Visualización gráfica de tipos de datos Haskell

María del Rosario Baena Priego
Roberto Aragón Pividal

GRADO EN INGENIERÍA DE COMPUTADORES
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO

Madrid, 23 de junio de 2014

Director: Manuel Montenegro Montes

Autorización de difusión y utilización

María del Rosario Baena Priego

Madrid, 23 de junio de 2014

Roberto Aragón Pividal

Madrid, 23 de junio de 2014

Agradecimientos

Agradecemos a Manuel Montenegro su inestimable ayuda y disponibilidad. Gracias también a la comunidad de Software Libre de JavaScript, que nos ha abierto un mundo nuevo de posibilidades.

Índice

Índice de figuras

Resumen

Este trabajo se centra en el estudio e implementación de una herramienta para la ayuda al aprendizaje del lenguaje Haskell en el entorno educativo. Si bien existen librerías y aplicaciones web que proporcionan una funcionalidad similar, éstas requieren la instalación de una plataforma Haskell completa o restringen demasiado el entorno de aprendizaje.

Nuestra herramienta consiste en una aplicación web que permite editar módulos y evaluar cualquier expresión Haskell de forma interactiva sin necesidad de que el usuario se instale la plataforma completa. Además, los tipos y estructuras de datos se pueden representar como objetos gráficos interactivos que el programador puede personalizar y mejorar, lo que ayuda tanto a la comprensión de la naturaleza de los tipos como al aprendizaje del lenguaje.

La propuesta de *software* y *hardware* de la solución probada permite disponibilidad y escalabilidad, ya que es posible su distribución en diferentes nodos gracias a su arquitectura basada en servicios REST y componentes desacoplados y balanceados. El desarrollo de la herramienta se fundamenta en un arquitectura moderna y de fácil mantenimiento con grandes capacidades de accesibilidad, extensibilidad y adaptación a distintos dispositivos.

Creemos que el resultado ha cumplido las expectativas y que la herramienta, aunque admite múltiples mejoras, proporciona un marco arquitectónico con gran potencial educativo, facilidad de uso y de evolución, lo que aumenta su utilidad.

Palabras clave: Haskell, estructura de datos, tipo de datos, representación gráfica, interactividad, aplicación web, programación funcional.

Abstract

This work is focused on the study and implementation of an aid tool for the learning of the Haskell language in the academic environment. While there are libraries and web applications that provide similar functionality, they require the installation of a complete Haskell platform or restrict too much the learning environment.

Our tool is a web application that allows editing modules and evaluate any Haskell expression interactively without the need to install the whole platform by the users. In addition, the types and data structures can be represented as interactive graphic objects that the programmer can customize and improve, which helps both to understand the nature of the types and to better learn the language.

The proposed and installed software and hardware solution allows availability and scalability, since it is feasible its distribution in different nodes through its REST-based service architecture with decoupled and balanced components. The development of the tool is based on an architecture modern and easy to maintain with good features like accessibility, extensibility and adaptation to heterogeneous devices.

We believe that the results have met the expectations and that the tool, even admitting many improvements, provides an architectonic framework with great educational potential, ease of use and development, which enhances its utility.

Keywords: Haskell, data structure, data type, graphical representation, interactivity, web application, functional programming.

Capítulo 1

Preliminares

1.1. Introducción / Introduction

Los primeros pasos en el aprendizaje de un nuevo lenguaje de programación son los más difíciles. Se requiere un gran tesón y mucha ayuda hasta que el esfuerzo se ve recompensado. De esa frustración surge nuestra motivación para apoyar la idea de una herramienta suficientemente amigable y, al mismo tiempo, accesible, que ayude al aprendiz de Haskell a *ver* como los tipos de datos y estructuras más complejas se vuelven más asequibles.

Nuestro propósito es familiarizar al aprendiz de Haskell con el lenguaje de programación, así que pensamos que una aplicación web más, como no, es la interfaz más familiar que podemos encontrar en la era de Internet.

Existen múltiples herramientas web en línea para la edición de programas y la interpretación de expresiones Haskell; también pueden instalarse y usarse múltiples librerías para la visualización de tipos y estructuras de datos, pero nuestra aportación plantea aunar ambas características y dotar a las representaciones gráficas de mayor iteractividad.

Un conocimiento analítico más profundo de las necesidades de la solución y un cuidadoso estudio de las alternativas tecnológicas nos llevaron hacia una arquitectura novedosa y, sin embargo, altamente flexible y manejable. El desarrollo de la aplicación valida esta sensación con cada nueva funcionalidad.

Creemos que el resultado obtenido con este método cumple los re-

quisitos iniciales y, además, abre un gran espectro de posibilidades para mejorar y avanzar hacia una una herramienta de aprendizaje mucho más completa que, sin embargo, puede mantener la usabilidad y sencillez gracias a su planteamiento y arquitectura.

En este primer capítulo realizamos una breve introducción a la programación funcional y a las herramientas disponibles para la visualización de estructuras de datos complejas, intérpretes de expresiones e IDE (*Integrated Development Environment*, Entorno de desarrollo integrado) para el desarrollo de proyectos en línea. Además, explicamos la motivación de este trabajo así como los propósitos iniciales, objetivos finales y el plan de trabajo realizado para el desarrollo de la herramienta resultante.

El capítulo siguiente, a modo de exposición de "materiales y métodos", plantea nuestra propuesta de arquitectura, HW (*hardware*, componentes electrónicos del ordenador) y SW (*software*, programas), explicando detalladamente cómo y porqué realizamos esta elección y bajo qué premisas se realiza la selección de componentes externos de la arquitectura.

En el capítulo 3 se describen con detalle, a modo de manual de usuario, las distintas páginas de las que consta nuestra aplicación web, para qué sirven y cómo utilizarlas correctamente, aunque su uso es intuitivo por diseño y no requiere una capacitación específica. El usuario y programador Haskell puede hacer un uso personalizado de las capacidades de representación gráfica de tipos, para lo cual se facilitan dos métodos diferentes de programación de tipos representables, necesarios para la visualización e interacción en la consola.

En el cuarto capítulo se describen los detalles de diseño y desarrollo de la solución y cómo debe instalarse para un correcto funcionamiento. De este modo se profundiza en cómo se realiza y cómo funciona la aplicación y se presentan los aspectos técnicos sobre las dependencias y requisitos de SW para su instalación, que son imprescindibles para un mantenimiento satisfactorio.

Uno de los propósitos iniciales del presente trabajo es el de proporcionar capacidades para personalizar y añadir nuevas representaciones gráficas de tipos y estructuras Haskell a programadores con los conocimientos necesarios. En el capítulo 5 abordamos algunos mecanismos y

métodos sencillos para extender la funcionalidad en éstas y en otras capacidades más allá de lo planteado inicialmente, y estudiamos qué esfuerzo y capacidades son necesarios para realizarlos.

Y, finalmente, el último capítulo está dedicado al estudio y discusión crítica de los resultados, a la exposición de conclusiones y, por supuesto, a un esbozo de las futuras mejoras y ampliaciones que serían necesarias en la solución propuesta. Por último explicamos la contribución detallada de cada autor al presente trabajo.

Introduction

The first steps in learning a new programming language are the hardest. Great determination and much help is needed until the effort is rewarded. From that frustration comes our motivation to support the idea of a sufficiently friendly and accessible tool to help to the Haskell apprentice to *see* how data types and more complex structures become more affordable.

Our purpose is to familiarize the apprentice with the Haskell language programming, so we thought it is a web application, again, the more familiar interface that we can find in the Internet age.

There are many online web tools for editing programs and interpretation of Haskell expressions. One can also install and use several libraries for displaying data types and structures, but our contribution is to join both features and to equip graphical representations with more interactivity.

A deeper analytical understanding of the requirements and a careful study of the technological alternatives lead us to a highly flexible, innovative and yet manageable architecture. The application development validates this feeling with each new feature.

We believe that the result we obtained with this method meets the initial requirements and also opens a wide spectrum of possibilities to improve and move towards a much more complete learning tool that, however, can maintain the usability and simplicity with its current approach and architecture.

In this chapter we provide a brief introduction to functional program-

ming and to some tools available for visualizing structures of complex data, expression interpretation and online IDEs for developing small projects. Furthermore, we explain the motivation of this work and the initial purpose, end goal and work plan for the development of the resulting tool.

The following chapter, as a sort of "materials and methods" explanatory, raises our proposed HW and SW architecture, giving details of how and why we made this choice and under what assumptions we make the selection of external components of the architecture.

Chapter 3 describes in detail, as a user's manual, the various pages of our web application consists on, what they are and how to use them properly, although its use is intuitive by design and should not require specific training. The user and Haskell programmer can customize graphing capabilities of types, for which two different programming methods are provided to obtain representation types necessary for the visualization and interaction in the console.

The details of the design and development of the solution are described in the fourth chapter and how it must be installed for proper operation. Thereby elaborates on how it performs and how the application works and detailed aspects of dependencies and software requirements are presented, which are essential for successful maintenance.

One of the initial purposes of the present work is to provide capabilities to customize and to add new graphical representations of types and data structures to Haskell programmers. In Chapter 5 we address some simple mechanisms and methods to extend the functionality in these and other areas far beyond which it is stated initially and to study the effort and skills that are needed to perform them.

And finally, the last chapter is devoted to the study and critical discussion of the results, to expose the conclusions and, of course, to outline future work and extensions that are necessary in the proposed solution. Finally we explain the detailed contribution of each author to this work.

1.2. Antecedentes

La programación funcional es un paradigma de programación declarativa basado en el concepto de función matemática, cuya principal característica es la ausencia del concepto de estado, ya que los programas funcionales puros no manejan datos mutables. Los resultados de las funciones sólo dependen de los parámetros de entrada.

Los programas escritos bajo este paradigma son construidos únicamente mediante funciones, dado que carecen de estado, verifican las propiedades de la transparencia referencial[1] y la carencia total de efectos laterales.

A diferencia de los lenguajes imperativos, estos lenguajes no poseen construcciones como secuencias o bucles, sino que se apoyan en el mecanismo de recursión para implementar todas la construcciones repetitivas. Además tampoco existe el concepto de asignación de variables, ya que todas las definiciones dentro de un programa son funciones o constantes.

Debido a que hay lenguajes funcionales que relajan alguna de las características anteriores, se establecen dos categorías: *puros* e *híbridos*. La diferencia fundamental está en que algunos lenguajes toman conceptos de los lenguajes imperativos como, por ejemplo, el uso de variables mutables. Al adoptar estos conceptos es difícil conservar la propiedad de transparencia referencial.

Entre los lenguajes funcionales puros se encuentra Haskell (Hudak et al., 2007), un lenguaje puramente funcional con semánticas no estrictas y fuerte tipificación estática[2]. Nació en los años 80 con el objetivo de crear un lenguaje funcional que aunara las principales características de los múltiples lenguajes existentes y permitiera su uso tanto en el mundo académico como en el profesional. Se ha convertido en un estándar de facto cuya última versión es Haskell 98 (Marlow y Jones, 2002), que especifica la versión mínima, compatible y libre del lenguaje. Las características más

[1] La *transparencia referencial* asegura que el resultado de una expresión siempre es el mismo independientemente de la historia del programa en ejecución y del orden de evaluación de otras subexpresiones

[2] Se dice de aquellos lenguajes cuya comprobación de tipos se realiza en tiempo de compilación. Esta comprobación permite prevenir errores de programación

interesantes son el soporte de tipos de datos, funciones recursivas, listas, tuplas, guardas y el ajuste de patrones.

La mayoría de las implementaciones de Haskell se distribuyen bajo licencia de código abierto y cumplen plenamente con el estándar Haskell 98. Quizá, la implementación más utilizada sea GHC (*Glasgow Haskell Compiler*) (Jones y Marlow, 2007). Desarrollado originalmente en la Universidad de Glasgow y continuado en el departamento de investigación de Microsoft por su creadores Peyton Jones y Simon Marlow. Licenciado como BSD (*Berkeley Software Distribution*) proporciona un entorno para la compilación y ejecución de programas soportando un gran número de extensiones, librerías, optimizaciones y proporcionando un interprete interactivo. GHCi (*GHC's interpreter*, Intérprete de GHC) (Iborra y Marlow, 2007) es el intérprete en línea de comandos que permite evaluar expresiones, interpretar programas, cargar módulos ya compilados y realizar operaciones de depuración.

1.2.1. Herramientas de visualización de datos

Los mecanismos de visualización que usan las distintas implementaciones del lenguaje a la hora de mostrar las estructuras de datos, están basadas en una conversión textual. Esta forma de mostrar los datos resulta insuficiente cuando se manejan tipos de datos complejos.

Existen algunos trabajos cuyo objetivo es facilitar la visualización de estructuras de datos complejas y ayudar a comprender mejor algunas de las características de los lenguajes funcionales. La mayor parte de estos trabajos se han implementado como extensiones o librerías utilizadas dentro del inteterprete GHCi, como Vacuum (Morrow y Seipp, 2009) y GHC-Viz (Felsing, 2012), y otros permiten observar y depurar las estructuras después de que el programa haya sido ejecutado, como GHood (Reinke, 2001).

Vacuum Esta librería, licenciada como LGPL3 (*Lesser General Public License Version 3*) y experimental, permite extraer representaciones gráficas de los valores de la pila de GHC en tiempo de ejecución. Dado que las representaciones generadas por si mismas no son de mucha utilidad,

se hace necesario traducirlas a formatos más legibles mediante el uso de otras librerías como: Vacuum-cairo (Stewart, 2009) o Vacuum-graphviz (Miljenovic y Seipp, 2009).

Vacuum-cairo, licenciado como BSD y experimental, permite la visualización gráfica de las estructuras de datos dentro del intérprete GHCi, mientras que Vacuum-graphviz, licenciado como LGPL3, permite convertir y exportar las representaciones gráficas generadas por Vacuum a imágenes PNG (*Portable Network Graphics*) o SVG (*Scalable Vector Graphics*).

La diferencia fundamental con el enfoque de nuestro trabajo, es que la representación que ofrecen estas librerías es estática, no permiten ningún tipo de parametrización e interación con la representación. Además, realizan una representación gráfica de las estructuras tal y como aparecen en el pila de GHC. Si la estructura a representar fuese, por ejemplo, un árbol de árboles, las representaciones serían un complejo gráfico de punteros. Las representaciones que nosotros proponemos son muchos más legibles y fáciles de manipular.

GHC-Viz Es una herramienta licenciada como BSD para la visualización dinámica de las estructuras de datos dentro del intérprete GHCi. Su objetivo es ayudar a los programadores a comprender como funciona la evaluación perezosa y como se comparten datos entre estructuras, a través de la representación gráfica de las mismas. Trabaja en combinación con el depurador GHCi, es decir, puede ser usado para mostrar las estructuras de datos mientras se realizan operaciones de cálculo. Esta forma de trabajar es posible, ya que abre un hilo de ejecución dentro del depurador que le permite mostrar paso a paso la evolución de las estructura de datos y mostrar diferentes representaciones.

Esta herramienta esta muy enfocada en mostrar el funcionamiento de dos características del lenguaje: la evaluación perezosa de estructuras y la compartición de datos. Nuestro enfoque se centra en realizar una representación parametrizada de estructuras de datos complejas.

GHood Es una extensión del depurador Hood (Gill, 2000). Este depurador se basa en la observación y traza de las estructuras de datos duran-

te la ejecución de un programa en GHC. Una vez finalizado, estas trazas, textuales, pueden ser evaluadas. GHood crea un animación sobre los datos obtenidos por el depurador, permitiendo realizar una evaluación más amigable. La manipulación y estudio de los datos se realizan fueran del entorno de compilación e intérprete de Haskell, mediante la ejecución de un programa Java. Este programa permite observar como cambian las estructuras de datos e incluye un proceso de evaluación perezosa de los datos.

Al igual que GHC-Viz, su objetivo es ayudar a comprender el funcionamiento de la evaluación perezosa del lenguaje y no en la representación de estructuras de datos complejas.

1.2.2. Herramientas en línea

Del mismo modo que se han creado proyectos para tratar de mejorar algunas características de Haskell para ofrecer ayudas en el desarrollo de programas, también se han desarrollado iniciativas en línea que proporcionan entornos de desarrollo, compiladores e intérpretes con el objetivo de facilitar el aprendizaje y desarrollo de programas.

Try Haskell es un intérprete[3] en línea con una funcionalidad limitada muy útil para tomar un primer contacto con el lenguaje. Proporciona un tutorial interactivo que sugiere expresiones comunes del lenguaje junto con una breve explicación de las mismas a medida que se avanza en el aprendizaje. El subconjunto de expresiones que pueden ser utilizadas son las soportadas por el intérprete Mueval[4], que proporciona un entorno seguro y reducido de librerías de GHC para evitar código malicioso.

Nosotros también proponemos incorporar un intérprete de expresiones. Sin embargo, proporcionamos sin restricción toda la potencia del intérprete GHCi y no nos limitamos al seguimiento de un tutorial.

GHC-IO es un intérprete[5] en línea nacido como caso de uso para la demostración de una nueva extensión del lenguaje Haskell implementada

[3] http://tryhaskell.org
[4] http://hackage.haskell.org/package/mueval
[5] http://ghc.io

en GHC denominada Safe Haskell (Terei et al., 2012). Esta extensión permite crear un entorno seguro de ejecución frente a código no confiable, limitando alguna de las operaciones de entrada/salida disponibles en el lenguaje, pero sin restringir la potencia del entorno GHC.

Este intérprete ofrece toda la potencia del interprete GHCi, pero no permite la gestión de módulos y, por tanto, tampoco permite trabajar con expresiones sobre los propios módulos que el usuario quiera probar.

Codepad es una sencilla aplicación web[6] *pastebin*[7] que permite compilar e interpretar código de distintos lenguajes. Permite la ejecución de pequeños ejemplos Haskell que son compilados con Hugs y pueden ser compartidos y de acceso público para servir de ayuda a otros programadores.

Es una herramienta libre pero muy primaria, cuya principal utilidad es poner en común pequeñas porciones de código que pueden ayudar y orientar a otros programadores. Quizá por su orientación a múltiples lenguajes tiene funciones muy generales y, aunque permite el registro de usuarios, carece de una gestión de módulos y de un intérprete de expresiones.

FP Complete creado por Aaron Contoner[8], un ex-alumno de la Fundación de Microsoft Coorporation, es el primer IDE basado en Web para Haskell. Actualmente el proyecto se esta enfocando en la implementación y comercialización de herramientas dirigidas a la industria. Aunque su página permite un uso gratuito, sus condiciones de uso son algo excesivas[9].

Se trata de un proyecto muy maduro que ofrece todas las posibilidades que son deseables en un IDE, pero carece de una consola interactiva que permita probar ciertas expresiones de los módulos que componen los proyectos sobre los que trabaja. Además, la licencia es demasiado restrictiva y para obtener más funcionalidades se requiere pagar una cuota mensual.

[6] http://codepad.org

[7] Son aplicaciones web que permiten a los usuarios compartir públicamente pequeños ejemplos de código.

[8] https://www.fpcomplete.com

[9] Entre sus términos de uso es muy llamativo el punto 4 (*Sharing content on the Website*) en el que se explica que al subir código a su sitio la compañia se reserva todo derecho de uso y explotación sobre éste de forma irrevocable y perpetua.

1.3. Objetivos

1.3.1. Motivación

Nuestra intención es desarrollar una herramienta educativa en línea similar a las descritas que permita a los programadores principiantes familiarizarse con la programación funcional y que, además, proporcione representaciones gráficas interactivas de las expresiones Haskell.

Todas las herramientas anteriores ayudan al programador en la tarea del desarrollo de programas, tanto en el proceso de depuración como en el de visualización y comprensión de ciertas características del lenguaje. El principal inconveniente de todas ellas es que, al ser extensiones de GHC, están orientadas a usuarios con una cierta experiencia tanto en la programación como en el manejo e instalación de la plataforma de desarrollo Haskell.

También hemos mencionado otras herramientas en línea que ponen a disposición de los usuarios intérpretes, tutoriales e IDEs que hacen más asequible acercarse al mundo de la programación funcional.

La diferencia fundamental de todas estas herramientas con la que proponemos en este trabajo está en que aunamos tanto la idea de tener un IDE para la gestión de los fuentes con un intérprete que permite ejecutar y trabajar con las expresiones asociadas a los fuentes. Y, sobre todo, en que la ponemos a disposición de los usuarios de forma totalmente libre, sin ningún tipo de restricción de uso.

1.3.2. Propósito inicial

La propuesta inicial de este trabajo es el desarrollo de un método de representación gráfica de estructuras de datos complejas de Haskell que permita la visualización en un navegador Web, debido a las facilidades que ofrece éste en cuanto representaciones gráficas.

Para ello se ha pensado en la creación de una consola web que permita evaluar las expresiones escritas en Haskell, al estilo de GHCi, con la diferencia fundamental de que los resultados puedan presentarse tanto en forma textual como gráfica.

Otra idea es que el usuario de la aplicación pueda subir módulos Haskell que serán compilados dentro del propio servidor. El usuario podrá acceder a la lista de módulos que ha subido. Cuando seleccione uno de ellos, se accederá a la consola en la que podrá introducir expresiones para su evaluación, que pueden incluir llamadas a las funciones contenidas en el módulo seleccionado.

El sistema debe permitir múltiples usuarios y, para cada usuario, debe permitir guardar todos sus módulos. Por tanto, cada usuario utiliza un identificador único y contraseña que le permite acceder a sus módulos e interactuar con la consola.

La aplicación debe proporcionar las siguientes funcionalidades:

- Autenticación de usuarios.
- Subida de módulos del ordenador del usuario al servidor.
- Eliminación de módulos.
- Edición y modificación de módulos.
- Ejecución de expresiones que contengan llamadas a funciones definidas en los módulos, y visualización del resultado tanto en forma textual como gráfica.

1.3.3. Objetivo final

Durante el proceso de desarrollo de la aplicación, sin embargo, se hacen necesarias una serie de funcionalidades adicionales para un mejor soporte y experiencia de uso al usuario.

Dado que la aplicación debe permitir el acceso a múltiples usuarios, se distinguen tres tipos de usuarios:

- Usuarios comunes que pueden acceder a la consola y gestionar sus módulos.
- Administradores de aplicación que, además de tener toda la funcionalidad de los usuarios comunes, pueden modificar los datos de los usuarios.

- Un administrador de servidor, encargado del alta y baja de todos usuarios de la aplicación.

Los usuarios, independientemente del perfil que tengan asignado, pueden gestionar sus módulos agrupándolos en proyectos lo que les permite una mejor organización. Por tanto, además de las acciones indicadas en el punto anterior (ver 1.3.2), pueden crear y eliminar proyectos, crear módulos Haskell nuevos y ejecutar cualquier expresión Haskell que utilice dichos módulos.

También se proporciona un proceso por el cual podrá restablecer su contraseña en el caso de que la haya olvidado.

Por el momento, la aplicación no proporciona un registro automático de usuarios. Los usuarios deberán solicitar al administrador del sitio el alta para poder acceder a al aplicación.

1.4. Plan de trabajo

1.4.1. Evaluación de herramientas y tecnología

La primera fase antes de comenzar con el desarrollo de la aplicación, teniendo definidos ya los requisitos funcionales que se deben cumplir, es realizar una evaluación de la tecnología y herramientas más apropiadas para el desarrollo e implementación.

Los criterios de evaluación utilizados para la elección de la tecnología son:

- Madurez de las herramientas y la experiencia en su manejo.
- Cumplimiento con los requisitos de la aplicación.
- Un diseño que cumpla con los estándares de usabilidad y accesibilidad.
- Uso en múltiples dispositivos: móviles inteligentes, tabletas, portátiles y ordenadores de sobremesa.
- Cumplimiento con unos estándares mínimos de seguridad y fiabilidad.

1.4.2. Entorno de desarrollo

En la segunda fase debemos crear el entorno de desarrollo. Para ello tomamos dos decisiones: crear una maquina virtual que contenga todo el software necesario para desplegar la aplicación y albergar el repositorio de fuentes, y seleccionar una herramienta de control de versiones que nos permite llevar una auditoria de los cambios realizados.

1.4.3. Maqueta: navegación y evaluación

En esta tercera fase, se ha elegido un modelo de desarrollo evolutivo, arrancamos el desarrollo a partir de un prototipo de navegación. Una vez realizado evaluamos la usabilidad, accesibilidad y navegabilidad de la aplicación.

1.4.4. Desarrollo de servicios

A la maqueta de navegación se le incorpora toda la lógica y servicios auxiliares para la gestión de usuarios, cambios de contraseña, gestión de proyectos e interacción con la consola de expresiones Haskell con el intérprete GHCi (servicio *PHPRest*).

1.4.5. Librería de soporte a la representación

Esta fase se dedica al desarrollo de la librería que permite realizar la visualización de las estructuras de datos Haskell en la consola.

1.4.6. Evaluación del producto

En esta última fase se realiza la evaluación del producto final. Es decir, se realizan pruebas funcionales de cada uno de los componentes que integran la aplicación, una auditoria de seguridad y la revisión de cumplimiento de los estándares de accesibilidad y usabilidad.

Capítulo 2

Propuesta de *hardware/software* y arquitectura

2.1. Arquitectura propuesta

Las contribuciones a GHC suelen realizarse como librerías instalables y descargables de la página de Haskell[1]. Sin embargo, una librería no aportaría la facilidad de uso y comodidad que se requiere para una herramienta educativa.

Además, un entorno de desarrollo de Haskell (GHC) requiere la descarga, instalación y configuración de múltiples paquetes de SW y, dependiendo del SO (*Sistema Operativo*) utilizado, puede requerir incluso compilar fuentes, hacer ajustes a la configuración global, etc.

Esta es la razón por la que la arquitectura propuesta está basada en el paradigma *cliente/servidor*, ya que requiere un único componente de software en cada cliente (navegador web), aunque el servidor (o nodo de servicio) requerirá una instalación más compleja.

