Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Nombre del Centro

Trabajo Fin de Grado

HERRAMIENTA WEB PARA LA VISUALIZACIÓN 3D DE REDES CEREBRALES

Alumno: Javier Moral Lara

Tutor: Prof. D. Juan Ruiz de Miras
Dpto: Lenguajes y Sistemas Informáticos

Febrero, 2017

Universidad de Jaén

Escuela Politécnica Superior de Jaén
Departamento de Informática

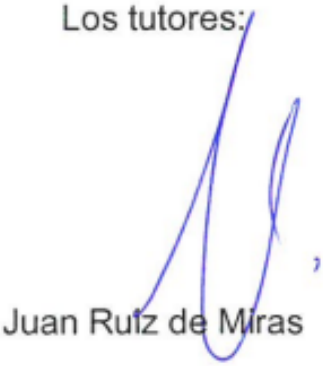
Don Juan Ruiz de Miras , tutor del Proyecto Fin de Carrera titulado: Herramienta web para la visualización 3D de redes cerebrales, que presenta Javier Moral Lara, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, febrero de 2017

El alumno:

Javier Moral Lara

Los tutores:



Juan Ruiz de Miras

Índice

1. Introducción

A continuación, se presenta una introducción a este trabajo de fin de grado, repasando de manera breve y concisa los objetivos y funciones genéricas del mismo, así como la motivación detrás del desarrollo.

Cada uno de los apartados irá acompañado de un epígrafe como este, sirviendo de breve introducción al tema expuesto y reflejando algunas reflexiones personales del autor.

Objetivos y funciones genéricas

Se busca implementar una herramienta que permita la fácil visualización de las estructuras conocidas como “brain network” o redes cerebrales. Dichas estructuras son un modelo de la interconexión de las neuronas entre distintas áreas del cerebro.

La aplicación debe presentar una interfaz sencilla e intuitiva. Nuestro principal usuario será una persona con conocimientos a nivel de usuario en el ámbito de las ciencias de la computación en general y de la informática gráfica en particular.

La naturaleza espacial de este tipo de datos los hace muy susceptibles de ser representados mediante técnicas 3D. Nuestra herramienta debe permitir una sencilla navegación a través de los datos. Mostrando un modelo del cerebro como referencia, representará las distintas áreas del mismo y las conexiones entre ellas.

En el apartado de representación, debemos poder configurar parámetros de la misma. Color, transparencia, sombreado del dibujo deben ser opciones que el usuario puede personalizar para hacer más cómoda su exploración de los datos.

Con respeto a las áreas a representar, la parcelación del cerebro no se da en términos exactos. El cerebro es un órgano y el desarrollo del mismo depende en los detalles del propio paciente, aun así, debemos proporcionar al usuario la posibilidad de elegir entre varias parcelaciones estándar.

A grandes rasgos, la función principal de nuestro software será proporcionar una herramienta lo más accesible posible para la visualización de las redes cerebrales. Accesible en cuanto a la plataforma, pero también accesible en cuanto al nivel de conocimiento del usuario.

En cuanto a la plataforma porque no queremos atarnos a ninguna en particular. El desarrollo para determinadas plataformas móviles va a acompañado de una inversión previa (licencias, permisos de publicación) a veces de manera absoluta como Android o mediante un modelo de subscripción como iOS y las posibilidades de monetización de esta herramienta son limitadas. En cuanto al nivel de conocimiento del usuario porque tenemos claro que el usuario base no tiene conocimientos sobre informática gráfica, ni si quiere tiene por qué tenerlos de informática general, pero creemos que puede ser una herramienta interesante para aplicaciones médicas y de enseñanza.

Si bien uno de los objetivos de este trabajo es el de explorar y trabajar las tecnologías web, más adelante se verá que no se trata de una decisión arbitraria y que hay una serie de razones válidas para la elección de las mismas.

Exploraremos los distintos estándares en el desarrollo web, como HTML5 o las últimas versiones el estándar ECMAScript que da lugar al lenguaje Javascript. Estudiaremos el estado del arte en Javascript, un ecosistema que en los últimos años ha vivido la irrupción de multitud de herramientas con su metodología correspondiente. Como resultado, demostraremos que es posible desarrollar una herramienta con unos requisitos en principio poco convencionales (aplicación gráfica 3D) usando solamente herramientas y tecnologías web.

Motivación

Las mejoras en el ámbito científico y en particular las aplicadas a la medicina han permitido el desarrollo de técnicas y herramientas para una compresión más profunda de un órgano complejo como el cerebro (mayores resoluciones en IRM, electroencefalografía más sensible, etc.). A medida que aumenta la cantidad de información que conseguimos recolectar sobre el cerebro, necesitamos que las técnicas de visualización de las mismas avancen a la par que las técnicas de recolección. Sin estas técnicas de visualización, el análisis, interpretación y emisión de un diagnóstico certero usando estos datos puede ser complejo y costoso, tanto en tiempo como en recursos humanos.

Buscamos desarrollar una herramienta que, sin llegar a ejercer un rol crítico en el diagnóstico de un paciente, sea capaz de ofrecer un análisis preliminar certero y que sirva para trazar un plan de ataque más profundo para el equipo médico que la emplee.

Además, al presentar los datos de una manera visual, intuitiva e interactiva, puede tener finalidades educativas, siendo útil para ayudar a estudiantes del área de la biomedicina a entender los mecanismos del cerebro.

Resultados esperados

El resultado esperado es el de llevar a buen término la implementación y despliegue de la aplicación descrita en el proyecto, proporcionando una herramienta usable para el usuario final.

Buscamos también que el desarrollo y despliegue de toda la aplicación se realice a coste cero, aprovechando para tal fin las herramientas y plataformas que ofrezcan servicios gratuitos.

Si bien en el análisis y elaboración de la documentación del proyecto se propone un sistema con características secundarias, como, por ejemplo, la capacidad del sistema para mantener información del estado de la navegación entre sesiones, el objetivo principal es el de implementar una herramienta de visualización en 3D en el navegador que permita mostrar los datos de las redes cerebrales. En caso de presentarse restricciones temporales, nos centraremos en la implementación de dichos requisitos principales y se diseñarán sin llegarse a implementar aquellos requisitos secundarios o que no estén tan focalizados en el ámbito de la informática gráfica, campo principal de este proyecto.

Estructura del proyecto

Sirva el índice como principal imagen de estructura del proyecto y los epígrafes presentados en cada capítulo y subcapítulo para obtener una idea más en profundidad de a qué se dedica cada apartado, se puede hacer una categorización general del proyecto.

Este primer apartado será una introducción al mismo.

El segundo explorará los requisitos de software. Será el resultado de las primeras etapas en un proyecto, entrevistas con usuarios, análisis de documentación previa y elaboración de un dosier sobre las distintas posibilidades de implementación del mismo.

En el tercero se mostrará la planificación para la ejecución. Será el apartado más de perfil de gestor, mostrando planificación temporal, de recursos tanto materiales y humanos. Como

resultado presentaremos una estimación de costos y un presupuesto para cubrir gastos de producción.

El cuarto mostrará el diseño del software. Será el resultado del análisis desde la perspectiva de la ingeniería del software. Proporcionaremos una visión de arquitectura de componentes lógicos para dar solución al problema planteado.

El quinto entrará en los detalles propios de la implementación. Será la parte más técnica de la documentación, entrando a discutir aspectos y decisiones tanto de la implementación como de software elegido.

El sexto recopilará las experiencias de las pruebas. Se hará un repaso por las pruebas de tipo software ejecutadas, principalmente para la detección y corrección de errores, pero también por las pruebas de validación de los requisitos funcionales.

El séptimo incluirá un breve manual de usuario, para facilitar al usuario el uso de la aplicación.

El octavo proporcionará un manual de instalación y despliegue de la aplicación.

En el noveno y último punto se presentarán las conclusiones de índole técnico, así como a nivel personal del encargado del proyecto.

2. Requisitos del software

Descripción funcional

Para la descripción del sistema software usaremos requisitos. Estos vendrán aunados en dos grandes grupos.

- Requisitos funcionales
- Requisitos no funcionales

Los requisitos funcionales son una descripción objetiva de las tareas que se espera que el usuario sea capaz de realizar con el software, su funcionalidad.

Los requisitos no funcionales son una medida subjetiva del grado de requerimiento de los requisitos funcionales.

Requisitos Funcionales

1. **Generales del sistema.**

   RF1.1 El sistema debe ser ejecutable desde los dispositivos y plataformas con mayor aceptación de mercado.

   RF1.2 El sistema debe poder trabajar con distintos perfiles de usuarios.

2. **Sobre el usuario**

   RF2.1 El usuario debe poder identificarse en el sistema con un par identificador-contraseña.

   RF2.2 El usuario debe poder configurar los datos de autenticación de su perfil.

   RF2.3 El usuario tendrá disponible un listado de redes cerebrales que podrá consultar, independientemente del dispositivo donde acceda.

   RF2.4 El usuario podrá crear nuevas redes a partir de ficheros estándar de representación de redes cerebrales.

   RF2.5 El usuario podrá eliminar redes cerebrales del sistema.

   RF2.6 El usuario podrá establecer cuáles de sus redes son públicas, de manera que otros usuarios del sistema puedan visualizarlas.

3. **Sobre representación de redes cerebrales**

   RF3.1 Se mostrará como guía un modelo del cerebro humano, para ayudar a situar los distintos elementos.

   RF3.2 Las conexiones entre las distintas áreas aparecerán con una representación explícita, mostrando además información concisa sobre ellas.

   RF 3.3 Las distintas áreas del cerebro o parcelación debe de ser explícita.

RF3.4 Se podrá elegir entre una representación más realista del cerebro o una más funcional para la exploración de datos.

RF3.5 El usuario podrá seleccionar y ajustar características estéticas de las representaciones explícitas de las conexiones (color, tamaño, tipo de sombreado).

RF3.6 El usuario podrá filtrar las distintas conexiones entre áreas en función del peso de las mismas, estableciendo un valor mínimo por debajo del cual, las conexiones no se mostrarán.

RF3.7 El usuario podrá explorar los datos desde distintas posiciones y puntos de vista.

RF3.8 La configuración de la representación del usuario debe poder estar disponible entre distintas sesiones de trabajo.

Requisitos No Funcionales

La navegación a través de los datos se debe realizar de una manera intuitiva para nuestro perfil de usuario. Dado que la herramienta tiene fines de exploración de datos y no de su procesamiento, la curva de aprendizaje de la misma se debe de mantener al mínimo, para que la ratio esfuerzo / recompensa justifique su uso.

El tiempo de respuesta de la aplicación deberá ser lo más ajustado posible, tanto en la carga como en la visualización de los datos.

El perfil de usuario no tiene conocimientos sobre informática gráfica, por lo que, en la medida de lo posible, debemos evitar el uso de términos técnicos asociados a dicho campo. Usaremos sinónimos cuando sea posible o metáforas

Perfil de los usuarios

Se presentan los dos tipos de usuario que presentará esta aplicación.

Usuario sin registrar: No requiere de autorización y por tanto de identificación. Podrá consultar aquellas redes cerebrales que hayan sido marcadas como públicas por sus dueños.

Usuario registrado: Requiere de identificación. Podrá consultar las redes cerebrales marcadas como públicas por el resto de usuarios. Además, podrá consultar todas aquellas redes de las que él sea el autor, así como modificarlas. Podrá además crear nuevas redes.

Estudio de posibles soluciones

Uno de los objetivos principales del proyecto es desarrollar el software manteniendo al mínimo el coste. Exploraremos solamente alternativas de coste cero, ya sean freeware o preferentemente con licencias de tipo abierto. Las herramientas que supongan un desembolso económico se intentarán evitar en la medida de lo posible. Fuera de esta evaluación se dejan alternativas válidas y de peso en la industria (como puede ser el caso de Oracle o Microsoft SQL) pero el coste de las mismas los hace prohibitivamente caro para un proyecto de bajo coste como el que aquí se expone. Queda exento de esta máxima el apartado de despliegue e infraestructura, ya que depende de las capacidades y del crecimiento del proyecto, este coste será ineludible. Haremos sin embargo una consideración especial a los servicios que ofrecen de forma gratuita.

Plataforma

Observando los requisitos funcionales RF2.3, que indica que el listado de redes debe estar presente al usuario independientemente del dispositivo y el requisito funcional RF3.6, el cual nos indica que los parámetros de visualización deben estar disponibles entre sesiones de trabajo, se hace evidente que no podemos construir una aplicación aislada. Nuestra aplicación debe de poder tener acceso a datos a través de la red para recuperar la información de las sesiones anteriores. La manera más sencilla de implementar este comportamiento es mediante un patrón cliente-servidor, en el que nuestro servidor actuará como almacén centralizado de información.

Como contamos con un presupuesto y un tiempo limitado, descartamos la posibilidad de lanzar varias versiones de la aplicación para distintas plataformas, buscamos una herramienta universal, que nos permita que el mismo código (o con variaciones minúsculas) se pueda ejecutar en distintas plataformas.

QT

QT es una librería multiplataforma para el desarrollo de aplicaciones. Cuenta con herramientas de desarrollo para los principales entornos software y permite emitir ejecutables

para las plataformas con mayor aceptación en el mercado, tanto de escritorio como móviles (Windows, MacOS, Linux, iOS, Android). Cuenta con soporte nativo para OpenGL. (Qt, 2016)

QT presenta varios puntos a su favor. En primer lugar, tiene bindings para múltiples lenguajes, lo que nos abre todo un abanico de posibilidades. De entre todos, destaca el lenguaje original en el que se construyó el proyecto C++. Los lenguajes compilados suelen proporcionar un buen rendimiento en las aplicaciones.

Los puntos en su contra son más dispersos. En primer lugar, el soporte para muchas de las plataformas móviles se ha incorporado en últimas versiones, lo que puede dar lugar a inconsistencias entre plataformas, obligándonos a disponer de infraestructura y materiales para testear y a, potencialmente, deber de mantener más de una versión de nuestro software, precisamente una de los inconvenientes que buscamos evitar. En segundo lugar, el licenciamiento. Aunque su versión para plataformas de escritorio es gratuita y publicada bajo una licencia abierta, a la hora de usar la herramienta para dispositivos móviles el licenciamiento no queda tan claro, pudiendo llegar incluso a enfrentarnos a problemas legales.

Tecnologías web

La tecnología entorno a la web ha sufrido grandes avances en la última década, convirtiéndola, no solo en un medio para visualizar hipertexto si no en una auténtica plataforma para ejecutar aplicaciones. Nombres importantes de la industria como Microsoft (con Office 365), Spotify, AirBnB, Pinterest y un largo etcétera ofrecen ya aplicaciones que poco tienen que envidiar a sus contrapartidas de escritorio.

Los principales navegadores modernos cuentan con versiones en prácticamente todas las plataformas del mercado, además, han centrado sus esfuerzos en implementar los últimos estándares del W3C, que cada vez van más orientados a hacer de la web una plataforma interactiva.

Esto es posible gracias a la homogeneidad que proporciona el navegador web, plataforma última sobre la que se ejecuta nuestra aplicación. Proporciona un entorno, basado en estándares abiertos y delegamos la responsabilidad de mantener la consistencia entre las plataformas a los propios fabricantes del navegador.

Sin embargo, esto trae una consecuencia inherente. Si bien las ventajas de tener un estándar estricto son más que evidentes, limita en gran medida las posibilidades de elección de lenguaje a desarrollar. El rendimiento normalmente ha sido una consideración negativa en este tipo de plataformas. La popularización de la web como sistema en los últimos años ha desencadenado una carrera por los principales navegadores para mejorar el rendimiento de sus intérpretes Javascript y abundan los libros y documentación sobre buenas prácticas a la hora de desarrollar para web.

Arquitectura de aplicación web

A continuación, exploramos las dos principales arquitecturas a la hora de diseñar una aplicación web.

Sin entrar en detalles de la arquitectura de la aplicación a nivel lógico, hay dos patrones hegemónicos a nivel conceptual. Uno pone el énfasis en la parte servidora y la otra descarga parte del procesamiento en el lado del servidor.

Arquitectura clásica

La arquitectura clásica de aplicaciones web dinámicas se basa en una aplicación o una extensión del servidor web para interpretar scripts que ante peticiones a distintos endpoints a través de HTTP renderizan un documento HTML que se envía al usuario.

En esta arquitectura las tareas de paginación, autenticación, enrutado o almacenamiento del estado de la sesión recaen sobre el componente servidor, dejando de lado del cliente las tareas de visualización del documento final.

A favor de la arquitectura clásica cuenta su sencillez a la hora de implementación. El hecho de que los datos de sesiones se almacenen por entero del lado de cliente, compartiendo solo la información indispensable le proporciona mayor seguridad a la aplicación. Es una arquitectura probada, con multitud de casos de uso exitosos. Por último y como veremos en el punto siguiente, presenta un amplio abanico de software, lenguajes y tecnologías con las que trabajar, al no estar atados a una plataforma más limitada en este aspecto, como es el caso del navegador web.

Como contrapunto, es una arquitectura que permite menos escalabilidad, debido a que el renderizado de los ficheros HTML es costoso en comparación con la simple serialización de los datos y el envío al frontend.

Single Page Application

La arquitectura Single Page Application o SPA se basa en una aplicación Javascript que se ejecuta del lado cliente y que es la encargada de las principales tareas (autenticación, paginación, enrutado, estado, que mostrar al usuario en cada interacción).

El papel del servidor en esta arquitectura es mínimo, actuando como un agente de autenticación contra una base de datos y proporcionando a la parte cliente con datos serializados (normalmente usando JSON u otro esquema de serialización) a medida que se realizan llamadas a el mismo.

Se trata de una arquitectura altamente escalable. La carga que supone realizar el renderizado recae sobre el cliente, dejando al servidor como una fachada entre la base de datos y el frontend. Es una arquitectura cada vez más popular, con numerosos frameworks y herramientas apoyados por los principales actores de la industria. Sirva como ejemplo React (Facebook, 2016), framework desarrollado por Facebook y usado en empresas como Pinterest (Chan, 2016) o AirBnB (AirBnB, 2016).

Como contrapunto, la complejidad añadida al diseño de la aplicación y la necesidad de incluir más componentes de lado cliente para manejar la navegación. La autenticación del usuario se vuelve más compleja siendo necesario requerir a herramientas y protocolos como Auth0 que requieren del uso de bibliotecas para tal uso tanto en cliente como en servidor. La limitación de lenguajes inherente al uso de del navegador reduce el abanico de posibilidades a la hora de elegir la herramienta adecuada para el trabajo adecuado.

Software para el backend

Procedemos a analizar las distintas soluciones para implementar el código de nuestro backend (parte de la aplicación que se ejecuta en el servidor).

El estándar web en el lado de servidor solo afecta a los protocolos de comunicación, por lo que, cualquier lenguaje que sea capaz de trabajar con conexiones de red y de interpretar y

escribir en dichos estándares será factible de ser usada. Tenemos por tanto más variedad en el lado de servidor a la hora de elegir nuestras herramientas.

PHP

Herramienta por antonomasia en el desarrollo web, PHP nace como en 1994 con el fin de crear pequeñas herramientas sencillas como contadores de visitas. (PHP, s.f.). Tiene una estructura de lenguaje de plantillas, con pequeños "snippets" que se incluyen en el código HTML o con ficheros independientes que el servidor HTTP lee e interpreta. Los endpoints de nuestra aplicación son los propios ficheros PHP.

Esta arquitectura hace difícil implementar aplicaciones estructuradas, incluso el hecho de que sean los propios ficheros PHP los que el usuario visita para desencadenar su ejecución deja expuesta en parte la estructura de nuestra aplicación.

Tampoco es un lenguaje que, de base, proporcione una estructura robusta y segura para el desarrollo de aplicaciones, siendo numerosos los casos de código inseguro en proyectos PHP que se encuentran en el mercado. (Cozic, 2016) Si bien es cierto que existen numerosos frameworks que intentan ofrecer herramientas robustas para desarrollar en PHP, como Symfony.

Ruby

Ruby es un lenguaje de interpretado, orientado a objetos de tipado dinámico desarrollado en 1995 por Yukihiro Matsumoto. Si bien se trata de un lenguaje de propósito general han surgido dentro de su ecosistema frameworks orientados a la programación web, destacando entre todos ellos Ruby On Rails.

Ruby on Rails es un framework "opinionado", es decir, impone una estructura muy clara al proyecto web y a las herramientas para trabajar con él. (Ruby on Rails, 2016) Si bien esto no suele ser un problema, ya que las herramientas que lo acompañan suelen ser de buena calidad y lo suficientemente maduras, no proporciona mucha flexibilidad.

Python

Python es un lenguaje interpretado, multiparadigma de tipado dinámico desarrollado por Guido van Rossum. Al igual que Ruby se trata de un lenguaje de propósito general alrededor

del cual ha surgido una variedad de frameworks para programación web, destacando entre todos Django, pero con notables excepciones como Flask.

Django al igual que Rails es un framework con una estructura, unas herramientas y un flujo de trabajo muy marcado. Es un proyecto grande, con muchas herramientas propias (presenta tu propio ORM) e incluso proporciona una plataforma de scaffolding.

Flask es un microframework que implementa solo funcionalidades básicas para la gestión de tareas básicas (rutas, sesiones) sin embargo posee una arquitectura modular que le permite trabajar e interactuar con otros componentes. En el apartado de la interconexión con bases de datos, existen extensiones para conectarlo al ORM SQLAlchemy. La renderización de documentos HTML se realiza mediante Jinja (un motor de plantillas que está disponible también en Django) aunque se puede elegir cualquier otro motor a tal efecto.

Go

Go es un lenguaje compilado, estáticamente tipado y con primitivas muy orientadas a la concurrencia (canales). (Wikipedia, 2016) Creado en 2009 es quizás el más joven de los lenguajes a considerar, pero el hecho de tener detrás a Google, junto con que entre sus diseñadores se encuentren personajes de la talla de Rob Pike o Ken Thompson le da a Go credibilidad.

Las capacidades para la concurrencia han hecho que Go destaque en sistemas cuya limitación principal son las operaciones de entrada / salida (por ejemplo, como servidor web). Un ejemplo claro es que la propia Google implementa su sistema de descargas mediante este lenguaje. (Fitzpatrick, 2013)

La propia biblioteca estándar de Go cuenta con herramientas para el manejo de HTTP, aunque hay conjuntos de herramientas para facilitar la tarea del programador.

El diseño del lenguaje, así como su estrategia (que pone énfasis en la configuración frente la convención, al contrario que Rails) no se presta a la existencia de muchos frameworks webs y los que hay no están tan maduros ni tan reconocidos. Sin embargo, las herramientas de la biblioteca estándar son más que suficientes para la implementación de un sistema web.

Sistema de gestión de bases de datos

En este apartado analizamos los principales motores de bases de datos relacionales. Como indicamos en la introducción, dejamos fuera aquellos de carácter propietario o con un elevado coste económico.

Los requisitos de la aplicación son lo suficientemente sencillos como para poder operar con un software de bases de datos modesto, si bien esto no es razón para escoger una herramienta con bajo desempeño o con pobres características.

MySQL

Software de bases de datos de código abierto desarrollado inicialmente por Sun Microsystems y actualmente. Durante años ha sido el motor de bases de datos relacional por antonomasia para aplicaciones web de bajo coste.

Presenta dos motores de almacenamiento, MyISAM, con una capacidad para transacciones superficial, pero un alto rendimiento en datos de lectura, e InnoDB, motor de almacenamiento con capacidad plena de transacciones y un rendimiento superior en escritura. Destacan sus capacidades para funcionar en modo de alta disponibilidad, permitiéndole funcionar en esquemas master / slave y master / master.

La compra de Oracle de Sun Microsystems ha supuesto que el proyecto haya ido siendo sucesivamente dejado de lado en favor del producto de bases de datos propio de Oracle.

PostgreSQL

Software de bases de datos de código abierto desarrollado inicialmente en la Universidad de Berkeley en la década de los 80 del siglo pasado. Inicialmente con código cerrado, su código fue liberado y desde hace tiempo está consiguiendo mucha tracción en las comunidades de desarrollo de software.

Al contrario que MySQL presenta un solo motor de almacenamiento, que concentra todos los esfuerzos de desarrollo. Esto, combinado con su largo ciclo de vida lo ha convertido de uno de los softwares de gestión de bases de datos más potentes de coste cero. Destaca también su soporte para tipos de datos JSON.

Software para el frontend

Analizamos las distintas opciones para construir la herramienta de frontend (parte cliente de la aplicación). Nos centramos en métodos para ayudarnos en el manejo de WebGL.

Las opciones en cuanto a lenguajes en este lado son limitadas. A pesar de los intentos de empresas como Microsoft en los 90 y 2000 de extender e incluir tecnologías propias, los estándares se han hecho con la web y los navegadores actuales miden parte de su calidad en el grado de adecuación con dichas recomendaciones.

No profundizamos en el estudio de frameworks para desarrollo de aplicaciones Javascript (como React, Vue, Angular) ya que suelen estar más orientados a aplicaciones de tipo SPA.

Emscripten

Compilador LLVM-a-Javascript (Zakai, s.f.) que permite compilar aplicaciones con soporte en la infraestructura LLVM a asm.js, un subconjunto de Javascript diseñado con el rendimiento como meta. Con soporte para OpenGL, este proyecto permite, entre otras opciones, que aplicaciones tan exigentes y en principio orientadas a escritorio como un motor de videojuegos ejecutarse en el navegador. (Mozilla Corporation, 2014)

Three.js

Creado por un programador bajo el seudónimo de mrdoob, Three.js es una biblioteca que proporciona un wrapper sobre las primitivas de WebGL además de proporcionar algunos contenedores, tipos de datos y herramientas extra. Creado en Javascript puro, proporciona un punto de partida válido para la construcción de aplicaciones WebGL.

dat.gui

Creado por el equipo de Chrome Experiments, perteneciente a Google, dat.gui es una biblioteca de controles para Javascript usada principalmente en proyectos gráficos y de motivación artística que son desarrollados por esta rama de la compañía. Proporciona una manera sencilla acompañada de una interfaz mínima y clara para mostrar por pantalla una serie de widgets (cuadros de texto, controles deslizantes, checkbox, botones)

jQuery

La biblioteca Javascript por antonomasia, desarrollada inicialmente en 2006 y con el objetivo de ocultar los distintos detalles de implementación de cada uno de los navegadores. Sin embargo, a pesar de ser una herramienta muy popular, no está desarrollada con el fin de actuar en grandes aplicaciones en el lado cliente, sino más bien como un conjunto de helpers para el programador y si no se usa con cuidado puede dar lugar al acoplamiento del código. (Osmani, 2015)

Webpack

Webpack es una herramienta de manejo de assets para una aplicación web. Se encarga de gestionar, procesar y preparar paquetes o bundles agrupando todos los ficheros estáticos necesarios en uno o varios ficheros o chuncks. El principal motivo es reducir el número de conexiones HTTP necesarias para cargar nuestra página, que, aún dependiente del navegador, suele estar limitado en HTTP 1 por debajo de las dos cifras en los principales navegadores. (Smith, 2012)

Webpack nos permite además expandir nuestras posibilidades con respecto a los lenguajes en el frontend. Podemos usar un preprocesador de CSS como es SASS para organizar de forma más efectiva nuestras hojas de estilo, podemos transformar código construido usando estándares más modernos del estándar ECMA (estándar básico de Javascript, como ya se ha comentado) usando transpilers como Babel e incluso podemos mejorar la legibilidad de nuestro código y detectar posibles fallos mediante el uso de linters, como ESLint.

Construido usando NodeJS permite usar el repositorio central de código del mismo (NPM) para la gestión de bibliotecas y sus dependencias, evitando tener que hacerlo de manera manual mediante etiquetas script en nuestro HTML.

Despliegue / infraestructura

Analizamos ahora las distintas plataformas para hospedar nuestro software. Nos centramos en aquellas que ofrecen soluciones gratuitas para proyectos con bajo requerimiento. Dejamos fuera del análisis por tanto la posibilidad de instalar la aplicación en una máquina alquilada bajo la fórmula de hosting clásico.

AWS

Amazon Web Services es un servicio puesto en marcha por Amazon de tipo IaaS (Infraestructure as a Service). El pilar básico de AWS es la instancia, una máquina virtual configurable organizada en tiers, pudiendo seleccionar las capacidades de cómputo, memoria, velocidad I/O etc.

Las instancias al crearse montan imágenes, con los datos de disco preparadas, permitiendo preparar una de estas imágenes y arrancar varias máquinas con el mismo software. Esto unido a la API que ofrecen para sus herramientas de monitorización permite crear sistemas auto escalables, en las que se crean máquinas y se destruyen sobre la marcha.

Al tratarse de un sistema IaaS la compatibilidad con el software no es un problema. Amazon proporciona imágenes base de los principales sistemas operativos para servidor, de los cuales todos soportan Python.

En lugar de instalar y configurar los servicios, Amazon provee además de herramientas de tipo SaaS (Software as a Service) entre las que se incluyen bases de datos relacionales, bases de datos de tipo clave-valor, almacenamiento de ficheros por volumen, CDN y un largo etcétera.

Ofrecen una opción gratuita con unas capacidades modestas.

Heroku

Heroku provee un servicio PaaS (Platform as a Service) que pone el enfoque en ofrecer una plataforma de despliegue fácil de usar para desarrolladores. El pilar básico de Heroku es el Dyno, un contenedor que ejecuta una aplicación.

Abstrae todas las operaciones de control de sistemas, como son la configuración de servidor web, bases de datos, etc. permitiendo definir y controlar los procesos de despliegue mediante ficheros de texto. Además, provee de ganchos o hooks para herramientas de control

de versiones, que permiten que una vez completado un commit, automáticamente se inicie el despliegue de la aplicación automáticamente. Los distintos servicios para que la operación trabaje se añaden como dynos extra.

