Graduado en Ingeniería Informática

Universidad a Distancia de Madrid

Departamento de Ingeniería Informática

TRABAJO DE FIN DE GRADO

UDIMA, un enfoque dinámico: Diseño centrado en el usuario e implementación de una app clave en adaptar la metodología universitaria al entorno móvil.

Autor: Francisco Javier Clares Mateo

Directora: Sonia Pamplona Roche

MADRID, FEBRERO DE 2020

Declaración de originalidad:

El autor de este trabajo, Francisco Javier Clares Mateo, con D.N.I. 77570265M, declara que el contenido de este trabajo de fin de grado es original, y en el caso de haber utilizado las ideas o contenidos de otros trabajos de otros autores están convenientemente citados, de acuerdo con las normas de citación establecidas.

El número de palabras de este trabajo, excluyendo los anexos es: <11.228-palabras>

Índice

Índice de figuras 5
Índice de tablas 6
Resumen 7
Palabras clave 7
Abstract 8
Keywords 8
1. Introducción 9
1.1. Planteamiento del problema 9
1.2. Objetivos 9
1.3. Visión 9
2. Trabajos relacionados 10
2.1. Universidad de Murcia App 10
2.1.1. Interfaz 10
2.1.2. Ventajas 10
2.1.3. Desventajas 10
2.1.4. Conclusión 11
2.2. La Complutense (Universidad Complutense de Madrid) 11
2.2.1. Interfaz 11
2.2.2. Ventajas 11
2.2.3. Desventajas 12
2.2.4. Conclusión 12
2.3. Universidad de Almería 12
2.3.1. Interfaz 12
2.3.2. Ventajas 13
2.3.3. Desventajas 13
2.3.4. Conclusión 14
2.4. Universidad de Barcelona 14

2.4.1. Interfaz 14
2.4.2. Ventajas 14
2.4.3. Desventajas 14
2.4.4. Conclusión 15
2.5. Universidad Miguel Hernández de Elche 15
2.5.1. Interfaz 15
2.5.2. Ventajas 15
2.5.3. Desventajas 15
2.5.4. Conclusión 16
2.6. Valoración final 16
3. Planificación temporal y costes 17
3.1. Especificación del calendario 17
3.2. Planificación del calendario 17
3.2.1. Distribución recursos de tiempo por fase 18
3.2.2. Diagrama de Gantt 18
3.3. Planificación de costes 23
3.3.1. Recursos Humanos 23
3.3.2. Recursos de Software 24
3.3.3. Recursos de Hardware 24
3.3.4. Otros 25
3.3.5. Estimación coste total proyecto Software 25
4. Desarrollo 25
4.1. Requisitos 25
4.1.1. Requisitos funcionales 27
4.1.2. Requisitos no funcionales 27
4.1.3. Requisitos software 27
4.2. Análisis 28
4.2.1. Justificación de la Arquitectura. 28
4.2.2. Estudio de viabilidad. 29
4.2.3. Casos de uso. 29

4.3. Diseño centrado en el usuario 32
4.3.1. Iteración I. 32
4.3.2. Iteración II. 44
5. Evaluación 53
5.1. Requisitos funcionales. 53
5.1.1. RF1: Gestionar eventos mediante calendario estudiante. 53
5.1.2. RF2: Contactar con los profesores del estudiante. 53
5.1.3. RF3: Disponer de espacio personal “Mis Asignaturas”. 54
5.2. Requisitos no funcionales. 54
5.2.1. RNF1: Seguridad. 54
5.2.2. RNF2: Disponibilidad. 54
5.2.3. RNF3: Usabilidad. 55
5.3. Puesta en Producción. 55
6. Conclusiones 58
6.1. Objetivos. 58
6.2. Publicación y Feedback final. 59
7. Bibliografía 60

Índice de figuras

Fig.1: Interfaz aplicación móvil de Universidad de Murcia. 10
Fig.2: Interfaz aplicación móvil de Universidad Complutense Madrid. 11
Fig.3: Interfaz aplicación móvil de Universidad de Almería. 13
Fig.4: Interfaz aplicación móvil de Universidad de Barcelona. 14
Fig.5: Interfaz aplicación móvil de Universidad de Elche. 15
Fig.6: Detalle horas asignadas a fases. 18
Fig.7: Detalle planificación cronograma general. 18
Fig.8: Diagrama de Gantt evolución general. 19
Fig.9: Detalle planificación de tareas Análisis. 20
Fig.10: Diagrama de Gantt fase de Análisis. 20
Fig.11: Diagrama planificación de Diseño funcional C.U. 21
Fig.12: Diagrama de Gantt fase de Diseño funcional C.U. 21
Fig.13: Diagrama planificación de Puesta en Producción. 22
Fig.14: Diagrama de Gantt fase de Puesta en Producción. 22
Fig.15: Evolución Udima APP con hitos. 23
Fig.16: Análisis de requisitos iniciales. 26
Fig.17: Estadísticas globales uso sistema operativo móvil. 28
Fig.18: Diagrama UML de casos de uso. 31
Fig.19: Interfaz principal Udima APP. 33
Fig.20: Estructura general BBDD MariaDB Udima APP. 34
Fig.21: Diseño interfaz asignaturas Udima APP. 38
Fig.22: Detalle tabla matrícula. 38
Fig.23: Detalle tabla asignación. 39
Fig.24: Detalle tabla asignaturas. 39
Fig.25: Diseño interfaz “Mi calendario”. 41
Fig.26: Nueva tabla “actividades” en base de datos. 42
Fig.27: Diseño interfaz “Mis profesores”. 44

Fig.28: Asociación tablas para interfaz “Mis profesores”. 45
Fig.29: Layout ”elemento_grid”. 45
Fig.30: Diseño interfaz “Contacto”. 47
Fig.31: Detalle código envío POST. 48

Índice de tablas

Tabla.1: Tabla salarial 2020 23
Tabla 2: Coste RRHH Udima APP 24
Tabla 3: Coste Software Udima APP 24
Tabla 4: Coste Hardware Udima APP 24
Tabla 5: Coste Otros Udima APP 25
Tabla 6: Estimación coste total de Udima APP 25
Tabla 7: Resultados TU01. 35
Tabla 8: Resultados TU02. 37
Tabla 9: Resultados TU03. 40
Tabla 10: Resultados TU04. 43
Tabla 11: Resultados TU05. 46
Tabla 12: Resultados TU06. 48
Tabla 13: Resultados TU07. 52

Resumen:

En los ámbitos pertenecientes al campo de la enseñanza, existe controversia sobre cuál es el método idóneo para ofrecer una educación y metodología de estudio capaz de proveer un desarrollo posterior profesional de calidad a los alumnos a lo largo de su etapa de formación estudiantil. El objetivo de este trabajo es justificar que la aparición de las nuevas tecnologías fomenta y genera un nuevo escenario más atractivo capaz de llevar a la enseñanza a un nivel más práctico, dinámico, portable y eficiente que pueda ofrecer la metodología clásica.

En el inicio del documento se realiza un análisis de las necesidades de los estudiantes en el entorno diario y se muestra la importancia de disponer de los elementos clave tales como un calendario de eventos importantes y sus alertas, un progreso visual del avance general de los estudios que se están cursando o bien un contacto directo y cómodo con los profesores asociados a las asignaturas escogidas; todo ello al alcance de la mano, en forma de aplicación móvil.

A lo largo del desarrollo de la aplicación se muestra mediante una metodología basada en test de usabilidad y prototipado evolutivo, como las capacidades y la implicación de los estudiantes aumenta conforme se tienen en cuenta sus aportaciones y se obtiene una evolución considerable y poco esperada con respecto al comienzo del mismo.

Como resultado del trabajo, se genera una herramienta que traslada, resume y engloba las funcionalidades más utilizadas por los estudiantes y que permite optimizar y estructurar su tiempo efectivo de estudio de una forma más proactiva.

A modo de conclusión, se deduce la necesidad de disponer de esta herramienta de apoyo, como simbiosis perfecta al aula virtual, aplicando una ayuda indispensable para el estudiante a la hora de la planificación y estructuración de los contenidos evaluables y, por tanto, una interacción más cercana y visual con la universidad.

Palabras clave: Aplicación, planificación, cercana, móvil, usabilidad.

Abstract:

Actually, there’s a big contrast between traditional teaching and teaching based on new technologies and there are different opinions that comment the inclusion of these new technologies in these methologies is an error.
The goal of this work is show how technology allows for better results and a more efficient organization of students relative to traditional methods.
At the beginning of the investigation, a study is carried out in the forums and students groups about what their requeriments are when considering having an app to support daily study.
Students prefer to have an event calendar with notifications, a section with the contact of all their teachers, and the overall visual progress of their studies.
Students prefer all of these functions on a mobile terminal, simplifying use and reducing management time.
Throughout the development, the feedback from the students, causes important changes in the app that were neither expected at the beginning of the app.
Finally, students would be able to perform better results if they had a mobile app that would help them with the management and the organization of subjects of course.

Keywords: App, planning, near, mobile, usability.

CAPÍTULO 1: INTRODUCCIÓN

1.1. Planteamiento del problema.

El estudiante necesita disponer de los contenidos de la Universidad UDIMA en una plataforma móvil atractiva, sencilla y visual, que le permita interaccionar de forma ágil a todo lo relacionado con sus estudios en la Universidad. Es necesaria la simplificación de los contenidos a este nivel visual y añadir una comunicación más eficiente y directa para generar esa comodidad de tener la Universidad a pocas interacciones de la mano del propio estudiante. Actualmente, aunque existe la plataforma web completa y bien estructurada, complica este acceso rápido y eficiente de los estudiantes a través de sus terminales móviles, así como, se estiman necesarias ciertas funcionalidades de control y organización que permitan mantener al estudiante ligado de una forma más estrecha a la evolución de sus cursos, controlando en todo momento su seguimiento de forma gráfica y sencilla.

1.2. Objetivos

El objetivo principal es proporcionar y condensar de forma específica y atractiva, el contenido relevante del estudiante para un acceso universal, visual y simplificado. Como objetivos secundarios se pueden destacar:

- Promover y mejorar la experiencia del estudiante.
- Captar y atraer a potenciales estudiantes.
- Generar nuevas metodologías ágiles de estudio.
- Simplificar y actualizar el contacto estudiante-profesor.
- Fomentar y ampliar la comunidad de estudiantes online.
- Diseñar e implementar aplicación móvil para los estudiantes.

1.3.Visión

Agilizar y actualizar el uso estructurado de los servicios de la Universidad UDIMA basándonos en la aplicación de nuevas tecnologías móviles centradas en el estudiante.

CAPÍTULO 2: TRABAJOS RELACIONADOS

El marco teórico de nuestro proyecto, consiste en una aplicación móvil de una universidad, con lo que, en este punto, se analizan las aplicaciones existentes en la actualidad. De esta forma, tras el análisis competitivo se tendrá como conclusión las características diferenciadoras de nuestra aplicación con respecto al resto.

2.1. Universidad de Murcia App:

2.1.1. Interfaz

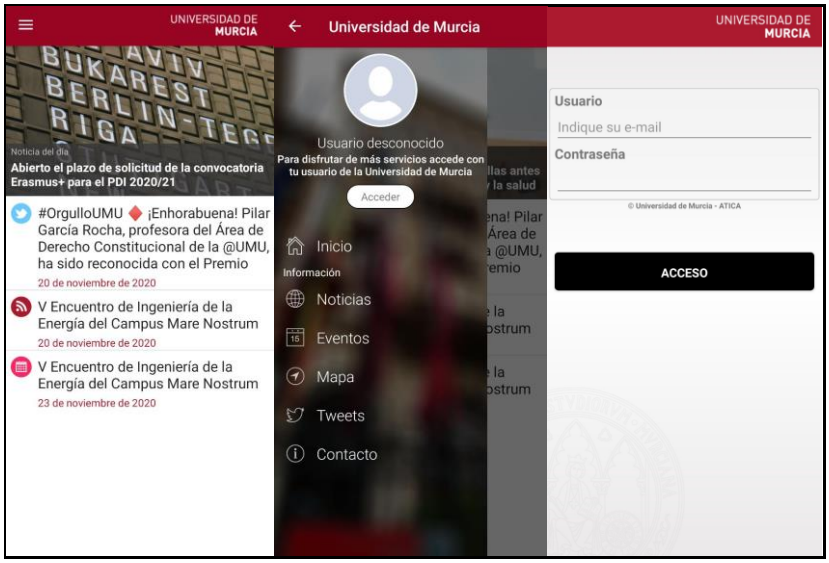


Fig.1: Interfaz aplicación móvil de Universidad de Murcia.

2.1.2. Ventajas

- Gestión eficiente de clases virtuales.
- Rápida, intuitiva y sencilla.
- Idea de primeros pasos y ayuda inicial coherente y útil.
- Calendario de eventos ágil.

2.1.3. Desventajas

- Interfaz de inicio con demasiada información y poca estructura visual y sencilla.
- Acceso autenticado demasiado rígido. Faltan opciones de inicio vía “Google” y a través de elementos de identificación del propio terminal: Huella, reconocimiento facial…
- Sección de “Tweets” irrelevante y poco útil para el estudiante.
- Sección de “Contacto”: Se hace necesario formulario de contacto, chat interno…

2.1.4. Conclusión

Aplicación con diseño elemental y funciones bien definidas, aunque rígidas y en ocasiones con información poco relevante y que desorienta al estudiante.

2.2. La Complutense (Universidad Complutense de Madrid)

2.2.1. Interfaz

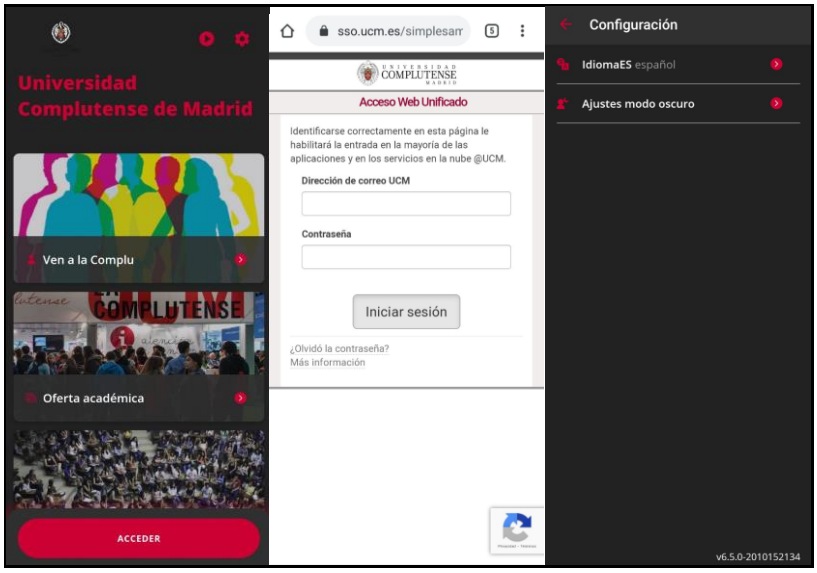


Fig.2: Interfaz aplicación móvil de Universidad Complutense de Madrid.

2.2.2. Ventajas

- Óptima ayuda inicial. Diseño minimalista pero centrado en el usuario. Buena interacción con un dedo a la hora de seguir el flujo de la misma.

- Modo oscuro: Muy acertada tanto la propia activación instantánea como la limpia transformación de la aplicación a éste.
- Interfaz muy visual, mostrando en la parte inferior la funcionalidad inicial más importante, login de usuario: Color diferenciador, lugar estratégico, visible, útil.
- Buena integración de RRSS de la Universidad, de un modo responsive y adaptado.
- Excelente inicio estructurado de la información en bloques de diseño visuales, de tamaño ideal y buena movilidad para el usuario.

2.2.3. Desventajas

- Integración login usuario: No te permite loguearte en la propia aplicación, redirige a su acceso web unificado con dirección de correo de la propia universidad. Falta de adaptación al usuario.
- Si el usuario desea usar el botón "Atrás" del propio dispositivo, no vuelve al flujo anterior, se cierra directamente la aplicación.
- Menú idioma sin funcionalidad: No permite seleccionar otro idioma diferente. Resta. Si no suma funcionalidad, mejor no incluirlo.
- Funcionalidad no correcta en botón "Home". No vuelve al inicio principal de la aplicación.

2.2.4. Conclusión

Aplicación basada en una capa inicial con diseño atractivo y dinámica, aunque con contenido demasiado adaptado a la versión web. Se denota más bien un sitio web embebido dentro de una capa Android en vez de un desarrollo completo en plataforma. Pinceladas interesantes pero escasas.

2.3. Universidad de Almería:

2.3.1. Interfaz

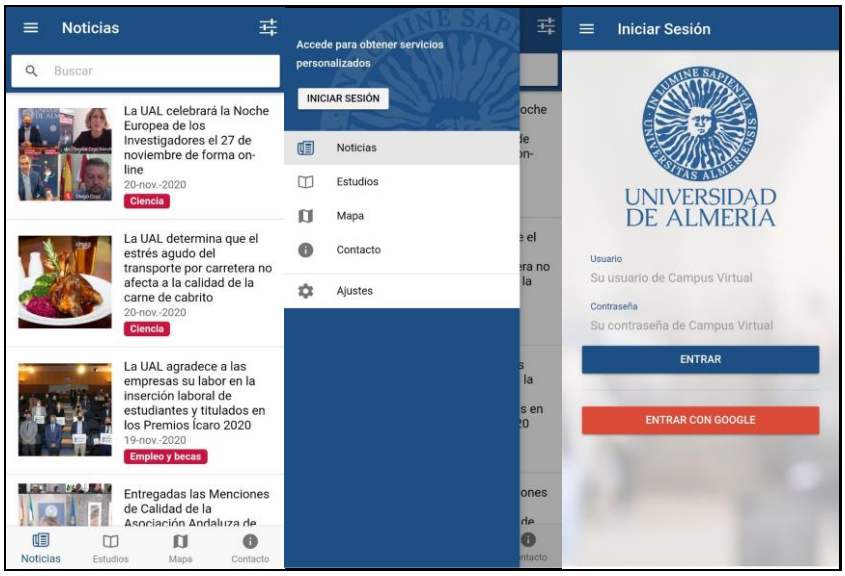


Fig.3: Interfaz aplicación móvil de Universidad de Almería.

2.3.2. Ventajas

- Posibilidad de elegir idioma en inglés, aunque únicamente traduce las cabeceras, no contenido.
- Estupenda interfaz de login: Posibilidad de entrar vía Google. Integrado en la propia aplicación.
- Sencilla pero útil sección con cita previa y contactos telefónicos.
- Buen filtro para noticias vía palabras clave por campo determinado de actuación.
- Mensajes de alerta: Aunque se echa en falta botón “Salir”, avisa correctamente al usuario si desea o no cerrar la aplicación tras pulsar tecla atrás en terminal móvil.

2.3.3. Desventajas:

- Diseño interfaz de bienvenida poco atractiva. Demasiado texto.
- Sección de noticias nada más comenzar. Poca utilidad para el estudiante.
- Sin flujo ni botones (home) que faciliten la navegabilidad del usuario.
- Los contenidos se redirigen a la web. Poca funcionalidad en aplicación, ni integración.

- Contacto con errores: No se muestra formulario web: Versión obsoleta SSL.
- Diseño general pobre: Colores poco llamativos y opciones sin utilidad.

2.3.4. Conclusión:

Aplicación atractiva, con interesante interfaz de login y opciones interesantes como "Google". Menú sencillo, funcional y sección inicial de noticias poco relevante.

2.4. Universidad de Barcelona:

2.4.1. Interfaz

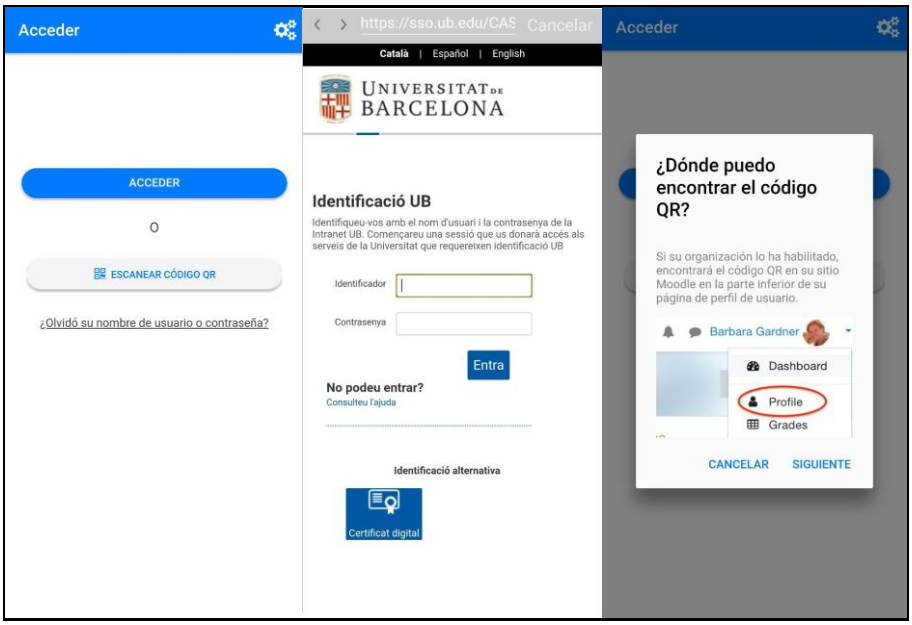


Fig.4: Interfaz aplicación móvil de Universidad de Barcelona.

2.4.2. Ventajas

- Menú de ajustes pensado en usuario: Editor de texto enriquecido…

2.4.3. Desventajas:

- Interfaz inicial de login consiste en el propio login web de la Universidad, dejando la sensación de poca adaptación a la plataforma móvil. No app móvil.
- Uso de login con certificado, muy poco útil y usable en plataformas móviles.
- No permite ninguna opción ni contenido sin realizar login en la Universidad.
- Diseño inexistente.

2.4.4. Conclusión:

Se trata de una aplicación nada intuitiva para el usuario, y básicamente se trata del sitio web embebido. Ausencia de elementos atractivos e interesantes y baja navegabilidad.

2.5. Universidad Miguel Hernández de Elche:

2.5.1. Interfaz:

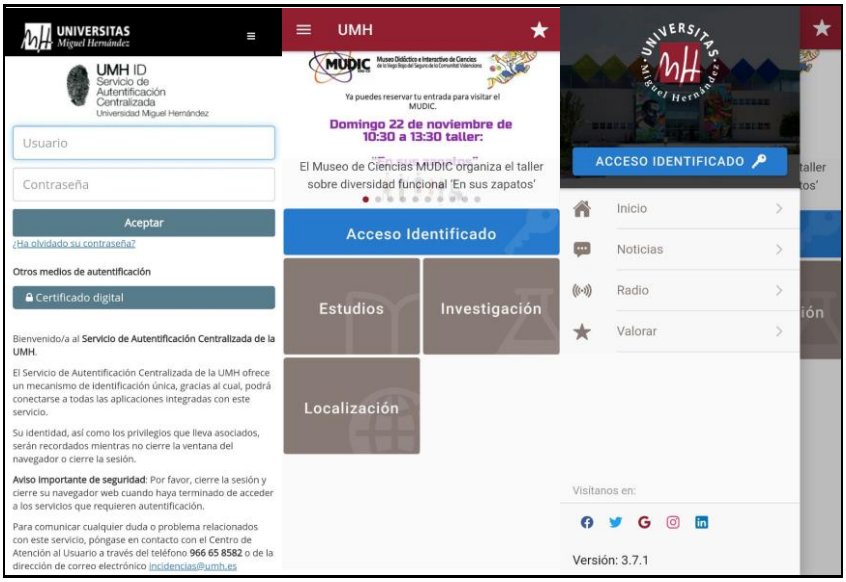


Fig.5: Interfaz aplicación móvil de Universidad de Elche.

2.5.2. Ventajas:

- Avisos y condiciones legales explícitos.
- Opción interesante de valoración y feedback por parte de los usuarios. Preocupación por la mejora continua, adaptación a cambios y centrados al usuario.
- Opción innovadora: Radio para los estudiantes. Falta por pulir, pero podría ser adaptada a charlas de profesores, consejos, apuntes de forma audible y más.
- Menú de opciones en panel izquierdo con útil diseño y acceso rápido.

2.5.3. Desventajas:

- Interfaz de login estilo web nada más iniciar la aplicación.
- Flujo incómodo para el usuario. No botones adelante, atrás, home, sobre todo, se echan en falta una vez entras a cualquier opción, y se pierde la usabilidad.

- Interfaz de inicio con poco diseño estructurado, sin simetría, colores poco atractivos, opciones muy sencillas y sin fundamento.

2.5.4. Conclusión:

Aplicación con menú lateral limpio y con un diseño ideal, aunque con interfaz inicial no simétrica y poco elegante, demasiado texto. La interfaz de login más orientada a web, debido al certificado digital, y con demasiada información.

2.6. Valoración final:

Tras el estudio de mercado, la mayoría de las Universidades han adquirido una plantilla del portal "Universia", con lo que el aspecto general y las opciones son réplicas. Tras finalizar el desarrollo de "Udima App", ésta se desmarca y diferencia de las demás en los siguientes puntos:

- Interfaz de bienvenida sin demasiada información, adaptando el enfoque del estudiante al área privada y al "no estudiante" a formar parte de la universidad.
- Amplía la funcionalidad al usuario "no estudiante", posibilitando el modo test de la aplicación, y acercando más la universidad a estos potenciales estudiantes. En las versiones estudiadas, se limitan a noticias e información general.
- Login propio: No se muestra la plataforma web embebida en la aplicación, sino que se establece un acceso privado adaptado al entorno.
- Flujo de navegabilidad más claro, con menor cantidad de opciones que permiten al estudiante obtener una grata experiencia.
- Importancia de los colores de la "Marca": En las aplicaciones estudiadas, hacen un guiño a los colores corporativos, pero de una forma leve. En Udima App, se ha alternado en todo momento con los colores propios (verde y rojo), ofreciendo una experiencia que permita recordar a los estudiantes su universidad por ellos.
- Las opciones internas propias del estudiante, no es posible compararlas, ya que no se dispone del acceso privado (no ser alumnos de dichas universidades).

CAPÍTULO 3: PLANIFICACIÓN TEMPORAL Y COSTES

El proyecto dispone de unas características bien definidas que marcarán la gestión del calendario y los hitos principales del mismo. Se definen las siguientes:

- **Ciclo de vida**: Prototipado evolutivo agile.
- **Metodología:** Scrum combinado con Kanban.
- **Estándares:** ISO 12207 (Procesos ciclo de vida) e ISO 9126 (Evaluación calidad).
- **Fases:** Cada una supone un punto importante dentro del calendario:
    - Fase I: Análisis: Objetivo y viabilidad.
    - Fase II: Diseño funcional centrado en usuario: Iteración y feedback continuo de las sub-fases de diseño, desarrollo y pruebas.
    - Fase III: Puesta en producción: Publicación de la aplicación.

Una vez caracterizado el proyecto, se determina la planificación temporal y de costes.

3.1. Especificación del calendario.

Se diferencian las jornadas laborales y su franja horaria de la siguiente forma:

- **Días laborales**: *(días: L-V; franja horaria: 16:00-18:00):* Se componen de un total de 55 días (26/10/2020 – 08/01/2021) x 2 Horas/día = 110 Horas.
- **Días no laborales** *(días: S-D; franja horaria: 07:00-15:00):* Se componen de un total de 22 días completos (24/10/2020 – 03/01/2021) x 8 Horas/día + 1 día incompleto (09/10/2020) x 6 horas = 182 Horas.

El total de horas del ciclo de vida de Udima APP es de **292 horas**.

3.2. Planificación del calendario:

A lo largo del proyecto se determinarán de forma flexible y global la relación de las jornadas laborales con las horas marcadas, siempre teniendo en cuenta que, dentro de las

fases generales, se trata de un proceso ágil entre los stakeholders, sin ser rígidos en aspectos limítrofes, pero sí, en los hitos principales, como los test de usabilidad.

3.2.1. Distribución de recursos de tiempo por fase:

Se distribuyen un total de horas planificado (292 horas) en las fases establecidas:

Fig.6: Detalle horas asignadas a fases.

3.2.2. Diagrama de Gantt:

Se detallan las tareas e hitos más importantes del proyecto y su duración en días, considerando la determinación del horario mostrado, ampliando la definición temporal, también de forma gráfica, en la que se indican tanto el plano general como los planos detallados de cada una de las fases ya descritas.

- **Planificación general**: Estimación temporal del proyecto completo:

La estimación general se basa en dos fechas principales a tener en cuenta:

- Fecha de inicio: 24/10/2020 – 07:00 am.
- Fecha de finalización: 09/01/2020 – 15:00 pm.

Se muestra el gráfico temporal resultante:

Nombre	Duración	Inicio	Terminado	Predecesores
Udima APP	36,75 days	24/10/20 7:00	9/01/21 15:00	
F1. Análisis	2,5 days	24/10/20 7:00	27/10/20 18:00	
F2. Diseño Funcional Centrado Usuario	32,75 days	28/10/20 16:00	6/01/21 18:00	2
F3. Puesta Producción	1,5 days	7/01/21 16:00	9/01/21 15:00	8

Fig.7: Detalle planificación cronograma general.

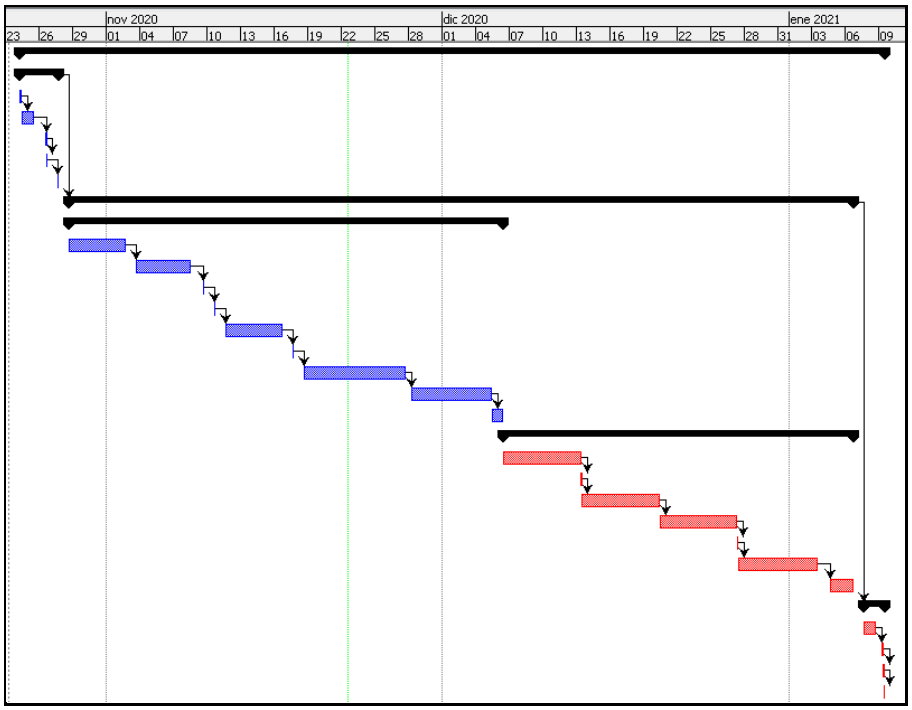


Fig.8: Diagrama de Gantt evolución general.

El proyecto se basa en técnicas centradas en el estudiante, el elemento indispensable, el cual genera el feedback necesario para adaptar la fase F2, de una forma muy dinámica y flexible, incorporando sprint cortos pero funcionales y mostrando de forma explícita los siguientes test de usabilidad (TU):

- F2. Diseño funcional centrado en el usuario:
  - Iteración 1:
    - TU01 – Login.
    - TU02 – Interfaz Home.
    - TU03 – Interfaz Asignaturas.
    - TU04 – Interfaz Mi Calendario.
  - Iteración 2:
    - TU05 – Interfaz Mis Profesores.
    - TU06 – Interfaz Contacto.
    - TU07 – Navegabilidad menús.

- F3. Puesta en producción:
    - TU08 – Interfaz Sistema Udima APP.

En cada uno de los test de usabilidad (TU) se establece un patrón funcional guiado por el desarrollador dónde se muestra en tiempo real el diseño y navegabilidad del apartado concreto, y mediante, una reunión global con los stakeholders seleccionados en un ambiente de tormenta de ideas ágiles y enfocadas a la usabilidad de los estudiantes. De esta forma, se involucran todos los interesados en un ambiente idóneo y de confianza, a la vez que se gana el tiempo necesario para el desarrollo evitando consultas individuales. Se detalla la planificación del cronograma de cada fase y sus tareas:

- **<u>Fase I: Análisis:</u>** Objetivo principal, elección de arquitectura razonada, estudio de mercado, planteamiento de objetivos secundarios e identificación de requisitos iniciales (no rígidos):

	Nombre	Duración	Inicio	Terminado	Predecesores
	Udima APP	36,75 days	**24/10/20 7:00**	**9/01/21 15:00**	
	F1. Análisis	2,5 days	**24/10/20 7:00**	**27/10/20 18:00**	
	F1.1 Estudio viabilidad	0,5 days	24/10/20 7:00	24/10/20 11:00	
	F1.2 Planificación cronograma y costes	1,5 days	24/10/20 11:00	25/10/20 15:00	3
	F1.3 Justificación Arquitectura	0,125 days	26/10/20 16:00	26/10/20 17:00	4
	F1.4 Estudio de mercado	0,125 days	26/10/20 17:00	26/10/20 18:00	5
	F1.5 Educción requisitos iniciales "ForoUdima"	0,25 days	27/10/20 16:00	27/10/20 18:00	6

Fig.9: Detalle planificación de tareas Análisis.

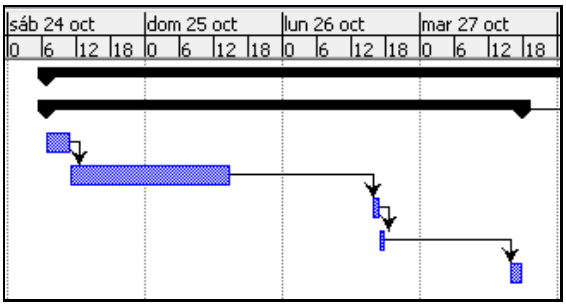


Fig.10: Diagrama de Gantt fase de Análisis.

- **<u>Diseño funcional</u>**: Parte central, dinámica, flexible iterativa con retroalimentación continua de los interesados: Diseño, codificación, pruebas y test de usabilidad (en reuniones ágiles con tormenta de ideas).

Nombre	Duración	Inicio	Terminado	Predecesores
Udima APP	36,75 days	**24/10/20 7:00**	**9/01/21 15:00**	
F1. Análisis	2,5 days	**24/10/20 7:00**	**27/10/20 18:00**	
F2. Diseño Funcional Centrado Usuario	32,75 days	**28/10/20 16:00**	**6/01/21 18:00**	**2**
F2.1 Iteración 1	18,75 days	**28/10/20 16:00**	**6/12/20 13:00**	
F2.1.1 Sprint 1: Interfaz principal	3 days	28/10/20 16:00	2/11/20 18:00	
F2.1.2 Sprint 2: Arquitectura	3 days	3/11/20 16:00	8/11/20 15:00	10
TU01: Login	0,25 days	9/11/20 16:00	9/11/20 18:00	11
TU02: Interfaz Home	0,25 days	10/11/20 16:00	10/11/20 18:00	12
F2.1.3 Sprint 3: Interfaz Asignaturas	3 days	11/11/20 16:00	16/11/20 18:00	13
TU03: Interfaz Asignaturas	0,25 days	17/11/20 16:00	17/11/20 18:00	14
F2.1.4 Sprint 4: Calendario: Diseño	4 days	18/11/20 16:00	27/11/20 18:00	15
F2.1.5 Sprint 5: Calendario: Avisos y Alertas	4 days	28/11/20 7:00	5/12/20 13:00	16
TU04: Interfaz Mi Calendario	1 day	5/12/20 13:00	6/12/20 13:00	17
F2.2 Iteración 2	14 days	**6/12/20 13:00**	**6/01/21 18:00**	
F2.2.1 Sprint 1: Interfaz Mis Profesores	3 days	6/12/20 13:00	13/12/20 11:00	
TU05: Interfaz Mis Profesores	0,25 days	13/12/20 11:00	13/12/20 13:00	20
F2.2.2 Sprint 2: Registro contacto	3 days	13/12/20 13:00	20/12/20 11:00	21
F2.2.3 Sprint 3: Contacto interactivo	3 days	20/12/20 11:00	27/12/20 9:00	22
TU06: Interfaz Contacto	0,5 days	27/12/20 9:00	27/12/20 13:00	23
F2.2.4 Sprint 4: Flujo de navegabilidad	3,5 days	27/12/20 13:00	3/01/21 15:00	24
TU07: Navegación menús	0,75 days	4/01/21 16:00	6/01/21 18:00	25

Fig.11: Diagrama planificación de Diseño funcional C.U.

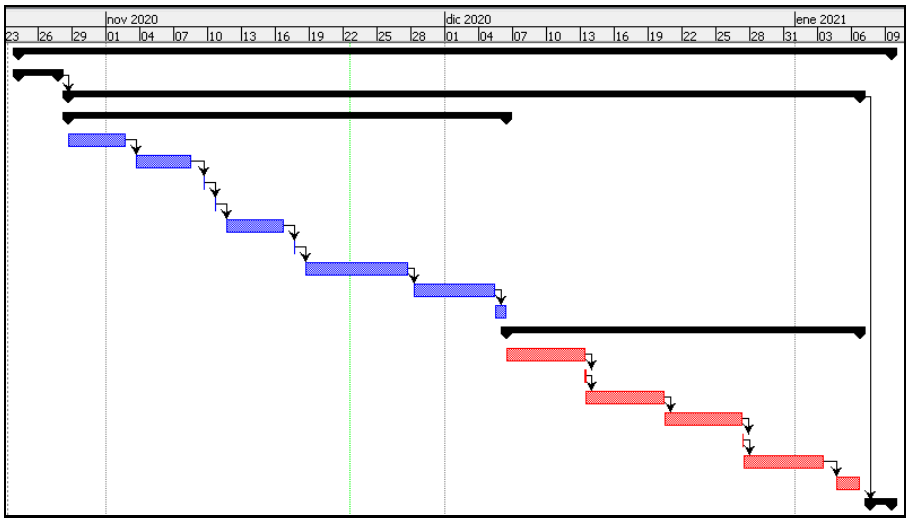


Fig.12: Diagrama de Gantt fase de Diseño funcional C.U.

- **<u>Puesta en Producción:</u>** Fase final del proyecto. Puesta en producción y test de usabilidad del sistema completo por parte de los interesados:

	Nombre	Duración	Inicio	Terminado	Predecesores
1	**Udima APP**	36,75 days	**24/10/20 7:00**	**9/01/21 15:00**	
2	**F1. Análisis**	2,5 days	**24/10/20 7:00**	**27/10/20 18:00**	
8	**F2. Diseño Funcional Centrado Usuario**	32,75 days	**28/10/20 16:00**	**6/01/21 18:00**	**2**
9	**F2.1 Iteración 1**	18,75 days	**28/10/20 16:00**	**6/12/20 13:00**	
19	**F2.2 Iteración 2**	14 days	**6/12/20 13:00**	**6/01/21 18:00**	
27	**F3. Puesta Producción**	1,5 days	**7/01/21 16:00**	**9/01/21 15:00**	**8**
28	F3.1 Sprint 1: Pruebas de sistema	0,5 days	7/01/21 16:00	8/01/21 18:00	
29	TU08: Interfaz sistema Udima APP	0,5 days	9/01/21 7:00	9/01/21 11:00	28
30	F3.2 Sprint 2: Publicación APP	0,25 days	9/01/21 11:00	9/01/21 13:00	29
31	F3.3 Sprint 3: Feedback + Soporte + Documen	0,25 days	9/01/21 13:00	9/01/21 15:00	30

Fig.13: Diagrama planificación de Puesta en Producción.

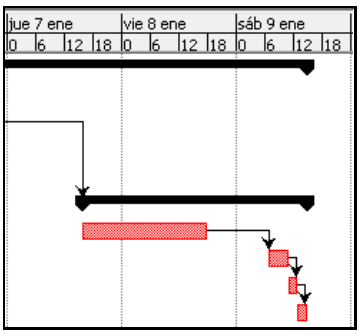


Fig.14: Diagrama de Gantt fase de Puesta en Producción.

Se establece una línea del tiempo, claramente diferenciada en una etapa inicial de Análisis y una etapa final encargada de la puesta en producción de la aplicación software. La etapa o fase predominante y por tanto, más importante, es la fase de diseño centrado en el usuario, una fase iterativa, adaptada a los cambios y opiniones de los estudiantes, y en la que, se entrelazan, de forma dinámica y sin orden establecido, las sub-fases de diseño de funcionalidad en forma de prototipos evolutivos, construcción de los mismos, y sus correspondientes pruebas con participación de todos los interesados, para poder afirmar que la aplicación tiene las funcionalidades esperadas, que están todas las que son y únicamente las que deben ser.

Se muestra gráfico de la evolución general con los hitos importantes:

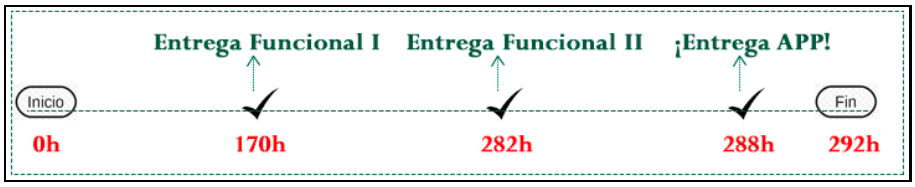


Fig.15: Evolución Udima APP con hitos.

3.3. Planificación de costes:

Para la planificación de costes, se detallan las siguientes consideraciones:

- La gestión del ciclo de vida de Udima APP es gestionado por un único Ingeniero, que adoptará diferentes roles durante el mismo: Analista, diseñador, desarrollador y técnico de sistemas.
- El adquiriente del proyecto software no facilita recursos hardware ni software.
- No se detallan costes de las actividades de Jefe de Proyecto habituales en metodología SCRUM.
- En términos de coste, un día de trabajo equivale a 8 horas.

En la estimación de costes generales del proyecto, se detallan a continuación las diferentes áreas a tener en cuenta:

3.3.1. Recursos Humanos:

Personal humano involucrado en el proyecto Software, con diferentes roles.

- **Tabla salarial/categoría**. Publicación BOE Núm.134, Sec. III. 13/05/2020:

BOLETÍN OFICIAL DEL ESTADO
Núm. 134 Miércoles 13 de mayo de 2020 Sec. III. Pág. 32827

TABLAS SALARIALES

Grupo	Categoría	Diario	Hora
E	Analista Informática	52,50 €	6,56 €
E	Diseñador	45,41 €	5,68 €
E	Programador-Analista de Informática	45,98 €	5,75 €
C	Operador Informática	40,75 €	5,09 €

Tabla 1: Tabla salarial 2020.

- **<u>Asignación Tareas/Recursos Humanos</u>**:

Se establece la siguiente relación a cada rol desempeñado según fase, tareas y duración de las mismas:

TABLAS SALARIALES					
Fase	Perfil Profesional	Duración	Esfuerzo	Coste hora	Coste total
F1: Análisis	Analista Informática	20h	100%	6,56 €	131,20 €
F2: Diseño C.U.	Diseñador	262h	20%	5,68 €	297,63 €
F2: Diseño C.U.	Programador-Analista de Informática	262h	80%	5,75 €	1.205,02 €
F3: Producción	Operador Informática	10h	100%	5,09 €	50,90 €
TOTAL COSTE RRHH					**1.684,75 €**

Tabla 2: Coste RRHH Udima APP.

3.3.2. Recursos de Software:

Elementos de Software de terceros para la realización del proyecto Udima APP.

RECURSOS SOFTWARE				
Recurso	Descripción	Precio	Duración	Coste total
Prototipado	Generar diseño de prototipos	6€ / mes	4 meses	24,00 €
Bases de Datos	Gestión de base de datos	1418€ / año	4 meses	472,67 €
Diseño	Gestión diseños elementos APP	60€ / mes	4 meses	240,00 €
Publicación	Alta Google Play publisher	25,00 €		25,00 €
TOTAL COSTE SOFTWARE				**761,67 €**

Tabla 3: Coste Software Udima APP.

3.3.3. Recursos de Hardware:

Elementos de Hardware necesarios para la infraestructura del proyecto Udima APP.

RECURSOS HARDWARE				
Recurso	Descripción	Precio	Duración	Coste total
Arquitectura	Servidor acceso global	20€ / mes	4 meses	80,00 €
TOTAL COSTE HARDWARE				**80,00 €**

Tabla 4: Coste Hardware Udima APP.

No se estiman costes de computador cliente del Ingeniero. Se presupone que ya dispone de propio y el contrato es únicamente por el desarrollo de la aplicación.

3.3.4. Otros:

Se asumen costes de energía eléctrica en el consumo de las horas de gestión de todo el ciclo de vida de la aplicación Udima APP.

OTROS				
Recurso	Descripción	Precio	Duración	Coste total
Electricidad	Consumo Computador	1,15€/150h	292h	2,24 €
TOTAL COSTE OTROS				2,24 €

Tabla 5: Coste Otros Udima APP.

3.3.5. Estimación coste total proyecto Software:

La estimación del coste total de nuestro proyecto Software, resulta de la suma de todos los costes anteriormente mencionados:

PLANIFICACIÓN COSTE UDIMA APP	
Coste	Coste total
Coste RRHH	1.684,75 €
Coste Software	761,67 €
Coste Hardware	80,00 €
Coste Otros	2,24 €
TOTAL COSTE	2.528,66 €

Tabla 6: Estimación coste total de Udima APP.

Se asumen las estimaciones con un riesgo de variación del 6% tanto en cronograma como en coste debido a factores de incertidumbre y cambios contenidos en cualquier proyecto.

CAPÍTULO 4: DESARROLLO.

En este capítulo se muestra un proceso ágil, iterativo, flexible, guiado por los test de usabilidad centrados en el estudiante, que marcarán la variación del análisis, requisitos, diseño, implementación y pruebas, para un resultado marcado por los requisitos deseados.

