Sistema centralizado de autenticación para entorno empresarial

Daniel Ignacio Salazar Recio

Proyecto presentado para la titulación de
Ingeniería Informática

Supervisado por:
Manuel Palomo Duarte

Universidad de Cádiz
Septiembre de 2017

Yo, Daniel Ignacio Salazar Recio confirmo que el trabajo presentado en este proyecto está hecho por mí. Cuando la información viene derivada de otras fuentes, confirmo que se indice explícitamente en la memoria.

Resumen

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam et turpis gravida, lacinia ante sit amet, sollicitudin erat. Aliquam efficitur vehicula leo sed condimentum. Phasellus lobortis eros vitae rutrum egestas. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec at urna imperdiet, vulputate orci eu, sollicitudin leo. Donec nec dui sagittis, malesuada erat eget, vulputate tellus. Nam ullamcorper efficitur iaculis. Mauris eu vehicula nibh. In lectus turpis, tempor at felis a, egestas fermentum massa.

Agradecimientos

Me gustaria dar las gracias a mi familia por haberme apoyado durante el desarrollo del proyecto en especial mis padres y mi hermano, así como a los compañeros de la carrera que me han ayudado con consejos y ánimos. También dar las gracias a mi tutor de proyecto Manuel Palomo por los consejos y supervisión.

Table of Contents

Lista de figuras

Lista de tablas

Definiciones y acrónimos

API	**A**pplication **P**rogramming **I**nterface
JSON	**J**ava**S**cript **O**bject **N**otation
Python	Lenguaje interpretado del lado del servidor.
HTML	**H**yper**T**ext **M**arkup **L**anguage

Chapter 1

Introducción

Con este Proyecto de Fin de Carrera se pretende la consecución de dos objetivos fundamentales: poner en práctica los conocimientos adquiridos en la titulación de Ingeniería en Informática y desarrollar una aplicación que de solución a un problema real de un entorno empresarial.

1.1 Objetivos y alcance

El proyecto consiste en la creación de un software que centralice y homogeinice la autenticación en un entorno empresarial. Una problemática habitual de una empresa que utilice múltiples productos es la diferencia entre los diferentes sistemas de autenticación de cada una de las aplicaciones, de forma que complica la gestión de estos usuarios debido a lo heterogéneo de los diferentes sistemas. Con este proyecto se propone un sistema que utiliza OAuth 2.0, de forma que sea sencillo de integrar con otras aplicaciones y facilitar también la gestión de permisos de cada una de las aplicaciones, centra- lizándolo en un único programa de gestión.

1.2 Estructura del documento

El documento se compone de los siguientes capítulos:

- Introducción: descripción del proyecto, objectivos y alcance del mismo y estructura básica del documento.
- Planificación: descripción del desarrollo de la planificación temporal y plazos de realización. Descripción general: descripción detallada sobre el proyecto, especificando tecnologías y herra- mientas usadas para su desarrollo.
- Análisis: fase de análisis del sistema, empleando la metodología seleccionada. Definición de re- quisitos funcionales del sistema, modelo conceptual y modelo de comportamiento.
- Diseño: diferentes fases del diseño técnico, arquitectura del sistema, diseño de la base de datos y diagramas de clase aplicadas al diseño.
- Implementación: aspectos más relevantes de la fase de implementación del sistema y explicación de los problemas encontrados durante el desarrollo.
- Pruebas y validaciones: plan de pruebas y automatización utilizado para el proyecto. Conclusiones: valoración y conclusiones personales obtenidas tras la realización del proyecto. Apéndices: ** Manual de instalación: manual para instalar correctamente la aplicación. ** Manual de usuario: manual para ayudar al usuario en el uso de la aplicación.
- Bibliografía: libros y referencias consultadas durante la realización del proyecto.
- Licencia GPL 3: texto completo sobre la licencia GPL 3, por la cual se rige el proyecto.

Chapter 2

Planificación

La planificación se divide en varias fases, a continuación se explicará en detalle cada una de ellas.

2.1 Fase inicial

La primera fase consistió en el planteamiento de la idea del proyecto, que en principio era desarrollar una aplicación que diera solución a un problema real del mundo empresarial. Tras la experiencia laboral adquirida en los últimos años decido realizar este proyecto conociendo los múltiples requisitos que pueden entrar en juego.

2.2 Fase de análisis

Se realizan diversas reuniones con empleados de una empresa para captar requisitos de manera informal. Tras estas reuniones se repasan estos requisitos y se escriben de manera formal, en forma de historias de usuario con tests de aceptación. En cada reunión posterior se plantean dudas y se van refinando esos requisitos. Finalmente se elabora un documento de requisitos que queda validado por los stakeholders del proyecto, en este caso los futuros usuarios de esta empresa.

2.3 Fase de aprendizaje

Para la realización del proyecto usaron tecnologías de las que se tenían conocimiento. La fase de apren- dizaje consistió principalmente en entender la problemática de la empresa en detalle, entendiendo que tenía que ser una solución que se aplicara de forma genérica.

2.4 Fase de diseño

Fase en la que se realiza el diseño técnico de la arquitectura de la aplicación. Hay que entender como funcionará la aplicación a alto nivel, estructurarla de forma que sea escalable y que pueda atender peti- ciones de múltiples usuarios.

También entra en esta fase el diseño de la estructura interna de la aplicación, patrones de diseño utiliza- dos, etc.

2.5 Implementación

Fase más extensa del desarrollo del proyecto. Consiste en implementar los requisitos especificados en la fase de análisis siguiendo para ello el diseño realizado en la fase anterior, procurando que la aplicación final satisfaga las necesidades y cumpla también los requisitos no funcionales.

2.6 Pruebas

Etapa importante en la que se comprueba una por una las funcionalidades del sistema verificando que no hay errores y que todo funciona como debe. Se realiza un plan de pruebas que pueda ser automatizado.

2.7 Redacción de la memoria

Esta fase se ha ido solapando con las demás ya que se ha realizado conjuntamente a las otras a medida que se iba desarrollando el proyecto.

2.8 Diagrama de Gantt

A continuación se muestra el diagrama de Gantt1 realizado con Excel, en el que se puede comprobar los plazos utilizados para las fases del desarrollo del proyecto.

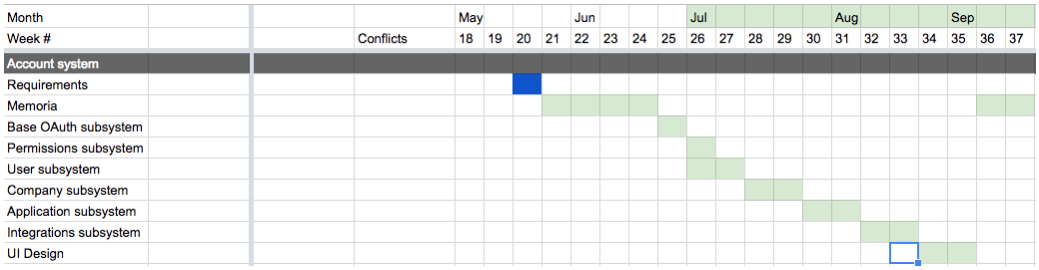


Figure 2.1: Diagrama de Gantt

Chapter 3

Descripción general del proyecto

Este proyecto tiene la condición de Software Libre, por lo que en caso de necesitar ser ampliado, cualquier persona podria hacerlo. El proyecto es una aplicación nueva, no es continuación de otro proyecto

3.1 Descripción

El proyecto consiste en una aplicación web, con una API pública, con distintos perfiles de usuario, en la que se llevará a cabo la configuración de las aplicaciones que tienen acceso al sistema.

3.2 Perfiles de usuario

A continuación se expondrán los diferentes perfiles detallando a que funcionalidad tendrá acceso cada uno.

3.2.1 Perfil Administrador

El administrador tendrá acceso a toda la gestión de usuarios, de productos y permisos. Por tanto el administrador podra crear nuevos usuarios con los perfiles que considere necesarios, nuevas aplicaciones y otorgar acceso a usuarios sobre aplicaciones.

3.2.2 Perfil Gestor de aplicaciones

Este perfil solo tendrá acceso a gestionar las aplicaciones existentes en el sistema, podrá también otorgar permisos sobre aplicaciones existentes a usuarios.

3.2.3 Perfil Usuario

Únicamente tendrá acceso a las aplicaciones visibles para este usuario.

3.3 Interfaz de Usuario

La interfaz será simple y funcional, ya que solo se utilizará a nivel interno en cada empresa. Visualizada en un navegador web, con un menú principal en el que se tendrá acceso a las diferentes funcionalidades, estando ocultas las que no pertenezcan al perfil del usuario.

3.4 Software

Al ser una aplicación web, ésta será multiplataforma, pudiendo funcionar sobre cualquier navegador actual, ya que cumple los estándares de la W3C.

Como lenguaje de servidor la aplicación utiliza Python, se toma la decisión de utilizarlo por la amplia documentación que hay disponible, por el

conocimiento del desarrollador, además de la multitud de librerías que existen para simplificar su utilización. Además se ha utilizado el framework MVC Django, que simplifica muchas tareas que de implementarlas únicamente con Python sin la ayuda de ninguna librería se harían muy tediosas.

Para las vistas se ha utilizado HTML, CSS y JavaScript, además de Bootstrap para simplificar el diseño de la aplicación, que es algo que escapa al alcance de este proyecto.

En la parte de los datos se ha usado MySQL como SGBD, utilizando Django ORM para abstraer el uso de la base de datos dentro de la aplicación.

Chapter 4

Análisis

4.1 Metodología de desarrollo

Para la realización del proyecto y su documentación se ha utilizado el Rational Unified Process (RUP), junto con el Lenguaje Unificado de Modelado (UML). Se ha elegido este sistema ya que es la metodología estándar más utilizada, además de ser un grupo de metodologías que se adaptan muy bien a las necesidades de un producto.

4.2 Especificación de requisitos del sistema

A continuación se enumeran los requisitos funcionales que se consideran fundamentales para el sistema. Éstos serán detallados utilizando casos de uso, describiendo tanto su escenario principal como sus posibles flujos alternativos. Además se detallará cada caso de uso con su diagrama de secuencia correspondiente.

4.2.1 Gestión de instalación

Una vez finalizada la instalación de la aplicación el administrador de la empresa debe terminar la configuración del sistema.

4.2.1.1 Caso de uso: Finalizar instalación

- **Descripción**: Caso de uso para la primera vez que se accede al sistema.
- **Actores**: Administrador de la empresa.
- **Precondiciones**: La aplicación ha sido instalada.
- **Postcondiciones**: La aplicación queda configurada para su uso.
- **Escenario principal**:
  - El sistema muestra un formulario al usuario para que introduzca los datos.
  - El administrador introduce el nombre de la empresa, su email y su contraseña, que será la contraseña del administrador del sistema.
  - El sistema valida los datos y muestra al usuario un formulario para introducir usuarios junto con su rol de acceso.
  - El administrador repite el paso anterior hasta que termine de introducir usuarios
  - El sistema manda un mail a todos los usuarios introducidos para terminar su configuración
  - El sistema queda configurado.
- **Escenarios alternativos**:
  - Alguno de los datos introducidos es inválido.
    * El sistema indica el error y el caso de uso vuelve al paso anterior.
  - El administrador decide dejar el proceso de introducción de usuarios para más tarde.
    * El caso de uso termina.
  - El administrador introduce un email que ya ha introducido previamente.

Figure 4.1: Diagrama de casos de uso de Gestión de instalación

        * El sistema indica el error y el caso de uso vuelve al paso anterior.

4.2.2 Gestión de usuarios

4.2.2.1 Caso de uso: Añadir usuario

- **Descripción**: Caso de uso para la creación de un usuario.
- **Actores**: Administrador de empresa.
- **Precondiciones**: El administrador se ha identificado correctamente en el sistema.
- **Postcondiciones**: Se crea un usuario con el perfil correspondiente.
- **Escenario principal**:
  - El administrador selecciona una empresa existente para añadir un usuario en ella.
  - El administrador introduce los datos del usuario y el nivel de privilegios.
  - El sistema valida que los datos son correctos y no hay ningún usuario con el mismo email.
  - El sistema crea el usuario y envía un mail al usuario para terminar la configuración
- **Escenarios alternativos**:
  - Alguno de los datos no es correcto.
    * El sistema indica el error y el caso de uso vuelve al paso anterior.
  - Ya existe algún usuario con el mismo email.
    * El sistema indica el error y el caso de uso vuelve al paso anterior.
  - En cualquier momento el administrador decide cancelar el proceso.
    * El caso de uso finaliza.

4.2.2.2 Caso de uso: Terminar configuración de usuario

- **Descripción**: El usuario registrado termina su configuración.
- **Actores**: Usuario
- **Precondiciones**: El usuario ha sido creado previamente por un administrador y el usuario ha recibido un email con un enlace.
- **Postcondiciones**: El usuario queda configurado y con acceso al sistema.
- **Escenario principal**:
  - El usuario abre el link que ha recibido por email.
  - El sistema muestra un formulario para configurar la contraseña y el resto de datos necesarios.
  - El usuario introduce los datos.
  - El sistema valida los datos.
  - El sistema guarda los datos y da acceso al usuario.
- **Escenarios alternativos**:
  - Alguno de los datos es incorrecto
    - El sistema lo indica y vuelve al paso anterior.