Al tratarse de un servicio PaaS debemos controlar que lenguajes y aplicaciones es capaz de desplegar en su infraestructura. Rails es su "first class citizen" dado que fue el primer entorno que soportó, pero durante su ciclo de vida ha ganado soporte para los principales sistemas y frameworks, incluyendo nuestra plataforma principal Python, así como nuestra plataforma secundaria, Node.

Su plan gratuito provee de un dyno para la ejecución de la aplicación web además de ofrecer una base de datos.

Dokku

Construido a partir de código liberado con licencia abierta por Heroku y sustituyendo el motor de contenedores por Docker, Dokku se puede definir como una alternativa a Heroku auto hospedada.

Presenta compatibilidad con la mayoría de ficheros y métodos de trabajo de Heroku, lo que permite que una aplicación pensada para desplegar en este se pueda desplegar en Dokku sin apenas esfuerzo extra.

Siguiendo el camino de compatibilidad con Heroku, también podemos desplegar Python en Dokku.

No tiene coste alguno en cuanto a licenciamiento, pero requiere de una máquina sobre la que ejecutarlo.

OpenShift

OpenShift es un servicio de tipo PaaS desarrollado por Red Hat y basado por completo en software libre. Basado también en la tecnología de contenedores, permite escoger el motor de contenedores a usar, dándonos libertad entre Docker y Kubernetes.

Como el resto de sistemas de este tipo, los despliegues en OpenShift se pueden realizar automáticamente mediante herramientas de control de versiones.

Tiene soporte para las principales plataformas de desarrollo incluyendo Python y Node.

Travis CI

Travis CI (Continuous Integration) es un servicio que permite automatizar la ejecución de tests de varios tipos (tests unitarios, de cobertura).

Al igual que Heroku, presenta hooks para trabajar con las principales herramientas de control de versiones, además de integración con plataformas como Heroku, condicionando el despliegue de la aplicación a la finalización de los tests de forma positiva.

Ofrece un plan gratuito para proyectos de software libre e integración con GitHub.

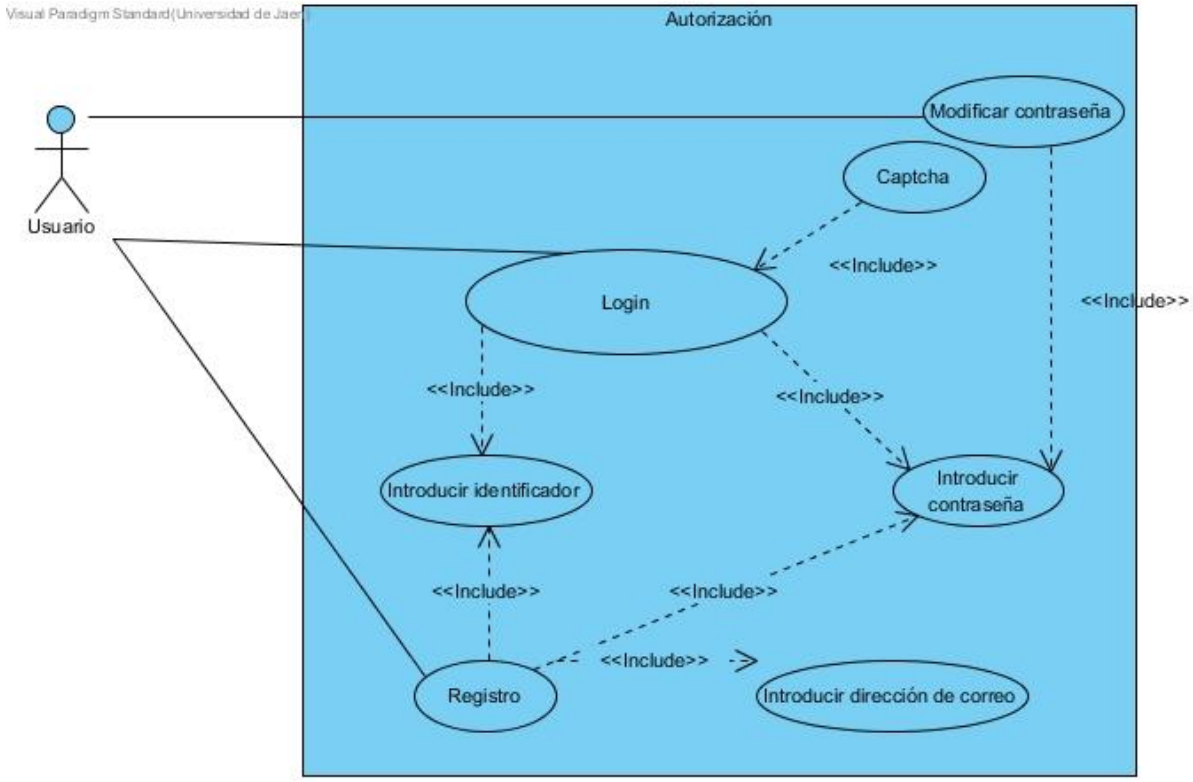
Casos de Uso de la aplicación

Detallamos los casos de uso de la aplicación web a desarrollar.

Diagramas de casos de uso

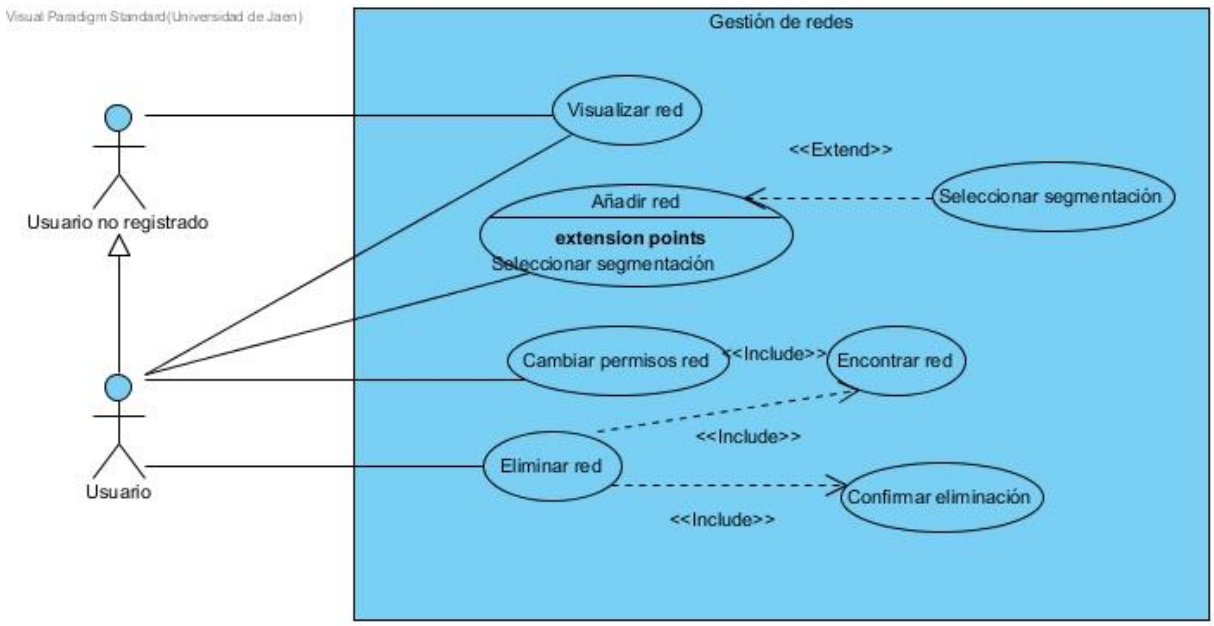
Autenticación del usuario

Desde el sistema de autenticación, el usuario puede introducir sus credenciales para iniciar una sesión de trabajo en el sistema. Además, puede crear un nuevo usuario en caso de no estar registrado.

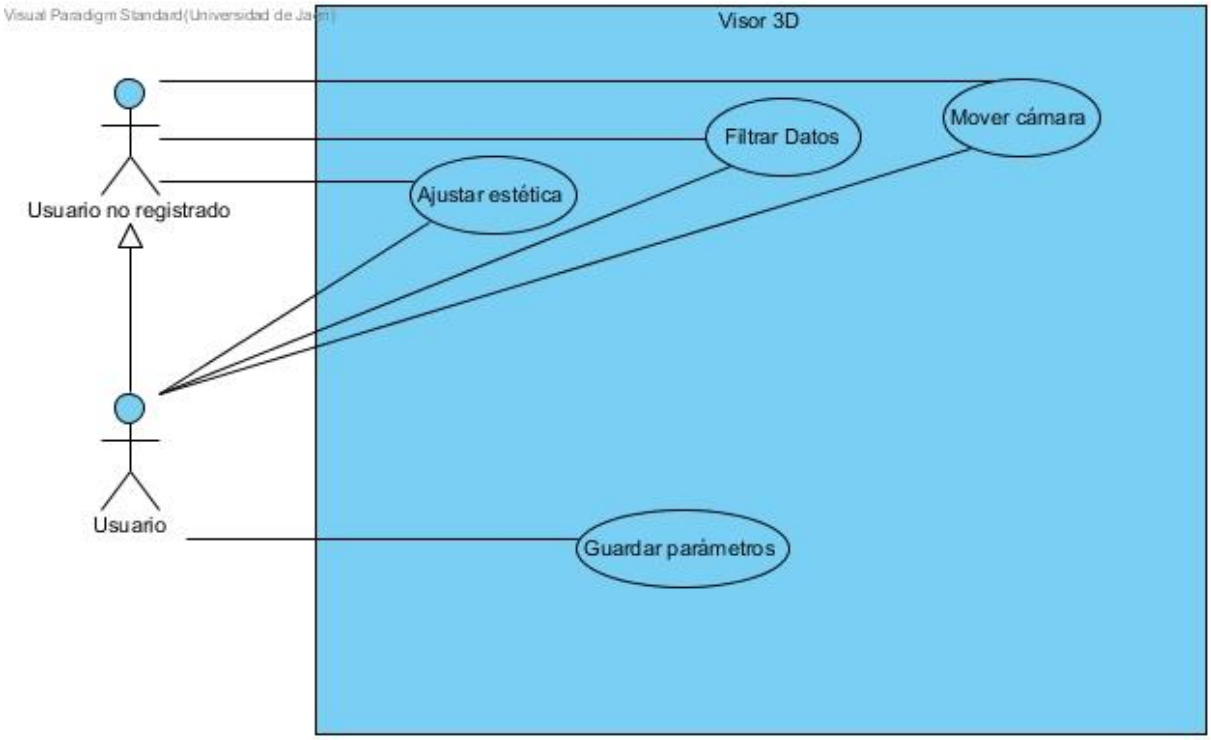


Gestión de redes cerebrales

Desde el sistema de visualización de redes, el usuario puede consultar las redes públicas, así como las suyas propias. Un usuario registrado podrá además añadir nuevas redes al sistema, cambiar la visibilidad de sus propias redes, así como eliminarlas.



Visualizar redes



Narrativas de los casos de uso

Autenticación del usuario

Actor principal: Usuario no registrado

Precondiciones: Ninguna

Camino principal:

1. El usuario introduce su identificador.

2a. El usuario introduce su contraseña.

Camino Alternativo

1.b Si el nombre de usuario no existe, se avisa de que las credenciales no son válidas. Se repite desde el punto 1.

2.b Si la contraseña no es correcta, se avisa de que las credenciales no son válidas. Se repite desde el punto 1.

3.a El usuario añade su dirección de correo.

3.b Si alguno de los datos no es válido para crear una cuenta, se avisa al usuario y se repite desde el punto 1.

Registro de nuevo usuario

Actor principal: Usuario no registrado

Precondiciones: Ninguna

Camino Principal:

1.a El usuario introduce el identificador que desea usar.

2.a El usuario introduce la dirección de correo que desea usar.

3.a El usuario introduce la contraseña que desea usar.

4. El confirma el registro

Camino Alternativo

1.b El identificador está en uso por otro usuario. Se repite desde el punto 1.a

2.b La dirección de correo está en uso. Se repite desde el punto 2.a

3.b La contraseña no cumple con los requisitos mínimos exigidos. Se repite desde el punto 3.a

Modificación de contraseña

Actor Principal: Usuario

Precondiciones: El usuario debe estár correctamente identificado en el sistema.

Camino Principal:

1.a El usuario introduce su contraseña actual.

2.a El usuario introduce su nueva contraseña.

Camino Alternativo

1.b La contraseña introducida no se corresponde con la actual. Se repite desde el punto 1.a

2.b La nueva contraseña no cumple los requisitos mínimos. Se repite desde el punto 1.a

Encontrar red

Actor Principal: Usuario

Precondiciones: Ninguna

Camino Principal:

1.a Se muestra al usuario un listado con las redes disponibles.

2.a El usuario introduce algún criterio de búsqueda.

Camino Alternativo:

2.b El usuario introduce un nombre de red. Puede continuar en 2.c o acabar.

2.c El usuario introduce una fecha de creación.

3.a Si el usuario no está registrado y la red que intenta visualizar es privada no se le permite el acceso. Se repite desde 1.a.

3.b Si el usuario intenta visualizar una red privada de la que no es dueño no se el permite el acceso. Se repite desde 1.a

Visualizar red

Actor Principal: Usuario

Precondiciones: Estar identificado en el sistema. Haber completado el caso de uso “encontrar red”.

Camino Principal:

1.a Se carga el visor 3D

Camino Alternativo

1.b Si el usuario no está registrado y la red que intenta visualizar es privada no se le permite el acceso. Se repite desde 1.a

1.c Si el usuario intenta visualizar una red privada de la que no es dueño no se le permite el acceso. Se repite desde el punto 1.a

Cambiar permisos de una red

Actor Principal: Usuario

Precondiciones: Estar autenticado en el sistema. Haber completado el caso de uso “encontrar red”.

Camino Principal:

1.a El usuario escoje que visibilidad tendrá la red.

Añadir Red

Actor Principal: Usuario

Precondiciones: Estar identificado en el sistema.

Camino Principal:

1.a El usuario proporciona el fichero desde el que se creará la red.

2. El usuario seleccionará o proporcionará el fichero con la parcelación a usar.

3. El usuario selecciona la visibilidad (permisos) que tendrá la red creada.

Camino Alternativo:

1.b El fichero proporcionado no es válido. Se repite desde el punto 1.a

Eliminar red

Actor Principal: Usuario.

Precondiciones: Estar identificado en el sistema. Haber completado el caso de uso “encontrar red”

Camino Principal:

1.a Se confirma que el usuario quiere realmente borrar la red.

Camino Alternativo:

1.b Si la confirmación no se realiza, se vuelve al flujo de trabajo normal.

Filtrar Datos

Actor Principal: Usuario

Precondiciones: Haber completado el caso de uso “Visualizar Red”

Camino Principal:

1.a El usuario introduce el valor mínimo del peso de las conexiones que se mostrarán en el grafo.

Ajustar estética

Actor Principal: Usuario

Precondiciones: Haber completado el caso de uso “Visualizar Red”

Camino Principal:

1.a El usuario escoje el color del que se mostrarán las conexiones entre zonas.

Camino Alternativo:

1.b El usuario escoje la forma con la que se mostrarán las conexiones entre zonas.

1.c El usuario escoje el color con el que se mostrará la base del cerebro.

1.d El usuario escoje el sombreado con el que se mostrará la base del cerebro.

1.e El usuario escoje el nivel de transparencia con la que se mostrará la base del cerebro.

Mover cámara

Actor Principal: Usuario

Precondiciones: Haber completado el caso de uso “Visualizar Red”

Camino Principal:

1. El usuario escoje desde que posición desea visualizar la escena.

Guardar parámetros

Actor Principal: Usuario

Precondiciones: Estar identificado en el sistema. Haber completado el caso de uso “Visualizar Red”.

Camino Principal:

1.a El usuario guarda los parámetros actualies de visualización en su perfil.

Gestión de redes cerebrales

Actor Principal: Usuario

Precondiciones: Tener permisos sobre la red. Haber completado el caso de uso "encontrar red"

Camino Principal:

1. Se le muestra al usuario las opciones sobre la red. En función de la escogida, se le deriva al caso de uso correspondiente.

Visualizar redes

Actor Principal: Usuario

Precondiciones: Tener permiso para visualizar la red.

Camino Principal:

1.a. El usuario elige la red a visualizar y pulsa el enlace correspondiente.

2. Se le muestra la red al usuario.

Criterios de validación del producto obtenido

La validación del producto viene supeditada al grado de cumplimiento de los requisitos funcionales y en menor medida al grado de cumplimiento de los requisitos no funcionales.

Si bien estos requisitos se han redactado con la idea de desarrollar un producto final que ofrezca al usuario la gestión sobre algunos aspectos de su cuenta o sobre la privacidad de los datos, debemos de considerarlos requisitos secundarios y centrar los esfuerzos del desarrollo en producir un prototipo mínimo funcional que incorpore las capacidades actuales que tienen los estándares y tecnologías web para construir aplicaciones gráficas.

Tomaremos como indispensables para la implementación del prototipo los requisitos funcionales desde el RF3.1 hasta el RF3.7 y el grado de cumplimiento de los mismos indicará el éxito o no del proyecto. El resto de RF serán secundarios, no por ello descuidándolos, pero haciendo que pasen a un segundo plano durante la implementación final del proyecto.

3. Plan del proyecto

En este capítulo mostraremos la parte más puramente gestora y organizativa de la ejecución del proyecto.

Una vez redactados los requisitos tenemos una carta de especificaciones que debemos desarrollar, sin embargo, las dos principales restricciones en el campo de la ingeniería son el tiempo y el presupuesto.

Planificaremos las fases de nuestro proyecto para hacer frente a estas dos restricciones, poniendo especial hincapié en cumplir los criterios de validación del producto especificados por el cliente.

Planificación temporal

Tareas y estimaciones

Tras recopilar todos los requisitos para nuestro proyecto, llega la hora de descomponer los mismos en tareas asignables. En la tabla 1.1 se muestran las tareas, así como su estimación temporal, suponiendo una jornada diaria de 8 horas.

Tabla 1.1. Descripción de tareas del proyecto y estimación de desarrollo temporal.

Tarea	Descripción	Estimación temporal (días)
Requerimientos de la aplicación	Entrevistas y elaboración de los requisitos funcionales y no funcionales de la aplicación.	2
Casos de uso	Elaboración de los casos de uso y redactado de los mismos, tanto en su modalidad gráfica como su modalidad narrativa.	2
Planificación temporal	Descomposición del proyecto en etapas y tareas y elaboración de una planificación temporal.	1

Estimación de costes	Cálculo aproximado del coste en función de la planificación temporal, horas / hombre y coste de los recursos materiales del proyecto.	2
Diseño de modelo de datos	Elaboración de un modelo de los datos de la aplicación en base a los requisitos recolectados. Diseño de la base de datos	2
Storyboards	Elaboración de diagramas para la interfaz persona / ordenador, proporcionando las interfaces para la consecución de los requisitos funcionales.	2
Diseño	Modelado general de los principales componentes de la aplicación así como la interacción entre los mismos en forma de diagramas de clase y diagramas de secuencia.	3
Estudio de posibles soluciones		
Estudio de herramientas software	Elaboración de un dosier poniendo de manifiesto pros y contras de las distintas tecnologías y herramientas disponibles.	1
Estudio de las soluciones de hosting	Elaboración de un dosier poniendo de manifiesto pros y contras de las distintas	1

	soluciones para hospedar nuestra aplicación así como las capacidades de distintos servicios gestionados.	
Fase de implementación		
Configuración de "boilerplate"	Etapa previa al desarrollo. Consiste en desarrollar y configurar una base para nuestro proyecto, entre las tareas se incluyen, preparar cuentas de usuario en los servicios seleccionados, preparación de infraestructura de tests unitarios así como configuración de los distintos entornos de desarrollo.	2
	Implementación de Backend	
Implementación de base de datos	Creación a partir del modelo propuesto anteriormente de una estructura lógica en forma de tablas relacionales.	1
Implementación de sistema de autenticación de usuario	Capacidad del sistema para aceptar nuevos usuarios y permitirles iniciar sesiones de trabajo, usando su par identificador-contraseña.	1
Implementación de serialización de redes cerebrales	Capacidad de transformar los ficheros de entrada proporcionados por el usuario en datos listos para consumir por nuestro sistema.	3

Implementación del sistema de almacenamiento de estado	Capacidad de la aplicación para almacenar el estado actual de la visualización del usuario.	5
Implementación del frontend		
Implementación del renderizado de escena	Capacidad para mostrar una escena 3D, además de permitir la modificación de parámetros estéticos de elementos estáticos (modelo 3D del cerebro)	4
Implementación de renderizado de red cerebral	Capacidad de consumir datos proporcionados por el backend pertenecientes a la red cerebral y de dibujar una escena 3D en función de los mismos.	6
Implementación de recuperación de estado desde backend	Capacidad de consumir datos proporcionados por el backend pertenecientes al estado de la visualización de la escena (parámetros estéticos elegidos, posición de cámara) y de actualizar el estado actual en función de los mismos.	5
Documentación		
Manual de uso	Se redactará el manual de usuario para facilitar al usuario final el uso de la aplicación.	2
Manual de despliegue	Se redactará el manual de despliegue para que otros	2

	administradores puedan desplegar la aplicación.	
Testeo	Pruebas de validación de los usuarios	3
TOTAL		50 días = 400 horas

Planificación mediante diagramas Gantt

Tras descomponer el proyecto en tareas concretas, analizamos y preparamos una estimación mediante el sistema de diagrama Gantt. Entre las ventajas que nos da este sistema está la de poder calcular el llamado camino crítico, un conjunto de tareas cuya demora en el tiempo afectan al resto de planificación.

Tabla de tareas y predecesoras

Tabla 1.2 Tareas, duración estimada, precedencias y personal encargado.

Número	Nombre	Duración estimada (días)	Predecesoras	Encargado
1	**Proyecto**	**37**		
2	Inicio	0		
3	**Etapa de diseño**	**14**		
4	Requerimientos de la aplicación	2		
5	Casos de uso	2	4	Analista
6	Storyboards	2	5	Analista
7	Diseño de modelo de datos	2	6	Analista
8	Diagramas de secuencia	3	7	Analista
9	Planificación temporal	1	8	Analista
10	Estimación de costes	2	9;13	Analista
11	**Estudio de soluciones**	**2**		
12	Estudio de herramientas software	1	7	Analista
13	Estudio de plataformas hospedaje	1	12	Programador web
14	**Etapa de implementación**	**20**	**3**	
15	Configuración de "boilerplate"	1	13	2
16	**Implementación backend**	**12**	**15**	

17	Implementación de base de datos	1	10;12	Programador web
18	Implementación del sistema de login y perfil de usuario	2	17	Programador web
19	Implementación de serialización de red cerebral	5	18	Programador web
20	Implementación de sistema de almacenamiento de estado	4	19	Programador web
21	**Implementación frontend**	**19**	**15**	
22	Implementación de renderizado de escena	4	12	Programador gráfico
23	Implementación de renderizado de red cerebral	6	22;19	Programador gráfico
24	Implementación de recuperación de estado desde backend	5	23;20	Programador web
25	**Documentación**		14	
26	Manual de usuario	2		Programador gráfico
27	Manual de despliegue	2		Programador web
28	Testeo	4	21	
29	Funcionalidad básica	Hito	23	
30	Fin de Proyecto	Hito	25	

Justificación de las precedencias

Las precedencias establecen el orden de las tareas en función de las necesidades y requisitos de cada una.

Habrá partes del sistema que dependan de otras (por ejemplo, no se puede hacer autenticación de usuario sin que exista una base de datos sobre la que almacenar sus credenciales.

En primer lugar, las tareas principales de la etapa de diseño siguen una planificación lineal, dependiente las unas de las otras.

Las etapas de la fase de diseño son independientes con respecto a las tecnologías que se van a usar, sin embargo, el diseño del modelo de datos condiciona la base de datos que se va a usar. De ahí que la tarea número 12, estudio de las herramientas de software, no pueda comenzar hasta una vez creado el modelo.

La estimación de costes dependerá de las herramientas y plataformas a utilizar, de ahí la decisión de aplazar este paso hasta una vez finalizadas las tareas relacionadas con el estudio de soluciones. Por ejemplo, el ejemplo de un sistema de hospedaje para el sitio puede influir en el coste final.

La etapa de implementación no comenzará hasta que a consecuencia de la estimación de costes se dé el visto bueno al desarrollo de la aplicación. No queremos comenzar un proyecto que sepamos de antemano que va a ser irrealizable bajo nuestras restricciones de tiempo y presupuesto.

La primera tarea dentro de la etapa de implementación es la configuración del boilerplate. Adaptamos y configuramos el entorno de desarrollo para trabajar con las herramientas que hemos seleccionado en el estudio de soluciones.

La siguiente etapa es el desarrollo del backend. En primer lugar, debemos de implementar nuestro modelo de datos y adaptarlo para el sistema seleccionado, creando ficheros SQL y comenzando a modelarlo en el lenguaje escogido.

Una vez implementada la base de datos, podemos empezar a construir el sistema de gestión y registro de los usuarios. Sobre esta base, construimos el sistema de serialización de red cerebral, ya que es el usuario el que crea las nuevas redes, es necesario que el sistema tenga soporte para perfiles individuales. Por último, implementamos el sistema de serialización de parámetros de visualización.

La otra etapa mayor de desarrollo es el sistema de implementación del frontend. La primera fase, la implementación de la visualización de escena no depende para nada del sistema de usuario, pudiéndose desarrollar en paralelo durante las fases tempranas del proyecto.

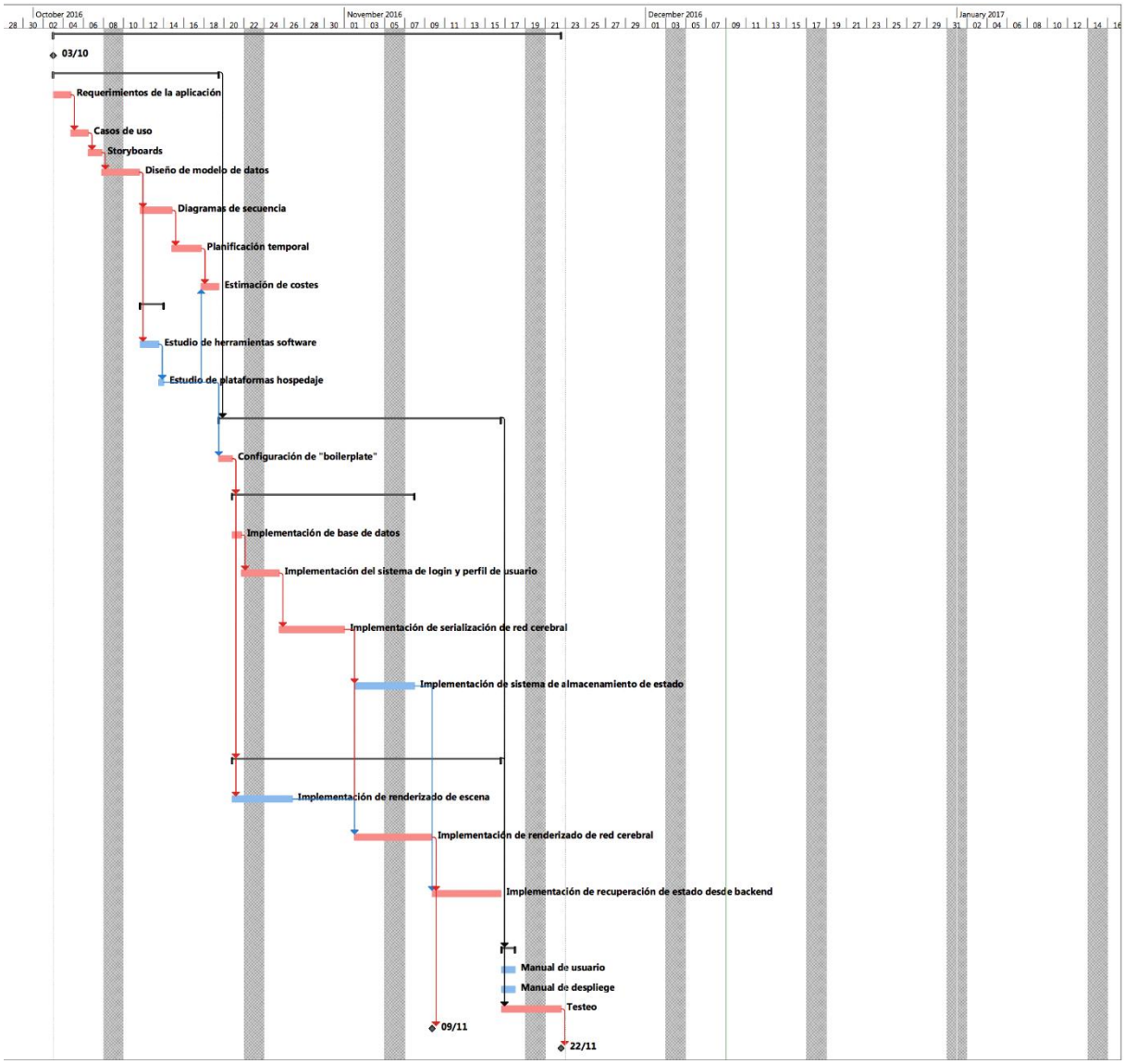
Una vez hemos implementado esta parte, el paso natural es implementar la visualización de los datos de red cerebral. Dichos datos deben de ser recuperados desde el backend de la aplicación, por lo que debemos esperar a que se termine esta fase. Aquí se marca un hito del proyecto y es que una vez podamos mostrar redes cerebrales por pantalla, cumpliremos los requisitos mínimos expuestos en el punto 2.5 sobre criterios de validación del proyecto. La última fase también depende de la implementación en el backend del usuario.