4.1. Requisitos

Tras el estudio de mercado en relación a las aplicaciones de otras universidades, se necesita conocer qué es aquello que demandan los estudiantes en una futura aplicación de su universidad. Para conocer sus preferencias se realiza una encuesta online a través de los siguientes canales:

- Foro Udima general.
- Foro Udima de Ingeniería Informática.
- Grupo de personas al azar no pertenecientes a Udima.

La encuesta es abierta y sencilla, limitando a tres respuestas (funcionalidad esperada por la aplicación) por estudiante. Tras el análisis y verificación de los resultados, se muestran las siguientes conclusiones:

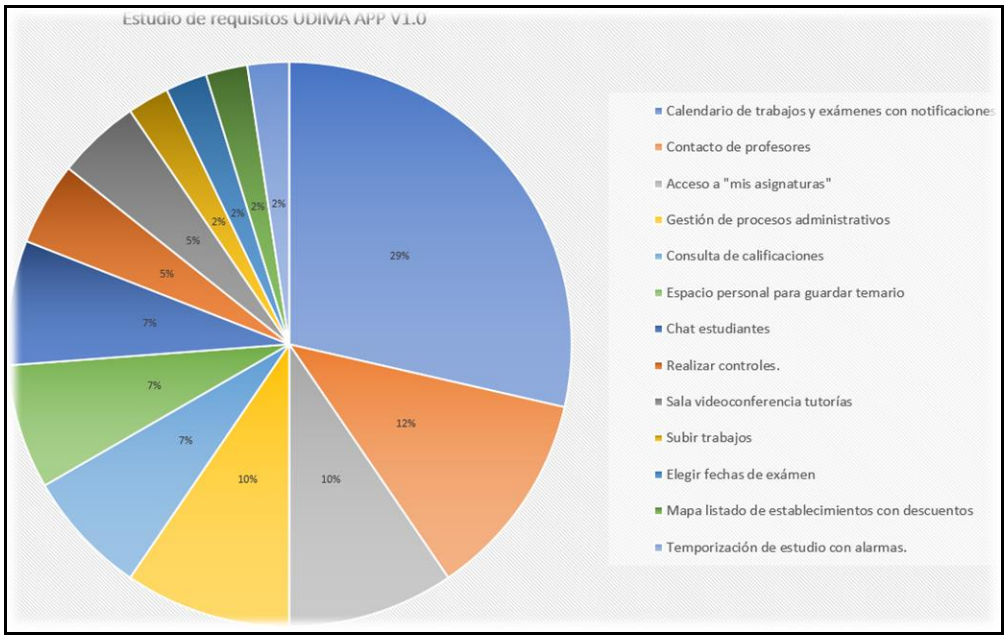


Fig.16: Análisis de requisitos iniciales.

El estudio de requisitos permite conocer desde el punto de vista del estudiante, cuales son las funcionalidades que deben estar en la aplicación, de modo inicial. Posteriormente, dependiendo de la planificación del calendario estimada y sobre todo de los test de usabilidad, estos requisitos podrán permanecer, cambiar, o incluso no poder llevarse a

cabo, de ahí, que insistamos en la agilidad y flexibilidad propia de este proyecto de este tipo.

4.1.1. Requisitos funcionales:

- Gestionar eventos mediante un calendario personal del estudiante (RF1): Fechas de exámenes, entregas, eventos importantes, con de notificaciones y alertas.
- Contactar con los profesores del estudiante (RF2). Directo, visibilidad de online/offline, chat, posible video tutoría.
- Disponer de un espacio personal (RF3): Temario descargable + material profesores + detalle "Mis asignaturas" (Calificaciones, estado...).
- Gestionar procesos administrativos (RF4): Matrícula, consulta expediente…
- Chatear con otros estudiantes (RF5).

4.1.2. Requisitos no funcionales:

- Seguridad (RNF1): La aplicación debe contar con un acceso controlado con contraseña cifrada y canales de comunicación cifrados en origen y destino.
- Disponibilidad (RNF2): El acceso a los datos del estudiante en plataforma cloud, debe estar disponible al menos al 98% de cada período de 24 horas.
- Rendimiento (RNF3): La tasa de errores no puede superar el 10% y los tiempos de respuesta deben estar comprendidos entre 5-20 segundos en cualquier petición.
- Usabilidad (RNF4): Fácil de usar, con menús visuales atractivos y mostrando las funcionalidades de una forma clara y directa.
- Legalidad (RNF5): Debe cumplir la Ley Orgánica española 3/2018 de 5 de diciembre de Protección de Datos y Garantía de los derechos digitales.

4.1.3. Requisitos software:

- Sistema operativo Android (RH1): Se recomienda a partir de la versión 11.0.

- Almacenamiento disponible (RH2): Se recomienda disponer de al menos 150 Mb libres en disco.
- Conexión a Internet (RH3): La aplicación no dispone de cometido offline.

Los requisitos iniciales del proceso de obtención indicado serán analizados a través de un modelo de prototipado evolutivo funcional, mediante la técnica ágil de desarrollo conjunto de aplicaciones (JAD), implicando así a los potenciales usuarios de la aplicación en el desarrollo y en la adaptabilidad de dichos requisitos, ahorrando tiempo en contrastar las opiniones de los estudiantes por separado.

4.2. Análisis.

El análisis del proyecto comienza en definir qué es lo que se quiere obtener, y en este contexto, es la creación de la aplicación móvil de la universidad, con aquellos requisitos obtenidos a partir del proceso de educción anteriormente descrito. La arquitectura de la aplicación es uno de los puntos más influyentes y está debidamente justificado.

4.2.1. Justificación de la arquitectura.

La aplicación en la versión V1.0 únicamente será válida para el sistema operativo Android (Ver RH1) por los siguientes motivos:

- Estadísticas de uso en la sociedad: Es el sistema operativo móvil más usado en todo el mundo. De este modo, se dispone del más alto porcentaje de cuota del mercado y se consigue llegar a un mayor número de potenciales usuarios que podrán beneficiarse de las funcionalidades descritas.

Fig.17: Estadísticas globales uso sistema operativo móvil.

- Acceso a terminales Android: Altamente relacionado con el punto anterior, ya que, entre otros motivos, el coste de adquisición de un terminal Android es mucho más bajo que el de un terminal Apple, con lo que, el acceso a este tipo de terminales Android es más atractivo y, por tanto, llega a un mayor índice de población.
- Compatibilidad: Para la configuración, flexibilidad y adaptabilidad con diferentes plataformas, así como la integración con otro tipo de sistemas y librerías, Android, sigue siendo más compatible, amigable y sobre todo menos rocosa a la hora de este tipo de conjugación.

Se continua con el análisis global, realizando un estudio de viabilidad inicial.

4.2.2. Estudio de viabilidad.

Tras el estudio y planificación del calendario, recursos humanos y mercado actual, conociendo los requisitos y preferencias de los potenciales estudiantes como principales interesados, el proyecto de la aplicación como complemento perfecto del aula virtual se considera además de necesaria, viable, en los términos indicados, y, además, establece una nueva vía de comunicación y mejora en el ámbito y las metodologías de estudio correspondientes a los escenarios cada vez más dinámicos.
Dentro de esta justificación, en el análisis, se adopta un proceso y metodología flexible y muy cambiante, justificando y rebatiendo al mismo tiempo cada requisito en cada test de usabilidad, dependiendo de todos los interesados. El desarrollo de ciertos requisitos puede quedar fuera de la primera versión por requerir demasiados recursos temporales y humanos y, por tanto, se irá adaptando el calendario sobre los mismos, sobre las peticiones de los estudiantes y sobre las condiciones que surjan en cada momento.
Aunque las funcionalidades iniciales puedan variar debido a esta naturaleza, las fechas previstas de test de usabilidad y entregas parciales y totales, se mantienen en la previsión.

4.2.3. Casos de uso.

Tras el análisis de requisitos, empleamos como notación formal y estandarizada de los mismos, mediante los casos de uso, sus actores y los escenarios principales y secundarios, representando la secuencia de acciones propia del estudiante con la aplicación. Cada caso de uso se corresponde con los requisitos funcionales anteriormente descritos:

- CU01: Añadir evento al calendario docente
    - Actor principal: Estudiante.
    - Actor secundario: Profesor.
        - Escenario principal:
            - 1. Profesor inserta fechas límite actividades estudiante.
            - 2. Profesor valida datos en el Sistema.
            - 3. Estudiante accede a “Mi Calendario”.
            - 4. Estudiante pulsa evento en calendario y obtiene info.
        - Estudiante secundario:
            - 1. Estudiante no recibe eventos en calendario.
            - 2. Estudiante crea nuevo ticket de soporte.
            - 3. Profesor reasigna eventos al id de Estudiante.

- CU02: Escribir correo electrónico a Profesor.
    - Actor principal: Estudiante.
    - Actor secundario: Profesor.

        - Escenario principal:
            - Estudiante accede a “Mis profesores”.
            - Estudiante pulsa icono “correo electrónico”.
            - Estudiante rellena consulta y pulsa enviar.
            - Profesor recibe la notificación.
            - Estudiante recibe mensaje en área del Profesor.

        - Escenario secundario:
            - Estudiante no dispone de los profesores correctos.
            - Estudiante pulsa en validar profesores.
            - Aparece lista de Profesores.

- CU03: Consultar estado de estudios.
    - Actor principal: Estudiante.
    - Actor secundario: Profesor.
        - Escenario principal:
            - 1. Estudiante pulsa en “Mis asignaturas”.
            - 2. Sistema muestra asignaturas matriculadas.
            - 3. Sistema muestra avance global estudios.
            - 4. Profesor actualiza notas de asignaturas.
            - 5. Estudiante confirma avance actualizado.
        - Escenario secundario:
            - 1. Estudiante no recibe actualización de notas.
            - 2. Estudiante genera incidencia en soporte.
            - 3. Profesor valida asignaturas y refresca panel.

- CU04: Petición de soporte.
    - Actor principal: Estudiante.
    - Actor secundario: Soporte técnico.
        - Escenario principal:
            - 1. Estudiante dispone incidencia.
            - 2. Estudiante pulsa en contacto servicio técnico.
            - 3. Servicio técnico resuelve incidencia.
            - 4. Servicio técnico envía confirmación por correo.
        - Escenario secundario:
            - 1. Estudiante encuentra soporte técnico “no disponible”.

- 2. Soporte técnico realiza intervención telefónica.

Se muestra el diagrama de casos de uso en UML:

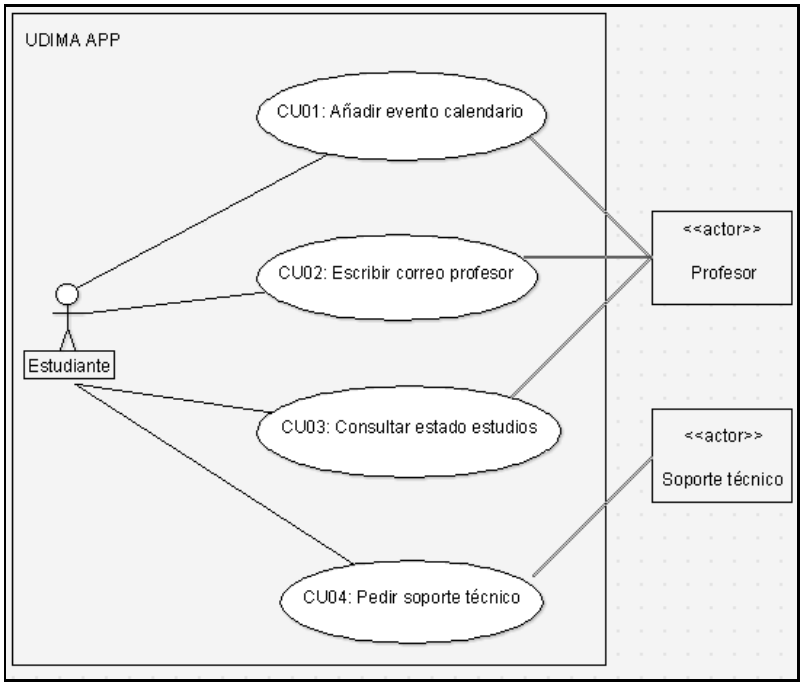


Fig.18: Diagrama UML de casos de uso.

4.3. Diseño centrado en el usuario.

Conforme a la planificación del cronograma se procede a desarrollar cada sprint dónde cada test de usabilidad se encarga probar las funcionalidades y aspectos relevantes de los Sprint anteriores implementados, con la siguiente metodología común:

- A: Análisis Sprint.
- I: Implementación Sprint.
- T: Test usabilidad Sprint.
- CL: Conclusiones test de usabilidad.

4.3.1. Iteración 1

Se desarrollan los aspectos relevantes de la misma:

A1.1 - F2.1.1 Sprint 1: Interfaz principal.

Dispone de los siguientes atributos:

- Facilidad de aprendizaje: Sencilla, botones de adecuado tamaño con funcionalidades principales, colores corporativos identificables.
- Satisfacción: Interfaz limpia, sin demasiado texto, visual.
- Efectividad: Navegación directa sin confusión.
- Memorabilidad: Pasos sencillos, fácilmente recordables.

Fig.19: Interfaz principal Udima APP.

El prototipo consiste en la oferta de matriculación inicial y, además, en el acceso al área privada del estudiante, con opciones que se desglosan en los siguientes sprints.

A1.2 - F2.1.2 Sprint 2: Arquitectura.

Se establecen los siguientes componentes:

- Base de datos principal: Modelo de base de datos relacional:
  - SGBD: MariaDB 10.0.38
  - Alojamiento: Localhost remoto vía ip: 149.202.228.229

- Estructura: Se muestra diagrama base de datos:

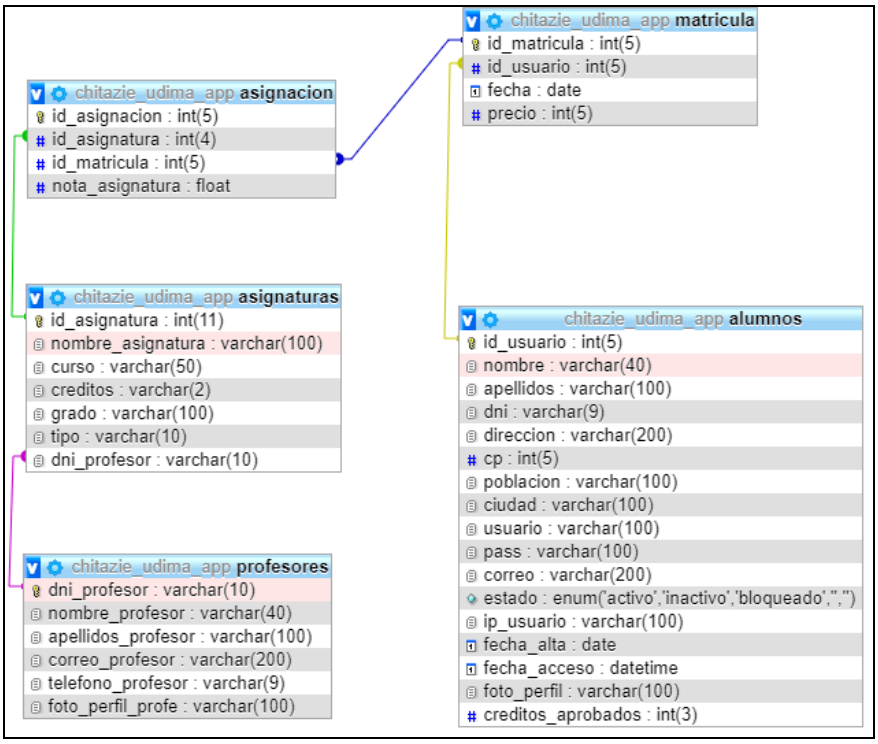


Fig.20: Estructura general BBDD MariaDB Udima APP.

- Tratamiento de datos aplicación-BBDD: Mecanismo síncrono de manipulación de los datos de forma bidireccional entre aplicación y base de datos:
  - Lenguaje PHP versión 7.3.6 con extensión mysqli.
- Lenguaje principal de implementación: Lenguaje usado en el desarrollo.
  - Java.
- IDE Android: Emulador implementación y pruebas de la aplicación:
  - Android Studio 4.1.0.0
- Herramientas de diseño: Adobe XD, Adobe Photoshop.

I1 – Implementación Sprint 1, Sprint 2.

La interfaz principal se compone de dos funcionalidades bien diferenciadas: matricúlate

(video promocional Udima) y login (acceso a la interfaz privada del estudiante). Cada botón dispone de un escuchador que recibe el evento onClick () y provoca la transición a una nueva Activity en Android mostrando cada parte de la aplicación por separado. Ver diagrama de flujo en Anexo A6.1.

T1.1 – TU01: Login.

Test de usabilidad de la interfaz de Login:

- Guion: Se facilita al estudiante un prototipo funcional de la aplicación y durante 5-8 minutos interactúa con él, facilitando las instrucciones de las tareas a realizar.
- Funcionalidades a probar:
  - TF01: Acceder con login correcto.
  - TF02: Acceder con login incorrecto.
  - TF03: Abandonar aplicación.
- Post-Uso: Interacción desarrollador – estudiante:
  - ¿Has sentido dificultad en el uso del login? ¿Te gusta el diseño?
  - ¿Echas de menos alguna función? ¿Cuál? ¿Por qué?
  - ¿Qué es lo que no te ha gustado? ¿Por qué?
- Resultados de la prueba: Impresiones de la interacción:

	TF01	TF02	TF03
Estudiante1			
Estudiante2			
Estudiante3			

Leyenda
Éxito
Dificultades
Fracaso

Tabla 7: Resultados TU01.

CL1.1 – Conclusiones TU01.

Se obtienen las siguientes conclusiones por parte de los estudiantes:

- Baja calidad logo principal de la aplicación. Tamaño demasiado grande.
- Botones mejor en colores lisos. Se hace necesario diferenciar el botón más importante, en este caso login.

- Ante el error de usuario/contraseña, especificar “cual” es exactamente el error, ya que puede ser contraseña errónea, usuario no existe…
- Falta funcionalidad “Recordar contraseña”.
- “Repetir contraseña” sólo útil en caso de registro, no en login.
- Necesario botón home.
- Botón salir, poco visual y sin mensaje de advertencia.

A lo largo del proyecto, y tras el último sprint de la iteración, se realizará una reunión con todos los interesados para valorar y/o validar las modificaciones según cronograma/coste.

T1.2 – TU02: Interfaz Home.

Test de usabilidad de la interfaz Home:

- Guion: Se facilita al estudiante un prototipo funcional de la aplicación y durante 5-8 minutos interactúa con él, facilitando las instrucciones de las tareas a realizar.
- Funcionalidades a probar:
    - TF04: Acceder con diferentes usuarios y comprobar datos concretos.
    - TF05: Gestionar el menú principal. Cambio entre secciones.
    - TF06: Cambiar contraseña del usuario.
    - TF07: Re-loguear con nueva contraseña.
    - TF08: Cambiar datos de usuario y comprobar nuevo perfil usuario.
    - TF09: Revisión envío de credenciales diferentes correos.
- Post-Uso: Interacción desarrollador – estudiante:
    - ¿Te parece atractivo el diseño del perfil de estudiante?
    - ¿Son completos los datos personales del estudiante en el perfil?
    - ¿Qué te ha parecido el cambio de contraseña? ¿Qué mejorarías?
    - ¿El menú es fluido y cómodo? ¿Te parece buena ubicación?
    - ¿La navegabilidad entre cambio de contraseña y login es útil?
    - ¿Qué echas de menos? ¿Por qué?

- ¿La plantilla del correo de confirmación te parece acertada?

- Resultados de la prueba: Impresiones de la interacción:

	TF04	TF05	TF06	TF07	TF08	TF09
Estudiante1						
Estudiante2						
Estudiante3						

Leyenda
Éxito
Dificultades
Fracaso

Tabla 8: Resultados TU02.

CL1.2 – Conclusiones TU02.

Se obtienen las siguientes conclusiones por parte de los estudiantes:

- Falta definición y claridad en opción actual en el menú lateral. El contraste del color hace que las letras apenas sean visibles.
- Información del grado siempre aparece fija en "Informática". Debe indicar el grado que se está cursando por parte del estudiante.
- Falta claridad y funcionalidad en usuarios bloqueado e inactivo. Si un usuario no tiene el estado "activo", no debe funcionar el login, y debe mostrar mensaje error.
- Posibilidad de cambio también de correo electrónico.
- Permitir preguntar al usuario si desea volver a loguear o hacerlo al salir de la aplicación para mejorar el flujo de navegabilidad.
- Diseño correo de envío de contraseña mejorable con colores corporativos y diseño más adaptado. Falta intensidad y visibilidad para atraer al usuario.
- Requerir mínima complejidad en nueva contraseña. Posibilidad de pedir contraseña anterior como medida de seguridad en el cambio de contraseña.

A1.3 - F2.1.3 Sprint 3: Interfaz Asignaturas.

En el sprint se diseña la interfaz obteniendo las asignaturas matriculadas por el estudiante concreto, con la particularidad de disponer o no de la nota, dependiendo del cierre o no de actas. En el caso de cierre de actas, aparecerá publicada la nota final, y en la zona

inferior de la interfaz, se tendrá en cuenta sólo aquellas asignaturas que tengan dicha nota final igual o mayor que cinco, para en este caso, actualizar el progreso en los estudios.

Fig.21: Diseño interfaz asignaturas Udima APP.

El progreso tiene en cuenta el total de créditos del grado, restando dinámicamente los créditos conseguidos en cada asignatura superada. Se muestran los campos más representativos de cada asignatura y el progreso en colores corporativos, con el detalle del verde en el progreso conseguido, para mejorar la usabilidad.

I1.3 – Implementación Sprint 3.

En la implementación, gira todo en torno al usuario, únicamente cuando dicho usuario existe, ha superado un login de éxito y ya está en la zona privada. No es lógico tener en cuenta el usuario en otro caso. Se asocia un id_matricula único al id_usuario:

id_matricula	id_usuario	fecha	precio
1	8	2020-11-24	1,500

Fig.22: Detalle tabla matrícula.

Cada asignación de una asignatura, se refiere al id_matricula:

id_asignacion	id_asignatura	id_matricula	nota_asignatura
1	1.388	2	6,57
2	1.514	2	7,55
3	1.198	2	4,8
4	1.757	2	(NULL)
5	1.755	1	9,5
6	1.756	2	5

Fig.23: Detalle tabla asignación.

Por tanto, se muestran aquellas asignaturas que han sido asignadas al id_matricula, correspondiendo al id_usuario que ha hecho login en la interfaz, y muestra la tabla dinámica, compuesta por un array de filas, y por tanto conociendo cuál es el total de asignaturas matriculadas al recibir la consulta.

En cada obtención de cada fila concreta, se comprueba la nota, si existe, y en caso de ser igual o superior a cinco, se guardan los créditos de la asignatura:

id_asignatura	nombre_asignatura	curso	creditos	grado	tipo	dni_profesor
1.198	Ingles	1	6	informática	T	(NULL)
1.375	Fundamentos de Programacion	1	6	informática	T	(NULL)
1.380	Tecnologia y Estructura de Computadores	1	6	informatica	B	000000002A

Fig.24: Detalle tabla asignaturas.

Sumándose al campo creditos_aprobados de la tabla alumnos. Con el total predefinido de créditos de los estudios cursados, la resta de los aprobados muestra el progreso en forma de un gráfico dinámico, implementado con MPAndroidChart. Ver diagramas de flujo en el Anexo A6.2.

Se destaca el cambio de contraseña en dos vertientes, o bien por petición del estudiante, introduciendo la nueva contraseña y actualizando en base de datos, tras codificar en MD5, o bien, generando una contraseña aleatoria, actualizando y enviando por correo electrónico para proceder al login de nuevo. Ver Anexo A6.3.

Se obtiene listado de asignaturas matriculadas y se suman créditos en función de si están aprobadas o no.

T1.3 – TU03: Interfaz Asignaturas.

Test de usabilidad de la interfaz Asignaturas:

- Guion: Se facilita al estudiante un prototipo funcional de la aplicación y durante 5-8 minutos interactúa con él, facilitando las instrucciones de las tareas a realizar.
- Funcionalidades a probar:
    - TF10: Añadir nuevas asignaciones de matrícula.
    - TF11: Cambiar notas y comprobar progreso dinámico.
- Post-Uso: Interacción desarrollador – estudiante:
    - ¿Los campos mostrados en las asignaturas te parecen relevantes?
    - ¿Qué te parece el diseño del gráfico de progreso?
    - ¿Añadirías alguna funcionalidad nueva? ¿Cuál? ¿Por qué?
- Resultados de la prueba: Impresiones de la interacción:

	TF10	TF11
Estudiante1		
Estudiante2		
Estudiante3		

Leyenda
Éxito
Dificultades
Fracaso

Tabla 9: Resultados TU03.

CL1.3 – Conclusiones TU03.

Se obtienen las siguientes conclusiones por parte de los estudiantes:

- Clasificación de las asignaturas por año y diferenciadas por el mismo en la tabla.
- Resaltar asignaturas aprobadas en color diferente a las no superadas.
- Mostrar mensajes de “aliento” tras cada porcentaje superado.
- Posibilidad de revisar matrículas anteriores con acceso a actas.
- Acceso directo a matriculación online, con capacidad de actualizar asignaturas matriculadas.
- Recibir feedback en la columna nota de los profesores.
- Cambiar colores de los porcentajes al aproximarse a la consecución del título.
- Acceso a documento PDF tras completar estudios con mensaje tutorizado.

A1.4 - F2.1.4 Sprint 4: Calendario Diseño.

En el sprint inicial del calendario del estudiante, se presenta la interfaz:

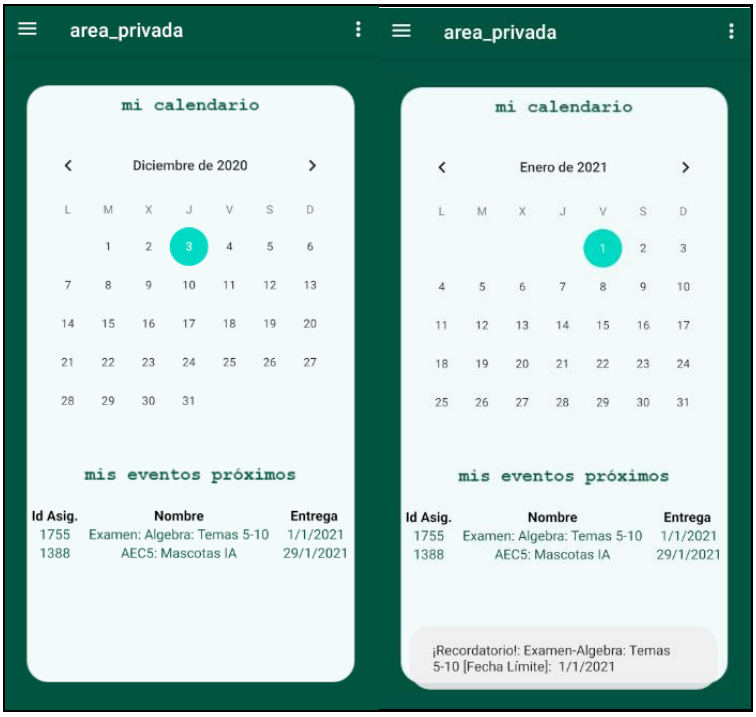


Fig.25: Diseño interfaz “Mi calendario”.

Cada id_asignatura tendrá asociada una actividad y una fecha límite. En el sprint F2.1.5 se especificará y se implementará la comparación entre la fecha del calendario y la fecha límite de las tareas correspondientes a las asignaturas matriculadas, que se conocen en la anterior interfaz de “mis asignaturas”.

A1.5 - F2.1.5 Sprint 5: Calendario: Avisos y alertas.

En el Sprint se diseña una vista principal con el calendario que marca el día actual. En la vista inferior, se muestran los eventos próximos, teniendo en cuenta las actividades asociadas a las asignaturas matriculadas, y, por tanto, la fecha límite de entrega de las mismas, no mostrando aquellas cuya dicha fecha es anterior. Cuando se pulsa en una fecha límite de alguna actividad marcada, se muestra un recordatorio del evento:

I1.5 - Implementación: Calendario: Avisos y alertas.

El primer cambio que se realiza en la implementación es en la base de datos, creando una tabla con las actividades asociadas a las asignaturas:

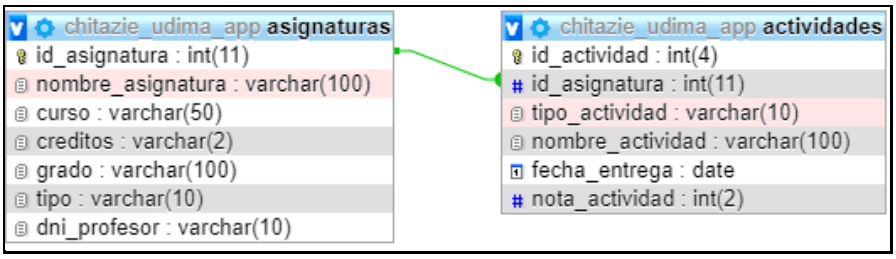


Fig.26: Nueva tabla “actividades” en base de datos.

Cada actividad dispone de un id_actividad única, que va asociado al id_asignatura de la tabla “asignaturas” y gracias a la fecha_entrega, que es la fecha límite, se puede controlar en relación a la fecha del calendario y también a la hora de controlar las actividades que se muestran en “mis eventos”.

En la implementación de código, se utiliza “Calendar.getInstance()”, y así obtenemos la fecha actual. Desde la llamada POST a “obteneractividades.php”, recibimos el JSON con “id_asignatura”, “tipo_actividad”, “nombre_actividad” y “fecha_entrega”. Con esta “fecha_entrega” formamos un nuevo Calendar, teniendo en cuenta que en Android los meses comienzan por “0” y, por tanto, tenemos que restar uno para que entienda nuestro mes real. Una vez se dispone del calendario actual y el calendario de cada fila del JSON se pueden comparar con la función “after” de Calendar, y en el caso de que la fecha del evento sea posterior, se añade una nueva fila a la tabla dinámica, llamando a la clase “tablaDinámica” creada ya en el caso de las asignaturas, y que se reutiliza aquí.

En relación al mensaje que se muestra al pulsar una fecha que tiene evento, se hace uso de la función “onSelectedDayChange” de CalendarView, dónde al pulsar en la fecha, se identifica el día, mes y año en sus variables y podemos compararlas con las que estamos recibiendo de cada fila del JSON, ya que la consulta devuelta está ordenada por la fecha_entrega. Se tiene en cuenta que ahora, hay que sumar una unidad al mes que se

pasa por parámetro en este método, para corresponder con el propio del array, y simplemente, con un IF dónde son iguales los tres parámetros, se lanza un "Toast" y se muestra el evento en la parte inferior de la interfaz. Ver diagrama de flujo en Anexo A6.4.

T1.5 – TU04: Interfaz Mi Calendario.

Test de usabilidad de la interfaz Asignaturas:

- Guion: Se facilita al estudiante un prototipo funcional de la aplicación y durante 5-8 minutos interactúa con él, facilitando las instrucciones de las tareas a realizar.
- Funcionalidades a probar:
    - TF12: Comprobar generación de eventos en fechas actividades.
    - TF13: Cambiar fechas y comprobar eventos próximos.
    - TF14: Cambiar usuarios y asignaciones en matrícula y comprobar mis eventos.
- Post-Uso: Interacción desarrollador – estudiante:
    - ¿Meterías más información en "mis eventos"? ¿Otro diseño?
    - ¿Las alertas y notificaciones te parecen funcionales? ¿Por qué?
    - ¿Añadirías alguna funcionalidad nueva? ¿Cuál? ¿Por qué?
- Resultados de la prueba: Impresiones de la interacción:

	TF12	TF13	TF14
Estudiante1			
Estudiante2			
Estudiante3			

Leyenda
Éxito
Dificultades
Fracaso

Tabla 10: Resultados TU04.

CL1.4 – Conclusiones TU04.

Se obtienen las siguientes conclusiones por parte de los estudiantes:

- Posibilidad de añadir marca en días con eventos registrados.
- Mejora: Evento sonoro y emergente en terminal móvil fuera de la app.
- Diseño "mis eventos" mejorable.

- Botón mostrar detalles del evento, con contenido de la actividad.
- Posible sección con contenido de las actividades y asignaturas.

4.3.2. Iteración 2

Se desarrollan los aspectos relevantes de la misma:

A2.1 - F2.2.1 Sprint 1: Interfaz Mis Profesores.

Dispone del siguiente diseño, conteniendo los datos referentes a la asignatura matriculada y el profesor concreto de la misma, y su información de contacto:

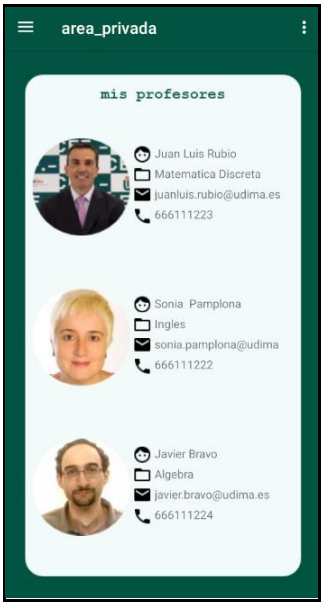


Fig.27: Diseño interfaz “Mis profesores”.

I2.1 - Implementación Sprint 1: Interfaz Mis Profesores.

Se toma como referencia el id_usuario que se identificó correctamente en login, para obtener los datos de cada profesor. Se deben cumplir las condiciones del INNER JOIN de la consulta, dónde se controla id_asignatura en la tabla asignación, y en la matrícula. Si está asociada al id_usuario, se consulta en la tabla profesores, unida a la tabla asignaturas, con la clave primaria dni_profesor.

id_asignatura	nombre_asignatura	curso	creditos	grado	tipo	dni_profesor
1.198	Ingles	1	6	informática	T	000000000A
1.375	Fundamentos de Programacion	1	6	informática	T	(NULL)
1.380	Tecnologia y Estructura de Computadores	1	6	informatica	B	000000002A

dni_profesor	nombre_profesor	apellidos_profesor	correo_profesor	telefono_profesor	foto_perfil_profe
000000000A	Sonia	Pamplona	sonia.pamplona@udima.es	666111222	soniap.png
000000001A	Juan Luis	Rubio	juanluis.rubio@udima.es	666111223	juanluisr.png
000000002A	Javier	Bravo	javier.bravo@udima.es	666111224	javierbravoa.png

Fig.28: Asociación tablas para interfaz “Mis profesores”.

En relación al código, se destaca el uso de “GridView” y una nueva clase “adaptadorGrid”. En “adaptadorGrid” se pasa como parámetro un array por cada elemento que se muestra en la interfaz, cada uno con el índice actual en el momento actual del bucle que recorre los profesores asignados en el listado de asignaturas matriculadas. Gracias a pasar el contexto, se pasa a gridProfesores el adaptador, se infla una nueva vista (elmento_grid), con los datos concretos del profesor desde base de datos (recibidos como JSON gracias a obtenerprofesores.php):

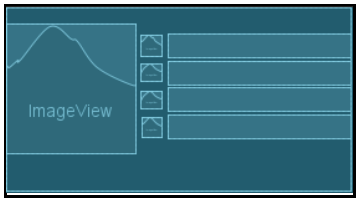


Fig.29: Layout ”elemento_grid”.

Mientras queden asignaturas, se muestran los profesores asignados, de forma dinámica, inflando la vista con el layout determinado.

Se muestra diagrama de flujo en Anexo A6.5.

T2.1 – TU05: Interfaz Mis Profesores.

Test de usabilidad de la interfaz Asignaturas:

- Guion: Se facilita al estudiante un prototipo funcional de la aplicación y durante 5-8 minutos interactúa con él, facilitando las instrucciones de las tareas a realizar.
- Funcionalidades a probar:
    - TF15: Añadir profesores aleatorios en distintas asignaciones.

- TF16: Comprobar funcionalidad contacto correo y teléfono.
- TF17: Comprobar creación dinámica de profesores.

- Post-Uso: Interacción desarrollador – estudiante:
  - ¿Qué te parece la ficha de profesor? ¿Alguna forma de contacto para añadir? ¿Cuál? ¿Por qué?
  - ¿Te parece bien que se repita la ficha si un profesor es repetido en otra asignatura? ¿Cambiarías la forma de mostrarlo? ¿Cómo?
  - ¿Te parece limpio y visual el diseño?
- Resultados de la prueba: Impresiones de la interacción:

	TF15	TF16	TF17
Estudiante1			
Estudiante2			
Estudiante3			

Leyenda
Éxito
Dificultades
Fracaso

Tabla 11: Resultados TU05.

CL2.1 – Conclusiones TU05.

Se obtienen las siguientes conclusiones por parte de los estudiantes:

- TF16 no ejecutable. No existe funcionalidad en el contacto de profesores. Necesario abrir aplicación de correo predeterminada y de llamada en terminal.
- Profesores repetidos. En caso de estar un mismo profesor en dos o más asignaturas, únicamente añadir asignatura adicional en ficha.
- Diseño elegante, sencillo y visual.
- Detalle de asignatura al pulsar sobre la misma.
- Posible sección chat con disponibilidad de profesores online/offline y determinar funcionalidad con respecto a la acción requerida.

A2.2 - F2.2.2 Sprint 2: Registro contacto.

Interfaz de contacto, ofreciendo la funcionalidad de a enviar petición al soporte de Udima.

Se muestra un mensaje de confirmación o error en caso de entregar o no el mensaje. Se dispone de un botón restablecer el cuál borra el contenido de ambos campos

Fig.30: Diseño interfaz “Contacto”.

I2.2 - Implementación Sprint 2: Registro contacto.

La implementación se establece con el punto de partida del nombre de usuario que hizo login, y se envía una petición POST a “contacto.php”, recibiendo usuario, asunto y texto. El correo de soporte, se deja predefinido en código, ya que es interno de Udima, pero también, gracias al usuario, se rescata el correo en base de datos para poder enviar una copia de la petición al correo personal del estudiante. En la consulta, se genera el html a enviar por correo, y mediante “mail” en php se produce el envío.

En la parte de desarrollo Android, se utiliza “StringRequest” y “RequestQueue” para enviar a “contacto.php” el texto introducido por el usuario. Si en “contacto.php” se envía correctamente, muestra el mensaje de “Mensaje enviado” y, por tanto, en modo de response no nulo, llega a Android, y muestra el mensaje de éxito que se visualiza en la interfaz anterior.

A destacar el uso de “Map”, para el envío de datos POST:

```
@Override
protected Map<String,String> getParams() throws AuthFailureError {
    Map<String,String> valores=new HashMap<~>();
    valores.put("nombreUsuario", nombreUsuarioRecibido);
    valores.put("asuntoContacto", contactoAsunto.getText().toString());
    valores.put("textoContacto", contactoMensaje.getText().toString());
    return valores;
}
```

Fig.31: Detalle código envío POST.

En el caso de pulsar el botón restablecer, simplemente se utiliza el método “setTest(“”)” en los elementos TextView “contactoAsunto” y “contactoMensaje”, quedando en blanco.

T2.2 – TU06: Interfaz Contacto.

Test de usabilidad de la interfaz Asignaturas:

- Guion: Se facilita al estudiante un prototipo funcional de la aplicación y durante 5-8 minutos interactúa con él, facilitando las instrucciones de las tareas a realizar.
- Funcionalidades a probar:
    - TF18: Enviar peticiones desde diferentes estudiantes.
    - TF19: Probar botón restablecer.
    - TF20: Envío de mensaje vacío.
- Post-Uso: Interacción desarrollador – estudiante:
    - ¿Desearías recibir el seguimiento de mensajes en la aplicación?
    - ¿Es necesario indicar la urgencia o prioridad?
    - ¿Es complejo? ¿Diseño acorde al resto? ¿Qué añadirías?
- Resultados de la prueba: Impresiones de la interacción:

	TF18	TF19	TF20
Estudiante1			
Estudiante2			
Estudiante3			

Leyenda
Éxito
Dificultades
Fracaso

Tabla 12: Resultados TU06.

CL2.2 – Conclusiones TU06.

Se obtienen las siguientes conclusiones por parte de los estudiantes:

- Falta capacidad de indicar la urgencia de la consulta: Alta, media, baja.
- Añadiría un plus a la aplicación, disponer de los mensajes enviados y recibidos por parte del soporte.
- Indicar en el botón de "Restablecer" un mensaje que informe que los datos han sido borrados.
- Capacidad de mensaje más visible a la hora de confirmar el envío y posible sonido.

Por el tiempo limitado del calendario, en la reunión del Sprint 2, el estudiante, elige, mejorar el diseño e implementar las funcionalidades requeridas en los demás test de usabilidad y retomar el "Contacto interactivo" en una versión futura de la aplicación. Se verá en "Flujo de navegabilidad".

A2.3 - F2.2.4 Sprint 4: Flujo de navegabilidad.

En el Sprint, se genera el mapa de flujo de navegabilidad completo, que se muestra en el Anexo, y tras la interacción entre estudiante-desarrollador en cada test de usabilidad, se han realizado las siguientes mejoras y cambios, tanto en diseño como en código:

- Se rediseña la interfaz principal. Botones sin relieve, logo más uniforme.
- Se incluye contenido en "Matricúlate" con un video promocional de Udima (con alta compresión webm), autoiniciado y con posibilidad de pause y reanudación.
- Se controla no posible login con usuario en blanco y usuario no correcto.
- Se implementa "olvidé mi contraseña", que consiste en generar una nueva contraseña aleatoria tras introducir el estudiante su usuario, enviar al correo electrónico registrado. Siempre si existe el usuario.
- Se mejora el perfil del estudiante, con diseño más limpio y la funcionalidad "cambiar contraseña" más funcional, actualizando la base de datos de forma útil.

- Se corrige el problema en “mis asignaturas” que seguía sumando los créditos ya aprobados anteriormente.
- Se establece un calendario más limpio y visual, y se limitan los eventos para no mostrar una lista interminable.
- Se recodifica de forma íntegra la sección “mis profesores”, generando un grid dinámico en función de los profesores asignados en asignaturas matriculadas.
- Se incluye botón “Salir” en menú principal, que devuelve el control a la interfaz de inicio.
- Se consigue una navegabilidad dentro del área_privada funcional, sin que se produzca pérdida de datos al pasar desde una sección a otra.
- Se establece bienvenida personalizada en menú con la inclusión del nombre del usuario actual.

I2.3 - Implementación Sprint 4: Flujo de navegabilidad.

Se muestran los detalles más importantes en las nuevas funcionalidades:

- **Matricúlate**: Se consigue con la inclusión de un elemento “WebView” que nos permite con el método “loadUrl” cargar la Url de nuestro vídeo.
- **Olvidé mi contraseña**: Se genera una nueva vista en forma de ventana, gracias a “AlertDialog.Builder” y se permite abrir dicha ventana en forma de elemento pop-up sobrepuesto en la interfaz de login. El estudiante introduce su nombre de usuario, y mediante petición POST se comprueba (comprueba.php). En caso de existir, se genera nueva contraseña aleatoria, se actualiza en el campo contraseña en base de datos (MD5) y se envía al correo registrado (olvidarpw.php).
- **Mis asignaturas:** Se añade un nuevo campo en la tabla “asignación”, llamado “es_mostrada”. Cuando se llama a “obtenermatricula.php”, se controla este campo. Si es “0”, y la asignatura tiene una nota igual o superior a 5, se suman los créditos, y se pone el valor de “es_mostrada” a 1. Por tanto, en el caso de que se

hayan sumado ya los créditos, en la primera comprobación, al tener el valor "1", nunca realizará la suma. Así, aunque el estudiante pulse en varias ocasiones sobre "mis asignaturas", nunca cambiará el porcentaje dinámico del progreso.

- **Mis profesores:** Se usa la clase "tabladinamica" y el elemento GridView, creando un xml concreto para el grid, y añadiendo como si fuera una fila más en dicha tabla.
- **Botón Salir:** Al pulsar la opción en el menú, se dirige al nuevo fragmento, añadiendo un "Intent" y activando el flag "clear_top", posteriormente cerrando la actividad con "finish". Se establece un "AlertDialog Builder" para mostrar al usuario la confirmación de sí desea o no salir realmente. Se tiene en cuenta también tanto la función indicada como "cerrar sesión" y también a la hora de salir de la aplicación de forma final.
- **Implementación Calendario**: Se cambia la implementación del calendario, ya que "CalendarView nativo en Android, no permite añadir eventos ni marcar fechas en el mismo. De esta forma, con "CompactCalendarView", se marcan los eventos próximos y ofrece una opción visual interesante a los estudiantes.
- **Navegabilidad de menú:** Se consiguen perdurar los datos propios de cada usuario, gracias a "setNavigationItemSelectedListener" del elemento navigationView, controlando en todo momento que id de fragmento se pulsa.

  De esta forma, y con "fragmentManager", se sustituye en "contenedor principal" que es la interfaz en blanco que se liga con el menú central" por el fragmento correspondiente a cada interfaz. De este modo, siempre muestra el mismo fragmento y con los datos pasados desde el área_privada con "bundle" privados. De esta forma, dentro del área del estudiante, siempre se conservarán sus datos en todas las interfaces y cambiarán en caso de que se produzca un login de un estudiante diferente.

T2.3 – TU07: Navegación menús.

Test de usabilidad de la interfaz Asignaturas:

- Guion: Se facilita al estudiante un prototipo funcional de la aplicación y durante 5-8 minutos interactúa con él, facilitando las instrucciones de las tareas a realizar.
- Funcionalidades a probar:
    - TF21: Probar menú principal y acceso a cada interfaz.
    - TF22: Probar olvidé mi contraseña.
    - TF23: Visualizar video promocional y botón salir.
- Post-Uso: Interacción desarrollador – estudiante:
    - ¿Qué te parece el menú? ¿Es cómodo e interactivo?
    - ¿Qué obtienes al probar "olvidé mi contraseña"?
    - ¿La navegabilidad es correcta y simple? ¿Confunde?
- Resultados de la prueba: Impresiones de la interacción:

	TF21	TF22	TF23
Estudiante1			
Estudiante2			
Estudiante3			

Leyenda
Éxito
Dificultades
Fracaso

Tabla 13: Resultados TU07.

CL2.3 – Conclusiones TU07.