2.1.1. Paradigma de arquitectura

Nuestra propuesta de arquitectura se basa en el paradigma *cliente/servidor*. Sin embargo, desde el punto de vista computacional, los dispositivos

[1] http://hackage.haskell.org

del lado del cliente tienen capacidad de cómputo suficiente, navegadores con un API (*Application Programming Interface*, Interfaz de programación de aplicaciones) muy avanzado y capacidades gráficas excelentes, por lo que proponemos una arquitectura que aproveche estos recursos.

Además, dado que el servidor ha de atender múltiples peticiones de ejecución simultáneamente y que las ejecuciones pueden tener un coste relevante, una arquitectura altamente distribuida parece la elección correcta. Balanceando la carga de computación de este modo podemos conseguir lo que podría denominarse una arquitectura de *cliente ligero* y *servidor ligero* ya que, a pesar de que la carga computacional global es alta, creemos que este balance libera suficientes recursos en cada parte para aligerar su carga individual.

2.1.2. Patrón de diseño

En cuanto al patrón de diseño, el hecho de elegir un cliente con cierta carga computacional (texto, interacción y representación gráfica), utilizando para ello sólo el API de los navegadores, suele tener un impacto importante en la mantenibilidad del código, por lo que optamos por elegir un patrón de diseño avanzado. Elegimos MVW (*Model-View-Whatever*, Modelo Vista y "lo que sea") por dos razones: porque permite mayor flexibilidad de otros patrones al elegir el modelo de control y porque existen librerías de JavaScript que lo implementan, aumentando aún más la mantenibilidad del código. En la práctica el modelo más cercano será MVC (*Model-View-Controller*, Modelo-Vista-Controlador), ya que la mayoría del trabajo se realizará utilizando controladores que no se comunican externamente.

2.1.3. Diagrama de despliegue de la arquitectura

En la figura 2.1 mostramos un posible despliegue de la arquitectura en el que se distribuyen los servicios en tres nodos: un primer nodo (*WebServer*) distribuye las páginas como contenido estático (HTML, CSS, JavaScript). Un segundo nodo, *WebServer* y *ExecutionServer*, sirve las peticiones REST (*REpresentational State Transfer*) de órdenes Haskell a interpretar. El tercer nodo, *DBServer*, sirve los recursos de autenticación, de repositorio

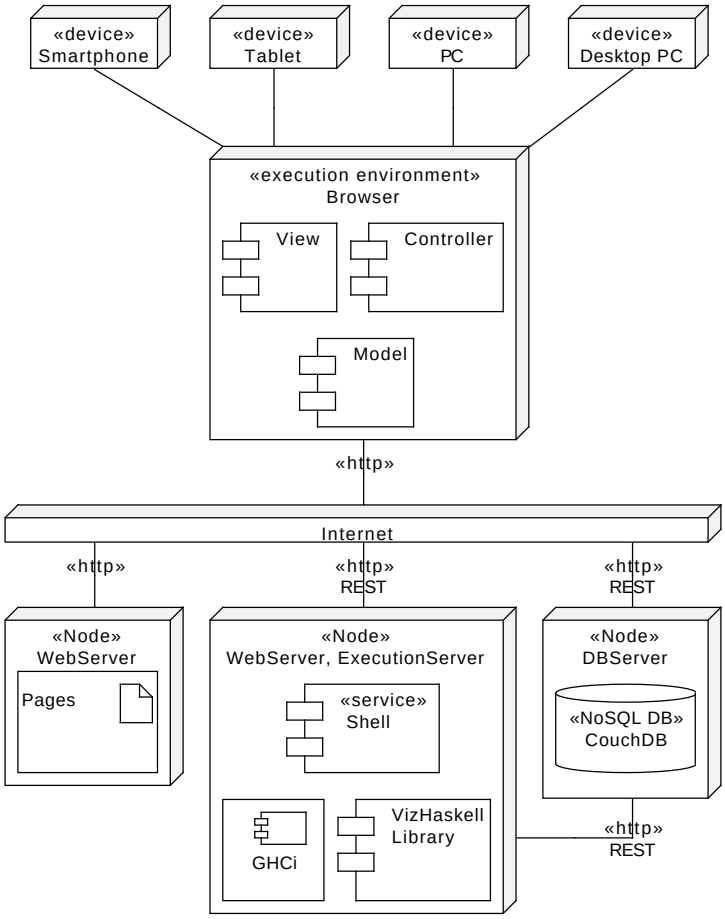

Figura 2.1: Diagrama de despliegue de distribución máxima de VizHaskell

de usuarios, proyectos y módulos.

Todos los dispositivos, posiblemente con sistemas diferentes, comparten una misma clase de entorno de ejecución, el navegador. En este entorno las capas MVC se comunican internamente entre sí, pero la comunicación externa sólo es realizada por la capa de modelo utilizando el protocolo HTTP (*Hypertext Transfer Protocol*, Protocolo de transferencia de hipertexto) en modo REST.

Puede observarse cómo los componentes de la aplicación deben desplegarse solo en los servidores *WebServer* y *ExecutionServer*, ya que los servicios web de persistencia son proporcionados por el gestor CouchDB (Anderson et al., 2010) de forma predeterminada.

Por último, nótese que existe una comunicación cliente/servidor entre el nodo *ExecutionServer* y el nodo *DBServer*. Esta comunicación es necesa-

ria para completar la fase de recuperación de módulos del proyecto activo en el servicio de evaluación, para comprobar la validez de la sesión de usuario y para implementar el cambio de contraseña, que se realiza con una autenticación especial.

En esta configuración los entornos de ejecución están equilibrados entre el cliente y los servidores, de modo que la carga de representación e interactividad es tarea del cliente y la carga de persistencia y ejecución de procesos es tarea del servidor.

2.2. Propuesta de *hardware*

2.2.1. *Hardware* del cliente

En los últimos años el acceso a Internet se ha popularizado para todo tipo de dispositivos. Estos dispositivos tienen en común varias capacidades clave: tienen una gran capacidad de cómputo gráfico y una pantalla decente. El inconveniente es la gran disparidad de sistemas operativos, lenguajes de programación y tamaños de pantalla que pueden presentar, por lo que la programación nativa se complica.

Por suerte, el API de los navegadores y su presencia en casi cualquier SO proporciona la independencia del dispositivo adecuada para llevar las aplicaciones a todos ellos.

Por todo ello nuestra propuesta para el HW del cliente es ambiciosa: pretendemos dar soporte multi-dispositivo, si bien no siempre es posible debido a las limitaciones de los navegadores antiguos. Nos apoyaremos para ello en capacidades comunes de hardware y estándares, así como en los marcos de diseño adaptable como Bootstrap (Cochran, 2012) que salvan en gran medida las diferencias entre navegadores.

2.2.2. *Hardware* del servidor

La aplicación que hemos desarrollado hace un uso intensivo de servicios de red, por lo que elegimos un SO en el que podamos obtener un al-

to rendimiento en este aspecto, como es el sistema GNU/Linux[1]. Nuestra propuesta requerirá un HW compatible con este SO, lo que nos permite elegir entre una gran gama de arquitecturas de procesador.

Los requisitos de los servidores en cuanto a capacidad de memoria y cómputo dependerán del número de usuarios al que se quiera dar servicio. No requerimos HW paralelo o HW dedicado, por lo que nuestra solución puede desplegarse en casi cualquier servidor de gama media para dar servicio a un conjunto de usuarios de un entorno educativo reducido.

La capacidad de comunicación es el aspecto más relevante, pero casi cualquier HW diseñado para ser utilizado como servidor de web puede utilizarse sin problema.

2.3. Propuesta de *software*

2.3.1. Elección del sistema operativo

Nuestra propuesta de SW comienza con la elección del SO. Para los clientes no existe elección. Sin embargo, al tratarse de un entorno multidispositivo requerimos lenguajes o entornos de ejecución en el cliente que corran en múltiples sistemas.

Nuestra propuesta es GNU/Linux (en cualquiera de sus "sabores") como SO, ya que es libre, estable, completo y amigable. Además, es el SO líder en servidores, especialmente en servidores web. La gran comunidad alrededor de este sistema garantiza una ayuda que no podemos encontrar en otros sistemas.

2.3.2. Comunicaciones

Hemos optado por asegurar las comunicaciones utilizando HTTPS (*Hypertext Transfer Protocol Secure*, Protocolo seguro de transferencia de hipertexto) para evitar la intercepción de contraseñas y para mejorar la seguridad en el transporte, a pesar de que la información de la aplicación no es muy sensible.

[1] http://www.drdobbs.com/which-os-is-fastest-for-high-performance/199101229

En cuanto al protocolo de servicios, optamos por REST. Es más ligero que SOAP (*Simple Object Access Protocol*) y está orientado a recursos. Los recursos son un artefacto directo en el patrón MVC y, además, tienen un soporte muy sencillo en AngularJS, por lo que parecen una elección natural.

Preferimos JSON (*JavaScript Object Notation*, Notación de objetos de JavaScript) a XML (*Extensible Markup Language*, Lenguaje de marcas extensible) como formato de transporte del protocolo REST porque es más ligero, por consistencia con el formato de CouchDB y D3 (*Data-Driven Documents*) y porque el núcleo de programación de la aplicación es JavaScript.

2.3.3. Lenguajes y componentes internos

La programación de los componentes internos del lado del cliente se realiza usando el patrón MVW en AngularJS (Knol, 2013). La configuración del editor CodeMirror y la representación gráfica requiere JavaScript con jQuery y D3. D3 (Bostock et al., 2011) es una librería de programación que proporciona separación entre datos (normalmente en formato JSON) y presentación, por lo que es ideal para nuestros propósitos.

La vista explota las capacidades de tratamiento de etiquetas personalizadas de AngularJS sobre una base de HTML5 y CSS3 con el marco de visualización Bootstrap (Cochran, 2012). Este marco de trabajo proporciona estilos, controles avanzados y diseños adaptables, por lo que es ideal para un entorno multidispositivo.

Para la implementación de nuestro servicio de ejecución de GHCi hemos elegido PHP (*PHP Hypertext Pre-processor*, recursivo de Preprocesador de Hipertexto PHP). PHP se puede ejecutar en casi cualquier servidor web utilizando un mínimo de recursos y tiene librerías excelentes para comunicaciones usando HTTP y SMTP (*Simple Mail Transfer Protocol*, Protocolo Simple de Transferencia de Correo).

2.3.4. Componentes externos

Nuestra propuesta incluye varias librerías y componentes JavaScript que se incluyen de manera estática en las páginas HTML5 de la aplicación:

- **AngularJS** versión 1.2.16 con los *plugins route*, *resource*, *file-upload*, *checklist-model*, *xeditable* y *base64*.
- **jQuery** versión 1.11.0.
- **Bootstrap** versión 3.1.1, configurada a medida.
- **CodeMirror** versión 4.2, incluye los *plugins haskell*, *css*, *closebrackets* y *matchbrackets*.
- **D3** versión 3.4.8.
- **wterm** versión 0.0.4 de Venkatakrihnan Ganesh, aunque fuertemente modificada para nuestras necesidades.

El compilador e intérprete de Haskell son GHC y GHCi, ya que es un requisito inicial. GHC es un estándar de facto y, además, la documentación y extensibilidad que proporcionan no tienen competidor entre otros productos equiparables.

2.3.5. El servidor de base de datos

Nuestra propuesta de servidor ligero incluye minimizar, también, la cantidad y complejidad de los componentes propios en el servidor. Por tanto proponemos la base de datos no-SQL CouchDB, ya que su interfaz de comunicación nativa está basada en servicios web utilizando el protocolo REST. Esto elimina la necesidad de creación de múltiples servicios o componentes que atendiesen las tediosas operaciones típicas de CRUD (*Create-Read-Update-Delete*, Creación-Lectura-Actualización-Eliminación) necesarias para soportar la persistencia de datos en la BD (Base de datos). Otra ventaja añadida de CouchDB es que proporciona autenticación y autorización usando *cookies* de sesión, por lo que esta tarea también podemos delegarla a la BD.

2.3.6. El servidor web

Elegimos Apache Webserver como la alternativa más popular. La comunidad y la base de conocimiento en Internet es una ventaja importante. Otra ventaja es la capacidad de *proxy* de Apache, necesaria para redirigir varios puertos HTTP a uno solo, lo que simplifica el despliegue de CouchDB.

2.3.7. Herramientas de desarrollo

Todo el desarrollo se ha realizado utilizando *VIM* como editor y *SVN* como control de versiones, ya que disponemos del editor y del repositorio centralizado en el mismo servidor de integración, por lo que se simplifica el ciclo de pruebas en gran medida.

Por supuesto, las pruebas han sido realizadas utilizando las últimas versiones de los navegadores que respetan los estándares HTML5 y CSS3 (puntuación superior a 400 en el índice de compatibilidad de Niels Leenheer[1]) tanto en versiones móviles como de escritorio. Algunas versiones antiguas, como la 10 y 17 de Firefox son soportadas, aunque con algunas capacidades ligeramente degradadas.

[1]Véase http://html5test.com/results/desktop.html, http://html5test.com/results/tablet.html y http://html5test.com/results/mobile.html

Capítulo 3

Descripción de la solución

3.1. Guía de arranque rápido y navegación

3.1.1. Manual de arranque rápido

Para poder utilizar la aplicación será necesario obtener una cuenta. Para ello deberá proporcionarse al *administrador del servidor* una dirección válida de correo electrónico y el nombre completo del usuario.

Una vez que el *administrador del servidor* realice el alta del usuario y le proporcione las credenciales de acceso de la cuenta solicitada, este podrá acceder a la aplicación autenticándose en la pantalla de *Inicio*.

Si la autenticación se realiza con éxito, se obtiene el acceso a la Consola. Para empezar a utilizarla el usuario debe crearse un proyecto nuevo utilizando el botón *Crear un nuevo proyecto* en el panel *Proyecto*. Una vez creado un proyecto nuevo, ya es posible, o bien crear nuevos módulos (botón *Crear un nuevo módulo* del panel *Módulo*) o bien subir módulos Haskell desde el navegador del usuario utilizando el botón *Añadir módulos* del panel *Módulo*.

Después de esto, el usuario ya puede introducir expresiones Haskell en la *Consola*, escribiendo tras el indicador Haskell que se encuentra bajo la solapa *GHC*. Las expresiones son enviadas al servidor y evaluadas por el intérprete *GHCi* que puede devolver una respuesta textual (si la respuesta no es representable gráficamente) o una representación gráfica. Existen también órdenes internas de la consola que permiten acceder a la funcionalidad de *GHCi* o acceder a atajos de expresiones Haskell. Las más

importantes son las siguientes:

- `:help, :h ó :?` Muestra la ayuda de las órdenes que pueden usarse. Si ejecutamos `:help<nombre de orden>` nos muestra una ayuda más extensa de la orden.
- `:browse` Muestra los identificadores exportados por un módulo Haskell.
- `:info` Muestra información disponible sobre los identificadores indicados.
- `:kind` Infiere el tipo de una expresión dada.
- `:main` Ejecuta la función principal de un módulo.
- `:type, :t` Muestra el tipo de una expresión dada.
- `:print-console` Imprime el contenido de la consola.
- `:reptree` Visualiza como un árbol la expresión dada.
- `:repstr` Visualiza como texto la expresión dada.
- `:repraw` Visualiza como árbol JSON la expresión dada.
- `:reptot` Visualiza como árbol de árboles la expresión dada.
- `:clear` Limpia el contenido de la consola.

Las funciones `:reptree`, `:repstr`, `:reptot` y `:repraw` son atajos que permiten ejecutar expresiones de representación gráfica en el servidor mediante el intérprete GHCi.

3.1.2. Navegación

Existen dos áreas fundamentales para navegar por la aplicación:

Barra de menú Situada en la esquina superior derecha de todas las páginas, permite moverse entre las páginas más importantes: *Consola*, *Contraseña* y *Desconectar*. El usuario administrador tendrá acceso también a la página de *Usuarios*.

Enlaces del pie de página Permite navegar a las páginas informativas de la aplicación, como son la página de *Diseño y tecnología*, el *Mapa del sitio* o la página de *Acerca de*.

Es importante saber que al abandonar la página de la *Consola*, los módulos abiertos se cerrarán, de modo que si hay algún cambio pendiente de salvar, el usuario será avisado para confirme o aborte la acción y así pueda tener la oportunidad de guardar sus cambios.

El único enlace que no aparece en alguna de estas áreas es el de *Restablecer contraseña*, que se encuentra sólo en la pantalla de entrada por comodidad.

3.2. Manual de usuario

A continuación se describen las páginas y componentes de la interfaz que el usuario debe conocer para utilizar más cómodamente la aplicación.

3.2.1. Páginas auxiliares y descripción de la navegación

3.2.1.1. Entrada a la aplicación

La página de entrada (figura 3.1) es la puerta de acceso a la aplicación (se redirige aquí si el usuario no está autenticado e intenta acceder a cualquier otra página que lo requiera). Contiene un enlace para acceder a la página de restablecimiento de contraseña, en caso de que el usuario no recuerde su contraseña y desee que le sea restablecida para poder acceder de nuevo.

Si el usuario introduce un correo electrónico y una contraseña válidas, se redirigirá la la página de la Consola. En caso contrario se mostrará un mensaje de error y no se dará acceso a la aplicación.

3.2.1.2. Diseño y tecnología

Esta es una página pública que describe las librerías y componentes externos usados por la aplicación, explicando para qué se utilizaron y enlazando a las páginas oficiales de cada producto (figura 3.2).

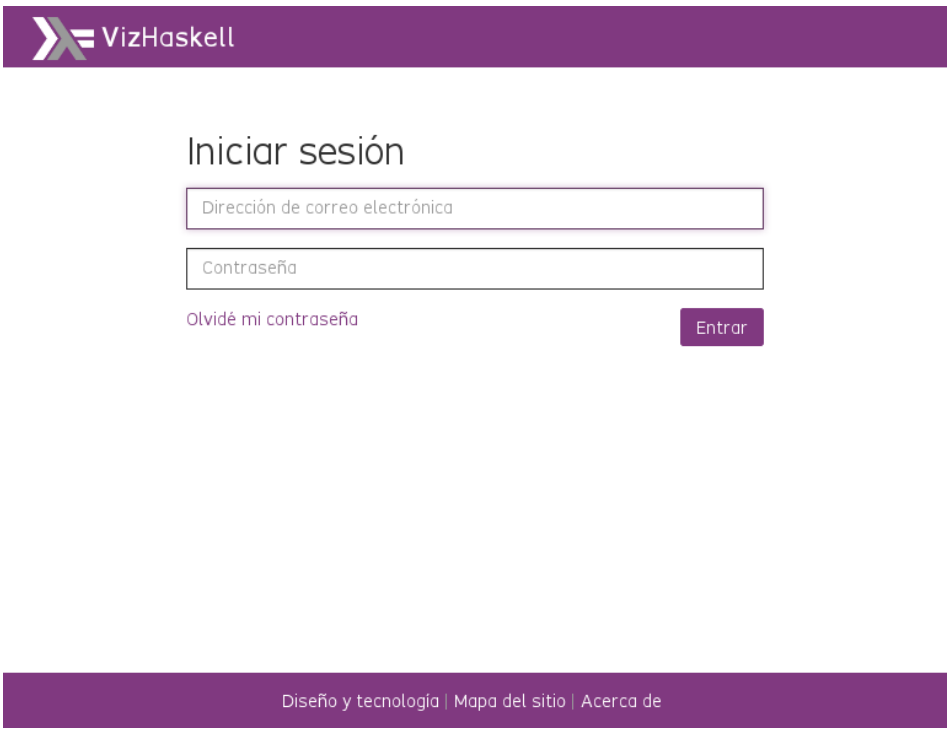

Figura 3.1: Página de entrada a la aplicación

3.2.1.3. Mapa del sitio

Esta página (figura 3.3) se adapta al perfil de acceso del usuario mostrando los enlaces a los que el usuario tiene acceso y su jerarquía de navegación. El mapa está disponible también públicamente, en cuyo caso sólo mostrará las páginas públicas accesibles.

3.2.1.4. Acerca de

Esta es la página pública inicial de la aplicación (figura 3.4) y contesta a las preguntas más básicas sobre ella de modo que el visitante ocasional tenga una idea rápida de lo que es y de cómo se usa.

3.2.1.5. Gestión usuarios

La página de *Usuarios* (figura 3.5) permite al administrador gestionar la lista de usuarios de la aplicación. Comprende el alta, baja, consulta y modificación de los usuarios del repositorio de usuarios. Este repositorio se utilizará para la autenticación y para el control de acceso de la aplicación.

Figura 3.2: Página de diseño y tecnología

Alta Pulsando el botón *Añadir usuario*, el administrador podrá introducir el nombre (completo o alias), la dirección de correo electrónico y la contraseña asignada al usuario a dar de alta en las cajas de texto ubicadas bajo la cabecera de las columnas de la tabla de usuarios que se muestra en la página. Pulsando el botón *Guardar cambios* la información será introducida en la base de datos, siempre que no exista la dirección de correo electrónico del usuario ya en el repositorio y que se introduzca el nombre y la contraseña correctamente. En caso contrario se mostrará un mensaje de error indicando el problema y se abortará el alta.

Baja Utilizando el botón *Eliminar usuario* que se encuentra a la derecha de cada línea de la tabla de usuarios, el administrador solicitará la baja del usuario y de todos sus datos del repositorio. Se mostrará una ventana de confirmación y, si se acepta, se procederá al borrado del usuario del repositorio.

Modificación La tabla de usuarios permitirá la edición en la propia tabla de todos los detalles de cada usuario al pulsar en el botón *Editar usuario*.

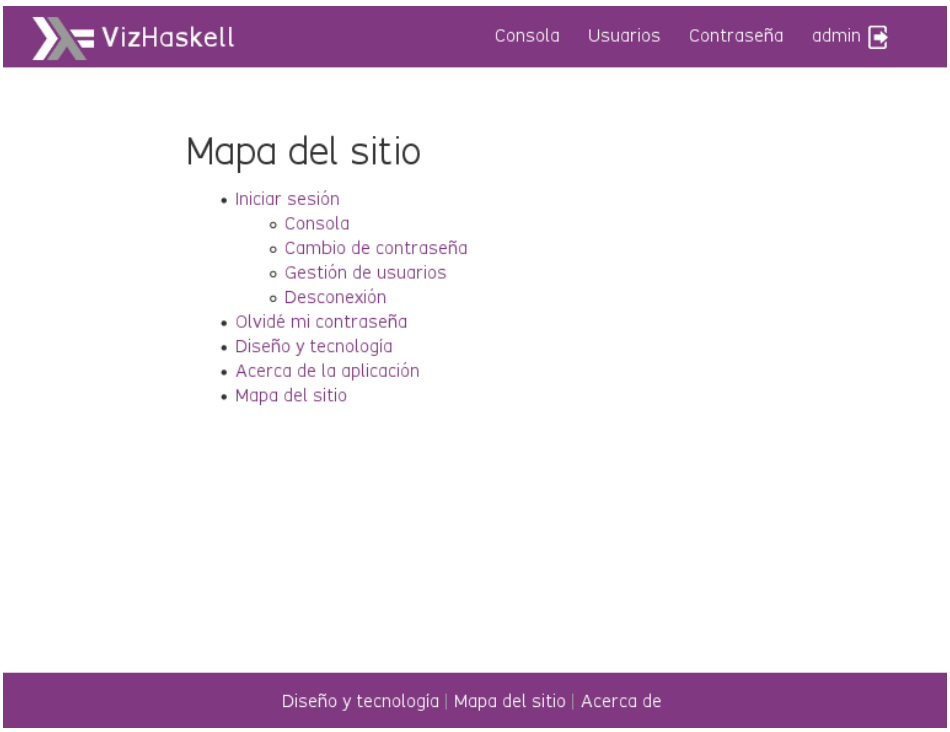

Figura 3.3: Página de mapa del sitio

Nuevamente habrá que pulsar el botón *Guardar usuario* para que los cambios se realicen en el repositorio.

3.2.1.6. Cambio de contraseña

Cualquier usuario puede solicitar un cambio de su contraseña utilizando la página de cambio de contraseña (figura 3.6). Esta página requiere autenticación y solicita una contraseña de más de ocho caracteres dos veces, para mayor seguridad.

3.2.1.7. Restablecimiento de contraseña

En esta página (figura 3.7) se solicita la dirección de correo válida del usuario y su nombre completo (nombre con el que se registró en la aplicación); de este modo se minimizan los intentos de restablecimiento de contraseña por parte de usuarios maliciosos. El sistema generará una contraseña aleatoria de ocho caracteres y la enviará por correo electrónico a la dirección especificada. Entonces cambiará la contraseña en el repositorio de modo que el usuarios pueda utilizarla en su próxima sesión.

Figura 3.4: Página de información acerca de la aplicación

3.2.2. La consola

3.2.2.1. Gestión de proyectos

Una vez autenticado, el usuario podrá utilizar la *Consola* (figura 3.8) no sólo para evaluar expresiones Haskell e interactuar con las representaciones gráficas, sino también para gestionar sus proyectos, subir y añadir módulos, así como utilizar el editor de módulos Haskell y CSS para mantenerlos.

Las funciones más habituales se describen a continuación:

Gestión de proyectos La gestión de proyectos comprenderá la posibilidad de crear nuevos proyectos (pulsando el botón *Crear un nuevo proyecto* y tecleando el nombre del proyecto) y de eliminar el proyecto actualmente seleccionado (pulsando el botón *Eliminar el proyecto seleccionado* y confirmando la acción para mayor seguridad). En este caso, todos los módulos asociados al proyectos también se eliminarán.