4.2.2.3 Caso de uso: Editar usuario

- **Descripción**: Caso de uso para la edición de un usuario.
- **Actores**: Usuario.
- **Precondiciones**: El usuario que se intenta editar coincide con el identificado en el sistema o bien el usuario identificado es un administrador.
- **Postcondiciones**: Se actualizan los datos del usuario.
- **Escenario principal**:
  - El sistema muestra los datos actuales del usuario.
  - El usuario modifica sus datos.
  - El sistema valida que los datos introducidos son correctos y no hay ningún otro usuario con el mismo email.

  - El usuario elige guardar los datos.
  - El sistema modifica el usuario.

- **Escenarios alternativos**:
  - Alguno de los datos no es correcto.
    * El sistema indica el error y el caso de uso vuelve al paso anterior.
  - Ya existe algún usuario con el mismo email.
    * El sistema indica el error y el caso de uso vuelve al paso anterior.
  - En cualquier momento el administrador decide cancelar el proceso.
    * El caso de uso finaliza.

4.2.2.4 Caso de uso: Ver usuario.

- **Descripción**: Caso de uso para seleccionar una empresa.
- **Actores**: Usuario del sistema.
- **Precondiciones**: El usuario tiene privilegios de usuario del sistema.
- **Postcondiciones**: Se muestran los datos del usuario.
- **Escenario principal**:
  - El usuario elige un usuario para ver sus datos.
  - El sistema muestra los datos del usuario.
  - El sistema muestra las aplicaciones a las que tiene acceso actualmente.

4.2.2.5 Caso de uso: Borrar usuario

- **Descripción**: Caso de uso para el borrado de un usuario
- **Actores**: Administrador.
- **Precondiciones**: El usuario identificado en el sistema tiene permisos para borrar usuarios o bien es administrador de la empresa del usuario a borrar.

- **Postcondiciones**: El usuario queda borrado
- **Escenario principal**:
  - El usuario selecciona un usuario para borrarlo.
  - El sistema muestra los datos actuales del usuario.
  - El usuario confirma que quiere borrar al usuario.
  - El sistema confirma el borrado.
- **Escenarios alternativos**:
  - En cualquier momento el administrador decide cancelar el proceso.
    * El caso de uso finaliza.

4.2.3 Gestión de empresas

A continuación se especifican los casos de uso necesarios para llevar a cabo la gestión de las empresas clientes del sistema.

Diagrama de casos de uso de Gestión de empresas

4.2.3.1 Caso de uso: Añadir empresa

- **Descripción**: Caso de uso para añadir una empresa
- **Actores**: Administrador del sistema.
- **Precondiciones**: El usuario tiene nivel de administrador del sistema.
- **Postcondiciones**: La empresa queda registrada en el sistema
- **Escenario principal**:
  - El administrador introduce los datos de la empresa.
  - El sistema valida que los datos son correctos y no hay ninguna empresa con el mismo nombre.
  - El administrador selecciona los productos a los que tendrá acceso la empresa.
  - El sistema crea la empresa.
- **Escenarios alternativos**:

  - Alguno de los datos no es correcto.
    * El sistema indica el error y el caso de uso vuelve al paso anterior.
  - Ya existe alguna empresa con el mismo nombre.
    * El sistema indica el error y el caso de uso vuelve al paso anterior.
  - En cualquier momento el administrador decide cancelar el proceso.
    * El caso de uso finaliza.

4.2.3.2 Caso de uso: Editar empresa

- **Descripción**: Caso de uso para la edición de una empresa.
- **Actores**: Administrador del sistema.
- **Precondiciones**: El usuario tiene privilegios de administrador del sistema.
- **Postcondiciones**: Se actualizan los datos de la empresa
- **Escenario principal**:
  - El sistema muestra los datos actuales de la empresa
  - El usuario modifica sus datos.
  - El sistema valida que los datos introducidos son correctos y no hay ningún otro usuario con el mismo email.
  - El usuario modifica los productos a los que tiene acceso la empresa.
  - El usuario elige guardar los datos.
  - El sistema modifica la empresa.
- **Escenarios alternativos**:
  - Alguno de los datos no es correcto.
    * El sistema indica el error y el caso de uso vuelve al paso anterior.
  - En cualquier momento el administrador decide cancelar el proceso.
    * El caso de uso finaliza.

4.2.3.3 Caso de uso: Ver empresa.

- **Descripción**: Caso de uso para la edición de una empresa.
- **Actores**: Usuario del sistema.
- **Precondiciones**: El usuario tiene privilegios de usuario del sistema.
- **Postcondiciones**: Se muestran los datos de la empresa
- **Escenario principal**:
  - El usuario elige una empresa de las que tiene acceso.
  - El sistema muestra los datos actuales de la empresa.
  - El sistema muestra los usuarios de la empresa.
  - El sistema muestra los productos contratados por la empresa.

4.2.3.4 Caso de uso: Borrar empresa.

- **Descripción**: Caso de uso para el borrado de una empresa.
- **Actores**: Administrador.
- **Precondiciones**: El usuario identificado en el sistema tiene permisos para borrar empresas.
- **Postcondiciones**: La empresa queda borrada.
- **Escenario principal**:
  - El usuario selecciona una empresa para borrarla.
  - El sistema muestra los datos actuales de la empresa.
  - El usuario confirma que quiere borrar a la empresa.
  - El sistema confirma el borrado.
- **Escenarios alternativos**:
  - En cualquier momento el administrador decide cancelar el proceso.
    * El caso de uso finaliza.

4.2.4 Gestión de productos

4.2.4.1 Caso de uso: Seleccionar producto

- **Descripción**: Caso de uso abstracto incluído en otros casos de uso para seleccionar una aplicación de una lista de disponibles
- **Actores**: Gestor de aplicaciones del sistema
- **Precondiciones**: El usuario identificado en el sistema es un gestor de aplicaciones.
- **Postcondiciones**: Se selecciona una aplicación para su uso en otra finalidad.
- **Escenario principal**:
  - El sistema muestra un listado de las aplicaciones disponibles.
  - El usuario selecciona la aplicación deseada.
- **Escenarios alternativos**:
  - No hay ninguna aplicación registrada.
    * El sistema indica el error y el caso de uso finaliza.

4.2.4.2 Caso de uso: Registrar aplicación

- **Descripción**: Registra una nueva aplicación en el sistema.
- **Actores**: Gestor de aplicaciones del sistema
- **Precondiciones**: El usuario identificado en el sistema es un gestor de aplicaciones.
- **Postcondiciones**: La aplicación queda registrada.
- **Escenario principal**:
  - El gestor introduce el código, el nombre y todos los demás datos de la aplicación.
  - El sistema comprueba que los datos cumplen el formato.
  - El sistema confirma el alta de la aplicación mostrando un mensaje.
- **Escenarios alternativos**:
  - Alguno de los datos introducidos tiene un formato incorrecto.

        * El sistema indica el error y se vuelve al paso anterior.
    - Falta algún campo obligatorio.
        * El sistema indica el error y se vuelve al paso anterior.
    - Ya existe alguna aplicación con ese código o nombre,
        * El sistema indica el error y se vuelve al paso anterior.
    - El gestor decide cancelar el registro en cualquier momento
        * El caso de uso finaliza

4.2.4.3 Caso de uso: Editar aplicación

- **Descripción**: Edita una aplicación existente en el sistema modificando sus datos
- **Actores**: Gestor de aplicaciones del sistema
- **Precondiciones**: El usuario identificado en el sistema es un gestor de aplicaciones.
- **Postcondiciones**: La aplicación queda modificada.
- **Escenario principal**:
    - Se realiza el caso de uso *seleccionar aplicación*
    - El sistema muestra sus datos actuales, permitiendo su edición.
    - El usuario modifica los datos.
    - El sistema comprueba que los datos son correctos.
    - El sistema confirma la modificación de la aplicación mostrando un mensaje.
- **Escenarios alternativos**:
    - Alguno de los datos introducidos tiene un formato incorrecto.
        * El sistema indica el error y se vuelve al paso anterior.
    - Falta algún campo obligatorio.
        * El sistema indica el error y se vuelve al paso anterior.
    - Ya existe alguna aplicación con ese código o nombre,
        * El sistema indica el error y se vuelve al paso anterior.
    - El gestor decide cancelar el registro en cualquier momento
        * El caso de uso finaliza

4.2.4.4 Caso de uso: Borrar aplicación

- **Descripción**: Borra una aplicación del sistema.
- **Actores**: Gestor de aplicaciones del sistema
- **Precondiciones**: El usuario identificado en el sistema es un gestor de aplicaciones.
- **Postcondiciones**: La aplicación queda eliminada.
- **Escenario principal**:
  - Se realiza el caso de uso *seleccionar aplicación*
  - El sistema muestra un diálogo de confirmación.
  - El usuario confirma que quiere eliminar la aplicación.
  - El sistema elimina la aplicación.
  - El sistema confirma la eliminación de la aplicación mostrando un mensaje.
- **Escenarios alternativos**:
  - El usuario selecciona que no desea eliminar la aplicación
    - El caso de uso se reinicia.
  - El gestor decide cancelar la eliminación en cualquier momento
    - El caso de uso finaliza

4.2.4.5 Caso de uso: Ver detalle de aplicación

- **Descripción**: Muestra los datos de una aplicación en detalle, así como sus credenciales.
- **Actores**: Gestor de aplicaciones del sistema
- **Precondiciones**: El usuario identificado en el sistema es un gestor de aplicaciones.
- **Postcondiciones**: Los datos de la aplicación se muestran por pantqalla
- **Escenario principal**:
  - Se realiza el caso de uso *seleccionar aplicación*
  - El sistema muestra los datos de la aplicación y sus credenciales.

- **Escenarios alternativos**:
  - El gestor decide cancelar el proceso en cualquier momento
    * El caso de uso finaliza

4.2.4.6 Caso de uso: Editar usuarios de la empresa con acceso a la aplicación

- **Descripción**: Cambia los usuarios de una empresa que tienen acceso a la aplicación.
- **Actores**: Administrador de empresa.
- **Precondiciones**: El usuario identificado en el sistema es un administrador de una empresa.
- **Postcondiciones**: Los usuarios con acceso a la aplicación quedan modificados.
- **Escenario principal**:
  - Se realiza el caso de uso *seleccionar aplicación*
  - El usuario selecciona el rol que quiere editar.
  - El usuario selecciona los usuarios de su empresa a añadir a la aplicación.
  - El usuario vuelve al paso 2 para seleccionar otro rol hasta que haya acabado con todos los roles.
  - El usuario selecciona guardar.
  - El sistema modifica los datos
- **Escenarios alternativos**:
  - El gestor decide cancelar el proceso en cualquier momento
    * El caso de uso finaliza

4.3 Integraciones

4.3.0.1 Caso de uso: Registrar aplicación externa

- **Descripción**: Registra una nueva aplicación externa en el sistema.
- **Actores**: Administrador de aplicaciones.
- **Precondiciones**: El usuario identificado en el sistema es un administrador de aplicaciones.
- **Postcondiciones**: La aplicación externa queda registrada.
- **Escenario principal**:
  - El usuario introduce los datos de la aplicación.
  - El sistema comprueba que son correctos.
  - El usuario introduce la url externa para hacer la integración.
  - El sistema prueba la conexión.
  - El sistema guarda los datos.
- **Escenarios alternativos**:
  - Los datos son incorrectos.
    - El sistema lo indica y vuelve al paso anterior.
  - La conexión con la url externa no se puede realizar.
    - El sistema lo indica y vuelve al paso anterior.
  - El gestor decide cancelar el proceso en cualquier momento
    - El caso de uso finaliza

4.3.0.2 Caso de uso: Obtener token de usuario desde aplicación interna

- **Descripción**: Un usuario obtiene un token a través de una aplicación interna.
- **Actores**: Usuario
- **Precondiciones**: El usuario existe en el sistema y tiene acceso a la aplicación.
- **Postcondiciones**: La aplicación recibe el token de usuario.

- **Escenario principal**:
  - La aplicación redirige al usuario a la web del sistema para identificarse.
  - El usuario se identifica en el sistema.
  - El sistema comprueba los datos son correctos.
  - El sistema comprueba que el usuario tiene acceso a la aplicación.
  - El sistema devuelve el token a la aplicación.
- **Escenarios alternativos**:
  - Las credenciales son incorrectas.
    * El sistema lo indica y vuelve al paso anterior.
  - El usuario identificado no tiene acceso a la aplicación.
    * El sistema lo indica y el caso de uso termina
  - El usuario decide cancelar el proceso en cualquier momento
    * El caso de uso finaliza

4.3.0.3 Caso de uso: Obtener token de usuario desde apliación externa

- **Descripción**: Un usuario obtiene un token a través de una aplicación externa.
- **Actores**: Usuario
- **Precondiciones**: El usuario existe en el sistema y tiene acceso a la aplicación.
- **Postcondiciones**: La aplicación recibe el token de usuario.
- **Escenario principal**:
  - La aplicación redirige al usuario a la web del sistema para identificarse.
  - El usuario se identifica en la aplicación.
  - El sistema redirige a la web de la aplicación externa.
  - El usuario se identifica en la aplicación externa.
  - El sistema devuelve el token a la aplicación.

