Portada

− Resumen (de una página)

− Índice

− Introducción.

− Objetivos (incluyendo descripción del problema, estudio de alternativas y metodología empleada).

− Descripción informática (puede incluir especificación, diseño e implementación).

− Conclusiones (incluyendo los logros principales alcanzados y posibles trabajos futuros).

− Bibliografía

− Apéndices

RESUMEN

El uso de las redes sociales está en auge, y bien sabemos todos que forman parte del día a día, de casi todos los ciudadanos. El concepto red social se extiende cada vez más, y en la actualidad van surgiendo diferentes tipos de las mismas, en las que podemos conversar, subir fotos, conocer gente, e incluso encontrar trabajo, pero a veces nos falta una visión más generalizada de un mismo usuario en concreto.

Ahí es donde surge la motivación de este proyecto. Es el intento de unión de dos de estas redes más utilizadas. Hablamos de TWITTER y LINKEDIN. Esta aplicación, obtiene información resumida de un usuario, en la que podemos ver, sus aficiones, gustos, y sus últimos tweets, y retweets, mezclado todo ello con unos datos más técnicos y profesionales, en los que observaremos el empleo actual (en caso de que lo haya) y un resumen de su CV. Es decir, obtenemos en un “click” una visión general del usuario en cuestión. Todo esto contribuye al nacimiento de Founbep.

Founbep es una aplicación web, implementada en Python, mediante el framework Django, usando además, HTML, CSS y SQLite como modelo de base de datos. El motor básico de esta app es el acceso a las API tanto de Twitter como de LinkedIn, donde accederemos a los datos de un usuario en concreto.

Índice

1. Introducción
   1. Contexto
   2. Motivación
   3. Estructura de la memoria
2. Objetivos
   1. Descripción del problema
   2. Objetivos
   3. Estudio de alternativas
      1. Lenguajes de programación
         1. Python
         2. Java
         3. PHP
         4. Framework Django
         5. HTML
         6. CSS
      2. Bases de datos
         1. SQLite
   4. Metodología empleada
3. Descripción informática (puede incluir especificación, diseño e implementación).
   1. Captura de requisitos
      1. Requisitos funcionales
      2. Requisitos no funcionales
   2. Casos de uso
      1. Diagrama de casos de uso
      2. Descripción de los casos de uso
   3. Arquitectura de alto nivel
   4. Diagrama E-R
   5. Algoritmo utilizado
   6. Descripción de los módulos
4. Pruebas
   1. Pruebas unitarias
      1. Pruebas de caja blanca
      2. Pruebas de caja negra
5. Conclusiones y trabajo futuro.
6. Bibliografía
7. Anexo

**ABREVIACIONES**

API **A**pplication **P**rogramming **I**nterface

HTML **H**yper**T**ext **M**arkup **L**anguage

CSS **C**ascading **S**tyle **S**heets

MVC **M**odelo-**V**ista-**C**ontrolador

Capítulo 1

**INTRODUCCION**

La finalidad de este proyecto de fin de carrera es la creación de una aplicación web, mediante la cual se podrá obtener una visión general en cuanto a los datos se refiere, de un usuario. Tendremos un híbrido entre información de gustos y aficiones del usuario, y por otro lado, datos profesionales, técnicos, experiencia laboral…Todo esto lo conseguiremos accediendo a las apis de LinkedIn y de Twitter, herramientas gratuitas para desarrolladores que nos aportan información pública y útil de los users registrados.



Figura1. Apariencia principal de FoundBep.

* 1. **Contexto**

Aprovechando el auge de las redes sociales, y del potencial de estas herramientas para conectar mundos y personas, surge una primera idea de la creación de *FoundBep.* Estas aplicaciones se han incorporado a nuestras vidas formando parte de ellas y ahora podemos compartir información de nuestras rutas en bicicleta, jugar contra nuestros amigos y superarles en puntuación, realizar compras desde casa sin ningún tipo de esfuerzo, y mil usos cotidianos que nos hacen ejercitar cuerpo y mente.

Estas redes cumplen de cierta forma la falta de roce que trae consigo el desarrollo de la vía moderna. Gracias a la facilidad que ofrecen para conocer personas y de establecer relaciones de amistad con gente que posee gustos e intereses similares a los nuestros, las redes sociales son un medio por el cual podemos estar en contacto con diversas personas alrededor del mundo. Y lo que es más importante, son fáciles de consultar y posibilitan la obtención de la información más relevante en tiempo real.

* 1. **Motivación**

Todas estas redes sociales, tienen en común, que conectan miles de usuarios mundialmente, ya sea mediante fotos, comentarios, *likes,* pero en muchas de ellas se echa en falta un perfil más generalizado de un usuario. Hablamos de general en cuanto a saber cuáles son sus aficiones, gustos, y saber por otro lado donde trabaja, cuáles son sus estudios… En el caso de Twitter, puedes conocer a un usuario por sus aficiones y gustos en cualquier ámbito social, desde política, hasta deportes, y por el contrario no sueles saber nada de ese mismo usuario en cuanto a su vida profesional. Puedes observar algún tweet aislado, pero con exactitud no sabrás cuál es su Curriculum Vitae, ni en que se especializó al salir de la carrera…cosa que por ejemplo si podrás ver en LinkedIn.

Las redes sociales, son aplicaciones que sacian la sed del usuario en un aspecto concreto, ya sea socializarse, desconectar, buscar sitios de ocio cerca de su ciudad, encontrar trabajo…es decir hay infinidad de usos, pero a veces nos gustaría encontrar varios de esos perfiles en una misma red social. Es cierto que en alguna de ellas se puede ver algo más generalizado, como quizás en el perfil de Facebook, pero sigue careciendo de poseer más detalles.

Ahí es donde entra en acción *FounBep*, mostrando de manera clara, y muy visual, los pilares que forman al usuario.

* 1. **Estructura de la memoria**

La memoria del proyecto está estructurado en los siguientes capítulos:

1. **Introducción**

En este capítulo se llevará a cabo una introducción al contexto de las redes sociales en la sociedad actual, para tener una idea general del proyecto.

1. **Objetivos**

Concretaremos los objetivos principales que tiene que cumplir la aplicación desde que el usuario se registra.

1. **Descripción informática**

Resumen de las tecnologías utilizadas en el proyecto, y donde se expondrán todas las herramientas y el porqué de su elección frente a otras.

1. **Pruebas**

Este capítulo contiene las pruebas unitarias ejecutadas sobre el proyecto.

1. **Conclusiones**

Apartado dedicado a las conclusiones que han nacido al realizar el proyecto. Se comentará adicionalmente si finalmente se han cumplido los objetivos previamente establecidos.

1. **Trabajo Futuro**

Relacionado con el apartado anterior, el capítulo de “trabajo futuro” comentara de forma breve las funcionalidades adicionales que podrían hacerse en versiones posteriores del proyecto, siendo factibles.

Capítulo 2

**OBJETIVOS**

**2.1 Descripción del problema**

**2.2 Objetivos**

Queda claro, que el objetivo principal de *FoundBep* es obtener de manera sencilla y clara un vistazo general del usuario. Esta herramienta sigue siendo un ejercicio sencillo y demostrativo, pero la idea de poseer información de alguien de manera más completa es la raíz principal en la que se basa este proyecto.

**2.3 Estudio de alternativas**

Para el desarrollo de una aplicación web necesitamos la elección de un lenguaje principal de programación, un *framework* que nos ayudará a la implantación, y un gestor de base de datos, por lo que en este apartado nos meteremos de lleno en la comparación de varios de los mismos, y el porqué de su elección final.

**2.3.1 Lenguajes de programación**

**2.3.1.1 Java**

La principal característica de Java es la de ser un lenguaje compilado e interpretado. Todo programa en Java ha de compilarse y el código que se genera, es interpretado por una máquina virtual. De este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son independientes de la plataforma.

Java es un lenguaje orientado a objetos, de propósito general. Aunque Java comenzara a ser conocido como un lenguaje de programación de applets que se ejecutan en el entorno de un navegador web, se puede utilizar para construir cualquier tipo de proyecto.



Figura 2. Java - Spring

**2.3.1.2 Python**

Es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa, y en menor medida programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. Su filosofía reside en que su sintaxis favorezca a un código legible.

Es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel. Es un lenguaje ideal para el scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.



Figura 3. Python – Django

**2.3.1.3 PHP**

PHP es un lenguaje de programación de uso general de código del lado servidor originalmente diseñado para el desarrollo de contenido web dinámico. Fue uno de los primeros lenguajes de programación del lado servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos.

PHP se utiliza para generar páginas web dinámicas. Recordar que llamamos página estática a aquella cuyos contenidos permanecen siempre igual, mientras que llamamos páginas dinámicas a aquellas cuyo contenido no es el mismo siempre. Por ejemplo, los contenidos pueden cambiar en base a los cambios que haya en una base de datos, de búsquedas o aportaciones de los usuarios, etc



Figura 4. PHP – Symfony2

Cualquiera de los lenguajes anteriores, nos habrían servido perfectamente para el desarrollo de nuestro proyecto, pero al final nos decantamos por el lado de Python. Java era el único de los tres al que tenía el placer de haber conocido de cerca, y partía con ventaja, pero Python es un lenguaje muy sencillo de comprender a pesar de no tener experiencia con él, así que la balanza se volcó hacia su lado.

