APLICACIÓN WEB Y ESTUDIO DE SISTEMAS DE VISUALIZACIÓN PARA UN SISTEMA DE PREDICCIÓN DE CRISIS EN EL TRASTORNO DE BIPOLARIDAD

David Peñas Gómez

DOBLE GRADO EN INGENIERÍA INFORMÁTICA - MATEMÁTICAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO
Curso 2015/2016

Director: María Victoria López López
Codirector: Guadalupe Miñana Ropero

APLICACIÓN WEB Y ESTUDIO DE SISTEMAS DE VISUALIZACIÓN PARA UN SISTEMA DE PREDICCIÓN DE CRISIS EN EL TRASTORNO DE BIPOLARIDAD

David Peñas Gómez

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Índice general

Índice de figuras

Resumen

Las tecnologías relacionadas con el análisis de datos masivos están empezando a revolucionar nuestra forma de vivir, nos demos cuenta de ello o no. Desde las grandes compañías, que utilizan big data para la mejora de sus resultados, hasta nuestros teléfonos, que lo usan para medir nuestra actividad física. La medicina no es ajena a esta tecnología, que puede utilizarla para mejorar los diagnósticos y establecer planes de seguimiento personalizados a los pacientes.

En particular, el trastorno bipolar requiere de atención constante por parte de los profesionales médicos. Con el objetivo de contribuir a esta labor, se presenta una plataforma, denominada *bip4cast*, que pretende predecir con antelación las crisis de estos enfermos. Uno de sus componentes es una aplicación web creada para realizar el seguimiento a los pacientes y reprentar gráficamente los datos de que se dispone con el objetivo de que el médico sea capaz de evaluar el estado del paciente, analizando el riesgo de recaída.

Además, se estudian las diferentes visualizaciones implementadas en la aplicación con el objetivo de comprobar si se adaptan correctamente a los objetivos que se prenden-den alcanzar con ellas. Para ello, generaremos datos aleatorios y representaremos estos gráficamente, examinando las posibles conclusiones que de ellos pudieran extraerse.

Palabras clave: big data, visualización de datos, trastorno bipolar, aplicación web, javascript, node.js, D3.js.

Abstract

Massive data analysis technologies are revolutionating our way of life, whether we are aware or not. From big companies, which use big data to get better results to our smartphones, which use it to track our phisycal activities. Medicine can use big data too in order to improve patient's diagnosis and make personalized treatments.

In particular, bipolar disorder requieres continuous medical supervision. To contribute to that task we have developed an integrated platform, named *bip4cast*, which aims to predict patient's crisis. This platform consists of a web application, an Android app and an activity monitor weared by the patient. The web app's purpose is to track patient's status and to make graphical representations using several charts of some of the data obtained from them. That way the doctor is able to evaluate the patient's status and analysing what is the risk of having a crisis.

In addition, some implemented charts are studied in order to check if they fit in their purpose to find the conection between the medicine, the patient's status and the risk of having a crisis. To achieve this, we represent some some random generated data, studying the results which can be concluded from them.

Keywords: big data, data visualization, bipolar disorder, web application, javascript, node.js, D3.js.

1 Introducción

El trastorno bipolar afecta, en sus diversas formas, alrededor del 2,4 % de la población mundial [García-Blanco et al., 2014]. Produce alteraciones en el estado anímico que llevan al paciente a experimentar euforia extrema o depresión grave. Suele ir acompañado de alteraciones en la conducta y en la forma de pensar, llegando incluso a producirse delirios o alucinaciones. Los diágnosticos suelen aparecer en la adolescencia o los primeros años de la edad adulta.

El tratamiento farmacológico es la principal forma de abordar la enfermedad. Tiene por objetivo acortar las crisis y prevenir que estas aparezcan. Los fármacos, especialmente en dosis altas (que suelen suministrarse durante las crisis) puede tener graves efectos secundarios. De aquí surge la necesidad de, en la medida de lo posible, evitar la aparición de crisis y con ello evitar la sobremedicación del paciente. Esta detección precoz es sumamente difícil, puesto que los primeros síntomas del inicio de una crisis pueden ser muy sutiles y pasar inadvertidos incluso para el propio paciente. Es imposible establecer un patrón común para el inicio de una crisis, pero entre los síntomas frecuentes que pueden servir como indicador de comienzo de una crisis están los trastornos del sueño y los cambios en la actividad física.

Las tecnologías de análisis de datos masivos, como pueden ser las tecnologías Big Data, pueden ayudar a la medicina tradicional a la hora de tomar mejores decisiones para determinar el tratamiento más adecuado a cada paciente. Por ello hemos creado una plataforma, *bip4cast*, que pretende cubrir aspectos claves para determinar el mejor tratamiento para cada paciente: la cuantificación de la actividad física y la calidad del sueño, el control farmacológico, y el seguimiento del estado del paciente por medio de las técnicas tradicionales (test heteroadministrados que el médico realiza al paciente). Detallaremos la estructura de la plataforma en las páginas siguientes.

A continuación se expone de forma general los aspectos de *bip4cast* en su conjunto, y no sólo de la aplicación web, con objeto de entender exactamente cuál es el objetivo que pretende lograrse, de qué recursos se dispone y qué herramientas se han creado a tal efecto. Así, se realiza un análisis del problema a resolver y de los datos necesarios para construir un modelo que sirva para abordar el problema. Tras ello, se expone la arquitectura de la plataforma y los requisitos específicos de la aplicación web. Para concluir, se presenta el plan de trabajo acometido para el desarrollo de la aplicación web.

1.1. bip4cast

Así pues, el objetivo es tratar de predecir, con una antelación razonable que deje margen para la actuación por parte del médico (antelación superior a 10 días), cualquiera de los dos tipos de crisis, maníaca o depresiva. Más concretamente, el objetivo de este trabajo es la implementación de una aplicación web que sirva como elemento de interacción con el resto de elementos de la plataforma y que sirva al personal médico en su misión de prevenir la aparición de crisis en los enfermos.

Para lograr este objetivo, se propone la creación de una plataforma integral que consta de:

- Un actígrafo modelo GT9X Link, suministrado por la empresa Actigraph y entregrado a cada paciente. El actígrafo recoge de forma continuada datos de actividad física del paciente. Esto es especialmente importante para determinar la calidad del sueño, un indicador clave para conocer el estado del paciente [Actigraph, 2016].
- Una aplicación para el sistema operativo móvil Android con un propósito doble: por un lado, servir para la comunicación médico-paciente y establecer recordatorios para las distintas tomas de la medicación. Por otro, controlar variables subjetivas que sirven para determinar el estado del paciente, como pueden ser número de cafés que toma, el número de cigarrillos que consume si es fumador, en el caso de las mujeres si tienen la menstruación o no, cómo el paciente se ve anímicamente o la capacidad de concentración que posee.
- Un modelo estadístico que, con la información recogida del paciente y conociendo sus datos fenotípicos sea capaz de realizar una predicción con una antelación razonable de la próxima crisis del paciente, de forma que el profesional médico pueda actuar y tratar de evitar que esa crisis llegue a materializarse.
- Una aplicación web, del mismo nombre, que sirva para la gestión del resto de elementos de la plataforma, así como para realizar las tareas más rutinarias del seguimiento de los pacientes, como puede ser:
  - Visualización del estado del paciente. Si el modelo estadístico predijese la aparición de una crisis con una probabilidad alta, sería la aplicación web quien lo notificase al médico.
  - Control de las prescripciones médicas.
  - Control de los test que el médico realiza a los pacientes para conocer su estado.
  - Mantener la comunicación con el paciente a través de la aplicación móvil de su smartphone.

Los datos recogidos por el acelerómetro serán enviados automáticamente a un servidor dedicado que los integrará en la base de datos. Tanto el servidor de datos del actígrafo como el de base de datos permanecerán en todo momento bajo custodia del centro sanitario, con el fin de garantizar la privacidad correcta de los datos de carácter personal. El teléfono móvil del paciente y la aplicación web constituyen otras dos fuentes de datos. El médico interactuará con esta última, y no tendrá acceso a los datos recogidos por

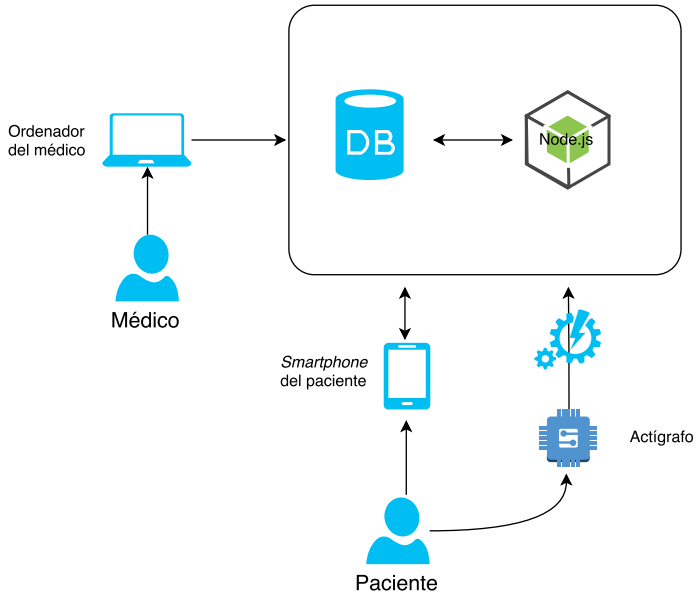

Figura 1.1: Esquema de la arquitectura de la plataforma.

la aplicación Android, cuyo fin exclusivo es servir de datos de entrada para modelo de predicción. La Figura 1.1 muestra un diagrama de la arquitectura de la plataforma.

El centro médico dispondrá de un servidor que alojará el servicio web y la base de datos de las siguientes características:

- Equipo con sistema de discos RAID2, con conexión constante a la red y conexión remota mediante claves.
- Sistema operativo Linux Server.
- 1 TB de disco duro.
- 16 GB de memoria RAM.
- Software necesario: MongoDB, instalación de node.js y npm.

1.1.1. La predicción

Para la predicción, se estudirán alternativas que permitan la implementación de un sistema adaptativo basado en redes bayesianas o redes neuronales, pues el modelo debe ajustarse a las características particulares de cada paciente. Para la implementación software del modelo, se utilizará R. Para la prueba del modelo, se seleccionará un conjunto pequeño de pacientes muy estables, a los que se les hará entrega de un actígrafo, con el objeto de recabar datos y realizar test supervisados del modelo y en general, de la plataforma.

Los datos de entrada para este modelo son los recogidos por el actígrafo, los datos fenotípicos del paciente, la medicación prescrita y los resultados de las pruebas realizadas

con la aplicación móvil, aunque también podrían añadirse otros datos recogidos del teléfono móvil que indiquen actividad física o agitación, como pueden ser consumo de datos móviles, minutos de conversación, actividad en redes sociales, etc. Otros aspectos a considerar de cara a adaptar lo mejor posible el modelo a cada paciente son:

- Avance y retroceso de la fase de sueño.
- Alteraciones en el transcurso del sueño.
- Siestas y tiempo total de sueño.
- Actividad en momentos clave: despertares precoces, actividad nocturna y matinal, etc.

1.2. Requisitos

Para el diseño de la plataforma, hemos estado asesorados en el terreno médico por el Dr. Diego Urgelés, médico psiquiatra del Hospital Nuestra Señora de la Paz de la Orden Hospitalaria San Juan de Dios. Además de como asesor médico, ha ejercido el clásico rol del *cliente* de un proyecto típico de desarrollo de software, estableciendo requisitos y funcionalidades que la plataforma debe poseer.

El objetivo de la plataforma es predecir crisis en enfermos con trastorno bipolar. Para ello es imprescindible controlar varios aspectos del tratamiento del paciente, que actúan como variables de un modelo predictivo.

Los datos fenotípicos

Los datos fenotípicos son los datos que constituyen la base para la predicción de las variables deseadas y que no cambian demasiado a lo largo del tiempo (pero que es imprescindible conocer). Nos referimos más concretamente a:

Datos personales Nombre completo, fecha de nacimiento y sexo.

Correo electrónico

Convivencia Indica con quién reside el paciente de forma habitual. Puede ser una de las siguientes opciones

- Solo.
- Pareja.
- Pareja e hijos.
- Familia de origen, es decir, padres, hermanos, etc.
- Otros.

Diagnóstico Especifica el diagnóstico del paciente siguiendo la clasificación CIE-10.

Edad de inicio de los síntomas

Sensibilidades Indica si el paciente es sensible o alérgico a litio, valproato o carbamacepina.

Estacionalidad Indica si el paciente posee estacionalidad, es decir, sufre crisis en épocas concretas del año de forma frecuente o periódica.

Síntomas psicóticos Indica si el paciente sufre de síntomas psicóticos.

Media de crisis Especifica el número medio de crisis maníacas o mixtas por año. Se toma como referencia el año completo anterior.

Período libre Número de meses que el paciente está libre de síntomas al año.

Otros Se deja un campo para que el médico pueda incorporar otra información que considerare relevante dejar plasmada.

Las recetas

Las recetas modifican el comportamiento del paciente y son tenidas en cuenta en el modelo predictivo. Una receta en la aplicación web *bip4cast* consta de:

Nombre Nombre del medicamento, o principio activo.

Tipo Tipología del medicamento. Puede ser uno de los siguientes:

- Litio.
- Antiepilépticos.
- Antipsicóticos.
- Ansiolíticos/hipnóticos.
- Antidepresivos.
- Otros.

Dosis En miligramos.

Hora Hora de toma.

Comienzo y final Fechas de comienzo y final de la receta. Es posible dejar una receta sin final (por ejemplo para medicación habitual).

Título y texto Descripción (breve y más amplia, respectivamente) de la toma. Puede contener, por ejemplo, instrucciones o indicaciones a tener en cuenta antes de consumir el fármaco.

Los mensajes

La plataforma *bip4cast* también incluye un sistema de comunicación con el paciente. El médico puede escribir mensajes directamente a la aplicación móvil del paciente a través de esta pestaña. Estos mensajes son guardados en la base de datos desde donde pueden ser consultados por el paciente.

Tienen una estructura similar a las recetas:

Comienzo y final Fechas de comienzo y final del mensaje, si se desea que éste se envíe de forma periódica en el periodo de tiempo indicado. En caso contrario, el mensaje se enviará el día seleccionado.

Hora Hora a la que el mensaje se desea enviar. Puede escogerse la opción de enviar a la hora actual.

Texto Cuerpo del mensaje.

Los test

Tradicionalmente el estado del paciente se conoce con uno de los métodos más usuales de la psiquiatría, los test. En nuestro caso no abandonamos el uso de los mismos, sino que los utilizamos para alimentar el modelo de predicción, y además, es el resultado de estos test lo que se pretende predecir con el uso de *bip4cast*.

Dada la importancia de estos test dentro del proyecto, conviene explicar muy brevemente en qué consiste cada uno de ellos.

Escala de Evaluación de la Actividad Global (EEAG) . Escala numérica (0-100) utilizada para valorar subjetivamente el desempeño social, laboral y psicológico. Esta valoración es independiente de alteraciones físicas o ambientales [Wikipedia contributors, 2016a].

Hamilton Depression Rating Scale (HDRS) . Cuestionario múltiple utilizado para valorar aspectos de los estados depresivos [Wikipedia contributors, 2016b], como puede ser estado anímico, sentimiento de culpabilidad, pensamientos suicidas, insomnio, agitación, ansiedad o pérdida de peso.

Young Mania Rating Scale (YMRS) . Escala de 11 ítems destinada a la medición de la intensidad de los síntomas maníacos, basada en una entrevista corta que al igual que la HDRS tiene en cuenta los comentarios subjetivos del entrevistado además de la observación del evaluador [Torrent, 2011]. Se trata de valorar aspectos influyentes en los estados maníacos como la euforia o la hiperactividad, el deseo sexual, la irritabilidad, la expresión verbal o las conductas agresivas.

Positive and Negative Syndrome Scale (PANSS) . Escala médica usada para medir la severidad de los síntomas de pacientes con esquizofrenia, basada en entrevistas de alrededor de 45 minutos [Wikipedia contributors, 2016c]. El nombre hace referencia a las escalas utilizadas: la positiva, que abarca excesos de las funciones normales (alucinaciones, hiperactividad); la negativa, que abarca deficiencias de las funciones normales (dificultad de pensamiento abstracto, falta de espontaneidad) y la general, que se ocupa de cuestiones que no tienen que ver con las anteriores (depresión, ansiedad, falta de atención, desorientación).

Los test siempre se realizan en una fecha determinada, aunque no todos tienen por qué realizarse el mismo día.

1.3. Etapas de desarrollo y plan de trabajo

Como integrante del proyecto para el desarrollo de *bip4cast* me fue asignada la tarea del desarrollo de la aplicación web y la visualización, objeto de la presente memoria.

Debo decir que no tenía ningún tipo de experiencia previa con el desarrollo de aplicaciones web. Más aún, desconocía por completo HTML y CSS (básico para hacer páginas estáticas) y por suspuesto las tecnologías tanto del lado de servidor que sirven desde

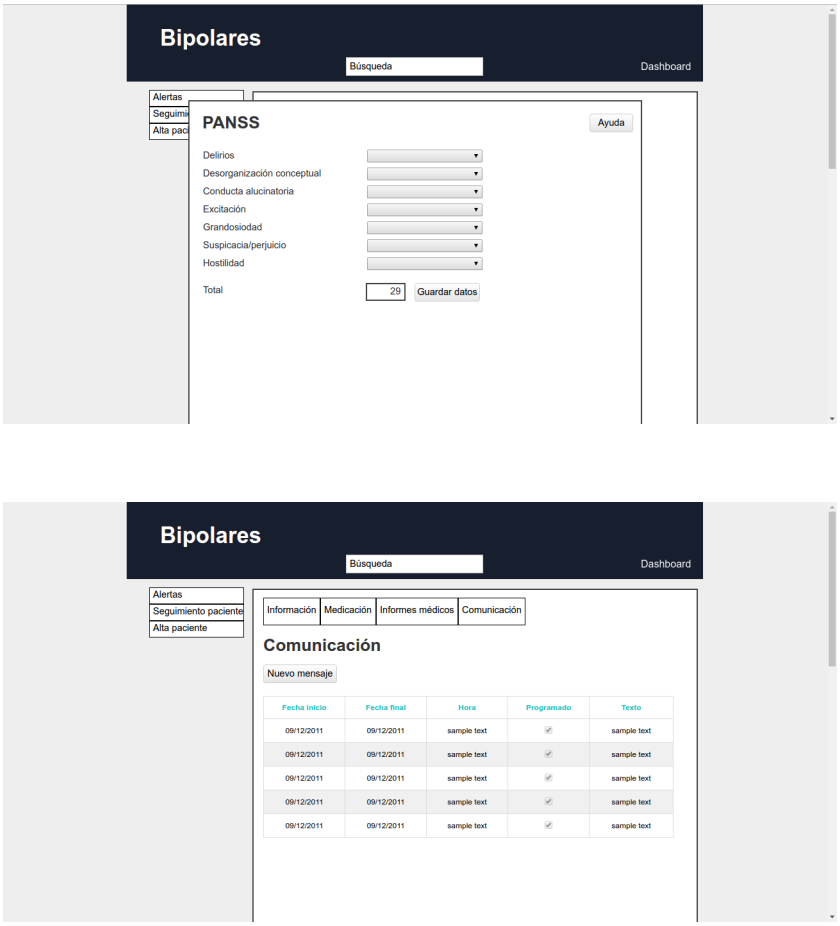

Figura 1.2: Imagenes del prototipo.

para captar datos de formularios como las del lado del cliente utilizadas, por ejemplo, para realizar animaciones. Por lo tanto el cometido de desarrollar una aplicación web era para mí un reto importante, puesto que no tenía contacto con ninguna de las tecnologías finalmente utilizadas para el desarrollo de esta aplicación.

