



UNIVERSIDAD DE GRANADA

PLATAFORMA PILOTO PARA LA GESTIÓN INSTITUCIONALIZADA
DE LOS SERVICIOS DE RELACIONES INTERNACIONALES DE
LA UNIVERSIDAD DE GRANADA

twinX

CLAUDIO LÓPEZ CARRASCOSA

Trabajo Fin de Grado
Grado en Ingeniería Informática

Tutora
Rosana Montes Soldado

E.T.S. INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, a 18 de noviembre de 2020

AUTORIZACIÓN

A GRADECIMIENTOS

ÍNDICE GENERAL

1.	INTRODUCCIÓN	9
1.1.	Motivación	9
1.2.	Alcance	10
1.3.	Objetivos generales	11
2.	CASO DE ESTUDIO: OFICINA DE RELACIONES INTERNACIONALES DE LA FACULTAD DE FILOSOFÍA Y LETRAS DE LA UNIVERSIDAD DE GRANADA	14
2.1.	Sobre la oficina	14
2.2.	Servicio al estudiantado	15
2.2.1.	Estudiantes salientes o <i>outgoing</i>	15
2.2.2.	Estudiantes entrantes o <i>incoming</i>	15
2.3.	Servicios al profesorado (PDI)	16
2.4.	Coordinación con la Oficina de Relaciones Internacionales de la UGR (ORI)	16
2.5.	La base de datos TWINS	17
2.5.1.	El modelo de datos	18
2.5.2.	Funciones implementadas	19
2.5.3.	La interfaz de usuario	20
2.6.	Estudio de necesidades de la ORI-FyL	26
3.	TWINX	30
3.1.	Desarrollo de twinX	30
3.1.1.	Diseño centrado en el usuario	30
3.1.1.1.	Pautas de Accesibilidad	31
3.1.1.2.	Encuesta SUS	31
3.1.2.	<i>Design Thinking</i> , DT	32
3.1.2.1.	Las fases del DT	32
3.1.2.2.	Herramientas del DT	33
3.1.3.	Aplicación de las metodologías	34
3.2.	Personas ficticias	34
3.3.	Entrevistas	35
3.4.	Malla receptora de información	39
3.5.	Descripción de la propuesta	39
3.5.1.	Elección del tipo de metodología de desarrollo	40
3.5.2.	Propuesta de producto	41
3.6.	Planificación	42
3.6.1.	La metodología Scrum	42
3.6.2.	Planificación en sprints	43
3.6.3.	Planificación temporal	45

3.7. Presupuesto	46
4. DISEÑO DE LA INTERFAZ DE USUARIO	48
4.1. Matriz de tareas de usuario	48
4.2. Mapa web (sitemap)	50
4.3. Iconografía (labelling)	51
4.4. Guía de estilo	51
4.5. Bocetos wireframe	53
4.5.1. Wireframes del módulo de gestión	53
4.5.2. Wireframes del módulo de calendario	61
5. DISEÑO TÉCNICO	65
5.1. Listado inicial del producto (product backlog)	65
5.2. Historias de usuario	68
5.3. Modelo de la base de datos	84
5.4. Arquitectura de twinX	87
6. IMPLEMENTACIÓN	89
6.1. Herramientas para el desarrollo de twinX	89
6.2. Creación de la base de datos	91
6.3. Desarrollo del Panel de Control	113
6.4. Desarrollo del módulo de gestión	122
6.4.1. Integración del menú de convenios	122
6.4.2. Integración del menú de estudiantes	131
6.4.3. Integración del menú de expedientes	135
6.4.4. Integración del dashboard	138
6.4.5. Integración de la mensajería	141
6.5. Desarrollo del módulo de calendario	143
Glosario	150

ÍNDICE DE FIGURAS

1.	Convenios	17
2.	Login de TWINS	21
3.	Calendario de TWINS	21
4.	Creación de un nuevo evento en TWINS	22
5.	Avisos de TWINS	23
6.	Eventos sin procesar de TWINS	23
7.	Menú principal de TWINS	24
8.	Tablas implicadas en consultar convenios en TWINS	25
9.	Menú de cambio de DNI	26
10.	Vista de alumnos en TWINS	27
11.	Consentimiento de datos en TWINS	27
12.	Vista de información de asignaturas en TWINS	28
13.	Persona #1	36
14.	Persona #2	37
15.	Persona #3	38
16.	Malla receptora de información	40
17.	Kanban al comienzo del desarrollo	45
18.	Diagrama de Gantt	46
19.	Sitemap de twinX	51
20.	Logo de TWINS	52
21.	Logo de twinX	52
22.	Esquema de colores de twinX	53
23.	Tipografía Segoe UI	54
24.	Wireframe de <i>Dashboard</i>	55
25.	Wireframe de lista de convenios	56
26.	Wireframe de vista de convenio básica	56
27.	Wireframe de vista de convenio avanzada	57
28.	Wireframe de vista de convenio extendida	57
29.	Wireframe de lista de expedientes	58
30.	Wireframe de lista de estudiantes	58
31.	Wireframe de búsqueda	59
32.	Wireframe de detalle de expediente	59
33.	Wireframe del modal de cambio de fase	60
34.	Wireframe de la vista de detalle de un estudiante	60
35.	Wireframe de lista de eventos	61
36.	Wireframe de lista de notificaciones	62
37.	Wireframe de creación de un evento	63

38.	Modelo de la base de datos de twinX	86
39.	Esquema MVC	88
40.	Portal de Gii	90
41.	Menú lateral del panel en twinX	118
42.	Generación de la tabla CRUD de User	119
43.	Menú de usuarios en twinX	120
44.	Vista de un registro en twinX	120
45.	Menú de fases de expedientes en twinX	121
46.	Vista de una fase de expediente en twinX	122
47.	Creación de un convenio en twinX	123
48.	Creación de un convenio en twinX 2	123
49.	Creación de un convenio en twinX 3	124
50.	Menú de convenios en twinX	128
51.	Vista de los convenios en twinX	132
52.	Creación de un estudiante en twinX	132
53.	Menú de estudiantes en twinX	133
54.	Vista de estudiante en twinX	134
55.	Vista de expedientes en twinX	136
56.	Dashboard de twinX	139
57.	Bandeja de entrada en twinX	142
58.	Vista de mensaje en twinX	142
59.	Menú de recordatorios en twinX	144
60.	Diagrama de paquetes de twinX	146

ÍNDICE DE TABLAS

1.	Estadísticas de TWINS	18
2.	Estimación horaria por bloques	47
3.	Presupuesto estimado	47
4.	Matriz de tareas de usuario	49
5.	Iconos de twinX	64
6.	Listado de historias de usuario	67
7.	Tabla de la HU1	68
8.	Tabla de la HU1.1	69
9.	Tabla de la HU1.2	70
10.	Tabla de la HU1.3	71
11.	Tabla de la HU1.4	72
12.	Tabla de la HU1.5	73
13.	Tabla de la HU1.6	73
14.	Tabla de la HU2	74
15.	Tabla de la HU2.1	75
16.	Tabla de la HU2.2	76
17.	Tabla de la HU2.3	77
18.	Tabla de la HU2.4	78
19.	Tabla de la HU2.5	79
20.	Tabla de la HU2.6	80
21.	Tabla de la HU3	81
22.	Tabla de la HU3.1	82
23.	Tabla de la HU3.2	83
24.	Tabla de la HU3.3	83
25.	Tabla de la HU4	84

INTRODUCCIÓN

Desde hace muchos años, las herramientas que hay para gestionar la información en oficinas y secretarías no son –por suerte– las mismas que antes. Y es que, por pequeña que sea la cantidad de información a manejar, hay cada día más y más formas de hacer las cosas mejor y teniendo en cuenta detalles y elementos que intervienen en la actualización haciendo, por lo general, el proceso más efectivo, eficiente y seguro.

Con este trabajo pretendemos dar un empujón a una solución que fue creada para hacer gestiones, pues estas han ido poco a poco tomando forma de problema. Cada vez hay más información que almacenar, se tiene que disponer de la misma de manera más efectiva y, en general, las exigencias van elevándose conforme pasa el tiempo. Por ello, con este trabajo, vamos a proponer una alternativa a modo de punto de partida, no solo para un ámbito en concreto, sino para muchas otras situaciones en que se tengan preámbulos parecidos al que describiremos.

1.1 MOTIVACIÓN

La Facultad de Filosofía y Letras es, con 14 enseñanzas de grado y 2000 asignaturas, una de las facultades con mayor volumen de información de la Universidad de Granada. En el curso 2018/2019 se encuentra en cuarto lugar en cantidad de estudiantes con un total de 4733 [14]. Es más, se tiene un registro de unos 517 estudiantes entrantes y otros 236 salientes hacia y desde España, haciendo algún programa de movilidad durante el curso 2019/2020 en dicha facultad [17].

Resulta obvio pensar en que la relación del tamaño de una entidad con la cantidad de datos que maneja es directamente proporcional. Centrándonos en los datos relacionados a todos los estudiantes que realizan algún programa de movilidad, sería impensable hoy en día manejar semejante cantidad de información sin la ayuda de alguna herramienta que nos permitiera conocer el estado de los mismos (tanto los que vienen del extranjero como los que van a otro país). No obstante, lo cierto es que hasta hace tan solo unos años, desde la Coordinación de Movilidad de la facultad comenzaron con el tratamiento de los datos mediante anotaciones de todo tipo y en cualquier lugar que pudiera parecer seguro y de la forma más organizada posible

(si cabía): desde documentos físicos hasta digitales, haciendo casi imposible la realización de numerosas tareas como son la clasificación, búsqueda y selección de estos datos.

Por ello, trataron de aproximarse a lo que comúnmente se utiliza en estos casos: un sistema de información. Hoy en día disponen de una herramienta que si bien les ha hecho reducir el tiempo de trabajo de semanas a unas pocas horas, sigue teniendo muchas limitaciones que no permiten adaptar la forma natural de interaccionar con la información y, por tanto, con el sistema que la alberga de manera total, poniendo en riesgo la consistencia de los datos, su seguridad y su accesibilidad, dificultando al mismo tiempo su integridad y su distribución.

Así, nace la idea de **twinX**. Se trata de una propuesta cuyo objetivo es, en cierto modo, replicar la funcionalidad de la herramienta actual creada para la Oficina de Relaciones Internacionales de la Facultad de Filosofía y Letras (ORI-FyL), conocida como **TWINS**. Será necesario establecer una serie de propuestas de rediseño (a nivel interno y al de la interfaz de usuario), un cambio de plataforma a la web y la ampliación de su alcance a la totalidad (o la mayoría) de usuarios que interaccionan con el sistema, entre otras cosas.

1.2 ALCANCE

La necesidad de una herramienta como la que ya ha sido creada para albergar la información en la actualidad no es una novedad. Son numerosas las facultades que tienen el mismo problema (aunque quizás no al mismo nivel de dificultad, pues posiblemente manejen una menor cantidad de datos), pero es desde luego una labor que no puede llevarse a cabo con directorios, archivos y demás elementos que complican mucho esta serie de tareas para cualquier cantidad de datos, teniendo en cuenta que el tratamiento que se da a veces no es simple, ni mucho menos.

Son varias las facultades que han mostrado interés por implementar TWINS. La Facultad de Ciencias Políticas trabaja en la actualidad con la herramienta y las facultades de Ciencias Económicas y Empresariales y de Bellas Artes están en ello también. En sus comienzos, la importación de datos era algo bastante tedioso y que llevó tiempo para su adaptación. Sin embargo, poco a poco, han notado una gran mejoría respecto a su anterior ritmo de trabajo.

No obstante, **twinX** no tiene un objetivo tan cerrado a nivel institucional. Se pretende hacer llegar una solución para la gestión de la mayor parte de la información referente a los procesos de movilidad a toda la Universidad de Granada, estableciendo las menores limitaciones posibles, de modo que pueda incluso llegar a usarse en otras universidades que no dispongan de los medios suficientes para organizar y disponer la información tal y como se espera hacer con la herramienta que proponemos.

Es más, algo que dota de atractivo a twinX es acercar la interacción con los datos a la mayor cantidad de usuarios posible que forman parte de los procesos de movilidad estudiantil, lo que presenta probablemente una alternativa muy razonable a las posibles soluciones que las distintas facultades de otras universidades puedan poseer para llevar a cabo sus gestiones en el mismo ámbito.

1.3 OBJETIVOS GENERALES

Este proyecto no es otra cosa que un camino hacia la posibilidad de transportar las gestiones que actualmente están teniendo lugar a una plataforma web para la Universidad de Granada.

Se trata de dejar atrás una aplicación enlazada a una base de datos local, todo ello realizado con Microsoft Access®y obtener a cambio una alta disponibilidad, un buen funcionamiento y una mejoría en la forma de gestionar el trabajo; es decir, se pretende llevar el actual TWINS al navegador, para que se pueda facilitar la posibilidad de acceso desde distintos sitios en lugar de tener los datos centralizados en un único archivo que tiene una infinidad de riesgos de ser borrado, modificado por error, corrompido, etc; todo ello, en aras de mejorar la labor de las distintas oficinas de relaciones internacionales dentro de la UGR.

No es más que la intención de mejorar y perfeccionar lo ya existente: poder consultar la información de una manera más rápida y efectiva, con una mayor comprensión de los menús por los que se pasa en el curso de la interacción con la aplicación, simplificar totalmente la vista, hacerla más atractiva y adaptarla a los sistemas web de hoy en día.

Para poder partir de lo que ya se tiene, se habrá de implementar al menos la misma funcionalidad de TWINS casi al completo. Dejando a un lado los procesos que se consideren redundantes o de poca utilidad, se habrá de incluir aquellos que sean de mayor importancia o representen un mayor manejo de los flujos de información con respecto a la totalidad de la aplicación y se incluirán otros o se modificarán los ya existentes para automatizar el intercambio de datos entre los mismos. Entre todos ellos, cabe mencionar:

- Gestión de [Convenios](#) entre universidades.
- [Acuerdos de estudios](#).
- Expedientes¹ de los estudiantes y sus datos personales.
- Información sobre los [Tutores académicos](#) y su gestión con respecto a los acuerdos de estudios.
- [Alteración de matrícula](#) para [estudiantes Incoming](#).
- Comunicaciones masivas, tanto al estudiantado como a los [Socios](#) (ver: [Nominación](#)).

¹ Con «expedientes» aquí nos referimos más bien a las etapas del estudiante en su proceso de movilidad y no al expediente académico universitario que alberga sus calificaciones.

Y con ello, los demás procesos derivados de estos siete módulos principales que se basan en la modificación, supresión y creación de la información que intercambiarán los distintos procesos de twinX.

Junto a esto y centrándonos en el día a día de la labor de una secretaría de internacionalización, hay un gran trabajo que hacer en cuanto a los acuerdos de estudios. Durante la confección de los mismos se intercambian muchas versiones de estos documentos entre los tutores académicos y los estudiantes que planean su movilidad, y es por ello que proponemos junto con twinX una nueva forma de hacer esto. La idea no es otra que permitir que mediante una identificación, tanto por parte del tutor como por la del interesado/a en hacer una movilidad se pueda confeccionar un acuerdo de estudios online, requiriendo los elementos necesarios para dicho fin y pudiendo, de forma mucho más cómoda, intercambiar las versiones.

Con la idea anterior y en aras de reducir el papeleo al menor posible, una parte importante está en los estudiantes que vienen a estudiar a la UGR o estudiantes *incoming*, como también se les conoce. Tienen que hacer saber a la secretaría qué asignaturas desean cursar y, tras ello, esperar la retroalimentación por parte de la oficina, quien determina si existen plazas disponibles para las asignaturas seleccionadas. Todo ello se haría mucho más fácil si los datos de cada uno de los estudiantes no tuvieran que ser importados a la aplicación, sino que ya existieran en ella misma directamente, pudiendo incluso dar indicaciones a los interesados sobre disponibilidad de plazas y derivados. En el mismo ámbito cabe mencionar los demás documentos que han de entregarse para justificar el fin de estancia en un país de destino, la modificación de los acuerdos de estudios o el [Consentimiento de cesión de datos](#): todo ello se vería reducido a la interacción con la plataforma twinX, ofreciendo a los usuarios una visión general y compacta del curso de los eventos que marcan los procesos de movilidad.

Es más, una de las tareas que más tiempo requiere es la localización de los distintos usuarios que registra TWINS y que se han puesto en contacto con la oficina, de modo que los moderadores que administran la información de los estudiantes tienen que buscar en distintos sitios según sea la naturaleza de la petición que tengan que atender. Por ello, y dado que el correo electrónico -dando por hecho que éste es el medio de comunicación habitual- es un servicio aislado de la aplicación, proponemos una unificación para simplificar no sólo el curso de las operaciones que el usuario tenga que desempeñar, sino también la consistencia de la información y la visualización de la misma, poniendo ante la vista una perspectiva más genérica aún del estudiante, lo que terminaría haciendo más efectivo el trabajo.

Todo ello ha de estar respaldado por una capacidad de la plataforma de albergar una cierta cantidad de conexiones al mismo tiempo. Si bien es verdad que no se contempla soportar una elevada carga de peticiones simultáneas debido a que los accesos se condensarán al comienzo y al final de un cuatrimestre académico, se ha de tener en cuenta que se ha de servir la plataforma a tantos estudiantes como deseen acceder a la misma. No obstante, este no es ni mucho menos el objetivo de este proyecto.

Tan solo se menciona como elemento diferenciador de la mejora que supone antes lo anteriormente desarrollado; es decir, ahora se tendrán que tener en cuenta ciertos factores que permitan la funcionalidad básica del sistema.

2

CASO DE ESTUDIO: OFICINA DE RELACIONES INTERNACIONALES DE LA FACULTAD DE FILOSOFÍA Y LETRAS DE LA UNIVERSIDAD DE GRANADA

Para poder realizar un trabajo efectivo y poder tener en cuenta todas las necesidades, hemos de hacer un análisis previo sobre el caso que nos ocupa. Es por ello por lo que tenemos que comprender cuál es su labor, qué servicios prestan al estudiantado, al profesorado, de qué herramientas disponen, cuáles son sus necesidades y por qué.

2.1 SOBRE LA OFICINA

La labor principal de la oficina es encargarse de la gestión de los programas de intercambio y la movilidad de los estudiantes e incluso de profesores. Como no podía ser de otra manera, proporcionan la información necesaria para los interesados en estos programas, tanto de fuera de la UGR como desde universidades extranjeras. Es más, también revisan convenios existentes con éstas y hacen otros nuevos para ofrecer cada vez más alternativas para poder mejorar nuestra formación.

En su día a día, atienden peticiones y dudas de los estudiantes que participan en alguno de los citados programas; es más, se dedican a asesorar y mostrar todas las alternativas de las que disponen cuando se nos presenta alguna situación complicada, de modo que podamos resolverlo de la forma en que más nos beneficie. Trabajan, en definitiva, con el futuro de los estudiantes, pues la movilidad la desarrollamos con el objetivo de complementar nuestra formación, algo fundamental y abrumador al mismo tiempo cuando se cruzan fronteras y se quiere seguir en el camino de la educación en una universidad que no es la de casa.

La oficina se sitúa junto a la secretaría, en la Facultad de Filosofía y Letras de la Universidad de Granada, en el Campus de Cartuja, y en ella trabajan alrededor de cuatro personas. Es, dadas las cifras que se tienen, una gran cantidad de información las que tan sólo unas pocas personas tienen que manejar con una herramienta que ha sido creada sobre la marcha para facilitar su importante labor; un trabajo que no puede parar ni tolera fallos, pues los estudiantes de movilidad son uno de los pilares fundamentales de la institución.

2.2 SERVICIO AL ESTUDIANTADO

2.2.1 Estudiantes salientes o outgoing

En relación con los estudiantes salientes, en la oficina se encargan de coordinar a los tutores académicos, que son quienes revisan los acuerdos de estudios que los candidatos proponen para iniciar su movilidad. Una vez éstos les han dado el visto bueno, en la oficina revisan cada uno de los mismos para asegurarse de que todo está en orden. Una vez iniciada la movilidad puede darse el caso en que los estudiantes deseen hacer alguna modificación a su acuerdo, debido a algún cambio imprevisto o a que alguna asignatura no resultara ser como se esperaba, todo ello en el destino. En ese caso, el proceso sería el mismo: tendrían que volverse a revisar de nuevo los documentos para comprobar que todo está en orden.

También escuchan casos de estudiantes con problemas particulares y que deben ser examinados con detenimiento, para ofrecer la mejor alternativa, ya sea hablando con los coordinadores de los destinos internacionales o arreglando algún dato en los convenios que haya causado algún inconveniente en la movilidad de algún(a) estudiante. De este modo, las futuras movilidades podrán hacerse con una mayor posibilidad de éxito, sobre todo cuando se trate de convenios nuevos. Son múltiples los casos en que se deba necesitar asistencia: una lengua de impartición de las asignaturas distinta a la esperada, un nivel requerido en un idioma que no se había comunicado al estudiante, etc.

Una vez los estudiantes vuelven de su movilidad, se inicia el proceso de reconocimiento de créditos, para el cual se establecen unas correspondencias entre las calificaciones obtenidas en el destino y las que se van a especificar en su expediente en la UGR. Para ello, esta información debe estar reflejada en un documento oficial y que la secretaría pueda aceptar, por lo que en muchos casos el personal tiene que ponerse en contacto con los responsables en el destino y solicitar los certificados que sean precisos. De esta manera, se puede tener la certeza de que es alguien de confianza quien los remite, ya que se ha tomar muy en serio la veracidad de los mismos.

2.2.2 Estudiantes entrantes o incoming

En cuanto a los *incoming*, el proceso es distinto. Si bien es verdad que se les atiende para problemas similares a los que los estudiantes salientes podrían tener, este grupo viene a la UGR con un acuerdo de estudios previo ya hecho, de modo que es entonces cuando precisan del visto bueno extra de la Oficina, quien les confirma que las asignaturas a las que quieren acceder según lo que establezcan sus acuerdos de estudios tienen plazas disponibles. Es entonces cuando se podrían matricular de las mismas.

Este proceso es conocido como [Alteración de matrícula](#) en el ámbito de la movilidad y se hace de manera manual según las plazas que establece la facultad para cada

asignatura. Es gracias a la ayuda de TWINS que puedan no sólo ver estas asignaciones de una mejor manera, sino que también tienen la posibilidad de generar los horarios para los estudiantes, algo fundamental y que les preocupa mucho cuando vienen a hacer su movilidad a la UGR. Pensemos que el simple hecho de que dos asignaturas tengan lugar en la misma hora supone un cambio inminente. Para ello, los interesados tienen que estudiar cuáles son las alternativas de que disponen, atendiendo al número de plazas restantes en las demás asignaturas, no dejando de lado si la franja horaria en la que se imparte clase es compatible con su horario final.

Como es lógico, tendrán que reportar estos cambios que hagan a sus universidades de destino tal y como éstas establezcan, pues al fin y al cabo el proceso para ellos será el mismo por lo general cuando vuelvan a casa.

2.3 SERVICIOS AL PROFESORADO (PDI)

Al igual que los estudiantes, el Personal Docente Investigador tiene posibilidad de participar en programas de movilidad; es decir, hay convenios que contemplan la acogida del profesorado.

Es en la ORI-FyL donde gestionan y registran estos convenios. Una vez hay constancia de ellos, el personal puede solicitar un programa, al igual que los estudiantes; no obstante, es la Oficina de Relaciones Internacionales central quien se encarga de sus gestiones a lo largo de la movilidad. Esto es, no establecen una interacción directa con la oficina de la facultad.

En su caso no tienen un acuerdo de estudios, sino que tienen un acuerdo de movilidad o *mobility agreement*, donde se pone de manifiesto la intención y los objetivos de los interesados con el programa que desean hacer.

2.4 COORDINACIÓN CON LA OFICINA DE RELACIONES INTERNACIONALES DE LA UGR (ORI)

Todo comienza cuando la ORI envía los datos de las adjudicaciones de las plazas de movilidad a la secretaría de la facultad. Es entonces cuando comienzan los trámites administrativos: se registra a cada estudiante de acuerdo a su destino para comenzar a confeccionar su expediente en base a su acuerdo de estudios, documentos firmados y demás información necesaria. Todo ello tendrá que quedar en conocimiento de la ORI una vez acabe la movilidad.

Establecen una estrecha comunicación también cuando se tratan asuntos económicos en relación a las becas. Con la confirmación de las fechas de llegada al destino y vuelta al origen se hace un contraste con la información presente en el convenio, que es otro acuerdo que los estudiantes se comprometen a cumplir. Se tiene en cuenta si el interesado/a ha realizado la movilidad durante la totalidad del periodo para la

Figura 1: Vista de Convenios

cual estaba prevista. De no ser así, la cantidad económica final tendrá que ser distinta a la prefijada para la ayuda a recibir por los estudiantes.

Por tanto, es de gran importancia guardar toda la información referente al proceso, pues al fin y al cabo la ORI tiene que coordinar que los distintos programas se están llevando a cabo sin incidencias, ya que, al fin y al cabo, es otro organismo asegurador del buen funcionamiento de esta alternativa al estudio continuado en la universidad que tanta importancia tiene hoy en día y que cada vez está más en auge.

2.5 LA BASE DE DATOS TWINS

TWINS alberga desde el curso 2018/2019 un volumen de datos que plasmamos en la tabla 1. Al tratarse de una base de datos en la herramienta ofimática Microsoft Access®, la aplicación consta más bien de una simple capa a modo de interfaz que permite al usuario interactuar con la base de datos. La presentación de los datos se posibilita gracias a la ejecución de consultas preestablecidas (*Query By Example*) que se almacenan y se indica en qué campos ha de mostrarse la información. Cuando se realiza alguna acción que requiera el borrado, inserción o actualización de registros se hace uso de las macros, que son trozos de código que disponen distintos flujos de información entre las tablas implicadas en dicha operación.

1 Tanto *incoming* como *outgoing*

Administración	
Estudiantes ¹	2055
Tutores	58
Convenios	607
Expedientes	5049
Base de datos	
Tablas	108
Relaciones	60
Macros	55
Formularios	152
Consultas QBE	343

Tabla 1: Estadísticas de TWINS

2.5.1 El modelo de datos

El modelo relacional propuesto por Codd es el elegido para el diseño de esta base de datos. Las distintas tablas de la misma se conectan mediante relaciones que establecen restricciones para mantener la consistencia entre los datos.

Las tablas son una forma característica de almacenamiento de este modelo, en contraposición con otras disposiciones de la información usadas por otros sistemas y que, al fin y al cabo, se diseñan de otro modo porque se han de satisfacer unas necesidades distintas.

En este caso, podemos entender que la herramienta TWINS fue creada en Access y, por tanto, en un modelo de datos relacional dado el fácil acceso al usuario no experto en la materia, facilitando la operatividad con las bases de datos.

Es, sin duda, el modelo que más se utiliza aún hoy en día, el cual promete una determinada efectividad siempre y cuando el volumen de información a manejar no sea excesivamente elevado. En este caso, aunque la información que se requiere manipular en la oficina no es fácilmente manejable por personas, sí que aún podemos continuar utilizando este modelo para el desarrollo de la nueva herramienta twinX. Se entiende que no se tendrán más de unos 1000 estudiantes por curso académico (en una sola facultad) y que el número de convenios y tutores no será ni mucho menos parecido, considerando, eso sí, que la cantidad de expedientes de TWINS será aproximadamente el doble que la de los estudiantes para los que se creen dichos registros.

Así pues, el modelo relacional parece ser suficiente para las funcionalidades básicas y necesarias para trabajar en la ORI-FyL que twinX pretende implementar.

No obstante, no olvidemos la intención de extender la funcionalidad del actual TWINS para que los propios estudiantes puedan dejar atrás el constante envío de documentos entre sus tutores académicos por correo electrónico, de modo que puedan dar el salto a una plataforma que implemente una interfaz que les permita prescindir de estos documentos, como el acuerdo de estudios, y trabajar con la información directamente (aunque se posibiliten las conversiones a certificados que puedan ser impresos). En este contexto, se ha de tener en cuenta la forma de tratar los datos que se quiere realizar, que tendrá que adaptarse a este modelo de datos que va a ser usado también en twinX.

2.5.2 Funciones implementadas

■ Funcionalidades básicas

Entre las funciones que hacen que TWINS cobre sentido, destacamos las siguientes:

- **Almacenamiento de información administrativa:** estudiantes, tutores, convenios, expedientes, etc.
- **Asociación de estudiantes con otras entidades:** estudiantes con su tutor, el convenio respecto del cual realizan su movilidad, sus expedientes, etc.
- **Alteración de matrícula:** para poder organizar a los estudiantes *incoming*
- **Envío masivo de correos electrónicos:** posibilita funciones como las **nombraciones** automáticas o la comunicación a los participantes en los programas de movilidad de su tutor académico.
- **Generación de documentos automática:** disposición de los distintos datos en documentos que son entregables a estudiantes para mostrarles un sumario de su situación académica
- **Creación, edición y borrado de los datos dentro de la aplicación**
- **Anotación de futuras modificaciones a documentos ajenos a la oficina:** como son, por ejemplo, los convenios, que quedan registrados en la sede de la UGR y no pueden ser modificados hasta una fecha concreta, fuera del control de la ORI-FyL.
- **Recuperación de información de otros formularios externos a la aplicación:** para el registro de nuevos estudiantes (que es hasta ahora la mejor aproximación a la automatización de dicho proceso de la que se dispone) y, por ejemplo, para registrar datos de estudiantes afectados de alguna manera por la COVID-19.

■ Funcionalidades específicas

Entre las que destacan la generación de informes con fines estadísticos para la oficina, información de asignaturas o menús para gestionar formularios creados por el personal para poder posteriormente transferir la información a la herramienta TWINS.

También disponen de un sistema de alertas en relación con las tareas a realizar por el personal y con una especie de calendario donde se notifican los eventos temporales más próximos a atender.

2.5.3 La interfaz de usuario

Como hemos señalado en anteriores secciones, toda la aplicación yace sobre la herramienta de Microsoft®. Por ello, tanto los datos, como la presentación o vista y el control ejercido sobre los mismos no tiene separación alguna.

Al margen de esto y analizando las ventajas que el uso de Access® nos podría aportar, tenemos una manera de crear una interfaz de usuario de una manera más sencilla de lo normal. Tan sólo basta con arrastrar y redimensionar un botón con el ratón y una casilla para mostrar texto y con unas simples instrucciones tenemos una acción que, al pulsar el nuevo botón, en la casilla podría mostrarse, por ejemplo, cuántos registros almacena una determinada tabla (o varias de ellas) en la base de datos. Las consultas a la base de datos pueden almacenarse y reutilizarse según se quiera. A las interfaces se las conoce como **formularios** en Access®.

También pueden programarse como si de una consulta se tratara, acciones que impliquen la modificación, borrado o creación de registros en la base de datos. A estas piezas de código funcionales se les da el nombre de *macros*.

Una vez mencionadas las posibilidades funcionales, abordemos el uso de las mismas que han hecho para construir TWINS.

Lo primero que nos encontramos tras abrir la aplicación, es una pantalla de login con los distintos usuarios que pueden acceder al sistema (figura 2)

Tras identificarnos, nos topamos, antes de llegar a la pantalla principal, con tres pantallas más:

- El calendario (figura 3) proporciona una vista genérica de todos los eventos que tienen lugar y que conciernen a la ORI-FyL o que tratan de asuntos relacionados con la misma. También se guardan plazos para realizar determinadas tareas. Los eventos del calendario² son públicos a todos los usuarios de TWINS y en la creación de las entradas en él se permite establecer un(a) encargado/a de la tarea (figura 4), de modo que de un vistazo puedan verse los avisos pendientes que se tienen para un determinado evento.
- La pantalla de avisos (figura 5) nos alerta de las acciones que tenemos que realizar con mayor urgencia, dado que han sido programadas para tenerlas listas para una fecha cercana y necesitan de la atención del usuario al que se notifica. Los avisos, al igual que un mensaje cualquiera, tienen emisor y receptor;

² Nótese la especificación de «evento de calendario» o «[Evento de expedientes de TWINS](#)» en las definiciones correspondientes para establecer la diferenciación entre eventos que tienen que ver con la planificación en el calendario y los que componen los [expedientes de TWINS](#)



The screenshot shows the initial login screen of the TWINS application. At the top, there is a header bar with the title "ACCESO A LA APLICACIÓN" on the left and a red "X" button on the right. On the right side of the header is a blue button labeled "Salir de TWINS" and the TWINS logo. Below the header is a table containing user information:

Usuario	Contraseña:	Menu Principal	44009	Unidad L	Bloquear	Desbloquear
CHELO		Menu Principal	44009	Unidad L	Bloquear	Desbloquear
CRISTINA		Menu Principal		Unidad L	Bloquear	Desbloquear
JAVIER		Menu Principal	44065	Unidad L	Bloquear	Desbloquear
JENNIFER		Menu Principal	48956	Unidad L	Bloquear	Desbloquear
JESÚS		Menu Principal		Unidad L	Bloquear	Desbloquear
LAURA		Menu Principal		Unidad L	Bloquear	Desbloquear
MIGUEL ANGEL		Menu Principal	44016	Unidad L	Bloquear	Desbloquear
OLGA		Menu Principal		Unidad L	Bloquear	Desbloquear
ROSA		Menu Principal	20104	Unidad L	Bloquear	Desbloquear
RRII		Menu Principal		Unidad L	Bloquear	Desbloquear
TODOS		Menu Principal		Unidad L	Bloquear	Desbloquear

Figura 2: Vista del login inicial de TWINS con todos los usuarios



The screenshot shows the TWINS calendar interface. At the top, there is a header bar with the title "CALENDARIO" on the left and a red "X" button on the right. On the right side of the header is a blue button labeled "Nueva Tarea" and the TWINS logo. Below the header is a table containing event information:

¿Tengo Avisos?	¿Finalizada?	Filtrar	Hoy es: 15/06/2020	Descripción Corta	% de Ejecución	% de Asignado
ENCARGADO	PENDIENTE	Faltan -137 Días para el fin del EVENTO	CREACIÓN DE CONVENIOS	0		
	PENDIENTE	Faltan -131 Días para el fin del EVENTO	VIDEO TUTORIALES OUTGOING			
	PENDIENTE	Faltan -131 Días para el fin del EVENTO	videos tutoriales INCOMING	0		
ENCARGADO	PENDIENTE	Faltan -108 Días para el fin del EVENTO	Mejorar el rescate de los datos AI	0		100
ENCARGADO	PENDIENTE	Faltan -98 Días para el fin del EVENTO	ASIGNACION DE TUTORES	0		101
	PENDIENTE	Faltan -90 Días para el fin del EVENTO	CORONAVIRUS			
	PENDIENTE	Faltan -7 Días para el fin del EVENTO	CHARLAS INFORMATIVAS			
	PENDIENTE	Faltan 69 Días para el fin del EVENTO				50
	FINALIZADA	Faltan -132 Días para el fin del EVENTO	Llegada incoming	0		100
	FINALIZADA	Faltan -115 Días para el fin del EVENTO	Alteración de matrícula INCOMING 2º Plazo	60		100

Figura 3: Vista del calendario en TWINS con los eventos existentes y visibles por todos los usuarios



Figura 4: Creación de un nuevo evento en TWINS con la posibilidad de asignar tareas a otros usuarios

es decir, hay alguien que los crea y los dirige a otra persona. Cuando se añade un evento al calendario, se especifican ambos actores, de modo que cuando está próxima la fecha de alguna tarea, se crea el aviso y notifica a aquel al cual le ha sido asignada. Normalmente, un usuario solo puede visualizar los avisos que le han sido dirigidos, salvo el administrador, quien puede ver todos los avisos de todos los usuarios aparte de los propios

Por supuesto, también se pueden crear avisos fuera del contexto de los eventos de calendario, lo que también implica la existencia de emisor, receptor y fecha de caducidad. Al margen de la importancia que se le da a la notificación del número restante de días para que una tarea caduque, esta última forma de hacer uso de los avisos podría verse como imaginando al creador, en la oficina, dejando una nota adhesiva en la mesa de su compañero/a.