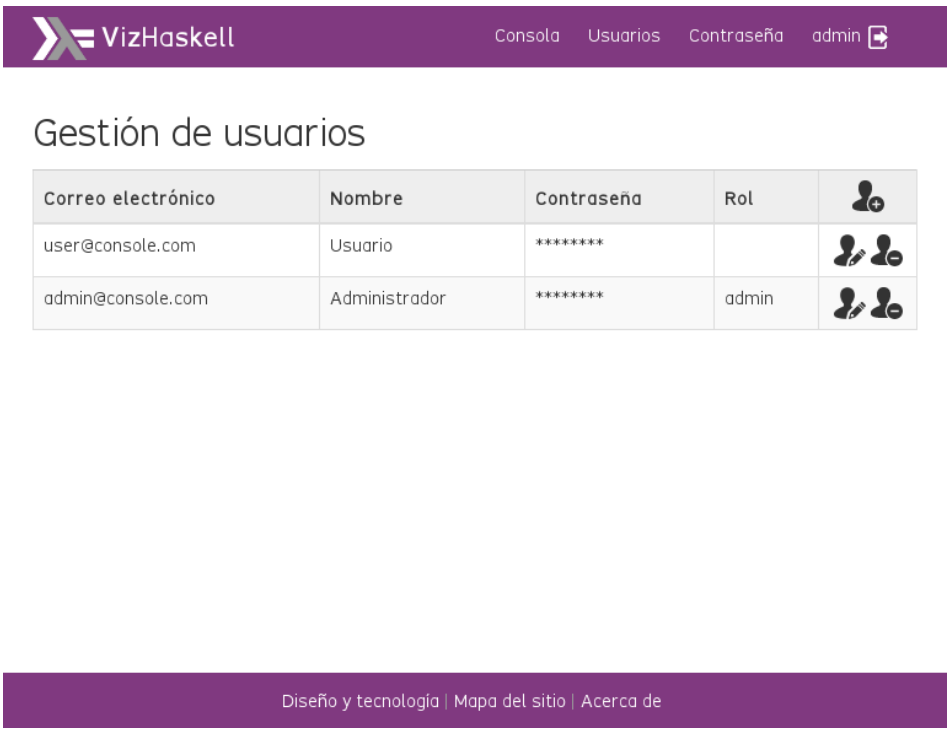

Figura 3.5: Página de gestión de usuarios

Añadir módulos Una vez que el usuario haya creado un proyecto nuevo o haya seleccionado un proyecto existente, se habilitará el botón *Anadir módulos* en el panel *Módulo*. En los navegadores que soportan subida múltiple de ficheros, esto abrirá un diálogo de selección múltiple de ficheros. Si se seleccionan varios módulos, estos se añadirán *temporalmente* al proyecto actuamente seleccionado en espera de confirmación. La confirmación puede realizarse pulsando, uno a uno, el botón *Subir el módulo nuevo al proyecto* a la derecha de cada módulo nuevo o todos a la vez, pulsando el botón *Subir todos los módulos* en la cabecera del panel *Módulo*. También es posible cancelar la subida de uno (o varios) módulos pulsando en el botón *Cancelar la subida del módulo* que aparecerá a la derecha del nombre de cada módulo. Si sólo se seleccionó un módulo, el botón de *Subir todos los módulos* no aparecerá.

Editar módulo Una vez haya módulos subidos en el proyecto, el usuario tendrá la posibilidad de editarlos y guardarlos en el servidor. Para editar un módulo, puede pulsar el botón *Editar módulo*, lo que abrirá una solapa nueva con un editor de texto con resaltado de sintaxis. Aquí el usuario po-

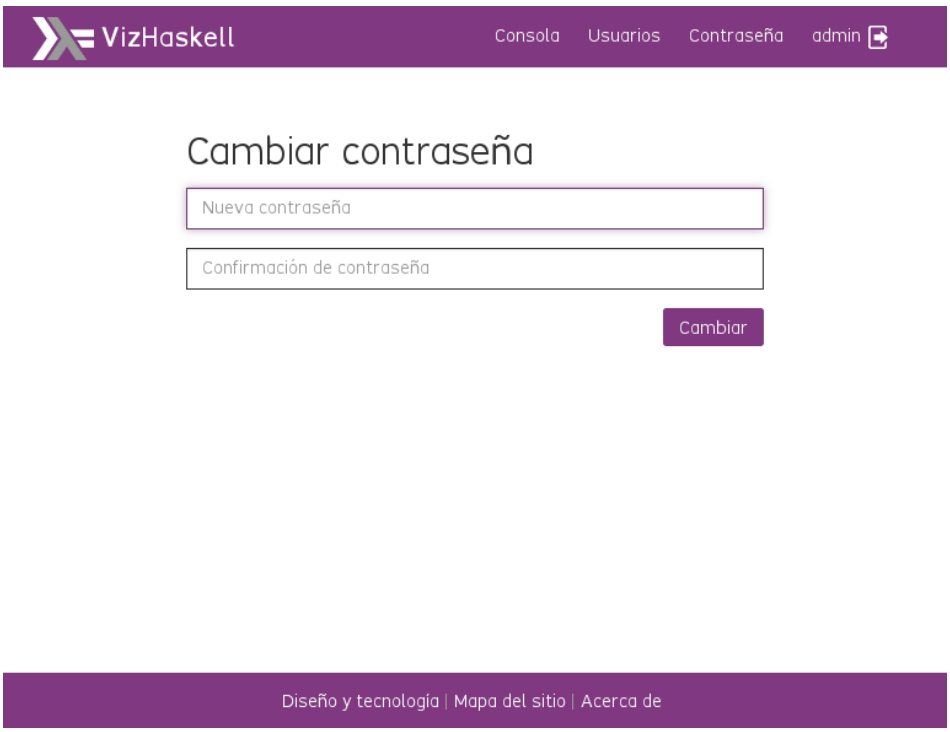

Figura 3.6: Caso de uso del cambio de contraseña

drá modificar el módulo, en cuyo caso se mostrará en la solapa el botón *Guardar módulo*. Alternativamente, es posible pulsar la tecla `CTRL+S` para salvar el contenido del módulo en el servidor. Podrá editar varios módulos a la vez, pero los cambios solamente estarán presentes en la siguiente evaluación de expresión que realice si los cambios han sido guardados. Es posible editar módulos de Haskell, de *literate* Haskell y módulos CSS (figura 3.9).

Eliminar módulo Los módulos subidos o creados en un proyecto se pueden eliminar usando el botón *Eliminar módulo*. Esta acción eliminará el módulo del proyecto en el servidor de forma irreversible, de modo que se solicita confirmación para mayor seguridad.

Crear un módulo nuevo Es posible también crear módulos vacíos (utilizando el botón *Crear un módulo nuevo* del panel *Módulo*. Esta acción creará un nuevo fichero del tipo módulo Haskell, módulo *literate* Haskell u hoja de estilos y lo añadirá al proyecto actualmente seleccionado.

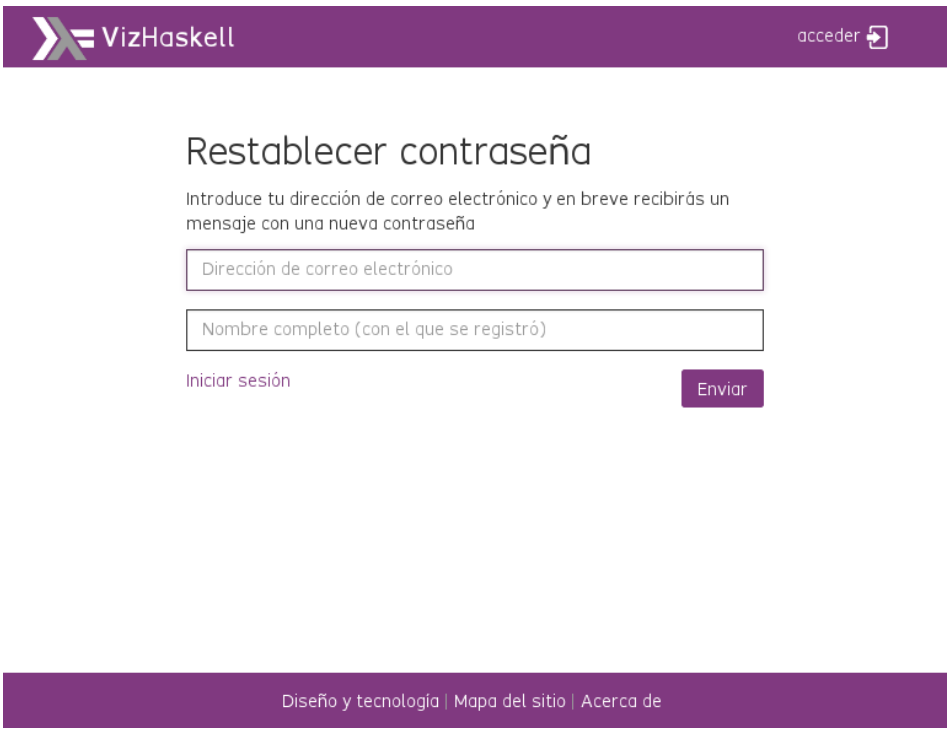

Figura 3.7: Página de solicitud de restablecimiento de contraseña

3.2.2.2. Evaluación de expresiones

Utilizando la solapa *GHC*, el usuario podrá escribir expresiones de Haskell y éstas serán enviadas al servidor para su evaluación por el intérprete de Haskell (GHCi). Además de la expresión escrita, se mandarán como importaciones los módulos del proyecto seleccionado, de modo que se dispondrá de los tipos de datos e importaciones que se definan en estos módulos.

La salida del intérprete de Haskell será mostrada en la consola después del texto de la expresión tecleada (o de su sustitución si se utilizó un atajo). Si la expresión escrita es representable gráficamente (cumple el *contrato de representación* descrito en la sección 3.3), se mostrará inmediatamente, y será posible interactuar con ésta (expandir nodos, representar subárboles, etc.). Si no es así, se mostrará la salida textual del intérprete.

Si la expresión introducida tiene algún error o si el intérprete encuentra algún problema en alguno de los módulos importados por el usuario, se mostrará el texto completo del error en la consola bajo la expresión. El usuario tendrá entonces la posibilidad de corregir el error, ya sea en la

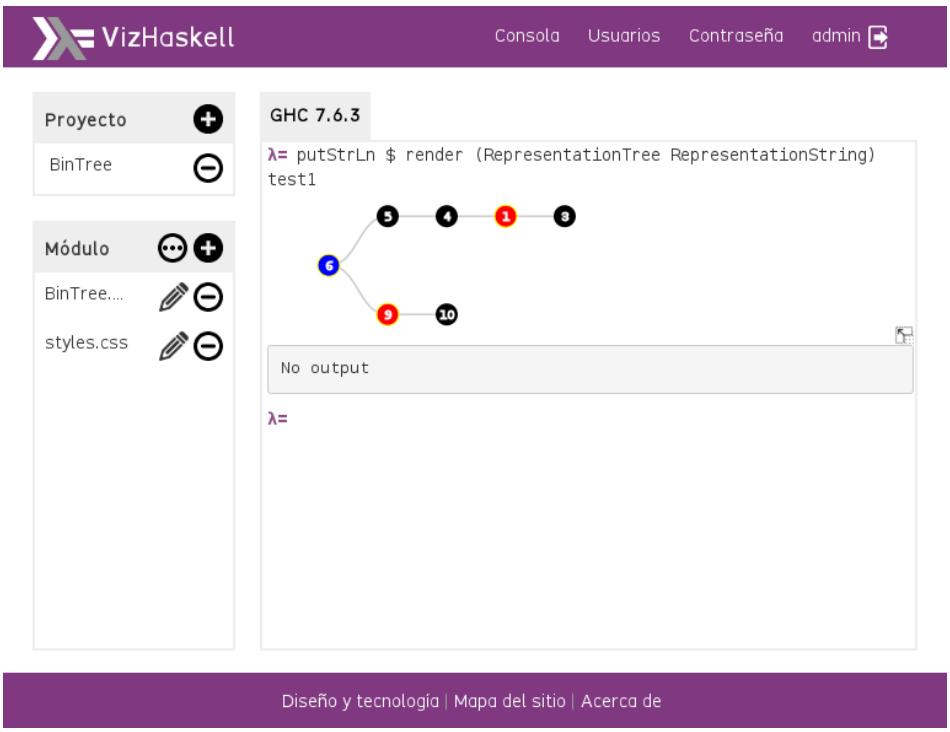

Figura 3.8: Página de la consola

expresión tecleada o editando el módulo o módulos que se indican en el error devuelto.

3.2.3. Accesibilidad y usabilidad

La aplicación cumple los estándares WGAC 2.0 AAA de accesibilidad (verificado usando AChecker), HTML5 y CSS3 (verificado usando los validadores de la W3C). La usabilidad de la aplicación es una prioridad y, por ello, se ha primado la facilidad y simplicidad de uso en todos los dispositivos por encima de otras consideraciones, al tratarse de una herramienta educativa.

3.3. Visualización de datos en VizHaskell

3.3.1. Alternativas de programación de la visualización

Los resultados de la evaluación de expresiones Haskell en la consola se muestran por defecto como texto sin formato. Sin embargo, es posible mostrarlos de forma gráfica cumpliendo con un sencillo *contrato de repre-*

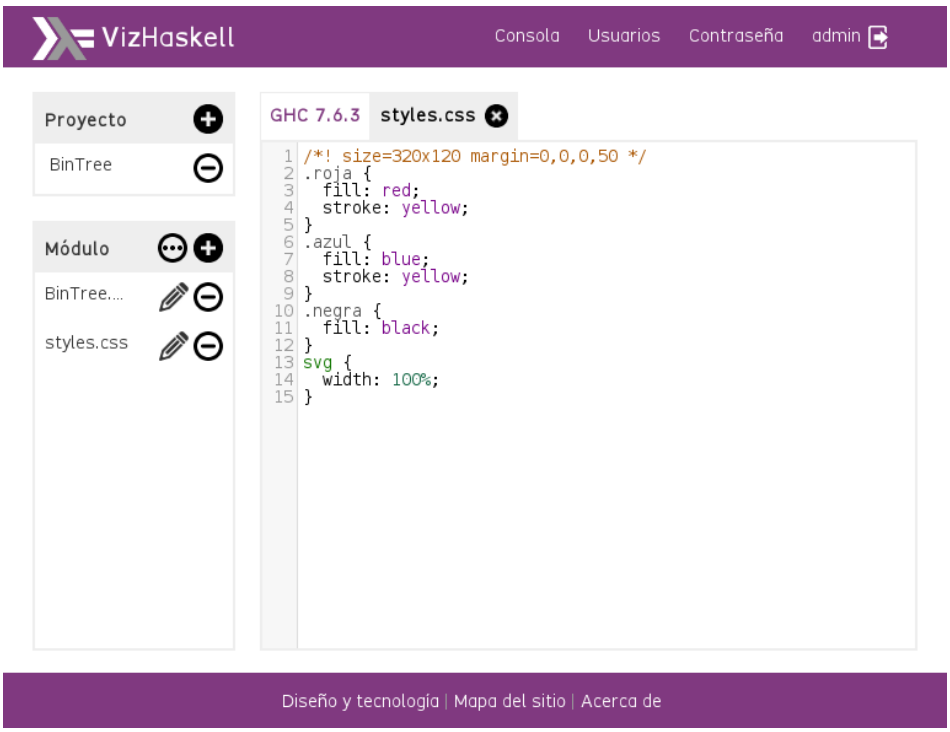

Figura 3.9: Consola con un editor CSS

sentación.

En la consola de VizHaskell, por convenio, cualquier resultado de una evaluación que produzca una cadena de texto JSON válida y que, además, contenga un campo `repType` con un valor soportado por el sistema de representación (actualmente sólo son válidos los valores `tree` y `text`), será considerada una representación gráfica y, por tanto, será traducida a un gráfico SVG que el navegador interpretará como un gráfico interactivo.

Por tanto, el programador de Haskell puede dotar a sus tipos de funciones que, cumpliendo este convenio, produzcan estructuras representables. A continuación describiremos los dos métodos actualmente soportados para conseguir este objetivo.

3.3.1.1. Implementación de bajo nivel: `ToJSON`

El contrato JSON puede cumplirse directamente implementando la función `toJSON` de la clase `ToJSON` de la librería `Data.Aeson` en los tipos definidos por el usuario. La ventaja principal de este método es su simplicidad e inmediatez, aunque tiene como inconvenientes que el programador de-

be tener cierto dominio de la librería `Data.Aeson` y que la representación generada puede ser incorrecta (por ejemplo si olvida el atributo `repType`).

3.3.1.2. Librería de soporte de representación VizHaskell

Un método alternativo es usar la librería de soporte que VizHaskell importa de forma automática. Esta librería presenta, no sólo tipos más sencillos de implementar, sino también una jerarquía de clases que permiten la incorporación de nuevos tipos de representación. La librería consta de cuatro clases principales:

`Representation` Clase base, sin métodos, ancestro de todas las clases de representación y que sirve de "marca" de las instancias que son representables (listado A.1).

`RepresentationString` Clase asociada al tipo de representación `text` que implementa la representación en forma de cadena de texto plano. Además, especifica que todas las instancias de *Show* pueden ser también representables como `RepresentationString` (listado A.2).

`RepresentationTree` Clase asociada al tipo de representación `tree` que implementa la representación en forma de árbol. Los tipos de usuario que utilicen esta clase deberán implementar obligatoriamente los métodos `contents` (contenido en cada nodo) y `children` (array de hijos de cada nodo) y opcionalmente los métodos `label` (una etiqueta que identifica el nodo de forma explícita en representaciones concretas) y `className`, un nombre de clase que se utilizará en el contrato CSS para dar estilos distintivos a cada nodo (listado A.3).

`RepresentationRaw` Clase que presupone que la instancia deriva de la clase `ToJSON` y, por tanto, implementa el método `toJSON` que devuelve una cadena JSON válida que cumple el contrato de representación. Esta clase es un "puente" entre la implementación de bajo nivel y la librería de soporte que permite, por tanto, utilizar los métodos comunes de la librería con tipos hechos a medida instanciando la clase `Data.Aeson.ToJSON` (listado A.4).

3.3.2. Librería de representación D3 de VizHaskell

Una vez que el programador ha implementado las funciones que cumplen el contrato JSON de representación y que ha realizado una petición de evaluación que se resuelve en una estructura representable, esta es recibida por el navegador para su conversión en un objeto gráfico.

Esta tarea la realiza la librería de representación D3 (`tree.js`), que es capaz de interpretar el contrato JSON y mostrar un gráfico SVG interactivo. Este gráfico, además, puede contener clases CSS (si el programador así lo decidió), por lo que pueden ser personalizadas con toda la potencia del lenguaje de estilos.

La representación se realiza nodo a nodo, ya que cada nodo debe tener un tipo de representación (`repType`) que indique a la librería cómo debe mostrarlo. Así se posible, por ejemplo, mostrar árboles cuyos nodos contengan árboles o simples etiquetas de texto.

3.3.3. Contratos de representación

Los contratos de representación establecen qué características de la representación gráfica actual pueden cambiarse y cómo debe hacerse, con independencia del método de implementación que se desee usar. Definen sólo unas pautas mínimas, pero pueden ampliarse para mejorar el rango y capacidades de nuevos métodos de representación, como veremos más adelante.

3.3.3.1. El contrato JSON

El contrato de visualización JSON define la estructura del documento de datos a representar gráficamente, con independencia de sus atributos visuales. Se define a partir de un documento JSON válido que define un objeto con los siguientes campos mínimos:

`repType` Cadena textual presente en cada objeto representable, indica el tipo de representación del objeto. Puede tener como valor `tree` o `text` (valor por defecto si `repType` está ausente).

value Valor del objeto representable y, por tanto, con su propio `repTyp e` que puede o no coincidir con su nodo contenedor. Así es posible, por ejemplo, que un objeto contenga un árbol (`repType="tree"`) o una simple etiqueta de texto (`repType="text"`)

className Cadena de texto opcional que representará la clase (o clases) que un objeto gráfico llevará asociada. Este atributo permitirá, por tanto, vincular el aspecto visual a cada objeto en virtud del contrato CSS.

Adicionalmente, los objetos del tipo de representación árbol (`repType ="tree"`) pueden llevar los siguientes campos:

children Array presente en cada nodo con "hijos" (aunque puede estar vacío) y, por tanto, raíz o rama de un árbol. Cada hijo debe ser, por supuesto, un nuevo objeto JSON con la misma estructura que su ancestro.

label Etiqueta que identificará un nodo (no necesariamente de forma única) y que servirá para mejorar la representación.

En el caso de objetos del tipo de representación texto (`repType="tex t"`), el texto del valor vendrá en un campo llamado **text**, que contendrá como valor una cadena de texto.

3.3.3.2. El contrato CSS

El contrato CSS es un sencillo método de asociar estilos de representación a cada nodo en las representaciones gráficas. Tiene dos partes:

1. El módulo CSS `styles.css`: si el proyecto activo contiene un módulo llamado `styles.css`, su contenido se enviará solo al elemento contenedor del gráfico SVG, lo que significa que los estilos que se definan en este módulo aplicarán solamente a ese gráfico. De este modo se evita que los estilos puedan provocar cambios en toda la página o que afecten a gráficos de resultados anteriores. Si el contenido del módulo CSS cambia, los estilos aplicarán a los nuevos gráficos, pero no a los ya mostrados.

2. El método `className`: Si el programador de tipos optó por implementar este método, los nodos gráficos llevarán una clase CSS con el valor que en el método se indicó y, de acuerdo al contrato, los estilos definidos bajo este nombre de clase en el módulo CSS `styles.css` serán aplicados, cambiando así el aspecto del nodo de forma consistente.

3.3.4. Ejemplos

3.3.4.1. BSTree

Listado 3.1: Código fuente del módulo `BSTree.hs`

```
1  module BSTree where
2
3  import Data.String
4  import VizHaskell.TreeRepresentation(TreeRepresentable
     (..))
5
6  data BSTree a = Empty | Node (BSTree a) a (BSTree a)
7    deriving (Show, Eq, Ord)
8
9  instance TreeRepresentable BSTree where
10   contents Empty        = Nothing
11   contents (Node _ x _) = Just x
12
13   children  Empty               = []
14   children (Node Empty x Empty) = []
15   children (Node l x Empty)     = [l]
16   children (Node Empty x r)     = [r]
17   children (Node l x r)         = [l,r]
18
19   className Empty = Nothing
20   className (Node l _ Empty) = Just "negra"
21   className (Node Empty _ r) = Just "roja"
22   className (Node _ _ _) = Just "azul"
23
24 insert :: Ord a => a -> BSTree a -> BSTree a
25 insert x Empty = Node Empty x Empty
26 insert x (Node l y r)
```

```
27    | x == y = Node l y r
28    | x <  y = Node (insert x l) y r
29    | x >  y = Node l y (insert x r)
30
31  test1 :: BSTree Int
32  test1 = foldr insert Empty [10,9,3,1,4,5,6]
```

En el listado 3.1 se muestra un ejemplo de cómo instanciando la clase `TreeRepresentation` de la librería de soporte a la representación de VizHaskell puede cumplirse el contrato JSON de representación de forma muy sencilla.

En las líneas 10 y 11 se implementa la función `contents`, de modo que la librería tenga información de cómo conseguir los contenidos de un nodo para después generar el campo `value` del contrato JSON.

En las líneas 13 a 17 se implementa la función `children`, y así se explica la relación *ancestro-descendiente* entre los nodos que forman el árbol.

Finalmente, en las líneas 19 a 22 se implementa parte del contrato CSS, al describirse cómo se producirá el campo `className` del contrato, lo que permitirá vincular los estilos visuales a cada nodo. Nótese cómo los nodos se colorean de negro si su hijo es inferior, rojo si es superior o azul si tiene hijos inferior y superior al mismo tiempo.

Por supuesto, para obtener el gráfico coloreado habría que cumplir la segunda parte del contrato, lo que requeriría añadir al proyecto el módulo `styles.css` con un contenido similar al siguiente:

```
.negra {
  fill: black;
}
.roja {
  fill: red;
}
.azul {
  fill: blue;
}
```

La orden a ejecutar en el intérprete para obtener el árbol gráfico sería:

```
putStrLn $ render (RepresentationTree RepresentationString) te
```

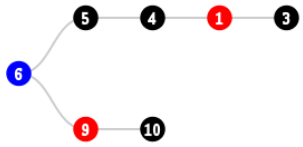

Figura 3.10: Árbol gráfico generado a partir de la función `test1` del módulo `BSTree`

`st1`, que indica que se quiere representar un objeto como árbol, mientras que sus elementos se quieren representar como cadenas. El resultado es el que se muestra en la figura 3.10.

3.3.4.2. `IntTree`

Listado 3.2: Código fuente del módulo `IntTree.hs`

```
1  {-# LANGUAGE OverloadedStrings #-}
2  module IntTree where
3
4  import Data.Aeson
5  import Data.Aeson.Types(Pair)
6
7  data IntTree = Empty | IntTree
8    { left :: IntTree,
9        val :: Int,
10     right :: IntTree } deriving (Show)
11
12 instance ToJSON IntTree where
13   toJSON (IntTree i v d ) = case (i, d) of
14     (Empty, Empty) -> addValue [] v
15     (Empty,     d) -> addValue [d] v
16     (i    , Empty) -> addValue [i] v
17     (i    ,     d) -> addValue [i, d] v
18     where addValue :: [IntTree] -> Int -> Value
19           addValue p v = object [
20             "repType" .= String "tree",
21             "children" .= p,
22             "value" .= object [
23               "repType" .= String "text",
```

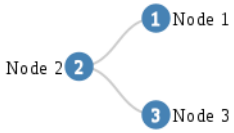

Figura 3.11: Árbol gráfico generado a partir de la función `it1` del módulo `IntTree`

```
24                "text" .= (show v)
25                ],
26              "label" .= ("Node " ++ show v)
27              ]
28
29 it1 :: IntTree
30 it1 = IntTree (IntTree Empty 1 Empty) 2 (IntTree Empty 3
     Empty)
```

En el listado 3.2 se muestra un ejemplo de cómo implementar el método `toJSON` para cumplir el contrato JSON de representación.