La primera fase de mi trabajo consistió en definir los requisitos que la aplicación debía implementar. A continuación, elaboré un par de prototipos con la herramienta Justinmind muy básicos con el objetivo de que el Dr. Urgelés los validase, además de para establecer una idea del tipo de aplicación que deseábamos. La Figura 1.2 muestra uno de esos prototipos.

Una vez cerrada la etapa de prototipado, pasé a una etapa de formación autodidacta en Web y las tecnologías relacionadas. Seguí el curso de la W3School[1] en HTML y CSS para inicializarme en el desarrollo de páginas web. A continuación tocaba decidir qué arquitectura implementar. Opté finalmente por utilizar Node.js junto con Express.js,

[1] http://www.w3schools.com/

además de usar MongoDB para la capa de persistencia. Las razones de esta decisión pueden entontrarse en el Capítulo 3. Decididas las tecnologías a emplear, era hora de aprender a manejarlas. Para ello resultó vital el libro de Ethan Brown *Web Development with Node and Express* [Brown, 2014], además de mucha otra documentación disponible en internet y en foros especializados.

Para la visualización me decanté por el curso Data Visualization and D3.js [Udacity, 2014] de la web de cursos masivos online Udacity. Aprendí algunos de los principios subyacentes en el diseño de buenas visualizaciones, así como lo fundamental para usar la librería dimple.js con la que se ha elaborado la visualización de datos de la aplicación.

1.4. Organización de la memoria

La presente memoria pretende explicar y exponer el desarrollo de unla herramienta web de la plataforma, detallando las tecnologías utilizadas y el proceso de implementación seguido. El proyecto completo está disponible en GitHub bajo licencia GPLv3.

Para comenzar presentan una serie de tecnologías punteras en este momento para la visualización de datos que cubre gran variedad de plataformas, como navegadores web, que se ejecutan de forma nativa en el sistema. Se hace especial hincapié en las herramientas que el software de análisis estadístico R proporciona, por tratarse de una herramienta de código abierto. A continuación se analizan las arquitecturas híbridas batch-processing streaming en sistemas big data, prestando especial atención a las arquitecturas lambda y kappa.

El Capítulo 3 presenta la arquitectura del servicio web, y además describe las tecnologías empleadas para su desarrollo. Finaliza con un recorrido por la aplicación, explicando y detallando sus funcionalidades. El cuarto capítulo entra en detalles de implementación del servicio, y se analizan cada uno de los componentes utilizados (librerías, módulos, scripts, etc.) tanto en el servidor como en el cliente. Además, se detalla la arquitectura de la base de datos.

Finalmente, se dedica un capítulo especial a la visualización de datos en la aplicación web, presentando las distintas gráficas implementadas. Igualmente, se estudia la idoneidad de dichas representaciones a través de datos generados aleatoriamente. Una sección especial aborda la teoría detrás de la generación de números aleatorios, relacionada con la simulación estadística. La memoria concluye con las conclusiones y el trabajo futuro.

2 Estado del arte

Entre los objetivos del big data se encuentra servir de ayuda a la toma de decisiones en numerosos campos: la empresa, la defensa, la medicina, etc. La gran cantidad de datos que se tienen y que se analizan de poco sirven si los resultados que se obtienen no pueden interpretarse adecuadamente. A esto se une que en ocasiones, los responsables de la toma de decisiones son completamente ajenos al análisis de datos (especialmente de grandes volúmenes de ellos). Por todo ello, la visualización se constituye en el tercer gran pilar sobre el que descansa el big data: una buena visualización de los datos y los resultados obtenidos permite abstraer a los responsables de la toma de decisiones del tipo de datos y la metodología empleada para su obtención y análsis, y además ofrece estos resultados de forma amena y amigable.

Para el procesamiento de grandes cantidades de datos con finalidades estadísticas se han empleado tradicionalmente de forma mayoritaria SAS y SPSS, especialmente utilizados en ciencias sociales, estudios de mercado y business intelligence. Frente a estas dos herramientas, de carácter privativo, se encuentra R, de software libre que se distribuye bajo licencia GNU GPL.

Proporciona un amplio abanico de herramientas para el análsis estádistico (modelos lineales y no lineales, análisis de series temporales, etc.) y para la representación gráfica (histogramas, representación de secuencias de ADN,...). Además, sus funcionalidades pueden ampliarse mediante paquetes y cuenta con una importante comunidad de usuarios y desarrolladores que garantizan la continuidad y soporte al proyecto.

2.1. Visualización de datos en medicina y biología

La gran cantidad de información que puede transmitirse de forma visual, junto con el desarrollo de internet en los últimos años ha propiciado que la demanda de aplicaciones o servicios para representar datos haya aumentado de forma exponencial. Es el caso del llamado periodismo de datos. Actualmente es raro encontrar noticias en diarios online o publicaciones especilizadas que no vayan acompañados de una infografía, por ejemplo. De hecho, los grupos editoriales y los periódicos son los principales clientes de estos servicios, algunos de los cuales detallamos a continuación.

2.1.1. D3.js

D3 se refiere a Data Driven Documents y según [Zhu, 2013]

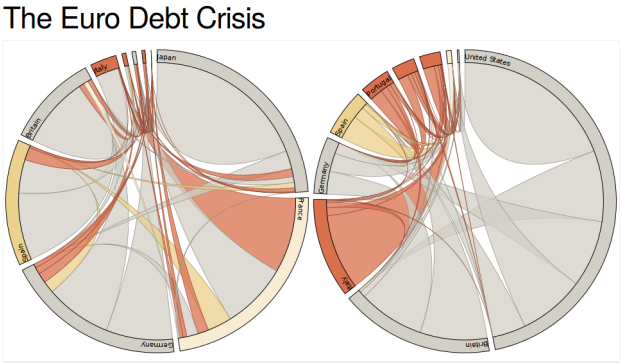

Figura 2.1: Ejemplo de diagrama realizado con D3.js.

> "D3.js es una librería de JavaScript para manipular documentos basados en datos. D3 ayuda a dar vida a los datos usando HTML, Scalable Vector Graphics (SVG) y Cascading Style Sheets (CSS). El énfasis de D3 en estándares web ofrece al usuario exprimir todas las capacidades de los navegadores modernos sin necesidad de usar un framework propietario, permitiendo así crear representaciones gráficas de gran calidad de datos usando una aproximación simple (documentos)."

En la actualidad, es uno de los métodos de representación más utilizados debido a los resultados vistosos y sobre todo interactivos que ofrece. Además, estar basado en JavaScript proporciona a D3.js una gran flexibilidad, a lo que hay que sumar una eficiente gestión de los datos en lo que se basan las representaciones gráficas. De esta forma, D3 es extremadamente rápido, soporta bien grandes conjuntos de datos, comportamientos dinámicos y animación a los resultados.

La prensa es uno de los principales usuarios de D3.js, especialmente la norteamericana, donde los artículos de periodismo de datos están acompañados de diagramas interactivos que invitan al usuario a profundizar más en la noticia. En la Figura 2.1 se muestra el diagrama de cuerda inspirado en el acompañaba un artículo del New York Times sobre la crisis europea del euro e ilustra la relación entre la deuda soberana de algunos países.

La API de D3 contiene centenares de funciones agrupadas en las siguientes unidades lógicas

- Selecciones
- Transiciones
- Arrays
- Matemáticas
- Color
- Escalas
- SVG
- Tiempo
- Capas
- Geografía
- Geometría
- Comportamientos

2.1.2. Tableau

Tableau es una compañía americana con sede en Seattle que ofrece una familia de productos de visualización gráfica especialmente orientado a business intelligence. Actualmente desarrollan su motor gráfico para varios soportes: escritorio, plataformas móviles y la nube. Es además uno de los líderes del sector, siendo usado por compañías de primer nivel como Deloitte, Pfizer, Toyota o SpaceX entre otras.

Permite la creación de visualizaciones de alto nivel con tan sólo arrastrar los datos y observar cambios en tiempo real mientras nuevos datos se incorporan al dataset que estamos representando. Su uso intuitivo y sencillo lo hace ideal para ser empleado en entornos ajenos al análisis de datos.

2.1.3. CartoDB

CartoDB es una plataforma Software as a Service especializada en la representación de datos geográficos sobre mapas interactivos. Nacida en España, ha obtenido gran repercursión internacional y ha cosechado gran éxito en diversas rondas de financiación, llegando a captar más de 30 millones de doláres. Entre sus clientes se encuentran la National Aeronautics and Space Administration (NASA), Banco Bilbao Vizcaya Argentaria (BBVA) o Twitter.

Cuenta además con una comunidad de desarrolladores y usuarios de software que despliega sus propias instancias del software gracias a que buena parte de las APIs son públicas.

2.1.4. La visualización de datos en R

R es un lenguaje y entorno de programación para análisis estadístico y gráfico de código libre. La visualización de datos en R es bastante extensa. La representación puede hacerse de numerosas formas en función del tipo de datos a representar. Dispone de paquetes para representar datos multivariantes, datos categóricos, binning, representación tridimensional e interactiva, etc [Hartvigsen, Gregg, 2014]. No sólo esto, sino que, por ejemplo, también se ofrecen herramientas específicas para la representación de grandes conjuntos de datos (que conllevaría mucho tiempo debido a la alta carga computacional que supone) o para la creación de árboles que pueden ser usados para representar dendrogramas derivados de los procesos de clustering. Merece especial mención por su amplio uso dentro de la comunidad R `ggplot2`, un paquete creado por Hadley Wickham, que apoyándose en los paquetes básicos (especialmente `lattice` y `graphics`) trata de mejorar sus posibilidades, atendiendo especialmente a muchos de los detalles incómodos como puede ser la generación de las leyendas. Apoyándose en la grammar of graphics, está compuesta de una serie de componentes independientes que pueden combinarse de diferentes maneras, de forma que no son necesarios grandes conocimientos para generar gráficos de gran calidad.

La interoperabilidad de R permite que la representación de datos pueda realizarse con

otras herramientas distintas de las que la propia distribución ofrece, por ejemplo PostScript, Portable Document Format (PDF) o SVG. Otra opción es exportar la visualización de datos a pequeñas aplicaciones web que por regla general suelen ser interactivas. Las herramientas más populares que utilizan R para la visualización de datos son:

shiny Framework desarrollado por la comunidad de R [Cheng et al., 2015]. Simplifica enormemente la creación de aplicaciones web interactivas con R. Según la propia web del proyecto, hace posible *"construir bonitas, sensibles y potentes aplicaciones con un mínimo esfuerzo"*.

plotly Es una aplicación web interactiva desarrollada sobre la librería D3.js que gestiona la representación gráfica de datos. Tiene APIs para trabajar con R, Phyton, Matlab y ggplot2 entre otros.

Actualmente R proporciona, a través de numerosos paquetes (listados todos ellos en el CRAN Task View `Medicine Image`), soporte para datos e imágenes referidos a visualización médica. Por ejemplo

- Imágenes por resonancia magnética.
- Positron Emission Tomography (PET).
- Electroencefalografía.

lo cual permite utilizar R para el análisis de la imagen médica. Fuera de este campo concreto de la medicina, la mayor parte de los datos médicos con que se trabaja son datos cuantitativos frente al tiempo, esto es, son en esencia series temporales.

2.2. Arquitecturas híbridas batch-processing streaming

Junto con la visualización, el procesamiento de datos constituye otro de los pilares fundamentales del big data. El procesamiento debe ir enfocado siempre a satisfacer las consultas de los datos en el menor tiempo posible, pero debido al gran volumen de datos que se maneja esto a veces no es posible. Además, otro de los objetivos principales de las tecnologías big data es el procesamiento de ingentes cantidades de datos que se recogen en tiempo real. De esta forma, existen varios paradigmas para el procesamiento de datos:

Procesamiento batch El procesamiento secuencial de datos por lotes ha sido la forma tradicional de procesar datos, y las tecnologías big data la han heredado, adaptada, y sirve perfectamente al problema del volumen de datos a analizar. Por otro lado, es una tecnología madura fácilmente escalable. Presenta varios inconvenientes: se centra en el procesamiento de datos estáticos y tiene una alta latencia. La herramienta estrella del procesamiento distribuido es Hadoop, puesto que engloba tanto el almacenamiento de datos distribuido como su procesamiento utilizando MapReduce. A partir de él se han ido desarrollado frameworks que trabajan sobre él, como pueden ser Pig, Cascading. Por otro lado, se encuentra Spark, que no utiliza MapReduce para el procesamiento de datos aunque sí realiza procesamiento batch.

Procesamiento en streaming El procesamiento de datos en tiempo real es idóneo para un procesamiento rápido de flujos de datos ilimitados y continuos (a diferencia de los procesamientos batch que estaban especializados en datos estáticos). Se proporciona de esta manera una baja latencia y una alta tolerancia a fallos, pero por contraposición la escalabilidad aún no está lo suficientemente asegurada. Ejemplos de implementación de estas tecnologías los encontramos en Storm, Flume, Trident y Spark streaming (la versión de Spark para procesamiento en streaming).

Frente a estas dos alternativas, se alza una tercera, unión de ambas. Son las llamadas arquitecturas híbridas [Marz and Warren, 2015], que combinan una parte de procesamiento batch y otra de procesamiento en streaming, de forma que pueden analizarse datos estáticos (por ejemplo, datos históricos) como datos nuevos en tiempo real (datos nuevos adquiridos). Es importante hacer notar que estas tecnologías híbridas no combinan ambas tecnologías en el sentido literal de la palabra, sólo las "encapsulan". Esto se debe a lo radicalmente distintos que son los dos paradigmas de procesamiento, lo que hace imposible su combinación.

La arquitectura de estas tecnologías responde al siguiente patrón:

Capa batch Esta capa está compuesta por el conjunto de todos los datos (dataset principal) y una tecnología de procesamiento batch. Los nuevos datos se incorporan al dataset principal a medida que van llegando para su almacenamiento.

Capa streaming El flujo de datos nuevos también llega a esta capa, completada por una tecnología streaming para su procesamiento.

Capa de combinación Los resultados de sendas capas se combinan aquí y se ofrecen al usuario.

2.2.1. Arquitectura-λ

La arquitectura lambda es un tipo de arquitectura híbrida ideada por Nathan Marz [Marz and Warren, 2015]. La idea básica es estructurar el procesamiento en tres capas, como se exponía en la sección anterior. Puede verse un diagrama de la arquitectura en la Figura 2.2.

Capa *batch*

Cuando se quiere ejecutar una consulta sobre todos los datos, la cantidad de tiempo que puede demorarse esa consulta es alta. La capa batch se encarga de preprocesar (mediante indexación) los datos contenidos en el dataset principal. Estos datos precomputados son las denominadas vistas batch. Son estas las que se analizan cuando se quiere hacer una consulta sobre el dataset. Las lecturas sobre las vistas batch pueden ser aleatorias.

De esta forma, la capa batch está formada por las vistas batch creadas y el dataset principal. Este tipo de procesamiento es ideal para sistemas batch como puede ser Hadoop. Esta capa está constantemente computando vistas batch desde cero. Además,

las vistas batch pueden procesarse en paralelo debido a su simplicidad, por lo que la aplicación de Hadoop en su procesamiento es idóneo (escalado horizontal).

Esta aproximación induce dos problemas. El primero es que crear las vistas batch supone un procesamiento de todo el dataset, por lo que es predecible que sea una operación de alta latencia. El segundo se deriva de éste primero. ¿Qué ocurre con los nuevos datos que se reciben mientras se crean las vistas batch? Actualmente, estamos obviando estos datos (no se están procesando) por lo que la información de consulta no es completa.

Capa de servicio

Una vez obtenidas las vistas batch, el siguiente paso es cargarlas en algún sitio desde donde puedan ser consultadas. Ese lugar es la capa de servicio. Es una base de datos distribuida especializada que permite las lecturas aleatorias de las vistas batch, aunque no es necesario que soporte escrituras aleatorias pues no son necesarias. Así, la complejidad de la base de datos se simplifica enormemente. ElephantDB o Cloudera Impala son ejemplos de bases de datos de esta capa.

Además es la encargada de mantener las vistas actualizadas. De esta forma, cuando nuevas vistas batch están disponibles, la capa de servicio automáticamente sustituye una antigua por la nueva.

Capa de velocidad

Por último, está la capa de velocidad, que trata de responder a las deficiencias de la capa batch. De esta forma, la capa de velocidad se encarga de recoger el flujo de datos que va llegando al sistema mientras se crean las vistas batch y actualiza el dataset con estos nuevos datos cuando las vistas están completas. Como su nombre sugiere, su propósito es garantizar que la nueva información está analizada para que pueda ser consultada en cualquier momento.

Podría pensarse que la capa de velocidad es similar a la capa batch puesto que ambas producen vistas basadas en los datos que reciben. Una de las diferencias entre ambas es que la primera se centra en los nuevos datos mientras que la segunda lo hace sobre el dataset completo a la vez. La otra gran diferencia es que, con el propósito de reducir al máximo la latencia, la capa de velocidad no examina todos los nuevos datos al mismo tiempo. En lugar de eso, actualiza los datos ya procesados (llamadas vistas en tiempo real)con los nuevos que entran (en contraposición con la capa batch, que recomputa la vista batch por completo).

De esta forma, la arquitectura lambda satisface todos los puntos deseables de un sistema big data:

Tolerancia a fallos Hadoop controla la posible caída de los nodos de procesamiento, mientras que la capa de servicios usa replicación para asegurar la disponibilidad de los datos en caso de caída de los sistemas. Además, tanto la capa de servicio como la batch son tolerantes a errores humanos.

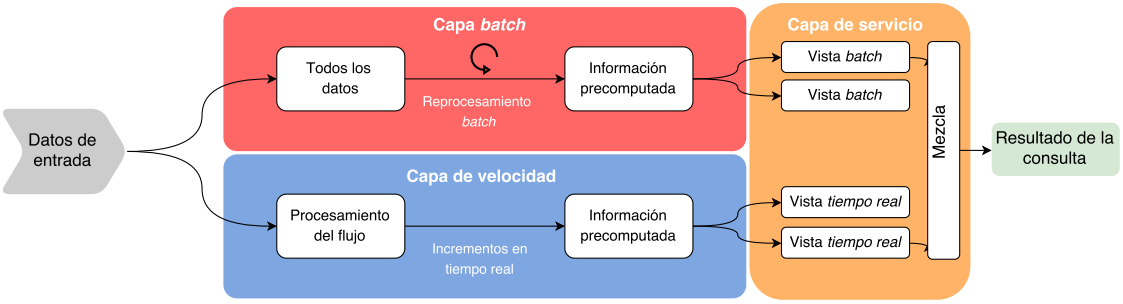

Figura 2.2: Diagrama de la arquitectura lambda.

Baja latencia en lecturas y escrituras La capa de velocidad se encarga, mediante las vistas en tiempo real, de tener analizados los nuevos datos que llegan al sistema.

Escalabilidad Las capas batch y de servicio son fácilmente escalables de forma horizontal.

Generalidad El sistema está diseñado para propósito general.

Extensibilidad Añadir nuevas vistas es tan sencillo como añadir nuevas funciones que operen sobre el dataset.

Consultas *ad hoc*

Mínimo mantenimiento El componente principal a mantener es Hadoop, que pese a que requiere algún conocimiento previo, es fácil de gestionar. Por otro lado, puesto que no escribimos de forma aleatoria, ahorramos muchos problemas derivados de esto, como pueden ser fallos de concurrencia.

Depuración Las entradas y salidas de los procesos de análisis de datos están siempre disponibles, de forma que es fácil depurar en caso de error.

A esto se suman algunas tendencias que concurren actualmente:

(1) Se está alcanzando el límite físico de las CPUs mononúcleo. De esta forma, si quieren escalarse los sistemas, es necesario paralelizar los cálculos y procesamientos.

(2) Generalización de las Infrastructure as a Service, como es el caso de Amazon Web Services (AWS), que permiten ajustar la capacidad de procesamiento a la demanda real. Estas simplifican enormemente la administración de un sistema y en general abaratan costes de cálculo.

