Facultad

de

Ciencias

DESPLIEGUE Y GESTIÓN MULTITENANT DE UNA APLICACIÓN WEB DE CONTROL DE SOLICITUDES DE ACTIVIDADES

(Deployment and management multitenant of a web applications control activities)

Trabajo de Fin de Grado

para acceder al

GRADO EN INGENIERIA INFORMÁTICA

Autor: Jorge Polanco Peña

Director: Rafael Menendez de Llano Rozas

Co-Director: José Miguel Prellezo Gutiérrez

Septiembre – 2016

ÍNDICE

Lista de tablas

Lista de imágenes

Lista de acrónimos

CIC	Consulting Informático Cantabria
MVC	Modelo Vista Controlador
HTML	HyperText Mark-up Language
CSS	Cascading Style Sheets
AJAX	Asynchronous JavaScript And XML
XML	eXtensible Markup Language
WSGI	Web Server Gateway Interface
ORM	Object Relational Mapping
IDE	Integrated Development Environment
VCS	Version Control System
DVCS	Distributed Version Control Systems
WYSIWYG	What You See Is What You Get
TDD	Test First Development
DBB	Behaviour Driven Development
JSON	JavaScript Object Notation

AGRADECIMIENTOS

A mi familia y amigos por todo el apoyo y confianza que han tenido en mi durante todo este tiempo.

A mis profesores por todo el tiempo y esfuerzo que han empleado para enseñarme todo lo que se, por haberme aconsejado siempre que lo necesitaba y por haberme abierto las puertas a un mundo al que ni siquiera llegaba a ver.

A CIC por permitirme trabajar con ellos y haberme enseñado que trabajar no siempre significa aburrimiento.

RESUMEN

Actualmente, todos los ayuntamientos ofrecen cursos o actividades para el entretenimiento del ciudadano. Inscribirse en estos cursos puede llegar a ser bastante tedioso, ya que es posible que haya que recorrer todo municipio para obtener los datos necesarios para la inscripción.

A causa de esto, muchos ciudadanos no se matriculan en estos cursos, o lo dejan a mitad del proceso porque creen que no merece la pena.

El ayuntamiento de Piélagos y CIC (*Consulting Informático de Cantabria)* se dieron cuenta de este suceso e idearon una aplicación Web que permitiese tanto la inscripción de los ciudadanos a los cursos de forma online como la gestión de las solicitudes.

Esta aplicación está desarrollada en Python [1], implementa una arquitectura MVC (*Modelo Vista Controlador*) y utiliza varios frameworks para su funcionamiento como por ejemplo Flask.

El objetivo de este proyecto es añadir nuevas características a la aplicación Web, además de rediseñarla para que funcione en distintos ayuntamientos dado su éxito.

Palabras clave: Python, Modelo Vista Controlador, aplicación web, cursos

ABSTRACT

Nowadays, all town halls offer courses or activities to entertain the citizens. Signing up for these courses can become quite tedious because you may have to go over all the town to obtain the necessary data for the inscription.

Because of this, many citizens do not register in these courses, or leave them halfway because they believe it is not worth it.

The town hall of Piélagos and CIC (*Computer Consulting Cantabria*) realised this event and they came up a Web application which would allow the citizens to get the registration online and control the activities.

This application is developed in Python [1], It implements a MVC (*Model View Controller*) architecture and uses several frameworks such as Flask.

The objective of this project is to add new features to the Web application, in addition to redesign it so as it works in different town halls given its success.

Keywords: Python, Model View Controller, web application, courses

1. Introducción

Todos los ayuntamientos ofrecen cursos o actividades deportivas para el entretenimiento del ciudadano. Apuntarse a estas actividades puede llegar a ser muy complicado dado que al final puedes recorrer todo el municipio en busca de los requisitos que son necesarios para la inscripción.

El ayuntamiento de Piélagos y CIC observaron que muchos de los ciudadanos no se matriculaban a estos cursos porque el método de inscripción era demasiado complicado. Así que idearon una aplicación Web para que se pudiesen matricular online, sin necesidad de salir de su casa

CIC vio el éxito de esta aplicación Web y desarrolló este proyecto para mejorarla. A continuación, se expondrá una breve descripción de la aplicación, después se comentarán cuáles son los objetivos del proyecto, y por último se indicará cual es la estructura del informe.

1.1. Aplicación Web

La aplicación en si es bastante intuitiva, por un lado, está la sección a la que pueden acceder todos los usuarios. Esta está formada por varias páginas:

- La página principal, que contiene un par de discursos y las últimas noticias que están relacionadas con los cursos ofertados.
- Las noticias, donde se muestran todas las noticias publicadas en la web.
- Los cursos, donde aparece una pequeña descripción de los cursos que puede ser modificada por el administrador del sistema cuando quiera, y un índice de todas las escuelas que ofrecen cursos (por ejemplo: escuela de karate, baile, ajedrez, …).

Luego está la sección para los usuarios que están registrados. Dependiendo del tipo de usuario (ciudadano, monitor, administrador, …) aparecerán unos componentes u otros.

La vista más simple, (y la que se va a comentar en este apartado) es la del ciudadano ya que solo tiene los componentes necesarios para realizar las solicitudes:

- Las notificaciones (que aparece en todos los tipos de usuarios), por lo general, una notificación es cualquier cambio en el estado de una solicitud.
- Las actividades disponibles, donde se muestran todos los datos de las actividades, incluyendo si aún hay plazas o no.
- La creación de solicitudes, donde el usuario puede rellenar una solicitud para cualquier curso.

1.2. Objetivos del proyecto

El objetivo principal del proyecto es convertir la herramienta que desarrolló CIC, en una herramienta de despliegue multi-cliente. Además de este, se han añadido nuevos objetivos, los cuales consisten en mejorar y ampliar las características de la herramienta.

A continuación, se comentarán cuáles son estos objetivos.

- Capacidad para activar/desactivar ciertas funcionalidades de la Web.
- Añadir nuevas funcionalidades a la aplicación actual. Estas funcionalidades se comentarán más tarde en el apartado de requisitos.
- Inclusión de testing a todos los niveles.
- Mejora de la usabilidad de ciertas partes de la aplicación.

A parte de estos objetivos que sirven para completar el proyecto, hay otros objetivos que son necesarios para la realización del proyecto, como, por ejemplo, estudiar el uso de las diferentes herramientas y frameworks.

1.3. Estructura del informe

A continuación, se comentará cual es la estructura del informe de este trabajo fin de grado.

- En el siguiente apartado, el apartado 2, se comentará las diferentes tecnologías y herramientas utilizadas. Además del estado del arte en el momento en el que se desarrolló el trabajo fin de grado.
- El apartado 3 es el más importante de este informe.
  En primer lugar, se analizará el funcionamiento del sistema, para ello se comentarán cuáles son los proyectos que forman el sistema además de los distintos tipos de usuario.
  Luego se comentarán los requisitos, tanto funcionales como no funcionales.
  Después, se explicará cual es la metodología utilizada, además de la arquitectura del sistema.
  Y, por último, se comentará como se desarrolló el proyecto, para ello se utilizará uno de los requisitos como ejemplo
- En el apartado 4 se mostrarán las conclusiones.
- En el apartado 5 se indicarán cuáles son los trabajos futuros.
- Y en el apartado 6 se expondrá la bibliografía.

Por ultimo hay un anexo con un diagrama que muestra el estado actual de la base de datos.

2. Tecnologías y herramientas utilizadas

A continuación, se comentará cual es el estado del arte antes de comenzar el proyecto, después se comentarán cuáles son las tecnologías que utilicé, y por ultimo comentaré cuales son las herramientas que usé.