En las líneas 14 a 17 se distinguen los diferentes casos de nodos con y sin contenido en sus subárboles izquierdo y derecho, pero siempre se construye una lista de objetos que se emitirá en el campo JSON `children` (línea 21). Obsérvese cómo cualquier objeto JSON (sea un nodo o el contenido de un nodo) debe llevar un campo de tipo de representación (líneas 20 y 23), de acuerdo al contrato.

Dado que la representación en texto plano es la dada por la función `show` automáticamente derivada (línea 10), la salida de la orden `print it1` o, simplemente `it1` sería:

```
IntTree {left = IntTree {left = Empty, val = 1,
 right = Empty}, val = 2, right = IntTree {left
 = Empty, val = 3, right = Empty}}
```

La representación gráfica la podemos obtener con la orden `putStrLn $ unpack $ encode it1`. En este caso se obtiene el árbol de la figura 3.11, cuyo JSON válido sería similar al siguiente:

```
{
```

```
    "repType": "tree",
    "label": "Node 2",
    "value": {
        "repType": "text",
        "text": "2"
    },
    "children": [
        {
            "repType": "tree",
            "label": "Node 1",
            "value": {
                "repType": "text",
                "text": "1"
            },
            "children": []
        },
        {
            "repType": "tree",
            "label": "Node 3",
            "value": {
                "repType": "text",
                "text": "3"
            },
            "children": []
        }
    ]
}
```

Nótese que podríamos haber obtenido los mismo resultados usando la clase de representación `RepresentationString` para obtener la representación de cadena compatible con `Show` y `putStrLn $ render RepresentationRaw it1` para obtener la representación como árbol, lo que demuestra que ambos métodos son intercambiables, ya que ambos cumplen el contrato JSON.

Capítulo 4

Diseño, desarrollo e instalación de la solución

4.1. Diseño de la solución

4.1.1. Dispositivos y navegadores soportados

La aplicación está preparada para funcionar correctamente con diferentes tipos de dispositivos y con diferentes tamaños de pantalla.

El diseño es adaptable, por tanto, al tamaño de la pantalla del navegador, permitiendo conmutar entre cuatro tamaños diferentes de pantalla, que podrían corresponderse, típicamente con *móvil inteligente* (figura 4.1), *tableta* (figura 4.2), *ordenador personal* (figura 4.3) de pantalla mediana y *ordenador de sobremesa* (figura 4.4) con pantalla grande.

En cuanto a los navegadores soportados, se ha probado la compatibilidad con las últimas versiones de los navegadores Chromium, Firefox y Opera en el escritorio y Chrome, Android Browser, Firefox Mobile y Opera Mobile en un *móvil inteligente*. Como ya se ha mencionado, se requiere un soporte mínimo de HTML5, por lo que navegadores antiguos presentan una calidad de uso y funcionalidad deficiente con HTML5. En general, la compatibilidad con navegadores vendrá determinada por la compatibilidad que proporciona Bootstrap 3.1.1, que es el marco de diseño que explota e independiza el interfaz gráfico de las páginas.

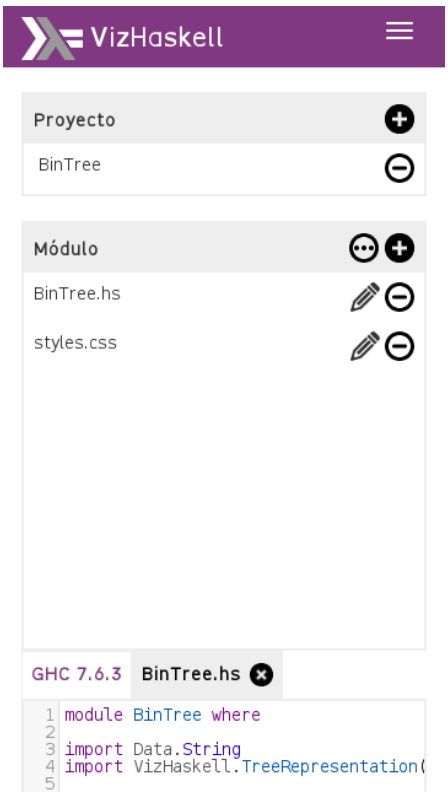

Figura 4.1: Consola vista en un *móvil inteligente*

4.1.2. Modelo de navegación

El sitio utiliza la arquitectura de página única de AngularJS (la navegación usa AJAX desde una única página para cargar las diferentes vistas), lo que proporciona un histórico transparente y gestión de rutas centralizado. Sobre esta arquitectura se han elegido tres opciones de navegación:

Barra horizontal superior Acceso a las páginas más importantes en función del perfil del usuario y al botón para cerrar o iniciar la sesión (en función del estado de la misma).

Lista horizontal en el pie Acceso a las páginas informativas de la aplicación. En el caso del mapa del sitio, éste es adaptativo, es decir, que incluye sólo los enlaces a las páginas a las que el usuario tiene acceso en función de su perfil.

Modelo de paneles y solapas En la página más importante de la aplicación, la consola del intérprete de Haskell, la navegación utilizada es la clásica de los IDEs de programación: un panel selector de proyecto

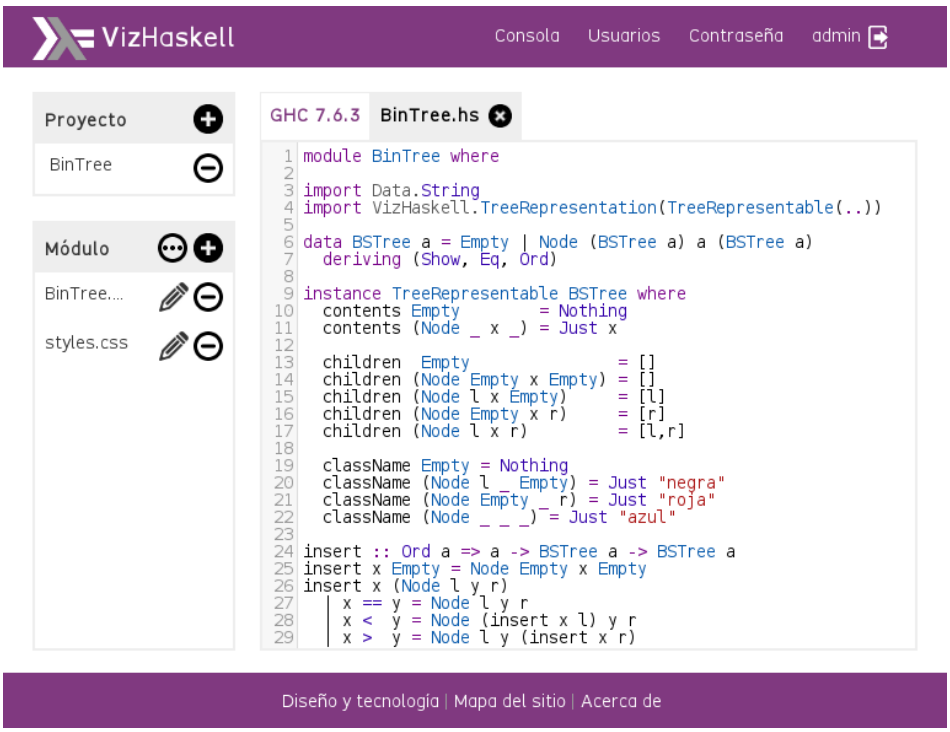

Figura 4.2: Consola vista en una *tableta*

que permite cambiar entre los proyectos del usuario y otro con una lista de módulos para editar. Las solapas de edición permiten cambiar entre módulos abiertos y existe una solapa permanente, la del intérprete de Haskell. Las acciones sobre módulos, proyectos y solapas se realizan mediante botones, para así facilitar la navegación en dispositivos móviles.

4.1.3. Sesión, autenticación y autorización

Se utiliza el correo electrónico como identificador único de usuario para minimizar los detalles necesarios para usar la aplicación. Además, éste es necesario también para enviar una nueva contraseña en caso de que el usuario la restablezca usando la pantalla de *Restablecimiento de contraseña*.

La autenticación exitosa incluye la creación de una sesión válida. Esto comprende no sólo la capacidad de navegar por las páginas autorizadas, sino la adquisición de credenciales para poder leer y escribir en la BD propia del usuario, en donde se almacenarán los proyectos y módulos.

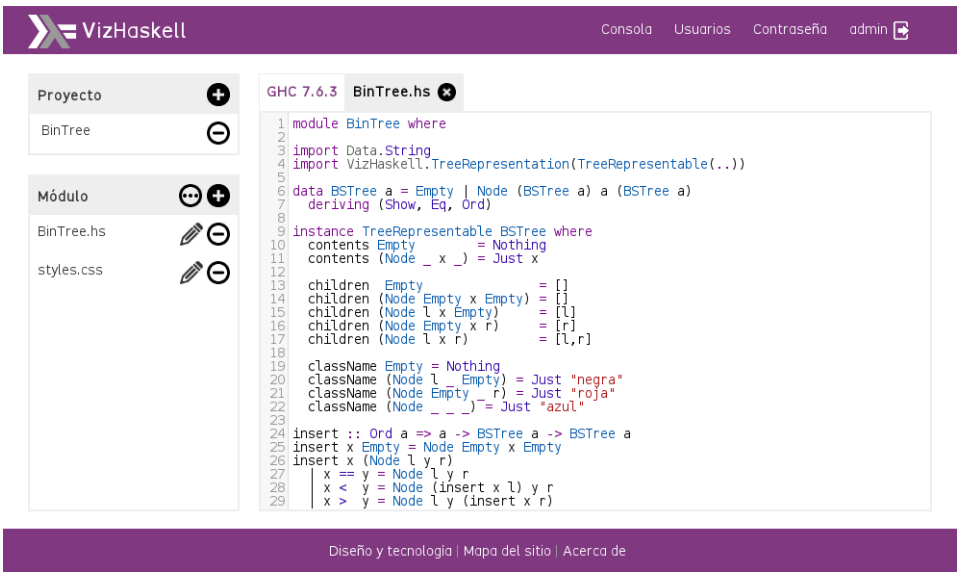

Figura 4.3: Consola vista en un ordenador personal

4.1.4. Roles

A partir de los requisitos se distinguen tres casos de uso diferenciados (roles), con distintos permisos y capacidades:

Usuario El usuario común es un estudiante o interesado en Haskell y, por lo tanto, puede acceder a la consola para escribir expresiones y obtener representaciones gráficas de éstas.

Administrador de aplicación Tiene acceso a toda la funcionalidad del Usuario y, además, tiene la capacidad de modificar los datos de acceso e autenticación de los usuarios (nombre, contraseña y rol) utilizando la pantalla de *Gestión de usuarios*.

Administrador de servidor Tiene acceso a toda la funcionalidad del administrador de la aplicación y, además, puede crear y eliminar usuarios. Esta capacidad está limitada a un único usuario que debe coincidir con el administrador de CouchDB, el único rol con capacidad para la creación de bases de datos.

4.1.5. Casos de uso

A continuación se describen los casos de uso más relevantes que se identifican en la aplicación.

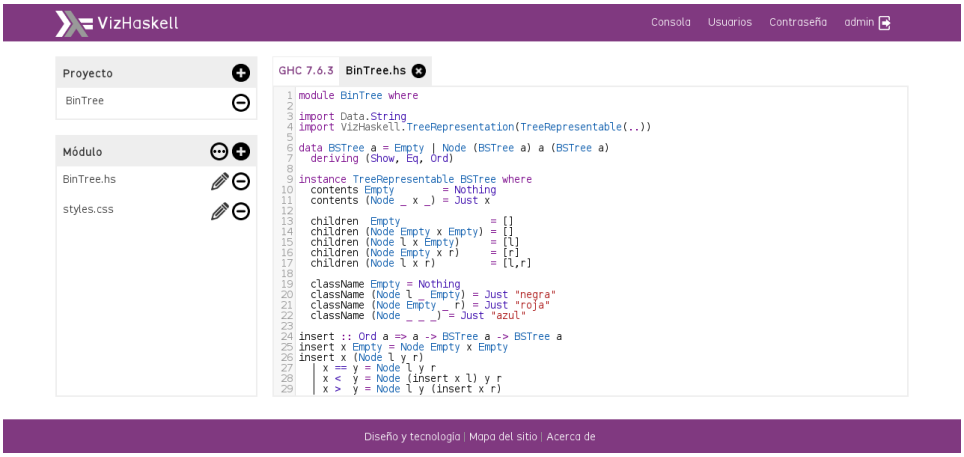

Figura 4.4: Consola vista en un ordenador sobremesa

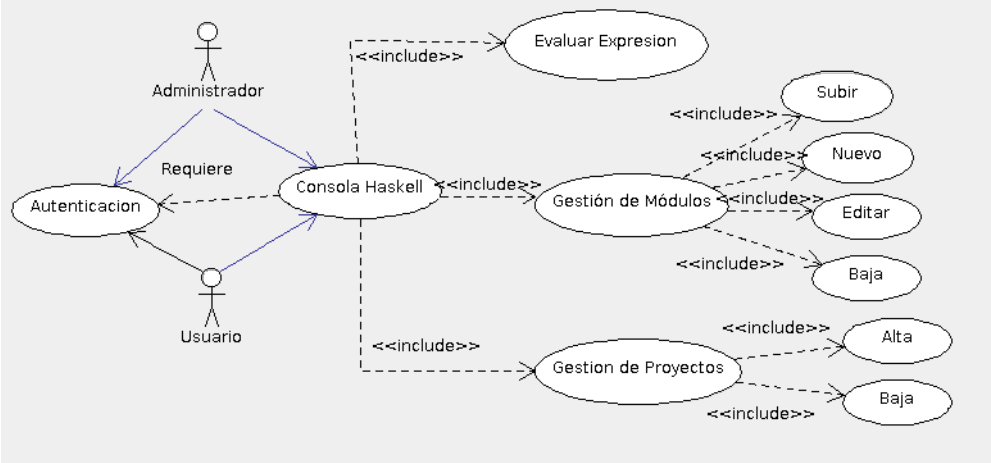

Figura 4.5: Caso de uso de la Consola

4.1.5.1. Consola

La consola permitirá a los usuarios (tanto administradores como usuarios comunes) escribir expresiones Haskell que se evalúan en un intérprete del servidor (figura 4.5). Para utilizarla será necesario que el usuario esté autenticado.

El usuario podrá, además de evaluar una expresión y obtener una representación gráfica automáticamente, subir módulos con tipos de datos e importaciones necesarias, crear nuevos módulos, editar y borrar.

Los módulos se podrán agrupar en proyectos. Cada proyecto tendrá un conjunto de módulos que será cargado por el intérprete cada vez que se evalúe una expresión. Se permitirá crear nuevos proyectos, cambiar entre

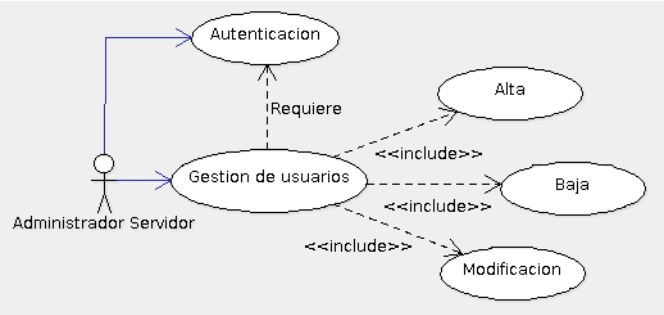

Figura 4.6: Caso de uso de gestión de usuarios - Administrador de servidor

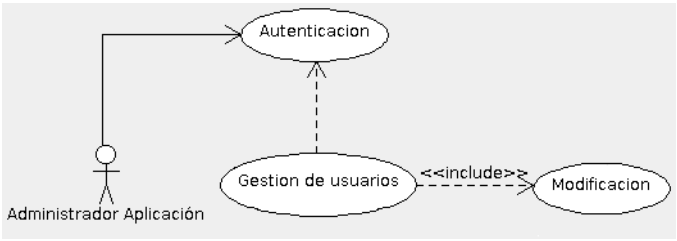

Figura 4.7: Caso de uso de gestión de usuarios - Administrador de aplicación

distintos proyectos y eliminar proyectos existentes.

4.1.5.2. Gestión usuarios

El administrador de servidor (figura 4.6) tiene la capacidad de dar de alta usuarios, eliminarlos o modificar los detalles de su registro. El administrador de la aplicación (figura 4.7) únicamente podrá modificar los detalles del registro de usuarios.

Será requerido, por tanto, una autenticación previa, ya que existe un control de acceso en función del perfil del usuario en esta página.

4.1.5.3. Cambio contraseña

Previa autenticación, el usuario podrá editar los detalles de su cuenta. En concreto, podrá cambiar su contraseña (figura 4.8). No se le solicitará contraseña anterior, ya que deberá estar autenticado, pero sí que confirme escribiendo dos veces la contraseña para evitar errores al teclearla.

Si las contraseñas introducidas coinciden y cumplen unos criterios mí-

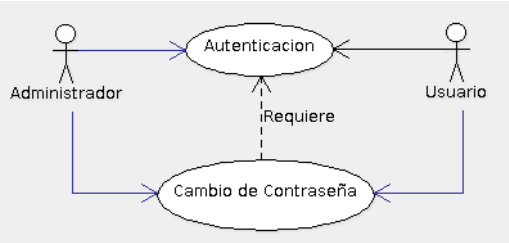

Figura 4.8: Caso de uso del cambio de contraseña

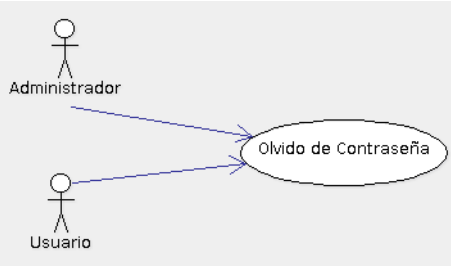

Figura 4.9: Caso de uso de restablecimiento de contraseña

nimos de calidad (longitud, clases de caracteres, etc.), se procederá a actualizar la contraseña del usuario en el repositorio. En caso contrario se mostrará un mensaje de error y se solicitarán de nuevo ambas contraseñas.

4.1.5.4. Restablecimiento contraseña

Cualquier usuario puede solicitar que el sistema restablezca su contraseña. Para ello se le solicitará, además de su dirección de correo, el nombre completo con el que se registró, para evitar así que otra persona pueda solicitar el restablecimiento suplantando su identidad. Es muy importante, por tanto, que el usuario utilice y recuerde el nombre completo con que se registró (figura 4.9).

Si la dirección de correo electrónico existe en el repositorio de usuarios, el sistema enviará un correo electrónico a dicha dirección incluyendo una contraseña alfanumérica generada aleatoriamente, que debe cambiarse en el siguiente acceso a la aplicación. Si la dirección de correo no corresponde con ningún usuario conocido, o si el nombre completo no coincide con el del usuario cuya dirección de correo se especificó, se mostrará un mensaje de error indicando un error en los datos.

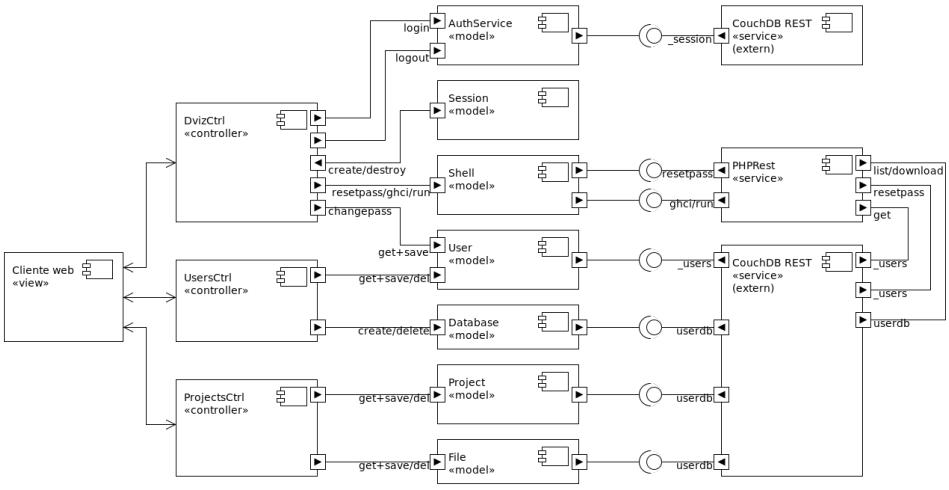

Figura 4.10: Diagrama de componentes UML de la aplicación

4.2. Componentes de la solución

4.2.1. Componentes propios de la solución

En la figura 4.10 se muestra el diagrama de componentes asociado a la aplicación que detallamos a continuación.

4.2.1.1. Controlador principal de la aplicación

Este controlador, `DvizCtrl` cuyo fichero fuente se encuentra en `app/control/dviz.js`, es el encargado de responder a las peticiones de inicio de sesión, cierre de sesión, cambio de contraseña y restablecimiento de contraseña.

Depende de los siguientes módulos o componentes para su correcto funcionamiento: `AuthService`, `Session`, `User` y `Shell`. La descripción detallada de los métodos de este controlador se muestran en la sección B.1.

4.2.1.2. Controlador de la gestión de usuarios

El controlador `UsersCtrl` (`app/control/users.js`) es el encargado de gestionar el alta, baja y modificación de usuarios de la aplicación.

Requiere los siguientes módulos para su correcto funcionamiento: `User`, `Database` y el módulo de AngularJS `xeditable`.

La descripción detallada de los métodos de este controlador se muestran en la sección B.2.

4.2.1.3. Controlador de la gestión de proyectos

El controlador `ProjectsCtrl` se encarga de la gestión de proyectos y módulos creados por el usuario. La definición de este controlador se encuentra en el fichero `app/control/projects.js`.

Para su correcto funcionamiento requiere de los siguiente módulos: `Shell`, `Project`, `File` , `Session` y el módulo de AngularJS `file-uploader`.

Este controlador trabaja sobre tres importantes estructuras de datos, que son los modelos sobre los que se apoya la vista para el correcto funcionamiento de la consola de la aplicación:

- `projects`: almacena la información sobre los proyectos recuperados de la base de datos así como los ficheros fuente asociados a cada uno de ellos.
- `tabFiles`: almacena la información sobre todos los ficheros que se desea mostrar para su edición en la vista de consola. Además, mantiene los cambios realizados sobre el código asociado a los ficheros.
- `projectSelected`:, apunta al proyecto actual sobre el que se está trabajando en la vista. Todas las modificaciones realizadas sobre esta estructura son automáticamente reflejadas en la estructura de datos `projects`, ya que ambas están vinculadas en el modelo.
- `uploader`: es el objeto encargado de realizar todas las operaciones relacionadas con la subida de ficheros a los proyectos.

Con el fin de controlar el estado de los elementos que participan en la consola se ha definido una serie de elementos de supervisión (`watch`), encargados de:

- Comprobar si se han realizado cambios en uno o varios ficheros que han sido editados. Para ello se comprueba la propiedad `modified` de la estructura `tabFiles`. Si hay cambios sin guardar se avisa al usuario con un mensaje, pidiéndole confirmación para abandonar la página y avisando de la pérdida de los cambios pendientes.

- Cuando se edita un nuevo fichero, lo que implica añadir un elemento a la estructura `tabFiles`, se redimensiona el elemento `list-files` de la vista si el tamaño de las solapas (`tab-files`) aumenta.
- Supervisar los posibles cambios en el objeto `Session` para así recuperar el nombre de la base de datos del usuario cuando esté disponible.

La descripción detallada de los métodos de este controlador se muestran en la sección B.3.

4.2.1.4. Servicios de autenticación/autorización

Para la gestión de los procesos de autenticación y autorización se han desarrollado dos componentes que forman parte del módulo `dvizServices`: `AuthService` y `Session`, definidos en el fichero `app/model/services.js`.

AuthService Proporciona los métodos necesarios para realizar las operaciones de autenticación, desconexión y chequeo de sesión. Solo dependen del servicio `$http` de AngularJS. La descripción detallada de sus métodos se muestra en la sección B.4.

Session Este servicio es compartido por casi todos los componentes de la aplicación. Es el encargado de mantener la información asociada con la sesión del usuario conectado. La descripción detallada de sus métodos se incluye en la sección B.5.

4.2.1.5. Servicios de recursos de la base de datos

En el módulo `dvizModels` se encuentran definidos todos los recursos que nos permiten interaccionar con todos los servicios de datos a través de servicios REST. Depende del módulo `ngResource` de AngularJS y está encargado del soporte de interacción con los servicios REST vía `$resource`[1]. Estos recursos se encuentran definidos en el fichero `app/model/models.js` y son los siguientes:

[1]Un servicio (`factory`) que nos permite interaccionar con las fuentes de datos a través de servicios REST

- User: modelo para la gestión de usuarios
- Database: modelo para la creación y eliminación de bases de datos de usuario.
- Project: modelo para la gestión de proyectos.
- File: modelo de gestión de módulos asociados a proyectos.
- Shell: modelo de interfaz con el servicio *PHPRest*

Para una descripción detallada de los recursos, vea la sección B.6.

4.2.1.6. Interceptores de peticiones HTTP

AngularJS proporciona la posibilidad de crear servicios que permiten interceptar tanto las peticiones como las respuestas HTTP para realizar un pre o post procesamiento. Este interceptor se encuentra definido en el fichero `app/main/interceptors.js`.

Hemos creado el servicio `sessionExpired` que nos permite comprobar si existe una sesión válida antes de realizar cualquier operación sobre el servidor. Para ello invocamos el método `AuthService.check` que comprueba si la sesión sigue activa o ha caducado. En caso de que la sesión caduque, se informa al usuario de ello y se le redirige a la página de inicio de sesión. En caso de que la sesión sea válida, se verifica si el perfil del usuario tiene o no autorización para ver la página solicitada. Si el usuario autenticado tiene el perfil de administrador podrá acceder a la pantalla de gestión de usuarios, en caso contrario se le denegará el acceso.