Adicionalmente, hubo un trabajo de investigación respecto al acceso a las Apis, tanto de Twitter como de LinkedIn. En el primer caso, tenemos librerías en casi todos los lenguajes principales, pero en el caso de LinkedIn, Python contaba con más documentación, foros, y centros de ayuda, con los cuales podría resolver los futuros problemas que surgieran, y pensando en el futuro, debido a la poca o nula experiencia en Python, valoramos el gran contenido de apoyo y documentación como algo fundamental.

**2.3.1.4 Framework Django**

Django es un framework web de código abierto escrito en Python que permite construir aplicaciones web más rápido y con menos código.

Django fue inicialmente desarrollado para gestionar aplicaciones web de páginas orientadas a noticias de World Online, más tarde se liberó bajo licencia BSD. Django se centra en automatizar todo lo posible y se adhiere al principio DRY (Don't Repeat Yourself).

También se adscribe al diseño *MVC* (Modelo-Vista-Controlador), por lo que las diferentes partes del sitio están claramente separadas. Por ejemplo, el código de acceso a los datos es completamente independiente al que gobierna el aspecto externo de la página.

Al tratarse de Python, Django permite que el desarrollador escriba código ágilmente. El resultado son menos líneas de código y, en consecuencia, menos probabilidades de que haya bugs. De ahí que digan que “fomenta el desarrollo rápido y el diseño limpio y pragmático”.

**2.3.1.5 HTML**

HTML es un lenguaje de programación que se utiliza para el desarrollo de páginas de Internet. Se trata de la sigla que corresponde a *HyperText Markup Language*, es decir, Lenguaje de Marcas de Hipertexto, que podría ser traducido como Lenguaje de Formato de Documentos paEste lenguaje es el que se utiliza para especificar los nombres de las etiquetas que se utilizarán al ordenar, no existen reglas para dicha organización, por eso se dice que es un sistema de formato abierto.

EL HTML se encarga de desarrollar una descripción sobre los contenidos que aparecen como textos y sobre su estructura, complementando dicho texto con diversos objetos (como fotografías, animaciones, etc).

Es un lenguaje muy simple y general que sirve para definir otros lenguajes que tienen que ver con el formato de los documentos. El texto en él se crea a partir de etiquetas, también llamadas tags, que permiten interconectar diversos conceptos y formatos.



Figura 5. Logo HTML 5

**2.3.1.6 CSS**

Al igual que HTML nos permite definir la estructura de nuestra página web, CSS es el encargado de establecer el diseño y estilo de la página en los diferentes dispositivos. Por tanto, las hojas de estilo nos permiten definir la representación de la página web de manera eficiente logrando separar el contenido de la forma.

El lenguaje CSS se basa en una serie de reglas que rigen el estilo de los elementos en los documentos estructurados, y que forman la sintaxis de las hojas de estilo. Cada regla consiste en un selector y una declaración, esta última va entre corchetes y consiste en una propiedad o atributo, y un valor separados por dos puntos.

Ejemplo:

*h2 {color: green;}*

*h2 ---> es el selector*

*{color: green;} ---> es la declaración*

*color ---> es la propiedad o atributo*

*green ---> es el valor*

El *Selector* especifica que elementos HTML van a estar afectados por esa declaración, de manera que hace de enlace entre la estructura del documento y la regla estilística en la hoja de estilo.

La *Declaración* que va entre corchetes es la información de estilo que indica cómo se va a ver el selector. En caso de que haya más de una declaración se usa punto y coma para separarlas.

El uso de CSS permite conseguir efectos visuales que antes solo eran posibles con el uso de tecnologías adicionales. Además ahorramos tiempo y trabajo al poder seguir varias técnicas (bordes redondeados, sobras en el texto, sobras en las cajas, gradientes…) sin necesidad de usar un editor gráfico.



Figura 6. Logo CSS 3

**2.3.1 Bases de datos**

**2.3.1.1 SQLite**

Django soporta de manera predeterminada la conexión con *postgresql*, *mysql*, *sqlite3* y *oracle*. En nuestro proyecto usaremos sqlite3.

Con toda esta filosofía en mente, vamos a comenzar a explorar la capa de la base de datos de Django. Primero, necesitamos tener en cuenta algunas configuraciones iniciales: necesitamos indicarle a Django qué servidor de base de datos usar y cómo conectarse con el mismo.

Asumimos que ha configurado un servidor de base de datos, lo has activado, y has creado una base de datos en este (por ej. usando la sentencia *CREATE DATABASE*). *SQLite* es un caso especial; es este caso, no hay que crear una base de datos, porque *SQLite* usa un archivo autónomo sobre el sistema de archivos para guardar los datos.

*SQLite* merece especial atención como herramienta de desarrollo. Es un motor de base de datos extremadamente simple y no requiere ningún tipo de instalación y configuración del servidor. Es por lejos el más fácil de configurar si sólo quieres jugar con Django, y viene incluido en la librería estándar de Python 2.5.



Figura 7. Diferentes tipos de BD que se pueden conectar a Django.

**2.4 Metodología empleada**

Para la creación de nuestro proyecto, seguiremos el Proceso Unificado de Desarrollo Software, un marco de desarrollo guiado por casos de uso, centrado en la arquitectura y con un ciclo de vida iterativo e incremental.

Está compuesto de cuatro fases denominadas *Inicio, Elaboración, Construcción y Transición*. Cada una de estas fases es a su vez dividida en una serie de iteraciones.

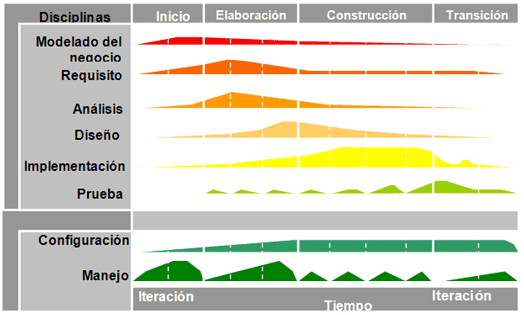


Figura 8. Proceso Unificado de Desarrollo SW

En cada iteración se llevaran a cabo las siguientes etapas:

* Análisis de requisitos

En esta fase se describen los objetivos y necesidades del sistema y poner a los desarrolladores y al cliente de acuerdo en esta descripción.

* Análisis y diseño

Describe como el SW será realizado en la fase de implementación. Se plasma en un modelo de diseño que consiste en una serie de clases con interfaces bien definidos.

* Implementación

En esta etapa se realiza la codificación del sistema siguiendo todos los objetivos y documentos creados en las etapas anteriores.

* Pruebas

Se comprueba que el funcionamiento es correcto analizando diversos aspectos: los objetos como unidades, la integración entre objetos, la implementación de todos los requisitos, entre otros.

* Evaluación

Se evalúan la facilidad de manejo y la utilidad de la interfaz a través de pruebas con varios usuarios.

Capítulo 3

**DESCRIPCIÓN INFORMÁTICA**

**3.1 Captura de requisitos**

En este apartado se llevara a cabo la definición de requisitos de SW del sistema. Tiene por objeto averiguar que se debe construir. Ello implica que el cliente y los desarrolladores tienen que tener claro lo que debe y no debe hacer el sistema.

El propósito de esta especificación es ayudar a tener una idea general del producto y ofrecer una guía para pasos posteriores. Además, servirá para ayudar a entender las necesidades y funcionalidades de SW de cara a la implementación.

**3.1.1 Requisitos funcionales**

Los requisitos funcionales describen los requisitos a tener en cuenta en cuanto a la funcionalidad del sistema. Será necesario que el sistema satisfaga estos requisitos.

Mediante casos de uso. El caso de uso es una funcionalidad que proporciona el sistema. Para el usuario, el caso de uso representa la forma de usar el sistema. También se especificara la forma de la interfaz de usuario para la ejecución de los casos de uso.

Lista de requisitos funcionales

RF1 – El usuario podrá darse de alta en la aplicación. Para ello tendrá que dejar su Twitter y su URL publica de LinkedIn para poder enlazar con sendas *apis.*

RF2 – El usuario una vez registrado, podrá autenticarse en la herramienta, con su *user* y su *password.*

RF3 – El usuario podrá ver información de su perfil.

RF4 – El usuario podrá buscar en el sistema a otros usuarios ya registrados anteriormente para ver sus perfiles.

RF5 – El usuario podrá dejar comentarios en los demás perfiles.

RF6 – El usuario podrá volver a su perfil.

RF7 – El usuario podrá cerrar sesión y volver a la pantalla de login/registro.

**3.1.2 Requisitos no funcionales**

Especifican propiedades del sistema como restricciones, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, fiabilidad, etc.

Lista de requisitos no funcionales

RNF 1 – Nuestro sistema tendrá una interfaz intuitiva y sencilla, llegando así a un mayor número de usuarios. (Principios Interacción Persona-Ordenador)

RNF 2 – El sistema deberá responder en tiempo real a las búsquedas del usuario, evitando respuestas lentas.

RNF 3 – El sistema podrá usarse desde cualquier ordenador y cualquier sistema operativo.