2.1. Estado del arte

El ayuntamiento de Piélagos observó que muchos de los ciudadanos no se matriculaban en los cursos que ofrecían porque el proceso de inscripción era muy largo, y en muchas ocasiones era innecesario. Por este motivo decidió contratar a CIC para que implementasen una aplicación Web.

CIC desarrolló todos los requisitos que el ayuntamiento de Piélagos pedía y comenzó a investigar.

Tras la investigación, CIC llegó a la conclusión de que los sistemas que encontró, o bien no cumplían los requisitos que el cliente solicitaba, o se salían demasiado del presupuesto. Así que, llegaron a la conclusión de que lo mejor era desarrollar una aplicación Web desde cero, totalmente personalizada para el cliente.

En un principio CIC se planteó la idea de desarrollar la aplicación con las mismas herramientas y tecnologías que utilizaban los otros sistemas, algunos de estos sistemas utilizan como lenguaje de programación C# y varios frameworks como .NET. Pero al final, llegaron a la conclusión de que utilizar otras herramientas y tecnologías de desarrollo software podría beneficiar al rendimiento del sistema.

Tras varias semanas trabajando en la aplicación, CIC implementó la aplicación en el servidor del ayuntamiento de Piélagos.

Al cabo de unos meses se observó que la aplicación era todo un éxito. La cantidad de ciudadanos que se inscribían a los cursos era mucho mayor que cuando no existía este sistema.

CIC se dio cuenta de que este sistema no solo funcionaba para el ayuntamiento de Piélagos, sino que podría funcionar para cualquier ayuntamiento que ofreciese cursos o actividades.

De forma que decidió desarrollar este proyecto para aumentar el alcance de la aplicación.

2.2. Tecnologías utilizadas

Para desarrollar este proyecto han sido necesarias el uso de varias tecnologías. A continuación, se comentarán cuáles son estas tecnologías.

2.2.1. Python

Python [1] es un lenguaje de alto nivel que permite usar varios tipos de programación, como por ejemplo la programación orientada a objetos o la programación funcional.

Este lenguaje tiene varias características, como por ejemplo el tipado dinámico. Además, su sintaxis es parecida a la de Java por lo que su aprendizaje es bastante sencillo.

Imagen 2.1: Logo de Python

2.2.2. HTML, CSS y JavaScript

Todas o casi todas las aplicaciones web utilizan estos tres lenguajes para el desarrollo de su interfaz gráfica.

Imagen 2.2: Logos de HTML, CSS y JavaScript

HTML [2] (*HyperText Mark-up Language*) es un lenguaje de programación basado en etiquetas con las que se pueden diseñar páginas web.

CSS [2] (*Cascading Style Sheets*) es un lenguaje utilizado para dar un estilo a los componentes de la página web.

JavaScript [2] es un lenguaje de programación que permite el desarrollo de páginas web dinámicas

2.2.3. AJAX

Imagen 2.3: Logo de AJAX

AJAX (*Asynchronous JavaScript And XML*) es una tecnología que utiliza JavaScript [2] y XML para realizar llamadas asincronas al servidor, de forma que no sea necesario recargar la página web cada vez que quieras realizar alguna acción, como por ejemplo obtener un dato.

2.2.4. MySQL

MySQL [3] es un sistema de gestión de bases de datos relacional desarrollado por Oracle.

Para acceder a la base de datos se ha utilizado el framework SQLAlchemy.

Imagen 2.4: Logo MySQL

2.2.5. Flask

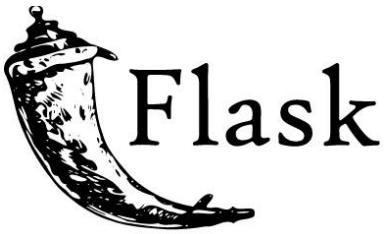


Imagen 2.5: Logo de Flask

Flask [4][5] es un microframework desarrollado para Python [1] y basado en la especificación de Werkzeug [7] y el motor de plantillas Jinja2 [6].

Flask es un microframework porque en un principio te proporciona lo mínimo para desarrollar una aplicación web, y puedes ir añadiéndole extensiones a medida que las necesites.

Este microframework proporciona su propio servidor para que puedas probar la aplicación siempre que quieras.

2.2.6. Jinja2

Jinja2 [6] es un lenguaje diseñado para Python [1] y utilizado en plantillas HTML [2]. Es muy utilizado y tiene varias características como, por ejemplo: la herencia de plantillas, su sintaxis configurable y la facilidad para depurar código.

Imagen 2.6: Logo de Jinja2

2.2.7. Werkzeug

Werkzeug [7] es una librería WSGI (*Web Server Gateway Interface*) para Python [1], es muy utilizada y tiene varias características como, por ejemplo, el uso de un potente depurador interactivo, el uso de objetos HTTP de petición y respuesta, y un soporte Unicode.

Imagen 2.7: Logo de Werkzeug

2.2.8. SQLAlchemy

SQLAlchemy [8] es un ORM (*Object Relational Mapping*) para Python [1]. Este ORM permite convertir los tipos datos de un lenguaje de programación (Python) en los de un sistema de base de datos relacional (MySQL) y viceversa.

Imagen 2.8: Logo de SQLAlchemy

2.2.9. UnitTest

UnitTest es un framework que ya viene implementado en Python [1] y que nos permite la ejecución y el desarrollo de tests para poder ver el funcionamiento de la aplicación.

2.2.10 SweetAlert

SweetAlert [9] es un componente que sustituye las ventanas modales generadas por JavaScript [2] (alert o confirm) por otras que tienen un mejor aspecto.

Imagen 2.9: Logo SweetAlert

2.3. Herramientas utilizadas

Además de estas tecnologías ha sido necesario el uso de varias herramientas que se mostrarán a continuación.

2.3.1. PyCharm

PyCharm [10] es un IDE (*Integrated Development Environment*) utilizado para la programación en Python [1]. Proporciona análisis de código, depuración gráfica, integración con VCS/DVCS y muchas otras características que facilitan el desarrollo de código.

Imagen 2.10: Logo de PyCharm

2.3.2 Alembic

Alembic [11] es una herramienta de migración de base de datos relacional que permite la creación, modificación o eliminación de cualquier elemento (tablas, columnas, índices, etc.).

Alembic proporciona un sistema de versiones y te permite utilizar una versión u otra, facilitándote de esta forma la creación de la base de datos. Además, se encarga de analizar el código y ver las diferencias que hay en la aplicación respecto a la versión anterior de forma que te autogenere los cambios que va a realizar en la siguiente versión.

2.3.3. Selenium

Selenium [12] es una herramienta que permite la ejecución de acciones en una página web, de forma que se pueda probar el funcionamiento de la página.

Para su funcionamiento Selenium utiliza drivers que permite usar un navegador. Aunque con ciertos drivers, como el de PhantomJS no es necesario el uso de un navegador.

Imagen 2.11: Logo de Selenium

2.3.4. Git

Git [13] es un sistema de control de versiones, gratis y de código abierto. Con esta herramienta se podrá subir todo el proyecto a un repositorio de CIC.

Además, Git ofrece varias características como por ejemplo la gestión de ramas de trabajo o la distribución del proyecto a todos los usuarios de forma que aumenta la seguridad en caso de ocurriese algún conflicto de versiones.