La fase de testeo de los requisitos se añade por convención al final del proyecto, sin embargo, una vez tengamos implementadas las funcionalidades mínimas requeridas podemos comenzar con el proceso de testeo de las mismas por parte de los usuarios finales.

Procesos como el desarrollo de tests unitarios se irán realizando como parte de las tareas de implementación.

La fase final de documentación se realizará una vez hayan completado las tareas de implementación del producto, si bien no es una condición estricta, ya que parte de dichas tareas se pueden realizar sobre la marcha durante la fase de implementación.

Diagrama Gantt



Recursos

Recursos humanos

En función de las distintas etapas del desarrollo necesitaremos personal cualificado para tal fin. Se requiere la presencia de tres roles principales, aunque todos serán asumidos por el alumno que ha realizado el TFG.

Calcularemos los salarios base en función del "XIII Convenio Colectivo de ámbito estatal para los centros de educación universitaria e investigación" publicado por el Ministerio de Empleo y Seguridad Social. (Ministerio de Empleo y Seguridad Social, 2015)

Para el rol de programador gráfico y dado el perfil particular que buscamos, consideramos una mayor dotación económica ya que el mercado laboral en este sector en particular es escaso.

Tabla 1.3 Salario mensual y anual del personal

Roles	Salario Mensual	Salario diario
Analista	1.548,37 €	77,42 €
Programador web	1.074,05 €	53,70 €
Programador gráfico	1.120,40 €	56,02 €

Se incluye también una estimación de los costes en recursos humanos asociadas a cada una de las tareas del proyecto.

Tabla 1.4 Coste de las tareas en función de las horas de trabajo y del salario del personal.

Tarea	Rol	Horas	Días trabajados	Coste
Analista		160	20	1.548,37 €
Requerimientos de la aplicación	Analista	*16*	2	154,84 €
Casos de uso	Analista	*16*	2	154,84 €
Storyboards	Analista	*16*	2	154,84 €
Diseño de modelo de datos	Analista	*16*	2	154,84 €
Diagramas de secuencia	Analista	*24*	3	232,26 €

Planificación temporal	Analista	*8*	1	77,42 €
Estimación de costes	Analista	*16*	2	154,84 €
Estudio de herramientas software	Analista	*8*	1	77,42 €
Testeo	Analista	*40*	5	387,09 €
			0	
Programador gráfico		112	14	751,84 €
Implementación de renderizado de escena	Programador gráfico	*40*	5	268,51 €
Implementación de renderizado de red cerebral	Programador gráfico	*56*	7	375,92 €
Manual de usuario	Programador gráfico	*16*	2	107,41 €
Programador web		200	25	1.400,50 €
Estudio de plataformas hospedaje	Programador web	*8*	1	56,02 €
Configuración de "boilerplate"	Programador web	*16*	2	112,04 €
Implementación de base de datos	Programador web	*8*	1	56,02 €
Implementación del sistema de login y perfil de usuario	Programador web	*16*	2	112,04 €
Implementación de serialización de red cerebral	Programador web	*48*	6	336,12 €

Implementación de sistema de almacenamiento de estado	Programador web	*40*	5	280,10 €
Implementación de recuperación de estado desde backend	Programador web	*48*	6	336,12 €
Manual de despliegue	Programador web	*16*	2	112,04 €

Recursos materiales

Para el desarrollo del proyecto necesitaremos diversos recursos materiales, tanto físicos como lógicos.

Algunos de ellos ya están en propiedad de antemano, por lo que amortizaremos su coste para la duración del proyecto. El ordenamiento con respecto a la amortización de bienes viene regulado por la Ley 27/2014 del 27 de noviembre sobre el Impuesto sobre Sociedades, publicada en el Boletín Oficial del Estado a fecha 28 de noviembre de 2014. (Gobierno de España, 2014)

Dicha ley estipula que el periodo de amortización para software es de 6 años, 2190 días, mientras que para los equipos para procesos de información (hardware) es de 8 años, 2920 días.

Software

Tabla 1.5 Costes de software en propiedad adquiridos mediante licencia.

Recurso	Coste total	Periodo amortización (días)	Amortizado/ día
Software			
Microsoft Windows 10 Pro	279,00 €	2190	0,13 €

Visual Paradigm Community Edition	0,00 €	2190	0,00 €
Microsoft Project Standard 2016	769,00 €	2190	0,35 €
Mac OS Sierra 10.12	0,00 €	2190	0,00 €
Microsoft Office Hogar y Estudiante para Mac	149,00 €	2190	0,07 €
Ubuntu Server 16.04			
Neovim Editor	0,00 €	2190	0,00 €
YouCompleteMe Autocompletion Plugin	0,00 €	2190	0,00 €
Atom Editor	0,00 €	2190	0,00 €
Tern Autocompletion Plugin	0,00 €	2190	0,00 €
Google Chrome	0,00 €	2190	0,00 €
Mozilla Firefox	0,00 €	2190	0,00 €
Git	0,00 €	2190	0,00 €

Hardware

Tabla 1.6 Costes de hardware necesarios para el desarrollo del proyecto.

Hardware	Coste Total	Periodo Amortización	Amortizado / día
MacBook Pro 15" con descuento estudiante	2.049,00 €	2920	0,70 €
Portatil HP	780,00 €	2920	0,27 €
Monitor Dell P2415Q	485,99 €	2920	0,17 €

Running Services

Servicios de tipo suscripción, normalmente mensual a los que nos suscribiremos durante el desarrollo de la aplicación o durante su mantenimiento. La decisión para incluir software en este coste es que, depende de la plataforma que usemos, el coste de usar determinado servicio (ej. Redis) puede ir asociado a una cuota mensual.

Tabla 1.7 Costes de software como servicio.

Recurso	Coste mensual	Días / mes	Coste diario
Software (Despliegue)			
Python v2.7.0	0,00 €	31	0,00 €
Node.js v12.0	0,00 €	31	0,00 €
THREE.js Librería Javascript	0,00 €	31	0,00 €
Dat.gui Biblioteca Javascript	**0**,00 €	31	0,00 €
Servicios			
Conexión a internet (Movistar)	41,00 €	31	1,32 €
Heroku Free Plan	0,00 €	31	0,00 €

Costes absolutos

Costes de materiales o servicios de cuota única usados con el fin exclusivo de la realización o consecución del proyecto, como bibliografía o costes editoriales.

Tabla 1.8 Coste de material bibliográfico.

Recurso	Coste
Impresión y encuadernación documentación	20,00 €
Learning Three.js: The Javascript 3D Library for WebGL	29,02 €
JavaScript: The Good parts	34,26 €
Total	83,28 €

Organización del personal

La jornada laboral será una jornada estándar de 8 horas. Se trabajará 5 días en semana y a efectos de cálculo, se tomarán meses de 4 semanas.

El analista será el encargado de evaluar las necesidades del producto y de desarrollar toda la documentación previa a la implementación, es decir, documentación de los requisitos funcionales, documentación de los requisitos no funcionales, evaluación de las soluciones disponibles, diseño de bases de datos, diseño lógico del programa, etc. Deberá de tener experiencia en este tipo de tareas además de contar con conocimientos de UML y ofimática básica para la elaboración de informes.

El programador web se encargará de la implementación de todo lo relacionado con el backend de la aplicación, implementación de la base de datos, sistema de autenticación de usuarios, etc. Deberá de tener experiencia en los lenguajes indicados, experiencia construyendo sobre tecnologías web comunes.

El programador gráfico se encargará de la implementación de todo lo relacionado con la visualización de gráficos 3D. Implementará los sistemas de renderizado, visualización de redes, etc. Deberá de tener experiencia en el campo de la informática gráfica además de contar con experiencia en entornos Javascript. Se valorará positivamente el conocimiento de tecnologías web en general.

Estimación de costos del proyecto

Las estimaciones en cuanto al coste del desarrollo se realizan en función de las estimaciones de tiempo realizadas en el apartado 3.1. A parte de los costes de desarrollo incluimos los costes de mantenimiento.

Dentro de los costes de mantenimiento incluimos gastos necesarios para el funcionamiento del sistema como el hospedaje.

Una vez desarrollado el producto, se incluye dentro del precio el coste de tres meses de mantenimiento. Una vez vencido este tiempo, se podrá prorrogar este periodo. El funcionamiento de la aplicación irá ligado a la compra del mantenimiento.

Estimamos que, una vez finalizado el proceso de desarrollo, durante los tres primeros meses las posibles tareas de corrección de errores no supondrán más de un total de siete días por cada mes de trabajo de los desarrolladores.

Descartamos costes económicos derivados del pago de las instalaciones al estar nuestra sociedad ubicada en un vivero de empresas promovido por un organismo público andaluz.

Coste recursos humanos desarrollo

Mostramos los costes derivados de pagar salarios a los distintos trabajadores involucrados en la realización del proyecto.

Tabla 1.9 Coste total de recursos humanos.

Roles	Salario Mensual	Salario diario	Días trabajados	Costo
Analista	1.548,37 €	77,42 €	20	1.548,37 €
Programador web	1.074,05 €	53,70 €	14	751,84 €
Programador gráfico	1.120,40 €	56,02 €	25	1.400,50 €
Total Desarrollo	**3.742,82 €**	**187,14 €**	**59**	**3.700,71 €**

Coste recursos humanos mantenimiento

Como se ha apuntado anteriormente, estimamos que el tiempo total para la localización y corrección de errores que queden cubiertos dentro del mantenimiento no supondrá un esfuerzo en horas mayor a siete días por cada mes.

Tabla 1.10 Coste de recursos humanos durante el periodo de mantenimiento.

Rol	Estimación días / mes	Costes mensuales
Programador web	7	375,92 €
Programador gráfico	7	392,14 €
Total mantenimiento mensual		**768,06 €**

Coste de software

A continuación, presentamos los costes de software. Durante la etapa de mantenimiento se da por cerrada la fase de diseño de la aplicación, por lo que no incluiremos costes relacionados con el software dedicado para tal fin, principalmente Microsoft Project, Visual Paradigm, Microsoft Office o Microsoft Windows.

Tabla 1.11 Desglose de costes de software.

Recurso	Coste total	Amortizado/ día	Uso desarrollo (días)	Uso mantenimiento (días / mes)	Coste amortizado (Desarrollo)	Coste amortizado (Mantenimiento)
Microsoft Windows 10 Pro	279,00 €	0,13 €	20	0	2,5479 €	0,0000 €
Visual Paradigm Community Edition	0,00 €	0,00 €	20	0	0,0000 €	0,0000 €
Microsoft Project Standard 2016	769,00 €	0,35 €	20	0	7,0228 €	0,0000 €
Mac OS Sierra 10.12	0,00 €	0,00 €	43	14	0,0000 €	0,0000 €
Microsoft Office Hogar y Estudiante para Mac	149,00 €	0,07 €	20	0	1,3607 €	0,0000 €
Ubuntu Server 16.04				14	0,0000 €	0,0000 €
Neovim Editor	0,00 €	0,00 €	43	14	0,0000 €	0,0000 €
YouCompleteMe Autocompletion Plugin	0,00 €	0,00 €	43	14	0,0000 €	0,0000 €
Atom Editor	0,00 €	0,00 €	43	14	0,0000 €	0,0000 €
Tern Autocompletion Plugin	0,00 €	0,00 €	43	14	0,0000 €	0,0000 €
Google Chrome	0,00 €	0,00 €	43	14	0,0000 €	0,0000 €

Mozilla Firefox	0,00 €	0,00 €	43	14	0,0000 €	0,0000 €
Git	0,00 €	0,00 €	43	14	0,0000 €	0,0000 €
Total				Total	10,9315 €	0,0000 €

Coste de hardware

Tabla 1.12 Desglose de costes de hardware

Hardware	Coste Total	AM/Día	**Uso Desarrollo**	**Uso Mantenimiento**	**Coste Amortizado (Desarrollo)**	**Coste Amortizado (Mantenimiento)**
MacBook Pro 15" con descuento estudiante	2.049,00 €	0,70 €	36	14	25,2616 €	9,8240 €
Portatil HP	780,00 €	0,27 €	36	14	9,6164 €	3,7397 €
Monitor Dell P2415Q	485,99 €	0,17 €	36	14	5,9917 €	2,3301 €

Costes totales

Tabla 1.13 Resumen de costes totales del proyecto.

	Desarrollo	**Mantenimiento**	**Meses contratados**	**Total**
RRHH	3.700,71 €	768,06 €	3	6.004,88 €
Software	10,93 €	0,00 €	3	10,93 €
Hardware	40,87 €	15,89 €	3	88,55 €
Running Costs	56,87 €	18,52 €	3	112,42 €
Costes Absolutos				83,28 €
Total				**6.300,06 €**

La estimación del coste total de ejecución del proyecto asciende a seis mil trescientos euros con seis céntimos.

4. Diseño

Diseño de base de datos

En este apartado presentamos lo relativo al diseño de la base de datos, así como estudios sobre las estimaciones de uso de la misma.

Los requisitos funcionales de la aplicación indican que hay tres entidades que deberán ser almacenadas por la aplicación. La primera, los usuarios. Es necesario un registro de los usuarios junto con sus datos para la autenticación. Del usuario almacenaremos pues el par usuario contraseña para identificarlo. Incluiremos también su dirección de correo electrónico. Si bien los requisitos funcionales no señalan nada a este respecto, guardaremos esta información por si fuese necesario ponerse en contacto con algún usuario en particular.

La siguiente entidad son las redes. Es necesario almacenar los datos de las mismas subidos por el usuario, así como una cantidad mínima de datos que la identifiquen. Estos metadatos serán, un nombre, una breve descripción y la fecha de registro en el sistema. Cada red será propiedad de un usuario, que podrá tener varias redes y podrá decidir si la misma es visible al público o desea almacenarla en privado. Las redes son representadas mediante un grafo, indicando conexiones entre los distintos nodos. Como red, entendemos solamente las conexiones entre nodos, esto es, la matriz de adyacencia del grafo. Las distintas regiones que conforman el grafo serán las parcelaciones. El nombre de las tablas no será único de manera global para el sistema, si no que presentan las siguientes restricciones.

- No podrán existir redes públicas con el mismo nombre.
- Un usuario no podrá crear una red privada con el mismo nombre que otra de sus colecciones privadas.
- Un usuario no podrá crear una red privada con el mismo nombre que una pública.

La tercera entidad son las parcelaciones. Cada red usa únicamente una parcelación. Los usuarios son libres de crear y añadir nuevas parcelaciones al sistema. Esto es, cada usuario podrá tener una o más redes creadas. Al contrario que las redes, el usuario no puede decidir si las parcelaciones son públicas o no, por defecto, serán todas privadas. Es cierto que, el sistema cuenta con dos parcelaciones por defecto, que al ser las más comunes en el ámbito de la neurología serán públicas para todos los usuarios. Por simplicidad, optamos por incluir estas parcelaciones en la base de datos y para diferenciarlas de aquellas subidas por el usuario, mantendremos también un campo en la base de datos indicando si son públicas o no. De esta manera, facilitamos la inclusión de nuevas parcelaciones públicas en el futuro.

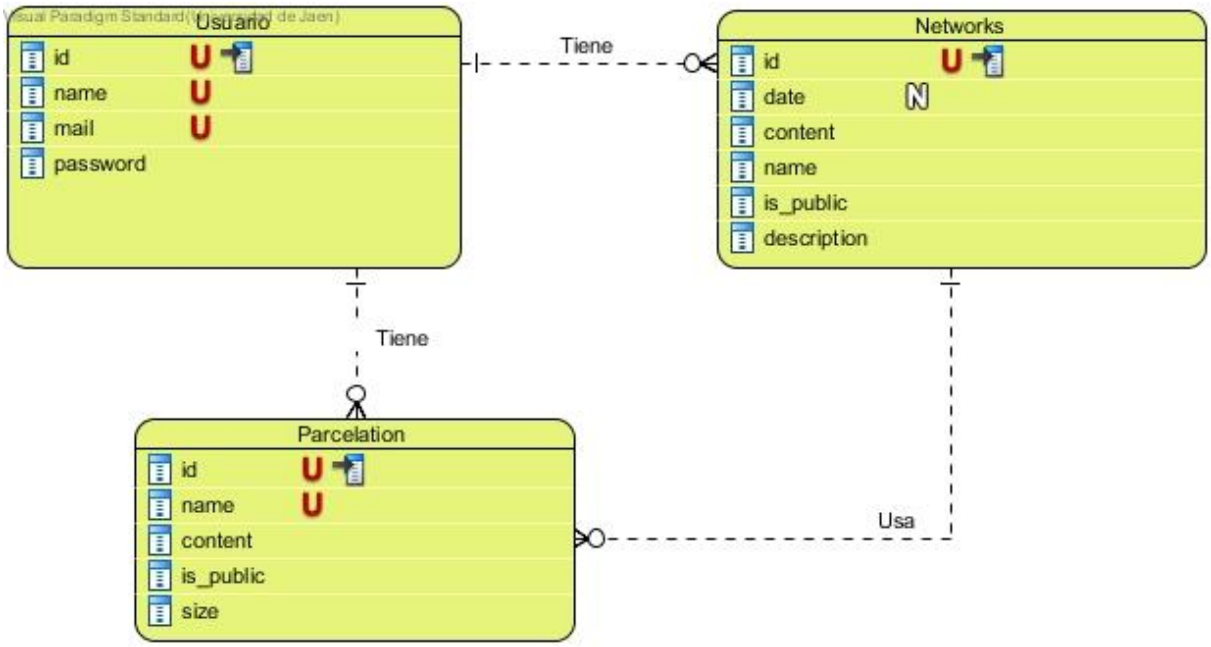


Figura 4.1 Diseño lógico de la base de datos.

Tenemos tres relaciones:

- R1: Tiene. Entre usuario y red. Multiplicidad 1:N
- R2: Tiene. Entre usuario y parcelación. Multiplicidad 1:N
- R3: Usa. Entre red y parcelación. Multiplicidad 1:N

Debido a que usaremos un ORM de tipo procedural, no es necesario el paso a tablas del esquema lógico. En su lugar modelamos el esquema lógico usando técnicas de diseño de programación orientada a objetos y el sistema ORM se encargará automáticamente de las operaciones de bajo nivel.

El ORM elegido permite sin embargo configurar algunos aspectos del diseño físico de datos, como el tamaño de los campos. Estos aspectos se discutirán más adelante.

Sin embargo, el trabajo del ORM no es suficiente para implementar algunas de las restricciones que se piden, en concreto la concerniente al nombre de las redes. Es por ello que, una vez generado el esquema físico de la base de datos, procedemos a analizarlo y a implementar manualmente disparadores (triggers) que aseguren dichas restricciones. Se prefiere usar esta técnica a implementarlo directamente en la lógica de la aplicación debido a la mejora de la integridad que supone. Es más probable que exista un fallo en nuestro código a que lo haga en el DBMS.

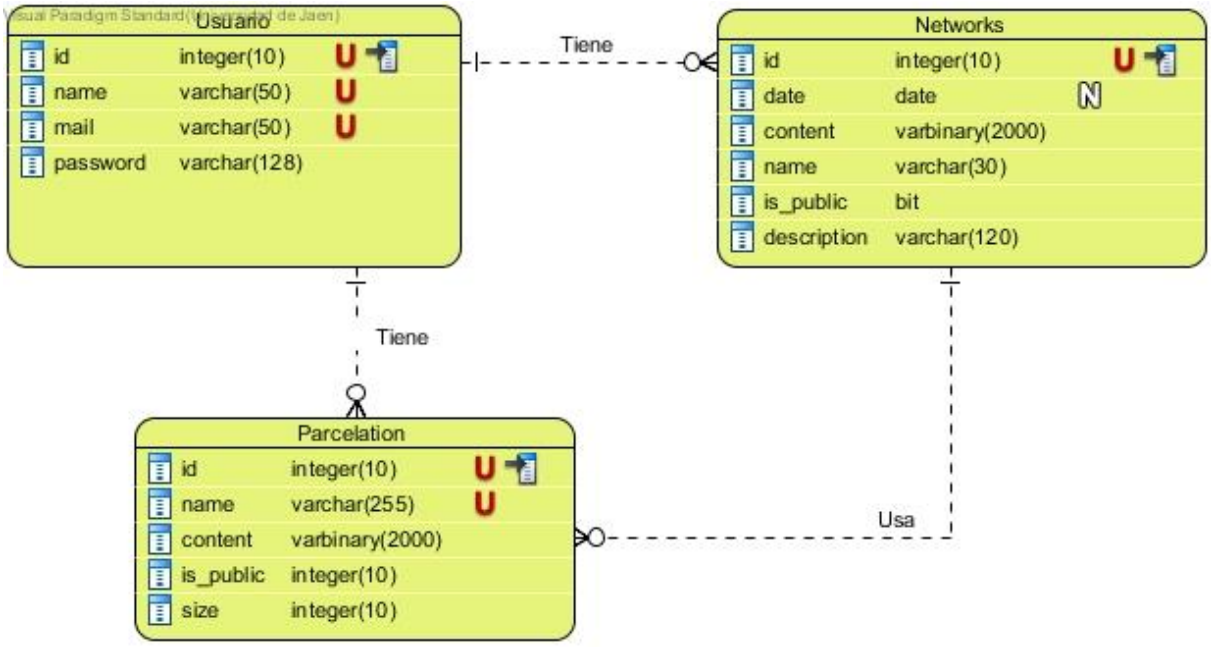


Figura 4.2 Esquema físico resultante de la base de datos.

Análisis de volumetría

Procedemos a realizar el análisis de volumetría. Conociendo el esquema físico de la base de datos y con ello el tamaño de los campos de datos, podemos realizar estimaciones de tamaño total que ocupará la misma en base a la cantidad de usuarios proyectados.

En primer lugar, analizaremos el tamaño en disco de los tipos de datos. Para los campos variables (varchar, bytea / binary) realizamos dos estimaciones.

La primera, una estimación pesimista, suponiendo que los campos se llenarán al máximo de lo permitido. El caso de los campos varchar merece especial mención. PostgreSQL utiliza

codificación UTF-8 para almacenar las cadenas de caracteres. En esta codificación, el subconjunto de caracteres de UTF-8 que pertenece también a la codificación ASCII solo ocupa un byte, pero el resto de caracteres (eñes, vocales con tildes, etc.) pueden llegar a ocupar hasta 4 bytes. Para el análisis de peor caso, tomaremos cadenas compuestas por caracteres UTF-8 de 2 bytes, por ejemplo, la eñe. Excluimos de esta regla el campo password, al tratarse de una codificación base64 del hash de la contraseña, que garantiza que el rango de caracteres se mantiene en el subconjunto ASCII de UTF-8.

La segunda estimación será un análisis más realista de los datos, con valores variables menores que el límite y con una distribución más habitual de los caracteres en la tabla de codificación.

A continuación, se mostrarán los datos ordenados por tablas de la base de datos, calculando el total por fila en cada uno de los casos. Tras mostrar los datos por tablas, se mostrará un resumen total de la estimación del tamaño de la base de datos.

Para el cálculo del número de filas, partimos de los siguientes supuestos:

- Se registrarán 10 usuarios.
- Cada usuario añadirá 10 redes cada uno.
- Cada usuario añadirá 4 parcelaciones propias.

Además, se contabilizarán las dos parcelaciones añadidas por defecto (AAL90 y Brodmann82) y la red de muestra añadida en el sistema.

Para los tamaños de datos, tomamos como referencia la documentación de PostgreSQL 9.6 (Group, 2016)

En primer lugar, analizamos el tamaño fijo de los campos de datos, es decir, el tamaño inmutable asociado a un campo por el mero hecho de aparecer en la tabla.

Tipo de dato	Tamaño fijo (Bytes)
integer	4 bytes
varchar	1 byte + tamaño de datos
date	4 bytes

boolean / bit	1 byte
bytea / varbinary	4 bytes + tamaño de datos

Tabla 4.1. Tamaño fijo de tipos de datos en PostgreSQL

Análisis pesimista

Como hemos comentado antes, en el análisis pesimista calculamos el volumen de datos suponiendo un llenado completo de los campos variables, con caracteres fuera del rango ASCII que requieren bytes extra. Los campos de parcelaciones y redes no tienen tamaño máximo, por lo que usamos una red de gran tamaño para realizar las estimaciones. La red cuenta con 4000 nodos, por lo que el volumen de datos será grande.

Mostramos la estimación de volúmenes ordenados según las tablas de la base de datos.

Columna	Tipo de dato	Tamaño Fijo	Tamaño variable (Bytes)	Tamaño total (Bytes)
id	integer	4	0	4
name	varchar(30)	1	60	61
mail	varchar(50)	1	100	101
_password	varchar(128)	1	128	129
Total / fila				295

Tabla 4.2. Análisis pesimista de volumen de datos en tabla Usuarios.

Columna	Tipo de dato	Tam. Fijo (Bytes)	Tam. variable (Bytes)	Tam. total (Bytes)
id	integer	4	0	4
content	bytea	4	7696183	7696187
user_id	integer	4	0	4
parcelation_id	integer	4	0	4
is_public	boolean	1	0	1
date	date	4	0	4

name	varchar(30)	1	60	61
description	varchar(120)	1	240	241
Total / fila				7696506

Tabla 4.3. Análisis pesimista de volumen de datos en tabla Networks.

Columna	Tipo de dato	Tam. Fijo (Bytes)	Tam. variable (Bytes)	Tam. total (Bytes)
id	integer	4	0	4
name	varchar(30)	1	23870	23871
user_id	integer	4	0	4
content	bytea	4	0	4
is_public	boolean	1	0	1
size	integer	4	0	4
Total / fila				23888

Tabla 4.4. Análisis pesimista de volumen de datos en tabla Parcelations.

Tabla	Tamaño por fila	Número de filas	Tamaño Total (Bytes)
Usuarios	295	10	2950
Networks	7696506	101	777347106
Parcelations	23888	42	1003296
Total BBDD		153	778353352
Total MB			742,30

Tabla 4.5. Estimación de tamaño total de la base de datos según el análisis pesimista.

Análisis optimista

Columna	Tipo de dato	Tamaño Fijo (Bytes)	Tamaño variable (Bytes)	Tamaño total (Bytes)
id	integer	4	0	4
name	varchar(30)	1	15	16
mail	varchar(50)	1	30	31
_password	varchar(128)	1	128	129
Total / fila				180

Tabla 4.6. Análisis realista de volumen de datos en tabla Usuarios.

Columna	Tipo de dato	Tamaño Fijo (Bytes)	Tamaño variable (Bytes)	Tamaño total (Bytes)
id	integer	4	0	4
content	bytea	4	15002	15006
user_id	integer	4	0	4
parcelation_id	integer	4	0	4
is_public	boolean	1	0	1
date	date	4	0	4
name	varchar(30)	1	16	17
description	varchar(120)	1	60	61
Total / fila				15101

Tabla 4.7. Análisis realista de volumen de datos en tabla Networks.

Columna	Tipo de dato	Tamaño Fijo (Bytes)	Tamaño variable (Bytes)	Tamaño total (Bytes)
id	integer	4	0	4
name	varchar(30)	1	10	11
user_id	integer	4	0	4
content	bytea	4	5445	5449
is_public	boolean	1	0	1
size	integer	4	0	4
Total / fila				5473

Tabla 4.8. Análisis realista de volumen de datos en tabla Parcelations.

Tabla	Tamaño por fila	Número de filas	Tamaño Total (Bytes)
Usuarios	180	10	1800
Networks	15101	101	1525201
Parcelations	5473	42	229866
Total BBDD		153	1756867
Total MB			1,68

Tabla 4.9. Estimación de tamaño total de la base de datos según el análisis realista

Conclusiones

Como podemos observar, el valor dominante en el volumen de la base de datos es el peso de las redes. En los análisis pesimistas con redes de gran tamaño, sólo esta tabla llega a pesar 740 megas, siendo el 99% del tamaño total de la base de datos. En los análisis más realistas, el volumen de dicha tabla supone el 86% del total, una cantidad porcentual nada despreciable.

Se incluye también el total de filas debido a que algunos proveedores de PaaS lo tienen en cuenta a la hora de tarificar el almacenamiento de base de datos.

Diseño de software

Interfaz de usuario

Para el diseño de la interfaz de usuario usaremos la técnica de storyboard. Con este sistema, podremos esbozar el aspecto final de la aplicación además de documentar las distintas interacciones entre las vistas que se mostrarán. Los dos pilares fundamentales para el diseño de la aplicación serán la simplicidad estética y una fácil navegación. Como se estipula en los requisitos funcionales, el software debería de tener una curva de aprendizaje plana, para que cumpla su misión como herramienta de apoyo a la exploración de datos.

Imagen 4.1 Storyboard. Listado de redes sin autenticación de usuario

Como se puede ver en la figura 4.1, cuando el usuario no ha iniciado sesión, se mostrarán para explorar solamente aquellas redes que sus respectivos dueños hayan marcado como públicas. Se mostrará un botón invitando al usuario a autenticarse en el sistema en la parte superior derecha. Los datos de usuario se suelen presentar en la mayoría de aplicaciones web en la parte superior de la pantalla, manteniendo este estándar de facto buscamos facilitar al usuario la navegación. Para visualizar la red en 3D, mantenemos un enlace por cada una de las redes.

En la parte superior incluimos dos enlaces a modo de breadcrum para facilitar la navegación. Un enlace al listado de redes y otro al perfil del usuario. Para visitar el perfil es necesario estar autenticado en el sistema, ya que no se permite la visualización de perfiles ajenos. Cuando un usuario sin autenticar intenta acceder a una vista reservada a usuarios autenticados, se le redirige a la vista de autenticación y si esta es satisfactoria, se le devolverá a la página a la que quería acceder originalmente (ver figura 4.2).