- Dejando atrás los plazos y los eventos del calendario, otro elemento de gran importancia y con lo que trabajan día a día en la ORI-FyL (y no sólo en un cierto momento del cuatrimestre académico) son los **eventos de expedientes de TWINS**. Pensemos que a diario surgen problemas, inconvenientes o simplemente hay que actualizar la situación de un estudiante. Por ello, la siguiente pantalla tras descartar las dos anteriores es la de eventos (de expediente) sin procesar (figura 6).

En ella, los usuarios de gestión pueden acceder a la información relacionada con el evento, como los demás eventos, los expedientes del alumno para el cual se está mostrando la información, o directamente las fichas del estudiante o el convenio en sí mismas.

Finalmente, llegamos a la pantalla principal (figura 7), de donde parten todos los menús (incluso los que acabamos de ver, que se muestran automáticamente identificarse correctamente en el sistema). En ella, no vemos ninguna información como la presen-

Creado por	Dirigido a	Descripción del Aviso	Nº Días	Fecha Aviso	Estado
MIGUEL ANGEL	CHELO	Mejoras rescate de información de los listados EXCELL OUTGOING - El Nombre del alumno debe aparecer separado - Código del centro que aparece si comillas '009' - Que aparezca el email de contacto del destino INCOMING - Que aparezca la titulación de destino del alumno	- 10 - 20 + + 10	05/07/2020	Pendiente Suspendido Terminado
MIGUEL ANGEL	MIGUEL ANGEL	ya se puede borrar los campos de mail tipo en titulaciones En ficha convenio, al envío de TOR 2018/2019 se hace muy mal, hay que darle una vuelta	- 10 - 11 + + 10	26/06/2020	Pendiente Suspendido Terminado
MIGUEL ANGEL	MIGUEL ANGEL	EL BORRADO DE TITULACIONES VER LA INTEGRIDAD CON CENTROS, TB VER SI SE PUEDE ELIMINAS EL ENLACE CON EMAIL_TIP012345678 DE SEGUIMIENTOS O DE NOMINACION INICIAL Y DE TUTORIZACION	- 10 - 2 + + 10	17/06/2020	Pendiente Suspendido Terminado

Figura 5: Vista de los avisos en TWINS desde el usuario administrador

Figura 6: Menú de eventos sin procesar en TWINS, una pantalla donde los usuarios ven fácilmente el trabajo acumulado

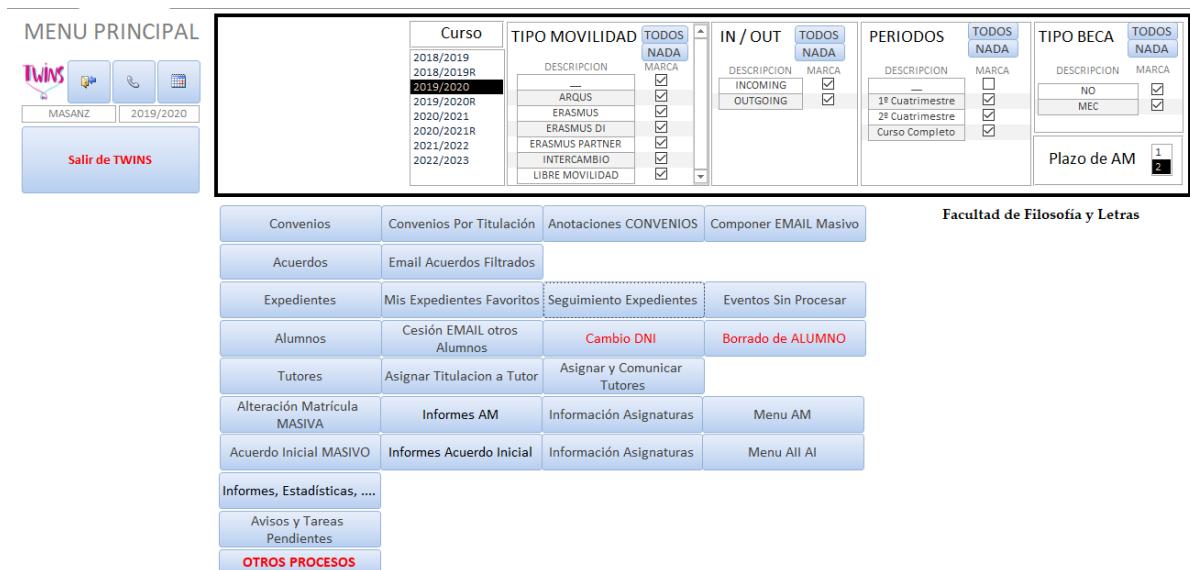


Figura 7: Pantalla principal de TWINS. Desde aquí podemos acceder a todos los menús anteriormente mostrados.

tada en capturas anteriores (ni indicador de notificaciones, mensajes o alertas; nada). Lo que sí podemos apreciar es una selección en la cabecera de la vista principal con la que se filtrará el contenido que aparezca tras acceder a cualquier elemento del menú. Esto es, al solicitar determinada información cuando entremos a cierto menú, sólo se nos mostrarán aquellos datos que estén especificados en dicho cuadro de filtrado. Así, las consultas que se pueden hacer a la base de datos son mucho más cómodas y accesibles a todos los usuarios que no tienen por qué tener conocimiento de estos sistemas.

Si accedemos, por ejemplo, a la vista de convenios que ya hemos mostrado con anterioridad (figura 1), el cuadro de filtrado se nos bloquea (ya hemos hecho una consulta y, para hacer otra, con otros parámetros, tendríamos que retroceder y volver a establecer nuestras preferencias de búsqueda). Se listan los convenios tal y como se han solicitado. El orden puede variarse a través de las flechas en las distintas columnas. Sin embargo y como es lógico, no se muestra toda la información de una tabla. Es más, realmente se están solicitando datos de distintas tablas (figura 8). Este es un ejemplo de cómo la herramienta de creación de formularios de Access nos permite disponer la información de una tabla de forma más concisa que mostrando la tabla en crudo.

Si nos fijamos en las distintas entradas de la lista en la figura 1 de nuevo, correspondiente cada una a un distinto convenio, vemos que se tienen elementos como un botón para acceder al formulario de la vista del mismo, los estudiantes *outgoing* o *incoming* que están de movilidad según ese convenio, y demás elementos que son de relevancia para la entrada en concreto.



Figura 8: Relación de tablas implicadas en la consulta realizada para listar los convenios en TWINS

A efectos de diseño de la interfaz, es apreciable que ciertos campos de cada entrada se presentan con un elemento de lista, también conocido como *dropdown*. No son editables hasta que no se accede a la vista del convenio, en contra de lo que puede dar a entender. Si generalizamos al resto de la aplicación, existen numerosos elementos con una disposición extraña como la que mencionamos, de modo que resultan atípicos para el usuario inexperto, lo que podría llevar a confusiones y una pérdida de confianza en el usuario que interactúe con el sistema, a pesar de que no habrá gran cantidad de gente que comience a utilizar el sistema por primera vez de forma avanzada como es la labor del personal de secretaría, quienes sí lo harán. De todos modos, no podemos dar nada por hecho y la labor de un experto en informática es, en esencia, facilitar y mejorar.

Es por ello por lo que gran parte de este proyecto se enfocará al rediseño de la aplicación de TWINS basado en técnicas de usabilidad. Esto será posible gracias a la aplicación de metodologías actuales y la realización de pruebas reales con usuarios que interaccionarán con el sistema.

Para finalizar, vamos a comentar aspectos genéricos sobre las distintas vistas de la interfaz de TWINS. Es un elemento apreciable el uso de colores vivos y que reclaman la atención del usuario. Éstos suelen ser utilizados mayormente en áreas donde se debe prestar mucha atención y se debe proceder con cuidado a la hora de hacer cambios, pues podría desencadenar a un estado de inconsistencia en el sistema difícil de revertir. Ejemplo de esto es el menú de cambio de DNI (figura 9), donde la cabecera es amarilla. La propia aplicación te impide editar el DNI si para el estudiante seleccionado se tienen registros de información. No obstante, no cabe duda de que se trata de un cambio importante, y por eso también, para acceder a dicho menú, hay que hacer doble click en el botón del menú principal (figura 7), ya que sus letras tienen un color rojo, como pasa con cualquier botón que contiene la aplicación y se resalta de esta manera.

También, como elemento en común cada vez que aparece el nombre de los estudiantes, se marca en rojo si es *outgoing* o en azul si, por el contrario, es *incoming*. Ejemplos de esto podemos encontrarlos en las figuras 10 y 11. En la primera, podemos también apreciar cómo se presenta un color verde en la columna de casillas más a la izquierda cuando el número de acuerdos de estudios activo para los estudiantes es de 0, lo que significa que «no hay peligro» al editar su información. Si por el contrario tuvie-

The screenshot shows a software interface for managing student records. At the top left is a logo for 'TWINS' with sub-options 'MASANZ' and '2019/2020'. To the right is a 'CURSO' dropdown menu with options: 2018/2019, 2018/2019R, 2019/2020 (highlighted in green), 2019/2020R, 2020/2021, 2020/2021R, 2021/2022, and 2022/2023. Below the menu is a 'Filtro' input field and a 'Filtrar' button. To the right is a 'IN / OUT' section with checkboxes for 'DESCRIPCION' (INCOMING checked) and 'MARCAS' (OUTGOING checked). The main area is titled 'ACUERDOS' and contains a table with columns: ACTIVO, TOTAL, IN / OUT, NEW, CURSO, AÑO / S ACUERDO / S, APPELLIDOS, NOMBRE, and DNI / ID. The table lists numerous entries, each with a red 'Cambiar DNI' button. The first few rows show '0' in the ACTIVO column and 'OUTGOING' in the IN / OUT column. The CURSO column shows '2018/2019' or '2018/2019 2019/2020' for most rows. The AÑO / S ACUERDO / S column shows various years from 2018/2019 to 2022/2023. The APPELLIDOS, NOMBRE column lists names like 'Apellidos de Prueba, 2256' through '2275'. The DNI / ID column lists corresponding DNI numbers.

Figura 9: También existe en TWINS un menú específico para el cambio de DNI

ran más de uno, la aplicación mostraría la cifra en amarillo, indicando el alto riesgo de modificar sus datos en caso de hacerlo de forma descuidada. Nótese que no se muestran datos con carácter sensible debido a cuestiones de protección de datos.

Comentemos, en último lugar, que la disposición de estos colores e indicadores no es en vano, no ya a partir de las aclaraciones anteriores, sino también haciendo alusión a la figura 12, donde no se tienen datos importantes que modificar, tan solo se trata de una mera consulta –sin derecho a modificación– a las asignaturas registradas para los estudiantes en la base de datos, no habiendo riesgo de comprometer la consistencia de la información. Es por ello por lo que la paleta de colores, aunque de discutible empleo en toda la aplicación, es muy reducida en esta vista concretamente, pues tan solo se resalta en azul la cabecera, y el resto queda en blanco y negro.

2.6 ESTUDIO DE NECESIDADES DE LA ORI-FYL

En general, la Oficina de Relaciones Internacionales de la facultad dispone de una herramienta que, dado que ha sido creada por un integrante de la secretaría de la facultad con una estrecha relación laboral con los miembros de la oficina, ha sabido cumplir los requisitos operacionales del servicio. Muchas funcionalidades de la misma son muy concretas a las necesidades de la ORI-FyL (informes imprimibles con documentación específica, estadísticas, pegatinas para fundas de plástico, etc.)

Figura 10: Vista de alumnos en TWINS

Figura 11: Menú de gestión del consentimiento de cesión de datos de los estudiantes salientes en TWINS

INFORMACION ASIGNATURAS							
Filtrar		AA_CODIGO / AA_NOMBRE	ACRED LING ??	AA_TITULACION	PLAZAS	VACANTES	S I M U L A C I O N
					VACANTES	Nº ALTAS	Nº BAJAS
	29911M8A	ACÚSTICA MUSICAL	B1-ES	299	0	0	-23
	2851142B	ADQUISICIÓN DEL INGLÉS COMO SEGUNDA LENGUA I	B1-ES	285	9	0	0
	2851142A	ADQUISICIÓN DEL INGLÉS COMO SEGUNDA LENGUA I	B1-ES	285	5	0	0
	2851148A	ADQUISICIÓN DEL INGLÉS COMO SEGUNDA LENGUA II	B1-ES	285	0	2	2
	2851148B	ADQUISICIÓN DEL INGLÉS COMO SEGUNDA LENGUA II	B1-ES	285	5	1	1
	26611D3A	AGREGACIÓN Y MONUMENTALIZACIÓN DEL PAISAJE EN ANDALUCÍA (IV Y III MILENIO A.C.)	B1-ES	266	4	4	4
	9999BELL	ALTA ASIGNATURA FACULTAD DE BELLAS ARTES	B1-ES	900			
	9999CIEN	ALTA ASIGNATURA FACULTAD DE CIENCIAS	B1-ES	900			
	9999EDUC	ALTA ASIGNATURA FACULTAD DE CIENCIAS DE LA EDUCACION	B1-ES	900			
	9999DEPO	ALTA ASIGNATURA FACULTAD DE CIENCIAS DEL DEPORTE	B1-ES	900			
	9999EMPR	ALTA ASIGNATURA FACULTAD DE CIENCIAS ECONOMICAS Y EMPRESARIALES	B1-ES	900			
	9999POLI	ALTA ASIGNATURA FACULTAD DE CIENCIAS POLITICAS Y SOCIOLOGIA	B1-ES	900			
	9999COMU	ALTA ASIGNATURA FACULTAD DE COMUNICACION Y DOCUMENTACION	B1-ES	900			
	9999DERE	ALTA ASIGNATURA FACULTAD DE DERECHO	B1-ES	900			
	9999MEDI	ALTA ASIGNATURA FACULTAD DE MEDICINA	B1-ES	900			
	9999PSIC	ALTA ASIGNATURA FACULTAD DE PSICOLOGIA	R1-ES	900			

Figura 12: Menú de información de las asignaturas en TWINS (en la universidad local, de destino, de cara a los estudiantes entrantes)

No obstante, debido a las limitaciones de la metodología escogida para posibilitar estas gestiones, se ha llegado a un punto en que es muy complicado realizar modificaciones y extensiones a la misma. Por ejemplo, la ORI-FyL destaca la necesidad de un portal directamente conectado a la aplicación principal que posibilite la realización de gestiones por parte de los estudiantes. De este modo, tanto el estudiante como el personal de secretaría ahorrarían tiempo y su interacción no sería íntegramente en el correo electrónico. Esto es también extensible al papel de los tutores académicos.

Actualmente se ha hecho uso de herramientas auxiliares como los Formularios de Google® que permiten volcar datos al formato de hoja de cálculo, con lo que permite a TWINS reconocer la información y que pueda, a través de macros, crear nuevos registros en la base de datos para almacenarlos. Sin embargo, se precisa de la interacción humana para dicha tarea, al igual para muchas otras que se puedan querer automatizar o adaptar, como por ejemplo:

- **Adaptar la mensajería** e implementarla dentro de la aplicación para llevar un mejor control de los casos de los estudiantes.
- **Automatización del seguimiento de las nominaciones** teniendo un portal externo donde otras universidades puedan nominar a sus estudiantes en la UGR.
- **Portal con vista para el estudiante entrante**, posibilitando que haga su propio registro en la plataforma y pueda ver su información en ella.

Al margen de las nuevas características a incluir, no podemos dejar de lado la necesidad de reconstruir un sistema desde cero, para que sea consistente, estable, con alta disponibilidad y fiabilidad y que pueda adaptarse a las necesidades tanto del personal de la oficina como a las del estudiantado. Es por tanto por lo que debemos descartar seguir utilizando la herramienta de Access® dadas sus limitaciones, expuestas en la sección [2.5.1](#).

3

TWINX

Una vez hemos analizado el material ya existente y desde el que partiremos para la construcción de twinX, vamos a ponernos manos a la obra con su desarrollo. Si bien es cierto que no se parte desde cero en cuanto a requisitos, sí que se va a reconstruir la herramienta desde cero, para poder aportar un gran significado a todos los módulos que vayan a formar parte de twinX y así podamos cohesionarlos de una mejor manera.

Para conseguir nuestro propósito, en este capítulo hablaremos, en primer lugar, sobre cómo vamos a usar el **diseño centrado en el usuario** y a aplicar sus metodologías para desarrollar twinX (sección [3.1](#)). A continuación, hablaremos de las **personas ficticias** que podrían usar la aplicación, de las **entrevistas** que se han tenido para posibilitar este proyecto, y describiremos de forma más precisa **nuestra propuesta**, donde elegiremos un tipo de metodología de desarrollo y definiremos las capacidades de la primera versión de twinX. Por último, expondremos la planificación diseñada para realizar los trabajos necesarios (sección [3.6](#)), donde hablaremos del uso que haremos de *Scrum*, el desdoble en *sprints* del desarrollo, el tiempo empleado en el proyecto en su totalidad y el presupuesto del mismo.

[3.1](#) DESARROLLO DE TWINX

Vamos a comenzar estableciendo las ideas principales del proyecto, reafirmando el propósito y viéndolo desde otras perspectivas que, aunque parecen obvias, no siempre se tienen en cuenta. Ello nos permitirá asentar las bases del producto software final y dotarlo de calidad.

[3.1.1](#) *Diseño centrado en el usuario*

Esta disciplina, conocida también como *User-Centered Design* o UCD, destaca por basar las fases del proceso de diseño estableciendo un enfoque constante en aprender del sujeto que utilizará el producto final. Es decir, para la consecución de los objetivos, es necesario tener una retroalimentación constante por parte del usuario final,

que será quien vaya orientando nuestros avances según sea su interacción con lo que vayamos desarrollando.

El proceso dispone de unas fases a seguir, como son:

- **Especificación del contexto de uso:** quiénes usarán el producto, para qué y bajo qué condiciones lo harán.
- **Especificación de requisitos:** identificar los objetivos que tienen que cumplirse para dejar a los usuarios satisfechos.
- **Crear soluciones de diseño:** en distintas etapas, desde un concepto poco definido hasta un diseño completo.
- **Evaluación de los diseños:** a través de las pruebas con los usuarios, de forma ideal.

3.1.1.1 Pautas de Accesibilidad

El no atender a causas de accesibilidad sería no aplicar correctamente, de alguna forma, el tipo de diseño escogido para el desarrollo del producto. Cuando se hacen pruebas, el objetivo no es otro que adaptar el producto para que pueda ser bien utilizado por el mayor número de usuarios posible y de la forma satisfactoria para ellos. Es por ello por lo que tenemos que abrir el abanico y contemplar que pueden ser numerosos los usuarios que necesiten adaptaciones para interactuar correctamente con el contenido web.

Para ello, se propone la utilización de los recursos especificados en *Web Content Accessibility Guidelines* (WCAG) [29]. Con ello, podemos hacer que la experiencia en el sitio web pueda ser mínimamente satisfactoria para todo usuario que necesite hacer uso de la misma. En efecto, se deberán emplear una serie de técnicas, especificadas en su web, para que twinX sea adaptable. Entre ellas, destacamos la viabilidad de la interacción con el teclado, un texto descriptivo para las posibles imágenes que se puedan incluir, regular el contraste, etc. Con este proyecto, la intención es alcanzar el nivel AA (segundo más exigente), para que la gran mayoría de personas con necesidades especiales puedan usar la plataforma sin problema alguno.

3.1.1.2 Encuesta SUS

La encuesta SUS o *System Usability Scale* [3] es una de las encuestas que se pueden utilizar para evaluar la usabilidad de una cantidad de productos o servicios.

Sus únicos 10 enunciados son de las características que más atractiva la hacen, puesto que los usuarios la rellenan de manera rápida y fácil. También destaca por ser de denominación común y tener un bajo coste, además de poder corregirse inmediatamente después de terminar la encuesta. Es más, es una encuesta que puede aplicarse a casi cualquier tipo de interfaz de usuario, por lo que es muy versátil. Finalmente, cabe destacar la facilidad de entender los resultados, puesto que vienen dados en forma de una puntuación en el rango de 0 a 100.

Sobre los enunciado de la encuesta, tienen una escala de 1 a 5 cada uno, siendo 1 el indicador de mayor desacuerdo y el 5 el de mayor acuerdo. También se suele aportar unas pequeñas instrucciones a los candidatos a tomar la encuesta, indicando la necesidad de contestar todas las preguntas y de no pensarse mucho la respuesta a las mismas.

Para probar la eficacia de esta herramienta como elemento conductor en la creación de una interfaz de usuario a la hora de hacer pruebas y recibir *feedback*, se hizo un experimento con una escala con adjetivos y no con números. Los resultados del análisis [3] vieron una mayor desviación de los resultados a la hora de expresar un mayor desacuerdo (adjetivos como «horrible» o «peor que lo imaginable» con números más próximos al 1). No obstante, en general, y gracias a análisis como el mencionado, se tiene evidencia que asegura que este tipo de encuestas es una buena herramienta a tener en cuenta a la hora de recibir información acerca de cuán bueno es nuestro diseño para los usuarios.

Por todo ello, esta será una de las herramientas que utilizaremos en las pruebas para constatar que los objetivos del proyecto se cumplen y cuáles pueden ser las conclusiones.

3.1.2 Design Thinking, DT

El «Pensamiento de Diseño» o *Design thinking* es una técnica de desarrollo que se centra en el usuario, pudiendo detectar y reaccionar ante cambios repentinos en el entorno de los usuarios y sus comportamientos. El objetivo está, mayormente, en abordar problemas con una pobre definición o que no se conocen a fondo para situar al usuario en el centro de todo y poder así enfocar el problema desde otras perspectivas, de manera que se pueda poner la atención en aquello que resulte de mayor importancia para los usuarios [12].

3.1.2.1 Las fases del DT

El proceso tiene unas fases no necesariamente secuenciales, de modo que puedan adaptarse lo mejor posible al proyecto, teniendo incluso la posibilidad de ejecutarse al mismo tiempo.

1. **Empatizar**, tratar de adoptar un conocimiento lo más empático posible del problema que se pretende resolver. Este es un elemento esencial, pues posibilita a los desarrolladores a descartar sus propias primeras conclusiones –erróneas a menudo– y a entrar en materia con la realidad del cliente y sus necesidades.
2. **Definir** las necesidades de los usuarios y sus problemas. Es la fase donde se reúnen y ordenan los elementos obtenidos como resultado de la fase anterior. A partir de entonces, se sintetizan para definir los problemas base que se identifican, los cuales dan pie a la creación de *personas*; esto es, la construcción de perfiles humanos en los cuales centrar el desarrollo.

3. **Idear** y hacer frente a lo que se da por hecho, creando formas alternativas de ver y tratar el problema con soluciones innovadoras, a partir de lo estudiado en las dos fases anteriores.
4. **Prototipado** de las soluciones pensadas, una fase experimental cuyo objetivo es el de encontrar la mejor solución para cada uno de los problemas encontrados. Los desarrolladores han de producir una versión de bajo coste del producto para investigar cuál es el resultado de haber llevado las ideas a la práctica.
5. **Pruebas** con lo obtenido, para analizar si realmente se ha llegado a un buen resultado o, si por el contrario, se ha de retroceder a otra fase para redefinir uno o más problemas.

Como hemos indicado, las fases no siempre siguen el mismo orden. Hay veces que se toman decisiones como la de saltar de la primera fase de empatía a la penúltima de prototipado, probablemente para aclarar las ideas y poder hacer una mejor definición, a través de la muestra de material al cliente que pueda animarlo a dar una mayor retroalimentación. Del mismo modo, si el prototipado no ha ido bien, se puede volver a la tercera y anterior fase, la de construcción de ideas. Incluso podría darse el caso que haciendo pruebas, los desarrolladores nos demos cuenta de que no se ha llevado a cabo una buena ejecución del proceso y sea preciso volver a la segunda etapa de definición de los problemas. Siempre es mejor ir hacia atrás en lugar de comenzar la casa por el tejado, así que toda maniobra que sea apropiada para una mejor construcción del producto y que lo dote de calidad siempre es bienvenida.

3.1.2.2 Herramientas del DT

Hay una serie de actividades o técnicas que nos pueden resultar útiles para llevar a cabo el trabajo de desarrollo con eficacia y que suelen tener éxito. Destacamos las siguientes:

- **Creación de personas:** perfiles ficticios de los distintos usuarios que utilizarán el producto software. Creadas en la fase de definición, no solo se especifica el propósito específico de interacción con el producto a mejorar o la necesidad por que exista el software que se quiere desarrollar. Describimos el contexto del personaje, sus inquietudes y, en definitiva, lo que hay detrás de esa persona en más ámbitos que puedan ayudar a comprender por qué es necesario que se tengan en cuenta ciertas cosas a la hora del desarrollo.
- **Brainstorming:** también conocida como nube de ideas que radican alrededor de un concepto central. Se trata de escribir conceptos que vengan a la cabeza de los intervenientes en la creación del esquema, sin importar los análisis o el futuro que puedan tener en el proceso. Cualquier cosa que tenga que ver con lo que se está tratando es válida, pues lo que aparentemente resulta absurdo podría en muchos casos resolver parcialmente el problema o ayudar a enfocarlo. Así, cuantas más propuestas, mejor se lleva a cabo este proceso, que tiene lugar en la fase de ideación.

- **Prototipado en papel:** la creación de prototipos (en la cuarta fase de prototipado) con un material del que todos disponemos es extremadamente sencilla a la par que útil. Cuando las cosas se plasman en un folio, podemos apreciar matices que no nos venían a la cabeza cuando la idea era tan solo un concepto. Si bien es cierto que depende de las dotes artísticas de la persona que dibuja el prototipo, el hacerlo con papel y lápiz ayuda a volcar la concentración de una manera diferente a como se hace cuando se programa o se diseña con herramientas informáticas.
- **Mapas de experiencia de usuario:** también conocidos como *Customer Journey Maps*, sirven para representar la experiencia de un usuario a lo largo del tiempo. En ellos plasmamos la forma en que un diseño cubre o no las necesidades de un usuario utilizando un producto o servicio. Es justo por eso por lo que estos mapas han de ser lo más descriptivos posibles, representando con gran detalle las acciones y subtareas que tiene que desempeñar un usuario al usar el sistema.

3.1.3 Aplicación de las metodologías

En relación con las fases del DT, podemos diferenciar a través de las cuales ya hemos ido pasando. El comienzo del proyecto vino acompañado de una serie de reuniones que se incluyen en los anexos. En este contexto, la fase de **empatización** se correspondería con las dos primeras reuniones, donde se establecieron un primer contacto y las bases del proyecto, atendiendo al testimonio de los usuarios reales de TWINS, lo que actualmente se está usando para resolver los problemas a los que se tienen que enfrentar en la ORI-FyL. Es más, en la **segunda reunión** se habló de algunas características ideales y que está costando implementar en el actual escenario, lo que podríamos englobar dentro de la **ideación**. Por último, ya en la tercera reunión, se tiene la **definición** de todos los conceptos necesarios para comprender el funcionamiento de TWINS y de la oficina en general. No obstante, a lo largo de esta sección, vamos a continuar esta fase con la definición de más elementos necesarios para llevar a cabo el desarrollo de twinX.

Con esas tres fases cubiertas, se puede dar comienzo a las otras dos: la de **prototipado** y **pruebas**, que serán desarrolladas de forma más extensa en las secciones ?? y ??.

3.2 PERSONAS FICTICIAS

Tal y como hemos comentado en la sección 3.1.2.2, una de las herramientas que nos permiten definir el alcance y las necesidades de la aplicación es la creación de perfiles de personas ficticias, candidatos a utilizar twinX en un futuro. De esta forma, tanto por la parte del desarrollo como por la del interesado en el producto final, pueden no solo hacerse una mejor idea de los objetivos del mismo, sino también justificar su creación.

Vamos a tratar de crear tres personalidades lo más variopintas posibles en aras de enfocarnos más aún en el usuario y dotar de mayor calidad el resultado final, de modo que podamos abarcar por completo el entorno de influencia del problema.

En vista de la persona en la figura 13, Eladio, podríamos pensar que para él, lo óptimo sería desarrollar una app móvil para poder gestionar su movilidad, en caso de que quisiera hacerla. Pertenece a las nuevas generaciones donde apenas se usa el ordenador, hasta llegar a la universidad, mayormente, por lo que está acostumbrado a una tablet. Al mismo tiempo, María Gracia (figura 14) trabaja gestionando información como la de estudiantes como Eladio o Marketa (figura 15) y no puede permitirse interaccionar con un dispositivo móvil, pues tardaría mucho en realizar su trabajo, y no lo haría de forma efectiva. Por tanto, una solución web parece una resolución del problema apta para la mayoría del mercado del producto.

Como decíamos, nos sirven elementos que no tengan que ver directamente con las habilidades tecnológicas de los candidatos; y es que, por ejemplo, María Gracia en este caso no podría tampoco ver una web que no tuviera la tipografía lo suficientemente grande como para leer, porque podría tener la vista cansada. O quizás Marketa, si no tiene una buena situación económica, tendría quizás que conectarse desde un dispositivo algo obsoleto, por lo que probablemente le sea muy costoso en cuanto a tiempo cargar una página web con numerosas imágenes.

En definitiva, podría pensarse que los perfiles de estas tres personas están hechos para hacer el análisis de cualquier aplicación o producto software, ya que no hemos ideado estas personalidades específicamente para el problema que nos compete. Es por ello por lo que podemos tomar estos perfiles ficticios como válidos, pues no están «hechos a medida» para nosotros. Gracias a ello, podemos adaptar el desarrollo a una serie de necesidades y/o conflictos que pueden generar en otras personas al mismo tiempo, obligándonos a buscar una solución homogénea para todos, funcional y, no menos importante, que resuelva el problema como es debido y dadas las necesidades.

3.3 ENTREVISTAS

Las reuniones con las personas que pueden entenderse como «el cliente» del producto software, han sido esenciales para el desarrollo del mismo. En primer lugar, se establecieron las bases y se definieron los problemas. Una vez lograda la comprensión, nos pusimos manos a la obra a definir cómo podríamos abordar el problema de forma que se pudiera aportar una solución de valor; si bien no definitiva y que resolviera el problema por completo, pero sí un buen comienzo como pretendemos alcanzar con este proyecto.