Imagen 2.12: Logo de Git

2.3.5. Balsamiq Mockups

Balsamiq Mockups [14] es una aplicación para crear maquetas de interfaces gráficas. Permite al diseñador utilizar distintos widgets pre construidos utilizando un editor WYSIWYG (*What You See Is What You Get*).

Imagen 2.13: Logo de Balsamiq Mockups

3. Desarrollo del proyecto

Tras haber investigado las herramientas y tecnologías utilizadas, y el funcionamiento del sistema, el cliente me planteo el problema. He de dejar claro que el cliente ahora mismo no es el ayuntamiento de Piélagos, sino CIC, ya que es esta empresa la que quiere añadir nuevas funcionalidades al sistema.

En resumen, el cliente quiere convertir el sistema en un sistema multi-cliente, además, quiere añadir nuevas funcionalidades al sistema como, por ejemplo, añadir listas de valores a los formularios.

En este apartado se comentará cual es la estructura del sistema y cuáles son los requisitos del sistema. Además, se hablará de la metodología utilizada y de cómo se desarrolló el proyecto.

3.1. Análisis del sistema

A continuación, se comentará cual es la estructura del sistema y cuáles son los tipos de usuarios que puede haber en el sistema.

3.1.1. Estructura del sistema

El sistema está dividido en tres proyectos:

- El proyecto seve-app, es el proyecto principal y en él se concentra casi toda la lógica de negocio del sistema.
  Como se puede observar en la imagen 3.1, el proyecto tiene varios componentes. A continuación, se explicará brevemente el contenido de los directorios más importantes:
  - El directorio 'alembic' almacena toda la información relacionada con la herramienta Alembic [11], entre esa información se encuentran todas las versiones de la base de datos.
  - El directorio 'tests' almacena los tests unitarios y de integración.
  - El directorio 'app' almacena todos los componentes del sistema como, por ejemplo, los controladores, los modelos, las plantillas y los componentes externos que utiliza.

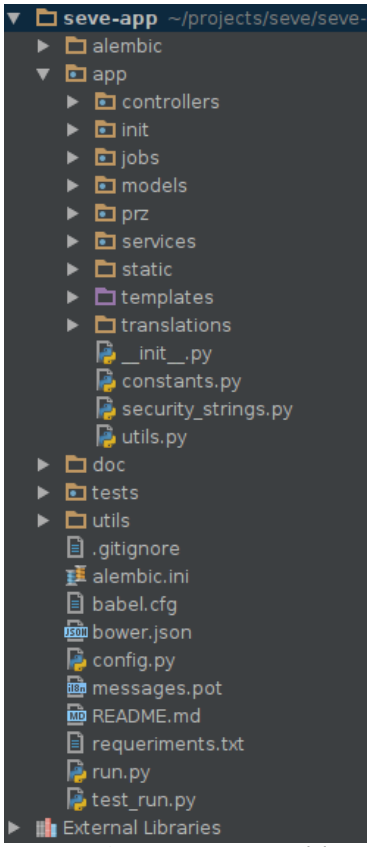


Imagen 3.1: Estructura del proyecto seve-app

- El proyecto seve-demo, es el encargado de inicializar el sistema. Además, almacena toda la información que hace referencia al cliente.
  Gracias a este proyecto es posible desarrollar un sistema multi-cliente. Solo hay que hacer múltiples copias de este proyecto, y modificar los datos del cliente, incluyendo otros datos que se necesitan, como el puerto del servidor donde se inicializa el sistema.

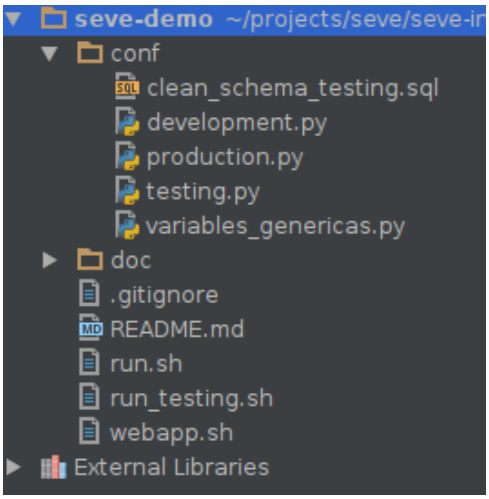


Imagen 3.2: Estructura del proyecto seve-demo

- Y, por último, seve-worker.
  Este proyecto es el encargado de realizar todas las tareas asíncronas como, por ejemplo, el envió de emails.

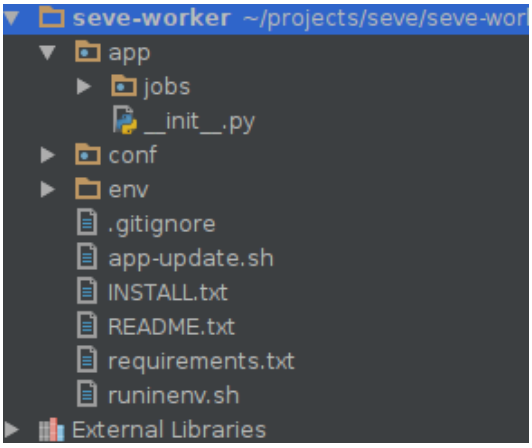


Imagen 3.3: Estructura del proyecto seve-worker

3.1.2 Usuarios del sistema

El sistema está desarrollado de forma que agilice el proceso de inscripción de un alumno en un curso, tanto antes como después de matricular al alumno al curso. Ya que es posible que el monitor del curso (por ejemplo) no acepte a un alumno por alguna razón.

Por esta razón hay cuatro tipos de usuarios en el sistema:

- El cliente/ciudadano: las funciones que puede realizar este usuario es inscribir a cualquier persona al curso y ver cómo va el proceso de inscripción. Además de pagar de forma online (si el usuario quiere) la matricula en caso de que se acepte la inscripción.
- El monitor del curso: este usuario tendrá la decisión final de si acepta a un alumno o no. Tiene acceso a la lista de alumnos y será el que indique si un alumno ha realizado la matrícula de forma manual.

- Los servicios sociales: los usuarios que tengan el rol de servicios sociales se encargarán de gestionar una parte de la inscripción del alumno en caso de que el cliente solicite una ayuda con el pago.
- El administrador: se encarga de gestionar todo el sistema. Para ello puede validar o anular el pago de ciertos alumnos, o modificar cualquier dato que hay en el sistema como, por ejemplo, una tarifa, los detalles de un curso, o la inscripción de un nuevo monitor.

3.2. Requisitos

A continuación, se especificarán todos los requisitos, tanto funcionales como no funcionales, de la forma más clara posible. Para ello utilizaremos la siguiente tabla:

[Id. del requisito]
[Nombre del requisito]
[Descripción del requisito]

Tabla 1: Formato de los requisitos

3.2.1. Requisitos funcionales

RF-01
Sistema multi-cliente
Mover todos los datos que hacen referencia al cliente (y no están guardados en la base de datos) del proyecto seve-app al proyecto seve-demo

RF-02
Creación, modificación y eliminación de listas de valores
El administrador del sistema podrá crear cualquier lista de valores, y podrá modificar o eliminar la lista o su contenido siempre y cuando alguno de los valores no se esté utilizando. **La lista de valores podrá tener hasta un nivel de profundidad 2. A continuación se mostrará un ejemplo.** Lista: Tipos de instrumentos Valores: Instrumentos de cuerda e instrumentos de viento Lista: Instrumentos musicales (lista padre: Tipos de instrumentos) Valores: Flauta (valor padre: instrumentos de viento) y saxofón (valor padre: instrumentos de viento)

RF-03
Lista de tipos de solicitud en la convocatoria
El administrador del sistema podrá seleccionar un conjunto de tipos de solicitud para una convocatoria. **El tipo de solicitud se podrá eliminar de la convocatoria a no ser que se esté utilizando en alguna escuela.**

RF-04
Escoger el tipo de solicitud en una escuela
El administrador del sistema podrá escoger un tipo de solicitud que haya sido escogido previamente en la convocatoria. **El tipo de solicitud se podrá cambiar siempre y cuando no se haya utilizado en alguna solicitud de la escuela.**

RF-05
Tarifa gratuita
El administrador del sistema podrá crear tarifas gratuitas. De forma que cuando el cliente haga una solicitud con esta tarifa, se salte ciertos pasos del proceso de inscripción.