**3.2 Casos de uso**

Los diagramas de casos de uso representan las diferentes operaciones que debe realizar el sistema, y la manera de actuar con su entorno, más concretamente, los usuarios y otras aplicaciones que interactúan con él.

Cada manera en que los usuarios emplean el sistema representara un caso de uso y uniendo todos los casos tendremos la especificación total del sistema.

**3.2.1 Diagrama de casos de uso**

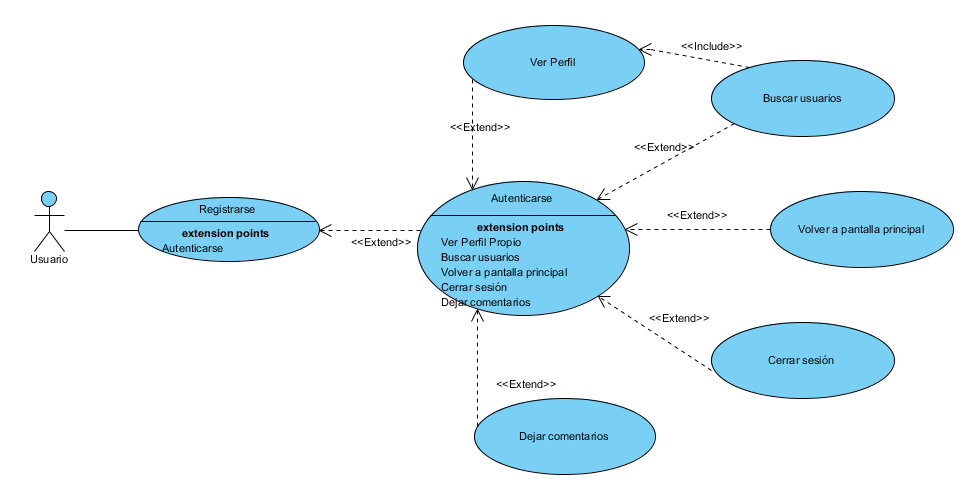


Figura 9. Diagrama de casos de uso.

**3.2.2 Descripción de los casos de uso**

* Caso de uso: RF1- **“Darse de alta”**

**- Objetivo:** Dar de alta al usuario en la herramienta.

**- Actores:** Nuevo usuario de la aplicación.

**- Precondición:** La única precondición para registrarse es que el usuario acceda a la herramienta desde un navegador web.

**- Descripción:** El usuario introduce sus datos (Twitter, LinkedIn, correo, password) y quedan almacenados para el posterior acceso a la aplicación.

|  |  |
| --- | --- |
| USUARIO | SISTEMA |
| 1. El usuario accede a la app y decide registrarse, con lo que introduce sus datos. Una vez completado el formulario, pulsa “Registrar”. | 2. El sistema corrobora que dicho *user* no está duplicado en la BBDD, y que todos los datos son correctos. OK. |
|  | 3. La aplicación registra dichos datos en la BBDD, y avisa al usuario con un mensaje por pantalla de que todo ha ido correctamente. |

Figura 10. Tabla Caso de uso “Darse de Alta”.

**-Casos alternativos:** 3. La aplicación ve que alguno de los datos introducidos son erróneos, y avisa al usuario por pantalla de que compruebe los mismos.

* Caso de uso: RF2- **“Autenticarse”**

**- Objetivo:** *Loguear* a un usuario en la aplicación.

**- Actores:** Usuario de la aplicación.

**- Precondición:** El usuario debe estar registrado en la herramienta.

**- Descripción:** El usuario introduce su usuario y contraseña para poder acceder al sistema.

|  |  |
| --- | --- |
| USUARIO | SISTEMA |
| 1. El usuario una vez registrado en la app, introduce sus datos y pulsa el botón “LogIn” | 2. El sistema comprueba que esos datos son correctos y que corresponden con un *user* registrado anteriormente en la app. |
|  | 3. Una vez realizada dicha comprobación, el sistema redirige a la pantalla principal de la aplicación. |

Figura 11. Tabla Caso de uso “Autenticarse”.

**- Casos alternativos:** 2. La aplicación no encuentra coincidencias para los datos introducidos, con lo avisa al usuario por pantalla del error.

* Caso de uso: RF3- **“Ver perfil”**

**- Objetivo:** El usuario visualiza un perfil, ya sea suyo o de otra persona.

**- Actores:** Usuario de la aplicación.

**- Precondición:** El usuario debe *loguearse* con anterioridad para que la aplicación le redirija a su página.

**- Descripción:** El usuario podrá visualizar los datos en pantalla correspondientes al perfil propio o de otro usuario.

|  |  |
| --- | --- |
| USUARIO | SISTEMA |
| 1. El usuario autenticado es redirigido a la página principal de la aplicación. | 2. El sistema se encarga de mostrar los datos que se han obtenido. |
| 3. El usuario visualiza los datos mostrados. |  |

Figura 11. Tabla Caso de uso “Ver Perfil”.

**- Casos alternativos:** 2. La aplicación no puede obtener ningún dato respecto al usuario registrado con lo que no podrán visualizarse dicho perfil.

* Caso de uso: RF4- **“Buscar perfil”**

**- Objetivo:** El usuario busca a otra persona registrada en la aplicación para visualizar su perfil

**- Actores:** Usuario de la aplicación.

**- Precondición:** El usuario debe *loguearse* con anterioridad para que la aplicación le redirija a su página.

**- Descripción:** El usuario introduce en el buscador un nombre de otro *user* para que el sistema muestre su perfil.

|  |  |
| --- | --- |
| USUARIO | SISTEMA |
| 1. El usuario autenticado es redirigido a la página principal de la aplicación. |  |
| 2. El usuario introduce en el buscador el nombre de otro usuario. | 3. El sistema comprueba que dicho usuario existe. |
|  | 4. La herramienta redirige a la página principal del usuario buscado y muestra su perfil. |

Figura 12. Tabla Caso de uso “Buscar Perfil”.

**- Casos alternativos:** 3. La aplicación no puede obtener ningún usuario de la barra de búsqueda, porque los datos introducidos no coinciden con la BBDD. Se muestra un mensaje de error por pantalla.

* Caso de uso: RF5- **“Escribir comentario”**

**- Objetivo:** El usuario deja un comentario en el perfil de otro usuario.

**- Actores:** Usuario de la aplicación.

**- Precondición:** El usuario debe buscar en la aplicación a otro usuario para poder dirigirse a su perfil.

**- Descripción:** El usuario una vez redirigido al perfil ajeno, introduce un comentario en la caja de texto, y una vez terminado, pulsa “Enviar”.

|  |  |
| --- | --- |
| USUARIO | SISTEMA |
| 1. El usuario autenticado es redirigido a la página principal de la aplicación. |  |
| 2. El usuario introduce en el buscador el nombre de otro usuario. | 3. El sistema comprueba que dicho usuario existe. |
|  | 4. La herramienta redirige a la página principal del usuario buscado y muestra su perfil. |
|  | 5. El usuario introduce un comentario en la caja de texto que aparece a la derecha y la envía. |
|  | 6. Se actualiza el listado de comentarios y se muestran. |

Figura 13. Tabla Caso de uso “Escribir comentario”.

* Caso de uso: RF6- **“Volver a pantalla principal”**

**- Objetivo:** El usuario regresa a la pantalla principal donde se muestra su perfil.

**- Actores:** Usuario de la aplicación.

**- Precondición:** El usuario debe *loguearse* con anterioridad para que la aplicación le redirija a su página.

**- Descripción:** El usuario una vez autenticado en la aplicación, puede volver a su perfil principal pulsando el botón “HOME”. Puede hacerlo en cualquier parte de la aplicación.

|  |  |
| --- | --- |
| USUARIO | SISTEMA |
| 1. El usuario autenticado es redirigido a la página principal de la aplicación. | 2. El sistema se encarga de mostrar los datos que se han obtenido. |
| 3. El usuario presiona el botón “Home” y es redirigido a su perfil nuevamente. |  |

Figura 14. Tabla Caso de uso “Volver a pantalla principal”.

**- Casos alternativos:** 3. El usuario en lugar de presionar dicho botón, busca en la aplicación otros usuarios y se muestra su perfil. Vuelva al paso 3.

* Caso de uso: RF7- **“Cierre de sesión ”**

**- Objetivo:** El usuario cierra su sesión y vuelve a la pantalla de “*login/signup*”

**- Actores:** Usuario de la aplicación.

**- Precondición:** El usuario debe *loguearse* con anterioridad para que la aplicación le redirija a su página.

**- Descripción:** El usuario una vez autenticado en la aplicación, puede cerrar sesión donde se volverá a la pantalla de carga inicial. Vuelta al RF1.

|  |  |
| --- | --- |
| USUARIO | SISTEMA |
| 1. El usuario autenticado es redirigido a la página principal de la aplicación. | 2. El sistema se encarga de mostrar los datos que se han obtenido. |
| 3. El usuario presiona el botón “Home” y es redirigido a su perfil nuevamente. |  |

Figura 15. Tabla Caso de uso “Cierre de sesión”.