Imagen 4.2 Autenticación y registro de usuario.

Mostramos todos los formularios relacionados con la identidad el usuario en la misma página. Cada uno de ellos tiene un epígrafe indicando la funcionalidad de cada uno. Los formularios se validarán en el lado del servidor, indicando los posibles fallos en la parte superior de la página, con una tipografía y color que resalten sobre el resto de la página, para facilitar su visualización al usuario.

Imagen 4.3 Storyboard. Listado de redes sin autenticación de usuario.

Una vez el usuario se autenticado en el sistema, se procederá a mostrar, además de las redes públicas, las redes privadas del propio usuario (ver figura 4.3). El formato de visualización será el mismo que con las redes públicas, solo se incluirá un botón para añadir nuevas redes. Junto al listado de redes privadas, se mostrarán las parcelaciones propias del usuario. Se añade la cantidad de redes que la usan por aportar más información sobre las mismas. Debajo, un botón para añadir nuevas redes.

Esta es la vista más cargada de todas las diseñadas. Intentamos reducir la carga cognitiva del usuario incluyendo títulos grandes en cada apartado, además de separarlos los unos de los

otros en la medida de lo posible (factor que dependerá del tamaño de la ventana de visión, es decir, de la resolución de la pantalla).

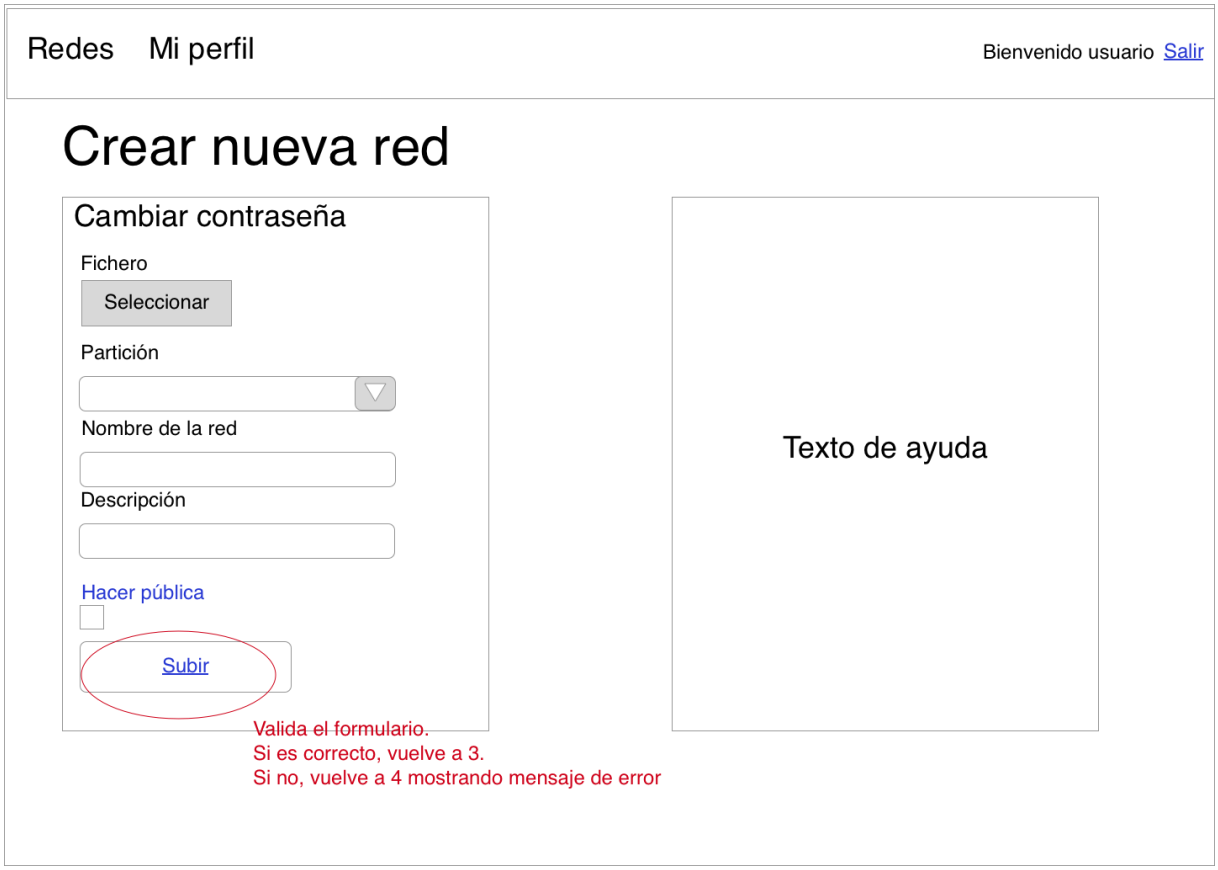


Imagen 4.4 Formulario de subida de una nueva red.

En la figura 4.4 se muestra el formulario para crear una nueva red. Solo será accesible para usuarios registrados en el sistema.

Se intenta que tanto el nombre como la ubicación de los campos del formulario sea intuitivo para los usuarios. Por si no fuese necesario, cuentan con ayuda en la parte derecha de la pantalla. La ayuda no se limita al proceso de subida y carga de los ficheros, sino que también muestra información sobre el formato esperado para los mismos.

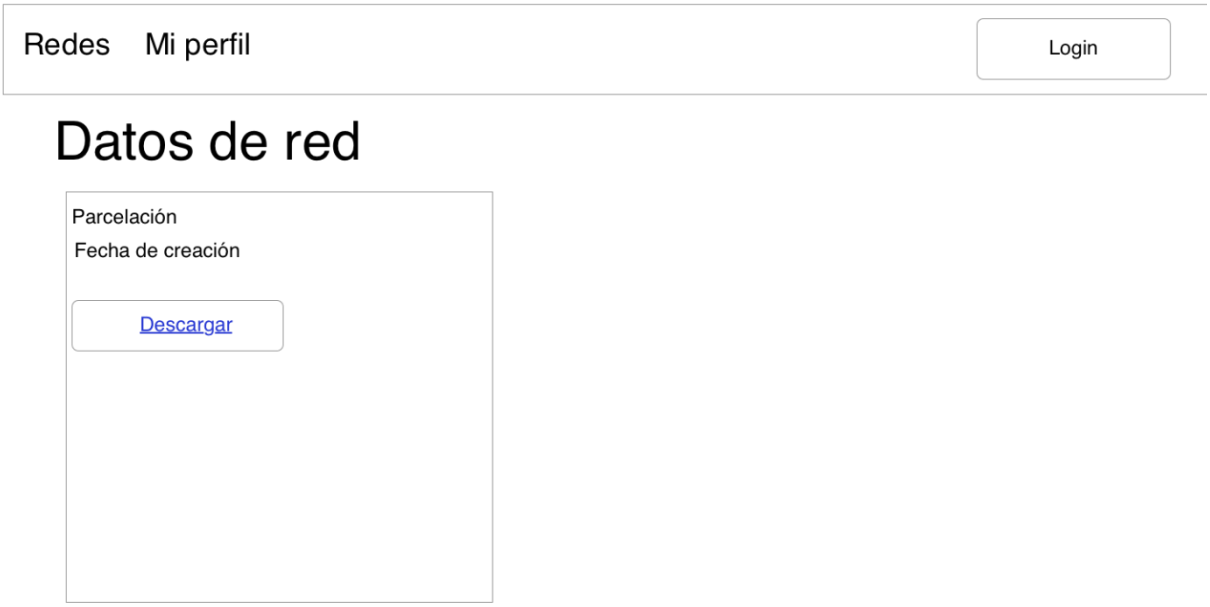


Imagen 4.5 Datos de una red sin autenticación de usuario.

En la vista relacionada con las propiedades de las redes (ver figura 4.5), podrá consultarse información básica sobre las mismas, esto es, la parcelación usada y la fecha de creación. Se podrá además descargar el fichero de la red para consulta de la misma en un software distinto a este o como referencia para crear nuevas redes. Debido a que la funcionalidad de descarga de la red es solo para usuarios más avanzados que busquen crear sus propias redes, se decide poner el botón en esta vista en lugar de en la vista de redes principal.

Esta vista también se mostrará a usuarios registrados en el sistema que intenten acceder a las propiedades de una red pública pero que no sea de su propiedad.

Imagen 4.6 Propiedades de red con autenticación de usuario.

Los usuarios registrados que accedan a las propiedades de una red de su propiedad verán además botones para ejecutar operaciones reservadas al propietario (figura 4.6). Dichas operaciones serán las de borrar la red del sistema y cambiar la privacidad de una red.

Imagen 4.7 Visionado de red cerebral.

La figura 4.7 muestra la vista principal del sistema de visualización. La información principal será el propio renderizado 3D del mismo. En la parte superior izquierda mostraremos información sobre el objeto que el usuario esté señalando con el ratón, permitiendo consultar datos sobre la red de manera precisa en lugar de mediante indicadores visuales. En la parte derecha se mostrará el GUI de la aplicación, con tres menús desplegables. Al pulsar en cada uno de los epígrafes, se desplegará el menú correspondiente.

- Opciones de filtrado (ver figura 4.7.1): Permite ajustar la cantidad de nodos y conexiones visibles en función de los parámetros de los mismos.
- Estética de escena (ver figura 4.7.2): Permite ajustar parámetros sobre la estética elementos invariantes de la escena (color de la malla del cerebro, transparencia de la misma, visualización o no de ejes)
- Estética de red (ver figura 4.7.3): Permite ajustar parámetros sobre la estética de los elementos que dependan de la red creada por el usuario (color de los vértices en función del peso, color de los nodos, etc.).

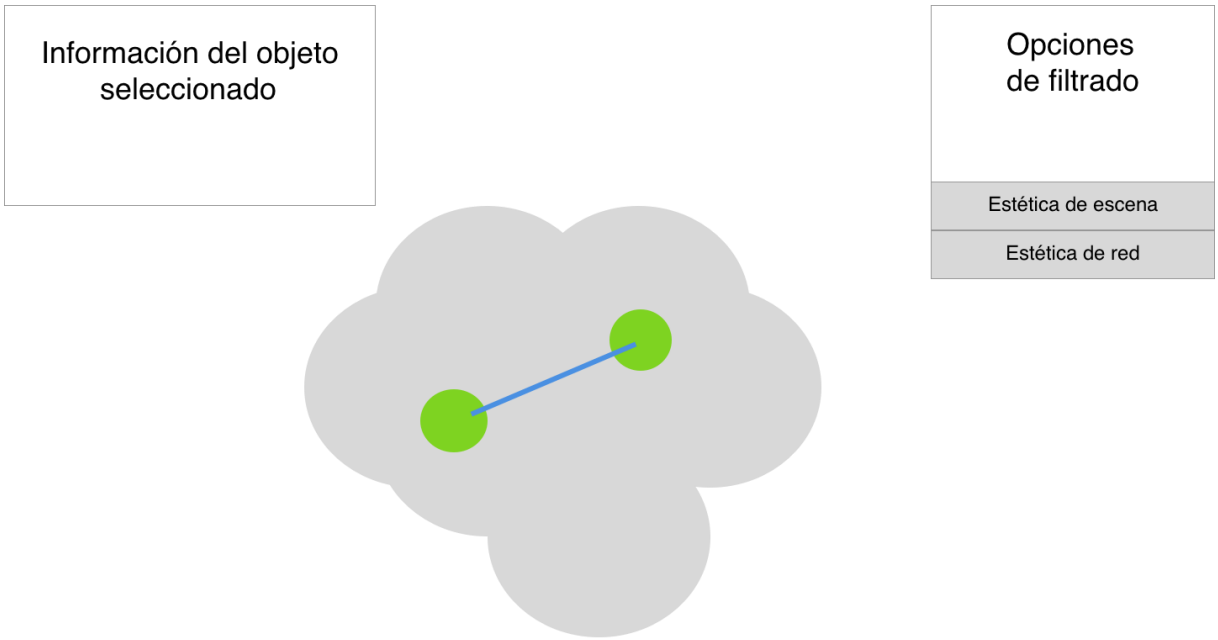


Imagen 4.7.1 Visionado de red cerebral mostrando opciones de filtrado.

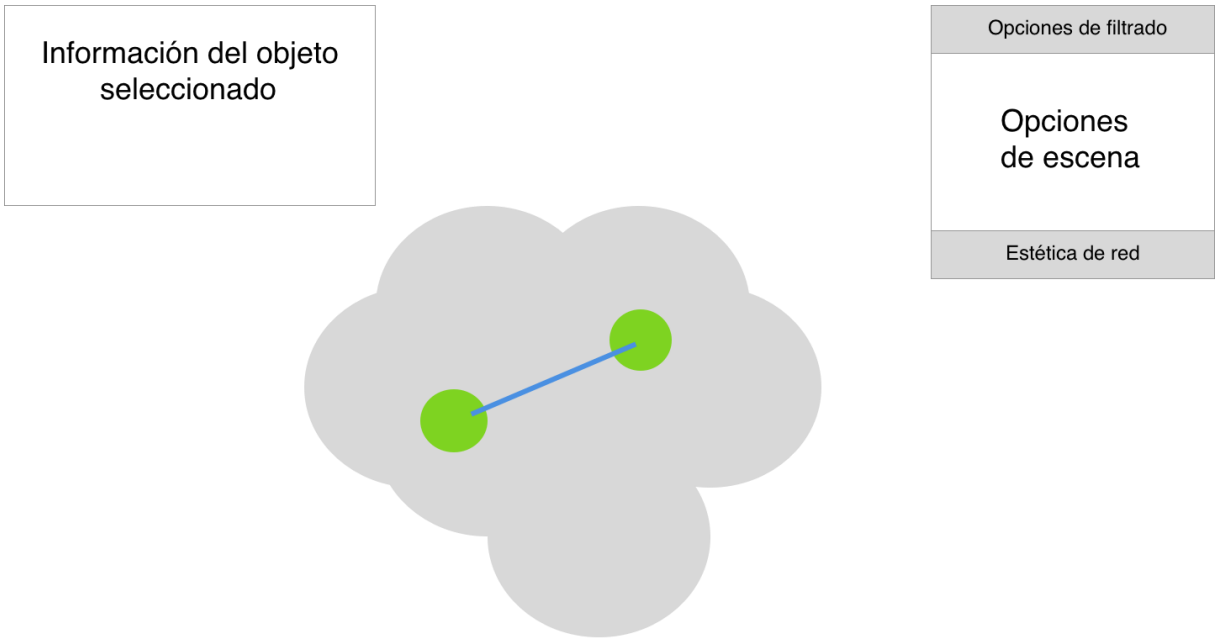


Imagen 4.7.2 Visionado de red cerebral mostrando opciones de escena.

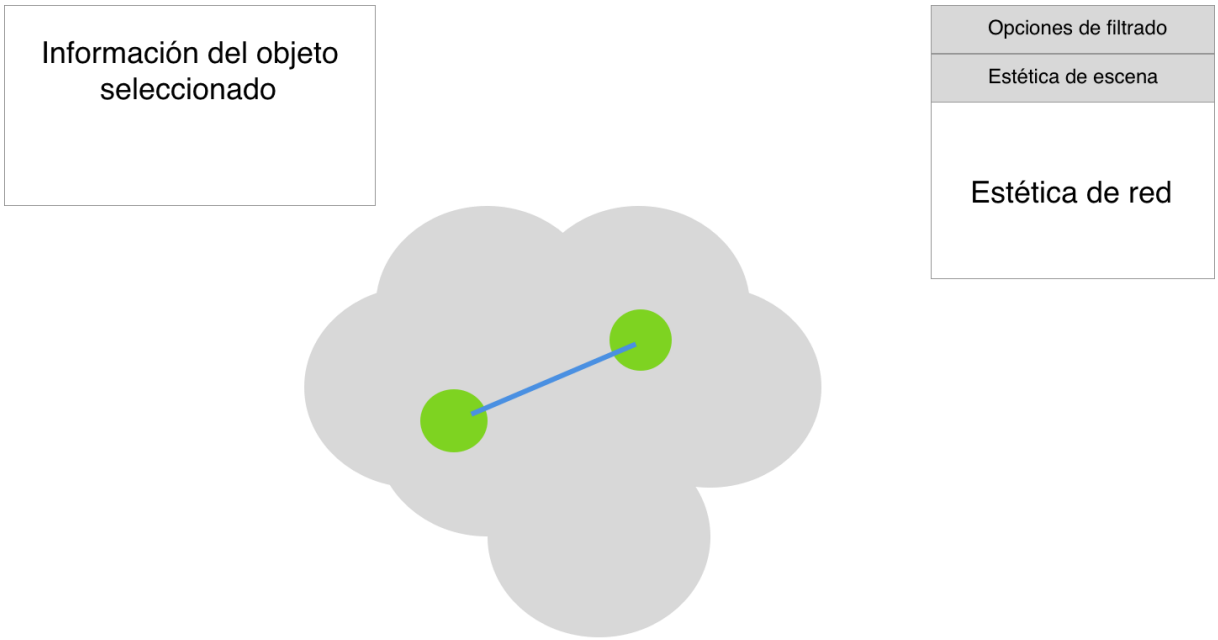
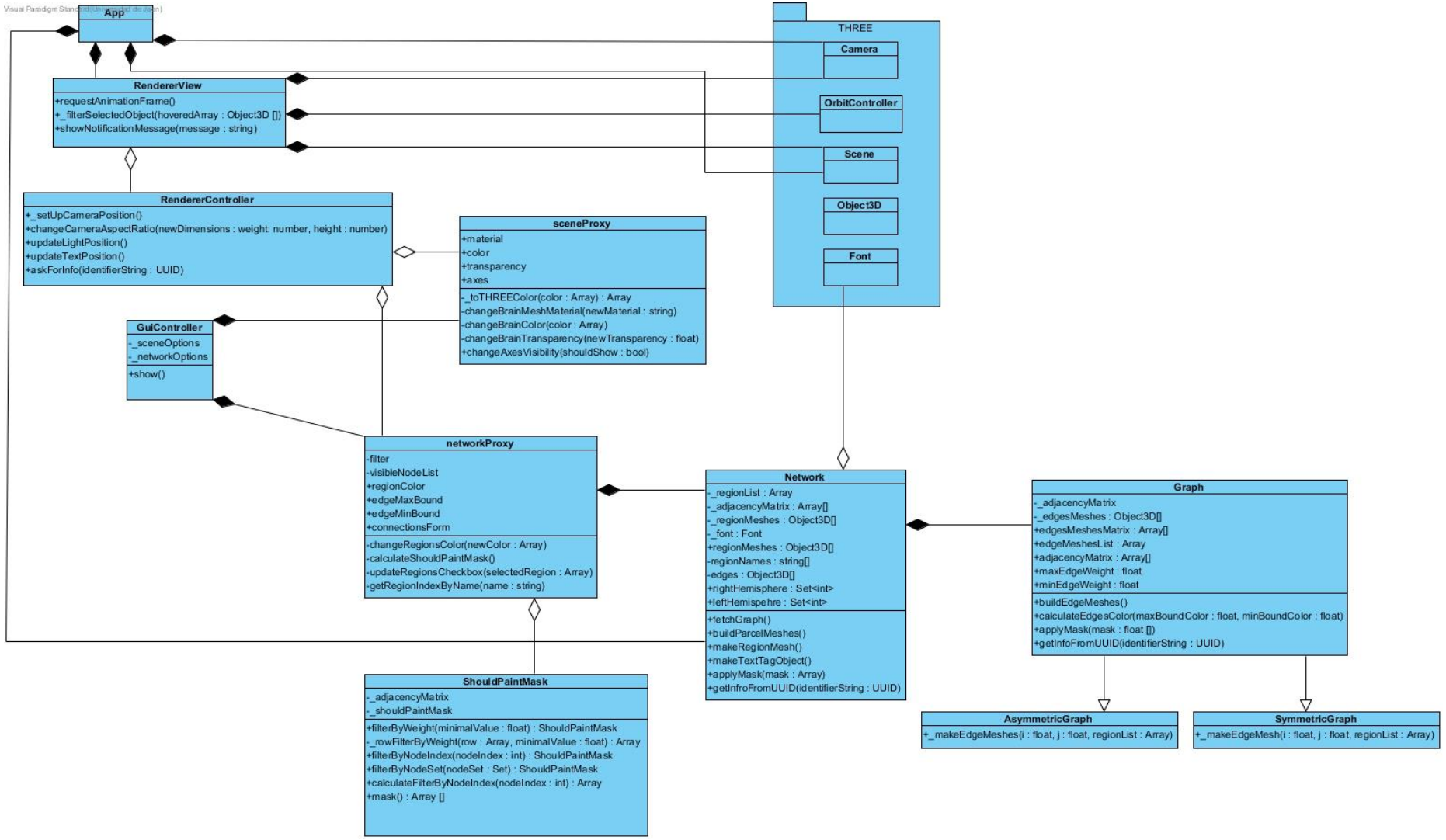
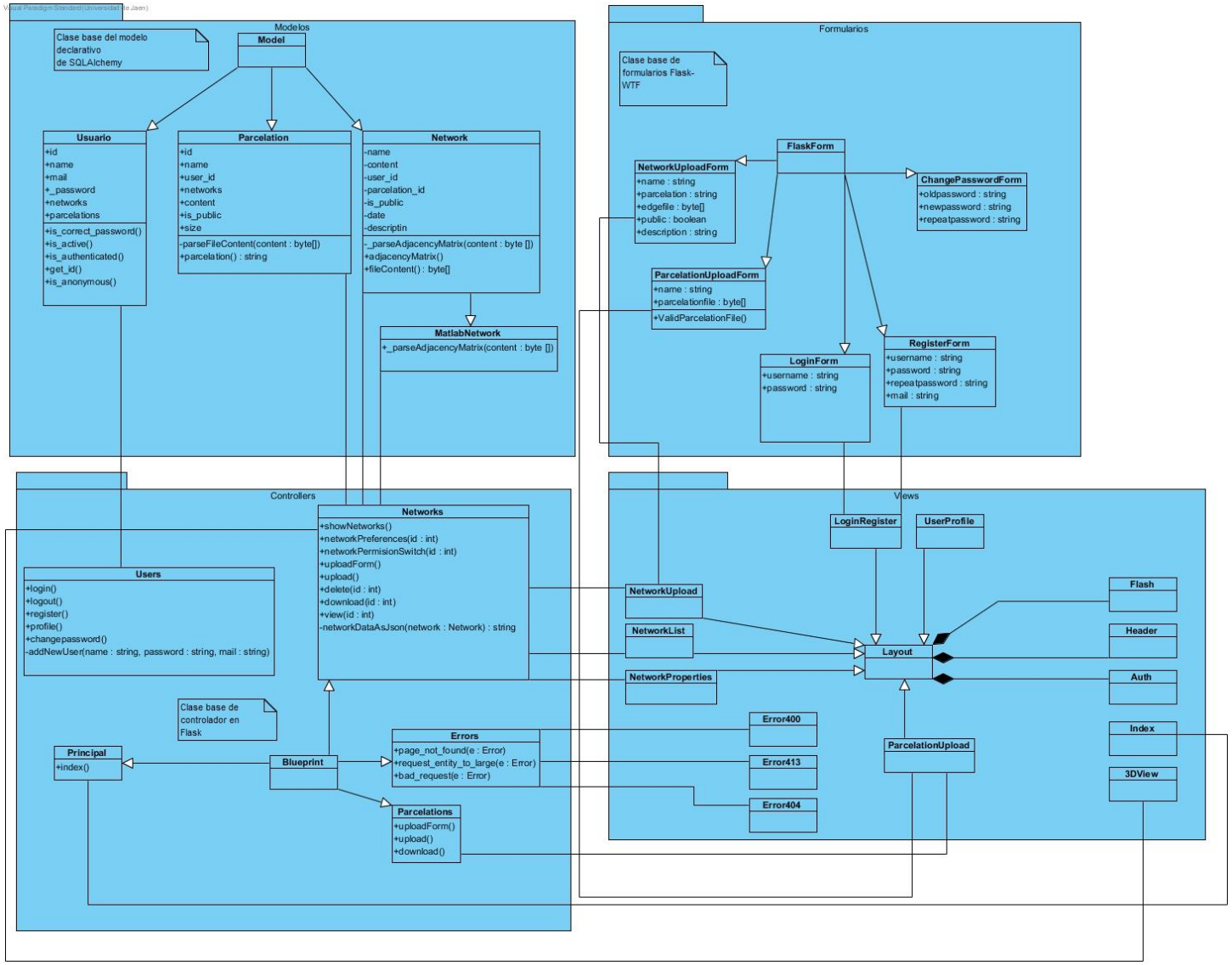


Figura 4.7.3 Visionado de red cerebral mostrando opciones de estética de red.

Diagramas de la aplicación




Modelos
Clase base del modelo declarativo de SQLAlchemy
Model
Usuario
+id
+name
+mail
+_password
+networks
+parcelations
+is_correct_password()
+is_active()
+is_authenticated()
+get_id()
+is_anonymous()
Parcelation
+id
+name
+user_id
+networks
+content
+is_public
+size
-parseFileContent(content : byte[])
+parcelation() : string
Network
-name
-content
-user_id
-parcelation_id
-is_public
-date
-descriptin
-_parseAdjacencyMatrix(content : byte [])
+adjacencyMatrix()
+fileContent() : byte[]
MatlabNetwork
+_parseAdjacencyMatrix(content : byte [])
Formularios
Clase base de formularios Flask-WTF
FlaskForm
NetworkUploadForm
+name : string
+parcelation : string
+edgefile : byte[]
+public : boolean
+description : string
ChangePasswordForm
+oldpassword : string
+newpassword : string
+repeatpassword : string
ParcelationUploadForm
+name : string
+parcelationfile : byte[]
+ValidParcelationFile()
LoginForm
+username : string
+password : string
RegisterForm
+username : string
+password : string
+repeatpassword : string
+mail : string
Controllers
Networks
+showNetworks()
+networkPreferences(id : int)
+networkPermisionSwitch(id : int)
+uploadForm()
+upload()
+delete(id : int)
+download(id : int)
+view(id : int)
-networkDataAsJson(network : Network) : string
Users
+login()
+logout()
+register()
+profile()
+changepassword()
-addNewUser(name : string, password : string, mail : string)
Clase base de controlador en Flask
Principal
+index()
Blueprint
Errors
+page_not_found(e : Error)
+request_entity_to_large(e : Error)
+bad_request(e : Error)
Parcelations
+uploadForm()
+upload()
+download()
Views
LoginRegister
UserProfile
NetworkUpload
NetworkList
NetworkProperties
Layout
Flash
Header
Auth
Index
3DView
Error400
Error413
Error404
ParcelationUpload

Diseño de la aplicación

Una vez presentados los diagramas de clases, procedemos a discutir los aspectos de diseño de la aplicación. Conste que este paso se realiza cronológicamente después de la elección de bibliotecas y software auxiliar a utilizar para desarrollar el proyecto, por lo que en el diseño se tienen en cuenta los aspectos intrínsecos de dichas herramientas.

Parte cliente

Fundamentalmente hay dos clases de información en la parte cliente. Las redes de datos y los datos de geometría para representarla. Las redes se codifican en dos clases distintas. La clase Network y la clase Graph. La clase Network será la encargada de recuperar los datos desde el servidor, implementando la lógica de las peticiones HTTP asíncronas. Una vez recuperada, se encarga de la lectura de los datos y de crear el grafo correspondiente. El grafo puede ser de tipo asimétrico o de tipo simétrico. Cada uno de los casos requiere representaciones gráficas diferentes, para lo que hacemos uso de la herencia, creando subclases. En todo momento se respeta el principio de sustitución de Liskov (Wikipedia l. e., 2016) siendo posible sustituir la una por la otra en cualquier momento sin necesidad de adaptar la lógica del resto de clases.

La clase Network se encarga de generar la geometría necesaria para la representación de los nodos y de almacenar los datos relativos a los mismos. La clase Network hace lo propio con la información de las redes. Ambas almacenan referencias a los objetos creados. La clase Graph mediante un array y la clase Network mediante una matriz homeomorfa a la matriz de adyacencia. Esto facilita la posterior aplicación de filtros para la visualización, permitiendo implementarlos como un bucle que recorre un array en lugar de recurrir a métodos más complejos. La clase Network se encarga de la creación de la clase Graph correspondiente y no tienen sentido la una sin la otra, representando una asociación por composición.

Una vez completadas dichas operaciones, se notifica a la clase aplicación para que sea esta las que incluya las referencias en la escena. El diseño rompe con el esquema clásico del patrón MVC en el que se usa el patrón observador para notificar a la vista de cambios en el modelo por dos razones. La primera, aprovechamos dos propiedades del lenguaje, el hecho de que todos los objetos sean tratados como referencias y la habilidad nativa para trabajar con eventos. La segunda y principal, el renderizado de la escena se ejecuta en el bucle principal de la aplicación. La vista necesita acceder al modelo de datos constantemente para realizar el

renderizado, por lo que incluyendo una referencia directa de esta a la clase escena ahorramos la sobrecarga asociada a las distintas llamadas a funciones necesarias en el patrón observador y conseguimos un renderizado más fluido. Uno de los objetivos de las aplicaciones gráficas es que la tasa de redibujado de la pantalla sea lo más próximo posible a 60 cuadros por segundo, por lo que necesitamos optimizar lo máximo posible la lógica de renderizado. El empleo de canales de publicación / subscripción ayuda a desacoplar el diseño de las clases, evitando tener que mantener referencias entre ellas lo máximo posible.

No obstante, cuando la notificación no requiere la urgencia de las operaciones de renderizado, si se sigue el esquema clásico MVC. Por ejemplo, cuando un usuario selecciona un nodo para ver información detallada sobre el mismo, el controlador informa al modelo de este evento y este notifica a la vista para que actualice el campo destinado a la información. A pesar de este comportamiento, seguimos usando el esquema publicación / subscripción.