Se obtienen las siguientes conclusiones por parte de los estudiantes:

- Iconos no funcionales: Ajustes, botón inferior junto a home.
- Interesante nueva funcionalidad de cambiar foto de perfil, incrementaría la calidad de la aplicación.
- Funcionalidad requerida en contacto con profesores, vía móvil y correo.
- Diseño "Matricúlate" huérfano, le falta atracción al usuario.
- Botón Atrás desde login hasta principal, en ocasiones no hace su función.
- Diseño final atractivo, alternando los colores corporativos y fluidez a la hora del uso con el menú.

CAPÍTULO 5: EVALUACIÓN

Se realiza una revisión formal del uso de la aplicación por parte del estudiante en modo de usuario final, realizando las pruebas de aceptación, con la finalidad de comprobar si los requisitos establecidos al comienzo y que se han refinado gracias a los test de usabilidad y la interacción continua y ágil entre todos los interesados, ha producido los resultados esperados en base a ellos.

5.1. Requisitos funcionales.

5.1.1. RF1: Gestionar eventos mediante calendario personal del estudiante:

Se obtienen las siguientes conclusiones tras la resolución de PA01 (Ver anexo A4.1):

- **Representación:** No se colapsan los resultados, ni se muestran demasiados elementos, únicamente los eventos próximos, en un límite básico de representación (<10 líneas).
- **Visibilidad**: Acertada del calendario, buena navegación.
- **Usabilidad**: Funcionalidad correcta. Se muestran únicamente aquellos eventos con fecha igual o superior al día actual.
- **Otros**: Se ofrecen marcas visuales en los días que tienen actividades pendientes. Al pulsar sobre la fecha, se muestra una alerta visual con el recordatorio de todas las actividades con dicha fecha límite.

5.1.2. RF2: Contactar con los profesores del estudiante:

Se obtienen las siguientes conclusiones tras la resolución de PA02 (Ver anexo A4.2):

- **Representación**: En función de los profesores asignados únicamente en las asignaturas matriculadas, se muestran en “grids” individuales de forma dinámica, sin dejar “grid” vacío, ni mostrar más elementos de los que realmente son.
- **Visibilidad**: Se muestran de forma clara los datos de contacto importantes.

- **Usabilidad**: Funciones “llamar” y “contactar” intuitivas y funcionales. Al tocar el icono se lanzan las aplicaciones nativas con las funciones reales y los datos concretos del profesor/a en cuestión. Presencia de scroll acertado.

5.1.3. RF3: Disponer de espacio personal “Mis Asignaturas”:

Se obtienen las siguientes conclusiones tras la resolución de PA03 (Ver anexo A4.3):

- **Representación**: Diseño intuitivo, con progreso general de los estudios, y con detalle de las asignaturas matriculadas. Se echa en falta más detalle en las mismas.
- **Visibilidad**: Colores intuitivos, corporativos, y gráfico centrado.
- **Usabilidad**: Sección de información al estudiante. Excelente comprobación de asignaturas ya aprobadas, para no aumentar el progreso total de forma irreal.

5.2. Requisitos no funcionales.

5.2.1. RNF1: Seguridad:

El servidor cuenta con un cifrado punto a punto en las conexiones TCP. El acceso de los usuarios es de forma controlada, generando un hash aleatorio y convirtiendo las credenciales en MD5 en base de datos, evitando texto plano. El fichero config.php con las credenciales para el acceso a la conexión de base de datos, está cifrado mediante certificado instalado en servidor destino. Se realiza comprobación del usuario previamente en base de datos, tanto para el login principal como para el envío de nueva contraseña (sólo disponible el envío del correo registrado, con nueva contraseña generada aleatoriamente, evitando intrusiones y vulnerabilidades). El cambio de contraseña sólo es posible dentro del área privada (Ver anexo A4.4).

5.2.2. RNF2: Disponibilidad:

El servidor alojado en Raiola Networks ofrece un 99% de disponibilidad 24/7 365 días.

5.2.3. RNF3: Usabilidad:

La aplicación cuenta con los siguientes aspectos destacados por el estudiante durante el proceso ágil retroalimentado y pruebas de aceptación:

- **Limpieza**. Sin saturación de elementos.
- **Color**. Fantástica utilización de los colores corporativos a lo largo de la navegabilidad de la misma.
- **Menú**. Sencillo, ágil, directo, mostrando sólo y nada más que las funciones importantes, sin adornos ni demasiada información redundante.
- **Conservación de datos**: Perfil con datos actualizados en todo momento.
- **Contraste**: Combinación de los diseños en botones muy acertado, mostrando en todo momento la interfaz actual del usuario, con opciones sencillas de navegabilidad para el usuario final.
- **Interacción**: Constante interacción con usuario. Entrega eficiente de notificaciones en correo, de forma segura y rápida.

5.3. Puesta en Producción.

Se abarca el Sprint "F3.1 Sprint 1: Pruebas del Sistema", englobando esta evaluación final como el último test de usabilidad "TU08: Interfaz sistema Udima APP, permitiendo a través de la retroalimentación de los interesados, obtener las siguientes nuevas mejoras:

- **Modo noche** (Ver Anexo A4.6): Adaptación a un diseño basado en tonos negros y luminosos para ofrecer una mejor visión al usuario en determinadas situaciones. En el desarrollo de este nuevo modo, se crea una nueva clase "modo" que permite comprobar si la aplicación está en modo noche (getDefaultNightMode tiene un valor de 2) y gracias al método "mostrarDialog", pasando por parámetro la propia actividad desde dónde se llama, su contexto, y, además, el layout modo día y el layout modo noche, el cual se muestra según el método esModoNoche, se infla un AlertDialog según el layout concreto. Ver diagrama de flujo en Anexo A6.7.

- **Nuevo diseño**: Iconos más representativos, botones adaptados, imágenes definidas, limpieza en las interfaces principales, mejor visibilidad y aspecto.
- **Nueva sección contenido** (Ver Anexo A4.7): Otorga la funcionalidad de compartir, descargar y visualizar el contenido de las asignaturas matriculadas. Se establece un "Spinner" con un "Adapter" de las actividades que se recogen desde "obtenercontenidoasignatura.php", pasando en modo POST la asignatura seleccionada en ese desplegable de las asignaturas matriculadas. Se crea un nuevo GRID con el contenido, con la importancia de construir la URL del fichero, dependiendo del curso, grado y la descripción de la misma, de tal forma, que se adapta dinámicamente al curso y grado concreto para la descarga, visualización o modo compartir previstos en la interfaz. Ver diagrama de flujo en Anexo A6.7.
- **Avisos en calendario** (Ver Anexo A4.8): Inclusión de notificación sonora y visual en el propio terminal, justo en el último día de entrega de la actividad. Se crea una nueva clase "guardarNotificacionWork" que extiende a Worker, una nueva API en Android, que nos permite guardar eventos y mantenerlos, aunque se apague el dispositivo. Creamos un canal de notificación "NotificationChannel" que permite pasar la descripción de la asignatura y que, conociendo en CompactCalendar los eventos que están guardados en la fecha límite, podemos llamar al método "lanzarNotificacion", y guardarla, para que en dicha fecha del evento aparezca, y esta fecha será el último día de la entrega de la actividad, en modo recordatorio final. Ver diagrama de flujo en Anexo A6.8.
- **Ver contraseña**: (Ver Anexo A4.9): Visualizar/ocultar la contraseña en la interfaz de login. Dependiendo si es modo noche o no, necesario para cambiar el diseño de los iconos de "ojo abierto y cerrado", lo único que se controla es si el "TextView" de la contraseña está oculto o no, con "getInstance" del método "getTransformationMethod" y dependiendo del estado, lo transforma en el modelo "Password" que estará oculto o modo "Texto" que estará visible. Con el método onClick se puede mostrar u ocultar al pulsar sobre el icono.
- **Media Expediente** (Ver Anexo A4.10): En la interfaz "Mi matrícula" se muestra

la media global de los estudios del estudiante. En relación al código, únicamente se realiza la suma de todas las notas disponibles y se dividen entre el total de asignaturas del estudiante, con la particularidad de usar un tipo "Double" para los decimales y dónde "Math.round" con su parámetro "100d" reduce a dos dígitos (número de ceros) la nota media final mostrada. Se muestra con "setText" en dicho Textview.

- **Modo Demo** (Ver Anexo A4.11): En la interfaz "Matricúlate" se permite el test de la aplicación con un usuario "temporal" y datos de "prueba" a un usuario interesado en Udima. Se crea "enviarpwtemp.php", para enviar usuario "temporal" y credencial de acceso. A modo de novedad, se controla que el usuario introduzca un correo válido (no se ha querido validar si está creado o no dicho correo), es decir, que contenga el patrón correcto de un email; y para ello se establece el método "validarCorreo" que utiliza "Patterns.EMAIL_ADDRESS" que nos permite comprobar con un patrón, y devolver desde el método "matcher" si es correcto o no (booleano). Si es correcto, se envían credenciales, y si no lo es, el AlertDialog, se queda en pantalla, mostrando un error de formato.
- **Petición de Contacto** (Ver Anexo A4.12): Mejora de la interfaz y funcionalidad. Es posible restablecer y enviar peticiones con copia al propio estudiante.
- **Interfaz inicio área privada:** Se implementa nueva interfaz, con accesos directos a matrícula y contenido del estudiante identificado. Además, se genera "obtenereventos.php" para listar todos los eventos (globales de la universidad), a modo de panel de información al estudiante, en una tabla dinámica. El menú lateral, incluye esta nueva área Inicio.
- **Interfaz matricúlate:** Nuevo diseño e inclusión de botones de información que nos envían a la web Udima (grados y másteres). Se añade otro botón para poder contactar con Udima a través del cliente de correo del dispositivo.

Debido al dinamismo, naturaleza ágil y centrada en el usuario de dicho proyecto, el diseño, interfaces y funcionalidades evolucionan iterativamente (Ver Anexo A2.1)

CAPÍTULO 6: CONCLUSIONES

6.1. Objetivos

Al comienzo del proyecto, se establecieron una serie de objetivos generales, los cuales analizamos a la finalización del proyecto:

- **Promover y mejorar la experiencia del estudiante.**

El producto final consiste en una aplicación capaz de simplificar y reducir el extenso contenido de la web principal de la universidad, adaptándolo a las secciones importantes para el día a día del estudiante y que se han desarrollado a lo largo del mismo: “Mi perfil”, “Mi matrícula”, “Mi calendario”, “Mis profesores”, “Mi contenido”. Sin duda, tras los test de usabilidad, la satisfacción y experiencia de uso de los estudiantes ha incrementado y ha fomentado un uso mayor y más interactivo gracias a la aplicación.

- **Captar y atraer a potenciales estudiantes.**

Se ha establecido una sección para atraer nuevos estudiantes, mostrando un video interactivo propio del sistema de matriculación y, además, generando un usuario temporal para que el futuro estudiante pueda tener una experiencia previa de la propia aplicación, trasmitiendo la confianza y la cercanía propia de la universidad.

- **Generar nuevas metodologías ágiles de estudio.**

Tras añadir los indicadores de “mi progreso” y “nota media de expediente” influyen en la psicología del estudiante, aumentando la motivación a mejorar sus técnicas de estudio y a la vez, generan esa simbiosis entre aplicación y usuario, necesaria para que se convierta en una rutina de estudio. Como parte principal, “Mi calendario”, generan ese aspecto visual, intuitivo y a la “mano” para mejorar la organización del estudiante y establecer una metodología ágil a la hora de entrega de actividades y realización de controles.

- **Simplificar y actualizar el contacto estudiante-profesor.**

El contacto desde la aplicación, de una forma ordenada y tan sencilla, permite al estudiante, desde su terminal móvil, poder realizar el contacto con los profesores de las asignaturas que tiene matriculadas. Se genera una experiencia positiva, ya que no necesita entrar en la web ni en su propio correo y buscar al remitente, sino que la estructura le permite disponer de las asignaturas clave y su contacto, de forma directa.

- **Fomentar y ampliar la comunidad de estudiantes online.**

La opción de "Mi contenido" se establece como vía futura de fomentar la comunidad de estudiantes, ya que favorece y estimula el modo de compartir tareas, documentos, incluso notas, que, aunque en esta primera versión, se queda en el camino previo, sí que es una opción futura, elegante y que cumple con la filosofía de nuestra aplicación.

- **Diseñar e implementar aplicación móvil para los estudiantes.**

El objetivo principal del proyecto, era el de construir de forma real la aplicación móvil para los estudiantes, y, además, se ha implementado en un host remoto, con una base de datos real, y con un acceso universal a través de Internet, en cualquier parte del globo. El diseño se ha realizado de una forma caracterizada por estar centrada en el usuario, y que, gracias a los test de usabilidad, se ha adaptado, tanto en diseño, requisitos, como implementación y pruebas, a los deseos de los propios estudiantes.

6.2. Publicación y Feedback final.

En este último capítulo, entramos de lleno, en los últimos Sprint del proyecto, "F3.2 Sprint 2: Publicación APP" y "F3.3 Sprint 3: Feedback + Soporte + Documentación", en los cuales, se establece a modo de trabajo futuro propuesto, con la autorización legal de la publicación en "Google Play" e integración con la base de datos de Udima.
La ambición del proyecto va más allá y se espera la continuación a una versión V2 futura, con las mejoras de diseño correspondientes, la creación de un soporte real a los

estudiantes, y la documentación de uso, en forma de manual propio de uso de la aplicación.
Los plazos de entrega se han correspondido con los previstos en el inicio, y a modo de conclusión, ha permitido entender que el desarrollo de una aplicación debe ser ágil, flexible, adaptable y con un buen protocolo a la hora de establecer y aceptar cambios propuestos. Por supuesto, el dato más importante, el usuario final, el verdadero motor de la aplicación, que condiciona, y que, en este caso, ha condicionado, y modificado en "cierta parte" los detalles previstos en el papel, y que ha permitido realizar una aplicación útil, de cara al uso específico de los estudiantes, que realmente son los verdaderos partícipes de ella.

Como se ha comprobado, en el último Sprint de desarrollo, "F3.1 Sprint 1: Pruebas de sistema" se ha necesitado un esfuerzo mayor de los recursos humanos y de cronograma, ya que las nuevas implementaciones no contempladas en los Sprints anteriores, han necesitado de cambios en la estructura principal de la aplicación, con nuevos flujos de trabajo, y reinicio del ciclo de vida general, con sus flujos reales y con la inclusión de las pruebas de integración.
Se han estimado, por tanto, 8 horas más de trabajo en relación con el previsto.

A modo de conclusión del proyecto, se otorga las gracias a todas las personas que han estado ahí, principalmente la directora de este TFG y por supuesto, las personas ajenas a la universidad que han aguantado tantas y tantas horas de diferentes estados de ánimo.

BIBLIOGRAFÍA

[1] S. Sánchez, M. A. Sicilia y D. Rodríguez. *Ingeniería del Software. Un enfoque desde la guía Swebok*. Madrid: Garceta Grupo Editorial, 2011.

[2] "Android Developer", 2020 [En línea]. Disponible en: https://developer.android.com/. [Accedido: 28-oct-2020].

Anexos

El proyecto se complementa con la utilización de los siguientes anexos:

Anexo A1. Test de Usabilidad:

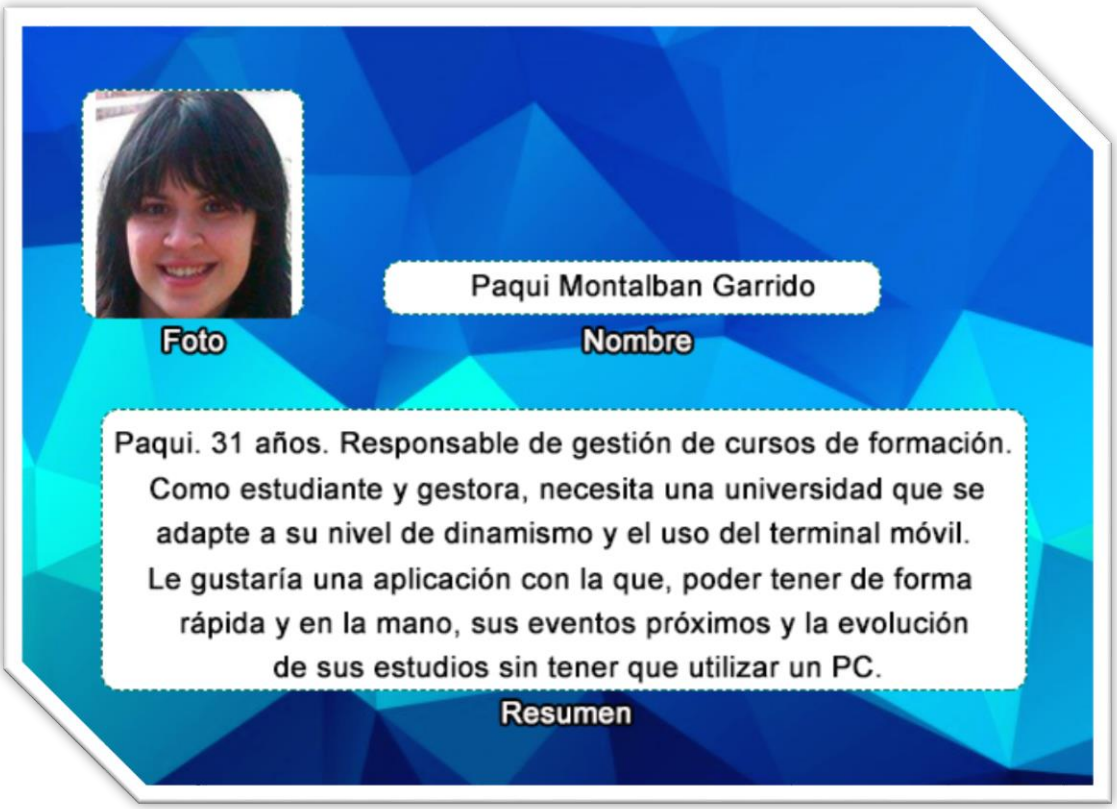
Para la realización de los test de usabilidad, a partir de ahora TU, se abordan los elementos siguientes:

- **Participantes**:

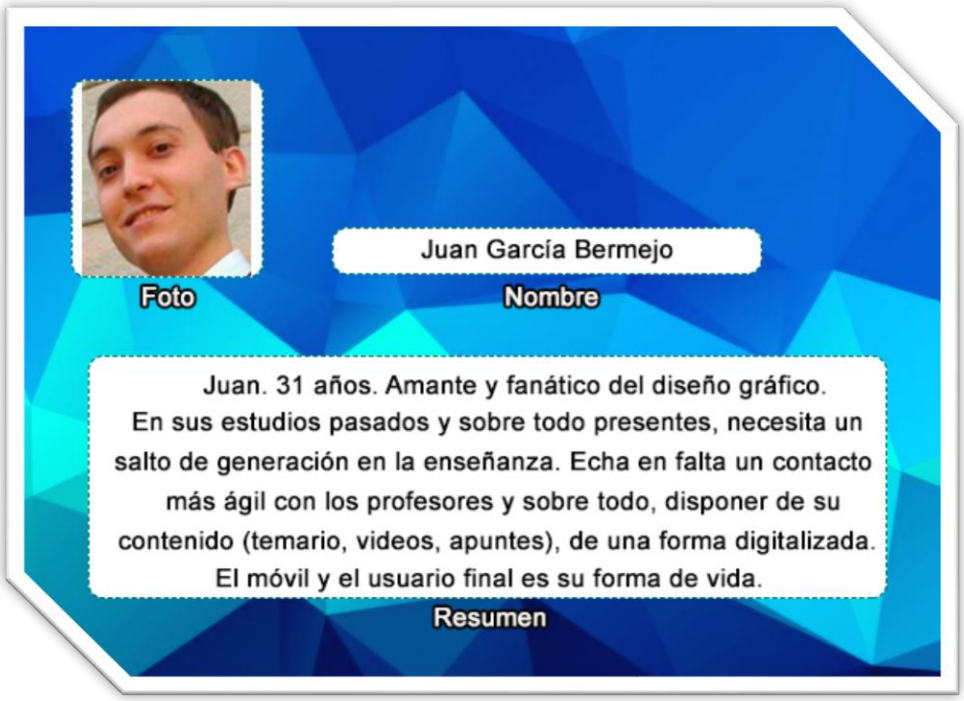
PARTICIPANTES			
Estudiantes	Número de test por estudiante	Modalidad	Cómo se ha realizado el test
3	8	Síncrona	Presencial con app en móvil Android

- **Análisis de Usuarios (Perfil):**

Perfil A: Rango 30-40 años. Independiente. Gestiones rápidas y eficientes.



Perfil B: Rango 30-40 años. Centrado en el usuario.



Perfil C: Rango 30-40 años. La enseñanza por vocación.

- **Contexto de Usuario**:

Eres una persona activa, que dispone de muy poco tiempo libre, ya que combinas los estudios en la Universidad y el trabajo a tiempo completo. Eres responsable y tu uso con el móvil es tan alto que se ha convertido en tu amigo diario. Por suerte, ha llegado a tus manos una aplicación móvil que te ayudará en la gestión del tiempo en relación a tus estudios y que te permitirá optimizar y aprovechar ese uso tan dinámico y constante del móvil que realizas en la actualidad. Especificando tu usuario facilitado por la Universidad, accederás a tu perfil, dónde podrás gestionar y visualizar tu contenido privado. A continuación, tomarás la aplicación y realizarás los test de usabilidad mostrados.

- **Test de usabilidad**:

Guion inicial común:

Bienvenid@ a la aplicación: A continuación, queremos que realices la interacción con la aplicación en función de los requisitos planteados y las funcionalidades de la misma.
Los estudiantes son nombrados de acuerdo con su perfil, y serán empleados con las siguientes abreviaturas: Perfil A (PA), Perfil B (PB), Perfil C (PC).

TU01: Login.

1. Indica el correo electrónico personal para el test.
2. Revisa el correo. Comprueba si ha llegado el usuario/contraseña. ¿Te gusta el formato? ¿Qué te parece el correo? ¿Qué cambiarías? ¿Harías la entrega de credenciales de otra forma? ¿Por qué? ¿Cómo?
3. Intenta entrar con un usuario diferente al entregado. Intenta entrar con el usuario facilitado y otra contraseña. ¿Has probado a iniciar sin usuario y contraseña? ¿Qué ocurre? ¿Está bien especificado cada campo? ¿Te parece sencillo? ¿Qué mensajes propondrías para controlar los errores?

4. No recuerdas tu contraseña. ¿Qué te parece la funcionalidad? ¿Es segura? ¿Ideas de mejora?

TU01: Resultados por estudiante.

- PA: Conclusiones:
    - 1: fmontalbanasesores@gmail.com
    - 2: No cambiaría nada, he recibido el correo correctamente con las credenciales del usuario y la bienvenida. Buen diseño. Me parecen correctas la entrega de contraseña por correo electrónico; muy funcional.
    - 3: Al probar con otro usuario y contraseña, indica un mensaje de error mostrando, usuario o contraseña no son correctos. En blanco no permite avanzar, indica que se inserte una contraseña. Me parece muy sencillo, mensajes cortos, directos y muy claros. No hay confusión.
    - 4: Me parece una buena funcionalidad y segura, ya que llega al correo que tienes registrado. Echo en falta poder cambiar el correo registrado.
- PB: Conclusiones:
    - 1: juan.garc.berm@gmail.com
    - 2: Me gusta, es sencillo. Quitaría el banner y dejaría únicamente el logo oficial de Udima, para un diseño más sencillo y limpio. La entrega es ideal, por correo electrónico, no veo otra forma mejor.
    - 3: Funciona perfectamente, no deja iniciar sin usuario ni con otro usuario incorrecto. Sencillo y mensajes muy claros.
    - 4: Excelente. Quizá el diseño es demasiado brusco. Buscaría algo más elegante y en otro color. En cuanto a funcionalidad, segura, sencilla y ágil.
- PC: Conclusiones:
    - 1: mariadolores.navarro8@murciaeduca.es
    - 2: Si, las instrucciones son muy claras. No cambiaría nada. No haría la entrega de otra forma.

- 3: Da error, usuario o contraseña incorrecta. Los campos están bien identificados y me parece muy intuitivo y visual. Como mensaje para contemplar los errores, pondría "¡Revisa tus datos!".
- 4: Me parece muy útil porque es muy habitual el olvido de contraseñas. Es segura. En el campo de "olvidé mi contraseña", en vez de indicar el usuario, se debería indicar el correo electrónico del usuario; así éste recibirá un correo recordando la contraseña.

TU02: Interfaz Home.

1. Accede al menú principal. ¿Qué te parece el inicio del área privada? ¿Es visual? ¿Qué cambiarías? ¿Por qué?
2. ¿Están todos los elementos importantes del perfil? ¿Echas en falta alguna funcionalidad en el menú que consideres interesante?
3. Intenta cambiar la contraseña. ¿Qué ocurre? ¿Te deja cambiar por una contraseña en blanco? ¿La navegabilidad entre los menús es fluida y sencilla? ¿Te parece útil?

TU02: Resultados por estudiante.

- PA: Conclusiones:
  - 1: En la pantalla central, no me convence mucho la flecha que señala al desplegable. En matricúlate, estaría bien que se escuchara el audio del vídeo.
  - 2: Si, están todos los elementos. En la foto de perfil podría dejar seleccionar desde la galería del propio móvil y subirla.
  - 3: Para cambiar la contraseña estaría bien que te pidiera la antigua. Quizás también una pregunta de seguridad para cambiarla. Por lo demás, es útil.
- PB: Conclusiones:
  - 1: Cambiará toda la interfaz. El color verde no me gusta, provoca que la aplicación esté condicionada a ese color, le resta luminosidad y la apariencia no es elegante. No tiene funcionalidad alguna, aparece una

imagen de “alumnos” que no tiene demasiado sentido, y el estudiante que ha ingresado no dispone de ninguna función interesante al iniciar su área. Pondría un nuevo diseño en blanco, más actual, limpio y con accesos directos a las opciones más usadas. Además, pondría los eventos más relevantes para el estudiante, a modo de recordatorio. En el menú lateral, también haría falta esta sección de inicio, para poder volver a ella, ver los eventos y poder ir a otras secciones. En la parte de matricúlate, el diseño es demasiado simple, con colores que no ayudan y dónde una persona ajena a la universidad, no tiene ningún tipo de opción, simplemente visualiza el video y nada más, la interacción es inexistente y no es llamativo para un futuro estudiante. Falta mostrar ofertas de los grados, másteres y también poder contactar con la universidad en busca de información. Se cambiaría completamente.

    - 2: El diseño vuelve a estar poco logrado, con el mismo color verde que desentona, no aporta. Los iconos no me gustan y pondría el diseño de una forma más cercana y simple. Los datos son los necesarios y principales. No añadiría más.
    - 3: Los botones del cambio de contraseña, no se llevan. Es necesario incorporar una línea, al igual que en el login, para que el diseño sea más limpio. Necesaria mayor seguridad, o bien introduciendo la contraseña antigua o bien con preguntas de seguridad previamente guardadas. El botón atrás no se lleva, el usuario usa el del terminal móvil. Necesario botón para mostrar y ocultar contraseña tanto aquí como en todas las zonas dónde se incorpore una contraseña.
- PC: Conclusiones:
    - 1: Creo que sería más visual con otra tipología de letra y alguna imagen interactiva. Se puede hacer más visual y atractivo al usuario. Cambiaría la tipología de letra, el menú desplegable lo pondría más destacado con algún color diferente al resto y añadiría una imagen interactiva. Haría esos

cambios porque sería todo más visual.

- 2: Los importantes sí. Incorporaría una opción de “mi horario” en el que se confeccione el horario por días y horas según las asignaturas cursadas.
- 3: Añades la contraseña dos veces y se cambia correctamente. No deja cambiar la contraseña en blanco, te indica un mensaje de “introduzca contraseña”. La navegabilidad es muy fluida y me parece útil.

TU03: Interfaz Asignaturas.

Nota aclaratoria: La suma de créditos para el porcentaje del progreso se suman sólo de las asignaturas matriculadas, si no se han sumado ya, y por supuesto, si la nota es igual o mayor que 5. Además, sólo aparecen las asignaturas matriculadas, de forma dinámica.

1. ¿Encuentras el diseño atractivo? ¿Qué cambiarías? ¿Por qué?
2. ¿Qué información relevante acerca de la matrícula no está? ¿Necesitas algún tipo de interacción con esta sección? ¿Cuál? ¿Por qué?
3. Añadimos la asignatura “Prueba de test de usabilidad” con 24 créditos a tu usuario, y con una nota de 4. ¿Qué ocurre? Describe si notas alguna novedad.
4. Cambiamos la nota de la asignatura a 5. ¿Qué ha ocurrido? Descríbelo.

TU03: Resultados por estudiante.

- PA: Conclusiones:
  - 1: Si, el diseño es atractivo, centrado y con los elementos importantes.
  - 2: No, veo todo lo necesario. Asignaturas matriculadas, porcentaje completado en colores corporativos, buena definición.
  - 3: Aparece la asignatura en mis asignaturas, con la nota mostrada. El porcentaje completado sigue igual.
  - 4: Ha cambiado la nota en mis asignaturas, y el porcentaje ha cambiado, sumando los créditos aprobados. Estaría bien que mostrara el número de créditos aprobados y pendientes, no sólo el porcentaje.

- PB: Conclusiones:
    - 1: El diseño en líneas generales está bien, cambiaría de nuevo colores y limpieza en la pantalla completa.
    - 2: Está todo lo importante, no añadiría nada más.
    - 3: Aparece la asignatura, pero no aprecio ningún cambio más.
    - 4: El porcentaje ha cambiado, ha reflejado el cambio. Aun cambiando de menú, no vuelve a sumar el porcentaje. Muy acertado, en colores realmente corporativos y llamativos, con gran identificación del verde al éxito y rojo al fracaso, en relación a los porcentajes.
- PC: Conclusiones:
    - 1: Sí, me gustan los iconos del menú porque son muy intuitivos visualmente, todo muy sencillo de manejar.
    - 2: Las fechas de matriculación para recordar en que año has cursado cada asignatura, independientemente del curso al que pertenezca.
    - 3: No me aparece nada. No veo los cambios.
    - 4: ¿Tengo que hacer algo? Se ve únicamente el cambio en el porcentaje, incrementando los créditos aprobados. Útil.

TU04: Interfaz Mi Calendario.

Nota aclaratoria: Las actividades que aparecen son únicamente actividades asociadas a las asignaturas matriculadas.

1. ¿Qué te parece el calendario? Prueba a moverte sobre él. Comprueba los eventos en relación a la fecha actual. ¿Es todo correcto?
2. ¿Reconoces algún distintivo en las fechas con algún evento? ¿Te parece suficiente? ¿Es fácil de identificar?
3. ¿Si pulsas sobre un evento sin actividad, ¿Te gustaría recibir algún mensaje o que tuviese alguna funcionalidad extra? ¿Cuál? ¿Por qué?
4. ¿Si pulsas sobre un evento con actividad, ¿Qué ocurre? Describe lo que ves. ¿Qué opinas?

5. Introducimos una actividad con fecha del 11/12/2020. ¿Aparece en los eventos? ¿Deberían aparecer los eventos pasados? ¿Por qué?
6. Introducimos una actividad con fecha del 23/01/2021. Describe lo que ocurre ahora.
7. ¿Te falta algún campo importante en los eventos o calendario? ¿Añadirías alguna información adicional o función del calendario? Razona la respuesta.

TU04: Resultados por estudiante.

- PA: Conclusiones:
    - 1: Me parece muy útil y con un gran diseño. Se puede mover fácilmente desde el táctil, y van cambiando los meses. Los eventos aparecen en la parte inferior, y todos con fecha posterior, muy logrado.
    - 2: No, no hay ningún distintivo en las fechas marcadas, no es nada fácil de identificar, tienes que leer la leyenda inferior para poder ir al día y pulsar.
    - 3: No, no tiene sentido. Sólo sería útil en días con alguna actividad, que es lo realmente importante.
    - 4: Al pulsar en el día que existe una actividad, se muestra un mensaje recordatorio en la parte inferior. Muy acertado.
    - 5: Estamos a 10/12/2020. No aparece nada. No aparecen eventos. No es necesario controlar los eventos pasados, no tiene sentido.
    - 6: Aparece el evento en la parte inferior, con la fecha 23/01/2021. Si pulsas en el día exacto, muestra el mensaje. Funciona correctamente.
    - 7: No añadiría nada más, me parece todo correcto.
- PB: Conclusiones:
    - 1: Calendario correcto, con diseño mejorable en cuanto a colores, pero útil, con facilidad para moverse de mes, y cambia perfectamente entre fechas. No hay eventos en la parte inferior.
    - 2: No aparece nada, simplemente el listado de eventos. Añadiría una marca en el día de un color que predomine, y, además, una notificación en el

propio dispositivo que te permita recordarlo en caso de no estar con la aplicación iniciada.

- 3: Si no tiene actividad, no es necesario.
- 4: Aparece un mensaje emergente en la parte inferior. El mensaje se visualiza correctamente y muestra un resumen de la actividad y la fecha. Gran combinación de mensaje y funcionalidad.
- 5: Estamos a 07/12/2020. No aparece nada. Podrían aparecer los eventos pasados con una descripción al pulsar sobre lo ocurrido, notas, apuntas, detalles…, sería muy práctico.
- 6: Ahora si aparece la actividad, al ser de fecha posterior. Es una buena forma de tener únicamente las actividades futuras, aunque sería interesante tener una zona para las pasadas.
- 7: En cuanto a funcionalidad no, cambiaría el diseño y añadiría detalles en las actividades o incluso acceso directo al contenido de las mismas.

- PC: Conclusiones:
  - 1: Muy útil y fácil. Al moverte sobre él, es muy fácil cambiar de mes, pinchar en los días con recordatorio y leer lo que hay en ese día. Todo correcto.
  - 2: Hay un punto rojo, pero no siempre aparece, y apenas se ve. Debería ser más claro.
  - 3: Poder añadir algún comentario; por ejemplo, poder escribir: “no olvidar calculadora científica”.
  - 4: Te sale un recordatorio y también la información escrita fija. Opino que es muy acertado.
  - 5: Estamos a 09/12/2020. No aparece. Sí, debería aparecer en eventos pasados, para comprobar fechas y saber lo que has hecho, como, por ejemplo, los exámenes a los que te has presentado. Sería genial poder poner en verde los exámenes a los que te vas a presentar y en rojo los que no o algo así.

- 6: No sé introducirla, no me sale ninguna opción para hacerlo. Tras incluir el evento en base de datos: Me aparece la asignatura como próximo evento, muy bueno.
- 7: Lo especificado anteriormente.

TU05: Interfaz Mis Profesores.

Nota aclaratoria: Únicamente aparecen los profesores que aparecen asignados como tal en las asignaturas que han sido matriculadas.

1. ¿Te parece buena la idea de que se repitan profesores, aunque sea el mismo en otra asignatura?
2. ¿La información que ves en cada perfil de profesor es útil? ¿A simple vista, notas funcionalidad o no?
3. ¿Te parece útil el modo de visualización si hay más de 3 profesores? ¿Puedes deslizar la pantalla?
4. Se añade como profesor nuevo a la asignatura "Prueba de test de usabilidad" llamado "Puntxi Doctor Honoris Causa". ¿Te aparece en el listado? ¿Te gustaría algún tipo de refresco de la interfaz o es cómoda la sincronización? ¿Puedes contactar con él? Describe el proceso de uso.

TU05: Resultados por estudiante.

- PA: Conclusiones:
  - 1: Si, así se diferencias las asignaturas y posibles funciones sobre las mismas.
  - 2: La información es útil, pero no noto funcionalidad. No aparece la opción de contactar por correo ni llamar al profesor.
  - 3: Si, me gustaría que hubiese un desplegable con todos los profesores y poder bajar y subir.
  - 4: Aparece al salir y entrar de la sección. Es buena la sincronización, no es necesario tener que salir del perfil.

- PB: Conclusiones:
    - 1: Quizá cambiaría el diseño para que aparecieran los profesores y en cada área, las asignaturas que tiene impartiendo, en plan listado. Así aprovecharíamos los campos de nombre, correo y contraseña que serían los mismos y no tiene lógica repetirlos.
    - 2: Me parecen los campos necesarios, pero la funcionalidad está demasiado oculta. Tienes que pulsar justo en el icono para que funcione.
    - 3: Si, pero lo haría en disposición horizontal en caso de repetir profesor, en un mismo enfoque.
    - 4: En el momento no aparece. Al cambiar de sección y volver sí, aunque estaría bien un botón tipo "Actualizar" o similar, para recargar la información. El diseño puede mejorar.
- PC: Conclusiones:
    - 1: Si, para evitar confusiones.
    - 2: Me sale en blanco.
    - 3: Si, aunque me harían falta más profesores para ver la funcionalidad correcta.
    - 4: Aparece tras salir y entrar. Está bien, aunque se requiere botón de enviar correo al profesor, no me aparece.

TU06: Interfaz Contacto.

Nota aclaratoria: La petición de contacto se mandará al correo de soporte registrado de la aplicación y además una copia al correo de vuestro usuario.

1. ¿Echas de menos especificar la prioridad? ¿Por qué? ¿Es importante? ¿Cómo lo pondrías en el actual diseño?
2. ¿Es necesario algún campo más? ¿El espacio para el texto es suficiente?
3. ¿Funciona el botón "Restablecer"? ¿Qué te parece su función? ¿Tendrías en cuenta alguna condición o crees necesario otro botón con alguna otra función nueva?

4. ¿Echas en falta tu nombre de usuario al enviar la petición? ¿Por qué?
5. Haz una prueba de envío de petición. ¿Te ha llegado algún correo electrónico? ¿Qué te parece?

TU06: Resultados por estudiante.

- PA: Conclusiones:
    - 1: No es necesario, el estudiante siempre pondría prioridad alta, y no sería real.
    - 2: No, es lo necesario, no pondría ningún campo más.
    - 3: Muy buena, muy útil cuando quieres borrar todo de golpe y no quieres ir línea a línea.
    - 4: No, ya que estás dentro del usuario, aunque sí debe llegar al destino. El problema es que no tengo muy claro a quién le llegaría este correo, al poner "Contacto".
    - 5: Funciona perfectamente. Llega a mi correo, con el detalle del título, texto, el correo electrónico, usuario y desde dónde se envía. Completo y muy funcional. El diseño también es bueno.
- PB: Conclusiones:
    - 1: Pondría un desplegable con la prioridad, y un diseño sin tanto icono, y más limpio.
    - 2: Además de la prioridad, pondría un campo dónde marcar si quieres recibir una copia o no del correo, e incluso poder añadir una imagen.
    - 3: Quitaría este botón. No se usa, pondría "Salir sin guardar", y que me llevara a la página inicial.
    - 4: No, no es necesario mi nombre, ya que estoy dentro de mi ficha.
    - 5: Es funcional. El diseño es como el de bienvenida, con un banner en color, y no demasiado visual. El texto que se recibe es el correcto, y, además, los campos son precisos y muy correctos.

- PC: Conclusiones:
    - 1: No.
    - 2: No, es suficiente.
    - 3: Si, funciona. Me parece muy útil para ahorrar tiempo en borrar todo si es necesario. No añadiría más botones.
    - 4: No es necesario porque estás dentro de tu perfil.
    - 5: Bien el diseño, con el logo, asunto, etc.

TU07: Navegación menús.

Nota aclaratoria: Este test de usabilidad es únicamente para comprobar la navegabilidad entre todas las actividades y menús de la aplicación.

1. ¿Qué te parecen los iconos para navegar por la aplicación? ¿Describen la finalidad de cada uno de ellos? ¿Llevan a confusión?
2. ¿Las ventanas de "cerrar sesión del usuario" y "salir de la aplicación" son intuitivas? ¿Qué cambiarías?
3. ¿El menú es práctico? ¿Mejorarías algún aspecto? ¿Echas de menos el botón "home" en algún lugar?
4. Reflexión: ¿Echarías en falta alguna sección que consideras importante para el estudiante en su día a día? ¿Cambiarías algún aspecto general de diseño?
5. ¿Qué te parece la bienvenida a la Universidad al inicio, al pulsar "Matricúlate", ¿La ves completa? ¿Qué se te ocurre para atraer a futuros potenciales estudiantes?
6. ¿Qué valoración le darías a la aplicación (0-10)? Confecciona una lista de cambios/mejoras importantes bajo tu punto de vista.

TU07: Resultados por estudiante.

- PA: Conclusiones:
    - 1: Están bien, muy claros. El problema es que, si pulsas atrás con el botón del móvil, te saca de tu zona privada. No son confusos.
    - 2: Me parecen sencillas y correctas. No las cambiaría.

    - 3: El menú es práctico, cómodo y visual, no es necesario nada más, cumple su función.
    - 4: Echo de menos una sección con el contenido: prácticas, temarios y demás de las asignaturas. El diseño en líneas generales me parece correcto.
    - 5: Como indiqué anteriormente, el video de matricúlate no se escucha. Por lo demás, lo veo bien.
    - 6: La nota es de 8. Es muy completa, y las únicas mejoras serían las mencionadas, no más.
- PB: Conclusiones:
    - 1: Quizá quitaría bastantes botones, como el de "Home", botón atrás, "Restablecer". La usabilidad actual implica que el usuario ya no usa los botones, sino que se mueve con los botones del terminal móvil.
    - 2: Cambiaría el diseño y los botones. Nada de redondeados y, además, los colores corporativos no pegan en estas ventanas.
    - 3: Quizá un menú inferior sin desplegable es más cómodo, pero cumple con la función.
    - 4: Echo de menos poder matricularse directamente y pagar con el móvil, aunque quizá es demasiado para la primera versión.
    - 5: Definido anteriormente. No es nada intuitiva ni marca al usuario que tiene curiosidad por la universidad. Falta atraer al usuario, falta poner los planes formativos, las ofertas, contacto, y demás cercanía para que se sientan escuchados y, sobre todo, que les motive. El diseño lo cambiaría completo.
    - 6: La nota sería de 8. Las funcionalidades son excelentes, y lo único que cambiaría sería un nuevo enfoque de diseño y la usabilidad general de la app.
- PC: Conclusiones:
    - 1: No da pie a confusión, totalmente destinados a su finalidad y los iconos muy claros y concisos.

- 2: Sí, no cambiaría nada.
- 3: Muy práctico. No nada. No.
- 4: Sí, la del horario. Quizás también un apartado para documentos, en el que profesores o alumnos puedan subir presentaciones, etc.
- 5: Me gusta. Sí. Para atraer a futuros estudiantes añadiría una navegación de prueba por un perfil modelo. Y también, mensajes atractivos que le hagan entrar en ella.
- 6: Le pondría un 9, porque para mi gusto es muy completa, visual, clara, intuitiva y útil. Añadiría los cambios y mejoras indicados anteriormente en las diferentes preguntas. Son detalles personales que me gustaría que apareciesen.

TU08: Interfaz sistema Udima App.

Nota aclaratoria: Debes realizar un recorrido por toda la aplicación. Tras finalizar, contesta a las siguientes preguntas:

1. ¿Cómo calificarías el diseño? ¿Qué cambiarías?
2. ¿Es necesaria alguna sección más? ¿Qué importancia tiene? ¿Por qué?
3. ¿Echas de menos alguna funcionalidad o dato adicional en las secciones ya existentes?
4. Dinos una frase con la que describirías la aplicación.

TU08: Resultados por estudiante.

- PA: Conclusiones:
    - 1: El diseño está bien en líneas generales. Quizá cambiaría el menú, porque los colores no contrastan demasiado bien con las letras.
    - 2: La sección de contenido, ya que es necesario tener el contenido de las asignaturas, para leerlo desde el móvil, por ejemplo. Me gustaría que aparecieran todas las asignaturas y al pulsar en cada una de ellas, te llevara al contenido de la propia.

    - 3: En principio nada más.
    - 4: Apoyo en la complicada vida del estudiante a distancia.
- PB: Conclusiones:
    - 1: Cambiaría el diseño totalmente. Mantenemos el color rojo corporativo para el fondo inicial. Los botones, siguiendo la tendencia actual del diseño UX/UI, son rectangulares con las esquinas ligeramente redondeadas. Estará colocados en la parte inferior para facilitar el acceso con una sola mano y sin tener que soltar el móvil. Propondría un diseño sencillo, minimalista e intuitivo según las tendencias UX/UI. Haría una llamada de atención a los botones importantes (tanto por su diseño como por el lenguaje empleado). Respetaría la identidad visual de UDIMA. Siempre, centrado en el usuario final, en este caso, el objetivo principal son los estudiantes que ya están matriculados.
    - 2: En cuanto a secciones, veo todas las necesarias y las funcionalidades están realmente conseguidas.
    - 3: Algunos retoques como los mencionados anteriormente, pero a excepción del diseño, todo correcto.
    - 4: Conocimiento móvil en beneficio de los estudiantes.
- PC: Conclusiones:
    - 1: Tal y como he dicho, el diseño me parece bonito y sencillo. No cambiaría nada.
    - 2: En cuanto a secciones, las mencionadas, tanto del contenido, como del horario, eventos pasados, exámenes para hacer y marcar verde/rojo…
    - 3: Las mencionadas a lo largo de los test.
    - 4: Herramienta básica y necesaria para focalizar la educación a distancia actual online.

Tras los test de usabilidad, y con la evolución de los cambios propuestos, se muestran los cambios de diseño, de forma gráfica, puntualizando las funcionalidades finales.

Anexo A2. Diseño aplicación.

Anexo A2.1 Cambios finales de diseño tras TU08:

Nota inicial: Para reflejar el proceso de evolución centrado en el usuario, y el dinamismo de un proyecto ágil de forma real, y tal como ha ocurrido a lo largo de este proyecto, las imágenes del diseño en los diferentes apartados posteriores podrán cambiar en función del punto exacto del proyecto, de ahí, este diseño tan dinámico. En este apartado, se presentan los cambios finales del diseño, ya mostrados de forma general en el mapa de Navegabilidad (Final).

