Trabajo Fin de Grado

Memoria

Plataforma integrada de pagos para la Universidad de Zaragoza

Payments integrated platform for the Zaragoza University

Autor

Samuel Gascón Gascón

Director y ponente

Pascual Pérez Sánchez (director)
Responsable técnico de la unidad de Administración Electrónica de la Universidad de Zaragoza

Joaquín Ezpeleta Mateo (ponente)
Departamento de Informática e Ingeniería de Sistemas

Escuela de Ingeniería y Arquitectura
2018

Plataforma integrada de pagos para la Universidad de Zaragoza

1.Resumen

Para permitir que los ciudadanos que así lo desean puedan relacionarse con la Universidad de Zaragoza usando exclusivamente medios electrónicos, es necesario disponer de una infraestructura de pago de tasas por medios electrónicos.

Un sistema de pago electrónico, seguro, flexible y multiplataforma es uno de los pilares básicos sobre el que debe apoyarse el servicio de Administración Electrónica.

En este momento la Universidad de Zaragoza ofrece sistemas de pago electrónico encastrados en algunos procedimientos concretos de administración electrónica como la solicitud de certificados académicos o la solicitud de título, pero de cara a generalizar la posibilidad de pago electrónico se necesita ofrecer un servicio genérico de pago de tasas, independientemente del procedimiento.

Al sistema desarrollado se le va a llamar Plataforma Integrada de Pagos para la Universidad de Zaragoza (*PIPUZ*).

El acceso y uso del sistema se hará utilizando una interfaz web para administradores y usuarios y una *API* de integración con las aplicaciones. Está compuesta de 5 módulos:

1. Panel de administración del sistema

2. Gestión de tasas, comercios y unidades contables

3. Gestión de usuarios del sistema y pagos

4. Integración con pasarelas de pago virtual (*TPV* y *Paypal*)

5. *API* de acceso al sistema para otras aplicaciones y procesos

El sistema está preparado para integrarse con los principales sistemas de información existente en la Universidad de Zaragoza: sistema de autenticación y autorización de usuarios, sistema de gestión de perfiles de tramitación, sistemas de gestión contable, sistemas de continuidad de los servicios (backups, alarmas, etc), sistema de gestión documental, etc.

La definición y especificación de la solución se ha realizado siguiendo la metodología *Scrum*.

El desarrollo de la solución se ha realizado utilizando *PHP* y siguiendo el paradigma *MVC* (Modelo Vista Controlador). Para el modelo se ha utilizado una base de datos *MySQL* con ayuda del *framework* de *PHP PDO*. Para la vista, se ha realizado una interfaz web con *HTML*, *CSS* con *Bootstrap* y *JavaScript* con *AngularJS*. Por último, para la parte del controlador, se ha utilizado una tecnología *REST* implementada en lenguaje *PHP* utilizando como herramienta el *framework Slim*.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

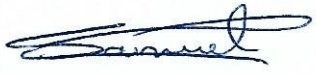
D./Dª. Samuel Gascón Gascón,

con nº de DNI 76971217S en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Plataforma integrada de pagos para la Universidad de Zaragoza

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 31 de agosto de 2018



Fdo: Samuel Gascón Gascón

Índice

2.Introducción

2.1. Objetivo y alcance

La gestión de pagos y la emisión de recibos en la Universidad de Zaragoza está completamente descentralizada siendo problemática tanto para *Ibercaja* (banco que recibe los pagos que se realizan a la Universidad de Zaragoza), como para el departamento de contabilidad de la universidad. Por ello, este trabajo se ha realizado con el objetivo de **desarrollar una aplicación para la gestión de pagos y emisión de recibos, de una manera unificada, en la Universidad de Zaragoza.**

El desarrollo de este trabajo se ha realizado dentro de la Universidad de Zaragoza en la unidad de Administración Electrónica y es la base para crear *PIPUZ*, que será la Plataforma Integrada de Pagos para la Universidad de Zaragoza.

Para que la solución resulte eficaz, todas las unidades de la Universidad de Zaragoza que deban emitir recibos o realizar cobros deberían utilizar la aplicación o integrar sus sistemas con ella.

Por otro lado, un objetivo adicional de este proyecto es el de mejorar los conocimientos del estudiante que lo desarrolla, donde ha aprendido a realizar un proyecto en el mundo laboral, a trabajar colaborativamente, y a utilizar las metodologías y las tecnologías estudiadas en el desarrollo de una aplicación web.

2.2. Soluciones existentes

Esta aplicación se incluye en el marco del comercio electrónico. El comercio electrónico es la compra y venta de productos o de servicios a través de medios electrónicos (Internet o redes informáticas). Al principio el término se aplicó a la realización de transacciones mediante medios electrónicos como el intercambio electrónico de datos, pero con la revolución de Internet y de la *World Wide Web*, a mediados de los 90, comenzó a aplicarse a la venta de bienes y servicios a través de Internet usando como pago medios electrónicos como las tarjetas de crédito.

El comercio electrónico más famoso que existe hoy en día es *Amazon*[1]. *Amazon* existe desde 1994, y comenzó siendo una tienda de libros pero pronto se diversificó a más productos. Otras famosas plataformas de comercio electrónico actuales son *Ebay*[2], *Aliexpress*[3], *Rakuten*[4]...

El comercio electrónico también se ha implantado en las páginas web de los comercios físicos, teniéndose que amoldar a las necesidades de los consumidores. Un ejemplo claro que fue pionero y tuvo un gran éxito, fue *Barrabés*[5], un comercio de esquí y montaña en los Pirineos que hoy en día es un referente para empresarios y escuelas de negocio.

1 https://www.amazon.es/
2 https://www.ebay.es/
3 https://es.aliexpress.com/
4 https://www.rakuten.es/
5 http://www.barrabes.com

Acercándose más al objetivo del proyecto, que es el de la gestión de pagos de las tasas de una administración pública, se encuentran varias plataformas.

Un ejemplo claro de este tipo de plataformas es el de la sede de la DGT[6], en donde existe la posibilidad de pagar tasas como la expedición de permisos de circulación, la obtención del permiso de conducir, la autorización de autoescuelas o centros... Esta plataforma da la posibilidad de elegir una o varias tasas para después pagarlas indicando el *NIF*/*CIF*, email y la forma de pago, que puede ser por cuenta corriente o por tarjeta. A continuación, una vez insertados los datos, será necesario firmar digitalmente para completar el pago.

Otra plataforma de estas características se encuentra en la sede electrónica del Gobierno Vasco[7]. Esta plataforma sirve para pagar las cartas de pago emitidas por el Gobierno Vasco a través de la web. En primer lugar, hay que indicar un código (el *Código de Procedimiento de Recaudación*) que se indica en la carta de pago recibida. A continuación se indican los datos que aparecen en la carta tales como el emisor, referencia, identificación e importe. Después se mostrará el listado de los pagos para posteriormente elegir una de las entidades financieras adheridas al sistema y pagar.