RF-06
Creación, modificación y eliminación de tipos de solicitud
El administrador del sistema podrá crear cualquier tipo de solicitud, y solo podrá modificarla cuando no la utilice ninguna escuela.

RF-07
Creación, modificación y eliminación de bloques de preguntas
El administrador del sistema podrá crear cualquier bloque de solicitud, y solo podrá modificarlos o eliminarlos cuando no lo utilice un tipo de solicitud.

RF-08
Creación, modificación y eliminación de preguntas
El administrador del sistema podrá crear cualquier tipo de pregunta, y solo podrá modificarla o eliminarla cuando el bloque de preguntas no se esté utilizando. **Las posibles respuestas de la pregunta son: número, texto, fecha, lista de valores y email**

RF-09
Implementación de provincias, municipios y localidades
Añadir todas las provincias, municipios y localidades de España al sistema.

RF-10
Personalizar bloques estándares del tipo de solicitud
Configurar los bloques estándares del tipo de solicitud para que el administrador pueda mostrar u ocultar sus preguntas.

RF-11
Configurar vínculos de las redes sociales
Permitir que el administrador pueda configurar los vínculos de las redes sociales que aparecen en la portada.

RF-12
Modificar archivo Excel
Modificar el archivo Excel que guarda las solicitudes realizadas para que también muestre el tipo de solicitud de la solicitud

RF-13
Modificación de ventanas modales
El sistema deberá usar ventanas modales con mejor aspecto

3.2.2. Requisitos no funcionales

RNF-01
Compatibilidad con los navegadores
El sistema deberá funcionar desde cualquier navegador

RNF-02
Tiempo de aprendizaje
La aplicación debe ser lo más intuitiva posible para que el tiempo de aprendizaje sea inferior a 15 minutos

RNF-03
Confidencialidad
La información manejada por el sistema está protegida de acceso no autorizado y divulgación.

RNF-04
Accesibilidad
El tiempo de carga de cualquier componente del sistema no debe ser mayor a 1 segundo

3.3. Metodología utilizada

Antes de comenzar a desarrollar el software hay que escoger una metodología de desarrollo. Para ello hay que tener en cuenta el ciclo de vida del proyecto.

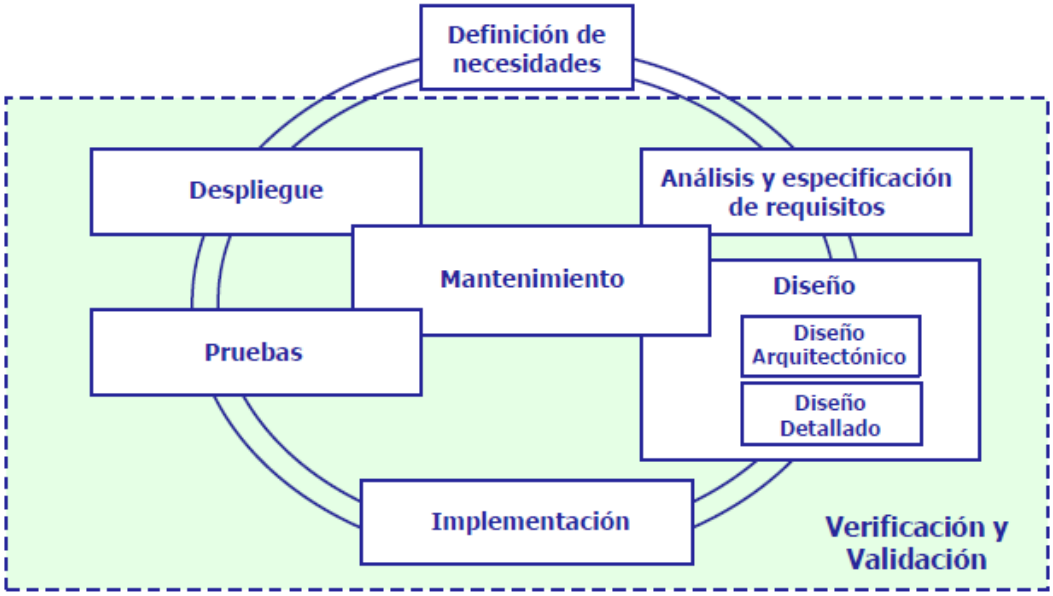


Imagen 3.4: Ciclo de vida de un Proyecto

Tras investigar encontré varias metodologías:

- Espiral: Consiste en repetir el ciclo de vida varias veces durante el desarrollo del software hasta llegar a la versión final del producto.
- Incremental: Está metodología consiste en escoger un módulo del producto y desarrollarlo hasta que esté terminado.
- Cascada: Consiste en ir pasando de una fase a otra hasta que el ciclo de vida del producto esté terminado.
- Sashimi: Es muy parecido a la metodología en cascada, con la diferencia de que las etapas anterior y posterior se pueden solapar con la actual, de forma que se aumente la eficiencia entre etapas.

Además de estas metodologías existen otras que son llamadas metodologías agiles como:

- Scrum.
- TDD (*Test Driven Development*)
- BDD (*Behaviour Driven Development*)

Dado que en el proyecto solo trabajaría yo, escogí la metodología incremental.

3.4. Arquitectura del proyecto

Como ya se comentó previamente, se va trabajar sobre la última versión de un sistema, y este sistema ya tiene una arquitectura software. Dicha arquitectura se basa en el Modelo Vista Controlador:

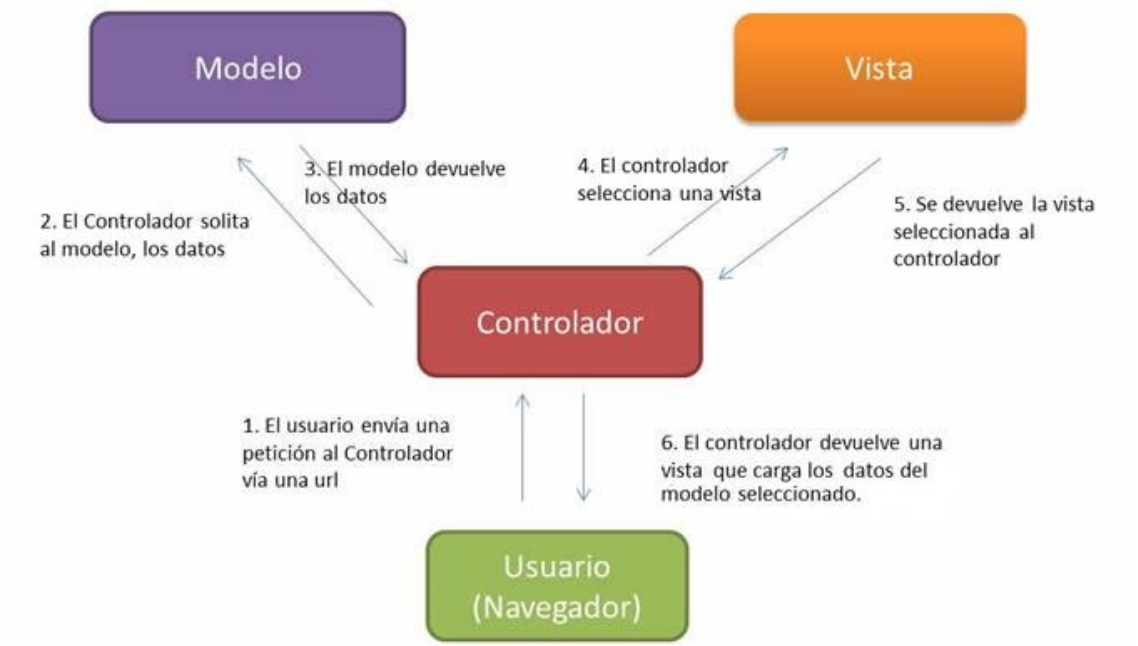


Imagen 3.5: Esquema del MVC

Como se puede observar en la imagen, la arquitectura MVC está formada por tres componentes:

- Modelo: gestiona la obtención, modificación, eliminación y creación de datos en el sistema. En resumen, controla lo que ocurre en la base de datos.
- Vista: se encarga de presentar la información (el modelo) con el formato adecuado.
- Controlador: Responde a las peticiones que realiza el usuario y utiliza los otros dos componentes para devolver la respuesta adecuada.

3.5. Desarrollo

Ahora que se han definido los requisitos y la metodología que se va a utilizar, llego el momento de comenzar a desarrollar los nuevos componentes del sistema.

Para explicar el proceso que seguí, utilizaré como ejemplo el requisito RF-02, el cual volveré a exponer a continuación.

RF-02
Creación, modificación y eliminación de listas de valores
El administrador del sistema podrá crear cualquier lista de valores, y podrá modificar o eliminar la lista o su contenido siempre y cuando alguno de los valores no se esté utilizando. **La lista de valores podrá tener hasta un nivel de profundidad 2. A continuación se mostrará un ejemplo.** Lista: Tipos de instrumentos Valores: Instrumentos de cuerda e instrumentos de viento Lista: Instrumentos musicales (lista padre: Tipos de instrumentos) Valores: Flauta (valor padre: instrumentos de viento) y saxofón (valor padre: instrumentos de viento)

3.5.1 Diseño de la interfaz gráfica

El primer paso que realicé fue diseñar la interfaz gráfica mediante la herramienta Balsamiq Mockups [14]. De esta forma me hacia una idea de cómo funcionarían los componentes.

Por lo general cuando terminaba de hacer el diseño, se lo presentaba al cliente para ver si esa era la idea que tenía. Y una vez que se lo presentaba, o bien realizaba los cambios necesarios, o comenzaba a desarrollar el modelo de datos.

En la siguiente página se encuentra la imagen 3.6, donde se muestra el diseño del requisito RF-02.

La idea es bastante sencilla. Meter todos los componentes de ese requisito en una sola vista. Los componentes que están rodeados con un rectángulo rojo son los encargados para obtener, modificar (si es posible) y eliminar una lista.

Una vez que el usuario seleccione la opción ‘Buscar lista’, ya no podrá crear una lista, porque, en teoría estaría modificando los datos de la lista que el usuario ha solicitado.

Por último, cuando el usuario selecciona la opción guardar, pueden ocurrir dos posibilidades: o se crea una lista nueva, o se modifica una lista ya creada.

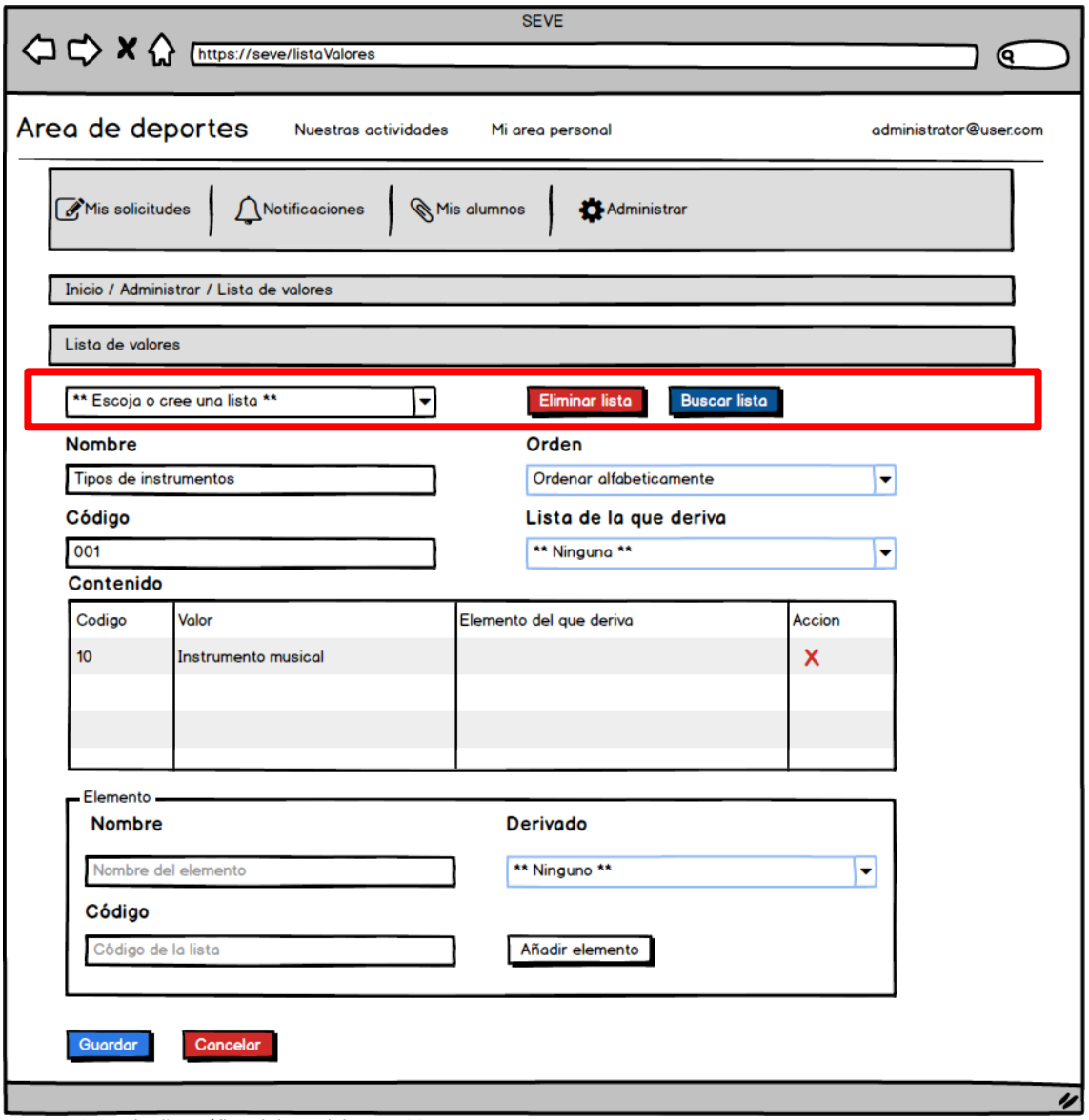


Imagen 3.6: Diseño gráfico del requisito RF-02

3.5.2. Modelo de datos

Ahora que se ha realizado un diseño de la interfaz gráfica de este requisito, se diseñará el modelo de datos para guardar estos componentes.