- **Escenarios alternativos**:
  - Las credenciales son incorrectas.
    * El sistema lo indica y vuelve al paso anterior.
  - El usuario identificado no tiene acceso a la aplicación.
    * El sistema lo indica y el caso de uso termina
  - El usuario decide cancelar el proceso en cualquier momento
    * El caso de uso finaliza

4.3.0.4 Caso de uso: Importar usuarios desde módulo externo

- **Descripción**: Se importan usuarios desde módulo externo
- **Actores**: Administrador de empresa
- **Precondiciones**: El usuario existe en el sistema y es administrador de una empresa
- **Postcondiciones**: Los usuarios quedan cargados en la aplicación
- **Escenario principal**:
  - El usuario selecciona el método de importación.
  - El usuario carga los datos siguiendo el método adecuado.
  - El sistema registra los usuarios en el sistema.

4.3.0.5 Caso de uso: Sincronizar usuarios con módulo externo

- **Descripción**: Se sincroniza con una api externa para cargar los usuarios.
- **Actores**: Administrador de empresa
- **Precondiciones**: El usuario existe en el sistema y es administrador de una empresa
- **Postcondiciones**: Los usuarios quedan cargados en la aplicación
- **Escenario principal**:

- El usuario selecciona el método de importación.
- El usuario introduce las apis necesarias con las credenciales.
- El sistema sincroniza con la api externa.

4.4 Modelo Conceptual de datos

4.5 Modelo de comportamiento del sistema

Para el modelo de comportamiento del sistema se mostrarán diferentes diagramas de secuencia del sistema. El diagrama define las interacciones entre actores y sistema, también se detallarán los contratosde las operaciones del sistema, para describir en detalle qué hace cada operación.

Al existir muchos casos de uso similares, sólo se detallarán los más relevantes de cada subsistema.

4.5.1 Caso de uso: Añadir usuario

4.5.1.1 Contrato de la operación "introducir datos de usuario"

- **Responsabilidades**: Registrar usuario en el sistema
- **Referencias cruzadas**: Caso de uso *editar usuario*. Caso de uso *añadir usuario*
- **Precondiciones**: No existe un usuario con email = w_email
- **Postcondiciones**: Se crea una instancia de usuario U, se asignan w_email y datos.

4.5.2 Caso de uso: Editar usuario

4.5.2.1 Contrato de la operación "seleccionar usuario"

- **Responsabilidades**: Abrir página de usuario
- **Referencias cruzadas**: Caso de uso *Ver usuario*. Caso de uso *editar usuario*
- **Precondiciones**: El usuario está creado en la empresa y el usuario logueado tiene permisos para verlo.
- **Postcondiciones**: Se muestra la página del usuario.

4.5.3 Caso de uso: Añadir empresa

4.5.3.1 Contrato de la operación "introducir datos de empresa"

- **Responsabilidades**: Registrar empresa en el sistema
- **Referencias cruzadas**: Caso de uso *añadir empresa*. Caso de uso *editar empresa*
- **Precondiciones**: No existe una empresa con code = w_code
- **Postcondiciones**: Se crea una instancia de empresa E, se asignan w_code y datos.

4.5.4 Caso de uso: Editar empresa

4.5.4.1 Contrato de la operación "seleccionar empresa"

- **Responsabilidades**: Abrir página de empresa
- **Referencias cruzadas**: Caso de uso *Ver empresa*. Caso de uso *añadir usuario*. Caso de uso *editar usuario*. Caso de uso *editar empresa*. Caso de uso *editar usuarios con acceso a aplicación*.
- **Precondiciones**: La empresa existe en el sistema

- **Postcondiciones**: Se muestra la página de la empresa con el listado de usuarios.

4.5.5 Caso de uso: Añadir aplicación

Caso de uso: Añadir aplicacion

4.5.5.1 Contrato de la operación "introducir datos de aplicación"

- **Responsabilidades**: Registrar aplicación en el sistema
- **Referencias cruzadas**: Caso de uso *añadir aplicación*. Caso de uso *editar aplicación*
- **Precondiciones**: No existe una aplicación con code = w_code
- **Postcondiciones**: Se crea una instancia de aplicación A, se asignan w_code y datos, se generan credenciales.

4.5.6 Caso de uso: Editar aplicación

4.5.6.1 Contrato de la operación "seleccionar aplicación"

- **Responsabilidades**: Abrir página de empresa
- **Referencias cruzadas**: Caso de uso *Ver aplicación*. Caso de uso *editar aplicación*. Caso de uso *borrar aplicación*
- **Precondiciones**: La aplicación existe en el sistema
- **Postcondiciones**: Se muestra la página de la aplicación.

4.5.7 Caso de uso: Borrar aplicación

4.5.7.1 Contrato de la operación "borrar aplicación"

- **Responsabilidades**: Borra aplicación en el sistema
- **Referencias cruzadas**: Caso de uso *borrar aplicación*.
- **Precondiciones**: Existe una aplicación con code = w_code
- **Postcondiciones**: Se elimina la instancia de aplicación A. Se borran todos los accesos existentes para la aplicación.

4.5.8 Caso de uso: Editar usuarios de empresa con acceso a aplicación

4.5.9 Caso de uso: Obtener token de aplicación interna

4.5.9.1 Contrato de la operación "obtener token de acceso"

- **Responsabilidades**: Genera un token para el usuario logueado.
- **Referencias cruzadas**: Caso de uso *Obtener token de aplicación interna*.
- **Precondiciones**: El usuario tiene acceso a la aplicación.
- **Postcondiciones**: Se genera un token de acceso para el usuario.