**3.3 Arquitectura de alto nivel.**

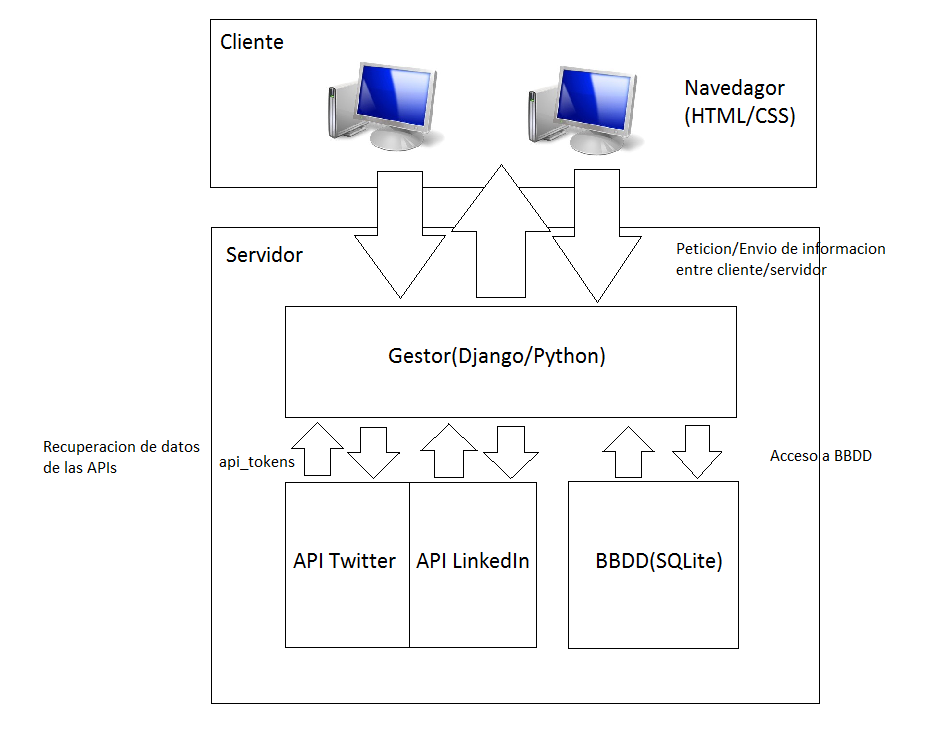


Figura 16. Diagrama alto nivel.

* Capa de cliente: En la que se encuentra nuestro navegador cliente, que envía y acepta peticiones HTTP a la capa servidor. Esta capa debe interpretar el código HTML y las hojas de estilos (CSS). Nuestros formularios y páginas se mostraran aquí.
* Capa servidor: Esta capa se encarga de recibir los datos de los formularios HTML para procesarlos y enviar peticiones a las Apis de Twitter y de LinkedIn. Por otra parte, se encarga de almacenar usuarios y sus comentarios en nuestra base de datos. Todo esto bajo la tutela de Django, y SQLite que es nuestro gestor de bases de datos.

Para definir mejor la arquitectura del sistema, la Figura 17 muestra el diagrama de alto nivel según los módulos de la aplicación.

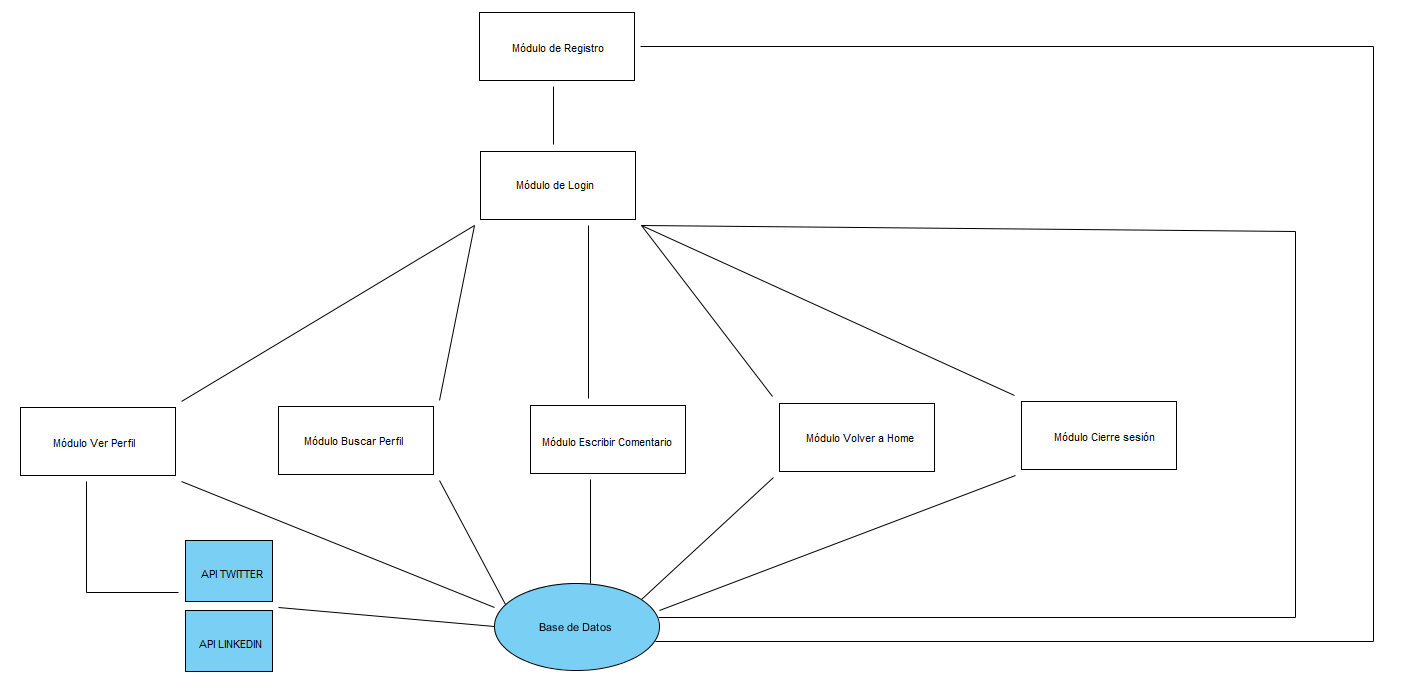


Figura 17. Diagrama alto nivel según módulos.

-Módulo “Registro”: módulo encargado de almacenar nuestro nuevo usuario en la base de datos.

-Módulo “Login”: módulo encargado de conectar al usuario previamente registrado, con nuestra aplicación, para posteriormente acceder con dichos datos a las apis.

-Módulo “Ver Perfil”: módulo encargado de conectar al usuario previamente registrado, con nuestra aplicación, para posteriormente acceder con dichos datos a las apis.

-Módulo “Buscar Perfil”: módulo encargado de buscar en nuestra BBDD coincidencias con el *user*, que metemos en el buscador, y en caso de encontrarlo, mostrar su perfil.

-Módulo “Escribir Comentario”: módulo encargado de capturar los comentarios de los usuarios y almacenarlos en la base de datos.

-Módulo “Volver a Home”: Módulo encargado de volver a la pantalla principal, en la cual se muestra el perfil del usuario *logueado* actualmente.

-Módulo “Cerrar Sesión”: Módulo encargado de cerrar la sesión del usuario activo para que otros usuarios no puedan acceder a su información.

Todos los módulos se conectan con la base de datos, ya que necesitan mostrar, consultar, y comprobar que los datos introducidos en los distintos formularios de la app, corresponden con datos.

**3.4 Diagrama E-R.**

Para comprender la estructura de la BBDD, mostraremos en la figura X, el diagrama entidad-relación correspondiente a la aplicación, y posteriormente explicaremos cada una de sus partes.

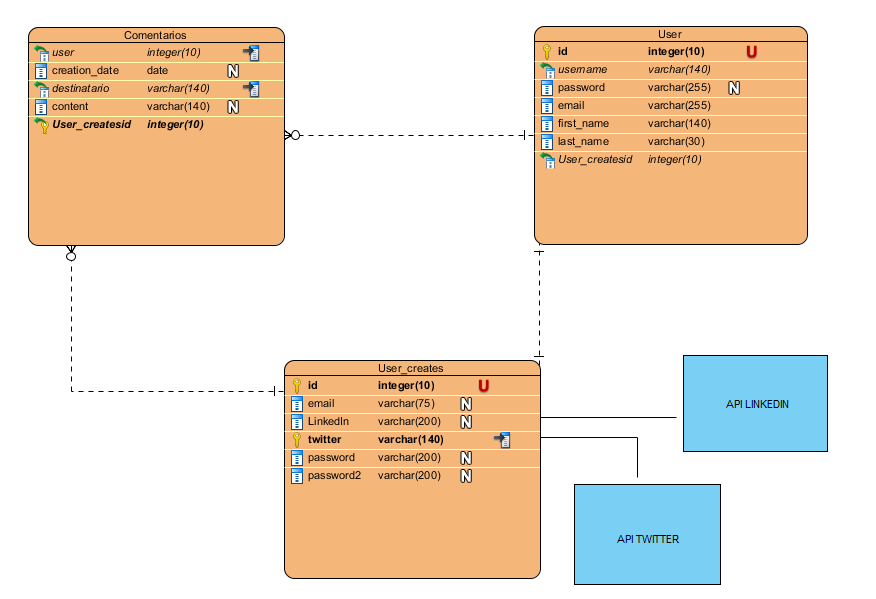


Figura 18. Diagrama ER.