Una vez definido el servicio, éste se registra añadiéndolo a la lista de interceptores del módulo `$httpProvider`:

```
// app es el modulo principal de la aplicacion
app.config(["$httpProvider",
  function($httpProvider) {
    $httpProvider.interceptors.push('sessionExpired');
  }
]);
```

4.2.1.7. Servicio web PHPRest

Este servicio se encarga de atender a las peticiones de restablecimiento de contraseña y la ejecución de las expresiones Haskell en el intérprete GHCi. El fichero `php/phprest.ini` contiene todas las variables de configuración necesarias para la correcta ejecución de los servicios proporcionados por la clase `PHPRest` definida en el fichero `php/server.php`.

Dado que el servicio de restablecimiento de contraseña realiza el envío de la nueva contraseña a través de un correo electrónico, se ha incluido la clase `SMTP` (definida en el fichero `php/class.smtp.php`) de la librería `PHPMailer` (Bointon et al., 2009) licenciada como LGPL2 (*Lesser General Public License Version 2*).

En el fichero de configuración (`phprest.ini`) se encuentran los datos de configuración necesarios para el envío de los correos electrónicos en el restablecimiento de contraseña, las credenciales de un usuario especial en la base de datos que restablece la contraseña de los usuarios, las URL (*Uniform Resource Locator*)s de ejecución del servidor CouchDB y las importaciones de los módulos Haskell necesarios para la evaluación y compilación de las expresiones y módulos ejecutadas desde la consola de aplicación (*prólogos*).

La clase `PHPRest`, definida en el fichero `server.php`, tiene una función principal (`exec`) que es la encargada de realizar la petición de servicio a partir del valor del parámetro `method`. En cada uno de los métodos se valida que los parámetros necesarios sean correctos y antes de realizar la ejecución de la acción solicitada.

La ejecución y validación de una expresión Haskell requiere comprobar si existe sesión válida de usuario. Para ello se obtiene la *cookie* de sesión. Si no está disponible se emite un error de no autorización. Una vez validada la sesión, se procede a bajar todos los ficheros asociados al proyecto a disco. Sólo se descargan aquellos que hayan sido modificados desde la última ejecución, ya que son necesarios para realizar la compilación y ejecución de la expresión. En caso de error, tanto de la compilación como de la ejecución de la expresión, se devuelve un mensaje informando del error. Si la ejecución es correcta, el intérprete puede devolver:

- Un resultado textual que es convertido en un JSON válido que contiene la propiedad `cmdResult` con el resultado de la ejecución.
- Si la respuesta es un JSON y se comprueba que es correcto, se devuelve en un documento nuevo JSON el atributo `cmdGraph` con dicha respuesta. Además, si se ha especificado un fichero CSS (*Cascading Style Sheets*), se añadirá el atributo `csss` con el contenido de dicho fichero.

Además de todas las validaciones anteriores, y dado que que este servicio debe funcionar de forma independiente del cliente, se han repetido validaciones que permiten detectar:

- Si el número y el contenido de los parámetros son válidos.
- Si las órdenes y expresiones ejecutadas en el intérprete están permitidas (configuradas en el fichero `phprest.ini`).
- Si el usuario, además de tener una sesión válida, tiene acceso a los documentos de la base de datos que se indica en la petición.

Si alguna de estas validaciones no se cumple, se rechaza la petición devolviendo una información clara y precisa del error.

4.2.1.8. Configuración de aplicación

La configuración del entorno de ejecución del cliente se realiza en distintos ficheros fuente del directorio `app/main`:

- En `app.js` se configura el marco MVC de AngularJS.
- En `dialog.js` encontramos funciones auxiliares para el manejo de diálogos.
- En `directives.js` se definen las directivas HTML5 a medida.
- En `routes.js` se configuran las distintas rutas de navegación.
- En `console.js` se configura el editor CodeMirror y la consola Wterm.

Para una descripción de los detalles de configuración en cada uno de los anteriores ficheros, refiérase a la sección B.7.

4.2.2. Componentes de terceros

En la sección 2.3 se enumeraron los componentes SW utilizados en el desarrollo de la aplicación. Aquí describimos aquellos componentes que han sido adaptados para una buena integración con la aplicación, de modo que si se realiza una actualización de estos componentes a una versión superior se comprueben si es o no necesario realizar de nuevo dichas modificaciones

Los componentes de terceros que han sido modificados se puede encontrar en el directorio `app/extmod`.

4.2.2.1. CodeMirror

La versión incorporada al proyecto es la última versión estable 4.2. Las modificaciones realizadas se ha realizado para que pase los estándares de accesibilidad.

En la función `Display` al elemento `input` se le ha añadido un elemento *label*, y a los atributos que no son parte del HTML (*HyperText Markup Language*, Lenguaje de marcas de hipertexto) se les ha añadido el prefijo `data-`.

4.2.2.2. XEditable

La versión incorporada al proyecto de este módulo de AngularJS es la versión 0.1.8.

Se le ha hecho una pequeña modificación para que acepte como elemento válido de formulario el objeto HTML `input` de tipo `password`. Para esto, sólo se ha añadido este tipo en la variable `types` de la línea 156 del fichero.

4.2.2.3. File-Uploader

La versión incorporada al proyecto de este módulo de AngularJS es la versión 0.5.6.

Las modificaciones realizadas son las siguientes:

- Se ha añadido un nuevo parámetro de configuración `isUploadForm`

que indica si el fichero se sube como un formulario *multipart* o se lee y se sube sólo su contenido.

- En la función `_xhrTransport` hemos añadido el siguiente código (que utiliza la variable `isUploadForm`):

```
        _xhrTransport: function(item) {
            ...
          if(that.isUploadForm) {
            var form = new FormData();
            item.formData.forEach(function(obj) {
              angular.forEach(obj, function(value, key)
                  {
                form.append(key, value);
              });
            });
            form.append(item.alias, item.file);
            xhr.send(form);
          } else {
            var fr = new FileReader();
            fr.readAsArrayBuffer(item.file);
            fr.onload = function(readerEvent) {
              xhr.send(readerEvent.target.result);
            };
}
```

4.2.2.4. Wterm

La versión incorporada al proyecto de este módulo es la 0.0.4. En esta versión hemos realizado muchas modificaciones para poder adaptarlo a las exigencias de nuestra consola.

- Se han añadido nuevos parámetros de configuración:

```
// DEFAULT_EXEC
// Function to be executed in the case of not
   finding a command register
DEFAULT_EXEC      : null,
// CMD_CLEAR
// Command name of clear
```

```
  CMD_CLEAR          : 'clear',
  // FILTERS
  // List of filters that must be validated before
     executing a command
  FILTERS            : []
```

- En la función `hide` se ha añadido la deshabilitación del elemento `input` para que la petición no pueda ser enviada más de una vez.
- En la función `show` se envia el foco al elemento `input`.
- Se ha añadido una nueva función que permite restablecer el área de entrada de expresiones de la consola:

```
var reset = function() {
  input.removeProp('disabled');
  input.val( '' );
  input.focus();
  var parent = $(element).parent();
  if(parent !== null || parent !== undefined){
    $(parent).scrollTop($(parent)[0].scrollHeight);
  }else{
    $(element).scrollTop($(element)[0].scrollHeight);
  }
};
extern.reset = reset;
```

- Se ha cambiado por completo la función que muestra en consola el resultado de la ejecución de las órdenes:

```
var update_content = function( p, cmd, data ) {
  content.append( '<div><span class="' + settings.
     THEME_CLASS_PREFIX
    + '-prompt-content">'
    + (p != null && p.trim() != ''? p: settings.PS1)
       + ' </span>'
    + '<span>' + cmd + '</span>'
    + '<div>' + ( ( data ) ? data : '' ) + '</div></
       div>' );
  reset();
```

```
};
extern.update_content = update_content;
```

- Se ha añadido a la función `clear_content` la llamada a la función `reset`.
- En la función `submit` se han realizado las siguientes modificaciones: (1) se ha añadido la posibilidad de validar filtros que puedan impedir la ejecución de una orden y (2) en el caso de que no haya una orden a ejecutar y el parámetro `DEFAULT_EXEC` se haya definido una función, esta será ejecutada por defecto, en caso contrario aparecerá un mensaje avisando de que la orden no está permitida.

4.2.2.5. Tree

A partir del trabajo de Schmuecker (2013), hemos realizado una adaptación para la representación de las estructuras de datos que implementen el contrato JSON para el tipo de representación árbol (`repType="tree"`). En concreto, hemos añadido soporte para la representación de árboles en árboles, etiqueta textual dentro de los nodos y generación del atributo de clase para cumplir el *contrato CSS*. Hemos eliminado el soporte de *arrastrar y soltar* y la ordenación automática.

El código final se incluye en la sección A.2.1, comentado para su mejor lectura y revisión.

4.2.3. Correspondencia entre vista y controlador

En las secciones anteriores hemos descrito los componentes SW de la aplicación. En la descripción de cada uno de los controladores hemos establecido las correspondencias de los controladores (capa de control) con los recursos (capa de modelo) definiendo sus dependencias. Además en la figura 4.10 se muestra la relación entre estas dos capas: control y modelo.

Ahora nos centraremos en establecer la correspondencia entre las capas de control y vista. El controlador `DvizCtrl` (ver 4.2.1.1) afecta a todas las pantallas de la aplicación, el controlador `UsersCtrl` (ver 4.2.1.2) afecta a la página de usuarios (`app/view/users`) y el controlador `ProjectsCtrl`

Figura 4.11: Cabecera de aplicación sin sesión de usuario

Figura 4.12: Cabecera de aplicación con sesión de usuario

(ver 4.2.1.3) afecta a la pantalla de consola y gestión de proyectos (`app/view/console.html`).

Las páginas *Acerca de* (`app/view/about.html`) y *Diseño y tecnología* a `pp/view/design.html` son páginas estáticas de información y no se ven afectadas por ningún controlador. Por tanto, omitiremos estas dos vistas.

4.2.3.1. Cabecera de aplicación

En la cabecera de la aplicación mostramos los menús necesarios para la navegabilidad por la aplicación. Estas opciones de menú están afectadas por el controlador principal `DvizCtrl`. Los menús dependen del estado del componente `Session`:

- En la figura 4.11, el menú *acceder* aparece cuando no hay sesión de usuario (`!Session.isAuthenticated()`) y la página de *inicio de sesión* no es visible.
- En la figura 4.12, los menús *Consola* y *Contraseña* aparecen cuando hay una sesión de usuario válida (`Session.isAuthenticated()`). El menú *Usuarios* solo está visible cuando haya una sesión válida y el usuario conectado es un administrador. El último menú aparece el nombre del usuario (`Session.getUsername()`), y es el encargado de realizar la operación de desconexión llamando al método `logout` del controlador.

4.2.3.2. Inicio de sesión

Esta página utiliza el controlador `DvizCtrl`. Cuando es llamada, el menú *acceder* se oculta. Los objetos afectados por el controlador son, como

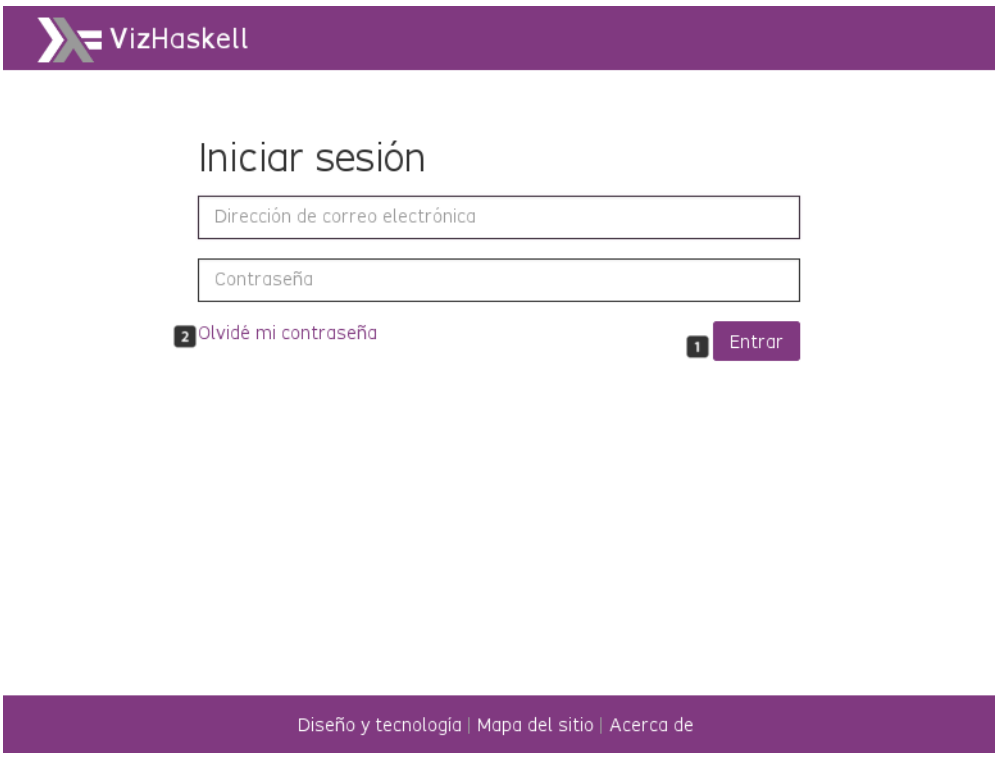

Figura 4.13: Inicio de sesión: correspondencias con el controlador

se indica en la figura 4.13:

1. El botón *Entrar*, que ejecuta el método `login` del controlador enviando el formulario con el correo electrónico y la contraseña.
2. El enlace de restablecimiento de contraseña, que lleva a la página de *Restablecimiento de contraseña* a través de la ruta `#/forgot`, que es interceptada por el módulo `$routeProvider`.

4.2.3.3. Restablecimiento de contraseña

Esta página usa el controlador `DvizCtrl`. Los objetos afectados por el controlador son, de acuerdo a su numeración en la figura 4.14:

1. El botón *Enviar*, que ejecuta el método `resetpass` del controlador enviándole el formulario con el correo electrónico y el nombre completo del usuario.
2. El enlace a la página de *Inicio de sesión*, a través de la ruta `#/login` que es interceptado por el módulo `$routeProvider`.

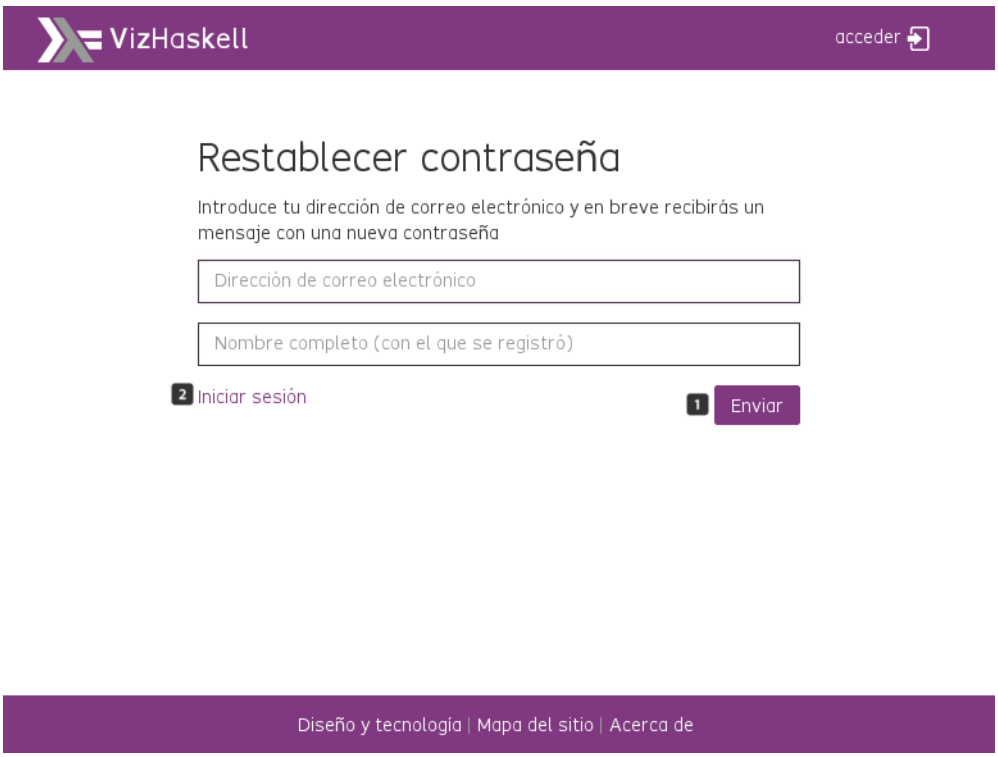

Figura 4.14: Restablecimiento de contraseña: correspondencias con el controlador

4.2.3.4. Cambio de contraseña

Esta página usa el controlador `DvizCtrl`. Tal y como se indica en la figura 4.15, el botón *Cambiar* ejecuta el método `changepass` pasándole el formulario con las dos contraseñas.

4.2.3.5. Gestión de usuarios

Esta página usa el controlador `UsersCtrl`. Cuando se accede a la página desde el menú o desde la página de *Mapa del sitio*, se ejecuta el método `init` del controlador que consulta en la BD los documentos de usuario. Los datos obtenidos son almacenados en el objeto `users`, que es el objeto mostrado la lista en pantalla. Según la numeración de la figura 4.16 los objetos que la componen son:

1. El botón de alta de usuario, que ejecuta el método del controlador `addUser` y que muestra el formulario que se identifica con el número 6.
2. El botón de cancelar, que llama al método `cancelUser`. En caso de al-

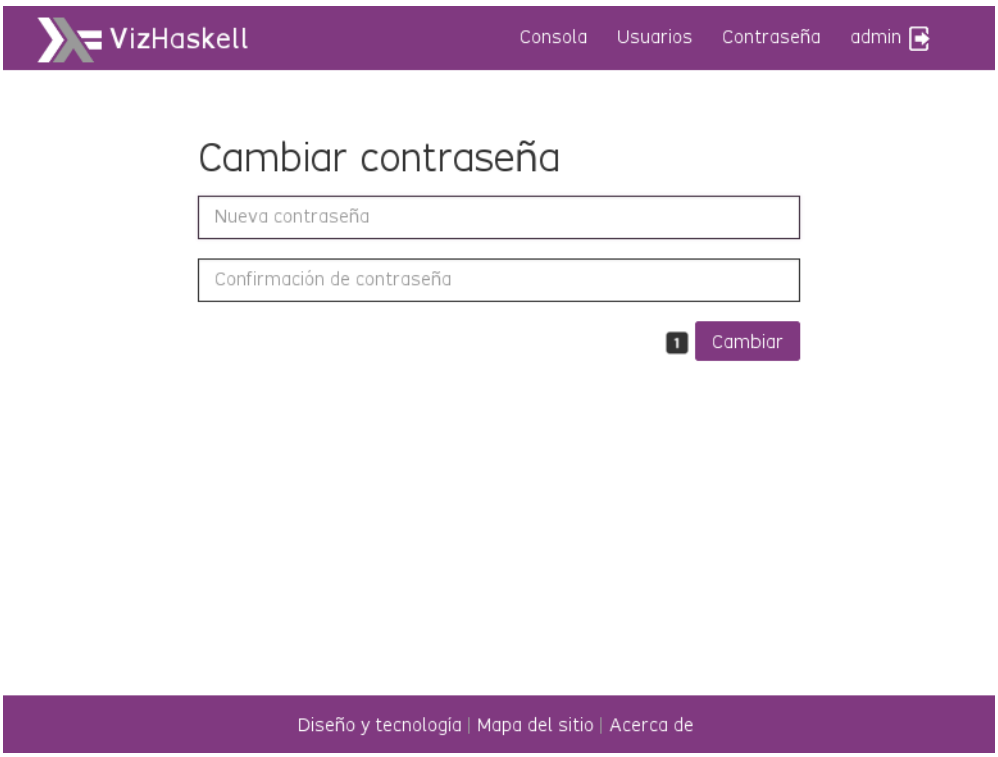

Figura 4.15: Cambio de contraseña: correspondencias con el controlador

ta elimina la nueva fila creada en la tabla y en caso de edición de un usuario oculta el formulario.

3. El botón eliminar un usuario, que llama al método `deleteUser` para borrar el usuario.

4. El botón edición de usuario, que llama al método `editUser` y muestra el formulario identificado con el número 7.

5. El botón de alta o modificación de usuario, que llama a los métodos `saveUser`, en el caso de edición de un usuario, o a `createUser`, en el caso de creación de un nuevo usuario.

Los botones 1 y 3 sólo estarán visibles para el usuario administrador de servidor (`Session.isRoot()`).

Los métodos `checkName`, `checkEmail` y `checkPassword` sólo se ejecutarán cuando estemos en modo edición (7) o en un alta de usuario (6). El método `showPassword` se ejecuta siempre que se muestre un elemento en la lista de usuarios.

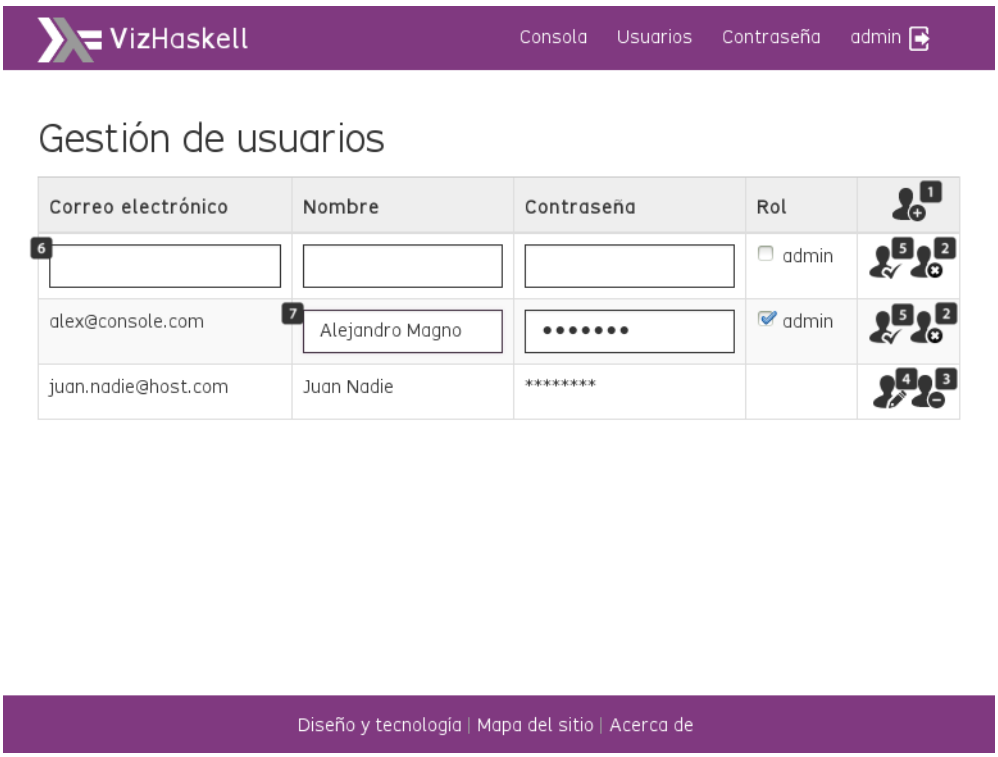

Figura 4.16: Gestión de usuarios: correspondencias con el controlador

4.2.3.6. Consola

Esta página usa el controlador `ProjectsCtrl`. Cuando se accede a la página desde el menú o bien desde la página del *Mapa del sitio*, se ejecuta el método `init` del controlador, que consulta en la BD y obtiene todos los documentos (proyectos) del usuario que son almacenados en el objeto `projects`.

De acuerdo a la numeración de la figura 4.17, sus componentes son:

1. La lista de proyectos, que se carga con el contenido de la estructura `projects`. Cuando se realiza un cambio de proyecto (`changeProject()`) se actualiza la variable `projectSelected` que es el modelo sobre el que se carga la lista de módulos. En la figura, los tres primeros elementos de la lista.
2. El botón de añadir un proyecto, que llama a una pantalla modal que permite introducir el nombre del proyecto y llamar al método `addProject()`.
3. El boton de eliminar un proyecto, que llama al método `deleteProjec`

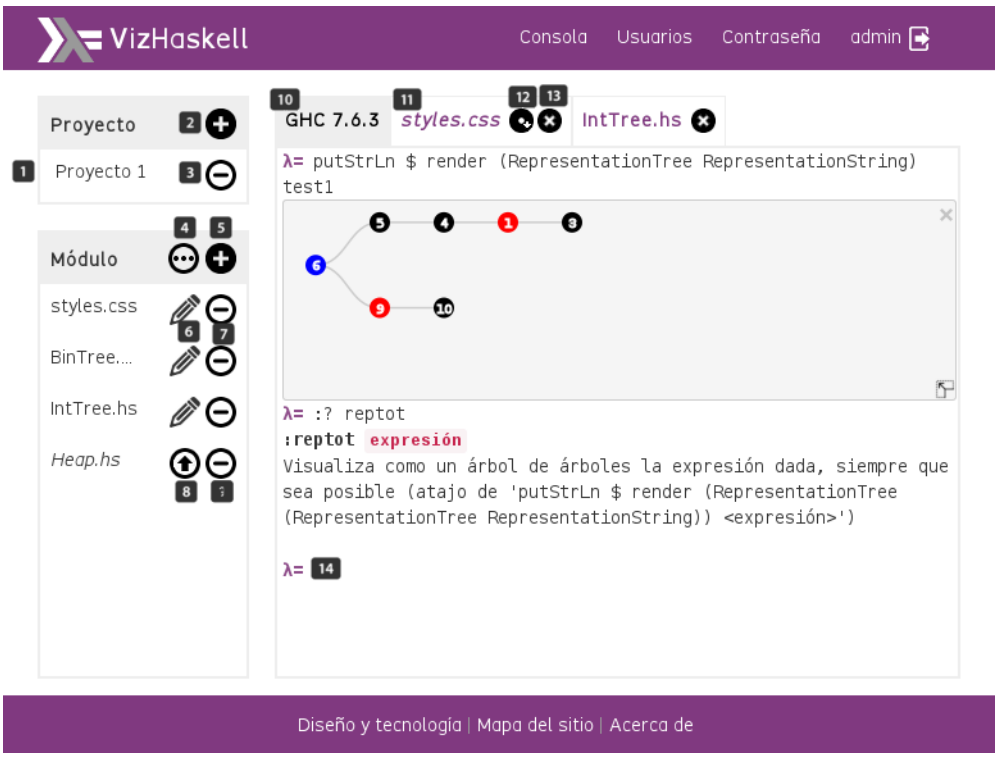

Figura 4.17: Consola: correspondencias con el controlador

t.