El resto de la aplicación difiere poco de una arquitectura MVC clásica. La clase dedicada a controlar la vista, RendererView, se encarga de modificar el DOM del documento para incluir las etiquetas HTML necesarias, a la vez que manejan la entrada del usuario y notifican a los controladores. Los controladores se encargan de modificar el modelo de datos en base a las notificaciones recibidas por la vista.

Parte servidor

El servidor sigue la interpretación del patrón MVC implementada por los creadores del framework. El modelo de datos es la parte más ortodoxa del diseño, más allá de las herencias de las clases con las clases base del ORM. Las operaciones de parseo y serialización de los ficheros proporcionados por el usuario los realiza el modelo. Diseñamos una subclase de la clase Network, para que el sistema pueda trabajar con ficheros Matlab además de con ficheros de texto convencionales.

Los controladores de la aplicación no se implementan como clases y por tanto no hay herencia, si no que se utilizan módulos Python y la técnica de decoradores en funciones. Aun así, decidimos representarlos como clases en nuestro diagrama.

Algo parecido pasa con las vistas, nuevamente, no se implementan como clases si no que se definen en plantillas HTML con un lenguaje propio incrustado.

Los controladores responden a las peticiones HTTP que realiza el usuario, carga los datos desde el modelo, que se encarga a su vez de cargar los datos desde la base de datos, y con estos genera un documento HTML final que se envía al usuario en la respuesta.

El controlador encargado de las redes es el más grande en lo que respecta a las líneas de código, debido a que la mayoría de las operaciones en la aplicación están relacionadas con las redes. Se decide que la operación de deserialización de las redes (es decir, recuperar los datos del modelo y construir una respuesta con la información de la red como de la parcelación) se realice en el controlador.

Diagramas de secuencia

Seguidamente mostramos los diagramas de secuencia de la operación. Estos representan en una escala temporal el intercambio de mensajes entre distintos elementos de la aplicación, proporcionando una excelente manera de entender el flujo de datos entre los distintos elementos de la aplicación.

Ver red (parte servidor)

En la figura 4.10 se muestra la operación que realiza la parte servidor cuando recibe del usuario una petición HTTP para visualizar una red.

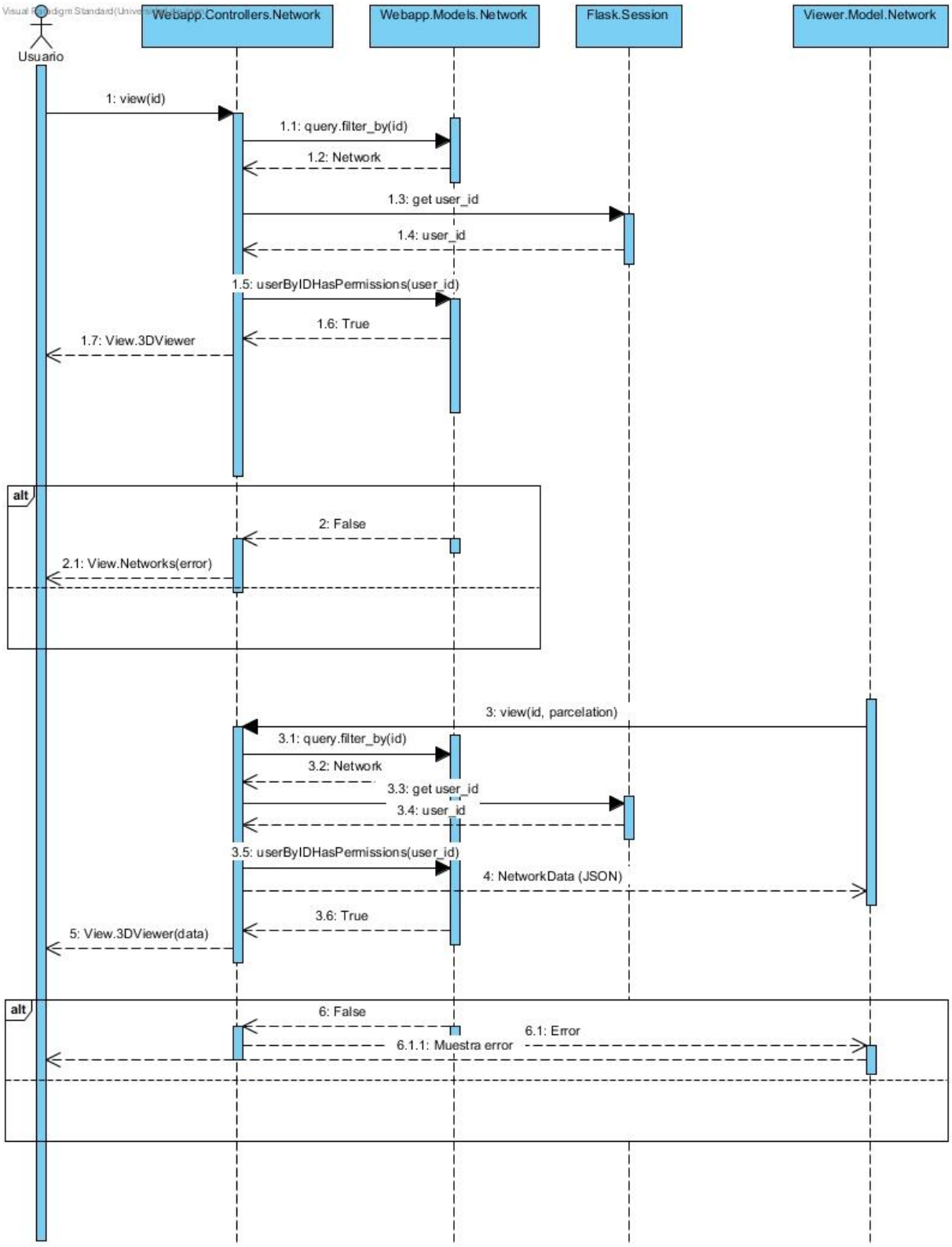


Figura 4.10. Diagrama de secuencia de la operación "Ver Red" en la parte servidor.

Una vez que el controlador registra la petición, consulta los permisos del usuario para visualizar la red en cuestión. Si los tiene, procede a la carga del documento HTML junto con el

código Javascript de la aplicación de parte servidor. Una vez que ésta ha cargado, inicia automáticamente una nueva petición, esta vez para recuperar los datos codificados en formato JSON. En este paso también se repiten las comprobaciones de seguridad para evitar posibles problemas de filtración de datos.

Ver red (parte cliente)

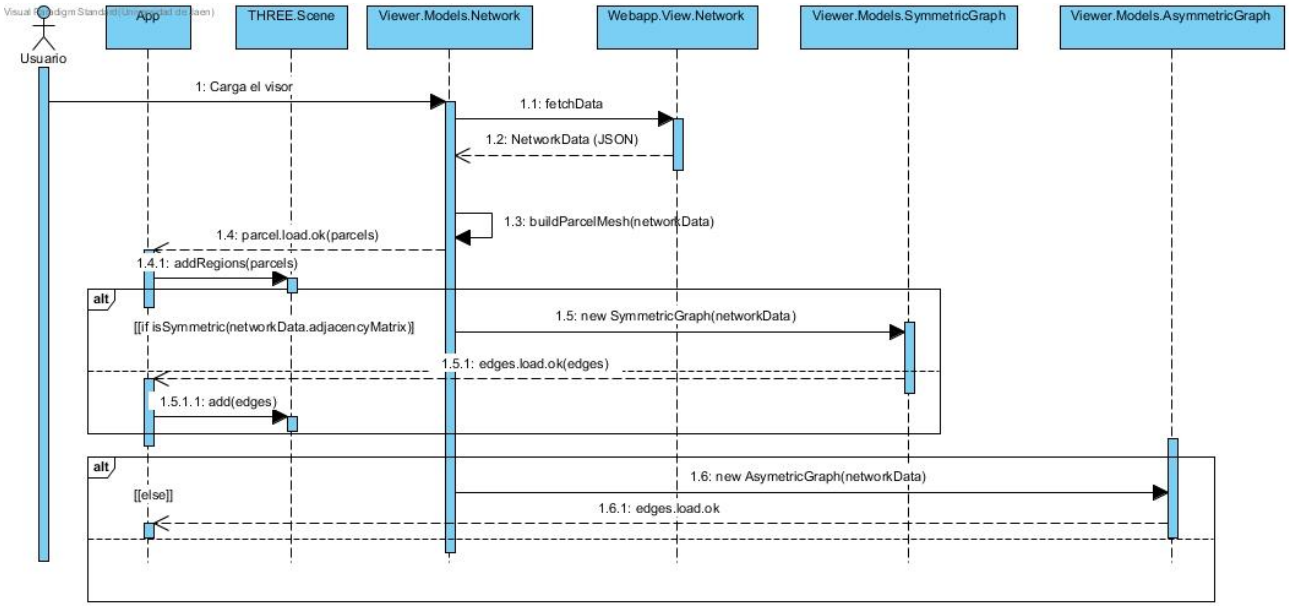


Figura 4.11. Diagrama de secuencia de la operación “Ver Red” en la parte cliente.

Cuando la parte cliente recibe la información de la red, comienza a procesarla (ver figura 4.11). En primer lugar, la clase Network se encarga ella misma de generar la geometría. Una vez que esta se ha generado, se notifica a la aplicación para que ésta añada las regiones a la escena. Una vez concluido el proceso, se presentan dos alternativas. Si la matriz de adyacencia del grafo representa una matriz asimétrica, se crea un objeto de la clase SymmetricGraph. Por el contrario, si la matriz es asimétrica, el objeto será de la clase AsymmetricGraph. En ambos casos, y por el principio de substitución de Liskov, el comportamiento de las clases será el mismo. Estas generarán la geometría correspondiente y notificarán a la aplicación.

Ya se concluyan las operaciones de manera satisfactoria o se produzca un error, la notificación a la aplicación siempre será mediante el empleo de canales publicación /subscripción

Filtrado de nodos

En la figura 4.12 se muestra el diagrama de secuencia correspondiente al filtrado por peso. El resto de operaciones de filtrado son en esencia de la misma forma, solo que se cambia el mensaje que se envía entre los objetos.

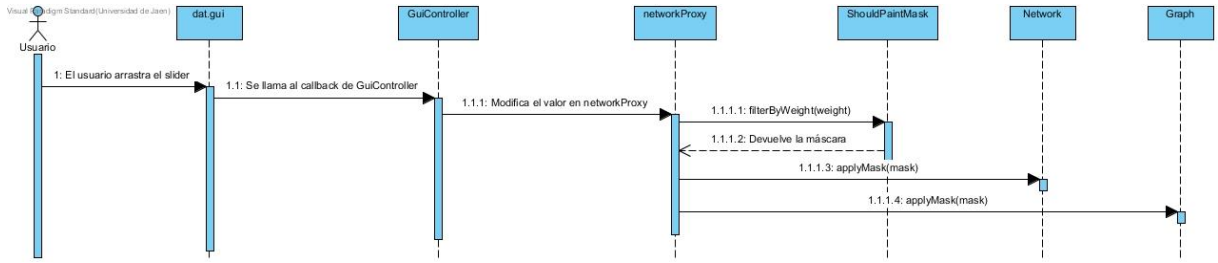


Figura 4.12. Diagrama de secuencia de la operación "Añadir Filtro"

La secuencia comienza con el usuario generando un evento recogido por dat.gui. Esta notifica al controlador de la interfaz gráfica que comienza la modificación de datos. Debido a las limitaciones de tipos de datos con los que dat.gui puede operar, en lugar de modificar directamente el modelo, hace uso de una clase que actúa como intermediario entre ellas. Cuando el proxy recibe el mensaje, genera una nueva máscara indicando que nodos se deben o no pintar, usando la clase ShouldPaintMask. Una vez generada la máscara, se envía a los modelos de red y grafo para que la apliquen. Estos modifican las geometrías para cambiar su visibilidad. Aprovechando que el modelo de memoria de Javascript se basa enteramente en referencias, estos cambios se ven automáticamente reflejados en el objeto escena (ya que técnicamente, se están modificando los mismos objetos) no siendo necesaria la notificación de la vista para que esta se actualice.

Mostrar información de un elemento.

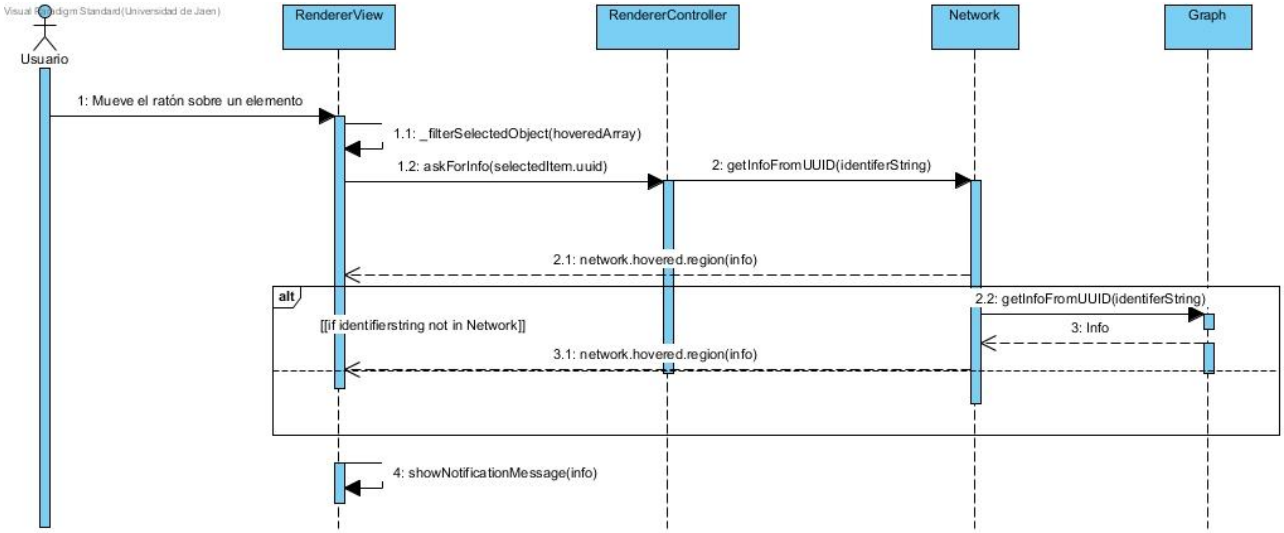


Figura 4.13. Diagrama de secuencia de la operación "Mostrar Información"

Cuando el usuario coloca el ratón sobre un elemento, se muestra en la pantalla información sobre el mismo (ver figura 4.13). Esta información está almacenada en el correspondiente modelo de datos. En primer lugar, es necesario identificar qué objeto se ha seleccionado. Dado que el proceso requiere de cálculo de colisiones, raytracing y otras operaciones ligadas a la biblioteca de manipulación 3D, dejamos esta operación al cargo de la vista. Una vez se ha identificado, se notifica al controlador que, a su vez, notifica al modelo. El receptor del mensaje es la clase Network, que comprueba si se trata de uno de sus modelos geométricos. Si no, pasa el mensaje a la clase Graph, aprovechando la relación de composición entre ambas. Una vez encontrado, se envía el mensaje de vuelta a la vista, notificándola para que actualice el documento.

Añadir nueva red

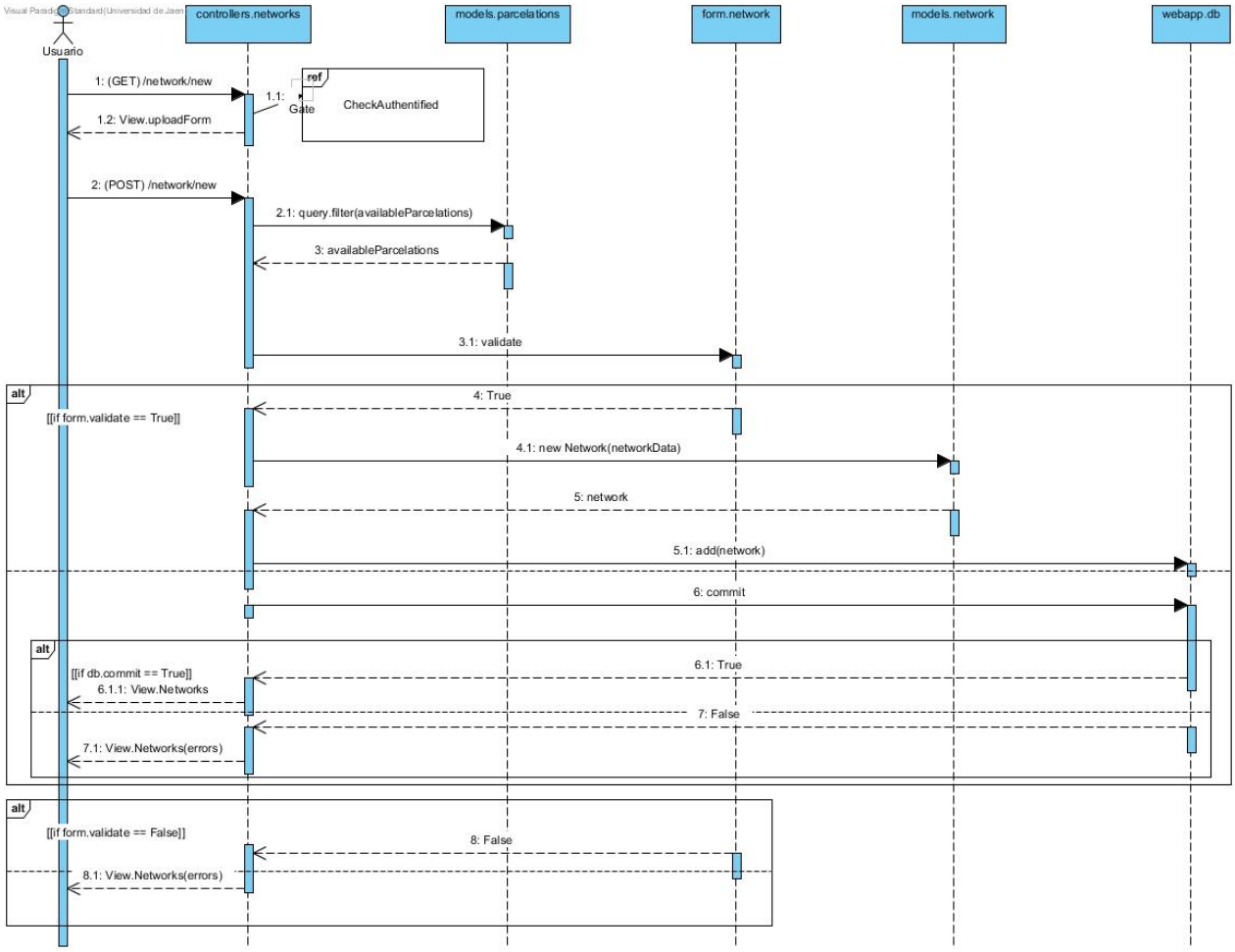


Figura 4.14. Diagrama de secuencia de la operación "Añadir Nueva Red"

Al intentar crear una nueva red (ver figura 4.14), el usuario solicita el documento de formulario mediante una petición GET. El servidor comprueba que se trata de un usuario autenticado y de ser así, le envía el documento con el formulario de subida. Posteriormente, se envía el formulario relleno usando una petición POST. Los datos pasan por el proceso de validación y si este es satisfactorio, se crea el objeto Network y se incluye en la base de datos, de lo contrario, se informa al usuario de los errores ocurridos durante la validación. Si se produce un error durante la inclusión del objeto en la base de datos, se notifica al usuario. El proceso para añadir una parcelación es prácticamente el mismo, por lo que se omite el diagrama.

Autenticación de usuario

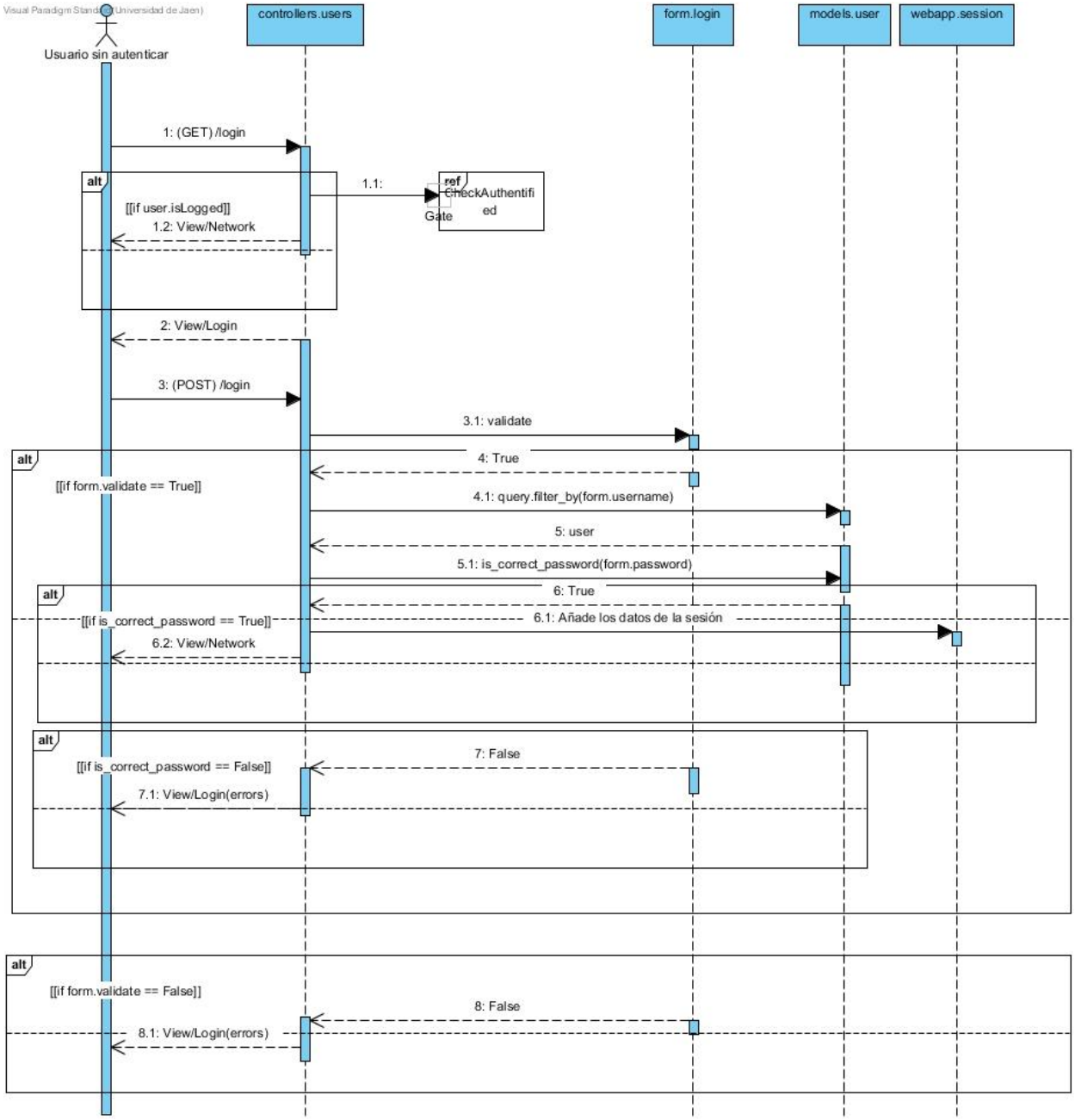


Figura 4.15. Diagrama de secuencia de la operación "Autenticación de usuario".

Como se muestra en la figura 4.15 el proceso de autenticación comienza cuando el usuario lanza una petición GET a la URL especificada. El servidor comprueba si el usuario ya está autenticado. Si lo está, lo redirige al listado de redes. Si no, sirve el documento. Al recibir los datos mediante POST, valida el formulario y presenta al usuario los errores en caso de producirse. Si no, carga el objeto usuario desde el modelo y comprueba que la contraseña sea

la correcta. Si la contraseña indicada en el formulario coincide con la almacenada en la base de datos, se crean los datos de sesión y se redirige al usuario a la lista de redes.

Registro de un nuevo usuario

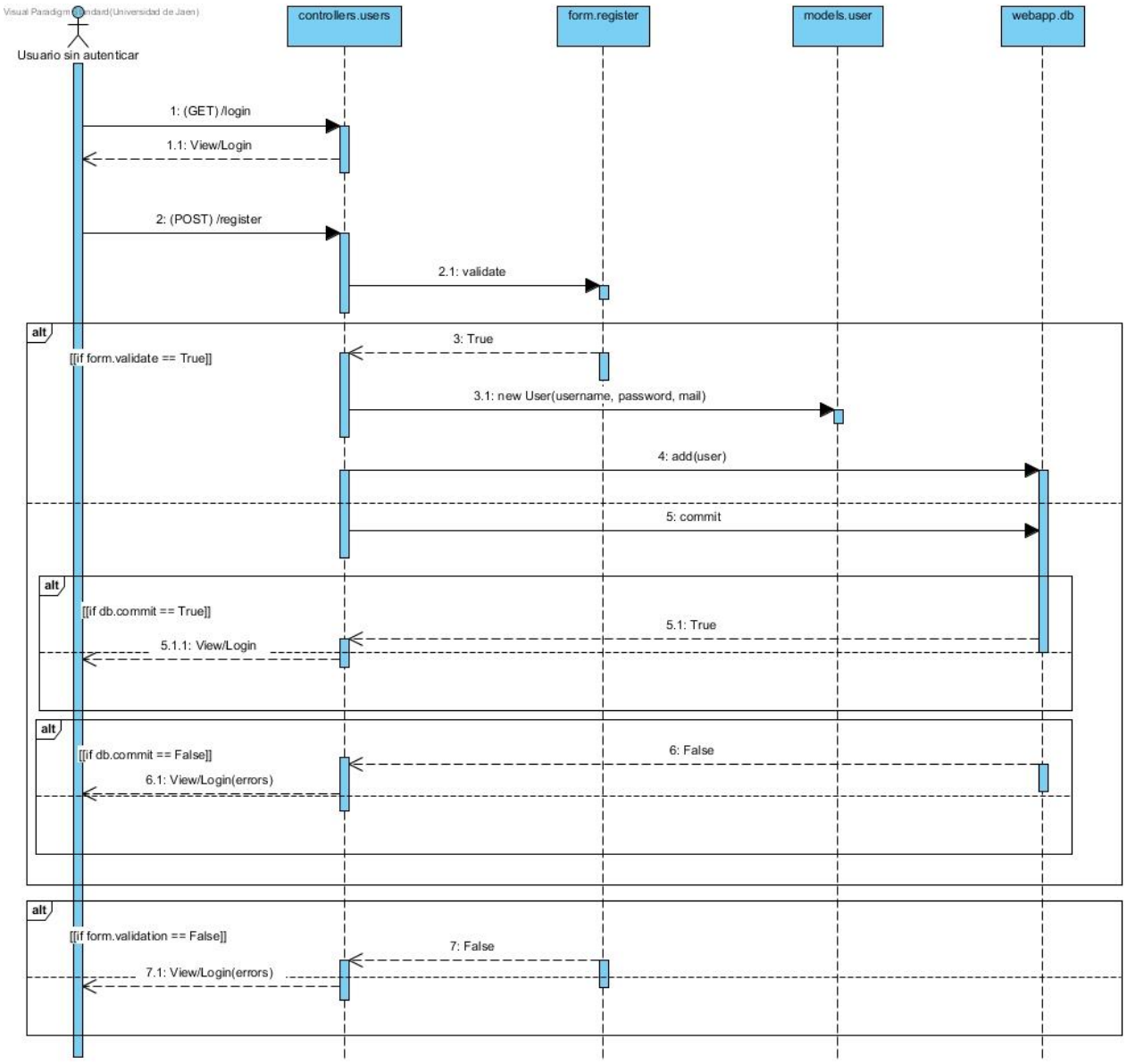


Figura 4.16. Diagrama de secuencia de la operación “Registro de Nuevo Usuario”.

El proceso de registro comienza cuando el usuario envía una petición GET al controlador de gestión de usuarios (ver figura 4.16). Este envía el documento HTML que contiene el formulario de autenticación y registro. Una vez rellenos los campos, se envía una petición POST con los datos del mismo. El servidor valida los datos del formulario y si encuentra algún error, envía nuevamente el documento HTML de registro junto con el informe de fallos de validación. Si no, crea el objeto usuario e intenta añadirlo a la base de datos. Si esta última operación genera

algún error, se notifica al usuario, de lo contrario, se informa al usuario de que el usuario se ha creado con éxito y se le pide que realice el proceso de autenticación con las nuevas credenciales creadas.

5. Implementación

Pasamos a discutir detalles propios de la implementación realizada.

Tecnologías usadas y su justificación

Plataforma

La principal razón contra el uso de aplicaciones de escritorio tipo QT es que, si bien presenta abstracciones y clases para trabajar con redes, requiere un esfuerzo añadido a la hora de diseñar y codificar la aplicación. Al usar un sistema web, podemos abstraer los detalles de implementación de este código, con la certeza de que el distinto software en el que se puede ejecutar nuestra aplicación estará probado y testado en estos aspectos, permitiendo centrarnos en la implementación de la propia aplicación. (JavaScript performance updates in Microsoft Edge and Chakra, 2016)

Entre los contras al sistema web pesa sobre todo el rendimiento. Al poner una capa extra entre nuestro código y el hardware (en este caso el navegador) empeoraremos el rendimiento, sin embargo, la popularización de estas tecnologías ha obligado a los fabricantes a poner especial énfasis en el rendimiento. Si bien no puede competir contra los lenguajes compilados, supone una mejora frente a otros lenguajes compilados. (Debian Project, s.f.)

