



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Puja Por Tu Resi

Gestión de residencias

Autor

Antonio Jiménez Martínez

Director

Francisco Javier Melero Rus



Escuela Técnica Superior de Ingenierías Informática
y de Telecomunicación

—
Granada, 16 de junio de 2017



puja por tu resi

Pujar Por Tu Resi

Gestión de residencias

Autor

Antonio Jiménez Martínez

Director

Francisco Javier Melero Rus

Pujar Por Tu Resi: Gestión de residencias

Antonio Jiménez Martínez

Palabras clave: residencia, estudiante, apuesta, API, Web, diseño responsivo, reserva, comercialización

Resumen

Este es un trabajo fin de grado (TFG), tiene el objetivo de aprender a desarrollar un proyecto a gran escala, enfrentarse a todos los problemas que lleva consigo y de aplicar todos los conocimientos obtenidos durante todos los años académicos.

Nuestro proyecto, un gestor de residencias, sirve para cubrir una necesidad en el sector estudiantil, los procesos de elegir o solicitar residencia son bastante complicados y tediosos. Sin embargo, a través de esta aplicación se convertirá en una tarea amena. Por lo tanto, nuestro software tiene como principio ofertarles a todos los estudiantes las distintas residencias y sus habitaciones.

Actualmente no existe nada parecido a este producto en el mercado, lo cual lo convierte en una novedosa herramienta para el usuario. Por otro lado sabemos que existe una alta competencia indirecta de todos los gestores de hoteles que existen actualmente. Nuestro producto puede verse desde dos perspectivas, la del estudiante y la de la propia residencia.

El usuario estudiante tiene la capacidad de pujar por varias habitaciones. En el momento en el que una de ellas le es asignada podemos realizar distintas tareas en la aplicación, entre las que destacamos comunicarte con la residencia a través de una mensajería interna, analizar la información de la residencia y de la habitación en concreto, realizar los pagos de las mensualidades y presentar incidencias a la residencia por el mal funcionamiento sobre un elemento de la misma o por el descontento en esta.

Por otro lado, tenemos la vista del usuario residencia, este puede darse a conocer a través de la aplicación y presentar sus servicios a los usuarios de esta. Entre las tareas que le corresponden al usuario podemos subrayar la capacidad de añadir habitaciones al sistema. Entre el conjunto de acciones, también encontramos comunicarse con los estudiantes, los cuales tienen una habitación en la residencia, ver las incidencias y actualizar el estado de estas, ver la información sobre las habitaciones y los estudiantes, además ver si los clientes de la residencia han realizado el pago del mes.

La aplicación es accesible actualmente desde la web y tiene un servidor al cual se accede a través de una API. El diseño de la web es *responsive*, adaptativo, lo cual puede ser visualizado cómodamente desde cualquier dispositivo desde móvil hasta escritorio.

Finalmente, hemos realizado una encuesta para posibles usuarios de la aplicación, la cual ha tenido bastante aceptación y ha ayudado identificar problemas y posibles trabajos futuros. Pero lo más importante ha sido que nuestro producto ha sido aceptado y de manera que puede llegar a comercializarse y empezar a usarse.

Puja Por Tu Resi: Manager college

Antonio Jiménez Martínez

Keywords: college, student, bid, Web, API, responsive design, booking, delivery

Abstract

This is a final project degree, it is developed with the main goal of learning to work on a large scale project, facing all the problems which may appear during that period and to apply all the knowledge obtained during my training.

Our project, a college manager, is focus on covering one necessity found in the student area, the task of choosing which college to live during their university studies is hard and tedious. However, the application will make that task really enjoyable. Therefore, our software has a principle offer to all students: the different colleges and rooms.

Nowadays there is nothing like that product in the market, so it is a really new tool for users. On the other hand, it is known that there is a high indirect competition from all hotel managers used nowadays. Our product can be analyzed and seen from two views, from the student and the college.

The student user has the capability to bid for several rooms, when one of them is assigned to the user, he can make different actions in the application, among all of them we can highlight communicating with the college through messages, analyzing the data of the college and the room, making the payment of every month and reporting incidences to the college about one element malfunction or the annoyance about anything.

So as to approach the view of the college user, he can be famous through the application and share his services to all the users of that. Among the tasks which that user can have access we could highlight adding rooms to the system. There is a wide range of actions as communicating with all students, who have a room of the college, seeing the incidences and updating its status, seeing the information of the rooms and the students, also checking if the students have made the payment of the month.

The application can be reached from the web and it has a serve, what is getting through an API. The design of the web is responsive, it means that can be adapted to be display in different kinds of devices such as desktop, tablet or mobile.

To sum up, we have done a survey for possible users of the application, it has been really interesting and it has helped to identify problems and futures works. Last but not least, our product has been accepted, so the next step could be delivery to real colleges and start to be used.

Yo, **Antonio Jiménez Martínez**, alumno de la titulació Grado en Ingeniería Informática de la **Escuela Técnica Superior Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 15426661D, autorizo la ubicació de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Antonio Jiménez Martínez

Granada a 16 de junio de 2017.

D. Francisco Javier Melero Rus, Profesor del Área de Lenguajes y Sistemas Informáticos del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Puja Por Tu Resi, gestor de residencias***, ha sido realizado bajo su supervisión por **Antonio Jiménez Martínez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 16 de junio de 2017.

El director:

Francisco Javier Melero Rus

Agradecimientos

Mis agradecimientos van dirigidos principalmente para mi familia y amigos los cuales me han apoyado durante el periodo de desarrollo.

Gracias a mi tutor, Francisco Javier Melero Rus, por haberme ayudado y sobre todo hacerme ver lo que yo no podía.

Agradezco a todas las comunidades de software donde he podido encontrar soluciones a muchos de los problemas que han aparecido en el desarrollo.

Índice general

Introducción	1
Metodología de desarrollo	2
Metodología Scrum	4
Primer Sprint.....	4
Segundo <i>sprint</i>	6
Tercer <i>Sprint</i>	7
Conclusión	8
Definición.....	9
Estado del arte.....	9
Especificación Requisitos software (ERS)	13
Introducción.....	13
Descripción general	14
Requisitos específicos.....	15
Historias de usuario	24
Diseño	27
Diagrama de clases	27
Diagrama de secuencia.....	30
Diagrama de actividad.....	35
Diagrama IFML.....	39
Implementación.....	42
Tecnología utilizada	42
Back end	42
Front end	48
Control de versiones.....	54
Seguridad web	56
Brute force.....	56
Command execution/injection	56
Cross-site request forgery (CSRF)	57
Upload file.....	58
Cross-site scripting (XSS).....	58
SQL injection.....	58
Accesibilidad web	60
Problemas y soluciones	63
Organización archivos <i>front-end</i>	63
Header and AJAX	64
Bootstrap-select.....	64
Módulo de pago.....	65
Página inicio.....	65

Clase <i>Agreement</i>	66
Organización de las fechas	66
Table <i>responsive</i> design	67
Testeo software	68
Test unitarios y funcionales	68
Test de usabilidad	70
Documentación	72
Actividades legales en la web	72
Política de privacidad	72
Política de <i>cookies</i>	72
Aviso legal	73
Condiciones generales de contratación y uso	74
Logo	75
Trabajos futuros	76
Comercialización	77
Licencia	79
Asignaturas relacionadas	80
Conclusión	81
Bibliografía	82
Apéndice	84
Acrónimos y abreviaturas	84
Política de privacidad de Puja Por Tu Resi	85
Política de <i>cookies</i> de Puja Por Tu Resi	86
Aviso legal de Puja Por Tu Resi	87
Condiciones generales de contratación y uso de PujaPorTuResi	88
FAQ de Puja Por Tu Resi	89
Despliegue	91
Formulario de Puja Por Tu Resi	92
Resultados encuesta de Puja Por Tu Resi	96
Métodos API	102

Índice de figuras

Figure 1 Diagram Gantt	3
Figure 2 Epic primer sprint	4
Figure 3 Gráfico primer sprint	6
Figure 4 Gráfico Segundo sprint	7
Figure 5 Gráfico tercer sprint	8
Figure 6 Gráfico todos los sprint.....	8
Figure 7 ScreenShot Gestión De Residencias (GDR).....	9
Figure 8 Screenshot Gestión de residencias de la tercera edad Geriges	10
Figure 9 Screenshot GreenStudent	11
Figure 10 Screenshot Software para residencias Inovaciones Informáticas S.A	12
Figure 11 Diagrama de clases	28
Figure 12 diagrama secuencia enviar mensaje.....	31
Figure 13 Diagrama secuencia pujar por una habitación	32
Figure 14 Diagrama secuencia pagar mensualidad	33
Figure 15 Diagrama de secuencia Crear una incidencia	34
Figure 16 Diagrama de actividad enviar mensaje	35
Figure 17 Diagrama de actividad pujar por una habitación	36
Figure 18 Diagrama de actividad pagar una mensualidad	37
Figure 19 Diagrama de actividad crear una incidencia.....	38
Figure 20 Diagrama IFML Search	40
Figure 21 Diagrama IFML Incidencias	40
Figure 22 Diagrama IFML Message.....	41
Figure 23 Conjunto de métodos de la PIA	43
Figure 24 Modelo Vista Controllador Symfony	44
Figure 25 Screenshot seguridad	45
Figure 26 Screenshot Modulo crear pdf con una plantilla	46
Figure 27 Screenshot Modulo de documentación	46
Figure 28 Screenshot CloudBackupBundle.....	47
Figure 29 Screenshot python script.....	48
Figure 30 Screenshot AJAX	49
Figure 31 Screenshot API google maps	50
Figure 32 Screenshot Floatthead.....	50
Figure 33 Screenshot slider	51
Figure 34 Screenshot drag and drop	52
Figure 35 Screenshot página initial	53
Figure 36 Screenshot Diseño responsive.....	54
Figure 37 Screenshot fuerza bruta code.....	56
Figure 38Screenshot Validate Class.....	57
Figure 39 Screenshot Access Control Method.....	58
Figure 40 Screenshot upload file	58
Figure 41 Screensot SQL injection	59
Figure 42 Herramienta TAW para analizar accesibilidad.....	61
Figure 43 Resultado accesibilidad A	62
Figure 44 Logo para añadir a la web respecto a la prioridad A	62
Figure 45 Estructura de archivos front-end.....	63
Figure 46 Error Access control Allow origing.....	64
Figure 47 Ejemplo element Boostrap-select	65
Figure 48 History Table responsive design	67
Figure 49 Logo de nuestra aplicación	75

Introducción

Hoy en día, donde todo se hace a través del Internet desde leer un periódico hasta realizar una reserva, hemos creado una aplicación web la cual es totalmente necesaria en el mercado actual, este será muy útil para muchos estudiantes universitarios. Cuando estos salen de casa para estudiar fuera y deciden entre vivir en un piso o en una residencia universitaria. En el caso de una residencia, nuestra aplicación le será de gran ayuda. El procedimiento actual para ir a vivir en una residencia es un poco tedioso, puesto que en primer lugar tienes que informate de todas las residencias de la ciudad, después ver localización, precio, fama, aspecto y equipamiento de la residencia y de las habitaciones. Para ello es necesario contactar por teléfono y ver la página web y si hay habitaciones disponibles ir a la residencia a verla. En el caso en que la residencia te guste te apuntas a una lista de espera y cuando te llamen eliges la habitación.

Con nuestra aplicación todo es más fácil puesto que puedes ver directamente que habitaciones hay libres y cómo son las habitaciones y qué precio y equipamiento tienen. En el caso en que estés interesado puedes pujar por ellas y cuando has sido aceptado firmar el contrato y entonces puedes realizar distintas acciones como poner incidencias, realizar el pago de un mes o contactar con la residencia.

Un caso interesante de donde hemos aprendido bastante es en Linköping (Suecia) , donde existe una empresa (studentbostader, n.d.) la cual tiene todos los pisos de alquiler legales de la ciudad, es decir, existe un monopolio. Esta empresa es gestionada a partir de una web donde, el usuario elige la habitación a través de un sistema de pujas y todos los pagos y la administración de esta es realizada a través de la web.

A raíz de la necesidad de un software que ayudase a ver las residencias disponibles y a firmar el contrato y la facilidad con la que todo funciona en la aplicación de alquileres en Suecia decidimos combinar ambas ideas y crear nuestro proyecto.

Actualmente en España no existe este tipo de aplicaciones, en cambio las aplicaciones de reserva de hoteles son muy utilizadas y estas podrían añadir nueva funcionalidad como reserva de residencias, es decir, existe una competencia futura.

Metodología de desarrollo

En esta sección explicaremos la planificación que hemos seguido para la realización de este proyecto.

Inicialmente comenzamos con una definición del proyecto, esta fue una de las principales fases puesto que si este era exitoso continuaríamos. Esta la podemos definir mediante las siguientes tareas:

- Requisitos del sistema y objetivos
- Historias de usuario.
- Estado del arte.
- Estimación temporal: 30 horas

Una vez el proyecto es aprobado pasamos a aplicar nuestra metodología. Esta será SCRUM, a partir de esta controlaremos y desarrollaremos nuestra idea. Esta metodología consiste en que vamos realizando ciclos o iteraciones. En cada una de ellas, trabajamos en las principales tareas de nuestro proyecto las cuales son diseño, implementación y testeo. El proceso de desarrollo ha tenido un total de 4 iteraciones, de las cuales hablaremos más tarde. A continuación, explicamos cada una de las principales tareas.

- 1) Diseño. En esta etapa se decide la estructura del proyecto, como se va a organizar y comunicar la información.
 - Diagrama de clases y modelo entidad relación.
 - Diagrama de actividad.
 - Diagrama de secuencia.
 - Diagrama IFML y bocetos iniciales de la web.
 - Estimación temporal: 40 horas
- 2) Implementación. Esta es la etapa más larga puesto que es un sistema software desarrollado por una única persona. Durante esta etapa hubo una parte de aprendizaje de las tecnologías que utilizaremos. Tiempo estimado 180 horas. En este apartado encontramos:
 - Problemas, los cuales definiremos después como nos hemos enfrentado y resuelto.
 - Tecnologías utilizadas.
 - Análisis de la seguridad web.
 - Comprobar la accesibilidad web.
- 3) Testeo. En esta fase se implementan unos tests para los métodos de la API de nuestro sistema. Además de estudiar la accesibilidad de nuestra web. La estimación temporal es 40 horas. Dos tipos de test:
 - Tests funcionales.
 - Test de usabilidad.

Para terminar, hemos desarrollado la documentación de la aplicación, esta es la parte final y principalmente consiste en definir las partes de las que no se ha hablado aun y unificar

toda la documentación. Tiempo estimado es de 60 horas. En este apartado trabajamos en los siguientes temas.

- Análisis del logo.
- Licencia
- Estudio de mercado y posible comercialización.
- Trabajos futuros.
- Política de privacidad y *cookies* y condiciones generales y aviso legal.
- FAQ.
- Despliegue.
- Asignaturas relacionadas

A continuación mostramos el diagrama Gantt (CCM Benchmark Group, 2017) , donde representamos en una vista cronología como se va a desarrollar el proyecto.

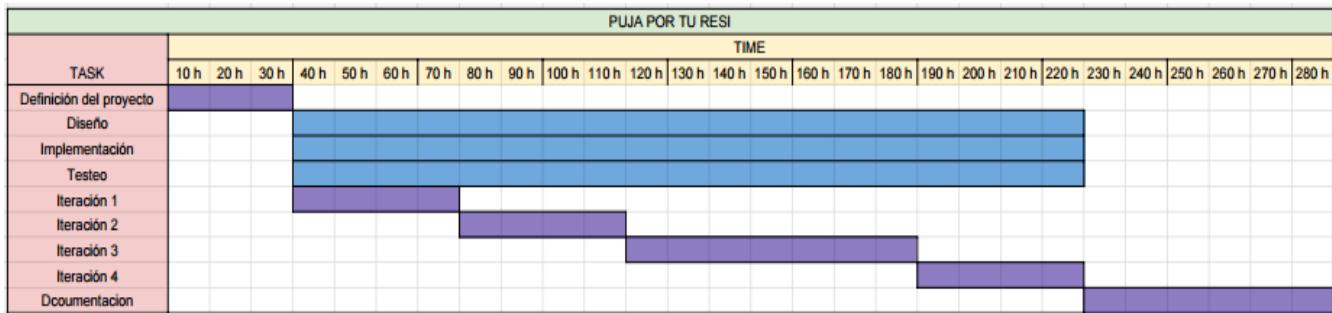


Figure 1 Diagram Gantt

Metodología Scrum

En un principio empezamos a trabajar con una metodología Scrum, en la cual tomábamos historias de usuarios y las dividíamos en pequeños tareas y calculábamos su duración para desarrollarlas entre 7-10 días, sin tener en cuenta la prioridad de las tareas. Pero al no apuntar las tareas y no registrar cuento se tardaba exactamente en realizarlas no hemos podido mejorar nuestras estimaciones. En ese momento nos dimos cuenta que lo mejor era utilizar un sistema que gestionase Scrum, para ello analizamos en Internet varias herramientas que existían sobre gestión ágil software y elegimos SCRUMDO (scrumdo, n.d.), ya que este estaba recomendado y parecía sencillo. Ahora podemos agrupar tareas, gestionar su estado, su duración y estas se almacenan en el sistema. Esto llevó consigo aprender mejor el funcionamiento de una metodología de desarrollo Scrum: entre sus elementos *sprint*, *label*, *epic*, estados y demás. También podemos ver la evolución de mis tareas. En este momento realizamos las tareas por prioridad. A continuación hablaremos de los distintos *Sprint*:

Primer Sprint

En la siguiente imagen podemos apreciar nuestro primer *Sprint*, aproximadamente en 40 horas, si cada día trabajamos 5 horas. La duración del *sprint* es de 8 días trabajo. En el siguiente Sprint veremos si necesité o me faltó tiempo.

Figure 2 Epic primer sprint

En este *sprint* tenemos actividades de diferentes *epic* (documentación, gestión de habitación, gestión API, diseño web). Entre las tareas que estimadas podemos destacar

1. Diagrama clases, tiempo necesario estimado 4 horas:
 1. Una residencia tiene varios responsables.
 2. Una residencia tiene varias cuentas bancarias; una activada.
 3. El equipamiento de la residencia pertenece a esta, no es otra clase.
 4. Incidencia y mensualidad están relacionados con la residencia de forma indirecta, a través del contrato.
 5. El equipamiento de una habitacional pertenece a esta, no a otra clase.
2. Gestión sesión usuario con las siguientes subtareas, tiempo necesario estimado 9 horas:

1. Analizar la herramienta de gestión sesión en Symfony.
2. Implementar mantener la sesión activa del usuario.
3. Implementar la API de habitación, tiempo necesario estimado 12 horas:
 1. Obtener datos de la habitación.
 2. Crear la tabla habitación en la base de datos.
 3. Eliminar habitación.
 4. Método: crear habitación.
 5. Validar todos los valores de entrada de la habitación.
 6. Tener la opción de crear la habitación sobre una ya creada.
 7. Modificar el estado de una habitación.
4. Decidir la interfaz del usuario Residencia, tiempo necesario estimado 15 horas:
 1. Vista perfil: datos, personal responsable, cuenta bancaria.
 2. Vista habitaciones: crear, editar, eliminar.
 3. Vista cliente: todos los estudiantes que tiene una habitación en dicha residencia.
 4. Vista mensajería: con mensajes, con conversaciones con todos los usuarios.
 5. Vista incidencias: con el estado, propietario, fecha, archivo.

A continuación podemos ver el gráfico de cómo se han desarrollado dichas tareas del primer *sprint*:



Figure 3 Gráfico primer sprint

Respecto a este periodo, podemos ver que las tareas se han ido convirtiendo de estado *To do* al estado *Doing* y *Done* progresivamente y se puede ver que se ha terminado en la fecha prevista. En cambio, podemos apreciar que existe una malformación o pico en la gráfica, esta se debe a que durante el desarrollo se añadieron más tareas las cuales estaban previstas para dicho modulo, como por ejemplo una de ellas fue, que en un principio se tenía pensado conocer el estado de una habitación por una variable. En cambio, ahora se puede pujar por la habitación en todo momento siempre y cuando esté disponible, es decir, no tenga ningún contrato en ese momento.

Por otro lado el *Sprint* se hizo en los días que se tenía en cuenta, como se puede ver en la gráfica.

Segundo sprint

Este fue programado para 70 horas, es decir un conjunto de 14 días. Esta estaba formada por los siguientes objetivos:

- User (college) view
- Sign agreement, web view student
- Sign agreement API: method y class
- Message API college user: method
- Message , web view college
- Main data profile college: API and web



Figure 4 Gráfico Segundo sprint

Este *sprint* se realizó sin problemas en 8 días, de los 14 previstos. Esto se debe a una mala estimación. El módulo *sign agreement* parecía complicado desde un principio y luego se pudo llevar a cabo sin inconvenientes. Este problema nos ayudó en etapas futuras a realizar una estimación del tiempo más concreta.

Tercer Sprint

En la tercera y última iteración, se encuentra toda la funcionalidad del usuario *Residencia*: cómo gestionar los estudiantes, ver la información de los pagos de las mensualidades, gestionar las incidencias, gestionar los datos bancarios, la lista de responsables de la empresa, etc.

Este se realizó en el tiempo estimado, un total de 45 horas (9 días laborales), lo que indica que después de un *sprint* sin documentar y tres *sprint* con seguimiento podemos decir que la estimación realizada en las tareas de mi proyecto es lógica.

A continuación, puede verse el gráfico de tiempo del último *sprint*. Como se puede ver durante el fin de semana del 17-18-19 de febrero no trabajamos en el proyecto. Por lo demás, parece que el balance *To do* y *Done* está correcto.

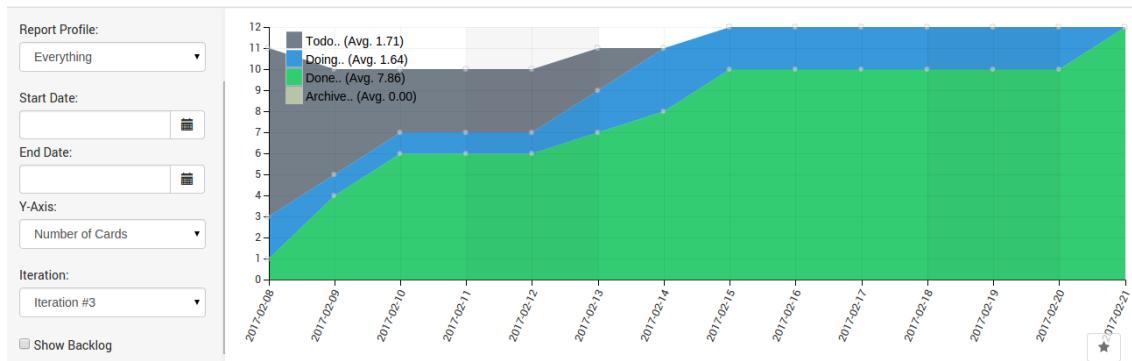


Figure 5 Gráfico tercer sprint

Conclusión

Para concluir, tenemos una imagen que describe el proceso de desarrollo en tres iteraciones. En la imagen podemos percibir que en cada iteración las tareas se han añadido a *To do* y que al final de la iteración las tareas han acabado como *Done*. Podemos subrayar que las escalas de tareas para hacer y hechas ha avanzado correctamente. Por lo tanto, la etapa de desarrollo ha concluido correctamente como esperábamos.

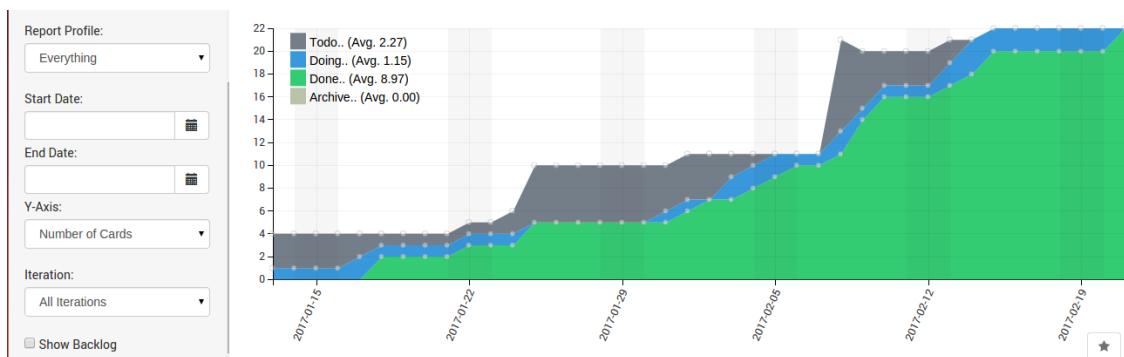


Figure 6 Gráfico todos los sprint

Definición

Estado del arte

Estado del arte, este apartado consiste en hacer un estudio de mercado sobre otras aplicaciones con funcionalidades similares. Comprobaremos si esta tecnología se está usando actualmente. También valoraremos las aplicaciones comparándolas con la nuestra, respecto a las distintas funcionalidades y demás. El objetivo de este apartado es que el lector se dé cuenta de que esta herramienta es una necesidad actual. A continuación aparece una lista con aplicaciones en el mismo ámbito y contexto tecnológico:

- Gestión De Residencias (GDR) (gestionderesidencias, n.d.). Este es un software para la gestión de residencias geriátricas. Este tipo de residencias van dirigidas a ancianos. Este software tiene un precio de 50€ al mes por residencia, más 2€ por residente. Por otro lado este es un software el cual tiene que estar instalado en dispositivo como ordenador. También dispone de un soporte técnico. El objetivo de la aplicación es aumentar la productividad de la residencia. Tiene como requisito la conexión a Internet, ya que la base de datos está en la nube.
- Entre los módulos en común con nuestra aplicación encontramos:
 - Gestión de mensajería.
 - Gestión de residentes.
 - Gestión de los pagos.
- Entre los módulos propios de la aplicación:
 - Para los empleados: gestión del tiempo de trabajo y la lista de tareas.
 - Gestión del menú del comedor.



Figure 7 ScreenShot Gestión De Residencias (GDR)

- Gestión de residencias de la tercera edad (geriges, n.d.). Se trata de una aplicación web para móvil y ordenador conectado a la nube; es decir, se puede conectar de forma remota. Este se puede adaptar a distintos tipos de residencias, es decir, existe distintos paquetes de pagos desde 49€ hasta 119€; además de los paquetes de soporte que va desde 40€ hasta 60€.
 - Entre los módulos en común con nuestra aplicación encontramos:
 - Gestión de residentes y del personal.
 - Gestión de los pagos.
 - Entro los módulos propios de la aplicación:
 - Para los empleados: gestión del tiempo de trabajo y la lista de tareas.
 - Gestión de actividades de la residencia.
 - Gestión del menú y dietas del comedor.
 - Gestión de lavandería



Figure 8 Screenshot Gestión de residencias de la tercera edad Geriges

- GreenStudent (Green Software, S.L., n.d.). Es un software para la gestión de residencias de estudiantes. Este tiene unos requisitos mínimos para su instalación, es compatible con Windows. Actualmente está siendo usado por más de 70 residencias. Su objetivo es gestionar las áreas de la residencia.
 - Entre los módulos en común con nuestra aplicación encontramos:
 - Gestión de solicitudes de estudiantes
 - Gestión de cobros a través de recibos a domicilio a los clientes.
 - Gestión de residentes.
 - Entro los módulos propios de la aplicación:

- Gestión de ocupación de las habitaciones.
- Gestión del almacén de los pedidos.
- Gestión de limpieza de habitaciones.
- Gestión auditoria interna
- Gestión de salones, para el alquiler de salones.
- Gestión administrativa de la caja y facturación diaria.