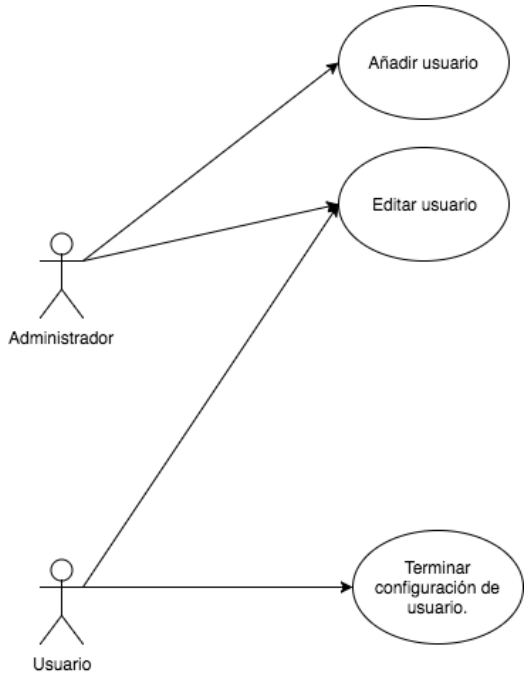


Figure 4.2: Diagrama de casos de uso de Gestión de usuarios

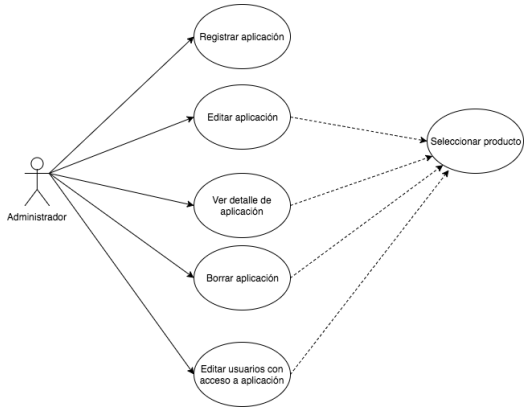


Figure 4.3: Diagrama de casos de uso de Gestión de productos

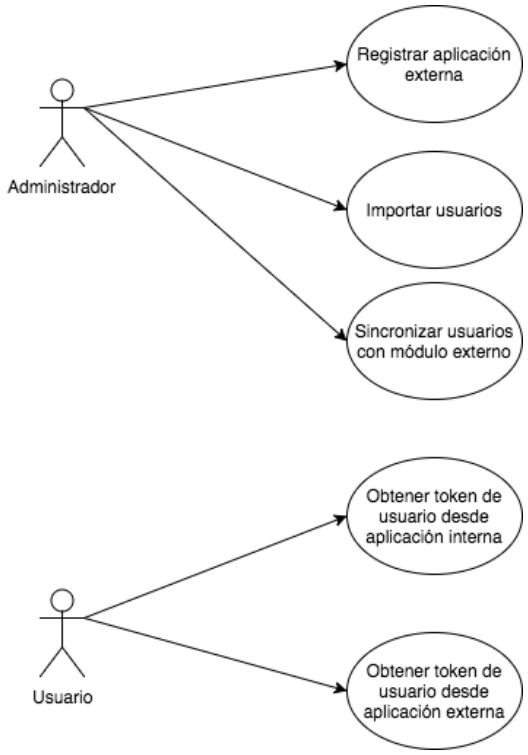


Figure 4.4: Diagrama de casos de uso de integraciones

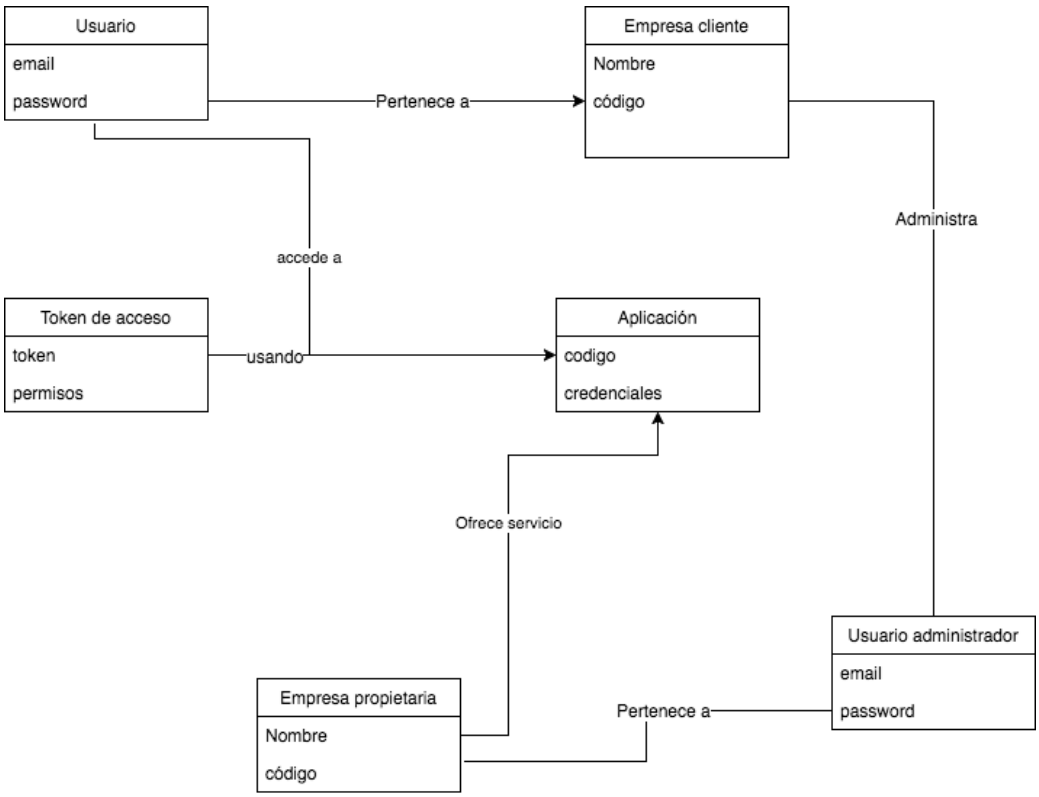


Figure 4.5: Diagrama de clases conceptual

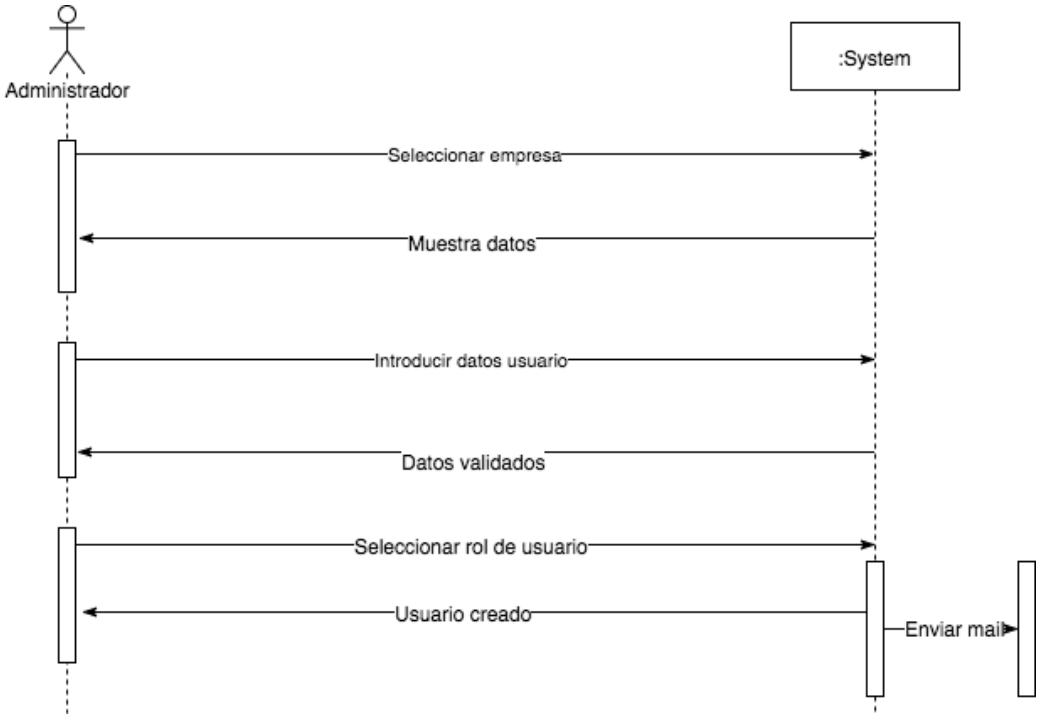


Figure 4.6: Caso de uso: Añadir usuario

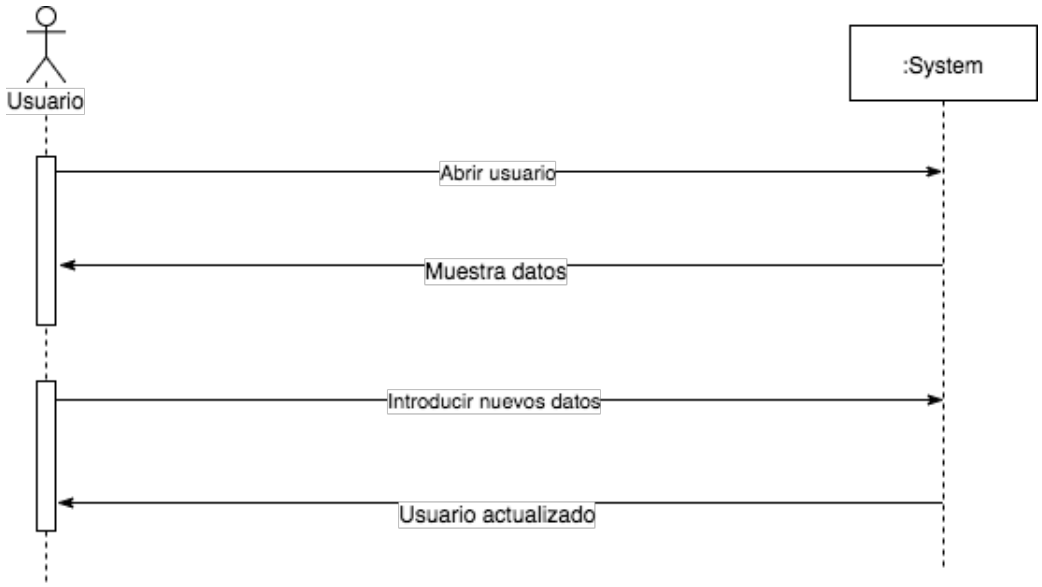


Figure 4.7: Caso de uso: Editar usuario

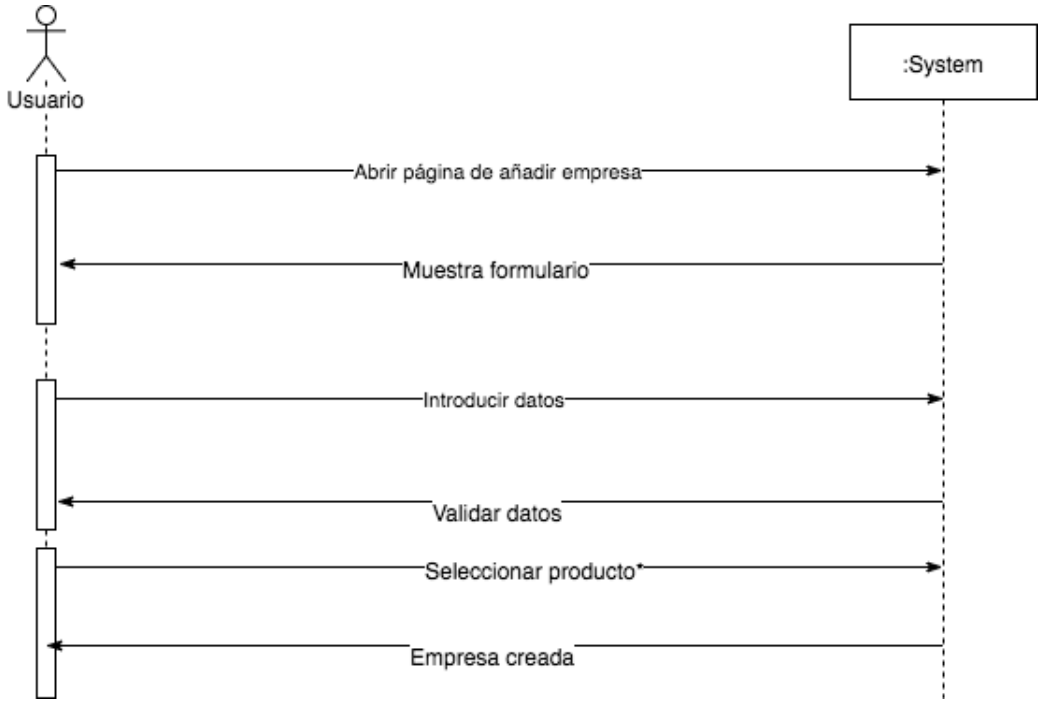


Figure 4.8: Caso de uso: Añadir empresa

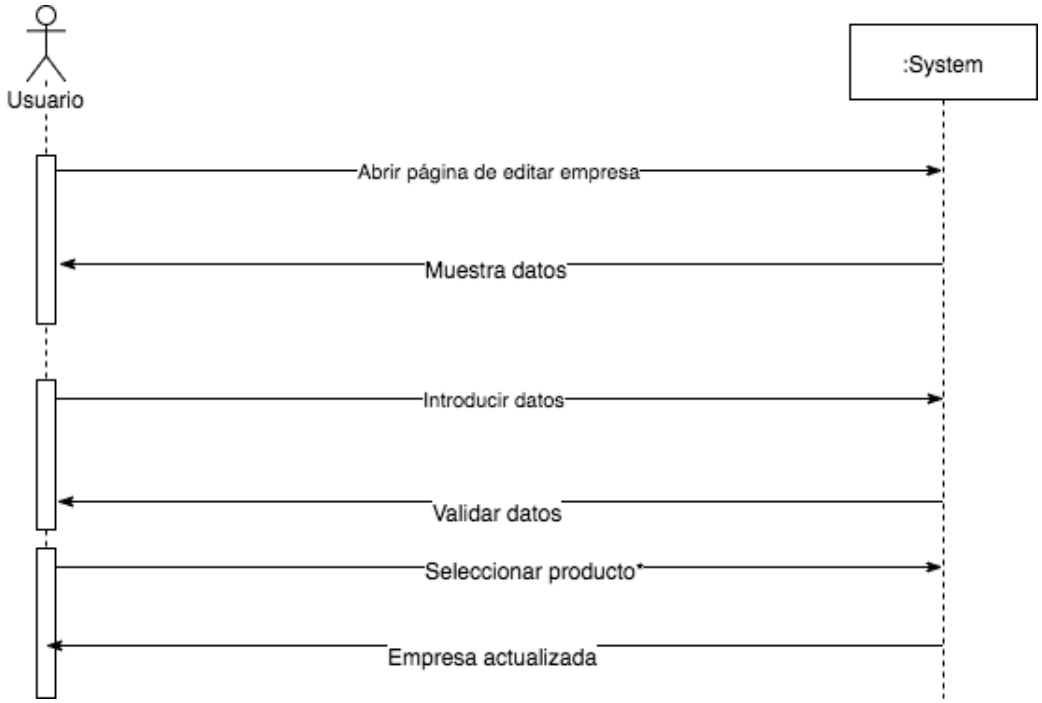


Figure 4.9: Caso de uso: Editar empresa

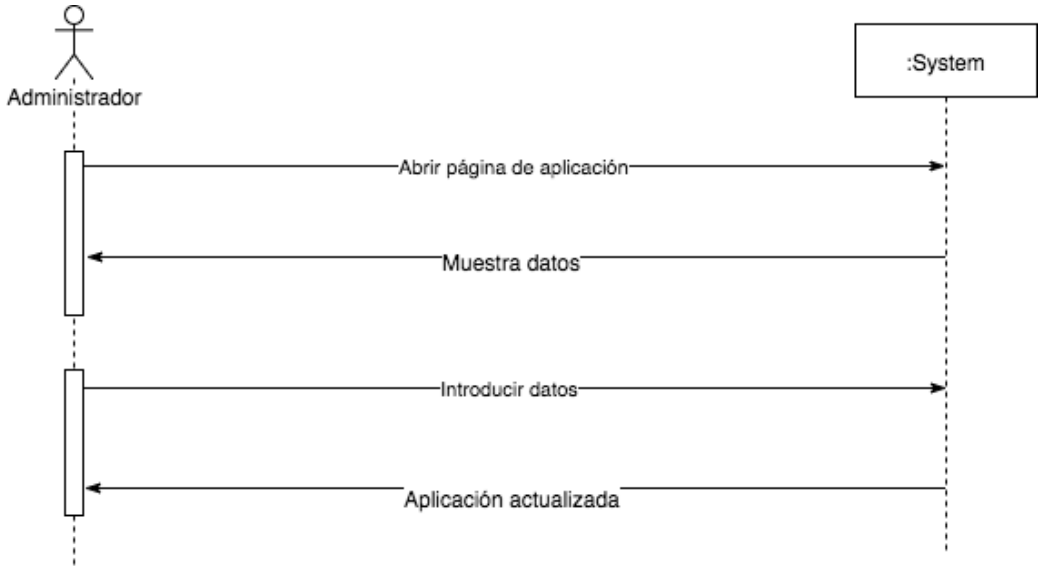


Figure 4.10: Caso de uso: Editar aplicacion

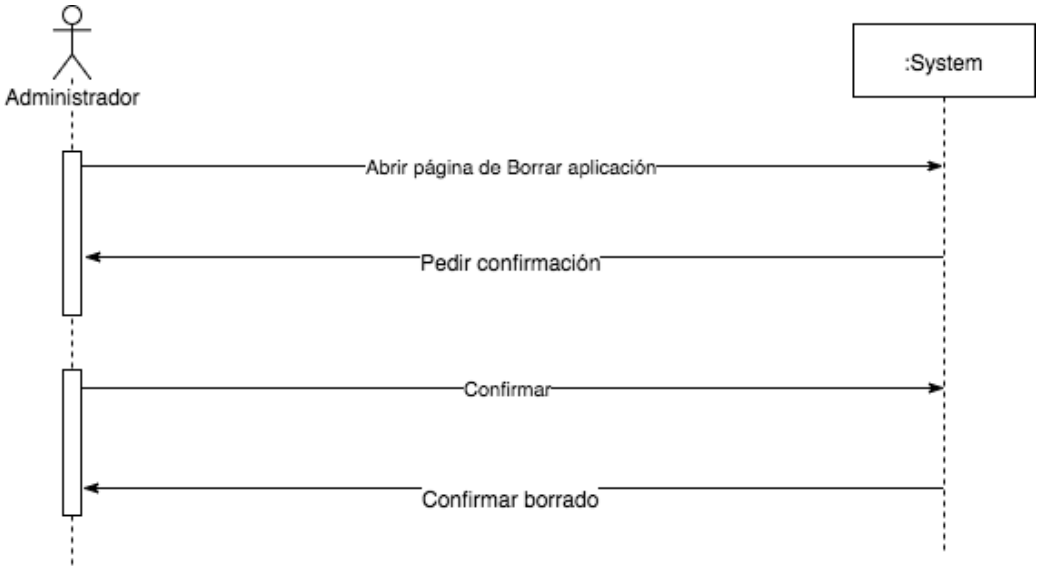


Figure 4.11: Caso de uso: Borrar aplicacion

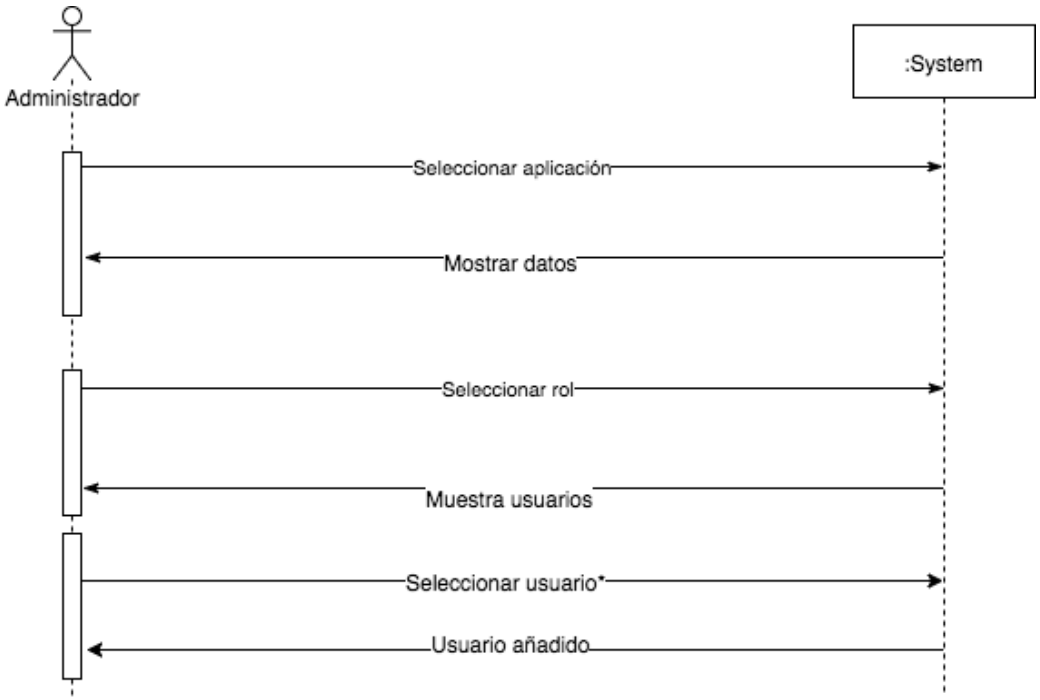


Figure 4.12: Caso de uso: Editar acceso de usuarios de empresa

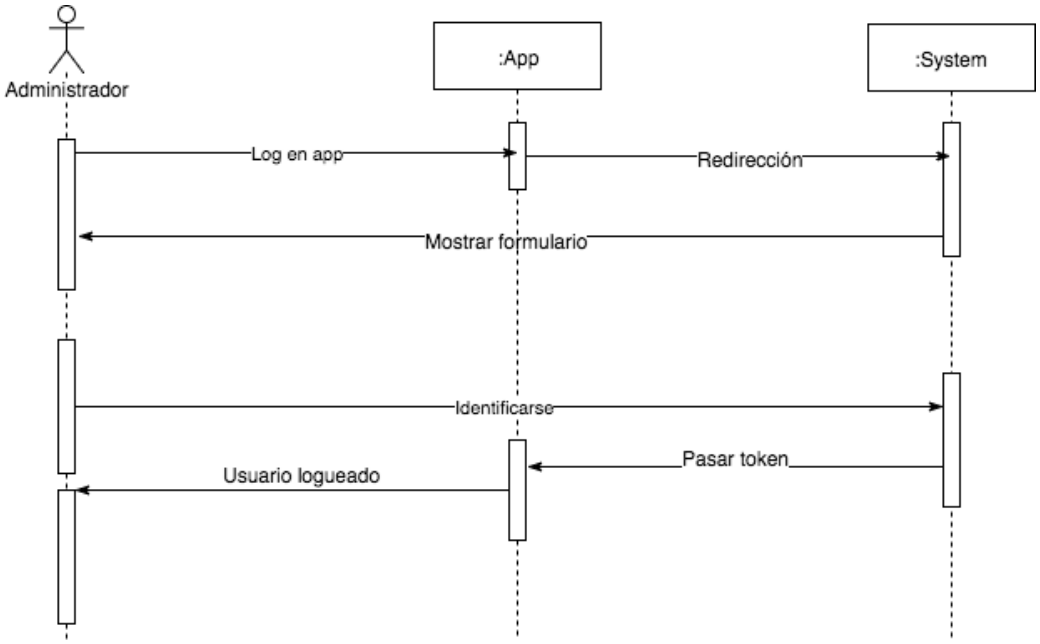


Figure 4.13: Caso de uso: Obtener token de aplicación interna

Chapter 5

Diseño

5.1 Introducción

La fase de diseño consiste en aplicar una serie de técnicas para transformar los requisitos elicitados en la fase de análisis en una estructura detallada para el sistema de forma que se pueda implementar fácilmente a partir de ese diseño.

Este diseño ha de realizarse a varios niveles de detalle, por un lado a nivel de arquitectura interna de la aplicación. Por otro lado el diseño del modelo de datos y finalmente a más alto nivel detallar todos los componentes del sistema y como interactúan entre si, componiendo un sistema escalable.

Siguiendo los requisitos de la fase anterior, esta tarea es relativamente sencilla y debe resultar en una serie de diagramas y especificaciones que sirvan como guión y documentación a las personas que vayan a participar, ahora o en un futuro en el desarrollo del sistema.

Este documento debe contener también un modelo de despliegue incluyendo infraestructura, aplicación y configuración.

5.2 Arquitectura de aplicación

La aplicación se rige por el patrón arquitectónico MVC. Este patrón permite separar la lógica de negocio de la presentación, así como de los datos.

- El modelo representa a las estructuras de datos. Las clases del modelo contienen funciones para modificar los datos, insertar y actualizar la base de datos.
- La vista es la información presentada al usuario. Una vista normalmente es una página web que puede contener datos del modelo para mostrarlos al usuario.
- El controlador es un intermediario entre las dos capas anteriores y otros recursos que puedan ser necesarios. Se encarga de procesar las peticiones y generar la página web que será presentada al usuario.

Este patrón favorece la reutilización de código y la claridad.

5.2.1 Elección del framework

Dado que en el lenguaje elegido para implementar la aplicación es Python, se ha elegido el framework Django, debido a la experiencia previa con esta tecnología. Este framework permite empezar a tener un software funcionando en muy poco tiempo, dedicando el trabajo casi exclusivamente a implementar el modelo y la lógica de negocio. Dado que el framework ya es conocido, la curva de aprendizaje es mínima.

5.2.2 Endpoints

5.2.2.1 Empresas

5.2.2.1.1 Visualización GET: /v1/companies

- Headers:

  - Content-Type: applicacion/json
  - Authorization: Bearer <token>
- Response (Lista de):
  - **“id”**: 123,
  - **“name”**: “Company”,
  - **“code”**: “company_1”
- Response codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

GET: /v1/companies/<company-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>
- Response (JSON Object):
  - **“id”**: 123,
  - **“name”**: “Company”,
  - **“code”**: “company_1”
- Response Codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)
  - 404 NOT FOUND (*company-id* no encontrado)

5.2.2.1.2 Creación POST: /v1/companies

- Headers:
  - Content-Type: applicacion/json

  - Authorization: Bearer <token>

- Request body (JSON Object):
  - **“name”**: “Company”,
  - **“code”**: “company_1”

- Response codes:
  - 201 CREATED (Empresa creada)
  - 400 BAD REQUEST (Dato incorrecto en el body)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.1.3 Actualización PUT: /v1/companies/<company-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>

- Request body (JSON Object):
  - **“name”**: “Company”,
  - **“code”**: “company_id”

- Response codes:
  - 200 OK (Empresa actualizada)
  - 400 BAD REQUEST (Dato incorrecto en el body)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.1.4 Borrado DELETE: /v1/companies/<company-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>

- Response codes:
  - 204 NO CONTENT (Empresa borrada)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.2 Usuarios

5.2.2.2.1 Visualización GET: /v1/companies/<company-id>/users

- Headers:
  - Content-Type: applicacion/json
  - Authorization: Bearer <token>
- Response (Lista de):
  - **“id”**: 123,
  - **“name”**: “User”,
  - **“email”**: “user@company.com”
- Response codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

GET: /v1/companies/<company-id>/users/<user-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>
- Response (JSON Object):