No es necesario interactuar con la base de datos, porque si se crea el modelo de datos utilizando Python [1], y después se usa la herramienta “Alembic” [11] para actualizar la base de datos, ya se tendrían las tablas para guardar estos objetos en la base de datos.

Así que, después de diseñar el modelo de datos, lo implementé.

```python
class LOV(db.Model):
    __tablename__ = 'lov'

    id = db.Column(db.Integer, primary_key=True)
    codigo = db.Column(db.String(128), nullable=False, unique=True)
    nombre = db.Column(db.String(128), nullable=False, unique=True)
    ordenarAlf = db.Column(db.Boolean, nullable=True)
    lista_padre = db.Column('lista_padre', db.Integer, db.ForeignKey('lov.id'))


class LovValores(db.Model):
    __tablename__ = 'lov_valores'

    id = db.Column(db.Integer, primary_key=True)
    lov_id = db.Column(db.Integer, db.ForeignKey('lov.id'))
    nombre = db.Column(db.String(128), nullable=False)
    codigo = db.Column(db.String(128), nullable=False, unique=True)
    valor_padre = db.Column('valor_padre', db.Integer, db.ForeignKey('lov_valores.id'))
```

Imagen 3.7: Modelos de los componentes del requisito RF-02

Una vez desarrollados los modelos hay que pedirle a “Alembic” [11] que genere una nueva versión, para ello hay que ejecutar la siguiente instrucción:

```
$ alembic revision --autogenerate -m "lista_valores"
```

Si no hay ningún fallo de compilación, te generará un fichero con este contenido:

```python
"""lista_valores

Revision ID: 1975ea83b712
Revises:
Create Date: 2016-02-08 11:40:27.089406

"""

# revision identifiers, used by Alembic.
revision = '1975ea83b712'
down_revision = '4895e356b712'

from alembic import op
import sqlalchemy as sa

def upgrade ():
    op.create_table('lov',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('codigo', sa.String(length=128), nullable=False),
    sa.Column('nombre', sa.String(length=128), nullable=False),
    sa.Column('ordenarAlf', sa.Boolean(), nullable=True),
    sa.Column('lista_padre', sa.Integer(), nullable=True),
    sa.ForeignKeyConstraint(['lista_padre'], ['lov.id'], ),
    sa.PrimaryKeyConstraint('id'),
    sa.UniqueConstraint('codigo'),
    sa.UniqueConstraint('nombre')
    )
```

```
    op.create_table('lov_valores',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('lov_id', sa.Integer(), nullable=True),
    sa.Column('nombre', sa.String(length=128), nullable=False),
    sa.Column('codigo', sa.String(length=128), nullable=False),
    sa.Column('valor_padre', sa.Integer(), nullable=True),
    sa.ForeignKeyConstraint(['lov_id'], ['lov.id'], ),
    sa.ForeignKeyConstraint(['valor_padre'], ['lov_valores.id'], ),
    sa.PrimaryKeyConstraint('id')
    )


def downgrade():
    op.drop_table('lov_valores')
    op.drop_table('lov')
```

Es recomendable ver si la versión se ha generado correctamente, ya que a veces puede añadir información de más que no es necesaria, o, es necesario cambiar algunos parámetros.

Por último, para generar las tablas en la base de datos, es necesario ejecutar un último comando que es:

```
$ alembic upgrade head
```

3.5.3. Controlador

Una vez que el modelo está hecho y la base de datos está actualizada, hay que desarrollar el controlador, el cual, está formado por dos partes.

La primera parte son clases que el controlador va a utilizar para comunicarse con la vista. Estas clases representan el formulario que se va a crear en la vista.

```
class CreateForm(PrzForm):
    listas = SelectField(choices=[], default=-1, coerce=int)

    nombre = StringField('nombre', validators=[DataRequired(), Unique(LOV.nombre, get_session=db.session)])
    ordenLista = SelectField(choices=[(0, u'Ordenar por código'), (1, 'Ordenar alfabeticamente')],
                             default=1, coerce=int)
    codigo = StringField('codigo', validators=[DataRequired(), Unique(LOV.codigo, get_session=db.session)])
    listaPadre = SelectField(coerce=int)

    listaValoresHijos = HiddenField("listaValoresHijos")
    nombreElem = StringField()
    codigoElem = StringField()
    elemPadre = SelectField(coerce=int)


class EditForm(CreateForm):
    def __init__(self, *args, **kwargs):
        super(EditForm, self).__init__(*args, **kwargs)
        read_only(self.nombre)
        read_only(self.codigo)
# ------------------------------------------------------------------------
```

Imagen 3.8: Clases que representan al formulario de la vista del requisito RF-02

En la imagen 3.8 se muestran estas clases que están formadas por atributos que hacen referencia a objetos de Flask [4][5], estos objetos son componentes del formulario HTML [2].

En el siguiente punto se mostrará cómo se crea uno de estos objetos utilizando Jinja2 [6].

Como se puede observar en la imagen todos los atributos hacen referencia a uno de los componentes de la vista. Excepto el campo listaValoresHijo, en él se guardará un string en formato JSON con los datos de los valores de la lista.

La segunda parte del controlador está formada por las funciones que va a utilizar ese controlador. A algunas funciones las llama directamente con una petición, y otras son funciones que llama otra función o son llamadas por una petición AJAX.

A continuación, se mostrará en la imagen 3.9 la función que se encarga de eliminar una lista si es posible.

```python
@bp.route("/del", methods=['POST'])
@login_required
@roles_required('ADMIN')
@Breadcrumb("admin.lista_valores")
def lista_valores_get_delete():
    """
    Elimina una lista de valores
    :return: pagina web
    """
    form = CreateForm(request.form)
    form.listas.choices = listar_lovs()
    form.listaPadre.choices = listar_lovs_padres()
    # Obtiene el id de la lista
    item = form.listas.data

    counter_preg = Pregunta.query.filter(Pregunta.lista == item).count()
    counter_lov = LOV.query.filter(LOV.lista_padre == item).count()
    # Comprueba si la lista es utilizada por otra lista
    if counter_lov >= 1 or counter_preg >= 1:
        flash("La lista de valores es usada por %s lista(s)" % str(counter_lov), 'error')
        flash("La lista de valores es usada por %s pregunta(s)" % str(counter_preg), 'error')
    else:
        # Elimina la lista y los valores
        for item_valores in LovValores.query.filter(LovValores.lov_id == item):
            db.session.delete(item_valores)
        db.session.commit()
        db.session.delete(LOV.query.get_or_404(item))
        db.session.commit()

    return redirect(url_for("lista_valores.lista_valores_get"))
```

Imagen 3.9: Función que elimina una lista de valores del requisito RF-02.

Como se puede observar, lo primero que se hace es obtener el formulario de la petición para poder obtener el id de la lista que se quiere utilizar. Después, se comprueba si se utiliza la lista. Para ello se utiliza SQLAlchemy [8] que utiliza los modelos.

Por último, en caso de que la lista se utilice, se indica que no se puede eliminar y si no se elimina la lista.

3.5.4. Vista

Una de razones por la que escogí este requisito es porque en la vista utilizo todos los posibles componentes. Como el fichero es muy grande, pondré pequeños fragmentos de código mostrando todas tecnologías utilizadas.

Lo primero que se hace en el fichero es heredar la plantilla base y las funciones más comunes. Después se empieza desarrollar el formulario.