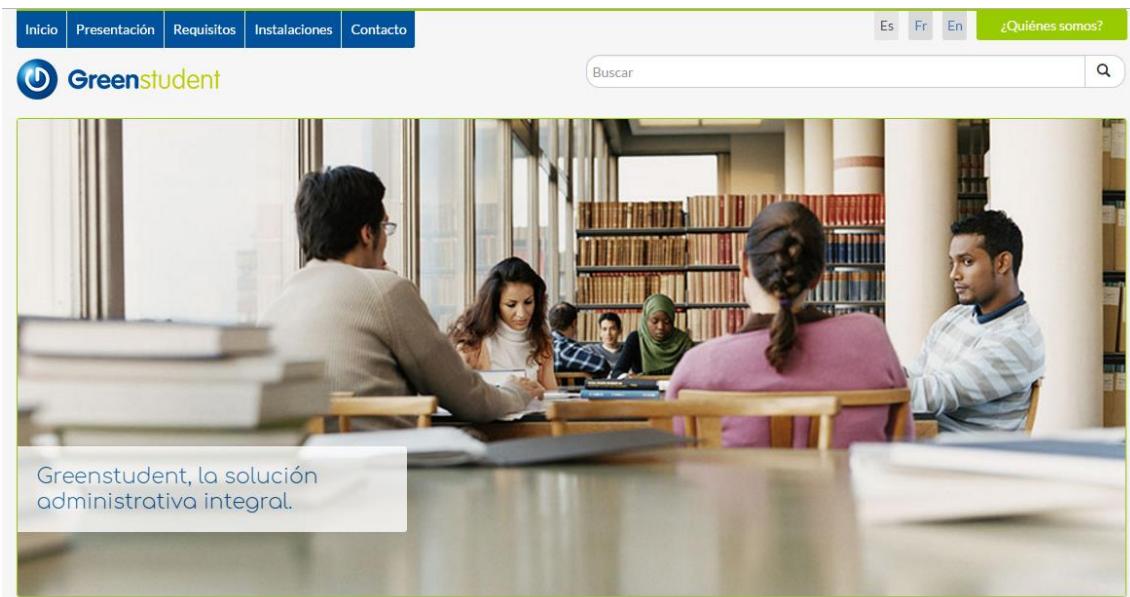
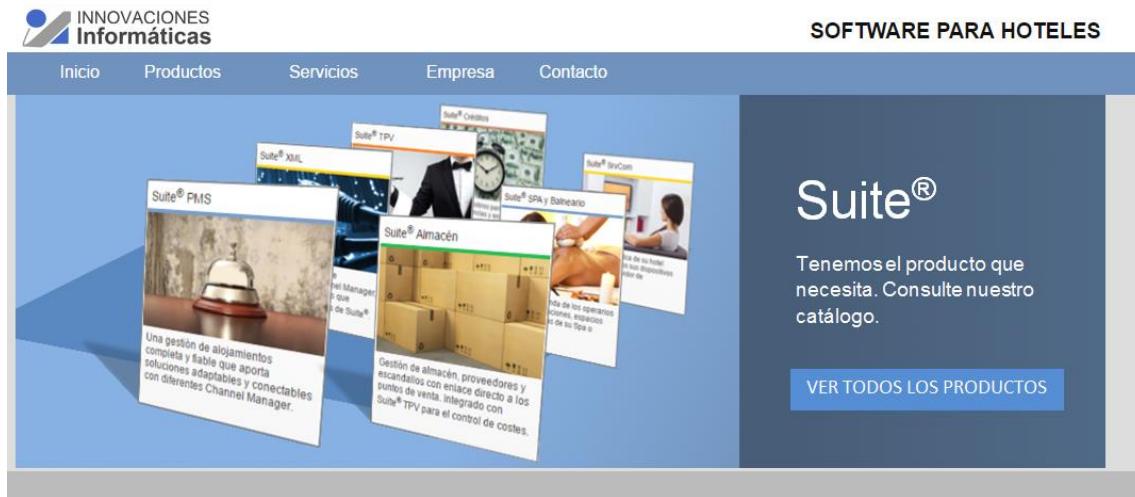


Figure 9 Screenshot GreenStudent

- Software para residencias (Innovaciones Informáticas S.A. , n.d.). Es principalmente un gestor de alojamientos. Es bastante flexible por lo que puede ser utilizado por residencias de estudiantes, militares o guardia civil. El coste de este depende del destinatario. Entre los principales módulos encontramos gestión de vehículos, limpieza de habitaciones, comedor y restaurantes. Por otro lado tiene otros módulos similares a nuestro proyecto como pagos, reservas y demás.



SOFTWARE PARA RESIDENCIAS

Figure 10 Screenshot Software para residencias Inovaciones Informáticas S.A

- Además encontramos páginas informativas sobre residencias universitarias, con la información de la residencia, fechas de inscripción y demás.

Para concluir podemos decir que ninguna de las aplicaciones web y software analizadas tienen una vista y funcionalidad para el cliente más importante, el estudiante. Toda ellas están hechas para ser utilizadas por el personal de la residencia. En cambio nuestra aplicación permite las dos opciones. De esta manera esta es más ventajosa. En cambio, nuestro sistema no tiene gestor de limpieza de habitaciones, ni gestor de menú y dietas de comedor, entre otros. Además, no hemos encontrado en ninguna de las aplicaciones analizadas un gestor de incidencias, el cual puede ser muy útil.

Especificación Requisitos software (ERS)

Introducción

En este documento se describirán los requisitos del sistema, según el Estándar de especificación de requisitos (ERS) IEEE 830 (IEEE Std. 830-1998, 2008); para el proyecto *PujaPorTuResi*, el cual será realizado como trabajo de fin de grado (TFG) por Antonio Jiménez Martínez, con tutor Francisco Javier Melero Rus en el curso académico 2016/2017.

Propósito

Este documento trata de plasmar todos los requisitos y funcionalidades que el sistema debe tener para su diseño, implementación y validación o testeo. Por otro lado, este documento va dirigido a cualquier usuario, desarrollador o examinador de la aplicación.

Ámbito del Sistema

El nombre seleccionado para el sistema es: *PujaPorTuResi*, es un nombre bastante coloquial con el objetivo de llamar la atención de los futuros usuarios; este también describe la esencia de la aplicación web ya que está consiste en que los usuarios pujen por habitaciones, en las cuales podrán residir.

Esta aplicación permitirá a estudiantes encontrar residencia a través de un buscador con distintos filtros como nombre residencia, precio, requisitos, fechas disponibilidad, etc. El usuario podrá pujar por las habitaciones que más interesantes le parezcan, través de un sistema de pujas. Una vez el usuario tiene su habitación asignada, este debe firmar el contrato; a partir de este momento el usuario puede hacer distintas tareas en la aplicación como: realizar los pagos mensuales, contactar con la residencia y exponer incidencias a esta; además de ver toda la información de la residencia, de la habitación y del contrato.

Por otro lado, la residencia juega un papel importante en el proyecto, esta será otro usuario el cual se dará de alta en la aplicación y añadirá habitaciones a la base de datos. Entre las funcionalidades de este usuario destacamos: conocer que estudiantes tiene un contrato con la residencia y ver su información de pagos y contactar con ellos. Además podrán resolver las incidencias expuestas por los estudiantes.

Entre los beneficios podemos destacar la automatización del proceso de asignación entre estudiante y residencia o el proceso de pagos. Respecto el proceso de asignación de una habitación a un estudiante tiene que ver la antigüedad del estudiante; es decir; cada estudiante tiene puntos, los cuales están determinados por la fecha en la que se registró el usuario. Entonces el estudiante pujará por una habitación y al final de la semana, se asigna la habitación al estudiante con más puntos que ha pujado por ella. En consecuencia la residencia no tiene que hacer nada en este proceso, ya que este es totalmente transparente y autónomo para ella.

En relación con el sistema de pagos, cuando la residencia y el estudiante firman el contrato, se generan los pagos que deben ser realizados, entonces el usuario entra al sistema, pincha en pagar el mes que elija y través de un TPV realiza el pago y este se queda almacenado en el sistema.

Dentro de los objetivos del proyecto podemos destacar varios:

- A través de la plataforma el estudiante se puede elegir entre un amplio conjunto de residencia y habitaciones para vivir durante un periodo académico.
- Las residencias que se ofrecen, patrocinan sus características.
- Los estudiantes puede mostrar sus quejas a través de la plataforma y la residencia tiene un sistema de gestión de estas.
- La residencia y los estudiantes pueden contactar entre si y viceversa a través del sistema.
- Los estudiantes puedan buscar habitaciones utilizando un filtro de equitación: como wifi, baño, televisión y demás.

Descripción general

Este apartado tiene como objetivo mostrar el contexto y los factores del proyecto, para que se comprenda mejor los requisitos.

Perspectiva del producto

El proyecto a realizar no tiene relación con ningún otro proyecto, de esta manera este no tendrá ningún subsistema y será construido desde la base. Así que no hay que tener en cuenta los requisitos de ningún otro subsistema.

Funciones del producto

La aplicación está compuesta por varios módulos divididos por su funcionalidad, podemos destacar:

- Gestión de usuarios (estudiantes, residencia): como hemos citado anteriormente existe dos tipos de usuarios estudiantes y residencias; el sistema provee un módulo para la gestión de datos personales de ambos usuarios. Además, en este módulo incluimos la gestión de las cuentas bancarias y del personal responsable, es decir, distintos métodos como crear, eliminar, activar y demás.
- Gestión del sistema de habitaciones: este módulo es específico para el usuario residencia, este módulo contempla la creación, eliminación de habitaciones además de ver que estudiantes están en cada habitación así como su contrato con esta.
- Gestión de pujas: este módulo administra las pujas de los usuarios realizadas sobre distinta habitaciones además de la asignación de una habitación al estudiante en la fecha de fin de pujas. Cuando una habitación es asignada a un estudiante todas las pujas del estudiante son eliminadas y todas las pujas de la habitación son eliminadas.
- Gestión de contratos: una vez la habitación es asignada a un estudiante, este descarga el contrato, lo firma y lo sube a la plataforma desde este momento existe un acuerdo entre la residencia y el estudiante. La lista de pagos mensuales se crea en este momento.
- Gestión del sistema de pagos del estudiante a la residencia: cada mes el estudiante accede al sistema elige el mes a pagar y realiza el pago, totalmente seguro a través de un TPV, el cual es externo al sistema. Por otro lado la residencia puede ver qué pagos han sido realizado por el usuario.

- Gestión sistema de incidencias: los estudiantes pueden mostrar sus quejas a la residencia a través de la aplicación. Y la residencia puede actualizar el estado de esta a través del sistema.
- Gestión del sistema de mensajería entre estudiantes y residencia y viceversa: el sistema permite enviar mensajes entre los usuarios de la aplicación.

Características de los usuarios

Los usuarios del sistema serán: estudiantes, residencia y administrador.

- Estudiante: tendrá conocimiento y experiencia de aplicaciones web de gestión de usuario (como redes sociales) y alojamiento (como aplicaciones de hoteles).
- Residencia: tendrá conocimiento y experiencia de aplicaciones web de gestión de alojamiento (como aplicaciones de alquiler de habitaciones o pisos).
- Administrador: tendrá perfecto conocimiento del funcionamiento del sistema. La única función que realizará será la asignación de las habitaciones a los estudiantes en la fecha final de pujas de forma totalmente automática.

Restricciones

- El lenguaje de programación utilizado para el servidor será *PHP (con framework Symfony)* y para el cliente *HTML, Javascript y CSS*.
- Respecto la seguridad del sistema: el sistema estará protegido de las principales vulnerabilidades de sistema web.
- La interfaz de usuario será realizada en un cliente web y la cual será visible en distintos dispositivos como ordenador, *tablet* o móvil utilizando un diseño *responsive*.
- El cliente y el servidor se comunicará a través de los métodos de la API descritos en el apéndice.

Requisitos específicos

En este apartado encontramos los requisitos, descritos con un mayor detalle, con el objetivo de que los desarrolladores puedan entender el sistema para implementarlo. Los requisitos podrán entenderse por distintos tipos de personas desde clientes hasta diseñadores.

Requisitos Funcionales

Identificación:	RF01
Nombre:	Registro usuario
Usuarios:	Estudiante / Residencia
Características:	Los usuarios deben estar registrados para poder autenticarse en el sistema.
Descripción:	<p>En primer lugar hay que especificar el tipo de usuario (estudiante o residencia), puesto que el formulario será distinto.</p> <ul style="list-style-type: none">• Datos necesarios para registro estudiante: DNI, nombre y apellidos, correo, contraseña.• Datos necesarios para registro residencia: CIF, nombre empresa, correo, dirección, teléfono, página web, contraseña,

	<p>equipamiento.</p> <p>Es necesaria una validación del formato de todos los datos de entrada.</p> <p>La contraseña debe tener 8 caracteres.</p> <p>No pueden existir dos usuarios con el mismo DNI o CIF.</p>
--	--

Identificación:	RF02
Nombre:	Autenticación de Usuario
Usuarios:	Estudiante / Residencia / Administrador
Características:	Los usuarios deben identificarse para acceder al sistema.
Descripción:	Para el acceso es necesario (DNI – CIF – Código Administrador) y contraseña.

Identificación:	RF03
Nombre:	Recordar contraseña
Usuarios:	Estudiante / Residencia
Características:	Los usuarios pueden pedirle al sistema que recuerde su contraseña.
Descripción:	Una nueva contraseña aleatoria, será enviada al correo del usuario.

Identificación:	RF04
Nombre:	Modificar perfil usuario
Usuarios:	Estudiante / Residencia
Características:	Los usuarios pueden modificar el perfil y la contraseña.
Descripción:	<p>Los datos que pueden ser alterados.</p> <ul style="list-style-type: none"> • Estudiante: correo, contraseña. • Residencia: correo, dirección, teléfono, página web, contraseña y equipamiento. <p>Es necesaria una validación sobre el formato de todos los nuevos datos de entrada.</p>

Identificación:	RF05
Nombre:	Añadir cuenta bancaria
Usuarios:	Residencia
Características:	Añadir una cuenta bancaria
Descripción:	<p>Los datos de creación son IBAN, BIC y el nombre del dueño de la cuenta.</p> <p>En esta cuenta se realizaran los pagos de los estudiantes.</p>

Identificación:	RF06
Nombre:	Eliminar cuenta bancaria
Usuarios:	Residencia
Características:	Eliminar una cuenta bancaria
Descripción:	Se eliminará la cuenta bancaria siempre y cuando esta no sea la que está activada en ese momento.

Identificación:	RF07
Nombre:	Activar cuenta bancaria
Usuarios:	Residencia
Características:	Activar una cuenta bancaria
Descripción:	En esta cuenta se realizaran los pagos de los estudiantes.

Identificación:	RF08
Nombre:	Añadir información responsable
Usuarios:	Residencia
Características:	Añadir la información del responsable de la residencia
Descripción:	Los datos para la creación son el nombre, apellidos, DNI del responsable y puesto.

Identificación:	RF09
Nombre:	Eliminar información responsable
Usuarios:	Residencia
Características:	Eliminar la información del responsable de la residencia
Descripción:	Se eliminará el responsable del sistema.

Identificación:	RF10
Nombre:	Consultar perfil usuario
Usuarios:	Estudiante / Residencia
Características:	Los usuarios pueden consultar el perfil.
Descripción:	Todos los datos del perfil pueden ser consultados, excepto la contraseña. Los estudiantes pueden además consultar los puntos para pujar (1 punto por cada día en el sistema)

Identificación:	RF11
Nombre:	Cerrar sesión usuario
Usuarios:	Estudiante / Residencia / Administrador
Características:	Cerrar la sesión del usuario.
Descripción:	Cierra la sesión del usuario y vuelve a la página principal (autenticación).

Identificación:	RF12
Nombre:	Añadir una habitación.
Usuarios:	Residencia
Características:	La residencia añade una habitación de su propia residencia.

Descripción:	<p>La habitación debe tener la siguiente información:</p> <ul style="list-style-type: none"> • Identificador: una manera de identificarla individualmente • 3 fotografías • Precio por mensualidad en euros • Planta en la que se encuentra. • Tamaño habitación: m² • Seleccionar un equipamiento de la lista • El equipamiento debe contener los siguiente aspectos: <ul style="list-style-type: none"> ○ Televisión. ○ Baño. ○ Escritorio. ○ Estantería. ○ Armario. <p>La habitación debe ser individual (para una persona).</p>
---------------------	---

Identificación:	RF13
Nombre:	Dar de baja una habitación.
Usuarios:	Residencia
Características:	La residencia da de baja una habitación de su propia residencia.
Descripción:	La habitación no puede estar ocupada por nadie.

Identificación:	RF14
Nombre:	Mostrar residentes.
Usuarios:	Residencia.
Características:	Analizar los residentes, los cuales tienen un contrato un con una de las habitaciones de la residencia.
Descripción:	Analizar el perfil y la habitación de cada uno de los residentes actuales de su residencia.

Identificación:	RF15
Nombre:	Mostrar incidencias.
Usuarios:	Residencia / Estudiante.
Características:	Analizar las incidencias.
Descripción:	La residencia podrá ver las incidencias expuesta por los residentes. Los estudiantes podrán ver el estado de las incidencias expuestas a la residencia.

Identificación:	RF16
Nombre:	Modificar estado incidencias.

Usuarios:	Residencia.
Características:	Selecciona la incidencia y modificar el estado.
Descripción:	Una vez la residencia está trabajando con una queja, esta puede cambiar el estado de la incidencia.

Identificación:	RF17
Nombre:	Notificar una incidencia.
Usuarios:	Estudiante.
Características:	Notificar una incidencia sobre la residencia.
Descripción:	El estudiante debe especificar si la incidencia es sobre la habitación o un servicio de la residencia (cantina, baños, zonas comunes, lavandería, gimnasio, zona estudio). Se debe especificar una descripción y una fotografía descriptiva del problema.

Identificación:	RF18
Nombre:	Pago mensualidad.
Usuarios:	Estudiante.
Características:	El pago de la mensualidad se puede hacer automáticamente en la aplicación.
Descripción:	El usuario podrá utilizar su tarjeta de crédito para realizar el pago a la residencia. El coste lo fija la residencia. Se creará un recibo del pago en el sistema, el cual se puede descargar.

Identificación:	RF19
Nombre:	Descargar resguardo del pago de la mensualidad
Usuarios:	Estudiante.
Características:	Descargar un documento en formato pdf como resguardo del pago de la mensualidad a la residencia.
Descripción:	En el resguardo aparecerá la información del contrato, mes del pago e identificador de la transacción del TPV.

Identificación:	RF20
Nombre:	Enviar mensaje.
Usuarios:	Estudiante / Residencia
Características:	Los usuarios pueden enviarse mensaje a través de la aplicación.
Descripción:	Los estudiantes pueden enviar un mensaje a la residencia. La residencia puede enviar un mensaje a todos los residentes o a uno específico. El mensaje se realizará a través de un formulario, el cual contendrá: destino, texto, un archivo adjunto (opcional)

Identificación:	RF21
Nombre:	Leer un correo.
Usuarios:	Estudiante / Residencia.
Características:	El usuario puede leer un correo.
Descripción:	El usuario puede abrir la bandeja de entrada y leer cualquier correo.

Identificación:	RF22
Nombre:	Contestar correo.
Usuarios:	Estudiante / Residencia.
Características:	El usuario puede contestar un correo.
Descripción:	El usuario puede abrir un correo y responder este de la misma manera que escribiendo uno nuevo.

Identificación:	RF23
Nombre:	Pujar por una habitación.
Usuarios:	Estudiante.
Características:	El estudiante selecciona una habitación de una residencia en el proceso de pujas.
Descripción:	El usuario busca las habitaciones disponibles, elige una y puja por ella. El máximo de pujas por usuario es cinco.

Identificación:	RF24
Nombre:	Buscar residencia y habitación.
Usuarios:	Estudiante
Características:	El estudiante puede ver todas las habitaciones disponibles de la aplicación. Además puede hacer una búsqueda a partir de un formulario.
Descripción:	El usuario puede buscar todas habitaciones disponibles o que cumplen unos requisitos como residencia, fechas, equipamiento, etc.

Identificación:	RF25
Nombre:	Sistema asigna habitación
Usuarios:	Administrador
Características:	Una habitación es asignada por el sistema al usuario.
Descripción:	El último día de la puja, la habitación, se le ofrece al usuario con más puntos. Este proceso es automático por el sistema.

Identificación:	RF26
Nombre:	Aceptar o rechazar habitación
Usuarios:	Estudiante
Características:	El estudiante acepta/rechaza la habitación que se le ha sido asignada.
Descripción:	Debe entrar en la página web para aceptarla o rechazarla en un máximo de una semana. Si la habitación no es aceptada en siete días, será rechazada de forma automática.

Identificación:	RF27
Nombre:	Genera contrato.
Usuarios:	Administrador.
Características:	El sistema genera un contrato entre la residencia y estudiante en el cual se le asigna la habitación durante un periodo académico.
Descripción:	Cuando el estudiante ha aceptado la habitación, el sistema genera un

	contrato para formalizar el proceso. Este debe ser impreso, firmado manualmente y subido a la plataforma.
--	---

Identificación:	RF28
Nombre:	Firmar contrato.
Usuarios:	Estudiante.
Características:	El usuario descarga el contrato, lo firma y lo sube a la web.
Descripción:	Una vez que la habitación ha sido asignada y aceptada por el usuario, se genera el contrato. El usuario tiene que descargarlo, firmarlo manualmente y subirlo al sistema. De esta manera el proceso es formalizado. El contrato es enviado a la residencia.

Requisitos No Funcionales

Identificación:	RNF01
Nombre:	Interfaz sencilla.
Características:	La interfaz debe ser sencilla para que sea fácil interactuar con la aplicación.
Descripción:	La interfaz debe ser sencilla, de esta manera el usuario comprenderá todas las funcionalidades sin necesidad de un manual.

Identificación:	RNF02
Nombre:	Interfaz plataforma.
Características:	La interfaz está desarrollada en la web.
Descripción:	Esta tendrá un diseño <i>responsive</i> , es decir, podrá ser visualizada en distintos dispositivos como ordenador o móvil o <i>tablet</i> .

Identificación:	RNF03
Nombre:	Interfaz para usuario.
Características:	La interfaz debe ser distinta para cada tipo de usuario.
Descripción:	La interfaz debe acoplarse a la funcionalidad de cada tipo de usuario.

Identificación:	RNF04
Nombre:	Disponibilidad.
Características:	La aplicación debe estar trabajando de forma continua sin parar.
Descripción:	La aplicación debe estar trabajando todos los días, para que la residencia no pierda clientes.

Identificación:	RNF05
Nombre:	Seguridad
Características:	La aplicación debe seguir el programa de protección de datos.

Descripción:	La aplicación debe seguir la Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal, (LOPD). Esto lleva consigo explicar que se va a realizar con los datos requeridos en el registro: quien podrá verlos, como serán almacenados, destinatarios de la información. También explicar que si se da de baja un usuario todos sus datos (perfil, correo, relación con residencia, incidencias, transacciones económicas) serán eliminados.
---------------------	---

Interfaces externas

La interfaz web tiene dos vistas una para el usuario estudiante y la otra para el usuario residencia. En cada una de ellas se puede realizar distintas acciones.

La interfaz será sencilla y estará realizada en cliente web, de manera *responsive* para todos los dispositivos.

Funciones

Las funciones del sistema las vamos a dividir por usuarios:

- Estudiante:
 - Registrarse o darse de baja en el sistema. (RF01, RF02, RF03, RF06, RF11)
 - Modificar y consultar información de perfil o contraseña. (RF04, RF10)
 - Buscar residencia y habitación y pujar por ella. (RF23, RF24, RF22, RF23)
 - Aceptar habitación y formalizar proceso. (RF26, RF28)
 - Pagar la mensualidad de la residencia. (RF18, RF19)
 - Enviar un correo a la residencia. (RF20, RF21, RF22)
 - Notificar una incidencia. (RF15, RF17)
- Residencia :
 - Registrarse en el sistema. (RF01, RF02, RF03, RF06, RF11)
 - Modificar y consultar información de perfil o contraseña. (RF04, RF05, RF06, RF07, RF08, RF09, RF10)
 - Dar de alta o baja un habitación. (RF12, RF13)
 - Ver la lista de residentes. (RF14)
 - Enviar un correo a los residentes. (RF20, RF21, RF22)
 - Ver y modificar el estado de una incidencia. (RF15, RF16)
- Administrador:
 - Asignar habitación al estudiante. (RF25, RF27)

Atributos del Sistema

- Seguridad: los datos que se almacenen en el sistema deben seguir la protección de la ley de datos. Por otro lado se le debe informar al usuario el uso que se le va a dar a los datos ofrecidos.
- Fiabilidad: la interfaz debe ser sencilla para que los usuarios puedan utilizar toda la funcionalidad sin necesidad de un manual.
- Disponibilidad: el sistema tiene que estar disponible todos los días, para que no haya ningún problema en ninguna transacción. Existirá un correo para ponerse en contacto con el administrador en caso de existir alguna incidencia.

- Mantenimiento: el sistema será revisado y mantenido tras la creación.
- Portabilidad: la aplicación podrá ser utilizada en la web.

Historias de usuario

Estas reflejan una necesidad desde el punto de vista de los usuarios de la aplicación. Además son simples y escuetas por lo que se puede implementar como una funcionalidad en un único periodo (*sprint*) de una metodología ágil como SCRUM. Esta tiene un estructura: **Yo como [Usuario (tipo)], necesito [Descripción de la funcionalidad], con la finalidad de [Descripción del beneficio].**

De esta manera las historias de usuario representan qué hacer y cómo hacerlo.

Esta representación de los requisitos es distinta de los casos de uso, los cuales describen el proceso sobre una interfaz. A continuación, mostraremos las historias de usuario de nuestro proyecto:

Identificación:	Historia Usuario 01
Rol:	Estudiante / Residencia
Funcionalidad:	Registro en el sistema.
Resultado:	Poder autenticarme en el sistema y utilizarlo.

Identificación:	Historia Usuario 02
Rol:	Estudiante / Residencia
Funcionalidad:	Autenticar o salir del sistema.
Resultado:	Poder utilizar o dejar de utilizar funcionalidades del sistema.

Identificación:	Historia Usuario 03
Rol:	Estudiante / Residencia
Funcionalidad:	Recordar contraseña.
Resultado:	Poder autenticar en el sistema en caso de no recordar la contraseña.

Identificación:	Historia Usuario 04
Rol:	Estudiante / Residencia
Funcionalidad:	Modificar y consultar el perfil de usuario.
Resultado:	Poder utilizar, analizar y actualizar mi información.

Identificación:	Historia Usuario 05
Rol:	Residencia
Funcionalidad:	Registrar una habitación en el sistema.
Resultado:	Para que esta pueda ser asignada a un estudiante.

Identificación:	Historia Usuario 06
Rol:	Residencia
Funcionalidad:	Eliminar una habitación en el sistema.
Resultado:	Esta habitación no será utilizada más por los estudiantes.

Identificación:	Historia Usuario 07
Rol:	Residencia

Funcionalidad:	Ver el listado de residentes en las habitaciones.
Resultado:	Poder analizar los clientes de la residencia.

Identificación:	Historia Usuario 08
Rol:	Estudiante
Funcionalidad:	Ver el listado de residencias y habitaciones disponibles.
Resultado:	Para poder pujar por una habitación.

Identificación:	Historia Usuario 09
Rol:	Estudiante / Residencia
Funcionalidad:	Mostrar las incidencias.
Resultado:	Para ver el estado de las incidencias y resolverlas.

Identificación:	Historia Usuario 10
Rol:	Residencia
Funcionalidad:	Actualizar estado incidencias.
Resultado:	Poder finalizar la incidencia una vez este solventado el problema.

Identificación:	Historia Usuario 11
Rol:	Estudiante
Funcionalidad:	Añadir una incidencia.
Resultado:	Para que la residencia la resuelva.

Identificación:	Historia Usuario 12
Rol:	Estudiante
Funcionalidad:	Realizar el pago de la mensualidad.
Resultado:	Para poder seguir disfrutando de los servicios de la residencia.

Identificación:	Historia Usuario 13
Rol:	Estudiante /Residencia
Funcionalidad:	Enviar mensaje.
Resultado:	Poder comunicarse con la residencia o con un cliente.

Identificación:	Historia Usuario 14
Rol:	Estudiante / Residencia
Funcionalidad:	Leer y responder mensaje.
Resultado:	Leer el mensaje recibido y responder si es necesario.

Identificación:	Historia Usuario 15
Rol:	Estudiante
Funcionalidad:	Pujar por una habitación

Trabajo Fin de Grado Ingeniería Informática: Puja Por Tu Resi
Antonio Jiménez Martínez

Resultado:	Poder vivir en ella un periodo académico.
-------------------	---

Identificación:	Historia Usuario 16
Rol:	Estudiante
Funcionalidad:	Aceptar habitación asignada y firmar contrato.
Resultado:	Vivir en ella un periodo académico de forma formal.