Dentro de la Universidad de Zaragoza, existen varias formas de pago, que dependen de las aplicaciones que los generan y las unidades o servicios que los emiten. Así, por ejemplo el pago de tasas de matricula está dirigido por la aplicación de gestión académica y permite el pago por domiciliación o pago telemático, para las tasas para las pruebas de acceso se emiten recibos normalizados para el pago en entidad bancaria, para el pago de concursos se utilizan cartas de pago sin normalizar y en otros casos se permite el pago directo por datáfono.

Haciendo un trabajo de recapitulación nos encontramos con una gran variedad de situaciones, favorecidas en buena media por la gestión descentralizada de la emisión de recibos y de la gestión de cobros existente en la Universidad de Zaragoza. Esta variedad dificulta el uso de sistemas de pago telemático e impide disponer de una plataforma centralizada para la emisión de recibos.

En este momento se estima que se hacen unos 23000 ingresos en ventanilla a favor de la universidad, de los cuales utilizan un recibo normalizado (norma bancaria de la *Asociación Española de Banca*) unos 16000.

La herramienta que se ha desarrollado no va a eliminar por si misma esta dispersión pero favorecerá sin duda un proceso de catalogación, regularización y normalización de todos estos pagos.

2.3. Metodologías y tecnologías utilizadas

Las metodologías que se utilizan más habitualmente en el mundo laboral, son las llamadas metodologías ágiles. Después de estudiar diferentes posibilidades se optó por utilizar la metodología *Scrum*[1], dado que es una metodología fácil de llevar a cabo y en la que hay numerosas interacciones entre el cliente y los desarrolladores para ir abordando los requisitos del sistema poco a poco o poder ir cambiándolos según las exigencias del cliente.

6 https://sede.dgt.gob.es/es/

7 https://euskadi.eus

También se han tenido que estudiar las diferentes posibilidades de elección de las tecnologías y herramientas para la implementación de la aplicación. En primer lugar, se ha decidido que la aplicación siga un patrón modelo vista controlador, en donde se separe completamente la vista, que es una aplicación web, el controlador, basado en un *API REST*[2], y el modelo que incluye toda la gestión y tratamiento de datos usados por el sistema.

Para la interfaz web se han utilizado los lenguajes *HTML*, *CSS* con el *framework* Bootstrap y JavaScript con el *framework AngularJS*. Para el *API REST* se ha decido implementarla en *PHP* con ayuda del *framework Slim*[3]. Por último, en la parte del modelo, se ha desarrollado una base de datos *MySQL*, y se ha utilizado el *framework PDO* de *PHP* para la interacción entre la base de datos y el controlador.

2.4. Organización del documento

Este documento consta de varias secciones en donde se explican las soluciones tomadas para el proyecto y una explicación argumentando el por qué de esa solución.

En primer lugar se explica cómo se ha realizado el proyecto, donde en un principio se hace una breve descripción para, a continuación, explicar la metodología y las tecnologías utilizadas.

A continuación se explican las fases que ha tenido este trabajo. Al haber utilizado como se ha mencionado anteriormente la metodología *Scrum*, se explican los roles que han tenido las personas que han participado en el proyecto. También hay que explicar los diferentes *sprints* (iteraciones) que ha habido y por último las pruebas que se han realizado al sistema.

La última sección está destinada a explicar cuáles han sido los resultados del trabajo, en donde se explica el *API REST* implementado, los diferentes módulos del sistema y la integración con las diferentes formas de pago desarrolladas.

El documento incluye además los siguientes anexos técnicos: 1- instalación del *framework Slim,* donde se explicarán los pasos para instalar y poner en funcionamiento dicho *framework;* 2- base de datos, que describirá las tablas y todos los atributos de la base de datos del sistema; 3- integración de la base de datos con *PDO*, donde se describen todas las funciones que realizan llamadas a las bases de datos; 4- *API REST*, en donde se describirán todas las llamadas a la *API* indicando los parámetros de entrada y la respuesta; 5- carta de pago, donde se explica como se forma una carta de pago; 6- instalación de las pasarelas de pago; explicando paso a paso los procesos necesario para integrar el *TPV* de *Ceca* y *Paypal* a la aplicación; 7- *AngularJS*, en donde se explicará el desarrollo de la interfaz indicando especialmente el intercambio de información entre los controladores de *AngularJS* y el servidor (que en este caso es el *API REST*); 8- diagrama de *Gantt*, donde se explicará detalladamente el cronograma del proyecto; y finalmente 9- pruebas realizadas, tanto las del *API REST,* como las de la interfaz web.

3.Desarrollo del proyecto

3.1. Descripción

La aplicación que se desea implantar en la Universidad de Zaragoza consiste en una aplicación web y en unos servicios web con los que puedan interactuar los diferentes sistemas de información de la universidad.

La aplicación web tiene dos módulos diferenciados. Por un lado está el módulo de usuario y por otro el módulo de administración.

El módulo de usuario está pensado para que los usuarios del sistema elijan tasas de la universidad que desean pagar y para pagar las tasas que se les han asignado a pagar desde una unidad de la organización. Además dispone de un entorno donde visualizar y gestionar toda la información relativa a los pagos entre el ciudadano y la universidad.

El módulo de administración es la herramienta disponible para los gestores de las unidades de tramitación y dispone de varias funcionalidades. En este modulo se podrán ver los pagos realizados por los usuarios y los pendientes de realizar, se podrá crear, actualizar y borrar una tasa, crear usuarios, generar un recibo que ha de pagar un usuario, así como la posibilidad de anotar los pagos realizados al contado por lo interesados, actualizar los pagos realizados en el banco mediante carta de pagos y crear, actualizar y borrar descuentos de las tasas.

La figura 1, muestra un esquema de la arquitectura del sistema. Esta es una arquitectura en tres capas, en donde los usuarios (y los administradores), interaccionan con el sistema a través de la interfaz web, la interfaz web está en contacto con el servidor web (que tendrá que ser *Apache* y en lenguaje *PHP* por orden de la administración electrónica) y este a su vez está conectado con la base de datos (que será *MySQL* también por orden de la administración electrónica).

Además, diferentes aplicaciones de la Universidad, podrán estar conectadas al servicio web, pudiendo realizar las misma llamadas que se hacen desde la interfaz web.

Por último, el servicio web también estará conectado con las diferentes pasarelas de pago y los usuarios también tendrán que interactuar con ellas. Estas pasarelas de pago, tienen que ser *PayPal*[4] y *TPV* de *CECA*[5]. Además, se tendrá que poder descargar cartas de pago en formato *PDF*.