4. El botón de subir módulos, que permite abrir una pantalla para seleccionar los ficheros que se quieren subir. En el momento que se aceptan los ficheros a subir se ejecuta el método `filters` del objeto `uploader` y aquellos ficheros que superan el filtro son añadidos a la lista de ficheros `uploader.queue`.

5. El botón de añadir un nuevo módulo, que llama a una pantalla modal que permite introducir el nombre del módulo y llamar al método `newFile`.

6. El botón de editar módulo, que permite editar un módulo llamando al método `openFile`, lo que actualiza la estructura `tabFiles` que controla las solapas de los ficheros (número 11).

7. El botón de eliminar módulo, que permite eliminar un módulo de la BD llamando al método `deleteFile`.

8. Este botón de subir módulo, que permite la subida del fichero a la BD

ejecutando el método `item.uploader()` del objeto `uploader`. Este método añade el fichero a la lista de ficheros de la variable `projectSelected` lo que causa que automáticamente se añada a la lista de ficheros.

9. Este botón realiza la cancelación de la subida del fichero llamando al método `removeFileUploader`.
10. Esta solapa muestra la pantalla de consola y tiene asociado el método `activateTabConsole`.
11. Las solapas asociadas a los ficheros son controladas por la estructura `tabFiles`. Dado que usan un modelo, cualquier cambio en la lista de ficheros causa un cambio en las solapas mostradas. Asociado al nombre del fichero hay un método que permite mostrar el editor con el contenido de ese fichero (`activateTab`).
12. Este botón permite guardar el contenido del fichero ejecutando el método `saveFile`.
13. Este botón permite cerrar la solapa asociada al fichero ejecutando el método `closeTab`.
14. Cuando se introduce una expresión en la consola y se pulsa *enter* se llama al método `sendCommand`.

4.2.4. Estructura de la base de datos

Como gestor de BD se ha elegido CouchDB, una BD no-SQL orientada a documentos.

Cada usuario tiene una base de datos propia y, su base de datos, puede almacenar un conjunto de proyectos (documentos de CouchDB) que contienen, a su vez una lista de módulos (adjuntos al documento).

Cuando el administrador de servidor da de alta un usuario en la aplicación, genera un nuevo documento en la base de datos de usuarios (`_users`) y crea la base de datos del al usuario. Los datos almacenados de cada usuario son: un identificador que corresponde con el correo electrónico del usuario, el nombre completo del usuario, el perfil y la contraseña encriptada.

Al crear una base de datos por usuario nos aseguramos que ningún otro usuario pueda acceder a una BD que no sea la suya añadiendo el identificador del usuario al atributo `members` del documento de seguridad de su base de datos (`_security`), con lo que garantizamos la seguridad e integridad de los datos. Este esquema de seguridad está incorporado en la propia BD CouchDB.

La creación de un nuevo proyecto implica la creación de un nuevo documento en la BD del usuario. La estructura del documento asociada a cada proyecto contiene los siguientes atributos o campos:

- Un identificador único, que se corresponde con el nombre del proyecto.
- Un número de revisión, generado automáticamente por el gestor de BD. Este número de revisión es importante tenerlo actualizado en la aplicación para evitar errores de inconsistencia.
- Una lista de ficheros que forma parte del proyecto. Estos se almacenan en un campo del documento denominado `_attachments`. Cada uno de estos elementos en la lista es un fichero y contienen los siguientes atributos:
  - `content-type`, tipo de fichero.
  - `digest`, el contenido del fichero codificado en MD5.
  - `length`, longitud del fichero.

4.3. Instalación de la solución

4.3.1. Requisitos de la instalación

Para instalar la aplicación Web es necesario tener instalados los siguientes componentes de *software*, programas:

- Apache2 versión 2.4.6 con los módulos `mod_rewrite`, y `mod_proxy`.
- PHP5.

- CouchDB 1.5.0. Si la instalación se realiza sobre una máquina con un SO Ubuntu 12.04 el paquete disponible en el repositorio está en la versión 1.4.0 que no es compatible con nuestra aplicación.
- Erlang 16b, ya que es una dependencia de CouchDB.

4.3.2. Estructura de fuentes

La estructura de directorios aplicación y que es necesario desplegar es la siguiente:

`app` se almacenan todos los ficheros fuente que conforman la parte cliente de la aplicación. Contiene los siguientes directorios:

- **`control`** capa donde se describen los controladores.
- **`model`** capa donde se describen los servicios de autenticación y recursos.
- **`view`** capa donde se definen las página HTML que conforma la aplicación.
- **`css`** los ficheros CSS que definen el aspecto de la aplicación
- **`ext`** los ficheros de los componentes externos.
- **`extmod`** los ficheros de los componentes externos que han sido adaptados a las exigencias de la aplicación
- **`fonts`** el fichero que definen la fuente usada en la vista de la aplicación.
- **`img`** las imágenes usadas en la presentación de la aplicación.
- **`main`** los ficheros que configuran la aplicación: interceptores, funciones comunes, directivas y configuración AngularJS.

`php` ficheros del servidor *PHPRest* encargado de llamar al intérprete con la expresiones Haskell y de realizar el restablecimiento de contraseña y envío de correo electrónico informando del cambio.

`hs` se encuentra los fuentes de la librería Haskell de representación visual de expresiones Haskell.

`bin` se localizan los *shell scripts* de los intérpretes disponibles.

Los directorios `bin` y `hs` no deben ser accesibles, por ello deben incluir un fichero `.htaccess` con el siguiente contenido.

```
<Files "*">
  Require all denied
</Files>
```

4.3.3. GHCi y sus dependencias

El intérprete GHCi requiere la instalación de la plataforma Haskell completa. Dado que el entorno lo hemos montado sobre una maquina virtual con un SO Ubuntu 12.04, debemos realizar los siguientes pasos:

- Instalar la plataforma completa: `apt-get install haskell-platform`
- Crear el directorio `.cabal` en `/var/www` y asignarle como propietario y grupo `www-data`.
- Ejecutar `cabal update` como usuario `www-data`, para ello ejecutar `su - www-data`.
- Instalar `Data.Aeson` ejecutando `cabal install aeson`

4.3.4. Apache Web Server con PHP5

- Configurar las rutas `couchURL` y `baseURL` en `/var/www/php/phprest.ini`
- En la configuración de Apache2, directorio `/var/www/`, quitar la opción `Indexes` (poner menos delante) y cambiar la opción `AllowOverrideNone` por `All` (esto permite tener en cuenta las opciones de configuración de los ficheros `.htaccess`). La opción `FollowSymLinks` puede estar habilitada.
- Habilitar el módulo `rewrite` ejecutando: `a2enmod rewrite`
- Reiniciar el servidor web ejecutando: `service apache2 restart`
- Copiar `couchdb.conf` (proxy) a `/etc/apache2/sites-available`:

```
ProxyPass /db http://localhost:5984
ProxyPassReverse /db http://localhost:5984
ProxyPassReverseCookiePath /db /
```

- Habilitar la configuración realizada ejecutando: `a2ensitecouchdb`
- Habilitar el módulo `proxy_http` con el comando `a2enmodproxy_http`
- Habilitar el módulo proxy ejecutando: `a2enmodproxy`
- Instalar el soporte JSON para Apache ejecutando: `apt-get install php5-json`
- Instalar el soporte PHP (si es necesario) y `php5-curl` ejecutando: `apt-get install php5-curl`
- Copiar o exportar (con `svnexport` los fuentes de la aplicación desde el repositorio SVN[1].

4.3.5. CouchDB 1.5.0

A continuación mostramos la instalación, configuración y personalización de CouchDB 1.5.0 y de las dependencias necesarias de los paquetes usados. Es importante notar que los paquetes no corresponden con el SO instalado si no con el sistema más avanzado Ubuntu 14.04. Esta es la versión más antigua que hemos encontrado que soporta la versión 1.5.0 de CouchDB.

- Copiar `local.ini` a `/etc/couchdb`.
- Modificar los métodos de autenticación en `default.ini` (`authentication_handlers`) y dejar sólo la autenticación por *cookies* y básica.
- Establecer el valor del propiedad `require_valid_user` a `true` en el fichero `default.ini`.
- Modificar el campo `validate_doc_update` de `/_users/_design/_auth` para permitir borrar documentos a usuarios del grupo ádmin':

[1] https://dalila.sip.ucm.es/svn_vizhaskell

```
if ((userCtx.roles.indexOf('_admin') !== -1) ||
+ (userCtx.roles.indexOf('admin') !== -1) ||
```

- Crear el administrador del servidor CouchDB (`admin@vizhaskell.com`) en el fichero `local.ini`. Este usuario será el administrador de la BD y el administrador de servidor de la aplicación.
- Añadir la BD del administrador (`admin$vizhaskell_com`) usando la herramienta Futon[1], poner el nombre nombre `admin@vizhaskell.com` en el array `Names` del atributo `Members` del documento `_security` de esa base de datos.
- Añadir en el documento `_security` de la BD `_users` el rol `admin` a la lista de roles del grupo `Admins` para que los administradores de aplicación puedan acceder a la lista `_all_docs` en la BD del sistema.
- Si se instala sobre Ubuntu 13.10 o inferior, se requerirá compilar e instalar Erlang 16b para Couchdb 1.5.0. Con Ubuntu 13.10 se pueden utilizar los paquetes de Ubuntu 14.04.

4.3.6. El *software*, programas de VizHaskell

Un vez que se hayan instalado todos los componentes de SW necesarios e instalada la aplicación en `/var/www/`, queda un último paso, se ha de instalar la librería VizHaskell.

Para ello hay que acceder al directorio `hs/CoreRepresentacion` y con el usuario (`www-data`) que ejecutará la aplicación, que además debe ser el propietario del directorio `/var/www/.cabal`, debemos instalar la librería ejecutando: `$cd hs/CoreRepresentacion && cabal install`

4.3.7. Esquema de autorización en base de datos

Dado que la aplicación distingue dos tipos de administradores, y como se explicó en el sección 4.3.5, se ha modificado el esquema de seguridad de tal manera que los usuarios administradores de aplicación (`admin`) puedan acceder a la BD de usuarios con el objetivo de que puedan actualizar los datos de los usuarios.

[1] http://localhost/vizhaskell/db/_utils

El usuario admin@vizhaskell.com es el único que puede crear y borrar usuarios, ya que esto implica la creación y borrado de la BD asociada al usuario.

Cuando un usuario es creado por la aplicación, se le asocia una BD privada. Para lograr esto, en el documento de seguridad asociado a la BD se añade el identificador del usuario como miembro. De esta forma, es el único usuario autorizado para gestionarla, a excepción del administrador de CouchdDB. Con este mecanismo de seguridad evitamos que un usuario pueda acceder a la BD de cualquier otro usuario.

4.3.8. Configuración de VizHaskell

Los posibles parámetros de configuración que puede ser configurados en la aplicación se encuentran en el fichero php/phprest.ini:

- Los parámetros relacionados con el envío de correos electrónicos:

```
[smtp]
host = "smtp.host.com"
port = 9999
username = "user@host.com"
password = "xxxxxxxx"
from = "user@host.com"
msgtemplate = "mensaje a enviar"
```

- El directorio de trabajo donde se descargarán los proyectos de los usuarios cuando se ejecute alguna expresión en la consola: workDIR.
- El directorio donde se encuentran los paquetes instalados de GHC: cabalHOME.
- Las credenciales del usuario administrador de BD, distinto al de la aplicación, para realizar el restablecimiento de contraseña: credentials. Este usuario también tendrá que estar configurado en el fichero local.ini de CouchDB.
- La lista de comandos permitidos que pueden ser ejecutados en el intérprete: ghval-cmd.

4.3.9. Licencias y atribución

Componente	Autor/es	Licencia
GHC[3]	Peyton Jones y Simon Marlow	BSD
AngularJS[4]	Google	MIT
CodeMirror[5]	Marijn Haverbeke	MIT
Bootstrap[6]	Twitter	MIT
CouchDB[7]	Apache	*Apache License 2.0*
D3[8]	Michael Bostock, Jeffrey Heer, Vadim Ogievetsky	BSD
JQuery[9]	John Resig	MIT
PHP[10]	Rasmus Lerdorf	*PHP License*
CheckList-model[11]	Vitaliy Potapov	MIT
Angular-xeditable[12]	Vitaliy Potapov	MIT
Wterm[13]	Venkatakrishnan Ganesh	
dndTree.js[14]	Rob Schmuecker	

[3] http://www.haskell.org/ghc/
[4] https://angularjs.org/
[5] http://codemirror.net/
[6] http://getbootstrap.com/
[7] http://couchdb.apache.org/
[8] http://d3js.org/
[9] http://jquery.com/
[10] http://www.php.net/
[11] http://vitalets.github.io/checklist-model/
[12] http://vitalets.github.io/angular-xeditable/
[13] http://wterminal.appspot.com
[14] http://www.robschmuecker.com/d3-js-drag-and-drop-zoomable-tree/

Capítulo 5

Extensibilidad

5.1. Alcance, esfuerzo y capacitación

Entendemos como extensibilidad la capacidad de una aplicación de SW para ser ampliada o modificada en cuanto a la cantidad o naturaleza de la funcionalidad que ofrece. La cualidad de extensible, sin embargo, es algo más difícil de valorar, ya que toda aplicación de SW puede extenderse sin problema en principio, si se dispone del código fuente.

En nuestro caso, clasificaremos los métodos en básicos y avanzados en función del esfuerzo que un programador con la formación necesaria en cada caso tenga que invertir en el desarrollo de cada método descrito.

En cuanto a los límites de la extensibilidad, creemos que no existe un término absoluto que se pueda establecer, si no que depende de la rentabilidad que el programador quiera obtener de su esfuerzo y de los recursos de que disponga para facilitar su tarea. Creemos que una buena documentación, transparente y orientada hacia este objetivo es la mejor forma de facilitar la extensibilidad y, por tanto, hemos invertido en esta tarea más que en la de proporcionar métodos de extensibilidad a medida que, por experiencia, sabemos que se agotan con demasiada facilidad.

5.2. Extensibilidad básica

5.2.1. Añadir artefactos a páginas existentes

Es posible ampliar la funcionalidad de páginas existentes añadiendo elementos o funciones nuevas en éstas. Esta tarea requiere conocimientos de JavaScript (AngularJS y jQuery, sobre todo), HTML5 y CSS3 (si se quieren dar estilos a los elementos de interfaz). Para ello seguiremos el siguiente método:

1. Navegar hasta la página en la que queremos realizar cambios.
2. Localizar la vista (fichero HTML) que corresponda a la página que queremos modificar. Requiere tres pasos:
   a) identificar la *ruta* de AngularJS de la vista actual;
   b) identificar la *plantilla* y el *controlador* (si tiene) que corresponden a la *ruta* identificada en el fichero de rutas `app/main/routes.js`.
   c) identificar el fichero de JavaScript en que reside el *controlador* identificado anteriormente.
3. Añadir los elementos de interfaz en la vista (fichero HTML).
4. Añadir un nuevo modelo, si es necesario.
5. Añadir la funcionalidad en el controlador.
6. Refrescar la página y comprobar si los cambios son correctos.

Para identificar la *ruta* de AngularJS nos fijaremos en la barra de dirección del navegador. La parte de la dirección mostrada que sigue a la almohadilla (símbolo #) es la *ruta* de AngularJS.

En cuando a la *plantilla* y el *controlador* son los dos campos llamados, respectivamente, *templateUrl* y *controller* de la entrada correspondiente a la *ruta* que hemos identificado antes. Si la entrada no tiene el campo *controller*, la funcionalidad a añadir puede incluirse en el controlador `DvizCtrl` (fichero `app/control/dviz.js`), ya que este controlador tiene alcance

en todas las vistas. Si la ruta tiene controlador, se encontrará en los ficheros `app/control/projects.js` o en `app/control/users.js`.

Para añadir elementos de interfaz, se recomienda utilizar las capacidades de AngularJS (si la complejidad de la interacción lo requiere y la página tiene un controlador) utilizando las etiquetas de AngularJS para el uso del modelo subyacente (si es necesario) siempre anteponiendo el prefijo `data-` para que la página pase las validaciones de accesibilidad y usabilidad.

Para añadir un nuevo modelo será necesario añadir una nueva entrada a la lista de modelos (fichero `app/model/models.js`) y, además, inyectar este modelo en la lista de dependencias del controlador (añadir el nombre del modelo en la lista de parámetros del constructor `dvizControllers.controller`).

5.2.2. Añadir un tipo de representación VizHaskell

Como ya se explicó en la sección 3.3, la librería de soporte a la representación de VizHaskell permite instanciar tipos representables a partir de las clases `RepresentationString` y `RepresentationTree`. Además, es posible crear nuevos tipos de representación utilizando como clase base `Representation` u otra de las clases representables de la librería.

Este procedimiento requerirá conocimientos de programación en Haskell, pero también JavaScript (D3), ya que será necesario crear una función que, a partir de los datos de la estructura, sea capaz de añadir el gráfico que los representa. Veamos cómo:

1. Incluir el módulo de representación en el directorio obtenido del repositorio `hs/CoreRepresentacion/VizHaskell`. Se recomienda comentarlo para que después sea posible generar la documentación Haddock automáticamente.

2. Añadir el nombre del módulo a la lista `exposed-modules` del fichero de generación de la librería, `hs/CoreRepresentacion/VizHaskell.cabal`.

3. Compilar e instalar la librería utilizando el mismo usuario bajo el que se ejecuta el servidor web (normalmente `cabal install`).

4. Instalar el fichero JavaScript que contenga la función de representación en la carpeta `app/main` si es un desarrollo propio o en `app/extmod` si es un desarrollo ajeno modificado.

5. Añadir la carga del fichero JavaScript al la lista de dependencias de la vista principal `app/dviz.html`. El orden importa, debería estar debajo de `d3.min.js` si usa D3, por ejemplo.

6. Añadir la llamada a la función de representación en un nuevo *caso* bajo la línea 46 del fichero `app/main/console.js`, eligiendo un nuevo `repType` que coincida con el que se emite desde el nuevo tipo de representación Haskell.

Cabe destacar que el contrato de representación puede ampliarse si es necesario, siempre que se devuelva un documento JSON válido y que se proporcione el `repType` en cada objeto anidado y, sobre todo, en la extructura principal. La función de representación debe, además, cumplir el *contrato CSS* (sección 3.3) si se desea poder personalizar el objeto usando CSS, es decir, emitir el atributo `class` utilizando el campo `className`.

5.2.3. Modificaciones de la consola

La consola es la página más importante y compleja de la aplicación. Sin embargo, existen varios métodos sencillos y rápidos para ampliar la funcionalidad, aunque la flexibilidad y la utilidad de este componente probablemente admitan muchas más mejoras y ampliaciones. Nombraremos algunas como trabajo futuro más adelante, en la sección 6.1.

5.2.3.1. Añadir atajos al intérprete

Con ciertos conocimientos de JavaScript (y jQuery), es posible añadir atajos (órdenes cortas del intérprete que van precedidas de dos puntos y que se sustituyan al ejecutarse por expresiones Haskell más complejas) de forma sencilla. Esta tarea requiere editar el fichero `app/main/console.js`:

1. Añadir el atajo (elija un nombre corto, expresivo y sin los dos puntos del prefijo) a la lista de órdenes de la línea 94.

2. Añadir un elemento más al array de objetos `console_commands`, por ejemplo debajo de la línea 176. Este elemento debe nombrarse igual que el atajo (esta vez con el prefijo de dos puntos) y tendrá como valor una función anónima que recibe el parámetro `token`. Este parámetro es un array que puede modificarse y que contiene la lista de palabras que forman la expresión completa. Eliminar el primer elemento (que es el nombre del atajo) y añadir entonces la lista de sustitución es el método más común (vea órdenes parecidas como `:repraw`, etc.). Finalmente se llamará a `ghcConsole.ajaxCommand` para enviar la lista de palabras modificada para su ejecución.

3. Añadir la documentación completa de la orden a la función `helpConsole` (por ejemplo, bajo la línea 280). Habrá que añadir un caso con el nombre del atajo (sin prefijo) en el que se asigna una estructura `<dl>` con el texto de la ayuda a la variable `helpText`.

4. Añadir la documentación corta del atajo a la variable `helpText` de la línea 295 (caso `default`), por ejemplo bajo la línea 323. El texto debería tener un `<dt>` y un `<dd>` de forma similar al resto de líneas con una explicación de lo que hace la orden.

Se debería verificar que la sustitución es correcta y que las ayudas (tanto la completa, `:h<atajo>`, como la corta, `:h`) son correctas después de refrescar la página de la consola.

5.2.3.2. Añadir una orden nueva al intérprete

Añadir una nueva orden al intérprete es una operación muy parecida a añadir un atajo y requiere los mismos conocimientos para realizarla.

La única diferencia es que en el segundo paso del procedimiento no se modificará la variable `token` ni se llamará a `ghcConsole.ajaxCommand`, si no que se realizará la tarea que se desee al llamar a esta orden. La función anónima deberá devolver obligatoriamente una cadena de texto que se mostrará bajo la orden y que representa la salida de ésta.

Si la orden realiza alguna tarea larga, es muy conveniente mostrar algún tipo de realimentación al usuario. Por ejemplo, puede mostrarse el indicador de espera (animación circular en la esquina superior derecha)

hasta que la tarea se complete. Para ello basta con ejecutar la orden `$("#tab-files").addClass("working")` antes de comenzar la tarea para que la animación aparezca y `$("#tab-files").removeClass("working")` al concluirla para que la animación desaparezca.

5.2.3.3. Añadir o eliminar una orden filtrada del intérprete

El intérprete de la consola filtra muchas de las órdenes de GHCi que se han considerado innecesarias o poco útiles. En ocasiones puede ser deseable restablecer alguna de esas órdenes filtradas. Este cambio es muy sencillo y sólo requiere dos pasos:

1. Añadir la orden (sin el prefijo) a la lista de órdenes admitidas por el filtro del código JavaScript (variable `cmds`) en la línea 94 del fichero `app/main/console.js`.
2. Añadir la orden (con el prefijo) a la lista de órdenes admitidas por el filtro en el servicio *Shell* del componente *PHPRest*: se deberá editar el fichero `php/phprest.ini` y añadir bajo la línea 16 una entrada similar a `gheval-cmds[]=":<orden>"`.

5.2.4. Añadir un nuevo intérprete

El intérprete de Haskell que utiliza la consola por defecto es GHCi. Es posible, sin embargo, cambiar este intérprete por otro compatible siguiendo unos sencillos pasos. Para ello será necesario editar un fichero fuente JavaScript, crear un *shell script* que actuará a modo de adaptador y configurar el servicio *PHPRest* (editando su fichero de configuración INI) para que lo utilice.

1. Editar el fichero `app/main/console.js` y realizar los siguientes cambios:
   a) Cambiar el nombre del intérprete por defecto en la línea 70: asignar a la variable `ghcConsole.shell` un nombre de intérprete que se usará más adelante.
   b) Cambiar la lista de órdenes admitidas en la línea 94 (variable `cmds`). Las órdenes admitidas dependen del intérprete y, por tanto, hay

que establecerlas aquí.

c) Cambiar los textos de ayuda de las órdenes en la función `helpConsole`: debería haber un caso con el nombre de cada orden que estableciese la variable `helpText` a un `<dl>` con la información de ayuda.

d) Reescribir el caso `default` de la función `helpConsole`, ya que tiene que contener una ayuda rápida de las órdenes admitidas.

2. Crear un *shell script* (similar a los ya existentes `gheval` y `ghrun`) que recibirá como primer parámetro el directorio de proyecto (el *script* deberá cambiarse a ese directorio antes de ejecutar el intérprete) y en el resto de los parámetros recibirá la orden de Haskell. Este *script* es el encargado de invocar al intérprete y devolver, por la salida estándar, la respuesta textual o el documento JSON válido que el servicio devolverá al navegador. Éste, a su vez, enviará el documento a la función de representación correspondiente al tipo de representación que contenga (`repType`). El *script* debe ser ejecutable, por supuesto.

3. Editar el fichero de configuración del servicio *PHPRest* (`phprest.ini`):

   a) Añadir una clave con el nombre de la orden que se eligió anteriormente en la sección `interpreters`. El valor de la clave será la ruta absoluta del *shell script* que ejecutará el intérprete y que recibirá el directorio del proyecto y la orden Haskell como parámetros.

   b) Añadir las órdenes admitidas (con prefijo): para ello habrá que dar como clave `<orden>-cmds[]` y como valor cada una de las órdenes admitidas con prefijo, una orden por línea, repitiendo la clave. El efecto es que las órdenes se añadirán a un array con el nombre `<orden>-cmds` y así se admitirán todas las órdenes del array.

   c) Si se desea incluir importaciones específicas (distintas de las comunes) o se requiere un prólogo (órdenes previas en todas las ejecuciones del intérprete), puede añadirse en la sección `prologs`, añadiendo una clave con el mismo nombre de orden y el prólogo como valor. Los caracteres de retorno de carro y nueva línea se incluirán también si se entrecomilla el prólogo.