Diseño

Diagrama de clases

El diagrama de clases es un modelo en un formato UML (*Unified Modeling Language*) el cual describe cómo es el sistema y cómo se organiza. Es uno de los pilares de programación orientada a objetos, también es el más utilizado. Además de identificar los atributos de la clase, expresa la relación entre ellos. La principal características es que a partir de este podemos generar el código de nuestro sistema, es una herramienta necesaria en el desarrollo software. A continuación, explicamos cada clase del diagrama de clases:

- **College:** almacena la información imprescindible del usuario *College*, *username* será el CIF de la empresa. Además almacenamos información del equipamiento, de la dirección con los valores longitud y latitud, además de datos de contacto como teléfono, email y url.
ID: *username*
- **Student:** almacena la información imprescindible del usuario *Student*, *username* es el DNI. El valor *creation_date*, será utilizado para saber cuántos puntos tiene el estudiante.
ID: *username*
- **Mail:** representa un correo entre estudiante y residencia. *Sender_type* denota que rol ha realizado el envío. De igual manera tenemos otra variable para comprobar si lo ha leído la residencia o el estudiante en ese momento.
ID: *id*
- **Incidence:** representa una incidencia generada por un estudiante hace una residencia. Es necesaria una fotografía (formato imagen). La incidencia tiene varios estados: [*OPEN*, *IN PROGRESS*, *DONE*].
ID: *id*
- **Rent:** representa el pago realizada cada mes desde el estudiante a la residencia. De forma automática el estudiante recibirá una notificación sobre el pago (*status_paid*: False) y cuando lo pague se actualizará la fecha de pago y (*status_paid*: True) y se añadirá el identificador de la transacción. Se creará un archivo con el resguardo del pago.
ID: *id*
- **Room:** representa la habitación de una residencia. La habitación debe tener un nombre que la identifique en la residencia. Esta debe tener un equipamiento adjunto. Entre el equipamiento contiene las características de la habitación y las imágenes de esta.
ID: *id*
- **Agreement:** es el contrato entre residencia y estudiante por una habitación entre dos fechas (*date_start_school* - *date_end_school*) por un precio mensual. *File_agreement* es el contrato firmado por ambos.
ID: *id*

- **Bid:** representa la apuesta de cada usuario sobre una habitación que se ofrece. Las apuesta se realiza por una habitación para dos fechas (*date_start_school* - *date_end_school*). El día de la puja, se ofrece la habitación al usuario con más puntos (antigüedad) que ha pujado por ella.
 ID: *id*
- **ResponsiblePerson:** esta representa el conjunto de empleados de la residencia con los que se puede contactar. Almacenemos el DNI del trabajador, además de su correo.
 ID: *id*
- **Bank:** este representa la cuenta bancaria donde se realiza la transferencia cada mes. La residencia puede tener varias cuentas bancarias pero solo una activada.
 ID: *id*

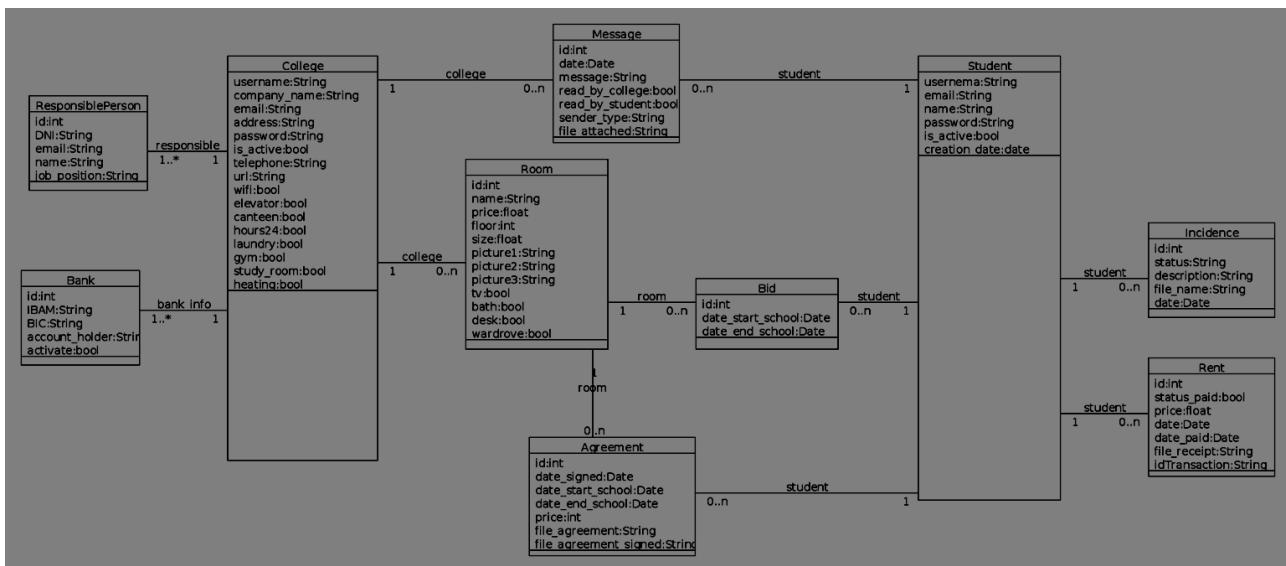


Figure 11 Diagrama de clases

Como podemos imaginar el diagrama de clases ha ido refinándose y mejorándose a lo largo del proceso de implementación. A continuación, vamos a mostrarle los grandes cambios realizados:

- Partimos de un diseño en el cual la mayoría de los módulos (incidencias, pagos, habitación, contrato, mensaje) estaban relacionados de forma directa con la clase Estudiante y la clase Residencia. Este proceso se ha actualizado, de esta manera, la residencia tiene habitaciones; por otro lado el estudiante realiza pagos y presenta incidencias. Ahora la única relación entre estudiante y residencia es el contrato. Así desde la residencia a través del contrato podemos conocer los pagos y las incidencias. Y al contrario desde el estudiante a través del contrato podemos conocer la habitación.
- Por otro lado, el equipamiento de la habitación o de la residencia estaba en otra clase, ahora este está alojado en la propia clase residencia o habitación.
- Respecto a la clase pagos, en un principio este alojaba información relacionada con la tarjeta con la que se realiza el pago, después nos dimos cuenta que estos datos no debían almacenarse, así que actualmente solo guardamos el identificador de la transacción recibido del TPV.

- Uno de los mayores cambio el cual cambio bastante el diseño fue:
 - Inicialmente las fechas de las pujas y del contrato estaban fijadas por la residencia y se almacenaban en la habitación. De esta manera los estudiantes pujaban durante un periodo y tenían la habitación asignada durante otro.
 - Actualmente las pujas se realizan cada semana y los estudiantes tienen flexibilidad para decidir cuánto quieren que dure el contrato académico. Este es elegido a la hora de realizar la puja y este es después fijado en el contrato.
- Para saber si un contrato está firmado necesitamos el archivo firmado por el usuario y la fecha de hacerlo. Estos datos no estaban definidos inicialmente.
- Otro de los datos añadido al final, es el BIC en la clase *Bank*, con este dato podemos saber dónde realizar el ingreso junto el IBAM.

Diagrama de secuencia

Este es un diagrama UML, el diagrama de secuencia, tiene como objetivo describir como los distintos objetos operan para realizar una tarea, además de especificar el orden en que se realizan dichas acciones. Representando el conjunto clases y objetos que intervienen y sus asociaciones. Son también conocidos como diagrama de eventos.

Entre los beneficios de su uso podemos destacar:

- Entender cómo interactúan los componentes
- Que objetos son necesarios para realizar una tarea.
- Identificar el orden de las llamadas, puesto que el diagrama muestra el tiempo en eje vertical, de arriba a abajo.

En este apartado vamos a describir los principales diagrama de secuencia de nuestro sistema, es decir, los principales casos de uso:

- Diagrama actividad: enviar mensaje. Este representa como enviar un mensaje a todos a un estudiante, a través de un método de la API y como el servidor se comporta.

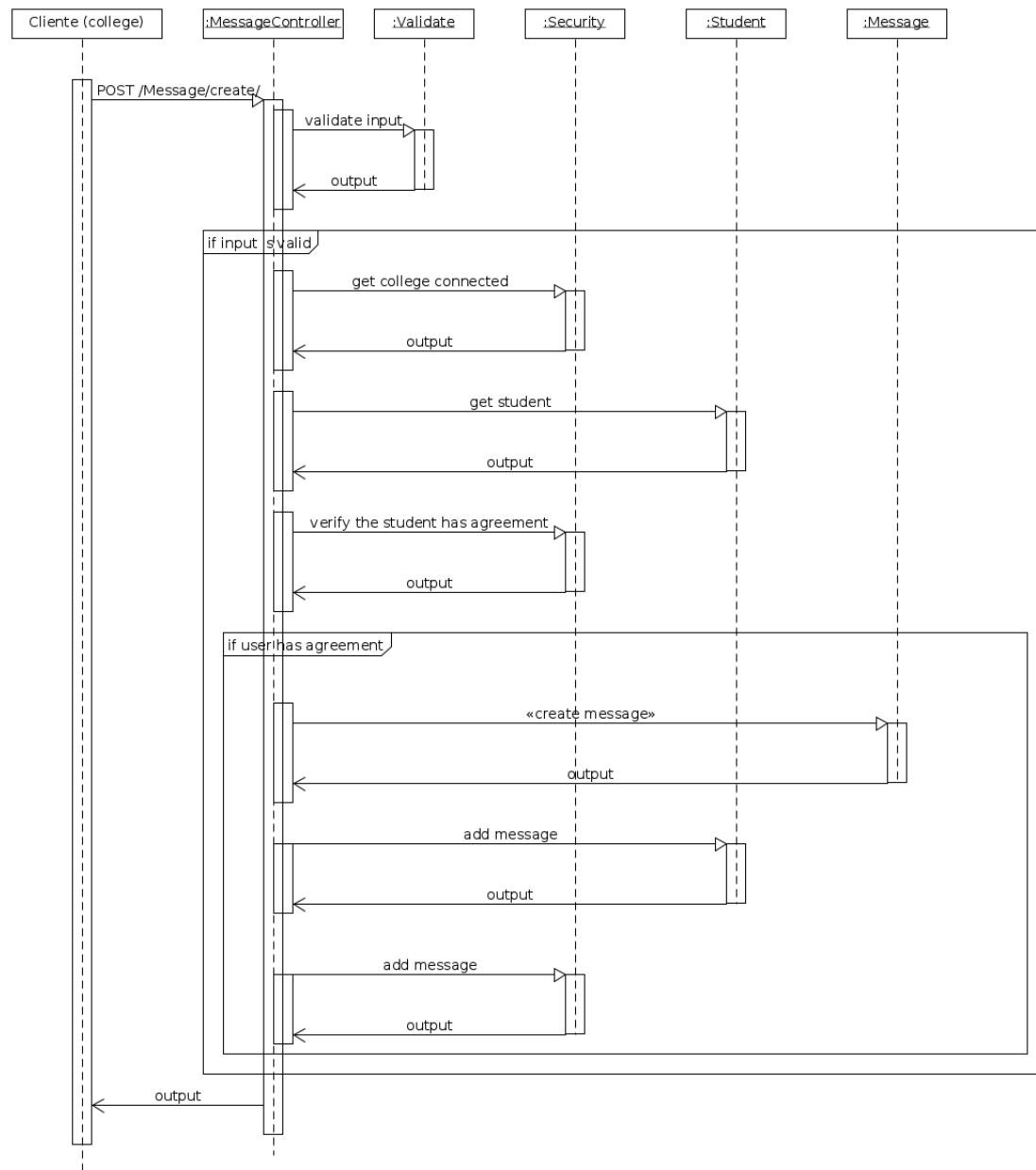


Figure 12 diagrama secuencia enviar mensaje

- Diagrama de secuencia de la actividad pujar por una habitación: este representa como un usuario estudiante puede pujar por una residencia.

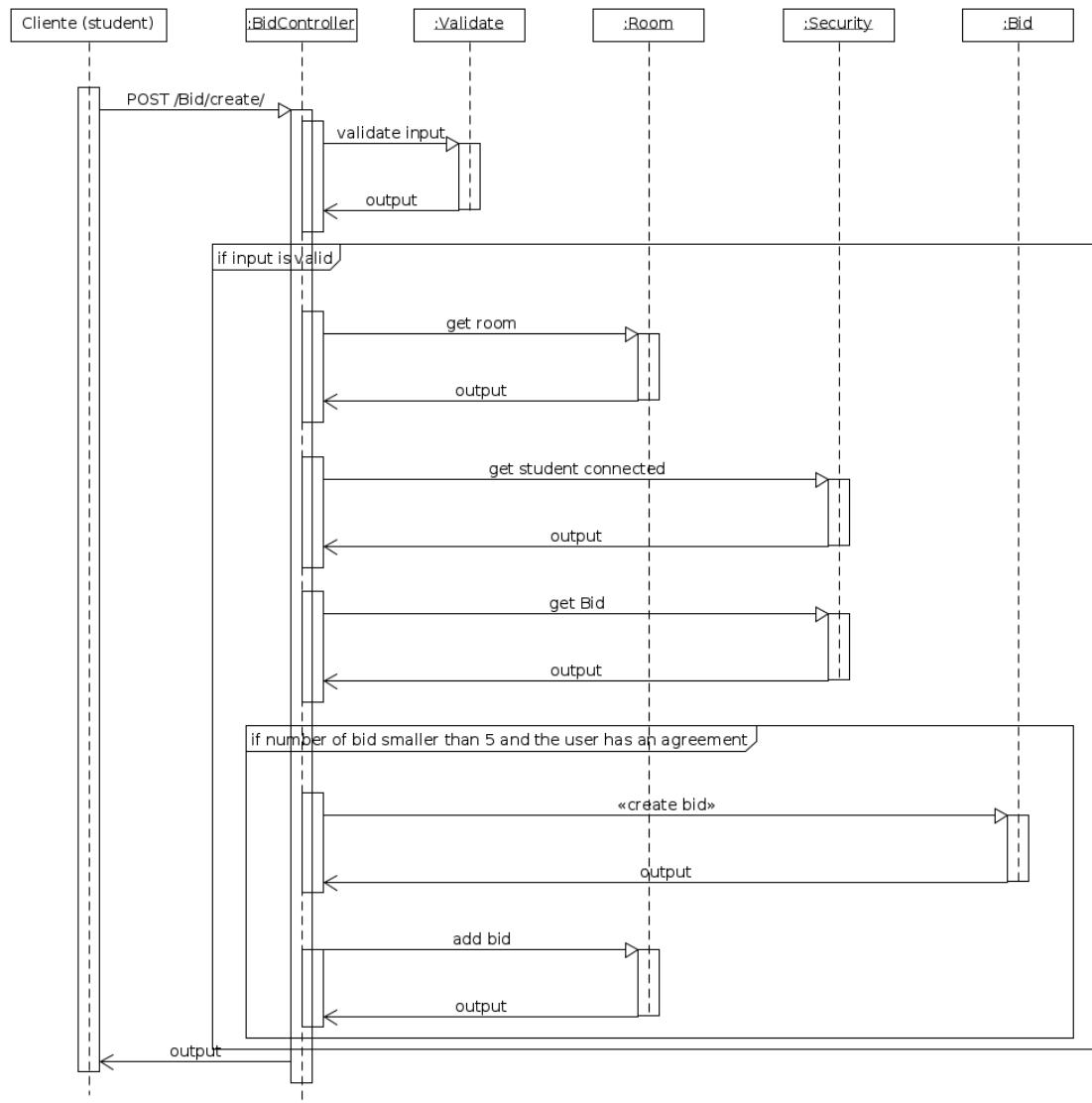


Figure 13 Diagrama secuencia pujar por una habitación

- Diagrama de secuencia de la actividad pagar una mensualidad, este diagrama de actividad describe como un usuario estudiante puede pagar una mensualidad de su contrato con la residencia, pasando a través del módulo TPV y como este hace una llamada a la API.

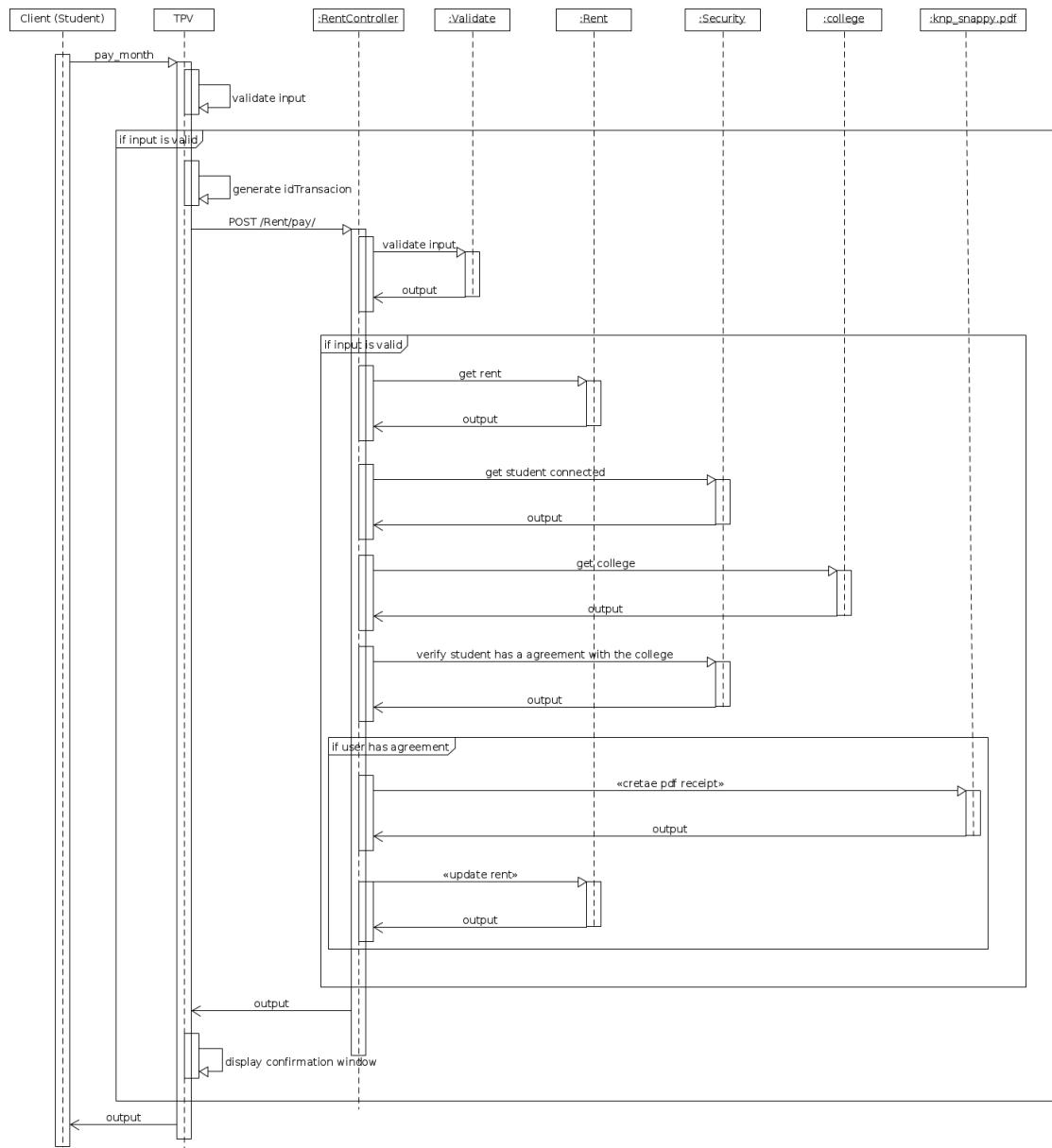


Figure 14 Diagrama secuencia pagar mensualidad

- Diagrama de secuencia de la actividad creación de una incidencia.

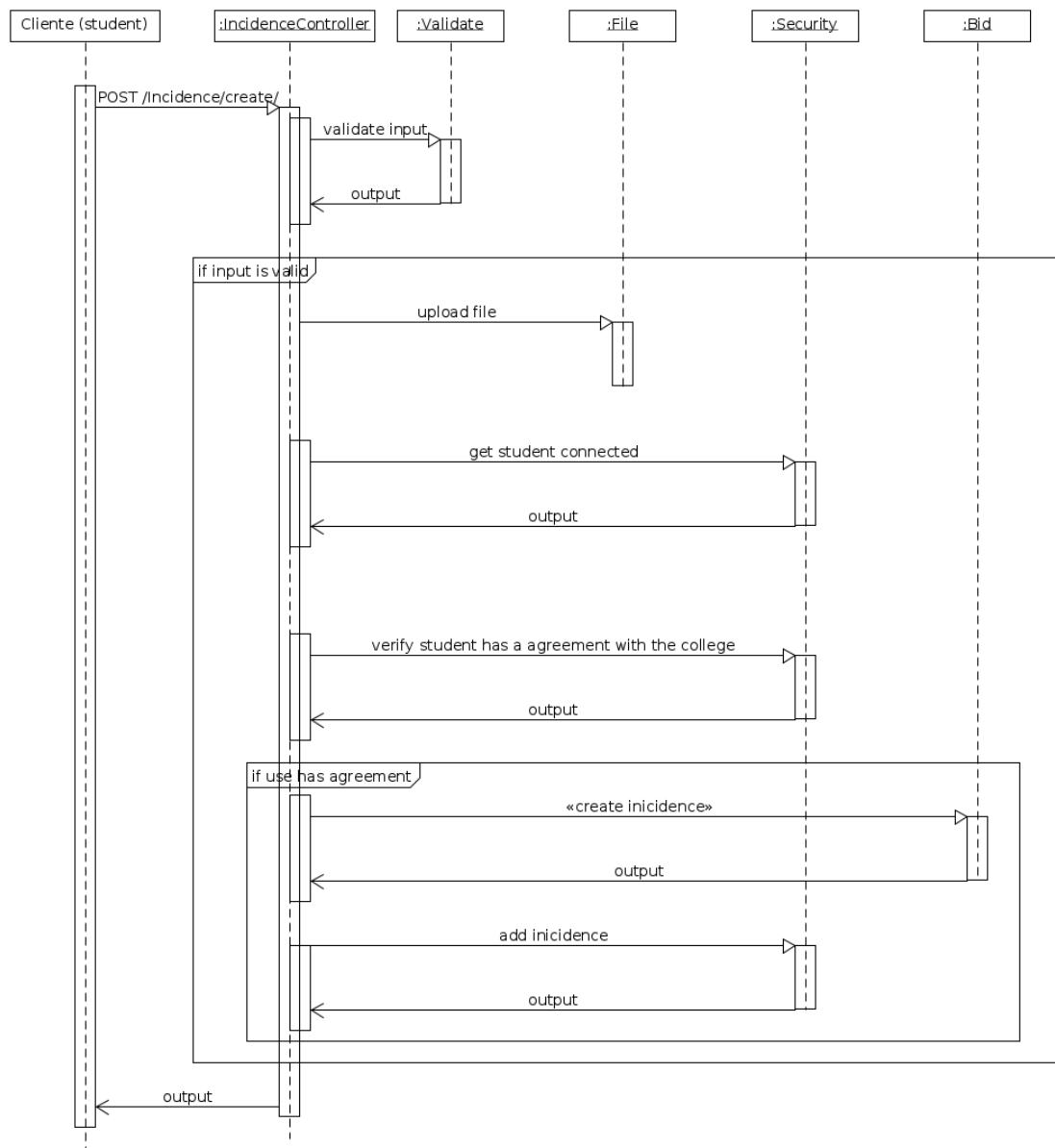


Figure 15 Diagrama de secuencia Crear una incidencia

Diagrama de actividad

Diagrama de actividad tiene como objetivo describir un proceso de negocio, el flujo de trabajo entre los usuarios y los componentes del sistema. Este está formado por secuencias de actividades para llegar a cabo un objetivo. En este apartado vamos a describir los principales diagrama de actividad de nuestro sistema, es decir, los principales casos de uso:

- Diagrama actividad: enviar mensaje. Este representa como enviar un mensaje a todos los estudiantes, en este encontramos una sucesión de acciones y un nodo de decisión. Además podemos ver como el objeto mensaje va creándose desde seleccionar mensaje hasta enviarlo.

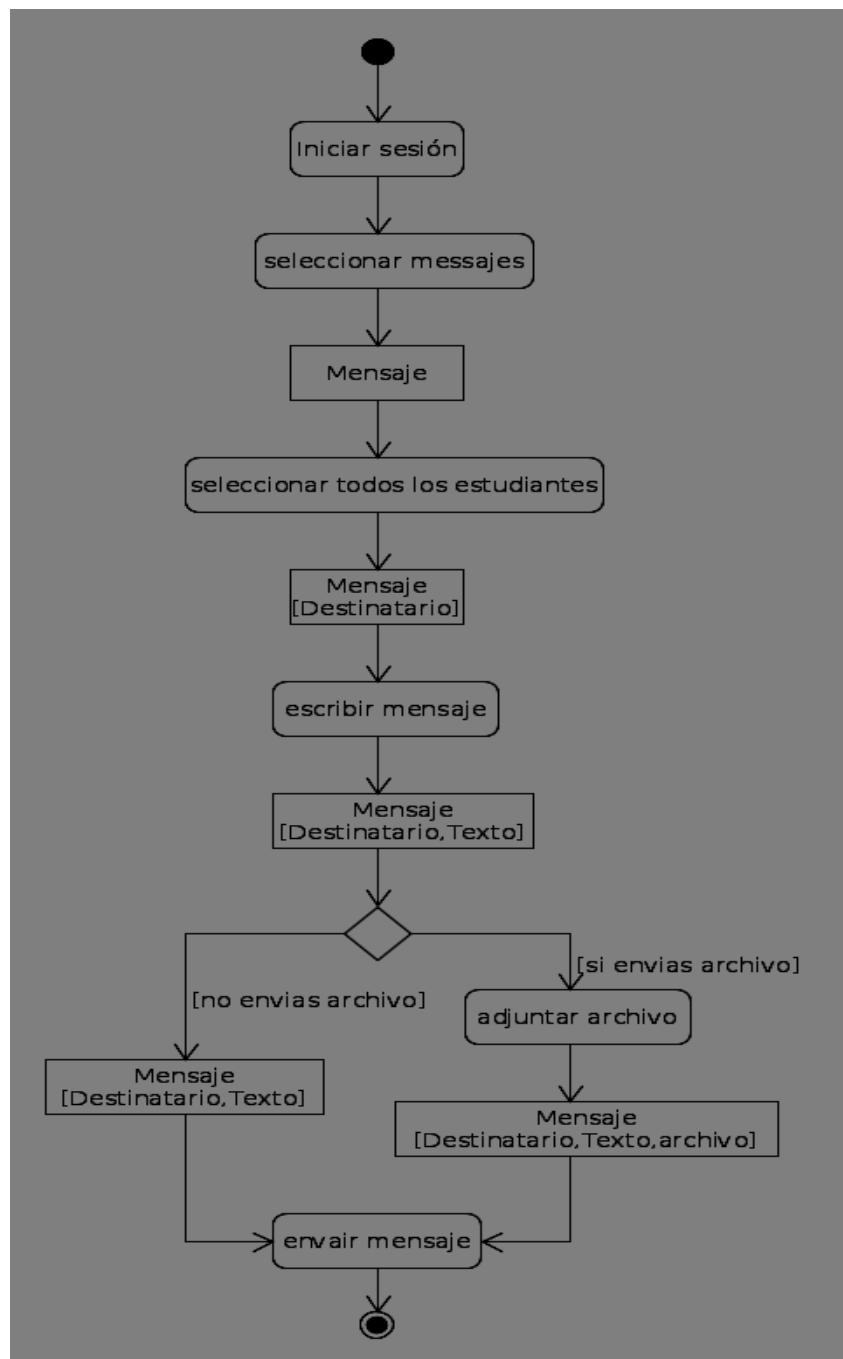


Figure 16 Diagrama de actividad enviar mensaje

- Diagrama de actividad pujar por una habitación: este representa como un usuario estudiante puede pujar por una residencia.

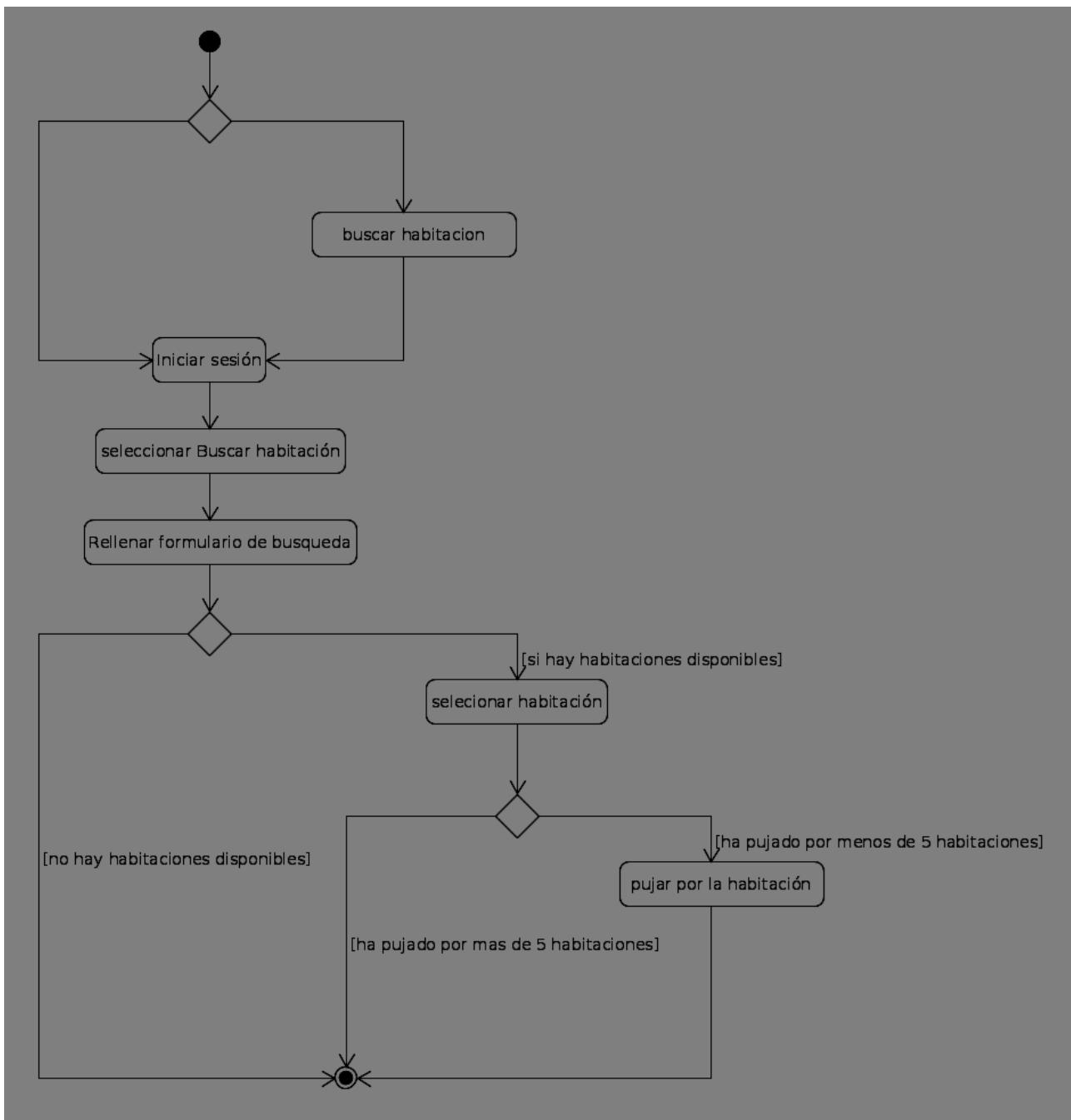


Figure 17 Diagrama de actividad pujar por una habitación

- Diagrama de actividad pagar una mensualidad, este diagrama de actividad describe como un usuario estudiante puede pagar una mensualidad de su contrato con la residencia.