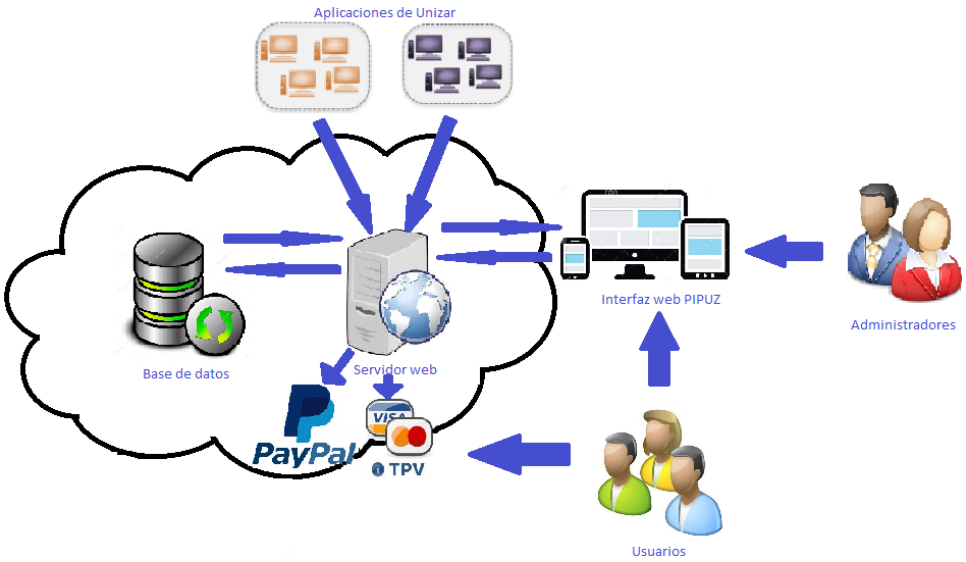


Figura 1: Esquema de la arquitectura del sistema

3.2. Metodología utilizada

Para la realización de este trabajo, ha sido necesario utilizar una metodología para estructurar, planificar y controlar el proceso de desarrollo del sistema de información, valorando la utilización de una metodología en cascada o una metodología ágil basada en *Scrum*.

Se ha tenido en cuenta el modelo en cascada por ser una metodología empleada durante la carrera y ser bien conocida. Esta metodología se basa en una ordenación de las etapas de desarrollo del software, de manera que hasta que no se acaba una etapa no se empieza la siguiente. Las etapas que se pensaron eran: análisis de los requisitos, diseño, implementación, pruebas y mantenimiento. Esta metodología tenía ciertas ventajas y desventajas. Una de las ventajas más importantes es que era un modelo conocido, ya que se había trabajado varias veces con él. Sin embargo, debido a que el proyecto no tenía unos requisitos fijos de antemano y que no se podía llevar una secuencia lineal, sino que había que hacer cambios y pruebas constantemente, se descartó esta metodología.

Valorados los problemas de aplicar una metodología clásica a este caso, se optó finalmente por *Scrum*. Esta metodología se caracteriza por hacer entregas parciales y regulares al cliente y cuenta con un conjunto de buenas prácticas que se apoyan unas en otras y ayudan a trabajar colaborativamente y a obtener un buen resultado. Este conjunto de buenas prácticas son:

- El desarrollo incremental de los requisitos del proyecto en bloques temporales cortos y fijos. En este trabajo, esta práctica ha sido fundamental, debido a que los requisitos no eran fijos, sino que eran cambiantes.

- La priorización de los requisitos por valor para el cliente y coste de desarrollo en cada iteración. Al no tener unos requisitos fijos, en cada iteración el cliente le podía dar una mayor importancia a un requisito u otro.

- El control empírico del proyecto. Por un lado, al final de cada iteración se demuestra al cliente el resultado obtenido, de manera que pueda tomar las

decisiones necesarias en función de lo que observa y del contexto del proyecto en ese momento. En este proyecto, el cliente era el director del trabajo, por lo que se le muestra en cada iteración el trabajo que se ha realizado y puede indicar las decisiones necesarias para mejorar el trabajo realizado o para poder continuar con otro requisito. También sirve para controlar si se está haciendo bien el trabajo o se está haciendo mal.

- La potenciación del equipo, que se compromete a entregar unos requisitos y para ello se le otorga la autoridad necesaria para organizar su trabajo. Con esto, se proporciona un trabajo constante cada iteración y con total libertad para su organización dentro de los requisitos establecidos.

- La sistematización de la colaboración y la comunicación tanto entre el equipo y como con el cliente. Esto ayuda para uno de los objetivos que se han planteado que era el de aprender a trabajar colaborativamente.

- El *timeboxing* de las actividades del proyecto, para ayudar a la toma de decisiones y conseguir resultados. El *timeboxing* es una técnica que fija el tiempo máximo para conseguir objetivos, tomar una decisión o realizar unas tareas. Por ello, esta práctica ayuda en la consecución de los objetivos y fuerza a tomar decisiones por lo que el trabajo será más rápido.

3.3. Tecnologías y herramientas utilizadas

La primera decisión en cuanto a tecnologías para la realización del sistema ha sido la de utilizar el patrón *MVC*[6] para separar completamente los datos, la lógica de negocio y la interfaz de usuario. Con este patrón se distinguen tres componentes diferentes que han de interactuar entre ellos que son el modelo, la vista y el controlador. Por ello, se va a hablar de estos tres componentes por separado.

3.3.1. Controlador

El controlador es la parte más importante del sistema ya que es el nexo de unión entre el modelo, donde se encuentran los datos, y la vista, donde actúan los usuarios.

Para el controlador ha desplegado un servidor web con el que no solo los usuarios puedan interactuar a través de la vista, sino que permita también el acceso de los diferentes sistemas de información que existen en la Universidad de Zaragoza utilizando servicios web.

Para la implementación de los servicios web se va a utilizar *REST* que es una arquitectura que se caracteriza por utilizar llamadas *HTTP* y donde el intercambio de datos puede apoyarse en el uso de varios formatos (*XML, JSON*, etc)

Se ha elegido esta opción frente a otras como por ejemplo *SOAP* porque ofrecen una gran escalabilidad debido a:

- El protocolo *HTTP* no tiene estado ya que en cada mensaje está contenida toda la información necesaria para comprender la petición.

- Se utilizan las operaciones definidas en *HTTP* que son *POST, GET, PUT* y *DELETE*, que se equiparan a las operaciones *CRUD* de una base de datos (crear, leer, actualizar y borrar).

- Los recursos en un sistema *REST* son direccionables únicamente a través de su *URI*, lo que hace que haya una sintaxis universal para identificarlos.

Debido a que el servidor de la administración electrónica es un servidor Apache, se impuso que se desarrollara la aplicación en este. Otra de las imposiciones que hubo acerca del controlador, fue el lenguaje de su desarrollo, que fue *PHP* debido a las siguientes razones:

- Dentro de la Administración Electrónica de la Universidad de Zaragoza, el lenguaje más utilizado es *PHP*, por lo que es el lenguaje más conocido por lo que no hará falta un tiempo de aprendizaje del lenguaje. También, para el futuro, será necesario que la persona responsable del mantenimiento del sistema sea conocedor del lenguaje.

- En la Administración Electrónica ya está instalado un servidor *Apache* con *PHP* y no hará falta instalar ningún otro servidor que sería necesario en otros lenguajes.

- *PHP* tiene la ventaja de que tiene un gran soporte.

- En *PHP*, si un usuario realiza una tarea compleja, la lentitud del sistema sólo afectará a ese único usuario.