En el modelo anterior podemos observar 3 entidades distintas. Por un lado, tenemos la entidad “*user*”, la cual proporciona Django. Se utiliza para facilitar la autenticación al sistema, para asociar contenido al propietario, para registros de perfiles… Sus principales atributos son los mostrados en la vista de su entidad.

Ya que Django nos proporciona esta funcionalidad, no la desperdiciaremos y usaremos esta tabla para registrar al usuario y manejarle por todo el sistema, sin tener que implementar funcionalidad adicional.

La entidad “*User\_Creates*” es una variante de la tabla anterior, en la cual añadimos los campos “LinkedIn” y “twitter” que serán los que usaremos para conectarnos a las apis. Cuando el usuario se registre por primera vez, crearemos una instancia para cada una de estas tablas. Por un lado tendremos un objeto “*user*” creado de esta manera:

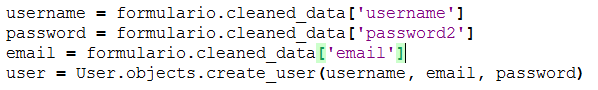


Figura 19. Código crear “user”.

Y por otro lado, la otra instancia se forma con los datos recuperados del formulario principal de registro:

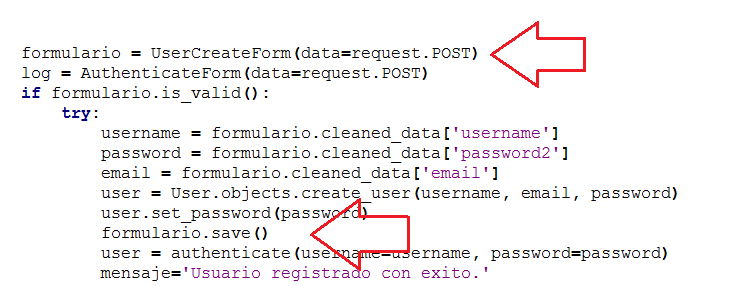


Figura 20. Código crear “User\_Create”.

Por otro lado tenemos la tabla “*comentarios*” en la cual un usuario podrá tener varios de estos, dirigidos hacia otros “*destinatarios*” que serán a su vez, otros usuarios.

Como se puede observar, es un modelo de base de datos sencillo, el cual solo se encarga de gestionar los usuarios, y sus comentarios hacia otros de los mismos. Esto se debe a que el acceso a la información no se almacena en nuestra BBDD, si no que se recupera de manera íntegra de las apis, para mostrarla directamente.

**Descripción y ejemplos de las tablas.**

Las siguientes capturas de pantalla están tomadas desde el sitio “Django/admin” reservado para *superusuarios*, en el que vemos la arquitectura de la base de datos.

* Tabla User.

Esta tabla es la sumistrada por Django, como hemos dicho anteriormente, en la que se pueden ver los campos “*Username*”, “*email*”, “*first\_name*” y “*last\_name*”. El campo “contraseña” se encuentra oculto, pero se puede ver entrando en cada uno de los usuarios.

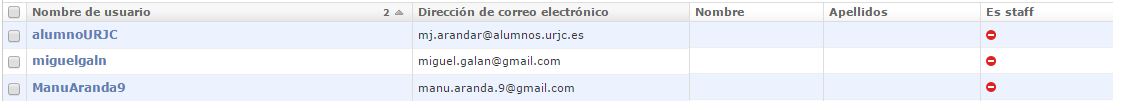


Figura 21. Vista de Admin/Django para “user”.

* Tabla User\_creates.

Esta tabla tiene nuestra versión personalizada del usuario del sistema, que accederá a las apis. Vemos como el campo “twitter” de esta tabla, coincide con el campo “*username*” de la tabla anterior.

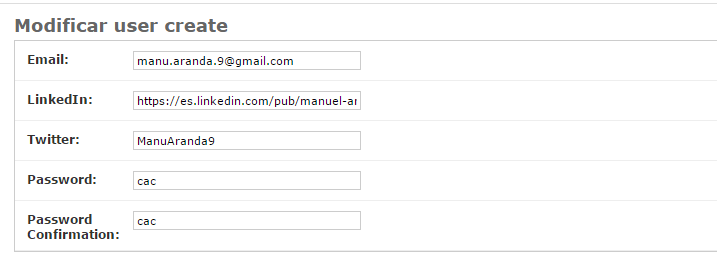


Figura 22. Vista de Admin/Django para “User\_create”.

* Tabla Comentario

Podemos observar en la figura X los campos que forman esta tabla. Anotar, que el campo “*creation\_date*” no aparece ya que es una propiedad que no se puede gestionar desde admin. En la figura +1 comprobamos que el campo User (*foreign\_key*), es un desplegable que referencia a todos los usuarios registrados en la aplicación.

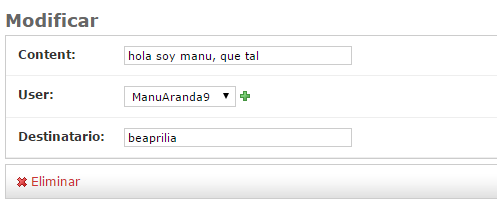


Figura 23. Vista de Admin/Django para “Comentario” (1).

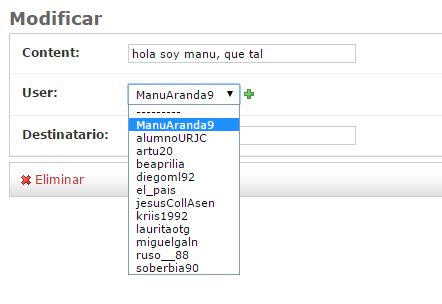


Figura 24. Vista de Admin/Django para “Comentario” (2).

**3.5 Algoritmo utilizado**

En este apartado realizaremos la descripción de los algoritmos utilizados para implementar cada uno de los requisitos funcionales de la aplicación.

*Mostrar datos en pantalla de autenticación;*

*Mostrar datos en pantalla de registro;*

*Si usuario quiere registrarse {*

*Usuario introduce datos en formulario de registro;*

*Si los datos introducidos son correctos {*

*Mostrar mensaje de éxito;*

*Guardar datos en la BD;*

*} Si no {*

*Mostrar mensaje de error;*

*}*

*Mostrar datos en pantalla de autenticación;*

*Mostrar datos en pantalla de registro;*

*}*

*//una vez el usuario se haya registrado, podrá logearse*

*Si usuario quiere loguearse {*

*Usuario introduce datos en formulario de login;*

*Si los datos introducidos coinciden en la BD con un usuario registrado {*

*mostrarPerfil (usuario);*

*} Si no {*

*Mostrar mensaje de error;*

*}*

*}*

*// Función que se encarga de mostrar un perfil, dado un usuario*

*Función mostrarPerfil (usuario) {*

*Redirección a pantalla principal;*

*Acceso al api de twitter;*

*Acceso al api de linkedin;*

*Se muestra la información de ese usuario en twitter;*

*Se muestra la información de ese usuario en linkedin;*

*Mostrar comentarios;*

*Si usuario!= usuarioLogueado{*

*Mostrar caja para introducir comentario;*

*}*

*Mostrar botón “volver a pantalla principal”;*

*Mostrar botón “cierre sesión”;*

*Mostrar buscador de usuarios;*

*Si usuario quiere volver a pantalla principal {*

*Pulsa botón Home;*

*Redirección a pantalla principal;*

*}*

*Si usuario quiere cerrar sesión {*

*Pulsa botón Cerrar Sesión;*

*Mostrar datos en pantalla de autenticación;*

*Mostrar datos en pantalla de registro;*

*}*

*Si usuario quiere buscar {*

*Introduce un usuario en el buscador;*

*Si usuario existe {*

*mostrarPerfil (usuario);*

*} Si no {*

*Mostrar mensaje de error;*

*}*

*}*

**3.6 Descripción de los módulos.**

En esta sección, se explican los principales módulos de los que se compone foundbep:

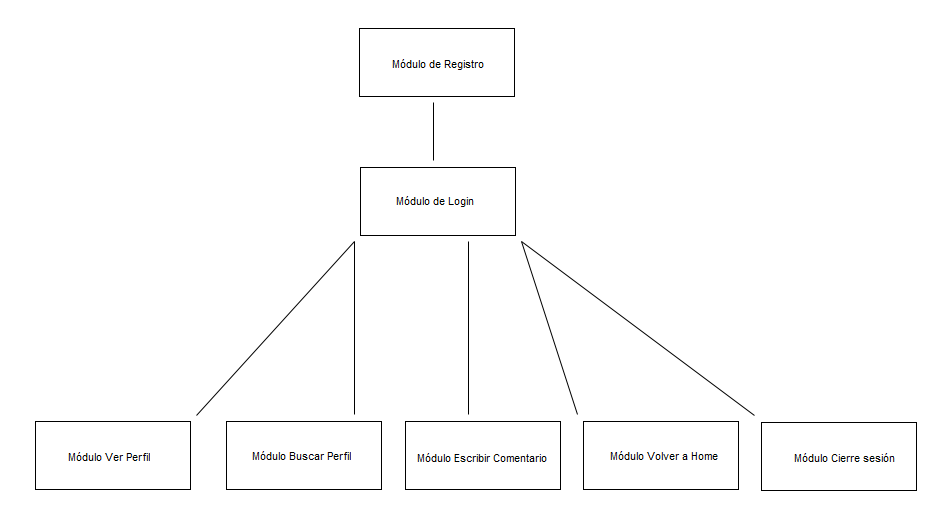


Figura 25. Diagrama de Módulos.