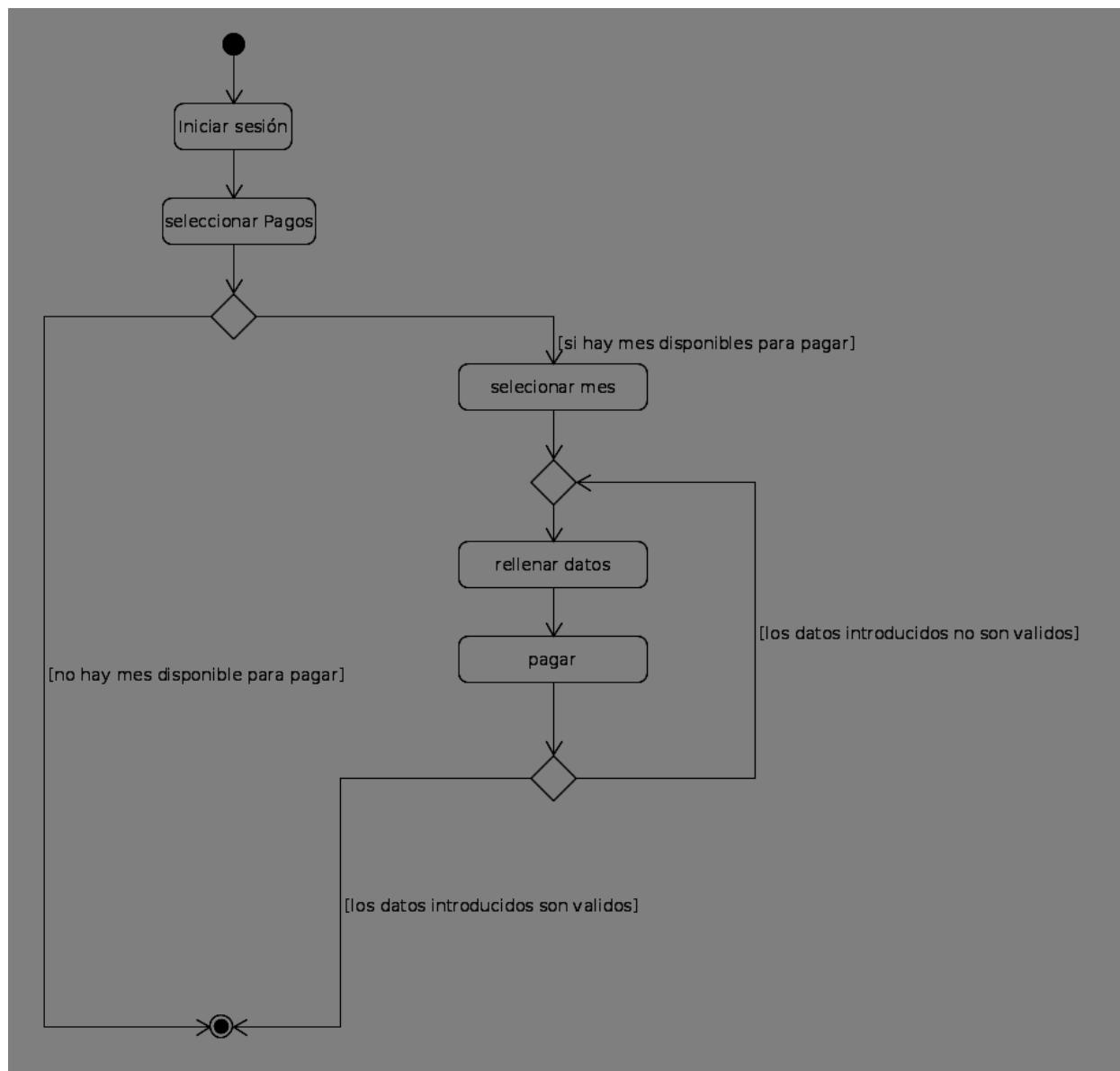


Figure 18 Diagrama de actividad pagar una mensualidad

- Diagrama actividad sobre creación de una incidencia

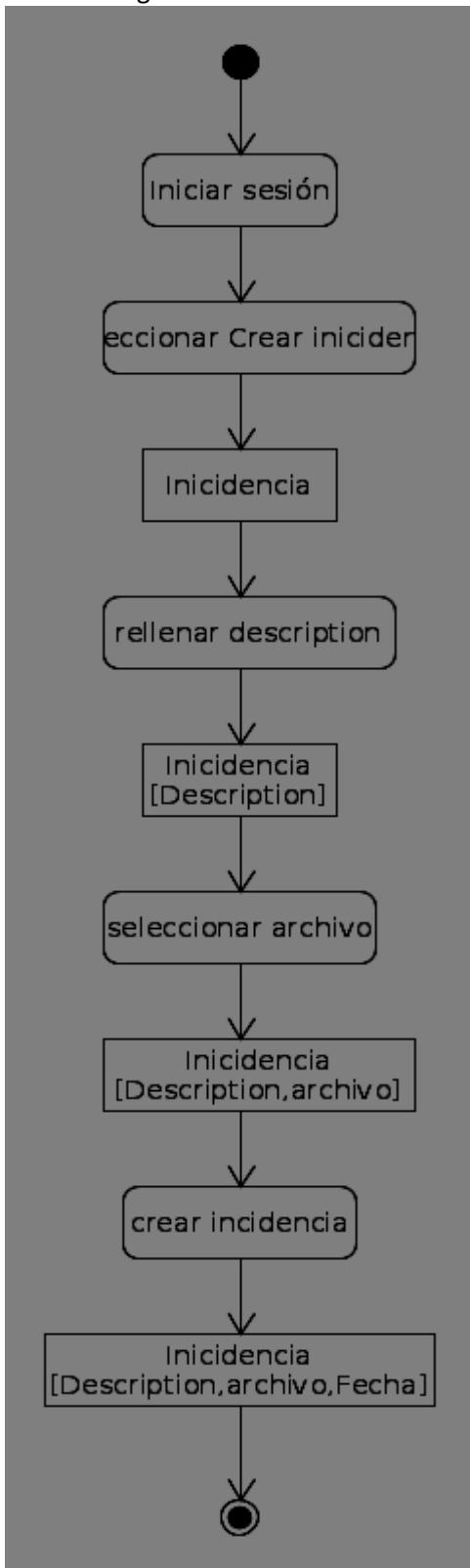


Figure 19 Diagrama de actividad crear una incidencia

Diagrama IFML

Respecto al diseño web, un estándar que se sigue es el diagrama IFML (Interaction Flow Modeling Lenguaje) (Object Management Group, Inc., 2015). Este es el sucesor de WebML. Este aparece en 2014 y sirve para modelar el flujo de interacción: qué información se le muestra al usuario y de qué manera, qué puede hacer el usuario y cuáles son las consecuencias. Respecto la arquitectura *modelo vista controlador*, podemos destacar:

- Vista: este diagrama representa como las vistas están compuestas y muestra el contenido de ellas.
- Controlador: IFML muestra la interacción del usuario y los eventos generados. Es decir, las ordenes que el usuario puede dar. Además de los parámetros necesarios para la interacción.
- Modelo: muestra los elementos de datos que puede ver el usuario.

Como hemos dicho anteriormente este es el estándar web aprobado por el OMG (Object Management Group).

Entre los elementos de este diagrama podemos destacar:

- Ítems atómicos: entidades de datos.
- Ítems compuestos: formados por varios ítems atómicos.
- Estructura contextuales: estructuras de navegación para acceder a ítems.

Una de las características principales de IFML es que es independiente de la plataforma. Es decir puede funcionar igual para interfaces web o de escritorio o de móvil.

A continuación, mostraremos el diagrama IFML de las principales vistas de nuestra web.

- En primer lugar, mostraremos la ventana principal y nos enfocaremos en la búsqueda de habitaciones.

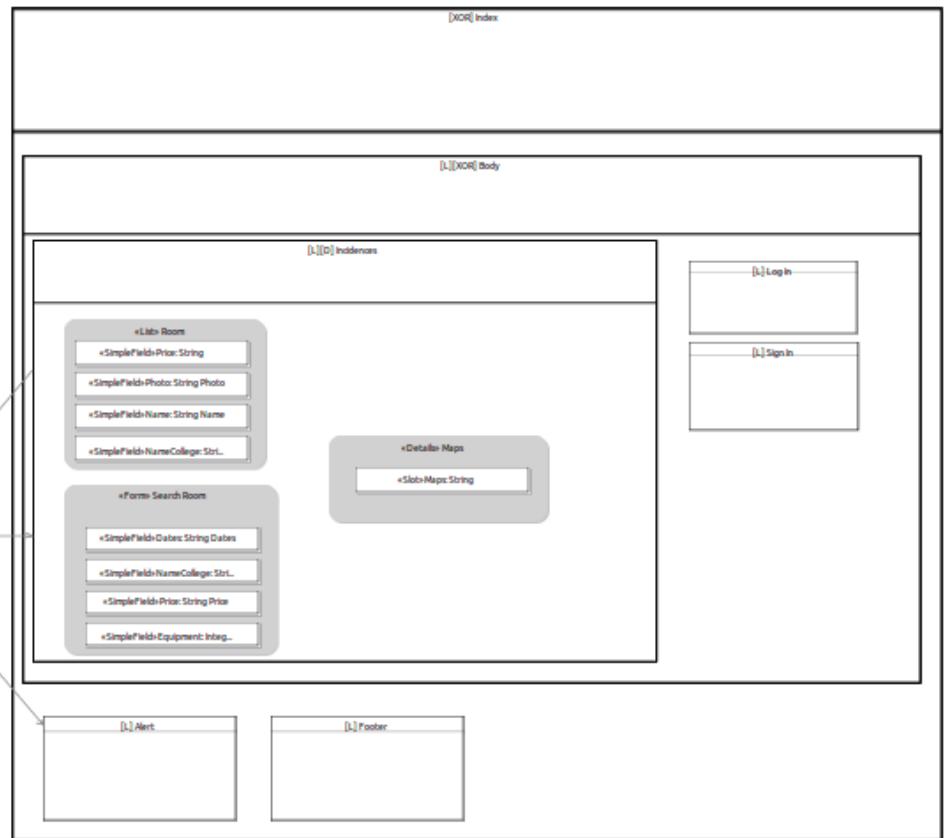


Figure 20 Diagrama IFML Search

- En este diagrama mostraremos la vista de la lista de incidencias.

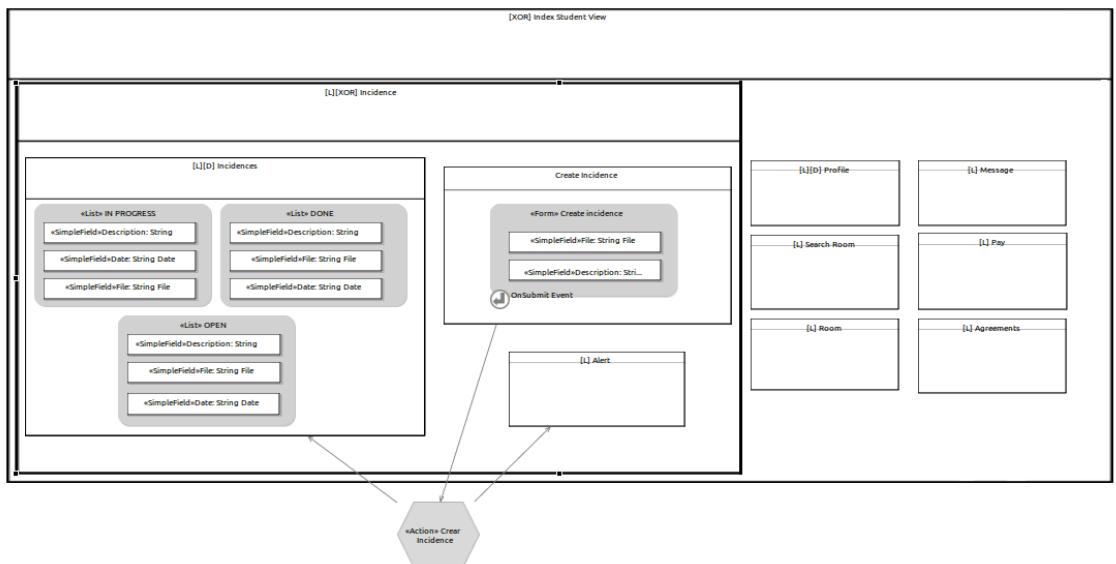


Figure 21 Diagrama IFML Incidencias

- Este diagrama muestra la vista de mensajes, donde mostramos la lista de mensajes y el formulario para enviar uno.

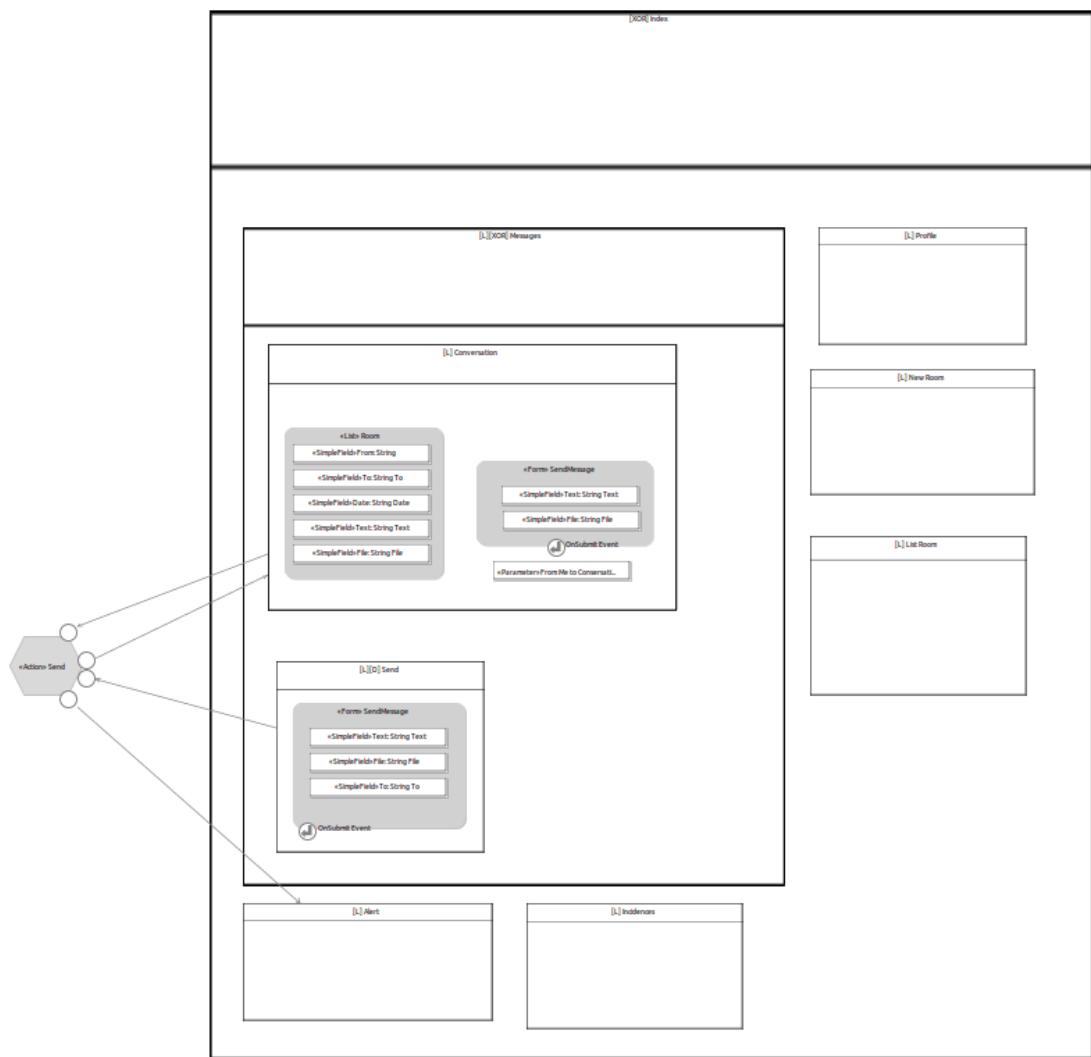


Figure 22 Diagrama IFML Message

Implementación

Tecnología utilizada

En este apartado explicaremos las tecnologías utilizadas, el porqué de su uso, la curva de aprendizaje necesaria y los estándares seguidos. Igualmente hablaremos del control de versión utilizado. La aplicación está dividida entre servidor o *back end* y el cliente o *front end*. A la hora de explicar que herramientas hemos utilizado, las clasificaremos por esta división.

Back end

Este es el servidor de nuestro proyecto donde encontramos la API, la base de datos y automatización de la asignación de habitaciones. Entre las tecnologías utilizadas destacamos. *Symfony* (*PHP*) para la API, *MySQL* para la base de datos y *Python* para la automatización. Una API es un conjunto de métodos bien definidos que son ejecutados en el servidor. Esta sigue una arquitectura software llamada *RestFul*, la cual está basada en el protocolo HTTP. Este se basa en su independencia de plataforma y lenguaje. Entre los métodos HTTP hemos usado *POST* y *GET*.

Ha sido una decisión difícil a la hora de elegir estas herramientas. Puesto que habíamos hecho antes otras APIs con otras tecnologías como *Django* o *Flask*, las cuales utilizan *Python*, un lenguaje fácil de aprender y flexible. Estas tecnologías son utilizadas por grandes empresas como Google, YouTube, Nokia, etc.

A continuación describimos cada herramienta:

- *Symfony*: es utilizado en el lado del servidor para crear la API. Este utiliza *PHP* como lenguaje. No lo habíamos utilizado antes por lo cual me costó elegirlo, aunque después de ver toda la documentación que tenía y es usado por un alto número de desarrolladores actualmente. Además este utiliza un diseño orientado a objetivos. Este *framework* tiene muchos útiles componentes los cuales pueden ser utilizados.

A la hora de una API, esta debe ser rápida y tener un buen rendimiento; lo cual está cubierto por *Symfony*. Además este es flexible, es decir, puede adaptarse a tus necesidades. A continuación puedes ver los métodos de la API.

POST	/Agreement/accept/	This method accept a Agreement between a student and a room. Can be called by user (student).
POST	/Agreement/assignedRooms/	This method assigned offered room to a student (with more point) the day of the bid. Can be called by user (ADMIN) automatically.
POST	/Agreement/create/{room_id}/{username}/{bid_id}	This method create a Agreement between a student and a room. Can be called by user(ADMIN) automatically.
GET	/Agreement/download/{filename}	Download agreement file. Can be called by user (Student/College/ADMIN).
GET	/Agreement/getCurrentSigned/	Get the current agreement signed. That function is called by a user (student).
GET	/Agreement/getList/	Get list of agreements. That function is called by a user (student).
POST	/Agreement/remove/	This method remove a Agreement between a student and a room (the agreement which is valid according to the dates). Can be called by user (Student).
POST	/Agreement/removeUnsigned/	This method remove a Agreement which is not signed for one week. Can be called by user (ADMIN).
GET	/Agreement/roomVerifyUnsigned/{room_id}	Verify if a room has a agreement without signed. Return the agreement. Can be called by user (College).
POST	/Bank/activate/{id}	This method activate a bank account of the college. Can be called by user (College).
POST	/Bank/create/	This method create a bank account for a college. Can be called by user (College).
GET	/Bank/get/	Get list of banks of a user (College). Can be called by user (College).
POST	/Bank/remove/{id}	This method remove a bank account of the college. Can be called by user (College/ADMIN).
POST	/Bid/create/	This method create a bid by the user (Student). An user cannot bid more than 5 times neither bid twice for the same room. Can be called by user (Student).
GET	/Bid/get/{id}	Get data of a bid. Can be called by user (College/Student/ADMIN).
GET	/Bid/getBidsRoom/{id}	Get all the bid of a room by its id. Can be called by user (College/Student/ADMIN).
POST	/Bid/remove/{id}	Remove a bid. Can be called by user (College/Student/ADMIN).
POST	/Bid/removeBidRoomStudent/	Remove the bid of a user (student) above a room by its id. Can be called by user (Student).
POST	/Bid/removeBidsRoom/{id}	Remove all the bids of a room by its id. Can be called by user (College/ADMIN).

Trabajo Fin de Grado Ingeniería Informática: Puja Por Tu Resi

Antonio Jiménez Martínez

<code>POST</code>	<code>/Bid/removeBidsStudent/{username}</code>	Remove all the bids of a student by username. Can be called by user (ADMIN) automatically.
<code>POST</code>	<code>/Incidence/create/</code>	This method create a incidence. Can be called by user (Student).
<code>GET</code>	<code>/Incidence/download/{filename}</code>	Download file of the incidence. Can be called by user (College/Student).
<code>GET</code>	<code>/Incidence/get/</code>	Get list of incident of a user in a JSON format. Can be called by user (Student/College).
<code>GET</code>	<code>/Incidence/getNumberOpen/</code>	Get number of incident In status OPEN. Can be called by user (College).
<code>POST</code>	<code>/Incidence/updateState/</code>	This method update the state of a incidence. Can be called by user (College/ADMIN).
<code>GET</code>	<code>/Message/countUnread/</code>	Get number of unread message of a user. Can be called by user (College/Student).
<code>GET</code>	<code>/Message/countUnreadStudent/</code>	Get number of unread message from all the student of a college. Can be called by user (College).
<code>POST</code>	<code>/Message/create/</code>	This method create a message by the user (Student/COLLEGE). Can be called by user (College/Student).
<code>GET</code>	<code>/Message/download/{filename}</code>	Download attached file of the message. Can be called by user (College/Student).
<code>GET</code>	<code>/Message/get/</code>	Get list of messages of a user (Student College). Can be called by user (College/Student).
<code>POST</code>	<code>/Message/openAll/</code>	This method set ReadBy=true of a all the messages of user. Can be called by user (College/Student).
<code>POST</code>	<code>/Message/openStudent/{username_student}</code>	This method set ReadByCollege=true of a all the messages of user with a specific student. Can be called by user (College).
<code>GET</code>	<code>/ProfileCollege/get/</code>	Get data of the user (College). Can be called by user (College).
<code>GET</code>	<code>/ProfileCollege/getStudentComplete/{username_student}</code>	Get student information get: list_rents, room, agreement, student_data. Can be called by user (College).
<code>GET</code>	<code>/ProfileCollege/getStudents/</code>	Get all the student of a college. Can be called by user (College).
<code>GET</code>	<code>/ProfileCollege/getStudentsComplete/</code>	Get all the student of a college. For every student get: list_rents, room, agreement, student_data. Can be called by user (College).
<code>POST</code>	<code>/ProfileCollege/updateAddress/</code>	This method update the address of a user (College). Can be called by user (College).
<code>POST</code>	<code>/ProfileCollege/updateEmail/</code>	This method update the email of a user (College). Can be called by user (College).
<code>POST</code>	<code>/ProfileCollege/updateEquipment/</code>	This method update the equipment of a user (College). Can be called by user (College).
<code>POST</code>	<code>/ProfileCollege/updatePassword/</code>	This method update the password of a user (College). Can be called by user (College).
<code>POST</code>	<code>/ProfileCollege/updateProfile/</code>	This method update the email,telephone,url and equipment of a user (College). Can be called by user (College).
<code>POST</code>	<code>/ProfileCollege/updateTelephone/</code>	This method update the telephone of a user (College). Can be called by user (College).
<code>POST</code>	<code>/ProfileCollege/updateURL/</code>	This method update the URL of a user (College). Can be called by user (College).
<code>GET</code>	<code>/ProfileStudent/get/</code>	Get data of the user (student) : name, username, email, ROLE, date_creation, point. Can be called by user (Student).
<code>POST</code>	<code>/ProfileStudent/updateEmail/</code>	This method update the email of a user (student). Can be called by user (Student).
<code>POST</code>	<code>/ProfileStudent/updatePassword/</code>	This method update the password of a user (student). Can be called by user (Student).
<code>POST</code>	<code>/Rent/createAll/{username_student}</code>	This method create all the rents of a user during his agreement. One per moth. Can be called by user (ADMIN) automatically
<code>GET</code>	<code>/Rent/download/{filename}</code>	Download receipt file. Can be called by user (Student/College).
<code>GET</code>	<code>/Rent/get/</code>	Get list of rents of a user. Can be called by user (Student/College).
<code>GET</code>	<code>/Rent/getReceiverBankAccount/</code>	Get the activate bank account of the college. Can be called by user (Student).
<code>GET</code>	<code>/Rent/getStudent/{username_student}</code>	Get list of rent of a user (Student). In JSON format. Can be called by user (College).
<code>GET</code>	<code>/Rent/getUnpaid/</code>	Get list of rents without pay of a user (Student). Can be called by user (Student).
<code>POST</code>	<code>/Rent/pay/</code>	Pay the rent. * This method generate file_receipt. * update the date_paid and set the cardNumber and cardHolder. Can be called by user (Student).
<code>POST</code>	<code>/ResponsiblePerson/create/</code>	This method create a ResponsiblePerson for a college. Can be called by user (College).
<code>GET</code>	<code>/ResponsiblePerson/get/</code>	Get list of responsiblePersons of a user (College). Can be called by user (College).
<code>POST</code>	<code>/ResponsiblePerson/remove/{DNI}</code>	This method remove a responsible person of the college. Can be called by user (College/ADMIN).
<code>POST</code>	<code>/Room/create/</code>	This method create a room for a College. Can be called by user (College).
<code>POST</code>	<code>/Room/create/</code>	This method create a room for a College. Can be called by user (College).
<code>GET</code>	<code>/Room/download/{filename}</code>	Download pictures rooms. Can be called by user (College/STUDENT/ADMIN).
<code>GET</code>	<code>/Room/get/{id}</code>	Get room of the id of a user (College). Can be called by user (College).
<code>GET</code>	<code>/Room/getAll/</code>	Get list of rooms of a College. Can be called by user (College).
<code>GET</code>	<code>/Room/getAllCompanyName/</code>	Get the companyName of all the colleges. This function can be called by User (College/Student/ADMIN).
<code>GET</code>	<code>/Room/getSearch/</code>	Get all the colleges with the data of college and all the OFFERED room. The college and the room should pass the restrictions: price, equipment, specific_college. This function can be called by User (College/Student/ADMIN)
<code>GET</code>	<code>/Room/getSearchAll/</code>	Get all the colleges with the data of college and all the OFFERED room. This function can be called by User (College/Student/ADMIN).
<code>POST</code>	<code>/Room/remove/{id}</code>	Remove room of the id of a user (College). Can be called by user (College).
<code>GET</code>	<code>/Security/checkSesion/</code>	This method verify if a user (which role) is connected in the system. Can be called by user (College/Student).
<code>POST</code>	<code>/Signin/college/</code>	This method sign up a user (College) in the system. It is not necessary to be authenticate.
<code>POST</code>	<code>/Signin/student/</code>	This method sign up a user (Student) in the system. It is not necessary to be authenticate.
<code>GET/POST</code>	<code>/login</code>	This method allow to a user to login the systme. Can be called by user (College/Student).
<code>GET/POST</code>	<code>/remmemberPassword</code>	This method allow to a user to remmber the password the systme. Can be called by user (College/Student).

Figure 23 Conjunto de métodos de la PIA

Este *framework* proporciona una estructura para los elementos más comunes además de seguir un paradigma de Modelo Vista Controlador (MVC), este provee las herramientas para crear el controlador y el modelo. Por otro lado la vista es creada en el cliente a partir de los datos obtenidos por la API.

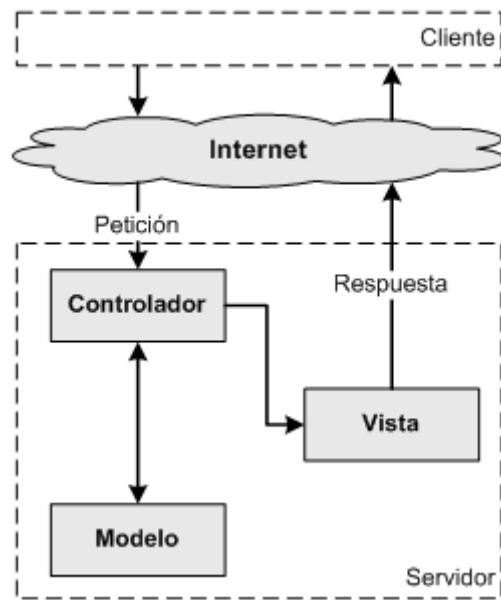


Figure 24 Modelo Vista Controlador Symfony

Al ser la primera vez en usarlo, la curva de aprendizaje fue dura, ya que es un *framework* con muchos componentes y una gran estructura por lo cual fue difícil de usar al principio, aunque ahora se ha convertido en una herramienta habitual.