(3) Por otro lado, el ecosistema big data se está enriqueciendo constantemente, especialmente con aplicaciones y servicios open source. Algunos, como Hadoop, los hemos mencionado anteriormente, pero también hay sistemas de bases de datos Not Only SQL (NoSQL) (Cassandra, HBase, Voldemort, MongoDB, etc.) o sistemas de paso de mensajes entre procesos (Apache Kakfa), o sistemas de procesamiento en streaming como los que se indicaron antes.

2.2.2. Arquitectura-κ

Aunque tras esta primera aproximación a la arquitectura lambda pudiera parecer una buena forma de solucionar las carencias de los sistemas batch y streaming, no le faltan detractores. El primero de ellos es Jay Kreps, un ingeniero de LinkedIn que en un hilo de O'Reilly Radar [Kreps, 2014] hizo un análisis pormenorizado de las ventajas e inconvenientes de la arquitectura de Marz.

Entre los puntos fuertes que observa Kreps está que la arquitectura lambda preserva inalterados los datos de origen, pues ya sabemos que se trabaja con las vistas batch y las temporales. De esta forma, como ya destacamos en su momento, el proceso de depuración es más sencillo, pues puede hacerse etapa por etapa del procesamiento de los datos. Otro de los puntos que destaca es que esta arquitectura se centra en el a veces ignorado problema del reprocesamiento de las arquitecturas en streaming, entendiendo éste como el nuevo análisis de los datos de entrada para volver a generar una salida cada vez que el código que calcula esta (las consultas) cambian. Mantener ese problema latente resulta preocupante, puesto que el código cambia constantemente.

Por otro lado, observa los siguientes dos puntos negativos

(1) El primero tiene que ver con la complejidad de gestionar un sistema que trabaja con dos paradigmas de procesamiento de datos tan distintos como son un sistema batch y otro streaming. Por un lado está el problema de implementar con dos tecnologías un software que debe proporcionar los mismos resultados (nada tiene que ver el código para hacer MapReduce y el código escrito en Storm o Spark). Una posible solución sería utilizar un framework para programar que abstrayese para qué sistema se está programando (si el batch o el streaming), como ocurre con Summingbird. Pero usando una herramienta de este tipo condenamos al ostracismo el rico ecosistema surgido alrededor de Hadoop.

(2) El segundo trata sobre lo complicado que puede resultar depurar en dos sistemas tan distintos.

Una alternativa

Para Kreps, el mayor lastre de la arquitectura lambda es depender de un sistema de procesamiento batch, que ya hemos mencionado que se ve afectado por una latencia desmesurada. Por tanto la cuestión que plantea es por qué no usar sendos procesos en streaming (más rápidos) para procesar los nuevos datos y los datos históricos, haciendo especial hincapié en mejorar el paralelismo de los procesos en streaming para obtener latencias bajas en el análisis de datos históricos.

Plantea esta arquitectura (que puede verse en la Figura 2.3), que requiere de un sistema de procesamiento en streaming y un módulo de almacenamiento rápido (por ejemplo, Apache Kafka):

1. Usar un clúster Apache Kafka como sistema de almacenamiento rápido, para mantener un registro completo (log) de los datos que se quiere que se vuelva a procesar.

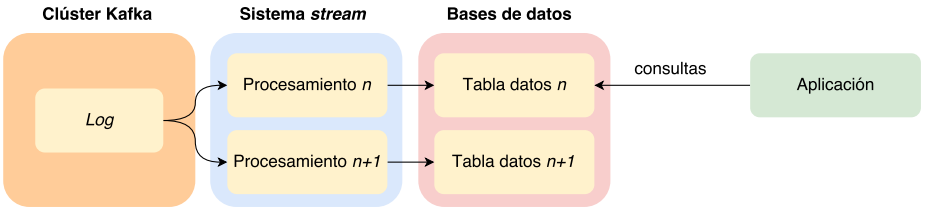

Figura 2.3: Esquema de la arquitectura kappa.

2. Cuando se quiera hacer el reprocesamiento, arrancar una segunda instancia del sistema de procesamiento en streaming que que empiece a procesar los datos seleccionados en el apartado anterior desde el principio, pero que vuelque su salida en una nueva tabla de resultados.
3. Cuando esta segunda instancia se ha puesto al día, establecer la nueva tabla como fuente de datos de las consultas, puesto que contiene la información actualizada.
4. Parar la versión antigua y borrar la antigua tabla de resultados.

A diferencia de la arquitectura lambda, en esta aproximación sólo se realiza el reprocesamiento cuando el código de la aplicación cambia. Observar que todo esto no significa que los datos no puedan almacenarse en HDFS, simplemente significa que no se puede ejecutar el reprocesamiento sobre este. Por otro lado, la arquitectura lambda requiere de dos procesos ejecutándose continuamente: por un lado el que crea las vistas batch y por otro el que crea las vistas temporales, mientras que con la arquitectura kappa basta tener dos procesos simultáneos cuando se necesita reprocesamiento. Por contra, se requiere el doble de espacio en la base de datos de la capa de servicio además de que ésta soporte un alto volumen de escritura.

3 Servicio web

3.1. Objetivo

El objetivo de la aplicación web *bip4cast* es servir como punto de interacción con el resto de elementos de la plataforma del mismo nombre. A través de ella el médico puede atender los dos aspectos principales de la plataforma: la gestión habitual de los pacientes y la visualización de los datos de que se dispone. La primera incluye:

- Control de las prescripciones médicas.
- Control de los test que el médico realiza a los pacientes para conocer su estado.
- Mantener la comunicación con el paciente a través de la aplicación móvil de su smartphone.

De esta forma el responsable médico tiene en su mano una herramienta con la capacidad de mostrarle todos los datos que la plataforma gestiona de una forma sencilla y amigable. No sólo al médico, sino a cualquier persona que no esté suficientemente familiarizada con la disciplina médica en cuestión.

Se ha decidido que la aplicación de gestión de la plataforma *bip4cast* sea web debido a:

La necesidad de un sistema portable Un sistema que influye de forma significativa en las personas y en su calidad de vida como es *bip4cast* debe responder a cualquier contingencia relacionada con un paciente que pudiera surgir en cualquier momento, incluso si el médico no está físicamente en su consulta. Por ello, es necesario que éste sea capaz de responder ante ella lo más pronto posible. El empleo de una arquitectura puramente web hace que los responsables médicos puedan hacerse cargo de ella, por ejemplo, desde su propio smartphone.

La capacidad cross-platform de las aplicaciones web El fuerte desarrollo de los navegadores web, íntimamente ligado a la explosión de Internet como plataforma de distribución de contenidos de todo tipo (imágenes, vídeo, servicios de tiempo real, etc.) ha propiciado que éstos puedan ser utilizados como nuevos soportes para la ejecución de aplicaciones. Tanto es así, que en los últimos tiempos se ha incrementado considerablemente el número de aplicaciones que corren directamente en los navegadores como resultado de servicios que se encuentran en la nube (los Software as a Service).

Además, el auge de los dispositivos móviles y el hecho de que cada uno de ellos incorpore un navegador web hace más universal si cabe este tipo de aplicaciones. Al ejecutarse en el navegador, y como resultado de la aceptación generalizada de los

estándares web por parte de los desarrolladores de navegadores web, es sumamente fácil desarrollar para numerosas plataformas (tanto móviles como de sobremesa) sin necesidad de que el desarrollador atienda las particularidades de cada una de ellas.

La madurez de las tecnologías utilizadas La Web de hoy en día nada tiene que ver con la de hace unos años. Existen numerosas tecnologías que permiten hacer casi cualquier cosa con un navegador web. En particular nos referimos a:

- Los motores JavaScript: El lenguaje de *scripting* web más utilizado es sin duda uno de los grandes responsables del auge de las aplicaciones web. Hace que las webs sean dinámicas y respondan ante la interacción del usuario. Hasta la fecha, el uso de JavaScript se ha hecho mayoritariamente en el lado del cliente, pero últimamente se ha experimentado un auge de las implementaciones de JavaScript en el lado del servidor.
- La utilización sencilla de frameworks de diseño. La interoperabilidad de las aplicaciones móviles puede ser también una de sus grandes debilidades. La tipología tan variada de dispositivos en los que las webs se visualizan (en especial los diferentes tamaños de pantalla de dichos dispositivos) puede resultar un auténtico quebradero de cabeza para los desarrolladores, que deben comprobar que las páginas web se visualizan correctamente en cada tipo de dispositivo. El uso de Bootstrap hace muy sencilla la aplicación de los principios del diseño web adaptable a las webs.
- La capacidad para construir un sistema altamente escalable: Aunque el proyecto *bip4cast* está en una fase muy inicial, su pretensión es ser un sistema ampliamente utilizado. Esto requiere que la plaforma sea fácilmente escalable y esto no es sencillo con las tecnologías web tradicionales (debido a problemas relacionados con la concurrencia y la gestión de peticiones). Por ello se ha empleado como tecnología subyecente en el lado del servidor node.js, acompañado de express.js para la gestión del tráfico HTTP.

3.2. Tecnologías del lado del servidor usadas

A continuación hablaremos con más detalle de cada una de las tecnologías utilizadas. Las agruparemos en función de si se emplean del lado del servidor o del lado del cliente.

Tradicionalmente se ha utilizado el webstack LAMP (Linux + Apache + MySQL + PHP) como conjunto de herramientas software para la creación de sitios y aplicaciones web. Últimamente se comienza a tomar cierto auge MEAN (MongoDB + Express.js + Angular.js + Node.js) o alguna de sus variantes (hay una que sustituye Angular.js por Ember.js como framework para las vistas). Esto constituye una arquitectura JavaScript end-to-end (es decir, en todo el sistema). En nuestro caso hemos optado por utilizar un motor de renderizado distinto, Handlebars, más sencillo que estos dos últimos y que detallamos más adelante.

3.2.1. Node.js

Node.js (o simplemente Node) es un entorno de ejecución basado en eventos, con base JavaScript, en el lado del servidor, utilizado fundamentalmente en el desarrollo de aplicaciones web. No es un framework JavaScript o un contenedor web. Tampoco es un servidor de aplicaciones como puede serlo JBoss o GlassFish. Las funcionalidades de node.js pueden ser expandidas a través de módulos (módulos JavaScript). Para la gestión de estos módulos (instalación, desinstalación, actualización, etc) node dispone de un gestor integrado, npm, que en palabras de los propios autores [npm documentation, 2016]

> "Es una forma de reutilizar código de otros desarrolladores, además de una forma de compartir tu código con ellos y hace fácil gestionar las diferentes versiones del código."

¿Por qué con Node.js es más sencillo crear aplicaciones en red escalables? Con los sistemas tradicionales, cada conexión se gestiona en un nuevo hilo, al que el sistema asigna recursos. Puesto que los recursos son limitados, a medida que crece el número de conexiones son necesarios más recursos o más maquinas para atender todas estas peticiones (lo que llamamos escalado horizontal) [Abernethy, 2011]. Node resuelve este problema de la siguiente forma: cada conexión, en lugar de generar un nuevo hilo, genera una ejecución de evento dentro de Node. Es decir, sólo existe un único hilo y además el sistema no se bloquea para operaciones de entrada/salida, que son asíncronas (lo cual evita *deadlocks*).

3.2.2. Express.js

Express.js es un framework web utilizado para el desarrollo de aplicaciones web y APIs, considerado el estándar *de facto* para la capa de servidor de Node.js. Se describe como [Brown, 2014] [Yaapa, 2013]

> "Un framework flexible y minimal para Node.js que provee de un conjunto robusto de características para consutruir aplicaciones web de página simple y/o multipágina-híbridas."

¿Qué quiere decir todo esto?

Flexibilidad La flexibilidad de Express viene por su buena integración con los paquetes de node.js que comentábamos anteriormente además de por la posibilidad de usar *middlewares*: métodos que se ejecutan cuando se realizan las peticiones.

Minimal Express, recién desplegado, proporciona el conjunto básico de herramientas para la gestión de peticiones, al contrario que otros, que incorporan una gran cantidad de funcionalidad muchas veces inutilizadas.

Aplicaciones web de página simple Express proporciona soporte para la creación de este tipo de aplicaciones web, en las que toda la aplicación se descarga en el navegador del cliente ganando así velocidad en la navegación puesto que tras la descarga incial apenas existe comunicación con el servidor.

Aplicaciones web multipágina e híbridas Las aplicaciones web multipágina se acercan bastante al concepto tradicional de sitio web, en el que cada página se obtiene por una petición independiente del cliente al servidor. Las aplicaciones web híbridas combinan esta aproximación como la de la aplicación de página simple.

3.2.3. MongoDB

MongoDB es un sistema de base de datos NoSQL orientado a documentos, es decir, que en lugar de guardar los datos en tablas (como se hace de forma tradicional) se guardan estructuras similares a JSON (más concretamente, una implementación propia llamada BSON).

Entre sus características principales se encuetran:

De propósito general Casi tan rápido como las bases de datos NoSQL de tipo clave-valor y con casi todas las funcionalidades de las relacionales.

Balanceo de carga Mongo se puede escalar de forma horizontal, de forma que el desarrollador determina cómo serán distribuidos los datos de una colección. Además, MongoDB tiene la capacidad de ejecutarse en múltiples servidores de forma que puede balancearse la carga y/o replicarse los datos para mantener el sistema en funcionamiento en caso de fallo.

Seguridad Implementa sistemas de autenticación y autorización, así como gestión de usuarios (cada uno de los cuales tiene distintos permisos).

Ejecución de JavaScript del lado del servidor Mongo posee la capacidad para realizar consultas usando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

MapReduce Permite utilizar MapReduce para el procesado de la información a través de funciones JavaScript que se ejecutan en los servidores.

3.2.4. Handlebars

Handlebars es un motor de plantillas web (web templating system). Basado en Moustache (otro motor de plantillas web), añade algunas funcionalidades lógicas (operadores `if`, `each`,...) que permiten iterar sobre los datos recibidos para la composición del documento HTML. Muy sencillo de utilizar puesto que, a diferencia de otros motores de renderizado (como Jade), mantiene la sintaxis original HTML.

En el caso de esta aplicación, se utiliza mayoritariamente del lado del servidor, puesto que express.js invoca al motor de renderizado con el contexto adecuado y éste envía al navegador web cliente el documento HTML final.

¿Cómo funciona? Para renderizar completamente un documento HTML, Handlebars necesita dos ficheros: la vista y el *layout*. Ambos son dos ficheros de extensión `.handlebars` que contienen páginas web o fragmentos de éstas. Contienen "huecos" o directivas que deben rellenarse (con información de un paciente, por ejemplo). Por otro lado necesita

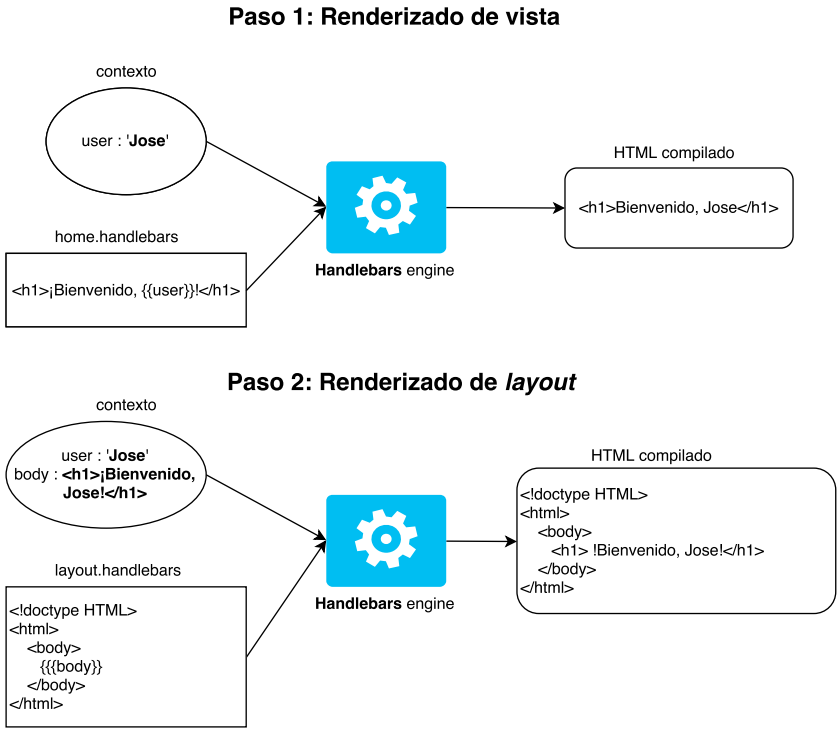

Figura 3.1: Proceso de renderizado de HTML con Handlebars.

el denominado *contexto*, es decir, la información con la que Handlebars rellena esos huecos presentes en las plantillas o vistas. Primero el motor renderiza las vistas, (es decir, rellena esos huecos con lo datos del contexto que tenga disponibles) y después se renderiza el *layout* con el resultado de la etapa anterior, obteniéndose así un documento HTML completo. En el esquema que se muestra en la Figura 3.1 puede verse el proceso completo.

3.3. Tecnologías del lado del cliente usadas

3.3.1. jQuery

jQuery es una biblioteca de JavaScript que permite interactuar y manipular documentos HTML a través de la interacción con el DOM, manejar eventos y responder así a la interacción del usuario con un sitio web.

Puesto que cualquier navegador moderno (tanto de sobremesa como móvil) es capaz de ejecutar JavaScript, jQuery funciona en cualquier plataforma. Además, está especialmente diseñado para superar las diferencias entre los motores JavaScript de los distintos navegadores.

3.3.2. Bootstrap

Bootstrap es un framework para el diseño de aplicaciones y sitios web, especialmente enfocado en diseño web adaptable. Es ampliamente utilizado en numerosas páginas web debido a su sencillez, la comunidad que está tras su desarrollo, la gran cantidad de recursos que hay disponibles de forma abierta (plantillas, scripts,...) y sus resultados de apariencia profesional y llamativa. Sigue una filosofía de diseño *mobile-first*, primando el diseño para estas plataformas (y desde ese diseño, pasar de forma automática a diseños para dispositivos con mayores resoluciones).

Incluye además gran cantidad de componentes listos para ser usados: botones, desplegables, paneles de navegación, barras de progreso, elementos de paginación... Así como elementos dinámicos, como pueden ser ventanas emergentes (llamadas *modals*), mensajes de alerta, carruseles,...

3.3.3. Dimple.js

Dimple.js es un API para D3.js para la realización de gráficos. El objetivo es simplificar la realización de gráficos, evitando la curva de aprendizaje de D3 para dibujar gráficos con muy poco esfuerzo. Independientemente de eso, no oculta la implementación de D3, de forma que éste puede seguir usándose para complementar posibles carencias de dimple.js o para la realización de gráficos complejos que dimple.js no contempla.

Es capaz de dibujar gráficas de los siguientes tipos:

- Gráficos de barras y de líneas
- Gráficos tarta y de anillo
- Diagramas de dispersión y de burbuja
- Gráficos de área

Cada dataset puede ser encapsulado en una *serie* de dimple y cada *serie* puede representarse gráficamente de una forma distinta, dando lugar a combinaciones entre varios formatos de gráficos (en bip4cast se han combinado gráficos de área y de líneas, por ejemplo). No se limita únicamente a dibujar las gráficas, sino que también incorpora elementos para animarlas cuando se pintan o para crear animaciones de forma automática (las llamadas *storyboards*).

3.4. La aplicación

La aplicación se basa en el concepto de *dashboard* o panel de control. La interfaz, basada en la plantilla homónima de Bootstrap, provee una interfaz de diseño web adaptable además de un diseño claro, moderno y eficaz. En la Figura 3.2 se muestra cómo se ve la aplicación en una tablet y un smartphone.

Cada acción diferenciada de la aplicación está representada en una vista distinta (dar de alta pacientes, controlar la medicación, visualizar los datos). Si cada acción diferen-

Figura 3.2: Diseño *responsive* de la aplicación.

ciada contiene otras acciones asociadas, como por ejemplo en el caso del control de la medicación puede ser añadir o modificar una receta, ésta se hace a través de los *modals*: pequeñas ventanas emergentes animadas. Se ha tratado de aprovechar los recursos de Bootstrap al máximo en aras de asegurar la mayor consistencia a la aplicación, además para conseguir un resultado profesional. Así, los mensajes de error, éxito, advertencia o información se realizan a través de los *alerts* o los iconos pertenecen a la librería de *glyphicons*.