* Módulo de Registro.

En este módulo, el cual se encuentra dentro de **views.py,** se implementa dentro de la función **nuevo\_usuario.** Su funcionalidad reside en registrar al usuario en el sistema.El siguiente pseudocódigo muestra la estructura del módulo.

*Recuperar datos del formulario de registro;*

*Si los datos son válidos {*

*Intentar {*

*Crear objeto de la tabla “user”;*

*Crear objeto de la tabla “user\_create”;*

*Guardar datos en BD;*

*Mostrar mensaje de éxito;*

*} Excepción {*

*Mostrar mensaje de error;*

*}*

*}*

* Módulo Login.

En este módulo, el cual se encuentra dentro de **views.py,** se implementa dentro de la función **login\_view.** Su funcionalidad reside en autenticar al usuario en el sistema una vez registrado.El siguiente pseudocódigo muestra la estructura del módulo.

*Recuperar datos del formulario de autenticación;*

*Si los datos son válidos {*

*Intentar {*

*Acceso con “user” y “password”;*

*Recupera los comentarios que tiene ese “user”;*

*} Excepción {*

*Mostrar mensaje de error;*

*Acceso = error;*

*}*

*Si acceso = error {*

*Autenticación (acceso);*

*} Si no {*

*Mensaje de error;*

*}*

*}*

* Módulo Ver Perfil

En este módulo, el cual se encuentra dentro de **views.py,** se implementa dentro de las funciones **getTweets()** y **getBiografia().** Su funcionalidad es la de acceder a las distintas apis, y con los datos proporcionados, recuperar información para mostrarla.El siguiente pseudocódigo muestra la estructura del módulo.

*//función getTweets()*

*Intentar {*

*Importar Api Twitter;*

*Obtener datos de la api(nombre, descripción, timeline, …);*

*} Excepción {*

*Mostrar error;*

*}*

*//------------------------------------------------------------------*

*//función getBiografia()*

*Intentar {*

*Importar Api LinkedIn;*

*Obtener datos de la api (nombre, centro de estudios, CV, …);*

*} Excepción {*

*Mostrar error;*

*}*

* Módulo Buscar Perfil.

En este módulo, el cual se encuentra dentro de **views.py,** se implementa dentro de la función **buscar ().** Su funcionalidad reside en buscar en nuestra BD coincidencias con el usuario introducido por el buscador.El siguiente pseudocódigo muestra la estructura del módulo.

*Recuperar datos del buscador;*

*Si los datos son válidos {*

*Recupera los comentarios que tiene ese “user”;*

*Para “user” recupera el LinkedIn asociado en la tabla “User\_Create”;*

*getTweets();*

*getBiografía();*

*}*

* Módulo Escribir Comentario.

En este módulo, el cual se encuentra dentro de **views.py,** se implementa dentro de la función **submit ().** Su funcionalidad reside en escribir un comentario y guardarlo en la BD con la información de quien lo escribe, y para quien va destinado.El siguiente pseudocódigo muestra la estructura del módulo.

*Recuperar datos del formulario;*

*Si los datos son válidos {*

*Recupera el usuario que envía dicho comentario;*

*Recupera el usuario al que va destinado el comentario;*

*Guarda toda la información en una objeto instanciado de la tabla “Comentario”*

*}*

* Módulo volver a “Home”

En este módulo, el cual se encuentra dentro de **views.py,** se implementa dentro de la función **cargarPantalla ().** Su funcionalidad reside en cargar la información del usuario autenticado en la app y mostrar su perfil.El siguiente pseudocódigo muestra la estructura del módulo.

*Si usuario quiere regresar a la pantalla principal {*

*Recupera información del usuario actualmente logueado en la app;*

*getTweets ();*

*getBiografía ();*

*}*

* Módulo Cerrar Sesión.

En este módulo, el cual se encuentra dentro de **views.py,** se implementa dentro de la función **logout\_view ().** Su funcionalidad reside en cerrar la sesión actual y volver a la pantalla de login/registro.El siguiente pseudocódigo muestra la estructura del módulo.

*Si usuario quiere cerrar sesión {*

*Logout();*

*Re direccionar a página inicial;*

*}*

Capítulo 4

**PRUEBAS**

Para comprobar el correcto funcionamiento de la aplicación se han realizado distintos tipos de pruebas que se describen a continuación.

**4.1 Pruebas Unitarias**

Con este tipo de pruebas se comprueba el correcto funcionamiento de cada módulo de forma independiente. Se distingue entre pruebas de **Caja Blanca** que son aquellas que verifican el funcionamiento interno del sistema. Las pruebas se planifican observando la estructura del algoritmo del sistema, sus condicionales, bucles, casos especiales… En las pruebas de **Caja Negra** no se tiene en cuenta el funcionamiento interno del sistema, solo se analizan entradas y salidas.

**4.1.1 Pruebas de Caja Blanca**

Como se ha comentado, el estudio se realiza a partir de los módulos de la aplicación, por lo que dividiremos este apartado según los módulos definidos en apartados anteriores.

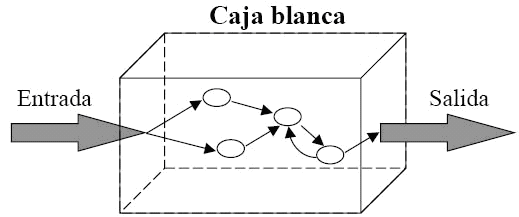
****

Figura X. Prueba de Caja Blanca.

En estas pruebas se mostrará el pseudocódigo de cada módulo seguido del diagrama que muestra los caminos del módulo. Cada círculo representado en el diagrama contiene un número que coincide con la línea del pseudocódigo a la que nos referimos.

A continuación, se mostrará la complejidad de la prueba (número de caminos independientes) y los caminos.

Una vez obtenidos los caminos, se mostrarán entradas de la prueba, así como las salidas de la prueba y la salida esperada (la que debería haber dado la prueba). Si ambas coindicen, el módulo funcionará correctamente.

**Módulo “Registro”**

1. *Recuperar datos del formulario de registro;*

2. *Si los datos son válidos {*

3. *Intentar {*

4. *Crear objeto de la tabla “user”;*

5. *Crear objeto de la tabla “user\_create”;*

6. *Guardar datos en BD;*

7. *Mostrar mensaje de éxito;*

8. *} Excepción {*

9. *Mostrar mensaje de error;*

10.  *}*

11. *}*

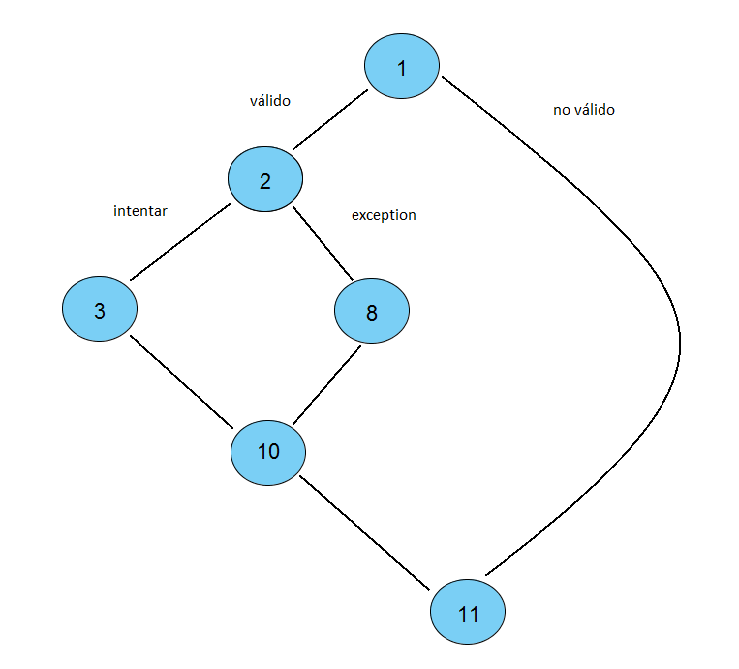


Figura X. Prueba Caja blanca caso de uso Registro

Complejidad = 3

Caminos:

* Camino 1: 1, 2, 3, 10, 11
* Camino 2 :1, 2, 8, 10, 11
* Camino 3: 1, 11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Valido | Intentar | Salida prueba | Salida esperada |
| Camino 1 | Si | Si | -Guardar datos en BBDD  -Mensaje éxito | -Guardar datos en BBDD  -Mensaje éxito |
| Camino 2 | Si | No | -Mostrar mensaje de error | -Mostrar mensaje de error |
| Camino 3 | No | - | -Seguir en pantalla | -Seguir en pantalla |

Figura x. Resultados de prueba caja blanca caso de uso Registro.