  - **“id”**: 123,
  - **“name”**: “User”,
  - **“email”**: “email@company.com”

- Response Codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)
  - 404 NOT FOUND (*company-id* o *user-id* no encontrado)

5.2.2.2.2 Creación POST: /v1/companies/<company-id>/users

- Headers:
  - Content-Type: applicacion/json
  - Authorization: Bearer <token>
- Request body (JSON Object):
  - **"name"**: "Nombre",
  - **"email"**: "user@company.com",
  - **"password"**: "password"
- Response codes:
  - 201 CREATED (Usuario creado)
  - 400 BAD REQUEST (Dato incorrecto en el body)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.2.3 Actualización PUT: /v1/companies/<company-id>/users/<user-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>
- Request body (JSON Object):
  - **"name"**: "User",
  - **"email"**: "email@company.com"

- **“password”**: “password”

- Response codes:
  - 200 OK (Usuario actualizada)
  - 400 BAD REQUEST (Dato incorrecto en el body)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.2.4 Borrado DELETE: /v1/companies/<company-id>/users/<user-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>
- Response codes:
  - 204 NO CONTENT (Usuario borrado)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.3 Aplicaciones

5.2.2.3.1 Visualización GET: /v1/applications

- Headers:
  - Content-Type: applicacion/json
  - Authorization: Bearer <token>
- Response (Lista de):
  - **“id”**: 123,
  - **“name”**: “application”,
  - **“client_id”**: “123123123”,
  - **“client_secret”**: “123123123”,

- Response codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

GET: /v1/applications/<client-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>
- Response (JSON Object):
  - **"id"**: 123,
  - **"name"**: "User",
  - **"client_id"**: "123123123",
  - **"client_secret"**: "123123123",
- Response Codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)
  - 404 NOT FOUND (*client-id* no encontrado)

5.2.2.3.2 Creación POST: /v1/applications/<client-id>

- Headers:
  - Content-Type: applicacion/json
  - Authorization: Bearer <token>
- Request body (JSON Object):
  - **"name"**: "Nombre",
- Response (JSON Object):
  - **"id"**: 123,

  - **“name”**: “User”,
  - **“client_id”**: “123123123”,
  - **“client_secret”**: “123123123”,

- Response codes:
  - 201 CREATED (Aplicación creada)
  - 400 BAD REQUEST (Dato incorrecto en el body)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.3.3 Actualización PUT: /v1/applications/<client-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>
- Request body (JSON Object):
  - **“name”**: “User”,
- Response codes:
  - 200 OK (Aplicación actualizada)
  - 400 BAD REQUEST (Dato incorrecto en el body)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.3.4 Borrado DELETE: /v1/applications/<client-id>

- Headers:
  - Content-Type: application/json
  - Authorization: Bearer <token>
- Response codes:
  - 204 NO CONTENT (Aplicación borrada)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.4 Access Tokens

5.2.2.4.1 Creación (authorization code) POST /v1/oauth2/authorization

- Headers:
  - Content-Type: application/json
- Request body (JSON Object)
  - **“username”**: “<user_email>”,
  - **“password”**: “<user_password>”,
  - **“client_id”**: “<client_id>”
  - **“client_secret”**: “<client_secret>”
  - **“redirect_uri”**: “http://auth_server”
- Response body:
  - Redirect to http://auth_server?code=code
- Response codes:
  - 302 Redirect (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.4.2 Creación (password grant) POST /v1/oauth2/access-tokens

- Headers:
  - Content-Type: application/json
- Request body (JSON Object)
  - **“grant_type”**: “password”,
  - **“username”**: “<user_email>”,
  - **“password”**: “<user_password>”,
  - **“client_id”**: “<client_id>”
  - **“client_secret”**: “<client_secret>”

- Response body:
  - **“token”**: “token”,
  - **“refresh_token”**: “refresh_token”,
  - **“expires_in”**: 123123 (seconds)
- Response codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.4.3 Creación (client_credentials grant) POST /v1/oauth2/access-tokens

- Headers:
  - Content-Type: application/json
- Request body (JSON Object)
  - **“grant_type”**: “client_credentials”,
  - **“client_id”**: “<client_id>”
  - **“client_secret”**: “<client_secret>”
- Response body:
  - **“token”**: “token”,
  - **“refresh_token”**: “refresh_token”,
  - **“expires_in”**: 123123 (seconds)
- Response codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.4.4 Creación (authorization_code grant) POST /v1/oauth2/access-tokens

- Headers:
  - Content-Type: application/json
- Request body (JSON Object)
  - **"grant_type"**: "authorization_code",
  - **"client_id"**: "<client_id>"
  - **"client_secret"**: "<client_secret>"
  - **"code"**: "<code>"
- Response body:
  - **"token"**: "token",
  - **"refresh_token"**: "refresh_token",
  - **"expires_in"**: 123123 (seconds)
- Response codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.2.2.4.5 Actualización (refresh_token grant) POST /v1/oauth2/access-tokens

- Headers:
  - Content-Type: application/json
- Request body (JSON Object)
  - **"grant_type"**: "refresh_token",
  - **"client_id"**: "<client_id>"
  - **"client_secret"**: "<client_secret>"
  - **"refresh_token"**: "<refresh_token>"
- Response body:

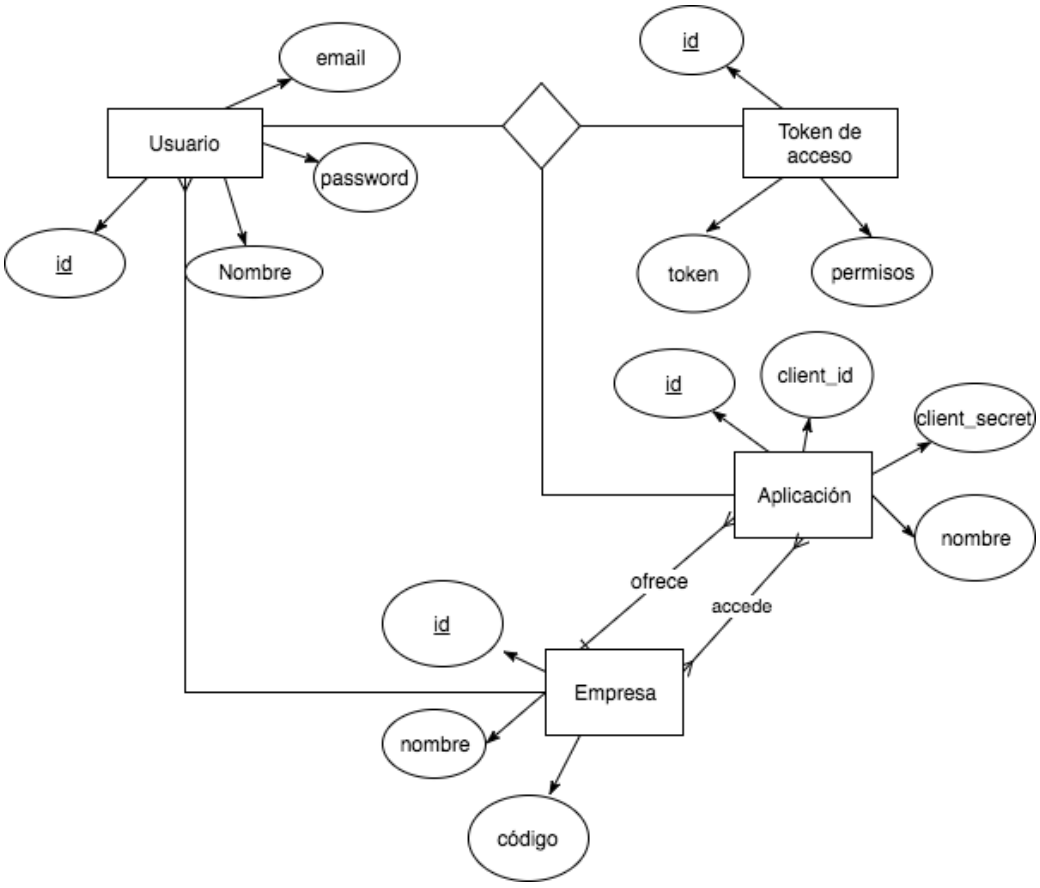
  - **“token”**: “token”,
  - **“refresh_token”**: “refresh_token”,
  - **“expires_in”**: 123123 (seconds)

- Response codes:
  - 200 OK (Success)
  - 401 UNAUTHORIZED (Token de acceso inválido o no enviado)
  - 403 FORBIDDEN (Sin permisos)

5.3 Base de datos

Para el diseño de la base de datos en la que se guardarán los datos manejados por la aplicación se usará un modelo relacional. Se usará MySQL como sistema de gestión de base de datos.

5.3.1 Modelo entidad-relación



??

5.3.2 Tablas y atributos

5.3.2.1 Usuarios

Atributos	Tipo	Nulo	Index	Descripción
id	INTEGER	NO	PRIMARY_KEY	Identificador autoincremental
email	VARCHAR(50)	NO	UNIQUE	Email de usuario, sirve para login

Atributos	Tipo	Nulo	Index	Descripción
password	VARCHAR(255)	NO	NO	Password cifrado de usuario
nombre	VARCHAR(255)	NO	NO	Nombre del usuario
apellido	VARCHAR(255)	NO	NO	Apellidos del usuario
company_id	INTEGER	NO	FOREIGN_KEY	Id de empresa

5.3.2.1.1 Normalización

- La tabla está en primera forma normal ya que:
  - Todos los atributos son atómicos.
  - Tiene clave primaria única (id).
  - La CP no puede ser nula.
- La tabla está en segunda forma normal ya que:
  - Está en 1FN.
  - Al ser la clave única no puede haber dependencias parciales.
- La tabla está en tercera forma normal ya que:
  - Está en 2FN.
  - No hay dependencias funcionales transitivas.
- La tabla está en forma normal de Boyce-Codd ya que:
  - Para toda dependencia funcional X->A X es superllave.

5.3.2.2 Aplicaciones

Atributos	Tipo	Nulo	Index	Descripción
id	INTEGER	NO	PRIMARY_KEY	Identificador autoincremental
client_id	VARCHAR(50)	NO	UNIQUE	client id
client_secret	VARCHAR(50)	NO	NO	client secret
name	VARCHAR(50)	NO	NO	Nombre de aplicación

5.3.2.2.1 Normalización

- La tabla está en primera forma normal ya que:
  - Todos los atributos son atómicos.
  - Tiene clave primaria única (id).
  - La CP no puede ser nula.
- La tabla está en segunda forma normal ya que:
  - Está en 1FN.
  - Al ser la clave única no puede haber dependencias parciales.
- La tabla está en tercera forma normal ya que:
  - Está en 2FN.
  - No hay dependencias funcionales transitivas.
- La tabla está en forma normal de Boyce-Codd ya que:
  - Para toda dependencia funcional X->A X es superllave.

5.3.2.3 Empresas

Atributos	Tipo	Nulo	Index	Descripción
id	INTEGER	NO	PRIMARY_KEY	Identificador autoincremental
code	VARCHAR(50)	NO	UNIQUE	Código identificador de la empresa
name	VARCHAR(50)	NO	NO	Nombre de empresa

5.3.2.3.1 Normalización

- La tabla está en primera forma normal ya que:
  - Todos los atributos son atómicos.
  - Tiene clave primaria única (id).
  - La CP no puede ser nula.
- La tabla está en segunda forma normal ya que:
  - Está en 1FN.
  - Al ser la clave única no puede haber dependencias parciales.
- La tabla está en tercera forma normal ya que:
  - Está en 2FN.
  - No hay dependencias funcionales transitivas.
- La tabla está en forma normal de Boyce-Codd ya que:
  - Para toda dependencia funcional X->A X es superllave.

5.3.2.4 Tokens de acceso

Atributos	Tipo	Nulo	Index	Descripción
id	INTEGER	NO	PRIMARY_KEY	Identificador autoincremental
token	VARCHAR(50)	NO	UNIQUE	Token de acceso de usuario
user_id	INTEGER	NO	FOREIGN_KEY	Id de usuario
application_id	INTEGER	NO	FOREIGN_KEY	Id de aplicación
permisos	VARCHAR(255)	NO	NO	Permisos de token de usuario
expires	INTEGER	NO	NO	Segundos de expiración de token

5.3.2.4.1 Normalización

- La tabla está en primera forma normal ya que:
  - Todos los atributos son atómicos.
  - Tiene clave primaria única (id).
  - La CP no puede ser nula.
- La tabla está en segunda forma normal ya que:
  - Está en 1FN.
  - Al ser la clave única no puede haber dependencias parciales.

- La tabla está en tercera forma normal ya que:
  - Está en 2FN.
  - No hay dependencias funcionales transitivas.
- La tabla está en forma normal de Boyce-Codd ya que:
  - Para toda dependencia funcional X->A X es superllave.

5.3.2.5 Refresh tokens

Atributos	Tipo	Nulo	Index	Descripción
id	INTEGER	NO	PRIMARY_KEY	Identificador autoincremental
token	VARCHAR(50)	NO	UNIQUE	Token de refresco de usuario
access_token_id	INTEGER	NO	FOREIGN_KEY	Id de usuario

5.3.2.5.1 Normalización

- La tabla está en primera forma normal ya que:
  - Todos los atributos son atómicos.
  - Tiene clave primaria única (id).
  - La CP no puede ser nula.
- La tabla está en segunda forma normal ya que:
  - Está en 1FN.
  - Al ser la clave única no puede haber dependencias parciales.
- La tabla está en tercera forma normal ya que:
  - Está en 2FN.

  - No hay dependencias funcionales transitivas.

- La tabla está en forma normal de Boyce-Codd ya que:
  - Para toda dependencia funcional X->A X es superllave.

5.3.2.6 Authorization codes

Atributos	Tipo	Nulo	Index	Descripción
id	INTEGER	NO	PRIMARY_KEY	Identificador autoincremental
code	VARCHAR(50)	NO	UNIQUE	Código de autorización
application_id	INTEGER	NO	FOREIGN_KEY	Id de usuario
user_id	INTEGER	NO	FOREIGN_KEY	Id de usuario
expires	INTEGER	NO	NO	Segundos de expiración de token

5.3.2.6.1 Normalización

- La tabla está en primera forma normal ya que:
  - Todos los atributos son atómicos.
  - Tiene clave primaria única (id).
  - La CP no puede ser nula.

- La tabla está en segunda forma normal ya que:
  - Está en 1FN.
  - Al ser la clave única no puede haber dependencias parciales.

- La tabla está en tercera forma normal ya que:
  - Está en 2FN.
  - No hay dependencias funcionales transitivas.
- La tabla está en forma normal de Boyce-Codd ya que:
  - Para toda dependencia funcional X->A X es superllave.

5.4 Arquitectura del sistema

El sistema completo se compone de varias partes, si bien la aplicación es agnóstica en cuanto a los componentes de los que depende, en este documento se detallará la tencología elegida para cada una de ellas.

La aplicación principal con la lógica de negocio está desarrollada en Python, y depende de una base de datos, en este caso se ha elegido MySQL. Esta aplicación se descompone a su vez en una API que será la que esté expuesta públicamente y un worker para tareas que se ejecutan en segundo plano. Ambos servicios necesitan de la base de datos por lo que tienen acceso a ésta.

Las peticiones de la API más usadas deberán estar en una caché para evitar la sobrecarga de la aplicación y la Base de Datos.

Todos estos servicios estarán controlados por un servidor de aplicaciones, que expondrá un punto de entrada al cual dará acceso un servidor web.

5.4.1 API

La api es un servicio desarrollado en Django, este servicio expone diferentes endpoints REST para realizar diversas labores en la aplicación, dentro de este servicio reside la lógica de negocio, que no está separada de la API en si misma. En caso de que otros servicios necesitaran de esta lógica de negocio la separación sería sencilla ya que el proyecto está dividido en äplicacionesÿ cada una de estas aplicaciones puede extraerse a una librería separada.

5.4.2 Base de datos

Como sistema de gestión de bases de datos se ha elegido MySQL debido a la experiencia previa con este sistema. La base de datos es accesible con un usuario con los permisos que exclusivamente necesita la aplicación.

5.4.3 Worker

Un worker es un servicio que trabaja ejecutando tareas asíncronas que va recibiendo en una cola, este worker descarga de trabajo a la API, de forma que esta pueda responder rápidamente a las peticiones que recibe, derivando los trabajos más pesados a esta cola. Este worker puede recibir tareas que ejecutará inmediatamente, o bien recibir tareas pospuestas para un momento concreto en el tiempo. Este sistema debe poder funcionar de forma distribuida, por lo tanto para funcionar necesita de un backend común para almacenar las tareas hasta que estas sean ejecutadas.

Como tecnología para el worker se ha elegido celery, ya que es la mejor opción existente actualmente en Python, permite funcionar con todos los requisitos expuestos anteriormente.

Como tecnología para el backend se ha elegido redis, que es un almacén de datos clave valor que se ejecuta de forma ligera.

5.4.4 Servidor de aplicaciones

Un servidor de aplicaciones es una pieza de software necesaria para ejecutar cierto tipo de aplicaciones y servirlas a un cliente a través de un puerto. Django y en general Python utilizan el protocolo Wsgi para comunicarse con servidores de este tipo

WSGI es una especificación que actúa de interfaz entre servidores web y aplicaciones web, es el estándar adoptado por Python para este tipo de comunicaciones.

Entre la aplicación y el servidor puede existir un middleware para procesar

la petición y enrutarla a la parte de código correspondiente. En nuestro modelo este middleware sería el router de Django, que se encarga de parsear la URL y enviarla a la vista correspondiente que se encargará de ejecutar la petición, una vez procesada, el middleware devuelve la respuesta al servidor de aplicaciones que se encarga de transmitirla al cliente.

Como implementación de servidor de aplicaciones hemos elegido uWSGI, que es una implementación del protocolo altamente configurable, que además proporciona mejor rendimiento que la mayoría de rivales.

5.4.5 Servidor web

En ocasiones es necesario un servidor web en modo reverse proxy que se encargue de enrutar la petición al servidor de aplicaciones correspondiente, como pueden existir varios servicios para responder a la petición, delante del servidor de aplicaciones se pone este reverse proxy que se encarga de enroutar la petición al servicio adecuado.

En esta ocasión hemos elegido NGINX por su facilidad de configuración. Este servidor también nos sirve para servir ficheros estáticos como imágenes o plantillas.

5.4.6 Infraestructura

Uno de los requisitos no funcionales de la aplicación es la alta disponibilidad, por ello necesitamos un hosting confiable del cual también podamos controlar los gastos.

Además de esto, el hosting no es lo único que necesitamos, también necesitamos poder asignar direcciones DNS, un servidor de email, entre otras cosas.