- *PHP* es un lenguaje que no necesita el montaje de una estructura por lo que los resultados son inmediatos.

Una vez elegido el lenguaje *PHP*, será necesario elegir un *framework* que ayude a la implementación del *API*. Barajando los diferentes *frameworks* que tiene el lenguaje elegido, se observa que los más famosos y más utilizados son *Symfony* y *Zend*. Analizando estas opciones, se buscan otras alternativas más ligeras para evitar la costosa curva de aprendizaje requerida por estas opciones. La opción que se eligió finalmente fue utilizar *Slim*, debido a que es un *framework* orientado a *REST* ya que soporta métodos *HTTP*. La razón de esta elección es su simplicidad, que ayuda a su aprendizaje y hace que la aplicación se desarrolle rápidamente y con muy poco código.

3.3.2. Modelo

El modelo es la representación de la información con la que el sistema opera, por lo que gestiona los accesos a dicha información tales como inserciones, consultas, borrados o actualizaciones.

Desde la administración electrónica, se impuso desde un primer momento que el sistema gestor de bases de datos fuera *MySQL*, debido a que la universidad cuenta con los servicios de este. Las razones por las que se impuso este sistema fueron las siguientes:

- Experiencia de uso, ya que es utilizado por el personal de la Administración Electrónica y por el alumno.

- Rapidez y ligereza.

- Es un sistema gratuito, y al ser una aplicación piloto (la base para la plataforma de pagos de la universidad), no es necesario utilizar una gran base de datos como Oracle.

Sin embargo, el alumno recomendó que se migrara a *Oracle* debido a la gran cantidad de registros que puede contener en un futuro, la seguridad en la gestión

de datos y transacciones y por ser la base de datos de referencia en el servicio informático de la universidad.

El *framework* elegido para interactuar entre la base de datos y el controlador, es *PDO* de *PHP*, el cual se puede integrar con diferentes sistemas gestores de base de datos, lo que hace que sea fácil su posible futura migración a Oracle. *PDO* es un *ORM* (*object relational mapper*) que hace un mapeo entre las tablas y relaciones de una base de datos relacional y estructuras de datos orientadas a objetos que son las que va a manejar la aplicación.

3.3.3. Vista

La vista es una interfaz web y es lo que los usuarios verán del sistema. Al ser una interfaz web, se va a utilizar *HTML* y *CSS*. Para ello se ha utilizado el *framework Bootstrap*[7]. *Bootstrap* es el *framework* más conocido y utilizado para integrar el uso de *CSS* que se caracteriza por ser fácil e intuitivo, compatible con todos los navegadores, que dispone de un gran soporte por su amplia comunidad de desarrolladores.

El nexo de unión entre la interfaz en *HTML* y el controlador se ha hecho con *JavaScript* utilizando *AngularJS*. La utilización de *AngularJS* se justifica por:

- Solución completa para crear una aplicación cliente en *JavaScript* que incluye: generación de vistas, rutas, organización de componentes en módulos, comunicación con el servidor…

- Reducción de código *JavaScript*.

- Experiencia de uso.

4.Fases del trabajo

Como ya se ha comentado con anterioridad, la metodología utilizada en el proyecto será *Scrum*. Esta metodología se caracteriza por la participación de diferentes personas ocupando cada una de ellas un rol y por la realización del trabajo incrementalmente en bloques cortos y fijos de tiempo denominados *sprints*.

4.1. Roles

Existen tres roles diferenciados en la metodología *Scrum*: el cliente, el *Scrum Master* y los desarrolladores.

- Cliente: en este proyecto, las veces de cliente las hace el director del proyecto, Pascual Pérez, al que se le entregarán los resultados al final de cada *sprint*.

- *Scrum Master*: es el responsable de que el trabajo siga las bases de *Scrum* y se encarga de resolver los obstáculos de los desarrolladores. En este proyecto, el *Scrum Master* también es Pascual Pérez.

- Desarrolladores: en el proyecto ha habido un único desarrollador que es el estudiante Samuel Gascón. Ha sido el encargado de escribir todo el código y de realizar las pruebas correspondientes.

4.2. Sprints

Al principio del proyecto, entre los diferentes miembros del proyecto se definió que los *sprints* durasen como norma general una semana (a excepción de semanas especiales como puente del Pilar, todos los Santos, Navidad...). Por lo tanto, cada 7 días, Pascual Pérez y Samuel Gascón se reunían para demostrar los resultados obtenidos durante la semana anterior y fijar los objetivos que había que cumplir para la próxima reunión.

El planteamiento principal era que el proyecto transcurriese entre septiembre y principios de enero. Sin embargo, la aparición de diferentes nuevos requisitos y algún que otro problema encontrado, hizo que el proyecto se alargara unos dos meses, hasta principios de marzo.

En total, ha habido 22 *sprints* entre el 6 de septiembre, que fue el inicio del desarrollo, hasta el 3 de marzo que ha sido el final. Las tareas realizadas en cada *sprint* se pueden ver en el anexo *Diagrama Gantt.*

Para el control de los objetivos marcados en cada iteración, se ha hecho uso de un sistema *Kanban*. Este sistema se caracteriza por la aparición de tres tableros en donde se encuentran las diferentes tareas que hay que realizar según estén por hacer (tablero llamado “TO DO”), se estén haciendo (tablero con nombre “DOING”) o estén hechas (“DONE”).

4.3. Pruebas

Como se acaba de comentar, en cada *sprint* se realizan pruebas de las funcionalidades que se han implementado, pero sin embargo, se ha decidido realizar unas pruebas más exhaustivas al finalizar toda la implementación, para determinar que el sistema no tiene ningún fallo.

4.3.1. Pruebas en API REST

Para comprobar que el *API REST* funciona correctamente, se han realizado unas pruebas con la herramienta *Postman*[8]*. Postman* es una herramienta que sirve para desarrollar y realizar pruebas de *API*s. Entre sus funcionalidades destacan:

- Tiene una interfaz intuitiva para enviar llamadas, guardar respuestas, agregar test y crear flujos de trabajo.

- Guarda el historial de las llamadas.

- Existe la posibilidad de crear variables.

- Crea colecciones y descripciones de llamadas.

Para las pruebas de la *API* desarrollada, se creará una colección de pruebas, en donde se añadirán todas las llamadas al *API REST.* Una vez añadidas todas las llamadas, se creará para cada llamada un test que verifique que los resultados de dichas llamadas son correctos. A continuación, se creará un archivo *CSV*, que contendrá todas las posibles combinaciones de parámetros que puedan tener las llamadas. Una vez realizado todo esto, se ordenará que *Postman* ejecute las llamadas en un orden determinado, pues hay algunas llamadas que se deben ejecutar antes que otras (por ejemplo hay que iniciar sesión antes de crear un pago).

En el anexo *Pruebas del API REST,* se pueden ver con más detalle las pruebas realizadas.

4.3.2. Pruebas del funcionamiento de la interfaz web

Para la realización de las pruebas de la interfaz web, no se ha empleado ninguna herramienta de automatización de pruebas, por lo que se han realizado manualmente.