**Módulo “Autenticación”**

1. *Recuperar datos del formulario de autenticación;*

2. *Si los datos son válidos {*

3. *Intentar {*

4. *Acceso con “user” y “password”;*

5. *Recupera los comentarios que tiene ese “user”;*

6. *} Excepción {*

7. *Mostrar mensaje de error;*

8. *Acceso = error;*

9.  *}*

10*. Si acceso!= error{*

11*. Autenticación (acceso);*

12*. } Si no {*

13*. Mensaje de error;*

14*. }*

15. *}*

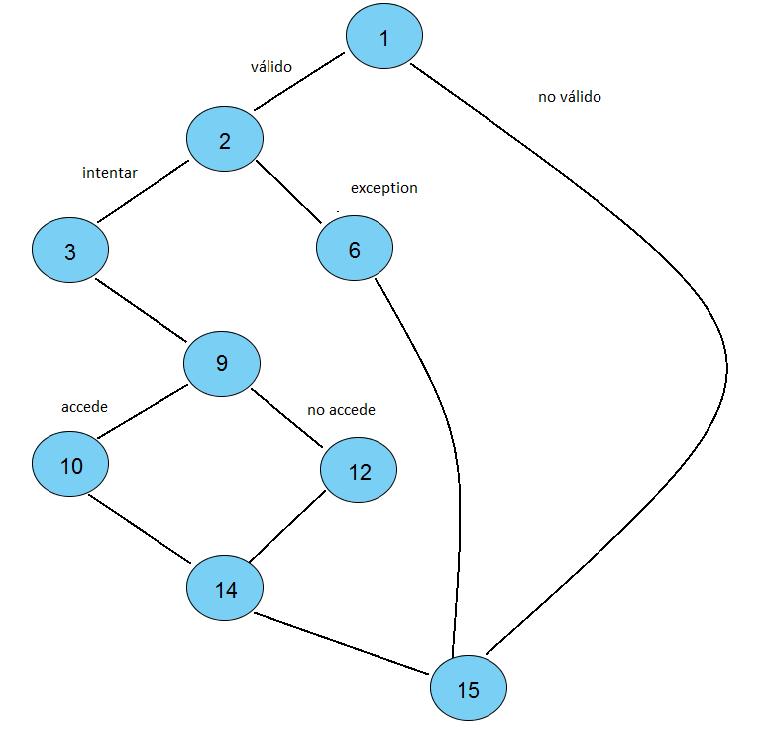
**

Figura X. Prueba Caja blanca caso de uso Autenticación

Complejidad = 4

Caminos:

* Camino 1: 1, 2, 3, 9, 10, 14, 15
* Camino 2: 1, 2, 3, 9, 12, 14, 15
* Camino 3: 1, 2, 6, 16
* Camino 4: 1, 15

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Valido | Intentar | Accede | Salida prueba | Salida esperada |
| Camino 1 | Si | Si | Si | -Recuperar comentarios del “user”  -Autenticación(acceso) | -Recuperar comentarios del “user”  -Autenticación(acceso) |
| Camino 2 | Si | Si | No | -Recuperar comentarios del “user”  -Mensaje de error | -Recuperar comentarios del “user”  -Mensaje de error |
| Camino 3 | Si | No | - | -Mensaje de error | -Mensaje de error |
| Camino 4 | No | - | - | -Mensaje de error | -Mensaje de error |

Figura x. Resultados de prueba caja blanca caso de uso Registro.

**Módulo “Ver Perfil”**

*//función getTweets ()*

1. *Intentar {*

2. *Importar Api Twitter;*

3. *Obtener datos de la api (nombre, descripción, timeline,…);*

4. *} Excepción {*

5. *Mostrar error;*

6. *}*

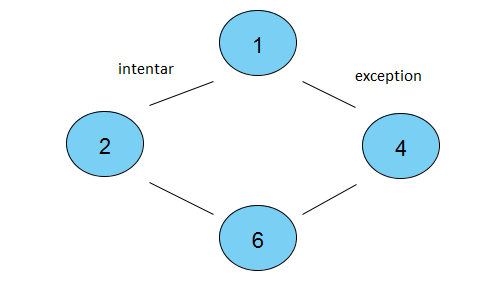
**

Figura X. Prueba Caja blanca caso de uso Ver Perfil (1)

Complejidad = 2

Caminos:

* Camino 1: 1, 2, 6
* Camino 2: 1, 4, 6

|  |  |  |  |
| --- | --- | --- | --- |
|  | Intentar | Salida prueba | Salida esperada |
| Camino 1 | Si | -Importar API Twitter  -Obtener Datos Api | -Importar API Twitter  -Obtener Datos Api |
| Camino 2 | No | -Mensaje de error | -Mensaje de error |

Figura x. Resultados de prueba caja blanca caso de uso Ver perfil (1).

*//función getBiografía ()*

1. *Intentar {*

2. *Importar Api LinkedIn;*

3. *Obtener datos de la api (nombre, centro de estudios, CV,…);*

4. *} Excepción {*

5. *Mostrar error;*

6. *}*

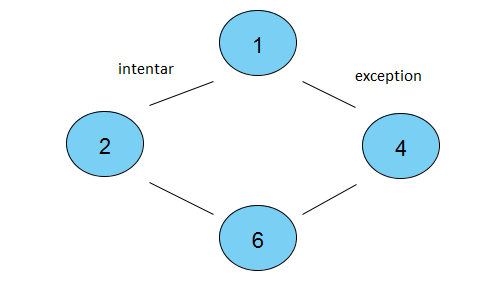


Figura X. Prueba Caja blanca caso de uso Ver Perfil (2)

Complejidad = 2

Caminos:

* Camino 1: 1, 2, 6
* Camino 2: 1, 4, 6

|  |  |  |  |
| --- | --- | --- | --- |
|  | Intentar | Salida prueba | Salida esperada |
| Camino 1 | Si | -Importar API LinkedIn  -Obtener Datos Api | -Importar API LinkedIn  -Obtener Datos Api |
| Camino 2 | No | -Mensaje de error | -Mensaje de error |

Figura x. Resultados de prueba caja blanca caso de uso Ver perfil (2).

**Módulo “Buscar Perfil”**

1. *Recuperar datos del buscador;*

2. *Si los datos son válidos {*

3. *Recupera los comentarios que tiene ese “user”;*

4. *Para “user” recupera el LinkedIn asociado en la tabla “User\_Create”;*

5. *getTweets ();*

6. *getBiografía ();*

7. *}*

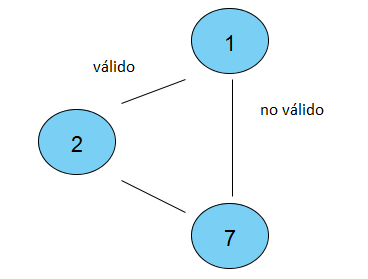
****

Figura X. Prueba Caja blanca caso de uso Buscar Perfil

Complejidad = 2

Caminos:

* Camino 1: 1, 2, 7
* Camino 2: 1, 7

|  |  |  |  |
| --- | --- | --- | --- |
|  | Válido | Salida prueba | Salida esperada |
| Camino 1 | Si | -Recuperar datos del user  -GetTweets, GetBiografia | -Recuperar datos del user  -GetTweets, GetBiografia |
| Camino 2 | No | -Seguir en pantalla | -Seguir en pantalla |

Figura x. Resultados de prueba caja blanca caso de uso Buscar Perfil.

**Módulo “Escribir Comentario”**

1. *Recuperar datos del formulario;*

2. *Si los datos son válidos {*

3. *Recupera el usuario que envía dicho comentario;*

4. *Recupera el usuario al que va destinado el comentario;*

5. *Guarda toda la información en una objeto instanciado de la tabla “Comentario”*

6. *}*

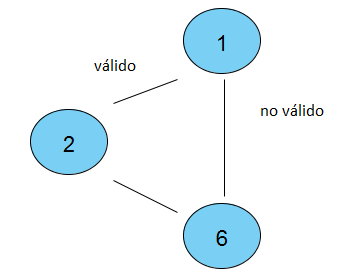


Figura X. Prueba Caja blanca caso de uso Escribir Comentario

Complejidad = 2

Caminos:

* Camino 1: 1, 2, 6
* Camino 2: 1, 6

|  |  |  |  |
| --- | --- | --- | --- |
|  | Válido | Salida prueba | Salida esperada |
| Camino 1 | Si | -Recuperar user  -Recuperar destinatario  -Almacenar comentario en BD | -Recuperar user  -Recuperar destinatario  -Almacenar comentario en BD |
| Camino 2 | No | -Seguir en pantalla | -Seguir en pantalla |

Figura x. Resultados de prueba caja blanca caso de uso Escribir Comentario.