Como decimos, el valor de las reuniones es muy alto. Sin ir más lejos, dependía de la interacción con el creador de TWINS, Miguel Ángel Sanz, de la comprensión de los procesos implicados y de la cesión del acceso al material para la realización del proyecto. No de menos importancia son las reuniones utilizadas para la realización

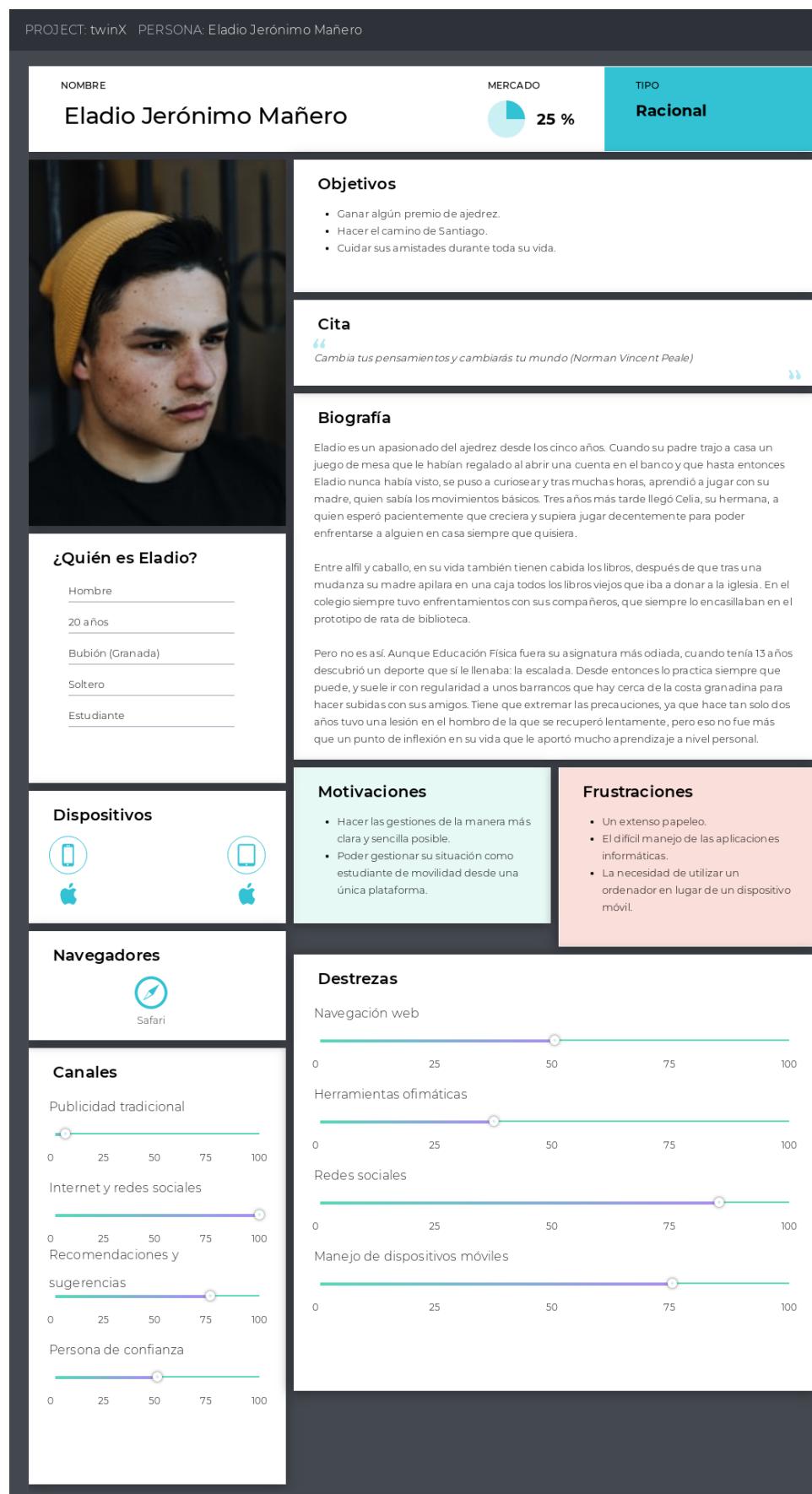


Figura 13: Persona #1: Eladio

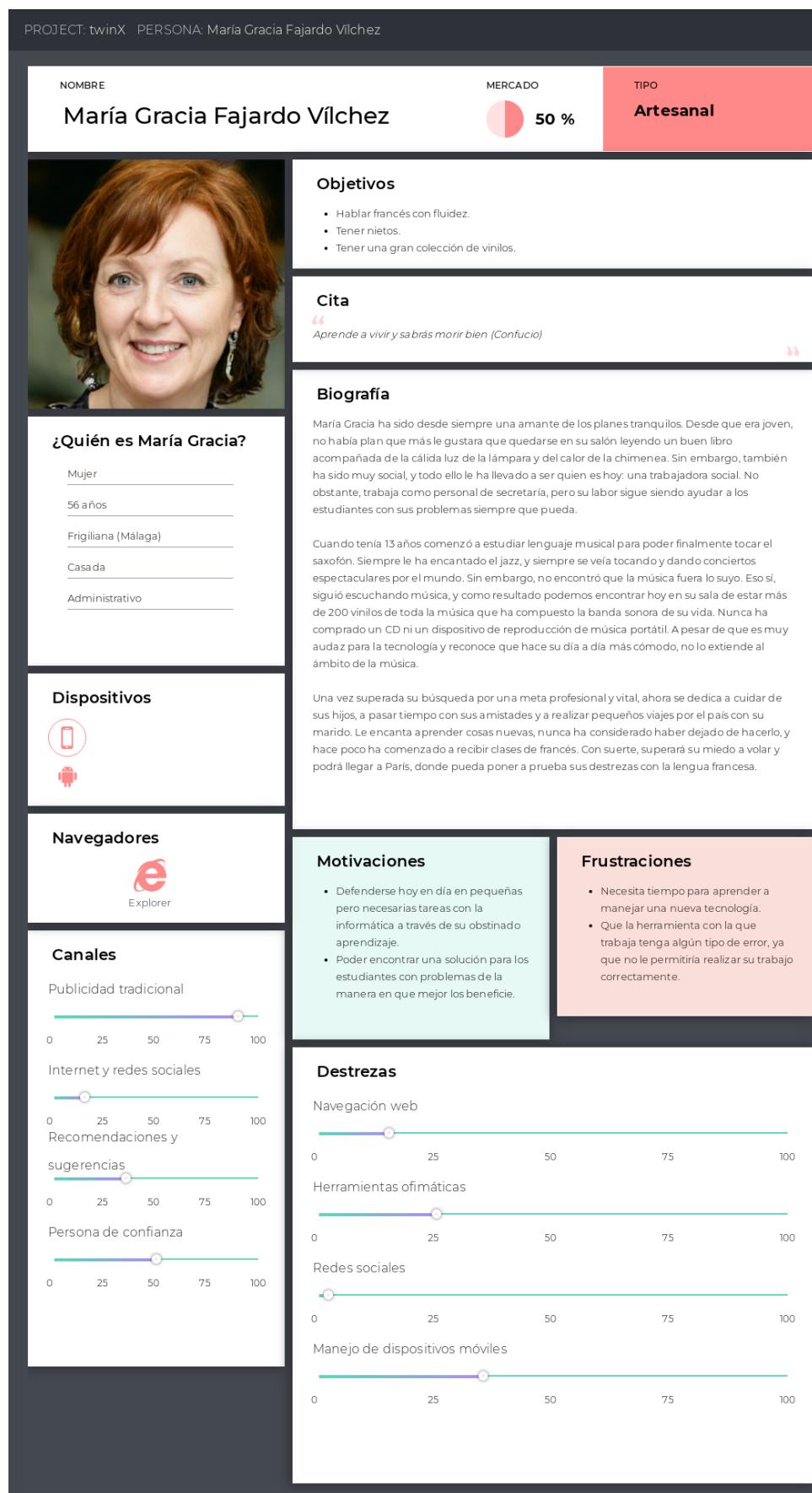


Figura 14: Persona #2: María Gracia

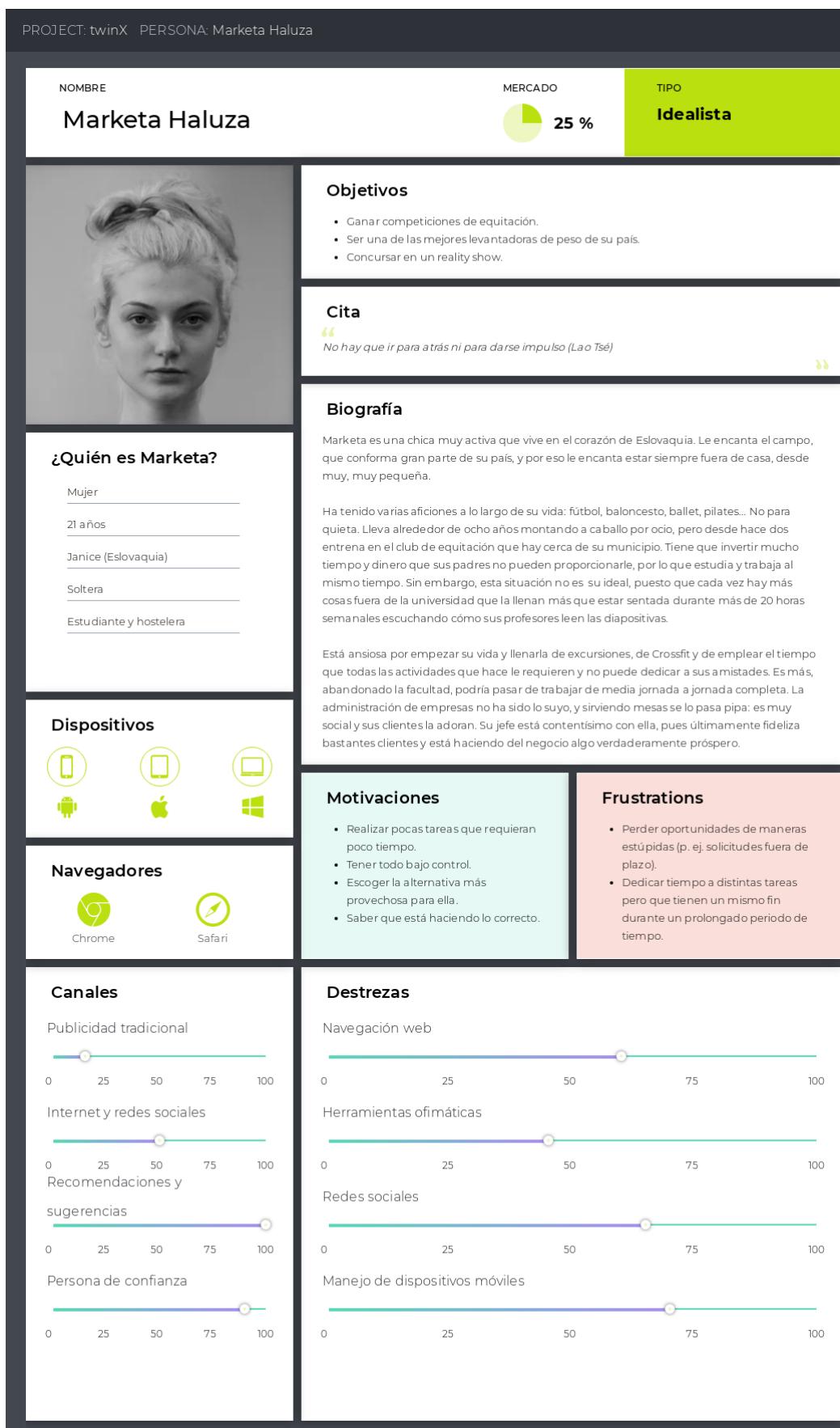


Figura 15: Persona #3: Marketa

de pruebas, ya que durante el proceso de desarrollo el enfoque estaba en simplificar y tratar de plasmar el núcleo de TWINS para construir twinX, lo que podría llevar a futuros problemas de conceptos o de carencias no detectadas a tiempo, para lo que es fundamental el contacto con los verdaderos usuarios finales de la aplicación –o al menos en sus primeras fases–.

Sin estas entrevistas, hubiera sido muy complicado llegar a hacer un producto que pueda servir a la ORI-FyL, el que era un objetivo implícito, pero el cual tiene prioridad como el que más, pues de nada serviría realizar un proyecto para crear un producto que nadie pudiera llegar a utilizar. Por tanto, las aportaciones de los participantes en las entrevistas han sido los que han capacitado al desarrollo el dar como resultado una pieza de valor.

3.4 MALLA RECEPTORA DE INFORMACIÓN

A las técnicas expuestas en [Herramientas del DT](#), podemos sumar la creación de una malla que nos permita definir las partes claves de nuestro proyecto. Si bien es cierto que la *Feedback Capture Grid* es mayormente utilizada para potenciar el aprendizaje tras las sesiones de pruebas, podríamos plasmar en ella los elementos que cualquier persona podría hacerse, sean o no actores que interactuarían o no con el producto final [11].

Su creación consiste en la división en cuatro cuadrantes de una hoja de papel, por ejemplo. En los diferentes espacios, empezando en orden de izquierda a derecha y por la parte superior, se toma nota de las buenas ideas que se han percibido tras conocer el producto, los problemas o las críticas que se le pueden encontrar, las preguntas que pueda haber y, por último, las ideas para la mejora del software que se pudieran tener, a modo de trabajos futuros.

En nuestro caso, hemos efectuado la malla (figura 16) para definir, a grandes rasgos, lo bueno que supone twinX ante TWINS, los impedimentos que tiene esta primera aproximación o proyecto piloto, las preguntas que podrían surgir por varias personas a la hora de conocer el producto (personal de administración, estudiantes, tutores...) y, por último, una serie de propuestas que se desarrollarán en la sección de trabajos futuros.

3.5 DESCRIPCIÓN DE LA PROPUESTA

Una vez hemos dado pinceladas de todo lo que tenemos, de qué podemos hacer, qué no y cómo llevarlo a cabo, vamos a establecer los claros objetivos de este proyecto y definir exactamente cuál será el producto tras su finalización, una vez conocemos todo el entorno que lo rodea.

Positivo		Crítico
<ul style="list-style-type: none"> Mejor diseño de la interfaz gráfica. Mayor facilidad de acceso: ya no se tendrá que disponer del archivo de configuración y el de datos, sino que twinX se alojará en un servidor web, accesible desde cualquier equipo. Reconstrucción de la base de datos, mejorando la consistencia y eliminando la redundancia de los datos. Posibilidad de extensión, de adición de características de manera sencilla y potente a través de la continuación en el desarrollo web 		<ul style="list-style-type: none"> La herramienta no contará con todas las características en su primera versión y la ORI-FyL no podrá hacer una migración inmediata. Se tendrán que adaptar ciertas formas de interactuar con el sistema y habrá que hacer modificaciones constantes hasta encontrar el método óptimo Se precisará de un equipo de desarrollo capaz de mantener la aplicación. La información con la que tratar es de carácter muy sensible y no se toleran pérdidas o corrupciones de la misma.
Preguntas		Ideas
<ul style="list-style-type: none"> ¿Podrán los estudiantes confeccionar su acuerdo de estudios desde la plataforma? ¿Podrán los coordinadores extranjeros nominar a sus estudiantes desde la plataforma? ¿Se podrán enviar mensajes desde la plataforma? ¿Podrán las OORRII de otras facultades utilizar twinX? ¿Podrán revisarse los acuerdos de estudios de los estudiantes que se vayan de movilidad desde el portal? ¿Tendrá que ser mediante el intercambio (subida y descarga) de documentos? 		<ul style="list-style-type: none"> Los estudiantes extranjeros podrían elegir sus asignaturas en la plataforma directamente, donde puedan ver las plazas libres y demás información relevante. Podría existir un bot de Telegram que informara de las actualizaciones del progreso de movilidad de los estudiantes, para que puedan recibir la información de forma instantánea y en su teléfono móvil, de forma sencilla. La elección de destinos podría estar también incluida en twinX. twinX podría estar disponible en varios idiomas, mayormente para la parte de los estudiantes y coordinadores externos.

Figura 16: Malla receptora de información

3.5.1 Elección del tipo de metodología de desarrollo

Para desarrollar twinX primero se ha de elegir una metodología. Tradicionalmente, los proyectos de software se construían siguiendo lo que se conocen como «metodologías pesadas» o tradicionales [8], largos procesos de desarrollo en los que se definen unas fases secuenciales por donde el producto iba pasando hasta tener una versión final. La metodología más sonada es la de «cascada» o *waterfall* en inglés [20]. En ella, tenían lugar unas estrictas etapas enfocadas a un grupo de tareas en concreto, como son la etapa de definición, la de diseño, la de codificación, la de pruebas y la de mejora. Todo ello daba lugar a documentaciones muy completas aunque innecesariamente extensas en muchos casos, a través de la implicación de mucha gente experta en la materia, capacitada para hacer las tareas definidas muy concretamente. La flexibilidad apenas estaba contemplada, puesto que la planificación era muy estricta y hacía peligrar la buena continuidad del proyecto.

Hoy en día, apenas se utiliza este tipo de metodologías. En nuestro caso, queremos construir lo que sería tan sólo el comienzo del producto final. En otras palabras, nuestras condiciones no encajarían en una metodología en cascada, pues no tenemos unos requisitos estrictamente fijados, ya que vamos a crear una versión de TWINS no definitiva y la cual no se verá con todas las características implementadas. Además, es un cambio tan drástico el que se va a dar al transportar la aplicación a la web, que se precisa de la continua atención de una especie de «equipo supervisor», conformado por uno o varios usuarios finales del producto, que nos pueda proporcionar un *feed-*

back sobre si nos estamos acercando o alejando de sus expectativas. Sobre esto último, diremos que no es que no se tuviera en cuenta en las metodologías tradicionales, pero no se primaba tanto como se debería, lo que hacía que el software perdiera calidad.

Por todo ello, podemos aventurarnos a decir que nuestro proyecto encaja más en una **metodología ágil**. Este tipo de metodologías se basan en la adaptación a medios con necesidades cambiantes, con un desarrollo muy de cerca con el cliente, quien está presente junto con el equipo de desarrollo en las numerosas reuniones que se hacen a lo largo de la vida del proyecto. Es más, en el *14th Annual State of Agile Report* [7] podemos encontrar unas estadísticas sobre los beneficios que ha traído el uso de metodologías ágiles en el último año, entre las que podemos destacar la capacidad de manejar los cambios de prioridades, la reducción de riesgos en el proyecto o el mantenimiento del software. En este caso, el equipo de desarrollo es ajustado, pero no por ello dejaremos de lado la necesaria intervención de dos personas fundamentales que hacen el papel de clientes y que son:

- María Consuelo Pérez Ocaña, Coordinadora de Internacionalización en la la Facultad de Filosofía y Letras.
- Miguel Ángel Sanz Sáez, personal de secretaría y creador de TWINS.

3.5.2 Propuesta de producto

En su primera versión, twinX incluirá la funcionalidad mínima de TWINS, de modo que de cara a las pruebas, la interacción con la nueva plataforma sea más bien realista, y aporte verosimilitud a la mejora, pudiendo así ser tomada como tal.

Más concretamente, contemplamos la implementación de:

- El añadido, modificación y eliminación de información esencial (como son asignaturas, tipo de expediente –similar a [Expediente de TWINS](#)–, mensajes predefinidos, etc.)
- La visualización de la información relevante a estudiantes (sus datos personales, expedientes abiertos, fases de los mismos, etc.)
- La visualización de los [Convenios](#), modificación de la información y el añadido de nuevos. Posibilidad de dos vistas: una de resumen y otra completa, tal y como se tiene en TWINS.
- Calendario con eventos, tareas, tal y como se disponía en TWINS
- El envío de mensajes, a modo de sustituir los avisos en TWINS, pero abriendo el abanico a tres tipos de mensaje: mensaje normal, recordatorio (mensaje de un usuario para sí mismo) y avisos (por proximidad a la fecha de un evento o al límite para realizar una determinada tarea).
- Una pantalla inicial de visualización de la información de mayor relevancia a modo de resumen.

- Organización de todo el contenido en tres grandes secciones: «gestión», «calendario» y «panel de control», para simplificar y mejorar la interfaz de usuario, facilitando la interacción con la misma.
- Búsqueda en los distintos listados (convenios, estudiantes, expedientes...) con filtros para facilitar la recuperación de la información.

3.6 PLANIFICACIÓN

Una vez sabemos qué son las metodologías ágiles, vamos a centrarnos en una en concreto con la que poder expandir nuestro proyecto de forma organizada y guiada. También necesitaremos organizar la acción en varias fases; esto es, dividir en distintas partes el contenido a desarrollar, de modo que lo más importante venga al principio, para poder obtener resultados de calidad. No menos importante, también tendremos que estimar un coste aproximado del proyecto para poder llevarlo a cabo, atendiendo a los distintos factores que intervienen.

3.6.1 La metodología Scrum

En relación con la sección [3.5.1](#), la metodología concreta a usar para el desarrollo de twinX será **Scrum**. Es ideal para incrementar la flexibilidad y producir un sistema que pueda responder ante tanto requisitos iniciales como los que se van añadiendo a lo largo del proceso de desarrollo que se puedan ir descubriendo.

En Scrum, se asume que el análisis, el diseño y los procesos de desarrollo son impredecibles, y por ello, tienen que ser iterativos; esto es, hay que estar constantemente comprobando el trabajo realizado, evaluando qué se ha hecho, para adaptarlo de la mejor manera posible a la situación del proyecto. Esta repetición recibe el nombre de **sprint**, y es una medida de esta metodología.

Los sprints son flexibles y no lineales. Se podrían tomar como una caja negra que precisa de controles externos. De este modo, en cada iteración se pueden tener en cuenta todos los factores que se crean necesarios, de forma que se evite convertir el proceso en algo caótico y que se aleje de los objetivos del proyecto y, al mismo tiempo, se pueda maximizar la flexibilidad [\[22\]](#).

Vamos a agrupar en tres bloques las fases de la metodología Scrum:

- **Preparación.** Podemos diferenciar dos subfases:
 - Planificación: donde se estiman el coste y el tiempo del proceso
 - Arquitectura: para decidir cómo se van a implementar los elementos de la lista de trabajos pendientes o *backlog*
- **Desarrollo:** donde se lleva a cabo la realización de los sprints acordados en la fase anterior. Se desarrollan nuevas funcionalidades, respetando las variables del tiempo, requisitos, calidad, coste y competición. El fin de esta fase está condicionado a cuán bien se interacciona con las variables mencionadas.

- **Finalización:** preparación para el lanzamiento del producto, donde se incluye la documentación final, y unas pruebas que se realizan con anterioridad a la versión final del software.

3.6.2 Planificación en sprints

A continuación, vamos a exponer la planificación que se ha pensado para llevar a cabo el desarrollo de este producto piloto para la plataforma twinX, de acuerdo con lo expuesto en la sección [3.5.2](#):

Sprint 0: puesta a punto

- **Toma de decisiones acerca del renombramiento de algunas entidades existentes en TWINS:** existen una serie de nombres que pueden ser confusos para una persona que trate de comprender el funcionamiento de TWINS desde cero; si bien esto no ocurre con los usuarios habituales de la aplicación de Access®, con el desarrollo de twinX estamos, por lo general, facilitando a todo el entorno de la ORI-FyL, y por tanto, se renombrará alguna entidad como por ejemplo «evento». El motivo no es otro que la confusión entre un evento de calendario cotidiano y un [Evento de expedientes de TWINS](#). Para ello, se tiene que hacer un segundo análisis de la base de datos ya creada y así poder mejorar la comprensión de nuevos usuarios en la plataforma.
- **Realización de un tutorial de Yii2 Framework:** sobre los detalles de la tecnología a usar para el desarrollo de twinX hablaremos en la sección [6](#). Sin embargo, antes de ponernos manos a la obra con la implementación del sistema web, habrá que adquirir los conocimientos necesarios para la toma de buenas decisiones y el respeto de las buenas prácticas del framework.
- **Modelado de la base de datos:** realizar el diseño lógico-conceptual de la base de datos es esencial a la hora del desarrollo. Una vez se tiene, podemos crear las tablas necesarias para hacer funcionar la aplicación. Es de suma importancia dedicar el tiempo suficiente a esta tarea, dado que los errores en el modelo pueden dar lugar a imprevistos temporales en el futuro.
- **Creación de bocetos:** necesarios para confeccionar las interfaces de usuario sobre un esquema ya pensado, de modo que las decisiones de diseño de importancia tengan lugar en esta fase, y no sobre la marcha, ya que es más costoso en cuanto a tiempo y la consistencia de las vistas puede verse comprometida.

Sprint 1: comienzo del desarrollo

- **Integrar el Panel de control de twinX:** desde donde se añadirán nuevos registros a la plataforma, del lado del [Administrador](#). Es la parte más importante si tenemos en cuenta que es la que nos posibilita hacer pruebas de la manera más sencilla, ya que no se espera que twinX tenga en su primera versión ninguna herramienta de migración masiva de datos.

- **Integrar el módulo de Gestión de twinX:** como parte fundamental de twinX, donde se tiene la mayor parte de la funcionalidad y las acciones que los *gestores de twinX* llevarán a cabo durante su jornada laboral a diario, pues viene siendo el corazón del actual TWINS.

Sprint 2: complementando twinX

- **Integración del módulo de calendario:** el cual tendrá los mismos elementos que el calendario en TWINS, con eventos, tareas asociadas a los mismos, y avisos, en forma de mensaje, para las fechas límite de las tareas.
- **Integración del módulo de mensajería:** para el intercambio de mensajes o notas, en sustitución del confuso menú de la figura 5.

Sprint 3: ensanchando horizontes

- **Integrar el frontend del estudiante:** con acciones básicas como el registro y la confección del acuerdo de estudios.
- **Integrar el frontend del tutor:** para interaccionar con sus estudiantes y poder revisar sus acuerdos de estudios.

Sprint 4: haciendo twinX aún más potente

- **Integración del reconocimiento de créditos:** para los estudiantes salientes, trasladar sus puntuaciones obtenidas en su universidad de destino a calificaciones ponderables en su expediente en la UGR.
- **Integración de la alteración de matrícula:** proceso mediante el cual los estudiantes entrantes pueden tener asignadas sus asignaturas, confeccionados sus horarios dependiendo de factores como las plazas vacantes o el posible solapado de clases en un tramo horario.
- **Funcionalidad de migración de datos masiva:** desde la sede de la UGR, quien comunica a la ORI-FyL los listados de los estudiantes de movilidad entrantes y salientes.

Todas estas tareas pueden ser, acorde con la metodología Scrum, modificadas y trasladadas a otros sprints. Sobre las subtareas que conllevaría cada épica¹ (las expuestas anteriormente), hablaremos en la sección ??.

Independientemente de la división del trabajo en todos estos sprints, no se contempla la realización de más de uno o dos de ellos, con la posibilidad de cambiar los contenidos de los mismos, pues también habría que incluir la realización de las pruebas y la redacción de esta memoria.

En adición a la planificación aquí expuesta, cabe señalar la utilización de un tablero **Kanban**, característico de las metodologías ágiles y de Scrum. En él, se disponen las

¹ Las épicas son tareas que almacenan dentro de ellos un gran número de subtareas.

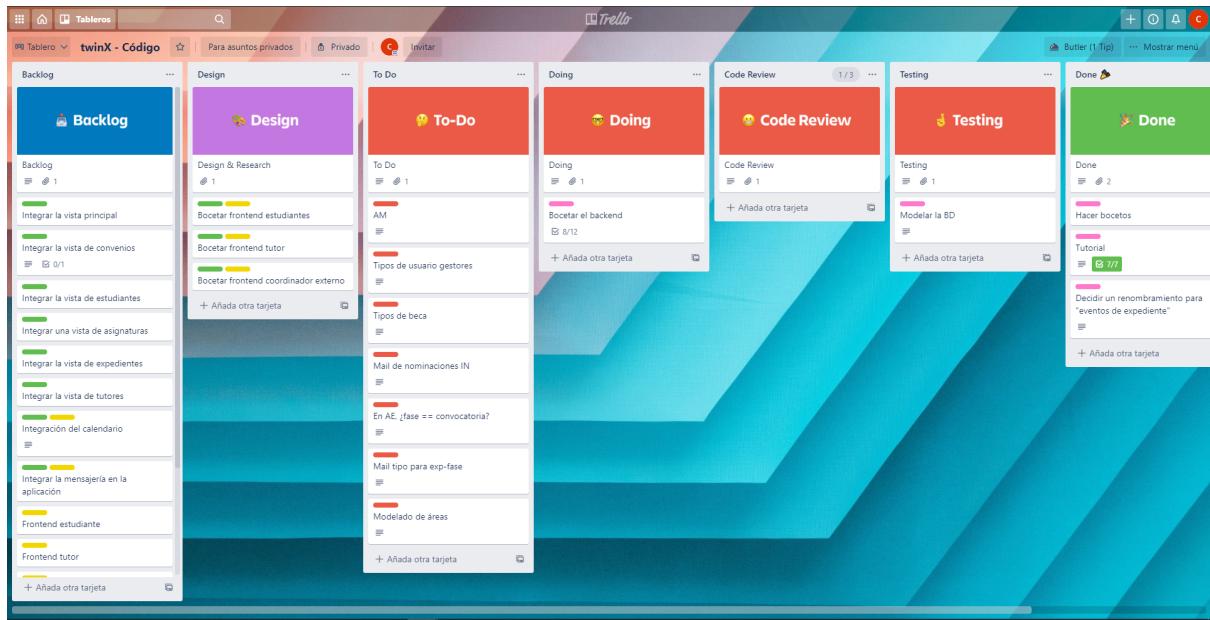


Figura 17: Kanban al comienzo del desarrollo

tareas en varias columnas. A cada tarea se le puede asignar un color para distinguirlo, por ejemplo, en los distintos sprints. Para la construcción de twinX hemos usado la herramienta *Trello* para la confección de un tablero interactivo (figura 17). Las distintas etiquetas representan distintos sprints, y a cada tarea (consideradas en su mayoría como épicas), se les puede adjuntar una lista de tipo *checkbox* para asegurarnos de que no queda tarea sin hacer dentro de cada épica.

3.6.3 Planificación temporal

Desde febrero de 2020 que dio comienzo la primera reunión para comenzar a planificar el proyecto hasta noviembre del mismo año, la planificación del trabajo se ha dispuesto como se muestra en el diagrama de Gantt de la figura 18. Su confección ha sido posible gracias a la herramienta Lucidchart [26].

Más en detalle, el tiempo dedicado a cada tarea en horas, de forma aproximada, lo podemos exponer en la tabla 2.

2 Hemos sacado fuera del Sprint o la realización del tutorial del framework porque lo hemos englobado dentro de la sección de aprendizaje.



Figura 18: Diagrama de Gantt con la planificación del proyecto real

3.7 PRESUPUESTO

A partir de la información anterior en cuanto al tiempo invertido en el proyecto y en vista de los elementos necesarios para llevarlo a cabo, proponemos el presupuesto de la tabla 3 para hacer frente a sus costes.

El precio por hora obtenido por el desarrollador se ajusta al estándar cobrado por un ingeniero junior que acaba de incorporarse al mundo laboral.

Sobre los demás elementos, diremos que la amortización solo ha sido necesaria aplicarla en caso del computador, que tras la finalización del proyecto seguirá siendo útil. Para la licencia de la suite de Office® hemos podido especificar la compra de un año completo como periodo mínimo de uso, siendo imposible el uso de la versión de estudiante, que excluye la herramienta de Access®, imprescindible para poder llevar a cabo todo nuestro estudio. Respecto del resto de software utilizado, no se incluye por ser de licencia libre, como Yii2 Framework.

Análisis	≈ 13 h 30 min
Reuniones	3 h 33 min
Estudio de TWINS	10 h
Aprendizaje	60 h
PHP	40 h
Tutorial de Yii2 Framework	20 h
Implementación	100 h
Sprint o ²	10 h
Sprint 1	60 h
Sprint 2	30 h
Pruebas	5 h
Memoria	120 h
Total del proyecto	298 h

Tabla 2: Estimación horaria por bloques

Recursos hardware	121.4 €
Concepto	Coste
Xiaomi Mi Air Notebook 13"	611 €
[Amortización a 5 años]	121.4 €
Recursos software	214 €
Concepto	Coste
Windows®10 Home	145 €
Suscripción a Microsoft Office®365 (1 año)	69 €
Servidor AWS (tarifa estándar gratuita)	0 €
Recursos humanos	2 384 €
Concepto	Coste
298 horas de trabajo (8€ / hora)	2 384 €
Coste total del proyecto	2 719.4 €

Tabla 3: Presupuesto estimado

4

DISEÑO DE LA INTERFAZ DE USUARIO

Una vez hemos definido cuáles son nuestros objetivos, analizado el estado del arte y establecido unas pautas para el desarrollo, vamos a comenzar con el diseño de la parte visual. Esto es, a su vez, un proceso que requiere de distintas definiciones, de manera que al término de este capítulo, no sólo tengamos el diseño de las interfaces de usuario, sino que también comprendamos por qué se han diseñado así y la intención de cada elemento que disponemos en pantalla.

4.1 MATRIZ DE TAREAS DE USUARIO

Un buen comienzo sería el de definir los roles de usuario en nuestra nueva aplicación. Hemos hablado ya en el capítulo anterior sobre la figura del [Gestor de twinX](#) y la del [Administrador](#). Incluyamos entonces también las de [Tutor\(a\) académico/a](#), estudiante y, de forma excepcional, la de coordinador(a) externo/a. Sobre esta última, recordemos que no tenía ningún tipo de interacción en TWINS y que hasta entonces, los usuarios de la plataforma habían establecido intercambios de información por correo electrónico con los coordinadores de otras facultades, que es el método estándar de realizar las [nominaciones](#). Sin embargo, aunque no para sus primeras versiones, se espera que twinX brinde la capacidad de hacer estas nominaciones a través de una interfaz gráfica, de una forma mucho más sencilla y segura que con el envío de un texto libre, donde puede haber errores y/o malas prácticas, dado el amplio uso que se puede hacer de la herramienta.

Es precisamente por la necesidad de dejar claras las acciones a llevar a cabo por cada usuario del sistema por la que elaborar una **matriz de tareas de usuario**. Esta herramienta nos permite establecer cuál es la frecuencia de uso de una característica de nuestra aplicación por un grupo de usuarios en concreto y cuál es la prioridad que debemos darle a la implementación de la misma. De esta forma, nos será más fácil diferenciar aquellas acciones que tan solo se lleven a cabo unas pocas veces al mes (o cada varios meses) y cuales, por el contrario, son ejecutadas casi a diario, de modo que se priorice su integración y su optimización. Sobre estas últimas, cabe decir que el potenciarlas no solo se procurará hacer en términos de procesamiento interno del código, sino también en cuanto a su facilidad de utilización por parte del usuario. Es

decir, tendremos que crear más accesos directos o simplificar lo máximo posible el curso de tareas a llevar a cabo para realizar estas acciones y obtener los resultados deseados. En adición a todo esto, podremos observar cuáles son los usuarios que menos usarán la plataforma, de modo que se reste prioridad a la implementación de sus interacciones, en beneficio de las que llevan a cabo los usuarios que, por el contrario, monopolizan el uso de la aplicación [31].

Para simplificar la matriz (tabla 4) y hacerla visualmente más sencilla de entender, vamos a asignar una serie de letras que correspondan a un nivel de utilización en concreto. Elegiremos la «A» para una frecuencia de uso alta, «M» para un uso medio y «B» para una baja frecuencia. A estas indicaciones añadiremos unas puntuaciones que nos permitirán determinar con mayor facilidad cuáles son las tareas de mayor importancia y cuáles son menos urgentes de implementar, al igual que los usuarios que desempeñarán un papel más o menos crítico en la aplicación. Respectivamente serán 3, 2 y 1 punto, de modo que las prioridades más altas recibirán una mayor puntuación y viceversa.

Usuarios / Tareas	Administrador	Gestor	Tutor	Estudiante	Coordinador externo	Puntuación
Acceder a twinX	M	A	B	B	B	8
Ver expedientes de un usuario	A	A				6
Cambiar de fase un expediente	A	A				6
Consultar la información de un convenio	A	A				6
Modificar los datos de un estudiante	M	M		B		5
Introducir un nuevo convenio	B	M				3
Realizar el acuerdo de estudios				B		1
Revisar propuesta de acuerdo de estudios		B	M			3
Consultar los expediente pendientes de procesamiento	M	A				5
Expresar consentimiento de cesión de datos				B		1
Añadir un nuevo tipo de expediente y/o fase de expediente	B					1
Marcar un expediente como favorito	B	M				3
Puntuación del grupo de usuarios	18	22	3	4	1	

Tabla 4: Matriz de tareas de usuario

Tras la atenta observación de la matriz, podemos afirmar que las acciones que mayor uso van a tener por los usuarios de twinX serían las ya existentes en TWINS, como es lógico, y que por supuesto, los grupos de usuarios con mayor necesidad de uso de la aplicación son los gestores y administradores. Sobre éstos últimos, cabe señalar que son una especie de gestores con más permisos de lo normal, de modo que pueden modificar la información que clasifica la situación de los estudiantes como son, por

ejemplo, las fases de los expedientes. Solo los administradores podrían hacerlo, ya que los gestores no tienen por qué tener acceso a dichas modificaciones; simplemente necesitan trabajar con las ya registradas y, en caso de necesitar más entradas o la modificación de otras existentes, tendrían que solicitarlo a los administradores.

Con esta matriz, además, justificamos que el contenido de la primera versión de twinX (descrito en la sección 3.5.2), es suficiente para cubrir la mayoría de las necesidades esenciales del trabajo de un secretario de la ORI-FyL, lo que es uno de los objetivos principales de este proyecto, aunque sea tan solo un comienzo a lo que finalmente llegaría a ser twinX.

4.2 MAPA WEB (SITEMAP)

Cuando un usuario llega a un sitio web por primera vez, incluso si es experimentado, no sabe qué hacer o dónde hacer clic. Lo normal es que esta duda no perdure por más de 15 segundos en la cabeza, aunque podría ser así si el sitio web en cuestión no tiene una estructura bien definida.

Para aclarar la estructura de un portal web, se puede hacer uso de los *sitemaps*. Es un esquema visual que permite mostrar la arquitectura de la información en el sitio, y qué secciones están enlazadas y/o relacionadas.

Aparte de definir, este tipo de diagramas nos permite comprender mejor nuestros objetivos, si vamos bien encaminados y tenemos una idea que es válida también para los usuarios finales del producto y si por algún motivo hemos de eliminar páginas innecesarias o que puedan ser redundantes.