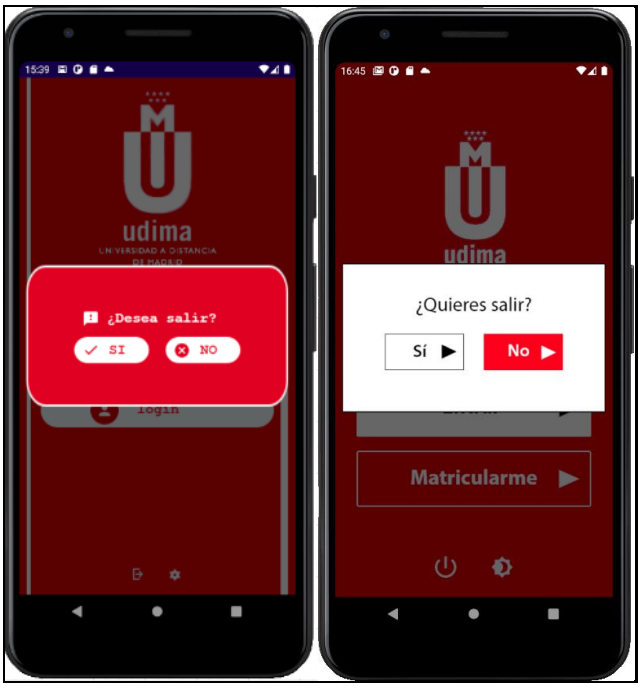
Anexo A2.1.1 Interfaz Principal:

- Evolución diseño (gráfico):

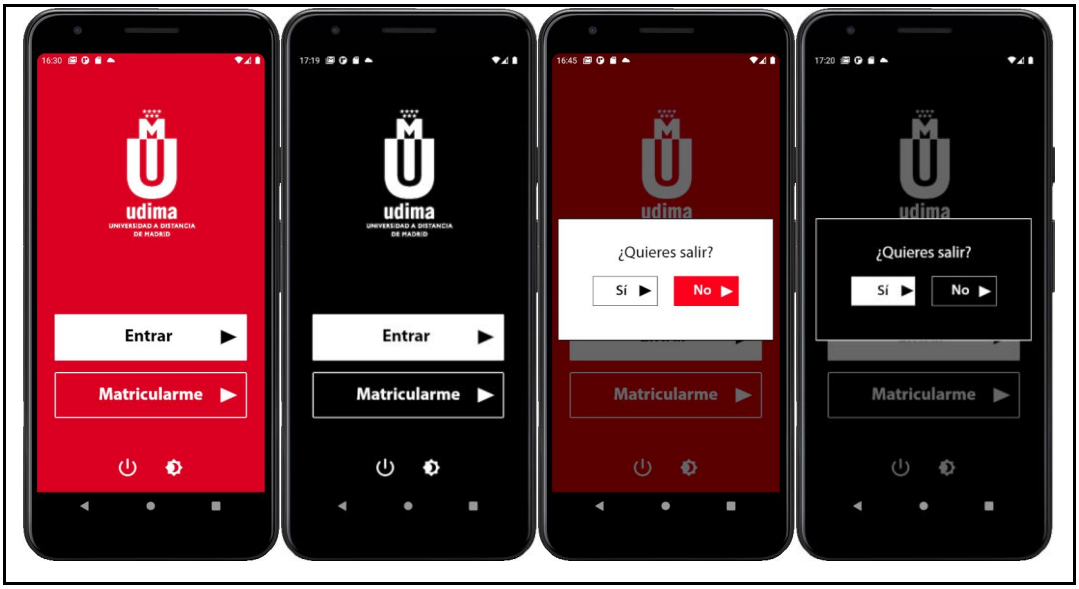
- Evolución diseño (detallado):
    - Eliminadas líneas verticales: Generaban aspecto estrecho y deteriorado.
    - Color barra de notificaciones: Color corporativo, mejor imagen.
    - Logo principal: Más pequeño, mejor definido y centrado.

- Botones: Nuevas características:
  - Tamaño: Más grande, facilitando la acción del usuario.
  - Forma: Cuadrada, siguiendo tendencia UX/UI actual, diseño minimalista, limpio y elegante.
  - Iconos: Los iconos variados a lo largo de los botones, causan un aspecto pobre y poco clarificante.
  - Colores: Se capta la atención del usuario en el botón principal, "Entrar", de ahí color blanco que resalta, y un pequeño triángulo como "llamada de acción".
  - Posición: Colocados en la parte inferior para facilitar el acceso con una sola mano y sin tener que soltar el móvil.
- Iconos parte inferior: Se añaden iconos más claros, para las nuevas funcionalidades, salir de la aplicación y modo noche.

- Nuevas funcionalidades:
  - Salir: Permite mostrar un dialog con la posibilidad de salir o no de la aplicación de forma definitiva (Cambio diseño limpio, en blanco, con botones también cuadrados siguiendo la misma línea anterior:

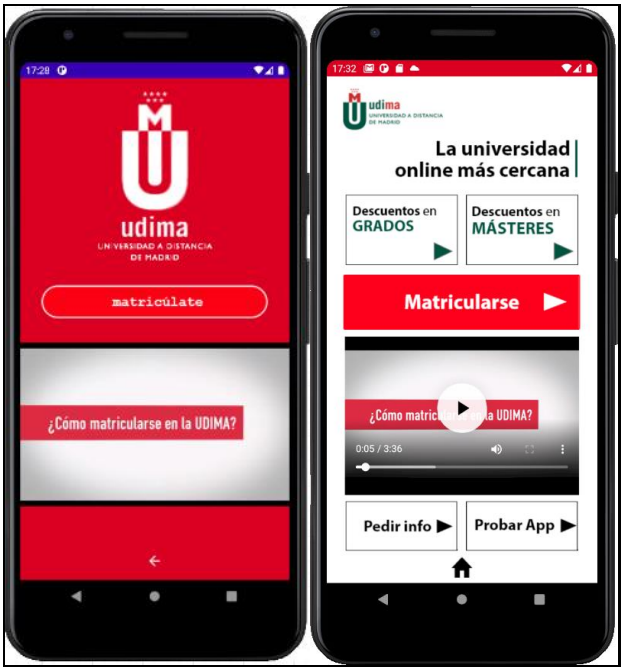


- Modo noche: Permite al usuario alternar entre el modo estándar y un modo mejor adaptado a la luminosidad, con un diseño basado principalmente en el color negro dominante y destellos claros para facilitar el seguimiento:

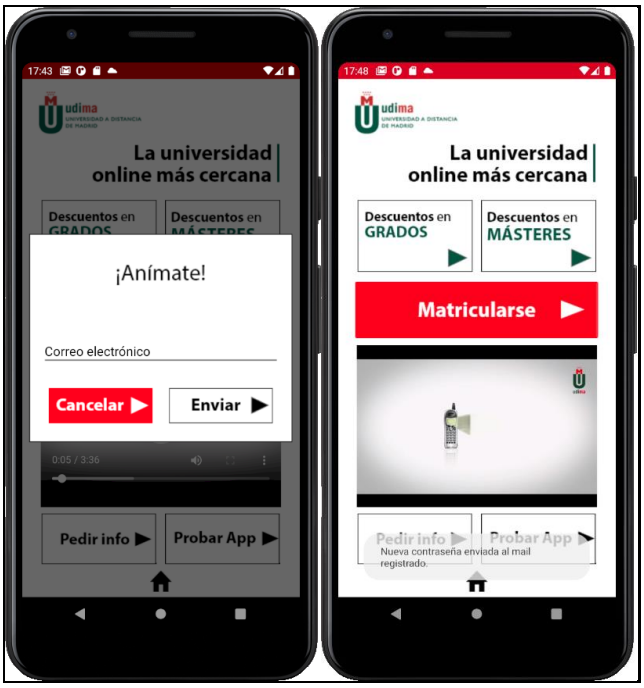


Anexo A2.1.2 Interfaz Matricúlate:

- Evolución diseño (gráfico):

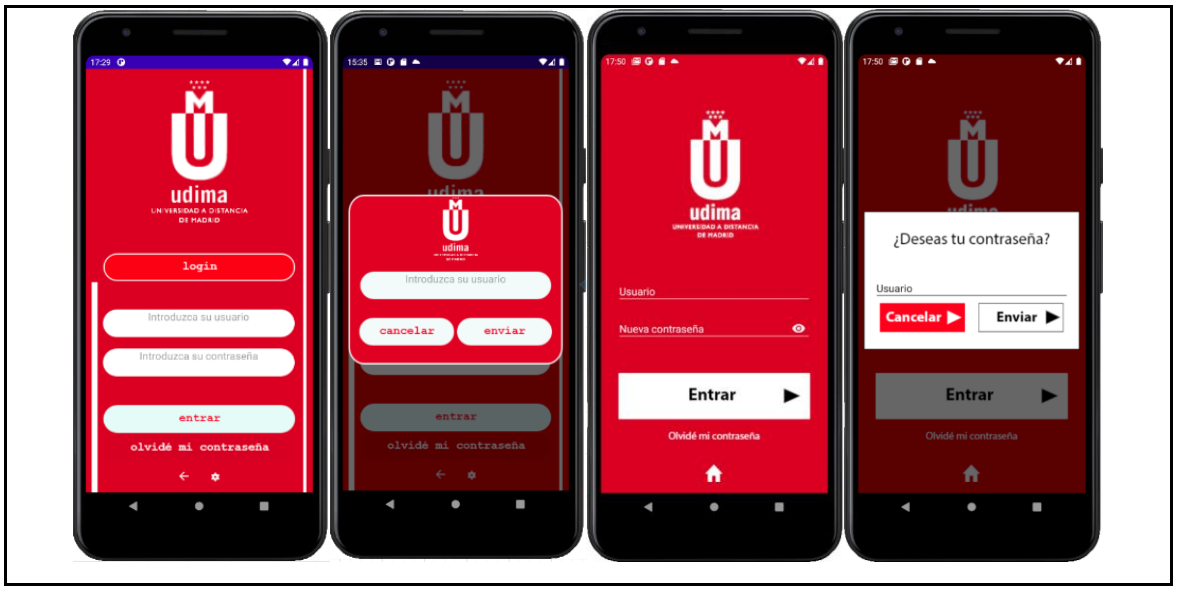


- Evolución diseño (detallado):
    - Aspecto: El color blanco resalta la sensación de amplitud, de transparencia y cercanía. Se establece eslogan principal de la Universidad. Se sigue la tencdencia de botones cuadrados con triángulo en forma de llamada de acción.
    - Navegabilidad: Icono home más representativo.
- Nuevas funcionalidades:
    - Botón Grados y Másteres: Abren una pestaña del navegador del dispositivo con las ofertas de dichos planes por parte de Udima.
    - Pedir info: Abre el cliente de correo electrónico con la dirección concreta de Udima para el contacto en busca de información.
    - Vídeo corporativo: Se añade sonido al mismo, para mayor realidad.
    - Probar App: El usuario introduce su correo electrónico y recibirá las credenciales de un usuario temporal para poder probar los menús internos de la aplicación:

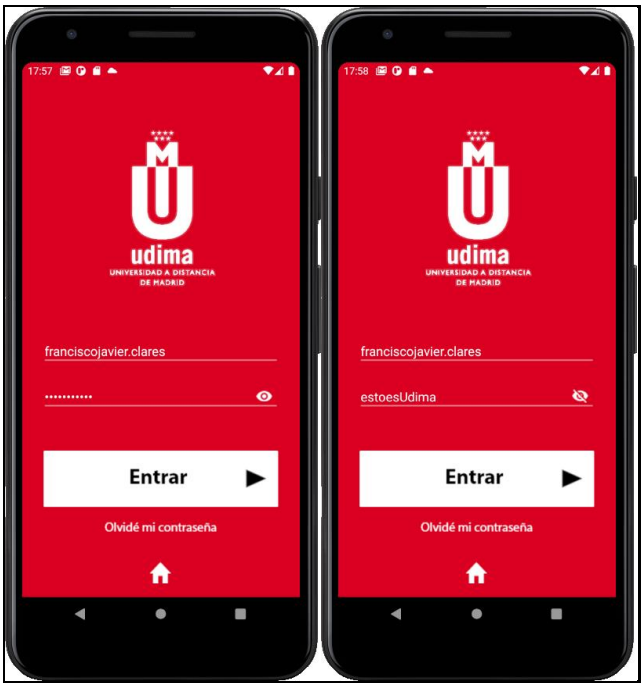


Anexo A2.1.3 Interfaz Login:

- Evolución diseño (gráfico):

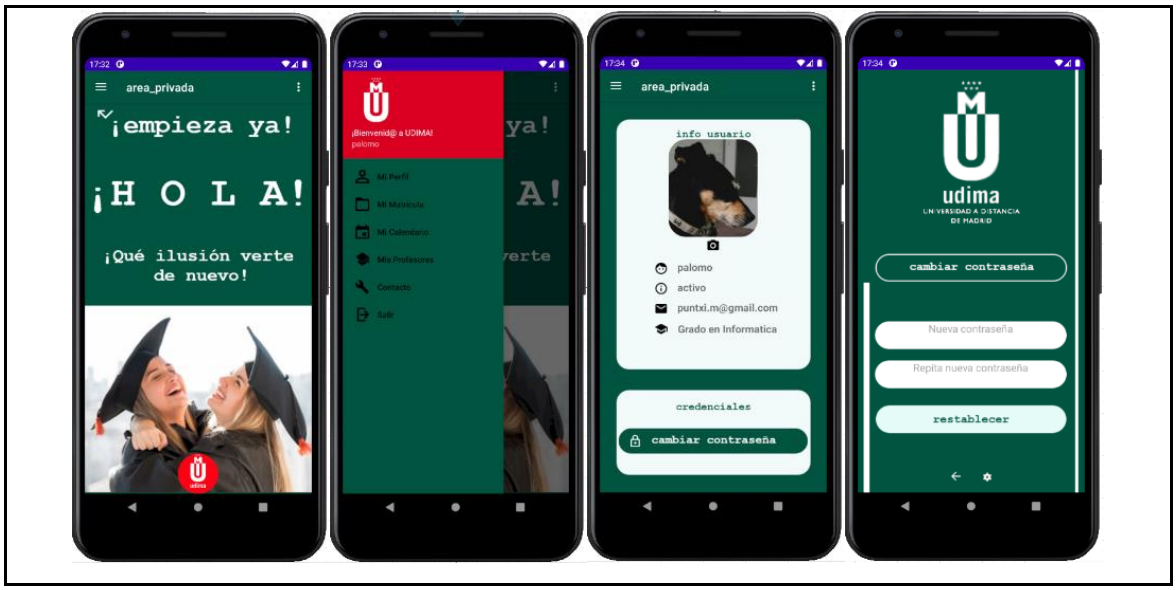


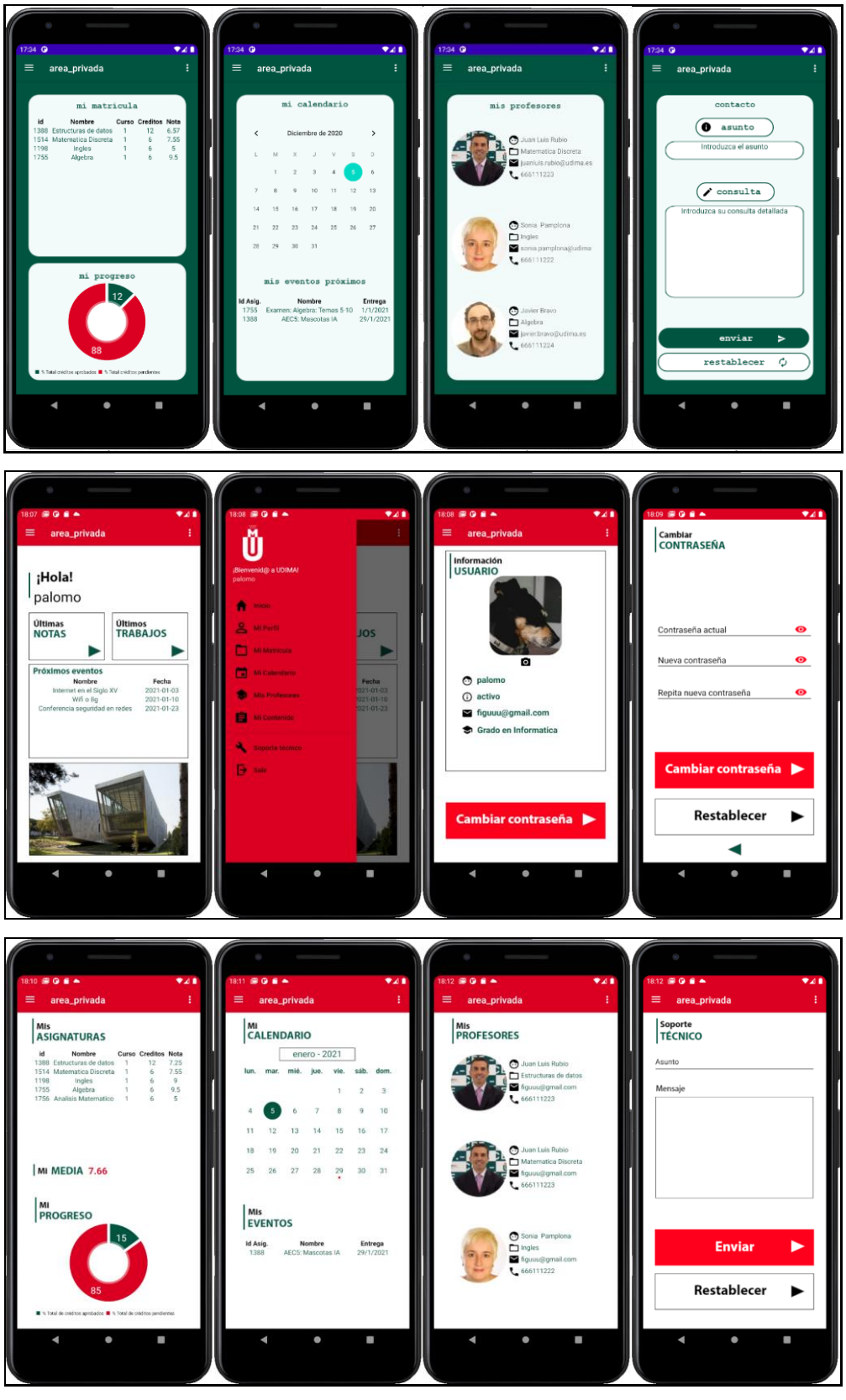
- Evolución diseño (detallado):
    - Aspecto: Diseño minimalista, eliminando botones de entrada de texto por línea sencilla. Botón “Entrar” de mayor tamaño y diferente color para mostrar la llamada de atención principal. Texto “olvidé mi contraseña” mejor definido y con un dialog en blanco, resaltando los botones cuadrados con las funcionalidades, evitando bordes redondeados. Se elimina el título “login” ya que no añade nada.
    - Navegabilidad: Únicamente botón Home para volver al inicio. Se elimina el icono “Ajustes”, mostrando una interfaz con lo realmente importante, eliminando detalles que resten.
- Nuevas funcionalidades:
    - Se añade icono ojo “mostrar/ocultar” contraseña, útil para mejorar la usabilidad en los posibles fallos de entrada de texto.



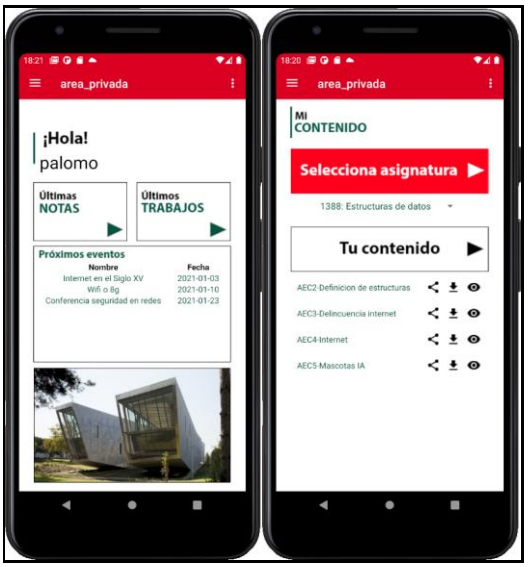
Anexo A2.1.4 Área privada:

- Evolución diseño (gráfico):

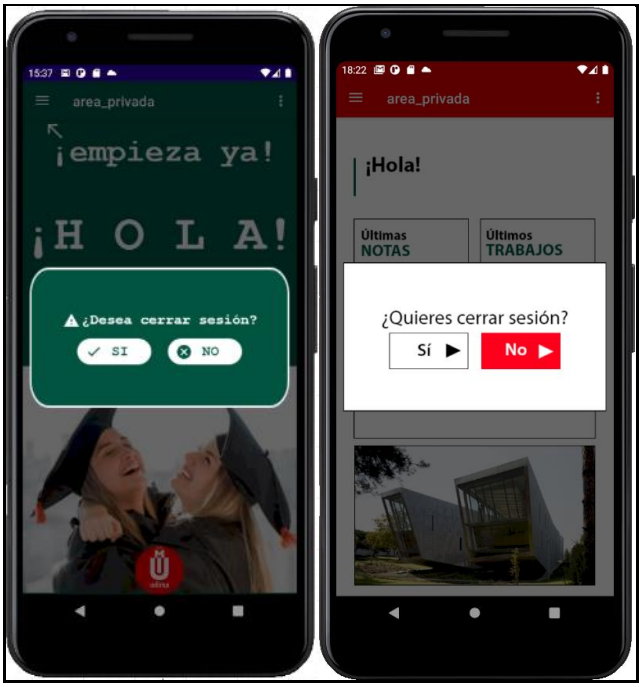



area_privada
mi matricula
mi calendario
mis profesores
contacto
asunto
consulta
enviar
restablecer
¡Hola!
palomo
Últimas NOTAS
Últimos TRABAJOS
Próximos eventos
Información USUARIO
Cambiar CONTRASEÑA
Contraseña actual
Nueva contraseña
Repita nueva contraseña
Cambiar contraseña
Restablecer
Mis ASIGNATURAS
Mi MEDIA 7.66
Mi PROGRESO
Mi CALENDARIO
Mis EVENTOS
Mis PROFESORES
Soporte TÉCNICO
Asunto
Mensaje
Enviar

- Evolución diseño (detallado):
    - Aspecto: Nuevo aspecto alternando color blanco principal y rojo corporativo. Letras en color verde (Myriad Pro), haciendo guiño al otro color de referencia. Eliminados logos redundantes que ofrecían una opacidad a la funcionalidad. El área de inicio, se le agrega funcionalidad, y un diseño marcado por la imagen de la universidad, sencillo pero que aporte al usuario.
    - Navegabilidad: El cambio de diseño provoca una mejor navegabilidad, impulsada por los campos de texto más amplios y clarificados, al eliminar botones rellenos inservibles.
- Nuevas funcionalidades:
    - Sección Inicio: Contiene accesos directos a los paneles de “mi matrícula” y “mi contenido”. Se añade una tabla dinámica con eventos generales de la universidad, mostrados a todos los alumnos y lanzados desde la base de datos. Se saluda al usuario con su nombre al comenzar.
    - Sección “mi contenido”: Sección que lista las asignaturas matriculadas, y al pulsar en cada asignatura, muestra una tabla dinámica con el contenido: Actividades, temario, con las opciones de visualizar pdf, descargar y compartir.



- Ampliación diseño: Mostrar dialog cerrar sesión:

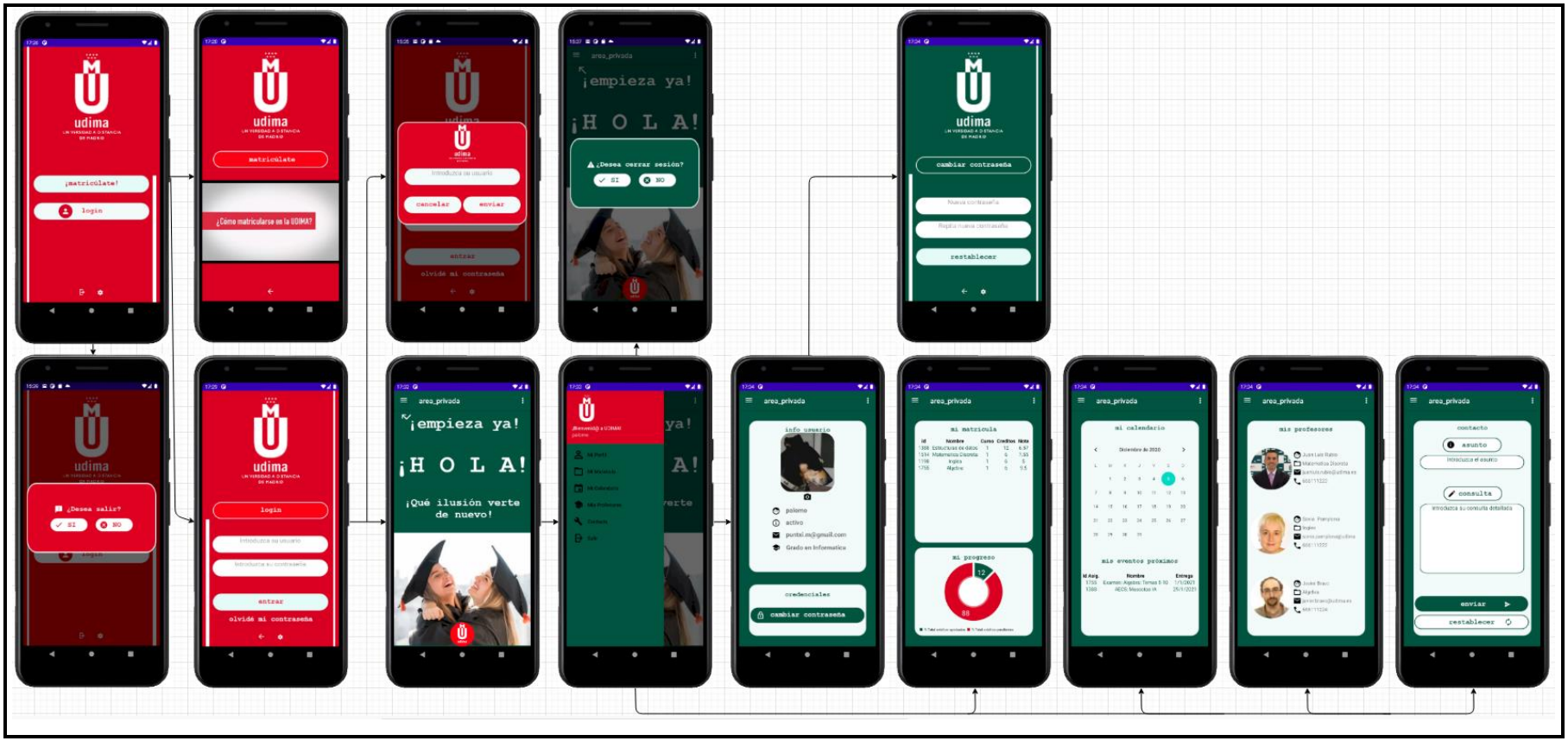


A modo de conclusión de diseño, valoramos de forma global las siguientes mejoras:

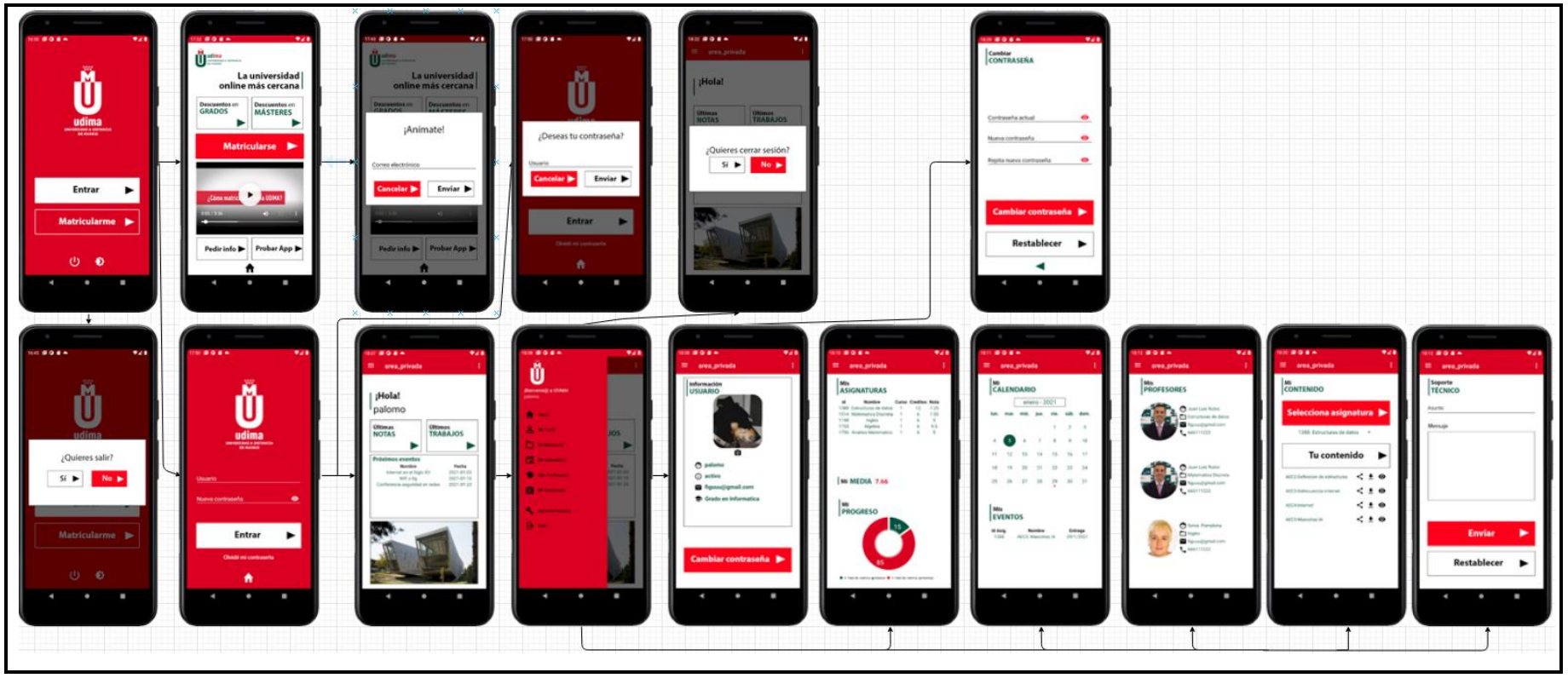
- Diseño sencillo, minimalista e intuitivo siguiente las tendencias en UX/UI actuales.
- Llamada a la acción de los botones importantes (tanto por su diseño como por el lenguaje empleado).
- Se respeta la Identidad Visual de Udima.
- Siempre centrado en el usuario final, en este caso estudiantes que ya están matriculados.
- Botones siempre en lugares estratégicos, cercanos a la parte inferior de la pantalla, para poder ofrecer funcionalidad cómoda en una mano al usuario.
- Guiños a la visión de UDIMA: Frase principal, logo corporativo, identidad…

A continuación, se muestran tres niveles de mapas de navegabilidad, el inicial durante el ciclo de vida (antes de TU08), final (después de TU08) y modo noche.

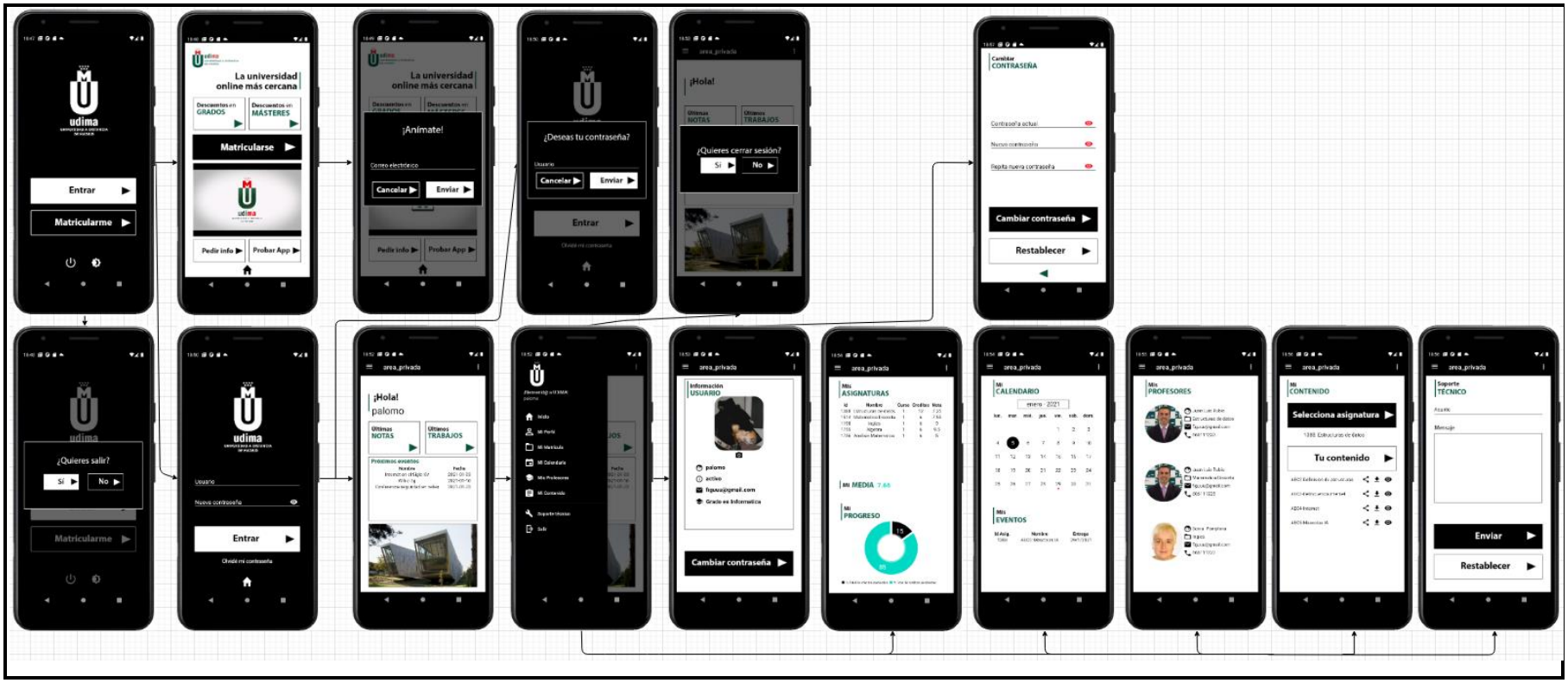
Anexo A2.2 Mapa de navegabilidad (Ciclo de vida).



Anexo A2.3 Mapa de navegabilidad (Final).



Anexo A2.4 Mapa de navegabilidad (Modo noche).

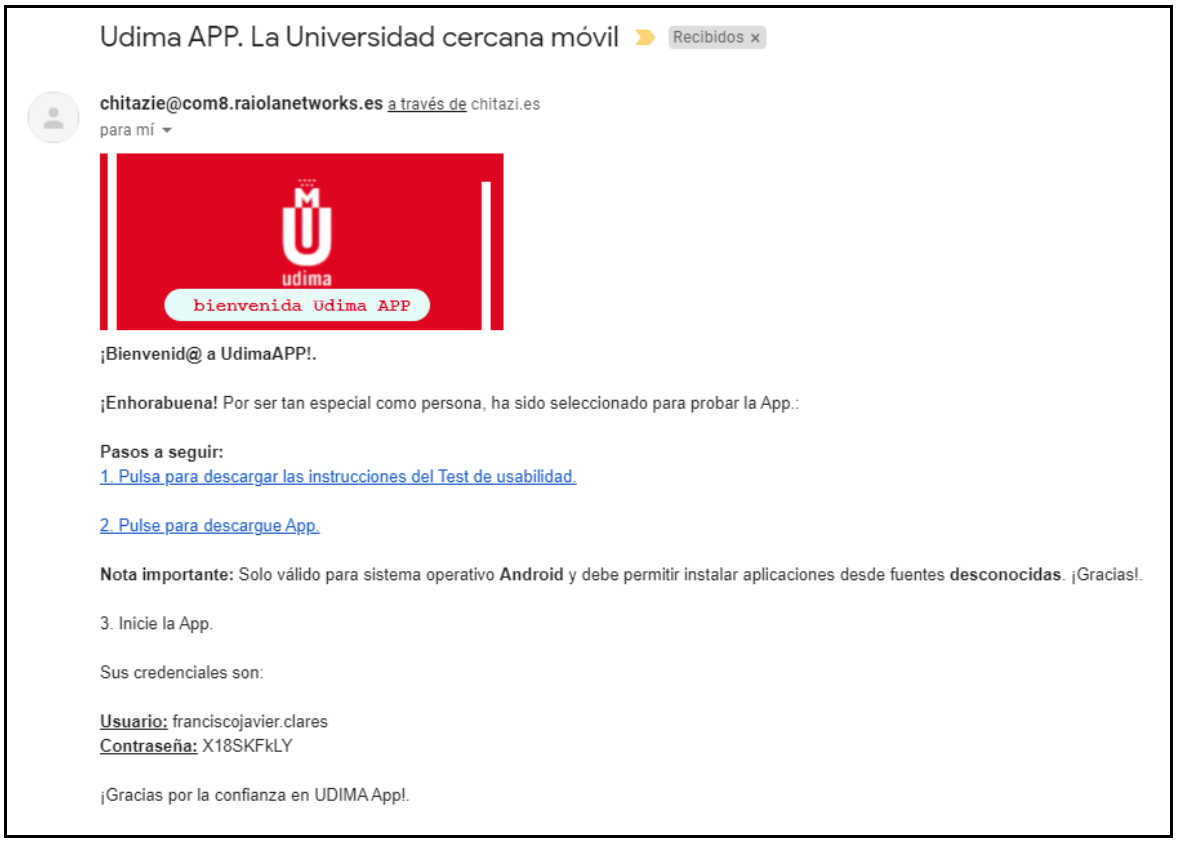


Anexo A3. Comunicación externa estudiante/aplicación.

Se muestra el detalle gráfico de las notificaciones que reciben los estudiantes:

- **Bienvenida a la aplicación:**

  - Notificación correo:



  - Archivos adjuntos:

Reciben el siguiente documento para realizar el test de usabilidad:

https://chitazi.es/udimaApp/documentos/udimaApp_Test_Usabilidad.pdf

Aplicación para su instalación (No incluida en Play Store por no ser oficial)

https://chitazi.es/udimaApp/appDescarga/udimaApp.apk

- **Olvidé mi contraseña:**

- **Petición de contacto:**

- **Probar la App (Usuario temporal):**

Anexo A4. Pruebas de aceptación.

Se muestran las pruebas realizadas por el estudiante en la validación y verificación final de la aplicación:

A4.1: Prueba de aceptación PA01

- Descripción de la prueba:

PA01	
Descripción	Mostrar recordatorio del evento en calendario.
Pre-condición	Estudiante logueado en aplicación.
Escenario	1. Añadir evento en fecha (28/01/2021) 2. Comprobar evento en calendario. 3. Deslizar hasta encontrar día (28/01/2021) 4. Pulsar en fecha encontrada.
Post-condición	OK: Se muestra mensaje emergente del evento en parte inferior

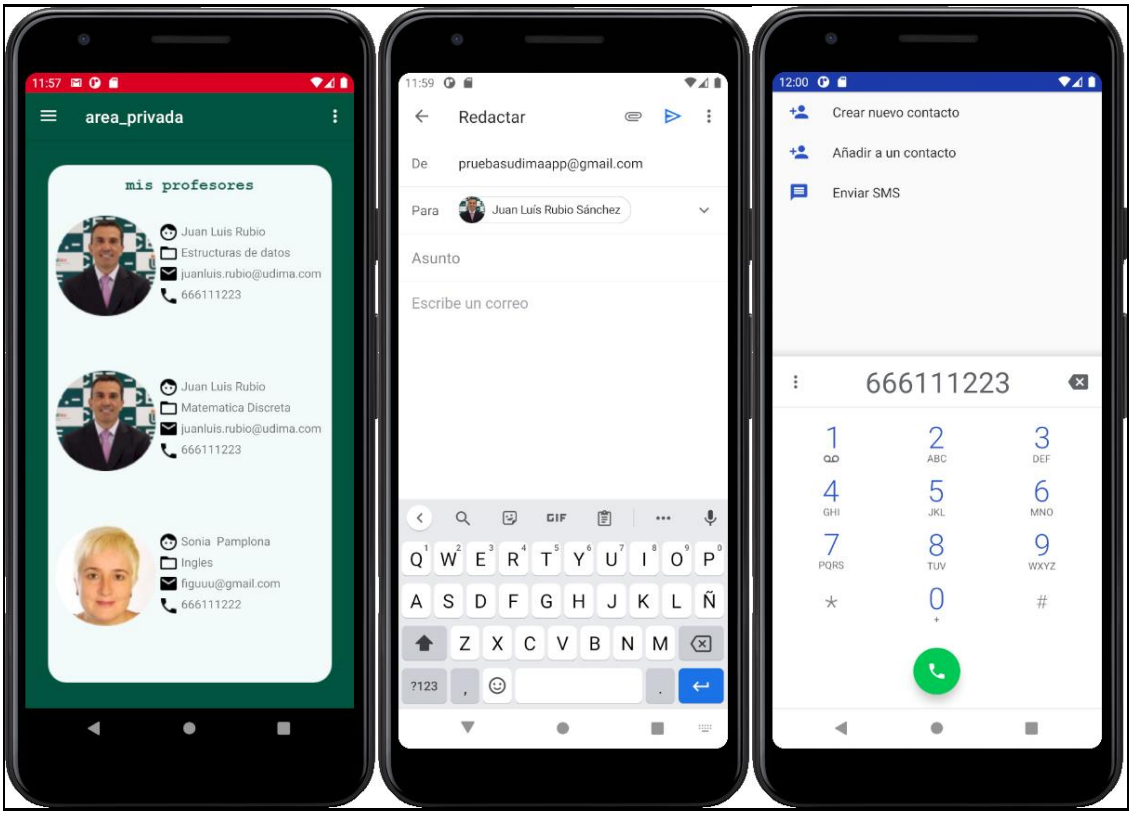
- Resultado gráfico:

A4.2: Prueba de aceptación PA02

- Descripción de la prueba:

PA02	
Descripción	Contactar con profesor específico de asignatura matriculada.
Pre-condición	Estudiante dispone de asignaturas matriculadas.
Escenario	1. Cargar profesores en perfil. 2. Pulsar sobre correo electrónico. 3. Pulsar sobre teléfono.
Post-condición	OK: Se muestran las aplicaciones nativas de contacto (Correo + llamada)

- Resultado gráfico:



A4.3: Prueba de aceptación PA03

- Descripción de la prueba:

PA03	
Descripción	Añadir asignatura aprobada.
Pre-condición	Estudiante dispone de asignaturas matriculadas.
Escenario	1. Cargar asignaturas en perfil. 2. Comprobar progreso actual. 3. Añadir asignatura con nota 5. 4. Comprobar progreso actual.
Post-condición	OK: Se incrementa el progreso con la suma de créditos de la asignatura.

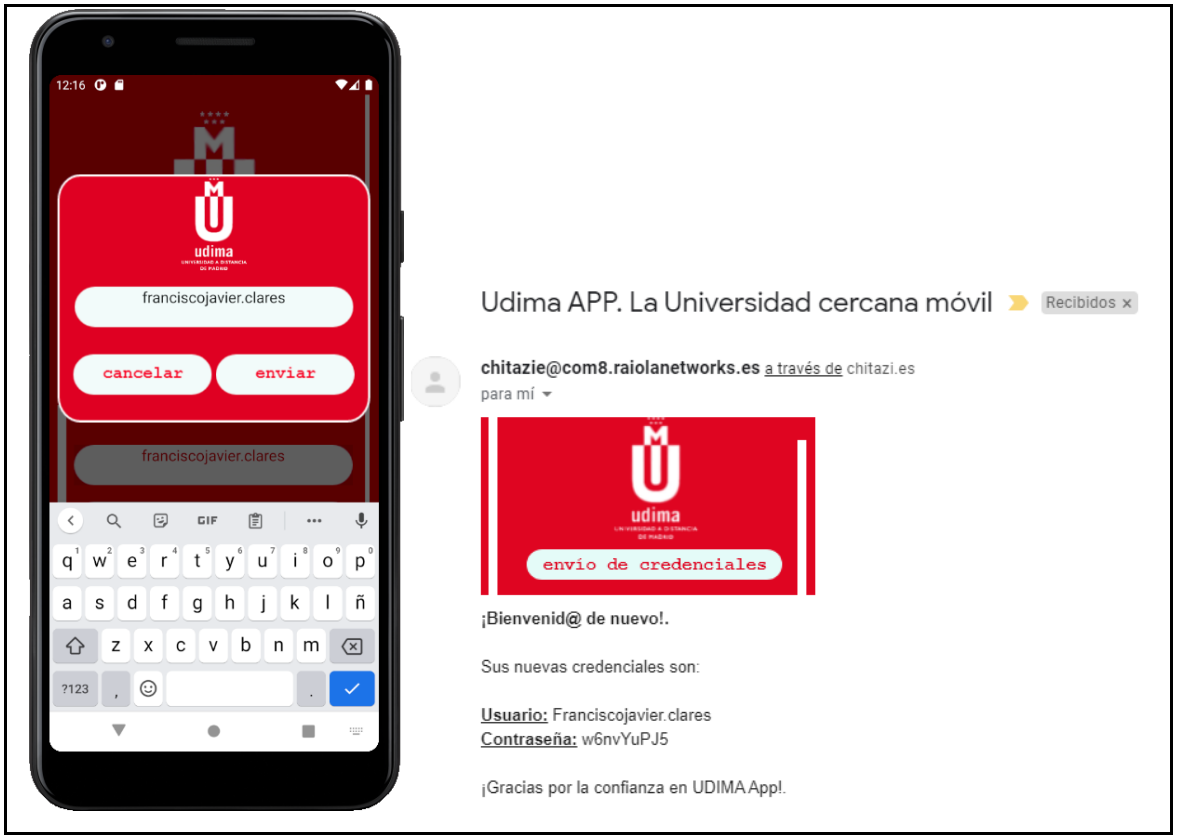
- Resultado gráfico:

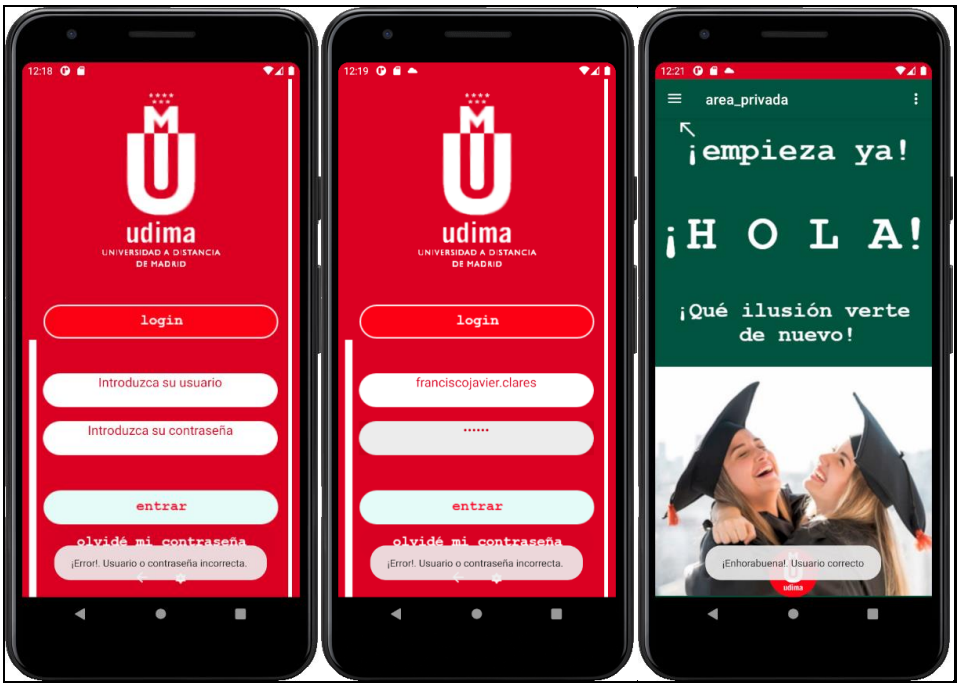
A4.4: Prueba de aceptación PA04

- Descripción de la prueba:

PA04	
Descripción	Comprobar login correcto.
Pre-condición	Estudiante dispone de correo de notificación de credenciales.
Escenario	1. Olvidé mi contraseña. 2. Recibir credenciales: franciscojavier.clares / w6nvYuPJ5 3. Introducir usuario / contraseña en blanco. 4. Introducir credenciales erróneas: franciscojavier.clares / prueba 5. Introducir credenciales correctas: franciscojavier.clares / w6nvYuPJ5
Post-condición	OK: Sólo acepta las credenciales correctas, en los demás casos ERROR.