Una vez realizados estos cambios bastará con refrescar la página (no es necesario reiniciar el servidor web) para comprobar si el nuevo intérprete se ha configurado con éxito.

5.2.5. Personalización de la visualización

La personalización de los estilos visuales requiere de conocimientos suficientes de CSS2 y, en concreto, de los estilos que aplican a los gráficos SVG[1].

Se dispone de toda la potencia de CSS para modificar los objetos gráficos de SVG de cada gráfico generado pero, por seguridad, los estilos son enviados a un *div* con el atributo `scoped="true"`, de modo que sólo aplican al *div* en el que el gráfico está incluido.

Dado que algunos atributos de los objetos no pueden ser manipulados usando CSS, la única alternativa es utilizar transformaciones visuales[2] en aquellos navegadores que lo soporten. Unidas a selectores[3] avanzados[4], la flexibilidad para modificar el gráfico final es casi total.

5.3. Extensibilidad avanzada

5.3.1. Extensibilidad de componentes de terceros

Otra de las ventajas de la estructura modular y de apoyarnos en componentes de terceros de gran aceptación es que la extensibilidad de la aplicación se apoya también en la extensibilidad de sus componentes externos. Tres de los componentes externos más importantes proveen mecanismos de extensibilidad que podemos aprovechar para ampliar las capacidades de la aplicación.

[1] http://www.w3.org/TR/SVG/styling.html#StylingWithCSS
[2] https://developer.mozilla.org/en-US/docs/Web/CSS/transform
[3] http://www.w3schools.com/cssref/css_selectors.asp
[4] http://www.w3.org/TR/2008/REC-CSS2-20080411/selector.html

5.3.1.1. jQuery

Esta librería dispone de extensibilidad basada en *plugins*. Existe un registro oficial de *plugins*[5] en el que cientos de desarrolladores publican tanto el código fuente como documentación y ejemplos de uso.

Para incluir un *plugin* será necesario tener conocimientos de JavaScript y, por supuesto, de jQuery, si bien la instalación de plugins suele ser muy sencilla. Bastará con incluir el enlace al fichero que incluye el plugin en la página `app/dviz.html`. El módulo debería dejarse en el directorio `app/ext` si no se modifica y en `app/extmod` si se requiere algún ajuste después de su instalación.

5.3.1.2. AngularJS

La arquitectura de AngularJS se basa en el concepto de *módulos*. Tanto el código propio de la aplicación como los componentes que amplían la funcionalidad de AngularJS son módulos[6], y la instalación y uso de estos módulos se rige por el concepto de *inyección de dependencias*: para usar un módulo (o la funcionalidad que provee) en otro debemos *inyectarle* aquel como una dependencia nueva.

Por ello es necesaria cierta fluidez en el manejo de la arquitectura, conceptos y técnicas de AngularJS si se desea añadir algún módulo nuevo.

5.3.1.3. CodeMirror

Este componente, de gran importancia y extensibilidad, puede ampliarse en múltiples facetas con poco esfuerzo, si bien su arquitectura está muy emparentada con jQuery y, por tanto, se requiere más destreza en esta librería. CodeMirror puede ampliarse añadiendo *addons*[7] (funcionalidades), *themes*[8] (combinaciones de colores para el resaltado de sintaxis en los editores), *language modes*[9] (modos de lenguaje que proporcionan resaltado de sintaxis), etc.

[5] http://plugins.jquery.com/
[6] http://ngmodules.org/
[7] http://codemirror.net/doc/manual.html#addons
[8] http://codemirror.net/demo/theme.html
[9] http://codemirror.net/mode/index.html

Algunas modificaciones sencillas serían las siguientes:

Añadir un *addon* Bastará con añadir el fichero con el *addon* al directorio `app/ext` o `app/extmod` (si se modifica) y añadir el enlace para que se cargue en la aplicación en el fichero `app/dviz.html`, después de la carga del fichero de CodeMirror, por supuesto. Es probable que el editor requiera configurar o activar la caracterísita del *addon*; la configuración del editor se realiza a partir de la línea 4 del fichero `app/main/console.js`.

Añadir un atajo de teclado Para añadir un atajo de teclado a CodeMirror habrá que añadir una clave con el nombre del atajo al objeto `extraKeys`[10] que se establece en la configuración del editor, en la línea 9 del fichero `app/main/console.js`. El nombre del atajo es fácil de encontrar en la documentación, pero la función a realizar requiere un buen conocimiento del API de CodeMirror.

Añadir un tipo de fichero (*language mode*) Para permitir la edición y subida de módulos con un nuevo tipo de fichero (modo de lenguaje) serían necesarias varias modificaciones.

1. Crear un método `put<extensión>` en el recurso `File` del fichero `app/model/models.js` (por ejemplo, bajo la línea 46) que tendrá como cabecera el `mime-type` del nuevo tipo de fichero.
2. Añadir el fichero JavaScript que implementa el modo de lenguaje en el directorio `app/ext` (o `app/extmod` si se modifica).
3. Incluir el enlace en `app/dviz.html`.
4. Añadir el *mime-type* del nuevo tipo de fichero al filtro de subida a partir de la línea 113 y la extensión (para navegadores más antiguos) en la línea 116 del fichero `app/control/projects.js` para permitir subir ficheros del nuevo tipo.
5. Añadir el *mime-type* del nuevo tipo de fichero a la variable `types` en la línea 206 y la extensión (para navegadores más antiguos) a la varia-

[10] http://codemirror.net/doc/manual.html#option_extraKeys

ble `exts` de la línea 205 del fichero `app/control/projects.js` para permitir añadir ficheros nuevos del nuevo tipo.

6. Añadir la llamada a la función correspondiente (`put<extensión>`) al nuevo módulo del recurso `File` en la línea 236 de `app/control/projects.js`.

5.3.2. Añadir una nueva página

Para añadir una nueva página con contenido dinámico (acceso a la BD) necesitaremos conocimientos de AngularJS, HTML5 y Bootstrap.

Necesitaremos crear los siguientes componentes nuevos:

- Una nueva vista (página HTML) en el directorio `app/view`, que no contendrá ni pie ni cabecera y que, normalmente llevará un `div` con un contenedor. Al ser dinámica, esta vista contendrá etiquetas específicas de AngularJS (con el prefijo `data-ng-`) para establecer el controlador, el modelo y los elementos del modelo presentes.
- Un nuevo modelo, que probablemente se deberá añadir al fichero `app/model/models.js`, en forma de *recurso*, con la configuración de acceso al servicio REST que implementa los métodos de acceso y modificación de datos.
- Un nuevo controlador (fichero JavaScript) en el directorio `app/control` y que, utilizando la inyección de dependencias, proveerá de los elementos de interfaz a la vista y los relacionará mediante funciones de interacción con el modelo.
- Una nueva ruta, que se añade al fichero `app/main/routes.js` en la llamada a `app.config` utilizando la función `when`, y que deberá tener el nombre de la vista como parámetro `templateUrl`, el nombre del controlador como parámetro `controller` y un título descriptivo (`title`).

Si se desea añadir a la barra de menú superior, habrá que modificar el fichero `app/view/header.html`. Si se quiere incluir en la lista de enlaces del pie, el fichero a modificar es `app/view/footer.html`. En cualquiera de

los dos casos, el enlace será parcial y apuntará a la ruta añadida anteriormente siempre precedido del separador de ruta parcial de AngularJS (#). Si la página requiere autenticación habrá que ocultar el enlace usando la directiva `data-ng-show="$session.isAuthenticated()"` y además evitar el acceso directo (utilizando la URL) añadiendo su ruta a la lista de rutas de autenticación requerida en la línea 13 de `app/main/interceptors.js`.

5.3.3. Añadir filtros al resultado de las órdenes

Con unas nociones suficientes de JavaScript y jQuery es posible filtrar la respuesta de las órdenes al intérprete para, por ejemplo, añadir, quitar o modificar patrones que respondan a una expresión regular.

Primero habría que introducir el filtro en la función anónima que se encuentra definida en el tercer parámetro de la llamada a `controller.sendCommand` de la línea 34 del fichero `app/main/console.js`. El documento JSON válido de respuesta se recibe en la variable `data`. Si contiene la clave `cmdResult`, el valor es una cadena de texto plano. Si, por el contrario, contiene la clave `cmdGraph`, entonces el valor es un objeto JSON válido con un objeto gráfico. Por último se cotejaría la expresión regular del filtro contra el valor del objeto deseado (`cmdGraph` o `cmdResult`) y, si coincide, se sustituiría. A partir de la línea 41 la función `update_content` utiliza esos valores para mostrarlos en la consola como respuesta a la orden.

Capítulo 6

Resultados

6.1. Resultados

6.1.1. Discusión de los resultados

Hemos obtenido una aplicación web con un conjunto reducido de funcionalidades pero suficiente para los requisitos planteados. No es un IDE completo de Haskell, ya que carece de muchas funcionalidades que podemos encontrar en otras herramientas más generales como un registro automático, perfiles de usuario avanzados, etc. Sin embargo, creemos que cumplir con unos requisitos mínimos con garantías es preferible.

También hemos obtenido una arquitectura flexible y potente, aunque algunos aspectos avanzados de seguridad y de extensibilidad no han sido explícitamente tratados e incorporados, nuevamente porque las necesidades iniciales no lo priorizaban. Aun así hemos implementado y propuesto mínimos satisfactorios.

Y, en cuanto a los componentes externos, tenemos módulos muy extensibles, lo que nos garantiza una gran generalidad. Esta cualidad tiene un precio, heredamos gran cantidad de código ajeno, con mayor riesgo de errores y dificultad de mantenimiento asociados.

Quizás tan importante como los artefactos son los conocimientos adquiridos, aunque no son un resultado tan fácilmente transferible como el código fuente y la documentación, en la que hemos invertido un esfuerzo considerable a pesar de que no pueda ser tan precisa como el código para un programador.

También creemos que algunos conceptos de diseño pueden ser de utilidad: los protocolos (contratos) en lugar de APIs cerrados para la representación y la extensibilidad basada en transparencia. La liberación del código fuente, que hemos aprovechado de otros en varios casos, es un resultado importante que nos gustaría devolver en la misma medida.

En cuanto al propósito inicial, lamentablemente el conjunto de representaciones gráficas que hemos presentado está lejos de ser completo. Nuestra idea estaba cerca de buscar una "representación universal" o, en cierta medida, muy general, pero nosotros no hemos podido conseguirla.

Aun así, creemos que la aplicación web obtenida es sencilla e intuitiva, con un estilo visual atractivo y de acuerdo a las últimas tendencias de diseño. Lamentablemente no hemos podido obtener una gran realimentación de usuarios. Nos habría encantado disponer de ella, pero la evaluación sistemática habría sido en sí misma un proyecto nuevo.

Esperamos que en el futuro nuestro trabajo pueda ser continuado y utilizado y que esa realimentación proporcione oportunidad para mejoras en todos los aspectos.

6.1.2. Conclusiones / Conclussions

No podemos estar de acuerdo con la idea, demasiado extendida, de que hacer una aplicación web es una tarea sencilla. Realizar una aplicación web seria soportando los últimos estándares de accesibilidad y usabilidad, con una interfaz adaptativa y que además soporte navegadores antiguos no es una tarea sencilla.

Para realizarla hemos necesitado nuevas herramientas, invertido tiempo en aprender a entenderlas, usarlas y sacar el mejor partido, y al final hemos obtenido casi el mismo resultado que utilizando herramientas conocidas. Pero en el cambio hemos ganado mucho más.

La falta de documentación de las herramientas más modernas se suple con una comunidad más activa y resolutiva, una nueva forma de aprender a programar mediante colaboración.

Gracias a esta colaboración hemos conseguido consensuar los objetivos del proyecto para obtener una herramienta funcional, trabajando en equipo con gran efectividad, gracias a la buena predisposición y actitud de

todos los miembros, incluyendo a nuestro director.

Creemos que la solución presentada puede ser una herramienta educativa de gran potencial, no tanto por lo que hemos conseguido implementar como por lo que aún puede realizarse con la arquitectura final. En concreto, a pesar de no centrarnos en la comunidad Haskell, podría tener interés en este entorno reorientando ligeramente el enfoque y mejorando los aspectos de seguridad.

Sin embargo, hemos aprendido que en cualquier proyecto hay que revisar las prioridades y estar abiertos a aceptar cambios si es necesario, sin perder de vista que el objetivo común es lo más importante.

También consideramos que nuestra actitud de aprendiz novel de Haskell ha sido útil para conseguir una herramienta amigable y usable para los neófitos en programación funcional, ya que hemos querido hacer la herramienta que nos gustaría usar.

Como conclusión final nos quedamos con dos descubrimientos. El primero es la efectividad del patrón de arquitectura MVC en el cliente, revelando a JavaScript como un lenguaje potente y complejo. Éramos escépticos sobre este lenguaje, pero la idea de "lenguaje de juguete" se ha esfumado. En especial, al conocer la gran comunidad existente y las capacidades del lenguaje en relación con el estándar HTML5. El segundo es la sencillez y potencia de los servicios REST con JSON, frente a la complejidad y rigidez de SOAP. Esta experiencia nos ha servido para contrastar esta tecnología y verificar la utilidad de esta combinación, que tan frecuentemente encontramos en las mejores aplicaciones en la Red.

Conclussions

We can not agree with the idea, too widespread, that making a web application is a simple task. Making a serious web application with support of the latest standards of accessibility and usability, with an adaptative interface and support for older browsers is not a simple task.

To accomplish this task we required new tools, invested time in learning, understanding, using them to get the best out of it, and in the end we obtained almost the same result as using familiar tools. But in the change we have get the most of it.

The lack of documentation of the latest tools is supplied with a more active and resolute community, a new way of learning to program through collaboration.

Through this collaboration we have achieved consensus on the objectives of the project to obtain a functional tool, working effectively as a team, thanks to the willingness and attitude of the members, including our advisor.

We believe that the presented solution can be an educational tool with great potential, not because of what we have achieved, but of the future implementations that can still be done with the final architecture. Specifically, although we have addressed the Haskell community, our approach would be interesting to them by redirecting our focus and slightly improving the safety aspects.

However, we have learned that in any project you must review the priorities and be open to accept changes if necessary, without losing sight in the common goal, that is the most important thing.

We also believe that our novel approach of Haskell apprentice has been helpful to get a friendly and usable tool for newbies in functional programming, because we wanted to make the tool we liked to use.

As a final conclusion we are left with two discoveries. The first is the effectiveness of architectural pattern MVC on the client, revealing that JavaScript is a powerful and complex language. We were skeptical about this language, but the idea of “toy language” is now gone. specially knowing the large existing community and the language capabilities in relation to the HTML5 standard. The second is the simplicity and power of REST services with JSON, given the complexity and rigidity of SOAP. This experience has helped us to test this technology and verify the incomes of this combination, so frequently found in the best applications on The Web.

6.1.3. Trabajo futuro

Como ya mencionamos en las conclusiones, a pesar de que la aplicación web es completamente funcional y satisface las necesidades iniciales, existen múltiples mejoras y extensiones que no hemos podido acome-

ter, pero que podrían aumentar la utilidad pública y la generalidad de la solución. A continuación enumeraremos algunas de las que consideramos más importantes:

- Añadir más tipos de representación: aunque nuestra librería de representación de árboles puede servir para representar otros tipos más simples, nuevos tipos enriquecerían la aplicación y atraerían a un público más amplio.
- Tratamiento de errores en consola: los mensajes de error que envía el intérprete cuando ocurre algún error proporcionan información detallada de la ubicación del error (módulo, línea y carácter). No sería difícil añadir un filtro a la respuesta de error que introdujese un enlace para poder abrir automáticamente el módulo afectado y posicionar el cursor en la posición del error, ayudando así a la corrección del mismo.
- Guardar estado del IDE: actualmente, al salir de la consola se pierde el estado de las solapas (módulos abiertos), el histórico y las posiciones del cursor en cada módulo abierto. Sería interesante poder guardar estos datos y restablecer esta información cada vez que se abra el proyecto, proporcionando mayor comodidad al manejo de la consola.
- Incluir una página de ayuda: aunque disponemos de la documentación necesaria para dar una ayuda rápida sobre el manejo de la aplicación, ésta no se encuentra en línea con la consola. En concreto, la información sobre la representación gráfica y el manual de arranque de este documento podrían ser parte de esa ayuda.
- Incluir una caja de búsqueda en Hoogle[1]: tanto si se dispone o no de ayuda en línea, incluir una enlace a este motor de búsqueda sería de gran utilidad.
- Incluir capacidad de ejecución de tutoriales interactivos: inspirados en la herramienta *TryHaskell*[2], creemos que esta capacidad sería una adición muy interesante.

[1] http://haskell.org/hoogle
[2] http://tryhaskell.org

- Incluir un proyecto *demo*: con no demasiado trabajo, sería posible incluir en la instalación uno o más proyectos identifificados de forma especial (p.e. como *proyecto demo*) en la base de datos del administrador. Éste tendría la capacidad para distribuir una copia actualizada de los proyectos a los usuarios que desee. Sería un pequeño pero útil paso hacia la incorporación de técnicas de enseñanza asistida más avanzadas.

- Mejorar las capacidades de interacción de las representaciones gráficas permitiendo, por ejemplo, la edición de la representación y la posterior conversión de la representación resultante a sus tipos en Haskell. Técnicamente esto es más sencillo de lo que parece, ya que sólo habría que invertir los métodos de representación actuales como, por ejemplo, implementar la función `fromJSON` a partir del documento de representación.

- Apertura al público de una web de demostración: añadir un registro abierto de usuarios (con o sin aprobación del administrador) con el que los usuarios accedan a una única base de datos pública de lectura y escritura (y no privada, como ocurre ahora) proporcionaría una visibilidad importante al proyecto. Técnicamente, la base de datos pública sería una *caja de arena* que se podría limpiar automáticamente a diario, sin mantenimiento manual.

- Edición colaborativa de proyectos: con algo más de trabajo y, posiblemente, un estudio académico más serio, podría implementarse la edición colaborativa o cooperativa de proyectos. Técnicamente se podría utilizar el recurso HTML5 *shared web-workers* que proporciona comunicación entre procesos en segundo plano.

- Permitir carpetas (anidadas) en proyectos: actualmente, sólo es posible subir módulos a un proyecto y las carpetas no están permitidas. Para proyectos más grandes, las carpetas pueden suponer una buena forma de organización.

- Utilizar *web-sockets* con un servidor web basado en Yesod, Snap, Happstack, HAppS u otro (reimplementando el servicio *Shell* en Haskell) po-

dría agilizar la evaluación de expresiones en GHCi, ya que no sería necesario crear un proceso de intérprete interactivo. Teóricamente esto podría acelerar tanto la comunicación como la ejecución de procesos. La integración con alguno de estos marcos de desarrollo web podría permitir, además, la creación de proyectos web que el usuario podría desplegar en el propio servidor, abriendo así la herramienta a la enseñanza de programación web con Haskell.

- Estudiar intérpretes seguros: existen varios intérpretes de Haskell que, supuestamente, proporcionan un entorno de ejecución más controlable y seguro. En el caso de una hipotética apertura al público, utilizar alguno de estos intérpretes se haría imprescindible por seguridad. También podría desarrollarse un intérprete a medida que minimizase los riesgos y proporcionase, al mismo tiempo, un entorno de ejecución con menos restricciones utilizando el concepto de *caja de arena*.

- Verificación de sintaxis inmediata: la mayoría de los editores de lenguajes de programación ya incorporan técnicas de compilación en segundo plano. Esto permite compilar mientras el usuario está escribiendo (o cuando el usuario termina de escribir), de modo que se pueden indicar errores y advertencias sobre la sintaxis de forma temprana, evitando una ejecución con errores. Técnicamente esto es factible utilizando *web-sockets* y algún verificador de sintáxis como *ghc-modi*[3] o *hdevtools*[4].

6.2. Contribución de los autores

El método de reparto de tareas para el presente trabajo se ha basado, en todas las etapas del diseño, desarrollo, pruebas, validación y documentación en el mismo concepto: trabajo en equipo a partes iguales utilizando una lista de tareas pendientes que era actualizada diariamente a medida que éstas se completaban, y que eran seleccionadas en función de la prioridad por el colaborador que terminaba su anterior tarea.

[3] http://www.mew.org/~kazu/proj/ghc-mod/en/ghc-modi.html
[4] https://github.com/bitc/hdevtools

Para ello hemos utilizado, como herramienta central, el control de versiones. Desde la primera documentación de requisitos hasta este documento, todo el trabajo ha sido desarrollado utilizando de forma diaria esta herramienta. No sólo se ha utilizado para almacenar el código y la documentación, si no que además se han utilizado los comentarios de las subidas para llevar un diario de las tareas realizadas.

Por ello consideramos que el mejor método para evaluar y explicar la contribución de cada autor sería acudir a la traza de comentarios de las subidas a nuestro repositorio[5], que explican de forma pormenorizada las contribuciones individuales, día a día y línea a línea (de código y documentación).

No obstante incluimos aquí un resumen para mayor comodidad.

6.2.1. María del Rosario Baena Priego

- Programación AngularJS
- Programación jQuery
- Programación en HTML5
- Programación y ajustes CSS
- Diseño y selección de componentes gráficos
- Configuración y programación de Bootstrap
- Diseño de la arquitectura MVC basada en AngularJS
- Evaluación de PHP en la fase de selección de lenguajes
- Diseño y desarrollo de una primera maqueta en JavaScript
- Diseño de interceptores para control de sesión y autenticación
- Pruebas de nuevas funcionalidades
- Configuración e instalación de CodeMirror
- Adaptación completa de WTerm

[5] https://dalila.sip.ucm.es/svn_vizhaskell

- Adaptación de xeditable y file-uploader
- Creación de la plantilla inicial de la memoria
- Búsqueda y redacción del estado del arte en la memoria
- Elaboración de los 2 capítulos más importantes de la memoria
- Escritura compartida del capítulo de resultados de la memoria

6.2.2. Roberto Aragón

- Evaluación de Yesod y Snap en la fase de selección de lenguajes
- Diseño de la arquitectura de servicios REST
- Programación en HTML5
- Programación y ajustes CSS
- Instalación, programación y pruebas con D3
- Programación del modelo (recursos y servicios AngularJS)
- Documentación de casos de uso y diagrama de componentes y despliegue
- Validación de accesibilidad y usabilidad
- Programación PHP del servicio PHPRest
- Instalación y configuración de CouchDB
- Programación Haskell (basada en código de Manuel Montenegro)
- Pruebas en integración
- Instalación de entornos de desarrollo e integración
- Mejora y simplificación de la plantilla de la memoria
- Escritura de 3 capítulos de la memoria, resumen e introducción
- Escritura compartida del capítulo de resultados de la memoria

Apéndice A

Listados seleccionados de código fuente

A.1. Módulos de la librería de soporte a la representación

A.1.1. La clase `Representation`

Listado A.1: Código fuente del módulo `Representation.hs`

```
1  {-# LANGUAGE FlexibleContexts #-}
2
3  {-|
4  Module      : VizHaskell.Core
5  Description : Definition of VizHaskell's main classes and @render@
       function.
6
7  This module defines the main classes for defining representation types
       and
8  representable datatypes.
9  -}
10 module VizHaskell.Core(
11           -- * Representation types
12           Representation,
13           -- * Representable values
14           RPair,
15           rPairRepresentation,
16           rPairValue,
17           buildRPair,
18           -- * Representable types
19           VizRepresentable(..),
```

```
20            -- * Representation function
21            render
22        ) where
23
24  import Data.Text.Lazy.Builder
25  import qualified Data.Text.Lazy as T
26  import Data.Aeson.Encode
27  import Data.Aeson
28
29  {-|
30    An empty class with no methods. Only those types denoting a
          representation
31    type (such as string representation, tree representation, etc.) may
          be
32    instances of this class
33  -}
34  class Representation rep
35
36  {-|
37    It associates a representable value @a@ with its representation type
          @rep@.
38    -}
39  data RPair rep a = RPair rep a
40
41  {-|
42    Representation type
43  -}
44  rPairRepresentation :: RPair rep a -> rep
45  rPairRepresentation (RPair rep _) = rep
46
47  {-|
48     Value to be represented
49  -}
50  rPairValue :: RPair rep a -> a
51  rPairValue (RPair _ a) = a
52
53  {-|
54     It builds a representable value
55  -}
56  buildRPair :: Representation rep => rep -> a -> RPair rep a
57  buildRPair = RPair
58
59
60  {-|
61    This class characterizes all the types which can be shown by
          VizHaskell's
62    web interface.
63  -}
```

```
64 class VizRepresentable a where
65   {-|
66     It returns a JSON object containing the representation of
67     the given value
68   -}
69   vizToJSON :: a -> Value
70
71
72
73 {-|
74   This is the main function called by VizHaskell's web interface, and
       it
75   returns a JSON representation of the value passed as second parameter
       , by
76   using the representation type given by the first parameter.
77 -}
78 render :: (Representation rep, VizRepresentable (RPair rep a)) => rep
     -> a -> String
79 render rep x = T.unpack (toLazyText (fromValue (vizToJSON (RPair rep x)
     )))
```

A.1.2. La clase `RepresentationString`

Listado A.2: Código fuente del módulo `StringRepresentation.hs`

```
1  {-# LANGUAGE FlexibleInstances, OverloadedStrings #-}
2
3  {-|
4  Module      : VizHaskell.StringRepresentation
5  Description : Representation of all the types implementing @Show@.
6
7  This module defines the simplest representation kind: a string
       representation.
8  It defines the corresponding 'RepresentationString' type, which is an
9  instance of 'VizHaskell.Core.Representation'. It also specifies that
       all
10 instances of 'Show' must also be instances of 'RepresentationString'.
11 -}
12 module VizHaskell.StringRepresentation(RepresentationString(..)) where
13
14 import VizHaskell.Core
15 import Data.Aeson
16 import Data.String
17
18
19 {-|
20   A representation type, which specifies that the value associated with
21   this representation (see 'VizHaskell.Core.RPair') must be shown as
       plain text.
```

```
22 -}
23 data RepresentationString = RepresentationString
24
25 {-|
26   'RepresentationString' is a representation type
27 -}
28 instance Representation RepresentationString
29
30 {-|
31   This specifies that all the types implementing the 'Show' class can
32   be represented with 'RepresentationString'
33 -}
34 instance Show a => VizRepresentable (RPair RepresentationString a)
       where
35   vizToJSON rs = object [ "text" .= show (rPairValue rs) , "repType" .=
         String (fromString "text") ]
```