Las pruebas realizadas pretenden verificar que todas las funcionalidades que ofrecen la aplicación web, obtienen una respuesta correcta en función de la entrada que se le proporcione al sistema.

Se han realizado las diferentes pruebas: registro de usuario, inicio de sesión, búsqueda de tasa, creación de pago, ver tus pagos, realizar pago, cancelar pago, ver tasas en administración, crear tasas, actualizar tasas, crear pagos a usuarios, actualizar pagos y crear, borrar y actualizar descuentos.

En el anexo *Pruebas de la interfaz web,* se pueden ver con más detalle las pruebas realizadas.

4.4. Cronología

El desarrollo del proyecto ha tenido una duración de 6 meses, en los que se ha ido anotando el trabajo realizado en cada sprint, empezando como norma general los miércoles (día de reunión entre el desarrollador y el cliente). Cómo se ha comentado anteriormente, se han realizado 22 *sprints,* que han transcurrido entre el 6 de septiembre y el 3 de marzo.

En la Figura 2 se puede ver el cronograma del proyecto. En el anexo *Diagrama Gantt*, se puede ver un diagrama *Gantt* con más detalle.

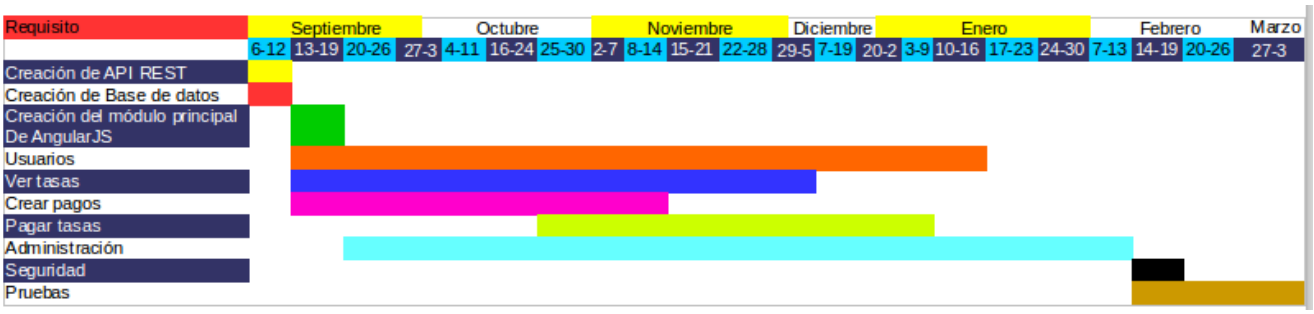


Figura 2: Cronograma del proyecto

5.Resultados del trabajo

A continuación se detallarán las diferentes partes con las que cuenta el sistema. En la Figura 3, se puede ver un esquema de la aplicación.

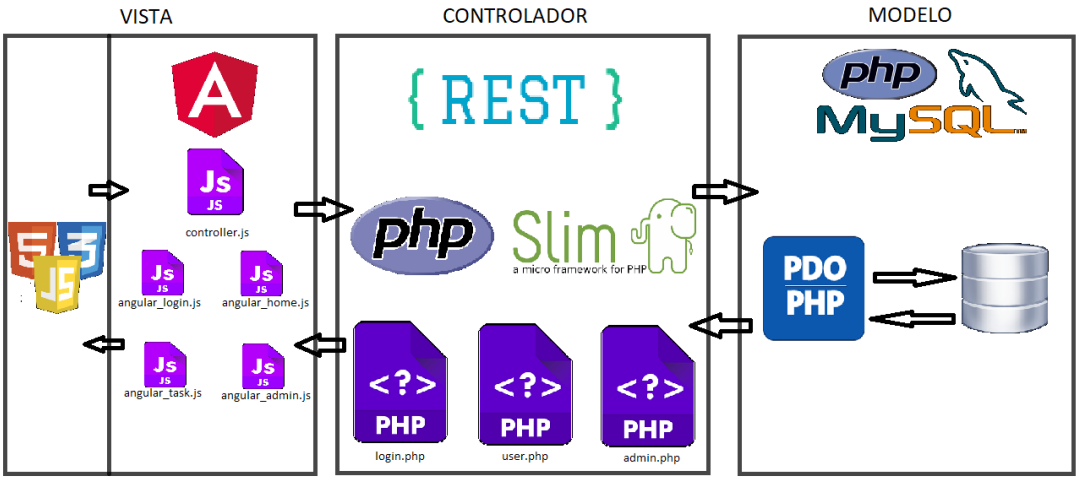


Figura 3: Esquema de la aplicación

5.1. Modelo

Como se ha comentado anteriormente, la parte del modelo está compuesta por una base de datos *MySQL* y la utilización del *framework PDO* para el mapeo de las consultas a la base de datos.

El modelo de la base de datos de la plataforma se puede ver en la *Figura 4*. Como se puede comprobar, hay un total de once tablas. En esas tablas representan todos los elementos que forman parte de la aplicación.

Los datos que forman parte del proyecto son los departamentos de la Universidad, los usuarios, los administradores de cada departamento, las tasas de cada departamento de la Universidad que los usuarios pagan, los atributos y descuentos que tiene cada tasa, los pagos que realizan los usuarios y las posibles formas de pago que existen en la plataforma.

En los anexos *Modelo* e *Integración de la base de datos con PDO*, se muestra con más detalle tanto la base de datos, como la utilización del *framework PDO.*