Al principio no quise utilizar los paquetes o componente que ofrecía y los implemente yo mismo para empezar a conocerlo *framework*, por ejemplo el módulo de usuarios o de mensajería o de pagos. Aunque por otro lado he utilizado varios módulos de *Symfony* como:

- Módulo de seguridad: para iniciar y cerrar sesión y almacenar el *token* y rol del usuario. Además para que ciertos métodos de la API pueden ser llamados por específicos roles de usuario nada más.

```
providers:
    our_db_provider:
        chain:
            providers: [ in_memory, college_our_db_provider, student_our_db_provider]
    college_our_db_provider:
        entity:
            class: AppBundle:College
            property: username
    student_our_db_provider:
        entity:
            class: AppBundle:Student
            property: username

# if you are using multiple entity managers
# manager_name: customer
# it is used to specify what request need authentication or not
firewalls:
    main:
        anonymous: ~
        form_login:
            login_path: login
            check_path: login
            default_target_path: login
            always_use_default_target_path: true
        logout:
            path: /logout
            target: /login
```

Figure 25 Screenshot seguridad

- Módulo base de datos: este es utilizado para la creación de base de datos y tablas. Además tiene métodos para leer, actualizar y añadir datos a la base de datos. Este permite hacer peticiones a la base de datos en un lenguaje DDL.
- Módulo para almacenar archivos en el servidor. Desde el cliente solo tenemos que añadir el archivo y el servidor verifica el formato y el tamaño. Y lo renombra y lo guarda en un directorio especificado en los atributos. Este ha sido beneficioso para adjuntar un archivo a los mensajes y a las incidencias.
- Módulo para crear pdf a través de una plantilla en TWIG. Este ha sido útil para la creación del contrato y los recibos. Este es llamado *KnpSnappyBundle*.

```
/**-
 * Create the receipt file by the date of the college, student, and the current rent.
 * Using knp snappy and twig to generate a pdf file.
 * The name of the user is random.
 */
public function create_receipt($college_data,$student_data,$rent_data,$college_bank_account)
{
    $filename=md5(uniqid()).'.pdf';
    $this->get('knp_snappy.pdf')->generateFromHtml(
        $this->renderView(
            'payment_receipt.html.twig',
            array(
                'rent' => $rent_data->getJSON(),
                'student' =>$student_data->getJSON(),
                'college' =>$college_data->getJSON(),
                'college_bank_account' =>$college_bank_account->getJSON(),
            )
        ),
        $this->container->getParameter('storageFiles')."/".$filename
    );
    return $filename;
}
```

Figure 26 Screenshot Modulo crear pdf con una plantilla

- Módulo de documentación, llamado *NelmioApiDocBundle*. Para cada método de la API es creado una descripción y los parámetros de entrada y la salida.

```
/**-
 * @ApiDoc(
 *     description="Pay the rent."
 *     This method generate file_receipt.
 *     update the date_paid and set the cardNumber and cardHolder. Can be called by user (Student).",
 *     requirements={
 *         {
 *             "name"="id",
 *             "dataType"="integer",
 *             "description"="ID of the rent"
 *         },
 *         {
 *             "name"="idTransaction",
 *             "dataType"="string",
 *             "description"="idTransaction of the simulator TPV"
 *         }
 *     }
 * )
 */

```

Figure 27 Screenshot Modulo de documentación

- Módulo validación tipo datos. Este es utilizado para validar el correo, el teléfono, la contraseña, la url, el tamaño de entrada, el número de cuenta, el BIC y demás. Este ha sido utilizado para validar datos de entrada, sobretodo registrar usuarios.
- Módulo llamado *CloudBackupBundle* para realizar un *backup* automático de la base de datos.

```
dizda cloud backup:
    # By default backup files will have your servers hostname as prefix-
    # such as: hostname_2014-01-01_21-08-39.tar-
    output_file_prefix: hostname-
    timeout: 300-
    restore: false # Set to true to enable restore command-
    processor:
        type: zip # Required: tar|zip|7z-
    cloud_storages:
        # Local storage definition-
        local:
            path: '%backup_dataBase_folder%-
        databases:
            mysql:
                all_databases: false # Only required when no database is set-
                database: "%database_name%" # Required if all_databases-
                db_host: "%database_host%" # This, and following is not requir-
                db_port: "%database_port%" # Default 3306-
                db_user: "%database_user%"-
                db_password: "%database_password%"
```

Figure 28 Screenshot CloudBackupBundle

Seguimos un estándar de programación definido por Symfony en el cual podemos destacar:

- Los comentarios:

```
/***
     * Descripción de la función
     *
     * @param $nombre Descripción parámetro
     *
     * @return tipo devuelto y descripción del elemento devuelto.
*/
```

- Añadir una línea en blanco antes del *return*.
- Declarar los atributos de la clase lo primero.
- Un estándar en el resultado de cada método de la API. El elemento de salida es un “JSON” el cual tiene siempre la misma estructura:
 - Success: si la operación ha funcionado correctamente
 - Message: en el caso de error: el mensaje describiendo el error.
 - Data: en el caso de no error: el resultado obtenido.
- MySQL: es un gestor de base de datos. Este es un componente de *LAMP* (Linux, Apache, MySQL, PHP). Hemos utilizado este gestor y no otros como *MongoDB* o *SQLite* o *MariaDB*, puesto que este es sencillo y cómodo de utilizar con *PHP* y *Symfony*. Además teníamos experiencia en su uso. Este es bastante escalable, rápido y tiene buen rendimiento.
Para gestionarlo de forma manual hacemos uso de una aplicación, llamada,

phpMyAdmin. Donde podemos crear, actualizar y eliminar bases de datos. Aunque el mayor uso se lo doy para exportar, al hacer un *backup* de los datos, e importar para usar los datos en otros servidores u ordenadores. *PhpMYAdmin* ha sido útil para generar el diagrama entidad relación de la base de datos automáticamente.

- *Python*: es usado para la automatización. Es muy popular y usado hoy en día para este servicio. No es necesario compilarlo, para poder ejecutarlo. Además puede ser ejecutado, sin depender del sistema operativo. A través de un *script* iniciamos sesión como administrador y asignamos cada habitación al usuario que ha pujado por ella con más puntos. Este *script* es ejecutado cada semana a través de CRON JOB. Este es una herramienta para lanzar un conjunto de comandos con una frecuencia determinada. En definitiva es un archivo con un formato específico para fijar una frecuencia y un comando a ejecutar, como por ejemplo ejecutar un *script*.

```
#login (call twice to login with the cookies)-
response = s.post("http://localhost:8000/login", data = form_data, headers = headers,cookies = n_cookies)
response = s.get("http://localhost:8000/login", headers = headers,cookies = s.cookies)

#API request /Agreement/assignedRooms/
print("remove_unsigned_agreement\n")
response = s.post( "http://localhost:8000/Agreement/removeUnsigned/", headers = headers,cookies = s.cookies)
print(response.text)
print("\n")-
```

Figure 29 Screenshot python script

Front end

Este es el cliente de nuestro proyecto o también llamado capa de presentación, actualmente está hecho en la web con una característica necesario hoy en día, tiene un diseño *responsive* que se adapta a distintos dispositivos como escritorio, móvil o *tablet*. Las tecnologías utilizadas en el lado del cliente son las más utilizadas como *JavaScript* o *Jquery*, *HTML* y *CSS*. A continuación describimos para que las hemos utilizado y porque las hemos elegido:

- *JavaScript*: este es uno de los lenguajes de programación más utilizado en la web actualmente. Este tiene como objetivo hacer dinámica la web, comunicarse con el servidor y con el cliente de la web. Aunque lo más importante es que reacciona ante eventos en la página web como click en un botón o escribir una letra y muchos más. El estándar seguido para la documentación en el código es el siguiente:

- Con este ejemplo mostramos el estandar:

```
/***
 * Descripción de la función
 * @param {type} descripción del parámetro
 * @return { type } descripción del valor a devolver.
 */
```

- Entre los principales uso podemos destacar:

- Comunicarse con el servidor a través de la API, utilizando *AJAX* (*Asynchronous JavaScript and XML*). Este es una tecnología que permite hacer peticiones al servidor. Podemos recibir la respuesta en *XML* o texto. En nuestro caso recibimos las respuestas en texto y la convertimos en *JSON*, el cual es un formato muy utilizado en *JavaScript*.

```
var xmlhttp = new XMLHttpRequest();
var url = window.location.protocol + "//" + window.location.host + port + "/ProfileCollege/get/";
xmlHttp.open("GET", url, true);
xmlHttp.withCredentials = true;
xmlHttp.send();
xmlHttp.onreadystatechange = function() {
    if (xmlHttp.readyState == 4 && xmlHttp.status == 200) {
        var output = JSON.parse(xmlHttp.responseText);
        console.log(output);
        if (output.success) {
            display_specific_college("college_profile_", output.data);
            display_username("tab_profile_college_username", output.data.username);
        } else {
            showErrorMessagesPage("showdata", output.message, output.success);
        }
    }
}
```

Figure 30 Screenshot AJAX

- Gestionar el historial, nuestra página es estática, esta solo se recarga cuando se inicia sesión o se recarga de forma manual. Utilizamos la librería *Pages*, esta nos permite añadirle un atributo a nuestra URL y a través de ella ejecutar un código u otro. Por ejemplo cuando tenemos el etiqueta “/profile” se muestra el perfil del usuario.
- La validación de cada valor de entrada en todos los formularios es realizara a través de JavaScript. Esta sirve para verificar el tamaño el tipo como DNI o correo o teléfono o url y demás.
- A la hora de mostrar la localización utilizamos la API de google maps. Esta ha sido práctico para recoger la latitud y la longitud de una dirección y después mostrarla en el mapa. Ha sido una tarea complicada aunque hemos encontrado bastante documentación para hacerlo.

```
/**-
 * Display the location of the latitude and longitude on map
 * @param {id} id of the div
 * @param {latitude} latitude of the position
 * @param {longitude} longitude of the position
 */
function init_map(id, latitude, longitude) {
    //get latitude,longitude from the college
    var uluru = {
        lat: latitude,
        lng: longitude
    };
    var map = new google.maps.Map(
        document.getElementById(id), {
            zoom: 13,
            center: uluru
        }
    );
    var marker = new google.maps.Marker({
        position: uluru,
        map: map
    });
}
```

Figure 31 Screenshot API google maps

- *Jquery*: es un lenguaje de programación web con el objetivo de simplificar y facilitar el uso de JavaScript. Esta tiene multitud de librerías, nosotros hemos hecho uso de:
 - *floatthead*: el objetivo que tienes es que la cabecera de una tabla se mantenga siempre aunque hagas *scroll* en la tabla.

```
/**-
 * keep in the top the thead of the table
 * @param tab_table
 */
function floatHead_table(tab_table) {
    var $table = $('#' + tab_table + ' table');
    $table.floatThead({
        scrollContainer: function($table) {
            return $table.closest('.wrapper');
        }
    });
}
```

Figure 32 Screenshot Floatthead

- *Bootstrap-select*: esta es utilizada para buscar en una lista y seleccionar uno o todos. Hemos sacado provecho de ella a la hora de seleccionar que usuarios enviar el mensaje.
- *Bootstrap-slider*: es una librería para crear una barrar con animación. En nuestro caso la hemos utilizado para crear los rangos de precio entre un mínimo y un máximo a la hora de buscar una habitación.

```
/**~  
 * Display the range of price~  
 */~  
function display_range_price(id) {~  
    $("#" + id).slider({~  
        range: true,~  
        min: 0,~  
        max: 2000,~  
        values: [75, 1500],~  
        slide: function(event, ui) {~  
            $("#amount").val(ui.values[0] + "€ - " + ui.values[1] + " €");~  
        }~  
    });~  
    $("#amount").val(get_min_range_price(id) + " € - " + get_max_range_price(id) + " €");~  
}
```

Figure 33 Screenshot slider

- *HTML5*: este es un lenguaje de marcas, con el objetivo de mostrarle al navegador su función semántica, a su vez es el nuevo estándar de la W3C. La gran ventaja es que es multiplataforma. Está formado por etiquetas con atributos. Este compone los elementos estáticos de la web, generando un código limpio. Cabe destacar varios elementos que hemos empleado:
 - *Icon*: existe varias fuentes para introducir iconos en cuentas web:
 - Font Awesome Icons
 - Bootstrap Icons
 - Google Icons
 - *Drag and Drop*, este es utilizado para mover elementos de un contenedor a otros. En nuestro caso ha sido utilizado para actualizar el estado de una incidencia moviéndolo de una caja a otra.

```
/*
 *Drag and Drop
 */
function allpwDrop(ev) {
    ev.preventDefault();
}

function drag(ev) {
    ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev, new_status) {
    ev.preventDefault();
    var data = ev.dataTransfer.getData("text");
    id_inicidence = data.replace("div_inicidence_", "");
    update_inicidence(id_inicidence, new_status);
    ev.target.appendChild(document.getElementById(data));
    if ("OPEN" == new_status) {
        document.getElementById(data).style.backgroundColor = "rgb(116, 207, 234)";
    } else if ("IN PROGRESS" == new_status) {
        document.getElementById(data).style.backgroundColor = "rgb(255, 186, 23)";
    } else if ("DONE" == new_status) {
        document.getElementById(data).style.backgroundColor = "rgb(39, 156, 38)";
    }
    console.log(new_status)
}
```

Figure 34 Screenshot drag and drop

- Hacemos uso de sus nuevos elementos específicos como: `<nav>`, `<header>`, `<footer>`, `<section>`.
- CSS3: este es el encargado de darle estilo y apariencia a los elementos de nuestra web. Así separamos el contenido de la apariencia, es decir, un mismo contenido puede tener distinta apariencias, así tenemos la flexibilidad para crearlas. Tiene la una gran ventaja en la reutilización ya que el diseño de un elemento se puede repetir muy frecuentemente. Este reduce la carga del navegador al cargar el contenido, puesto que toda la apariencia la tenemos almacenada en otro archivo.

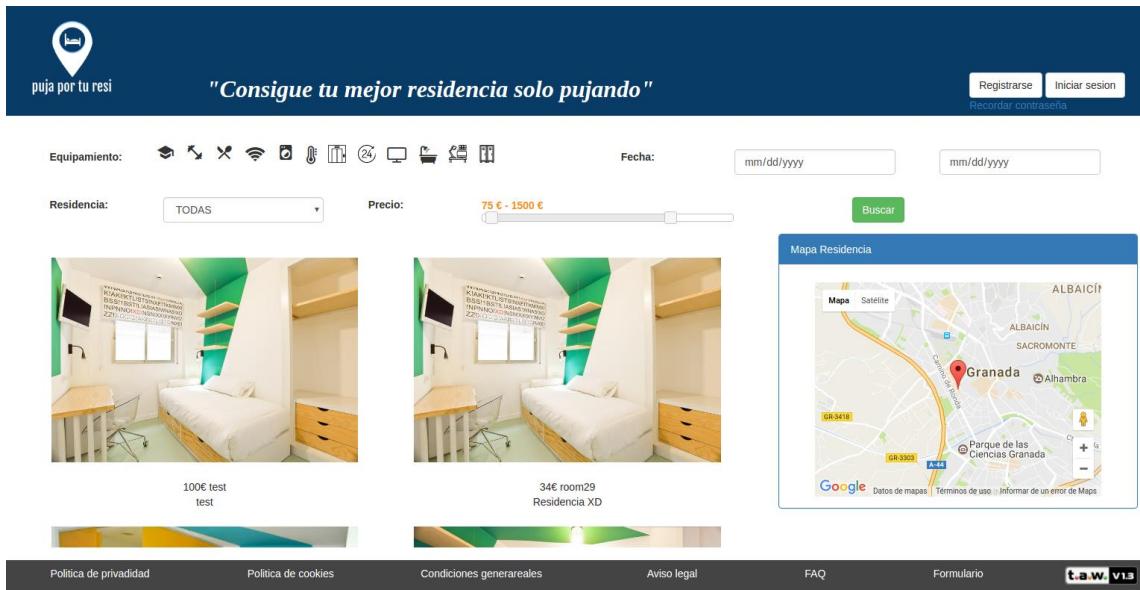


Figure 35 Screenshot página inicial

- Utilizamos una herramienta *Bootstrap* que nos permite crear elementos en HTML y CSS con la característica de que pueda adaptarse a distintas resoluciones de distintos dispositivos. Este es llamado diseño *responsive*. Además esta herramienta es sencilla y rápida. Asimismo cubre gran cantidad de elementos desde contenedores, botones, alarmas, *input*, formularios, tablas y demás. Esta tecnología fue creada para crear Twitter.

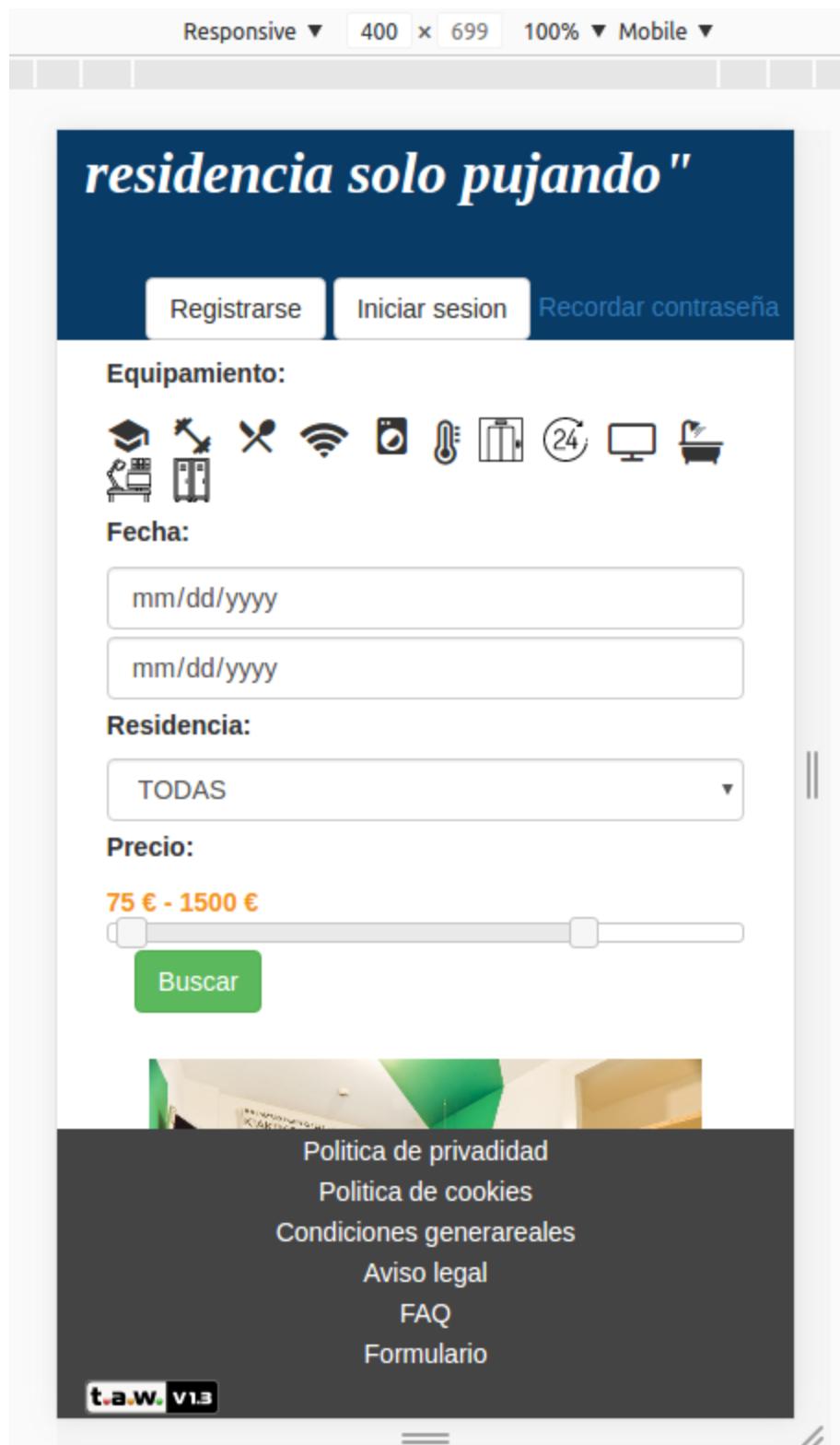


Figure 36 Screenshot Diseño responsive

Control de versiones

El proyecto sea gestionado a través de un repositorio en *git*. Nos hemos decantado por un repositorio de control de versión llamado Github. Este es uno de los más utilizados de forma gratuita. Entre sus principales funcionalidades nos ha permitido utilizar *branches*, *issues*, *milestones*. Aunque su mayor beneficio lo encontramos en la oportunidad de tener todo el

historial de nuestro código y poder acceder a este de forma fácil desde cualquier lugar. Además podemos gestionar el proyecto desde cualquier ordenador tan fácil como clonar lo y empezar a picar código. En conclusión este ha sido útil para el desarrollo de nuestro proyecto, ya que hemos podido llevar un seguimiento de todos los cambio en el código. Estos son los repositorios utilizados en nuestro proyecto en Github:

- https://github.com/softwarejimenez/API_puja_por_tu_resi
- https://github.com/softwarejimenez/web_puja_por_tu_resi

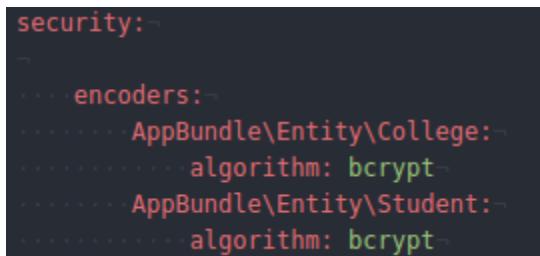
Seguridad web

La seguridad es uno de los temas más importantes a tener en cuenta en un proyecto web, según OWASP (OWASP, 2017) “25 Percent of Web Apps Still Vulnerable to Eight of the OWASP Top Ten” (DarkReading, 2017), teniendo en cuenta este dato, vamos a hacer hincapié en las posibles flaquezas y debilidades de nuestro sistema. En este apartado lo primero que hacemos es analizar las vulnerabilidades del proyecto y ver cómo se puede proteger de la mejor manera.

Brute force

El problema radica en la autenticación de los usuarios, para la cual utilizamos la combinación nombre usuario y contraseña. El ataque consiste en el uso de todos los posibles combinaciones de usuario y contraseña; utilizando diccionarios y frecuentes contraseñas.

La manera en la que lo solucionamos consiste en usar una encriptación en la contraseña a partir de la tecnología *bcrypt* (Assurance Technologies, LLC, 2017); esta usa una función *hash* y el valor total es 31 caracteres en base 64. Por lo tanto nuestro sistema resiste al ataque “fuerza bruta”.



```
security:
    encoders:
        AppBundle\Entity\College:
            algorithm: bcrypt
        AppBundle\Entity\Student:
            algorithm: bcrypt
```

Figure 37 Screenshot fuerza bruta code

Command execution/injection

Esta vulnerabilidad consiste en la ejecución de un comando maligno en el sistema, en consecuencia el sistema puede ser dañado; por ejemplo mostrar la tabla usuario. Esta vulnerabilidad es debida a un mal uso de la validación de entrada de datos.

La manera de solucionarlo es validando todos los datos de entrada y otra manera es no permitiendo que los valores de entrada sean ejecutados por el sistema operativo o por el gestor de la base de datos.

```
class Validate
{
    /**
     * Validate URL
     *
     * @param validator_module $validator
     * @param string $url      url
     *
     * @return bool
     */
    public function validateURL($validator,$url)
    {

        /**
         * Validate IBAN
         *
         * @param validator_module $validator
         * @param string $IBAN     IBAN
         *
         * @return bool
         */
        public function validateIBAN($validator,$IBAN)
        {
    }
```

Figure 38 Screenshot Validate Class

Cross-site request forgery (CSRF)

Este consiste en que el atacante quiere realizar una acción con los mismos privilegios que el usuario, sin autenticarse.

En nuestra aplicación sería vulnerable por ejemplo en el cambio de los datos bancarios, para así recibir los pagos de las mensualidades a otra cuenta bancaria. Para ello utilizamos el método (Bank/create/) POST en la API.

La manera que tenemos que mitigar este ejemplo consiste en que para poder ejecutar este método el usuario debe estar autenticado y los cambios solo serán realizados en dicho usuario.

Esta vulnerabilidad la tenemos resuelta porque la plataforma Symfony nos permite asegurar que una función solo puede ser ejecutada por un *role* de usuario autenticado. Esto se llama: *access_control*. Así que esta herramienta deniega la ejecución de la petición a la API. Otra manera de solucionarlo por parte del usuario que utiliza la aplicación es no permitir al navegador guardar el usuario y contraseña y tener la opción de recordar.

```
access_control:
    - { path: ^/api/doc, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/Signin/, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/rememberPassword, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/lucky/, roles: IS_AUTHENTICATED_ANONYMOUSLY }
#AGREEMENT CONTROLLER
    - { path: ^/Agreement/create/ , roles: ROLE_ADMIN }
    - { path: ^/Agreement/remove/ , roles: ROLE_STUDENT }
    - { path: ^/Agreement/accept/ , roles: ROLE_STUDENT }
    - { path: ^/Agreement/assignedRooms/ , roles: ROLE_ADMIN }
    - { path: ^/Agreement/download/ , roles: [ROLE_STUDENT, ROLE_COLLEGE, ROLE_ADMIN] }
    - { path: ^/Agreement/getCurrentSigned/ , roles: ROLE_STUDENT }
    - { path: ^/Agreement/getList/ , roles: ROLE_STUDENT }
    - { path: ^/Agreement/roomVerifyUnsigned/ , roles: ROLE_COLLEGE }
    - { path: ^/Agreement/removeUnsigned/ , roles: ROLE_ADMIN }
```

Figure 39 Screenshot Access Control Method

Upload file

Esta debilidad consiste en subir a la web un archivo el cual puede dañar el sistema, como por ejemplo sobrescribir archivos existentes necesarios o subir *script* o subir archivos muy grandes lo que reducirían los recursos del sistema o que muchos usuarios a la vez suben muchos archivos pequeños, lo que hace el sistema lento. Estos problemas han sido resueltos de la siguiente manera:

- Validando el formato y tamaño del archivo.
- El nombre de archivo es un nuevo nombre aleatorio utilizando “md5(uniqid())”
- Solo se puede subir un archivo no más por usuario y solo cuando se firma el contrato.

```
$file=$request->files->get('file_agreement_signed');
if (!$this->get('app.validate')->validatePDFFile($this->get('validator'),$file) and !$this->get('app.validate')->validateImageFile($this->get('validator'),$file)){
    return $this->returnjson(false,'Archivo no es valido (PDF- IMG).');
}
$filename=md5(uniqid()).'.'.$file->getClientOriginalExtension();
$file->move($this->container->getParameter('storageFiles'),$filename);
```

Figure 40 Screenshot upload file

Cross-site scripting (XSS)

Esta vulnerabilidad consiste en el atacante deja un *script* en un dato de entrada el cual será generado como dato de salida y será mostrado a otros usuarios. Entonces este *script* será ejecutado con los privilegios del otro usuario. Este *script* puede manejar *cookies* o enviar mensajes o modificar datos del usuario y demás.

Este puede ser utilizado en los mensajes que son enviados de un usuario a otros. Esto puede tener un *script* y ejecutarse sin que el propio usuario lo sepa.

La forma de mitigar esta debilidad es escapar los datos de entrada HTML al ser enviado al servidor. En el cliente utilizamos la función *escape(input)* para solventarlo

SQL injection

El servidor utiliza un gestor de base de datos SQL. Entonces el servidor realiza peticiones SQL con los datos de entrada de los usuarios. Entonces el atacante puede utilizar esta funcionalidad para concatenar diferentes comandos en SQL y por ejemplo mostrar todos los usuarios y contraseñas de la base de datos.

La forma en que he mitigarlo es utilizando los métodos “findby()” y “findAll()” de Symfony de esta manera no es posible la concatenación de varios comandos SQL y no es posible *SQL injection* en Symfony.

```
$colleges = $this->getDoctrine()->getRepository('AppBundle:College')->findAll();  
if (!$colleges) {  
    return $this->returnJson(false,'No hay ninguna residencia.');//  
}else {  
}
```

Figure 41 Screensot SQL injection

Accesibilidad web

Esta tiene en cuenta que personas con algún tipo de discapacidad, como sensorial o motora, puedan acceder a la web; es decir, estas pueden percibir, entender, navegar e interactuar con la propia web. La accesibilidad web está definida en WCAG (W3C, n.d.) (Web Content Accessibility Guidelines) desde 2008, por otro lado, esta pertenece a W3C (World Wide Web Consortium).

En la definición encontramos unas pautas para desarrollar contenidos accesibles y como poder evaluarlos, la evaluación combina de forma automática y manual. Todos estos criterios no dependen del navegador o del lenguaje utilizado en el desarrollo, es decir, es neutro.