Las referencias espaciales que hagamos a medida que se describa la aplicación se harán bajo el supuesto de que el dispositivo con el que se trabaja tiene una resolución de escritorio (mayor de 1024 por 768 píxeles).

La pantalla principal de la aplicación es el dashboard. Por el momento es una página estática, pero queda preparada para que en futuras revisiones de la aplicación contenga información relevante que deba ser mostrada cada vez que el profesional acceda a la aplicación.

La Figura 3.3 muestra el dashboard. Podemos ver en la parte superior la barra de navegación que será visible en todo momento, compuesta de, de izquieda a derecha: el nombre de la aplicación, enlaces al dashboard, a la vista de visualización y a la de alta de nuevo paciente, la barra de búsqueda de pacientes y el menú de ayuda.

3.4.1. Alta de pacientes

La tercera opción del menú principal es el formulario de alta de pacientes. A través de este formulario nuevos pacientes se dan de alta en la aplicación y a partir de ese momento pueden empezar a ser seguidos por el sistema. Puede verse en la Figura 3.4 y en la Figura 3.5 el formulario completo.

Los datos aquí recogidos son los denominados fenotípicos (que se describieron en Sec-

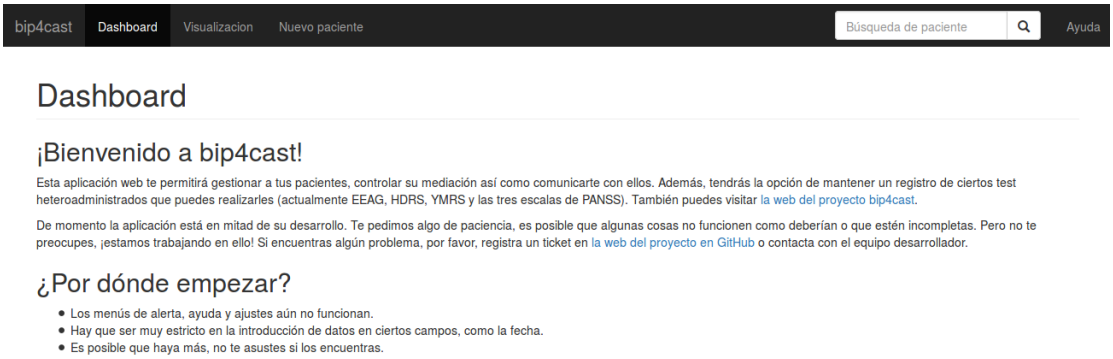

Figura 3.3: Vista principal del dashboard.

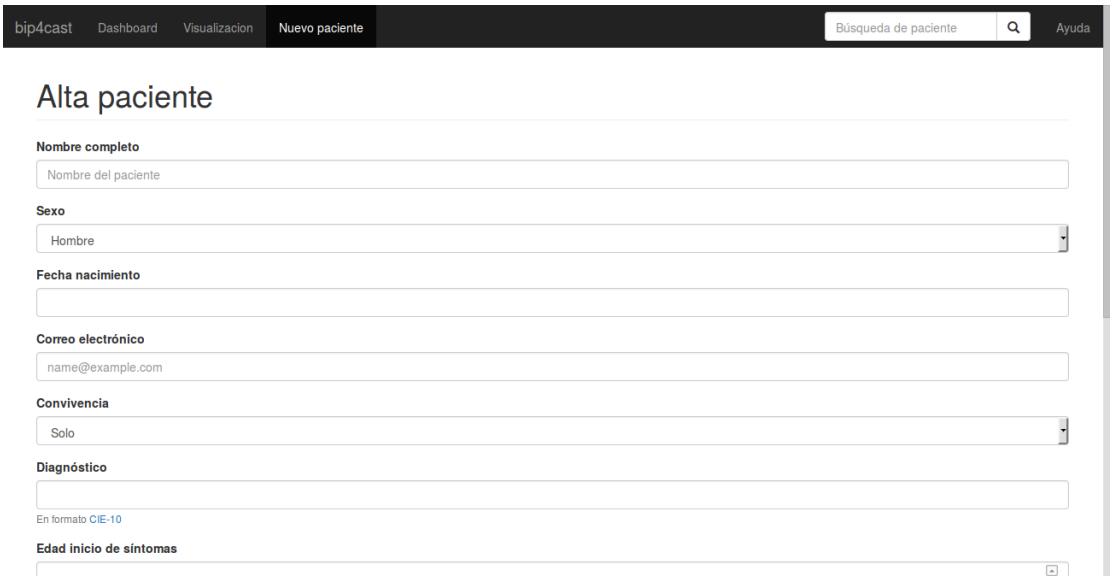

Figura 3.4: Vista del alta de paciente - 1.

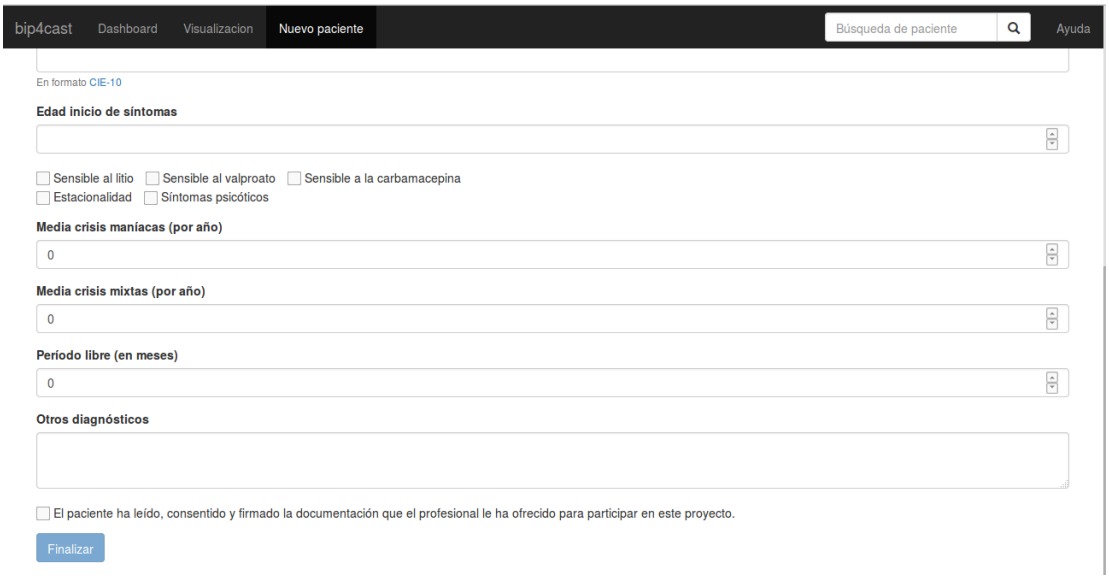

Figura 3.5: Vista del alta de paciente - 2.

ción 1.2) que incluyen datos personales, como edad o sexo, además de aspectos relativos a la enfermedad como la edad de inicio de los síntomas o el número de crisis al año, así como aspectos farmacológicos como posibles alergias a la medicación tradicional (litio, carbamacepina y valproato).

Para concluir, el botón para finalizar el proceso de alta se habilita cuando el médico afirma que el paciente ha dado su consentimiento escrito para participar en el programa. Una vez que el alta se completa, se asigna al nuevo paciente un identificador único y sus datos serán agregados a la base de datos MongoDB.

3.4.2. Resultados de la búsqueda

En la barra de navegación aparece una barra de búsqueda para los pacientes. La cadena de texto introducida servirá para localizar a los pacientes por nombre. La búsqueda se realiza sobre el campo *nombre completo* del paciente, de forma que es capaz de localizar nombres que contengan total o parcialmente la cadena introducida.

Los resultados de la búsqueda se muestran en una nueva pantalla. Si no se han encontrado resultados, un mensaje de información lo notifica. En la Figura 3.6 puede verse la vista generada al buscar “jai” en la barra superior.

Junto a los resultados y a título informativo, aparecen la fecha de nacimiento del paciente y su sexo. El nombre del paciente es un hipervínculo al perfil médico que sobre él guarda la aplicación.

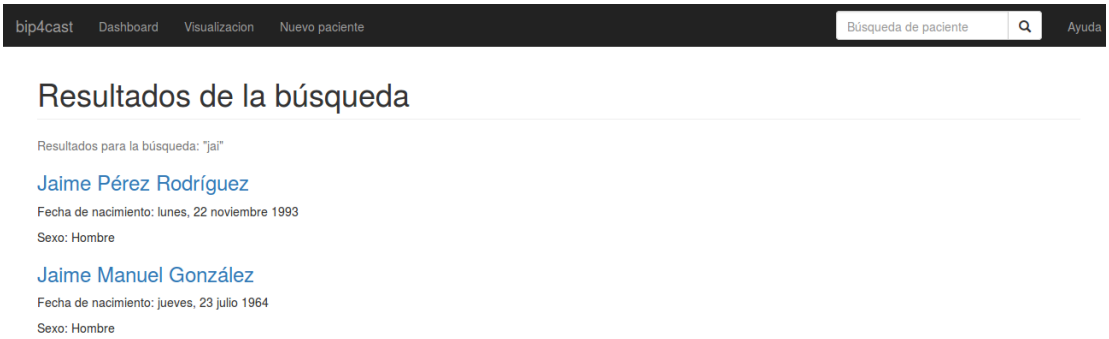

Figura 3.6: Resultados para la búsqueda “jai”.

3.4.3. El perfil del paciente

Pinchar en el nombre del paciente nos redirige a la ventana principal de su perfil. A la izquierda aparece un nuevo menú, sólo activo cuando se visitan las distintas vistas que componen el perfil del paciente para la navegación entre ellas. Este menú lateral se compone de: principal, medicación, test y comunicación.

Principal

La ventana principal, como puede observarse en la Figura 3.7, muestra los datos personales y fenotípicos del paciente, los mismos que se describieron en la Subsección 3.4.1. A la derecha aparecen sendos botones para modificar los datos del paciente o bien para eliminarlos.

Si se pulsa el botón “Editar paciente”, se despliega un *modal* (Figura 3.8) que muestra un formulario similar al de Alta de pacientes, con los datos actualmente guardados rellenos en cada campo. Cuando el usuario termine de hacer las modificaciones deseadas, puede validar los cambios. Un mensaje de confirmación informará de las modificaciones realizadas (Figura 3.9).

Algo similar ocurre cuando se pulsa en el botón “Eliminar paciente” (Figura 3.10). Esta es la forma de dar de baja un paciente del sistema y eliminar con él todos los datos que el sistema posee (incluidos mensajes, recetas, resultados de test,...). Puesto que los cambios son irreversibles, se pide confirmación para proceder a la eliminación definitiva.

Figura 3.7: Ventana principal del paciente.

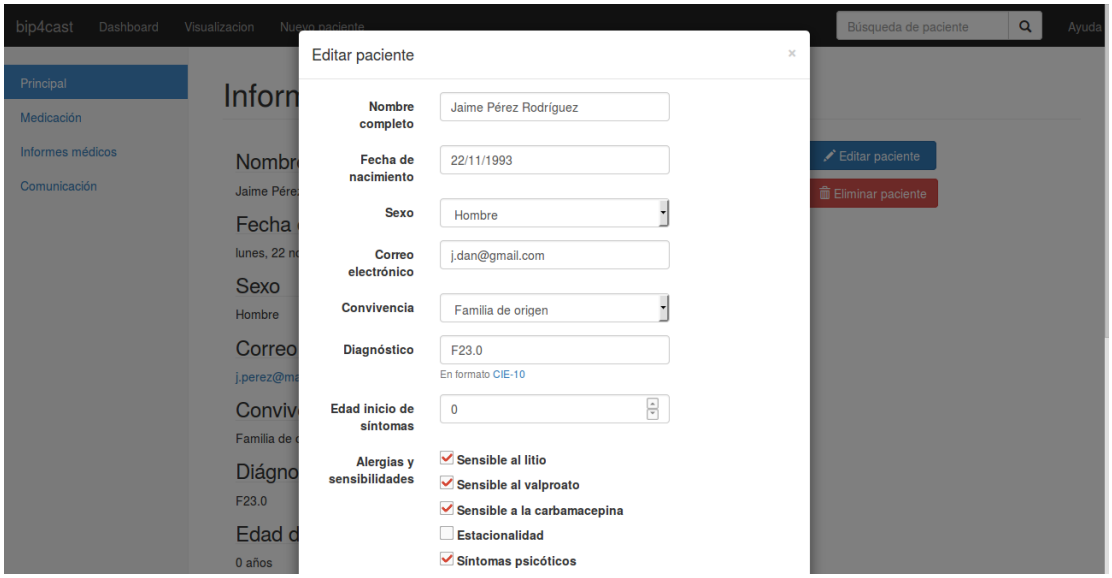

Figura 3.8: *Modal* desplegado para editar un paciente.

Figura 3.9: Mensaje de confirmación tras editar los datos del paciente.

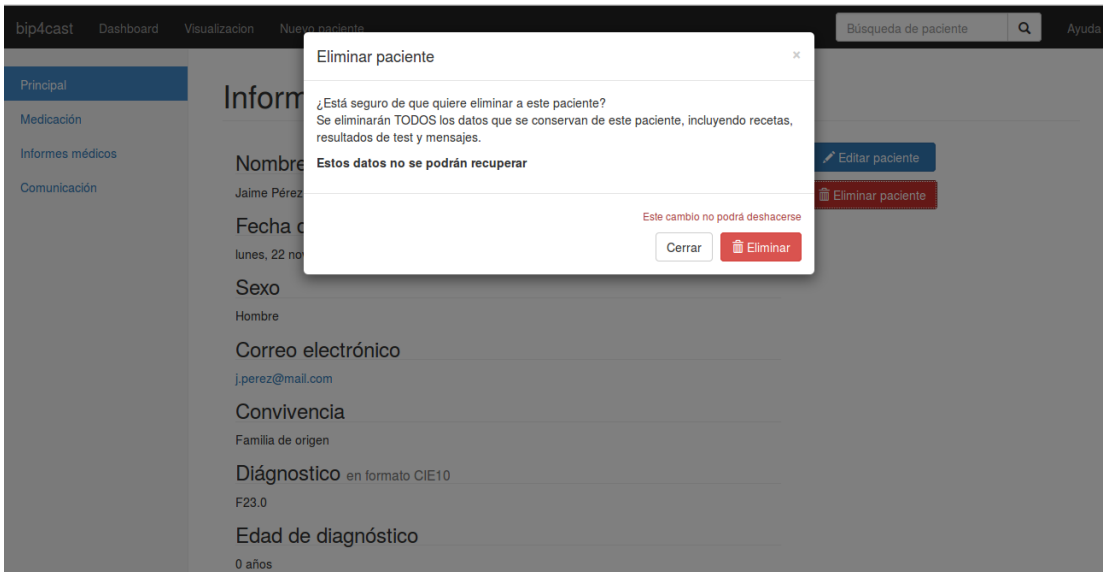

Figura 3.10: *Modal* para la eliminación del paciente.

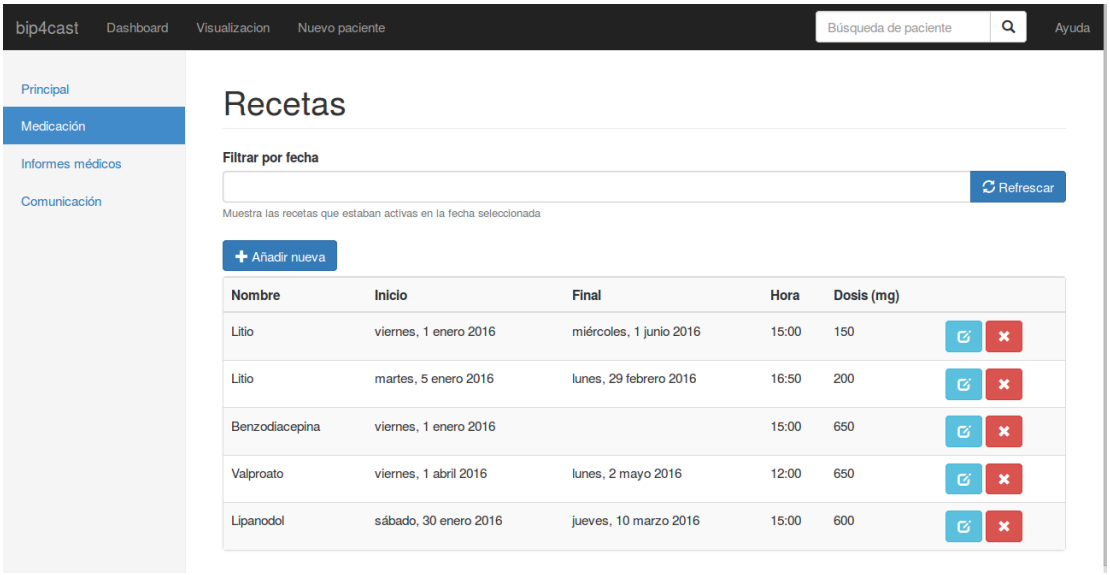

Figura 3.11: Vista de la pantalla de recetas.

Recetas

La ventana de recetas (Figura 3.11) se encarga de la gestión de las prescripciones médicas a los pacientes. Todas estas aparecen en la parte baja de la ventana, contenidas en una tabla que muestra el nombre, las fechas en las que se encuetran activas, la hora y la dosis prescrita, así como sendos botones para editar y eliminar cada receta de forma individual.

Así mismo, se dispone de un campo de texto, que permite filtrar las recetas por fecha de actividad, es decir, se filtrarán las recetas que estaban activas el día introducido en el campo y estas aparecerán en la tabla.

El botón para añadir una nueva receta separa la barra de filtrado de la tabla que incluye las recetas. Este botón despliega un *modal* con un formulario que permite agregar nuevas prescripciones.

Cada receta dispone de dos botones, representados con los símbolos de editar y eliminar, para realizar dichas acciones con la prescripción. Ambos funcionan de forma similar, pues cuando son accionados despliegan sendos *modals* para editar los datos guardados o para eliminarlos, respectivamente. Es importante hacer notar que el *modal* para agregar un nuevo medicamento y para editarlo es el mismo (es decir, vienen del mismo fragmento HTML), un pequeño *script* se encarga de modificar su comportamiento dinámicamente para adaptarse a una u otra situación. Un ejemplo de modal puede verse en la Figura 3.12.

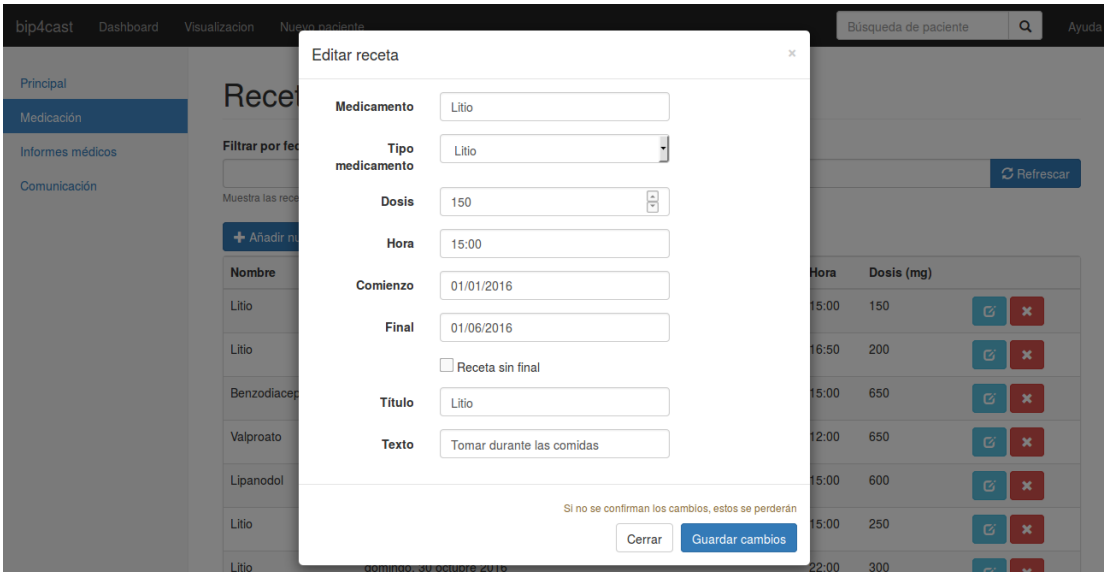

Figura 3.12: *Modal* para editar la información de una receta.