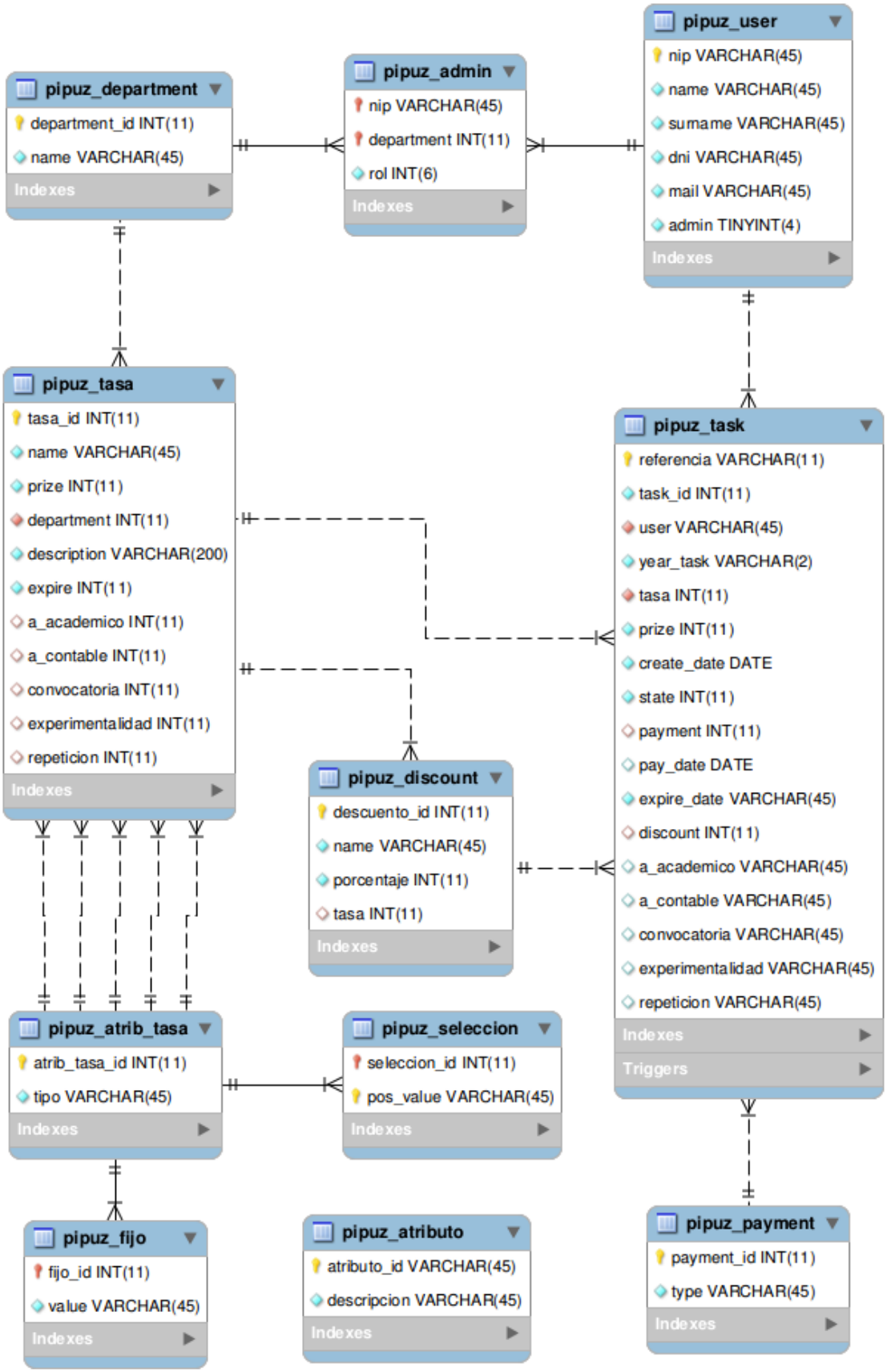


Figura 4: Modelo de la base de datos de PIPUZ

5.2. API REST

Como se ha comentado en los anteriores puntos, se ha decidido que se va a utilizar un servicio web para el desarrollo del controlador para así poder integrarlo también con los diferentes sistemas de la Universidad de Zaragoza que emitan recibos o realicen cobros. Este servicio web será un *API REST* implementado con el *framework Slim*, en el que se han diferenciado tres módulos diferentes.

En primer lugar está el módulo de *login*. Este módulo sirve principalmente para el inicio de sesión de los usuarios y para el registro en el sistema.

Para el inicio de sesión se deberá de indicar el *NIP* de la Universidad y la contraseña administrativa. Todo ello irá cifrado por *TLS* y se devolverá un *token* de autenticación que deberá ir en la cabecera de todas las llamadas posteriores. La comprobación de que el usuario y contraseña son correctos se harán contra el *LDAP* de la Universidad. Para el registro en el sistema, se deberá indicar el *NIP* de la Universidad y la contraseña administrativa. En caso de no pertenecer a la Universidad de Zaragoza, el usuario se deberá de registrar en ella.

El siguiente módulo es el de usuario, que contiene las llamadas que puede realizar un usuario normal.

Las llamadas destacadas de este modulo son las de crear un pago, ver un pago, ver el historial de pagos, mostrar tasas, buscar una tasa y realizar un pago.

Para la creación de un pago, se deberá indicar la tasa que el usuario desea pagar y los atributos de la tasa en caso de que existan. Para ver un pago, el usuario deberá indicar con los dos primeros dígitos el año en el que se inició el pago y a continuación el identificador del pago. El historial de los pagos se puede ver completo, o filtrado por pagos por pagar, pagados, cancelados o caducados. Al mostrar todas las tasas, existe la posibilidad de filtrarlas por departamento. Si se quiere buscar una tasa, se deberá hacer indicando parte del nombre de dicha tasa, ya que en la base de datos se comprobarán los nombres de las tasas. Por último, para realizar un pago, se podrá hacer con *Paypal*, con *TPV* o pedir una carta de pago.

Por último está el modulo de administración, que sólo podrán realizar llamadas a este los usuarios que tengan permisos específicos. Las operaciones destacadas en este módulo son la creación y actualización de tasas, que los administradores podrán realizar para los departamentos que tengan acceso, crear un usuario, el cuál debe de estar registrado en la Universidad y se le enviará un correo indicando que ha sido registrado en el sistema, crear un pago para un usuario específico, sólo cuando el administrador tenga acceso al departamento de la tasa y podrá ser al contado o para que lo realice el usuario desde su cuenta del sistema, y la creación, actualización y borrado de descuentos para las tasas del departamento al que tengan permiso los administradores.

En el anexo *Instalación del framework Slim,* se muestra con detalle la instalación del *API REST* con el *framework Slim*, y en el anexo *API REST*, se muestran con más detalle todas las llamadas del sistema.

5.3. Módulo de usuarios

El módulo de usuario en este sistema es al que van a poder tener acceso los usuarios normales que quieran realizar un pago. A continuación se va a mostrar las diferentes acciones que se pueden realizar en él.

En un primer lugar, aparecerá una pantalla donde se muestran todos los pagos. Puede que no exista ninguna sesión iniciada en el sistema, por lo que no se podrá realizar ninguna acción. Si se desea realizar alguna acción, se deberá iniciar sesión en el sistema, y en caso de no tener una cuenta, deberás registrarte con el usuario de la Universidad, y en caso de que no estés en la Universidad, deberás de registrarte en ella.

Dentro del módulo de usuario, se pueden realizar tres acciones: crear un pago, cancelar un pago y realizar un pago.

Para la creación del pago existen tres posibilidades:

- Seleccionar una tasa dentro de las tasas disponibles y tasas más populares que aparecen en la página principal. En las tasas disponibles, existe una paginación, en la cual aparecen siete tasas por página.

- Utilizar el buscador que aparece en la parte superior en todas las páginas del sistema. Este buscador es en tiempo real, y a medida que se escriba se van actualizando los resultados.

- Utilizar el botón crear pago, en donde seleccionarás el departamento de la tasa que deseas crear y la tasa correspondiente.

Para acceder a cancelar o a realizar el pago, existen dos posibilidades. En primer lugar, al crear un pago el sistema te redirige a la página del pago. La otra posibilidad, es accediendo al botón superior del sistema “Pagos de <tu nombre>”. A continuación te redirigirá a la pantalla de pagos. En esta pantalla, puedes ver todos los pagos o puedes filtrar por pagos por realizar, realizados, caducados o cancelados. Sólo se pueden realizar acciones sobre los pagos por realizar. Los demás pagos únicamente se pueden ver.