Para implementar el rectángulo en rojo que aparece en la imagen 3.6 (por ejemplo) solo hay que escribir las líneas que aparecen en la imagen 3.10.

```
<div class="row">
    {{ form.hidden_tag() }}
    {{ m.render_field_with_errors(6, 'text', 'Lista de Valores', form.listas,
            class="form-control", prz_row="0")  }}
    <div class="form-group col-xs-6" style="...">
            <button class="btn btn btn-secondary" type="button" onclick="buscaLista()">Buscar lista</button>
            <button class="btn btn btn-warning" type="button" onclick="eliminaLista()">Eliminar lista</button>
    </div>
</div>
```

Imagen 3.10: Código HTML y Jinja2

Las líneas que empiezan y acaban con llaves son instrucciones de Jinja2 [6]. Como se puede ver solo hace falta una línea para renderizar el componente que muestra todas las listas de valores y su label. En el caso de que solo quieras renderizar el componente, solo se tiene que escribir: `{{ form.listas|safe }}`

Una vez que ya están todos los componentes, es hora de programar en JavaScript [2], ya que, todas las acciones se hacen con JavaScript [2] y AJAX quitando la obtención, guardado y eliminación de una lista.

A continuación, en la imagen 3.11, habrá una función que se encarga de guardar los valores de la lista en el campo listaValoresHijos.

```
/**
 * Guarda los campos de la tabla en un input
 */
function guardaFormulario(){
    var tabla = document.getElementById("bstable");
    var input = document.getElementById("listaValoresHijos");

    var valores = {};

    var elemPadre;
    for(var i =1; i<tabla.rows.length ; i++){
        elemPadre= tabla.rows[i].cells[2].innerHTML;
        valores[i]={"codigo": tabla.rows[i].cells[0].innerHTML,
                    "nombre": tabla.rows[i].cells[1].firstElementChild.value,
                    "elemPadre": elemPadre.split(" ")[0]};
    }
    input.value  = JSON.stringify(valores);
}
```

Imagen 3.11: Código JavaScript

Esta función es bastante sencilla, guardo todos los valores con un bucle for en una variable y luego convierto esos datos en un JSON.

En esta vista también se utilizan las ventanas modales, por ejemplo, cuando se selecciona el botón 'Eliminar lista' se ejecutará el código que aparece en la imagen 3.12.

```
swal({title: "Va a eliminar una lista de valores.", text: "¿Está seguro?", type: "warning",
    confirmButtonColor: "#DD6B55", confirmButtonText: "Si", showCancelButton: true, cancelButtonText: "No"},
        function (isConfirm) {
            if (isConfirm){
                var form = document.getElementById("formId");
                form.action = "{{ url_for('lista_valores.lista_valores_get_delete') }}";
                form.submit();
            }
        }
);
```

Imagen 3.12: Código de la ventana modal

El código para generar esta ventana modal es bastante sencillo y el resultado está bastante mejor que el alert o el confirm de JavaScript [2]. Este fragmento se ejecuta para asegurarse de que el usuario quiere eliminar la lista de valores. En caso de que de que se presione el botón 'Si' se ejecutará la función que se muestra en la imagen.

Y, por último, hay que comentar el uso de las funciones AJAX. En esta vista se utiliza para obtener los valores de la lista padre.

```
$.get("/u/lista_valores/elementos_lista_padre/" + valorSelect,
        function (response) {
            for (var index in response) {
                option = document.createElement('option');
                option.value = response[index][0];
                option.innerHTML = response[index][1];
                select_elems.appendChild(option);
            }
        },
        "json"
);
```

Imagen 3.13: Función AJAX

Como se puede observar en la imagen 3.13, su uso es bastante sencillo. La función llama a un método del servidor, y cuando obtenga la respuesta, ejecuta lo que hay en el interior de la función. La propia función AJAX transforma el JSON que retorna el servidor en un objeto JavaScript [2].

3.6. Tests

Una vez que se completaron los requisitos, era necesario desarrollar las pruebas para comprobar que funcionaba correctamente. Al principio, el sistema no tenía ni un solo test, por lo tanto, las únicas pruebas que se habían hecho, eran las pruebas de aceptación con el cliente.

Como era indispensable el desarrollo de las pruebas, se decidió dejar las pruebas unitarias para un trabajo futuro y empezar con las pruebas de

integración. La diferencia entre ambas pruebas es que cuando se necesite acceso a otro componente del sistema, se utilizarían Mocks.

Generalmente, ese componente sería la base de datos, y para acceder a la base de datos hay que utilizar el framework SQLAlchemy [8]. Por lo que, en cierto sentido, las pruebas unitarias no tienen mucho sentido ya que no es necesario probar el correcto funcionamiento del framework.

Así que, diseñe la ejecución de las pruebas, para que lo primero que se ejecute, fuesen las pruebas unitarias (cuando existiesen), y a continuación las pruebas de integración.

Para ejecutar las pruebas es necesario ejecutar el archivo run_testing.sh que se encuentra en el proyecto seve-demo. Este archivo creará una base de datos de prueba idéntica a la original, pero con las tablas vacías, y a continuación levantará el sistema para ejecutar las pruebas. A continuación, el archivo run_testing.sh llamará a un archivo del proyecto seve-core cuyo nombre es test_run.py.

El archivo test_run.py ejecuta todos los test que hay en la aplicación (primero los unitarios y luego los de integración) pero se puede configurar para que ejecute un test en concreto.

Todas las pruebas que hay dentro de un test son pruebas de caja negra, y se puede distinguir tres tipos de pruebas diferentes:

- Las pruebas normales, que solo utilizan UnitTest para comprobar el funcionamiento del método.
- Las pruebas que utilizan peticiones HTTP. Se utilizan en los métodos que son llamados por funciones AJAX y retornan un JSON. Para la ejecución de estas pruebas se utiliza un componente de Flask que simula un cliente. A este cliente se le puede indicar si el usuario está registrado, pero no se le puede indicar que tipo de usuario es.
- Las pruebas que usan Selenium [12]. Se utilizan en los métodos que son llamados por el cliente directamente, y tienen restricciones como, por ejemplo, `@roles_required('ADMIN')`

Tras finalizar el desarrollo de las pruebas de un componente, se comprobaba que su funcionamiento era correcto. Por lo general, los tests se ejecutaban con éxito, pero a veces fallaba. Esto se debía a que al desarrollar el componente no se tenía en cuenta ese caso porque no podía ocurrir.

Aun así, se buscaba una solución y se modificaba el componente. De forma que, si en el futuro se cambiaba el uso de ese método, no se produjese ese fallo.

4. Conclusiones

Tras la finalización del proyecto y su presentación al cliente puedo decir que el desarrollo de este proyecto ha ido bastante bien. Se han cumplido todos los objetivos del proyecto y el cliente está contento con el resultado.

He aprendido a manejar un lenguaje de programación y varios frameworks, y he de admitir que no ha sido tan difícil como me esperaba, aunque los primeros días sentí un poco de pánico al ver que no conocía nada de lo que iba a utilizar.

Me he dado cuenta de que usar los frameworks te facilita mucho el desarrollo del sistema. Por ejemplo, Flask [4][5], con cinco o diez líneas ya puedes crear una página web que calcule la suma de dos números. Y si programas un poco más crearás una página con un buen estilo.

Además, he vivido una experiencia (trabajar en CIC) que pocos universitarios han experimentado y que en mi opinión todos deberían vivir para que se hiciesen una idea de lo que les depara la vida laboral.