Para su confección, es necesario disponer la página de inicio y las secciones que parten de ahí y que llevan a otros sitios, a modo de árbol. Una buena estrategia es la de tratar de minimizar el número de pasos que hay que dar para llegar a encontrar la información deseada desde el comienzo del diagrama. Esto es lo que conocemos como **la regla de los 3 clics**; aunque no oficial, es sin duda una apuesta segura para la simplicidad, facilitando que los usuarios menos experimentados puedan usar el sitio sin tener que dedicar mucho tiempo a ello, en cuyo caso podrían abandonar y buscar otra alternativa [2].

En la figura 19 exponemos el *sitemap* de twinX, diferenciando claramente sus tres secciones principales: gestión, calendario y panel de control. En la parte de gestión, destaca la cantidad de posibilidades de navegabilidad existentes, dado que por ejemplo, al buscar un expediente en su menú, podríamos acceder al perfil de un estudiante o viceversa. Lo mismo pasa con los convenios y los acuerdos de estudios: todos estos accesos directos se ponen para facilitar el trabajo del usuario y mejorar su experiencia, usando los menús ya creados y evitando la redundancia de los mismos.

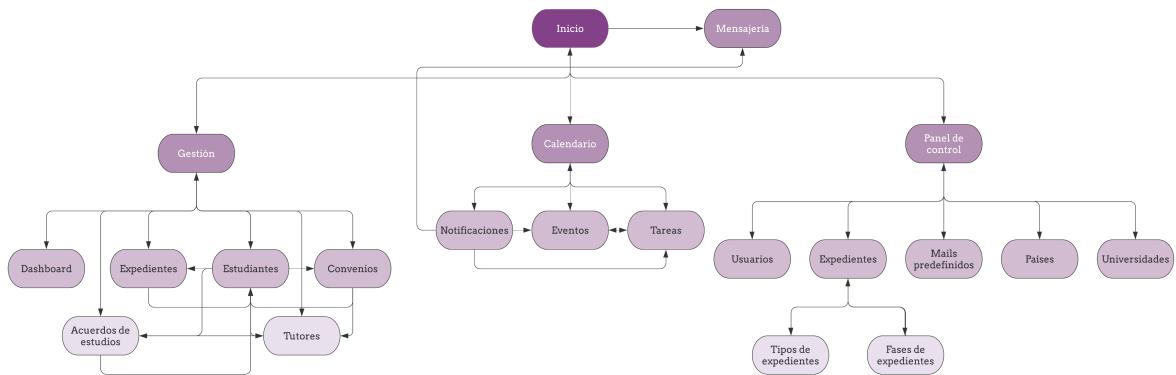


Figura 19: Sitemap de twinX

4.3 ICONOGRAFÍA (LABELLING)

Resultaría muy extraño navegar hoy en día por una web que no tuviera un icono ofreciendo alguna indicación. Quizás, al pensar en la palabra «ícono» nos vengan a la cabeza expresiones faciales que utilizamos a diario para comunicarnos (los famosos emoticonos), pero lejos de adornar, estos símbolos gráficos nos facilitan mucho la navegación por internet con indicaciones visuales que nos hacen ahorrar tiempo para encontrar un menú desplegable, el botón para iniciar sesión o simplemente la salida.

Como no podía ser de otra forma, nuestro portal no es una excepción y también incluye iconos para mejorar la experiencia de navegación. En la tabla 5 presentamos los iconos existentes en twinX.

4.4 GUÍA DE ESTILO

En [reunión de presentación de TWINS](#), Miguel Ángel comentó los significados del logo de TWINS (figura 20) y el por qué de los dos colores. Representan la unión de los estudiantes OUT (en color rojo) con los estudiantes IN (en azul), que se engarza en un corazón morado con motivo de la unión de dos universidades. Hemos querido mantener este concepto para la construcción de twinX, y la apuesta por una tipografía más vistosa como es Quicksand [10]. Para la coloración, hemos escogido los dos colores emblemáticos de la aplicación para mezclarlos un morado que va a ser el protagonista de los detalles en toda la web. Como resultado, se tiene el logo de la figura 21, el utilizado en la cabecera de la web.

Sobre los colores, hemos usado la herramienta **Color Tool** de *Material Design* para establecer el esquema de colores (figura 22). Hemos usado mayormente el tono normal (#ba68c8) y el oscuro (#883997) para twinX, siendo éste último el empleado en las partes seleccionadas o cabeceras de tablas y el primero, para todo lo demás en cuanto a detalles (hiperenlaces, en su mayoría).



Figura 20: Logo de TWINS

twinX

The logo for twinX consists of the lowercase letters "twinX" in a large, bold font. The colors transition from red for "t" and "w", through purple for "i" and "n", to blue for "X".

Figura 21: Logo de twinX

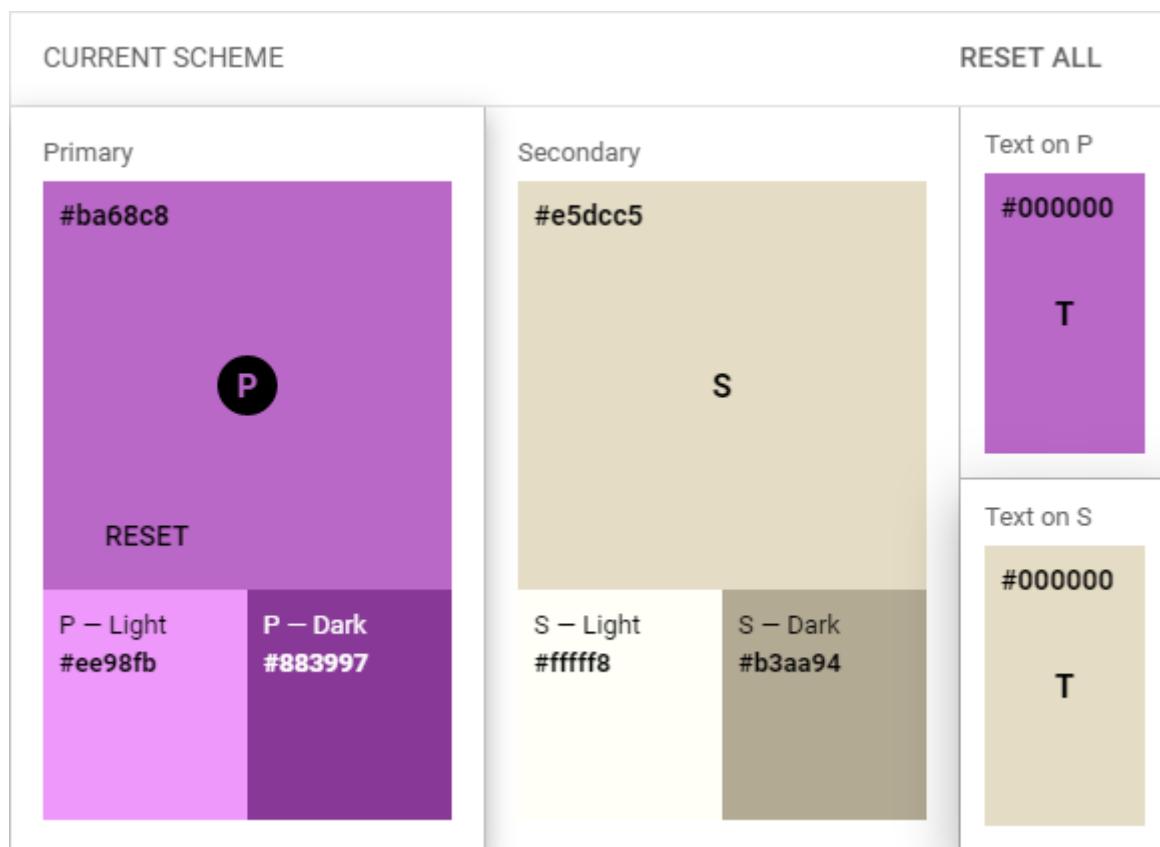


Figura 22: Esquema de colores de twinX

La fuente utilizada ha sido **Segoe UI** [19], por tener buena concordancia con la escogida para el logo, su buena legibilidad y su simplicidad (figura 23).

4.5 BOCETOS WIREFRAME

Con gran parte de los elementos necesarios definidos para poder crear twinX, necesitamos tener una idea clara de interfaz de usuario en mente antes de poder comenzar a integrarla. Pensemos lo fácil que es corregir errores de diseño tan solo arrastrando elementos y disponiéndolos en el boceto. Gracias a eso, podemos ahorrar mucho tiempo a la hora de programar según la guía que suponen los *wireframes*.

4.5.1 Wireframes del módulo de gestión

Para el módulo de gestión, tenemos los siguientes bocetos:

- **Dashboard:** una sección nueva que proponemos para twinX, donde se puede ver de un vistazo la información más importante para el usuario. Recordemos que en TWINS, al principio se disparaban una serie de vistas de calendario, avisos y tareas, etc. Todo ello obligaba al usuario a cerrar ventanas posiblemente

Aa Bb Cc Dd Ee Ff Gg Hh Ii
Jj Kk Ll Mm Nn Oo Pp Qq
Rr Ss Tt Uu Vv Ww Xx Yy Zz
1234567890!@#%&*?,:\$£€

Figura 23: Tipografía Segoe UI

indeseadas. Con esta nueva aproximación, el objetivo es aumentar la efectividad en el trabajo, pudiendo redirigirse a múltiples instancias del portal web desde un único lugar.

- **Lista de convenios:** manteniendo la información de mayor utilidad para el usuario, como el número de estudiantes salientes y/o entrantes que hay en destino del total de estudiantes que han sido nominados para ese convenio.
- **Vista de convenios básica contraída:** en TWINS había dos posibilidades para visualizar un convenio: estaba la vista básica, con la información de mayor relevancia para las consultas diarias; y la avanzada, que mostraba todos los detalles que se tenían sobre un convenio en concreto. Sin embargo, estaban en vistas distintas. En twinX, proponemos un único menú de convenio con un botón para cambiar entre la vista básica y la vista avanzada.
- **Vista de convenios avanzada contraída:** con todas las secciones contenedoras de los datos que alberga un convenio en su totalidad.
- **Vista de convenios extendida:** para ver los detalles de una o varias secciones del convenio en concreto.
- **Lista de expedientes:** se incluyen accesos directos a acciones rápidas que facilitan el trabajo de los gestores, como se venía haciendo en TWINS.
- **Lista de estudiantes:** de nuevo, con accesos directos para agilizar las operaciones.
- **Búsqueda con desplegable:** con la posibilidad de aplicación de distintos filtros para obtener unos mejores resultados. Éstos también tienen accesos directos que agilizan el trabajo. Por ejemplo, si se quisieran consultar los expedientes de un usuario, se podría ir directamente a ellos, sin tener que recuperar previamente la información del usuario en cuestión por completo.
- **Vista de expediente:** con información de relevancia para el gestor sobre el estudiante y una lista de las fases por las que el expediente ha pasado hasta la más actual. Se permite adjuntar documentación a cada fase y procesar un cambio de

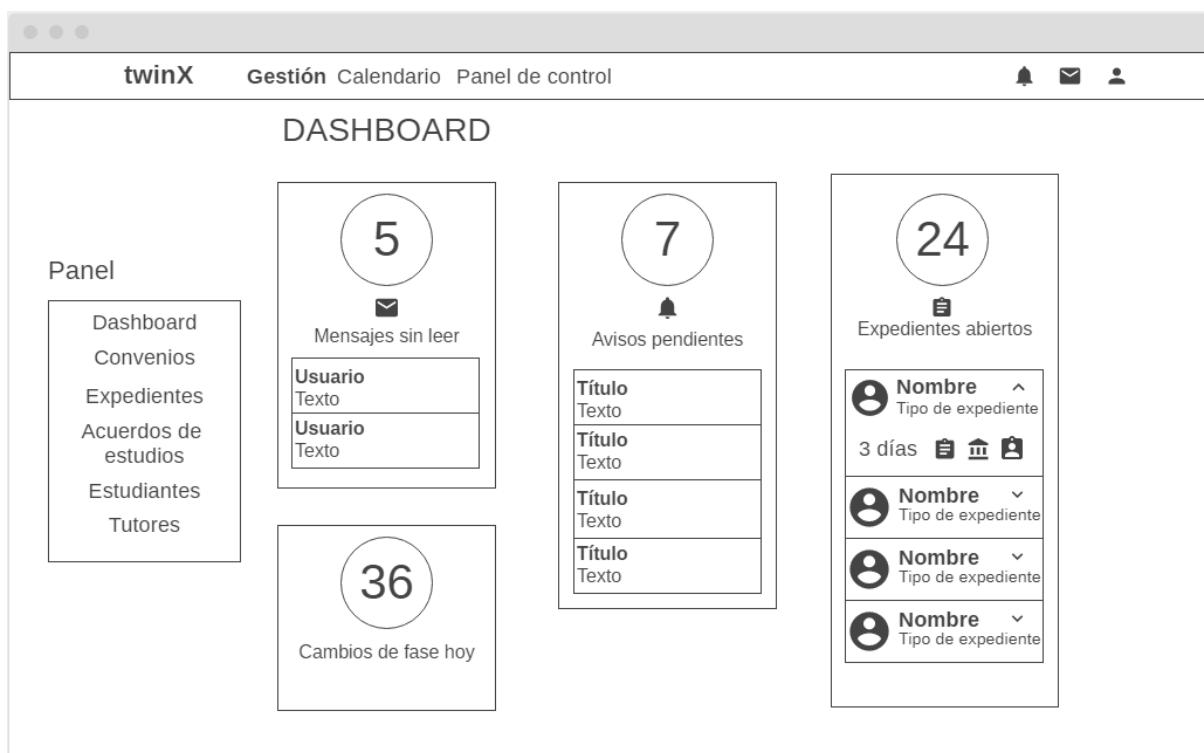


Figura 24: Wireframe del Dashboard donde el usuario verá de un vistazo toda la información de mayor importancia al iniciar la aplicación

fase desde esa misma vista, para simplificar el desdoble en menús. También se incluye la posibilidad de crear una nueva fase.

- **Diálogo modal de cambio de fase:** avisa al usuario de las acciones que se van a desarrollar tras la confirmación del procesamiento de la fase, que será principalmente el envío automatizado de mails predefinidos.
- **Vista de estudiante:** donde se dispone toda la información acerca de un estudiante.



Figura 25: Wireframe de lista de convenios



Figura 26: Wireframe de vista de convenio básica. Secciones contraídas.



Figura 27: Wireframe de vista de convenio avanzada. Secciones contraídas. Acceso desde el botón «+» de la botonera en la esquina superior derecha del contenido principal.



Figura 28: Wireframe de vista de convenio extendida, donde se pueden ver los detalles de una o varias secciones en concreto bajo demanda.



Figura 29: Wireframe de lista de expedientes



Figura 30: Wireframe de lista de estudiantes



Figura 31: Wireframe de búsqueda. Diálogo modal, desplegable desde la barra. Capacidad de filtrado



Figura 32: Wireframe de detalle de expediente



Figura 33: Wireframe del modal de cambio de fase. Advierte sobre las acciones a llevar a cabo tras la tramitación del cambio de fase en un expediente concreto.

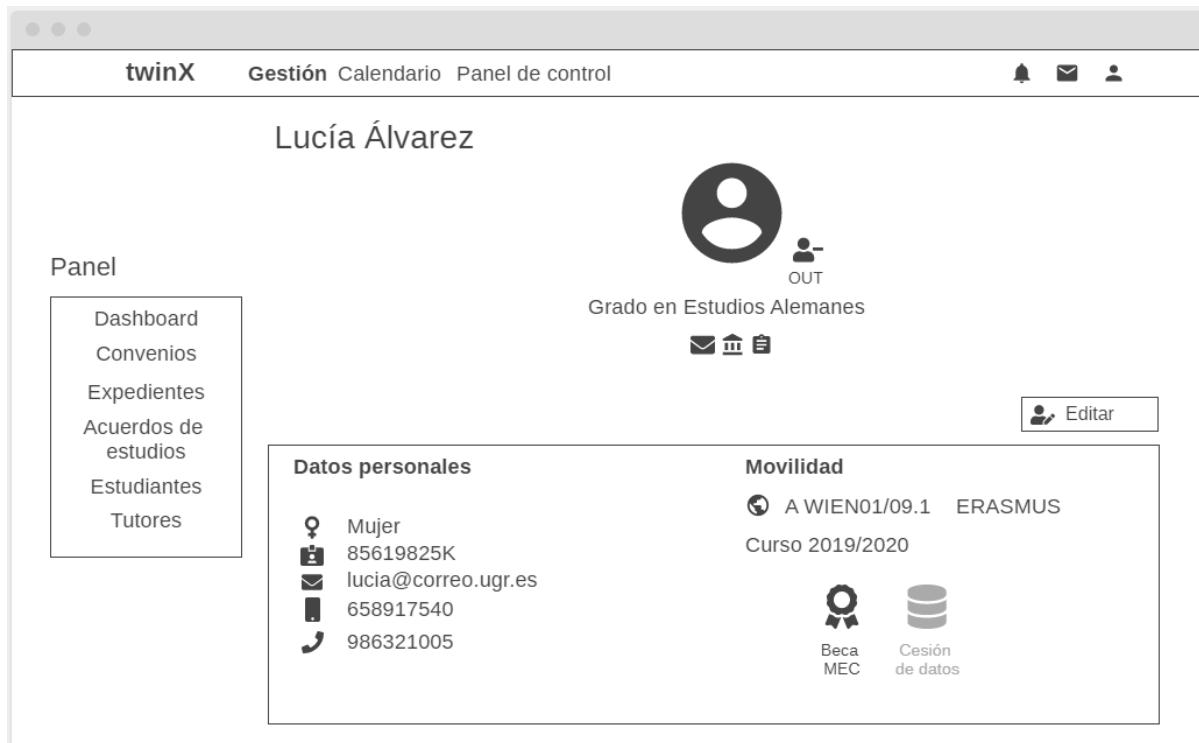


Figura 34: Wireframe de la vista de detalle de un estudiante

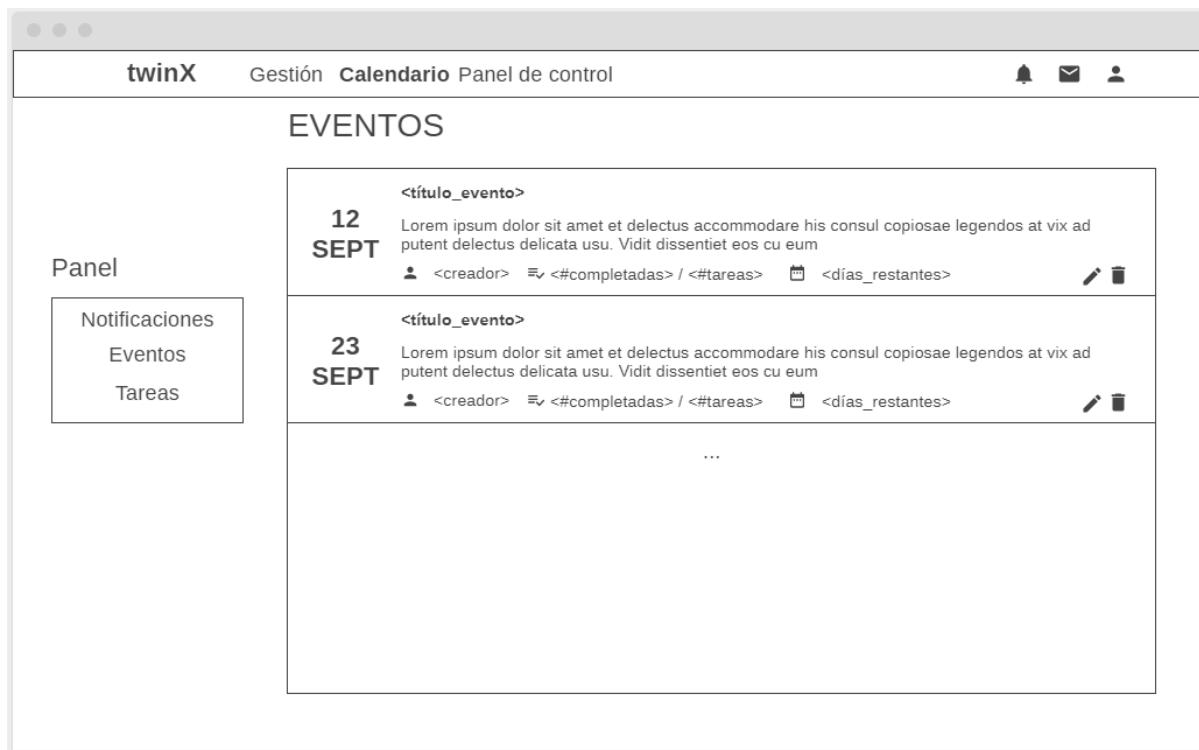


Figura 35: Wireframe de lista de eventos

4.5.2 Wireframes del módulo de calendario

Por su parte, los bocetos diseñados para el módulo del calendario son los siguientes:

- **Lista de eventos:** alberga todos los eventos que se han creado en el sistema, con la información más relevante sobre los mismos y acceso a acciones como editar y eliminar.
- **Lista de notificaciones:** como los mensajes, recordatorios y/o avisos de tareas pendientes, que aparecerán en este menú, accesible también desde la barra persistente situada en la cabecera de la aplicación.
- **Vista de creación de un evento:** respeta en la medida de lo posible la vista de creación de un nuevo evento en TWINS, con los elementos esenciales. Si bien es cierto que podría no ser la más inmediata a la hora de pensar en una interfaz similar, se ha tratado de ajustar a las necesidades reales de los usuarios finales. En este caso, se pretende permitir la inclusión de un número indefinido de avisos para con el evento, lo cual es extensible al ámbito de las tareas asociadas al mismo evento (en TWINS se tenían siempre dos, lo que obligaba a ocupar más espacio en las tablas de la base de datos, sobre todo para aquellos eventos donde ni siquiera había un solo aviso definido).

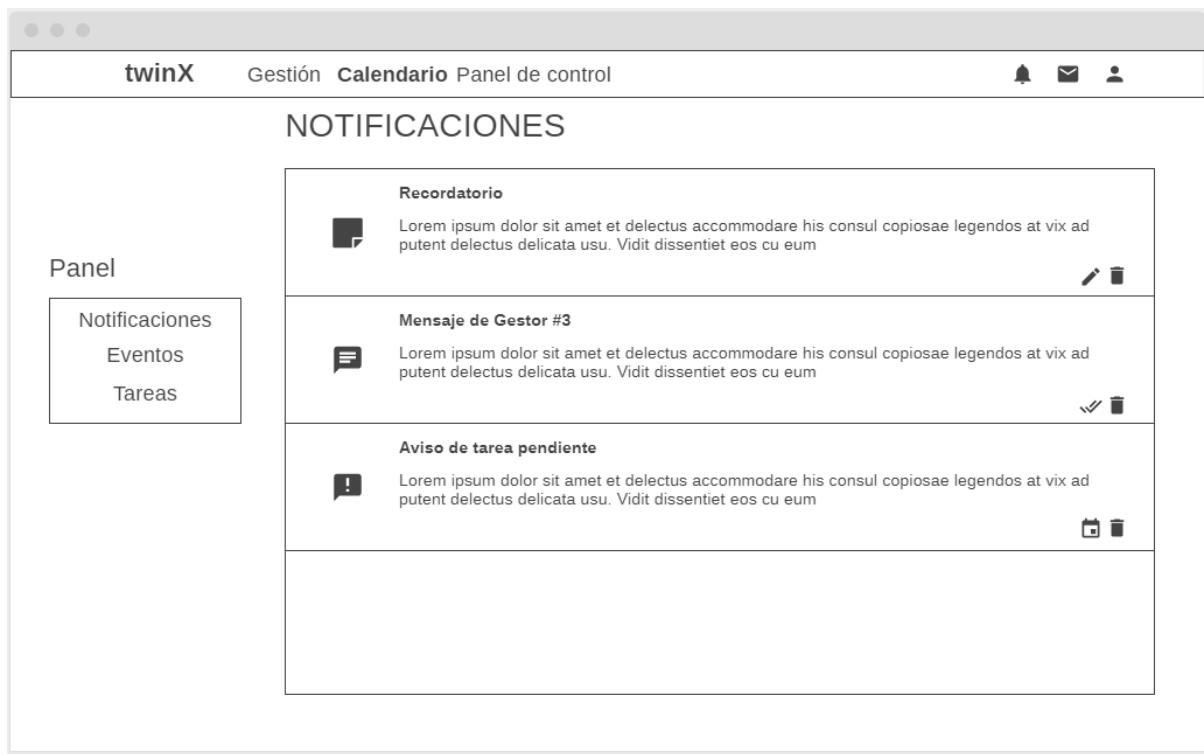


Figura 36: Wireframe de lista de notificaciones. Pueden aparecer tres tipos de notificaciones, dependiendo del tipo de mensaje recibido o generado por la propia plataforma (a través de avisos y/o recordatorios)



Figura 37: Wireframe de creación de un evento. Se distinguen dos vistas: la información del evento en general (izquierda) y la de sus subtareas (derecha)

Etiqueta	Descripción	Icono	Etiqueta (inglés)
Sobre	<ul style="list-style-type: none"> ■ Consultar el menú de envío de emails de un expediente concreto en el panel de control. ■ Acceso directo a la bandeja de entrada de mensajes. Icono persistente en la cabecera de twinX, junto al indicador de mensajes sin leer. 		<i>Envelope</i>
Ojo	Ver el detalle de los datos almacenados para una determinada entrada en una lista.		<i>Eye</i>
Lápiz	Editar la información de una entrada concreta en una lista.		<i>Pencil</i>
Papelera	Eliminar una entrada de una lista.		<i>Trash</i>
Flecha de desplegable (<i>dropdown</i>) hacia abajo	Indicador de la dirección del despliegue del menú. <i>dropdown</i>		<i>Caret down</i>
Flecha derecha	Ejecuta el procesamiento de una fase de expediente.		<i>Arrow right</i>
Libro	Menú de expedientes		<i>Book</i>
Usuario con corbata	Menú de tutores.		<i>User tie</i>
Pergamino	Menú de acuerdos de estudios.		<i>Scroll</i>
Universidad	Menú de convenios.		<i>University</i>
Estudiante graduado	Menú de estudiantes.		<i>User graduate</i>
Campana	<ul style="list-style-type: none"> ■ Menú de notificaciones. ■ Acceso directo al menú de notificaciones. Icono persistente en la cabecera de twinX, junto al indicador de notificaciones pendientes. 		<i>Bell</i>
Nota adhesiva	Menú de recordatorios.		<i>Sticky note</i>
Ojo tachado	Marcar un mensaje como no leído en la bandeja de entrada.		<i>Eye slash</i>
Doble tic	Marcar un mensaje como leído en la bandeja de entrada.		<i>Check double</i>
Tic	<ul style="list-style-type: none"> ■ Indicador de valor «verdadero» en una vista. ■ Marcar un recordatorio como completado. 		<i>Check</i>
Cruz	<ul style="list-style-type: none"> ■ Indicador de valor «falso» en una vista. ■ Marcar un recordatorio como no completado. 		<i>Times</i>

Tabla 5: Iconos de twinX

5

DISEÑO TÉCNICO

Hemos definido hasta ahora gran parte de los ingredientes que llevará twinX: quiénes lo van a usar, qué es capaz de hacer, cuánto vamos a tardar en hacerlo y por qué hace falta algo como twinX.

Es hora de ponernos manos a la obra. Pero antes de ello, es necesario aún definir muchas más cosas, todo ello en relación con lo anterior pero a más bajo nivel: cuáles van a ser exactamente nuestras tareas, qué subtareas asociadas tendremos que desarrollar y cómo va a ser la base de datos. Será vez puestos todos los ingredientes a nuestra disposición para cocinarlos cuando podamos comenzar a utilizar nuestras herramientas para conseguir nuestro objetivo.

5.1 LISTADO INICIAL DEL PRODUCTO (PRODUCT BACKLOG)

Es cierto que ya sabemos lo que va a ser capaz de hacer twinX en su primera versión, pero no tenemos claros los pequeños pasos a dar para poder completar las grandes tareas que ya conocemos. Por ello, vamos a elaborar un listado de todo lo que tenemos que hacer en orden de preferencia, paso a paso, para poder conseguir nuestros objetivos de manera organizada.

La priorización de la lista de tareas suele estar hecha por el llamado *product owner*; esto es, el propietario del producto final (en este caso, las dos personas mencionadas en la sección [3.5.1](#)). Sin embargo, en este caso, vamos a obviar la labor de los propietarios, dado que no se está construyendo un producto desde cero, sino que se parte de lo que ya se construyó en TWINS y porque además, la intención de este proyecto no es obtener un producto inmediatamente utilizable, sino como propuesta de donde partir.

Más técnicamente, cada uno de los componentes de esta lista son acciones que el usuario podrá desarrollar. Esto es algo característico de Scrum y su constante enfoque en el usuario, todo ello en relación con su activa participación con el equipo de desarrollo durante todo el proceso. Pues bien, estas capacitaciones que el usuario final tendrá reciben el nombre de **historias de usuario**, y es la forma que se tiene en esta metodología ágil de nombrar a los clásicos requisitos. Suponen una reducción en

la documentación, ya que no se necesitan especificar demasiados elementos a priori. No obstante y como ya comentamos en secciones pasadas, el número de tareas y, por tanto, de historias de usuario, puede variar, pues al comienzo de cada sprint se realiza una reunión para evaluar los trabajos del equipo de desarrollo y se vuelve a revisar la priorización que se ha hecho de las tareas en el *product backlog*.

En referencia a esas «grandes tareas» que hemos mencionado anteriormente y que ya tenemos definidas, podríamos decir que dentro de ellas se esconden otras de menor generalidad que nos son más sencillas de descomponer o comprender a la hora del desarrollo. Es decir, son una serie de historias de usuario encerradas en otra más grande que alberga a todas ellas. Esto se conoce en el argot de Scrum como **épicas**. Así, «hacer uso del panel de control» sería una épica, pues dentro de ella irían historias de usuario como «ver un listado con todos los tipos de expedientes».

En la tabla 6 encontramos el backlog genérico del desarrollo, como producto de la síntesis que se puede hacer de las secciones 3.5.2 y 3.6.2, donde hemos incluido las historias de usuario a desarrollar para la primera versión de twinX, en sus épicas correspondientes. Entendemos que todas las historias numeradas como «x.1», «x.2», ..., «x.n» pertenecen todas a la épica «x». Recordemos también que el significado de «historia de usuario» no es otro que una acción a desarrollar por el usuario final del sistema, por lo que siempre tienen que estar descritas desde su punto de vista. Es asunto del desarrollador el cuestionarse las tareas (de código en su mayoría) que implica realizar cada una de ellas, por lo que para la elaboración del backlog no nos centraremos en esas tareas a llevar a cabo en cuanto a implementación, sino a satisfacción de las necesidades del usuario.

Identificador	Descripción	Sprint
HU.1	Acceso al panel de control.	1
HU.1.1	Listado de usuarios en todo el sistema: consulta, búsqueda, edición y eliminación de registros.	1
HU.1.2	Listado de tipos de expedientes almacenados: consulta, búsqueda, edición y eliminación de registros.	1
HU.1.3	Listado de fases de expedientes almacenadas: consulta, búsqueda, edición y eliminación de registros.	1
HU.1.4	Listado de mails predefinidos almacenados: consulta, búsqueda, edición y eliminación de registros.	1
HU.1.5	Listado de universidades almacenadas: consulta, búsqueda, edición y eliminación de registros.	1
HU.1.6	Listado de países almacenados: consulta, búsqueda, edición y eliminación de registros.	1
HU.2	Gestionar el trabajo diario de la oficina: estudiantes, expedientes, acuerdos de estudios, convenios y tutores.	1
HU.2.1	Ver la información más importante y de atención prioritaria de un vistazo.	1
HU.2.2	Gestionar los estudiantes: listado y vistas independientes con posibilidad de edición de sus datos.	1
HU.2.3	Gestionar los convenios: listado y vistas independientes con posibilidad de edición de los distintos campos.	1
HU.2.4	Gestionar los expedientes de los estudiantes: listado y vistas independientes con posibilidad de modificar sus fases.	1
HU.2.5	Gestionar los acuerdos de estudios: listado y vistas independientes con posibilidad de consultar los estudiantes relacionados y sus datos.	1
HU.2.6	Ver todos los tutores del sistema y consultar los acuerdos de estudios asociados a los mismos.	1
HU.3	Planificación de eventos y tareas.	2
HU.3.1	Crear nuevos eventos con tareas asociadas a los mismos.	2
HU.3.2	Configurar avisos de los eventos y tareas que se creen.	2
HU.3.3	Configurar recordatorios personales.	2
HU.4	Hacer uso de una mensajería entre los usuarios.	2

Tabla 6: Listado de historias de usuario

5.2 HISTORIAS DE USUARIO

A continuación, vamos a ofrecer una serie de tarjetas de todas y cada una de estas historias de usuario, planificadas para los dos primeros sprints y priorizadas como se ha especificado en la tabla 6.

Identificación	HU1 - Acceso al panel de control	Sprint: 1
Descripción	El panel de control posibilita al administrador crear, modificar y eliminar la información con la que trabajan los gestores en twinX	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta acceder al panel de control desde la sesión de un usuario sin permisos de administrador y el sistema no permite el acceso. 	
Tareas	<ul style="list-style-type: none"> ■ Crear el módulo de panel de control ■ Configurar el <i>layout</i> ■ Agregar el módulo a config/main.php ■ Agregar el nuevo módulo a la cabecera del sitio en el frontend 	

Tabla 7: Tabla de la HU1

Identificación	HU1.1 - Listado de usuarios en todo el sistema: consulta, búsqueda, edición y eliminación de registros.	Sprint: 1
Descripción	Un listado para que el administrador pueda ver todos los usuarios del sistema: tutores, estudiantes, gestores y otros posibles administradores. Se podrán editar los permisos y/o datos del usuario en concreto.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ El usuario a borrar es de tipo «estudiante» y tiene al menos un acuerdo de estudios registrado en el sistema, por lo que se impide la acción. ■ Se intenta cambiar el rango de un estudiante a «gestor», pero hay datos relacionados con el estudiante ya guardados (acuerdo de estudios, expedientes...), por lo que se impide la acción. ■ Se crea un usuario nuevo, pero se avisa al administrador que dicho usuario aún tiene que activar su cuenta. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de Usuario con Gii. ■ Construir la tabla CRUD¹ con Gii. 	

Tabla 8: Tabla de la HU1.1

¹ Create, Read, Update & Delete (crear, leer, actualizar y borrar)

Identificación	HU1.2 - Listado de tipos de expedientes almacenados: consulta, búsqueda, edición y eliminación de registros.	Sprint: 1
Descripción	El administrador podrá crear, modificar, buscar y eliminar tipos de expedientes para que los gestores puedan especificar nuevos estados en la movilidad de un estudiante concreto.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta borrar un tipo de expediente que está en uso y se deniega la acción. ■ Se modifica el nombre de un expediente en uso y se alerta al administrador de ello. La acción se realiza después de una confirmación. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de tipo de expediente con Gii. ■ Construir la tabla CRUD con Gii. 	