Para cancelar un pago, se deberá pulsar sobre el botón Cancelar pago. Una vez hecho esto, no se podrá realizar ninguna otra opción en esta tarea.

Para realizar el pago, existen tres posibilidades: *TPV*, *Paypal* y *Carta de pago*. En el caso de *TPV* y *Paypal*, se te redirigirá a la seleccionada pasarela de pago que se comentará más adelante. En el caso de la *Carta de pago*, se generará un documento *PDF*, con los datos del pago y un código de barras y un código *QR*, para que el usuario vaya al banco y lo pago desde allí.

5.4. Módulo de administración

En este módulo, únicamente tendrán acceso los usuario que sean administradores. En el módulo habrá que seleccionar el departamento que se quiera administrar, pudiendo únicamente seleccionar los departamentos de los que seas administrador. Dentro de este módulo existen diferentes apartados y los administradores podrán acceder según los roles que tengan en el departamento seleccionado anteriormente.

Hay ocho apartados en el módulo:

- Pagos: En esta sección, se ven todos los pagos que se han realizado en el departamento. Existe la posibilidad de realizar una búsqueda de un pago en tiempo real.

- Crear tasa: Aquí, se podrán crear tasas dentro del departamento indicando nombre, descripción, precio, caducidad y con la posibilidad de añadir atributos.

- Actualizar tasa: En este apartado, se seleccionará una tasa del departamento y se podrán modificar su descripción, precio y días de caducidad, así como modificar, borrar o añadir atributos.

- Crear usuario: Un administrador, tendrá la posibilidad de crear un usuario. Al usuario se le comunicará por email que ha sido registrado en el sistema.

- Generar recibo: Los administradores también podrán crear un pago a un usuario específico seleccionando la tasa. Al usuario, se le avisará por email y se le creará el pago en su pantalla de pagos del módulo de usuarios.

- Pago al contado: En caso de que se realice el pago de una tasa al contado, se indicará el usuario y la tasa. Al usuario, se le avisará por email y se le creará el pago con estado pagado en su pantalla de pagos del módulo de usuarios.

- Actualizar Carta de Pago: Los administradores tendrán que indicar aquí los pagos que se han realizado en el banco por medio de una *Carta de Pago* indicando el identificador del pago, el usuario y la fecha del pago.

- Descuentos: En esta pantalla se podrá crear, actualizar o borrar un descuento de una tasa.

5.5. Seguridad

Un aspecto muy importante a tener en cuenta en una aplicación web es el tema de la seguridad y especialmente cuando nos encontramos con una aplicación en la que se realizan pagos.

Para empezar, el servidor en el que se ha realizado la aplicación piloto, cuenta con un certificado para poder realizar las llamadas cifradas por *TLS*. Cuando la aplicación se suba a un entorno de producción, también será importante que cumpla con este requisito.

Por otro lado, a la hora de realizar los pagos, las dos pasarelas de pago utilizadas, *TPV de CECA* y *PayPal,* son pasarelas de pago seguras y fiables que tienen un gran control de la seguridad.

También es importante evitar las vulnerabilidades web. Se han tratado dos posibles vulnerabilidades web: inyección SQL[9] y *XSS*[10] *(Cross Site Scripting).*

- Inyección SQL: Esta vulnerabilidad trata sobre escribir en parámetros de entrada de la página web, código en *SQL,* de tal manera que se realicen acciones sobre la base de datos que el sistema no deja realizar a los usuarios. Para ello se han implementado dos contramedidas que consisten en validar cualquier entrada de datos al sistema, escapando cualquier carácter especial (añadir delante el simbolo \), y añadir los datos a la consulta SQL en tiempo de ejecución.

- *XSS:* Es una vulnerabilidad mediante la cual un atacante logra inyectar *scripts* en la página web. Se utiliza el comando *htmlspecialchars* para evitar esta vulnerabilidad.

5.6. Integración con pasarelas de pago

Existen cuatro formas de pago dentro de la plataforma: al contado, carta de pago, *TPV* y *PayPal*. En este apartado se van a explicar tres: carta de pago, *TPV* y *PayPal*.

5.6.1. Carta de pago

La carta de pago es un recibo que se emite al usuario desde la web para que este la lleve al banco y la pague desde allí. Esta carta de pago es imprescindible que cumpla con la norma n.º57[11].

Para la creación de la carta de pago, se crea un *PDF* desde el *API REST,* convirtiendo un *HTML* gracias a la librería *Dompdf.*

Dentro de la carta de pago, se informarán de los datos de la Universidad, de la tasa y del usuario que quiere efectuar el pago. En la figura 5, se puede comprobar un ejemplo de carta de pago.

Los campos que contiene el archivo son:

- Dirección y *CIF* de la Universidad.

- Último día de pago: Es el último día que tiene validez la carta de pago para proceder a pagarlo.

- Emisora-sufijo: Es el *CIF* de la Universidad.

- Referencia: Número de referencia de la tasa a pagar.

- Importe: Valor del importe total a pagar.

- Fecha de expedición: Fecha en la que se creó la tarea del pago.

- Titular: Persona que realiza el pago.

- *DNI/NIE/CIF*: Número de identificación del titular.

- Descripción: Descripción del pago.

- Código de barras: Código de barras para realizar el pago. Se crea con la librería *Picquer.*

- Desglose: Descripción detallada del pago.

- Código *QR*: Código *QR* para realizar el pago. Se crea con la librería *PHP QR Code.*

- Modos de pago: Descripción de como se puede realizar el pago.

En el anexo *Carta de pago,* se explican los pasos para la creación del código de barras, del código *QR* y del paso de *HTML* a *PDF.*

Universidad Zaragoza

RECIBO DE PAGO
Bill for payment

Pedro Cerbuna 12 50009 Zaragoza - España CIF Q-5018001-G

ÚLTIMO DÍA PAGO Last day of payment	EMISORA-SUFIJO Emitter - Sufix	REFERENCIA Reference	IDENTIFICACIÓN Identification	IMPORTE Amount
06/09/2018	05018001	18683537034	077720	EUR***130

NÚMERO Number	FECHA DE EXPEDICIÓN Date of Issue	TITULAR Holder	DNI/NIE/CIF Passport or ID number
28635	30/08/2018	Samuel Gascón Gascón	76971217S

DESCRIPCIÓN
Description
Pago de la tasa para participar en el trofeo rector

CÓDIGO DE BARRAS
Barcode

DESGLOSE
Detatch
Pago de la tasa para participar en el trofeo rector

MODOS DE PAGO
Payments Ways
Acceda a Ibercaja para proceder al pago

EJEMPLAR PARA LA ENTIDAD FINANCIERA
Copy for bank

Figura 5: Ejemplo de Carta de pago

5.6.2. TPV

El *TPV* utilizado ha sido el *TPV* de *Ceca* debido a que existe un acuerdo de la Universidad con *Ibercaja* que trabaja con este *TPV*.