Comunicación

Como se menciono en la Sección 1.2, los mensajes y las recetas son muy parecidos. Tanto es así que esta vista está basada en la de los mensajes, con una apariencia muy similar (Figura 3.13).

De nuevo, hay dos elementos principales: un campo de texto para filtrar los mensajes activos en la fecha introducida y una tabla que muestra los mensajes. Están separadas por un botón que permite agregar un nuevo mensaje. Este botón, al pulsarse, despliega un *modal* que muestra un formulario para introducir los datos.

De la misma forma que en las recetas, la tabla de mensajes muestra inicio y final de mensaje, hora y si es programado o no, así como el texto. Junto a todos estos datos, aparecen dos botones: para editar o eliminar el mensaje.

Los test

La vista de informes médicos (Figura 3.14) muestra los resultados del test más reciente del que se dispone.

En la parte superior se muestra una campo de texto para localizar el informe en la fecha correspondiente a la fecha introducida. Debajo, dos botones permiten añadir un nuevo test y buscar entre los existentes. Esta búsqueda examina los test realizados en el intervalo de fechas seleccionadas junto al test o los test que se realizaron en esa fecha.

La parte inferior muestra los resultados. Se ha utilizado un panel de pestañas *tabpane* de Bootstrap como contenedor de los resultados. Así, cada pestaña muestra los resultados pormenorizados de cada tipo de test, aprovechando el gran espacio horizontal de la parte

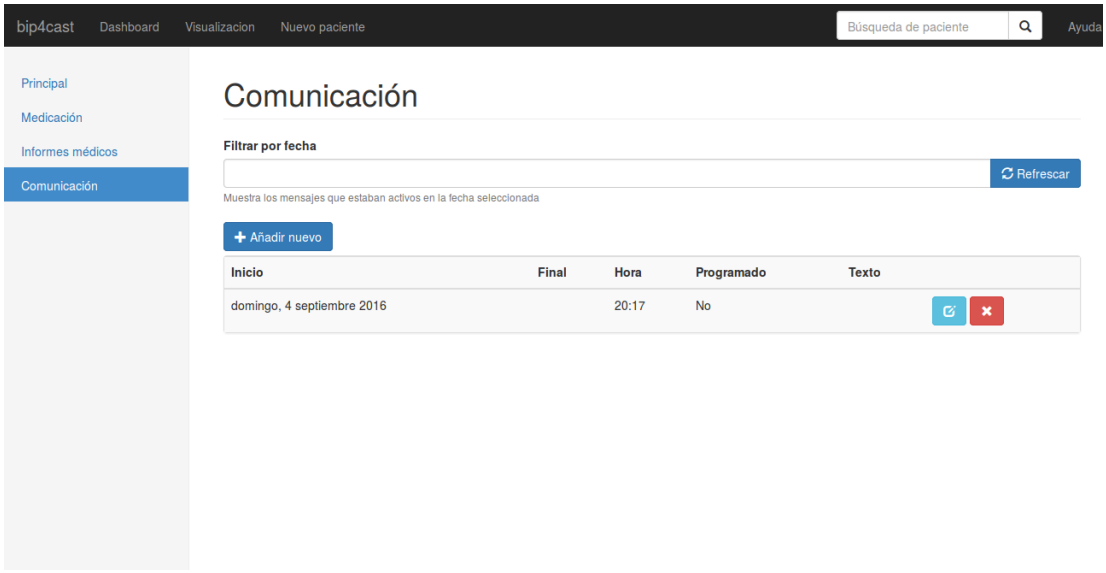

Figura 3.13: Vista de la pantalla de comunicaciones.

Figura 3.14: Vista principal de la ventana de test.

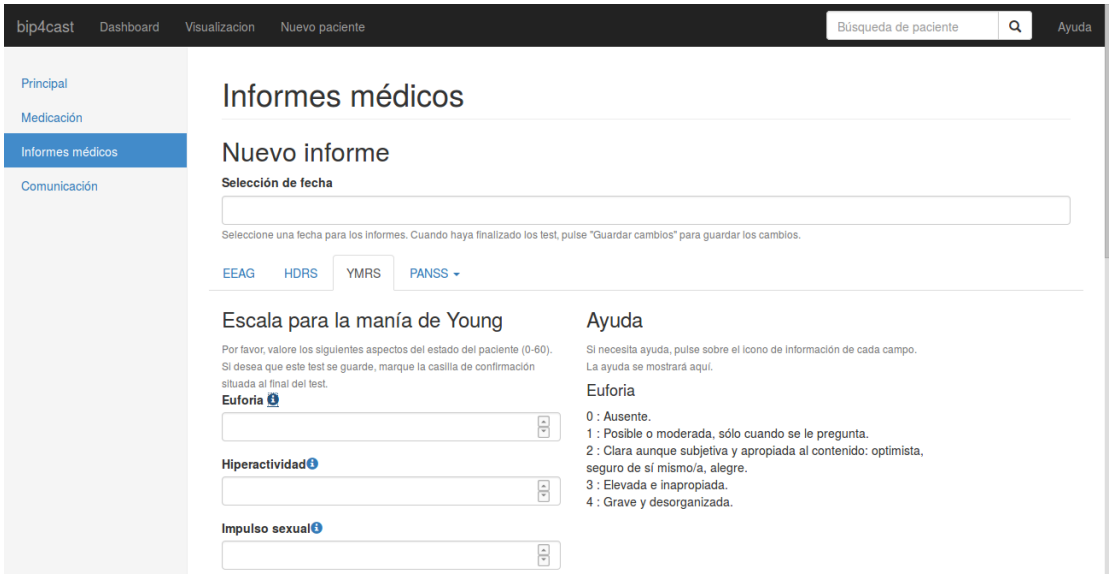

Figura 3.15: La ayuda aparece en la parte derecha de la pantalla.

baja. Este panel cambia dinámicamente en función del test mostrado. Los resultados de cada escala del PANSS están agrupados en un menú contráctil (concretamente, un *collapse-pane*) organizado como un acordeón. Cada escala despliega sus resultados al ser seleccionada. La suma total de los valores de cada test aparece al lado del nombre dentro de un pequeño círculo (un *badge*).

Añadir nuevos test

El botón de añadir nuevos test conduce a una nueva vista que sirve para añadir nuevos test a la base de datos.

Se dispone de un campo de texto para introducir la fecha del test. De forma similar a la vista de los resultados de los test, la parte inferior dispone de un panel de pestañas (*tabpane*) donde cada pestaña contiene un test. Las escalas del PANSS aparecen en un desplegable. La navegación entre cada una de las pestañas cambia el panel inferior de forma dinámica apareciendo éste partido en dos: la mitad izquierda contiene cada uno de los aspectos que el test valora, acompañado de un icono de una *i* minúscula. Éste activa la mitad derecha del panel, que muestra la ayuda individualizada de cada pregunta. Esta ayuda avanza a medida que lo hace el *scroll* del usuario (Figura 3.15), de forma que la ayuda está siempre presente, incluso en las preguntas que se encuentran en las últimas posiciones.

Al final de cada test se muestra una casilla de comprobación que indica al sistema que esos datos deben ser guardados (podría darse el caso de querer pasar un único test a un paciente un día concreto). Para finalizar el proceso, debe pulsarse el botón “Guardar datos”. En ese momento, un *script* comprueba que los datos introducidos son correctos, validándolos, y en caso de que haya valores que no son los adecuados el usuario es

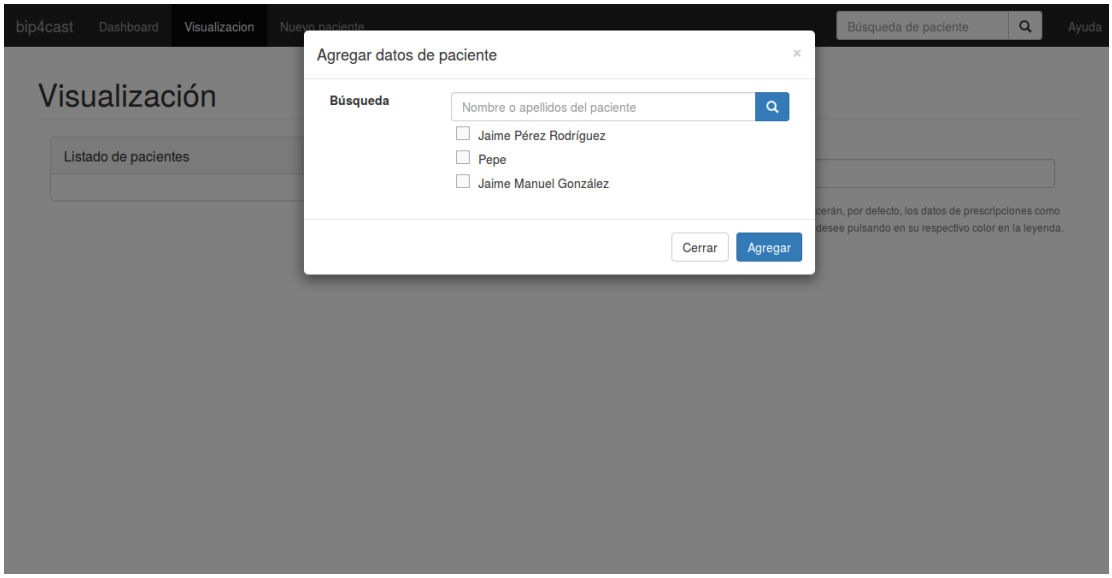

Figura 3.16: *Modal* para la búsqueda de pacientes.

notificado. Los datos no serán salvados hasta que los datos introducidos sean correctos. Un mensaje de confirmación informará al usuario de que el proceso ha finalizado.

3.4.4. Visualización

La última de las opciones del menú de navegación superior es la de la visualización. En esta vista se representarán gráficamente los datos almacenados de cada paciente relativos a medicación y resultados de los test.

La vista está dividida en dos: la parte izquierda tiene un contenedor que almecena los pacientes cuyos datos van a ser representados. Para agregar nuevos pacientes, se despliega el *modal* que se ve en la Figura 3.16 que permite buscarlos en la base de datos, con un formato similar a la barra de búsqueda del menú superior. Introducida la consulta y devueltos los resultados por el sistema, podemos seleccionar qué pacientes queremos agregar a la vista principal para representar los datos. Los agregados aparecerán en el contenedor "Listado de pacientes". Una vez introducido el intervalo de fechas que se quiere representar en los campos correspondientes, seleccionando un paciente y una gráfica, en la parte derecha aparece esta con los datos seleccionados.

De este apartado se hablará más en profundidad en el Capítulo 5.

4 Implementación

Habiendo hecho un repaso generalizado de la arquitectura de la plataforma y en lo referente a las tecnologías utilizadas para la construcción del servicio web, vamos a comentar en el presente capítulo detalles de la implementación tanto en el lado del servidor como en el lado del cliente.

La aplicación web está compuesta de numerosos ficheros y directorios. El árbol de directorios que se presenta en la Figura 4.1 corresponde con la estructura "pública" de la aplicación (tal y como se muestra en el repositorio público de GitHub). Se han omitido algunas carpetas irrelevantes al objeto del presente capítulo, como el directorio donde se guardan los módulos de node o los logs del servicio.

4.1. Arquitectura

Para el desarrollo de esta aplicación web se ha optado por una arquitectura basada en tres capas (*three-tier*):

Capa de presentación Constituida por el navegador web cliente. En este caso compuesta por las distintas vistas HTML y las librerías JavaScript utilizadas desde el propio navegador.

Capa de proceso La capa de proceso la compone el entorno de ejecución Node.js acompañado del framework Express.js para la gestión de las peticiones HTTP. Además, integra todos los módulos de Node.js necesarios. Toda la lógica de la aplicación está contenida en esta capa.

Capa de datos La constituye básicamente una base de datos MongoDB dividida en tres colecciones. Esta base de datos es compartida entre el servicio web y la aplicación Android de la plataforma. Puesto que el acceso requerido por la aplicación web se reduce únicamente a tres de las colecciones contenidas en esa base de datos, para simplificar asumiremos que la base de datos está compuesta únicamente de las tres colecciones, a las que nos referiremos más adelante.

La Figura 4.2 muestra un esquema de la arquitectura diseñada para esta aplicación.

4.2. El servidor

El servidor es un servicio Node.js, tal y como se dijo en la Sección 3.2. Sobre él, hay instalados diversos módulos, siendo el más importante Express. En efecto, es la pieza

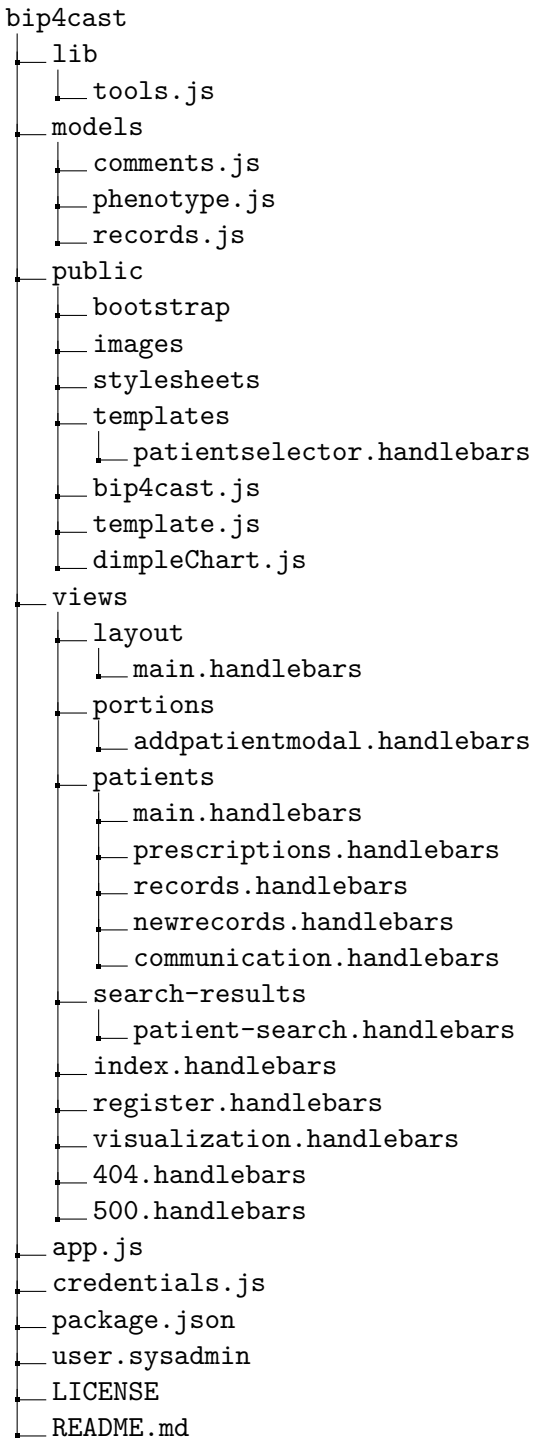

Figura 4.1: Árbol de directorios del proyecto.

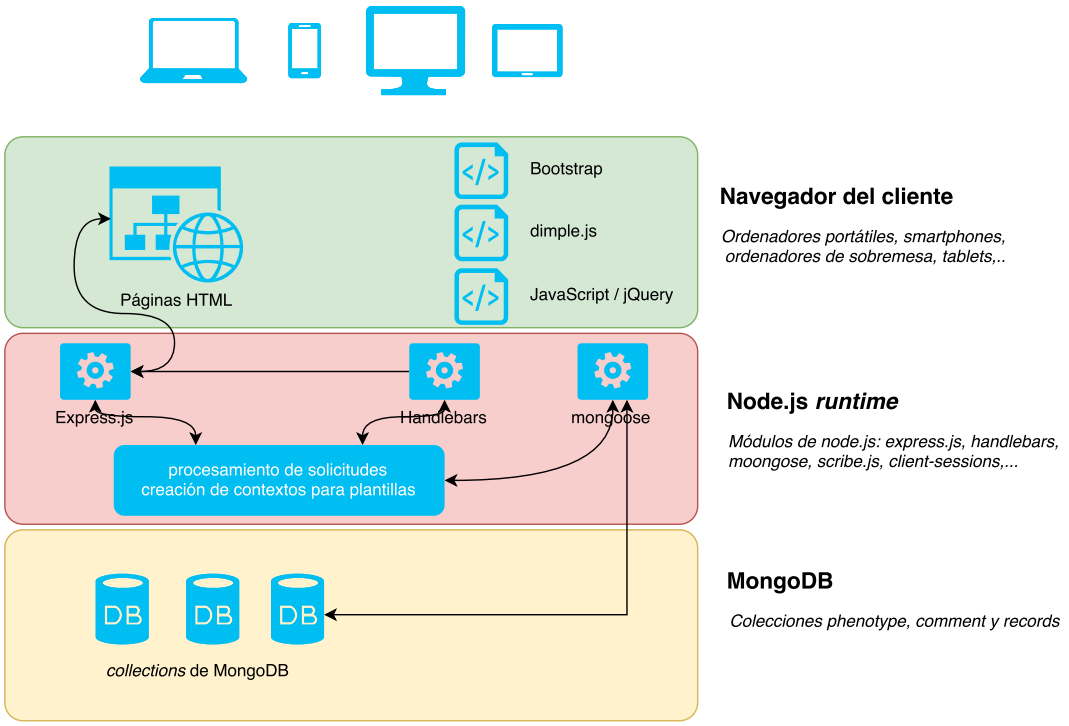

Figura 4.2: Arquitectura de la aplicación web *bip4cast*.

más importante del servicio puesto que es el encargado de la gestión de las rutas URL (es decir, es el responsable de gestionar el tráfico HTTP). Cada una de las posibles URL dentro de servicio son atendidas por una función dentro del fichero principal de la aplicación, `app.js`, donde se realiza una implementación de la atención de la petición mediante una función anónima. De la misma forma, el procesamiento de los formularios, vía POST, se realiza también dentro de este fichero JavaScript.

Express.js no es el único módulo instalado. También se dispone de los siguientes, de los que iremos detallando su cometido:

path Incluido en la distribución de Node. Permite operaciones de acceso a archivos y directorios.

serve-favicon Pequeño middleware[1] para proporcionar un *favicon*.

cookie-parser Otro middleware, en este caso, para acceder y parsear *cookies*.

body-parser Middleware para parsear el cuerpo de una petición (objeto *req* que reciben todas las funciones que tratan peticiones). De esta forma, cuando el manejador de la petición HTTP se ejecuta, pueden accederse a los atributos de la petición ([Brown, 2014, pp. 57-60]). Esto es muy útil, por ejemplo, para leer los campos y sus valores al procesar un formulario.