He de decir que completar alguno de los requisitos del cliente ha sido bastante complicado, por ejemplo, implementar la lista de valores en la respuesta de la pregunta, ya que cabe la posibilidad de que esa pregunta esté formada por dos campos, la lista padre y la hija.

Antes de terminar, como ya se ha comentado antes, he trabajado sobre un sistema que ya había desarrollado CIC. Y para poder cumplir los objetivos del proyecto he añadido alrededor de 15 componentes al sistema, entre ellos están los modelos, los controladores y las vistas. Además, he tenido que modificar cerca de un tercio de los componentes que ya existían para implementar los nuevos cambios.

Por último, a nivel personal, puedo decir que el desarrollo de este proyecto es más que satisfactorio, ya que he aprendido a usar tecnologías y herramientas que se que utilizaré en el futuro. Además, estoy bastante contento con el resultado que he obtenido.

5. Trabajos Futuros

De momento, la aplicación aún está en la fase de implementación, por lo que hay varias tareas por hacer:

- Terminar las pruebas de integración y las pruebas unitarias
- Inclusión de un módulo multi-lenguaje para poder vender el sistema en algunas comunidades autónomas como, por ejemplo, el País Vasco, Cataluña o Galicia.
- Reserva de espacios deportivos.
- Integración con pasarelas de pago (PayPal).
- Configuración del workflow de aprobación de la solicitud.
- Generar un mecanismo de integración para que los softwares de recaudación de los ayuntamientos puedan solicitar información al sistema.

6. Bibliografía

[1] Página web de Python, www.python.org, 10/3/2016

[2] Página web de HTML, CSS y JavaScript, www.w3schools.com, 1/4/2016

[3] Página web de MySQL, www.mysql.com, 16/3/2016

[4] Página web de Flask, http://flask.pocoo.org, 20/3/2016

[5] Miguel Grinberg, “Flask Web Development”, Editorial O’REILLY, 2014

[6] Página web de Jinja2, http://jinja.pocoo.org/docs/dev/, 12/4/2016

[7] Página web de Werkzeug, http://werkzeug.pocoo.org/, 20/4/2016

[8] Página web de SQLAlchemy, http://www.sqlalchemy.org/, 26/3/2016

[9] Página web de SweetAlert, http://t4t5.github.io/sweetalert/, 15/7/2016

[10] Página web de PyCharm, https://www.jetbrains.com/pycharm/, 11/3/2016

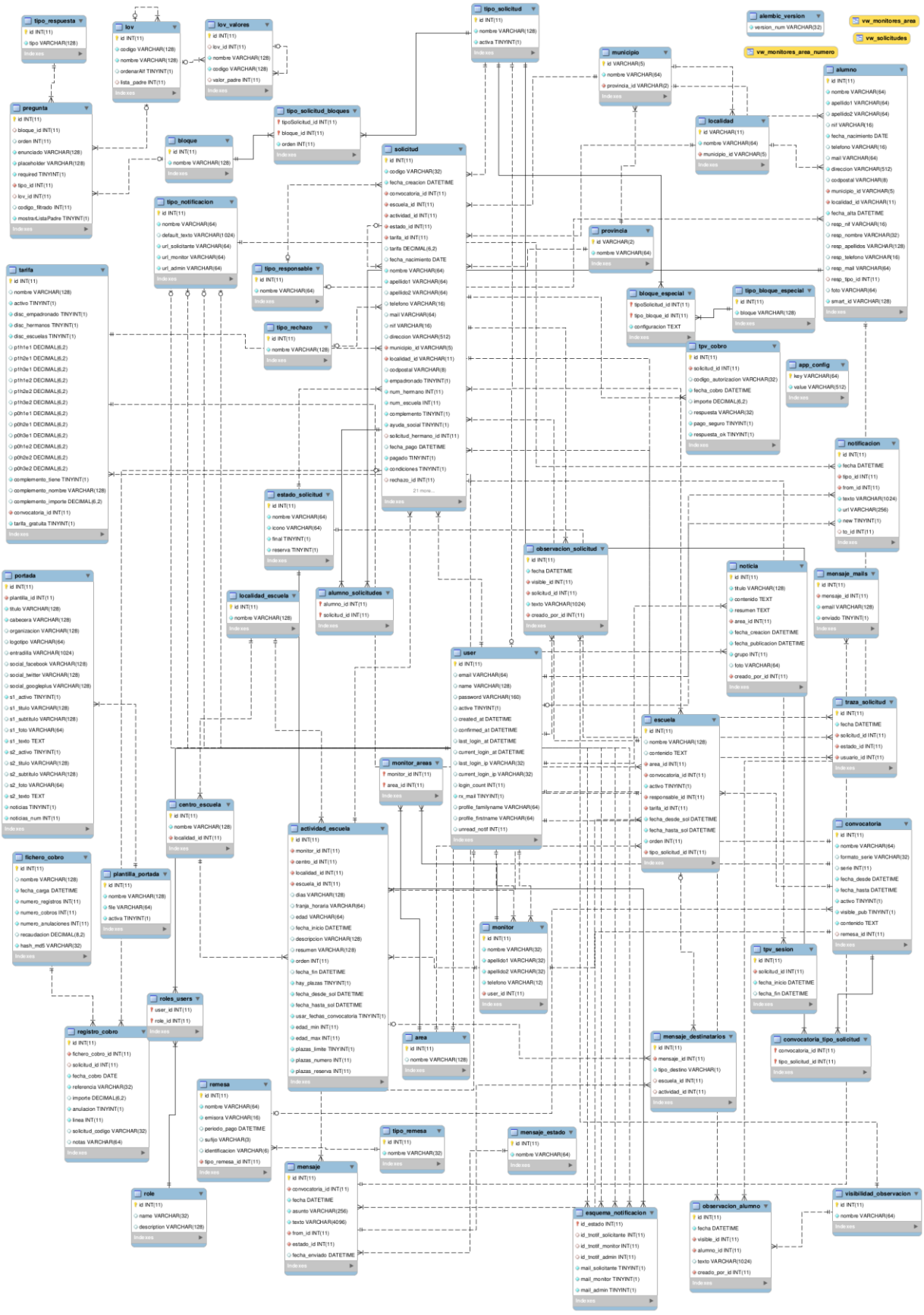
[11] Página web de Alembic, http://alembic.zzzcomputing.com/en/latest/, 1/2/2016

[12] Página web de Selenium, http://www.seleniumhq.org/, 16/6/2016

[13] Página web de Git, https://git-scm.com/, 20/3/2016

[14] Página web de Balsamiq Mockups, https://balsamiq.com/products/mockups/, 19/3/2016

ANEXO


tipo_respuesta
lov
lov_valores
tipo_solicitud
alembic_version
vw_monitores_area
vw_solicitudes
vw_monitores_area_numero
municipio
alumno
pregunta
bloque
tipo_solicitud_bloques
solicitud
localidad
tipo_notificacion
provincia
tarifa
tipo_responsable
tipo_rechazo
bloque_especial
tipo_bloque_especial
tpv_cobro
app_config
notificacion
estado_solicitud
observacion_solicitud
noticia
mensaje_mails
portada
localidad_escuela
alumno_solicitudes
user
traza_solicitud
escuela
monitor_areas
centro_escuela
actividad_escuela
convocatoria
fichero_cobro
plantilla_portada
monitor
tpv_sesion
roles_users
registro_cobro
area
mensaje_destinatarios
convocatoria_tipo_solicitud
remesa
tipo_remesa
mensaje_estado
mensaje
role
esquema_notificacion
observacion_alumno
visibilidad_observacion