- Resultado gráfico:



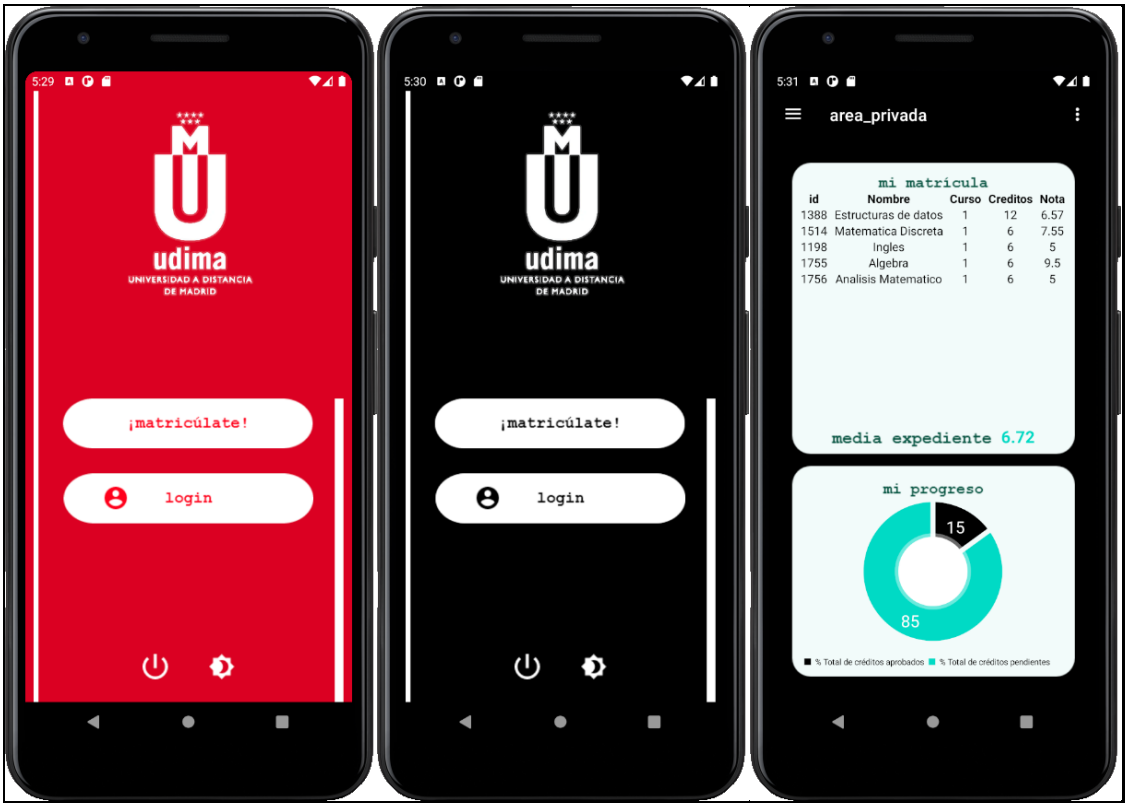


A4.5: Prueba de aceptación PA05

- Descripción de la prueba:

PA05	
Descripción	Activar modo noche.
Pre-condición	Estudiante dispone de la última actualización de la APP.
Escenario	1. Pulsar icono modo noche. 2. Comprobar diseño secciones. 3. Pulsar icono modo noche. 4. Comprobar diseño secciones.
Post-condición	OK: El diseño se alterna según el modo contrario al actual.

- Resultado gráfico:

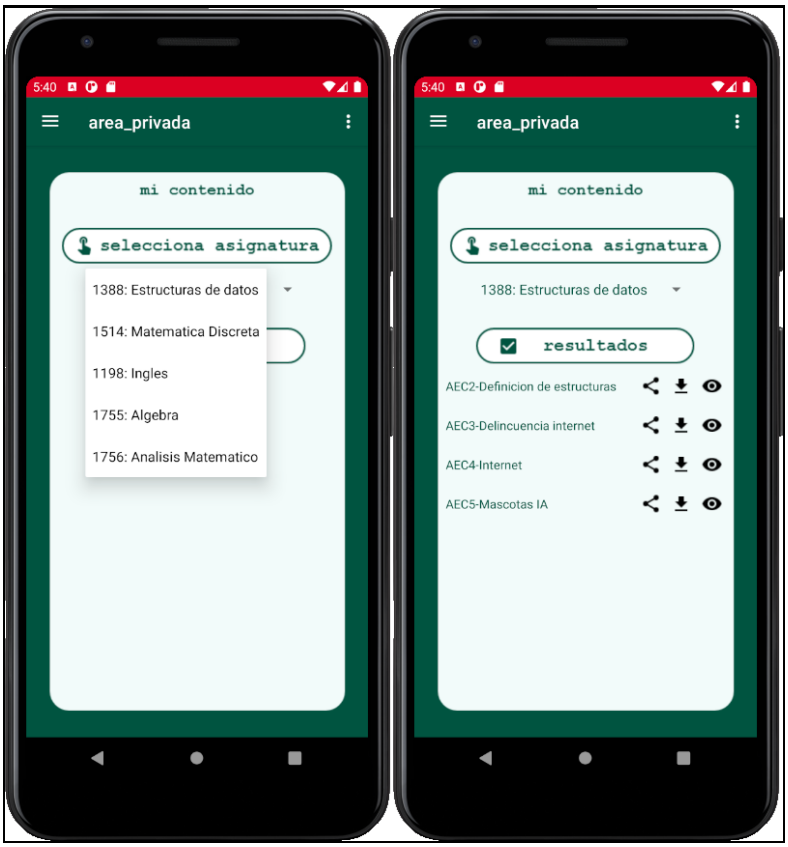


A4.6: Prueba de aceptación PA06

- Descripción de la prueba:

PA06	
Descripción	Comprobar contenido en lista de asignaturas.
Pre-condición	Estudiante dispone asignaturas matriculadas.
Escenario	1. Pulsar en "Mi contenido". 2. Pulsar en "Selecciona Asignatura" 3. Comprobar listado de asignaturas. 4. Comprobar lista de contenido asignatura "Estructuras de datos".
Post-condición	OK: Se muestras todas las asignaturas matriculadas. OK: Se muestran el contenido de la asignatura.

- Resultado gráfico:

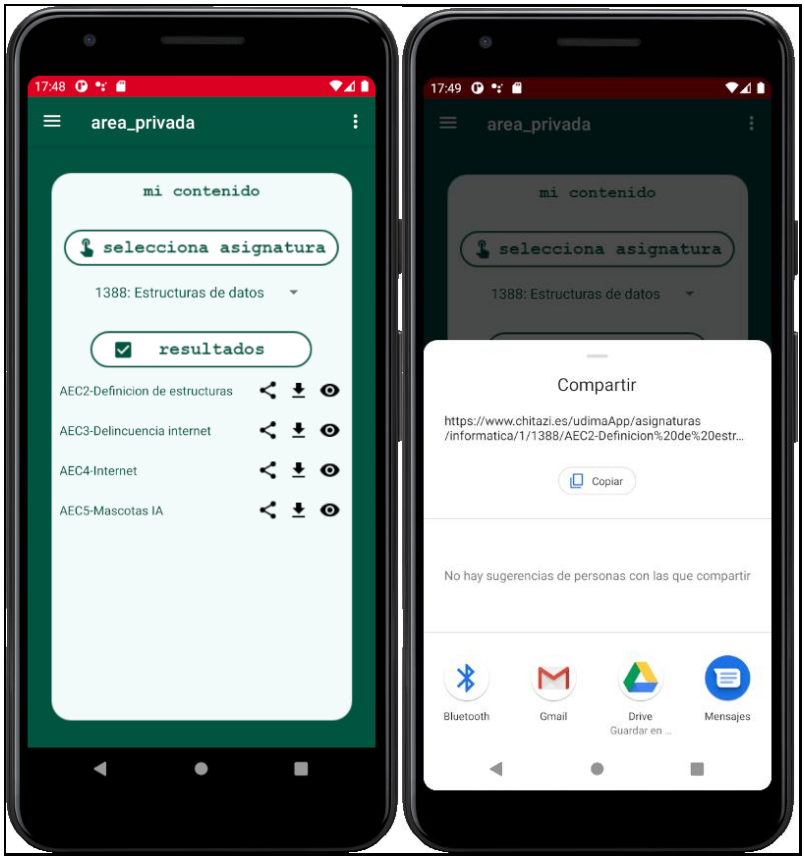


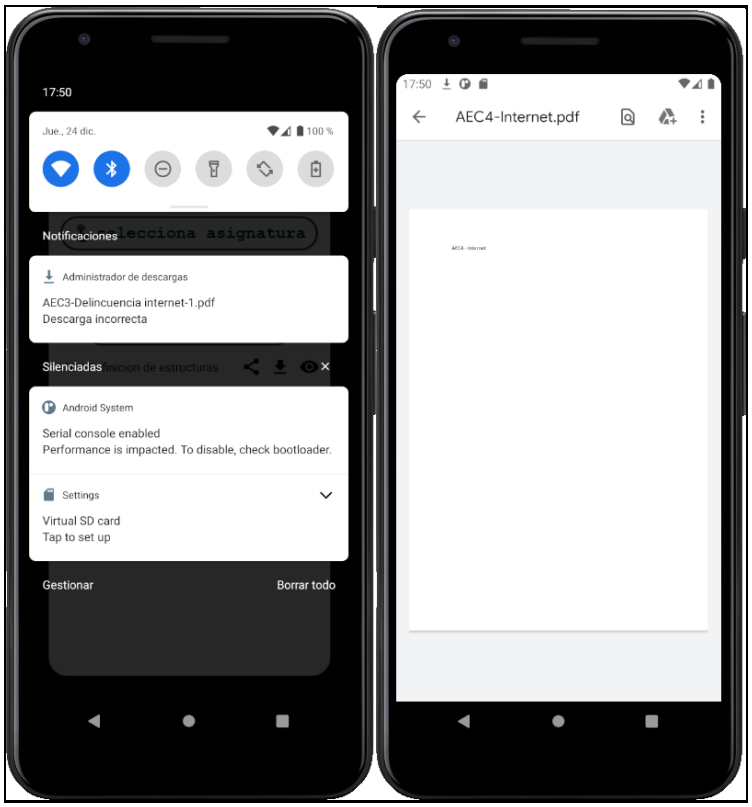
A4.7: Prueba de aceptación PA07

- Descripción de la prueba:

PA07	
Descripción	Comprobar interacción con contenido asignatura.
Pre-condición	Asignatura dispone de contenido.
Escenario	1. Seleccionar asignatura "Estructura de datos" 2. Compartir "AEC2 - Definición de estructuras". 3. Descargar "AEC3 - Delincuencia Internet". 4. Visualizar "AEC4 - Internet".
Post-condición	OK: Se muestra el contenido de la asignatura. OK: Aparece el menú "Comparte" con las opciones del sistema. OK: Se descarga el documento PDF en el dispositivo. OK: Se muestra el contenido PDF de forma predeterminada.

- Resultado gráfico:



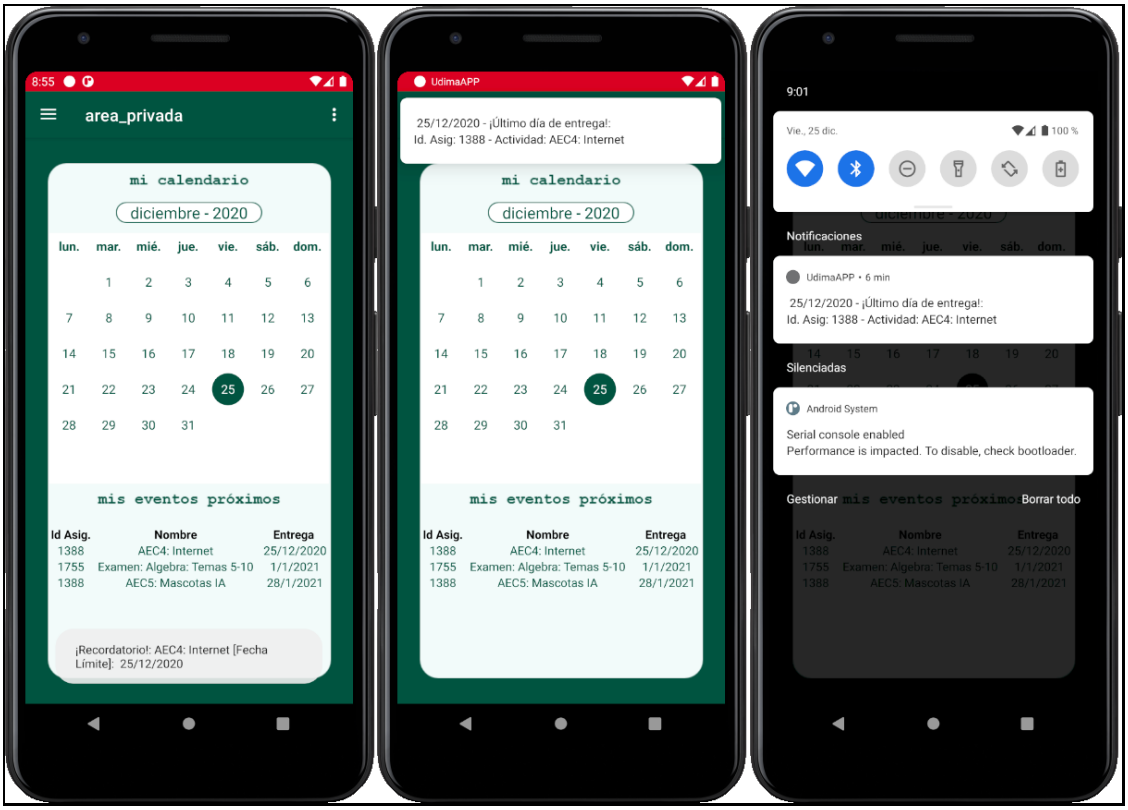


A4.8: Prueba de aceptación PA08

- Descripción de la prueba:

PA08	
Descripción	Recibir notificaciones externas a la aplicación.
Pre-condición	Estudiante de actividades con fecha fin posterior a la actual.
Escenario	1. Agregar actividad "AEC4-Internet" con fecha 25/12/2020 2. Pulsar en "Mi calendario". 3. Comprobar mensaje local en aplicación. 4. Comprobar recepción de notificación visual y sonora.
Post-condición	OK: Se muestra mensaje en la parte inferior de la App. OK: Aparece una notificación en la zona de notificaciones del móvil.

- Resultado gráfico:

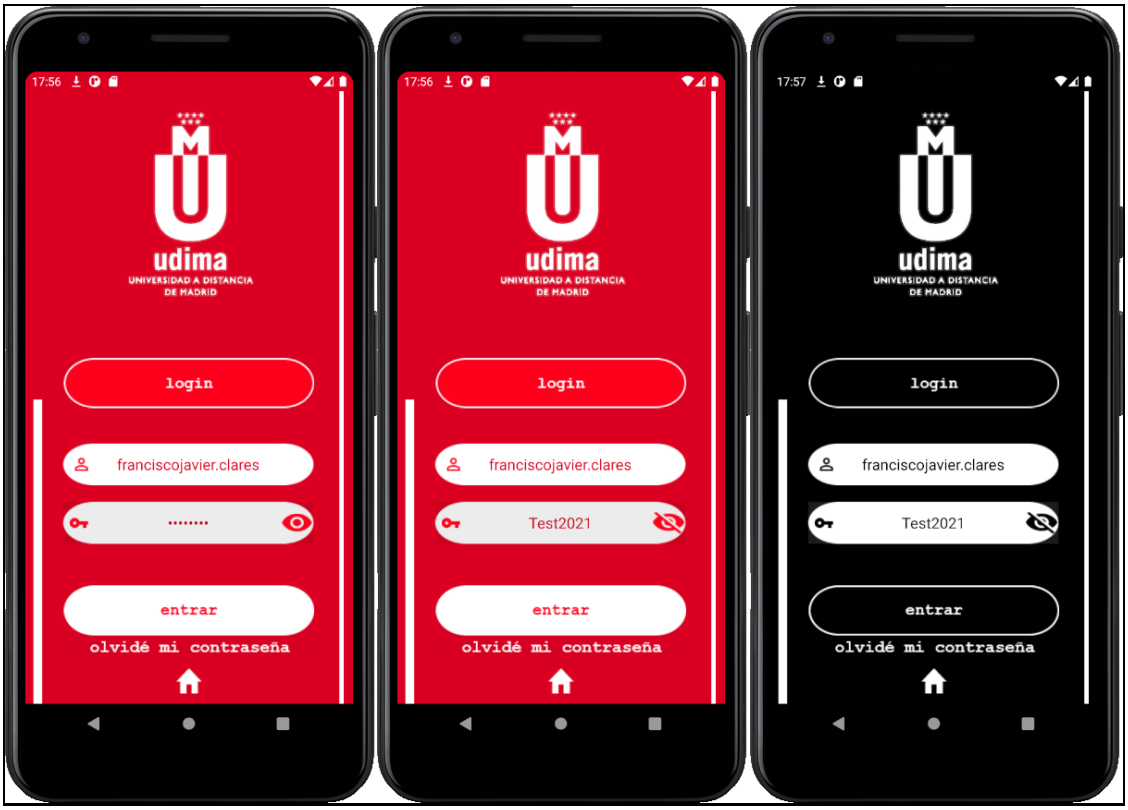


A4.9: Prueba de aceptación PA09

- Descripción de la prueba:

PA09	
Descripción	Ver/ocultar contraseña.
Pre-condición	Estudiante dispone de usuario registrado.
Escenario	1. Pulsar en Login. 2. Introducir usuario: franciscojavier.clares y contraseña: Test2021 3. Pulsar icono "Ver contraseña". 4. Pulsar icono "Ocultar contraseña". 5. Comprobar en modo "noche".
Post-condición	OK: Se muestra la contraseña. OK: Se oculta la contraseña. OK: Se cambia el icono a ojo abierto u ojo cerrado. OK: Funciona en modo noche, cambiando texto color.

- Resultado gráfico:



A4.10: Prueba de aceptación PA10

- Descripción de la prueba:

PA10	
Descripción	Comprobar actualización media de expediente.
Pre-condición	Estudiante debe tener asignaturas matriculadas.
Escenario	1. Pulsar en "Mi matrícula". Ver expediente actual: 6,72 2. Cambiar nota Inglés a: 9 3. Refrescar sección "Mi matrícula". Ver expediente actual: 7,52
Post-condición	OK: Se actualiza la nota media del expediente correctamente.

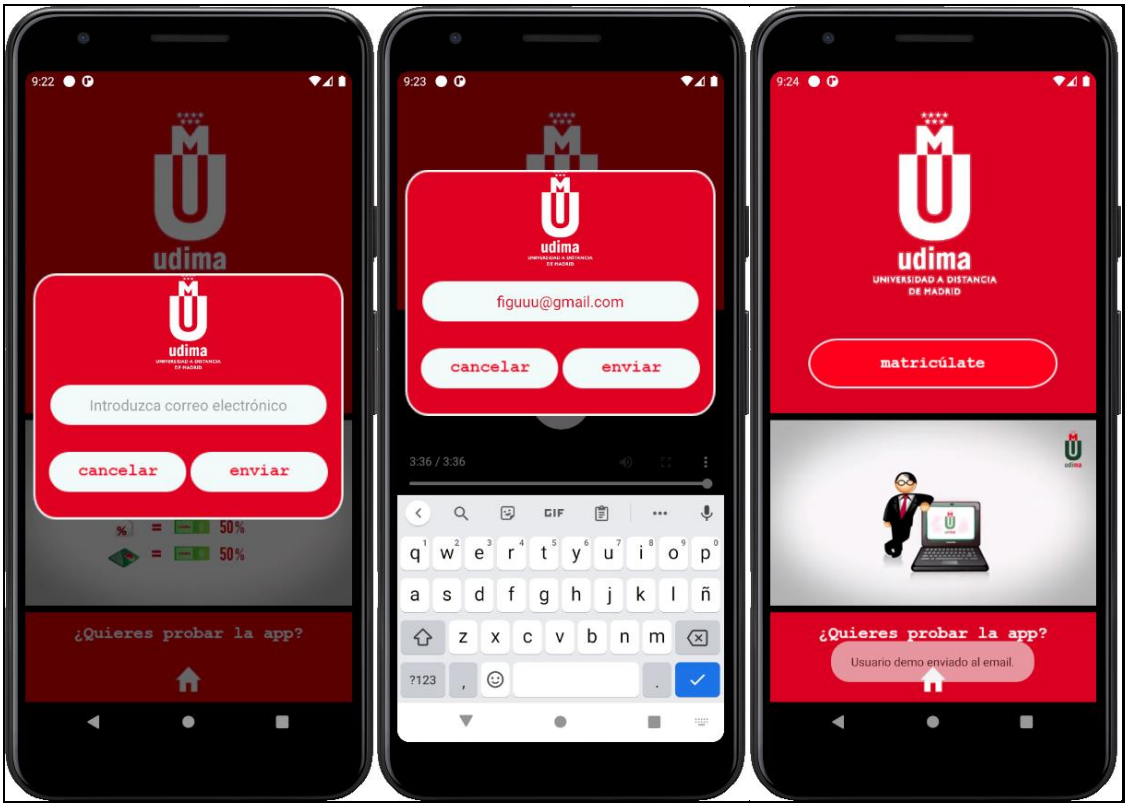
- Resultado gráfico:

A4.11: Prueba de aceptación PA11

- Descripción de la prueba:

PA11	
Descripción	Probar la aplicación sin disponer de usuario registrado.
Pre-condición	Estudiante debe disponer de la App instalada.
Escenario	1. Pulsar en "Matricúlate". 2. Pulsar en "¿Quieres probar la App? 3. Introducir correo "figuuu@gmail.com" y pulsar "Enviar". 4. Comprobar recepción correo con usuario y contraseña temporal. 5. Acceder a las secciones de la aplicación.
Post-condición	OK: Se recibe correctamente el correo y funciona la app en modo demo.

- Resultado gráfico:

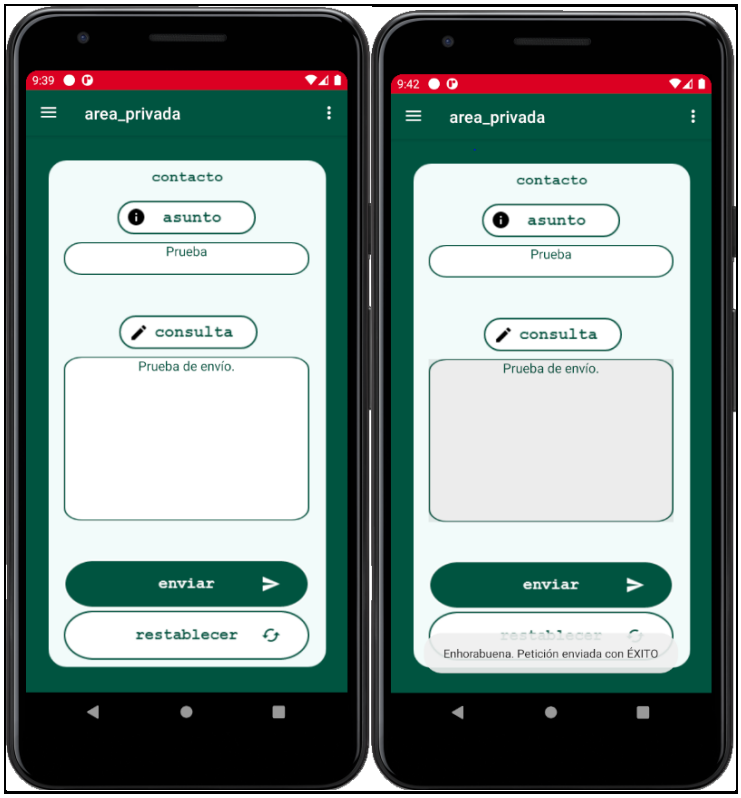


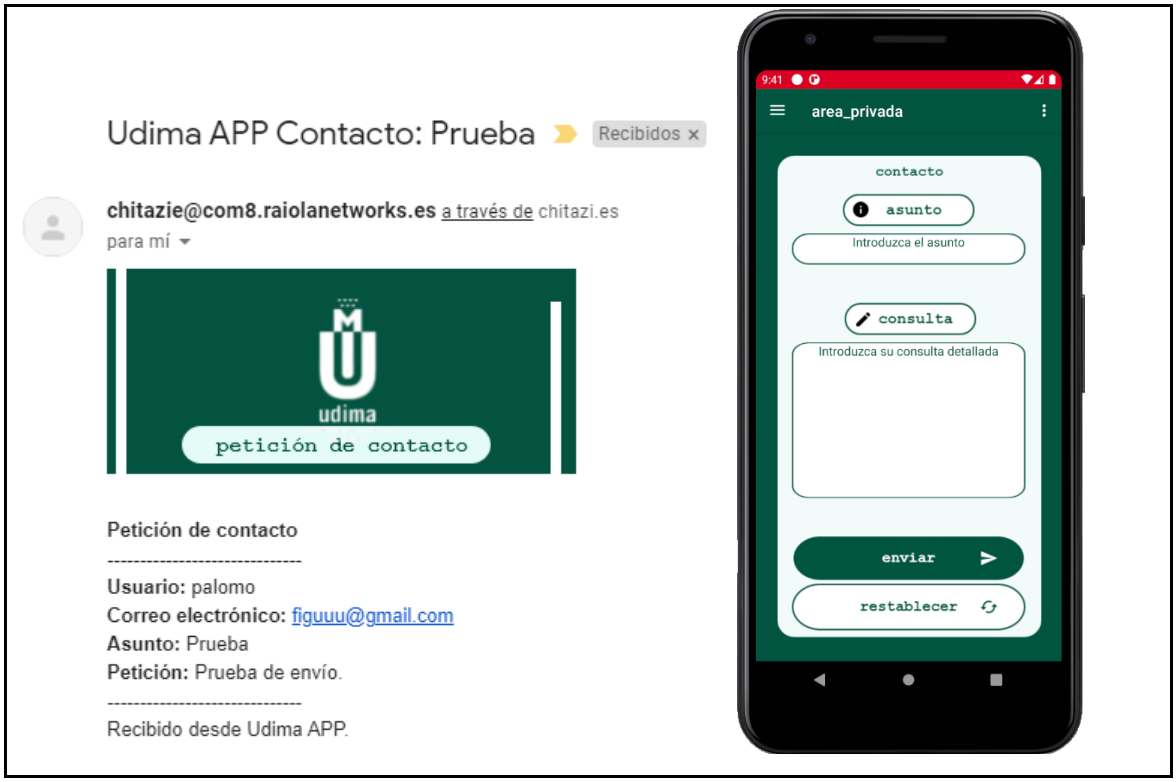
A4.12: Prueba de aceptación PA12

- Descripción de la prueba:

PA12	
Descripción	Enviar petición de contacto.
Pre-condición	Estudiante debe disponer de usuario registrado.
Escenario	1. Pulsar en "Contacto". 2. Introducir "Prueba" en Asunto. 3. Introducir "Prueba de envío" en Consulta. 4. Enviar y comprobar recepción del correo. 5. Restablecer.
Post-condición	OK: Se recibe correctamente el correo con los datos de Asunto y Consulta. OK: Al pulsar en "Restablecer" se borran los datos introducidos.

- Resultado gráfico:



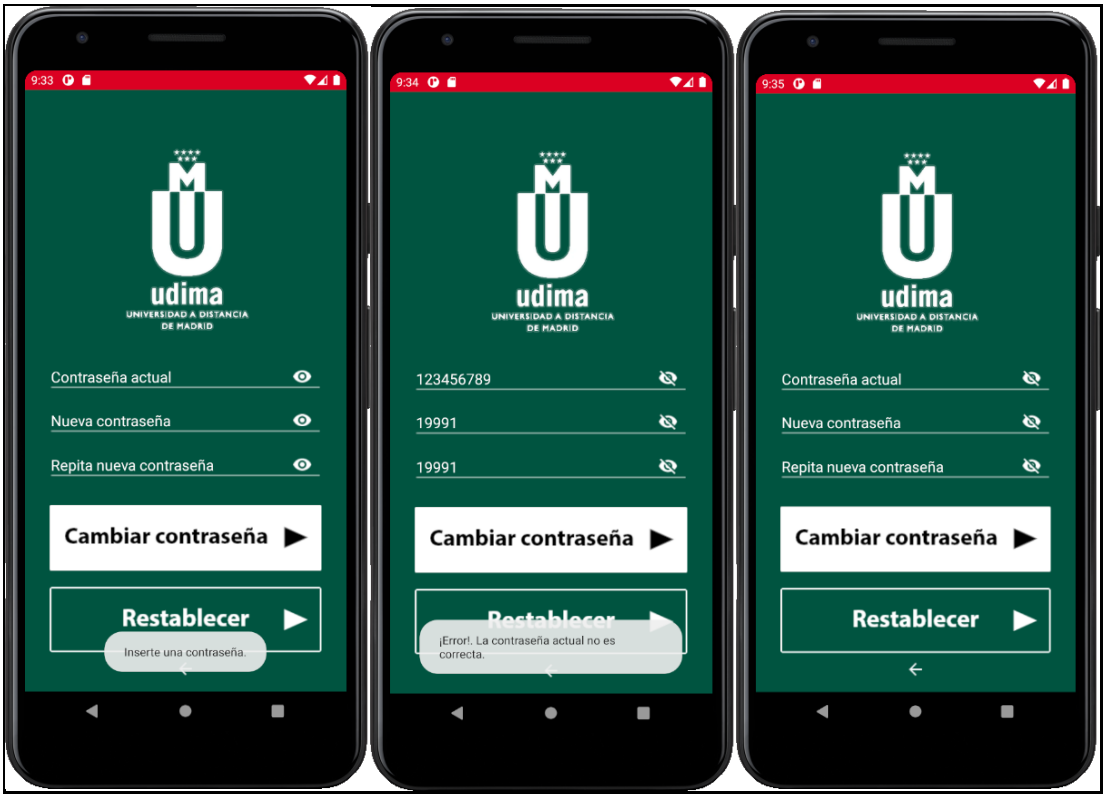


A4.13: Prueba de aceptación PA13

- Descripción de la prueba:

PA13	
Descripción	Cambiar password usuario.
Pre-condición	Estudiante debe disponer de usuario registrado.
Escenario	1. Pulsar en "Mi perfil - Cambiar contraseña". 2. Dejar en blanco y pulsar "Cambiar contraseña". 3. Introducir contraseña actual: 123456789 4. Introducir nueva contraseña: 19991 5. Pulsar Cambiar contraseña. 6. Pulsar Restablecer.
Post-condición	OK: Se recibe mensaje de error: "Inserte una contraseña". OK: Se recibe mensaje de error: "La contraseña actual no es correcta.". OK: Se elimina el valor de los campos.

- Resultado gráfico:

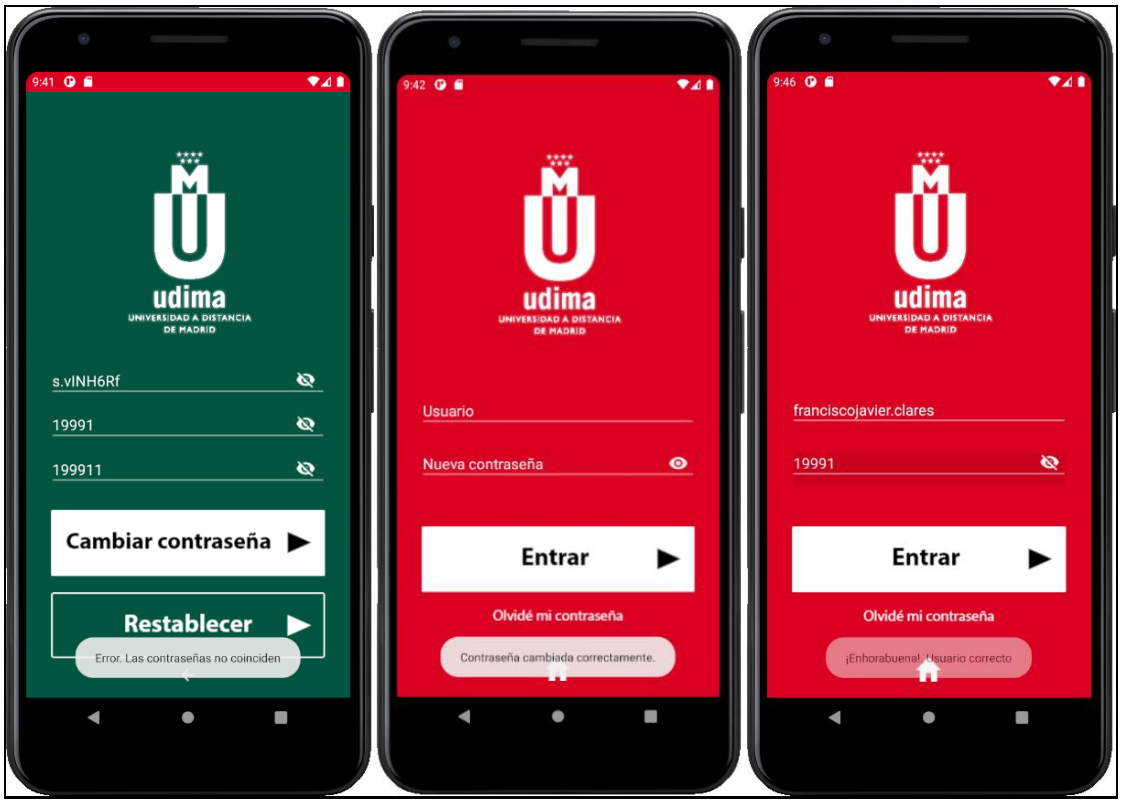


A4.14: Prueba de aceptación PA14

- Descripción de la prueba:

PA14	
Descripción	Cambiar password usuario.
Pre-condición	Estudiante debe disponer de usuario registrado.
Escenario	1. Pulsar en "Mi perfil - Cambiar contraseña". 2. Introducir contraseña actual: s.vINH6Rf 3. Introducir nueva contraseña: 19991 4. Repetir nueva contraseña: 199911 5. Pulsar Cambiar contraseña. 6. Repetir nueva contraseña: 19991 7. Pulsar Cambiar contraseña. 8. Iniciar usuario con nueva contraseña: 19991.
Post-condición	OK: Se recibe mensaje de error: "Inserte una contraseña". OK: Se recibe mensaje de éxito: Contraseña cambiada correctamente. OK: Usuario inicia correctamente al perfil.

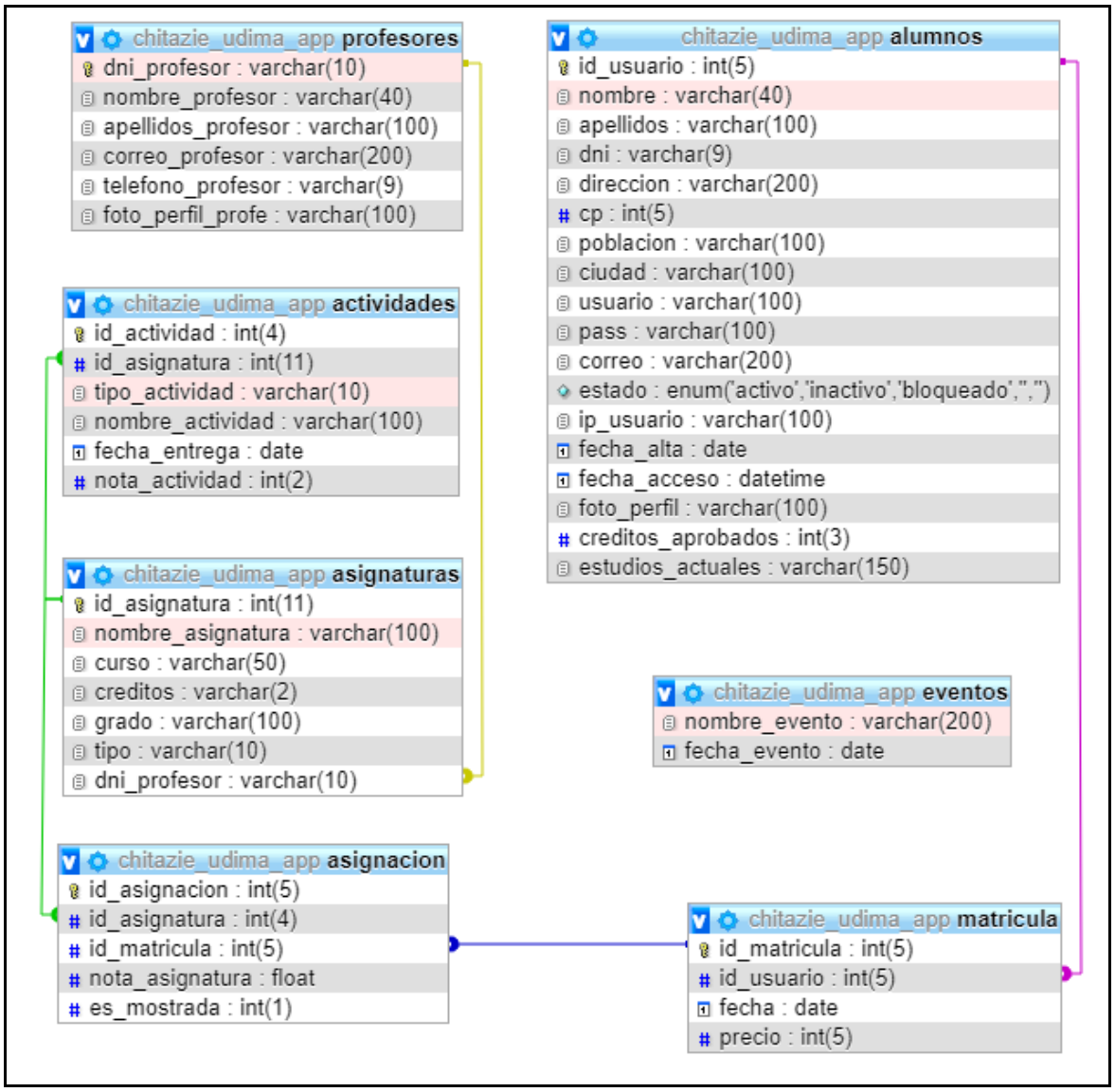
- Resultado gráfico:



Anexo A5. Base de datos.

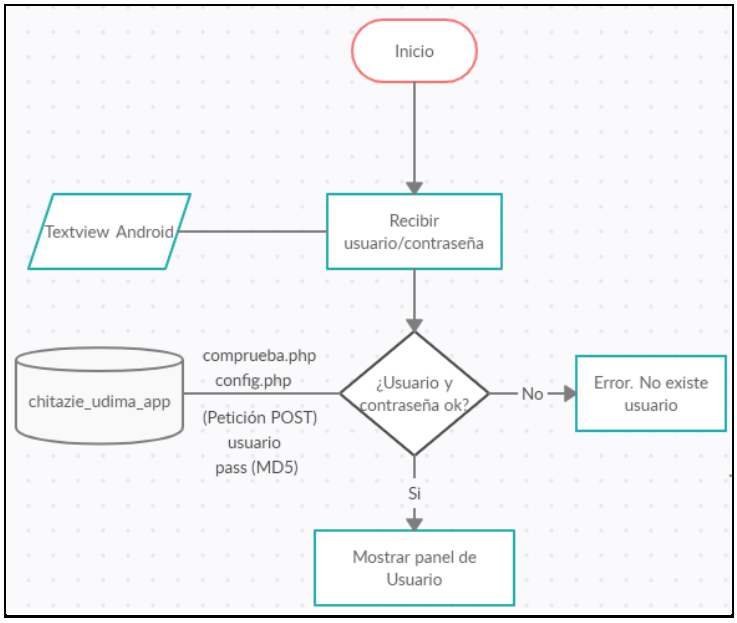
A5.1 Diseño final de base de datos.

Se muestra la implementación final de la base de datos, en el modelo real de E/R para el diseño, en función de los atributos y las relaciones entre claves primarias y foráneas:

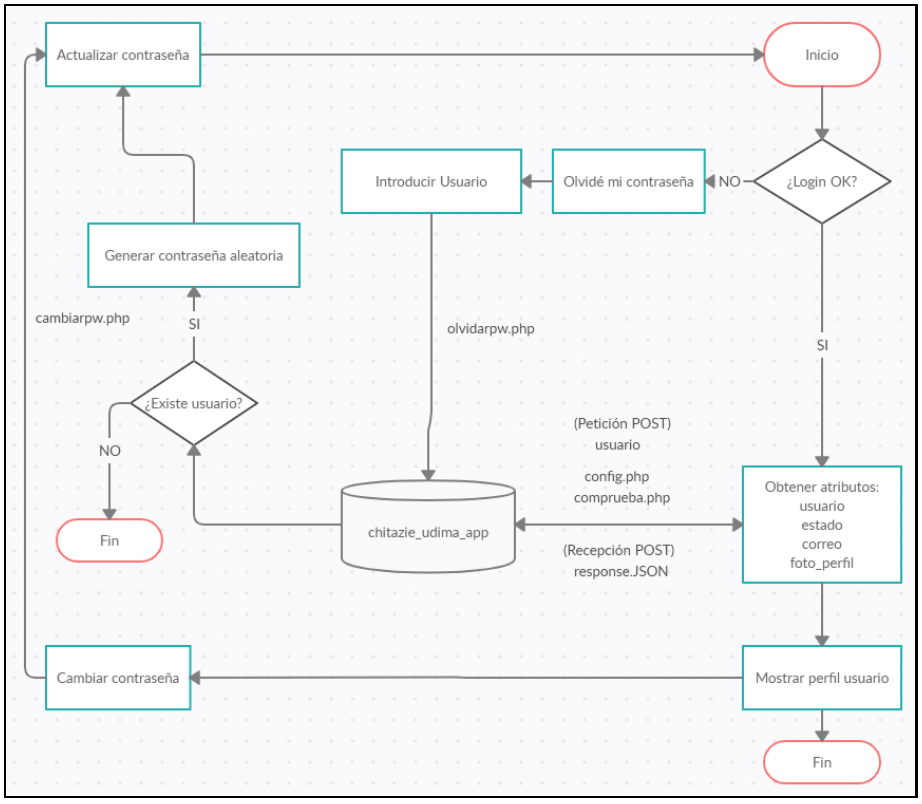


Anexo A6. Diagramas y notaciones gráficas implementación.

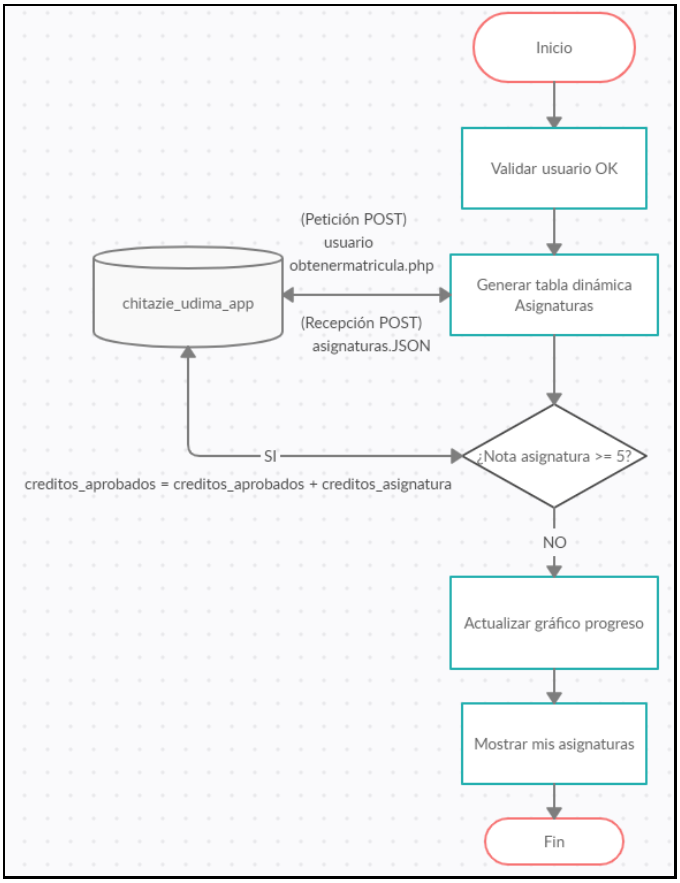
A6.1: Diagrama de flujo Sprint 1 y Sprint2.