La accesibilidad web parte de cuatro fundamentos teóricos:

- Perceptible: contenido percibido por al menos un sentido.
 - Alternativas textuales.
 - Adaptable para poder verse o por vista o sonido.
- Operable: todo tipo de usuarios puedan interactuar con el contenido.
 - Mediante teclado.
 - El contenido debe estar un tiempo mínimo.
 - No usar destellos.
- Comprensible, es decir, legible y entendible.
- Robusto: puede ser utilizado por distintos navegadores.

WCAG verifica si una web cumple la accesibilidad atendiendo a tres niveles, teniendo en cuenta el número de pauta que cumple, debemos fijar una imagen con el nivel de accesibilidad que cumple nuestra web.

- Nivel “A”: se puede comprobar de forma automática. Verifica prioridad uno.
- Nivel “AA”: valida prioridad uno y dos. El contenido se presenta de varias maneras.
- Nivel “AAA”: verifica prioridad uno, dos y tres. Son accesible para personas con cualquier discapacidad.

Para el análisis de la accesibilidad de nuestra web hemos utilizado TAW (TAW, Centro TIC, n.d.) una herramienta con la que podemos realizar el análisis de forma local en el ordenador, es decir, la web no está en Internet.

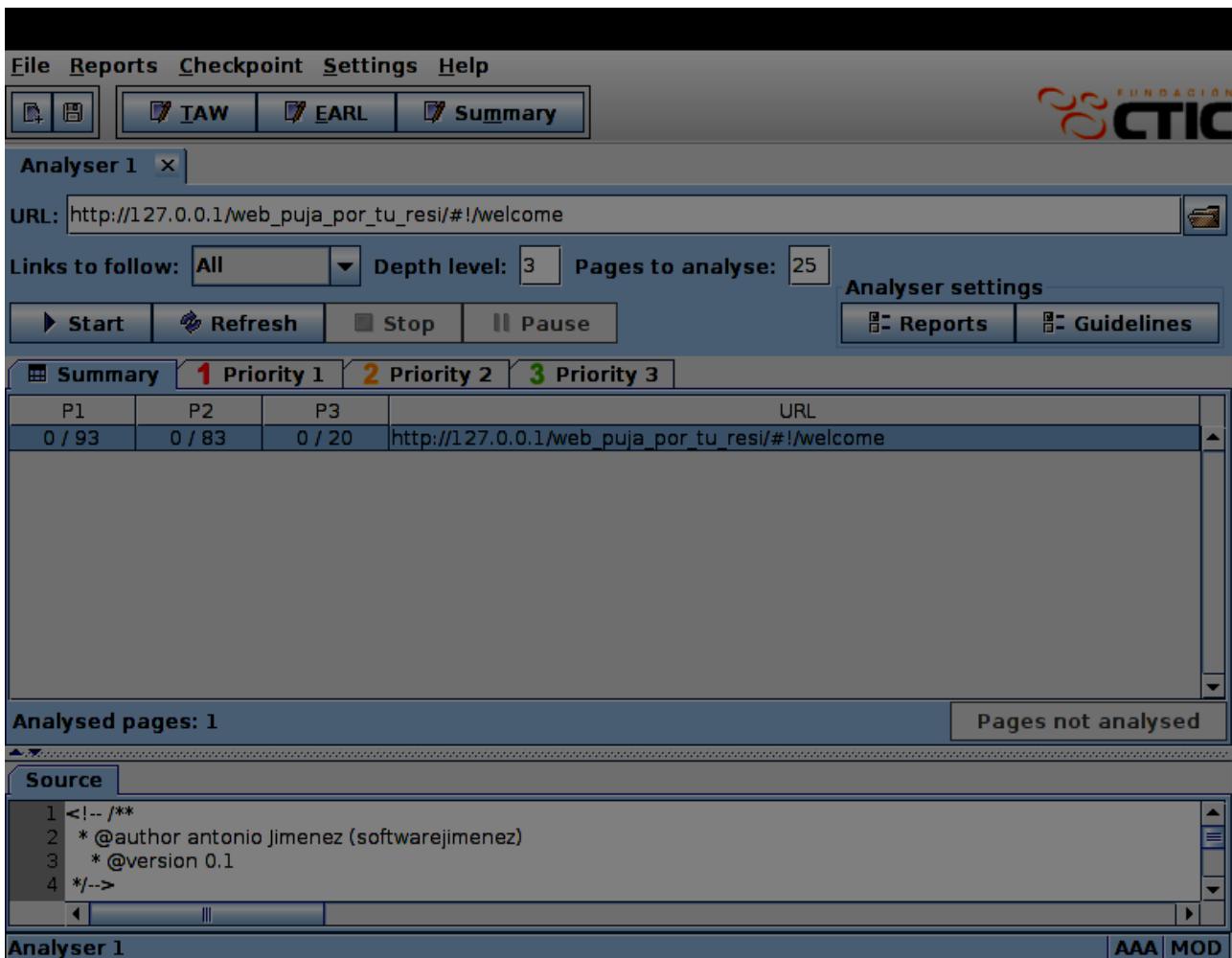


Figure 42 Herramienta TAW para analizar accesibilidad

Al principio hallamos los siguientes errores:

- Prioridad 2:
 - 11.2 Avoid deprecated features of W3C technologies. <center> is now deprecated and should not be used.
 - 3.5 Use header elements to convey document structure and use them according to specification. Header levels must not increase by more than one level per heading. Do not use heading to create font effects; use style sheets to change font styles.
- Prioridad 3:
 - Missing lang attribute: the primary language of this document has not been set.

Una vez solucionados todos ellos, aparece uno nuevo:

- Prioridad 2:
 - 3.5 Use header elements to convey document structure and use them according to specification. There are no header elements in this document.

Una vez solucionados todos los criterios de accesibilidad automáticos de la página inicial, pasamos a resolver los criterios de accesibilidad de revisión manual.

En primer lugar pasamos a revisar los criterios de prioridad 1, entre los errores más importantes explicamos:

- Toda la información está disponible también sin color.
- El documento puede ser leído sin hoja de estilo.
- La web se puede utilizar sin los *script*, ni los *applets*.
- El contenido de la web debe ser simple y limpio.

Una vez pasado los tests automáticos y manuales de prioridad 1, podemos decir que nuestra página inicial tiene accesibilidad “A”, con el objetivo de que puedan acceder a la web el mayor número de usuarios.

Test summary outcome		
	Automatic	Human review
Priority 1	1>0	2>0
Priority 2	2>0	2>83
Priority 3	3>0	2>20

Figure 43 Resultado accesibilidad A

El resultado de la herramienta TWA es el siguiente:

Congratulations! This Web page has no Priority 1 accessibility issues. The human review has satisfied all Priority 1 checkpoints.

This Web page satisfies the WAI-A conformance level. You can use any of the following logos to indicate that the Web page has been evaluated with taw 3.



Figure 44 Logo para añadir a la web respecto a la prioridad A

For the logos you can use the following alternative text: **Valid taw 1.3**

Problemas y soluciones

Organización archivos *front-end*

Durante el desarrollo del cliente web, ha habido momentos en los que había demasiadas funciones, html id, html class y demás. Entonces decidimos organizar la estructura, siguiendo un estándar y organizar las funciones por ficheros. De esta manera tenemos una división teniendo en cuenta la distinta vista:

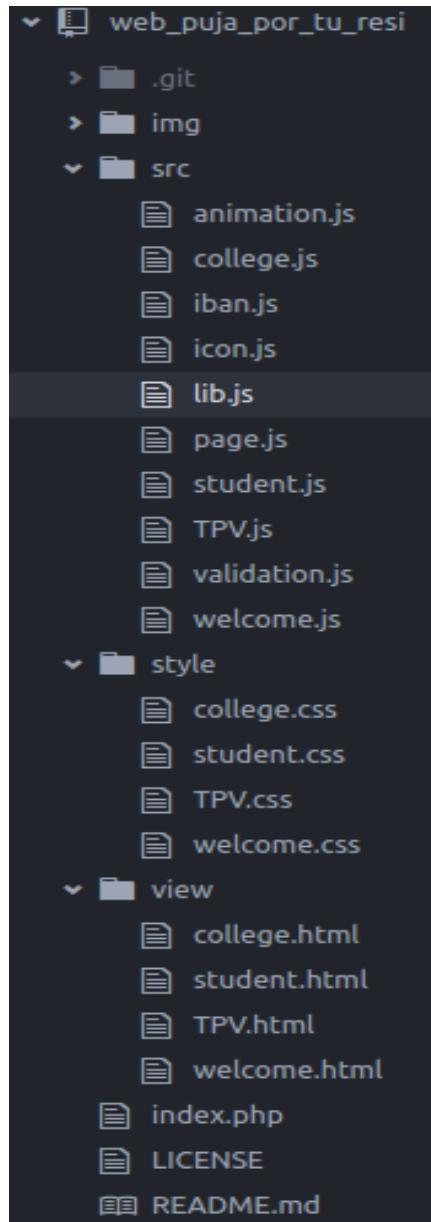


Figure 45 Estructura de archivos *front-end*

- Welcome: la página inicial. Así tenemos 3 archivos:
 - welcome.css donde tenemos el estilo de la página de bienvenida.
 - Welcome.js donde tenemos la funcionalidad de esta página como crear un usuario o iniciar sesión.

- Welcome.html donde tenemos la estructura html de la página de bienvenida.
- Student / college: es la vista del estudiante / residencia la cual está dividida en varios módulos como perfil, buscar habitación, mi habitación, incidencias, mensajes y pagos.
 - student.css / college.css donde tenemos los estilos de estos módulos.
 - student.js / college.js donde tenemos las funciones para gestionar los datos y hacer las vistas dinámicas.
 - student.html / college.html donde se encuentra la estructura de todos los módulos.
- index.php: esta es la página principal. Esta mostrara una vista u otra dependiendo de los privilegios y roles del usuario.
- Validation.js: este archivo tiene todas las funciones para validar los datos: DNI CIF tarjeta de crédito, CVV, teléfono, url, email y demás.
- Lib.js: tenemos las funciones que se usan en la vista de estudiante y residencia como mostrar estudiante o habitación o residencia o contrato o apuestas y demás.
- Animation.js: este tiene las funciones que dan dinámica a la web, como la rotación de las imágenes, mostrar el mapa, mostrar un elemento, mostrar mensaje de error y demás.
- Icon.js: este tiene un conjunto de funciones para crear iconos.
- TPV.js. este script simula un TPV para el módulo de pago.

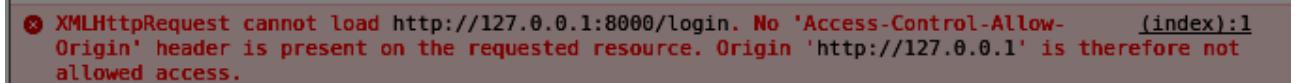
Respecto a nuestro cliente web, este sería híbrido puesto que no almacena los datos pero si los procesa, es decir los crea, modifica, muestra, elimina, ordena y demás.

Llevamos a cabo un diseño modular, el cual no se centra en cada página web, sino en los módulos como una habitación, esta será mostrada en varia vista y siempre siguiendo el mismo modulo. O el modulo residencia el cual será utilizado varia veces como en el perfil de una residencia o en la residencia propietaria de una habitación.

Estos módulos son reutilizable e independientes de la vista.

Header and AJAX

El servidor está en escuchando en un puerto y el cliente en otro. Entonces cuando se comunican mediante llamada http (AJAX), aparece el siguiente problema.



XMLHttpRequest cannot load http://127.0.0.1:8000/login. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://127.0.0.1' is therefore not allowed access.

Figure 46 Error Access control Allow origing

Como se puede apreciar el problema estaba relacionado con los *header* de la respuesta. La solución fue crear en el *ListenerEvent*, el cual modifica todas las respuesta del servidor añadiendo 'Access-Control-Allow-Origin' = 'http://127.0.0.1' para todas las repuesta.

Bootstrap-select

Durante la etapa de desarrollo, se prevé que una funcionalidad tan sencilla como enviar

un mensaje a varios usuarios podría ser desarrollada en unas horas. En cambio apareció un problema el cual hizo que el proceso se alargara hasta un día.

El problema surge cuando necesito un elemento en el formulario para elegir a que usuario enviar un mensaje. De esta manera este debería tener varias funcionalidades, como:

- Desplegar todos los usuarios.
- Desplegar todos los usuarios que contiene una o más letras (buscador).
- Ver los usuarios seleccionados.
- Seleccionar y deseleccionar todos los usuarios de forma fácil y rápida (un botón).

Después de intentar hacerlo con diferentes elementos de *html* como *input*, *select*, *option* y eventos como *onkeypress* y demás. Decidimos buscar ejemplos o casos de uso en Internet.

Encontramos un elemento de *html5* muy interesante, llamado *datalist* el problema es que no es soportado en Internet Explorer y Safari, de esta manera decidí que debo seguir buscando.

Entonces encuentro una biblioteca que utiliza Jquery y Boostrap y que hace todo lo que necesitaba, esta es llamada bootstrap-select (*bootstrap*, n.d.), después de analizar cómo funciona y ver varios ejemplos. Decido utilizarla, añadido los DCN (Content Delivery Network) y copio un ejemplo.

En conclusión, he encontrado un elemento de Boostrap que me ayuda para gestionar los usuarios a los que le mensaje le llegara y me he dado cuenta que siempre hay que añadir un poco más de tiempo en la estimación en un *Sprint* en la metodología SCRUM.

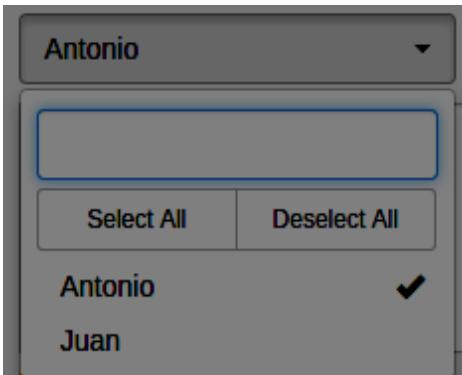


Figure 47 Ejemplo elemento Bootstrap-select

Módulo de pago

En un principio implementamos un módulo de pago dentro de la aplicación y almacenamos en la base de datos información sobre la tarjeta con la que se había realizado el pago.

En cambio tras analizar otros sistemas y hablarlo con el tutor, no dimos cuenta que el pago debería ser totalmente independiente y transparente para nuestra aplicación, es decir, el sistema debería enviar a un simulador de TPV la cuantía y la información del destinatario (la cuenta bancaria activada en ese momento). De este modo decidimos crear un módulo TPV extra e independiente, este será un simulador el cual generará la id de la transferencia. Este nuevo módulo recibirá la cuantía, IBAN y BIC de destino; y devolverá si la transferencia ha sido efectuada con éxito y el identificador de la transacción.

Página inicio

En un principio la página de inicio era un simple formulario para darse de alta o iniciar sesión, entonces entre el tutor y el estudiante llegamos a la conclusión que esta vista no llamaba la atención de los visitantes.

Como consecuencia creamos una página de bienvenida con la idea de nuestra de la web, como el eslogan del proyecto, “consigue tu mejor vida en residencia con solo una puja”. Actualmente mostramos la búsqueda de habitaciones, es decir, le expresamos al usuario que es lo que puede hacer.

Como conclusión, la página de inicio debe ser una portada a nuestro negocio. Según estudios la decisión si quedarse o no en una página web por parte de los clientes es de 5 segundos (Cabo, 2016). Asíque tenemos que llamar la atención de los clientes. Entre los objetivos de la página de bienvenida encontramos:

- Explicar lo que hacemos a partir de un eslogan, el cual sea fresco, innovador y corto. Estos no leerán un largo texto.
- Mostrar que oferta la página web, en nuestro caso una lista de habitación y residencias por la que puedes pujar.
- No mostrar una cantidad excesiva de información, para no confundir al cliente, si no mostrar una página ordenada y clara para la vista del cliente.
- Mostrar una imagen, puesto que el cliente siempre analizara las imágenes antes de leer un largo texto.

Es resultado es una página de bienvenida sencilla en la cual encontramos varios paneles para iniciar sesión, registrarse y buscar habitación.

Clase Agreement

En la clase contrato en primer lugar pensamos decidimos que un contrato era una relación entre una habitación, una residencia y un estudiante.

Entonces no dimos cuenta que no era necesario tener la relación residencia y contrata, puesto que una habitación pertenece a la residencia.

En un principio pensamos que una habitación solo podía tener un contrato y que un estudiante solo podía tener un contrato. Pero nos dimos cuenta que a la hora de la escalabilidad, una habitación y un estudiante tendrán varios contratos en el sistema. La manera de diferenciarlos y saber cuál es el activo se debe la fecha inicio y fin del contrato.

Organización de las fechas

En un principio las fechas estaba almacenadas en la clase habitación, es decir, la residencia asignaba las fechas del periodo del contrato y del periodo de puja. En cambio, nos hemos dado cuenta que esto no genera ninguna flexibilidad, ni escalabilidad. De esta manera hemos decidido organizarlas así:

- Cuando el usuario busca una habitación indica la fecha de entrada y salida.
 - En el caso en que el usuario esté interesado por la residencia y puje por ella. Las fechas del periodo del contrato se almacenan en la clase *Bid*.
- La asignación de habitaciones se realiza una vez por semana, fijamos este día como el viernes a las 00:00.
 - En la asignación, cogemos cada habitación con pujas.

- En el caso de que dos pujas coincidan en fechas se le asignara al usuario con más puntos.
- Restricciones:
 - Una habitación o estudiante tiene varios contratos en distintas fechas, aunque no puede tener colisiones de fechas en contratos.

Table responsive design

Uno de los problemas que tuvimos fue que las tablas no eran cómodas de ver cuando veíamos la web en el móvil. Entonces decidimos aplicar otra técnica para mostrarlas, es decir, aplicamos un diseño *responsive* para las tablas (COYIER, 2011). A continuación podemos ver una imagen de como era antes y de cómo es ahora.

The figure shows two versions of a table side-by-side. On the left, labeled 'new', is a vertical stack of cards for a room booking. It includes fields for Habitacion (room17), Residencia (Residencia XD), Precio (800€), Contrato (with a file download icon), Fecha inicio (2017-08-26), Fecha fin (2018-07-26), Firmar, and Rechazar. On the right, labeled 'Old', is a horizontal table with a blue header row and white data rows. The columns are labeled Habitacion, Residencia, Precio, Contrato, and Fecha inicio. The data row corresponds to the information in the 'new' card.

Habitacion	Residencia	Precio	Contrato	Fecha inicio
room17	Residencia XD	800€	file_download	2017-08-26

Figure 48 History Table responsive design

Testeo software

Test unitarios y funcionales

En este apartado vamos a comprobar que nuestro código funciona como esperábamos en todos los posibles casos. Para ellos vamos a utilizar una herramienta de Symfony llamada PHPUnit ([symfony, n.d.](#)). Para su uso es necesario instalarlo con el comando: `sudo apt install phpunit`.

Durante el proceso de desarrollo de los tests de nuestra API, hemos creado varios tests para cada controlador. Durante el procedimiento nos ha aparecido varias dificultades como

- Autenticar al usuario para realizar la petición.
- Crear un archivo temporal ficticio para los tests.
- Tomar la respuesta de la petición en formato *String*.

La elaboración de los tests intento realizarse en las primeras etapas del desarrollo, en cambio no fue posible puesto que no sabíamos cómo se iban a llamar los métodos de la API. De esta manera se ha creado un test por cada principal método de los controladores. Por otro lado, podemos decir que en las metodologías ágiles los tests y el desarrollo se realizan de forma simultánea.

Los tests creados han sido utilizados cada vez que se añade una funcionalidad para asegurar que lo anterior funciona correctamente. Lo ideal es seguir una filosofía de TDD (*test-drive development*) en la cual cada vez que se realiza un nuevo *commit* o incremento de código este es testeado automáticamente, para ello es necesario un *script* el cual lanza los tests antes de cada *commit* en *git*.

Asimismo, los tests sirven para encontrar bug o errores en nuestro código; por ejemplo durante el proceso de desarrollo hemos movido la fechas del contrato desde la clase *room* hasta la clase *agreement* y uno de los métodos del controlador *room* llamado *room/remove/id* utilizaba las fechas en la clase *room*, este error fue detectado gracias a los tests. Es decir, otra de las finalidades es la búsqueda y detención de errores.

Además estas tienen como principal objetivo llevar un control de calidad para poder darle al usuario una seguridad de lo que está haciendo, es decir, mostrando la calidad del software y reduciendo el riesgo de fallos. Igualmente los tests también sirven para verificar el buen funcionamiento de las historias de usuario y de las precondiciones y postcondiciones.

La mayoría de los tests son funcionales o también llamados integración test. Estos crean un escenario del caso y testean la definición, ejecución y validación, entre los ejemplos podemos destacar:

- *Controlador Room*: testeamos la creación de una habitación y la destrucción de esta.
- *Controlador Incidence*: testeamos la creación de una incidencia y la actualización de su estado.

Según este estudio “the economic impacts of inadequate infrastructure for software testing” ([NITS, 2002](#)), el desarrollo de test es una parte principal en el desarrollo software, respecto a la detención de errores. Puesto que cuanto antes encuentres un error, más fácil será resolverlo puesto que el proyecto puede seguir creciendo haciendo imposible encontrar una

manera de arreglar un error.

Respecto a nuestro caso principal es el testeo de API, para ello realizamos la implementación sobre métodos de la API de nuestro servidor, comprobando que tenemos la funcionalidad esperada. Además a través de nuestra herramienta *phpUnit* podemos ver el tiempo y memoria necesaria. A la hora de hacer la comprobación detectamos si los valores de salida o respuestas son los esperados además de comprobar el código de la petición.

Test de usabilidad

El test de usabilidad consiste en que los posibles usuarios usen y valoren la web a través de un formulario, es decir, evaluar el producto a través de los propios usuarios. Con el objetivo de comprobar que es cómodo de usar y que lo usuario lo entiende como esperábamos.

La usabilidad consiste en que la aplicación se puede utilizar por diferentes usuarios para conseguir objetivos con eficiencia, efectividad y satisfacción.

Los usuarios que han realizado el test han realizado las siguientes tareas, con el siguiente orden:

- Darse de alta.
- Buscar habitaciones a través de un filtro.
- Pujar por varias habitaciones.
- Una habitación ha sido asignada
- Descargar el contrato y firmarlo y subirlo a la web
- Crear una incidencia.
- Recibir y enviar un mensaje.
- Realizar el pago de un mes.
- Cerrar sesión.

Además, para aquellos que no han podido acceder a la web, hemos creado dos demos los cuales están en YouTube, con dirección web:

- <https://www.youtube.com/watch?v=kliUUXpZlg0> (vista del estudiante)
- https://www.youtube.com/watch?v=i3NzDyx_N2g (vista de la residencia)

Los cuales han sido bastante útil. Los usuarios son normalmente posibles usuarios es decir, estudiantes de diferentes modalidades que han vivido o conocen como es vivir en una residencia.

Respecto al análisis de los resultados no hemos dado cuenta que la página puede tener repercusión en los usuarios puesto que la extensa mayoría de lo que lo han realizado están contentos con el sistema, lo entienden y lo utilizarían en el futuro como podemos ver en los resultados. El formulario utilizado en este test junto con los resultados puedes encontrarlo en el apéndice.

Por otro lado no ha ayudado a darnos cuenta de que partes son más interesantes y comprensible y cuáles no.

- Los usuarios están contentos normalmente con:
 - Sistema de pujas
 - Tener todas las residencias y habitaciones en una misma web.
 - Poder hacer con el móvil.
 - Realizar todo de forma remota.
 - Posibilidad de encontrar la mejor habitación.
- Entre las deficiencias podemos mostrar:
 - Una búsqueda compleja.
 - No recoge todas las residencias.
 - No me da toda la información que necesitaría como que hay alrededor de la residencia.
 - No existe una aplicación móvil.

Uno de los apartados interesantes de la encuesta es el de que modulo te gustaría utilizar extra, listados por orden de preferencia:

- Menú del comedor
- Chat grupal

- Módulo de pago con PayPal.
- Organización de actividades.

Finalmente, podemos decir que la encuesta nos ha servido para darnos cuenta que el producto puede llegar al público y ser utilizado aunque antes teneos algunos puntos que resolver y hacerle frente. Este tipo de test es uno de los más interesantes para darte cuanta de los límites del proyecto, es decir, podemos ver como los usuarios se sentirían utilizando nuestra aplicación además de ver si entienden el funcionamiento.

Por otro lado, hemos aplicado las 10 reglas de la usabilidad de J Nielsen (Nielsen, 1994):

1. Nuestro sistema siempre representa el estado, a través de los contratos, de las pujas realizadas, de las incidencias y de los mensajes.
2. Nuestro sistema utiliza un lenguaje cotidiano para los usuarios.
3. Nuestro sistema permite a los usuarios interactuar con libertad a través de los módulos, el único requisito es estar registrado.
4. Nuestro sistema utiliza estándares con el objetivo de que sea más intuitivo para los usuarios.
5. Nuestro sistema recoge los errores de los usuarios y los muestra por pantalla, por ejemplo errores de la validación de datos de entrada.
6. Nuestro sistema recoge los datos de una base de datos almacenada en un servidor.
7. Nuestro sistema es eficiente.
8. Nuestro sistema tiene un diseño minimalista con el objetivo de que sea claro para el usuario.
9. Nuestro sistema muestra los errores para que los usuarios lo entiendan y no los vuelvan a cometer.
10. Nuestro sistema proporciona ayuda a través de las preguntas frecuentes, FAQ.

Documentación

Actividades legales en la web

Política de privacidad

Este apartado trata sobre la protección de datos (Rodríguez, 2008). Nos regimos a partir de la ley de la constitución española, art. 18.4: "La ley limitará el uso de la informática para garantizar el honor y la intimidad personal y familiar de los ciudadanos y el pleno ejercicio de sus derechos". Esta tiene como objetivo proteger a las personas ante el uso de sus datos personales de manera no autorizada. La agencia de protección de datos se encarga de controlar el cumplimiento de la ley de protección de datos; esta es un ente con derecho público. Esta agencia debe comprobar que la ley de protección de datos se aplica correctamente sobre reclamaciones presentadas por los afectados; y en caso necesario sancionar las aplicaciones que utilicen los datos de forma errónea ante la ley.

LOPD (Ley Orgánica de protección de datos de carácter personal) (López-Vidriero, 2005), esta protege el tratamiento de los datos personales, es decir, cualquier operación o procedimiento técnico sobre cualquier información perteneciente a persona física. El responsable y encargado del tratamiento o fichero, debe clarificar el uso que le dará a los datos y como serán almacenados y las medidas de seguridad seguidas dependiendo del tipo de datos. Encontramos distintas medidas de seguridad: nivel alto, medio y básico. Respecto a los datos utilizados en nuestra aplicación, estos deberían ser de nivel básico. Ya que los datos son datos personales, los cuales no ofrecen una definición de las características o de la personalidad. De esta manera, según la ley, nuestro sistema debe cumplir los siguientes artículos:

- Art. 89 Funciones y obligaciones del personal.
- Art. 90. Registro de incidencias.
- Art. 91. Control de acceso.
- Art. 92. Gestión de soportes y documentos.
- Art. 93. Identificación y autenticación.
- Art. 94. Copias de respaldo y recuperación.

Por otro lado es necesario mostrar a los usuarios de la aplicación web: quién es el responsable de los datos, cuál es el tratamiento de los datos y si estos son utilizados por terceros. Además hay que explicar cuáles son las medidas de seguridad utilizadas para almacenar los datos y que datos son almacenados y para qué cada uno.

El documento Política de privacidad, el cual será expuesto a los usuarios de nuestra aplicación se encuentra en el apéndice. En este documento informamos que datos son recogidos y que uso se les da. Además informa sobre la posibilidad de acceso, rectificación y cancelación de los datos. Mostrar la identidad del responsable del tratamiento.

Política de cookies

En este apartado hablaremos de la política de *cookies* (Agencia Española de Protección de Datos, 2017) y definiremos la política de *cookies* de nuestra aplicación web.

En principio *cookies* es un fichero que se alojan en la memoria del dispositivo cuando se accede a la web para distintos servicios y funcionalidades de la aplicación web. La *cookies* puede ser

utilizada por la misma web o por terceros.

La ley de *cookies* forma parte del artículo 22 de la ley de servicios de la sociedad de la información y de comercio electrónico (LSSI), el cual dice “Los prestadores de servicios podrán utilizar dispositivos de almacenamiento y recuperación de datos en equipos terminales de los destinatarios, a condición de que los mismos hayan dado su consentimiento después de que se les haya facilitado información clara y completa sobre su utilización, en particular, sobre los fines del tratamiento de los datos, con arreglo a lo dispuesto en la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal”.

La política de *cookies* debe estar accesible con un link en todas las vista de la aplicación web (Cuestiones Derecho, 2013), es decir, el titular de la aplicación debe exponer en un informe sobre los tipos de *cookies* que se instalarán, cuales son su objetivo y como deshabilitarlos. Además esta debe mostrar que *cookies* son usadas, de que tipo son, la temporalidad, titularidad y funcionalidad.

De esta manera el usuario puede decidir no instalar las *cookies* o no navegar por la web. En todo momento los usuarios pueden decidir no utilizar las *cookies* a través del navegador. Además de que los usuarios pueden borrar y ver la *cookies* individualmente.