En los últimos tiempos han ido ploriferando plataformas en la nube como Amazon Web Services (AWS) o Google Cloud Platform. AWS nos ofrece de serie una capa gratuita durante un año, con lo cual nos decidimos por esta plataforma, ya que ofrece todos los servicios que necesitamos.

Otro de los requisitos no funcionales es la escalabilidad, con esto nos referimos a que el servicio pueda adecuarse a los cambios en el tráfico de la aplicación. AWS ofrece de serie servicios autoescalables, de forma que la aplicación se ofrezca en servidores más potentes o se incremente el número de servidores que la sirven automáticamente. Con lo cual en este aspecto también nos puede ser útil.

5.4.7 Escalabilidad

Ya hemos hablado previamente de escalabilidad, pero en un software de estas características, es impor- tante tener claros los elementos que tienen que intervenir para que el servicio pueda responder a una carga alta de tráfico.

5.4.7.1 Balanceador de carga

Un balanceador de carga es un servicio que permite distribuir la carga de trabajo entre varios nodos, evitando así la sobrecarga de uno de estos nodos, lo que provocaría una caída del rendimiento del servicio. Este componente aumenta la fiabilidad del servicio a través de la redundancia, es decir el servicio está replicado en varios nodos y es el balanceador de carga el que se encarga de distribuir las peticiones a base de varios criterios, entre los que estarían la carga de cada nodo, si un nodo se identifica como no saludable, etc.

Este servicio divide la carga usando interfaces de red, por lo que actúa en la capa 4 del modelo OSI, esto quiere decir que utiliza las direcciones IP y el puerto TCP de origen y destino para enrutar la petición.

AWS ofrece Elastic Load Balancer como componente para balanceo de carga. ELB es fácilmente configurable, solo hay que añadir los hosts que servirán el tráfico y ELB se encargará de repartirlo. La pega es que ELB, al ser tan sencillo, también es bastante limitado, no ofrece autoescalabilidad, y reparte la carga por igual, por lo que si las diferentes máquinas configuradas tienen capacidad diferente, unas podrían llegar al límite de carga antes que otras. Con el nivel de carga estimado inicialmente parece suficiente con este mod-

elo, pero aun así hay que tener previsión para mover a un modelo diferente, autoescalable, que permita reducir costes. Para ello, Amazon ofrece una alternativa llamada Application Load Balancer, que es un balanceador de carga pensado para aplicaciones. Este componente actúa en la capa de aplicación, basándose en el contenido de la petición para enrutarla al objetivo correspondiente. Está pensado para arquitecturas de aplicación modernas, como microservicios y aplicaciones basadas en contenedores tipo Docker o Kubernetes, de la cual nos podríamos beneficiar para mejorar la disponibilidad.

5.4.7.2 Servicios autoescalables

Amazon ofrece el servicio Elastic Beanstalk para ejecutar aplicaciones. Este servicio es una abstracción que ofrece Amazon para desplegar aplicaciones sin preocuparnos por el provisionamiento de la máquina, autoescalado y monitorización.

5.4.7.3 Alta disponibilidad

Es importante tener en cuenta las posibles pérdidas de servicio de alguna de nuestras instancias, ni si- quiera un proveedor como AWS es 100 % fiable y hay que estar preparado para posibles fallos.

Para ello AWS ofrece diferentes availability zones donde ejecutar nuestra aplicación, de forma que podemos poner instancias de nuestra aplicación en diferentes AZ y si una de estas AZ cae, tendremos disponibles otras sin que nuestro servicio se vea afectado. En estos casos es el load balancer quién se encarga de redirigir el tráfico en caso de caídas.

5.5 Despliegue

Otra parte importante del diseño de la aplicación es definir claramente como será el proceso de despliegue, esto es, como se llevará el software al entorno

de producción, como se cargará la configuración y como se preparará la infraestructura para que todo funcione.

5.5.1 Despliegue de software

El paquete de software que se entregue tiene que ser en todo momento replicable en cualquier otro entorno, de forma que si queremos volver atrás tengamos la seguridad de que todo va a funcionar. Por tanto todas las tareas de compilación y preparación del software deben hacerse antes del despliegue. Lo que se entregará será un paquete listo para ser ejecutado en cualquier entorno.

5.5.1.1 Paquetizado

Existen diferentes alternativas para este método, en nuestro caso usaremos paquetes Debian, ya que el entorno de despliegue será una máquina con esta distribución. Para preparar un paquete debian primero necesitaremos preinstalar nuestra aplicación en un entorno virtual de Python (virtualenv).

El problema principal de virtualenv es que mantiene las rutas *shebang* de los ficheros que genera de forma absoluta, por tanto mantendrá los de la máquina en la que lo instalemos, pudiendo ser diferentes a las de la máquina en producción. Para solucionar esto existe una herramienta llamada *dh-virtualenv*.

Una vez preparado el virtualenv con la aplicación instalada será eso lo que empaquetemos en el Debian, y será el paquete Debian lo que se entregará a los servidores de producción, de forma que la versión sea replicable.

5.5.1.2 Gestión de configuración

Siguiendo las recomendaciones de buenas prácticas en diseño de software, la configuración no puede estar en el repositorio junto con el código. Hay diferentes razones por las que no hacer esto, algunas de ellas son que la

configuración es dependiente del entorno, por tanto tendremos tantas configuraciones como entornos tengamos, lo cual hace imposible mantener todas en el repositorio, además de esto, muchos valores de configuración son secretos, mantenerlos en claro en el repositorio no es una opción.

Para ello, la configuración se cargará siempre desde variables de entorno, para evitar de ninguna forma mantener ficheros con configuración. Estas variables de entorno se cargarán desde el software de gestión de configuración, que en nuestro caso será Ansible.

5.5.2 Gestión de infraestructura

La infraestructura debe estar preparada y configurada para soportar nuestra aplicación, para ello hay que provisionarla con ciertos valores y necesitaremos una herramienta adecuada que nos permita automatizar el proceso para que también pueda ser replicado, para ello usaremos Puppet, además de Terraform para prerparar el entorno de AWS.

5.6 Métricas y monitorización

Para una aplicación de esta escala se hace imprescindible generar métricas de las cuales en el futuro se pueda extraer información, igualmente la aplicación tiene que estar monitorizada, para en caso de fallos, informar a los administradores del sistema para resolverlo lo antes posible.

5.6.1 Métricas

Las métricas son puntos de datos generados por la aplicación que se envían a un servidor de métricas, estas métricas son configuradas de forma que se pueda extraer información útil de ellas.

Para un software comercial es imprescindible tener este tipo de información para conseguir una mayor monetización, además de poder añadir nuevas características en base a la información extraída.

Estas métricas se componen de un backend o base de datos y de un dashboard para configurarlas y visualizarlas, para el backend hemos elegido *InfluxDB* que es una de las bases de datos más utilizadas para este propósito.

InfluxDB es una base de datos de código abierto que almacena series basadas en el tiempo. Está escrita en *Go* y optimizada para ser usada en entornos de tiempo real y de alta disponibilidad.

Como dashboard hemos utilizado *Grafana*, es un dashboard configurable totalmente compatible con *InfluxDB*.

5.6.2 Monitorización

La monitorización es una parte importante para controlar la estabilidad del sistema, sin unas herramientas adecuadas no podemos asegurar la disponibilidad y el buen funcionamiento de nuestro software, para ello existen diferentes herramientas, como la anteriormente mencionada *Grafana* y a nivel de procesos *monit*, monit nos asegura que nuestros servicios estarán siempre levantados, si hay alguna caída, la herramienta los volverá a levantar.

5.6.3 Alertas

En caso de caída de servicio necesitamos tener un sistema de alertas que nos comunique el fallo correspondiente, para ello se utilizará además de *monit*, *Sensu*, que es un sistema configurable de checks para nuestros servicios, *Sensu* es a su vez compatible con sistemas de alerta a equipos de operaciones, como por ejemplo *OpsGenie*.

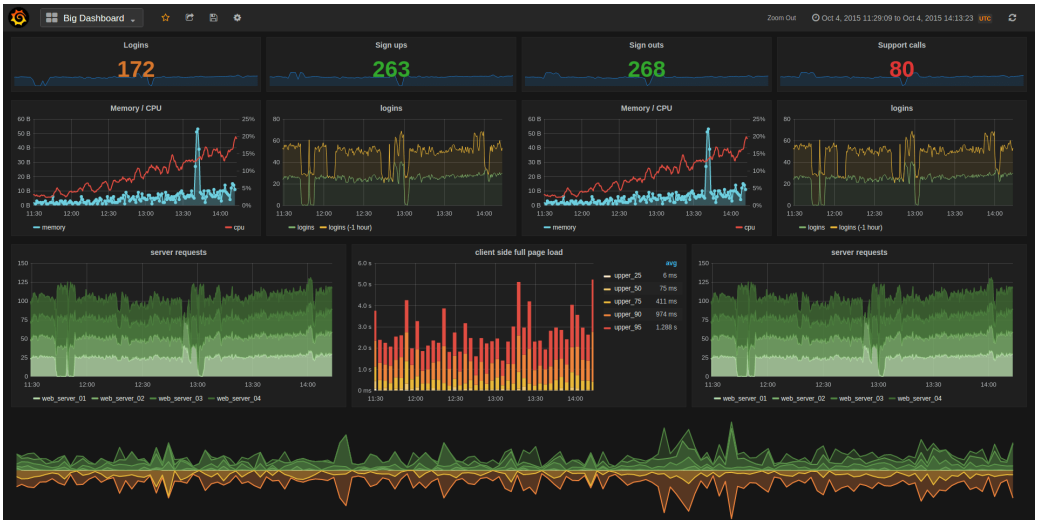


Figure 5.1: Dashboard de ejemplo de Grafana

Chapter 6

Implementación

Llegada la etapa de implementación, hay que transformar a código lo analizado y diseñado en etapas anteriores. La idea original era hacer una aplicación web, esta aplicación estaría separada en dos subsistemas, uno de administración y otro de integración con aplicaciones.

6.1 Subsistema de administración

Este subsistema consiste en una aplicación web con diferentes paneles para gestionar cada una de las partes del sistema, usuarios, empresas, etc.

A este subsistema tendrán acceso los administradores de la empresa principal con licencia del software, cada usuario tendrá visibilidad limitada según sus permisos. Empresas externas clientes de la empresa licenciada podrán tener acceso pero con visibilidad muy limitada, solo para gestionar los usuarios propios y el acceso a las aplicaciones que tenga contratadas.

6.1.1 Lenguajes

Para esta sección se ha utilizado *Python* como lenguaje principal, usando *Django* como framework de desarrollo web. La razón por la que se eligen *Python* y *Django* es por la actual experiencia del autor del proyecto en estos

lenguajes, además de existir una comunidad bastante grande alrededor de ellos, con una documentación amplia disponible.

Python es un lenguaje fácil de aprender y con parecido al lenguaje natural, por lo que facilita la labor a la hora de hacer un proyecto de estas características, *Django* además provee de diferentes herramientas para abstraer el desarrollo de una aplicación web e implementar solo la lógica de negocio.

Django sigue una variación del patrón MVC, en el cual separa la capa de modelo de datos del resto, en esta capa está la lógica del dominio del negocio, aunque se puede separar en más subniveles. En lugar del clásico controlador, *Django* usa el concepto de Vista, que son funciones que reciben la petición del servidor y devuelven una respuesta acorde utilizando la lógica de los modelos. El equivalente a la vista del MVC sería en este caso el sistema de plantillas, plantillas que dado el caso la View de *Django* renderiza con los datos necesarios.

Para la interfaz se ha aprovechado la librería *django-admin* que autogenera formularios para los modelos implementados, además sobre esta se utiliza una capa llamada *django-suit* para mejorar el resultado de la interfaz que se mostrará al usuario, esto ahorra trabajo de maquetación y diseño, trabajos que escapan al alcance de este proyecto.

6.2 Subsistema de integración

Este subsistema consiste en una API REST para que las aplicaciones puedan interactuar con el sistema central, este subsistema pone a disposición de la aplicación tokens de acceso con los cuales puede acceder a la API y realizar diferentes labores, además de identificar al usuario.

6.2.1 Lenguajes

Para este subsistema se ha utilizado también *Python* y *Django* pero con el añadido de usar *django-rest-framework* como capa por encima de *Django*.

Este framework nos proporciona varias herramientas para construir una API REST con *Django*, como son los serializadores y las vistas genéricas.

6.2.2 Herramientas utilizadas

Para el desarrollo del proyecto se hace necesario el uso de una serie de herramientas, como editores de código, sistemas de control de versiones, etc.

A continuación se detallarán todas las herramientas usadas en este proyecto.

La herramienta principal que se ha utilizado ha sido un IDE, en este caso *PyCharm*. *PyCharm* es un entorno escrito en Java y pensado en un principio para desarrollar proyectos Python, es una evolución de *IntellijIdea*, el cual es una herramienta para desarrollar *Java*, pero conforme ha avanzado el tiempo se ha ido ampliando a más lenguajes, como por ejemplo Python. La integración con este último es perfecta, proporcionando útiles herramientas como el autocompletado.