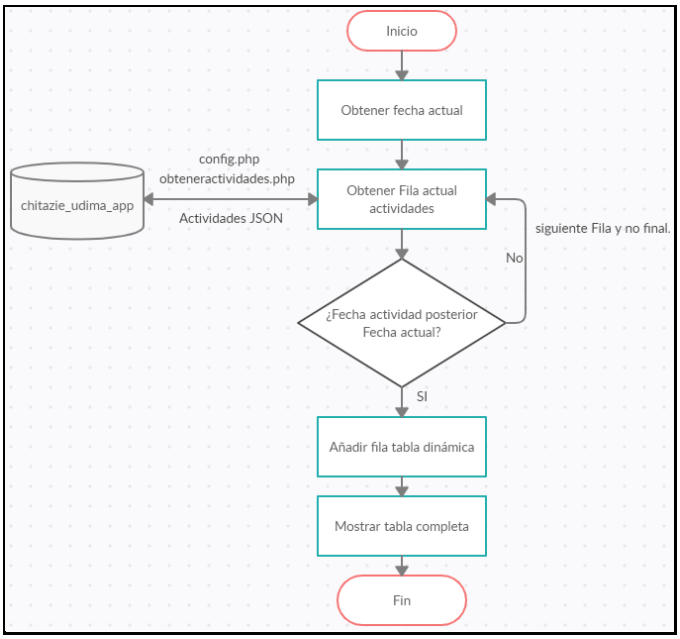
A6.2: Diagrama de flujo interfaz home y login.



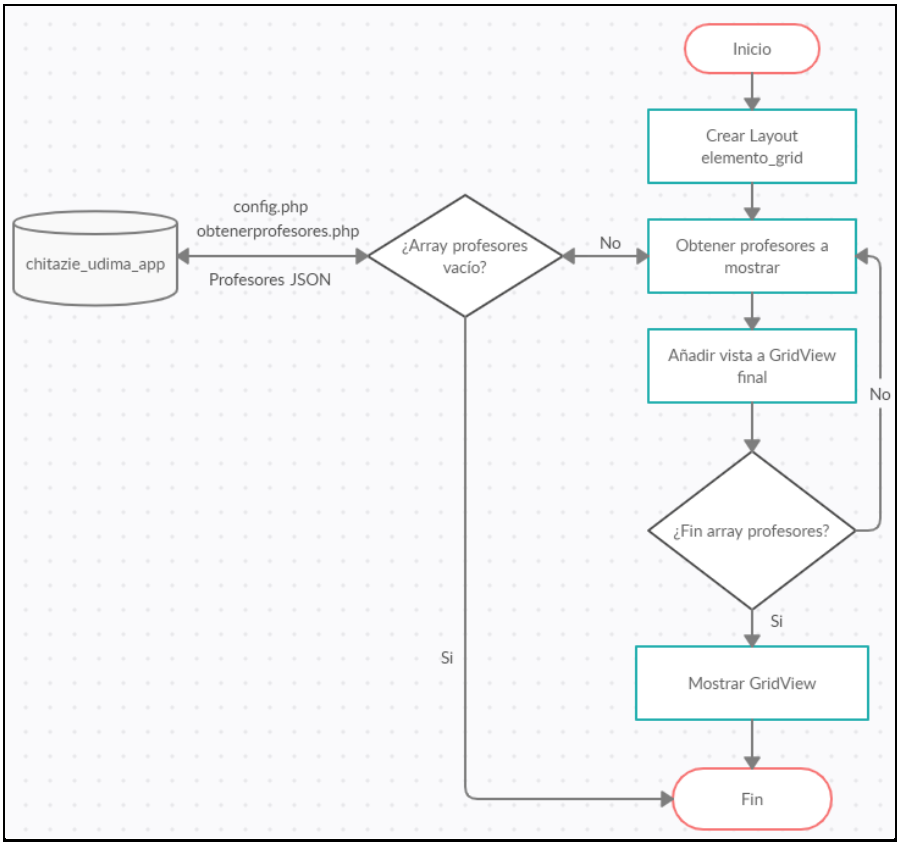
A6.3: Diagrama de flujo interfaz “mis asignaturas”.



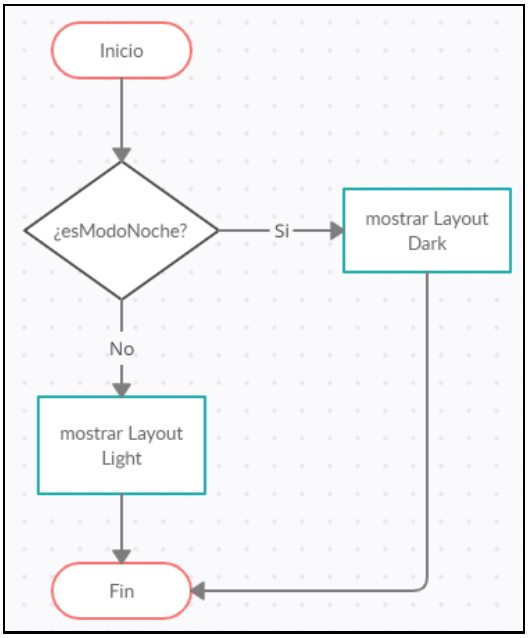
A6.4: Diagrama de flujo interfaz “mi calendario”.



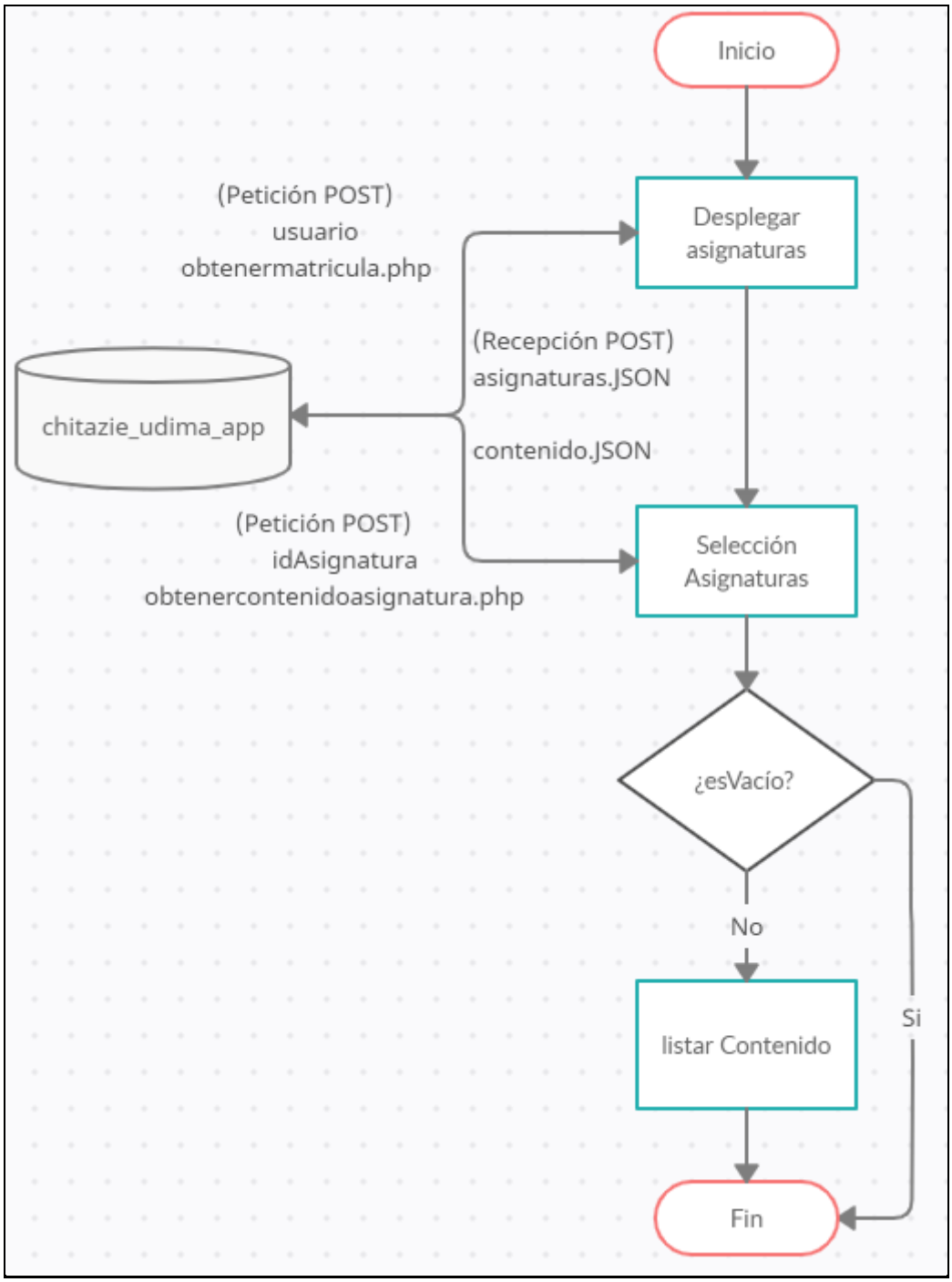
A6.5: Diagrama de flujo interfaz “mis profesores”.



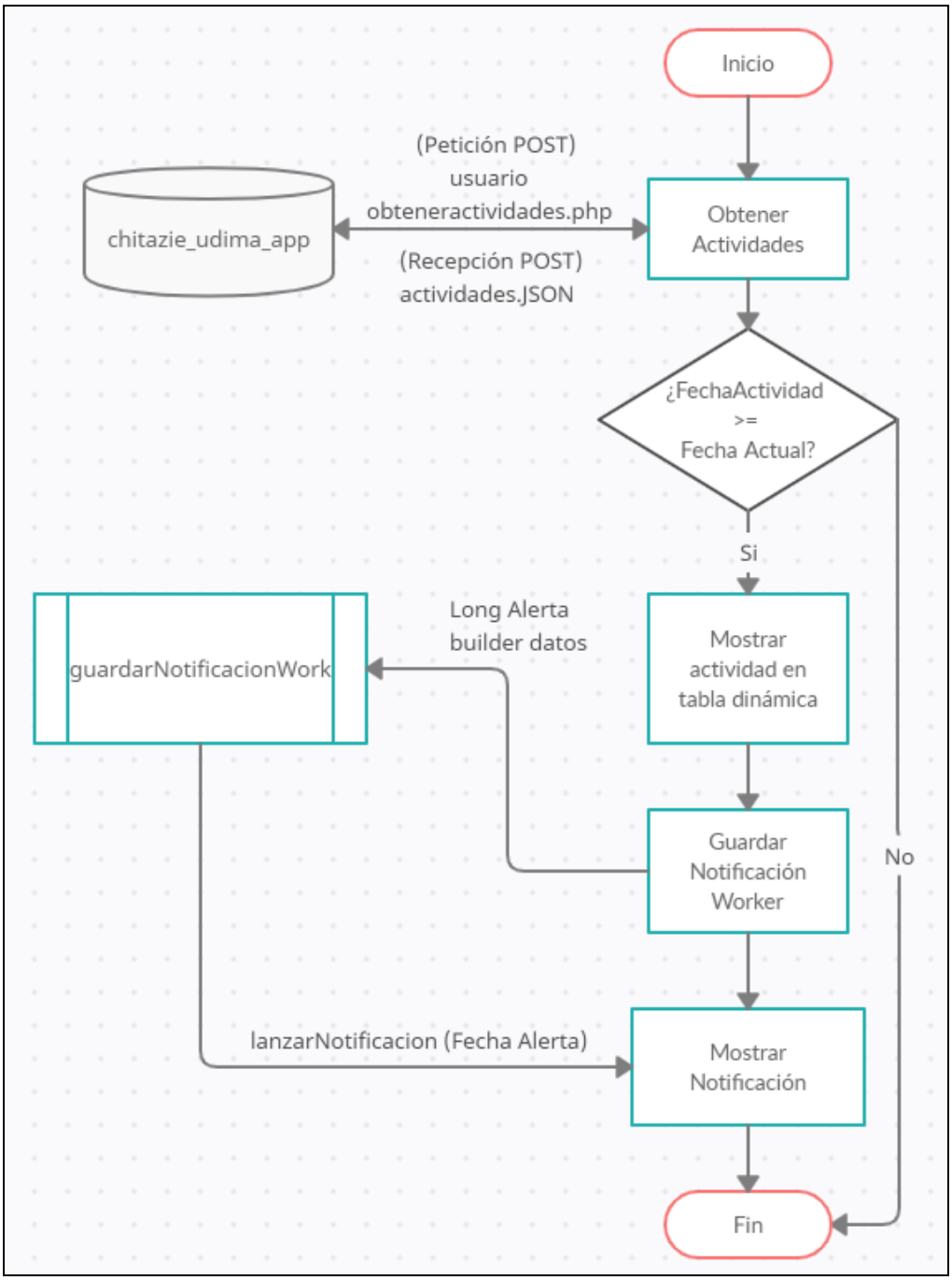
A6.6: Diagrama de flujo modo Noche.



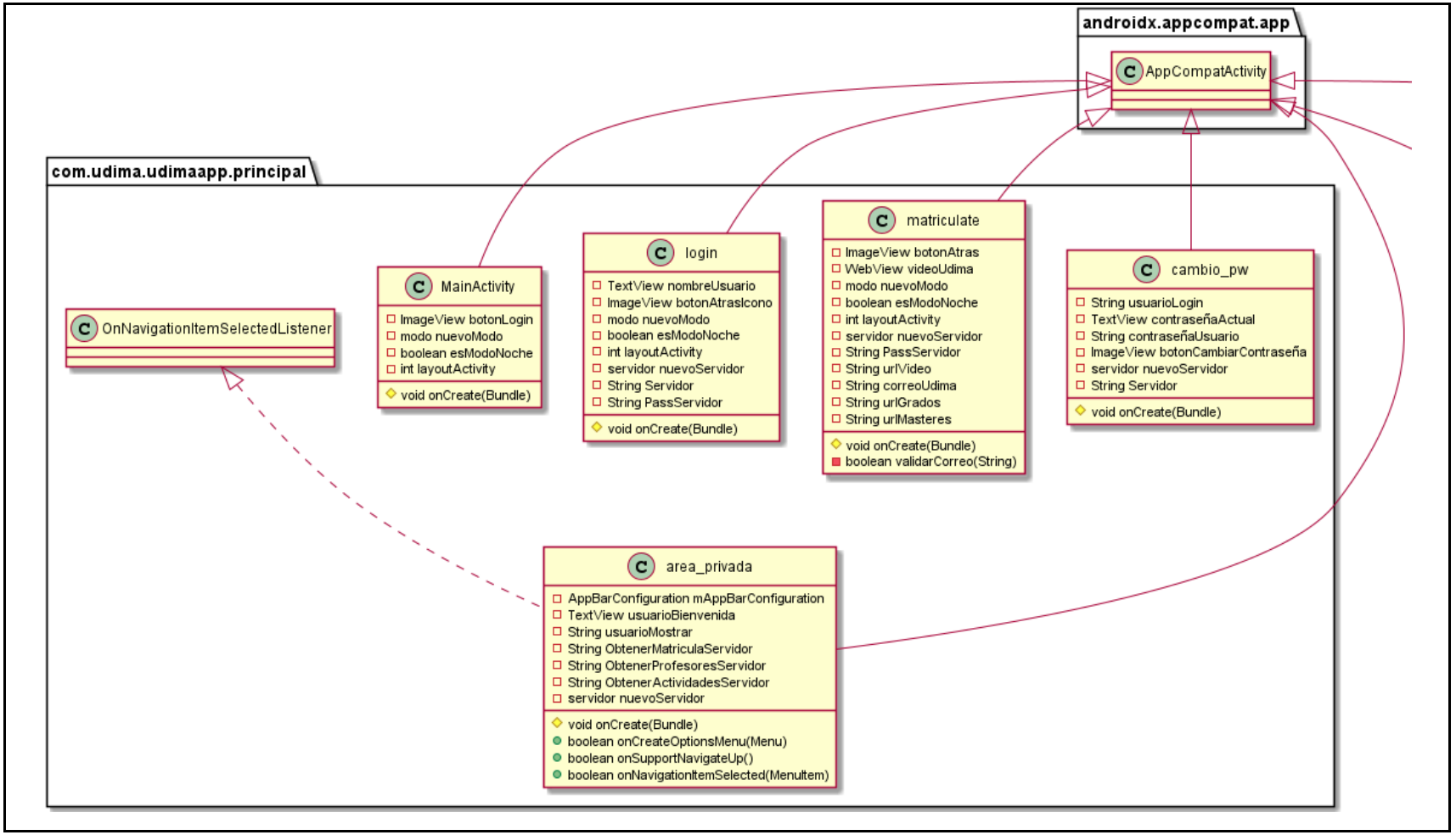
A6.7: Diagrama de flujo interfaz “Mi contenido”.



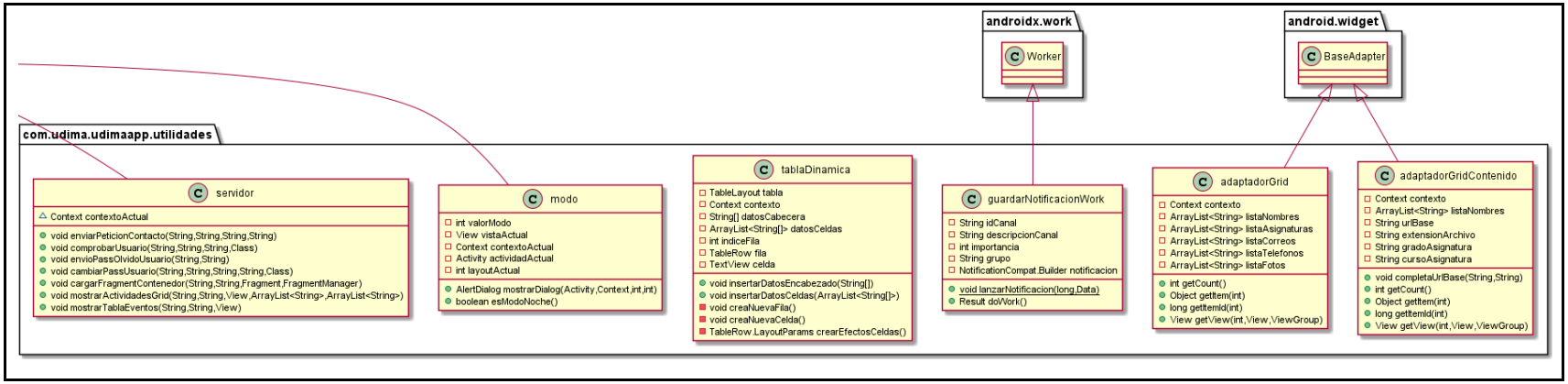
A6.8: Diagrama de flujo notificación Worker “Mi Calendario”.



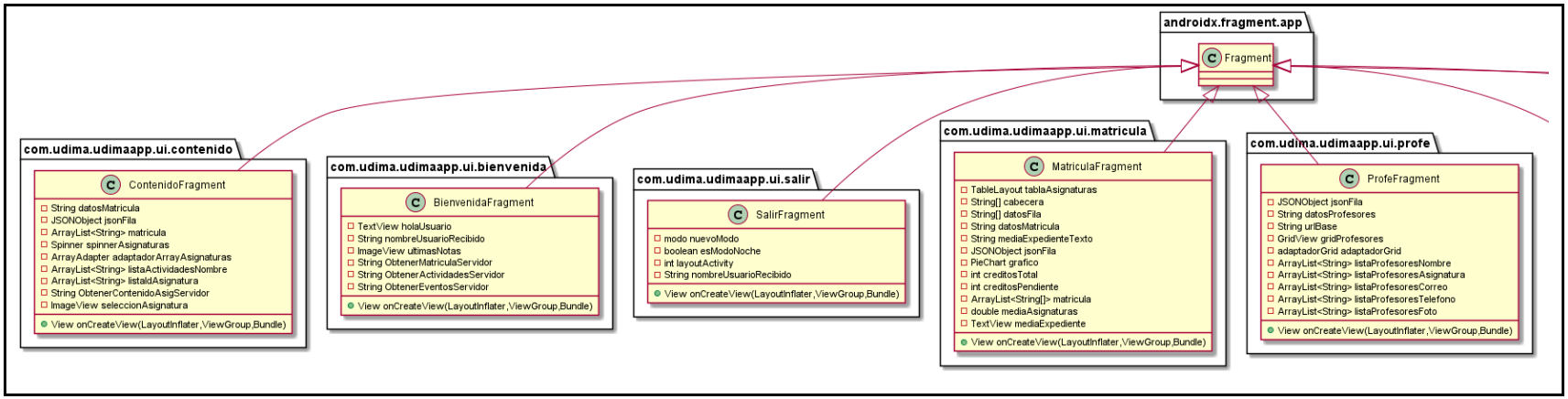
A6.9.1: Diagrama UML UdimaApp: Paquete Principal (1ª Parte).



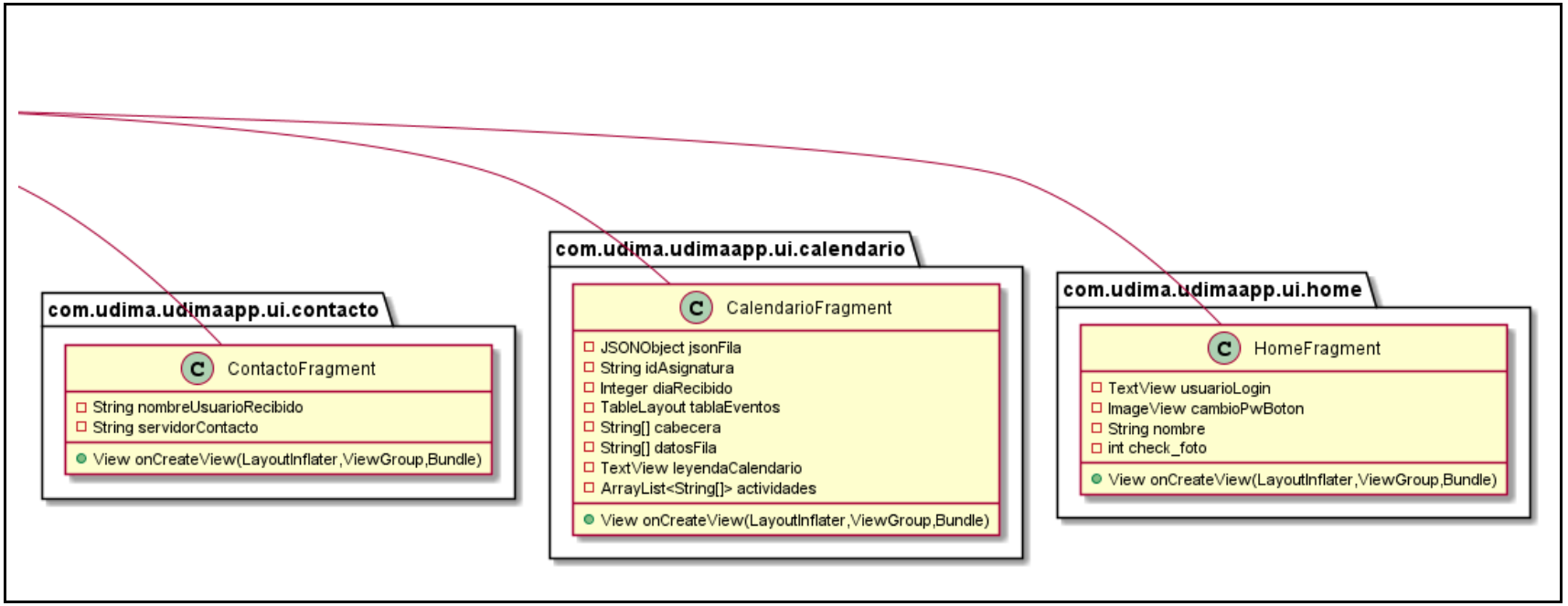
A6.9.2: Diagrama UML UdimaApp: Paquete Utilidades (2ª Parte).



A6.9.3: Diagrama UML UdimaApp: Paquete Ui (1ª Parte).

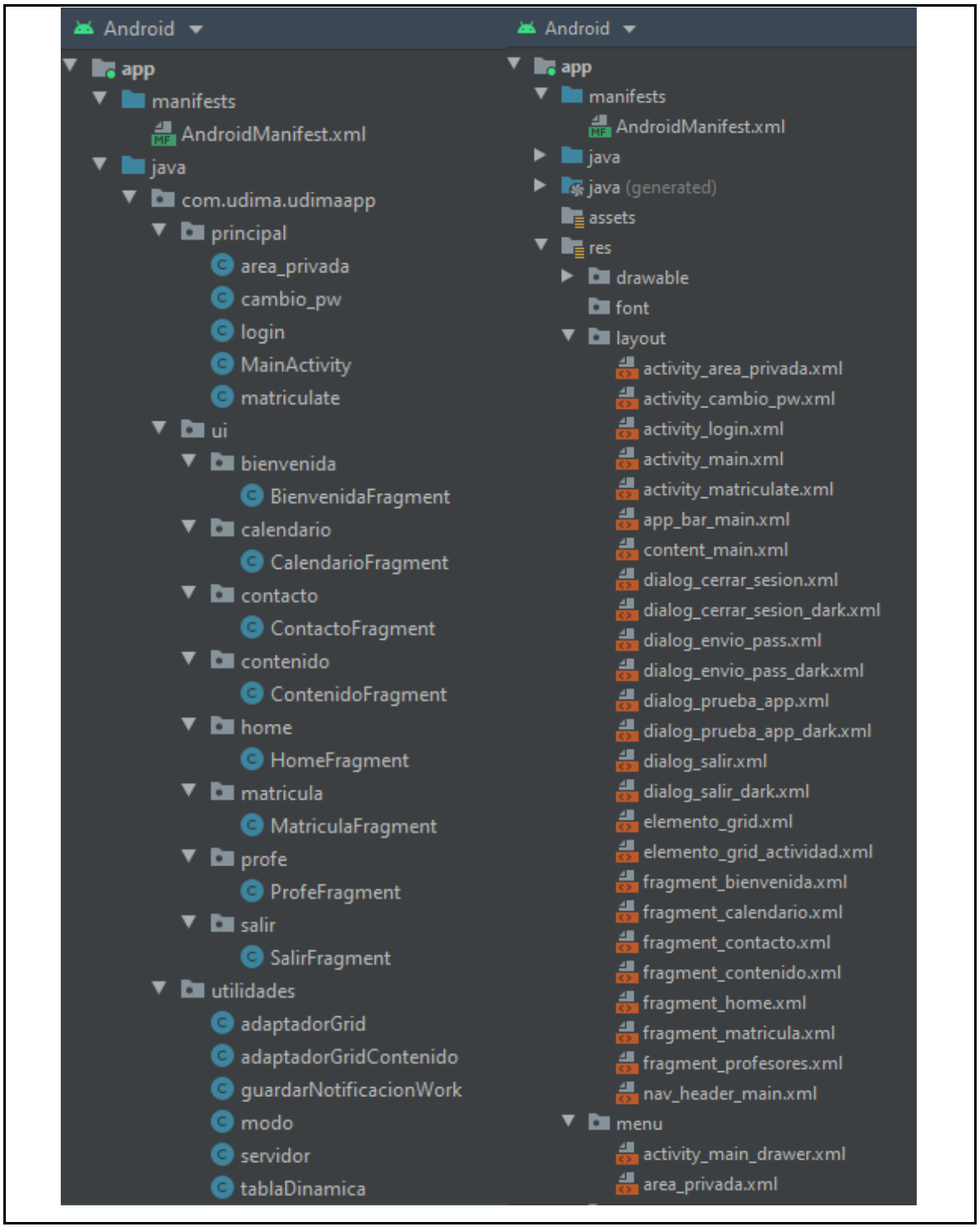


A6.9.4: Diagrama UML UdimaApp: Paquete Ui (2ª Parte).



A6.9.5: Estructura paquetes y ficheros.

Se muestra estructura de paquetes y ficheros de desarrollo funcionalidades (**Izquierda**) y ficheros de diseño de interfaces y actividades (**Derecha**):



Anexo A7. Código relevante.

A continuación, se muestran los aspectos más importantes del código elaborado en java para la aplicación dentro de la estructura definida en el anexo A6.9.5:

A7.1: Paquete “principal”.

- Clase “área_privada”:

Define el área privada del estudiante, desde dónde se llama a todos los fragmentos, y la que establece la funcionalidad del menú.

```
66          //Obtener usuario principal autentificado en el sistema.
67          Bundle datos = getIntent().getExtras();
68          usuarioMostrar = datos.getString( key: "pasarNombreUsuario");
```

```
83      @Override
84      public boolean onCreateOptionsMenu(Menu menu) {
85          // Inflamos menú y mostramos nombre de usuario recibido de login.
86          getMenuInflater().inflate(R.menu.area_privada, menu);
87          usuarioBienvenida = (TextView) findViewById(R.id.correo_menu_bienvenida);
88          usuarioBienvenida.setText(usuarioMostrar);
89          return true;
90      }
91
92      @Override
93      public boolean onSupportNavigateUp() {
94          NavController navController = Navigation.findNavController( activity: this, R.id.nav_host_fragment);
95          return NavigationUI.navigateUp(navController, mAppBarConfiguration)
96                  || super.onSupportNavigateUp();
97      }
98
```

  - Inflate: Se usa para poder inflar las vistas, en este caso el menú lateral.
  - Bundle: Medio de comunicación que proporciona la vía para poder enviar datos entre actividades, fragmentos y en todas sus formas. En este caso, con “getExtras()”, recibimos el usuario (usuarioMostrar) autenticado que llega desde la interfaz de login, necesario para la obtención de datos POST.
  - Se carga el layout host_fragment que contiene la navegación entre fragmentos en el menú.

```
public boolean onNavigationItemSelected(MenuItem item) {
    int id = item.getItemId();
    FragmentManager fragmentManager = getSupportFragmentManager();
    //Si el usuario elige Home.
    if (id == R.id.nav_home) {
        Bundle datosHome = getIntent().getExtras();
        //Creamos fragment concreto.
        HomeFragment fragmentoHome = new HomeFragment();
        //Recibimos el usuario desde login que nos valdrá para toda la aplicación.
        responseLogin = datosHome.getString( key: "response");
            if (responseLogin != null) {
                Bundle pasarDatos = new Bundle();
                pasarDatos.putString("datosHome", responseLogin);
                //Pasamos datos por el Bundle.
                fragmentoHome.setArguments(pasarDatos);
            }
        //Reemplazamos el contenedor principal por el fragmento principal.
        fragmentManager.beginTransaction().replace(R.id.contenedor, fragmentoHome).commit();
```

- Home: Perfil del estudiante. “FragmentManager” es el adaptador que nos permite poder cargar el fragmento necesario en nuestro “contenedor” principal. Se crea un fragmento específico de tipo “Home”, enviamos los datos del estudiante, recibidos desde servidor, a través de un Bundle y posteriormente con “replace”, cambiamos la parte central por nuestro fragmento Home.

```
    } else if (id == R.id.nav_bienvenida) {
        //Creamos fragmento concreto del tipo y llamamos a Servidor.
        Bundle datosBienvenida = new Bundle();
        BienvenidaFragment fragmentoBienvenida = new BienvenidaFragment();
        datosBienvenida.putString("nombreUsuario", usuarioMostrar);
        fragmentoBienvenida.setArguments(datosBienvenida);
        fragmentManager.beginTransaction().replace(R.id.contenedor, fragmentoBienvenida).commit();
```

- Bienvenida: Fragmento de inicio, al ingresar en área privada. Se utiliza “bundle” de nuevo para enviar el dato del nombre de usuario, y reemplazamos fragmento.

```
    //Si el usuario elige Mi matrícula.
} else if (id == R.id.nav_matricula) {
    //Creamos fragmento concreto del tipo y llamamos a Servidor.
    MatriculaFragment fragmentoMatricula = new MatriculaFragment();
    nuevoServidor.cargarFragmentContenedor(usuarioMostrar,ObtenerMatriculaServidor,fragmentoMatricula,fragmentManager);
```

- Matrícula: Se crea un fragmento concreto de Matrícula.
- Se utiliza “cargarFragmentContenedor” desde la clase “Servidor”, la cual, nos devuelve el json con los datos de la matrícula a partir del usuario y la url dónde está el código PHP de la petición POST.

```
        //Si el usuario elige Mi calendario.
    } else if (id == R.id.nav_calendario) {
        //Creamos fragmento concreto del tipo y llamamos a Servidor.
        CalendarioFragment fragmentoCalendario = new CalendarioFragment();
        nuevoServidor.cargarFragmentContenedor(usuarioMostrar,ObtenerActividadesServidor,fragmentoCalendario,fragmentManager);

        //Si el usuario elige Mis profesores.
    } else if (id == R.id.nav_profesores) {
        //Creamos fragmento concreto del tipo y llamamos a Servidor.
        ProfeFragment fragmentoProfesores = new ProfeFragment();
        nuevoServidor.cargarFragmentContenedor(usuarioMostrar,ObtenerProfesoresServidor,fragmentoProfesores,fragmentManager);

        //Si el usuario elige Mi contenido.
    } else if (id == R.id.nav_contenido) {
        //Creamos fragmento concreto del tipo y llamamos a Servidor.
        ContenidoFragment fragmentoContenido = new ContenidoFragment();
        nuevoServidor.cargarFragmentContenedor(usuarioMostrar,ObtenerMatriculaServidor,fragmentoContenido,fragmentManager);
```

- Se concretan de la misma forma anterior, los demás fragmentos (que son las opciones del menú), utilizando cada una su código PHP concreto, alojado en el servidor, aprovechando la misma clase de “Servidor”.

```
        //Si el usuario elige Mi contacto.
    } else if (id == R.id.nav_contacto) {
        Bundle datosContacto = new Bundle();
        //Creamos fragmento concreto del tipo
        ContactoFragment fragmentoContacto = new ContactoFragment();
        datosContacto.putString("nombreUsuario", usuarioMostrar);
        //Le pasamos en el Bundle el usuario para poder mandar el contacto concreto a dicho correo y a soporte.
        fragmentoContacto.setArguments(datosContacto);
        fragmentManager.beginTransaction().replace(R.id.contenedor, fragmentoContacto).commit();

        //Si el usuario elige Salir.
    } else if (id == R.id.nav_salir) {
        Bundle datosBienvenida = new Bundle();
        //Creamos fragmento concreto del tipo.
        SalirFragment fragmentoSalir = new SalirFragment();
        //Pasamos el bundle con el nombre de usuario, necesario para mostrar bienvenida al salir con datos.
        datosBienvenida.putString("nombreUsuario", usuarioMostrar);
        fragmentoSalir.setArguments(datosBienvenida);
        fragmentManager.beginTransaction().replace(R.id.contenedor, fragmentoSalir).commit();
    }
```

- En los fragmentos “Contacto” y “Salir” se necesita pasar el nombre del usuario logueado, y reemplazar el contenedor principal.

- Clase “cambio_pw”:

Clase que permite cambiar la contraseña de usuario al estudiante, con la seguridad propia de tener que introducir la contraseña actual del mismo. Controla la validación en base de datos de la contraseña actual, y actualiza el campo, en el caso de ser correcta e introducir la nueva.

```
43          int modoActual = AppCompatDelegate.getDefaultNightMode();
44          if (modoActual == 2){
45              botonCambiarContraseña.setImageResource(R.drawable.cambiapw_01_banner_dark);
46          }
```

- getDefaultNightMode(): Nos devuelve el valor del modo noche. En nuestro caso, si el valor es “2”, cambiamos el botón principal de cambiar contraseña al color negro.

```
49      botonCambiarContraseña.setOnClickListener(new View.OnClickListener() {
50          @Override
51          public void onClick(View v) {
52              // Recibimos contraseña introducida por el usuario.
53              contraseñaInicio = contraseña.getText().toString();
54              contraseñaRepite = repiteContraseña.getText().toString();
55              contraseñaUsuario = contraseñaActual.getText().toString();
56
57              //Si no introduce nada, mostramos mensaje de error.
58              if ((contraseñaInicio.isEmpty()) || (contraseñaRepite.isEmpty())){
59                  Toast.makeText(getApplicationContext(), text: "Inserte una contraseña.",Toast.LENGTH_SHORT).show();
60              }else {
61                  //Si ha introducido la misma contraseña en ambos campos, cambiamos contraseña del usuario.
62                  if (contraseñaInicio.equals(contraseñaRepite)) {
63                      //Toast.makeText(getApplicationContext(), "Ok, contraseñas correctas", Toast.LENGTH_SHORT).show();
64                      Class actividadDestino = login.class;
65                      nuevoServidor.cambiarPassUsuario(Servidor,usuarioLogin,contraseñaUsuario,contraseñaInicio,actividadDestino);
66
67                      //Si no coinciden las contraseñas, mostramos error y no llamamos al método cambiarContraseña.
68                  } else {
69                      Toast.makeText(getApplicationContext(),  text: "Error. Las contraseñas no coinciden", Toast.LENGTH_SHORT).show();
70                  }
71              }
72          }
73      });
```

- isEmpty(): Controla que la contraseña introducida no esté en blanco.
- Equals: Controla que la nueva contraseña sea la misma en los dos campos “Nueva contraseña” y “Repita nueva contraseña”.
- Si son iguales, se llama a “cambiarPassUsuario” de la clase “Servidor”, que permite comprobar la antigua contraseña en base de datos, y si es correcta, actualiza dicho campo con la contraseña actual, cargando la actividadDestino que será de nuevo Login, para entrar con la nueva contraseña cambiada.

```
93      botonOcultarPwActual.setOnClickListener(new View.OnClickListener() {
94          @Override
95          public void onClick(View v) {
96              if (contraseñaActual.getTransformationMethod().equals(PasswordTransformationMethod.getInstance())){
97                  botonOcultarPwActual.setImageResource(R.drawable.ojocerrado_icono_light); //Cargamos icono.
98                  contraseñaActual.setTransformationMethod(HideReturnsTransformationMethod.getInstance()); //Convertimos en campo oculto.
99              }else { //Si no está oculta, ocultar y poner ojo abierto.
100                 contraseñaActual.setTransformationMethod(PasswordTransformationMethod.getInstance());
101                 botonOcultarPwActual.setImageResource(R.drawable.ojoabierto_icono_light);
102             }
103         }
104     });
```

- getTransformationMethod(): Determina si el “TextView” de la contraseña está en modo “Password” o “Text”, es decir, si está oculta o no.
- setTransformationMethod(): Cambia el modo del “TextView” de la contraseña introducida, de modo que si se pulsa en el “ojo abierto”, se muestra la contraseña y si se pulsa en el “ojo cerrado” se oculta.

- Clase “login”:

Por una parte, encargada de controlar que existe el usuario con las credenciales facilitadas y, por otra parte, permite enviar una contraseña aleatoria si el usuario está registrado. En caso de introducir credenciales correctas, se muestra área_privada:

```
48          botonAtrasIcono.setOnClickListener(new View.OnClickListener() {
49              @Override
50              public void onClick(View v) {
51                  Intent i = new Intent( packageContext: login.this, MainActivity.class);
52                  startActivity(i); //Intent para ir desde login hasta MainActivity, la actividad anterior.
53              }
54          });
```

- Intent: Permite, entre otras funciones, ir desde una actividad a otra. En este caso, al pulsar botón atrás, nos devuelve a “MainActivity”.

```
57      botonValidarUsuario.setOnClickListener(new View.OnClickListener(){
58          @Override
59          public void onClick(View view){
60              String nombreUser = (String) nombreUsuario.getText().toString();//Nombre y pass recibidos del usuario.
61              String passwordUsuario = (String)  passUsuario.getText().toString();
62              Class actividadDestino = area_privada.class; //Clase necesaria para el destino en clase Servidor.
63              nuevoServidor.comprobarUsuario(nombreUser,passwordUsuario,Servidor,actividadDestino); //Comprueba login en servidor.
64          }
65      });
```

- Se captura nombre de usuario y contraseña a partir de los “TextView” “nombreUser” y “passwordUsuario”.
- comprobarUsuario: A partir de los datos recogidos, y el código PHP para comprobar si son correctos, se valida el usuario en base de datos, y en caso de ser éxito, se realiza un “Intent” dentro de la instancia “Servidor”, y nos lleva a la clase actividadDestino, que es área_privada.

```
//Inflar un AlertDialog para introducir usuario y enviar nueva contraseña aleatoria, si éste existe.
botonOlvidarContraseña.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view){
        AlertDialog dialog; //Dependiendo del modo actual, nos muestra el dialog en dicho modo.
        dialog = nuevoModo.mostrarDialog( actividad: login.this, view.getContext(),layoutActivity,layoutActivityDark);
        dialog.create();
        ImageView botonEnvio = (ImageView) dialog.findViewById(R.id.botonEnviarEnvio);
        ImageView botonCancelar = (ImageView) dialog.findViewById(R.id.botonCancelarEnvio);
        TextView usuarioEnviarPw = (TextView) dialog.findViewById(R.id.envioPassUsuario_tv);

            //Si pulsa enviar, comprobamos si existe el usuario que ha introducido en cadena.
            botonEnvio.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    String cadena = usuarioEnviarPw.getText().toString();
                        if (!cadena.isEmpty()){ //Debe introducir usuario.
                            nuevoServidor.envioPassOlvidoUsuario(PassServidor,cadena);
                            dialog.dismiss();
                        }else{ //Si pulsa enviar sin introducir usuario, error.
                            Toast.makeText(getApplicationContext(), text: "Error. Introduzca un usuario",Toast.LENGTH_SHORT).show();
                        }
                }
            });
            //Si pulsa cancelar, cerramos AlertDialog dialog.
            botonCancelar.setOnClickListener(new View.OnClickListener(){
                @Override
                public void onClick(View v) { dialog.dismiss(); }
            });

        dialog.show();
    }
});
```

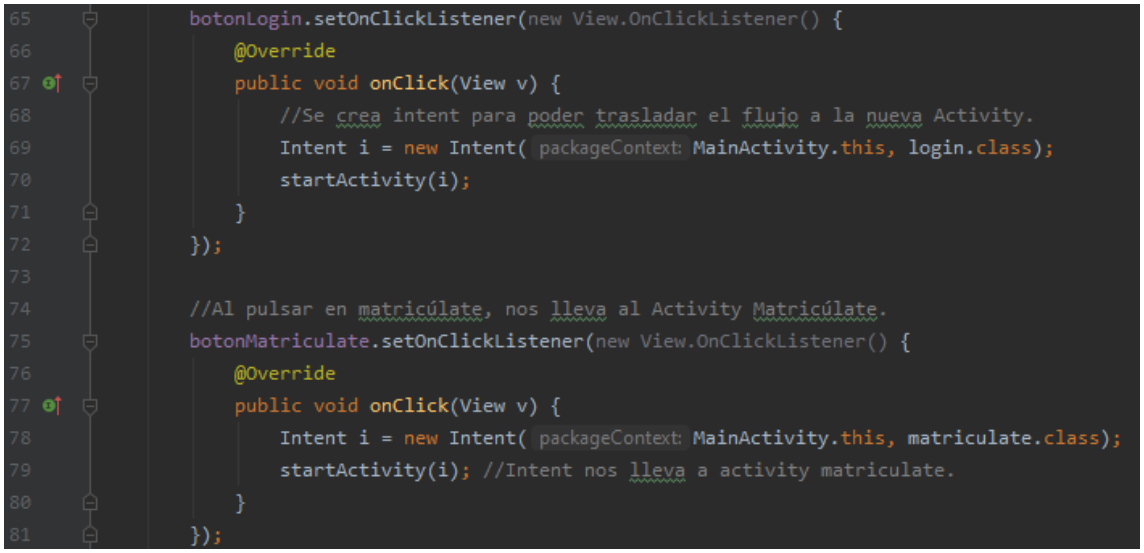
- AlertDialog: Ventana de alerta, emergente, en la que se carga el TextView necesario para introducir el nombre de usuario, para pasar al servidor. Se utiliza “create()” para crear la ventana y “show()” para mostrarla.
- mostrarDialog: Método de la clase “Modo”, a la que pasamos el contexto actual dónde nos encontramos que la propia vista “view”, la actividad (estamos en login, de ahí login.this, al ser la clase), y también los dos layout, el modo día y modo noche. Este método, se encarga de formar dicha ventana emergente con el layout concreto, dependiendo si se encuentra en modo noche o no. Por esta razón, para cada dialog, se observan layouts duplicados, cada uno para cada modo, con el diseño concreto.
- botonEnvio: Escuchador que captura cuando el usuario pulsa el botón de “Enviar”, la instancia de “Servidor”, y el método “envioPassOlvidoUsuario”, permite crear una contraseña aleatoria, actualizar en base de datos y enviar al correo electrónico registrado del usuario.

- Clase “MainActivity”:

Clase principal de la aplicación, que dispone las opciones de ir a “Matricúlate” o bien a “Login”. Como funciones adicionales, permite activar/desactivar “modo noche” y salir de la aplicación.

```
public void onClick(View v) {
    AlertDialog dialog; //Mostramos dialog con el layout light o modo noche según sea el modo actual.
    dialog = nuevoModo.mostrarDialog( actividad: MainActivity.this, v.getContext(),layoutActivity,layoutActivityDark);
    dialog.create();
    //Identificadmos los botones del dialog.
    botonSalirSi = (ImageView) dialog.findViewById(R.id.botonSalirSi);
    botonSalirNo = (ImageView) dialog.findViewById(R.id.botonSalirNo);
        botonSalirNo.setOnClickListener(new View.OnClickListener() {
            @Override //Si pulsa no, no hacemos nada.
            public void onClick(View v) { dialog.dismiss(); }
        });
        botonSalirSi.setOnClickListener(new View.OnClickListener() {
            @Override //Si pulsa si, terminamos todas las actividades y salimos de la app.
            public void onClick(View v) { finishAffinity(); }
        });
    //Mostramos el dialog.
    dialog.show();
}
});
```

- Dialog: Al igual que en el caso anterior, se dan las opciones de salir de la aplicación con “finishAffinity()” o bien, cerrar dialog con “dismiss()”.



```
botonLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Se crea intent para poder trasladar el flujo a la nueva Activity.
        Intent i = new Intent( packageContext: MainActivity.this, login.class);
        startActivity(i);
    }
});

//Al pulsar en matricúlate, nos lleva al Activity Matricúlate.
botonMatriculate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent( packageContext: MainActivity.this, matriculate.class);
        startActivity(i); //Intent nos lleva a activity matriculate.
    }
});
```

- botonLogin: Si pulsa en “Entrar”, Intent dirige el flujo a login.
- botonMatriculate: Si pulsa en “Matricularse”, Intent dirige a matriculate.

```
        botonDarkMode.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Si nos encontramos en modo noche, activa modo día.
                if (esModoNoche) {
                    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);
                    Toast.makeText(getApplicationContext(), text: "Modo noche desactivado", Toast.LENGTH_SHORT).show();
                } else { //Si estamos en modo día, activamos modo noche.
                    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES);
                    Toast.makeText(getApplicationContext(), text: "Modo noche activado", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
```

- esModoNoche: True si es modo noche, y por tanto con AppCompatDelegate, activamos modo día (NIGHT_NO). En caso contrario, activamos, modo noche (NIGHT_YES).