[1] Pequeño programa auxiliar de Node.js

client-sessions Permite gestionar las sesiones. Las sesiones se utilizan por el momento para el envío de mensajes al navegador cliente (por ejemplo, para confirmar que una operación de editar una receta se ha realizado satisfactoriamente).

connect Connect es el framework encargado de gestionar los middleware que procesan cada petición (*request*) antes que el manejador de esa solicitud actúe. En particular se utiliza junto con el módulo `connect-rest`.

connect-rest Middleware para `connect` para la implementación de una API REST. En el caso de *bip4cast* utilizado para la obtención de datos de medicación y test para representar en las gráficas de visualización.

scribe-js Framework que implementa un sistema de logging al que puede accederse vía web. Guarda registros en ficheros JSON, organizados por días y por categorías indicadas por colores según el nivel de importancia del mensaje.

http-auth Paquete para la gestión de autenticación vía HTTP. Utilizado para acceder al panel de control de los registros de log de Scribe.js.

express-handlebars Motor de renderizado de Handlebars diseñado específicamente para Express.js.

express Módulo de Express.

mongoose Librería que proporciona conexión y acceso a una base de datos MongoDB. Más concretamente es un Object Data Mapping (ODM), es decir, su cometido es hacer la traducción de cada colección a un objeto JavaScript. De esta forma, podemos tratar las distintas colecciones de la base de datos como objetos, lo cual simplifica enormemente su interacción con ellas.

El fichero comienza con la carga de los módulos listados más arriba y la inicialización del sistema de logger y de Express. Después se enlazan los distintos módulos con la instancia de Express lanzada, se configuran los directorios de objetos estáticos (hojas de estilos, scripts, etc.). Se crean las sesiones y se conecta a la base de datos, creándose los objetos `phenotype`, `records` y `comments`, derivados de los Schemas de `moongose`. A partir de este momento puede interactuarse con cada una de estas colecciones vía JavaScript.

El siguiente bloque corresponde con la gestión de cada una de las rutas dentro de la aplicación, es decir, se implementan las funciones que manejan cada URL solicitada al servidor, así como el procesamiento de los formularios. Después, las dos funciones de la API de obtención de datos para dibujar las gráficas del apartado de visualización. Y por último, los colectores (*catchers*) de errores 404 y 505.

4.2.1. La API

Se ha implementado una pequeña API para servir datos de pacientes. Se utilizan para obtener los datos que visualizar en las gráficas, pero en un futuro podrían servir para formar con todos los datos de que se dispongan un repositorio *open data* público. De esta forma, cualquiera podría obtener los datos convenientemente anonimizados y construir

herramientas propias con ellos, enriqueciendo el ecosistema *bip4cast*.

Se ha utilizado el middleware `connect-rest` para configurar las rutas de esta API. A modo de ejemplo, se muestra en el Código 4.1 la implemetación de la función de la API que sirve datos de medicación.

```
1  rest.post('/prescriptions/:pat_id', function(req,
2  content, cb){ // obtain pat_id from url
3    var pat_id = req.params.pat_id;
4    var begindate = tools.toDate(content.beginDate);
5    var enddate = tools.toDate(content.endDate);
6    // search data in database
7    comments.find({ user_id : pat_id, prescription : true}, function(err,
         results){
8      if(err) return cb({error : 'Internal server error.'});
9      if(results.length === 0)
10       cb(null, results);
11     else{
12       var data = results.map(function(p){
13         return{
14           beginDate : p.dateStart,
15           endDate : p.dateEnd,
16           type : p.type
17         }
18       });
19       // send results
20       cb(null, tools.createPrescriptionsJSON(data, begindate, enddate));
21     }
22   });
23 });
```

Código 4.1: Implementación de la función para obtener datos de medicación de la API.

4.2.2. tools.js

Este fichero auxiliar contiene gran cantidad de funciones utilizadas en toda la aplicación. Se trata en su mayor parte de funciones para procesar los datos de los formularios o para formatear datos que provienen de la base de datos antes de ser enviados a la plantilla.

Más concretamente, procesa datos en general como pueden ser fechas (convirtiendo el formato Date y strings en castellano y viceversa) y horas; datos de paciente como la convivencia con la familia o género; relativos a los test como el procesamiento de éstos cuando se agregan nuevos o viceversa (al ser mostrados en pantalla para su consulta) y para la creación de objetos JSON a partir de los datos almacenados que se sirven al API cuando se solicitan datos para las gráficas.

4.3. La base de datos

Todos los datos que maneja la aplicación web *bip4cast* están almacenados en una base de datos MongoDB.

La base está dividida en tres colecciones: `phenotype`, `comments` y `records`.

4.3.1. Colección `phenotypes`

Esta colección contiene los datos personales y fenotípicos de cada paciente, así como su identificador (ver Subsección 3.4.1). Cada documento de esta colección tiene los siguientes atributos:

name De tipo String. Nombre completo del paciente.

email De tipo String. Una dirección de correo electrónico válida.

birthDate De tipo Date. Fecha de nacimiento del paciente.

gender De tipo Boolean. Sexo del paciente: `false` si masculino, `true` si femenino.

pin De tipo Number. Es el PIN asignado al paciente para usarlo en la aplicación móvil.

cohabitation De tipo Number. Indica con quién vive el paciente. Puede tomar uno de los siguientes valores: 0 si vive solo, 1 si lo hace en pareja, 2 si lo hace con pareja e hijos, 3 si lo hace con su familia de origen y 4 si no es ninguna de las anteriores (otros casos).

diagnosis De tipo String. Diágnostico del paciente en formato CIE-10.

diagnosisAge De tipo Number. Edad de inicio de los síntomas.

senLit De tipo Boolean. Indica si el paciente es sensible o alérgico al litio.

senVal De tipo Boolean. Indica si el paciente es sensible o alérgico al valproato.

senCar De tipo Boolean. Indica si el paciente es sensible o alérgico a la carbamacepina.

seasonality De tipo Boolean. Indica si el paciente sufre de estacionalidad en sus crisis.

psycSymp De tipo Boolean. Indica si el paciente padece síntomas psicóticos.

maniaCrises De tipo Number. Media de crisis maníacas al año.

mixedCrises De tipo Number. Media de crisis mixtas al año.

freePeriod De tipo Number. Número de meses que el paciente está libre de síntomas al año.

others De tipo String. Puede albergar observaciones u otra información relevante del paciente.

El atributo `_id` que identifica al documento (y que MongoDB genera y asigna por defecto a cada elemento contenido en una colección) servirá como identificador de paciente en la aplicación.

4.3.2. Colección `comments`

Esta colección tiene una doble función: almacenar las recetas y los mensajes. Ya hemos comentado anteriormente que las recetas y los mensajes comparten cierta estructura, y con el objetivo de evitar duplicidades y mejorar el almacenamiento, se ha decidido que tanto las recetas como los mensajes compartan colección.

Los atributos comunes tanto a recetas como a mensajes son:

user_id De tipo String. Identificador de usuario al que pertenece este documento.

prescription De tipo Boolean. Si es `true`, este documento es una receta médica, en caso contrario, se trata de un mensaje.

dateStart De tipo Date. Fecha de inicio del elemento.

dateEnd De tipo Date. Fecha de final del elemento.

time De tipo Number. La hora del elemento, representada como los minutos transcurridos desde las 00:00 (por ejemplo, 901 hace referencia a las 15:01).

text De tipo String. Cuerpo del mensaje u observaciones de la toma del medicamento en el otro caso.

Los campos propios de las recetas son los siguientes:

name De tipo String. Nombre del medicamento o principio activo.

type De tipo Number. Indica el tipo de medicamento: 0 si es litio, 1 si es un antiepiléptico, 2 si es antipsicótico, 3 si es un ansiolítioco o un hipnótico, 4 si es un antidepresivo o 5 si es de otra categoría.

dose De tipo Number. Dosis en miligramos del medicamento prescrito.

title De tipo String. Breve resumen o descripción de la toma.

Puede haber recetas que no tengan una fecha final, sino que están prescritas por el médico hasta que éste realice nuevos cambios. Internamente, estas recetas tendrán por fecha final (atributo `dateEnd`) el valor `null`.

De forma similar, los mensajes pueden enviarse de forma periódica (como si fuesen recordatorios programados) en un intervalo de tiempo concreto o a una hora determinada. Más concretamente, si el la fecha de final es nula (valor `null`) se entiende que el mensaje se enviará una vez en la fecha indicada (fecha de inicio).

4.3.3. Colección `records`

Esta colección almacena los resultados de los test que el profesional médico realiza a cada paciente. Los atributos de los documentos de esta colección son:

user_id De tipo String. Identificador del usuario al que pertenece el test.

date De tipo Date. Fecha en que se realizó el test.

eeag De tipo Number. Valor del resultado del test EEAG.

hdrs De tipo Document. Guarda de forma individualizada los resultados de cada pre-

gunta. Todos los campos son de tipo Number.

ymrs De tipo Document. Igual que el anterior, contiene los valores de cada una de las preguntas que forman parte del test. Todos los campos son de tipo Number.

panss De tipo Document. Contiene otros tres documentos, correspondientes a cada uno de los escalas del PANSS. Igual que los anteriores, guarda las respuestas de las preguntas de cada test. Todos los campos de cada documento son de tipo Number.

La conexión con la base de datos y las distintas transacciones se realizan a través de `mongoose`. Para ello, se dispone de tres ficheros, dentro del directorio `models`, que contienen los Schemas que establecen la correspondencia entre la colección y un objeto JavaScript con el que manejarla. Estos ficheros crean el Schema, lo asocian a su correspondiente colección de MongoDB y exportan un objeto que será utilizado en `app.js`, el archivo principal de la aplicación.

4.4. Las plantillas Handlebars

Todas las plantillas están almacenadas dentro del directorio `views`.

Las plantillas principales se encuentran en el propio directorio. Hablamos de la de la página principal (`index.handlebars`), la de alta de pacientes (`register.handlebars`), la de visualización (`visualization.handlebars`) y las que muestran los errores (`404.handlebars` y `505.handlebars` respectivamente).

Por otro lado, dentro del directorio `patients` se encuentran las plantillas de cada una de las opciones del perfil de paciente: la ventana principal (`main.handlebars`), la del control de la medicación (`prescription.handlebars`), los mensajes (`comunication.handlebars`) y los test (`records.handlebars`), además de la vista para añadir nuevos test (`newrecord.handlebars`).

El directorio `layout` contiene el fichero `main.handlebars`, que es el layout principal. Esta es la plantilla común a todas las vistas servidas, y es donde los fragmentos HTML del resto de plantillas se incrustan. Como tal, es el fichero que más se parece a un HTML final, por lo que incluye la declaración HTML y el encabezado de la misma y los distintos scripts (D3.js, jQuery,...) y ficheros CSS necesarios. Dentro del tag `body` están los elementos visibles de las vistas, como la barra de navegación y el menú de perfil de paciente (cuando este aparece), además del *container* que incluye la parte principal de cada vista. Se incluye en la parte final un *helper* para que cada plantilla pueda incluir un bloque de scripts si es necesario.

4.5. En el lado del cliente

El trabajo de implementación en el lado del cliente descansa fundamentalmente en: la creación de las plantillas para cada una de las vistas, que ya hemos comentado anteriormente y el *scripting* con JavaScript y jQuery para agregar funcionalidad y dinamismo a cada una de las vistas.

4.5.1. Los scripts en las vistas

Todas las vistas traen cargadas consigo jQuery, que es la principal forma de interacción con el DOM que hemos utilizado. Se ha aprovechado el *helper* dejado en el *layout* principal para ubicar esas líneas de código.

Uno de los usos más típicos del *scripting* en el lado del cliente es para la validación de formularios antes de que éstos sean enviados al servidor para su procesamiento. Este será uno de los usos más extendidos del scripting en toda la aplicación debido a la gran interacción a través de formularios que el médico tendrá con el sistema.

Se ha utilizado un plugin jQuery llamado `inputmask` para poner máscaras en los campos de texsto que requieren un formato especial, como las fechas. Este plugin reduce el proceso de validación de cada una de las vistas.

visualization.handlebars

Esta es sin duda la vista que más trabajo por parte del motor JavaScript del navegador cliente requiere. No es de extrañar dado el papel que tiene dado su carácter más dinámico y visual. En este caso se agregan además las librerías de dimple.js, D3.js (necesario para ésta última) y Handlebars.

La primera parte consta de pequeñas funciones para dotar de funcionalidad a la vista, como por ejemplo para añadir nuevos pacientes al panel a partir de los resultados de búsqueda del *modal*, o la función para alternar entre los distintos dataset a representar gráficamente, que incluye las llamadas AJAX a la API para obtener nuevos datos a dibujar así como eliminar los previamente representados.

dimpleChart.js

Es el script encargado de la representación gráfica. El cuerpo principal del script está constituido por la declaración de las variables necesarias para dibujar: el objeto `myChart` representa la tabla y es a través del cual se interactúa. A `myChart` se le agregan las distintas *series* de datos que contienen cada uno de los dataset a representar (uno por cada test, hasta 6 en total; y otro por la medicación), además de configuración de los ejes y la leyenda.

Existen 3 tipos distintos de gráficas: la llamada "Distribución", que representa tanto las tomas de medicamentos como los resultados de los test, histogramas y mapas de calor. Para las dos últimas, se visualiza uno por cada tipo de medicamento. Las dos primeras se han implementado en dimple.js, mientas que el mapa de calor lo ha sido directamente en D3.js. Esto hace que el mapa de calor tenga un trato ligeramente diferenciado frente a las otras dos representaciones.

Cada vez que se desea cambiar el usuario cuya información desea visualizarse o la gráfica a representar se destruye la tabla y las series y vuelven a inicializarse vacíos. Este es el propósito de las funciones `initDistributionChart()`, `initHistogramChart()` y `deleteChart()` respectivamente.

Por otro lado, y puesto que cada *serie* comienza con los datos vacíos, estos deben ser rellenados. Esa labor la realizan, en el caso de la gráfica "Distribución", las funciones `loadPrescriptionDataset(dataset)` y `loadRecordsDataset(dataset)`, que inicializan los datos de cada serie correspondiente a la medicación y a los test respectivamente. En el caso del histograma, esa tarea recae en `loadPrescriptionHistogramDataset(drug, dataset)` y para el mapa de calor `loadPrescriptionHistogramDataset(drug, dataset)`. Todas estas funciones realizan las labores de filtrado correspondientes en función de la gráfica a representar.

Dotar a la layenda de la gráfica "Distribución" de interactividad es el principal cometido de la función `afterFirstDraw()`, que se ejecuta tras dibujar dicha gráfica, una vez que la leyenda está inicializada. La idea parte de desvincular la leyenda del gráfico para trabajar con ella de forma independiente. Se basa en lo siguiente: un *listener* detecta los clicks sobre los cuadros de la leyenda y cuando estos se producen se averiguan qué cuadros siguen activos (es decir, sus respectivos datos tienen que seguir apareciendo en la gráfica). A continuación, los siete datasets involucrados se mezclan (con ayuda de la función `mergeSeriesData()` para luego filtrarse, descartando aquellos que no van a ser representados. Una vez que el dataset está filtrado, se invoca a las funciones de carga arriba mencionadas y la gráfica se vuelve a redibujar.

En el perfil del paciente

En las distintas vistas que conforman el perfil del paciente, los scripts implementados están orientados a la validación de datos y a la interacción con algunos elementos propios de Bootstrap como pueden ser *modals*.

Así, en la vista principal (`main.handlebars`) existe una función que se ejecuta al desplegarse el *modal* para editar los datos del paciente que sirve para rellenar los campos con los valores ya existentes y evitar que el usuario tenga que volver a introducir datos que no desea modificar. Así mismo, se implementa otra función que comprueba la validez de los datos introducidos.

Por otro lado, en la vista de las recetas (`prescription.handlebars`) se implementan *listeners* asociados sendos *modals* (editar o añadir receta y eliminarla) y que se activan cuando al ser mostrados. En la Sección 3.4.3 explicamos que el mismo *modal* servía tanto para añadir una nueva receta como para editarla. Modificar dinámicamente el comportamiento de dicho *modal* en función de la acción que desee realizarse es el cometido de la primera función implementada. Averiguando qué botón lanzó el evento de desplegar el formulario se ajusta su comportamiento. En caso de que dicha acción sea editar una receta, los datos ya guardados se colocan en sus respectivos campos, siendo así innecesario reescribir valores que no desean ser modificados. Además, para evitar posibles interferencias cuando este *modal* se comporta de una forma u otra, proporcionamos una función que resetea los campos del formulario. El otro *listener* se encarga de configurar el desplegable para eliminar una receta, solicitando confirmación al usuario antes de realizar la acción. Por último, una última función comprueba que existe una fecha válida antes de solicitar filtrar las recetas.

En el caso de la vista de mensajes (`communication.handlebars`) aplica la totalidad de lo expuesto en el párrafo anterior, al tener ambas estrcturas y utilidad similar.

Para la vista de los test (`records.handlebars`), la función implementada valida la fecha introducida antes de filtrar los test.

Para finalizar, la vista que crea nuevos test tiene dos funciones orientadas a la validación: la primera valida que el valor introducido en cada campo de cada pregunta es el adecuado a través de un *listener* que hay asociado a cada campo de texto y que se ejecuta cuando éste deja de estar seleccionado (esto se realiza obteniendo los atributos `min`, `max` y `step` de cada pregunta y viendo si el valor es correcto) además de ir calculando la puntuación total del test hasta el momento. Si el valor introducido es correcto, el campo se sombrea de verde (es decir, se cambia dinámicamente la clase del campo de texto para dar *feedback* al usuario). De forma similar ocurre si el valor no es correcto, sombreándose el campo en rojo. Para lograr esto se modifican las clases de los campos de texto a través del *listener* asociado, pasando a ser estas las que Bootstrap incorpora para señalizar errores o aciertos de esta forma. Además de sombrear los campos de texto, en caso de fallo se despliega un *popover* (un pequeño bocadillo flotante) que se destruye cuando el valor introducido es correcto.

La otra función implementada en esta vista valida, antes de enviar los test para su procesado y almacenamiento, que de aquellos test que contienen datos que tienen que ser guardados (es decir, tienen marcada la casilla “Datos válidos”) son correctos.

4.5.2. Herramientas auxiliares

Plantilla del lado del cliente

Todas las plantillas de Handlebars se renderizan en el servidor. Todas salvo una, que se encuentra en la vista de visualización. Es la que se ocupa de mostrar los pacientes que desean ser agregados al contenedor “Lista de pacientes” de la parte izquierda de la vista.

Handlebars funciona aquí de la misma forma que si estuviese en el lado del servidor: el contexto se compila y se muestra el fragmento HTML resultante. ¿Por qué es necesario hacerlo del lado del cliente? El proceso de añadir nuevos pacientes al panel lateral izquierdo es un proceso enteramente en el lado del cliente, que no depende en absoluto del servidor. Sería absurdo mandar una petición al servidor para renderizar un pedazo de plantilla. No ocurre esto, por ejemplo, en los resultados mostrados en el *modal* de búsqueda: puesto que en ese caso se realiza una consulta AJAX al servidor, este retorna los resultados ya formateados y la propia función AJAX agrega directamente el código HTML en el pequeño `div` del *modal* destinado a ello.

bip4cast.js

Se trata de un pequeño fichero JavaScript que pretende aglutinar un conjunto de herramientas que pudieran ser necesarias a lo largo de las vistas del sitio. Por el momento, sólo contiene una función implementada: `showDismissibleAlert(title, message, alertType)`, que muestra un mensaje (`div` de la clase `alert alert-dismissible`) en un `div` reservado a tal efecto en todas las vistas. De esta forma, los mensajes de ayuda se muestran siempre en el mismo lugar. *Dismissible* hace referencia a que a la izquierda del mensaje aparece una pequeña x para cerrar el mensaje, desapareciendo éste de la pantalla. De todas formas, están programados para que a los 5 segundos éstos desaparezcan automáticamente.

5 Resultados de visualización

Además de servir como punto de gestión de una consulta psiquiátrica, esta herramienta también destaca por la visualización de los datos que maneja. De esta forma permite el análisis cualitativo y cuantitativo de los tratamientos que los pacientes siguen, así como extraer conclusiones de dichos datos.

La confidencialidad que el uso de los datos médicos exige ha provocado que, en este caso, hayamos optado por usar juegos de datos sintéticos para la prueba de la aplicación. No corresponden a pacientes reales, aunque ese aspecto no es relevante puesto sólo queremos comprobar la idoneidad de las representaciones gráficas esocogidas. Contamos con tres pacientes ficticios de los que consultaremos visualmente sus datos, realizando un análisis de los mismos.