Tabla 9: Tabla de la HU1.2

Identificación	HU1.3 - Listado de fases de expedientes almacenadas: consulta, búsqueda, edición y eliminación de registros.	Sprint: 1
Descripción	El administrador podrá crear, modificar, buscar y eliminar fases de expedientes para poder especificar los eventos que suceden durante la tramitación de un expediente para un estudiante en concreto.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta borrar una fase de expediente que está en uso y se deniega la acción. ■ Se modifica el nombre de una fase de expediente en uso y se alerta al administrador de ello. La acción se realiza después de una confirmación. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de fase de expediente con Gii. ■ Construir la tabla CRUD con Gii. ■ Enlazar los nombres de los tipos de expedientes haciendo una consulta a la tabla correspondiente. ■ Adjuntar una vista para enlazar envíos de mails por defecto al procesar la fase. 	

Tabla 10: Tabla de la HU1.3

Identificación	HU1.4 - Listado de mails predefinidos almacenados: consulta, búsqueda, edición y eliminación de registros.	Sprint: 1
Descripción	El administrador podrá crear, modificar, buscar y eliminar mails predefinidos que poder adjuntar a una o varias fases en concreto para que sean enviados de forma automatizada al procesar la fase en cuestión.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta eliminar un mail predefinido que está en uso en alguna fase y se deniega la acción 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo con Gii. ■ Contruir la tabla CRUD con Gii. ■ Modificar los atributos a mostrar para que se muestren los nombres y no las claves primarias en base de datos. 	

Tabla 11: Tabla de la HU1.4

Identificación	HU1.5 - Listado de universidades almacenadas: consulta, búsqueda, edición y eliminación de registros.	Sprint: 1
Descripción	El administrador podrá crear, modificar, buscar y eliminar universidades para que puedan ser elegidas a la hora de especificar los datos de un convenio dado.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta eliminar una universidad en uso y se impide la acción. ■ Se intenta modificar una universidad en uso y se alerta al usuario de los riesgos. La acción se realiza tras una confirmación. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de universidad con Gii. ■ Construir la tabla CRUD con Gii. 	

Tabla 12: Tabla de la HU1.5

Identificación	HU1.6 - Listado de países almacenados: consulta, búsqueda, edición y eliminación de registros.	Sprint: 1
Descripción	El administrador podrá crear, modificar, buscar y eliminar países para su asociación con universidades, todo ello en relación con la creación de convenios.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta eliminar un país en uso y se impide la acción. ■ Se intenta modificar un país en uso y se alerta al usuario de los riesgos. La acción se realiza tras una confirmación. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de país con Gii. ■ Construir la tabla CRUD con Gii. 	

Tabla 13: Tabla de la HU1.6

Identificación	HU2 - Gestionar el trabajo diario de la oficina: estudiantes, expedientes, acuerdos de estudios, convenios y tutores.	Sprint: 1
Descripción	El menú de gestión es el que frecuentarán los trabajadores de la oficina, donde gestionarán todo lo que rodea a la movilidad internacional en la facultad.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta acceder a gestión desde la sesión de un usuario sin permisos de gestor y el sistema no permite el acceso. 	
Tareas	<ul style="list-style-type: none"> ■ Crear el módulo de gestión ■ Configurar el <i>layout</i> ■ Agregar el módulo a config/main.php ■ Agregar el nuevo módulo a la cabecera del sitio en el <i>frontend</i> 	

Tabla 14: Tabla de la HU2

Identificación	HU2.1 - Ver la información más importante y de atención prioritaria de un vistazo.	Sprint: 1
Descripción	Disposición de un panel que resuma todo lo más importante que un gestor deba atender en un día concreto; esto es, que resuma los asuntos a atender en una sola vista. Esta utilidad recibe el nombre de <i>dashboard</i>	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Si una tarjeta tiene mucho contenido, solo se mostrará una parte del mismo, para no entorpecer la visibilidad de las demás tarjetas. ■ Si una tarjeta no tiene contenido, se tendrá que avisar de ello en lugar de dejarla en blanco. 	
Tareas	<ul style="list-style-type: none"> ■ Crear la interfaz gráfica siguiendo los bocetos. ■ Asociar cada tarjeta de información con los registros necesarios en la base de datos. 	

Tabla 15: Tabla de la HU2.1

Identificación	HU2.2 - Gestionar los estudiantes: listado y vistas independientes con posibilidad de edición de los distintos campos.	Sprint: 1
Descripción	Los gestores dispondrán de una lista con todos los estudiantes con algún proceso de movilidad registrado. Desde ahí, podrán acceder a otros menús en relación con un estudiante en concreto.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta acceder al acuerdo de estudios de un estudiante que no tiene. Se muestra un mensaje de error. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de estudiante con Gii. ■ Costruir la tabla CRUD con Gii. ■ Desarrollar la vista de un estudiante. ■ Programar acciones especiales para cada entrada en la lista (acceso a acuerdo de estudios, convenio, expedientes...) 	

Tabla 16: Tabla de la HU2.2

Identificación	HU2.3 - Gestionar los convenios: listado y vistas independientes con posibilidad de edición de los distintos campos.	Sprint: 1
Descripción	Los gestores dispondrán de una lista con todos los convenios registrados en el sistema. Podrán consultarlos, buscarlos, añadir nuevos, editarlos y borrarlos. También podrán ver los estudiantes asociados a un convenio concreto para los que se tiene un acuerdo de estudios vigente.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta eliminar un convenio que tiene algún acuerdo de estudios vigente. No se lleva a cabo la acción y se alerta al usuario. 	
Tareas	<ul style="list-style-type: none"> ■ Crear el modelo de convenio con Gii. ■ Crear la tabla CRUD con Gii. ■ Crear la interfaz para la vista de convenios acorde con los bocetos y las necesidades. 	

Tabla 17: Tabla de la HU2.3

Identificación	HU2.4 - Gestionar los expedientes: listado y vistas independientes con posibilidad de edición de los distintos campos.	Sprint: 1
Descripción	Los gestores dispondrán de una lista con todos los expedientes registrados en el sistema. Podrán consultarlos, buscarlos, añadir nuevos, editarlos y borrarlos. Dispondrán de un menú interno al que acceder para ver los historiales de los expedientes: las fases por las que han pasado, ejecutar algún cambio de fase o consultar datos relacionados con los estudiantes, sus acuerdos y los convenios asociados.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta procesar más de una fase al mismo tiempo. No se lleva a cabo la acción y se alerta al usuario. 	
Tareas	<ul style="list-style-type: none"> ■ Crear el modelo de expediente con Gii. ■ Crear la tabla CRUD con Gii. ■ Diseñar la interfaz gráfica de acuerdo con los bocetos y las necesidades. ■ Crear los accesos directos a la vista de estudiante y a la de convenio. 	

Tabla 18: Tabla de la HU2.4

Identificación	HU2.5 - Gestionar los acuerdos de estudios: listado y vistas independientes con posibilidad de edición de los distintos campos.	Sprint: 1
Descripción	Los gestores dispondrán de una lista con todos los acuerdos de estudios registrados en el sistema. Podrán consultarlos, buscarlos, añadir nuevos, editarlos y borrarlos. También podrán ver el convenio al que se asocia un acuerdo de estudios dado y los expedientes de ese acuerdo para con el estudiante que lo tiene registrado.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta eliminar un acuerdo vigente. No se lleva a cabo la acción y se alerta al usuario. 	
Tareas	<ul style="list-style-type: none"> ■ Crear el modelo de convenio con Gii. ■ Crear la tabla CRUD con Gii. ■ Crear los accesos directos a la vista de estudiante y a la de convenio. 	

Tabla 19: Tabla de la HU2.5

Identificación	HU2.6 - Ver todos los tutores del sistema y consultar los acuerdos de estudios asociados a los mismos.	Sprint: 1
Descripción	Los gestores dispondrán de una lista con todos los tutores activos en el sistema (esto es, tutorizando algún acuerdo de estudios). Podrán consultarlos, buscarlos y ver información relacionada, como los acuerdos de estudios que tutoriza. También se podrá asignar un tutor a un acuerdo de estudios existente y que no tenga tutor previamente.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta asignar un tutor a un acuerdo de estudios que ya tiene tutor o el cual no se encuentra vigente. No se lleva a cabo la acción y se alerta al usuario. 	
Tareas	<ul style="list-style-type: none"> ■ Crear el modelo de tutor con Gii. ■ Crear la tabla CRUD con Gii. ■ Crear los accesos directos a la vista de acuerdo de estudios. 	

Tabla 20: Tabla de la HU2.6

Identificación	HU3 - Planificación de eventos y tareas.	Sprint: 2
Descripción	Los gestores tendrán la posibilidad de crear eventos y tareas para organizar mejor su trabajo	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta visualizar el calendario sin permisos mínimos de gestor. La plataforma deniega el acceso. 	
Tareas	<ul style="list-style-type: none"> ■ Crear el módulo de calendario. ■ Configurar el <i>layout</i>. ■ Agregar el módulo a config/main.php. ■ Agregar el nuevo módulo a la cabecera del sitio en el <i>frontend</i>. 	

Tabla 21: Tabla de la HU3

Identificación	HU3.1 - Crear nuevos eventos con tareas asociadas a los mismos.	Sprint: 2
Descripción	Los gestores podrán crear eventos comunes a todo el personal de gestión de la plataforma y complementarlos usando tareas.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta crear un evento con una fecha anterior a la actual y el sistema alerta del error e impide la operación. ■ Se intenta eliminar un evento que no ha finalizado y que tiene tareas pendientes. El sistema avisa y efectúa la acción tras una confirmación. 	
Tareas	<ul style="list-style-type: none"> ■ Crear los modelos de evento y tarea con Gii. ■ Crear la tabla CRUD con Gii y modificarla de acuerdo con los bocetos. ■ Diseñar la interfaz de usuario para la creación de eventos y tareas. 	

Tabla 22: Tabla de la HU3.1

Identificación	HU3.2 - Configurar avisos de los eventos y tareas que se creen.	Sprint: 2
Descripción	Los gestores podrán asignar avisos a los eventos y tareas creados para ser notificados cuando elijan y a la persona a la que se los asignen.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta poner un recordatorio en una fecha anterior a la actual. El sistema lo notifica e impide la acción. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de la tabla deadline_aviso con Gii. ■ Modificar las vistas de calendario y tarea para agregar los recordatorios. ■ Añadir al menú de notificaciones la aparición de avisos por evento o tarea pendiente. 	

Tabla 23: Tabla de la HU3.2

Identificación	HU3.3 - Configurar los recordatorios personales.	Sprint: 2
Descripción	Los gestores tendrán la posibilidad de programar recordatorios para recibir un aviso en el futuro.	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta programar un recordatorio para una fecha anterior a la actual. El sistema lanza una aviso e impide la acción. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de recordatorio con Gii. ■ Construir la tabla CRUD con Gii. ■ Configurar el panel de notificaciones para que muestre también los recordatorios programados. 	

Tabla 24: Tabla de la HU3.3

Identificación	HU4 - Hacer uso de la mensajería entre los usuarios.	Sprint: 2
Descripción	Los usuarios pueden enviarse mensajes entre ellos mismos: consultar los mensajes recibidos, escribir nuevos y cambiar la etiqueta de los ya existentes (importante, eliminado, etc.)	
Pruebas de aceptación	<ul style="list-style-type: none"> ■ Se intenta enviar un mensaje a un usuario no existente. El sistema alerta del error y el mensaje no es enviado. 	
Tareas	<ul style="list-style-type: none"> ■ Construir el modelo de mensaje con Gii. ■ Construir la tabla CRUD con Gii. ■ Modificar la interfaz para asemejarla a una bandeja de entrada de mensajes. ■ Configurar las notificaciones de recepción de mensajes. 	

Tabla 25: Tabla de la HU4

Hemos plasmado un total de 19 historias de usuario, con las tareas que implicará cada una. Es obvio que en la práctica, no solo variará el número de tareas asociado a las HHUU, sino que muy probablemente, tampoco acabemos implementando exactamente las que tenemos aquí, sino que podrían surgir otras o incluso cambiar la priorización de unas a otras en el cambio del sprint 1 al 2. Sobre la implementación de las historias de usuario y el progreso que las acompaña, hablaremos en el capítulo 6

5.3 MODELO DE LA BASE DE DATOS

Cada vez tenemos una idea más clara de cómo va a ser twinX, pero antes hemos de especificar uno de los puntos más importantes para hacer posible el producto final como es la base de datos. En ella, tenemos que especificar las restricciones que hay, cuáles atributos tienen que ser compartidos entre tablas (claves externas), cuáles no se pueden repetir en una relación (claves primarias) y cuáles no pueden estar vacíos.

Nótese la importancia de dedicar tiempo al desarrollo de esta tarea: estableciendo las restricciones necesarias, aunque tengamos algún olvido a la hora de implementar la interfaz con la que el usuario interaccionará, siempre tendremos el respaldo de que la base de datos no comprometerá su estado de consistencia. Por ejemplo, no podemos dejar que se elimine un acuerdo de estudios para el cual hay algún estudiante registrado. O incluso tampoco podemos crear más de un acuerdo de estudios para un estudiante por curso.

El modelo de la base de datos creado es el que se propone en la figura 38. El análisis ha dado lugar a un total de 30 tablas, donde se contemplan funcionalidades que no están previstas de implementación en los dos primeros sprints, como son por ejemplo el historial de mensajes enviados, los acuerdos favoritos de un gestor o la confección del acuerdo de estudios a través de la plataforma. No obstante, el diseño se ha pensado para tener en cuenta el futuro desarrollo de twinX y así no desviarlo de la línea de las ideas principales que ya hay definidas. El diagrama ha sido realizado con la herramienta dbdiagram.io [25].

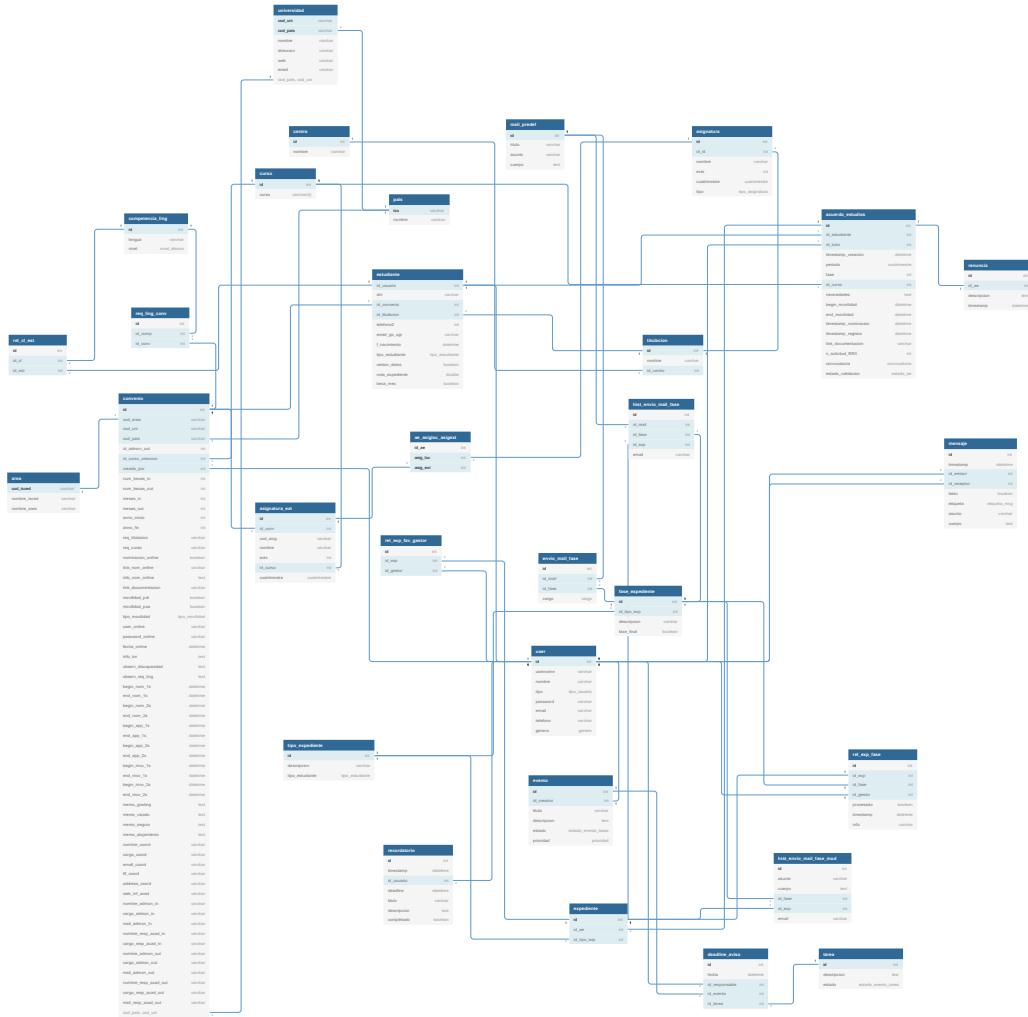


Figura 38: Modelo de la base de datos de twinX

El cambio con respecto a TWINS es muy grande. Si bien es cierto que en la aplicación ya existente se pueden encontrar tablas para fines varios que no figuran en la planificación de twinX, hay muchas tablas que pueden evitarse; aunque, sobre todo, prima la duplicidad y redundancia de atributos. Por ejemplo, en TWINS se almacenaban algunos atributos como la nota extra que se tiene al optar por una plaza de movilidad en función de las competencias lingüísticas, algo que es fácilmente calculable comparando los requisitos que exige el convenio y las competencias del estudiante. En función de ello, se puede asignar una mayor nota si procede.

Del modelo, destaca la extensión de la tabla de convenios, donde hay mucha información que guardar. Hay campos para los que ha sido complicado escoger un nombre para definir el atributo; más concretamente, estos pueden ser los de comienzo y fin de las nominaciones, apliaciones y los de la movilidad (begin_nom_1s correspondería a «fecha de comienzo de la movilidad en el primer semestre», etc.). Uno de los puntos clave estará en disponer toda esta información de manera organizada, para ofrecer al usuario una buena experiencia pero aportando, al mismo tiempo, facilidad de uso y practicidad para con su trabajo diario.

Otra tabla importante, aunque no extensa, es la de usuario. Tiene muchas relaciones, ya que puede ser definida como una especificación con tutor, gestor, administrador, y estudiante. Ya que en futuras versiones twinX pretende ofrecer acceso a los estudiantes también para sus gestiones, se ha incluido en esta tabla los atributos comunes a todo el público del portal. Como consecuencia, esto nos ahorra tener una tabla con gestores o tutores, pues tan solo habría que especificar esto con un atributo (`tipo_usuario`), algo que en TWINS estaba separado, aumentando el número de tablas. Esto también es aplicable a la tabla `deadline_aviso`, donde se aglomeran los avisos tanto ligados a eventos en general como a las tareas creadas para un evento.

Por último, comentaremos que para el envío de mensajes predefinidos de forma automatizada al cambiar un [Expediente de twinX](#) de [Fase de expediente](#) tiene un registro a modo de historial en la tabla `hist_envio_mail_fase` y que se establece una alternativa (`hist_envio_mail_fase_mod`) para cuando se quiera enviar un email que no está predefinido, conducta que ya se aprecia en TWINS, donde se guarda el texto completo de cada correo que se manda, esté o no predefinido en la base de datos.

5.4 ARQUITECTURA DE TWINX

Para el desarrollo de twinX vamos a emplear, como era de esperar una vez sabido que se trata de un sistema web, la patrón arquitectónico Modelo-Vista-Controlador (MVC). Enfocado en la modularización, separamos estas tres entidades para poder dejar de un lado los datos (**modelo**), que complementan a la interfaz gráfica (**vista**), todo ello impulsado y organizado por el **controlador**, que actúa de mediador entre ambas entidades.

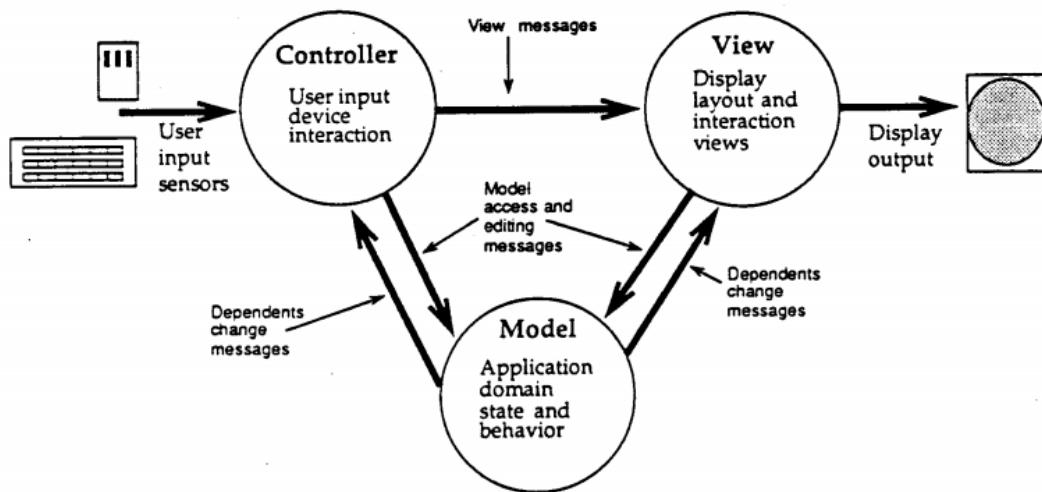


Figura 39: Esquema del patrón arquitectónico Modelo-Vista-Controlador

Este aislamiento de unidades funcionales facilita la comprensión y hace que sea mucho más sencillo modificar una unidad particular de todo el sistema, sin necesidad de conocer el funcionamiento de las demás. Pensemos que en proyectos de mayor envergadura, esto es muy potente, puesto que una persona podría dedicarse al mantenimiento de una única parte de todo el conjunto, lo que dotaría al desarrollador de independencia para poder realizar su trabajo de forma individual sin tener que recurrir a la constante supervisión por los responsables de otros módulos que, aunque puedan estar relacionados y/o comunicados están, ante todo, aislados ??.

Su funcionamiento es sencillo: en primera instancia, el usuario establece conexión con el controlador, el cual se conecta con el modelo, le solicita los datos con los que llenar la interfaz. Una vez sabe qué datos necesita ver el usuario, le hace saber a la vista la respuesta obtenida del modelo, y es entonces cuando se renderiza y se muestra en el navegador. Lo mismo ocurre cuando el usuario modifica algún dato en pantalla: el controlador recupera los cambios de la vista para enviárselos al modelo, el cual lo procesa y genera una respuesta, informando nuevamente al controlador del éxito o fracaso de la acción. Una vez éste se hace con la respuesta, vuelve a llamar a la vista con la información actualizada. Este comportamiento puede verse resumido de forma gráfica en la figura 39.

El framework escogido para el desarrollo de twinX, Yii2, impulsa este patrón mediante su funcionamiento separado en modelos, vistas y controladores, como ya veremos en el capítulo próximo. Sin embargo, el MVC no constituye la arquitectura de la aplicación al completo, sino que también intervienen la capa de conexión con la base de datos y la capa lógica. Ambas están suministradas por el propio framework, con el manejo de las mismas de forma interna, dando acceso a ellas a través de objetos.

6

IMPLEMENTACIÓN

Hasta ahora, con todas las piezas de información que hemos generado acerca del desarrollo, hemos podido, en efecto, discernir entre tres grandes grupos de tareas: la gestión, el calendario y el panel de control. Así pues, estableceremos la separación del desarrollo en estos tres grandes bloques de trabajo, con la previa especificación de las herramientas a utilizar y del trabajo anterior al comienzo del proceso de programación, propiamente dicho, que implican la puesta a punto de la base de datos y del servidor para probar y desarrollar nuestra aplicación.

6.1 HERRAMIENTAS PARA EL DESARROLLO DE TWINX

Para llevar a cabo la programación de este sitio web, aunque utilicemos herramientas más sofisticadas, éstas están basadas en los clásicos lenguajes web: **HTML** (*HyperText Markup Language*) [30] para la estructuración de la web y la identificación de los distintos elementos, **CSS** [5] (*Cascading Style Sheets*) para aplicar estilo a la vista (posicionamiento, colores, visibilidad, etc.) y JavaScript [18], para implementar funciones más complejas a cada componente de la web. Para el manejo de estos dos últimos, contaremos con la ayuda de dos librerías con un gran potencial. Para el estilo, usaremos **Bootstrap** [4] en su cuarta versión y para JavaScript utilizaremos **jQuery** [13]. Nos permitirán ahorrar mucho tiempo, ya que en ambos casos podemos hacer uso de funciones o palabras clave que evitarán el tener que escribir numerosas líneas de código en cada uno de los lenguajes a los que complementan.

Como ya hemos mencionado en varias ocasiones, el desarrollo va a ser llevado a cabo con la ayuda de **Yii2 Framework**¹. En su segunda versión, este framework hace uso del lenguaje de programación PHP para orquestar múltiples tareas que resultan engorrosas de llevar a cabo sin su ayuda y que pueden extenderse demasiado, como son, por ejemplo, las operaciones con la base de datos o la generación de código del que partir para dar forma a la vista, modelo o controlador de alguna de las secciones de twinX [32]. Hoy en día, en proyectos de desarrollo, sería muy complicado

¹ Sus siglas significan *Yes, it is!*

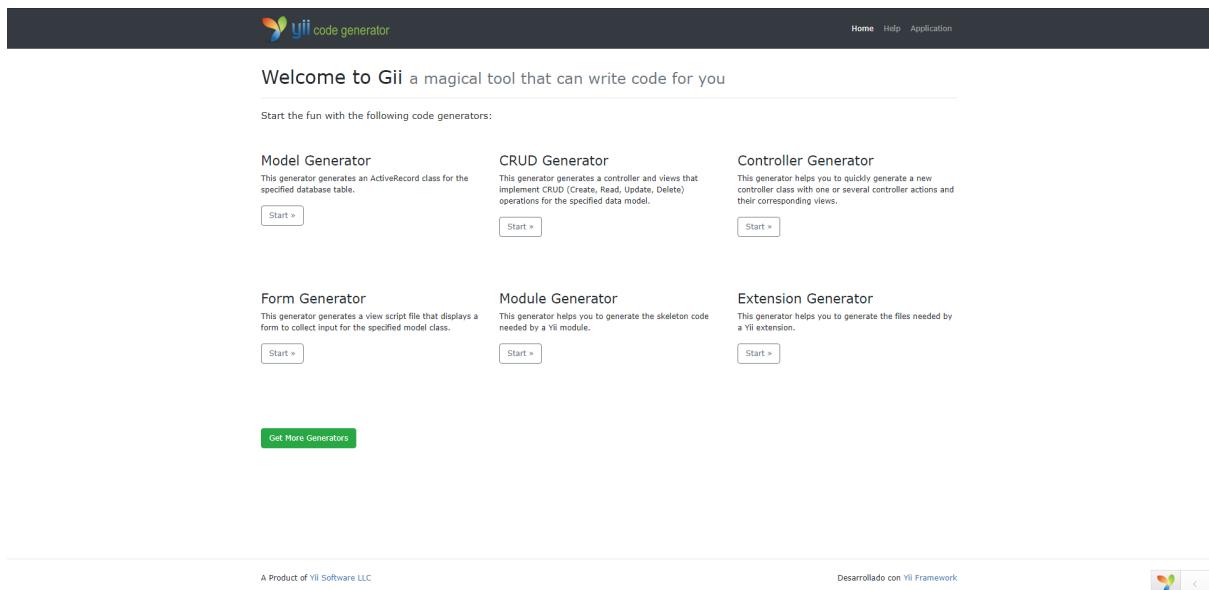


Figura 40: Portal de Gii desde donde podemos seleccionar un tipo de elemento a crear para que genere su código fuente

prescindir de una herramienta como esta, ya que las alternativas son numerosas y completas.

Entre sus características, destacamos el generador de código **Gii** (figura 40), que puede automatizar la creación, por ejemplo, de la clase de un modelo, o incluso de una tabla CRUD al completo (esto es, su vista y su controlador). Tan solo tenemos que indicar cuál es la tabla de la base de datos sobre la que queremos generar los fragmentos de código, y atendiendo a las restricciones creadas en la misma, se puede obtener fácilmente una clase instanciable para recuperar o guardar información en su tabla. Del mismo modo, al crear una tabla CRUD, tendríamos directamente en nuestro sitio web una interfaz donde pueden verse los registros almacenados en la base de datos, con la posibilidad de crear nuevos, eliminarlos o incluso filtrarlos y editarlos. No cabe duda del potencial de esta característica, la cual nos permitirá ahorrar mucho tiempo y seguridad a la hora de desarrollar y de depositar la confianza en código automáticamente generado y sin errores.

Para comenzar a usar Yii, lo primero que tenemos que hacer es hacernos con el gestor de dependencias de PHP **Composer** [1] y descargarnos el esqueleto de un proyecto en Yii2 avanzado [33]. Para ello, basta con escribir:

```
composer create-project -prefer-dist yiisoft/yii2-app-advanced twinX
```

Y con ello ya tendríamos la carpeta con nuestro proyecto. Concretamente este directorio es el que tenemos que servir y al que nos conectaremos para visualizar nuestra

web. También tenemos que seguir los pasos en el repositorio de GitHub de yiisoft para aplicar los ajustes necesarios derivados del renombramiento y redireccionamiento de URLs en el servidor. Además, tendremos que instalar algunas dependencias haciendo uso de la orden `composer install` y de iniciar el entorno de desarrollo con el archivo `ini` de PHP que incluye el repositorio que hemos clonado.

A continuación, procederemos con la instalación de la pila software **XAMPP** (Apache, MariaDB, PHP y Perl). En Windows, es una instalación sencilla a través de una interfaz de usuario, por lo que no requiere un gran tiempo. Es importante que, una vez instalado, situemos la carpeta del proyecto dentro del directorio `htdocs`, en la raíz de la carpeta de instalación de `xampp`. Se recomienda instalar en el directorio C: en Windows. En nuestro caso, hemos hecho los ajustes necesarios para poder situar esta carpeta en otro directorio desde donde se trabaja con el repositorio GitHub del proyecto, para que de forma más cómoda podamos salvaguardar el progreso del desarrollo de forma periódica sin tener que copiar archivos de un directorio que esté fuera del repositorio.

Esta pila contiene la aplicación **phpMyAdmin**, que ha sido también una gran ayuda para poder visualizar las tablas en base de datos, acceder a registros concretos o ejecutar código MySQL desde una interfaz gráfica. Frente a una terminal desde donde manejar la base de datos, presenta una menor tasa de errores al introducir órdenes, pues la información puede verse a golpe de click y con una presentación más vistosa.

Sobre GitHub, el repositorio se creó al comienzo de la redacción de esta memoria [6], donde mediante la creación de ramas hemos ido guardando en distintos *commits* las progresivas versiones del proyecto. Una vez creada una pieza de valor, se hacía una mezcla (*merge*) a la rama `master`, de modo que cada rama se creaba con un fin, para añadir una nueva funcionalidad, sin depender del funcionamiento de las demás y que poco a poco se puedan ir integrando todas las características en su totalidad.

Finalmente, otro de los protagonistas del desarrollo es el IDE² con que se ha programado twinX. En este caso, haciendo uso de su licencia educativa, se ha usado **PhpStorm** [15]. Ha sido esencial, pues hemos descartado otras herramientas como Visual Studio Code ya que no tienen tan buen integración con PHP y Yii en sí como tiene este IDE. Destacan también características como su guardado automático inteligente, su potente búsqueda de archivos y sus cómodos atajos de teclado. Todo ello han hecho el proceso de desarrollo muy cómodo y liviano.

6.2 CREACIÓN DE LA BASE DE DATOS

Tal y como hemos indicado en la sección 5.3, para crear el esquema del modelo de la base de datos, hemos usado la herramienta **dbdiagram.io** [25]. Ésta tiene mucho potencial, pues no es un simple creador de diagramas. Su principal característica es la confección de la parte gráfica mediante una especie de lenguaje creado por los

² *Integrated Development Environment*

mismos creadores, DBML [24]. Con él, se pueden especificar tablas, atributos, estructuras de enumeración y características de los atributos, como claves externas, primarias, cardinalidades y nulidad. Gracias a ello, se ha podido mejorar progresivamente el modelo de la base de datos, pues un cambio en el código implica una variación en la visualización de las tablas, bien sea con algún atributo de más o alguna nueva relación entre tablas.

No solo es posible ver con mayor claridad las características del modelo en el código en DBML, sino que también tiene la gran posibilidad de exportarlo a otro lenguaje como es MySQL. De este modo, con tan solo unos clicks, se obtienen las órdenes que posteriormente nuestra base de datos entenderá y podrá crear todas las tablas por nosotros, con todas las restricciones establecidas y ahorrando, así, mucho tiempo.