**Módulo “Volver a Home (pantalla principal)”**

1. *Si usuario quiere regresar a la pantalla principal {*

2.  *Recupera información del usuario actualmente logueado en la app;*

3. *getTweets ();*

4. *getBiografía ();*

5. *}*

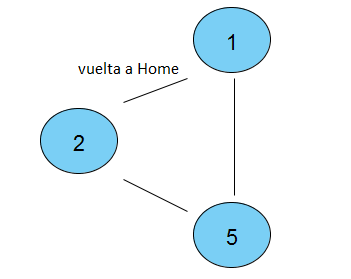


Figura X. Prueba Caja blanca caso de uso Volver a pantalla Principal

Complejidad = 2

Caminos:

* Camino 1: 1, 2, 5
* Camino 2: 1, 5

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vuelta Atrás | Salida prueba | Salida esperada |
| Camino 1 | Si | -Recuperar datos del user logueado  -GetTweets, GetBiografia | -Recuperar datos del user logueado  -GetTweets, GetBiografia |
| Camino 2 | No | -Seguir en pantalla | -Seguir en pantalla |

Figura x. Resultados de prueba caja blanca caso de uso Volver a pantalla Principal.

**Módulo “Cierre sesión”**

1. *Si usuario quiere cerrar sesión {*

2.  *Logout ();*

3. *Re direccionar a la pantalla inicial;*

4. *}*

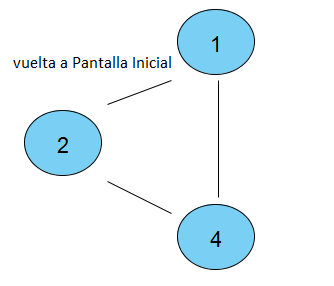


Figura X. Prueba Caja blanca caso de uso Volver a pantalla Inicial

Complejidad = 2

Caminos:

* Camino 1: 1, 2, 4
* Camino 2: 1, 4

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vuelta Atrás | Salida prueba | Salida esperada |
| Camino 1 | Si | -LogOut;  -Pantalla Inicial; | -LogOut;  -Pantalla Inicial; |
| Camino 2 | No | -Seguir en pantalla | -Seguir en pantalla |

Figura x. Resultados de prueba caja blanca caso de uso Volver a pantalla Principal.

**Conclusiones**

**4.1.2 Pruebas de Caja Negra**

En el método de la **Caja Negra** se decide no tener en cuenta el funcionamiento interno de un sistema y solo se analizan sus entradas y salidas. Se aplica tanto como estrategia de testeo, fijándose más en el exterior (usuario) o en la conexión entre diferentes sistemas (interfaz), que como necesidad cuando no es accesible o no es practico estudiar el funcionamiento interno del sistema en análisis.

Para Caja Negra o pruebas de caja opaca se requiere menos habilidad técnica, menos tiempo y menos herramientas. Por ende, menos costo. Pero solo te permite detectar errores y fallos pero no te acerca a la solución de éstos.

Para realizar estas pruebas se elegirá la técnica de partición en clases de equivalencia, que consiste en identificar las condiciones de entrada de los módulos y obtener sus reglas, así como las clases válidas e inválidas obtenidas mediante dichas reglas. Una vez obtenidas estas clases, se buscan los casos de pruebas correspondientes.

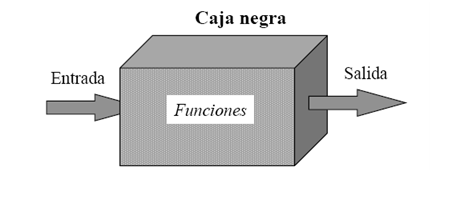


Figura x Caja Negra

**“home.html”**

**-** Registro de la aplicación

|  |  |  |  |
| --- | --- | --- | --- |
| Entrada | Reglas a seguir | Clases Válidas | Clases No Válidas |
| E-Mail | -alfanumérico@alfanumérico.com  - No vacío | **1**. alfanumérico@alfanumérico.com que tenga menos de 200 caracteres y que sea No vacío. | **2.** noalfanumérico@noalfanumérico.com  **3.** “” (vacío) |
| LinkedIn | - No vacío  - No acentos | **4.** Que no sea vacío y que no tenga acentos en la URL. | **5.** Vacío  **6.** Contenga acentos. |
| Twitter | - No vacío | **7.** Que no sea vacío. | **8.** Vacío |
| Contraseña | - <200  - No vacío | **9.** Con menos de 200 caracteres y que no sea vacío. | **10.** >200 caracteres  **11.** Vacío |
| Repetir contraseña | - <200  - No vacío | **12.** Con menos de 200 caracteres y que no sea vacío. | **13.** >200 caracteres  **14.** Vacío |

Figura X. Clases de entrada para “registro”

**NOTA:** Para crear a nuestro nuevo usuario, tanto el campo “Twitter” como “LinkedIn” dependen directamente tanto del *user* de Twitter que tenga el usuario, como de la *url* de su perfil público, que posea. Es decir, que en dichos campos, solo podemos controlar que no sea vacío, ya que las demás reglas, son implícitas a la hora de escribir sobre los campos. Cuando el usuario, introduzca alguno de los campos, siguiendo las clases correctamente, pero no coincidan con usuarios creados en las 2 apis, no dará error el cual se pueda observar en pantalla, pero no se mostrara información alguna, lógicamente.

Caso válido:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E-mail | LinkedIn | Twitter | Contraseña | Repetir contraseña | Clases |
| Manu.aranda.9@gmail.com | https://es.linkedin.com/pub/manuel-aranda/b0/436/689 | ManuAranda9 | 1234 | 1234 | 1,4,7,9,12 |

Figura x. Clase válida para “registro”

Casos no válidos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E-mail | LinkedIn | Twitter | Contraseña | Repetir contraseña | Clases |
| ….@------ | https://es.linkedin.com/pub/manuel-aranda/b0/436/689 | ManuAranda9 | 1234 | 1234 | 2 |
| (vacío) | https://es.linkedin.com/pub/manuel-aranda/b0/436/689 | ManuAranda9 | 1234 | 1234 | 3 |
| Manu.aranda.9@gmail.com | (vacío) | ManuAranda9 | 1234 | 1234 | 5 |
| Manu.aranda.9@gmail.com | http://es.linkedin.com/pub/mánuel-árándá/b0/436/689 (acento en la a) | ManuAranda9 | 1234 | 1234 | 6 |
| Manu.aranda.9@gmail.com | https://es.linkedin.com/pub/manuel-aranda/b0/436/689 | (vacío) | 1234 | 1234 | 8 |
| Manu.aranda.9@gmail.com | https://es.linkedin.com/pub/manuel-aranda/b0/436/689 | ManuAranda9 | 12345…(>200) | 1234 | 10 |
| Manu.aranda.9@gmail.com | https://es.linkedin.com/pub/manuel-aranda/b0/436/689 | ManuAranda9 | (vacío) | 1234 | 11 |
| Manu.aranda.9@gmail.com | https://es.linkedin.com/pub/manuel-aranda/b0/436/689 | ManuAranda9 | 1234 | 12345…(>200) | 13 |
| Manu.aranda.9@gmail.com | https://es.linkedin.com/pub/manuel-aranda/b0/436/689 | ManuAranda9 | 1234 | (vacío) | 14 |

Figura x. Clases no válidas para “registro”

**-** Autenticación en la aplicación

|  |  |  |  |
| --- | --- | --- | --- |
| Entrada | Reglas a seguir | Clases Válidas | Clases No Válidas |
| Twitter | - No vacío | **1.** Que no sea vacío. | **2.** Vacío |
| Contraseña | - <200  - No vacío | **3.** Con menos de 200 caracteres y que no sea vacío. | **4.** >200 caracteres  **5.** Vacío |

Figura x. Clases válidas para “autenticarse”

Caso válido:

|  |  |  |
| --- | --- | --- |
| Twitter | Contraseña | Clases |
| ManuAranda9 | 1234 | 1,3 |

Figura x. Clase válida para “registro”

Casos no válidos:

|  |  |  |
| --- | --- | --- |
| Twitter | Contraseña | Clases |
| (Vacío) | 1234 | 2 |
| ManuAranda9 | 1234…(>200) | 4 |
| ManuAranda9 | (Vacío) | 5 |

Figura x. Clase válida para “registro”

Para este módulo, de tan solo 2 casillas, además de comprobar que nuestros datos cumplen las clases, el sistema comprueba que existen en el sistema un usuario que corresponda con el introducido, y que además contenga esa contraseña. En caso negativo, aunque cumplan las clases anteriormente mencionadas, se mostrará en pantalla un error correspondiente a fallo en los datos introducidos.

**“inicio.html”**

**-** Búsqueda de perfil

|  |  |  |  |
| --- | --- | --- | --- |
| Entrada | Reglas a seguir | Clases Válidas | Clases No Válidas |
| User/Twitter | - No vacío | **1**. No vacío. | **2.** Vacío |

Figura X. Clases de entrada para “Buscar Perfil”

Caso válido:

|  |  |
| --- | --- |
| User/Twitter | Clases |
| ManuAranda9 | 1 |

Figura x. Clase válida para “registro”

Casos no válidos:

|  |  |
| --- | --- |
| User/Twitter | Clases |
| (Vacío) | 2 |

Figura x. Clase válida para “registro”

El funcionamiento de este módulo es similar al de “autenticarse” ya que los datos introducidos en el input, serán comprobados internamente en el sistema para ver si hay coincidencias de usuarios para mostrar. Tan solo podemos “controlar” que dicho campo no este vacío cuando se pulse el botón de “Buscar”.