Como se explicaba en la Subsección 3.4.4, la vista principal del servicio de visualización (Figura 5.1) se compone de dos mitades claramente diferenciadas: la izquierda muestra qué datasets están cargados en la aplicación, y permite cambiar entre los distintos tipos de gráficas que pueden mostrarse con dichos datos. La mitad derecha muestra los selectores de fecha (el intervalo de tiempo representado) así como el espacio que ocuparán las gráficas cuando estas se generen.

Una vez agregado un paciente, tenemos tres tipos de gráficas a visualizar. La primera, denominada "Distribución" que se compone de dos diagramas, uno de área apilada, que representa la medicación prescrita en ese intervalo de tiempo, y uno de líneas, que representa los resultados de los test realizados al paciente en dicho intervalo. La segunda, son histogramas de cada uno de los medicamentos prescritos. Finalmente, el tercer tipo de gráfica consiste en un mapa de calor sobre la medicación.

5.1. Distribución

El diagrama combina tanto los datos de la medicación como de los resultados de test en el intervalo de tiempo seleccionado. Por defecto, se muestran todos los datos disponibles en ese intervalo. En la parte derecha aparece la leyenda interactiva: haciendo click en cada uno de los colores, los datos asociados desaparecen y vuelven a aparecer en el gráfico (Figura 5.2). De esta forma se consigue un diagrama personalizado que se presta fácilmente a análisis.

Los medicamentos están clasificados en 6 categorías: litio, antiepilépticos, antipsicóticos, ansiolíticos o hipnóticos, antidepresivos y otros. Cada tipo de medicamento está agrupado en el diagrama de área y es representado mediante un color. Cuanto más ancha sea el área, más tomas de la misma tipología se producen. De esta forma obtenemos por un

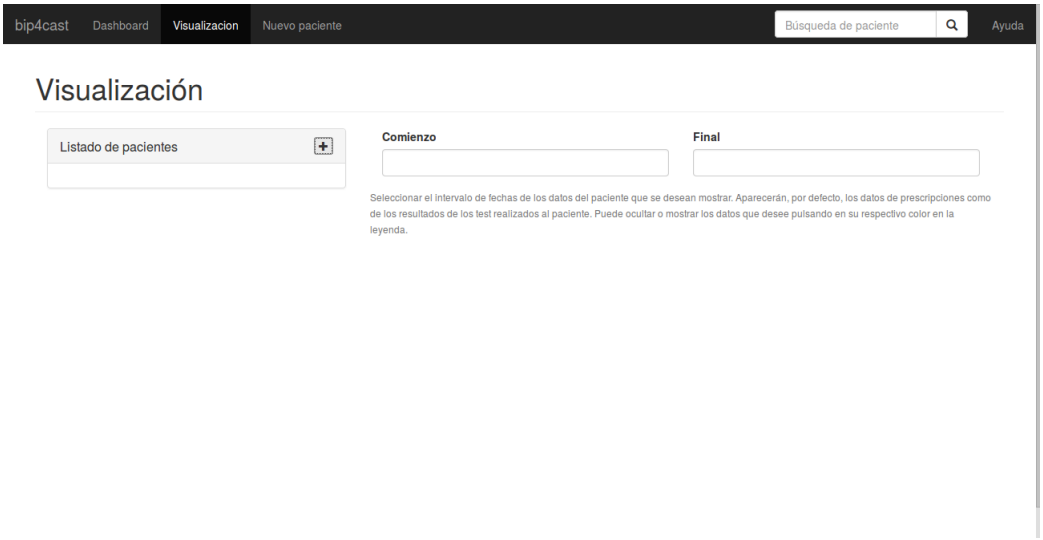

Figura 5.1: Vista principal de la visualización.

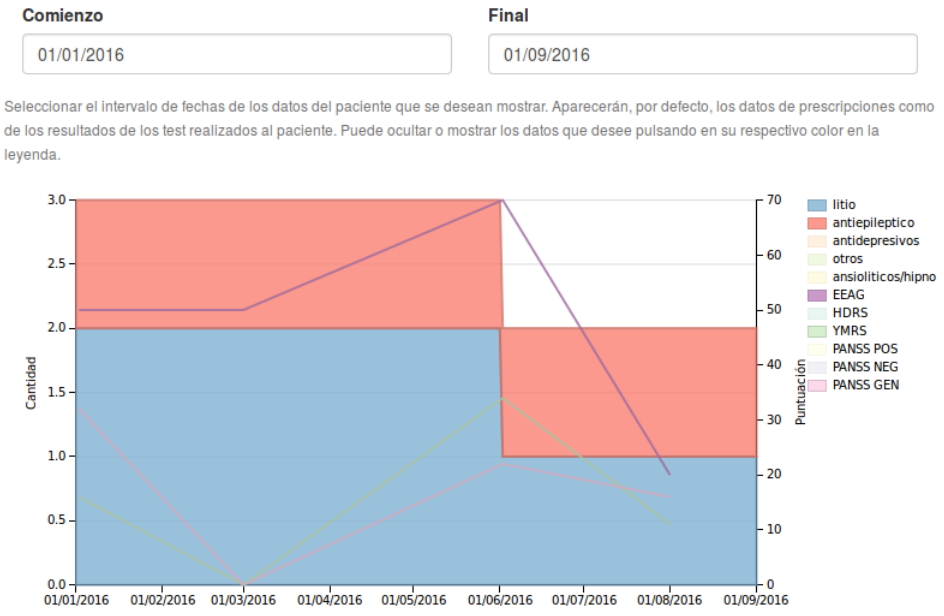

Figura 5.2: Subgráfica obtenida mediante interacción.

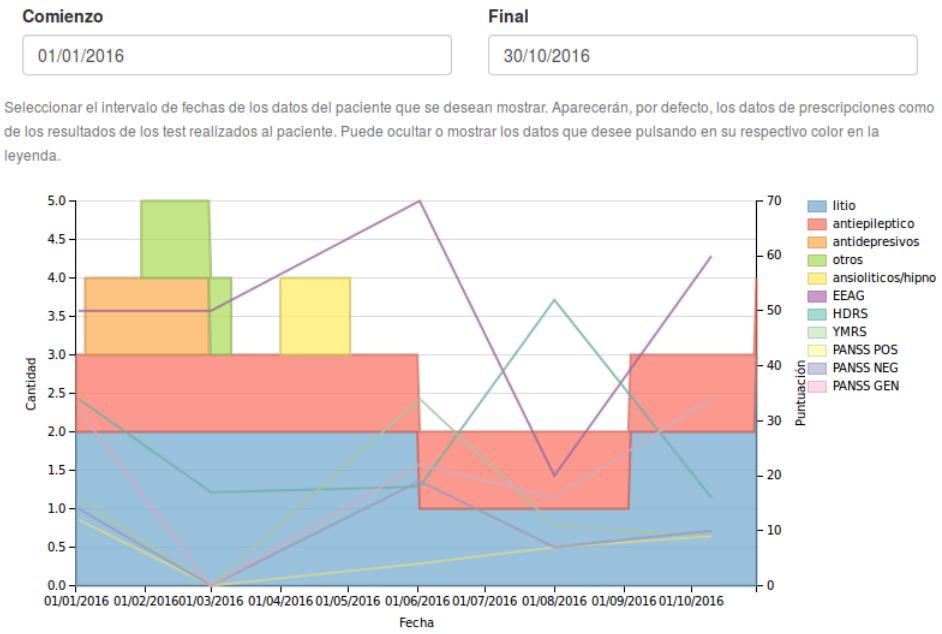

Figura 5.3: Datos del primer paciente.

lado cuánta cantidad de medicamentos y por otro de qué clases toma el paciente. Como ambos conjuntos de datos (medicamentos y test) están superpuestos, puede apreciarse qué impacto tienen los unos sobre los otros de una forma eficaz.

Analizamos la gráfica del primer paciente, que aparece en la Figura 5.3. Se puede observar que parte de un estado en el que se consumían hasta cuatro tipos distintos de medicamentos. La reducción de las tomas de litio y la eliminación de los ansiolítocos parece provocar una fuerte caída la EEAG y un incremento en la HDRS, que vuelve a recuperarse en cuanto se restaura la cantidad de litio que se tomaba al principio. Esto sugiere que este paciente es muy sensible al litio, por lo que puede existir correlación entre el litio y el estado depresivo del paciente.

5.2. Histogramas

Los histogramas miden la distribución de una muestra. En nuestro caso, hemos querido medir, para un tipo medicamento, la frecuencia de días en que se realizan el número de tomas. Así puede estudiarse si un paciente toma a lo largo de cierto intervalo de tiempo mucha o poca cantidad de algún tipo de medicamento, y cuántos días ocurre eso. Existe un histograma para cada tipo de medicamento, que puede mostrarse pinchando en la opción correspondiente del desplegable “Histogramas” de la izquierda.

En el caso del segundo paciente, seleccionamos por ejemplo el histograma correspondiente a los antiepilépticos (Figura 5.4). Destaca sobre el resto la columna derecha, es

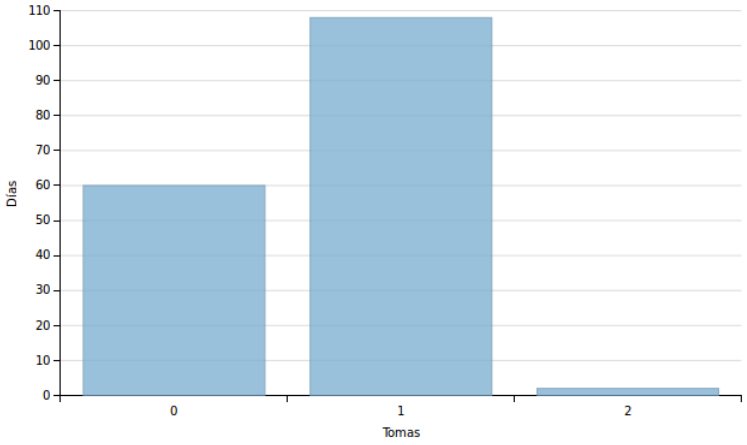

Figura 5.4: Histograma correspondiente a los ansiolíticos.

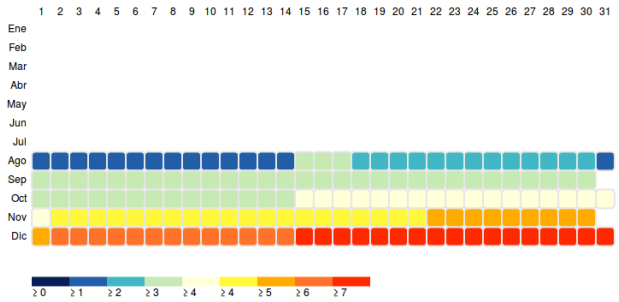

Figura 5.5: Mapa de calor de antidepresivos.

decir, durante casi 110 días hubo una toma de antiepilépticos, y durante 2, dos tomas. Sería interesante conocer en qué dos días se incrementaron las tomas de antiepilépticos y qué relación hay entre ellos para sacar conclusiones sobre el consumo de este medicamento. Para eso pueden utilizarse cualquiera de las otras dos gráficas de la aplicación, ya que contienen información precisa de las fechas de las tomas.

5.3. Mapa de calor

Los mapas de calor (*heatmaps*) consituyen una herramienta visual muy propicia para medir indicadores o cantidades absolutas. En nuestro caso hemos optado por representar el año natural, asignando a cada día del año una celda. De esta forma y como hicimos con los histogramas, seleccionado un tipo de medicamento, podemos ver la cantidad de dicho tipo que el paciente tomaba en el intervalo de tiempo seleccionado. Los colores más fríos se asocian a cantidades más bajas de medicamento, mientras que los más cálidos significan cantidades más altas.

Para la imagen de la figura, se ha optado por representar el mapa de calor de los antidepresivos del tercer paciente de nuestra muestra. Como se muestra en la Figura 5.5, el paciente consume gran cantidad de estos medicamentos en los dos últimos meses del año, sobre todo hacia finales del año. Por otro lado, durante el verano, pese a que las cantidades son más bajas que durante el invierno, durante los días centrales del mes de agosto presenta un pico en el consumo. Las fechas, significativas por ser periodos del año muy concretos podrían sugerir que el paciente sufre de cierta estacionalidad o bien tendencia a la depresión durante los días de vecaciones.

5.4. Generando datos

Como decíamos al principio de este capítulo, se ha trabajado con datasets sintéticos para probar las distintas gráficas implementadas en la aplicación. Cuando su uso alcance la masa crítica de usuarios se dispondrá de datos suficientes para realizar tareas de inferencia sobre los datos. Ya hemos comentado con anterioridad que no existía un patrón que todos los pacientes siguieran. Aún así, sería de gran utilidad conocer si, por ejemplo, la medicación de un paciente o los efectos que esta provoca sigue cierta distribución de probabilidad. Algo similar podríamos decir de los patrones de sueño, que se cree que tienen mucha influencia en la aparición de crisis.

El objetivo de las visualizaciones es que se puedan realizar rápidamente análisis de grandes cantidades de datos, extraer conclusiones y actuar en consecuencia. Por ello, es importante escoger qué visualizaciones deben implementarse para alcanzar estos objetivos de forma efectiva. Hasta que el momento de masa crítica de usuarios llegue, podemos utilizar una serie de datos simulados para comprobar que las visualizaciones son correctas. Para comenzar, podemos suponer normalidad en las variables aleatorias (una por cada tipo de medicación). Si llegado el momento de trabajar con datos reales esta hipótesis no fuese correcta, se podrían realizar los ajustes convenientes para adaptar la visualización a la distribución inferida.

Para ello se han implementado sendas funciones que generan datos de medicación en un intervalo de tiempo dado siguiendo cierta distribución de probabilidad. El esquema general de este algoritmo de generación es:

1. Decidir aleatoriamente el número de prescripciones que van a realizarse en ese intervalo.
2. Escoger aleatoriamente la longitud de esas prescripciones (en días).
3. Asignar aleatoriamente qué tipo de medicación se tomará en cada prescripción generada.

Todos los datos generados aleatoriamente en el algoritmo anterior siguen la misma distribución.

El fichero `randomData.js` implementa las funciones JavaScript necesarias para generar datos siguiendo una distribución normal $N(\mu, \sigma)$ (siendo estos dos argumentos parametrizables por el usuario). Se implementa también una función para la chi cuadrado χ^2_n

con n grados de libertad, aunque esta con objetivos de test de hipótesis y no con finalidad de representación. Esto último queda fuera del ámbito del presente trabajo.

Para la implementación de la correspondiente a la distribución normal, se ha seguido el método polar de Marsaglia:

```
function gaussian(mean, stdev) {
    var y2;
    var use_last = false;
    return function() {
        var y1;
        if(use_last) {
           y1 = y2;
           use_last = false;
        }
        else {
            var x1, x2, w;
            do {
                 x1 = 2.0 * Math.random() - 1.0;
                 x2 = 2.0 * Math.random() - 1.0;
                 w  = x1 * x1 + x2 * x2;
            } while( w >= 1.0);
            w = Math.sqrt((-2.0 * Math.log(w))/w);
            y1 = x1 * w;
            y2 = x2 * w;
            use_last = true;
       }

       var retval = mean + stdev * y1;
       if(retval > 0)
           return retval;
       return -retval;
   }
}
```

Código 5.1: Implementación del método de Marsaglia en JavaScript.

Aprovechando esta implementación, y basándonos en la definición de la variable χ^2_n ($\chi^2_n = Z^2 + \overset{n}{\cdots} + Z^2$) donde $Z \sim N(0,1)$, hemos obtenido una función para esta distribución:

```
function chiSquare(freedDeg){
    return function() {
      if (freedDeg === undefined) {
        freedDeg = 1;
      }
      var i, z, sum = 0.0;
      for (i = 0; i < freedDeg; i++) {
        z = gaussian(0,1)();
        sum += z * z;
      }
      return sum;
    }
}
```

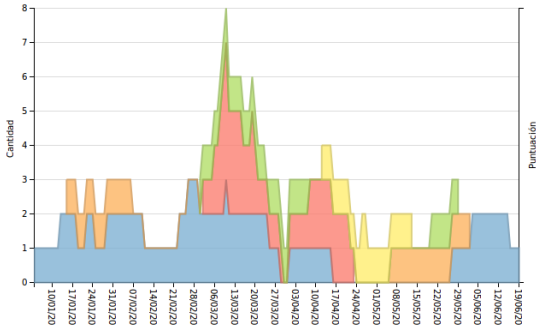

Figura 5.6: Gráfica *Distribución* usando datos generados aleatoriamente.

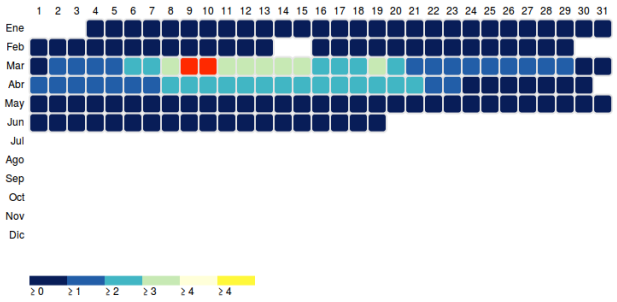

Figura 5.7: Mapa de calor para ansiolíticos usando datos generados aleatoriamente.

La fundamentación del método de Marsaglia se explica en detalle en la Sección 5.5.

Para cada una de estas funciones existe otra que devuelve los valores de la distribución pero convirtiéndolos a enteros (para su uso en las funciones de generación de datasets) dentro de un intervalo dado.

Vamos a realizar pruebas de visualizaciones implementadas generando un dataset. Actuando bajo la hipótesis de normalidad antes mencionada, el dataset sigue una distribución $N(15, 5)$. Esta hipótesis no resulta descabellada debido a que los pacientes siguen tratamientos farmacológicos estrictos para tratar de evitar las crisis, y las tomas pueden llegar a incrementarse considerablemente cuando éstas se materializan.

En la Figura 5.6 puede verse el diagrama de distribución para el dataset. Se observa con claridad que la distribución que siguen los medicamentos es una normal por el aspecto que presenta la gráfica. El valor más frecuente es el de color rojo, que se corresponde con los ansiolíticos y los hipnóticos. Si representamos el mismo dataset en forma de mapa de calor, obtenemos el que se muestra en la Figura 5.7. Rápidamente se aprecia, esta vez más en detalle que en el gráfico anterior, el pico de tomas los días 9 y 10 de marzo. Los colores azules nos dicen que la cantidad de litio consumida fuera de los días centrales del mes de marzo es baja. Como desventaja, es más complicado identificar la distribución normal de los datos debido a que se representa la agrupación de datos con intensidades de color, en lugar de con barras o líneas.

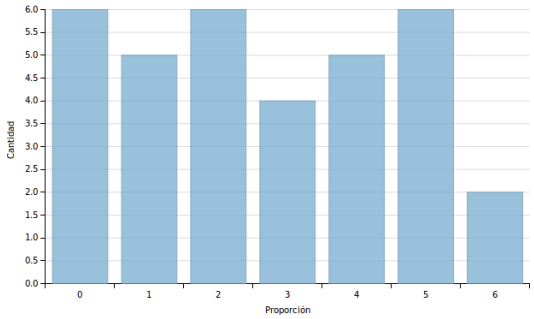

Figura 5.8: Histograma para ansiolíticos generado aleatoriamente.

Por otro lado, representando el histograma para el mismo tipo de medicamentos se obtiene el de la Figura 5.8. De aquí podemos deducir que los ansiolíticos e hipnóticos están presentes siempre durante el periodo de tiempo representado, llegando incluso hasta las seis tomas diarias (aunque sea únicamente en dos días). Por otra parte, se aprecia que se recetaron este tipo de medicamentos durante 34 días durante el intervalo bajo estudio.

En definitiva, el gráfico "Distribución" y los mapas de calor permiten visualizar muy bien la normalidad de los datos, al contrario que el histograma. En efecto, la reducción de dimensionalidad realizada para representar el histograma (perdemos las fechas de las tomas), aunque permite apreciar las cantidades de forma adecuada, no recoge su distribución en el tiempo. Por ello, sucesivas versiones de la aplicación se podría mantener éste último puesto que podría tener utilidad médica pese a que no la tiene estadística, al menos a priori.