Una vez nos hemos decantado por desarrollar una aplicación web, exploramos las distintas opciones de arquitectura.

Arquitectura de la aplicación

Analizando los distintos pros y contras de cada una de las arquitecturas, decidimos usar una arquitectura clásica cliente/servidor, que se encargará de los aspectos más delicados de implementar (como la autenticación del usuario) mientras que en la parte cliente implementaremos toda la lógica necesaria para la visualización tridimensional propiamente dicha.

Software para el backend

Se decide descartar PHP a pesar de ser una alternativa sencilla y de bajo costo, sin embargo, no cuenta con mucho apoyo entre la comunidad, muchos de los ejemplos y documentación están desactualizados y el conjunto de herramientas para trabajar con él está algo desactualizado, salvo excepciones de pago como PHPStorm).

El rol de nuestro backend es mínimo en comparación con el esfuerzo que supone la implementación del frontend. Buscamos por tanto un programa pequeño, que implemente la funcionalidad básica pero que se preste a la extensión futura. Los frameworks de mayor tamaño, Rails y Django, presentan demasiada rigidez en este sentido además de una curva de aprendizaje muy marcada.

Las alternativas que nos quedan son pues Go y Flask. Go, al tratarse de un lenguaje compilado presenta un rendimiento superior. Su diseño sin embargo lo hace más propicio a aplicaciones minúsculas, como parte de una infraestructura de micro servicios. Tratar de implementar una aplicación algo más compleja en Go es una tarea más complicada. No existen frameworks al uso, todo se construye en base a las bibliotecas estándar del sistema, que permiten la manipulación de peticiones y respuestas HTTP a muy bajo nivel, debiendo el programador implementar el resto de mecanismos, como sesiones, acceso a bases de datos, etc. Flask por el otro lado presenta un equilibrio perfecto entre ambos mundos. Su arquitectura modular lo hace extensible y ligero, pero tiene suficientes herramientas y extensiones oficiales del framework para construir aplicaciones de complejidad moderada. A pesar de encontrarse todavía en versión beta, cuenta con una comunidad estable y una gran cantidad de herramientas auxiliares para la construcción de una web. Python, además, posee una extensa cantidad de bibliotecas, tanto incluidas en el propio lenguaje (manejo de ficheros CSV) como añadidas por la comunidad como manipulación de ficheros Matlab, bibliotecas de operaciones matemáticas orientadas a gráficos por ordenador y un largo etcétera, que pueden ser de ayuda durante el desarrollo, implementando funcionalidad usando sistemas ya testados por otros usuarios.

Sistemas de gestión de bases de datos

A pesar de la enorme influencia que MySQL ha tenido y tendrá, la mayoría de la comunidad mira de manera positiva a PostgreSQL. En nuestro caso las ventajas de replicación

de MySQL no suponen un punto a su favor, porque las previsiones de uso de nuestra aplicación no parecen indicar que vaya a tener ese volumen de uso. Los tipos de datos nativos para JSON de PostgreSQL son una cualidad de la que nuestra aplicación puede sacar provecho.

Software para el frontend

Una parte importante de la funcionalidad de nuestro sistema se implementará de lado del servidor, por lo que debemos centrar nuestros esfuerzos en la parte gráfica de nuestra herramienta. Emscripten proporciona un buen rendimiento, siendo la opción elegida por motores gráficos como Unreal Engine o Unity para compilar versiones web de sus aplicaciones. Su contrapunto está en la complejidad subyacente. Crear una aplicación usando Emscripten supone implementar y programar una aplicación OpenGL desde cero, debiendo de diseñar, implementar y probar todo un conjunto de clases y módulos para el funcionamiento de la misma. Three.js proporciona un punto de partida muy sólido, dándonos estructuras de datos preparadas para trabajar con WebGL sin implementar mucho código, con un sistema cohesionado y con apoyo de la comunidad.

Para la interfaz nos decantamos por dat.gui. jQuery nos obligaría a añadir más código en forma de formularios HTML para poder desarrollar una interfaz, mientras que con dat.gui, expresamos el aspecto y funcionalidad de manera declarativa y es este software el que se encarga de incrustar y manejar el código necesario para su funcionamiento.

Usaremos Webpack para la gestión de nuestros assets. Nos obliga a que nuestra infraestructura soporte un lenguaje extra (NodeJS) sin embargo, la popularidad del mismo no supondrá un obstáculo para dicho fin y no podemos obviar las múltiples ventajas que nos proporciona la herramienta en todas las etapas del desarrollo.

Despliegue / infraestructura

AWS proporciona la herramienta más flexible de todas, pero su uso y correcta configuración lleva asociado un aprendizaje y una familiarización con sus términos y particular forma de trabajar. Supone además tener que realizar operaciones de administración de sistemas, al ser necesaria la instalación y configuración de los servicios que se van a usar. Su opción

gratuita es bastante limitada en cuanto a potencia. Las herramientas que proporciona AWS como su versión gestionada de base de datos relacional eliminarían la necesidad de tareas de administración de sistemas. El principal problema de este planteamiento es que, aunque basadas en herramientas de código abierto, las APIs de dichas herramientas no suelen ser compatibles con herramientas estándar. Pasamos de construir una aplicación web a construir una aplicación para AWS.

Heroku y OpenShift vienen a cubrir necesidades muy parecidas y ofrecen soluciones muy solapadas. Heroku cuenta con una documentación más rica y más soporte de la comunidad, además se añade Dokku, que nos permite hacer pruebas y despliegues en local para acostumbrarnos a la manera de trabajar del sistema. Toda la configuración de los servicios la realiza la plataforma, ahorrándonos las tareas de administración y permitiendo un perfil DevOps. Su opción gratuita solo ofrece una opción para base de datos basada en PostgreSQL.

Serialización de datos de redes

Uno de los problemas principales de la aplicación es cómo almacenar los datos de las redes del usuario para su posterior uso. En principio, podríamos simplemente almacenar los ficheros enviados en el sistema de archivos local del usuario. Esto plantea dos problemas. El primero, el de la seguridad, pues estamos proporcionando a un usuario potencialmente hostil la posibilidad de escribir datos aleatorios en nuestro sistema de ficheros. El segundo, el del espacio en disco y persistencia en plataformas PaaS. La mayoría de entornos de este tipo usan tecnologías de contenedores para sus despliegues (Heroku, 2016). Esta tecnología no proporciona persistencia en los sistemas de ficheros de la aplicación, provocando así la pérdida de datos cada vez que el contenedor se descargue del sistema (comportamiento habitual en el modelo gratuito de Heroku). Debido a estos dos factores, decidimos almacenar los datos en nuestro DBMS.

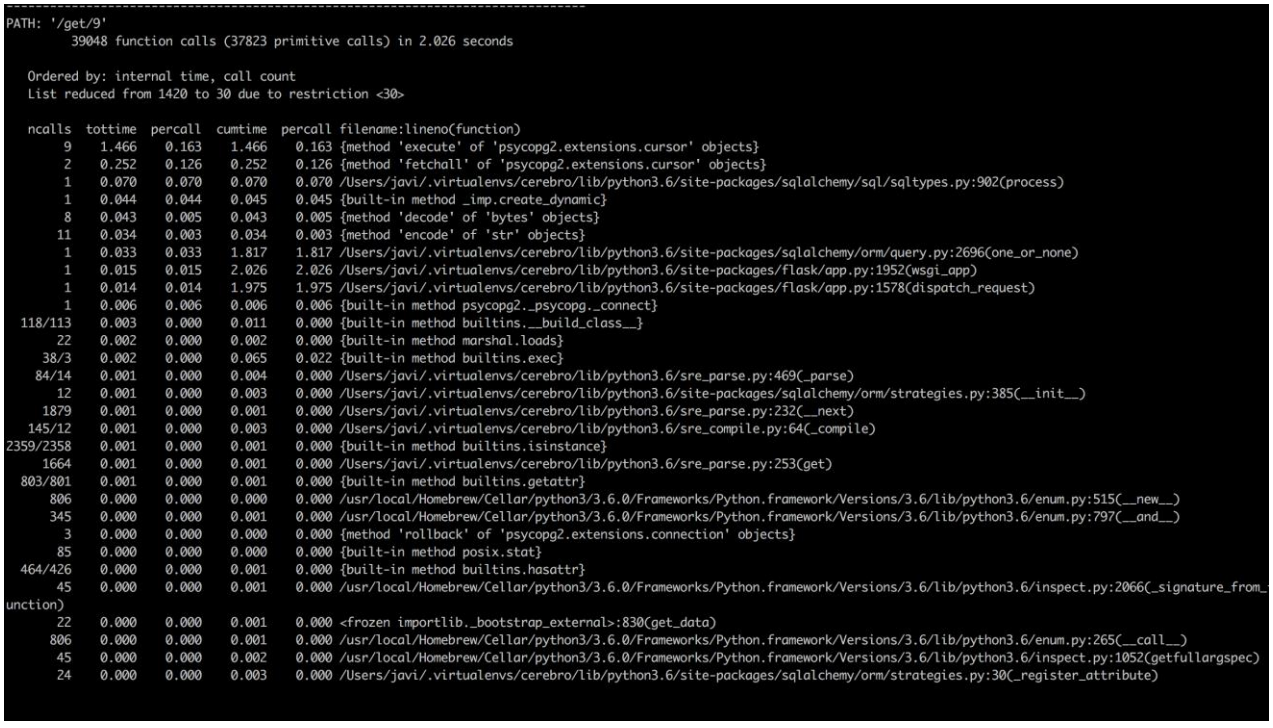
La opción más válida a primera vista sería cargar los datos en un campo de tipo TEXT de la base de datos. Durante las pruebas, se descubrió que, con las redes de gran tamaño, el volumen de datos es superior al que PostgreSQL puede almacenar en este tipo de campo.

La siguiente opción es usar un campo BLOB. Este tipo de campos sí están pensados para el almacenamiento de grandes volúmenes de datos, pero como inconveniente, requiere cierta

transformación y codificación previa. Para mejorar el rendimiento de la aplicación, procedemos a estudiar los distintos métodos de compresión y a comentar sus ventajas y desventajas.

La primera opción sería almacenar los datos tal cual están en el fichero, leyendo el mismo y almacenando su contenido tras convertirlo a bytes. La principal desventaja de este planteamiento es que, ante cada petición del usuario, habría que parsear y codificar los datos a formato JSON. Modificar el cliente para trabajar con la codificación propia de los ficheros introduce complejidad en el código. Manteniendo la notación de JSON nos aseguramos de que la lógica del parser en el lado cliente funcione, al ser ejecutada en un navegador con millones de usuarios en todo el planeta, es más probable que se encuentren y notifiquen errores. Por otro lado, mejora el rendimiento. Debido a que las operaciones con datos JSON son muy frecuente en las aplicaciones web, el código de parseo de JSON suele estar fuertemente optimizado.

La segunda opción se basa en JSON directamente. Al cargar el fichero, se serializan los datos usando dicho formato y se almacena la cadena resultante en formato binario. Para atender a las peticiones de los usuarios, solo tendríamos que recuperar la información desde la base de datos y codificarla en formato UTF-8. Para construir el objeto completo, concatenaríamos las distintas cadenas JSON para construir la respuesta.



```
PATH: '/get/9'
         39048 function calls (37823 primitive calls) in 2.026 seconds

   Ordered by: internal time, call count
   List reduced from 1420 to 30 due to restriction <30>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        9    1.466    0.163    1.466    0.163 {method 'execute' of 'psycopg2.extensions.cursor' objects}
        2    0.252    0.126    0.252    0.126 {method 'fetchall' of 'psycopg2.extensions.cursor' objects}
        1    0.070    0.070    0.070    0.070 /Users/javi/.virtualenvs/cerebro/lib/python3.6/site-packages/sqlalchemy/sql/sqltypes.py:902(process)
        1    0.044    0.044    0.045    0.045 {built-in method _imp.create_dynamic}
        8    0.043    0.005    0.043    0.005 {method 'decode' of 'bytes' objects}
       11    0.034    0.003    0.034    0.003 {method 'encode' of 'str' objects}
        1    0.033    0.033    1.817    1.817 /Users/javi/.virtualenvs/cerebro/lib/python3.6/site-packages/sqlalchemy/orm/query.py:2696(one_or_none)
        1    0.015    0.015    2.026    2.026 /Users/javi/.virtualenvs/cerebro/lib/python3.6/site-packages/flask/app.py:1952(wsgi_app)
        1    0.014    0.014    1.975    1.975 /Users/javi/.virtualenvs/cerebro/lib/python3.6/site-packages/flask/app.py:1578(dispatch_request)
        1    0.006    0.006    0.006    0.006 {built-in method psycopg2._psycopg._connect}
  118/113    0.003    0.000    0.011    0.000 {built-in method builtins.__build_class__}
       22    0.002    0.000    0.002    0.000 {built-in method marshal.loads}
     38/3    0.002    0.000    0.065    0.022 {built-in method builtins.exec}
    84/14    0.001    0.000    0.004    0.000 /Users/javi/.virtualenvs/cerebro/lib/python3.6/sre_parse.py:469(_parse)
       12    0.001    0.000    0.003    0.000 /Users/javi/.virtualenvs/cerebro/lib/python3.6/site-packages/sqlalchemy/orm/strategies.py:385(__init__)
     1879    0.001    0.000    0.001    0.000 /Users/javi/.virtualenvs/cerebro/lib/python3.6/sre_parse.py:232(__next)
   145/12    0.001    0.000    0.003    0.000 /Users/javi/.virtualenvs/cerebro/lib/python3.6/sre_compile.py:64(_compile)
2359/2358    0.001    0.000    0.001    0.000 {built-in method builtins.isinstance}
     1664    0.001    0.000    0.001    0.000 /Users/javi/.virtualenvs/cerebro/lib/python3.6/sre_parse.py:253(get)
  803/801    0.001    0.000    0.001    0.000 {built-in method builtins.getattr}
      806    0.000    0.000    0.000    0.000 /usr/local/Homebrew/Cellar/python3/3.6.0/Frameworks/Python.framework/Versions/3.6/lib/python3.6/enum.py:515(__new__)
      345    0.000    0.000    0.001    0.000 /usr/local/Homebrew/Cellar/python3/3.6.0/Frameworks/Python.framework/Versions/3.6/lib/python3.6/enum.py:797(__and__)
        3    0.000    0.000    0.000    0.000 {method 'rollback' of 'psycopg2.extensions.connection' objects}
       85    0.000    0.000    0.000    0.000 {built-in method posix.stat}
  464/426    0.000    0.000    0.001    0.000 {built-in method builtins.hasattr}
       45    0.000    0.000    0.001    0.000 /usr/local/Homebrew/Cellar/python3/3.6.0/Frameworks/Python.framework/Versions/3.6/lib/python3.6/inspect.py:2066(_signature_from_
unction)
       22    0.000    0.000    0.001    0.000 <frozen importlib._bootstrap_external>:830(get_data)
      806    0.000    0.000    0.001    0.000 /usr/local/Homebrew/Cellar/python3/3.6.0/Frameworks/Python.framework/Versions/3.6/lib/python3.6/enum.py:265(__call__)
       45    0.000    0.000    0.002    0.000 /usr/local/Homebrew/Cellar/python3/3.6.0/Frameworks/Python.framework/Versions/3.6/lib/python3.6/inspect.py:1052(getfullargspec)
       24    0.000    0.000    0.003    0.000 /Users/javi/.virtualenvs/cerebro/lib/python3.6/site-packages/sqlalchemy/orm/strategies.py:30(_register_attribute)
```

Figura 5.1. Resultado del análisis de rendimiento mediante profiling.

Como se muestra en la figura 5.1 usando el módulo cProfiler de Python, vemos que se trata de la opción más acertada en cuanto a rendimiento, siendo capaz de generar la respuesta en 2,026 segundos para una red con 4000 nodos.

Detalles de implementación

En este apartado discutimos los detalles de la implementación.

El proyecto presenta dos partes diferenciadas, la parte de software que se ejecuta en el servidor, en adelante backend, y la parte que se ejecuta en el cliente, en adelante frontend.

Parte cliente

Para el desarrollo del frontend usamos Webpack como herramienta principal. El estándar del lenguaje a usar será ES2015, usando Babel para compilarlo a una versión que soporten los distintos navegadores. Diseñaremos la aplicación usando el patrón MVC.

La clase aplicación se encarga de crear las instancias de los objetos necesarios para el funcionamiento del programa. Estos serán, la vista del render, el controlador del render, el controlador del GUI, el controlador de movimiento y los modelos. El GUI usa la biblioteca dat.gui, por lo que genera y configura automáticamente la vista, debiendo solo programar los callbacks en el controlador.

Al iniciar la aplicación, el modelo de datos iniciará automáticamente la recuperación de los datos de la red codificados en JSON, así como la malla de triángulos que representa el cerebro. Usaremos la API Fetch (Foundation M. , 2016) para descargar los datos. Este método proporciona una interfaz basada en Promises en lugar de la interfaz clásica de AJAX basada en callbacks, lo que nos permite programar las descargas de manera asíncrona, mejorando la respuesta del software a la interacción del usuario. Una vez descargados, comenzaremos a parsear los datos creando las geometrías necesarias para la representación. Al completarse esta operación, se notificará a la aplicación mediante PubSub.js. Esta herramienta permite crear buses de eventos basados en el patrón Publicador / Subscriptor, más característica de Javascript que otros sistemas de notificación como podría ser el patrón Observador (Osmani, 2015).

La clase aplicación, al recibir la notificación, añadirá los datos de la geometría a la escena, con sus correspondientes etiquetas. Los modelos correspondientes almacenan los objetos originales, mientras que la escena tiene referencias a los mismos. Esto nos permite que el

modelo pueda modificar los datos y que estos cambios se reflejen en la escena. En las aplicaciones MVC clásicas se hace uso del patrón observador para que la vista se actualice cuando el modelo cambia. En nuestro caso, decidimos que la vista del render acceda directamente a la escena, manteniendo una referencia a la misma. Usar un patrón de getters y setters no proporciona mayor seguridad debido a la naturaleza del modelo de memoria de Javascript, además, debemos tener en cuenta que debemos mantener la tasa de cuadros dibujados lo más alta posible para que la visualización de los datos sea fluida, por lo que intentamos optimizar esta parte de la aplicación eliminando o reduciendo el número de funciones llamadas.

A pesar de ser una herramienta estéticamente cuidada, dat.gui proporciona widgets solo para manejar los tipos de datos más básicos. Debido a que se busca controlar tipos de datos complejos en nuestra aplicación, debemos construir una interfaz entre dicho controlador y el modelo que controla. Usamos para ello el patrón Proxy (Osmani, 2015) implementado mediante ES6. Crearemos dos proxys, uno para los datos de escena y otros para los datos de red. En ambos casos el funcionamiento será el mismo, el proxy expondrá tipos de datos simples con los que dat.gui puede trabajar, y se configuran disparadores que ejecutan la lógica compleja cuando esta interfaz cambie.

Para el filtrado selectivo de datos, usamos la clase ShouldPaintMask. Esta clase calcula unas máscaras, en función de los filtros aplicados, atendiendo a valores como el peso de las conexiones, el hemisferio o las conexiones de cada nodo. Al completar el cálculo de la máscara, esta se devuelve en formato de matriz binaria, en el que cada elemento representa si el nodo debe dibujarse o no. Esta matriz se envía a los modelos correspondientes (el de red y el de grafo) que la aplican sobre cada elemento, usando la propiedad “visible” de los objetos THREE. Aplicar la máscara en los vértices es inmediato, no así en los nodos, en los que solo se ocultarán los elementos si ninguno de los vértices que quedan visibles conectan con el nodo en cuestión. A continuación, se muestra el algoritmo usado para discriminar los nodos.

```
ApplyMask (mask: Number[][])
  PaintRegion := Bool[] = False
      For i in mask:
         For j in region:
            If mask[i][j] is not 0 and
            mask[j][i] is not 0
            Then
               PaintRegion[i] = True
            End If
         End for
      End for
Return PaintRegion
```

Parte servidor

Organizamos la aplicación usando blueprints, que en jerga del framework, vendrían a ser los controladores. Utilizamos diversas herramientas de extensión como Flask-Login para la gestión de sesiones y usuarios, Flask-Webpack para la integración con el código del frontend y Flask-SQLAlchemy para la gestión de la base de datos.

La configuración del sistema web se podrá realizar de dos maneras, o bien mediante un fichero de configuración, en el que se indicarán los distintos parámetros mediante un par clave valor, o mediante variables de entorno. Este último método nos facilita el despliegue en entornos PaaS, ya que, para modificar un fichero de texto, tendríamos que incluir el cambio en el repositorio de código, siendo potencialmente inseguro. Usando el sistema de variables de entorno podemos confinar valores secretos al entorno de ejecución de la aplicación.

Flask-SQLAlchemy es una extensión para el framework de SQLAlchemy, un ORM declarativo capaz de trabajar con multitud de bases de datos, abstrayendo los detalles propios a cada una. Esto nos permite modelar los datos de la aplicación como objetos Python y que sea

el propio sistema el encargado de realizar las operaciones contra la base de datos. Un problema que surge de este sistema, es que cuando el tamaño de las redes es muy elevado (en torno a los 2000 nodos) la carga de los objetos con la recuperación completa de los datos ralentiza mucho las operaciones. Indicamos en nuestro modelo declarativo que haga una carga en diferido de la columna de contenido lo que supone que, a la hora de recuperar redes desde la base de datos, no se recupere el campo de contenido hasta que este vaya a ser leído, evitando así cargar con el gran volumen de datos que supone.

Con respecto a los usuarios, almacenamos la contraseña de los mismos usando Bcrypt (Wikipedia, 2016) algoritmo de resumen (hashing) criptográficamente seguro basado en Blowfish, y que ha sido implementado en las distintas distribuciones BSD como algoritmo de contraseñas de sistema. Configuramos el algoritmo para que trabaje con doce rondas, valor recomendado por el propio framework.

Para la gestión de ficheros subidos por el usuario, creamos formularios con validadores personalizados usando Flask-WTF. Los validadores cargan el fichero recibido por el servidor en formato bytes, los transforma a una cadena, los mueve a un buffer sobre el que se permiten operaciones de entrada / salida y se posteriormente se parsean para comprobar que el fichero cumple con los criterios. El sistema espera entrada de ficheros en formato texto, con extensiones txt, csv o edge. En cualquier caso, el formato del fichero será CSV, con codificación UTF-8 y usando tabulador como carácter delimitador.

Para almacenar las redes, serializaremos los datos usando JSON y los convertiremos a formato binario para poder almacenarlos en un campo tipo BLOB. Serializamos los datos en JSON y los convertimos en formato binario. Cuando un usuario pide los datos al servidor, recuperamos los datos desde PostgreSQL, los convertimos a UTF-8 y componemos la respuesta mediante concatenación de cadenas.

6. Pruebas

En este capítulo describimos las distintas pruebas a las que hemos sometido el proyecto para verificar que se adhiere a los requisitos impuestos.

Durante el desarrollo, cobran especial importancia las pruebas unitarias o unit testing. Dichas pruebas están construidas con el fin de poner a prueba situaciones a las que el software

se puede enfrentar, como subida de ficheros mal formateados, intento de acceso a redes privadas, o valores fuera de rango en los filtros del visor. Son de ayuda a la hora de modificar o refactorizar el código, para comprobar que el comportamiento del software sigue siendo el esperado.

Pruebas unitarias

Con respecto a las pruebas unitarias tomamos dos actitudes diferenciadas según estemos probando el lado cliente o el lado servidor que describiremos a continuación. A pesar de no seguir una filosofía de desarrollo guiado por tests estricta, intentamos que, a la hora de introducir nueva funcionalidad, esta vaya acompañado de los tests pertinentes para cerciorarnos de su correcto funcionamiento.

En la parte servidor es más sencillo emular la acción del usuario mediante peticiones HTTP y analizar de manera programada la respuesta del servidor, al ser esta un documento HTML codificado en modo texto. Por eso, para realizar las pruebas unitarias en el servidor, tomaremos una filosofía más centrada en probar el comportamiento del software ante determinadas acciones. Creamos pruebas que simulen acciones cotidianas del usuario, ya sean válidas o no y analizamos la respuesta. Las situaciones que se prueban son las siguientes.

- Carga de la página de inicio.
- Autenticación y cierre de sesión por parte de un usuario. Se prueban distintas combinaciones, válidas e inválidas, de pares usuario contraseña.
- Registro de un usuario. Se registra un usuario probando distintas combinaciones, válidas e inválidas, de campos para el formulario.
- Visualización de redes. Se intenta visualizar distintas redes con distintos usuarios y configuraciones de permisos.
- Subida de una red. Se añade una red con distintas combinaciones de valores para los campos del formulario, así como fichero de descripción de la red.

Esta metodología nos permite realizar refactorizaciones del código teniendo la certeza de que este sigue funcionando con el comportamiento deseado Durante el desarrollo se ha demostrado la utilidad de este método ya que han sido necesarias varias reimplementaciones de métodos y cambios en el programa para las pruebas de rendimiento con distintos tipos de redes.

En cuanto al software utilizado, trabajamos con el paquete unittest de Python (Foundation P. S., 2016). Flask implementa un cliente de pruebas para este tipo de situaciones, que usamos para realizar distintas peticiones al servidor, configurando los parámetros para cada una.

En la parte cliente, este tipo de metodología es imposible. El resultado de las operaciones es, en última instancia, un mapa de bits que se dibuja en la pantalla y la interacción del usuario depende enteramente del estado de este mapa de bits, por lo que programar tests precisos para la aplicación es una tarea fuera de los límites de este proyecto. Hay algunas aplicaciones y frameworks como Selenium (SeleniumHQ, 2016) que permiten probar aplicaciones en lado cliente, pero están más orientadas a trabajo con documentos HTML, no con aplicaciones gráficas. Por lo tanto, la metodología para implementar los tests en el lado cliente será la de probar el comportamiento de cada módulo por separado, confirmando que los valores resultantes o que el comportamiento de los mismos esté dentro de los valores previamente calculados.

Dividimos los tests unitarios por módulos y clases. Se listan las partes del código que se prueban, así como una breve descripción de los tests. Para poder comprobar que los resultados son correctos, generamos una red de 3 nodos, lo que nos permite establecer y calcular manualmente los valores esperados para programarlos en los tests correspondientes.

- Extensión THREE.Scene. Se prueban los métodos añadidos al prototipo del objeto Scene para manejar las geometrías propias de nuestro programa.
- Proxy Scene. Se pone a prueba que la modificación de valores del proxy conlleva la modificación de los valores en los respectivos modelos de datos, comprobando que dichos valores se corresponden con lo esperado.
- Matrices. Se ponen a prueba los resultados de las operaciones del módulo de matrices con datos previamente calculados.
- SymmetricGraph. Se pone a prueba la correcta construcción de las geometrías partiendo de los datos de la red de prueba, que responde correctamente a la petición de devolver información sobre un objeto tridimensional y que aplica correctamente la máscara de filtrado.
- AsymmetricGraph. Se pone a prueba la correcta construcción de las geometrías partiendo de los datos de la red de prueba, que responde correctamente a la

petición de devolver información sobre un objeto tridimensional y que aplica correctamente la máscara de filtrado.

- Proxy Network. Se pone a prueba que la modificación de valores del proxy conlleva la modificación de los valores en los respectivos modelos de datos, comprobando que dichos valores se corresponden con lo esperado.
- Network. Esta es la clase más compleja de probar, ya que gran parte de su comportamiento se realiza de forma asíncrona. Se prueba que ejecuta correctamente la petición HTTP asíncrona una vez creada, el correcto parseo de los datos y la generación de geometrías. También se prueba que una vez finalizado este último paso, se publica el mensaje correspondiente en el bus de eventos. Se prueba que se publica la información correspondiente a un objeto seleccionado mediante el ratón. Por último, comprobamos la correcta aplicación de una máscara de filtrado.
- ShouldPaintMask. Se prueban los distintos métodos de filtrado, comprobando que generan los valores esperados.

En lo referente al software utilizado usamos Webpack para generar un documento HTML con el Javascript incrustado donde se ejecutarán los tests. Como framework de testing hemos usado (MochaJS, 2016). Como biblioteca de afirmaciones (assertion library) se ha usado Chai (ChaiJS, Chai, 2016) ya que nos permite escribir la lógica y requerimientos de los tests con una sintaxis sencilla y potente. Para generar los objetos mock usaremos SinonJS (SinonJS, 2016). Al ejecutar las pruebas directamente en el navegador, generándolas automáticamente usando Webpack, podemos poner a prueba el comportamiento del programa en distintas máquinas y plataformas, asegurándonos de que funciona correctamente en todas ellas.