En la práctica, el código que tenemos en DBML para generar el modelo de la figura 38 es:

```
// ENTIDADES BASICAS

Table centro as CEN {
    id int [pk, increment]
    nombre varchar [not null, unique]
}

Table curso as CURSO {
    id int [pk, increment]
    curso varchar(4) [not null]
}

Table mail_predef as MAIL {
    id int [pk, increment]
    titulo varchar [not null, unique]
    asunto varchar [not null]
    cuerpo text [not null]
}

Table titulacion as TIT {
    id int [pk]
    nombre varchar [not null, unique]
    id_centro int [not null]
}

Ref: TIT.id_centro > CEN.id

Table asignatura as ASIG {
    id int [pk]
```

```

    id_tit int [not null]
    nombre varchar [not null]
    ects int [not null]
    cuatrimestre cuatrimestre [not null]
    tipo tipo_asignatura [not null]
}

```

Ref: ASIG.id_tit > TIT.id

```

Table asignatura_ext as EXT_ASIG {
    id int [pk]
    id_conv int [not null]
    nombre varchar [not null]
    ects int [not null]
    id_curso int [ref: > CURSO.id, not null]
    cuatrimestre cuatrimestre [not null]
}

```

Ref: EXT_ASIG.id_conv > CON.id

```

Table pais as P {
    iso varchar [pk]
    nombre varchar [not null]
}

```

```

Table universidad as UNI {
    cod_uni varchar [not null]
    cod_pais varchar [not null]
    nombre varchar [not null]
    direccion varchar
    web varchar
    email varchar

    Indexes {
        (cod_pais, cod_uni) [pk]
    }
}

```

Ref: UNI.cod_pais > P.iso

```

Table area as AR {
    cod_isced varchar [pk]
    nombre_isced varchar [not null]
}

```

```

    nombre_area varchar
}

// USUARIOS Y TIPOS

Table user as U {
    id int [pk, increment]
    username varchar
    nombre varchar
    tipo tipo_usuario
    password varchar
    email varchar
    telefono varchar
    genero genero
}

Table estudiante as EST{
    id_usuario int [not null]
    dni varchar [not null, unique] // PUEDE SER
        NIE
    id_convenio int [not null]
    id_titulacion int [not null]
    telefono2 int
    email_go_ugr varchar
    f_nacimiento datetime [not null]
    tipo_estudiante tipo_estudiante [not null]
    cesion_datos boolean
    nota_expediente double
    beca_mec boolean
}

```

Ref: EST.id_titulacion > TIT.id

Ref: U.id – EST.id_usuario

Ref: EST.id_convenio > CON.id

// CONVENIOS Y ACUERDOS

```

Table competencia_ling as CL {
    id int [pk, increment]
    lengua varchar [not null]
    nivel nivel_idioma [not null]

```

```

}

Table rel_cl_est {
    id int [pk, increment]
    id_cl int [ref: > CL.id, not null]
    id_est int [ref: > EST.id_usuario, not null]
}

Table req_ling_conv {
    id int [pk, increment]
    id_comp int [ref: > CL.id, not null]
    id_conv int [ref: > CON.id, not null]
}

Table convenio as CON {
    id int [pk, increment]
    cod_area varchar [not null]
    cod_uni varchar [not null]
    cod_pais varchar [not null]
    id_admon_out int

    id_curso_creacion int [ref: > CURSO.id, not
                           null]
    creado_por int [not null]

    num_becas_in int [not null]
    num_becas_out int [not null]

    meses_in int [not null]
    meses_out int [not null]

    anno_inicio int [not null]
    anno_fin int [not null]

    req_titulacion varchar
    req_curso varchar

    nominacion_online boolean
    link_nom_online varchar
    info_nom_online text

    link_documentacion varchar
}

```

```
movilidad_pdi boolean
movilidad_pas boolean

tipo_movilidad tipo_movilidad [not null]

user_online varchar
password_online varchar
fecha_online datetime

info_tor text

observ_discapacidad text
observ_req_ling text

begin_nom_1s datetime
end_nom_1s datetime
begin_nom_2s datetime
end_nom_2s datetime
begin_app_1s datetime
end_app_1s datetime
begin_app_2s datetime
end_app_2s datetime
begin_mov_1s datetime
end_mov_1s datetime
begin_mov_2s datetime
end_mov_2s datetime

memo_grading text
memo_visado text
memo_seguro text
memo_alojamiento text

nombre_coord varchar
cargo_coord varchar
email_coord varchar
tlf_coord varchar
address_coord varchar [note:"Por defecto igual
que la de la universidad"]
web_inf_acad varchar

nombre_admon_in varchar
cargo_admon_in varchar
mail_admon_in varchar
```

```

nombre_resp_acad_in varchar
cargo_resp_acad_in varchar

nombre_admon_out varchar
cargo_admon_out varchar
mail_admon_out varchar
nombre_resp_acad_out varchar
cargo_resp_acad_out varchar
mail_resp_acad_out varchar

}

Ref: P.iso < CON.cod_pais
Ref: CON.creado_por > U.id
Ref: CON.cod_area > AR.cod_isced
Ref: CON.(cod_pais, cod_uni) > UNI.(cod_pais, cod_uni)

```

```

Table acuerdo_estudios as AE {
    id int [pk, increment]
    id_estudiante int [not null]
    id_tutor int [not null]

    timestamp_creacion datetime [not null]
    periodo cuatrimestre [not null]
    fase int [not null]
    id_curso int [ref: > CURSO.id, not null]
    necesidades text
    begin_movilidad date // comienzo individual
        del estudiante
    end_movilidad date // fin individual del
        estudiante
    timestamp_nominacion datetime
    timestamp_registro datetime [not null]
    link_documentacion varchar
    n_solicitud_RRII int
    convocatoria convocatoria [not null]
}

```

```

Ref: AE.id_estudiante > EST.id_usuario
Ref: AE.id_tutor > U.id

```

```
Table ae_asigloc_asigext {
```

```

    id_ae int
    asig_loc int [ref: > ASIG.id]
    asig_ext int [ref: > EXT_ASIG.id]

    Indexes {
        (id_ae , asig_loc , asig_ext) [pk]
    }
}

```

```

Table renuncia as REN{
    id int [pk, increment]
    id_ae int [not null, ref: - AE.id]
    descripcion text [not null]
    timestamp datetime [not null]
}

```

// EXPEDIENTES, TIPOS DE EXPEDIENTE Y FASES

```

Table expediente as EXP{
    id int [pk, increment]
    id_ae int [not null]
    id_tipo_exp int [not null]
}

```

Ref: EXP.id_ae > AE.id
 Ref: EXP.id_tipo_exp > TIPO_EXP.id

```

Table tipo_expediente as TIPO_EXP {
    id int [pk, increment]
    descripcion varchar [not null]
    tipo_estudiante tipo_estudiante [not null]
}

```

```

Table fase_expediente as FAS_EXP {
    id int [pk, increment]
    id_tipo_exp int [not null]
    descripcion varchar [not null]
    fase_final boolean [default: 0]
}

```

Ref: FAS_EXP.id_tipo_exp > TIPO_EXP.id

```
Table envio_mail_fase {
```

```

    id int [pk, increment]
    id_mail int [ref: > MAIL.id, not null]
    id_fase int [ref: > FAS_EXP.id, not null]
    cargo cargo [not null]
}

```

```

Table hist_envio_mail_fase {
    id int [pk, increment]
    id_mail int [ref: > MAIL.id, not null]
    id_fase int [ref: > FAS_EXP.id, not null]
    id_exp int [ref: > EXP.id, not null]
    email varchar [not null]
}

```

```

Table hist_envio_mail_fase_mod { // Mail modificado
    id int [pk, increment]
    asunto varchar
    cuerpo text [not null]
    id_fase int [ref: > FAS_EXP.id, not null]
    id_exp int [ref: > EXP.id, not null]
    email varchar [not null]
}

```

```

Table rel_exp_fase as REL_EF {
    id int [pk]
    id_exp int [not null]
    id_fase int [not null]
    id_gestor int [not null]
    procesado boolean
    timestamp datetime [not null]
    info varchar
}

```

Ref: REL_EF.id_exp – EXP.id

Ref: REL_EF.id_fase – FAS_EXP.id

Ref: REL_EF.id_gestor – U.id

```

Table rel_exp_fav_gestor {
    id int [pk, increment]
    id_exp int [ref: > EXP.id]
    id_gestor int [ref: > U.id]
    //tipo_usuario == GESTOR
}

```

```
// CALENDARIO

Table evento as EV {
    id int [pk, increment]
    id_creador int [not null]
    titulo varchar [not null]
    descripcion text
    estado estado_evento_tarea [not null]
    prioridad prioridad [not null, default: 'MEDIA']
}
}
```

Ref: EV.id_creador - U.id // tipo == GESTOR

```
Table tarea as TASK {
    id int [pk, increment]
    descripcion text [not null]
    estado estado_evento_tarea [not null, default:
        'PENDIENTE']
}
}
```

```
Table deadline_aviso as DLA {
    id int [pk, increment]
    fecha datetime [not null]
    id_responsable int [not null]
    id_evento int [not null]
    id_tarea int [not null]
}
}
```

Ref: DLA.id_responsable > U.id // tipo == GESTOR

Ref: DLA.id_evento > EV.id

Ref: DLA.id_tarea > TASK.id

```
Table recordatorio {
    id int [pk, increment]
    timestamp datetime [not null]
    id_usuario int [ref: > U.id, not null]
    deadline datetime [not null]
    titulo varchar [not null]
    descripcion text
    completado boolean
}
}
```

```
// MENSAJES
Table mensaje as MSG{
    id int [pk, increment]
    timestamp datetime [not null]
    id_emisor int [not null]
    id_receptor int [not null]
    leido boolean
    etiqueta etiqueta_msg [not null]
    asunto varchar
    cuerpo text [not null]

}

Ref: MSG.id_emisor > U.id
Ref: MSG.id_receptor > U.id

Enum tipo_asignatura {
    TRONCAL
    OBLIGATORIA
    OPTATIVA
}

Enum cargo {
    COORDINADOR
    ADMON_IN
    ADMON_OUT
    RESP_ADMON_OUT
}

Enum nivel_idioma {
    B1
    B2
    C1
    C2
}

Enum etiqueta_msg {
    IMPORTANTE
    ELIMINADO
}
```

```
Enum estado_evento_tarea {
    PENDIENTE
    EN_PROCESO
    TERMINADO
}

Enum prioridad {
    ALTA
    MEDIA
    BAJA
}

Enum estado_ae {
    REVISION
    DENEGADO
    ACEPTADO
    VIGENTE
}

Enum convocatoria {
    PRIMERA
    SEGUNDA
    EXTRAORDINARIA
}

Enum tipo_movilidad {
    ERASMUS
    ARQUIS
    ERASMUS_DI
    ERASMUS_PARTNER
    INTERCAMBIO
    LIBRE_MOVILIDAD
}

Enum tipo_usuario {
    SUPERUSUARIO
    GESTOR
    ESTUDIANTE
    TUTOR
}

Enum genero {
```

```

    F
    M
    O
}

Enum tipo_estudiante {
    INCOMING
    OUTGOING
}

Enum cuatrimestre {
    PRIMERO
    SEGUNDO
    C_COMPLETO
}

```

Y como resultado, podemos traducir todo ello a código MySQL, quedando de la siguiente manera:

```

CREATE TABLE `centro` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`nombre` varchar(255) UNIQUE NOT NULL
);

CREATE TABLE `curso` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`curso` varchar(4) NOT NULL
);

CREATE TABLE `mail_predef` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`titulo` varchar(255) UNIQUE NOT NULL,
`asunto` varchar(255) NOT NULL,
`cuerpo` text NOT NULL
);

CREATE TABLE `titulacion` (
`id` int PRIMARY KEY,
`nombre` varchar(255) UNIQUE NOT NULL,
`id_centro` int NOT NULL
);

CREATE TABLE `asignatura` (
`id` int PRIMARY KEY,

```

```
`id_tit` int NOT NULL,  
`nombre` varchar(255) NOT NULL,  
`ects` int NOT NULL,  
`cuatrimestre` ENUM ('PRIMERO', 'SEGUNDO', 'C_COMPLETO') NOT NULL,  
`tipo` ENUM ('TRONCAL', 'OBLIGATORIA', 'OPTATIVA') NOT NULL  
);  
  
CREATE TABLE `asignatura_ext` (  
`id` int PRIMARY KEY,  
`id_conv` int NOT NULL,  
`nombre` varchar(255) NOT NULL,  
`ects` int NOT NULL,  
`id_curso` int NOT NULL,  
`cuatrimestre` ENUM ('PRIMERO', 'SEGUNDO', 'C_COMPLETO') NOT NULL  
);  
  
CREATE TABLE `pais` (  
`iso` varchar(255) PRIMARY KEY,  
`nombre` varchar(255) NOT NULL  
);  
  
CREATE TABLE `universidad` (  
`cod_uni` varchar(255) NOT NULL,  
`cod_pais` varchar(255) NOT NULL,  
`nombre` varchar(255) NOT NULL,  
`direccion` varchar(255),  
`web` varchar(255),  
`email` varchar(255),  
PRIMARY KEY (`cod_pais`, `cod_uni`)  
);  
  
CREATE TABLE `area` (  
`cod_isced` varchar(255) PRIMARY KEY,  
`nombre_isced` varchar(255) NOT NULL,  
`nombre_area` varchar(255)  
);  
  
CREATE TABLE `user` (  
`id` int PRIMARY KEY AUTO_INCREMENT,  
`username` varchar(255),  
`nombre` varchar(255),  
`tipo` ENUM ('SUPERUSUARIO', 'GESTOR', 'ESTUDIANTE', 'TUTOR'),  
`password` varchar(255),  
`email` varchar(255),  
`telefono` varchar(255),  
`genero` ENUM ('F', 'M', 'O')
```

```

);

CREATE TABLE `estudiante` (
`id_usuario` int NOT NULL,
`dni` varchar(255) UNIQUE NOT NULL,
`id_convenio` int NOT NULL,
`id_titulacion` int NOT NULL,
`telefono2` int,
`email_go_ugr` varchar(255),
`f_nacimiento` datetime NOT NULL,
`tipo_estudiante` ENUM ('INCOMING', 'OUTGOING') NOT NULL,
`cesion_datos` boolean,
`nota_expediente` double,
`beca_mec` boolean
);

CREATE TABLE `competencia_ling` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`lengua` varchar(255) NOT NULL,
`nivel` ENUM ('B1', 'B2', 'C1', 'C2') NOT NULL
);

CREATE TABLE `rel_cl_est` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_cl` int NOT NULL,
`id_est` int NOT NULL
);

CREATE TABLE `req_ling_conv` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_comp` int NOT NULL,
`id_conv` int NOT NULL
);

CREATE TABLE `convenio` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`cod_area` varchar(255) NOT NULL,
`cod_uni` varchar(255) NOT NULL,
`cod_pais` varchar(255) NOT NULL,
`id_admon_out` int,
`id_curso_creacion` int NOT NULL,
`creado_por` int NOT NULL,
`num_becas_in` int NOT NULL,
`num_becas_out` int NOT NULL,
`meses_in` int NOT NULL,
`meses_out` int NOT NULL,

```

```

`anno_inicio` int NOT NULL,
`anno_fin` int NOT NULL,
`req_titulacion` varchar(255),
`req_curso` varchar(255),
`nominacion_online` boolean,
`link_nom_online` varchar(255),
`info_nom_online` text,
`link_documentacion` varchar(255),
`movilidad_pdi` boolean,
`movilidad_pas` boolean,
`tipo_movilidad` ENUM ('ERASMUS', 'ARQUUS', 'ERASMUS_DI',
→ 'ERASMUS_PARTNER', 'INTERCAMBIO', 'LIBRE_MOVILIDAD') NOT NULL,
`user_online` varchar(255),
`password_online` varchar(255),
`fecha_online` datetime,
`info_tor` text,
`observ_discapacidad` text,
`observ_req_ling` text,
`begin_nom_1s` datetime,
`end_nom_1s` datetime,
`begin_nom_2s` datetime,
`end_nom_2s` datetime,
`begin_app_1s` datetime,
`end_app_1s` datetime,
`begin_app_2s` datetime,
`end_app_2s` datetime,
`begin_mov_1s` datetime,
`end_mov_1s` datetime,
`begin_mov_2s` datetime,
`end_mov_2s` datetime,
`memo_grading` text,
`memo_visado` text,
`memo_seguro` text,
`memo_alojamiento` text,
`nombre_coord` varchar(255),
`cargo_coord` varchar(255),
`email_coord` varchar(255),
`tlf_coord` varchar(255),
`address_coord` varchar(255) COMMENT
→ 'Por defecto igual que la de la universidad',
`web_inf_acad` varchar(255),
`nombre_admon_in` varchar(255),
`cargo_admon_in` varchar(255),
`mail_admon_in` varchar(255),
`nombre_resp_acad_in` varchar(255),
`cargo_resp_acad_in` varchar(255),

```

```

`nombre_admon_out` varchar(255),
`cargo_admon_out` varchar(255),
`mail_admon_out` varchar(255),
`nombre_resp_acad_out` varchar(255),
`cargo_resp_acad_out` varchar(255),
`mail_resp_acad_out` varchar(255)
);

CREATE TABLE `acuerdo_estudios` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_estudiante` int NOT NULL,
`id_tutor` int NOT NULL,
`timestamp_creacion` datetime NOT NULL,
`periodo` ENUM ('PRIMERO', 'SEGUNDO', 'C_COMPLETO') NOT NULL,
`fase` int NOT NULL,
`id_curso` int NOT NULL,
`necesidades` text,
`begin_movilidad` date,
`end_movilidad` date,
`timestamp_nominacion` datetime,
`timestamp_registro` datetime NOT NULL,
`link_documentacion` varchar(255),
`n_solicitud_RRII` int,
`convocatoria` ENUM ('PRIMERA', 'SEGUNDA', 'EXTRAORDINARIA') NOT NULL
);

CREATE TABLE `ae_asigloc_asigext` (
`id_ae` int,
`asig_loc` int,
`asig_ext` int,
PRIMARY KEY (`id_ae`, `asig_loc`, `asig_ext`)
);

CREATE TABLE `renuncia` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_ae` int NOT NULL,
`descripcion` text NOT NULL,
`timestamp` datetime NOT NULL
);

CREATE TABLE `expediente` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_ae` int NOT NULL,
`id_tipo_exp` int NOT NULL
);

```

```

CREATE TABLE `tipo_expediente` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`descripcion` varchar(255) NOT NULL,
`tipo_estudiante` ENUM ('INCOMING', 'OUTGOING') NOT NULL
);

CREATE TABLE `fase_expediente` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_tipo_exp` int NOT NULL,
`descripcion` varchar(255) NOT NULL,
`fase_final` boolean DEFAULT 0
);

CREATE TABLE `envio_mail_fase` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_mail` int NOT NULL,
`id_fase` int NOT NULL,
`cargo` ENUM ('COORDINADOR', 'ADMON_IN', 'ADMON_OUT', 'RESP ADMON_OUT')
→ NOT NULL
);

CREATE TABLE `hist_envio_mail_fase` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_mail` int NOT NULL,
`id_fase` int NOT NULL,
`id_exp` int NOT NULL,
`email` varchar(255) NOT NULL
);

CREATE TABLE `hist_envio_mail_fase_mod` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`asunto` varchar(255),
`cuerpo` text NOT NULL,
`id_fase` int NOT NULL,
`id_exp` int NOT NULL,
`email` varchar(255) NOT NULL
);

CREATE TABLE `rel_exp_fase` (
`id` int PRIMARY KEY,
`id_exp` int NOT NULL,
`id_fase` int NOT NULL,
`id_gestor` int NOT NULL,
`procesado` boolean,
`timestamp` datetime NOT NULL,
`info` varchar(255)
);

```

```

);

CREATE TABLE `rel_exp_fav_gestor` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_exp` int,
`id_gestor` int
);

CREATE TABLE `evento` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`id_creador` int NOT NULL,
`titulo` varchar(255) NOT NULL,
`descripcion` text,
`estado` ENUM ('PENDIENTE', 'EN_PROCESO', 'TERMINADO') NOT NULL,
`prioridad` ENUM ('ALTA', 'MEDIA', 'BAJA') NOT NULL DEFAULT "MEDIA"
);

CREATE TABLE `tarea` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`descripcion` text NOT NULL,
`estado` ENUM ('PENDIENTE', 'EN_PROCESO', 'TERMINADO') NOT NULL DEFAULT
→ "PENDIENTE"
);

CREATE TABLE `deadline_aviso` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`fecha` datetime NOT NULL,
`id_responsable` int NOT NULL,
`id_evento` int NOT NULL,
`id_tarea` int NOT NULL
);

CREATE TABLE `recordatorio` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`timestamp` datetime NOT NULL,
`id_usuario` int NOT NULL,
`deadline` datetime NOT NULL,
`titulo` varchar(255) NOT NULL,
`descripcion` text,
`completado` boolean
);

CREATE TABLE `mensaje` (
`id` int PRIMARY KEY AUTO_INCREMENT,
`timestamp` datetime NOT NULL,
`id_emisor` int NOT NULL,

```

```
`id_receptor` int NOT NULL,  
`leido` boolean,  
`etiqueta` ENUM ('IMPORTANTE', 'ELIMINADO') NOT NULL,  
`asunto` varchar(255),  
`cuerpo` text NOT NULL  
);  
  
ALTER TABLE `titulacion` ADD FOREIGN KEY (`id_centro`) REFERENCES  
→ `centro` (`id`);  
  
ALTER TABLE `asignatura` ADD FOREIGN KEY (`id_tit`) REFERENCES  
→ `titulacion` (`id`);  
  
ALTER TABLE `asignatura_ext` ADD FOREIGN KEY (`id_curso`) REFERENCES  
→ `curso` (`id`);  
  
ALTER TABLE `asignatura_ext` ADD FOREIGN KEY (`id_conv`) REFERENCES  
→ `convenio` (`id`);  
  
ALTER TABLE `universidad` ADD FOREIGN KEY (`cod_pais`) REFERENCES `pais`  
→ (`iso`);  
  
ALTER TABLE `estudiante` ADD FOREIGN KEY (`id_titulacion`) REFERENCES  
→ `titulacion` (`id`);  
  
ALTER TABLE `estudiante` ADD FOREIGN KEY (`id_usuario`) REFERENCES  
→ `user` (`id`);  
  
ALTER TABLE `estudiante` ADD FOREIGN KEY (`id_convenio`) REFERENCES  
→ `convenio` (`id`);  
  
ALTER TABLE `rel_cl_est` ADD FOREIGN KEY (`id_cl`) REFERENCES  
→ `competencia_ling` (`id`);  
  
ALTER TABLE `rel_cl_est` ADD FOREIGN KEY (`id_est`) REFERENCES  
→ `estudiante` (`id_usuario`);  
  
ALTER TABLE `req_ling_conv` ADD FOREIGN KEY (`id_comp`) REFERENCES  
→ `competencia_ling` (`id`);  
  
ALTER TABLE `req_ling_conv` ADD FOREIGN KEY (`id_conv`) REFERENCES  
→ `convenio` (`id`);  
  
ALTER TABLE `convenio` ADD FOREIGN KEY (`id_curso_creacion`) REFERENCES  
→ `curso` (`id`);
```

```
ALTER TABLE `convenio` ADD FOREIGN KEY (`cod_pais`) REFERENCES `pais`
→  (`iso`);

ALTER TABLE `convenio` ADD FOREIGN KEY (`creado_por`) REFERENCES `user`
→  (`id`);

ALTER TABLE `convenio` ADD FOREIGN KEY (`cod_area`) REFERENCES `area`
→  (`cod_isced`);

ALTER TABLE `convenio` ADD FOREIGN KEY (`cod_pais`, `cod_uni`)
→  REFERENCES `universidad` (`cod_pais`, `cod_uni`);

ALTER TABLE `acuerdo_estudios` ADD FOREIGN KEY (`id_curso`) REFERENCES
→  `curso` (`id`);

ALTER TABLE `acuerdo_estudios` ADD FOREIGN KEY (`id_estudiante`)
→  REFERENCES `estudiante` (`id_usuario`);

ALTER TABLE `acuerdo_estudios` ADD FOREIGN KEY (`id_tutor`)
→  REFERENCES `user` (`id`);

ALTER TABLE `ae_asigloc_asigext` ADD FOREIGN KEY (`asig_loc`)
→  REFERENCES `asignatura` (`id`);

ALTER TABLE `ae_asigloc_asigext` ADD FOREIGN KEY (`asig_ext`)
→  REFERENCES `asignatura_ext` (`id`);

ALTER TABLE `renuncia` ADD FOREIGN KEY (`id_ae`)
→  REFERENCES `acuerdo_estudios` (`id`);

ALTER TABLE `expediente` ADD FOREIGN KEY (`id_ae`)
→  REFERENCES `acuerdo_estudios` (`id`);

ALTER TABLE `expediente` ADD FOREIGN KEY (`id_tipo_exp`)
→  REFERENCES `tipo_expediente` (`id`);

ALTER TABLE `fase_expediente` ADD FOREIGN KEY (`id_tipo_exp`)
→  REFERENCES `tipo_expediente` (`id`);

ALTER TABLE `envio_mail_fase` ADD FOREIGN KEY (`id_mail`)
→  REFERENCES `mail_predef` (`id`);

ALTER TABLE `envio_mail_fase` ADD FOREIGN KEY (`id_fase`)
→  REFERENCES `fase_expediente` (`id`);
```

```
ALTER TABLE `hist_envio_mail_fase` ADD FOREIGN KEY (`id_mail`)
→ REFERENCES `mail_predef` (`id`);

ALTER TABLE `hist_envio_mail_fase` ADD FOREIGN KEY (`id_fase`)
→ REFERENCES `fase_expediente` (`id`);

ALTER TABLE `hist_envio_mail_fase` ADD FOREIGN KEY (`id_exp`) REFERENCES
→ `expediente` (`id`);

ALTER TABLE `hist_envio_mail_fase_mod` ADD FOREIGN KEY (`id_fase`)
→ REFERENCES `fase_expediente` (`id`);

ALTER TABLE `hist_envio_mail_fase_mod` ADD FOREIGN KEY (`id_exp`)
→ REFERENCES `expediente` (`id`);

ALTER TABLE `rel_exp_fase` ADD FOREIGN KEY (`id_exp`) REFERENCES
→ `expediente` (`id`);

ALTER TABLE `rel_exp_fase` ADD FOREIGN KEY (`id_fase`) REFERENCES
→ `fase_expedient` (`id`);

ALTER TABLE `rel_exp_fase` ADD FOREIGN KEY (`id_gestor`) REFERENCES
→ `user` (`id`);

ALTER TABLE `rel_exp_fav_gestor` ADD FOREIGN KEY (`id_exp`) REFERENCES
→ `expediente` (`id`);

ALTER TABLE `rel_exp_fav_gestor` ADD FOREIGN KEY (`id_gestor`)
→ REFERENCES `user` (`id`);

ALTER TABLE `evento` ADD FOREIGN KEY (`id_creador`) REFERENCES `user`
→ (`id`);

ALTER TABLE `deadline_aviso` ADD FOREIGN KEY (`id_responsable`)
→ REFERENCES `user` (`id`);

ALTER TABLE `deadline_aviso` ADD FOREIGN KEY (`id_evento`) REFERENCES
→ `evento` (`id`);

ALTER TABLE `deadline_aviso` ADD FOREIGN KEY (`id_tarea`) REFERENCES
→ `tarea` (`id`);

ALTER TABLE `recordatorio` ADD FOREIGN KEY (`id_usuario`) REFERENCES
→ `user` (`id`);
```

```
ALTER TABLE `mensaje` ADD FOREIGN KEY (`id_emisor`) REFERENCES `user`  
    →  (`id`);
```

```
ALTER TABLE `mensaje` ADD FOREIGN KEY (`id_receptor`) REFERENCES `user`  
    →  (`id`);
```

Otra muy buena opción hubiera sido la de las migraciones. Las migraciones en Yii2 son unas clases especiales capaces de crear tablas en la base de datos. Lo mejor de todo ello es que una vez estén creados los modelos en la aplicación, podemos actualizar las migraciones si queremos modificar algún atributo o tabla, por lo que tan solo tendríamos que crear las nuevas migraciones pertinentes para que los cambios surtan efecto no solo en la base de datos, sino también en la clase del modelo, lo que permite que nos despreocupemos sobre si un cambio en la base de datos llevará a algún tipo de error en el código ya escrito.

6.3 DESARROLLO DEL PANEL DE CONTROL

El desarrollo comenzó por el panel de control con un claro propósito: el usuario tiene que poder cambiar la información con la que se trabaja desde la interfaz, sin tener que recurrir a la base de datos a cambiarla. Por ejemplo, imaginemos que un [Gestor de twinX](#) quiere añadir un nuevo convenio con un país con el cual nunca se ha concertado ninguno con anterioridad. Si no permitimos que se añada un nuevo país desde la interfaz, el usuario medio no tendría posibilidad de llevar a cabo esta acción sin la intervención del administrador de la base de datos que lo hiciera de forma manual.

Es cierto que, de algún modo, este tipo de tareas no se realizan a diario ni es el objetivo de este proyecto en sí, aunque sí lo es el de ofrecer una solución coherente. Además, del mismo modo en que sería el encargado de administrar una base de datos quien pudiera hacer esta labor en caso de ausencia del panel de control, tiene que ser el usuario con el rol de [Administrador](#) quien pueda acceder a este apartado de twinX que permita modificar y generar nueva información estática con la que poder trabajar.

Así pues, el desarrollo de esta parte se centra en el manejo básico de entidades esenciales como son: universidades, países, tipos de [expedientes de twinX](#), [eventos de expedientes de twinX](#), [mensajes predefinidos de twinX](#), titulaciones, centros y usuarios (desde donde actualizar los permisos de los distintos usuarios o incluso eliminarlos del sistema). Mayormente, las necesidades se cubrirán con la creación de tablas CRUD sin grandes modificaciones, ya que es un menú que no será frecuentemente utilizado por el personal de secretaría y no necesitan ver gran cantidad de información interrelacionada.

Antes de comenzar con el código, lo primero que haremos será configurar algunas herramientas, como son la región, el idioma, nombre de la aplicación, la utilización de

URLs limpias y el formato horario. Esto es común a toda la aplicación y, por tanto, se hace en el archivo en `common/config/main.php`.

```
<?php

return [
    'aliases' => [
        '@bower' => '@vendor/bower-asset',
        '@npm'    => '@vendor/npm-asset',
    ],

    'vendorPath' => dirname(dirname(__DIR__)) . '/vendor',

    'language' => 'es-ES',
    'timeZone' => 'Europe/Madrid',
    'components' => [
        'cache' => [
            'class' => 'yii\caching\FileCache',
        ],
        'urlManager' => [
            'enablePrettyUrl' => true,
            'showScriptName' => false,
        ],
        'rules' => [
            ],
    ],
    'formatter' => [
        'class' => 'yii\i18n\Formatter',
        'dateFormat' => 'dd/MM/yyyy',
        'datetimeFormat' => 'dd/MM/yyyy H:i:s',

        'timeFormat' => 'H:i:s',
        'locale' => 'es-ES',
        'defaultTimeZone' => 'Europe/Madrid',
    ],
    'name' => 'twinX',
]
```

Dentro del directorio del proyecto, encontramos las carpetas `backend` y `frontend`. En nuestro caso, al centrarnos en los dos primeros sprints, el segundo directorio no tiene

mucho sentido, pues nos permitiría servir otro sitio web que estaría controlado a través del *backend*. La verdadera utilidad del *frontend* vendría en los siguientes sprints cuando se implementasen historias de usuario relacionadas con la intervención de los estudiantes, tutores o coordinadores externos, pero nosotros nos centraremos en el *backend*.

También eliminamos el paquete de `bootstrap` con Composer e instalaremos `bootstrap4` en su lugar, dado que el que viene por defecto es la versión anterior y el actual permitirá dar un aire más actual a twinX. Es un proceso más tedioso de lo que parece, porque hay que sustituir todas las dependencias de `bootstrap` por `bootstrap4`, lo que nos llevará a más de un error inesperado al usar algún trozo de código ya programado que use la antigua librería.

Lo primero que hacemos es crear un módulo llamado «panel» desde el generador Gii. Esto crea el directorio `backend/modules/panel`, con los subdirectorios correspondientes. Entonces, con lo que acabamos de hacer, podemos englobar todas las funcionalidades que hemos comentado en un directorio denominado `panel`. Es decir, esto, junto a la utilización de URLs limpias, nos permitiría identificar las distintas localidades del módulo como por ejemplo `localhost/panel/pais`.

El primer menú que crearemos será el de usuarios. Al seguir los pasos que se especifican en el repositorio de GitHub [33], tenemos ya creada una clase de migración para una tabla llamada `user`. Hemos aprovechado la migración y hemos añadido nuevos atributos a la misma, para poder equipararlos con los necesarios en la tabla en nuestro modelo. Es importante usar esta misma tabla, con la migración por defecto, ya que contiene atributos como `password_hash` y otros derivados del motor genérico del sitio para la identificación de los usuarios que ya viene implementada por defecto, como son la recuperación de la contraseña, el envío de emails para la confirmación de la cuenta, etc. Entonces, al tener todos esos atributos, aparecerán en la tabla de usuarios en la vista. Por tanto, nuestra misión principal aquí y en la creación de las demás CRUD del panel es modificar los atributos que se muestran, no solamente en la vista de la tabla, sino también en la vista de detalle (donde se muestran todos los datos al completo) y en el formulario de creación / edición del registro.

Por ejemplo, en el siguiente código de `backend/modules/panel/user/index.php` hemos comentado los atributos innecesarios:

```
<?php

use yii\helpers\Html;
use yii\grid\GridView;
use yii\helpers\Url;

/* @var $this yii\web\View */
/* @var $dataProvider yii\data\ActiveDataProvider */
```

```
$this->title = 'Usuarios';
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="user-index">

    <h1><?= Html::encode($this->title) ?></h1>

    <?= GridView::widget([
        'dataProvider' => $dataProvider,
        'columns' => [
            'id',
            'username',
            // 'auth_key',
            // 'password_hash',
            // 'password_reset_token',
            'email:email',
            'status',
            'created_at:datetime',
            //'updated_at',
            //'verification_token',
            'nombre',
            'tipo_usuario',
            'telefono',
            'genero',
        ],
        [
            'class' => 'yii\grid\ActionColumn',
            'template' => '{view} {update} {delete}',
            'header' => 'Acciones'
        ]
    ])>
); ?>

</div>
```

Sin embargo, es importante denotar que no siempre esto será suficiente. Cuando tenemos relaciones muchos a muchos, no bastará con tan solo mostrar la clave externa a las dos relaciones que se unen, porque no le diría nada al usuario. Tendremos que generar atributos legibles y comprensibles por el usuario. Afortunadamente, esto no es nada difícil, puesto que por ejemplo, el modelo de universidad tiene asociado un objeto del tipo País. ¿Por qué? Sencillamente porque en su tabla tiene un atributo con una clave externa al país donde está la universidad. Yii detecta automáticamente esto y nos permite acceder al objeto en su totalidad para poder consultar otros atributos, ya que por ejemplo, podríamos necesitar mostrar el nombre de la universidad y el país en su conjunto. Esto es más frecuente en las vistas del módulo de gestión.

Una vez creada la primera vista de la aplicación, tenemos que atender a cuestiones de organización. De acuerdo con los bocetos, hemos de disponer de una cabecera donde acceder a las tres distintas grandes secciones de la aplicación y, por otro lado, un menú lateral cuyo contenido varíe en función de la sección en que nos encontramos. Para ello, tenemos que hablar de los *layouts*. Son el marco de la foto, el recubrimiento que tiene cada vista en la web y como tal, es heredado por cada una de los componentes de una sección. Por ejemplo, la cabecera de «gestión» tendrá siempre los mismos elementos que tendrá la cabecera del panel, pero no tendrán los mismos menús laterales. Para ello, en backend/views/layouts tenemos definidas la auth.php, que sirve para la pantalla de login, ya que carece de menú lateral, la base.php, que es la que tiene la cabecera y la sidebar_base.php, y luego, en cada módulo, se definen los *layouts* propios al mismo. En el caso del panel, en backend/modules/panel/views/layouts tenemos panel.php que hereda del ya mencionado base.php y que carga el _sidebar_panel.php, que es el menú lateral del panel, con sus elementos y no los de otro. Pues bien, en las siguientes secciones hemos tratado los distintos menús de la misma manera. La especificación de qué *layout* utilizar se hace en backend/config/main.php, donde también se añade cada uno de los módulos que conforman la aplicación.

De la implementación del *sidebar* de esta parte (el panel), destacamos el haber agregado un menú desplegable o *dropdown* (figura 41), que aunque podríamos haber usado el de Bootstrap 4, se ha decidido hacer por cuenta propia, ya que el predefinido no mantenía el menú persistente. Esto es algo bastante importante, porque orienta al usuario y le indica en todo momento dónde se encuentra.