5.5. Generación de números aleatorios

Comprobar la validez de las representaciones estudiadas nos ha llevado a crear un generador de datos para logarlo. En esta sección pretendemos profundizar en los fundamentos matemáticos que sustentan las funciones implementadas en el marco de este proyecto para la creación de esos datasets sintéticos. La generación de números aleatorios siguiendo cierta distribución de probabilidad constituye un pilar básico en los experimentos de Monte Carlo. Pese a que ese no es el tema de estudio, nos apoyaremos en los resultados de esta rama de la estadística para el propósito que nos ocupa.

Los lenguajes de programación de propósito general incluyen, al menos, una función para generar números aleatorios comprendidos en cierto intervalo $[a, b]$. La distribución que siguen esos números es una uniforme en dicho intervalo (denotada por $U(a, b)$). Aprovecharemos este hecho para implementar una variable aleatoria normal $N(\mu, \sigma^2)$.

5.5.1. Método de Box-Muller

La forma más directa y a la vez, más ingenua, de generar una variable aleatoria conocida su función de distribución es precisamente hallar la inversa de ésta última. Queda plasmado en este resultado [Fishman, 2013]

Teorema 5.1. Sea X una variable aleatoria con función de distribución $F(z)$ y sea F^{-1} la función inversa de esta. Entonces la variable aleatoria $U = F(X)$ tiene distribución uniforme en el intervalo $(0, 1)$. Si Y es una variable aleatoria que sigue dicha distribución, entonces la variable aleatoria $X = F^{-1}(U)$ satisface la distribución F.

El principal problema que plantea esta aproximación es precisamente el cálculo de la función inversa de una función de distribución F. En efecto, hallar una expresión analítica para esa función puede ser sumamente complicado computacionalmente. No es, ni mucho menos, el único método para el cálculo de variables aleatorias.

El método de Box-Muller es un método de aceptación-rechazo para simular variables aleatorias que sigan una distribución normal [Fishman, 2013, p. 190]. Es importante mantener en mente las siguientes dos propiedades de la distribución normal:

1. Si $Y \sim N(0, 1)$, entonces $Z = \mu + \sigma Y$ es una $N(\mu, \sigma^2)$.
2. Reproductividad de la normal: Si $Z_1, \ldots, Z_n$ son variables aleatorias con distribución $N(\mu_1, \sigma_1^2), \ldots, N(\mu_n, \sigma_n^2)$ respectivamente, entonces la variable aleatoria $Z_1 + \cdots + Z_n$ sigue la distribución $N(\mu_1 + \cdots + \mu_n, \sigma_1^2 + \cdots + \sigma_n^2)$. Esta propiedad nos permitirá generar cualquier distribución $N(\mu, \sigma^2)$ a partir de una $N(0, 1)$.

El método polar de Marsaglia se basa en el siguiente resultado:

Teorema 5.2 (Transformación de Box-Muller)**.** Si U_1 y U_2 son variables aleatorias independientes idénticamente distribuidas $U(0, 1)$, las variables

$$X = \sqrt{-2\log(U_1)}\cos(2\pi U_2) \quad \text{e} \quad Y = \sqrt{-2\log(U_1)}\,\text{sen}(2\pi U_2)$$

son variables independientes con distribución normal $N(0, 1)$.

5.5.2. Método de Marsaglia

Aplicar el resultado anterior requiere calcular senos y cosenos, una tarea que computacionalmente puede resultar costosa. Pese a esto, existe un método de aceptación-rechazo basado en la tranformación de Box-Muller que evita el cálculo de funciones trigonométricas.

Sean U_1, U_2 variables independientes idénticamente distribuidas $U(0, 1)$. Definimos las variables aleatorias $V_1 = 2U_1 - 1$ y $V_2 = 2U_2 - 1$. Ambas siguen la distribución $U(0, 2)$. De esta forma, el par (V_1, V_2) representa las coordenadas cartesianas de un punto aleatorio distribuido uniformemente sobre el cuadrado $[-1, 1] \times [-1, 1]$. Sean (R, θ) las coordenas polares de dichos puntos, es decir,

$$R^2 = V_1^2 + V_2^2 \quad \text{e} \quad \tan(\theta) = \frac{V_1}{V_2}. \tag{5.1}$$

Proposición 5.1. *Si $R^2 = V_1^2 + V_2^2 \leq 1$, es decir, el punto de coordenadas (V_1, V_2) está dentro de la circunferencia unidad, R^2 y θ son, respectivamente, variables independientes con distribución $U(0,1)$ y $U(0,2\pi)$.*

Retomando el resultado del teorema 5.2, podemos usar las coordenadas polares y escribir que

$$X = \sqrt{-2\log(R^2)}\cos(\theta) \quad \text{e} \quad Y = \sqrt{-2\log(R^2)}\,\text{sen}(\theta).$$

Utilizando la Ecuación 5.1 y sustituyendo, se obtiene finalmente que

$$X = V_1\sqrt{\frac{-2\ln(R^2)}{R^2}} \quad \text{e} \quad Y = V_2\sqrt{\frac{-2\ln(R^2)}{R^2}}.$$

Así, X e Y son variables aleatorias con distribución $N(0,1)$, y las hemos calculado sin necesidad de utilizar funciones trigonométricas.

De esta forma, puede crearse un algoritmo [Gentle, 1998] que simula una variable aleatoria $N(0,1)$ de la siguiente forma

1. Se generan dos números aleatorios a_1, a_2.
2. Se calculan $b_1 := 2a_1 - 1$, $b_2 := 2a_2 - 1$.
3. Tomar $A = b_1^2 + b_2^2$.
4. Si $A > 1$, volver al paso 1. En caso contrario, devolver los valores

$$x = b_1\sqrt{\frac{-2\ln(A)}{A}} \quad \text{e} \quad y = b_2\sqrt{\frac{-2\ln(A)}{A}}.$$

Este algoritmo devuelve un par de variables aleatorias, basta quedarse con una de ellas. La implementación en JavaScript es la que se muestra en el Código 5.1.

6 Conclusiones y trabajo futuro

El análsis de datos masivos está llamado a ser el futuro, o al menos eso parece a día de hoy. Tiene la capacidad de ser implementado en numerosos campos, y no sólo en el empresarial o investigador, como venía pasando hasta ahora. La potencia de los grandes centros de datos, las nuevas arquitecturas que sacan provecho máximo a la capacidad de cálculo, la versatilidad de los sistemas cloud y muy especialmente, las ingentes cantidades de datos que generamos a diario alumbran un futuro realmente prometedor para el big data.

Muchos sectores no son ajenos a este cambio y comienzan a prepararse para la revolución de los datos que está por venir. La medicina, como cualquier otro campo, contempla posibilidades de cambio muy profundas con la ayuda de la informática y el análisis de datos. La mejora en los diagnósticos o los tratamientos personalizados para cada paciente son sólo dos ejemplos de cómo la informática puede introducir grandes mejoras. La plataforma *bip4cast* trata de formar parte de esa revolución planteando un sistema integral de seguimiento y prevención de crisis en los pacientes con trastorno bipolar.

Como hemos mostrado a lo largo de estas páginas, se ha desarrollado la aplicación web *bip4cast* cumpliendo los requisitos propuestos por el Dr. Urgelés y los objetivos que para ella se establecieron. La aplicación desarrollada es plenamente funcional, está lista para ser desplegada en el servidor y ser utilizada. Puesto que la plataforma está en una fase muy temprana, el primer cometido de la aplicación web será la captación de datos de los pacientes derivado del uso diario de los profesionales médicos para la gestión de sus consultas.

En el apartado de las visualizaciones del Capítulo 5, hemos realizado pruebas para comprobar si ciertos tipos de representaciones se ajustan bien a los datos representados y de verdad permiten sacar conclusiones sobre la distribución de éstos. Las elecciones de los gráficos de área y los mapas de calor parecen acertadas, mientras que el histograma no lo es tanto, puesto en que su uso perdemos variables que afectan a la distribución, como la fecha de las tomas.

Respecto al trabajo futuro, destacamos varios aspectos. El primero tiene que ver con la integración. Pese a que la base de datos y el servicio están integrados, el resto de elementos de la plataforma no. A corto plazo, sería necesaria la integración con la aplicación Android para hacer funcional el apartado de comunicaciones con el cliente. En el medio plazo, es necesaria la creación de un framework que recolecte los datos tanto del actígrafo como del smartphone del paciente y los integre en la base de datos de forma automatizada. En ese sentido, los actígrafos pueden presentar problemas pues la obtención de datos ha de hacerse a través de una herramienta propia del fabricante de los

mismos. Así mismo, sería necesario realizar la integración con el modelo una vez que éste se implemente. Dicha integración no será complicada en absoluto, puesto que la modularidad de Node.js permite que existan paquetes encargados de conectar con servicios de R, como por ejemplo `rstat` o OpenCPU, que proporciona una API para análisis de datos usando R. Otra alternativa sería utilizar RServe, un protocolo de comunicación con una sesión remota de R. Existe una implementación de un cliente RServe para JavaScript disponible en GitHub.

La arquitectura usada, especialmente el *runtime* Node.js, permite que esta sea un sistema robusto y fácilmente escalable. Aún así, por el momento es difícil prever cómo se comportará con exactitud el sistema cuando la cantidad de datos almacenados y las conexiones que tenga que soportar crezcan. Para ello hay que esperar a que la plataforma esté en primer lugar plenamente integrada y después sería conveniente la realización de pruebas de carga para determinar el rendimiento del sistema.

La visualización de los datos tiene mucho potencial, especialmente en el campo médico donde la toma de decisiones puede suponer influir en la calidad de vida de las personas. La aplicación web está preparada para representar nuevos tipos de datos. En efecto, la API puede incrementarse para servir nuevos tipos de datos de forma sencilla, las librerías de representación gráfica están cargadas en el cliente y las funciones de representación pueden reutilizarse. Por el momento hemos simulado la representación de datos que siguen una distribución normal con mapas de calor, histogramas y gráficos de área. Una línea de trabajo futuro en este sentido vendría dada por la prueba de nuevas visualizaciones y estudiar su comportamiento con datos bajo otras distribuciones distintas de la normal.

Pese a que los requisitos están cubiertos, sería recomendable realizar pruebas de usabilidad de cara a versiones y mejoras posteriores, teniendo en cuenta la experiencia de los usuarios así como sus sugerencias, puesto que puede ser también un foco de innovación y de incorporación de nuevas características a la aplicación en diversos aspectos: visualización, medicación, nuevos test que los médicos realizasen a sus pacientes, etc.

Índice alfabético

Glosario

AJAX *Asynchronous JavaScript And XML*, técnica de desarrollo web para crear aplicaciones web asíncronas. 45

Apache Kafka Sistema de mensajería distribuido de alto rendimiento cuyo objetivo principal es proporcionar un sistema unificado de baja latencia para la gestión de flujos de datos en tiempo real.. 16

API Application Programming Interface. 11, 12, 21, 40

AWS Amazon Web Services. 15

basado en eventos Paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los eventos que provoca bien el propio sistema o bien los usuarios como resultado de su interacción con él. 21

BBVA Banco Bilbao Vizcaya Argentaria. 11

binning técnica de representación que consiste en juntar varias partes pequeñas no para formar una más grande, sino para compactificar la representación. 11

Bootstrap Framework de diseño de sitios web basado en HTML y CSS, que incluye plantillas para formularios, botones, menús, extensiones JavaScript y que presta especial atención al diseño adaptable. 20, 24

business intelligence Conjunto de estrategias y herramientas para la transformación de datos en información significativa y útil para el análisis del negocio. 9, 11

CIE *Clasificación Internacional de Enfermedades*, criterio de codificación de las enfermedades publicado por la Organización Mundial de la Salud (OMS).. 4, 42

contenedor web Componente de un servidor web que interactúa con servlets de Java. 21

CRAN Task View Agrupación de paquetes y funciones de R. Cada task view corresponde a una disciplina diferente. 12

cross-platform capacidad para el uso de un programa o dispositivo sin inconvenientes en distintas plataformas de hardware y sistemas operativos. 19

CSS Cascading Style Sheets. 10

dataset conjunto de datos. 11, 13, 24, 45, 49

deadlock Bloqueo permanente de un conjunto de procesos en un sistema concurrente. 21

diseño web adaptable filosofía de diseño y desarrollo web cuyo objetivo es adaptar la aparienca de las páginas web al dispositivo que se esté utilizando para visualizalas. 20, 24

DOM *Document Object Model*, interfaz para representar documentos HTML, XMHTL y XML. A través de él, los programas pueden acceder al documento y modificar su estructura y contenido, incluso dinámicamente. 23, 45

EEAG Escala de Evaluación de la Actividad Global. 6, 43, 51

escalado horizontal En escalabilidad de sistemas informáticos, tipo de escalabilidad que incrementa el rendimiento global del sistema al añadir más unidades de procesamiento (nodos). 14, 21

express.js Framework para node.js para la construcción de aplicaciones web y APIs, y en especial para atender las peticiones web de estas. 20

fenotípico Perteneciente o relativo al fenotipo: manifestación variable del genotipo de un organismo en un determinado ambiente, comprendiendo tanto rasgos visibles como conductuales. 2, 4, 25

framework Estructura conceptual y tecnológica que sirven de base para la organización y desarrollo de software. 10, 12, 21

grammar of graphics Abstracción que hace que pensar, razonar y comunicar gráficos sea más sencillo desarrollado por Leland Wilkinson. 11

Hadoop Framework de soporte para aplicaciones distribuidas de licencia libre. 12, 13

HDFS *Hadoop Distributed File System*, sistema de ficheros usado por Hadoop. 17

HDRS Hamilton Depression Rating Scale. 6, 51

heteroadministrado Dicho de un test, cuya administración la realiza el observador al observado. 1

Infrastructure as a Service Modelo de distribución de infraestructuras físicas como un servicio. En lugar de adquirir servidores o espacio en centros de datos, los clientes compran estos servicios a proveedores externos. 15

JavaScript Lenguaje de programación interpretado ampliamente utilizado en aplicaciones web. 10, 20

JSON *JavaScript Object Notation*, formato de texto ligero para el intercambio de datos, subconjunto de la notación de objetos de JavaScript.. 22, 40

lado del cliente Conjunto de rutinas y procedimientos que ejecuta un navegador cliente (que solicita información a un servidor). 20

lado del servidor Conjunto de rutinas y procedimientos que ejecuta un servidor para satisfacer las peticiones que realizan los clientes. 20

NASA National Aeronautics and Space Administration. 11

node.js Entorno de ejecución para la capa de servidor que emplea JavaScript, asíncrono, basado en el motor V8 de Google. 20, 64

NoSQL Not Only SQL. 15, 22

ODM Object Data Mapping. 40

OMS Organización Mundial de la Salud. 63

open source Modelo de distribución de software, que hace que este sea distribuido y desarrollado libremente. 15

O'Reilly Empresa editorial estadounidense principalmente enfocada a los libros de tecnología e informática. 16

PANSS Positive and Negative Syndrome Scale. 6, 34, 44

PDF Portable Document Format. 12

PET Positron Emission Tomography. 12

R Lenguaje y entorno de programación para análisis estadístico y gráfico. 3, 8, 9, 11, 12, 60

REST *Representational State Transfer*, estilo de arquitectura software. 40

SAS Lenguaje de programación para el análisis estadístico de datos. 9

servidor de aplicaciones Aplicación software que proporciona servicios de apliScación a un servidor Web. 21

Software as a Service Modelo de uso de software basado en la utilización del software a través de navegadores web. 11, 19

SPSS Programa estadístico informático propiedad de IBM. 9

Summingbird API desarrollada por Twitter usada para desarrollar procesos de tipo batch MapReduce que pueden ser ejecutados en arquitecturas batch, streaming e híbridas.. 16

SVG Scalable Vector Graphics. 10, 12

V8 Motor para interpretar código JavaScript de código abierto desarrollado por Google e incluido en su navegador Chrome. 65

web templating system Utilidad que, tomando una plantilla de una página o sitio web y datos suministrados por el servidor, conforma documentos HTML. 22

webstack Conjunto de aplicaciones software utilizado para el desarrollo web. 20

YMRS Young Mania Rating Scale. 6

Bibliografía

Michael Abernethy. ¿Simplemente qué es Node.js?, June 2011. URL https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/.

Actigraph. ActiGraph Link | ActiGraph, 2016. URL http://actigraphcorp.com/products-showcase/activity-monitors/actigraph-link/.

Ethan Brown. *Web Development with Node and Express: Leveraging the JavaScript Stack*. O'Reilly Media, Beijing ; Sebastopol, CA, edición: 1 edition, July 2014. ISBN 978-1-4919-4930-6.

Joe Cheng, J. J. Allaire, Yihui Xie, Jonathan McPherson, RStudio, jQuery Foundation and contributors, Twitter Inc., Jacob Thornton and Bootstrap contributors, Alexander Farkas, Scott Jehl (Respond.js), Stefan Petre, Andrew Rowls, Dave Gandy, Brian Reavis (selectize.js), Kristopher Michael Kowal and es5-shim contributors, Denis Ineshin, Sami Samhuri, SpryMedia Limited, John Fraser, John Gruber, Ivan Sagalaev, and R. Core Team. shiny: Web Application Framework for R, August 2015. URL https://cran.r-project.org/web/packages/shiny/index.html.

G. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2013. ISBN 9781475725537. URL https://books.google.es/books?id=dogHCAAAQBAJ.

Ana C. García-Blanco, Pilar Sierra, and Lorenzo Livianos. Nosología, epidemiología y etiopatogenia del trastorno bipolar: Últimas aproximaciones. *Psiquiatría Biológica*, 21(3):89–94, September 2014. ISSN 11345934. doi: 10.1016/j.psiq.2014.07.004. URL http://linkinghub.elsevier.com/retrieve/pii/S1134593414000670.

J.E. Gentle. *Random Number Generation and Monte Carlo Methods*. Statistics and computing. Springer, 1998. ISBN 9780387985220. URL https://books.google.es/books?id=wybvAAAAMAAJ.

Hartvigsen, Gregg. *Primer in Biological Data Analysis and Visualization Using R*. Columbia University Press Group Ltd, March 2014. ISBN 978-0-231-16699-7.

Jay Kreps. Questioning the Lambda Architecture - O'Reilly Radar, July 2014. URL http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html.

Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications, May 2015. ISBN 978-1-61729-034-3.

npm documentation. What is npm? | npm Documentation, 2016. URL https://docs.npmjs.com/getting-started/what-is-npm.

Carla Torrent. Escala de Young para la Evaluación de la Manía, October 2011. URL http://www.acmcb.es/files/425-2431-DOCUMENT/Torrent-42-6Oct11.pdf.

Udacity. Data Visualization and D3.js Course | Udacity, 2014. URL www.udacity.com/course/data-visualization-and-d3js--ud507.

Wikipedia contributors. Global Assessment of Functioning, July 2016a. URL https://en.wikipedia.org/w/index.php?title=Global_Assessment_of_Functioning&oldid=731955585. Page Version ID: 731955585.

Wikipedia contributors. Hamilton Rating Scale for Depression, May 2016b. URL https://en.wikipedia.org/w/index.php?title=Hamilton_Rating_Scale_for_Depression&oldid=722263561. Page Version ID: 722263561.

Wikipedia contributors. Positive and Negative Syndrome Scale, June 2016c. URL https://en.wikipedia.org/w/index.php?title=Positive_and_Negative_Syndrome_Scale&oldid=724022764. Page Version ID: 724022764.

Hage Yaapa. *Express Web Application Development*. Packt Publishing Ltd, January 2013. ISBN 978-1-84969-655-5. Google-Books-ID: nQ_I1yuV_DwC.

Nick Qi Zhu. *Data Visualization with D3.Js Cookbook*. Packt Publishing, 2013. ISBN 978-1-78216-216-2.