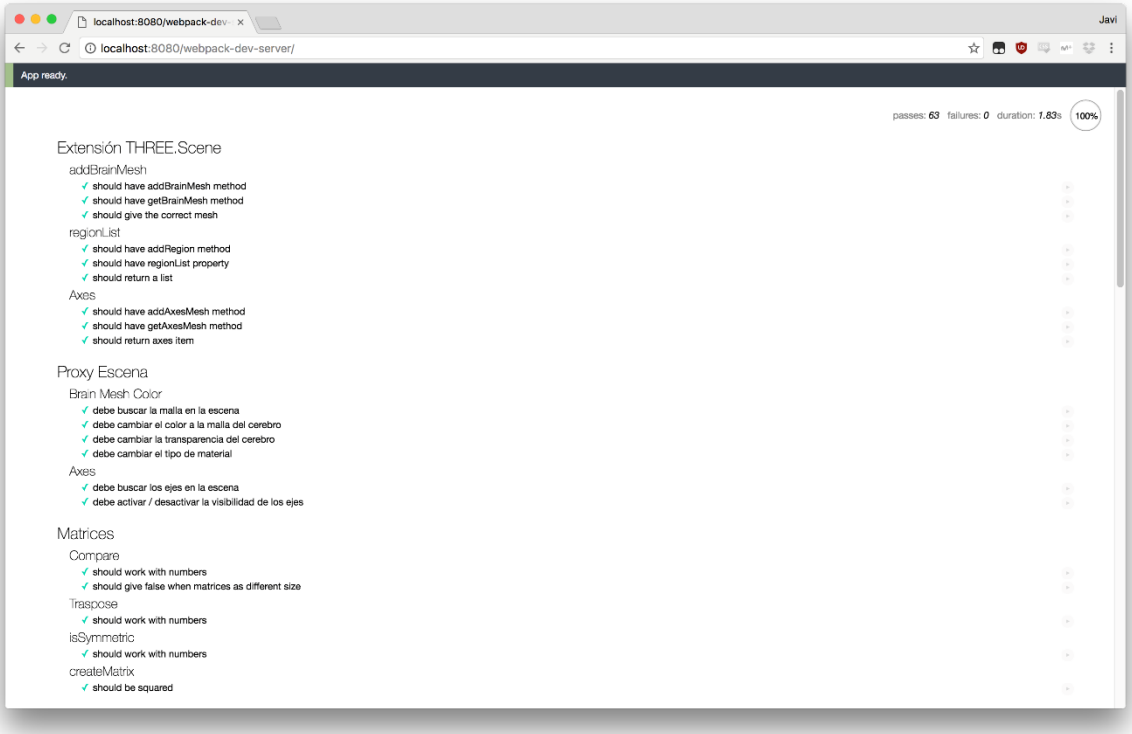


Figura 6.1. Tests unitarios ejecutándose en Chrome

SinonJS nos permite simular la respuesta a peticiones AJAX, insertando datos válidos o inválidos según la necesidad del test, por lo que nos permite probar las funciones asíncronas del software, así como el mecanismo de paso de mensajes mediante el patrón publicador/subscriptor. La figura 6.1 muestra un ejemplo de la ejecución de tests de unidad.

Pruebas de integración

Nuestro proyecto solo cuenta con dos componentes, el lado cliente y el lado servidor. El intercambio de información se realiza mediante métodos HTTP, que tanto el cliente (navegador web) como el servidor (dependerá del seleccionado para el despliegue) implementan. Al estar basado en un estándar abierto y conocido (Internet Engineering Task Force, 1999) que a su vez está implementado en software con multitud de usuarios en todo el mundo, confiamos en que el protocolo de intercambio de información funcionará correctamente.

A pesar de esto, debemos de comprobar que los datos enviados por el lado servidor son legibles por el lado cliente y que este genera los datos correctos. Para ello contamos con las redes de pruebas incluidas en el repositorio del proyecto. Si al trabajar con estas redes

detectamos algún fallo o alguna disonancia con el resultado esperado, entonces habremos localizado un problema de integración entre las distintas partes. El hecho de tratarse de un intercambio de datos principalmente unidireccional (del lado servidor al lado cliente) simplifica mucho las pruebas de integración.

Cabe señalar que, una vez implementado el formato final de intercambio de datos entre ambas partes, no se han producido errores significativos en este campo o al menos, no que no hayan sido detectados antes por la batería de tests unitarios.

Pruebas de validación

Las pruebas de validación son llevadas a cabo junto con el tutor de este TFG y usando como sujeto de pruebas al personal médico con el que este ha mantenido contacto a lo largo de todo el proceso de desarrollo. Durante la realización de las mismas, se ha recibido feedback de los usuarios y se procedió a implementar las siguientes modificaciones.

Carga de redes

Los usuarios consideran que el formulario de carga de redes no es lo suficientemente intuitivo y solicitan la inclusión de un texto de ayuda. Dicho texto deberá dar pistas sobre que espera el sistema en cada uno de los campos, así como clarificar el formato de los ficheros a subir.

El propio formulario de subida de ficheros también fue objeto de crítica. Originalmente, el nombre de la red estaba en primer lugar seguido del selector de fichero, después el dropdown para la partición y después la descripción. Los usuarios encontraban confuso este orden, pues pensaban que en el campo de nombre debían de completar los datos con el nombre del fichero a subir. Se modificó el orden de los campos del formulario y se volvió a validar con los usuarios, que, junto con el texto de ayuda, lo encontraron más intuitivo.

Esta petición entraría dentro de los requisitos no funcionales números uno y tres, que buscan que la herramienta sea sencilla de utilizar y no requiera una formación exclusiva.

Tras modificar las plantillas pertinentes y redactar un documento de ayuda, se incluye este cambio.

Filtrado

Además de la opción de filtrado por peso de los nodos, se pide la posibilidad de filtrar por hemisferios, mostrando solo las conexiones y nodos de cada uno de los mismos, y el filtrado por nodo, mostrando solo las conexiones que partan o vayan a un nodo en particular.

Si bien el requisito funcional RF3.6. sólo hace referencia a la capacidad para filtrar por peso, gracias al diseño de la infraestructura resulta sencillo implementar estos cambios.

Se ajusta y extiende la clase ShouldPaintMask, que implementa el filtrado del visor, añadiendo los nuevos requerimientos y se modifica el GUI para que estén presentes en la aplicación. Tras una nueva evaluación por parte de los usuarios, estos dan el visto bueno.

Capacidad de subir ficheros Matlab

Matlab es la herramienta habitual de trabajo para los usuarios, si bien exportar los ficheros al formato estándar es sencillo, piden la posibilidad de enviar los ficheros Matlab directamente, evitando así el paso intermedio.

Este requisito entraría dentro del requisito funcional R.F.2.4. ya que los ficheros Matlab son, junto con los ficheros de texto, un estándar en la representación de redes cerebrales.

Se extienden tanto el modelo de redes como los validadores de ficheros, para que sean capaces de entender y extraer información de ficheros Matlab, usando la librería Scipy de Python.

Modificaciones no implementadas

Durante las pruebas de validación el personal médico señaló que les gustaría poder incluir redes de vóxeles. Al contrario que las redes basadas en parcelaciones estándar, estas redes tienen un gran número de regiones distintas de pequeño tamaño (en torno a las 4000) y las redes resultantes por tanto son de mayor volumen y requieren un diseño distinto al adoptado.

Este requisito no entra dentro de ninguno de los requisitos funcionales, además, requiere un rediseño de la mayor parte de la arquitectura, tanto en el lado cliente, como del lado servidor, para poder implementarlo correctamente, por lo que queda fuera del proyecto. De todas formas y dado que se trata de un proyecto con carácter didáctico, se mostrará más adelante algunas de las ideas al respecto para la implementación del mismo.

El requisito funcional RF.3.8. no se ha podido implementar por restricciones de tiempo.

Resultados de las pruebas de validación

A pesar de los puntos expuestos en el apartado anterior, el tutor de este TFG, como mediador entre el grupo de usuarios y el alumno que desarrolla el mismo, da por buenas las pruebas de validación, haciendo constar que los usuarios finales están contentos con el producto.

7. Manual de usuario

Acceso al sistema

La aplicación es accesible a través del navegador web navegando hacia la siguiente dirección:

https://secret-refuge-83011.herokuapp.com

Al cargar la página, se nos muestra una pantalla de bienvenida con carácter informativo sobre algunas características del proyecto. Para visualizar las redes, pulsamos el botón “Explorar Redes”

Listado de redes

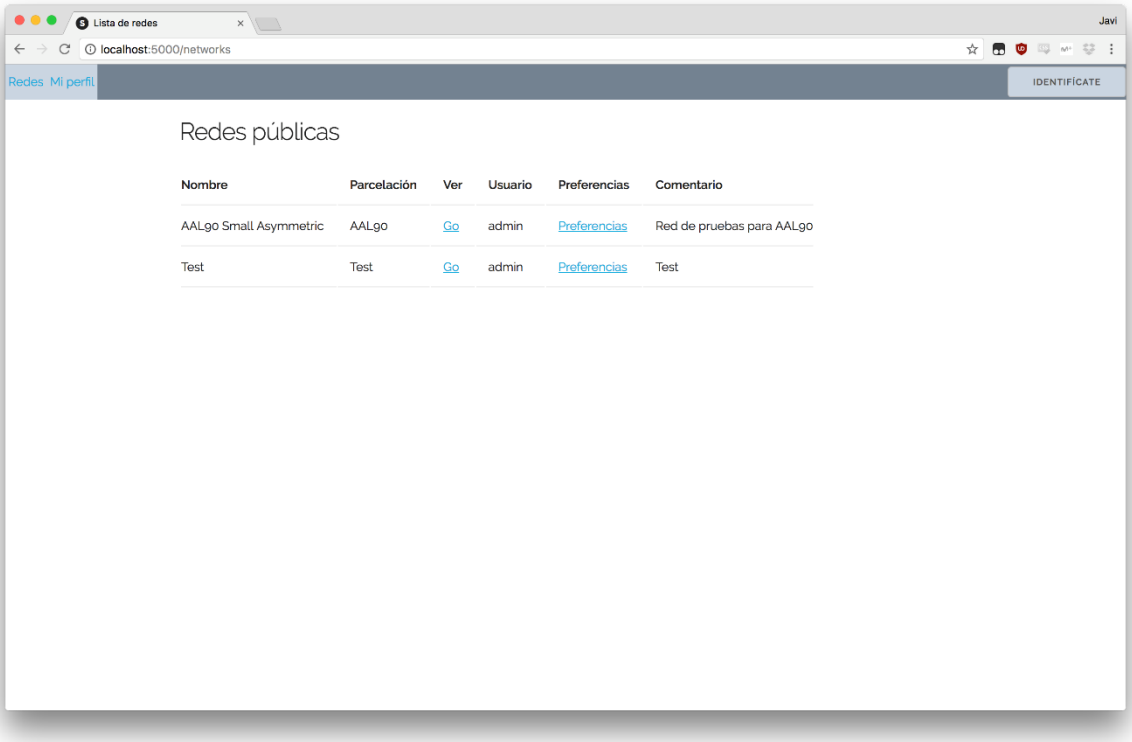


Figura 7.1. Captura de pantalla del listado de redes públicas.

Aquí podremos ver las redes que los usuarios han marcado como redes públicas (figura 7.1). Para visualizar las redes, pulsamos el enlace correspondiente a la red que deseamos ver en la columna “Ver”. Cada red está identificada con su nombre y con una breve descripción.

Autenticación y registro en el sistema

Tanto para identificarnos en el sistema como para crear una nueva cuenta, pulsamos el botón de la esquina superior izquierda “Identifícate”. Esto nos lleva a la pantalla de Identificación y Registro (figura 7.2).

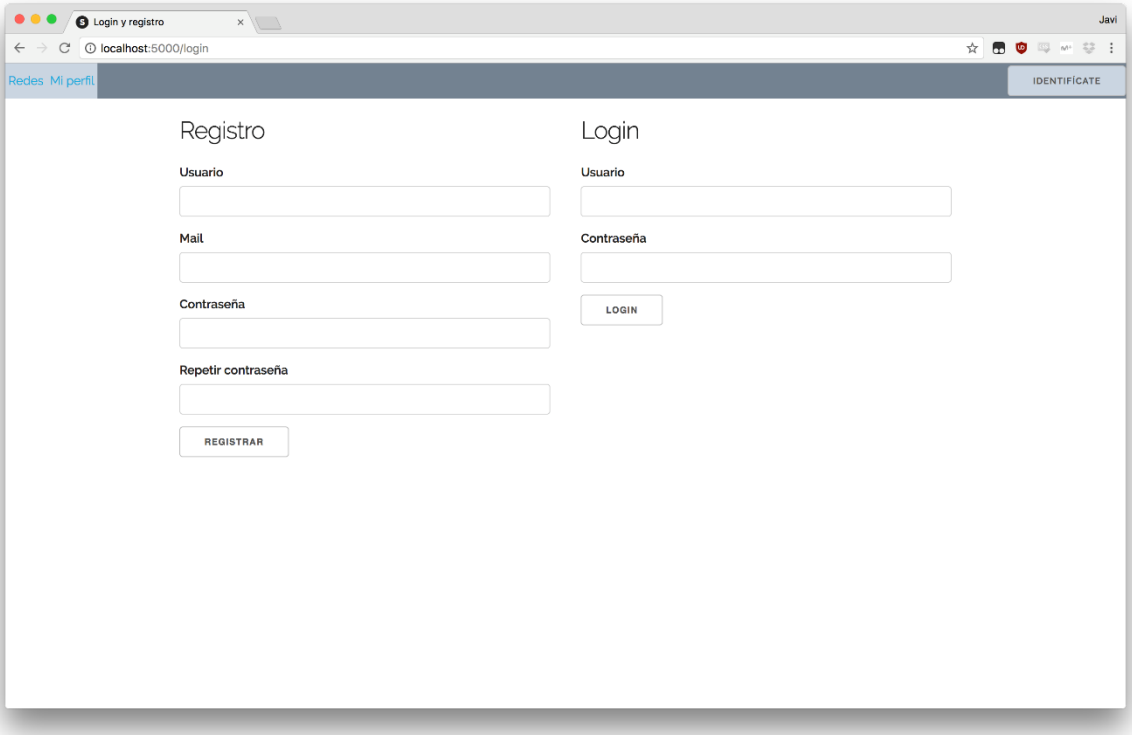


Figura 7.2. Identificación y registro

Como podemos observar, se nos mostrarán dos formularios distintos. El primero, para registrar un nuevo usuario. Rellenamos los campos con nuestras credenciales, teniendo en cuenta que la contraseña debe de tener más de 6 caracteres y que los campos “Contraseña” y “Repetir contraseña” deben de contener el mismo texto, como medida de validación. Una vez rellenado el formulario, si la cuenta se ha creado satisfactoriamente el sistema nos informará y podremos proceder a acceder con nuestra cuenta recién creada. Si por el contrario ocurrió un error durante el proceso (por ejemplo, el nombre de usuario no está disponible, la dirección de email es repetida) el sistema nos lo notificará y tendremos que repetir el proceso.

El proceso de autenticación es parecido al anteriormente descrito, debemos rellenar el formulario con nuestros datos de acceso y pulsar en el botón “Login” para completar el proceso. Si la autenticación se realiza satisfactoriamente, nos redireccionará a la lista de redes, de lo contrario se mostrará un mensaje de error (figura 7.3).

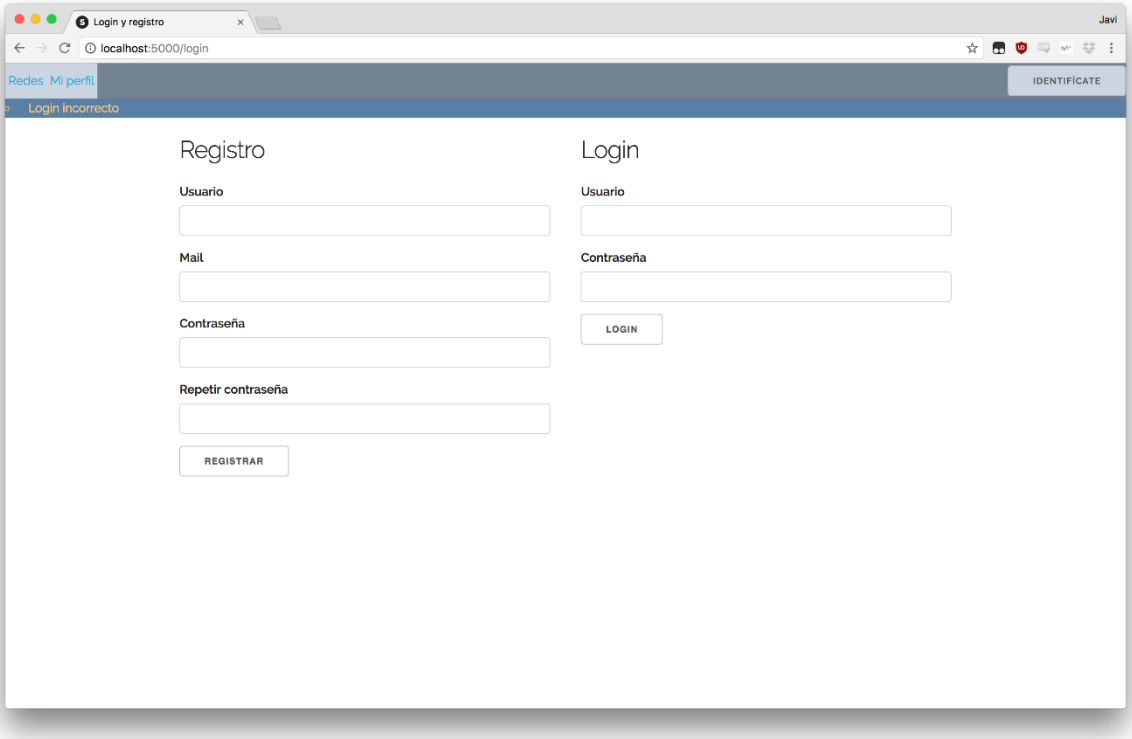


Figura 7.3. Página de login mostrando mensaje de error.

Los errores se mostrarán en la parte superior de la aplicación, justo debajo de la barra de navegación.

Listado de redes privadas

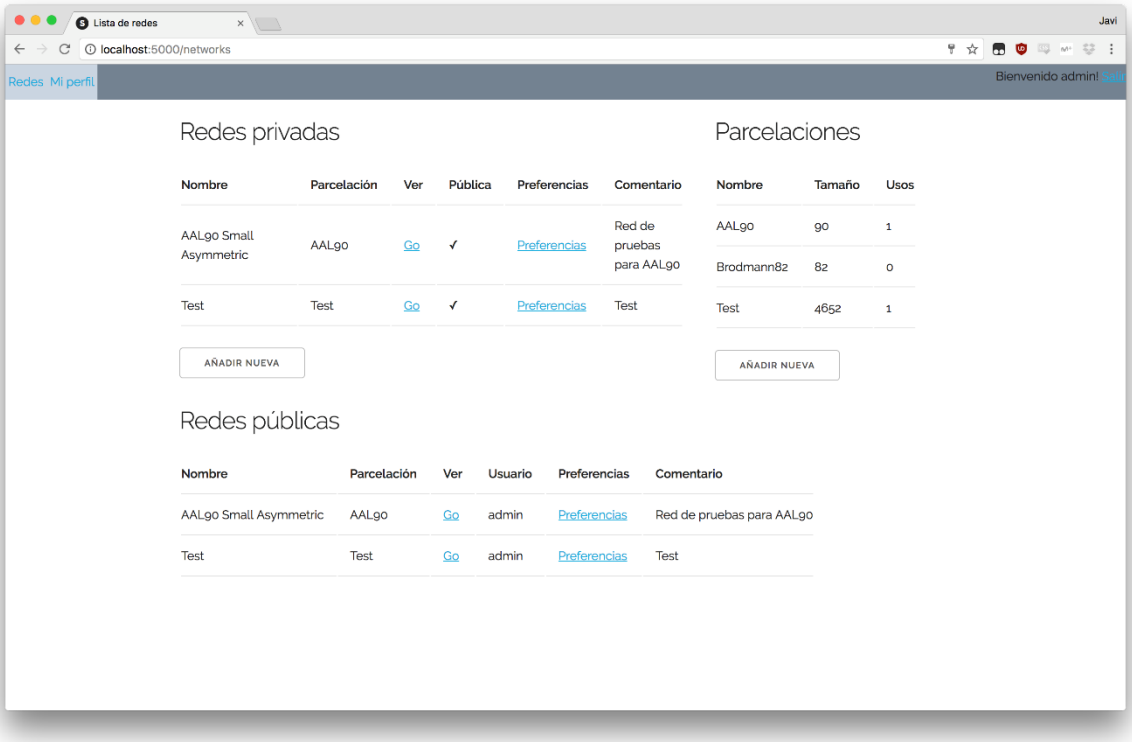


Figura 7.4. Listado de redes cuando el usuario está autenticado.

Al acceder con nuestras credenciales al sistema, la lista de redes se mostrará más completa (ver figura 7.4). En la parte superior se muestra un listado completo de nuestras redes, junto con un indicador del estado de privacidad de nuestras redes.

Crear nueva red

Para crear una nueva red debemos estar autenticados en el sistema. En la pantalla de listado de redes pulsamos el botón “Añadir nueva” en el apartado de redes privadas. El sistema nos mostrará el formulario de creación de redes (figura 7.5).

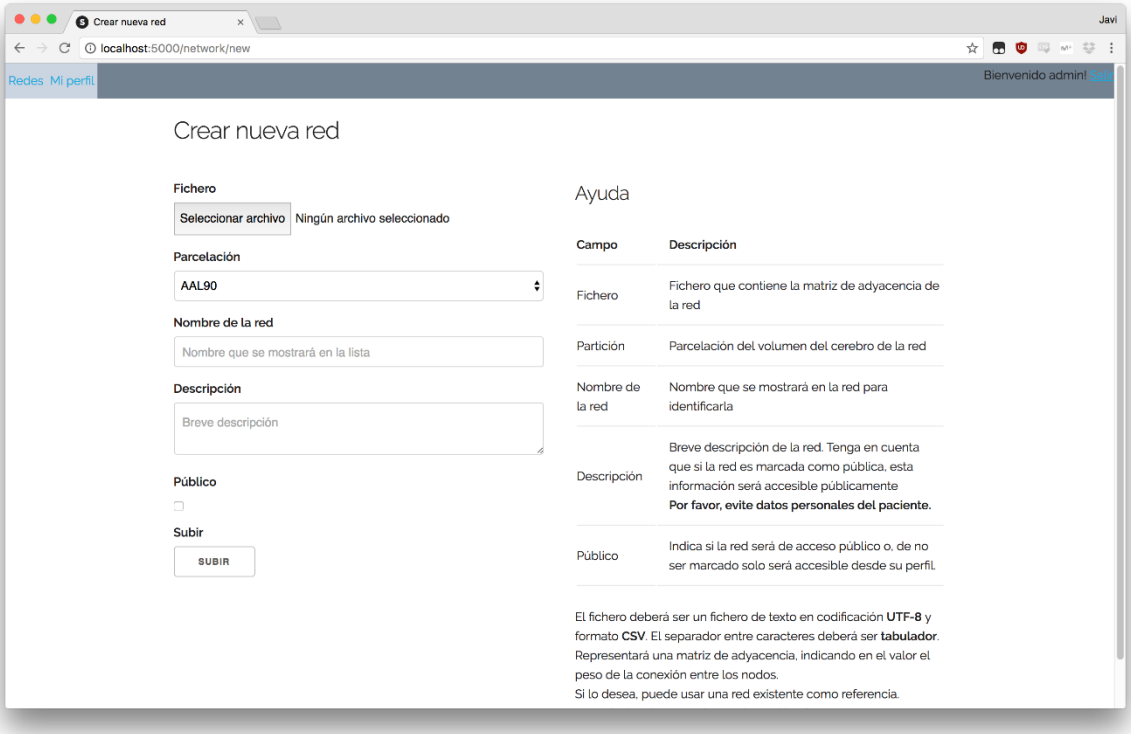


Figura 7.5. Formulario de creación de red.

En la propia pantalla se muestra una ayuda para el proceso. Describimos los campos del formulario a continuación.

- Fichero. Fichero que describe la matriz de adyacencia de la red. Deberá ser o bien un fichero CSV con codificación UTF-8 y con tabuladores como separación o bien un fichero Matlab. El nombre del dataframe que contenga la matriz deberá ser "matrix" o el sistema no podrá reconocer correctamente los datos. Este campo es obligatorio. El número de filas y columnas debe corresponderse con el número de nodos de la parcelación.
- Parcelación: Parcelación sobre la que está creada la red. Podremos elegir entre las parcelaciones por defecto del sistema (AAL90 y Brodmann82) y aquellas que hayamos creado.
- Nombre de la red. Identificador con el que se identificará a la red. No puede ser un nombre repetido ni compartir nombre con alguna red pública. El nombre no podrá superar los 30 caracteres. Este campo es obligatorio.
- Descripción. Breve texto descriptivo de la red. No podrá superar los 120 caracteres.

- Público. Indica si queremos que la red sea pública o privada. Para mantener la red privada, dejar el cuadro sin marcar.

Una vez completado el formulario, pulsamos el botón "Subir" para crear nuestra red. Si el proceso se completa satisfactoriamente, el sistema vuelve automáticamente a la lista de redes y podremos comprobar que se ha creado, de lo contrario, el sistema mostrará los errores acontecidos durante la creación de la misma.

Crear nueva parcelación

Para crear una nueva parcelación debemos estar autenticados en el sistema. En la pantalla de listado de redes pulsamos el botón "Añadir nueva" en el apartado de parcelaciones. El sistema nos mostrará el formulario de creación de parcelaciones.

En la propia pantalla se muestra una ayuda para el proceso. Describimos los campos del formulario a continuación.

- Fichero: Fichero que describe la lista de nodos de la red. Deberá ser un fichero CSV con codificación UTF-8 y con tabuladores como. Este campo es obligatorio.
- Nombre de la parcelación. Nombre con el que se identificará a la parcelación en el sistema. Este campo es obligatorio.

Todas las parcelaciones nuevas que se incluyan en el sistema serán de carácter privado y ningún otro usuario podrá utilizarlas, sin embargo, una red que se marque como pública mostrará los datos de la parcelación en cuestión.

Descarga de los datos de una red

Es posible descargar el fichero de descripción de una red de aquellas redes a las que tengamos acceso. Para ello, consultamos las propiedades de una red pulsando en el enlace "Preferencias" desde el listado de redes (ver figura 7.6).

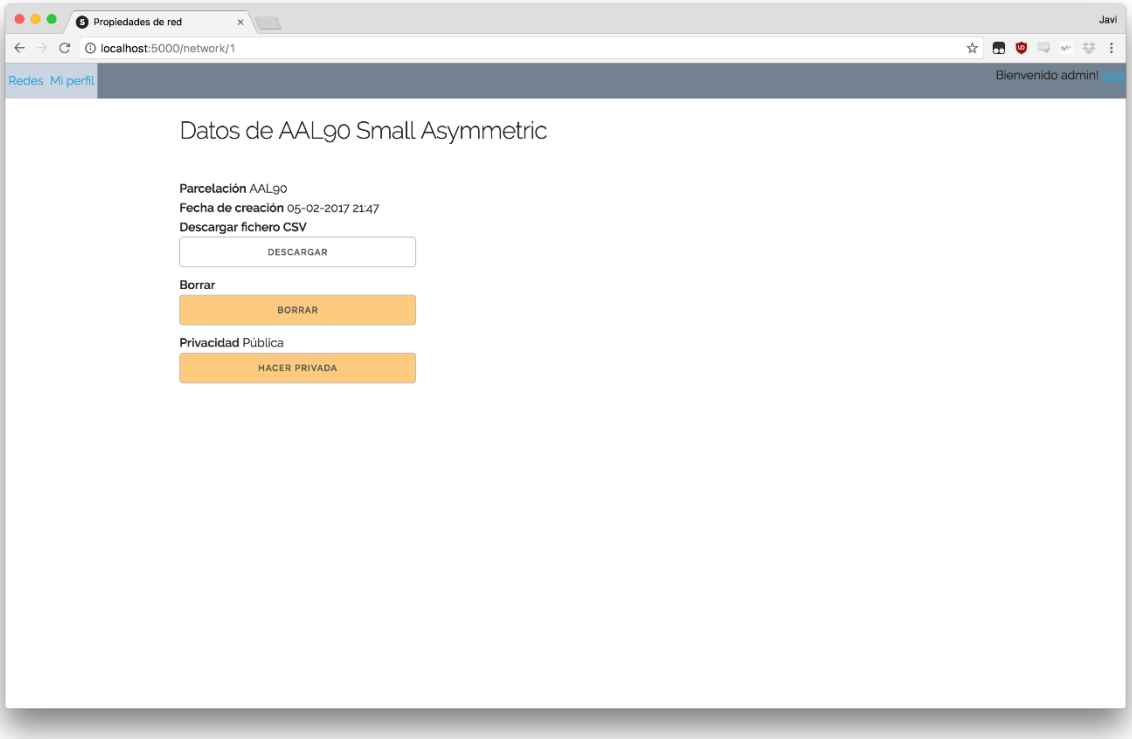


Figura 7.6. Preferencias de una red de propiedad del usuario.

En el apartado “Descargar fichero CSV” se muestra el botón “Descargar”. El navegador comenzará el proceso de descarga. El fichero tendrá el nombre de la red.

Borrado de una red

En las preferencias de una red que sea de nuestra propiedad se mostrará el apartado “Borrar” junto con un botón homónimo. El botón tiene un color destacado, indicando que es una operación destructiva. Al pulsarlo se eliminará la red del sistema.

Cambio de privacidad de una red

En las preferencias de una red que sea de nuestra propiedad se mostrará el apartado “Privacidad” indicando las preferencias de privacidad de una red junto con un botón para el cambio. El texto del botón cambiará en función del cambio que deseemos realizar. Tenga en cuenta que si algún usuario tiene una red con el mismo nombre no podrá hacer pública una red.