Para conseguir el funcionamiento de *dropdown*, hemos escrito el siguiente código con jQuery (backend/web/js/dropdown_expediente_panel.js):

```
let submenusExpedientes = ['tipo-expediente', 'fase-expediente',
    ↵ 'envio-mail-fase'];
$(document).ready(function() {
    submenusExpedientes.forEach(function (str) {
        if (window.location.pathname.includes(str))
            toggleCollapse();
    });
});
```

Tras la creación del menú CRUD de los usuarios (el modelo ya está creado con anterioridad con la migración), se crean, por un lado, la clase de controlador, UserController.php, dentro de backend/modules/panel/controllers. Un nivel más arriba, en views, se tiene una nueva carpeta llama user, que posee los siguientes archivos:

- **_form.php**: es el formulario (ActiveForm de Yii, que renderiza un elemento form de HTML) que es utilizado por las vistas que necesiten cargar el formulario de datos de un usuario en la base de datos.

Menú

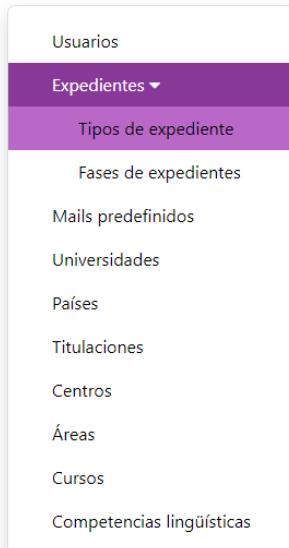


Figura 41: Menú lateral del panel en twinX

- **create.php:** es el receptor de la acción `create` del controlador, el cual renderiza el formulario `_form.php` para crear un usuario nuevo.
- **update.php:** también carga el mismo formulario, pero recibe la acción `update` del controlador, el cual envía la información que ya se tiene sobre un usuario en concreto.
- **view.php:** la vista de los datos del usuario en detalle.

Cada uno de estos archivos conforma la vista de un componente; en este caso, el de usuario.

Sobre los controladores, diremos que son los orquestadores de lo que se ve en pantalla, hasta el punto de que, de alguna manera, al escribir en la barra de direcciones `localhost/panel/user/view?id=3`, lo que ocurre es que se llama al método `actionView` de la clase `UserController`. Es decir, se llama al método `actionX` de la clase `yController`, donde claramente «`x`» es `view` en nuestro caso e «`y`», `User`. Con esto, y tras la atenta visualización de la ausencia, como se podría esperar, de un archivo `delete.php` en la vista, podemos imaginar que al ir a `localhost/panel/user/delete?id=3` se ejecutará el método `actionDelete`, que al igual que `actionView` acepta el parámetro `$id`. Pensemos, pues, que para eliminar un registro no necesitamos ninguna vista, pero sí un método en el controlador. Al fin y al cabo, ese método ordenará al modelo eliminar el registro, que es lo único que necesitábamos.

Los modelos son clases más sencillas, las cuales tienen la especificación de los atributos que tiene la tabla en concreto en base de datos, los métodos para recuperar los objetos de otros modelos a los que referencian (por ejemplo, como hemos dicho antes,

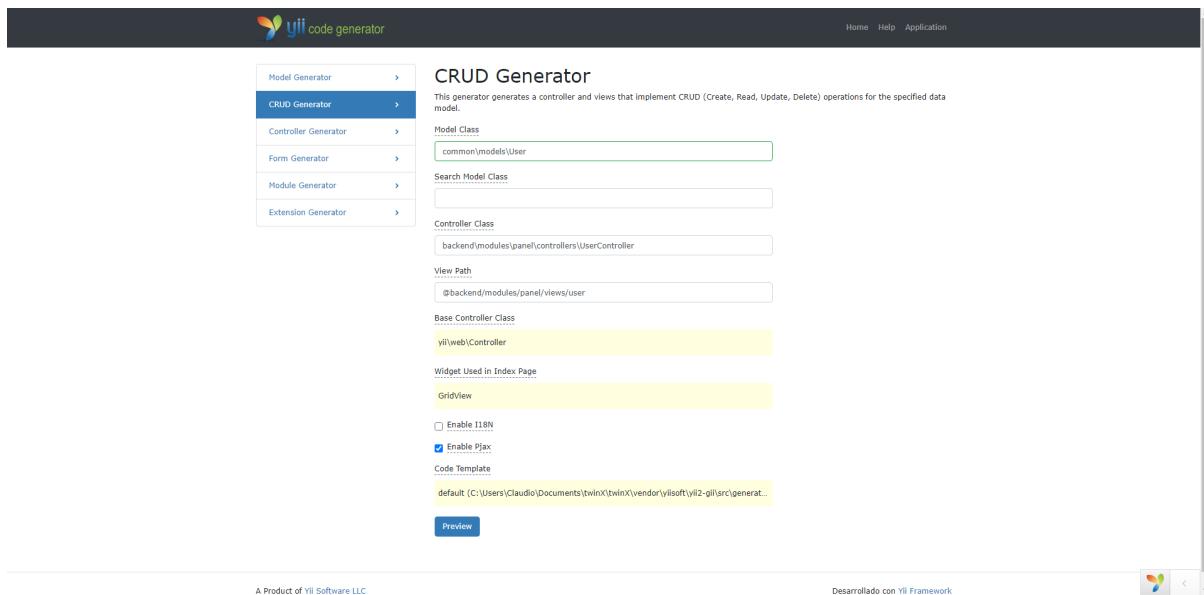


Figura 42: Generación de la tabla CRUD de User

una universidad pertenece a un país y, por tanto, el objeto del modelo de País estará dentro de Universidad), y otros métodos que generalmente creemos nosotros para acceder a datos externos de manera más rápida.

Una vez hemos hecho un recorrido por los tres grandes tipos de clases que tenemos, modelos, vistas y controladores, podemos entonces comprender el significado de nuestro patrón arquitectónico, el significado y la influencia que tiene en twinX y lo coherente que es el establecer esta separación.

Volviendo al desarrollo, tras crear con Gii la tabla CRUD (figura 42), el resultado es el menú de la figura 43. Normalmente, las tablas CRUD suelen tener un botón verde que permiten crear un nuevo registro, pero en el caso de los usuarios, se ha eliminado, ya que al crearlo manualmente desde este menú, no se genera un hash para la contraseña, al mismo tiempo que tampoco tiene gran relevancia el poder crear usuarios desde el panel de control, cuando el principal interés está puesto en gestionar los permisos de los integrantes de la comunidad.

Como ya hemos señalado con anterioridad, se tomó la decisión de mantener las vistas de los registros almacenados sin apenas modificar en este módulo de panel de control. Por defecto, lo que podemos ver en la interfaz de vista de un registro es lo que encontramos en la figura 44.

Por defecto, Gii pone los botones justo a la izquierda de las cabeceras de las tablas y de las vistas. Sin embargo, hemos tomado la decisión de modificar esto, como se puede apreciar en la figura 44, donde los botones de «editar» y «eliminar» están a la derecha. El motivo es porque mayormente en las tablas del menú index de cada categoría, como la de la figura 45 se podía ver un exceso de información de haber dejado el botón en la izquierda, pues tendríamos el título, el indicador del total de elementos de la

The screenshot shows the twinX application's user management interface. At the top, there is a navigation bar with links for 'Gestión', 'Calendario', and 'Panel de control'. On the right side of the header, there are notification icons (with a red '3' badge) and a 'Salir (supersu)' button.

Menú

Usuarios

- Expedientes ▾
- Mails predefinidos
- Universidades
- Países
- Titulaciones
- Centros
- Áreas
- Cursos
- Competencias lingüísticas

Usuarios

Mostrando 1-6 de 6 elementos.

ID	Nombre de usuario	Correo electrónico	Estado	Fecha de registro	Nombre	Tipo de usuario	Teléfono	Género	Acciones
2	supersu	supersu@twinx.com	10	22/10/2020 11:	Claudio López	SUPERUSUARIO	+34123456789	M	
8	chelo	chelo@twinx.com	10	11/11/2020 19:	Chelo	GESTOR	123	F	
9	msanz	msanz@twinx.com	10	11/11/2020 19:	Miguel Ángel	GESTOR	123	M	
10	augTutor	augusto@twinx.com	10	11/11/2020 19:	Augusto Carvajal	TUTÓR	123	O	
11	luci	lucia@twinx.com	9	11/11/2020 19:	Lucía Álvarez	ESTUDIANTE	123	F	
12	johnsm	johnsm@twinx.com	9	11/11/2020 19:	John Smith	ESTUDIANTE	123	M	

Figura 43: Menú de usuarios en twinX

The screenshot shows the twinX application's degree management interface. At the top, there is a navigation bar with links for 'Gestión', 'Calendario', and 'Panel de control'. On the right side of the header, there are notification icons (with a red '3' badge) and a 'Salir (supersu)' button.

Menú

- Usuarios
- Expedientes ▾
- Mails predefinidos
- Universidades
- Países
- Titulaciones**
- Centros
- Áreas
- Cursos
- Competencias lingüísticas

Grado en Ingeniería Informática

ID	296
Nombre	Grado en Ingeniería Informática
Centro	E.T.S. Ingenierías Informática y de la Telecomunicación

[Editar](#) [Eliminar](#)

Figura 44: Vista de un registro en twinX (menú de titulaciones)

Mostrando 1-5 de 5 elementos.					
#	ID	Descripción	Tipo de expediente	Fase final	Acciones
1	1	Aceptado y enviado a destino	Modificación acuerdo de estudios [O]	<input checked="" type="checkbox"/>	
2	2	Reclamo TOR a estudiante	Reconocimiento OUT [O]	<input checked="" type="checkbox"/>	
3	4	Comienzo de nominación IN	Nominación IN [I]	<input checked="" type="checkbox"/>	
4	5	Aceptado Acuerdo y Subido a SEDE	Modificación acuerdo de estudios [O]	<input checked="" type="checkbox"/>	
5	6	Finalización Nominación	Nominación IN [I]	<input checked="" type="checkbox"/>	

Figura 45: Menú de fases de expedientes en twinX

tabla y el botón. Por tanto, hemos dejado todos los botones a la izquierda en las sucesivas vistas de los menús de toda la plataforma. Por supuesto, no hemos hecho los cambios de forma manual, sino que se ha modificado el código del generador de Gii para que todas las genere automáticamente como describimos. Esto es posible modificando el archivo `index.php` para la tabla del menú y el `view.php` para la vista de un registro en el directorio `vendor/yiisoft/yii2-gii/src/generators/crud/default/views`³. Sobre el primero de los archivos, diremos que también hemos hecho las pertinentes modificaciones para que cada una de las vistas tenga el botón del ojo (acceso a la vista en detalle) el lápiz (para editar la entrada) y la papelera (para la eliminación) para interaccionar con cada registro en concreto.

Por último, vamos a destacar la vista de una fase de expediente. Recordemos que el procesar una fase implicaba enviar varios correos a responsables y a destinatarios relacionados con la situación del estudiante (o incluso a él mismo). Por tanto, hemos incluido en la vista de las fases, los mensajes que se enviarían tras procesarla (figura 46). También podemos verlos pulsando en el botón amarillo con el sobre de la figura 45, donde se dan aún más opciones al usuario. La implementación de esta parte no es compleja pero sí tediosa, por lo que vamos a omitir su explicación por el momento, dado que en la próxima sección veremos otro ejemplo, ya que en el módulo de gestión 6.4 hemos diseñado una vista compuesta de forma similar.

³ Este directorio no se encuentra en el repositorio de GitHub [6], dado que forma parte de los archivos del motor de Yii y se encuentran dentro del archivo `.gitignore`, por lo que no se guardan, dado que pueden obtenerse clonando el repositorio de Yii [33]

Fase #1

ID	1
Tipo de expediente	Modificación acuerdo de estudios
Descripción	Aceptado y enviado a destino
Fase final	x

Envío de mails en fase

Mail predefinido	Receptor(a)
Ejemplo de mail predefinido	COORDINADOR

Figura 46: Vista de una fase de expediente en twinX

6.4 DESARROLLO DEL MÓDULO DE GESTIÓN

Para desarrollar esta parte de la aplicación, también se ha creado un módulo y las tablas CRUD pertinentes para cada una de las secciones de Gestión. Recordamos que es el corazón de la aplicación, donde los gestores acudirían diariamente a desempeñar su trabajo: consulta de la información referente a un estudiante, tramitación de algún expediente, cambios en su acuerdo de estudios, etc.

6.4.1 Integración del menú de convenios

Uno de los mayores desafíos ha sido, como era de imaginar a la hora de ver el modelo de la base de datos (figura 38), el diseñar un formulario para los convenios recogido y no puesto tal cual sale de Gii. Para ello hemos empleado el objeto de Bootstrap card y accordion. Conforme vamos desplegando cada una de las secciones, las otras se cierran para no ocupar espacio en la pantalla y hacer que el usuario se centre en la introducción de los datos en una sección concreta. También hemos usado tarjetas a distintos niveles para estructurar los formularios de la mejor manera posible (figuras 47 y 48)

Por defecto, Gii espera que en campos donde en la tabla se tiene una restricción de clave externa se introduzca, mediante un campo de texto libre, un identificador, por ejemplo, del país del convenio. Sin embargo, esta es una de los numerosos cambios que se comienzan haciendo al recibir el código generado: adaptarlo al usuario. Para ello y durante todo el proyecto en general, hemos hecho uso de librerías del autor Kartik [28] y como resultado, tenemos elementos que ya están disponibles en Yii2 como es la inserción de un menú dropdown para escoger entre varias opciones, pero poten-

Nuevo convenio

Menú

- Dashboard
- Convenios**
- Estudiantes
- Acuerdos de estudios
- Expedientes
- Tutores

Identificación

Becas

Número de becas IN	[+]	Número de becas OUT	[+]
Meses IN	[+]	Meses OUT	[+]

Plazos

Personal de administración

Nominaciones online Nominación online

Requisitos

Anotaciones

Guardar

Figura 47: Creación de un convenio en twinX

Incoming

Responsable en administración incoming	Responsable académico incoming
Nombre	Nombre
Cargo	Cargo
Correo electrónico	

Outgoing

Responsable en administración outgoing	Responsable académico outgoing
Nombre	Nombre
Cargo	Cargo
Correo electrónico	

Figura 48: Creación de un convenio en twinX: introducción de datos de los responsables externos

Figura 49: Creación de un convenio en twinX: introducción de datos externos al modelo Convenio

ciado con una búsqueda, por ejemplo, como se aprecia en la figura 49. Pensemos que elementos como este son indispensables, pues en el uso cotidiano, twinX almacenaría cientos de estudiantes, decenas de universidades y convenios tal y como hace TWINS hasta el momento; por tanto, la opción de un desplegable estático no funcionaría en este contexto. Su uso es verdaderamente sencillo, una vez que nos familiarizamos con la sintaxis:

```
<?php
( . . . )
<?= $form->field($model, 'cod_pais')->widget(Select2::className(), [
    'data' => ArrayHelper::map(Pais::find()->all(), 'iso', 'nombreISO'),
    'theme' => Select2::THEME_KRAJEE_BS4,
    'options' => [
        'placeholder' => 'Seleccione un país',
    ]
]) ?>
```

Otra dificultad de la implementación del formulario de los convenios estaba en la selección de los requisitos lingüísticos. Éstos se almacenan en una tabla aparte, pero existe una relación muchos a muchos que relaciona un convenio con varias competencias o la pertenencia de una sola competencia a muchos convenios. El problema estaba cuando queríamos asociar un par (`id_convenio, id_requisito`) sin tener de antemano la ID del convenio que se está creando. Para ello ha sido necesario crear una **transacción**. Es decir, primeramente tenemos que asegurarnos de que guardamos todos los datos del convenio. Acto seguido, asociar los requisitos lingüísticos que se

han escogido para el convenio a crear y, después, guardar los cambios. Si alguna de estas acciones no prospera, tendremos que cancelar todo y devolver la tabla a su estado anterior. Esto podemos conseguirlo sobrecargando el método de guardado, `save`. Es más, también hemos de tener en cuenta que el tratamiento de los requisitos puede darse en la edición del convenio, con lo cual, puede haber requisitos ya en la tabla que tenemos que recuperar, y a la hora de ejecutar la transacción, evaluar cuáles estaban ya en la tabla de la relación convenio - requisito y dejarlos intactos, cuáles han de ser eliminados y cuáles insertados como nuevos registros. Para ello, tenemos que crear una nueva clase `ConvenioForm` que `eConvenio` (modelo) y herede sus métodos. Entonces, programamos toda esta funcionalidad que hemos descrito y solo cuando sea necesario, hacemos la llamada a este nuevo modelo. Una de las características más novedosas es la sobrecarga del método `afterFind`, que ejecuta justo después de buscar la recuperación de los requisitos actuales del convenio (si los tiene). Gracias a ello, podemos trabajar con un array de requisitos y ejecutar las acciones descritas:

```
<?php  
(...)  
  
class ConvenioForm extends Convenio  
{  
    public $requisitos = [];  
    private $_requisitos;  
  
    public function save($runValidation = true, $attributeNames = null)  
    {  
        $transaction = \Yii::$app->db->beginTransaction();  
  
        try {  
            if (!parent::save($runValidation, $attributeNames)) {  
                return false;  
            }  
  
            $this->addNewRequisitos();  
            $this->deleteOldRequisitos();  
  
            $transaction->commit();  
        }  
  
        catch (Exception $e) {  
            $transaction->rollBack();  
  
            throw $e;  
        }  
    }  
}
```

```
        return true;
    }

protected function addNewRequisitos()
{
    $nuevos = [];

    if ($this->isNewRecord) {
        $nuevos = $this->requisitos;
    }
    else if(!empty($this->requisitos)){
        foreach ($this->requisitos as $requisito) {
            if (!in_array($requisito, $this->_requisitos)) {
                $nuevos[] = $requisito;
            }
        }
    }

    if(!empty($nuevos)) {
        foreach ($nuevos as $requisito) {
            $relacion = new ReqLingConv();

            $relacion->id_conv = $this->id;
            $relacion->id_comp = $requisito;

            if (!$relacion->save()) {
                throw new Exception('Error al guardar el requisito');
            }
        }
    }
}

protected function deleteOldRequisitos()
{
    foreach ($this->reqLingConvs as $requisito) {
        if (!in_array($requisito->id, $this->requisitos) && $requisito->delete)
            throw new Exception('Failed to save related records.');
    }
}

public function afterFind()
{
    foreach ($this->reqLingConvs as $requisito) {
        $this->requisitos[] = $requisito->id;
    }
}
```

```

    $this->_requisitos = $this->requisitos;

    parent::afterFind();

}

}

```

El proceso llevado a cabo en `EstudianteForm` es muy similar, dado que también necesitamos, en el formulario de un estudiante, especificar las competencias lingüísticas que posee.

En el menú de convenios (figura 50) hemos tenido que cambiar bastantes cosas. Para ello, ha sido necesario crear bastantes métodos en el modelo de Convenio. Por ejemplo, para obtener el ratio de estudiantes nominados del total de estudiantes que contiene el convenio. O los convenios con su código completo (código ISO del país, código de la universidad y área de conocimiento según el ISCED⁴). Del mismo modo, hemos tenido que ajustar la visualización de los nombres de la universidad o del área completos, ya que la generación de código automática solo mostraba los ID en la base de datos:

```

<?php
(...)

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        [
            'attribute' => 'numAcuerdos',
            'label' => 'Nº de acuerdos'
        ],
        [
            'attribute' => 'nominadosAcuerdos',
            'label' => 'Nominados totales',
            'format' => 'raw'
        ],
        [
            'attribute' => 'codConvenio',
            'label' => 'Convenio',
            'format' => 'raw',
        ],
        [
            'attribute' => 'nombreCodUni',

```

⁴ International Standard Classification of Education

Figura 50: Menú de convenios en twinX

```

'label' => 'Universidad'
],
[
    'attribute' => 'areaCompleta',
    'label' => 'Nombre de la área'
],

'tipo_movilidad',

[
    'class' => 'yii\grid\ActionColumn',
    'template' => '{view}',
    'header' => 'Acciones'
],
],
]);
?>

```

Todos los atributos que vemos en el código son generados por métodos creados en el modelo:

```

<?php
(...)

public function getNombreCodUni()
{
    return $this->codUni->getNombreCodigo();
}

```

```
public function getAreaCompleta()
{
    return $this->codArea->getAreaCompleta();
}
```

También podríamos haber hecho ese acceso desde la vista, pero aunque no es una buena práctica, nos complicaría mucho las cosas. El tener los atributos definidos en el modelo, nos permite acceder a ellos desde una clase especial de la que aún no hemos hablado. Son las clases de búsqueda, y se encuentran en `common/models/search`, nombradas como `ConvenioSearch` en caso del convenio. Es ahí donde tenemos que decir qué atributos propios hemos definido en nuestro modelo, para poder unir las tablas a las que refieren y así poder hacer tanto búsquedas como ordenaciones, introduciendo términos en los campos de texto de debajo de la cabecera de la tabla y haciendo click en los nombres de los atributos respectivamente. Para conseguir este comportamiento, ha sido necesario hacer en primer lugar, los atributos seguros (`safe`):

```
<?php
(...)
class ConvenioSearch extends Convenio
{
    public $codConvenio, $nombreCodUni, $areaCompleta;

    /**
     * {@inheritDoc}
     */
    public function rules()
    {
        return [
            (...),
            [['codConvenio', 'nombreCodUni', 'areaCompleta'], 'safe'],
        ];
    }

    ...
}
```

Después, hacer la unión de la tabla con las demás y configurar las ordenaciones:

```
<?php
(...
public function search($params)
{
    $query = Convenio::find();
```

```

////
$query->joinWith(['codPais', 'codArea', 'codUni']);

// add conditions that should always apply here

$dataProvider = new ActiveDataProvider([
    'query' => $query,
]);

////
$dataProvider->sort->attributes['codConvenio'] = [
    'asc' => ['pais.iso' => SORT_ASC],
    'desc' => ['pais.iso' => SORT_DESC],
    'default' => SORT_DESC
];

$dataProvider->sort->attributes['nombreCodUni'] = [
    'asc' => ['universidad.nombre' => SORT_ASC],
    'desc' => ['universidad.nombre' => SORT_DESC],
    'default' => SORT_DESC
];

$dataProvider->sort->attributes['areaCompleta'] = [
    'asc' => ['area.cod_isced' => SORT_ASC],
    'desc' => ['area.cod_isced' => SORT_DESC],
    'default' => SORT_DESC
];

(...)

}

(...)

?>

```

Y por último, añadir el filtrado a las búsquedas que se hagan introduciendo texto:

```

<?php
(...)

->andFilterWhere(['like', 'pais.iso', $this->codConvenio])
->orFilterWhere(['like', 'area.cod_isced', $this->codConvenio])
->orFilterWhere(['like', 'universidad.cod_uni', $this->codConvenio])
->andWhere(['like', 'universidad.nombre', $this->nombreCodUni])
->orWhere(['like', 'universidad.cod_uni', $this->nombreCodUni])
->andWhere(['like', 'area.cod_isced', $this->areaCompleta])

```

```
->orFilterWhere(['like', 'area.nombre_isced', $this->areaCompleta])
->orFilterWhere(['like', 'area.nombre_area', $this->areaCompleta]);

(...)

?>
```

Esto es extensible a todas las clases donde hemos usado atributos modificados y Yii no conoce la manera de aplicar los filtros, por lo que es obligatorio hacerlo cuando queremos ofrecer otra manera de comunicar la información distinta a la que ofrece el framework por defecto. En muchos casos esto se complica bastante. Por ejemplo, en la unión de las tablas:

```
<?php
(...)

$query->joinWith(['ae.estudiante.usuario', 'ae.estudiante.convenio.codPais',
    'tipoExp', 'relExpFases.fase', 'ae.estudiante.convenio.codArea']);

?>
```

Este es el código de la clase `ExpedienteSearch.php`, dadas las numerosas referencias a otras clases externas a ella.

Una vez comentadas las vistas de creación, actualización y de menú principal, vamos a abordar la de vista (figura 51). Los datos se han separado en distintas secciones de nuevo, pero sin el comportamiento de acordeón que impedía abrir más de una sección al mismo tiempo. Para la vista se debe dar más libertad al usuario, y siempre puede contraer las secciones a su gusto; en cambio, para el formulario, es más importante forzar centrar su atención en una parte de los datos en concreto.

Destaca también el testigo de convenio activo, que indica si la fecha de vigencia del convenio es superior a la actual (convenio activo), si es inferior (inactivo) o si iguala al año (próxima caducidad)

6.4.2 Integración del menú de estudiantes

Respecto de los convenios, una de las cosas más destacable ya ha sido comentada: hemos necesitado ejecutar otra transacción para elegir las competencias lingüísticas de los estudiantes en su formulario (figura 52) y para ello hemos creado una clase `EstudianteForm` que extiende a `Estudiante`.

Desde el menú de estudiantes (figura 53) podemos pivotar hacia su convenio (ícono verde) o hacia su acuerdo de estudios (ícono turquesa). Al lado del nombre del estudiante, figura un círculo de color rojo o azul, según si es saliente o entrante, respec-

Figura 51: Vista de los convenios en twinX

Figura 52: Creación de un estudiante en twinX

Nombre	Convenio	Titulación	Username	Email	DNI	Acciones
Lucía Álvarez	FRANKFU01/09.0	Grado en Estudios Ingleses (123)	luci	lucia@twinx.com	123456799N	
John Smith	BRNO01/09.0	Grado en literaturas comparadas (121)	johnsm	johnsm@twinx.com	A1658X	

Figura 53: Menú de estudiantes en twinX

tivamente, como apuesta de rediseño de llenar el fondo del nombre del estudiante de uno de estos colores en TWINS según su modalidad (figura 10).

La vista del estudiante (54) también ha tenido un importante rediseño. Se ha vuelto a tener en cuenta la necesidad de tener accesos directos a demás información del usuario y respecto del boceto que creamos para esta vista (figura 34) destaca el haber prescindido del excesivo uso de iconos que se hizo en el wireframe, algo bastante positivo, ya que no se alcanzaba a entender bien en todos los casos a qué dato correspondía cada campo.

Llegados a este punto, señalamos la generación automática de la nota de competencia lingüística y de participación de un estudiante. En TWINS, ambas puntuaciones tenían que guardarse en dos atributos por separado, pero se ha creído conveniente desarrollar un algoritmo para el cálculo de las notas. De acuerdo con las exigencias del convenio, si un estudiante tiene la misma competencia que se le requiere, no se le atribuye una nota extra. Sin embargo, a partir de B1 (que es lo mínimo que se puede requerir), se puede premiar con 1 punto adicional si el estudiante tiene un nivel B2, 1.5 puntos si es C1 o 2 puntos si su título es de C2, independientemente del nivel que se parta; esto es, si el estudiante tiene un C1 pero le requieren un B2 o B1, se le suman 1.5 puntos extra, y 2 puntos en caso de poseer un nivel C2 pero le requirieran C1 o inferior.

Entonces, en el modelo de Estudiante, el algoritmo es el siguiente:

```
<?php
(...)
public function getNotaCompetenciaLing()
{
```

The screenshot shows the twinX application interface. At the top, there's a navigation bar with links for 'Gestión', 'Calendario', 'Panel de control', and 'Salir (supersu)'. On the left, a sidebar menu titled 'Menú' includes 'Dashboard', 'Convenios', 'Estudiantes' (which is selected and highlighted in purple), 'Acuerdos de estudios', 'Expedientes', and 'Tutores'. The main content area displays a student profile for 'John Smith'. It features a circular placeholder for a profile picture. Below the name, there are several tabs: 'Mail' (with 3 notifications), 'Acuerdos de estudios' (selected and highlighted in green), 'Editar' (Edit), and 'Eliminar' (Delete). A large table is present, divided into two sections: 'Datos personales' (Personal Data) and 'Movilidad' (Mobility). The 'Datos personales' section contains fields like 'Username' (johnsm), 'Género' (Masculino), 'DNI' (A1658X), 'Email' (johnsm@twinx.com), 'Fecha de nacimiento' (31/10/1997), and 'Fecha de registro' (11/11/2020 19:15). The 'Movilidad' section includes 'Convenio' (CZ BRNO01/09.0), 'Competencias lingüísticas' (INGLÉS (C1)), 'Titulación' (Grado en literaturas comparadas (121)), 'Cesión de datos' (Becario MEC), 'Nota del expediente' (6.9), 'Nota de competencias lingüísticas' (1.5), and 'Nota de participación' (8.4). There are also 'x' buttons next to some mobility-related fields.

Figura 54: Vista de estudiante en twinX

```

$compConvenioRaw = $this->convenio->reqLingConvs;
$compEstudianteRaw = $this->relClEsts;
$compConvenio = [];
$compEstudiante = [];
$extra = 0;
$valores = [
    'B1' => 0.5,
    'B2' => 1,
    'C1' => 1.5,
    'C2' => 2
];

foreach ($compConvenioRaw as $cC) {
    $compConvenio []= $cC->comp;
}

foreach ($compEstudianteRaw as $cE) {
    $compEstudiante []= $cE->cl;
}

foreach ($compEstudiante as $cE){
    if(in_array($cE, $compConvenio))
        unset($compConvenio[$cE->id]); // No tenemos en cuenta las
        ↳ competencias que requiere el convenio
}

foreach ($compEstudiante as $cE){
    foreach ($compConvenio as $cC){

```

```

        if($cE->lengua == $cC->lengua){
            if($valores[$cE->nivel] > $valores[$cC->nivel])
                $extra += $valores[$cE->nivel];
        }
    }

    return $extra;
}

(...)

?>

```

Entonces, el valor devuelto por el método es la nota de competencia lingüística que se muestra en la figura 54 y la nota de participación, la suma de ésta con la del expediente, ya es con esa nota con la que participa en el sorteo de las plazas de movilidad. Con este algoritmo, ahorramos un total de dos atributos en la tabla de estudiante, y teniendo en cuenta que la cantidad de registros a almacenar en el día a día es elevado, podemos considerar esta como una mejora significativa.

6.4.3 Integración del menú de expedientes

Sin pasar por el menú de acuerdos de estudios debida la carencia de novedades respecto a todo lo ya expuesto, que se repite a lo largo del desarrollo, entramos de lleno en los expedientes. Si bien ni el menú ni el formulario presentan características excesivamente novedosas, sí podemos encontrar un gran cambio en su vista (figura 55).

Un expediente puede ser el de la nominación de un estudiante *incoming*, y en todo momento necesitamos conocer en qué parte del proceso se encuentra su nominación. Recordemos que esto se posibilitaba con las [eventos de expedientes de twinX](#), y que cada una de ellas implicaba un envío de correos electrónicos en su [procesamiento](#). Pues bien, al igual que en TWINS, lo más oportuno es ver el expediente del estudiante y su situación actual, con un historial de todo lo ya ocurrido en dicho registro. Esto lo hemos hecho posible embebiendo un menú en la interfaz de la vista del expediente; en este caso, aunque pueda pensarse que es el de la tabla `fase_expediente`, en realidad se trata del de `rel_fase_exp` (relación de fase con expediente), donde se guarda información de la fase con respecto del expediente. De este último, aclaremos que en su tabla en la base de datos no se almacena otra información que la relación de un acuerdo y un tipo de expediente.

En este caso, aunque utilicemos la librería de JavaScript **AJAX** (integrada en Yii como **Pjax**), necesitamos crear vistas compuestas; esto es, vistas que mantengan la información del expediente en la parte de arriba mientras que en la parte de abajo se pueda estar renderizando el formulario de una fase o el menú que muestra todas las fases

ID	6
Nombre	John Smith
Convenio	CZ BRNO01/09.0
Titulación	Grado en literaturas comparadas (121)
Competencias lingüísticas	INGLÉS (C1)
Tipo de expediente	Nominación IN [I]

Fase	Gestor	Procesado	Fecha de modificación	Información	Acciones
Comienzo de nominación IN	Chelo (chelo)	X	14/11/2020 13:05:46 (hace un día)		
Finalización Nominación	Chelo (chelo)	✓	14/11/2020 12:54:20 (hace un día)		

Figura 55: Vista de expedientes en twinX

de un expediente. Estas dos últimas son las únicas posibilidades de variación de la interfaz que comentamos, pero volviendo al *modus operandi* de Yii, recordemos que crear y actualizar son dos acciones distintas a ojos del controlador, aunque ambas usen el mismo formulario. Por tanto, necesitaremos los siguientes elementos:

- **_view.php**: donde almacenaremos la vista compartida y persistente de Expediente.
- **view.php**: la vista predeterminada con la información del expediente y sus fases.
- **view-new-fase.php**: carga la vista del expediente y el formulario de creación de una nueva fase.
- **view-update-fase.php**: carga la vista del expediente y el formulario de modificación de una fase ya existente.

Dentro de las tres últimas, llamamos a una instancia del controlador

`RelFaseExpController.php` desde el que se llama al menú, al creador y al actualizador respectivamente, en sus métodos `action` correspondientes:

```
<?php include('_view.php') ?>

<?= $relExpFase->actionCreate($model) ?>

<?php $this->title = 'Expediente #' . $model->id; ?>
```

Dado que en las tres vistas se precisa de una instancia del controlador, es en `_view.php` donde se sitúa la creación de la misma:

```
<?php
( . . . )
```

```
$relExpFase = new
    \backend\modules\gestion\controllers\RelExpFaseController('rel-exp-fase',
    Yii::$app->getModule('rel-exp-fase'));

(...)

?>
```

Entonces, dado que es al controlador de Expediente al que se llama para cargar las distintas vistas compuestas (ya que nos situamos en `localhost/gestion/expediente`), es él mismo quien tiene que orquestar las llamadas a los métodos `action` para cargarlas, por lo que tenemos que crear acciones nuevas que llamen a los archivos de vista nuevos. A su vez y en aras de reutilizar código, éstos métodos harán una llamada a `actionView`, que es el mismo método que carga la vista del expediente, pero con nuevos parámetros, para que podamos enlazar el flujo de control con `RelExpFaseController.php`.

Una vez llegada la ejecución a uno de los archivos que permiten crear o modificar una fase para el expediente, se llama al método `actionCreate` o `actionUpdate` de `RelExpFase.php`, pero con una novedad: se pasa como argumento el Expediente actual, que por tanto el formulario ya tiene que llevar relleno en el campo destinado a ello. Lo mismo pasa con la actualización. En el caso de la vista de la lista de las fases de un expediente (archivo por defecto `view.php`), también se necesita el ID del expediente que se visualiza, para devolver tan solo aquellas fases pertenecientes al mismo. Esto se consigue modificando el método `actionIndex` de `RelExpFaseController.php` de la siguiente forma:

```
<?php
(...)

public function actionIndex($idExpediente = false)
{
    $path = '';
    $query = RelExpFase::find();

    if($idExpediente) {
        $path = '@backend/modules/gestion/views/rel-exp-fase/';
        $query = $query->where(['id_exp' => $idExpediente]);
    }

    $dataProvider = new ActiveDataProvider([
        'query' => $query,
    ]);
}
```

```
return $this->render($path . 'index', [
    'dataProvider' => $dataProvider,
    'idExpediente' => $idExpediente,
]);
}

(...)

?>
```

Los métodos `actionCreate` y `actionUpdate` tienen una implementación similar. Nótese que no hemos impedido el funcionamiento corriente de los métodos, sino que los hemos complementado con atributos que poseen valores por defecto porque no siempre tienen que requerirse, en caso de necesitar hacer uso de la interfaz de forma independiente a la vista de expedientes como aquí se necesita. Toda esta estructura se tiene también para la visualización de los correos (`MailPredefinido`) que se envían en cada fase en el módulo del panel de control, cuya implementación es similar y por ello, no la hemos tratado.