- Clase “Matriculate”:

Clase que engloba la interfaz para potenciales estudiantes, con opciones para ofertas en grados/másteres, y envío de credenciales demo para probar la aplicación de forma externa.

```
        //Al pulsar muestra info de los grados en navegador dispositivo.
        botonGrado.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Uri uriGrados = Uri.parse(urlGrados);
                matriculate.this.startActivity(new Intent(Intent.ACTION_VIEW,uriGrados));
            }
        });

        //Al pulsar muestra info de los másteres en navegador dispositivo.
        botonMaster.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Uri uriMasteres = Uri.parse(urlMasteres);
                matriculate.this.startActivity(new Intent(Intent.ACTION_VIEW,uriMasteres));
            }
        });
```

- ACTION_VIEW: Acción de Intent, que nos permite tras convertir la URL en URI con “parse”, mostrar dicha URL en el navegador del móvil.

```
botonProbarApp.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        AlertDialog dialog; //Cargamos layout en función del modo actual.
        dialog = nuevoModo.mostrarDialog( actividad: matriculate.this, v.getContext(),layoutActivity,layoutActivityDark);
        dialog.create();
        //Identificamos los botones reales del layout.
        ImageView botonEnvio = (ImageView) dialog.findViewById(R.id.botonEnviarPruebaApp);
        ImageView botonCancelar = (ImageView) dialog.findViewById(R.id.botonCancelarPruebaApp);
        TextView usuarioEnviarPw = (TextView) dialog.findViewById(R.id.envioPassTemporal_tv);

        botonEnvio.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Obtenemos el correo del usuario.
                String cadena = usuarioEnviarPw.getText().toString();
                //Validamos si el formato del correo es correcto.
                if (!validarCorreo(cadena)){
                    Toast.makeText(dialog.getContext(), text: "Error. Correo no válido.",Toast.LENGTH_SHORT).show();
                }else{
                    //Si el formato es correcto, enviamos usuario temporal al correo registrado en el campo.
                    nuevoServidor.envioPassOlvidoUsuario(PassServidor,cadena);
                    dialog.dismiss();
                }
            }
        });
```

- botonProbarApp: Se captura el correo electrónico introducido y se valida con "validarCorreo". Si cumple el patrón, se utiliza "envioPassOlvidoUsuario" para enviar credenciales temporales al correo electrónico y así poder entrar con el usuario "temporal".

```
    //Método para comprobar si el correo cumple con el patrón establecido lógico.
    private boolean validarCorreo (String correo){
        Pattern patronEmail = Patterns.EMAIL_ADDRESS; //Utilizamos patrón desde patterns.
        return patronEmail.matcher(correo).matches(); //comprobamos con el patrón.
    }
}
```

- Pattern: Se utiliza "EMAIL_ADDRESS" para comprobar el patrón típico del correo, y así validar que es el formato correcto. No se valida si el correo existe por no incluir "webs externas" para no crear dependencia.

```
//Al pulsar abre cliente correo electrónico usuario para pedir información.
botonInfo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        matriculate.this.startActivity(new Intent(Intent.ACTION_SENDTO).setData(Uri.fromParts( scheme: "mailto",correoUdima,  fragment: null)));

    }
});
```

- ACTION_SENDTO + "mailto": Permite abrir la aplicación cliente de correo electrónico e incluir el correo de contacto, para petición de información.

A7.2: Paquete “ui”.

- Clase BienvenidaFragment:

Fragmento destinado a contener la interfaz de bienvenida del estudiante, con accesos directos a “Notas” y “Trabajos” y un listado dinámico de eventos de la universidad.

```
47          if (getArguments() != null) {
48              //Guardamos el nombre de usuario.
49              nombreUsuarioRecibido = this.getArguments().getString( key: "nombreUsuario");
50              holaUsuario.setText(nombreUsuarioRecibido);
51              servidor nuevoServidor = new servidor(BienvenidaFragment.this.getContext());
52              nuevoServidor.mostrarTablaEventos(nombreUsuarioRecibido,ObtenerEventosServidor,root);
```

- mostrarTablaEventos: Se crea instancia de “servidor” con el contexto propio del fragmento, y con la vista mostrada “root”. Con el usuario, se recibirán los eventos disponibles en la tabla “eventos”.

```
60          ultimasNotas.setOnClickListener(new View.OnClickListener() {
61              @Override
62              public void onClick(View v) {
63                  servidor nuevoServidor = new servidor(BienvenidaFragment.this.getContext());
64                  MatriculaFragment fragmentoMatricula = new MatriculaFragment();
65                  FragmentManager fragmentManager = getActivity().getSupportFragmentManager();
66                  nuevoServidor.cargarFragmentContenedor(nombreUsuarioRecibido,ObtenerMatriculaServidor,fragmentoMatricula,fragmentManager);
67              }
68          });
69
70          //Al pulsar en acceso directo últimas notas, muestra matrícula.
71          ultimosTrabajos.setOnClickListener(new View.OnClickListener() {
72              @Override
73              public void onClick(View v) {
74                  servidor nuevoServidor = new servidor(BienvenidaFragment.this.getContext());
75                  FragmentManager fragmentManager = getActivity().getSupportFragmentManager();
76                  ContenidoFragment fragmentoContenido = new ContenidoFragment();
77                  nuevoServidor.cargarFragmentContenedor(nombreUsuarioRecibido,ObtenerMatriculaServidor,fragmentoContenido,fragmentManager);
78              }
```

- cargarFragmentContenedor: Clase de “servidor” que permite gracias a “fragmentManager” y la URL del código PHP con “StringRequest” POST, mostrar los datos, del fragmento destino, “contenidoFragment” y “matriculaFragment”, cargando los datos propios del usuario de cada sección concreta.
- getActivity(): Se utiliza ya que necesitamos obtener el fragmentManager de la actividad, y estamos dentro de un fragmento.

- Clase CalendarioFragment:

Fragmento destinado a contener el calendario del estudiante, con los eventos propios del mismo en las fechas siguientes al actual, con alertas en el dispositivo y con mensajes emergentes al pulsar día con evento.

```
62          //Creamos nuevo calendario de tipo CompactCalendar.
63          CompactCalendarView calendario = (CompactCalendarView) root.findViewById(R.id.miCompactCalendario_cv);
64          //Establecemos formato con mes completo, año completo separado por guión: "diciembre-2020".
65          SimpleDateFormat formatoMes = new SimpleDateFormat( pattern: "MMMM - yyyy",Locale.getDefault());
66          //Establecemos zona horaria la que dispone el dispositivo por defecto.
67          calendario.setLocale(TimeZone.getDefault(),Locale.getDefault());
68          //Veremos que se muestran los días con una abreviatura de tres letras en la cabecera.
69          calendario.setUseThreeLetterAbbreviation(true);
70          //Obtenemos la fecha actual del dispositivo en el formato que hemos definido.
71          String fechaActual = formatoMes.format(calendario.getFirstDayOfCurrentMonth());
72          //Mostramos justo en la parte superior del calendario.
73          leyendaCalendario.setText(fechaActual);
74          //Obtenemos instancia fecha actual
75          Calendar calendarioActual = Calendar.getInstance();
76          //Creamos nueva tabla dinámica pasando TabletLayout por parámetro.
77          tablaDinamica tablaDinamica = new tablaDinamica(tablaEventos, getContext());
```

- CompactCalendarView: Permite establecer un calendario con eventos, con posibilidad de añadir, eliminar, y marcarlos con distintivo en el mismo.
- SimpleDateFormat: Se utiliza para mostrar el mes y año en la parte superior del calendario, en el "formatoMes" y calculando la fecha actual del dispositivo con "TimeZone.getDefault".
- tablaDinamica: Necesaria para poder mostrar los eventos de forma dinámica.

```
80          //Comprobamos si el estudiante tiene actividades.
81          if (CalendarioFragment.this.getArguments() != null) {
82              try {
83                  //Obtenemos Array de Json del servidor, para tratar cada campo por separado.
84                  datosActividades = CalendarioFragment.this.getArguments().getString( key: "cadenaServidor");
85                  JSONArray actividadesArrayJson = new JSONArray(datosActividades);
```

- datosActividades: Desde área_privada, y "cargarFragmentContenedor" se envía mediante "Bundle" el json "cadenaServidor", que permite recuperar los datos, en este caso de los eventos a utilizar en el calendario.

```
for (int i = 0; i < actividadesArrayJson.length(); i++) {
    jsonFila = actividadesArrayJson.getJSONObject(i);
    //Obtenemos datos concretos del json actual.
    idAsignatura = jsonFila.getString( name: "id_asignatura");
    tipoActividad = jsonFila.getString( name: "tipo_actividad");
    nombre_actividad = jsonFila.getString( name: "nombre_actividad");
    fechaActividad = jsonFila.getString( name: "fecha_entrega");
    //Separamos la fecha en los dígitos año, mes y día, sin los guiones.
    fechaFormato = fechaActividad.split( regex: "-");
    //Obtenemos año, mes y día de la actividad actual.
    añoRecibido = Integer.parseInt(fechaFormato[0]);
    mesRecibido = Integer.parseInt(fechaFormato[1]);
    diaRecibido = Integer.parseInt(fechaFormato[2]);

    //Formamos una nueva instancia de fecha Calendar con los datos obtenidos.
    Calendar calendarioRecibido = Calendar.getInstance();
    calendarioRecibido.set(Calendar.YEAR,añoRecibido);
    //Tenemos que restar una unidad, ya que los meses empiezan en 0 en Calendar.
    calendarioRecibido.set(Calendar.MONTH,mesRecibido-1);
    calendarioRecibido.set(Calendar.DAY_OF_MONTH,diaRecibido);

    //Formatos a mostrar de fecha con barras, actividad, aviso y notificación.
    String nuevaFecha = " "+diaRecibido+"/"+mesRecibido+"/"+añoRecibido;
    String nombreCompleto = tipoActividad+": "+nombre_actividad;
    String eventoDescripcion = "¡Recordatorio!: " + nombreCompleto + " [Fecha Límite]: " + nuevaFecha;
    String aviso = nuevaFecha +" - ¡Último día de entrega!:";
    String detalleNotificacion = "Id. Asig: "+idAsignatura+ " - Actividad: "+tipoActividad+": "+nombre_actividad;
```

- jsonFila.getString: Desde JSONArray, se pueden obtener todos los datos de las actividades con su id en base de datos, con “name”.
- fechaFormato: Se usa Split y el guión, para dividir la fecha mostrada así en base de datos, y poder generar un Calendar con ese día, mes y año. Hay que restar una unidad al mes, ya que Calendar de Android, comienza los meses en 0, en vez de en 1.

```
if (calendarioRecibido.after(calendarioActual)){
    //Obtenemos los milisegundos de la fecha de la actividad a mostrar.
    Long calendarioActualMs = calendarioRecibido.getTimeInMillis();
    //Creamos evento con color, descripción y logitud en ms para mostrar en calendario.
    Event evento = new Event(Color.RED,calendarioActualMs,eventoDescripcion);
    //Añadimos evento al calendario para que se muestre en el día exacto.
    calendario.addEvent(evento);
    //Creamos fila con los datos obtenidos de la actividad.
    datosFila = new String[]{idAsignatura, nombreCompleto, nuevaFecha};
    //Añadimos a la tabla dinámica actividades, la fila datosfila.
    actividades.add(datosFila);

    //Pasamos datos del título y texto que guardaremos con work para notificación en terminal.
    Data datos = new Data.Builder()
            .putString("titulo",aviso)
            .putString("texto",detalleNotificacion)
            .build();

    //Diferencia necesaria para que el sistema pueda calcular el día exacto para la notificación.
    Long alerta = calendarioActualMs - System.currentTimeMillis();
    //Creamos y guardamos la notificación en el día exacto de la actividad.
    guardarNotificacionWork.lanzarNotificacion(alerta,datos);
}
```

- Se dispone de la fecha actual del sistema (calendarioActual) y la fecha de la actividad más antigua desde base de datos, ya que el Array comienza por orden (calendarioRecibido). Sólo en el caso, que el evento desde base de datos sea de fecha igual o posterior a la actual, se añade y se marca en el calendario, si es anterior, ya estaría pasada y no tendría sentido.
- actividades: Array para ir guardando las filas de eventos y posteriormente mostrar en tabla dinámica.
- Bundle datos: Necesario para poder enviar los datos concretos de la alerta que se mostrará en el dispositivo el mismo día de finalizar dicho evento.
- lanzarNotificacion: De la clase "guardarNotificacionWork", permite guardar la alerta en el dispositivo y mostrarla el día concreto, que se calcula, con la diferencia entre la fecha recibida (calendarioRecibido) y la fecha actual (System) en milisegundos, para la exactitud del mismo.

```
148         //Al terminar de recorrer el array, insertamos en tabla dinámica la cabecera y el array de actividades.
149         tablaDinamica.insertarDatosEncabezado(cabecera);
150         tablaDinamica.insertarDatosCeldas(actividades);
```

- insertarDatosCeldas: Se inserta el array de actividades, es decir los eventos en la tabla dinámica, de este modo, se pueden visualizar todos en este formato, siempre y cuando sean posteriores a la fecha actual.

```
153         calendario.setListener(new CompactCalendarView.CompactCalendarViewListener() {
154             @RequiresApi(api = Build.VERSION_CODES.O)
155             @Override
156             public void onDayClick(Date dateClicked) {
157                 //Obtenemos la lista de eventos que han sido guardados en calendario.
158                 List<Event> eventosLista = calendario.getEvents(dateClicked);
159                 //En el día que se pulsa, se recorre en busca de todos los eventos y se muestra con Toast.
160                 for (int indice=0;indice<eventosLista.size();indice++){
161                     Event eventoActual = eventosLista.get(indice);
162                     String nombre = (String) eventoActual.getData();
163                     Toast.makeText(getContext(), nombre, Toast.LENGTH_SHORT).show();
164                 }
165             }
```

- getEvents: Se obtiene la lista de eventos tras pulsar en la fecha concreta. Se obtiene el nombre del evento, con "getData()", y se muestra un mensaje con el mismo ("Toast").

- Clase ContactoFragment:

Clase que contiene el fragmento capaz de poder enviar una incidencia a modo consulta al servicio técnico.

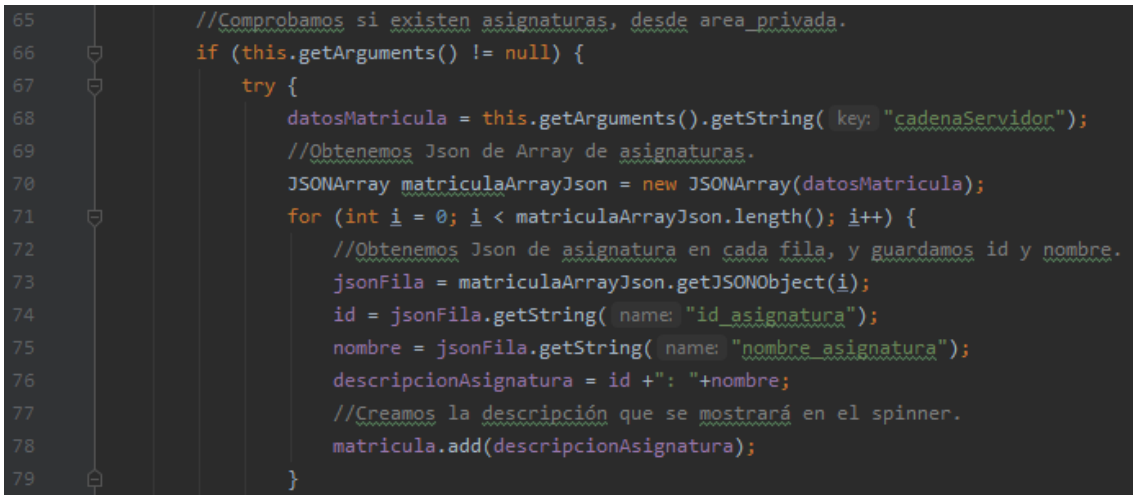
```
botonEnviar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Obtenemos asunto y texto por parte del usuario.
        asuntoContacto = contactoAsunto.getText().toString();
        textoContacto = contactoMensaje.getText().toString();
        //Creamos nueva instancia servidor con el contexto actual.
        servidor nuevoServidor = new servidor(ContactoFragment.this.getContext());
        //Llamamos al método, con los datos a pasar por POST en dicho servidor.
        nuevoServidor.enviarPeticionContacto(nombreUsuarioRecibido,asuntoContacto,textoContacto,servidorContacto);
     }
});

//Si pulsa restablecer, dejamos los campos vacíos, agregando la cadena vacía en ambos.
botonRestablecer.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        contactoAsunto.setText("");
        contactoMensaje.setText("");
    }
});
```

  - botonEnviar: Desde el método “enviarPeticionContacto”, se envía por parámetro los datos del usuario, el asunto, texto de la petición, y la URL para poder realizar el envío a través de mail() por PHP.
  - botonRestablecer: Simplemente, se vacían los “TextView”.

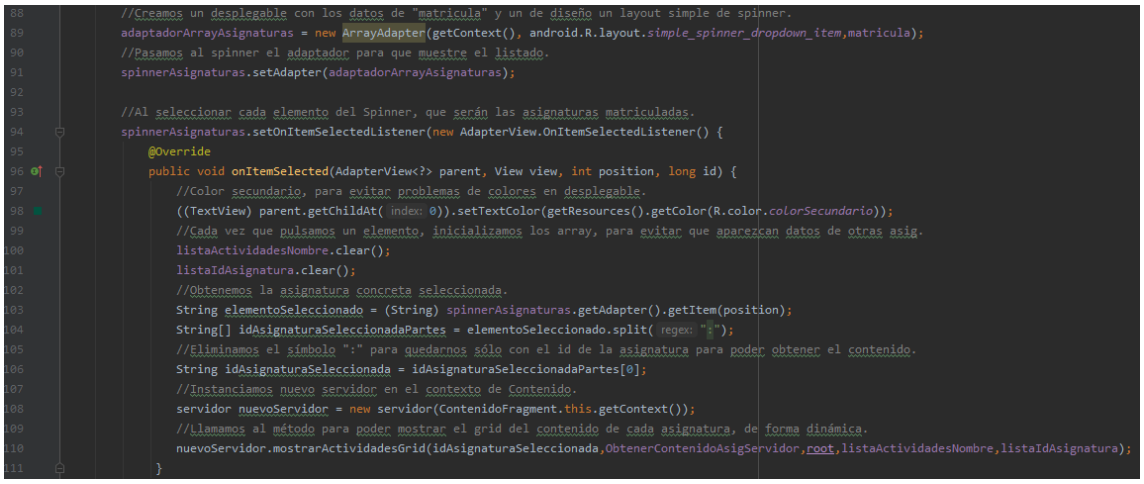
- Clase ContenidoFragment:

Clase que contiene el fragmento que permite mostrar una tabla dinámica con el contenido de las asignaturas, con las opciones de mostrar PDF, compartir y descargar el archivo.



```
//Comprobamos si existen asignaturas, desde area_privada.
if (this.getArguments() != null) {
    try {
        datosMatricula = this.getArguments().getString( key: "cadenaServidor");
        //Obtenemos Json de Array de asignaturas.
        JSONArray matriculaArrayJson = new JSONArray(datosMatricula);
        for (int i = 0; i < matriculaArrayJson.length(); i++) {
            //Obtenemos Json de asignatura en cada fila, y guardamos id y nombre.
            jsonFila = matriculaArrayJson.getJSONObject(i);
            id = jsonFila.getString( name: "id_asignatura");
            nombre = jsonFila.getString( name: "nombre_asignatura");
            descripcionAsignatura = id +": "+nombre;
            //Creamos la descripción que se mostrará en el spinner.
            matricula.add(descripcionAsignatura);
        }
```

- matriculaArrayJson: Array de Json con las asignaturas que nos devuelve desde base de datos en caso de no ser vacío. Gracias a "getString" y a cada id de los campos en base de datos, se guarda el contenido, y se agrega al array "matricula" que llevará las asignaturas matriculadas.



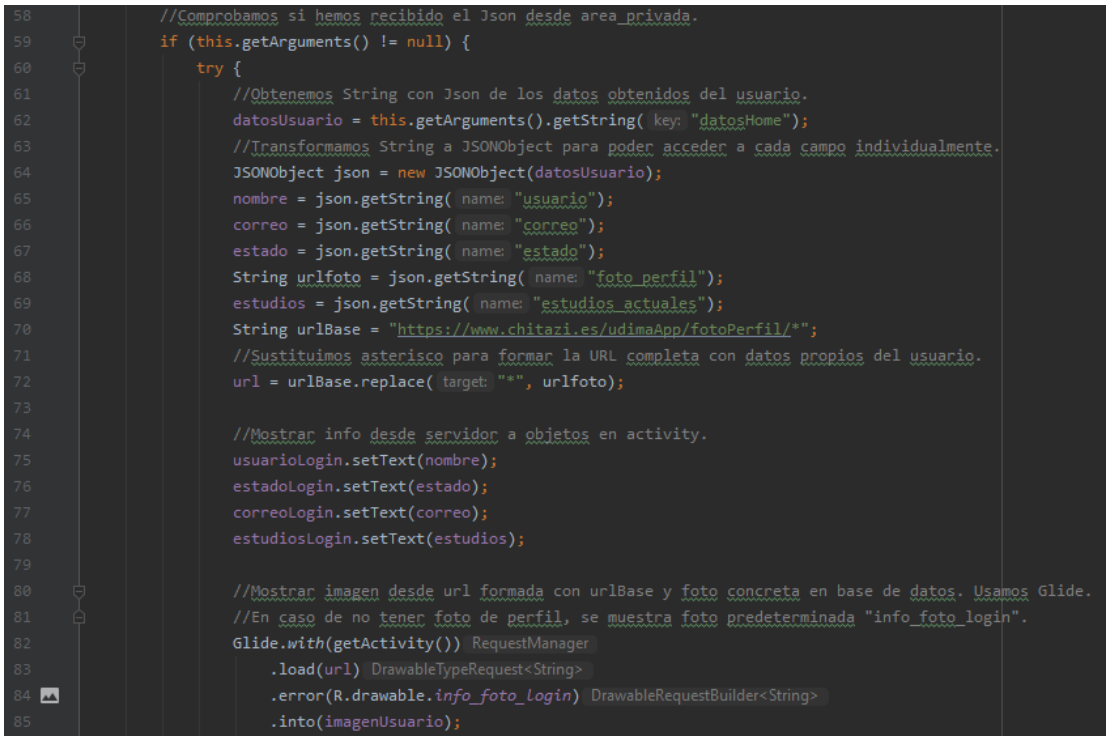
```
//Creamos un desplegable con los datos de "matricula" y un de diseño un layout simple de spinner.
adaptadorArrayAsignaturas = new ArrayAdapter(getContext(), android.R.layout.simple_spinner_dropdown_item,matricula);
//Pasamos al spinner el adaptador para que muestre el listado.
spinnerAsignaturas.setAdapter(adaptadorArrayAsignaturas);

//Al seleccionar cada elemento del Spinner, que serán las asignaturas matriculadas.
spinnerAsignaturas.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        //Color secundario, para evitar problemas de colores en desplegable.
        ((TextView) parent.getChildAt( index: 0)).setTextColor(getResources().getColor(R.color.colorSecundario));
        //Cada vez que pulsamos un elemento, inicializamos los array, para evitar que aparezcan datos de otras asig.
        listaActividadesNombre.clear();
        listaIdAsignatura.clear();
        //Obtenemos la asignatura concreta seleccionada.
        String elementoSeleccionado = (String) spinnerAsignaturas.getAdapter().getItem(position);
        String[] idAsignaturaSeleccionadaPartes = elementoSeleccionado.split( regex: ":");
        //Eliminamos el símbolo ":" para quedarnos sólo con el id de la asignatura para poder obtener el contenido.
        String idAsignaturaSeleccionada = idAsignaturaSeleccionadaPartes[0];
        //Instanciamos nuevo servidor en el contexto de Contenido.
        servidor nuevoServidor = new servidor(ContenidoFragment.this.getContext());
        //Llamamos al método para poder mostrar el grid del contenido de cada asignatura, de forma dinámica.
        nuevoServidor.mostrarActividadesGrid(idAsignaturaSeleccionada,ObtenerContenidoAsigServidor,root,listaActividadesNombre,listaIdAsignatura);
    }
```

- spinnerAsignaturas: Se carga el "adaptadorArrayAsignaturas", que contiene un layout de desplegable simple, y como contenido el array de las asignaturas "matricula". El spinner, permitirá abrir el desplegable y controlar que asignatura de la lista ha sido seleccionada.
- onItemSelected: En cada selección de una asignatura, se inicializa los Array de nombre e id de asignatura para el contenido, ya que en caso contrario, se mostrarían sobrepuestos contenidos de asignaturas que no son la seleccionada. Con el método "getItem" y la "posición" seleccionada de la asignatura, se obtiene el texto completo de la actividad. Usando "Split" y el delimitador ":" entre id de contenido y descripción, se filtra el "id del contenido" necesario, para pasarlo al grid con el método "mostrarActividadesGrid" y de esta forma, en esta vista concreta, poder establecer las funcionalidades de descarga, visualización y compartir, propias de cada contenido. Se utilizan dos listados de array String: listaActividadesNombre y listaIdAsignatura, ya que GridView, realiza con cada índice, un elemento de cada Array, y de ahí, dos listas con índices diferentes.

- Clase HomeFragment:

Clase perteneciente al fragmento Home, que permite mostrar los datos propios del usuario y, además, llevarnos a la actividad “cambio_pw” ya analizada anteriormente.

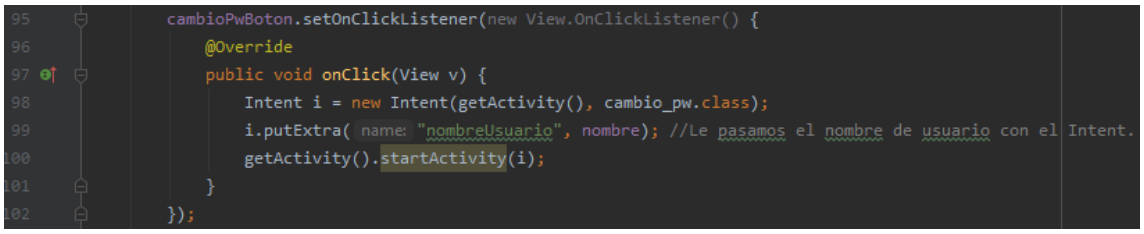


```
//Comprobamos si hemos recibido el Json desde area_privada.
if (this.getArguments() != null) {
    try {
        //Obtenemos String con Json de los datos obtenidos del usuario.
        datosUsuario = this.getArguments().getString( key: "datosHome");
        //Transformamos String a JSONObject para poder acceder a cada campo individualmente.
        JSONObject json = new JSONObject(datosUsuario);
        nombre = json.getString( name: "usuario");
        correo = json.getString( name: "correo");
        estado = json.getString( name: "estado");
        String urlfoto = json.getString( name: "foto_perfil");
        estudios = json.getString( name: "estudios_actuales");
        String urlBase = "https://www.chitazi.es/udimaApp/fotoPerfil/*";
        //Sustituimos asterisco para formar la URL completa con datos propios del usuario.
        url = urlBase.replace( target: "*", urlfoto);

        //Mostrar info desde servidor a objetos en activity.
        usuarioLogin.setText(nombre);
        estadoLogin.setText(estado);
        correoLogin.setText(correo);
        estudiosLogin.setText(estudios);

        //Mostrar imagen desde url formada con urlBase y foto concreta en base de datos. Usamos Glide.
        //En caso de no tener foto de perfil, se muestra foto predeterminada "info_foto_login".
        Glide.with(getActivity()) RequestManager
            .load(url) DrawableTypeRequest<String>
            .error(R.drawable.info_foto_login) DrawableRequestBuilder<String>
            .into(imagenUsuario);
```

- getArguments().getString (key: “datosHome”): Se recibe el Json con los datos desde área privada, gracias al nombre de usuario. A partir de ahí, se obtienen los elementos según id en base de datos. En particular, necesitamos la url de la foto, y se forma reemplazando el “*” con replace en “urlBase” por el nombre de la foto en base de datos, y así se obtiene el enlace concreto, de la ruta completa, dónde está la foto en el servidor. Después, con la librería “Glide” y dicha “url”, se muestra la foto de perfil.



```
cambioPwBoton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent(getActivity(), cambio_pw.class);
        i.putExtra( name: "nombreUsuario", nombre); //Le pasamos el nombre de usuario con el Intent.
        getActivity().startActivity(i);
    }
});
```

- Intent para poder pasar a “cambio_pw” en caso de pulsar el botón “Cambiar contraseña”.

```
iconoCamara.setOnClickListener(new View.OnClickListener() {
    @RequiresApi(api = Build.VERSION_CODES.R)
    @Override
    public void onClick(View v) {
        //El destino es la carpeta Pictures.
        File destinoFoto = getContext().getExternalFilesDir(Environment.DIRECTORY_PICTURES);
        try {
            //Formamos el archivo con el nombre de usuario y extensión png.
            File foto = File.createTempFile(nombre, suffix: ".png",destinoFoto);
            //Formamos la dirección interna de directorio dónde se guardará la foto, privada.
            Uri fotoUri = FileProvider.getUriForFile(getContext(), authority: "com.udima.udimaapp",foto);
            //Con MediaStore activamos la opción de realizar foto.
            Intent tomarFoto = new Intent (MediaStore.ACTION_IMAGE_CAPTURE);
            //Con la opción EXTRA_OUTPUT nos permite guardar la foto en directorios del teléfono.
            tomarFoto.putExtra(MediaStore.EXTRA_OUTPUT,fotoUri);
            startActivityForResult(tomarFoto, check_foto);
            //String rutaFoto = foto.getAbsolutePath();
            iconoCamara.setImageURI(tomarFoto.getData());
        } catch (IOException e) {
            e.printStackTrace(); //Mostramos excepción del sistema en caso de ser efectiva la acción.
        }
    }
});
```

  - getExternalFilesDir: Establece el directorio DIRECTORY_PICTURES como destino.
  - createTempFile: Estable un nuevo “File” temporal con extensión “png” el nombre del usuario y el destino mencionado.
  - MediaStore.ACTION_IMAGE_CAPTURE: Acción para utilizar cámara del dispositivo y capturar foto.
  - MediaStore.EXTRA_OUTPUT: Guarda la foto creada temporal, en el directorio, autorizando previamente la identidad “com.udima.udimaapp”.

- Clase MatriculaFragment:

Clase que contiene el fragmento, dónde se muestran las asignaturas matriculadas, la media del expediente y el progreso en forma de gráfico de los créditos aprobados/pendientes.

```
//Comprobamos los datos desde area_privada. Si no son nulos.
if (this.getArguments() != null) {
    try {
        datosMatricula = this.getArguments().getString( key: "cadenaServidor");
        //Creamos Array de Json con String recibido desde area_privada.
        JSONArray matriculaArrayJson = new JSONArray(datosMatricula);
        for (int i = 0; i < matriculaArrayJson.length(); i++) {
            //Obtenemos el Json concreto con la asignatura y guardamos los datos.
            jsonFila = matriculaArrayJson.getJSONObject(i);
            id = jsonFila.getString( name: "id_asignatura");
            nombre = jsonFila.getString( name: "nombre_asignatura");
            curso = jsonFila.getString( name: "curso");
            creditos = jsonFila.getString( name: "creditos");
            nota = jsonFila.getString( name: "nota");
            //Convertimos String a Double para cálculos con decimales.
            notaAsignatura = Double.parseDouble(nota);
            //Se van sumando todas las notas de las asignaturas para la media.
            notaTotalSuma = notaTotalSuma + notaAsignatura;
            creditosAprobados = jsonFila.getInt( name: "creditos_aprobados");
            datosFila = new String[]{id, nombre, curso, creditos, nota};
            //Añadimos al array de asignaturas cada fila.
            matricula.add(datosFila);
        }

        //Obtener media de la nota y mostrar en TextView. Usar 100d para dos decimales.
        mediaAsignaturas = (Math.round((notaTotalSuma / (matriculaArrayJson.length())) * 100d) /100d);
        //Parseamos a String para poder mostrar en Textview.
        mediaExpedienteTexto = Double.toString(mediaAsignaturas);
        mediaExpediente.setText(mediaExpedienteTexto);
```

- datosMatricula: Si hay asignaturas, se obtiene el Array Json con el contenido, y con los id que tienen los campos en base de datos, se recuperan los valores para mostrar.
- mediaAsignaturas: Tras acabar el recorrido del Array, la media es, suma de todas las notas de las asignaturas “notaTotalSuma”, entre el total de las asignaturas, que lo ofrece la longitud del Array de Json. Se utiliza 100d, para obtener dos decimales (nótese que es Double). Se convierte a String para poder mostrar en el TextView central.

```
//Creamos nueva tabla dinámica, añadimos cabecera y datos contenidos en matricula.
tablaDinamica tablaDinamica = new tablaDinamica(tablaAsignaturas, getContext());
tablaDinamica.insertarDatosEncabezado(cabecera);
tablaDinamica.insertarDatosCeldas(matricula);
```

- tablaDinamica: Se crea la nueva tabla dinámica de asignaturas, con la cabecera definida y el array de asignaturas por filas, contenido en “matricula”.

```
        grafico.getDescription().setEnabled(false);
        //Sin etiquetas.
        grafico.setDrawEntryLabels(false);
        //Mostrar porcentaje en gráfico.
        grafico.setUsePercentValues(true);

        //Calculamos los créditos que restan, teniendo en cuenta los créditos aprobados desde base de datos.
        creditosPendiente = creditosTotal - creditosAprobados;

        //Creamos un Array con los campos que tendrá el gráfico.
        ArrayList<PieEntry> detalleCreditos = new ArrayList<>();

        //Mostramos leyenda y los valores que se asignan a cada parte zona del gráfico.
        detalleCreditos.add(new PieEntry(creditosAprobados,  label: "% Total de créditos aprobados"));
        detalleCreditos.add(new PieEntry(creditosPendiente,  label: "% Total de créditos pendientes"));

        //Evitamos mostrar texto en el gráfico.
        PieDataSet valoresCreditos = new PieDataSet(detalleCreditos,  label: "");

        //Ancho de separación entre ambas franjas del gráfico.
        valoresCreditos.setSliceSpace(6f);
        valoresCreditos.setSelectionShift(0.2f);
        //Mostramos color verde créditos aprobados y rojo sin aprobar, en gráfico.
        valoresCreditos.setColors(new int[]{R.color.colorSecundario, R.color.colorPrincipalPlus}, getContext());

        //El color del valor del porcentaje, en blanco.
        PieData datos = new PieData((valoresCreditos));
        datos.setValueTextSize(20f);
        datos.setValueTextColor(Color.WHITE);

        //Mostramos el gráfico con los datos.
        grafico.setData(datos);

        //Devolver vista principal.
        return root;
```

- Piechart gráfico: Combinado con “PieEntry” que permite establecer las variables a mostrar en el gráfico “creditosAprobados” y “creditosPendiente”, usando porcentaje, y estableciendo los colores corporativos de dicho gráfico. Posteriormente, se establecen los datos, con setData.

- Clase ProfeFragment:

Clase que engloba el fragmento dónde se muestran los profesores que están asignados a las asignaturas que han sido matriculadas por el usuario concreto. Se utiliza GridView con un layout personalizado dónde se muestra la foto, correo electrónico, teléfono, nombre y asignatura de cada profesor, permitiendo las funciones de contacto.

```
48          //Comprobamos si hay profesores.
49          if (this.getArguments() != null) {
50              try {
51                  datosProfesores = this.getArguments().getString( key: "cadenaServidor");
52                  //Recibimos el Array de profesores.
53                  JSONArray profesoresArrayJson = new JSONArray(datosProfesores);
54                  for (int i = 0; i < profesoresArrayJson.length(); i++) {
55                      //Obtenemos cada profesor, cada fila.
56                      jsonFila = profesoresArrayJson.getJSONObject(i);
57                      //Obtenemos cada dato concreto y añadimos en su array correspondiente.
58                      nombreAsignatura = jsonFila.getString( name: "nombre_asignatura");
59                      listaProfesoresAsignatura.add(i,nombreAsignatura);
60                      nombreProfesor = jsonFila.getString( name: "nombre_profesor");
61                      apellidosProfesor = jsonFila.getString( name: "apellidos_profesor");
62                      listaProfesoresNombre.add(i, element: nombreProfesor+" "+apellidosProfesor);
63                      correoProfesor = jsonFila.getString( name: "correo_profesor");
64                      listaProfesoresCorreo.add(i,correoProfesor);
65                      telefonoProfesor = jsonFila.getString( name: "telefono_profesor");
66                      listaProfesoresTelefono.add(i,telefonoProfesor);
67                      fotoProfesor = jsonFila.getString( name: "foto_perfil_profe");
68                      //Para obtener la foto del profesor, sustituimos "*" por el nombre de la foto en bbdd.
69                      String url = urlBase.replace( target: "*",fotoProfesor);
70                      listaProfesoresFoto.add(i,url);
71
72                  }
```

- datosProfesores: Igual que en otros fragmentos, gracias a Json, se obtienen los datos concretos a mostrar en Grid. Se utilizan ArrayList<String> por separado, debido a que en GridView, cada vista del layout lo carga independiente y el índice, debe ser el mismo para cada recorrido, y no se puede implementar en un solo Arraylist, debido a las funcionalidades específicas.

```
80          //Creamos un nuevo adaptadorGrid con todos los elementos a tratar dentro de dicha clase, para poder realizar todas las funciones concretas.
81          adaptadorGrid = new adaptadorGrid(getContext(),listaProfesoresNombre,listaProfesoresAsignatura,listaProfesoresCorreo,listaProfesoresTelefono,listaProfesoresFoto);
82          //Añadimos a gridview.
83          gridProfesores.setAdapter(adaptadorGrid);
```

- Se crea un adaptador del Grid, dónde se pasa por parámetro todos los ArrayList, y este adaptador se asigna al Grid “gridProfesores”.

- Clase SalirFragment:

Clase del fragmento Salir, que permite al estudiante salir de su perfil, y, además, conserva la página de bienvenida en caso de no salir de dicha área. Se debe conservar y cargar el fragmento de bienvenida con fragmentManager para conservar los datos concretos.

```
botonCerrarSesionSi.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Se crea Intent para volver a MainActivity, activity principal.
        Intent nuevoIntent = new Intent(getContext(), MainActivity.class);
        //Mostramos aviso de cerrar todas las activity que queden abiertas en el flujo.
        nuevoIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(nuevoIntent);
        //Cerramos actividad y el Intent nos lleva a MainActivity.
        getActivity().finish();
    }
});
```

- FLAG_ACTIVITY_CLEAR_TOP: El Intent que nos lleva a la actividad principal si el estudiante decide salir, necesita eliminar las actividades que queden abiertas en el flujo, y también finalizar la actividad con finish().

```
botonCerrarSesionNo.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        //Necesario para que la pantalla de inicio quede con los datos del usuario activo.
        Bundle datosBienvenida = new Bundle();
        BienvenidaFragment fragmentoBienvenida = new BienvenidaFragment();
        datosBienvenida.putString("nombreUsuario", nombreUsuarioRecibido);
        //Adjuntamos el bundle con el usuario necesario para mostrar perfil bienvenida.
        fragmentoBienvenida.setArguments(datosBienvenida);
        //Se obtiene fragmentManager para poder reemplazar el contenedor central por el fragment bienvenida.
        FragmentManager fragmentManager = getActivity().getSupportFragmentManager();
        fragmentManager.beginTransaction().replace(R.id.contenedor, fragmentoBienvenida).commit();
        //Cerramos dialog.
        dialog.dismiss();
    }
});
```

- beginTransaction().replace: Es necesario, por una parte, crear el fragmento bienvenida, obtener el usuario que se envía desde área privada, y después, en caso de que el usuario no desee salir, reemplazar el contenedor central, por el fragmento bienvenida, con los datos del “nombreUsuario”, para evitar que quede en blanco, sin los datos de usuario.

A7.3: Paquete “utilidades”.

- Clase adaptadorGrid:

Clase que permite utilizar un layout concreto en forma de GridView, para mostrar los profesores y obtener las funcionalidades concretas. Se han separado en Grid y

GridContenido, para establecer las diferencias visuales, en vez de crear clase genérica adaptador.

```
31      //Creación de variables. Contexto y Array de cada campo a mostrar.
32      private Context contexto;
33      private ArrayList<String> listaNombres;
34      private ArrayList<String> listaAsignaturas;
35      private ArrayList<String> listaCorreos;
36      private ArrayList<String> listaTelefonos;
37      private ArrayList<String> listaFotos;
38
39      public adaptadorGrid(Context contexto, ArrayList<String> listaNombres, ArrayList<String> listaAsignaturas,
40                           ArrayList<String> listaCorreos, ArrayList<String> listaTelefonos, ArrayList<String> listaFotos){
41          //Se instancia el adaptador, con el contexto de la actividad y los Array a mostrar.
42          this.contexto = contexto;
43          this.listaNombres = listaNombres;
44          this.listaAsignaturas = listaAsignaturas;
45          this.listaCorreos = listaCorreos;
46          this.listaTelefonos = listaTelefonos;
47          this.listaFotos = listaFotos;
48      }
```

- Constructor adaptadorGrid: El contexto es necesario para poder establecer las funciones desde el fragmento o actividad concreta. Se establece por parámetro todos los campos a usar en forma de ArrayList.

```
50      //Método que obtiene el total de elementos. Utilizamos Size de uno de los arrays, son todos de igual longitud.
51      @Override
52      public int getCount() { return listaNombres.size(); }
```

- getCount(): Se obtiene el tamaño del array. Son todos de igual tamaño.

```
56      //Método para obtener elemento concreto en una posición determinada del array.
57      @Override
58      public Object getItem(int posicion) { return listaNombres.get(posicion); }
```

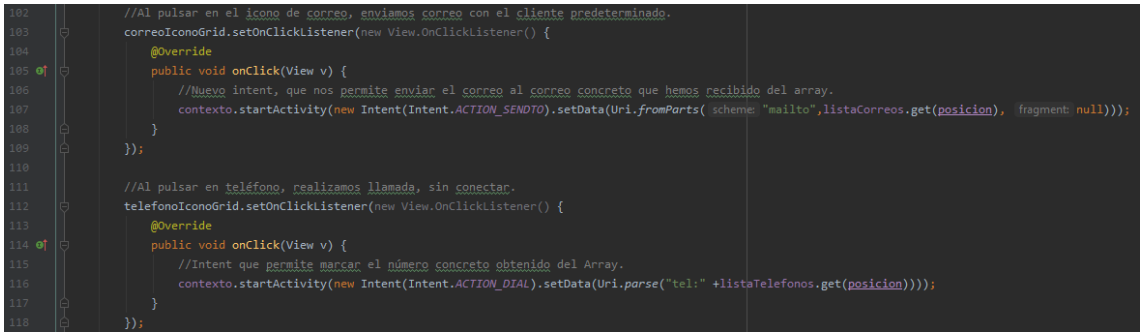
- getItem(): Método para obtener un elemento según su índice.

```
62      //Método para obtener el id del elemento en la posición concreta.
63      @Override
64      public long getItemId(int posicion) { return posicion; }
```

- getItemId(): Obtener identificador del elemento.

```
70      public View getView(int posicion, View vista, ViewGroup vistagrupo) {
71          //Si la vista no es nula.
72          if (vista == null) {
73              //Creamos un infladar de Layout para poder mostrar la vista concreta creada.
74              LayoutInflater inflater = (LayoutInflater) contexto.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
75              //La vista para el grid es elemento_grid, diseñada en los Layouts.
76              vista = inflater.inflate(R.layout.elemento_grid, vistagrupo,  attachToRoot: false);
77          }
```

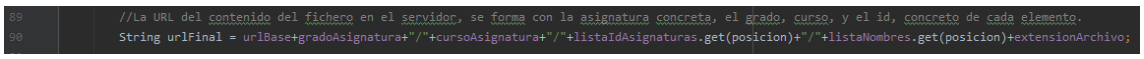
- getView(): Se infla el layout concreto “elemento_grid” de la vista “vista” del fragmento origen.



- ACTION_SENDTO: El array “listaCorreos” con el método get(posición), permite obtener la dirección de correo concreta del profesor en la posición del Array, y mostrarlo en esa vista concreta. Permite enviar el correo con la aplicación predeterminada del dispositivo.
- ACTION_DIAL: Igual que el anterior, pero permitiendo que el dispositivo “incorpore” el número de teléfono de dicho profesor, para la llamada.

- Clase adaptadorGridContenido:

Es necesario extender esta clase, debido a que el layout es diferente, y, además, el número de elementos también, y las funciones también. Si se aprovechara el mismo adaptadorGrid anterior, no se podrían definir las diferentes funcionalidades, ya que depende del layout, vista y fragmento concreto.