Cambio de contraseña

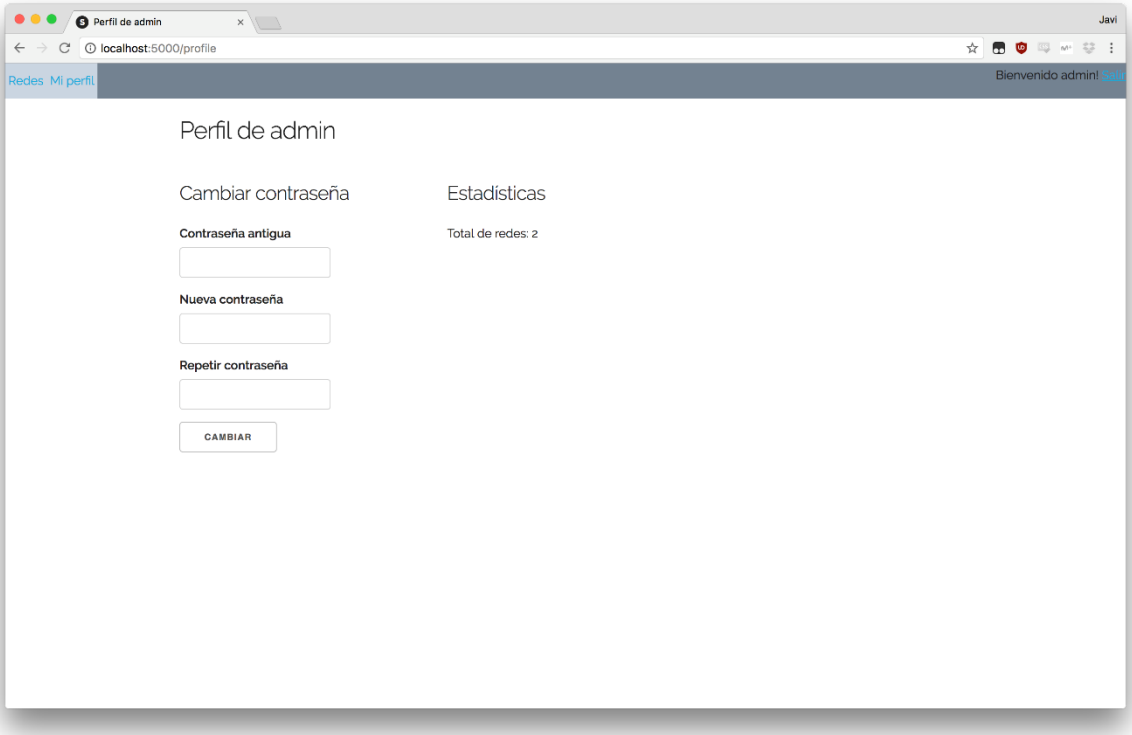


Figura 7.7. Perfil del usuario.

El cambio de contraseña se realiza en el perfil del usuario. Para acceder a este, pulsamos el botón “Mi perfil” en la barra de navegación. Tenga en cuenta que se debe estar autenticado en el sistema, de no estarlo, se nos pedirá que introduzcamos nuestras credenciales.

Una vez en el perfil (ver figura 7.7), rellenamos el formulario y pulsamos el botón “Cambiar” Los campos del formulario son los siguientes:

- Contraseña antigua: Contraseña actual con la que accedemos al sistema.
- Nueva contraseña: Contraseña con la que queremos acceder al sistema una vez realizado el cambio. Deberá ser de más de 6 caracteres.
- Repetir contraseña: Campo de validación de la contraseña introducida.

Los tres campos son obligatorios. Si el cambio de contraseña de produce satisfactoriamente, volveremos al perfil indicándonos del cambio. Si se produce un fallo durante la validación del formulario se nos mostrarán los mensajes de errores.

Visualización de redes

Desde el listado de redes, al pulsar en el enlace “Go” en la sección

Figura 7.8. Visualización 3D de la red cerebral.

Se nos mostrará la pantalla de visualización de redes y se procederán a cargar los datos (figura 7.8). En la esquina superior derecha se nos mostrarán las opciones de visualización y filtrado. En la esquina superior izquierda podremos ver la sección de mensajes. Aquí se mostrarán los mensajes de error que pudieran ocurrir durante la carga de los datos. Si la carga se realiza satisfactoriamente, se nos mostrará información sobre el elemento de la red seleccionado.

Para seleccionar un elemento de la red, simplemente colocamos el ratón sobre el mismo.

Para navegar por la escena tenemos tres movimientos posibles de cámara.

- Zoom: Acerca o aleja la cámara al punto de referencia. Se activa con la rueda del ratón.

- Orbit: Orbita la cámara con respecto al punto de referencia, situado en el centro. Mantenemos pulsado el botón izquierdo del ratón y movemos el mismo en la dirección en la que queremos mover la cámara.
- Pan: Mueve el punto de referencia de la cámara. Mantenemos pulsado el botón derecho del ratón y movemos el mismo en la dirección en la que queremos mover el punto de referencia.

Filtrado

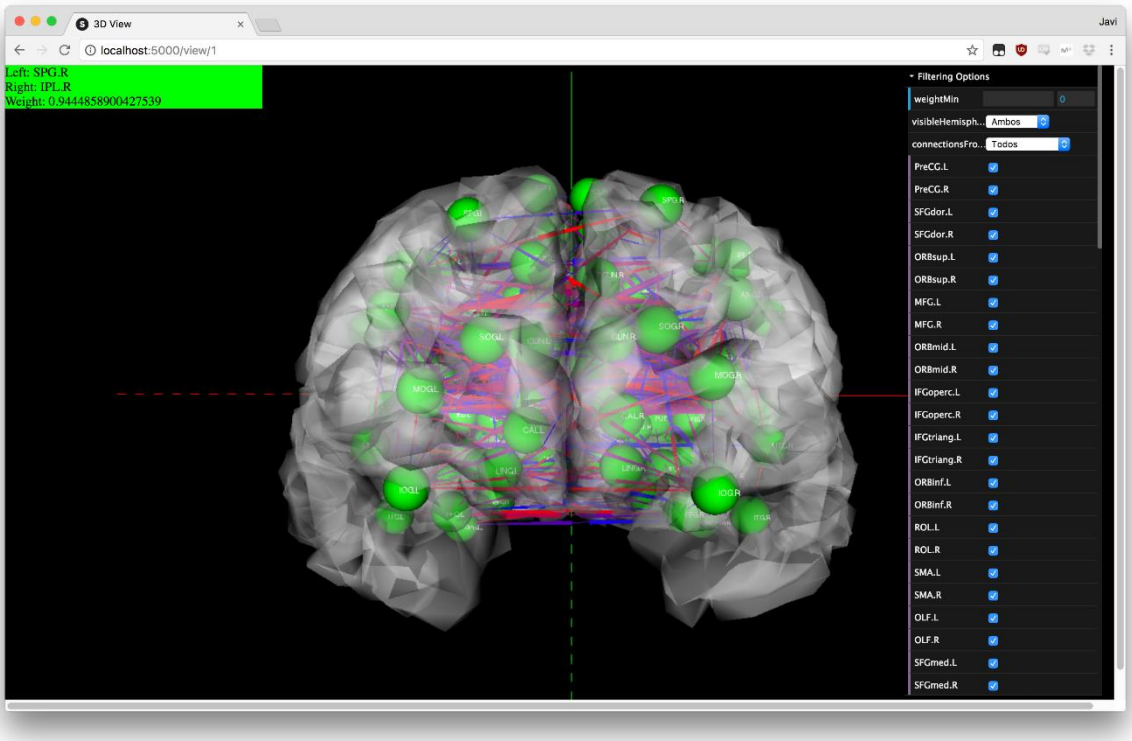


Figura 7.9. Visualización de la red mostrando las opciones de filtrado.

Al pulsar sobre el apartado “Filtering Options” se nos mostrará un menú desplegable con las mismas (ver figura 7.9). Describimos las opciones de filtrado de la aplicación.

- weightMin: Peso mínimo que debe de tener un vértice del grafo para aparecer en el visor. Nos permite descartar aquellas conexiones menos relevantes. Podemos seleccionar el valor ya sea pulsando con el botón izquierdo y arrastrando en el slider o escribiendo un valor numérico directamente en el campo. Independientemente del método utilizado, el valor se actualiza en su contraparte automáticamente.

- visibleHemisphere: Permite discriminar conexiones y nodos según el hemisferio en el que estén.
- connectionsFrom: Permite mostrar solo aquellas conexiones que parten de un nodo en concreto.
- Listado de regiones: Permite indicar manualmente que regiones se muestran u ocultan.

Los distintos filtros se aplican en cadena, haciendo que la salida de un filtro sea la entrada de otro.

Opciones de visualización de la escena

Es posible ajustar algunos parámetros de la escena, como ocultar el eje de referencia, o cambiar la apariencia de la geometría del cerebro. Pulsando sobre "Visualization Parameters" se muestran las opciones de visualización (ver figura 7.10).

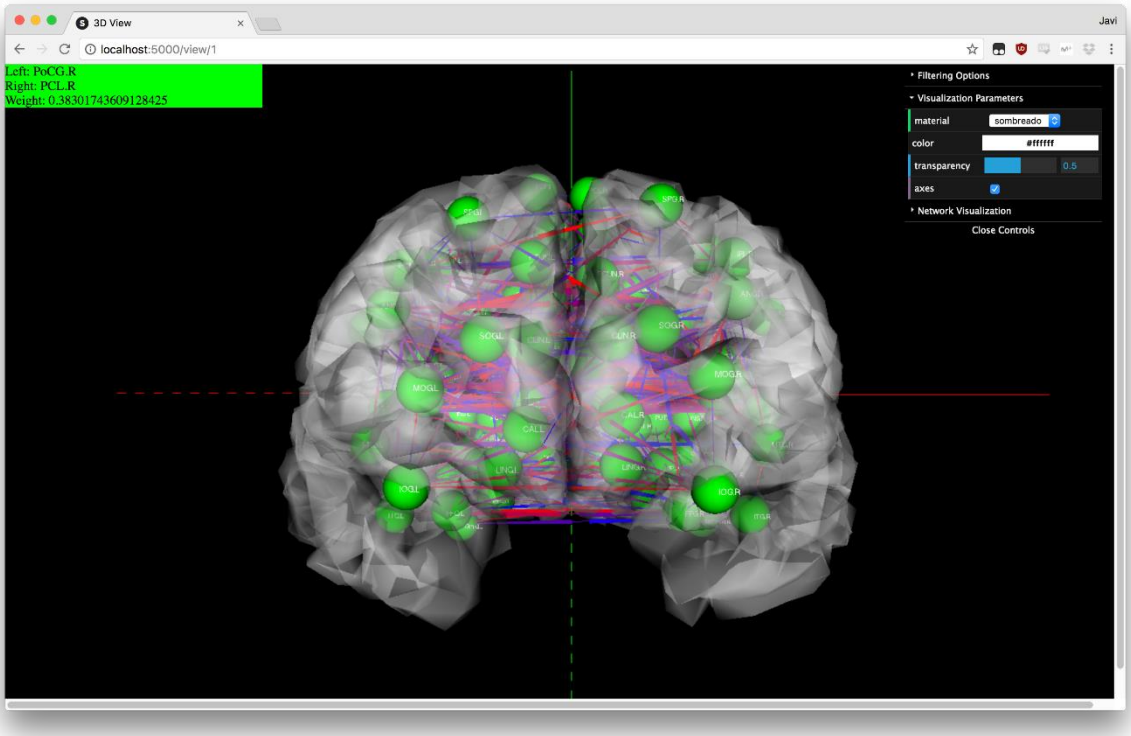


Figura 4.10. Visualización de la red mostrando las opciones de escena.

Describimos las opciones de visualización.

- Material: Tipo de sombreado utilizado para dibujar la geometría del cerebro. Las opciones son sombreado y plano. Sombreado ofrece un aspecto más realista mientras que plano ofrece mayor rendimiento y elimina las sombras.
- Color: Color del material del cerebro.
- Transparency: Transparencia del cerebro. El valor 0 indica falta de transparencia, obstaculizando la visualización de los datos de la red. El valor 1 indica transparencia total, volviendo la geometría del cerebro invisible, permitiendo una mejor visualización de la red.
- Axes: Muestra u oculta los ejes de referencia. La separación entre segmentos de línea en la parte discontinua del eje es de 2 unidades.

Opciones de visualización de la red

Las opciones de visualización de la red permiten ajustar parámetros visuales específicos a la red (ver figura 7.11).

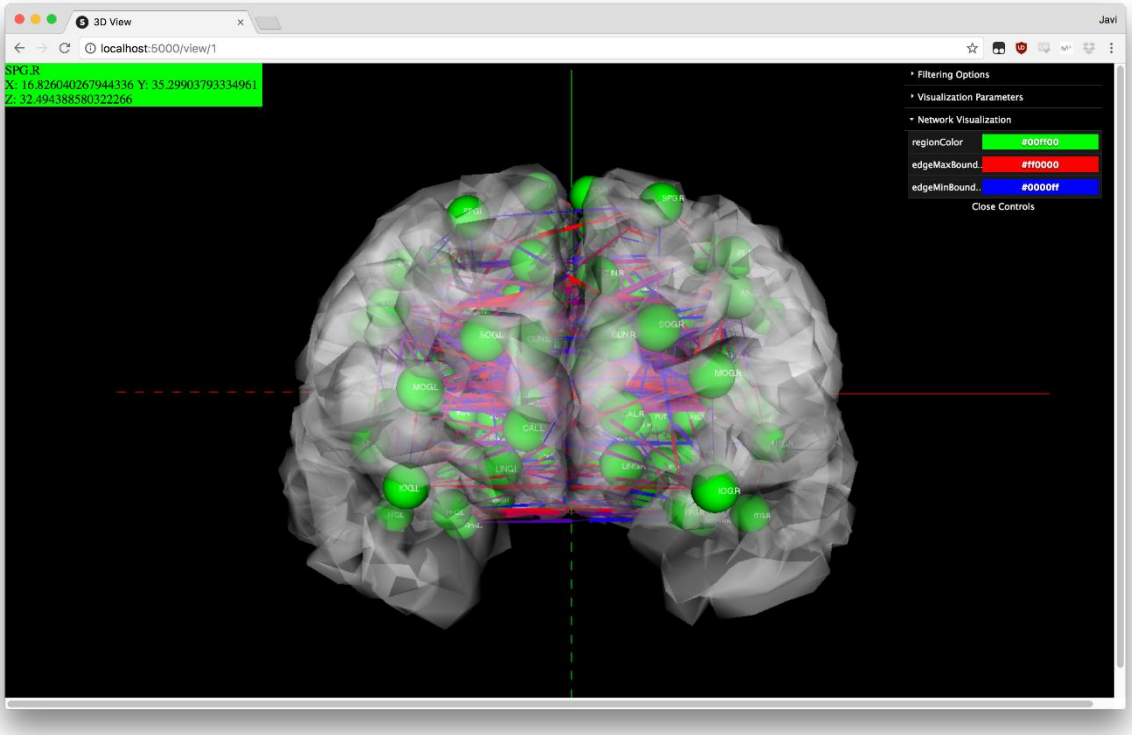


Figura 7.11. Visor de redes mostrando las opciones de visualización de la red.

Describimos los tres parámetros de la visualización.

- regionColor: Color de las geometrías que indican regiones en la parcelación de la red cerebral.
- edgeMaxBound: Color que tendrá el vértice de máximo peso en el grafo.
- edgeMinBound: Color que tendrá el vértice de mínimo peso en el grafo.

El color del resto de vértices se calculará mediante una interpolación lineal entre estos dos últimos colores. Si se desea información precisa del mismo, coloque el puntero del ratón sobre dicho vértice y la información aparecerá en la esquina superior izquierda de la pantalla, en el apartado de información.

8. Manual de instalación

En este capítulo describimos los pasos para replicar la instalación en otra cuenta de Heroku. Se debe de tener en cuenta que, como plataforma privada, su infraestructura y procesos pueden estar sujetos a cambios.

Desarrollo

Para instalar la aplicación en un entorno local de desarrollo, primero debemos instalar las aplicaciones necesarias.

- Python 3.6
- PIP 9.0.1
- NodeJS 7.4 / 7.5
- NPM 4.1.2

Usando la consola de comandos, nos situamos en el directorio de nuestro código fuente y ejecutamos:

```
$ npm install
```

```
$ pip install –r requirements.devel.txt
```

Y automáticamente se descargarán e instalarán las dependencias necesarias para la ejecución del programa. El uso de virtualenv en Python está recomendado, pero no es obligatorio.

Seguidamente, debemos de configurar la aplicación. Tenemos dos métodos, por variables de entorno, en las que podemos modificar parámetros relacionados con el tiempo de ejecución y por fichero de configuración, en el que podemos modificar todas las variables de configuración Los parámetros configurados por fichero de configuración sobreescriben aquellos configurados mediante variables de entorno. El problema de usar fichero es que en Heroku, para aplicarlos, hay que modificar el código y crear un commit en Git. Se muestra el listado de parámetros configurables por variables de entorno.

- DEBUG: Indica si la aplicación se ejecuta en modo depuración. Si su valor es "True", se mostrará una página de depuración cuando ocurra un error. Si es False, se mostrará un error HTTP código 500.
- DATABASE_URL: URL de conexión de la base de datos.
- SECRET_KEY: Clave secreta usada como NONCE en las operaciones criptográficas (cálculo de claves Bcrypt y firmado de cookies / CSRF). Es recomendable cambiarla en producción y configurarla siempre mediante variable de entorno para no incluirla en el repositorio.
- CSRF_ENABLED: Indica si queremos activar la protección contra Cross-Site Request Forgery.
- WTF_CSRF_ENABLED: Indica si queremos activar la inclusión del campo oculto de protección CSRF en los formularios.

Para configurar una variable de entorno, ejecutar el siguiente comando, cambiando el nombre del parámetro por el que se desee modificar.

$ export <Parámetro>={True | False | Valor}

A continuación, mostramos un listado del resto de parámetros de configuración.

- SQLALCHEMY_DATABASE_URI: Mismo comportamiento que DATABASE_URL. A pesar del distinto nombre, la configuración de este parámetro sobreescribe a la variable de entorno.
- BCRYPT_LOG_ROUNDS: Rondas del algoritmo de generación de contraseñas. Cambiarlo rompería la compatibilidad con las contraseñas almacenadas con un número distinto de rondas.

- WEBPACK_MANIFEST_PATH: Fichero Manifest generado por Webpack para los assets estáticos.
- SQLALCHEMY_TRACK_MODIFICATIONS: Parámetro de funcionamiento de SQLAlchemy.
- MAX_CONTENT_LENGTH: Tamaño máximo permitido para la subida de ficheros.
- ALLOWED_EXTENTIONS: Extensiones de fichero permitidas para la subida de ficheros.

El fichero de configuración deberá llamarse "config.py" y almacenarse en la carpeta "instance" en el directorio raíz del repositorio. Dentro del mismo se incluye un fichero de configuración de pruebas llamado "config.py.example" como guía para el formato y sintaxis del mismo.

Para crear el esquema de la base de datos y cargar los datos de prueba, ejecutamos:

```
$ python database_setup.py
```

Además de preparar la base de datos, el script generará credenciales aleatorias para el usuario admin.

Para ejecutar la aplicación es necesario que el sistema encuentre el fichero "manifest.json" dentro del directorio "webapp/static/". Para generarlo, ejecutamos:

```
$ npm run dev
```

Webpack se quedará en ejecución, pendiente de los cambios que se produzcan en los ficheros de la parte cliente y actualizando los ficheros paquete y el fichero manifest.

En otra terminal, ejecutamos:

```
$ python webapp_run.py
```

También se incluye un script Shell para facilitar la tarea, pero se recomienda que, en la primera ejecución, se usen los pasos individuales:

```
$ sh run.sh
```

Despliegue

Instalamos la aplicación Heroku CLI en su versión para nuestro sistema operativo. Después, usando la línea de comandos, vamos al directorio donde está el código fuente. Heroku usa como plataforma de despliegue Git. Al crear una aplicación usando Heroku CLI se creará automáticamente una rama remota. Al subir los datos a esta, se activarán los mecanismos de despliegue, configurados mediante los ficheros "Procfile" y ".buildpacks" situados en el directorio raíz. Para crear la aplicación en Heroku, ejecutamos:

```
$ heroku create –region eu
```

Por defecto, la aplicación no cuenta con base de datos, debiendo nosotros crearla y unirla a nuestra aplicación. Desde Heroku CLI, podemos realizar este paso y unirla automáticamente a la aplicación ejecutando el siguiente comando desde el directorio donde se encuentra nuestro código.

```
$ heroku addons:create heroku-postgresql:hobby-dev
```

Automáticamente se creará una base de datos y se añadirá a nuestra aplicación la variable de entorno $DATABASE_URL con la URL de conexión a la base de datos.

Podemos proceder al despliegue simplemente publicando el código a la rama "heroku" de nuestro repositorio Git.

```
$ git push heroku master
```

Para preparar la base de datos, ejecutamos:

```
$ heroku run python database_setup.py
```

El script automáticamente creará el esquema de la base de datos, creará el usuario "admin", generará automáticamente una contraseña y añadirá las parcelaciones públicas junto a la red de pruebas, todas ellas pertenecerán al usuario recién creado. Para cambiar la contraseña, entrar al sistema con las credenciales generadas y cambiarlas desde el perfil del usuario, ya que esta se encuentra cifrada en la base de datos.

9. Contenido del CD

En el directorio raíz del CD se incluyen este documento y la carpeta “proyecto” con el código fuente del programa y la carpeta “doc” con documentación autogenerada. En el directorio raíz del código se incluye un fichero README.html, con un fichero describiendo la estructura de directorios del fichero, así como notas para su despliegue, tanto en local como en producción.

Sobre la carpeta de documentación, está dividida a su vez en dos subcarpetas, “webapp” y “assets”. La primera contiene la documentación autogenerada de la parte servidor mientras que la segunda, la de la parte cliente.

10. Conclusiones

Experiencia personal

La principal conclusión que se extrae es que las tecnologías web han evolucionado hasta un punto en el que se pueden implementar aplicaciones normalmente relegadas al ámbito del escritorio. Las herramientas para trabajar con gráficos multimedia en el navegador han avanzado mucho desdeVRML y DHTML. WebGL se ha consolidado como un estándar potente y ha creado un ecosistema alrededor con proyectos tan interesantes como THREE.js

En cuanto a Javascript y su ecosistema, este ha madurado mucho desde sus comienzos y ha comenzado a consolidarse alrededor de algunas herramientas como Webpack que se están convirtiendo en el estándar de facto. Una de las críticas principales a este lenguaje es la cantidad de frameworks y herramientas que proliferaban y que tenían utilidades muy solapadas entre sí. Por suerte esto está cambiando. Como crítica, la implementación de los últimos estándares en los navegadores viene a un ritmo lento y configurar correctamente todas las herramientas para trabajar con estas características supone un esfuerzo considerable, aunque en mi opinión personal, acaba viéndose recompensado por la flexibilidad que aportan.

Con respecto a Python, ha demostrado ser un lenguaje potente y flexible, siendo capaz de realizar tareas tan dispares, como procesamiento de imagen, como implementación de sistemas web como se ha demostrado en este TFG. El ecosistema que rodea al lenguaje es amplio, con

un fuerte soporte de la comunidad e infinidad de paquetes que han ahorrado mucho tiempo evitando "reinventar la rueda" en casos como, por ejemplo, la lectura de ficheros de Matlab.

Propuestas de mejora

La última propuesta por el grupo de usuarios es el paso natural para la mejora o continuación de este TFG. Según indican los comentarios de los usuarios, este tipo de redes son tendencia actualmente, por lo que es una característica que destacaría mucho en un software de estas características.

Trabajar con este tipo de redes plantea dos retos, primero, el procesamiento en el lado servidor. Al trabajar con ficheros pesados, los hilos encargados de procesar la petición HTTP se bloquean y dejan al usuario sin respuesta. Esto hace que el navegador indique al usuario que se ha producido un timeout, a pesar de que la respuesta no se ha perdido en la red, si no que se sigue procesando. Una manera de solventar este problema sería implementar un sistema colas de trabajos asíncronos. El servidor programaría un trabajo en el sistema y se mantendría informado del estado del mismo mediante la técnica de pooling, pero seguiría libre para atender a las peticiones de los usuarios. Una opción sería un sistema de colas de trabajo distribuidas como. Celery Project (Project, 2016)

El segundo reto, sería la implementación de un sistema eficiente en el lado cliente. Existen aplicaciones que logran un renderizado eficiente de grandes cantidades de geometría, pero suelen estar implementados en lenguajes de programación menos abstractos que Javascript, con un control más manual de la distribución de memoria, lo que les permite optimizar más la ejecución.

Otra propuesta para la mejora de este TFG sería la implementación de una herramienta para el análisis topológico de las redes neuronales. Consistiría en representar las mismas mediante un grafo bidimensional y poder explorar otras propiedades del mismo, aplicando teoría de grafos. Graphviz (Graphviz, 2016) sería una herramienta adecuada para la representación de grafos, pero las operaciones a realizar sobre los mismos deberían someterse a estudio, pasar por una evaluación de requisitos y en definitiva, ser fruto de otro TFG completamente distinto.

Bibliografía

AirBnB. (2016). *react-dates*. Obtenido de http://airbnb.io/react-dates/?selectedKind=DateRangePicker&selectedStory=default&full=0&down=1&left=1&panelRight=0&downPanel=kadirahq%2Fstorybook-addon-actions%2Factions-panel

ChaiJS. (2016). *Chai*. Obtenido de Chai: http://chaijs.com

ChaiJS. (s.f.). *Chai*. Obtenido de Chai: http://chaijs.com

Chan, J. (18 de November de 2016). *How we switched our template rendering engine to React*. Obtenido de Pinterest engineering blog: https://engineering.pinterest.com/blog/how-we-switched-our-template-rendering-engine-react

Cozic, L. (2016). *SQL injections vulnerabilities in Stack Overflow PHP Questions*. Obtenido de https://laurent22.github.io/so-injections/

Debian Project. (s.f.). *The Computer Language Benchmarks Game*. Obtenido de https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=node&lang2=python3

Facebook. (2016). *A Javascript Library For Building User Interfaces*. Obtenido de React: https://facebook.github.io/react/

Fitzpatrick, B. (26 de July de 2013). *dl.google.com: Powered by Go*. Obtenido de https://talks.golang.org/2013/oscon-dl.slide#1

Foundation, M. (2016). *Fetch API*. Obtenido de Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Foundation, P. S. (2016). *Python Documentation*. Obtenido de unittest - Unit testing Framework: https://docs.python.org/3.6/library/unittest.html

Gobierno de España. (28 de November de 2014). *Ley 27/2014, de 27 de noviembre, del Impuesto sobre Sociedades.* Obtenido de Agencia Estatal Boletín Oficial del Estado: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2014-12328

Graphviz. (2016). *Graphviz - Graph Visualization Software*. Obtenido de Graphviz | Graphviz - Graph Visualization Software: http://www.graphviz.org

Group, T. P. (2016). *Chapter 8. Data Types*. Obtenido de PostgreSQL 9.6.1 Documentation: https://www.postgresql.org/docs/current/static/datatype.html

Heroku. (2016). *Dynos and the Dyno Manager*. Obtenido de Heroku Dev Center: https://devcenter.heroku.com/articles/dynos#dynos

Internet Engineering Task Force. (1999). *RFC Internet Engineering Task Force*. Obtenido de RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1: https://tools.ietf.org/html/rfc2616

JavaScript performance updates in Microsoft Edge and Chakra. (2016, 06). Retrieved from Microsoft Edge Developer: https://blogs.windows.com/msedgedev/2016/06/22/javascript-performance-updates-anniversary-update/#GW7SwMhCKZV4ZBfI.97

MessagePack. (2016). *MessagePack: It's like JSON but faster*. Obtenido de http://msgpack.org

Ministerio de Empleo y Seguridad Social. (11 de February de 2015). *Resolución de 30 de enero de 2015, de la Dirección General de Empleo.* Obtenido de Agencia Estatal Boletín Oficial del Estado: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2015-1343

MochaJS. (2016). *Mocha - The fun, simple, flexible Javascript test framework*. Obtenido de MochaJS: https://mochajs.org

Mozilla Corporation. (12 de March de 2014). *Mozilla and Epic Preview Unreal Engine 4 Running in Firefox*. Obtenido de The Mozilla Blog: https://blog.mozilla.org/blog/2014/03/12/mozilla-and-epic-preview-unreal-engine-4-running-in-firefox/

Osmani, A. (2015). *Learning JavaScript Design Patterns.* O'Reilly.

PHP. (s.f.). *History of PHP*. Obtenido de PHP Languaje: http://php.net/manual/en/history.php.php

Project, C. (2016). *Homepage | Celery: Distributed Task Queue*. Obtenido de Celery: Distributed Task Queue: http://www.celeryproject.org

Qt. (2016). *Qt Platform*. Obtenido de https://qt.io

Ruby on Rails. (2016). *Doctrine*. Obtenido de Rails Application Framework: http://rubyonrails.org/doctrine/#convention-over-configuration

SeleniumHQ. (2016). *What is Selenium?* Obtenido de Selenium - Web Browser Automation: http://www.seleniumhq.org

SinonJS. (2016). *SinonJS - Documentation*. Obtenido de SinonJS: http://sinonjs.org

Smith, P. (2012). *Professional Website Performance: Optimizing the Front-End and Back-End.* Indianapolis, IN: Wiley.

Wikipedia. (2016). *Bcrypt*. Obtenido de Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Bcrypt

Wikipedia. (2016). *Go (programming languaje).* Obtenido de Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Go_(programming_language)

Wikipedia, l. e. (2016). *Principio de sustitución de Liskov*. Obtenido de Wikipedia, la enciclopedia libre: https://es.wikipedia.org/wiki/Principio_de_sustitución_de_Liskov

Zakai, A. (s.f.). *Emscripten: An LLVM-to-JavaScript Compiler*. Obtenido de Github: https://github.com/kripken/emscripten

Zhang, Y. (28 de January de 2015). *What are pros and cons of PostgreSQL and MySQL*. Obtenido de Quora: https://www.quora.com/What-are-pros-and-cons-of-PostgreSQL-and-MySQL