6.4.4 Integración del dashboard

De nuevo, poco hay que decir sobre la implementación del menú de tutores, y por ello, pasamos a una de las grandes novedades que supone twinX sobre TWINS: la vista de dashboard (figura 56). Su objetivo no es otro que el presentar la información de mayor relevancia al usuario en cuanto abra la aplicación. Esta facilidad se ofrecía en TWINS en la forma de tres menús que bloqueaban toda la pantalla, ofreciendo información sobre el calendario, los avisos (mensajes en nuestro caso) y expedientes sin procesar, obligando al usuario a salir de las tres pantallas hasta llegar al menú principal. De esta manera, la aparición del dashboard no es bloqueante, y con tan solo una vista se resumen todos los asuntos que precisan de la atención del gestor al arrancar twinX.

En este caso, no ha sido necesaria la creación de ningún menú CRUD ni ningún modelo nuevo. Simplemente hemos creado la vista `dashboard.php` en su directorio (`backend/modules/gestion/views/dashboard`) y en la carpeta de controladores, el `DashboardController.php`, poseedor de tan solo un método `actionIndex` que carga el archivo de la vista.

Dejando aparte las cuestiones de estilo, la información puede mostrarse a través de consultas a los demás modelos ya creados. Como mejora adicional, aunque aún no hayamos hablado de los mensajes y los recordatorios, vamos a presentar la última clase especial que tiene un gran potencial en Yii y de la cual hemos hecho uso en la implementación del dashboard. Son las clases **Query**. En ellas, cualquier método que se cree, puede ser ejecutado a modo de QBE a la hora de hacer una consulta a la base de datos, de modo que podamos ahorrar y reutilizar código. Por ejemplo, este es el método `notificaciones` de `RecordatorioQuery.php`:

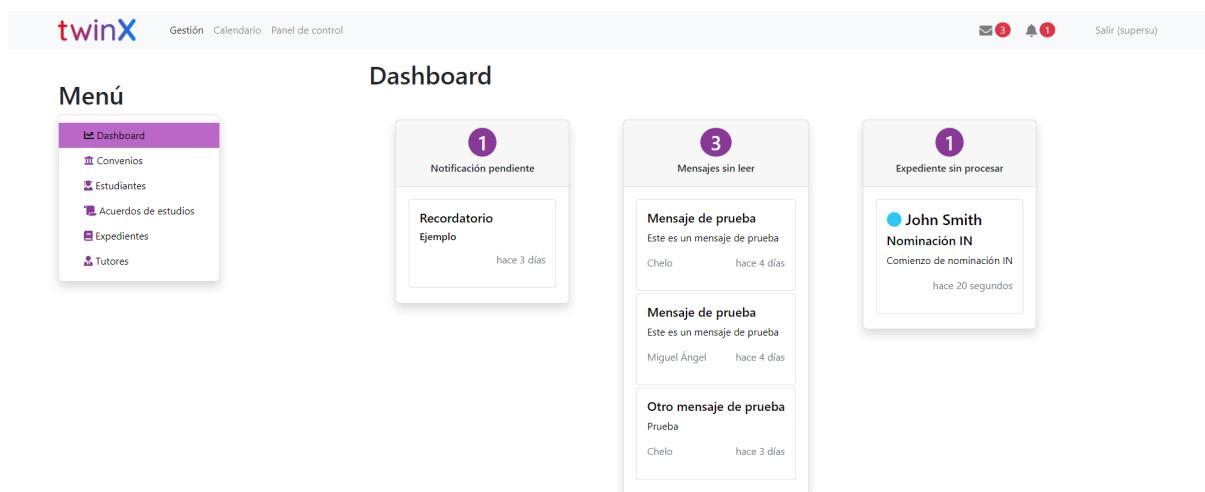


Figura 56: Dashboard de twinX

```
<?php
(...)

public function notificaciones($db = null)
{
    return Recordatorio::find()
        ->where(['id_usr_aviso' => Yii::$app->user->id])
        ->andWhere(['<>', 'completado', '1'])
        ->andWhere(['<', 'deadline', date('yy-m-d H:i')]);
}

(...)
?>
```

Es sencillo, pero dado que en el dashboard necesitamos trabajar con las notificaciones pendientes en dos ocasiones (extraer el número de recordatorios pendientes y su vista de detalle en la sección de notificaciones), es muy práctico tenerlo para llamarlo de la siguiente manera:

```
<?php
(...)

$notificaciones = \common\models\Recordatorio::find()->notificaciones();

(...)

foreach ($notificaciones->with('emisor')->all() as $notificacion) {
```

```

echo '<div class="card p-3 notificacion-dashboard mb-1">';
$output = '<h5>Recordatorio</h5>';
$output .= '<h6>' . $notificacion->titulo . '</h6> ';
$output .= '<p>' . $notificacion->descripcion . '</p>';
$output .= '<p class="text-muted d-flex flex-row-reverse">' .
    Yii::$app->formatter->asRelativeTime($notificacion->timestamp)
    . '</p>';
echo \yii\helpers\Html::a($output,
    ['/calendario/recordatorio/view', 'id' =>
    $notificacion->id]);
echo '</div>';
}

(...)

?>

```

Es decir, en \$notificaciones almacenamos la *query*, y la usamos llamando al método count() para obtener el número de registros o el array con todas ellas (all()) si necesitamos trabajar con cada una de ellas por separado.

En última instancia, es interesante también considerar la mejora que presenta la introducción de secuencias width(<atributo>). Al especificar el atributo del modelo de la clase para la que se hace la consulta, se está evitando a toda costa lo que se conoce como el fenómeno de la «carga perezosa» o *lazy loading*. Por ejemplo, para la carga de los expedientes sin procesar utilizamos varias sentencias with que nos permiten almacenar el contenido del atributo que se especifica, evitando posteriormente el tener que hacer consultas adicionales:

```

<?php
foreach ($expedientes->with('exp')->with('exp.ae')->with('fase')->all()
    as $expediente) {
    echo '<div class="card p-3 notificacion-dashboard mb-1">';
    $output = '<h4>' .
        $expediente->exp->ae->estudiante->nombreEstudiante . '<h4>';
    $output .= '<h5>' . $expediente->exp->tipoExp->descripcion .
        '</h5> ';
    $output .= '<p>' . $expediente->fase->descripcion . '</p>';
    $output .= '<p class="text-muted d-flex flex-row-reverse">' .
        Yii::$app->formatter->asRelativeTime($expediente->timestamp)
        . '</p>';
    echo \yii\helpers\Html::a($output, ['/gestion/expediente/view',
        'id' => $expediente->id_exp]);
    echo '</div>';
}
?>

```

Es decir, como tenemos que recuperar atributos del acuerdo de estudios (ae) de exp y la fase, podemos ordenar que se almacenen los valores para no tener que recuperarlos de nuevo. Este comportamiento recibe el nombre de «carga entusiasta» o *eager loading*. Más específicamente, sin ninguna orden *with* en toda la vista del dashboard, tendríamos un total de 45 *queries*, mientras que con todas ellas, rebajamos el número hasta 38.

6.4.5 Integración de la mensajería

Por último en este módulo, hemos implementado la mensajería de twinX (figura 57). En lugar de tener un *actionIndex* en *MensajeController.php* general para la lista de mensajes, se ha sustituido por un *actionRoute*, que diverge la vista del menú en bandeja de entrada o elementos: enviados y carga los mensajes que mostrar en función de si se es emisor o receptor de los mismos:

```
<?php
(...)
public function actionRoute($bandeja)
{
    $searchModel = new MensajeSearch();
    $query = $searchModel->searchQuery(Yii::$app->request->queryParams);

    if($bandeja == 'INBOX')
        $query->andWhere(['id_receptor' => Yii::$app->user->id]);
    else
        $query->andWhere(['id_emisor' => Yii::$app->user->id]);

    $dataProvider = $searchModel->search($query);

    return $this->render('index', [
        'searchModel' => $searchModel,
        'dataProvider' => $dataProvider,
        'bandeja' => $bandeja
    ]);
}
(...)
?>
```

De la vista de menú, se ha creído conveniente crear un pequeño atajo para marcar un mensaje como leído o no leído. También se ha tratado de simplificar la vista (figura 58) condensando los atributos de valor de un mensaje en su cabecera y su asunto como título del contenido. Destaca también la ausencia de opciones como la edición y eliminación de un mensaje ya enviado, para mantener el significado de la entidad *Mensaje* y no tratarla como una tupla más de una tabla en base de datos.

Figura 57: Bandeja de entrada en twinX

Figura 58: Vista de mensaje en twinX

6.5 DESARROLLO DEL MÓDULO DE CALENDARIO

La parte del calendario en TWINS podía resultar un poco confusa al tratar de entender cómo funciona: un calendario con eventos comunes a todo el mundo. Estos eventos con un máximo de dos personas a las que poder avisar, tenían la capacidad de encontrar asociadas numerosas tareas que dividían el evento en distintas etapas. Las tareas, a su vez, contaban con otro máximo de dos personas a las que poder asignar esa tarea y con una fecha límite concreta. Tenían características incluso como la atribución de un porcentaje de progresión en la completitud de la tarea o también el evento (figura 4).

En el boceto de la figura 37 representamos una aproximación a lo que podría ser el calendario en twinX, pero aunque quizás no tan complicado de implementar, su desarrollo necesita bastante tiempo, ya que en twinX, de acuerdo con el modelo de la base de datos (figura 38), se tenía una relación muchos a muchos para poder avisar a cuantos gestores se quisiera para un evento o una tarea creados. Esto complica la implementación, haciéndola menos inmediata e inteligible, pues pensemos que tendríamos que desempeñar de nuevo las mismas acciones que para generar vistas compuestas; en este caso, varias tablas CRUD embebidas, unas dentro de otras.

Ya hemos demostrado que es posible hacer este tipo de direccionamiento de los flujos de información para obtener los resultados que se quieren, pero por motivos de tiempo, se ha decidido cambiar la implementación de esta parte por otra, dentro del módulo del calendario, ya que no afecta a la consecución de los objetivos generales del proyecto. Más concretamente, al comienzo del segundo sprint 3.6.2 se tomó la decisión de dejar fuera de la programación la HU3.1 y la HU3.2. En consecuencia, se pudo llevar a cabo el desarrollo de la HU3.3 y la HU4 con mayor margen temporal y con la posibilidad de establecer un mayor enfoque en los resultados y la calidad de los mismos con respecto, como siempre, a los objetivos generales del proyecto.

Sobre la codificación de esta parte, las tareas a realizar eran similares: creación de un módulo, generación del modelo de Recordatorio y del menú CRUD para el mismo. Hemos, sin embargo, filtrado la aparición de recordatorios en el menú principal (figura 59) a tan solo los que crea el usuario. Cabe destacar que un gestor puede dirigir recordatorios a otro compañero, por lo que le aparecerán en su menú. Sin embargo, si otro usuario crea un recordatorio para nosotros, no lo veremos ahí, sino en la sección de notificaciones, que tiene también un testigo rojo a modo de contador en la cabecera de la web para que nos demos cuenta siempre que tengamos alguno pendiente. También aparecen en el dashboard, por lo que al abrir la aplicación no solo veríamos que tendríamos un recordatorio pendiente, sino que también podríamos apreciar de qué se trata. Si hacemos clic en él, nos redirigirá a la vista de detalle, muy similar a la de los mensajes. Una vez se marcan como completados, desaparecerán de notificaciones.

Para las notificaciones no nos ha hecho falta crear otro modelo, tan solo una vista y un controlador sencillos, al igual que para el dashboard, ya que las consultas que se ha-

Título	Usuario a avisar	Fecha y hora límite	Completado	Acciones
Ejemplo	Miguel Ángel (msanz)	12/11/2020 13:50:00	x	

Figura 59: Menú de recordatorios en twinX

cen a la base de datos se realizan sobre instancias de modelos ya creados. Retomemos la utilidad de haber construido el método en `RecordatorioQuery.php` para recuperar todos los recordatorios pendientes para explicar el funcionamiento de `actionIndex` en `NotificacionesController.php`; ahorraremos aún más código:

```
<?php
(...)

class NotificacionesController extends \yii\web\Controller
{
    public function actionIndex()
    {
        $dataProvider = new ActiveDataProvider([
            'query' => Recordatorio::find()->notificaciones()
        ]);

        return $this->render('index', [
            'recordatorios' => $dataProvider
        ]);
    }
}

?>
```

A modo de resumen, podemos elaborar el diagrama de paquetes de la figura 60 para ver la disposición de los archivos del proyecto en sus paquetes y directorios y la interacción entre los mismos. Se ha omitido el contenido de cada uno de los

subdirectorios de `views` por motivos de extensión del diagrama, al igual que de los directorios `query` y `search`, que no aportan nombre de clases nuevas.

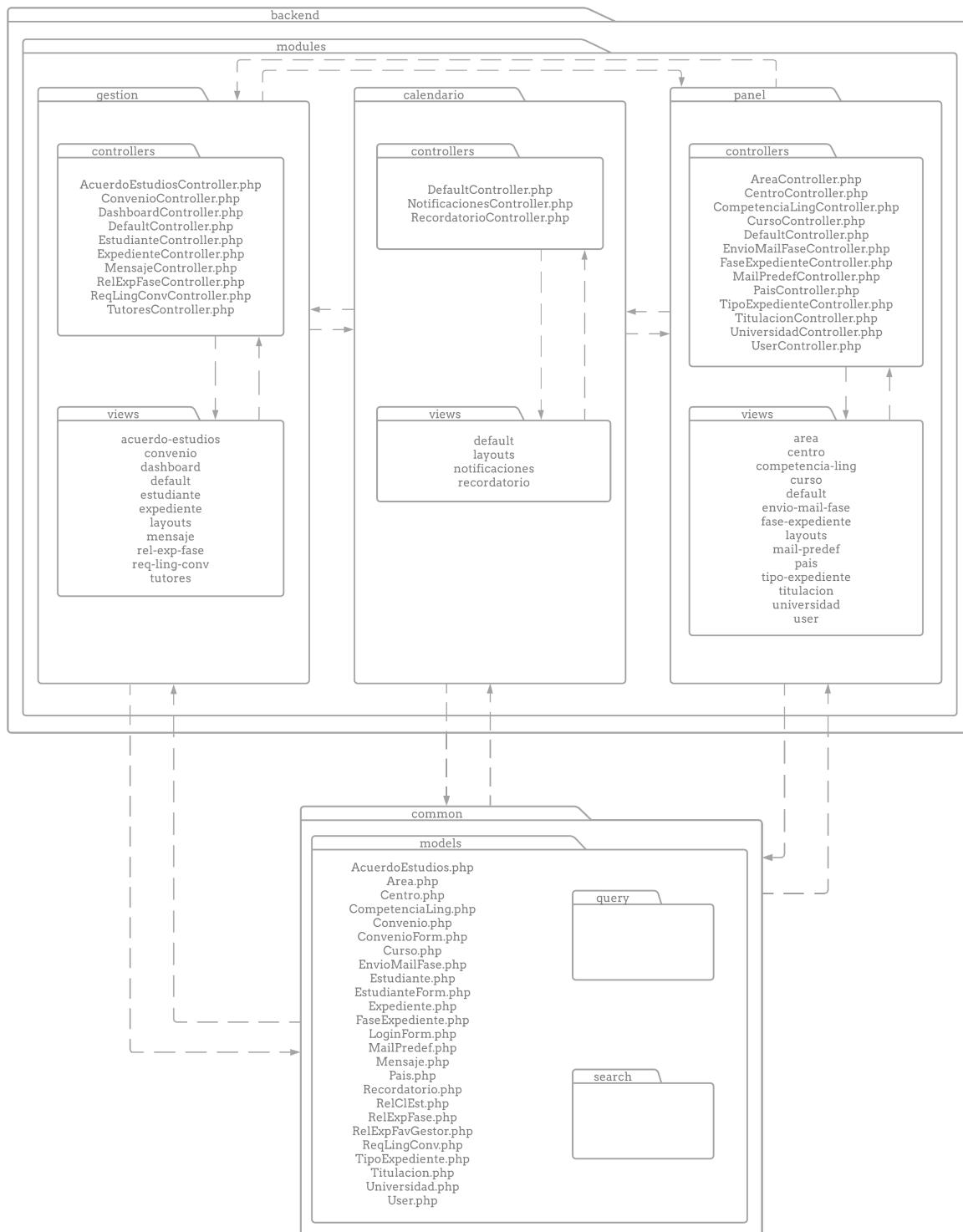


Figura 60: Diagrama de paquetes de twinX

BIBLIOGRAFÍA

- [1] Nils Adermann. *Composer*. URL: <https://www.getcomposer.org/>.
- [2] Nick Babich. "Sitemaps & Information Architecture (IA)". En: *Xd Ideas* (dic. de 2019). URL: <https://xd.adobe.com/ideas/process/information-architecture/sitemap-and-information-architecture/>.
- [3] A. Bangor. "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale". En: *Journal of Usability Studies* 4.3 (2009), págs. 114-123.
- [4] Bootstrap. *Bootstrap · The most popular HTML, CSS, and JS library in the world.Bootstrap · The most popular HTML, CSS, and JS library in the world.* URL: <https://getbootstrap.com/>.
- [5] Bert Bos. *Cascading Style Sheets*. URL: <https://www.w3.org/Style/CSS/>.
- [6] Claudio López Carrascosa. *claudiolc/twinX*. 2020. URL: <https://github.com/claudiolc/twinX>.
- [7] Digital.ai. *State of Agile Report | State of Agile*. 2020. URL: <https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494>.
- [8] Torgeir Dingsøyr y col. *A decade of agile methodologies: Towards explaining agile software development*. 2012.
- [9] Elastic. ¿Qué es Elasticsearch? URL: <https://www.elastic.co/es/what-is/elasticsearch>.
- [10] Google Fonts. *Quicksand - Google Fonts*. URL: <https://fonts.google.com/specimen/Quicksand>.
- [11] Interaction Design Foundation. *Test your prototypes: how to gather feedback and maximize learning*. URL: <https://www.interaction-design.org/literature/article/test-your-prototypes-how-to-gather-feedback-and-maximise-learning>.
- [12] Interaction Design Foundation. *What is Design Thinking? | Interaction Design Foundation*. URL: <https://www.interaction-design.org/literature/topics/design-thinking>.
- [13] The jQuery Foundation. *jQuery*. URL: <https://jquery.com/>.
- [14] Oficina de Gestión de la Comunicación. Unidad de Documentación Edición e Información. *Memoria Académica. Anexo estadístico*. 2019. URL: <https://canal.ugr.es/wp-content/uploads/2019/09/UGR-EN-CIFRAS-18-19-web-1.pdf>.
- [15] JetBrains. *PhpStorm: el IDE rápido e inteligente para programación en PHP de JetBrains*. 2000-2020. URL: <https://www.jetbrains.com/es-es/phpstorm/>.

- [16] Glenn E. Krasner. "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80". En: *JOOP* (1988). URL: <https://www.lri.fr/~mbl/ENS/FONDIHM/2013/papers/Krasner-JOOP88.pdf>.
- [17] Decanato de Filosofía y Letras. *Memoria de Gestión 2019. Facultad de Filosofía y Letras.* 2019. URL: [https://filosofiayletras.ugr.es/pages/facultad/normativa/memorias-de-gestion/memoriagestion2019/!](https://filosofiayletras.ugr.es/pages/facultad/normativa/memorias-de-gestion/memoriagestion2019/).
- [18] MDN. *JavaScript* | MDN. URL: <https://developer.mozilla.org/es/docs/Web/JavaScript>.
- [19] Microsoft. *Segoe UI Font Family - Typography* | Microsoft Docs. URL: <https://docs.microsoft.com/es-es/typography/font-list/segoe-ui>.
- [20] Kai Petersen, Claes Wohlin y Dejan Baca. "The Waterfall Model in Large-Scale Development". En: *Lecture Notes in Business Information Processing Product-Focused Software Process Improvement* (2009), 386–400. DOI: [10.1007/978-3-642-02152-7_29](https://doi.org/10.1007/978-3-642-02152-7_29).
- [21] Mario Román. *templates/classicthesis-tex-es at master · mroman42/templates*. 2018. URL: <https://github.com/mroman42/templates/tree/master/classicthesis-tex-es>.
- [22] K. Schwaber. *SCRUM Development Process*. 1997. URL: https://rd.springer.com/chapter/10.1007/978-1-4471-0947-1_11.
- [23] U.S. General Services Administration | Technology Transformation Services. *Usability.gov | Improving the User Experience*. URL: <http://www.usability.gov>.
- [24] Holistics Software. *DBML - Database Markup Language* | DBML. URL: <https://www.dbml.org/home/#intro>.
- [25] Holistics Software. *dbdiagram.io - Database Relationship Diagrams Design Tool*. 2020. URL: <https://dbdiagram.io/home>.
- [26] Lucid Software. *Online Diagram Software & Visual Solution* | Lucidchart. 2020. URL: www.lucidchart.com.
- [27] UXPressia®. *Customer & User Journey Map (CJM)* | Customer Experience Mapping Tool - UXPressia. URL: <https://uxpressia.com>.
- [28] Kartik Vilweswaran. *Krajee Yii Extensions - © Kartik*. URL: <https://demos.krajee.com/>.
- [29] W3C. *Web Content Accessibility Guidelines (WCAG) Overview*. Ed. por Shawn Lawton Henry. Jul. de 2005. URL: <https://www.w3.org/WAI/standards-guidelines/wcag/>.
- [30] W3C. *World Wide Web Consortium (W3C)*. URL: <https://www.w3.org/>.
- [31] Chauncey Wilson. *Method 18 of 100: The User/Task Matrix*. En: *100 User Experience (UX) Design and Evaluation Methods for Your Toolkit*. Oct. de 2011. URL: <https://dux.typepad.com/dux/2011/10/method-18-of-100-the-usertask-matrix.html>.

- [32] Yii. *Yii PHP Framework*. 2008 - 2020. URL: <https://www.yiiframework.com/>.
- [33] Yiisoft. *yiisoft/yii2-app-advanced: Yii 2.0 Advanced Application Template*. 2020. URL: <https://github.com/yiisoft/yii2-app-advanced>.

GLOSARIO

Transcript of Records Documento expedido por la universidad de destino tras la finalización de la movilidad de un estudiante en el que se establece la lista de asignaturas en el destino realizadas por el estudiante en cuestión y la calificación obtenida en las mismas. [1](#)

Acuerdo de estudios Documento de validez legal que recoge, previa movilidad, la relación de asignaturas que el estudiante va a realizar en su universidad de destino junto a la equivalencia con las asignaturas existentes en su facultad para su grado. Es el documento marco acordado entre el estudiante y su **Tutor(a) académico/a** académico para poder posteriormente realizar el reconocimiento de créditos del estudiante, junto con el *Transcript of Records*. [1](#), [12](#)

Administrador Usuario con el mayor número de permisos en twinX. Puede gestionar las piezas de información básicas y necesarias para trabajar con los estudiantes, como las universidades o los países. [1](#)

Alteración de matrícula Proceso mediante el cual se conceden una serie de asignaturas -comúnmente a **estudiantes Incoming**- mediante períodos de adjudicación donde se tienen en cuenta diferentes baremos en los que basar las distintas asignaciones. [1](#), [12](#), [16](#), [20](#)

Consentimiento de cesión de datos Expresión por parte del estudiante de su voluntad en que sus datos, tales como nombre y correo electrónico, sean facilitados a otros futuros interesados en hacer un programa de movilidad en el mismo destino. [1](#), [13](#)

Convenio Documento en el que se establece el acuerdo de dos universidades en países distintos de acoger a un número de estudiantes de una universidad a cambio de que la otra acepte al mismo número de estudiantes. Entre la información que figura en él se encuentra un identificador que suele contener alguna(s) letras que permiten identificar al país, seguido de uno o varios números, las fechas de relevancia para nominar estudiantes en el destino, la persona encargada de garantizar la validez del convenio y sus datos de contacto, los requisitos lingüísticos que se han de reunir para ser aceptado en dicha universidad, el número de meses que durará la movilidad, etc. [1](#), [12](#)

Estudiante Incoming Estudiante extranjero que viene de acogida a la universidad local o de origen, como también se la conoce. También se les atribuye el nombre de estudiantes entrantes. [1](#), [12](#)

Evento de expedientes de TWINS Representa un estado concreto en que se encuentra el expediente de TWINS de un estudiante. Sobre éste último, se entiende que

está realizando un programa de movilidad o tiene voluntad de hacerlo. [1](#), [21](#), [23](#)

Expediente de TWINS En la aplicación de TWINS, representa un acontecimiento que incumbe a un estudiante, como por ejemplo, la modificación de su acuerdo de estudios. Dado que es un registro de suma importancia, se divide en [eventos de expedientes de TWINS](#), los cuales van acumulándose y conformando el expediente durante el tránscurso del mismo. Todo expediente está iniciado y finalizado por un [Evento de expedientes de TWINS](#).

La movilidad de un estudiante podría concebirse como una línea temporal, la cual está compuesta por unos puntos que la dividen; entonces, los expedientes serían estos puntos. Del mismo modo, la línea estaría comenzada y finalizada por dos expedientes también. [1](#), [19](#), [21](#), [23](#)

Expediente de twinX La definición de [Expediente de TWINS](#) es aplicable a este contexto, salvo que ahora los [eventos de expedientes de TWINS](#) reciben el nombre de [Fase de expediente](#) para evitar confusiones con un evento del calendario [1](#)

Fase de expediente La definición de [Evento de expedientes de TWINS](#) es aplicable a este contexto, tan solo se ha cambiado el nombre de la entidad para evitar confusiones con los eventos del calendario. [1](#)

Gestión de twinX Módulo de la aplicación que contiene las acciones que llevan a cabo los [gestores de twinX](#): consulta y modificación de expedientes, estudiantes y convenios, actualización, creación y borrado de esa información, etc. [1](#)

Gestor de twinX Módulo de la aplicación que permite añadir información a bajo nivel (al [Administrador](#)), como las universidades, los países o los emails predefinidos. [1](#)

Nominación Proceso por el cual se da a conocer a la universidad de destino el/la estudiante o los estudiantes que han sido elegidos por la universidad de origen para realizar su movilidad de acuerdo con el convenio que previamente se ha establecido entre las mismas. Puede requerirse por las universidades de destino a modo de formulario online en alguna plataforma de éstas, por lo que ha de estar explícito en el convenio, así como las credenciales que pueden ser necesarias para loguearse en el sitio como prueba de veracidad de los datos que se les hacen llegar. [1](#), [12](#), [20](#)

Panel de control de twinX Módulo de la aplicación que permite añadir información a bajo nivel (al [Administrador](#)), como las universidades, los países o los emails predefinidos. [1](#)

Reconocimiento de créditos Proceso administrativo realizado por el personal de secretaría en el cual se toma la relación establecida en el Acuerdo de Estudios y el [Transcript of Records](#) para constar efectivamente las asignaturas que ha superado el estudiante y hacerlo así constar en su expediente académico. Dependiendo del destino, se puede precisar de una carta de equivalencias que

permite identificar cuál es la calificación más exacta dada la obtenida en el destino, de modo que pueda ser adaptado al sistema de calificaciones de la universidad de origen. [1](#)

Socio Nombre genérico usado por el personal de secretaría para referirse a cualquier universidad extranjera con la que se tenga un [Convenio](#). [1](#), [12](#)

Tutor(a) académico/a Figura de responsabilidad para el estudiante de acuerdo con su destino que se encarga de verificar las distintas propuestas de [Acuerdos de estudios](#) que el estudiante realiza hasta que finalmente se confecciona la versión final y es aceptada por su tutor(a). [1](#), [12](#)

ANEXOS

REUNIÓN DE TOMA DE CONTACTO

18 de febrero de 2020

Duración: 45 minutos

Facultad de Filosofía y Letras, Campus de Cartuja – Universidad de Granada

Chelo Pérez presenta al resto del personal que trabaja en la Oficina de Relaciones Internacionales de la Facultad de Filosofía y Letras, a quienes pone a disposición para que se les haga consultas de cualquier tipo que tengan relación con el problema, pues todos conocen las dificultades a las que se han de enfrentar cada día. A continuación, Chelo presenta a Miguel Ángel Sanz, quien trabaja en la secretaría de la facultad. El interés en conocerlo se basa en que es quien ha diseñado una herramienta que resuelve parcialmente algunos de los problemas a los que se tienen que enfrentar a diario en la oficina mediante la creación de una base de datos en Access®.

A pesar de su interés en la reutilización de esta herramienta que tras un año de esfuerzos está posibilitando gestionar de mejor manera la información en la oficina, se les comunica que el proyecto a construir empezaría desde cero, pudiendo así adaptarlo a otras tecnologías que facilitarían sus labores diarias considerablemente, más aún de lo que la actual solución lo hace. Es más, cabe destacar la imposibilidad de utilizar un software como es Microsoft Office® en un proyecto como este, dada la privatización de la licencia y las restricciones que éste pueda poner al desarrollo, que aún no se conocen, pero siempre serán menores si no se tiene fijada una herramienta sobre la cual tenga que radicar el resto del sistema.

REUNIÓN DE PREPARACIÓN DEL PROYECTO

17 de marzo de 2020

Duración: 34 minutos

A través de Google Meet

Participan Chelo Pérez Ocaña, Claudio López Carrascosa y Rosana Montes Soldado

La reunión da comienzo comentando lo hablado hasta ahora, tomando las partes esenciales del problema como punto de partida: gestión y almacenamiento de convenios, estudiantes y sus expedientes (asuntos a tratar con la secretaría). A partir de estos temas centrales, surgen aclaraciones y matices a discutir. Algunos ya se habían hablado, pero se repetían para que Rosana pudiera tomar nota y comprender el problema mejor. Otros, aunque habían sido comentados, se resalta la importancia de los mismos y se describe la forma en que se da respuesta en TWINS.

Otra cosa bastante destacable es cómo Chelo describe que al principio, el usar TWINS era un verdadero desafío, pues había muchas cosas que no funcionaban y que no habían sido del todo trasladadas a la aplicación de Access, para lo que tuvo que colaborar de forma estrecha con Miguel Ángel Sanz, su creador.

Sobre TWINS se comentan las cosas que se han llegado a conseguir tras los esfuerzos hechos, como una buena gestión de los convenios con otras universidades, los eventos que dispara en cuanto a envío de correos electrónicos automatizado –unos 2000 al día–. No menos importante es, la forma en que todas estas operaciones quedan registradas para su posterior volcado en un historial, donde se almacenan debidamente todos los cambios que se ha ido haciendo a cualquier información almacenada en la base de datos.

En general, Chelo resalta la gran ventaja que todo ello supone, pues por ejemplo, cuando antes necesitaban dedicar alrededor de un mes al proceso de nominaciones de estudiantes, con TWINS solo precisan de una hora (y porque se va ejecutando el proceso de envío de emails independiente a cada universidad). Es más, también se resaltan mejoras como, por ejemplo, la de poner formularios a disposición de los estudiantes e integrar la información en la aplicación directamente; justo lo que han tenido que buscar para procesar la información de todos los estudiantes de movilidad una vez comenzó la pandemia por la COVID-19.

Finalmente, pasamos a tratar el objetivo de mi Trabajo Fin de Grado. Con varias ideas sobre la mesa, finalmente se habla de una conversión de TWINS a la web, puesto que sería mucho más útil en cuanto a extensibilidad a otras facultades y; es más, una vez se haga el desarrollo web (con las mejoras que ya ello tiene de partida), sería más fácil integrar más servicios y añadir características. Para la realización, aparte del compromiso de Chelo para con su disponibilidad y colaboración con el trabajo, comenta la posibilidad de ceder acceso a TWINS para poder examinarlo bien y hacer un análisis completo, algo que resulta esencial para poder llevar a cabo el proyecto.

REUNIÓN DE PRESENTACIÓN DE TWINS

7 de abril de 2020

Duración: 1 hora y 56 minutos

A través de Google Meet

Participan Claudio López Carrascosa, Miguel Ángel Sanz Sáez y Rosana Montes Soldado

Comienza la reunión. Miguel Ángel hace uso de la palabra para hacer una pequeña presentación de TWINS. Nos explica que su nombre vino dado tras el encargo a sus hijas gemelas del diseño del logo, quienes tras explicarles el significado de los colores rojo (estudiantes salientes) y azul (estudiantes entrantes) en el ámbito de la oficina, dieron lugar a lo que es hoy por hoy el logotipo de la aplicación. Está compuesto por la palabra «gemelas» en lengua inglesa, rotulada con los dos colores. De ambos extremos de la palabra salen dos lazos que se unen en una especie de corazón morado, simbolizando la unión entre dos universidades.

Acto seguido, nos comenta los objetivos de TWINS y por qué decidieron comenzar este proyecto: para manejar grandes cantidades de información, para la comunicación entre distintos actores y para llevar un correcto seguimiento de los casos de los estudiantes.

También destaca aspectos de la aplicación, como el filtro de búsqueda, de suma importancia para poder localizar con eficacia a los estudiantes y/o documentos que se precisen; algo tan sencillo que ya, sin más, adelanta mucho trabajo.

Terminada la presentación, comienza a hacer una demostración en directo de TWINS. Nos habla de su funcionamiento, su estructura, y nos describe todos los elementos que lo componen y que ha mencionado en la presentación.

Tras aproximadamente dos horas de presentación, concluye la reunión, tras haber ido por muchos de los detalles de la aplicación (aunque no todos, por lo que probablemente se precise de más reuniones). Con todo ello, puede hacerse un análisis y así arrancar el proyecto de twinX.

REUNIÓN DE ORIENTACIÓN PARA LA INSTALACIÓN DE TWINS

18 de abril de 2020

Duración: 18 minutos

A través de Google Meet

Participan Claudio López Carrascosa y Miguel Ángel Sanz Sáez

En esta reunión se dieron las instrucciones necesarias para la instalación de TWINS, a partir de los dos archivos necesarios para ello: por un lado, el de la aplicación; por otro, el de los datos a importar por la primera.

Todo ello fue indicado por Miguel Ángel tras haber recibido firmado por parte de Rosana (tutora) y Claudio (estudiante) el correspondiente compromiso de confidencialidad para con la información de carácter sensible que alberga TWINS, si bien es cierto que ya había sido sustituida por otras cadenas de texto gran parte de la misma, imposibilitando la difusión de los datos. Sin embargo, estos cambios no supusieron ningún impedimento para el correcto análisis de la aplicación, que se centró mayormente en sus funcionalidades.