Por otro lado, existe un conjunto de *cookies* las cuales están exentas de ser informadas, es decir, no es necesario pedir consentimiento de uso, como:

- *Cookies* totalmente necesarias para el servicio.
- *Cookies* necesarias para la comunicación.
- Entre las *cookies* exentas podemos destacar:
 - *Cookies* de autenticación
 - *cookies* de seguridad
 - *cookies* de carga
 - *cookies* para personalizar interfaz
 - *cookies* de la cesta de la compra
 - *cookies* para llenar formularios.

La política de *cookies* de nuestra página web, la puedes encontrar en el apéndice.

Aviso legal

En este apartado describimos el texto legal de la web llamado aviso legal (López-Vidriero, 2005). Este recoge las cuestiones de la LSSI (Ley de servicios de la información). Es decir, informar quien es el responsable de la web, debemos encontrar la siguiente información:

- Nombre y domicilio.
- Inscripción en el registro mercantil.
- Título académico oficial, en el caso de que se ejerce una profesión.
- Número de identificación fiscal (CIF).
- Códigos de conducta.
- Derechos de los contenidos mostrados en la web, respecto quien los proporciona.

El documento que mostramos en nuestro web respecto aviso legal, lo puedes encontrar en el apéndice.

Condiciones generales de contratación y uso

En este apartado se explicara en qué consisten las condiciones generales de contratación y/o uso y como son explicadas en nuestra aplicación web. Este texto legal consiste en indicar (Rodríguez, 2008):

- Si en los precios mostrados los gastos de envío e impuestos están incluidos.
- Describir el proceso de compra.
- Obligaciones del vendedor y del comprador.
- Formas de pago de la compra.
- Idioma del contrato.
- Existe dos maneras de confirmar al comprador que la operación se ha realizado con éxito.
 - Un correo electrónico.
 - Pantalla de confirmación.

Este documento tiene como objetivo regular el acceso y la utilización de la aplicación web, bien por usuarios o bien por terceros.

Las condiciones generales de contratación y/o uso de nuestra aplicación web las puedes encontrar en el apéndice.

Logo

En este apartado hablaremos de cómo hemos elegido el logo o ícono de nuestra aplicación. Este fue elegido al principio del desarrollo del proyecto y este es utilizado en la documentación y en la página web, exactamente en el *header* de la página web.

Mis habilidades en el diseño de imágenes e iconos son escasas, por lo cual decidí utilizar una aplicación para el diseño, esta es “freeLogoDesign” (Free Logo Design, n.d.) una web donde puedes fácilmente crear tu logo con un título una imagen y un diseño.

De esta manera a partir del nombre de nuestra aplicación *PujaPorTuResi* y dos imágenes una sobre una de una cama y otra de un puntero de localización creamos nuestro ícono.

Este está disponible en dos colores, los cuales serán usados dependiendo del fondo, así un ícono es blanco y otro azul. El formato de nuestro ícono es “.ico”, el cual es el formato más popular para íconos.



Figure 49 Logo de nuestra aplicación

Es difícil crear el ícono ideal, el cual llame la atención del cliente, atractivo, cómodo puesto que será visible todo el tiempo, defina la aplicación, etc. Además otra complicada cuestión es el tamaño y el sitio en el que lo sitiamos (Creative Bloq Staff , 2014).

El color es otro atributo importante en un logo, teniendo en cuenta que la paleta de colores de nuestra web es azul, así que nuestro ícono tiene distintas tonalidades de esta.

Otra característica es que este será único, solo será utilizado en nuestra web y esta será identificada y asociada a través del logo.

Trabajos futuros

Esta sección trata sobre los posibles trabajos adicionales sobre el proyecto. Tras ver como los posibles usuarios están satisfechos con nuestro producto nos gustaría continuar y mejorarlo y añadir nuevos módulos e interfaces, además de un marketing para hacerlo llegar a residencias. Seguidamente mostraremos una lista de los posibles trabajos:

- Aplicación móvil en los distintos sistemas operativos: *Android, IOS, Window*. Con el objetivo de atraer más usuario y facilitar el uso.
 - En un principio nos gustaría hacer solamente la vista del cliente *Student* para *Android* y si este es utilizado implementarlo en otros sistemas las dos vistas.
- Solucionar los problemas expuesto por posible usuario como:
 - Dificultad para crear alto número de habitaciones. Implementaríamos una función para crear un conjunto de habitaciones iguales únicamente modificando el nombre del a habitación.
- Modulo extra para la gestión de comedor. Puesto que la mayoría de las residencias tiene un comedor donde ofrecen un menú semanalmente. Nos gustaría que los usuarios pudiesen ver en cualquier momento el menú de la residencia y los alérgenos de este. Por otro lado la residencia tiene que actualizar el menú periódicamente.
- Modulo actividades extras: Esta sección extra informaría sobre qué actividades la residencia propone, entonces los usuarios puede reservarlas y pagarlas. De esta manera no es necesario un recepcionista para esta tarea. Entre las actividades podría ser viajes, actividades deportivas, cenas, fiestas de inauguración y clausura y demás.
- Chat grupal entre todos los estudiantes de una residencia donde ellos pueden comunicarse y hablar sobre sus planes y actividades además de explicar problemas y como resolverlos.

En este chat aparecerían únicamente los estudiantes, es decir, la residencia no puede acceder a este.

Tendríamos que cambiar el diseño puesto que sería un mensaje para mucho o para una conversación específica, no para una un único usuario como lo hacemos ahora mismo.

- Módulo de pago con *Paypal*, el cual permite hacer el pago del mes no con tarjeta, sino con el sistema *Paypal* también. Con el objetivo de atraer a más clientes, puesto que los jóvenes utilizan este sistema con frecuencia.
- La capacidad de que los usuarios valoren por la residencia y sus habitaciones. De esta manera ayudara a otros estudiantes a decidir la mejor habitación.
- Añadir la opción de buscar por ciudad. En el servidor tenemos almacenada la localización de la residencia. De esta manera podemos mostrar todas las residencias de esta la ciudad especificada.
- Comercialización:
 - Llevar el producto a distintas residencias para ver la aceptación y si lo necesitan.
 - En el caso de que estén interesadas, realizar un marketing a través de las redes sociales.
 - Creación de empresa y las gestiones necesarias para ello.
 - Venta del producto a residencias.

Comercialización

Respecto a la comercialización del proyecto, es totalmente viable vender el producto a las residencias. Pero antes de ellos es necesario darse a conocer en el mercado. A través de las recomendaciones de mi profesor hemos decidido que cuando termine la universidad, continuaré con el proyecto y creare varios perfiles de redes sociales y lo anunciaré. También me gustaría mostrárselo a las residencias para ver qué opinión tienen y si lo necesitan. De esta manera, nos gustaría comprobar el potencial tecnológico y de mercado. Aun así crear una empresa del proyecto, es una idea arriesgada puesto que podría crearse una competencia rápidamente o los usuarios podrían estar descontentos con la aplicación.

Para verificar la viabilidad del proyecto podemos utilizar un análisis DAFO en el cual determinamos las debilidades, amenazas, fortalezas y oportunidades, este nos sirve para darnos cuenta si la idea puede llegar a convertirse en una empresa y comercializarla o es un riesgo muy grande y no es posible.

Una vez verificado el mercado y la viabilidad del proyecto será necesario un plan de negocio (Entrepreneur Media, 2017) donde defina la empresa, el mercado al que va dirigido, los costes de los recursos, los objetivos, los competidores, misión y visión de la empresa, precio del servicio, definir los canales de distribución y comunicación, recursos humanos. Grosso modo podemos definir un poco cada uno de ellos:

- La empresa gestionará, distribuirá y mantendrá la aplicación para las residencias. Esta estará ubicada en España, no tendrá un sitio fijo puesto que los empleados trabajaran de forma remota.
- Mercado, entre los posibles cliente podemos definir:
 - Clientes directos: serían las residencias y los portales de anuncios.
 - Clientes indirectos: estudiantes que disfruten de una residencia.
- Entre los recursos necesarios:
 - Servidores para gestionar el cliente y el servidor.
 - Dominio web.
 - Ordenadores.
- Entre los objetivos del plan de negocios podemos destacar:
 - Estar siempre disponibles para nuestros clientes, tanto la web como el soporte.
 - Los usuarios estén en contacto entre sí en todo momento.
 - Gestionar la empresa de manera eficiente.
 - Identificar inversiones sólidas.
- Los competidores es uno de los mayores riesgos del proyecto. Actualmente no existe ninguna aplicación web que permita gestionar residencias y buscar residencias. Por otro lado, si la idea tiene existo, los competidores podrán crearse rápidamente y ganar bastante terreno. Estos competidores son los gestores de reservas de hoteles.
- Misión y visión, la empresa tendría un camino bien definido, el cual sería vender el producto o servicio a residencia y a través de la satisfacción de los estudiantes; más residencias estarían interesadas en formar parte de esta red, puesto que estarían visibles para los estudiantes y serían más conocidas.
- En un principio el servicio será un cuota mensual, dependiente del número de habitaciones de la residencias, la cual sería:
 - 99€ al mes por residencia con menos de 50 habitaciones.
 - 200€ al mes con residencias con menos de 100 habitaciones.
 - 300€ al mes con residencias con 100 o más habitaciones.
- Entre los canales de distribución y comunicación serán utilizados:
 - Anuncios en Google y Facebook.
 - Creación de redes sociales para dar propaganda.

- Aunque el más importante será la distribución “de boca a boca” puesto que los estudiantes y residencias hablarán sobre ellos y todos ellos lo utilizarán.
- Recursos humanos, entre la plantilla necesaria, será escasa y la forma de trabajo será de forma remota, con el objetivo de ahorrar los gasto de mantenimiento y gestión de oficinas.
 - Desarrollador software.
 - Soporte.
 - Consultorías legales, cuando sean necesarias.
 - Consultorías de seguridad software, cuando sean necesarias.

Después del plan de negocios es necesario encontrar inversores para empezar, la financiación. De esta manera intentaremos buscar apoyo de administraciones públicas como una beca o ayuda pública o subvenciones de creación de empresas o *Startup* para recién graduados.

Además hay que decidir qué tipo de empresa seria, hay que tener en cuenta el servicio ofrecido, la financiación, el mercado y los socios, el capital social, responsabilidad. En mi opinión esta sería un tipo de empresa de responsabilidad limitada. Tras decidir qué tipo de empresa hay que tener en cuenta Consejería de Hacienda y Registro Mercantil Provincial. Además de pedir una licencia de actividad y el registro de ficheros de carácter personal en la agencia española de protección de datos (Dirección General de Industria y de la Pequeña y Mediana Empresa, n.d.).

Licencia

En este apartado se hablará sobre la licencia elegida para el proyecto. Además de que otras licencias se podrían haber elegido. Una licencia software vincula a un programador con su código y los requisitos y términos que debes seguir para usar el programa, si es posible. Es decir, esta describen los derechos del código sobre el desarrollador, en qué zona geográfica es válido y durante qué tiempo, además de los deberes del usuario que lo usa. El principal objetivo es que el código tiene *copyright*.

En cuanto a las licencias de código libre o código abierto. Esta permite al cualquier usuario utilizar o modificar o distribuir el programa sin ningún inconveniente. Por otro lado existen licencias que añaden algunas restricciones como no es posible comercializar con el código o siempre debe aparecer el autor del código.

La licencia utilizada es *GNU GENERAL PUBLIC LICENSE, Version 3*. Esta fue elegida puesto que mi proyecto es software libre el cual puede ser utilizado por cualquiera. Lo más importante es que este puede ser comercializado siempre y cuando sea distribuido bajo una licencia GPL, esta restricción es llamada *copyleft*. Podemos decir que esta es la licencia más utilizada y popular. Además esta fue utilizada en el proyecto del sistema operativo Linux, creado por distintos desarrolladores.

Entre las famosas licencias que pensé utilizar podemos destacar (Open Source Initiative, n.d.):

- Eclipse Public License v1.0: es una licencia de código abierto con la especificación de que el programador no se hace cargo de los problemas en la ejecución del código.
- Licencia apache: esta es de código abierto también en la cual siempre debes especificar quien es el autor, es decir, el desarrollador. Tiene derechos de autor a la hora de patentar. El usuario que la use debe tener una licencia Apache también.
- MIT license: es una licencia de código abierto creada por MIT. Es una licencia popular también. Destaca por ser bastante simple permisiva en comparación con otras licencias de software libre. Además los usuarios que la usen el código deben mantener los derechos de autor.

Asignaturas relacionadas

En el desarrollo del proyecto hemos aplicado conocimiento obtenidos en distintas asignaturas a lo largo de la carrera, seguidamente expongo las principales:

- *Fundamentos de programación y programación orientada a objetos* y metodología de la programación, estas son la base de todo proyecto software.
- *Fundamentos de base de datos y estructura de datos* nos ha ayudado crear el diseño de nuestra base de datos y estructuras eficientes.
- *Ingeniera, empresa y sociedad* y *Diseño y gestión de proyectos* nos ha enseñado cómo gestionar un proyecto y como comercializarlo.
- *Sistemas de información basados en web* ha sido esencial para desarrollar el proyecto; desde la creación del cliente, como del servidor junto con la API.
- *Desarrollo de software* ha contribuido con todo el proceso de diseño del software.

Las demás asignaturas, han ayudado de forma indirecta puesto que sin ellas no sería posible.

Conclusión

Como conclusión podemos afirmar que nos hemos enfrentado a un proyecto a gran escala y que hemos podido ser capaces de aprender y llevarlo a cabo. Aunque lo más importante es que tras ver los resultados de nuestra encuesta, no damos cuenta de que hemos creado una aplicación útil, la cual puede ser utilizada en el mercado actual por usuarios reales. Por lo tanto, al final ha sido una satisfacción la creación del proyecto y sin duda continuaremos trabajando en ella.

Como hemos hablado anteriormente hemos aprendido a utilizar nuevas herramientas la cuales no serán de gran utilidad en etapas futuras y hemos mejorado nuestros conocimientos en otras.

A lo largo de todo el proceso hemos tenido en cuenta el usuario, hemos intentado crear una interfaz adecuada para el usuario, con todo lo imprescindible para este. Este tipo de usuario está creciendo en la actualidad, cada vez más los estudiantes acuden a la web para buscar hoteles, residencias, piso para alquilar, etc. Así que este será bien aceptado entre el público.

Bibliografía

- Assurance Technologies, LLC. (Abril de 2017). *passlib.hash.bcrypt*. Obtenido de <https://passlib.readthedocs.io/en/stable/lib/passlib.hash.bcrypt.html>
- Creative Bloq Staff. (Mayo de 2014). Obtenido de <http://www.creativebloq.com/illustrator/create-perfect-favicon-12112760>
- W3C. (s.f.). *Web Content Accessibility Guidelines*. Obtenido de <https://www.w3.org/TR/WAI-WEBCONTENT/>
- Agencia Española de Protección de Datos. (2017). Obtenido de <https://www.agpd.es/portalwebAGPD/canaldocumentacion/cookies/index-ides-idphp.php>
- bootstrap. (s.f.). *bootstrap-select*. Obtenido de <https://silviomoreto.github.io/bootstrap-select/>
- Cabo, N. (19 de Junio de 2016). *Test de 3 segundos para tu página web*. Obtenido de <https://nachocabo.com/2016/06/19/test-3-segundos-web/>
- CCM Benchmark Group. (Marzo de 2017). Obtenido de <http://es.ccm.net/contents/580-diagrama-de-gantt>
- COYIER, C. (April de 2011). *Responsive Data Tables*. Obtenido de <https://css-tricks.com/responsive-data-tables/>
- Creative Bloq Staff . (Mayo de 2014). Obtenido de <http://www.creativebloq.com/illustrator/create-perfect-favicon-12112760>
- Cuestiones Derecho. (2013). Obtenido de <http://cuestionesderecho.es/creacion-del-aviso-legal-y-cumplimiento-de-la-lopd-desde-cero/>
- DarkReading. (Febrero de 2017). *25 Percent of Web Apps Still Vulnerable to Eight of the OWASP Top Ten*. Obtenido de <http://www.darkreading.com/application-security/25-percent-of-web-apps-still-vulnerable-to-eight-of-the-owasp-top-ten/d/d-id/1328142>
- Dirección General de Industria y de la Pequeña y Mediana Empresa. (s.f.). *Crea tu empresa "paso a paso"*. Obtenido de <http://www.creatuempresa.org/es-es/pasoapaso/paginas/creatuempresapasoapaso.aspx>
- Entrepreneur Media. (2017). *Tu Plan de Negocios paso a paso*. Obtenido de <https://www.entrepreneur.com/article/269219>
- Free Logo Design. (s.f.). Obtenido de <http://preview.freelogodesign.org/?lang=EN>
- geriges. (s.f.). *Gestión de Residencias de la Tercera Edad*. Obtenido de <http://www.geriges.com/>
- gestionderesidencias. (s.f.). Obtenido de <http://www.gestionderesidencias.es/>
- Green Software, S.L. (s.f.). Obtenido de <https://student.greensoft.es/es/>
- IEEE Std. 830-1998. (2008). *Especificación de Requisitos según el estándar*.
- Innovaciones Informáticas S.A.. (s.f.). *SOFTWARE PARA RESIDENCIAS*. Obtenido de <http://www.innovaciones-inf.es/software-para-residencias.jsp>
- López-Vidriero, E. S. (2005). *Protección de Datos Personales. Manual Práctico para Empresas*. ICEF Consultores. FC Editorial.
- Nielsen, J. (1994). *Enhancing the explanatory power of usability heuristics*. Boston.
- NITS. (mayo de 2002). *Software errors cost U.S. economy \$59.5 billion annually*. Obtenido de <https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf>
- Object Management Group, Inc. (2015). *IFML: The Interaction Flow Modeling Language*. Obtenido de <http://www.ifml.org/>.
- Open Source Iniciative. (s.f.). Obtenido de <https://opensource.org/>
- OWASP. (March de 2017). *Wiki OWASP*. Obtenido de https://www.owasp.org/index.php/Main_Page
- Rodríguez, M. D. (2008). *Manual de Derecho Informático 10a edición*. Thomson-Aranzadi . scrumdo. (s.f.). <https://app.scrumdo.com/>.
- studentbostader. (s.f.). Obtenido de <https://www.studentbostader.se>
- symfony. (s.f.). *symfony testing*. Obtenido de <http://symfony.com/doc/current/testing.html>

TAW, Centro TIC. (s.f.). *TAW*. Obtenido de
<http://www.tawdis.net/tools/accesibilidad/desktop/?lang=es>

Apéndice

Acrónimos y abreviaturas

- ERS: Estándar de especificación de requisitos
TFG: Trabajo de fin de grado
UML: Unified Modeling Language
IFML: Interaction Flow Modeling Languaje
OMG: Object Management Group
WCAG: Web Content Accessibility Guidelines
W3C: World Wide Web Consortium
DCN Content Delivery Network
LOPD: Ley Orgánica de protección de datos de carácter personal
LAMP: Linux, Apache, MySql, PHP
CSS: Cascading Style Sheets
TPV: Terminal punto de venta
API: Application Programming Interface
TDD: test-drive development

Política de privacidad de Puja Por Tu Resi

De acuerdo a la Ley Orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal (LOPD), se informa a los usuarios que los datos personales que nos faciliten serán incorporados al fichero PujaPorTuResi, el cual está registrado ante la Agencia Española de Protección de Datos, propiedad y responsabilidad para PujaPorTuResi, S.L.

Respecto a la información almacenada y su uso.

- Nombre y apellidos: utilizados para nombrar quien firma el contrato.
- DNI: para identificar a una persona de forma individual y necesario para firmar el contrato.
- Email: para contratar en caso de problemas.

Le avisamos que la información provista por los usuarios es no cedida a terceros para su análisis, esta es utilizada únicamente por la aplicación y la empresa PujaPorTuResi S. L. La información es requerida en el formulario de registro.

Respecto a la seguridad del fichero, según la ley este es de nivel básico. Por lo tanto nuestro sistema debe cumplir los siguientes artículos:

- Art. 89 Funciones y obligaciones del personal
- Art. 90. Registro de incidencias
- Art. 91. Control de acceso
- Art. 92. Gestión de soportes y documentos
- Art. 93. Identificación y autenticación
- Art. 94. Copias de respaldo y recuperación

Le recordamos que en cualquier momento podrá ejercitar los derechos de acceso, rectificación, cancelación, y, en su caso, oposición, enviando una solicitud por escrito, acompañada de una fotocopia de su D.N.I., dirigida a: XXX o mediante la dirección de correo electrónico PujaPorTuResi@gmail.com

Usted garantiza que los datos aportados son verdaderos, exactos, completos y actualizados, siendo responsable de cualquier daño o perjuicio, directo o indirecto, que pudiera ocasionarse como consecuencia del incumplimiento de tal obligación. En el caso de que los datos aportados pertenezcan a un tercero, Usted garantiza que ha informado a dicho tercero de los aspectos contenidos en esta Política y obtenido su autorización para facilitar sus datos a las Sociedades Titulares para los fines señalados.

El titular de la página web y responsable del tratamiento de los datos:

Nombre: PujaPorTuResi S. L.

Dirección: XXX

Teléfono: XXX

Correo: PujaPorTuResi@gmail.com

Política de *cookies* de Puja Por Tu Resi

Cookies es un fichero que se alojan en la memoria del dispositivo cuando se accede a la web para administrar servicios y funcionalidades de la aplicación web. PujaPorTuResi S. L. utiliza solo un *cookies* para la autenticación, este tiene como objetivo que el usuario pueda acceder a la web y que se mantenga conectado de mantera constante hasta el cierre de sesión. Esta es únicamente utilizada por PujaPorTuResi S. L. y en ningún momento por terceros.

Esta *cookies* es de tipo técnica puesto que permite al usuario navegar a través de la web. Además esta es temporal puesto que es eliminada cuando cierra la sesión en el navegador.

Aviso legal de Puja Por Tu Resi

El titular de la página web es PujaPorTuResi S. L. con domicilio en XXX y con número CIF XXX. Los datos se encuentran registrados en el registro mercantil de Granada, XXX.

Respecto a los contenidos mostrados en la web, con información sobre características, ubicación, información de contacto, habitaciones y demás de la residencia son creados por ella misma, de esta manera el titular de la página web no es responsable de la veracidad de dicha información.

Condiciones generales de contratación y uso de PujaPorTuResi

Se expone el documento contractual que regirá la contratación de una habitación de una residencia a través del a empresa PujaPorTuResi S.L. La aceptación del documento conlleva que el usuario:

- Comprende el texto expuesto.
- Que tiene capacidad suficiente para contratar una habitación.
- Que asume todas las obligaciones aquí dispuestas.

Todos los precios de las habitaciones son con impuesto incluidos.

Respecto al proceso de asignación de una habitación funciona a través de un sistema de pujas, el último día de la puja se le asigna la habitación al usuario con más puntos. El usuario firma el contrato por un periodo académico y cada mes tiene que realizar el pago mensual.

El usuario está obligado a aceptar o rechazar la habitación que se le ha sido asignada, además de pagar todos los meses la renta durante su contrato.

La residencia está obligada a alquilarle la habitación al usuario durante el periodo de contratación, además de permitirle el uso de todos los servicios mostrados en la aplicación web.

El proceso de pago se realiza a través de un TPV independiente de la aplicación el cual es seguro. Este permite realizar el pago con tarjeta de crédito.

Una vez que la habitación es asignada el usuario, se genera el contrato el cual tiene que ser firmado por el usuario y subirlo a la plataforma. Este contrato está escrito en español.

A la hora de confirmar el pago de una mensualidad, aparece una pantalla de confirmación. Para asegurar que el pago se ha realizado correctamente.

FAQ de Puja Por Tu Resi

En un principio pensamos hacer un manual de usuario para los distintos usuarios, en cambio, decidimos que las webs hoy en día no tienen este tipo de documento puesto que todo es bastante comprensible de cómo hacer, en otras palabras, es estándar. En cambio sí podemos encontrar preguntas frecuentes para usuario los cuales son primerizos o no tiene claro algún aspecto de la aplicación. Esta tiene sentido hacerla en el contexto de la web. Así la web mostrara un conjunto de preguntas y respuesta las cuales puede ser útiles para el usuario respecto al tema de la web.

Una de la grande utilidades de FAQ es que lo usuario no tienen que escribir un mensaje para hacer un pregunta, puesto que ya tiene la respuesta. Asique, en este apartado encontraremos preguntas y respuesta reales.

A continuación FAQ de nuestra aplicación para el usuario Estudiante:

- ¿Cómo puede solicitar una habitación?
 - Tienes que darte de alta y buscar hasta 5 habitaciones de tu gusto para pujar por ellas y a final de semana puede que se te haya asignado una. En este momento puede aceptar o rechazar la habitación.
- ¿Cómo recojo las llaves?
 - Una vez has firmado el contrato, ponte en contacto con la residencia para recoger las llaves o el día que empieza el contrato debe haber personal de la residencia para ayudarte.
- ¿Quiero registrarme pero dice “usuario no es correcto”?
 - Para el usuario debes introducir tu DNI, este es único para cada persona.
- ¿Puedo pasar mis puntos a otra persona?
 - No, los puntos son personales para cada usuario.
- ¿Cuándo debo devolver las llaves?
 - La residencia te informara sobre el procedimiento, en cambio, comúnmente se realiza el último día de contrato.
- ¿Cuándo se debe hacer el pago de la mensualidad?
 - Este debe hacerse entre el día uno y día cinco de cada mes.
- ¿Cómo puedo poner una falta a la residencia?
 - Si tienes cualquier problema con la residencia, a través de la aplicación puede presentar una incidencia al a residencia.
- ¿Cuántos puntos necesito para pujar?
 - No existe un mínimo, puede pujar hasta el mismo día que te registras, en cambio la habitación se le asignara al que tenga más punto de todas las pujas en la mismas fechas.
- ¿Puedo reservar una habitación aunque no sea un estudiante?
 - Es un servicio destinado a estudiantes, en cambio, puede ser utilizado por cualquier persona.
- ¿Qué es “incluido en el pago de la mensualidad”?
 - El pago del contrato incluye: los gasto de agua y luz y la habitación. Además del equipamiento de cada habitación y residencia como wifi o comedor o gimnasio y demás.
- ¿Cuándo me registre escribí mal mi correo electrónico es posible cambiarlo?
 - Si es posible cambiarlo desde la web, en la ventana del perfil.
- ¿Cómo funciona el sistema de puntos?
 - Cada día desde que te registrar almacenas un punto. Es decir, desde el primer día empiezas a colecciónar puntos.
- ¿Cuántas ofertas de habitaciones puedo tener?

- Solo es posible tener una oferta por estudiante.
- ¿Si tengo un contrato puedo pujar?
 - No, puesto que ya dispones de una habitación.
- ¿Cómo puedo firmar el contrato?
 - Bastante sencillo, solo tienes que descargar el contrato, firmarlo y subirlo a la web.
- ¿Cuándo se realiza la asignación de habitación?
 - El viernes a las 00:00 de cada semana se realiza la asignación de habitaciones.

Respecto al usuario residencia, existe un soporte el cual les permite realizar cualquier pregunta o queja. Este estará disponible a través de correo o teléfono.

Despliegue

Para que el proyecto se puede utilizar en cualquier ordenador con sistema operativo Linux, es necesario los siguientes pasos:

- Instalar LAMP (Linux, Apache, MySQL, PHP): de esta manera instalamos el servidor apache, Mysql y PHP.
 - Mientras el proceso el *pront* saltara para añadir la contraseña para el usuario del gestor de base de datos.
- Despliegue del servidor: clonamos el proyecto de Github e instalamos todos los requerimientos necesarios a través de *Composer*.
 - Creamos la base de datos para la API y actualizamos los datos.
 - Ejecutamos el servidor en segundo plano.
- Desplegamos el cliente en la carpeta por defecto del servidor apache */var/www/html/*. Una vez clonado el repositorio de Github, ya podemos usar nuestra aplicación web.
- Ahora podemos abrir: http://127.0.0.1/web_puja_por_tu_resi/ y empezar a usarla.

En el repositorio podrás encontrar el script “dependencies.sh” necesario para ejecutar lo dicho anterior mente.

https://github.com/softwarejimenez/API_puja_por_tu_resi/blob/master/dependencies.sh

Formulario de Puja Por Tu Resi

5/7/2017

Formulario proyecto PujaPorTuResi

Formulario proyecto PujaPorTuResi

En este formulario recopilaremos la opinión y la nota que distintos usuarios fija en nuestro aplicación. Todos los contenidos serán para beneficio estudiantil para la realización de un TFG.

Es necesario utilizar el sistema antes de contestar a la preguntas.

*Obligatorio

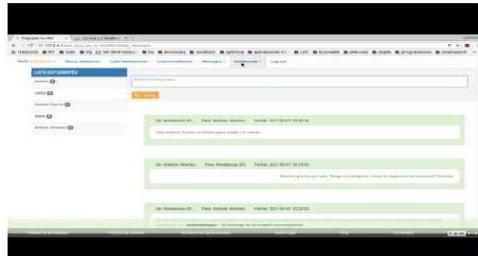
DEMO PujaPorTuResidencia vista Estudiante



<http://youtube.com/watch?v=kliUUXpZlq0>

Es video te será de ayuda para contestar a las preguntas que aparecen a continuación

DEMO PujaPorTuResidencia vista Residencia



http://youtube.com/watch?v=i3NzDvx_N2g

Es video te será de ayuda para contestar a las preguntas que aparecen a continuación

1. ¿Eres o has sido estudiante, de que rama? *

Marca solo un óvalo.