Para la detección de errores se hace casi obligado el uso de un debugger, en este caso hemos usado el debugger integrado de PyCharm, permitiendo utilizar puntos de ruptura en el código para comprobar el estado del sistema en un momento dado.

Para el despliegue de la aplicación se ha utilizado un entorno compuesto por un servidor Nginx, base de datos MySQL y el intérprete de *Python*, todo ello sobre un sistema GNU/Linux.

Otra herramienta utilizada que facilita el trabajo enormemente ha sido Git. Git es un sistema de control de versiones que facilita el desarrollo colaborativo y el mantenimiento de un software, versionando todos los cambios que se vayan produciendo en el código. Esto permite que si queremos volver a una versión anterior del sistema podamos hacerlo sin problema alguno, además de la creación de ramas de desarrollo, pudiendo fusionar ramas sin problema alguno.

6.3 Detalles de implementación de la arquitectura del sistema

6.3.1 Capa modelo

Como hemos dicho se ha utilizado *Django* como framework de desarrollo, *Django* trae integrado un ORM que abstrae el acceso a base de datos, funcionando de forma agnóstica con cualquier SGBD. Para construir un modelo definimos sus atributos de clase, además de sus relacione, utilizando los modelos que nos proporciona *Django*. *Django* a su vez, utilizando el comando `make migrations` generará las migraciones correspondientes que una vez ejecutadas con `migrate` se aplicarán en la base de datos y crearán las tablas necesarias.

Un ejemplo de modelo es el siguiente:

```
class Account(models.Model):
    objects = AccountManager()

    EMAIL_FIELD_MAX_LENGTH = 255

    id = models.AutoField(primary_key=True,
                          help_text="the account id,
                             e.g. 1234")
    uid = UUIDField(default=get_uuid,
                    help_text="A unique uuid string")
    enabled = models.BooleanField(default=True,
                                  help_text=("A flag
                                     that indicates
                                     if the account
                                     is enabled or "
                                             "not. A
                                                disabled
                                                account
```

```
                                                    cannot
                                                    log
                                                    in
                                                    the
                                                    system
                                                    and"
14                                                "will
                                                    not
                                                    receive
                                                    notifications."))
    email =
       models.EmailField(max_length=EMAIL_FIELD_MAX_LENGTH,
16                             help_text="Account
                                  email, e.g.
                                  example@gmail.com")
    password = models.CharField(max_length=56,
18                               help_text="The
                                    account holder's
                                    password.")
    name = models.CharField(max_length=40,
20                            help_text="The account
                                 holder's name.")
```

Una capa intermedia de los modelos es la capa de Model Managers, esta capa permite abstraer diferentes consultas a la base de datos para no repetirlas cada vez.

Un ejemplo sería el siguiente:

```
class AccountManager(models.Manager):
2    @transaction.atomic
    def create(self, **kwargs):
4        account = super(AccountManager,
            self).create(**kwargs)
         account.groups.add(self.default_group)
6        return account
```

```
    @cached_property
    def default_group(self):
        return
           Group.objects.get_by_natural_key('default')

    def get_by_id_or_uid(self, uid):
        if isinstance(uid, int) or uid.isdigit():
            params = {'pk': uid}
        elif len(uid) == 32:
            params = {'uid': uid}
        else:
            from api.models import Account
            raise Account.DoesNotExist()

        return self.get(**params)
```

En cuanto a las migraciones, son sentencias que permiten versionar los cambios en la base de datos, un ejemplo sería el siguiente:

```
class Migration(migrations.Migration):

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Account',
            fields=[
                ('id',
                   models.AutoField(serialize=False,
                   primary_key=True,
                   db_column=b'ac_id')),
                ('enabled',
                   models.BooleanField(default=True,
```

```
                    db_column=b'ac_enabled')),
                ('email',
                    models.EmailField(max_length=255,
                    db_column=b'ac_email')),
                ('password',
                    models.CharField(max_length=56,
                    db_column=b'ac_password')),
                ('name',
                    models.CharField(max_length=40,
                    db_column=b'ac_name')),
                ('surname',
                    models.CharField(default=b'',
                    max_length=40,
                    db_column=b'ac_surname',
                    blank=True)),
            ],
            options={
                'db_table': 'Account',
            },
        ),
```

6.3.2 Capa View (Controlador)

La capa controlador es la que se encarga de recibir las peticiones de la api y devolver una respuesta adecuada, es la capa de interfaz al exterior por lo que es bastante importante.

El controlador se compone de un Router y las Vistas, el Router se encarga de parsear las urls y pasar la petición a la vista correspondiente, un ejemplo de router sería el siguiente:

```
urlpatterns = [
    url(
        regex=r'^$',
        view=views.UserListView.as_view(),
```

```
        name='list'
    ),
    url(
        regex=r'^~redirect/$',
        view=views.UserRedirectView.as_view(),
        name='redirect'
    ),
    url(
        regex=r'^companies/(?P<slug>[\w.@+-]+)/users/(?P<username>[\
        view=views.UserDetailView.as_view(),
        name='detail'
    ),
    url(
        regex=r'^~update/$',
        view=views.UserUpdateView.as_view(),
        name='update'
    ),
]
```

Las vistas genéricas de *django-rest-framework* se componen de varios atributos para configurarlas, permitiendo generar una respuesta sin escribir apenas código, un ejemplo sería el siguiente:

```
class
   AccountDetailView(generics.RetrieveUpdateDestroyAPIView):
    queryset = Account.objects.all()
    serializer_class = AccountSerializer
    permission_classes = (UserCanManagePermission,)
    lookup_field = 'id'
    lookup_url_kwarg = 'account_id'
```

La vista del ejemplo genera acciones para el GET en detalle, para el PUT y para el DELETE, utilizando el queryset configurado.

6.3.3 Capa Templates (Vista)

Para esta capa hemos utilizado *django-admin* con la capa de *django-suit*, las vistas se configuran mediante código Python y en parte se autogeneran a partir de los modelos.

Ejemplo:

```
class AccountAdmin(DjangoObjectActions,
   admin.ModelAdmin):
    model = Account
    list_display = ('id', 'name', 'enabled')
    list_filter = ['enabled']
    search_fields = ['name']

    readonly_fields = []

    def get_readonly_fields(self, request, obj=None):
        return [f.name for f in
           self.model._meta.fields]
```

Chapter 7

Seguridad

Cuando comercializamos una aplicación que contendrá datos sensibles de usuarios y empresas como esta un aspecto importante es asegurar que todos estos datos son accedidos solo por la gente que debe, además de asegurar la persistencia e integridad de estos datos.

7.1 Seguridad para el software

Es importante poder tracear de que forma se ha accedido al sistema y quién ha modificado cada cosa. Para ello es importante mantener un registro de auditoría del sistema.

7.1.1 Subsistema de administración

Para el subsistema de administración se mantiene un log de eventos en forma de tabla en la base de datos, este log indica cada acción realizada, sobre que tabla y qué usuario, además se almacenan también los intentos de acceso al sistema de administración.

7.1.2 Subsistema de integración

Al igual que en el otro subsistema, en este caso se hace necesario saber qué aplicación está accediendo y en nombre de quién, para ello en la misma tabla anterior añadimos un campo application_id, para saber desde qué aplicación se está realizando la modificación, siendo nulo en caso de haberse realizado directamente desde el panel de administración.

7.1.3 Logs

Además de los datos anteriores, es necesario guardar logs de aplicación en el servidor de logs correspondiente, para ello utilizamos la herramienta *syslog-ng* para redirigir los logs generados por la aplicación al servidor correspondiente. La aplicación debe generar logs útiles de todas las acciones que se realicen, de forma que sean trazables.

7.2 Seguridad de los datos

Al ser una aplicación multi-tenant los datos no deben ser accesibles de un cliente a otro, ya que en principio no estarían físicamente separados, la forma de hacerlo será por software, en forma de esquemas diferentes de base de datos, asegurando así que no serán accedidos de un tenant a otro.

7.2.1 Copias de seguridad

Al no ser datos extremadamente críticos se decide hacer una copia de seguridad al día, esta copia está gestionada por AWS, de forma que no tenemos que preocuparnos mucho más que de configurarla.

7.2.2 Mecanismos de integridad

La base de datos ayudada por el ORM mantiene en todo momento integridad referencial, al ser una base de datos SQL con sus relaciones definidas.

7.3 Seguridad para el usuario

También se debe asegurar que cada usuario y cada aplicación solo accede a los datos que debe, para ello se configuran permisos para cada usuario y luego se añade la capa de OAuth, para la cual solo se otorgan permisos limitados por token. A continuación se explicará el protocolo OAuth 2.0 en detalle.

7.3.1 OAuth

OAuth 2.0 es el protocolo estándar de la industria para autorización. OAuth 2.0 se enfoca en la simplicidad para proveer flujos de autorizaciones para aplicaciones, ya sean de tipo web, escritorio, móvil, etc. OAuth es una especificación desarrollada en el marco de la IETF.

7.3.1.1 Flujos de OAuth

OAuth dispone de varios flujos de funcionamiento, en función de si la aplicación es pública o privada. A continuación se explicarán todos los flujos disponibles.

7.3.1.1.1 Flujo de autorización en 3 pasos Este flujo está pensado para aplicaciones públicas de terceros, en las cuales no queremos que la aplicación tenga acceso a las credenciales de usuario, para ello en primer lugar se envía al usuario al servidor de autorización, mostrando una interfaz para introducir sus credenciales, estas credenciales son introducidas directamente en el servidor, por lo que la aplicación no tiene acceso a ellas.

Una vez validadas el servidor devuelve un código temporal de autorización a la aplicación, que debe disponer de un servidor con una url para recoger este código.

Una vez obtenido el código este debe ser intercambiado por un token de acceso definitivo, para ello con el código y las credenciales de aplicación, ésta realiza una petición al servidor de autorización para intercambiar el código por un token, el servidor devuelve el token y el código deja de ser válido.

7.3.1.1.2 Flujo de autorización en 2 pasos Este flujo existe para aplicaciones públicas confiables, en principio aplicaciones desarrolladas por el mismo proveedor de autorización pero que funcionan de forma pública, tipo aplicaciones móvil, etc. En estas aplicaciones conocemos que no almacenarán las credenciales del usuario por lo que son confiables.

En primer lugar la aplicación realiza una petición al servidor con las credenciales de aplicación y de usuario, el servidor responde directamente con el token de acceso y la aplicación ya puede empezar a funcionar.

7.3.1.1.3 Flujo de autorización para aplicaciones Este flujo existe para aplicaciones privadas desarrolladas por el proveedor de autorización, que son totalmente confiables, estas aplicaciones utilizan solamente sus credenciales de aplicación, sin necesitar las de usuario, y en principio tendrían más privilegios que las normales, ya que no están disponibles de forma pública.

En primer lugar la aplicación realiza una petición al servidor enviando sus credenciales, el servidor responde con un token de acceso.

7.3.2 Permisos

Cada usuario tiene unos permisos definidos con lo que puede ver en el sistema, además de esto se pueden configurar permisos personalizados por aplicación que serán otorgados luego al usuario.

7.3.3 Scopes

Los scopes son permisos que la aplicación obtiene sobre un usuario para actuar en su nombre.

7.4 Seguridad del hardware

Esta capa a pesar de ser importante está gestionada por AWS, al haber configurado los sistemas de forma que sean fácilmente replicables no tenemos que preocuparnos por este aspecto.

Chapter 8

Conclusion

8.1 Thesis summary

In summary, pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nunc eleifend, ex a luctus porttitor, felis ex suscipit tellus, ut sollicitudin sapien purus in libero. Nulla blandit eget urna vel tempus. Praesent fringilla dui sapien, sit amet egestas leo sollicitudin at.

8.2 Future work

There are several potential directions for extending this thesis. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida ipsum at tempor tincidunt. Aliquam ligula nisl, blandit et dui eu, eleifend tempus nibh. Nullam eleifend sapien eget ante hendrerit commodo. Pellentesque pharetra erat sit amet dapibus scelerisque.

Vestibulum suscipit tellus risus, faucibus vulputate orci lobortis eget. Nunc varius sem nisi. Nunc tempor magna sapien, euismod blandit elit pharetra sed. In dapibus magna convallis lectus sodales, a consequat sem euismod. Curabitur in interdum purus. Integer ultrices laoreet aliquet. Nulla vel dapibus urna. Nunc efficitur erat ac nisi auctor sodales.

Appendix 1: Some extra stuff

Add appendix 1 here. Vivamus hendrerit rhoncus interdum. Sed ullamcorper et augue at porta. Suspendisse facilisis imperdiet urna, eu pellentesque purus suscipit in. Integer dignissim mattis ex aliquam blandit. Curabitur lobortis quam varius turpis ultrices egestas.

Appendix 2: Some more extra stuff

Add appendix 2 here. Aliquam rhoncus mauris ac neque imperdiet, in mattis eros aliquam. Etiam sed massa et risus posuere rutrum vel et mauris. Integer id mauris sed arcu venenatis finibus. Etiam nec hendrerit purus, sed cursus nunc. Pellentesque ac luctus magna. Aenean non posuere enim, nec hendrerit lacus. Etiam lacinia facilisis tempor. Aenean dictum nunc id felis rhoncus aliquam.

Chapter 9

References