- urlFinal: Se forma dependiendo de los datos de la asignatura concreta, y resulta que, el destino, dónde está alojado el archivo PDF del contenido, dependen, sus directorios, del grado matriculado, curso, id de la asignatura y el nombre de la actividad, con extensión pdf, y la url base del servidor. El archivo alojado tendrá esa combinación, y se forma gracias a los datos pasados desde el fragmento con los ArrayList en el constructor de la clase, en la instancia.

```
95          public void onClick(View v) {
96              Intent nuevoIntent = new Intent();
97              //Intent con la url completa, avisando del tipo de fichero pdf.
98              nuevoIntent.setDataAndType(Uri.parse(urlFinal),  type: "application/pdf");
99              contexto.startActivity(nuevoIntent);
100         }
101     });
```

- setDataAndType: Se especifica al Intent que el archivo es de tipo “pdf” y de esta forma, permite abrir el lector de PDF con el contenido del fichero.

```
103 //Descargar archivo al dispositivo.
104 descargaIconoGrid.setOnClickListener(new View.OnClickListener() {
105     @Override
106     public void onClick(View v) {
107         //Utilizamos DownloadManager, en el contexto de la vista.
108         DownloadManager downloadManager=(DownloadManager) contexto.getSystemService(Context.DOWNLOAD_SERVICE);
109         Uri uri=Uri.parse(urlFinal);
110         //Pasamos la Uri de la URL final del archivo alojado en servidor.
111         DownloadManager.Request request=new DownloadManager.Request(uri);
112         //Mostramos la notificación de la descarga en el dispositivo.
113         request.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED);
114         //Creamos el directorio de descargars oculto en los datos de la aplicación y Downloads.
115         String directorioDescargas = Environment.DIRECTORY_DOWNLOADS + "/";
116         //Establecemos el destino creado, creándose el archivo con el nombre de la actividad terminado en ".PDF".
117         request.setDestinationInExternalFilesDir(contexto, directorioDescargas, subPath: listaNombres.get(posicion)+extensionArchivo);
118         //Ponemos en cola la petición para la descarga, que comenzará en cuanto esté disponible.
119         //Si no está disponible, mostrará un error típico de DownloadManager, ya contemplado.
120         downloadManager.enqueue(request);
121     }
122 });
```

- DownloadManager.Request: Permite con el contexto y el servicio DOWNLOAD, descargar a partir de la Uri que hemos formado anteriormente. Se establece el directorio “directorioDescargas” con el nombre del fichero, su extensión, y en el directorio oculto raíz + Downloads. Se utiliza VISIBILITY para mostrar el mensaje típico cuando se descarga el archivo, para notificar al usuario.

```
125     comparteIconoGrid.setOnClickListener(new View.OnClickListener() {
126         @Override
127         public void onClick(View v) {
128
129             //Para evitar problemas con los espacios, convertimos cada espacio a su código "%20".
130             String cadena = urlFinal.replace( target: " ", replacement: "%20");
131             //Mostramos un título para la descarga.
132             String titulo = "Descarga el contenido Udima";
133             Intent nuevoIntent = new Intent();
134             //Creamos Intent para compartir enlace de descarga. Útil para enviar a otros estudiantes.
135             nuevoIntent.setAction(Intent.ACTION_SEND);
136             //Será un enlace en texto plano.
137             nuevoIntent.setType("text/plain");
138             //Indicamos título y detalle de la descarga.
139             nuevoIntent.putExtra(Intent.EXTRA_SUBJECT,titulo);
140             nuevoIntent.putExtra(Intent.EXTRA_TEXT,cadena);
141             //Mostramos el Intent con el detalle del mensaje de Compartir, para dar mayor usabilidad y claridad.
142             contexto.startActivity(Intent.createChooser(nuevoIntent, title: "Compartir con..."));
143         }
144     });
```

- urlFinal.replace: Se reemplazan espacios por el símbolo “%20”, ya que, al compartir, en ciertas aplicaciones como los navegadores, no entienden los espacios y necesitan ser traducidos para que en enlace final sea mostrado correctamente.
- ACTION_SEND: Indicando al Intent, que es texto plano, para enviar el enlace a compartir, con un título y texto para que aparezca a la hora de compartirlo (EXTRA_SUBJECT, EXTRA_TEXT).

- Clase guardarNotificacionWork:

Clase que permite guardar una notificación de un evento en la fecha calculada entre la actual y la del evento en milisegundos. Una vez llegada la fecha del evento, el dispositivo recuperar ese trabajo (work) y lanza la notificación de aviso al usuario.

```
35      //Constructor, con el contexto y los parámetros para Worker.
36      public guardarNotificacionWork(Context contexto, WorkerParameters parametrosWorker){
37          super(contexto,parametrosWorker);
38      }
```

- WorkerParameters: Parámetros necesarios para construir el Worker según el contexto concreto del fragmento o actividad.

```
40  //Método para lanzar la notificación.
41  public static void lanzarNotificacion (long duracion, Data datos){
42
43      //Creamos la notificación, con la duración que calculamos en calendario y datos pasados "titulo" y "texto".
44      OneTimeWorkRequest notificacion = new OneTimeWorkRequest.Builder(guardarNotificacionWork.class)
45              .setInitialDelay(duracion, TimeUnit.MILLISECONDS)
46              .setInputData(datos)
47              .build();
48
49      //Encolamos el trabajo del worker con la notificación.
50      WorkManager manager = WorkManager.getInstance();
51      manager.enqueue(notificacion);
52  }
```

- OneTimeWorkRequest: Se construye una instancia de “guardarNotificacionWork”, que incluye la diferencia en milisegundos para que el sistema sepa la fecha dónde lanzar la notificación, y los datos, un tipo de “Bundle”, que se envía a la clase “doWork()”, para poder enviar la notificación personalizada.

- WorkManager: Instancia del manejador del trabajo, que permite ir encolando (“enqueue”) cada notificación como un trabajo a recordar en la pila de trabajos generales.

```
    //Método que captura los datos de la notificación, y permite definir los parámetros de la misma.
    public Result doWork() {
        //Recogemos datos que hemos pasado desde calendario, con detalle actividad.
        String titulo = getInputData().getString( key: "titulo");
        String texto = getInputData().getString( key: "texto");
        //Se crea un canal de notificación, con los parámetros definidos en las variables.
        NotificationChannel canal = new NotificationChannel(idCanal,descripcionCanal,importancia);
        //Permitir que sea luminoso.
        canal.enableLights(true);
        //Establecer color principal para notificación. Depende de versión Android.
        canal.setLightColor(R.color.colorPrincipal);
        //Habilitar vibración.
        canal.enableVibration(true);
        //Establecer icono en la notificación.
        notificacion.setSmallIcon(R.mipmap.ic_launcher_round);
        //Mostrar texto del aviso.
        notificacion.setTicker("Aviso importante");
        //Si no activamos autoCancel, no desaparece de la pantalla, es molesto.
        notificacion.setAutoCancel(true);
        //Establecemos título y detalle actividad recogidos en la parte superior.
        notificacion.setContentTitle(titulo);
        notificacion.setContentText(texto);
        //Establecemos canal y grupo.
        notificacion.setChannelId(idCanal);
        notificacion.setGroup(grupo);
        //Dejamos el sumario en las notificaciones del terminal.
        notificacion.setGroupSummary(true);
        //Utilizamos el Manager para utilizar el sistema del dispositivo y mostrar la notificación en ese canal.
        NotificationManager manager = (NotificationManager) getApplicationContext().getSystemService(Context.NOTIFICATION_SERVICE);
        manager.createNotificationChannel(canal);
        //Construimos.
        manager.notify( id: 1,notificacion.build());

        //Devolvemos el resultado de la notificación únicamente cuando es correcto y se ha llevado a cabo todo de forma correcta.
        return Result.success();
    }
}
```

- getInputData(): Se obtienen los datos pasados desde el manager visto anteriormente.
- NotificationChannel: Canal para la notificación, con su id, descripción e importancia (IMPORTANCE_HIGH, que permite que sea una notificación sonara y visual, y que quede en la barra de notificaciones, acaparando el canal del dispositivo frente a otras alertas).
- Se establece icono, colores, texto y detalle de la notificación.
- Context.NOTIFICATION_SERVICE: El método desde el manager creado, para poder mostrar la notificación creada en nuestro canal. Se realiza el “createNotificationChannel” y “notify” notifica.

- Clase modo:

Clase que permite comprobar si la aplicación está en modo noche o no y, además, devolver un dialog con un layout preconfigurado según el modo concreto.

```
//Devolvemos el AlertDialog con un layout concreto, según el modo nocturno o día.
public AlertDialog mostrarDialog(Activity actividad, Context contexto, int layout, int layoutDark){
    this.contextoActual = contexto;
    this.actividadActual = actividad;
    this.layoutActual = layout;
    this.layoutDarkActual = layoutDark;
    //Creamos el constructor del AlertDialog.
    AlertDialog.Builder constructor = new AlertDialog.Builder(contextoActual);

    if (esModoNoche()){
        //Si es modo noche, cargamos layoutDark.
        vistaActual = actividadActual.getLayoutInflater().inflate(layoutDarkActual, root: null);
    }else{
        //Sino Layout normal.
        vistaActual = actividadActual.getLayoutInflater().inflate(layoutActual, root: null);
    }
     //Pasamos la vista al constructor.
    constructor.setView(vistaActual);
    //Creamos la vista con el layout en el AlertDialog.
    AlertDialog dialog = constructor.create();
    //Se indica fondo transparente para evitar bordes del layout.
    dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));

    //Devolvemos dialog al contexto.
    return dialog;
}
```

  - AlertDialog.Builder: Constructor que permite construir el dialog en el "contextoActual" que recibe por parámetro, y permite establecer la "vistaActual" inflada con el layout concreto, dependiendo si es modo noche o no, comprobado en "esModoNoche()". Se establece "TRANSPARENT" pare evitar bordes o fondos secundarios.

```
//Comprobamos si es modo noche.
public boolean esModoNoche(){
    boolean esNoche = false;
    //Obtenemos valor por defecto que tiene el dispositivo.
    valorModo =  AppCompatDelegate.getDefaultNightMode();
    //En nuestro caso, si es 2, es modo noche, devolvemos true.
    if (valorModo == 2){
        esNoche = true;
    }
    //Cualquier otro valor, será false.
    return esNoche;
}
}
```

- valorModo: El modo noche por defecto tiene el valor “2”. Si es así, se devuelve true y se conoce que está en modo noche.

- Clase servidor:

Clase que engloba todas las peticiones al servidor, dónde se encuentra el código PHP que recibe por POST las variables necesarias para devolver los datos necesarios para mostrar en la aplicación. En este apartado, cada petición debe comenzar y terminar, para poder obtener el resultado desde el fichero PHP, por lo tanto, no se detallan las partes comunes de los métodos:

```
61          }, new Response.ErrorListener() {
62              @Override //Controlamos error en el método POST con el servidor.
63              public void onErrorResponse(VolleyError error){
64                  Toast.makeText(contextoActual, error.toString(), Toast.LENGTH_SHORT).show();
65              }
66          }){
67              @Override //Método para pasar los valores por POST al servidor php.
68              protected Map<String,String> getParams() throws AuthFailureError {
69                  Map<String,String> valores=new HashMap<~>();
70                  valores.put("nombreUsuario", usuario);
71                  valores.put("asuntoContacto", asuntoContacto);
72                  valores.put("textoContacto", textoContacto);
73                  return valores;
74              }
75          };
```

- onErrorResponse: Utilizado para capturar y controlar el error en el caso de que el servidor tenga algún tipo de error, bien sea en código, en no disponibilidad, y, por tanto, indicamos al usuario este tipo de problema.
- getParams(): En esta parte, y gracias a “valores”, se especifican los valores que se pasarán por POST al fichero PHP pasado por parámetro en cada método, y que recibirá en las variables, que tienen el mismo nombre, como por ejemplo “nombreUsuario”. Es la parte fundamental de la comunicación, y cada método, necesitará un campo concreto, y que se llame de una forma concreta, idéntica a la que está la variable en el código PHP en el servidor.

```
41          //Al instanciar un nuevo Servidor, pasamos el contexto.
42          public servidor(Context contexto) { this.contextoActual = contexto; }
45
```

- El servidor debe llevar el contexto a la hora de instanciarlo, sino sería imposible conocer dónde se encuentra.

```
46      //Método para enviar petición de contacto. Usamos usuario, asunto y texto del asunto.
47      public void enviarPeticionContacto (String usuario, String asuntoContacto, String textoContacto, String Servidor){
48          //Usamos StringRequest con método POST, pasando el servidor, enviando petición POST al servidor con valores en Map.
49          StringRequest stringRequest = new StringRequest(Request.Method.POST, Servidor, new Response.Listener<String>() {
50              @Override
51              //Controlamos la respuesta del servidor. Es síncrona, debe estar definida en cada método, repitiéndose.
52              public void onResponse(String response) {
53                  if (!response.isEmpty()) {
54                      //Si no es vacío, significa que se ha enviado la petición desde el servidor correctamente.
55                      Toast.makeText(contextoActual, text: "Enhorabuena. Petición enviada con ÉXITO", Toast.LENGTH_SHORT).show();
56
57                  }else{ //En caso contrario, error.
58                      Toast.makeText(contextoActual, text: "Error. Petición NO enviada", Toast.LENGTH_SHORT).show();
59                  }
```

- StringRequest: Se indica que la petición al servidor es de tipo POST y, además, se incluye la dirección del “Servidor”.
- Se envía usuario, asunto y texto de la petición de Servicio técnico desde la app, y si recibe respuesta del servidor, ha sido correcto.

```
82      public void comprobarUsuario (String usuario, String password, String Servidor, Class actividadDestino) {
83          //Usamos StringRequest con método POST, pasando el servidor, enviando petición POST al servidor con valores en Map.
84          StringRequest stringRequest = new StringRequest(Request.Method.POST, Servidor, new Response.Listener<String>() {
85              @Override
86              //Controlamos la respuesta del servidor.
87              public void onResponse(String response) {
88                  if (!response.isEmpty()) {
89                      Toast.makeText(contextoActual, text: "¡Enhorabuena!. Usuario correcto", Toast.LENGTH_SHORT).show();
90                      //Toast.makeText(getApplicationContext(), "El response es: "+response, Toast.LENGTH_SHORT).show();
91                      Intent i = new Intent(contextoActual, actividadDestino);
92                      //String nombre = (String) nombreUsuario.getText().toString();
93                      i.putExtra( name: "pasarNombreUsuario",usuario);
94                      i.putExtra( name: "response",response);
95                      contextoActual.startActivity(i);
96                  }else {
97                      Toast.makeText(contextoActual, text: "¡Error!. Usuario o contraseña incorrecta. ", Toast.LENGTH_SHORT).show();
98                  }
```

- Recibe usuario, password, Servidor para la petición POST y la clase destino. En este caso, en caso de ser correcto usuario/password, se crea un Intent, que nos lleva a la “actividadDestino” pero también, se le pasan el nombre del usuario, clave durante todo el área privada y la respuesta desde el servidor, para poder capturar los datos del Json.

```
119     public void envioPassOlvidoUsuario (String PassServidor,String UsuarioEnvioPass){
120         //Usamos StringRequest con método POST, pasando el servidor, enviando petición POST al servidor con valores en Map.
121         StringRequest stringRequest = new StringRequest(Request.Method.POST, PassServidor, new Response.Listener<String>(){
122             @Override
123             //Controlamos la respuesta del servidor.
124             public void onResponse(String response) {
125                 if (!response.isEmpty()) {
126                     //Si no es vacío, se ha generado nueva contraseña aleatoria y enviada al correo.
127                     Toast.makeText(contextoActual,  text: "Nueva contraseña enviada al mail registrado.", Toast.LENGTH_SHORT).show();
128                 }else{ //Si no existe el usuario, no se envía contraseña aleatoria.
129                     Toast.makeText(contextoActual,  text: "¡Error!. No existe el usuario. ", Toast.LENGTH_SHORT).show();
```

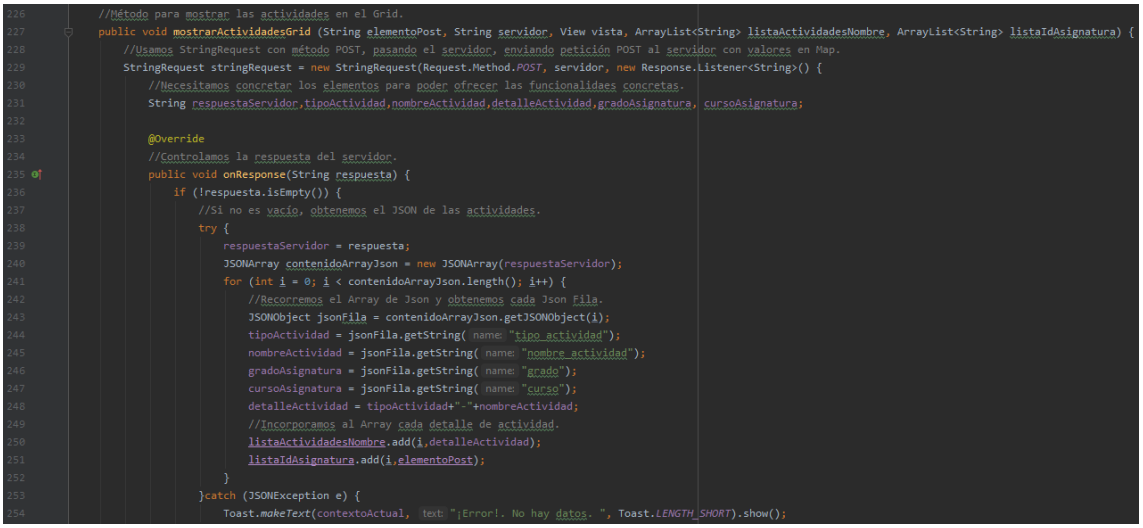
- Recibe el Usuario, y calcula en el fichero PHP la nueva contraseña aleatoria y la envía al correo registrado, en caso de que exista dicho usuario.

```
150 //Método para cambiar contraseña al Usuario.
151 public void cambiarPassUsuario (String cambioPassServidor,String UsuarioEnvioPass, String contraseñaActualUsuario, String nuevaContraseña, Class actividadDestino){
152     //Usamos StringRequest con método POST, pasando el servidor, enviando petición POST al servidor con valores en Map.
153     StringRequest stringRequest = new StringRequest(Request.Method.POST, cambioPassServidor, new Response.Listener<String>(){
154         @Override
155         //Controlamos la respuesta del servidor.
156         public void onResponse(String response) {
157             if (!response.isEmpty()) {
158                 //Si no es vacío, se cambia la contraseña en base de datos y volvemos a Actividad principal.
159                 Toast.makeText(contextoActual,  text: "Contraseña cambiada correctamente.", Toast.LENGTH_SHORT).show();
160                 Intent i = new Intent(contextoActual, actividadDestino);
161                 contextoActual.startActivity(i);
```

- Recibe la contraseña actual y la nueva, y al comprobar en base de datos en el código PHP si son correctas, envía el mensaje de vuelta, del cambio correcto, y en caso contrario muestra el error.

```
187 //Método usado en area privada, para cargar cada fragmento con el envío concreto del JSON de ese fragmento.
188 public void cargarFragmentContenedor (String usuarioFragmento, String servidor, Fragment fragmento, FragmentManager fragmentoManager) {
189     //Usamos StringRequest con método POST, pasando el servidor, enviando petición POST al servidor con valores en Map.
190     StringRequest stringRequest = new StringRequest(Request.Method.POST, servidor, new Response.Listener<String>() {
191         @Override
192         //Controlamos la respuesta del servidor.
193         public void onResponse(String respuesta) {
194             if (!respuesta.isEmpty()) {
195                 //Si no es vacío, usamos Bundle para pasar el Json.
196                 String respuestaServidor = respuesta;
197                 Bundle pasarDatosServidor = new Bundle();
198                 //Usamos la key "cadenaServidor" para poder usarla en cualquier fragmento.
199                 pasarDatosServidor.putString("cadenaServidor", respuestaServidor);
200                 //Se usa el fragmento pasado por parámetro que será del tipo concreto de la Actividad.
201                 fragmento.setArguments(pasarDatosServidor);
202                 //Se usa fragmentManager para reemplazar el contenedor principal (Bloque central de area_privada por el fragmento).
203                 fragmentoManager.beginTransaction().replace(R.id.contenedor, fragmento).commit();
204             }else {
205                 Toast.makeText(contextoActual,  text: "¡Error!. No hay datos. ", Toast.LENGTH_SHORT).show();
```

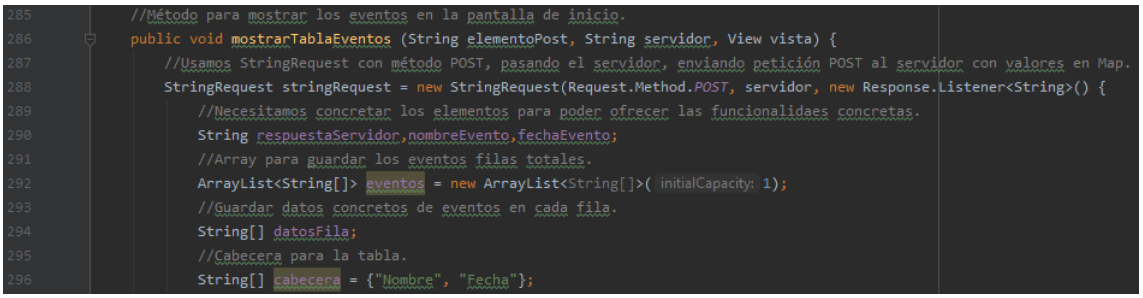
- Método más importante e indispensable, ya que, en el área privada, permite que, al pasar el fragmento, el servidor para la obtención del Json en base de datos y el fragmentManager, en caso de recibir respuesta del servidor, reemplazar el “contenedor” central, por el fragmento específico del área que ha elegido el estudiante y pasar el Json “cadenaServidor” para ser tratado en cada área o fragmento particular.



```
//Método para mostrar las actividades en el Grid.
public void mostrarActividadesGrid (String elementoPost, String servidor, View vista, ArrayList<String> listaActividadesNombre, ArrayList<String> listaIdAsignatura) {
    //Usamos StringRequest con método POST, pasando el servidor, enviando petición POST al servidor con valores en Map.
    StringRequest stringRequest = new StringRequest(Request.Method.POST, servidor, new Response.Listener<String>() {
        //Necesitamos concretar los elementos para poder ofrecer las funcionalidaes concretas.
        String respuestaServidor,tipoActividad,nombreActividad,detalleActividad,gradoAsignatura, cursoAsignatura;

        @Override
        //Controlamos la respuesta del servidor.
        public void onResponse(String respuesta) {
            if (!respuesta.isEmpty()) {
                //Si no es vacío, obtenemos el JSON de las actividades.
                try {
                    respuestaServidor = respuesta;
                    JSONArray contenidoArrayJson = new JSONArray(respuestaServidor);
                    for (int i = 0; i < contenidoArrayJson.length(); i++) {
                        //Recorremos el Array de Json y obtenemos cada Json fila.
                        JSONObject jsonFila = contenidoArrayJson.getJSONObject(i);
                        tipoActividad = jsonFila.getString( name: "tipo_actividad");
                        nombreActividad = jsonFila.getString( name: "nombre_actividad");
                        gradoAsignatura = jsonFila.getString( name: "grado");
                        cursoAsignatura = jsonFila.getString( name: "curso");
                        detalleActividad = tipoActividad+"-"+nombreActividad;
                        //Incorporamos al Array cada detalle de actividad.
                        listaActividadesNombre.add(i,detalleActividad);
                        listaIdAsignatura.add(i,elementoPost);
                    }
                }catch (JSONException e) {
                    Toast.makeText(contextoActual, text: "¡Error!. No hay datos. ", Toast.LENGTH_SHORT).show();
```

- Se recorre el "contenidoArrayJson" con los datos obtenidos tras la petición POST, se obtiene cada campo de la base de datos, y se guarda en variables. En cada vuelta del bucle, se añade el "detalleActividad" y "elementoPost" de la asignatura en cada ArrayList para ello. Seguiramente, se crea el adaptadorGridContenido, con los ArrayList y el contexto, de forma que puedan identificarse y cargarse el layout creado para ese grid.
- completaUrlBase: Método necesario para completar la URL destino del fichero, que como se especificó en secciones anteriores, se forma dinámicamente con el curso y grado específico del estudiante.



```
//Método para mostrar los eventos en la pantalla de inicio.
public void mostrarTablaEventos (String elementoPost, String servidor, View vista) {
    //Usamos StringRequest con método POST, pasando el servidor, enviando petición POST al servidor con valores en Map.
    StringRequest stringRequest = new StringRequest(Request.Method.POST, servidor, new Response.Listener<String>() {
        //Necesitamos concretar los elementos para poder ofrecer las funcionalidaes concretas.
        String respuestaServidor,nombreEvento,fechaEvento;
        //Array para guardar los eventos filas totales.
        ArrayList<String[]> eventos = new ArrayList<String[]>( initialCapacity: 1);
        //Guardar datos concretos de eventos en cada fila.
        String[] datosFila;
        //Cabecera para la tabla.
        String[] cabecera = {"Nombre", "Fecha"};
```

- Método similar al anterior, pero adaptado a una tabla dinámica, necesaria a la hora de mostrar, por ejemplo, las asignaturas en la sección de "Mi matrícula. No se detalla más.

```
345         //Se usa Volley para poder enviar datos al servidor con POST, con add stringRequest.
346         RequestQueue requestQueue = Volley.newRequestQueue(contextoActual);
347         requestQueue.add(stringRequest);
```

- Librería Volley necesaria para obtener la respuesta del servidor y poder comunicarnos de modo POST (en este caso) y obtener la respuesta, gracias a añadir el "stringRequest" que se crea al comienzo de cada método. Es necesario el contextoActual.

- Clase tablaDinamica:

Clase concreta que permite crear una tabla de forma dinámica, dependiendo del número de elementos, de forma, que no se limita a un número fijo de éstos.

```
34      //Constructor, con un TableLayout y el contexto.
35      public tablaDinamica(TableLayout tabla, Context contexto){
36      this.tabla = tabla;
37      this.contexto = contexto;
38      }
```

- Al instanciar, necesitamos TableLayout en el diseño y el contexto.

```
40      //Método para insertar los datos en el encabezado.
41      @RequiresApi(api = Build.VERSION_CODES.M)
42      public void insertarDatosEncabezado(String[] filaInicial){
43          this.datosCabecera = filaInicial;
44          indiceColumna = 0;
45          //Creamos nueva fila, inicial, de ahí índice 0.
46          creaNuevaFila();
47          //Mientras no lleguemos al final del array.
48          while (indiceColumna < datosCabecera.length) {
49              //Creamos celda y mostramos texto en TextView.
50              creaNuevaCelda();
51              celda.setText(datosCabecera[indiceColumna]);
52              //Detalle en color negro y negrita.
53              celda.setTextColor(contexto.getColor(R.color.black));
54              celda.setTypeface(Typeface.DEFAULT_BOLD);
55              //Añadimos a la fila el efecto de la tabla.
56              fila.addView(celda,crearEfectosCeldas());
57              //Sumamos el índice para que las próximas filas empiecen debajo.
58              indiceColumna++;
59          }
60      //Añadimos a la tabla, tras finalizar el recorrido de los datos cabecera.
61      tabla.addView(fila);
62  }
```

- indiceColumna: Índice que marca el comienzo de la cabecera, ya que es la primera fila. Se añaden en dicha posición los elementos de la cabecera, "datosCabecera" y se definen las propiedades como el color (negro), letra en negrita y los efectos de la separación entre celdas, con el método "crearEfectosCeldas()".

```
64      //Método para insertar los datos en cada celda.
65      @RequiresApi(api = Build.VERSION_CODES.M)
66      public void insertarDatosCeldas(ArrayList<String[]> filaDatos){
67          this.datosCeldas = filaDatos;
68          int indiceArrayList = 0;
69          String[] filaActual;
70          String datoCeldaActual;
71          //Recorremos desde el índice 1, ya está la cabecera, hasta el final del Array de las celdas.
72          for (indiceFila=1; indiceFila<=datosCeldas.size(); indiceFila++){
73              //Creamos nueva fila, y obtenemos índice de esa fila actual.
74              creaNuevaFila();
75              filaActual = datosCeldas.get(indiceArrayList);
76              //En cada fila actual, rellenamos los datos en cada celda de la fila.
77              for(indiceColumna=0; indiceColumna<datosCabecera.length; indiceColumna++){
78                  datoCeldaActual = filaActual[indiceColumna];
79                  creaNuevaCelda();
80                  celda.setText(datoCeldaActual);
81                  celda.setTextColor(contexto.getColor(R.color.colorSecundario));
82                  fila.addView(celda,crearEfectosCeldas());
83              }
84              //Avanzamos posición.
85              indiceArrayList++;
86              //Añadimos fila con todas sus celdas a la tabla.
87              tabla.addView(fila);
88          }
89      }
```

- indiceFila: Se establece en 1, ya que la fila 0 es la cabecera. Recorremos el array de filas hasta el tamaño total de filas que tengamos (elementos), y dentro de cada fila, comenzamos desde la columna inicial, la columna 0 ("indiceColumna), para, por un lado obtener los detalles del elemento con "get" y poder mostrarlo en el TextView creado, tras crear la nueva celda, con el método "creaNuevaCelda()". Al final, de cada fila, tras insertar las celdas en las columnas, se añade la fila completa a la tabla, gracias a "addview(fila)".

```
//Método para crear una nueva fila, simplemente una nueva TableRow en el contexto.
private void creaNuevaFila() { fila = new TableRow(contexto); }
```

- Una fila es un "TableRow" en el "TableLayout" definido.

```
//Método para crear nueva celda.
@RequiresApi(api = Build.VERSION_CODES.M)
private void creaNuevaCelda(){
    //Creamos nuevo TextView genérico.
    celda = new TextView(contexto);
    //Centramos en la tabla.
    celda.setGravity(Gravity.CENTER_HORIZONTAL);
}
```

- Cada celda es un TextView que contendrá los datos obtenidos en el Array, es decir, la info de cada elemento a mostrar. Se centra con Gravity.CENTER_HORIZONTAL.

```
//Formato de las celdas en la tabla.
private TableRow.LayoutParams crearEfectosCeldas(){
    //Creamos el efecto de la tabla.
    TableRow.LayoutParams efectosTabla = new TableRow.LayoutParams();
    //Sin márgenes, dejamos comentado por si queremos poner márgenes.
    //efectosTabla.setMargins(2,2,2,2);
    //Tamapo 1 de distancia, para no ser muy gruesa.
    efectosTabla.weight=1;
    //Devolvemos efecto y aplicamos a dónde se llame.
    return efectosTabla;
}
```

- Se establece como efectos, el grosos de 1 punto, en la tabla. Se dejan los márgenes sin valor, aunque se comenta el código para mostrar cómo se realizaría.

A7.4: Código servidor.

A continuación, se muestra de forma breve el código creado en PHP y alojado en el servidor, que hace posible la conexión entre la base de datos y la aplicación.

- comprueba.php

Recibe usuario y contraseña y valida si está o no de forma correcta en base de datos.

```php
<?php

include "config.php";

//Se recibe usuario y password introducido por el estudiante en la App.
$usuario = $_POST['nombreUsuario'];
$contraseñaTexto = $_POST['password'];
//Se Agrega seguridad a la contraseña, cifrado en md5.
$contraseña = md5($contraseñaTexto);

//Comprobamos si la conexión es correcta.
$conexion = abrirConexion();

//Validamos usuario y contraseña en base de datos.
validarLogin($conexion, $usuario, $contraseña);

//Cerramos conexión.
cerrarConexion($conexion);

?>
```

  - md5: La contraseña en base de datos está cifrada en md5. Es necesario convertirla a "md5" para poder compararla con la almacenada.
  - Conexión: En todos los ficheros, es imprescindible, abrir la conexión y cerrarla:

```php
<?php

//Establecer conexión con base de datos.
function abrirConexion(){

//Datos de conexión de la base de datos en el servidor.
$bdservidor = "          ";
$bdusuario = "          ";
$bdpass = "          ";
$basedatos = "chitazie_udima_app";

//Establecer conexión.
$conexion = new mysqli($bdservidor, $bdusuario, $bdpass, $basedatos);

    if ($conexion->connect_error) {
        echo "Error al conectar a la Base de Datos.";
        exit;
    }
    //Devolver conexión a la base de datos, si no hay error.
    return $conexion;
}

//Cerrar conexión de la base de datos.
function cerrarConexion(){
    mysqli_close($conexion);
}
```

- Mysqli: Intenta realizar la conexión con los datos concretos de la base de datos (ocultos). En caso de éxito, devuelve la conexión para poder usarla en las diferentes áreas.
- Mysqli_close: Cierra la conexión de forma segura.

- Olvidarpw:

Con el usuario introducido en la aplicación, comprueba si existe, obtiene su correo electrónico registrado, genera nueva contraseña aleatoria, actualiza ese campo en el usuario y además, envía correo electrónico con dicha nueva contraseña.

```php
//Se define correo del servicio técnico, recibirá copia de correo.
$correoSat = "figuuu@gmail.com";
//Controlar si existe usuario (1) o no (0).
$existeUsuario = 0;
//Se obtiene el nombre de usuario por el estudiante desde la App.
$usuario = $_POST['nombreUsuario'];

//Se abre conexión con base de datos.
$conexion = abrirConexion();

//Se comprueba si existe usuario en base de datos.
$existeUsuario = comprobarUsuario($conexion, $usuario);

//Si existe, se crea contraseña aleatoria y se cambia en base de datos. Se envía correo con las credenciales.
if ($existeUsuario == 1){
    $contraseñaAleatoria = crearContraseñaAleatoria();
    $contraseñaAleatoriaMd5 = md5($contraseñaAleatoria);
    cambiarPassword($conexion, $usuario, $contraseñaAleatoriaMd5);
    $correo = obtenerCorreoUsuario($conexion, $usuario);
    $mensaje = "
    <html>
    <head>
    <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
    <title>Udima APP. La Universidad cercana móvil.</title>
    </head>
    <body>
    <img src='https://chitazi.es/udimaApp/imagenes/cabecera_udima_mail.png' alt='LogoWeb'><br>
    <b>¡Bienvenid@ de nuevo!.</b><br><br> Sus nuevas credenciales son:<br><br>
    <b><ins>Usuario:</ins></b> $usuario<br><b><ins>Contraseña:</ins></b> $contraseñaAleatoria<br>
    <br>¡Gracias por la confianza en UDIMA App!.
    </body>
    </html>";
    enviarCorreo($mensaje,$correo,$correoSat);
```

- Métodos: crearContraseñaAleatoria (crea la nueva contraseña), cambiarPassword (cambia la contraseña actual por la nueva en base de datos), obtenerCorreoUsuario (devuelve el correo registrado del usuario), enviarCorreo (envía el "mensaje" en html con la información de usuario y contraseña, tanto al correo del usuario como al correo del servicio técnico como copia de seguridad.

- Cambiarpw:

Permite, cambiar la contraseña del usuario, siempre que éste introduzca al actual de forma correcta.

```php
<?php
//Incluimos archivo config para poder usar funciones definidas en él.
include "config.php";

//Se obtiene usuario y contraseña actual y nueva contraseña, introducidos por estudiante en App.
$usuario = $_POST['nombreUsuario'];
$contraseñaTexto = $_POST['password'];
$contraseñaActualTexto = $_POST['passwordActual'];

//Se añade encriptación md5 a contraseñas.
$contraseña = md5($contraseñaTexto);
$contraseñaActual = md5($contraseñaActualTexto);

//Se abre conexión con base de datos.
$conexion = abrirConexion();

//Se cambia la contraseñaActual del usuario por contraseña.
cambiarPasswordActual($conexion, $usuario, $contraseña, $contraseñaActual);

//Se cierra conexión.
cerrarConexion($conexion);
```

  - cambiarPasswordActual: Si "contraseñaActual" coincide con la almacenada en el usuario en base de datos, se cambia por "contraseña".

- obtenerMatricula, obtenerProfesores, obtenerActividades, obtenerContenidoAsignatura, obtenerEventos:

El formato es similar en todos los ficheros:

```php
<?php
//Incluimos archivo config para poder usar funciones definidas en él.
include "config.php";

//Se obtiene nombre de usuario introducido por estudiante en App.
$usuario = $_POST['nombreUsuario'];

//Se abre conexión con base de datos.
$conexion = abrirConexion();

//Se obtienen los profesores del estudiante en base de datos.
obtenerProfesores($conexion, $usuario);

//Se cierra conexión.
cerrarConexion($conexion);

?>
```

- Se diferencia en el dato concreto que recibimos desde la aplicación y que, dependiendo de éste, nos permitirá obtener Profesores, contenidos, asignaturas. Cada una llama a su método particular, para evitar confusiones y clarificar más aún el código, y no hacerlo genérico e inteligible.

- Contacto

Permite obtener el asunto y texto de la aplicación en el apartado “Servicio técnico” y enviarlo al soporte con copia al usuario.

```php
<?php
//Incluimos archivo config para poder usar funciones definidas en él.
include "config.php";

//Se obtiene usuario, asunto y texto introducidos en el formulario de contacto de la App.
$usuario = $_POST['nombreUsuario'];
$asunto = $_POST['asuntoContacto'];
$texto = $_POST['textoContacto'];

//Se define correo del servicio técnico, recibirá copia de correo.
$correoSat = "figuuu@gmail.com";

//Se abre conexión con base de datos.
$conexion = abrirConexion();

//Se obtiene el correo del estudiante.
$correoUsuario = obtenerCorreoUsuario($conexion, $usuario);

//Mensaje a enviar.
$mensaje = "
    <html>
    <head>
    <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
    <title>Udima APP. La Universidad cercana móvil.</title>
    </head>
    <body>
    <img src='https://chitazi.es/udimaApp/imagenes/cabecera_soporte_udima_mail.png' alt='LogoWeb'><br>
    <br><b>Petición de contacto</b><br>
    <b>------------------------------</b><br>
    <b>Usuario: </b>$usuario<br><b>Correo electrónico: </b>$correoUsuario<br><b>Asunto: </b>$asunto<br>
    <b>Petición: </b>$texto<br>
    <b>------------------------------</b><br>
    Recibido desde Udima APP.
    </body>
    </html>";

//Se envía mensaje al correo de soporte y del usuario.
enviarCorreo($mensaje,$correoUsuario,$correoSat);

//Se cierra conexión.
cerrarConexion($conexion);
```

- Con el usuario, se obtiene el correo “obtenerCorreoUsuario”, necesario para que le llegue el correo electrónico. Se genera el código “html” de

"mensaje" con los datos concretos recibidos por POST, y con el método "enviarCorreo".

- Enviarpwtemp:

Se crea un método alternativo a "olvidapw" para controlar el usuario temporal de prueba en el caso de la actividad "matricúlate", y así enviar credenciales al introducir correo electrónico un usuario que no es todavía miembro de Udima.

```php
<?php
//Incluimos archivo config para poder usar funciones definidas en él.
include "config.php";

//Controlar si existe usuario (1) o no (0).
$existeUsuario = 0;

//Se define correo del servicio técnico, recibirá copia de correo.
$correoSat = "figuuu@gmail.com";

//Se obtiene el nombre de usuario por el estudiante desde la App.
$usuario = $_POST['nombreUsuario'];

//Se abre conexión con base de datos.
$conexion = abrirConexion();

//El usuario introducido en la App en matricúlate, es el correo para recibir las credenciales temporales.
$correo = $usuario;
//El usuario es el usuario tempora en base de datos.
$usuario = temporal;
//Se crea contraseña aleatoria.
$contraseñaAleatoria = crearContraseñaAleatoria();
$contraseñaAleatoriaMd5 = md5($contraseñaAleatoria);
cambiarPassword($conexion, $usuario, $contraseñaAleatoriaMd5);
$mensaje = "
    <html>
    <head>
    <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
    <title>Udima APP. La Universidad cercana móvil.</title>
    </head>
    <body>
    <img src='https://chitazi.es/udimaApp/imagenes/cabecera_udima_mail.png' alt='LogoWeb'><br>
    <b>¡Bienvenid@ a UDIMA!.</b><br><br>¡Queremos enseñarte como sería tu perfil si te matricularas con nosotros!. <b>¿Te apetece?.</b>
    <br><br> Las credenciales de prueba son:<br><br><b><ins>Usuario:</ins></b> $usuario<br><b><ins>Contraseña:</ins></b> $contraseñaAleatoria<br>
    <br>¡Esperamos que formes parte de nuestro gran equipo!.<br>
    <br>¡Gracias por la confianza en UDIMA App!.
    </body>
    </html>";

//Se envia credenciales usuario temporal al correo introducido en la App.
enviarCorreo($mensaje,$correo,$correoSat);

//Se cambia el correo de dicho usuario para probar cambiarpw en App.
cambiarCorreo($conexion, $usuario, $correo);

//Se cierra conexión.
cerrarConexion($conexion);

?>
```

  - Correo = usuario: El usuario ingresa su correo electrónico, de modo que se asigna dicho correo nombrado como usuario al correo, y como usuario, se especifica el temporal.
  - Se recuerda que en todos los ficheros hay que incluir "config.php", siendo éste el fichero que incluye todos los métodos vistos y toda la configuración de la conexión.

A continuación, se muestra este fichero de configuración "config.php":

- Config:

Incluye los métodos anteriormente nombrados, “abrirConexion()” y “cerrarConexion(). Se muestran brevemente los aspectos más destacados de las funciones:

```php
//Obtener valor de consulta en base de datos.
function obtenerValorBbdd($conexion, $sentencia){
    //Realizar consulta.
    $consulta = mysqli_query($conexion,$sentencia);
    //Obtener filas.
    $numfilas = mysqli_num_rows($consulta);
    mysqli_data_seek($consulta,0);
    //Obtener resultado.
    $valor = mysqli_fetch_array($consulta);
    return $valor;

}
```

- En la gran mayoría de funciones, se necesita obtener el valor concreto de un campo de base de datos. Con la sentencia por parámetro, realizamos la consulta o “query”, devolvemos las filas, si existen, y con “fetch_array”, se obtiene el array de resultados, devolviéndolo en la función. Con el array, conociendo el “id” del campo de base de datos que deseamos obtener, se consigue el valor.

```php
//Comprobar si existe usuario en base de datos.
function comprobarUsuario($conexion, $usuario) {
    $existeUsuario = 0;
    $sentencia = "SELECT * FROM alumnos where usuario = '$usuario'";
    $consulta = mysqli_query($conexion,$sentencia);

    //Si existen registros, existeUsuario = 1.
    if ($registro = mysqli_fetch_assoc($consulta)){
        $existeUsuario = 1;
    }else { //Si no existe usuario, valor 0.
        $existeUsuario = 0;
    }
    return $existeUsuario;
}
```

- Comprobar si existe usuario, y será cierto, si al consultar el contenido global en la tabla “usuario” de dicho id de “usuario”, si devuelve o no datos. En caso afirmativo, se pone “existeUsuario” al valor 1, que será

interpretado por los métodos ya vistos.

```php
//Crear contraseña aleatoria.
function crearContraseñaAleatoria() {
    $caracteres = "0123456789abcdefghijklmnopqrstuvwyxzABCDEFGHIJKLMNOPQRSTUVWYXZ.!*";
    //Se forma contraseña aleatoria con los caracteres definidos, de longitud 9.
    $contraseñaAleatoria = substr(str_shuffle($caracteres), 0, 9);
    return $contraseñaAleatoria;
}
```

- Caracteres: Usados para formar la contraseña. Se pueden añadir más.
- Substr: Genera la contraseña, indicando que son 9 caracteres.

```php
//Enviar correo a las direcciones de email y mensaje indicado por parámetro.
function enviarCorreo($mensaje,$correo,$correoSat){
    $titulo = "Udima APP. La Universidad cercana móvil";
    //Se define cabecera indicando codificación y origen.
    $cabecera = "MIME-Version: 1.0\r\n";
    $cabecera .= "Content-type: text/html; charset=iso-8859-1\r\n";
    $cabecera .= "<Android>\r\n";
    $bool = mail($correo,$titulo,$mensaje,$cabecera);
    $bool = mail($correoSat,$titulo,$mensaje,$cabecera);
    if($bool){
        echo "Mensaje enviado";
    }else{
        echo "Mensaje no enviado";
    }
}
```

- Cabecera: Se especifica cabecera de tipo correo, para Android, y la codificación iso.
- Mail: Función para poder enviar el correo en PHP. Si es cierto, mensaje de éxito y sino de fallo.

```php
//Cambiar contraseña con control de introducir contraseña actual como medida de seguridad.
function cambiarPasswordActual($conexion, $usuario, $password, $passwordActual){

    $sentenciaPass = "SELECT pass FROM alumnos where usuario = '$usuario'";
    $valor = obtenerValorBbdd($conexion, $sentenciaPass);
    $contraseñaUsuario = $valor['pass'];

    //Si la contraseña actual introducida es realmente la actual en base de datos.
    if ($contraseñaUsuario == $passwordActual){
            echo "Contraseña actual correcta";
            //Se cambia la contraseña por la nueva introducida.
            $sentencia = "UPDATE alumnos SET pass = '$password' WHERE usuario = '$usuario'";
            $consulta = mysqli_query($conexion,$sentencia);
            echo "La contraseña se ha cambiado correctamente";

        }else{
            //echo "La contraseña actual no es correcta.";
        }
```

- obtenerValorBbdd: Obtiene el array y al conocer que “pass” es el id del password del usuario, se rescata y se compara con la introducida como actual. En caso de ser igual, se realiza UPDATE, que actualiza la contraseña nueva introducida en “password”.

```php
//Obtener eventos de la universidad.
function obtenerEventos($conexion, $usuario){

    //Obtener id de usuario.
    $sentencia = "SELECT * FROM eventos";
    $consultaEventos = mysqli_query($conexion,$sentencia);
    $numfilas = mysqli_num_rows($consultaEventos);

    //Array para guardar los resultados.
    $arrayResultados = array();

    for ($i=0; $i<$numfilas; $i++){
        //Se obtienen datos del evento.
        $valor = mysqli_fetch_array($consultaEventos);
        $nombre_evento = $valor['nombre_evento'];
        $fecha_evento = $valor['fecha_evento'];

        //Se guarda la fila actual en el array.
        $arrayResultados[] = array('nombre_evento' => $nombre_evento, 'fecha_evento' => $fecha_evento);
    }
    //Se devuelve el array de los eventos.
    echo json_encode($arrayResultados);
}
```

- No se especifican todos los métodos “obtener” ya que siguen la misma línea. Se obtienen las filas a través de la consulta de la sentencia (en algunos casos, se puede observar en el código que es necesario INNER para poder enlazar varias tablas diferentes con una clave común). Se recorre el array de filas, y se guardan los datos en cada variable. Finalmente, se forma un array con todos los datos, y se codifica en un json, con “json_encode”. A lo largo de este informe, se ha comprobado que, en cada fragmento, actividad y área privada, se recibe el Json, y a partir de ahí, bien sea Array de Json o no, gracias al id de cada campo en base de datos, se pueden capturar cada campo concreto, y de esta forma, poder mostrar en la aplicación los valores concretos de matrícula, contenido, perfil, profesores, relacionados siempre con usuario (estudiante).