Para la implementación de la integración del sistema con este *TPV* se ha utilizado la librería de *Ceca* para *TPV* de *PHP*.

La solución elegida para el desarrollo de la integración, tiene las siguientes fases:

- Petición del usuario para la utilización del *TPV*.

- Generación de un formulario automático para el usuario en donde aparecerán fijos la información necesaria del *TPV* y la información de la tasa a pagar.

- Envío del formulario por parte del usuario. Automáticamente el usuario se redirigirá a la página web del *TPV*.

- El usuario deberá de rellenar los datos de su tarjeta para poder realizar el pago correctamente.

- Si todo esta bien, el *TPV* enviará al servidor los datos del pago. En caso contrario cancelará el pago.

- En el servidor, se comprueba que la tarea del pago existe en el sistema por el usuario indicado y si el importe es el correcto. En caso correcto da la confirmación al *TPV* de que ejecute el pago y se actualiza la base de datos. En caso contrario se cancelará el pago.

- El *TPV* devolverá la confirmación al usuario y podrá volver al sistema.

En el anexo *Instalación del TPV de CECA,* se explica en detalle su instalación.

5.6.3. PayPal

La otra pasarela de pago que se ha decidido utilizar es *PayPal*, debido a que es uno de los medios de pago online más utilizados para compras online en todo el mundo, y por ser un medio rápido, cómodo y seguro. Todo aquel que quiera utilizar este medio como pago, tiene que disponer de una cuenta en este entorno.

Para esta implementación, se ha utilizado la librería de *PayPal PayPal-PHP-SDK* para *PHP*.

Los pasos que se siguen para realizar un pago son los siguientes:

- El usuario solicita que se desea realizar un pago mediante *Paypal*, indicando la tasa.

- Desde el servidor, se genera una *URL*, que se enviará al usuario para realizar el pago en la plataforma.

- Desde la web de esta aplicación, al recibir la *URL* mencionada anteriormente, al usuario se le redirigirá automáticamente a esta.

- El usuario deberá de acceder a su cuenta de *PayPal* y confirmar el pago.

- A continuación, si todo ha salido bien, se le enviarán al servidor los datos del pago. En caso contrario, se cancelará el pago.

- Una vez recibidos los datos en el servidor, se comprobará que la tarea del pago existe en el sistema por el usuario indicado y si el importe es el correcto. En caso correcto da la confirmación a *Paypal* para que se ejecute el pago, actualizará la base de datos y se le indicará al usuario que el pago se ha realizado correctamente. En caso contrario se cancelará el pago.

En el anexo *Instalación del TPV de Paypal,* se explica en detalle su instalación.

6.Conclusiones

Revisando los objetivos iniciales y el alcance establecido para el proyecto, se ha superado ampliamente la idea inicial de desarrollar una aplicación piloto. Los desarrollos realizados, a falta de algunos ajustes en tiempo de implantación, incluyen todas las funcionalidades que se esperaban en una aplicación de este tipo y el resultado final es plenamente operativo.

El objetivo principal era crear la base de la aplicación *PIPUZ* (Plataforma Integrada de Pagos para la Universidad de Zaragoza), que en un futuro se utilizará en la Universidad de Zaragoza para poder gestionar pagos y emitir recibos de una manera unificada.

Durante el trabajo, por parte del equipo de la administración electrónica, se me ha dado una serie de libertades en la toma de decisiones. Esas decisiones han sido la utilización de la metodología *Scrum*, el desarrollo de un *API REST* con el *framework Slim*, la estructuración de la base de datos y la utilización de *AngularJS*.

Para el futuro, existen dos mejoras que se recomienda realizar. La primera de ellas, que ya está prevista, es la del cambio de base de datos de *MySQL* a *Oracle*, que conlleva a una gran mejora debido a que *Oracle* está mejor capacitado para el trabajo con grandes cantidades de datos, que seguro va a tener esta aplicación en el futuro. La otra mejora consiste en la incorporación de otros métodos de pago como *Stripe,* que es un nuevo método de pago, que está integrado en la propia web sin necesidad de salir a la web de la pasarela de pago empleada, lo que conlleva a un uso más sencillo por parte de los usuarios.

Por otro lado, existían diferentes objetivos personales por parte del estudiante, que también han podido completarse. Se ha conseguido experiencia en la realización de un proyecto en el mundo laboral, se ha aprendido a realizar un *API REST* en *PHP* con el *framework Slim* y se ha adquirido más experiencia en *HTML*, *AngularJS* y *MySQL*. También se ha mejorado la forma de trabajar gracias a la metodología *Scrum*, que ha sido la utilizada en el proyecto.

El trabajo ha tenido dos partes bastante diferenciadas, una de ellas era la interfaz web y la otra el *API REST* con la base de datos. Valorando cada una de las partes, me he sentido más cómodo realizando la parte del *back-end* ya que tengo una gran experiencia en las bases de datos y la programación. Por otro lado, aunque la parte de la interfaz no sea mi gran punto fuerte, ha habido aspectos de los que estoy satisfecho.

Por último, también me siento satisfecho de haber podido trabajar en un proyecto en el mundo laboral, en el que he tenido que estar colaborando constantemente y que seguro que me ayudará en mi futuro.

7.Bibliografía

[1] SCRUM. Accedido el 15/09/2017, de Proyectos Ágiles. Sitio web:

https://proyectosagiles.org/que-es-scrum/

[2] REST. Accedido el 06/09/2017, de Wikipedia. Sitio web:

https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional

[3] Creando una REST Api para PHP con el Framework Slim 3. Accedido el 06/09/2017, de Anexsoft. Sitio web:

http://anexsoft.com/p/106/creando-una-rest-api-para-php-con-el-framework-slim-3

[4] Obed Alvarado, Paypal con PHP. Accedido el 22/11/2017. Sitio web:

https://obedalvarado.pw/blog/integracion-pagos-linea-usando-paypal-php/

[5] TPV de Ceca. Accedido el 25/10/2017, de Cecabank. Sitio web: https://comercios.ceca.es/docs_constpv/img/manual_comercios.pdf

[6] MVC. Accedido el 06/09/2017, de Wikipedia. Sitio web:

https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

[7] Bootstrap. Accedido el 16/09/2017. Sitio web:

https://getbootstrap.com/

[8] Probar Api REST con Postman, Accedido el 16/02/2018, de Los Andes Training. Sitio web:

https://losandestraining.com/2017/08/30/como-probar-una-api-rest-con-postman/

[9] Inyección SQL. Accedido el 25/10/2017, de PHP. Sitio web:

http://php.net/manual/es/security.database.sql-injection.php

[10] Evitar ataque XSS con PHP, Accedido el 25/10/2017, de Manuais. Sitio web:

https://manuais.iessanclemente.net/index.php/Evitar_ataques_XSS_y_CSRF_con_PHP

[11] Norma bancaria 57, Accedido el 31/01/2018, de Caixabank. Sitio web:

https://www.caixabank.es/deployedfiles/empresas/Estaticos/pdf/Transferenciasyficheros/20150115_FOLLETO57.pdf