- SÍ, CIENCIAS
- SÍ, CIENCIAS SOCIALES Y JURIDICAS
- SÍ, ARTES Y HUMANIDADES
- SÍ, CIENCIAS DE LA SALUD
- SÍ, INGENIERÍA
- No

5/7/2017

Formulario proyecto PujaPorTuResi

2. ¿Has vivido en una residencia alguna vez? *

Marca solo un óvalo.

- Sí
 No

3. ¿Utilizarías la aplicación? *

Marca solo un óvalo.

- Sí
 No
 Tal vez

4. ¿Recomendaría la aplicación? *

Marca solo un óvalo.

- Sí
 No
 Tal vez

5. ¿Cuál es la parte más útil e interesante en tu opinión? *

6. ¿Cuál es la parte menos útil e interesante en tu opinión? *

7. ¿Te parece una herramienta competitiva en el mercado? *

Marca solo un óvalo.

- Sí
 No

8. ¿Qué nota le pondrías a la aplicación? *

Marca solo un óvalo.

1 2 3 4 5 6 7 8 9 10

Diseño

En este apartado se realizaran cuestiones relacionadas con el diseño.

9. El tamaño de la letra es adecuado *

Marca solo un óvalo.

- Sí
 No

10. ¿Tiene un diseño cómodo? *

Marca solo un óvalo.

- Sí
 No
 Otro: _____

11. Entiendes el funcionamiento o has necesitado ayuda en algún modulo: *

Selecciona todos los que correspondan.

- Lo he entendido todo.
 No he entendido el módulo PERFIL
 No he entendido el módulo PAGOS
 No he entendido el módulo INCIDENCIAS
 No he entendido el módulo MENSAJES
 No he entendido el módulo Mi HABITACIÓN
 No he entendido el módulo BUSCAR HABITACIÓN
 No he entendido el módulo INICO SESION y REGISTRO
 No he entendido el módulo PUJAS
 Otro: _____

Contenido

En este apartado se realizaran cuestiones relacionadas con el diseño.

12. Te gustaría utilizar algún modulo extra: *

Selecciona todos los que correspondan.

- Menu comedor
 Actividades organizadas
 Chat grupal de todos los estudiantes
 Modulo de pago con Paypal
 Otro: _____

13. Añadirías alguna pregunta extra a FAQ *

Marca solo un óvalo.

- No
 Otro: _____

Trabajo Fin de Grado Ingeniería Informática: Puja Por Tu Resi
Antonio Jiménez Martínez

5/7/2017

Formulario proyecto PujaPorTuResi

14. Consideras que el contenido esta organizado de forma correcta *

Marca solo un óvalo.

- Sí
 No
 Otro: _____

15. Necesitarías mas información sobre las habitaciones o la residencia. En caso afirmativo añádelo. *

Marca solo un óvalo.

- No
 Otro: _____

16. Necesitarías mas información sobre las incidencias. En caso afirmativo añádelo. *

Marca solo un óvalo.

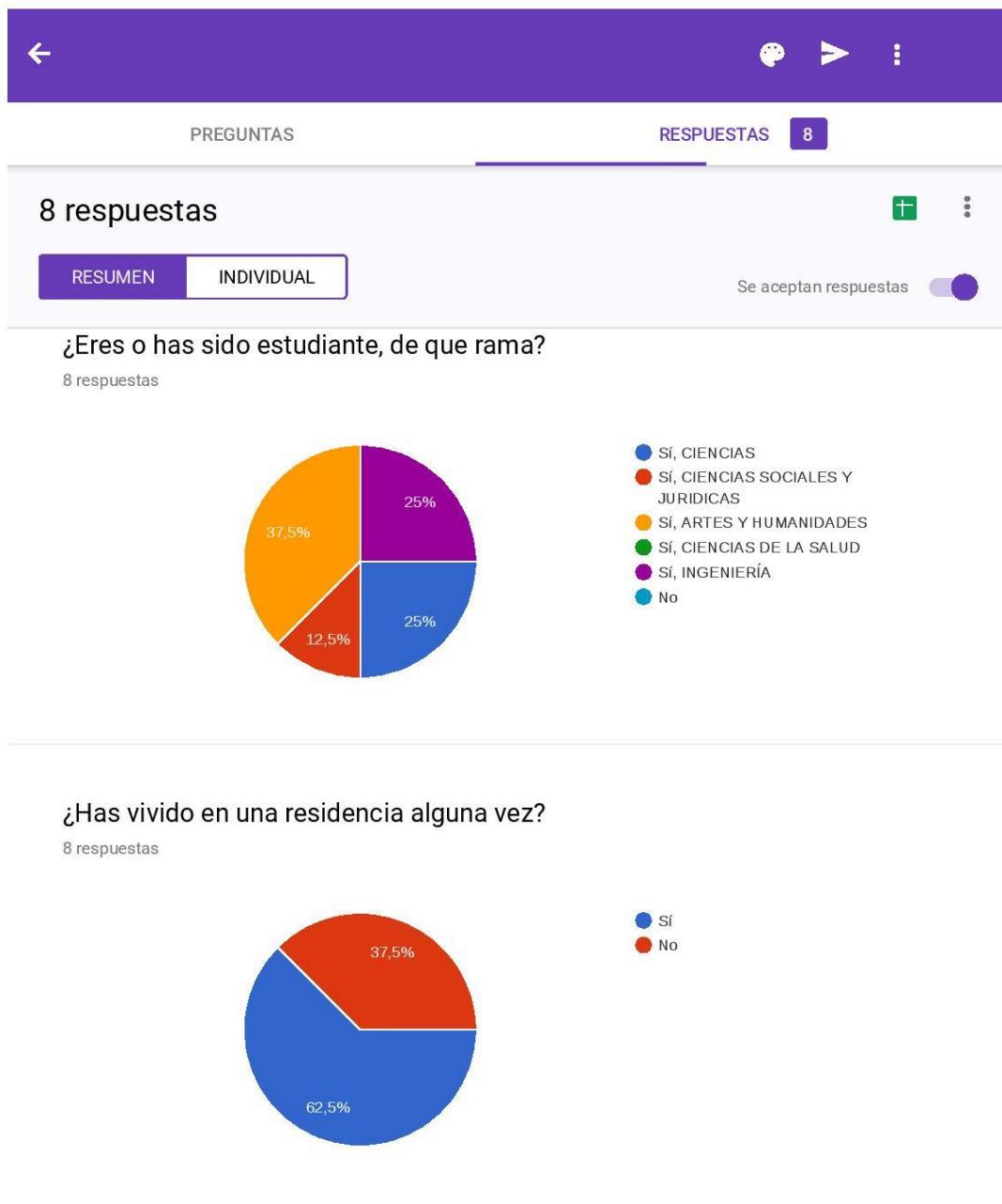
- No
 Otro: _____
-

Con la tecnología de
 Google Forms

Resultados encuesta de Puja Por Tu Resi

5/7/2017

Formulario proyecto PujaPorTuResi - Formularios de Google

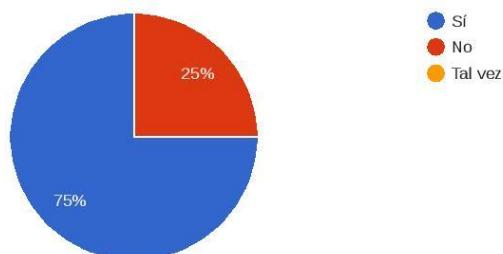


<https://docs.google.com/forms/d/1r9kUGCWhCMr0C0QVOFvm72DMStBpFlvW7lqz9F-AxUs/edit#responses>

1/7

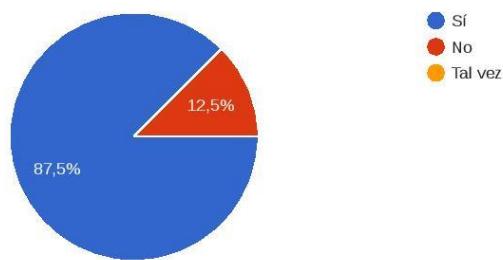
5/7/2017

Formulario proyecto PujaPorTuResi - Formularios de Google



¿Recomendaría la aplicación?

8 respuestas



¿Cuál es la parte más útil e interesante en tu opinión?

8 respuestas

las pujas

El poder elegir la habitación que mejor me convenga sin tener que buscar en mil sitios diferentes

Puedo estar en contacto con la residencia en todo momento.

Puedo hacerlo todo desde el móvil. Muy simple.

bien

No es necesario ir a la residencia para gestionar el proceso.

Podría buscar una habitación interesante.

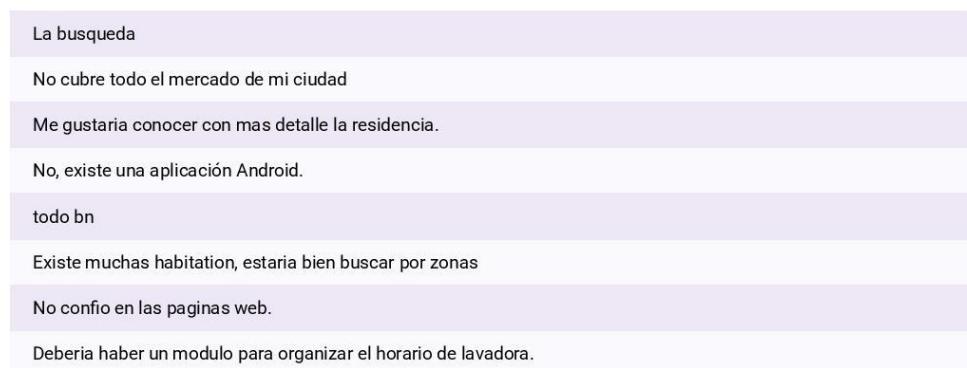
No es necesario ir a todas las residencias

5/7/2017

Formulario proyecto PujaPorTuResi - Formularios de Google

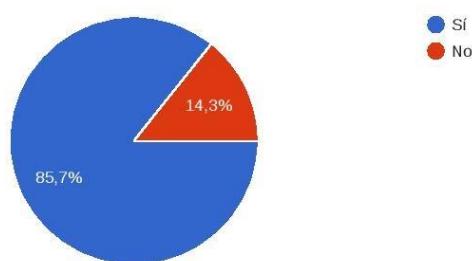
¿Cuál es la parte menos útil e interesante en tu opinión?

8 respuestas



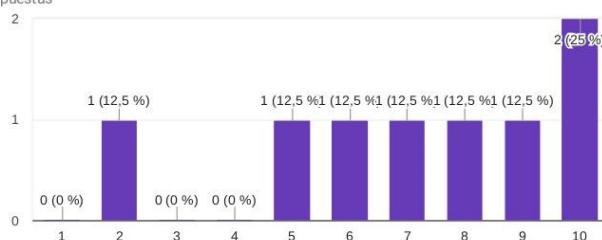
¿Te parece una herramienta competitiva en el mercado?

7 respuestas



¿Qué nota le pondrías a la aplicación?

8 respuestas



5/7/2017

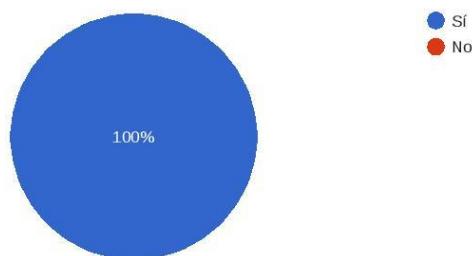
Formulario proyecto PujaPorTuResi - Formularios de Google

Q

Diseño

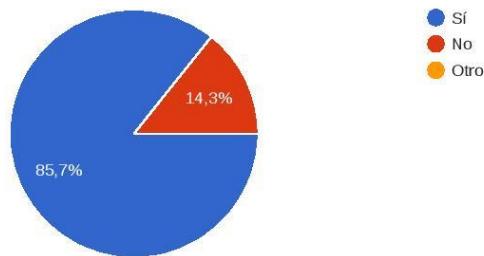
El tamaño de la letra es adecuado

8 respuestas



¿Tiene un diseño cómodo?

7 respuestas



Entiendes el funcionamiento o has necesitado ayuda en algún modulo:

8 respuestas

5/7/2017

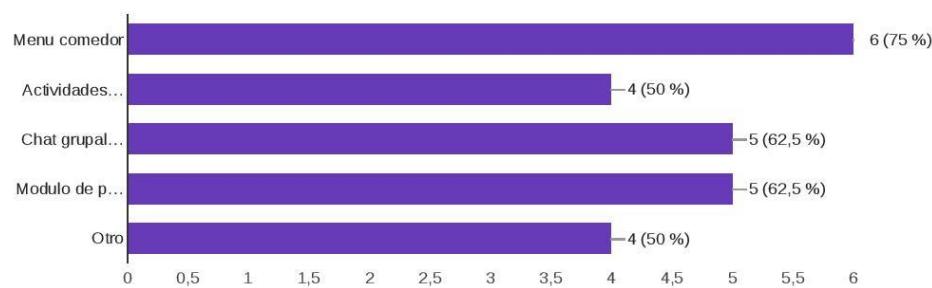
Formulario proyecto PujaPorTuResi - Formularios de Google



Contenido

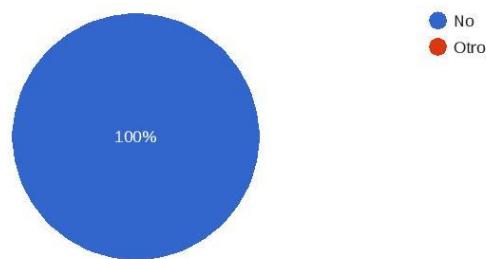
Te gustaría utilizar algún modulo extra:

8 respuestas



Añadirías alguna pregunta extra a FAQ

8 respuestas

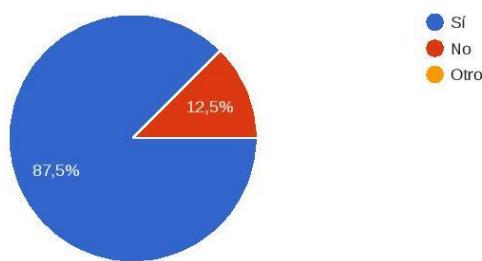


5/7/2017

Formulario proyecto PujaPorTuResi - Formularios de Google

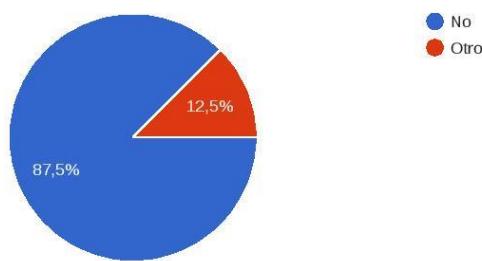
Consideras que el contenido esta organizado de forma correcta

8 respuestas



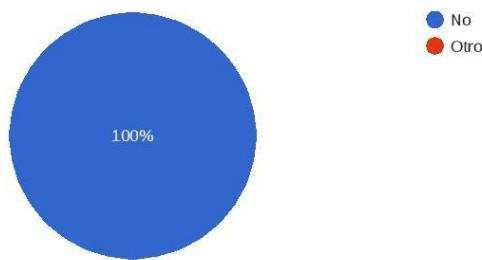
Necesitarías mas información sobre las habitaciones o la residencia. En caso afirmativo añádelo.

8 respuestas



Necesitarías mas información sobre las incidencias. En caso afirmativo añádelo.

7 respuestas



others

POST[/Agreement/accept/](#)

This method accept a Agreement between a student and a room. Can be called by user (student).

Documentation [Sandbox](#)

Requirements

Name	Requirement	Type	Description
room_id		Integer	Id of the room.
file_agreement_signed		File	file agreement signed
agreement_id		Integer	agreement id

POST[/Agreement/assignedRooms/](#)

This method assigned offered room to a student (with more point) the day of the bid. Can be called by user (ADMIN) automatically.

Documentation [Sandbox](#)**POST**[/Agreement/create/{room_id}/{username}/{bid_id}](#)

This method create a Agreement between a student and a room. Can be called by user(ADMIN) automatically.

Documentation [Sandbox](#)

Requirements

Name	Requirement	Type	Description
room_id		Integer	Id of the room.
student_username		String	Username of a student
bid_id		String	bid_id with the date_start_school and date_end_school
username			

GET**/Agreement/download/{filename}**

Download agreement file. Can be called by user (Student/College/ADMIN).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
filename		String	Filename agreement

GET**/Agreement/getCurrentSigned/**

Get the current agreement signed. That function is called by a user (student).

Documentation Sandbox**GET****/Agreement/getList/**

Get the current agreement signed. That function is called by a user (student).

Documentation Sandbox**POST****/Agreement/remove/**

This method remove a Agreement between a student and a room (the agreement which is valid according to the dates). Can be called by user (Student).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
room_id		Integer	Id of the room.
agreement_id		Integer	agreement id

POST /Agreement/removeUnsigned/

This method remove a Agreement which is not signed for one week. Can be called by user (ADMIN).

Documentation Sandbox**GET** /Agreement/roomVerifyUnsigned/{room_id}

Verify if a room has a agreement without signed. Return the agreemnt. Can be called by user (College).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
room_id			

POST /Bank/activate/{id} This method activate a bank account of the college. Can be called by user (College).**Documentation** Sandbox**Requirements**

Name	Requirement	Type	Description
id		Integer	ID of the bank account

POST /Bank/create/ This method create a bank account for a college. Can be called by user (College).**Documentation** Sandbox**Requirements**

Name	Requirement	Type	Description
IBAN		string	IBAN of the bank account.
BIC		string	BIC of the bank account.
account_holder		string	account_holder of the bank account.

GET	/Bank/get/	Get list of banks of a user (College). Can be called by user (College).
-----	------------	---

Documentation Sandbox

POST	/Bank/remove/{id}	This method remove a bank account of the college. Can be called by user (College/ADMIN).
------	-------------------	--

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
id		Integer	ID of the bank account

POST	/Bid/create/	This method create a bid by the user (Student). An user cannot bid more than 5 times neither bid twice for the same room. Can be called by user (Student).
------	--------------	--

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
room_id		Integer	Id of the room .
date_start_school		datetime	Date when the user (Student) starts to live and pay.
date_end_school		datetime	Date when the user (Student) stops to live and pay.

GET	/Bid/get/{id}	Get data of a bid. Can be called by user (College/Student/ADMIN).
-----	---------------	---

Documentation Sandbox

Requirements

Name	Requirement	Type	Description

bid_id	Integer	Id of the bid .
id		

GET /Bid/getBidsRoom/{id}

Get all the bid of a room by its id. Can be called by user (College/Student/ADMIN).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
room_id		Integer	Id of the room .
id			

POST /Bid/remove/{id}

Remove a bid. Can be called by user (College/Student/ADMIN).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
bid_id		Integer	Id of the bid .
id			

POST /Bid/removeBidRoomStudent/

Remove the bid of a user (student) above a room by its id. Can be called by user (Student).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
room_id		String	Id of the room .

POST /Bid/removeBidsRoom/{id}

Remove all the bids of a room by its id. Can be called by user (College/ADMIN).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
room_id		String	Id of the room .
id			

POST /Bid/removeBidsStudent/{username}

Remove all the bids of a student by username. Can be called by user (ADMIN) automatically.

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
username		String	username of the student .

POST /Incidence/create/

This method create a incidence. Can be called by user (Student).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
description		String	Description of the incidence
file_name		File	File with the picture. Image format.

GET /Incidence/download/{filename}

Download file of the incidence. Can be called by user (College/Student).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
filename		String	filename of the file to download

GET /Incidence/get/ Get list of incident of a user in a JSON format. Can be called by user (Student/College).

Documentation Sandbox

GET /Incidence/getNumberOpen/ Get number of incident In status OPEN. Can be called by user (College).

Documentation Sandbox

POST /Incidence/updateState/

This method update the state of a incidence. Can be called by user (College/ADMIN).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
id		Integer	ID of the incidence
status		String	New state of the incidence. state=['OPEN' or 'IN PROGRESS' or 'DONE']

GET /Message/countUnread/

Get number of unread message of a user. Can be called by user (College/Student).

Documentation Sandbox

GET /Message/countUnreadStudent/

Get number of unread message from all the student of a college. Can be called by user (College).

Documentation Sandbox**POST** /Message/create/

This method create a message by the user (Student/COLLEGE). Can be called by user (College/Student).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
message		String	Text of the message, max 5000 character.
file_attached		File	File of the attachedfile file. It is optional. It can be image or pdf format.
username_student		String	Optianl atrribute. If the college create the message, need the student. .

GET /Message/download/{filename}

Download attached file of the message. Can be called by user (College/Student).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
filename		String	Filename of the file to download

GET /Message/get/ Get list of messages of a user (Student || College). Can be called by user (College/Student).**Documentation** Sandbox**Requirements**

Name	Requirement	Type	Description

username_student	String	Optianl atribute. If the college create the message, need the student. .
------------------	--------	--

POST /Message/openAll/

This method set ReadBy=true of a all the messages of user. Can be called by user (College/Student).

Documentation Sandbox

POST /Message/openStudent/{username_student}

This method set ReadByCollege=true of a all the messages of user with a specific student. Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
username_student			

GET /ProfileCollege/get/ Get data of the user (College). Can be called by user (College).

Documentation Sandbox

GET /ProfileCollege/getStudentComplete/{username_student}

Get student information get: list_rents, room, agreement, student_data. Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
username_student		String	Username of the student

GET /ProfileCollege/getStudents/ Get all the student of a college. Can be called by user (College).

Documentation Sandbox

GET /ProfileCollege/getStudentsComplete/

Get all the student of a college. For every student get: list_rents, room, agreement, student_data. Can be called by user (College).

Documentation Sandbox

POST /ProfileCollege/updateAddress/

This method update the address of a user (College). Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
address		String	Address of the College
lat		float	Latitude of the Address of the College
long		float	Longitude of the Address of the College

POST /ProfileCollege/updateEmail/

This method update the email of a user (College). Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
email		String	New email of the user

POST**/ProfileCollege/updateEquipment/**

This method update the equipment of a user (College). Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
wifi		boolean	True if the college has wifi.
elevator		boolean	True if the college has elevator.
canteen		boolean	True if the college has canteen.
hours24		boolean	True if the college has hours24 receptions.
laundry		boolean	True if the college has laundry.
gym		boolean	True if the college has gym.
study_room		boolean	True if the college has study_room.
heating		boolean	True if the college has heating.

POST**/ProfileCollege/updatePassword/**

This method update the password of a user (College). Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
old_password		String	Old password of the user
new_password		String	New Password of the user

POST**/ProfileCollege/updateProfile/**

This method update the email,telephone,url and equipment of a user (College). Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
------	-------------	------	-------------

email	String	New email of the user
telephone	String	Telephone of the user
URL	String	URL of the user
wifi	boolean	True if the college has wifi.
elevator	boolean	True if the college has elevator.
canteen	boolean	True if the college has canteen.
hours24	boolean	True if the college has hours24 receptions.
laundry	boolean	True if the college has laundry.
gym	boolean	True if the college has gym.
study_room	boolean	True if the college has study_room.
heating	boolean	True if the college has heating.

POST /ProfileCollege/updateTelephone/

This method update the telephone of a user (College). Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
telephone		String	Telephone of the user

POST /ProfileCollege/updateURL/

This method update the URL of a user (College). Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
URL		String	URL of the user

GET /ProfileStudent/get/

Get data of the user (student) : name, username, email, ROLE, date_creation, point. Can be called by user (Student).

Documentation Sandbox

POST /ProfileStudent/updateEmail/

This method update the email of a user (student). Can be called by user (Student).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
email		String	New email of the user

POST /ProfileStudent/updatePassword/

This method update the password of a user (student). Can be called by user (Student).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
old_password		String	Old password of the user
new_password		String	New Password of the user

POST /Rent/createAll/{username_student}

This method create all the rents of a user during his agreement. One per moth. Can be called by user (ADMIN) automatically

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
username_student		String	Username of the user Student

GET**/Rent/download/{filename}**

Download receipt file. Can be called by user (Student/College).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
filename		String	Filename receipt

GET**/Rent/get/**

Get list of rents of a user. Can be called by user (Student/College).

Documentation Sandbox**GET****/Rent/getReveiverBankAccount/**

Get the activate bank account of the college. Can be called by user (Student).

Documentation Sandbox**GET****/Rent/getStudent/{username_student}**

Get list of rent of a user (Student). In JSON format. Can be called by user (College).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
username_student		String	Username of the student

GET**/Rent/getUnpaid/**

Get list of rents without pay of a user (Student). Can be called by user (Student).

Documentation Sandbox

POST /Rent/pay/

Pay the rent. * This method generate file_receipt. * update the date_paid and set the cardNumber and cardHolder. Can be called by user (Student).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
id		integer	ID of the rent
idTransaction		string	idTransaction of the simulator TPV

POST /ResponsiblePerson/create/

This method create a ResponsiblePerson for a college. Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
DNI		string	DNI of the responsible person.
email		string	email of the responsible person.
name		string	name of the responsible person.
job_position		string	job_position of the responsible person.

GET /ResponsiblePerson/get/

Get list of responsiblePersons of a user (College). Can be called by user (College).

Documentation Sandbox

POST /ResponsiblePerson/remove/{DNI}

This method remove a responsible person of the college. Can be called by user (College/ADMIN).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
DNI		String	DNI of the responsible person

POST /Room/create/ This method create a room for a College. Can be called by user (College).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
name		String	Name of the room by the college.
price		float	Price of the room per month.
floor		integer	floor of the room.
size		integer	size of the room in m ² .
picture1		File	Image File of the room.
picture2		File	Image File of the room.
picture3		File	Image File of the room.
tv		boolean	True if the room has a tv.
bath		boolean	True if the room has bath.
desk		boolean	True if the room has a desk.
wardrobe		boolean	True if the room has a wardrobe.

GET /Room/download/{filename}

Download pictures rooms. Can be called by user (College/STUDENT/ADMIN).

Documentation Sandbox**Requirements**

Name	Requirement	Type	Description
filename		String	Filename receipt

GET	/Room/get/{id}	Get room of the id of a user (College). Can be called by user (College).
-----	----------------	--

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
id			

GET	/Room/getAll/	Get list of rooms of a College. Can be called by user (College).
-----	---------------	--

Documentation Sandbox

GET	/Room/getAllCompanyName/
-----	--------------------------

Get the companyName of all the colleges. This function can be called by User (College/Student/ADMIN).

Documentation Sandbox

GET	/Room/getSearch/
-----	------------------

Get all the colleges with the data of college and all the OFFERED room. The college and the room should pass the restrictions: price, equipment, specific_college. This function can be called by User (College/Student/ADMIN)

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
college_company_name		String	Name of the college (companyName).
price_max		float	Price max of the room per month.
price_min		float	Price min of the room per month.
tv		boolean	True if the room has a tv.
bath		boolean	True if the room has bath.

Name	Requirement	Type	Description
desk		boolean	True if the room has a desk.
wardrobe		boolean	True if the room has a wardrobe.
wifi		boolean	True if the college has wifi.
elevator		boolean	True if the college has elevator.
canteen		boolean	True if the college has canteen.
hours24		boolean	True if the college has hours24 receptions.
laundry		boolean	True if the college has laundry.
gym		boolean	True if the college has gym.
study_room		boolean	True if the college has study_room.
heating		boolean	True if the college has heating.
date_start_school		datetime	Date when the user (Student) starts to live and pay.
date_end_school		datetime	Date when the user (Student) stops to live and pay.

GET /Room/getSearchAll/

Get all the colleges with the data of college and all the OFFERED room. This function can be called by User (College/Student/ADMIN).

Documentation Sandbox

POST /Room/remove/{id} Remove room of the id of a user (College). Can be called by user (College).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
id			

GET /Security/checkSesion/

This method verify if a user (which role) is connected in the system. Can be called by user (College/Student).

Documentation Sandbox

POST /Signin/college/ This method sign up a user (College) in the system. It is not necessary to be authenticate.

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
username		String	Username of the user (CIF)
password		String	Password of the user
email		String	Email of the user
companyName		String	Company name of the college.
address		String	Address of the College
lat		float	Latitude of the Address of the College
long		float	Longitude of the Address of the College
telephone		String	Telephone of the user
url		String	Url name of the college.
wifi		boolean	True if the college has wifi.
elevator		boolean	True if the college has elevator.
canteen		boolean	True if the college has canteen.
hours24		boolean	True if the college has hours24 receptions.
laundry		boolean	True if the college has laundry.
gym		boolean	True if the college has gym.
study_room		boolean	True if the college has study_room.
heating		boolean	True if the college has heating.

POST /Signin/student/

This method sign up a user (Student) in the system. It is not necessary to be authenticate.

Documentation Sandbox

Requirements

Name	Requirement	Type	Description

username	String	Username of the user (DNI)
password	String	Password of the user
email	String	Email of the user
name	String	Complete name of the user

GET | POST /login This method allow to a user to login the systme. Can be called by user (College/Student).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
_username		String	Username of the user (DNI/CIF)
_password		String	Password of the user

GET | POST /remmemberPassword

This method allow to a user to remmeber the password the systme. Can be called by user (College/Student).

Documentation Sandbox

Requirements

Name	Requirement	Type	Description
_username		String	Username of the user (DNI/CIF)