A.1.3. La clase `RepresentationTree`

Listado A.3: Código fuente del módulo `TreeRepresentation.hs`

```
1  {-# LANGUAGE FlexibleInstances, OverloadedStrings, FlexibleContexts #-}
2
3  {-|
4  Module      : VizHaskell.TreeRepresentation
5  Description : Representation of all the types that can be shown as a
       tree.
6
7  This module defines the 'TreeRepresentation' class, with allows one to
       define
8  a data type as being representable by a tree.
9  -}
10 module VizHaskell.TreeRepresentation(
11         TreeRepresentable(..),
12         RepresentationTree(..)
13       ) where
14
15 import VizHaskell.Core
16 import Data.Aeson
17 import Data.Aeson.Types(Pair)
18 import Data.String
19 import qualified Data.Text
20 import qualified Data.Vector as V
21
22
23 {-|
24   Class defining all the datatypes that may be represented
25   as a tree.
26 -}
```

```
27 class TreeRepresentable t where
28   {-|
29     It returns the contents of a node.
30   -}
31   contents  :: t b -> Maybe b
32
33   {-|
34     It returns the children of a node.
35   -}
36   children  :: t b -> [t b]
37   {-|
38     Additional info attached to the given node.
39     Unless explicitly overwritten by concrete representations,
40     it defaults to @Nothing@.
41   -}
42   label     :: t b -> Maybe String
43   {-|
44     CSS style being applied to the
45     given node. Unless explicitly overwritten by concrete
          representations,
46     it defaults to @Nothing@.
47   -}
48   className :: t b -> Maybe String
49
50   label     _ = Nothing
51   className _ = Nothing
52
53
54 {-|
55   This representation type specifies that the value associated with
56   this representation (see 'VizHaskell.Core.RPair') must be shown as
57   a tree. It receives another representation type, which refers to
58   the representation type of the values contained within the nodes.
59 -}
60 data RepresentationTree rep = RepresentationTree rep
61
62 instance Representation rep => Representation (RepresentationTree rep)
63
64 {-|
65   All the types implementing 'TreeRepresentable' are representable with
66   a 'RepresentationTree', provided their nodes are also representable.
67 -}
68 instance (TreeRepresentable t, Representation rep, VizRepresentable (
       RPair rep b))
69     => VizRepresentable (RPair (RepresentationTree rep) (t b)) where
70     vizToJSON rpair =
71       case contents repValue of
72         Nothing -> object [ "value" .= object [] , "children" .= Array
```

```
            (V.empty), "repType" .= String "tree"]
73        Just val -> object ([
74                             "repType" .= String "tree",
75                             "value" .= vizToJSON (buildRPair repNodes
                                  val),
76                             "children" .= Array (V.fromList
77                                (map (vizToJSON . buildRPair rep) (
                                     children repValue)))
78                            ] ++ showIfMaybe "label" (label repValue)
79                              ++ showIfMaybe "className" (className
                                   repValue))
80      where rep@(RepresentationTree repNodes) = rPairRepresentation
           rpair
81            repValue = rPairValue rpair
82            showIfMaybe :: Data.Text.Text -> Maybe String -> [Pair]
83            showIfMaybe attr Nothing = []
84            showIfMaybe attr (Just str) = [attr .= String (fromString
                 str)]
```

A.1.4. La clase `RepresentationRaw`

Listado A.4: Código fuente del módulo RawRepresentation.hs

```
1  {-# LANGUAGE FlexibleInstances, OverloadedStrings, FlexibleContexts #-}
2
3  {-|
4  Module      : VizHaskell.RawRepresentation
5  Description : Special representation of raw JSON data
6
7  This module defines the 'RawRepresentation' class, with allows one to
      use
8  a data type that already is representable with a valid JSON contract
9  -}
10 module VizHaskell.RawRepresentation(RepresentationRaw(..)) where
11
12 import VizHaskell.Core
13 import Data.Aeson
14 import Data.Aeson.Types(Pair)
15 import Data.String
16 import qualified Data.Text
17 import qualified Data.Vector as V
18
19 data RepresentationRaw = RepresentationRaw
20
21 instance Representation RepresentationRaw
22
23 instance ToJSON a => VizRepresentable (RPair RepresentationRaw a) where
24   vizToJSON rpair = toJSON (rPairValue rpair)
25 {-|
```

```
26    The raw representable object is supposed to implement toJSON, so no
          other
27    kind of processing is needed but to call it.
28  -}
```

A.2. Librerías de JavaScript

A.2.1. Librería dndTree modificada

Listado A.5: Código fuente de la librería `tree.js`

```
1   // Compatible with simple tree.addTree
2   function addTree(id, json, csss, options) {
3     // Calculate total nodes, max label length
4     var totalNodes = 0;
5     var maxLabelLength = 0;
6     // panning variables
7     var panSpeed = 200;
8     var panBoundary = 20; // Within 20px from edges will pan when
          dragging.
9     // Misc. variables
10    var i = 0;
11    var duration = 750;
12    var root;
13
14    if(csss && csss !== "")
15      d3.select(id).append("style")
16      .attr("scoped", "true")
17      .attr("type", "text/css")
18      .text(csss);
19
20    // size of the diagram
21    var viewerWidth = options.canvas.width;
22    var viewerHeight = options.canvas.height;
23
24    var tree = d3.layout.tree()
25      .size([viewerHeight, viewerWidth]);
26
27    // define a d3 diagonal projection for use by the node paths later on
          .
28    var diagonal = d3.svg.diagonal()
29      .projection(function(d) {
30        return [d.y, d.x];
31      });
32
33    // A recursive helper function for performing some setup by walking
34    // through all nodes
35    function visit(parent, visitFn, childrenFn) {
```

```
36      if(!parent) return;
37      visitFn(parent);
38      var children = childrenFn(parent);
39      if(children) {
40        var count = children.length;
41        for(var i = 0; i < count; i++) {
42          visit(children[i], visitFn, childrenFn);
43        }
44      }
45    }
46
47    // Call visit function to establish maxLabelLength
48    visit(json, function(d) {
49      totalNodes++;
50      maxLabelLength = d.label ? Math.max(d.label.length, maxLabelLength)
          : 5;
51    }, function(d) {
52      return d.children && d.children.length > 0 ? d.children : null;
53    });
54
55    // TODO: Pan function, can be better implemented.
56    function pan(domNode, direction) {
57      var speed = panSpeed;
58      if (panTimer) {
59        clearTimeout(panTimer);
60        translateCoords = d3.transform(svgGroup.attr("transform"));
61        if (direction == 'left' || direction == 'right') {
62          translateX = direction == 'left' ?
63            translateCoords.translate[0] + speed :
64            translateCoords.translate[0] - speed;
65          translateY = translateCoords.translate[1];
66        } else if (direction == 'up' || direction == 'down') {
67          translateX = translateCoords.translate[0];
68          translateY = direction == 'up' ?
69            translateCoords.translate[1] + speed :
70            translateCoords.translate[1] - speed;
71        }
72        scaleX = translateCoords.scale[0];
73        scaleY = translateCoords.scale[1];
74        scale = zoomListener.scale();
75        svgGroup.transition().attr("transform",
76          "translate(" + translateX + "," + translateY + ")scale(" +
              scale + ")");
77        d3.select(domNode).select('g.node').attr("transform",
78          "translate(" + translateX + "," + translateY + ")");
79        zoomListener.scale(zoomListener.scale());
80        zoomListener.translate([translateX, translateY]);
81        panTimer = setTimeout(function() {
```

```
82          pan(domNode, speed, direction);
83        }, 50);
84      }
85    }
86
87    // Define the zoom function for the zoomable tree
88    function zoom() {
89      svgGroup.attr("transform",
90        "translate(" + d3.event.translate + ")scale(" + d3.event.scale +
              ")");
91    }
92
93    // define the zoomListener which calls the zoom function on the "zoom
         " event
94    // constrained within the scaleExtents
95    var zoomListener = d3.behavior.zoom().scaleExtent([0.1, 3]).on("zoom
         ", zoom);
96
97    // Add svg div resizable and closeable container
98    var baseDiv = d3.select(id).append("div")
99      .attr("class", "canvas");
100
101   // Avoid focus steeling of the canvas by console command line
102   baseDiv.on("click", function() { d3.event.stopPropagation(); });
103
104   // Add a close anchor at the top right side
105   baseDiv.append("button")
106     .attr("class", "close hidden-print") // Don't print this button
107     .on("click", function() { d3.select(this.parentNode).remove(); })
108     .html("&times;");
109
110   // define the baseSvg, attaching a class for styling and the
         zoomListener
111   var baseSvg = baseDiv.append("svg")
112     .attr("width", "95%") // Leave space to the resize handler
113     .attr("height", "95%")
114     .call(zoomListener);
115
116   // Helper functions for collapsing and expanding nodes.
117   function collapse(d) {
118     if (d.children) {
119       d._children = d.children;
120       d._children.forEach(collapse);
121       d.children = null;
122     }
123   }
124
125   function expand(d) {
```

```
126     if (d._children) {
127       d.children = d._children;
128       d.children.forEach(expand);
129       d._children = null;
130     }
131   }
132
133   // Function to center node when clicked/dropped so node doesn't get
        lost
134   // when collapsing/moving with large amount of children.
135   function centerNode(source, svg, onlyvert) {
136     scale = zoomListener.scale();
137     x = -source.y0;
138     y = -source.x0;
139     var labeloffset = maxLabelLength * 5;
140     if(onlyvert) x = x * scale + labeloffset;
141     else x = x * scale + labeloffset + viewerWidth / 2;
142     y = y * scale + viewerHeight / 2;
143     svg.select("g").transition()
144       .duration(duration)
145       .attr("transform", "translate(" + x + "," + y + ")scale(" + scale
            + ")");
146     zoomListener.scale(scale);
147     zoomListener.translate([x, y]);
148   }
149
150   // Toggle children function
151   function toggleChildren(d) {
152     if(d.children) {
153       d._children = d.children;
154       d.children = null;
155     } else if(d._children) {
156       d.children = d._children;
157       d._children = null;
158     }
159     return d;
160   }
161
162   // Toggle children on click.
163   function click(d) {
164     if(d3.event.defaultPrevented) return; // click suppressed
165     d = toggleChildren(d); // expand
166     if((!d.children || d._children) &&
167         d.value && d.value.repType && d.value.repType === "tree")
168       addTree(id, d.value, null, options);
169     else update(d);
170     // Prevent focus steeling from canvas by stoping click propagation
          to
```

```
171     // the console
172     d3.event.stopPropagation();
173   }
174
175   function update(source) {
176     // Compute the new height: function counts total children of root
          node and
177     // sets tree height accordingly. This prevents the layout looking
          squashed
178     // when new nodes are made visible or looking sparse when nodes are
           removed
179     // This makes the layout more consistent.
180     var levelWidth = [1];
181     var childCount = function(level, n) {
182       if (n.children && n.children.length > 0) {
183         if (levelWidth.length <= level + 1) levelWidth.push(0);
184         levelWidth[level + 1] += n.children.length;
185         n.children.forEach(function(d) {
186           childCount(level + 1, d);
187         });
188       }
189     };
190     childCount(0, root);
191     var newHeight = d3.max(levelWidth) * 50; // 25 pixels per line
192     tree = tree.size([newHeight, viewerWidth]);
193
194     // Compute the new tree layout.
195     var nodes = tree.nodes(root).reverse(),
196         links = tree.links(nodes);
197
198     // Set widths between levels based on maxLabelLength.
199     nodes.forEach(function(d) {
200       d.y = (d.depth * (maxLabelLength * options.node.sep));
201       // Alternatively, to keep a fixed scale one can set a fixed depth
            per
202       // level. Normalize for fixed-depth by commenting out below line
203       // d.y = (d.depth * 500); //500px per level.
204     });
205
206     // Update the nodes...
207     node = svgGroup.selectAll("g.node")
208       .data(nodes, function(d) {
209         return d.id || (d.id = ++i);
210       });
211
212     // Enter any new nodes at the parent's previous position.
213     var nodeEnter = node.enter()
214       .append("g")
```

```
215       .attr("class", "node")
216       .attr("transform", function(d) {
217         return "translate(" + source.y0 + "," + source.x0 + ")";
218       })
219       .on("click", click);
220
221     nodeEnter.append("circle")
222       .attr("r", options.node.radius)
223       .attr("class", function(d) {
224       return d._children && d._children.length ?
225            "w-children" : d.className ?  d.className : "wo-children";
226       });
227
228     nodeEnter.append("text")
229       .attr("x", function(d) {
230         return d.children || d._children ? -10 : 10;
231       })
232       .attr("dy", ".35em")
233       .attr('class', 'nodeText')
234       .attr("text-anchor", function(d) {
235         return d.children || d._children ? "end" : "start";
236       })
237       .text(function(d) { return d.label; })
238       .style("fill-opacity", 0);
239
240     // Update the text to reflect whether node has children or not.
241     node.select('text')
242       .attr("x", function(d) {
243         return d.children || d._children ? -10 : 10;
244       })
245       .attr("text-anchor", function(d) {
246         return d.children || d._children ? "end" : "start";
247       })
248       .text(function(d) {
249         return d.label;
250       });
251
252     // Change the circle fill depending on whether
253     // it has children and is collapsed
254     node.select("circle")
255       .attr("r", options.node.radius)
256       .attr("class", function(d) {
257       return d._children && d._children.length ?
258            "w-children" : d.className ?  d.className : "wo-children";
259       });
260
261     // Put value inside node
262     nodeEnter.append("text")
```

```
263       .attr("dx", function(d) {
264         return d.value && d.value.text ? -3 * d.value.text.length : -4;
265       })
266       .attr("dy", "4")
267       .attr("stroke", "#fff")
268       .text(function(d) {
269         return d.value && d.value.text ? d.value.text : ".";
270       });
271
272     // Transition nodes to their new position.
273     var nodeUpdate = node.transition()
274       .duration(duration)
275       .attr("transform", function(d) {
276         return "translate(" + d.y + "," + d.x + ")";
277       });
278
279     // Fade the text in
280     nodeUpdate.select("text")
281       .style("fill-opacity", 1);
282
283     // Transition exiting nodes to the parent's new position.
284     var nodeExit = node.exit().transition()
285       .duration(duration)
286       .attr("transform", function(d) {
287         return "translate(" + source.y + "," + source.x + ")";
288       })
289       .remove();
290
291     nodeExit.select("circle")
292       .attr("r", 0);
293
294     nodeExit.select("text")
295       .style("fill-opacity", 0);
296
297     // Update the links...
298     var link = svgGroup.selectAll("path.link")
299       .data(links, function(d) {
300         return d.target.id;
301       });
302
303     // Enter any new links at the parent's previous position.
304     link.enter().insert("path", "g")
305       .attr("class", "link")
306       .attr("d", function(d) {
307         var o = {
308           x: source.x0,
309           y: source.y0
310         };
```

```
311         return diagonal({source: o, target: o});
312       });
313
314     // Transition links to their new position.
315     link.transition()
316       .duration(duration)
317       .attr("d", diagonal);
318
319     // Transition exiting nodes to the parent's new position.
320     link.exit().transition()
321       .duration(duration)
322       .attr("d", function(d) {
323         var o = {
324           x: source.x,
325           y: source.y
326         };
327         return diagonal({
328           source: o,
329           target: o
330         });
331       })
332       .remove();
333
334     // Stash the old positions for transition.
335     nodes.forEach(function(d) {
336       d.x0 = d.x;
337       d.y0 = d.y;
338     });
339   }
340
341   // Append a group which holds all nodes and which
342   // the zoom Listener can act upon.
343   var svgGroup = baseSvg.append("g");
344
345   // Define the root
346   root = json;
347   root.x0 = viewerHeight / 2;
348   root.y0 = 0;
349
350   // Layout the tree initially and center on the root node.
351   update(root);
352   centerNode(root, baseSvg, true);
353 }
```

Apéndice B

Descripción detallada de los componentes propios del SW

B.1. Métodos del controlador principal de la aplicación

login Como parámetro de entrada se debe indicar el formulario que contiene tanto el correo electrónico (`email`) como la contraseña (`password`). Con estos datos se invocará el método `login` del servicio `AuthService` pasándole tanto las credenciales de autenticación, *email* y *password*, como las funciones de éxito o error a ejecutar tras la respuesta AJAX (*Asynchronous JavaScript And XML*). Si las credenciales son correctas se creará una sesión válida mediante la ejecución del servicio `Session.create`, redirigiendo al usuario a la pantalla de consola. En caso de error, éste será mostrado mediante la función `addNotify` (ver 4.2.1.8).

logout Con la ejecución de este método se realiza el cierre e invalidación de la sesión. Para ello, se ejecuta el método `logout` del servicio `AuthService` que elimina del servidor la sesión, y se ejecuta el método `Service.destroy` para invalidar la sesión activa del usuario. Una vez cerrada la sesión se redirige al usuario a la pantalla de inicio (*Acerca de*).

changepass Este método solo puede ser invocado si existe una sesión válida del usuario. Como parámetros de entrada se debe indicar el for-

mulario que contiene los campos de nueva contraseña (`newpass`) y verificación de contraseña (`confpass`). Para realizar el cambio de contraseña se debe tener en cuenta el perfil del usuario:

- Si el usuario es el administrador del servidor (`Session.isRoot`), el cambio se realiza enviado una petición mediante la URL `/db/_config/admins` con el *email* y nueva contraseña.
- Para el resto de usuarios, primero se recuperan los datos del usuario mediante la llamada a `User.get` y después se ejecuta el método `user.$save` con la contraseña modificada sobre el objeto recuperado.

En cualquiera de los dos casos, si el cambio tiene éxito, es necesario realizar una llamada al servicio `AuthService.login` ya que el servidor invalida la sesión activa del usuario y es necesario crearla de nuevo. En caso de error, se muestra al usuario mediante la función de notificación (`addNotify`).

resetpass Como parámetro de entrada es necesario enviar los datos del formulario con los campos `email` (correo electrónico) y `nombre` (nombre completo del usuario). Para realizar el restablecimiento de contraseña ejecuta el método `Shell.execute` con el identificador de operación `resetpass`, *email*, nombre y las funciones a ejecutar en caso de éxito o fracaso de la operación. Tanto si la operación se ejecuta con éxito como si ocurre un error, se informa al usuario de ello con un mensaje emergente.

B.2. Métodos del controlador de la gestión de usuarios

init Esta función es la encargada de obtener todos los usuarios de la BD. Para ello ejecuta el método `query` del recurso `User`, al que se le pasan dos funciones que serán ejecutadas dependiendo del éxito o fracaso de la ejecución del método. Si no se producen errores, los usuarios (si los hubiera) se cargan en la estructura `users` que gestiona el controlador. Si hay algún error en la ejecución se informará de ello.

checkName Esta función se ejecuta cuando desde la vista se introduce el nombre del usuario. Dado que es un campo obligatorio, se verifica que se haya introducido un valor válido y no vacío.

checkEmail Esta función es ejecutada cuando desde la vista se introduce el correo electrónico que identifica al usuario. Se valida que el valor introducido sea correcto y no se admite un valor vacío.

checkPassword Esta función es ejecutada cuando desde la vista se introduce la contraseña. Dado que es un campo obligatorio se verifica que se introduzca un valor válido y no vacío.

showPassword Esta función solo se utiliza para ofuscar la contraseña en la vista, ya que está encriptada y, por tanto, no ofrece ninguna utilidad y debe ocultarse por seguridad.

editUser Cuando esta función es ejecutada, se muestra un formulario que permite la modificación de los datos del usuario. El único dato que no está disponible para modificación es el correo electrónico, ya que es el identificador único.

addUser Añade un nuevo elemento al objeto `users` del modelo y muestra el formulario para introducir los datos del nuevo usuario.

cancelUser Esta función permite cancelar tanto las operaciones de modificación de un usuario como el alta.

createUser Esta función recibe los datos del formulario para la creación del usuario, mostrados mediante la función `addUser`. Con estos datos se crea un nuevo recurso `User` y se ejecuta el método `$save` sobre el recurso creado. A este método se le pasan dos funciones, una para el caso de éxito y otra para el caso de error. Si la operación se realiza con éxito, se procede a crear la base de datos asociada al usuario mediante el método `save` del recurso `Database`. Tanto si la operación tiene éxito como si no es así, se informa de ello.

El nombre de la base de datos asociada al usuario creado se generará a partir del correo electrónico del usuario, sustituyendo los caracteres "@" por "$" y los puntos por guiones bajos. Estos cambios son necesarios debido a que CouchDB no acepta estos caracteres como nombre de base de datos válida.

saveUser La actualización de datos se realiza invocando el método `save` del recurso `User`. Se pueden modificar todos los datos asociados al usuario salvo el correo electrónico, ya que éste sirve como identificador único. Para comprobar si se ha realizado una modificación de la contraseña del usuario es necesario recuperar la contraseña anterior mediante el método `get` del recurso `User`. Si se ha modificado, en el envío de la petición se incluirá la nueva contraseña, en caso contrario este dato será omitido.

deleteUser El borrado de un usuario requiere confirmación, ya que supone la eliminación tanto del usuario como de su base de datos. Para ello, se llamará a la función `modalConfirm` (que se encuentra en el fichero `app/main/dialog.js`), a la que se le pasan dos funciones, una para el caso de confirmación de operación y otra para el caso de disconformidad.

Si se confirma la operación de borrado, se borrará el usuario mediante el método `delete` del recurso `User`. Si tiene éxito se elimina la BD asociada al usuario mediante la ejecución del método `delete` del recurso `Database`.

B.3. Métodos del controlador de la gestión de proyectos

init Esta función es la encargada de recuperar todos los proyectos del usuario de la base de datos. Para ello se invoca el método `query` del recurso `Project` pasándole dos funciones que serán ejecutadas en caso de éxito o error. Si la ejecución tiene éxito, los datos recibidos de la base de datos son cargados en la estructura `projects`, seleccionando el primer elemento de esta estructura como el proyecto actual (`projectSelected=projects[0]`). Si, por el contrario, ocurre algún error, éste será mostrado en usando la función `addNotify`.

uploader Esta variable permite configurar y gestionar la subida de ficheros al proyecto actual. El módulo requiere la configuración de los siguientes parámetros:

```
1   $scope.uploader = $fileUploader.create({
2     method: 'PUT',
3     removeAfterUpload: false,
4     isUploadForm: false,
5     scope: $scope
6   });
```

en la línea 2 se indica el método *http* que debe ejecutarse para el subida del fichero; en la 3, que la eliminación de los ficheros de la lista se realizará de forma manual; en la 4 que la subida no se realizará mediante un formulario, sino subiendo el contenido del fichero y, finalmente, en la 5 que el entorno del controlador es compartido por el módulo.

El módulo `file-uploader` permite definir el comportamiento de una serie de métodos en el proceso de gestión de la subida de ficheros. Nosotros detallaremos aquellos que hemos programado para el funcionamiento correcto de la subida de ficheros.

filters Se trata de una lista de funciones que pueden ser definidas para cancelar la subida de algunos ficheros. Hemos definido los suientes filtros: (1) Solo se permite la subida de ficheros *Haskell* (.hs), *literate Haskell* (.lhs) o *Cascading Style Sheets* (.css); y (2) no se permite la subida de ningún fichero que ya se encuentre en el proyecto, es decir, la validación se realiza recorriendo la lista de ficheros asociada al proyecto actual (`projectSelected.files`).

beforeupload Esta función se ejecuta antes de realizar el proceso de subida del fichero. Nos permite definir la URL de subida del fichero y especificar el *Content-Type* del fichero. En la URL especificamos la base de datos del usuario, el identificador del proyecto, la revisión asociada al proyecto y el nombre del fichero a subir.

success Esta función se ejecuta cuando la subida de un fichero se ha completado con éxito. En este caso añadimos el fichero a la lista de ficheros de la estructura `projectSelected.files`, actualizamos la revi-

sión asociada al proyecto y eliminamos el fichero de la lista de ficheros pendientes de subir del objeto `uploader`.

completeall Se ejecuta cuando la subida de todos los ficheros se ha completado, independientemente si se ha subido con éxito o no. En este caso recorremos la lista de ficheros a subir y, si el atributo `isError` está marcado, guardamos el nombre del fichero para mostrar posteriormente un mensaje con aquellos ficheros que no han sido subidos correctamente.

removeFileUploader Esta función permite eliminar aquellos ficheros que, estando inicialmente preparados para ser subidos, se decide no confirmar. Para ello sólo se necesita indicar el índice que ocupa dentro de la lista del fichero para que sea eliminado.

addProject Recibe como parámetro el nombre del proyecto. Para realizar el alta se ejecuta el método `save` del recurso `Project`, pasándole los siguientes parámetros: el nombre del proyecto, el nombre de la base de datos del usuario, la función a ejecutar en caso de éxito y la función a ejecutar en caso de error. En caso de error el mensaje se muestra usando la función `addNotify`. Si el alta se realiza con éxito se realizan las siguientes acciones:

1. Se añade un nuevo elemento a la estructura `projects`:

```
$scope.inserted = {
  "id" : data.id, // Identificador del proyecto
  "name": name,   // nombre del proyecto
  "rev": data.rev,// revision del documento
  "files": []     // lista de ficheros
}
$scope.projects.push($scope.inserted);
```

2. Se actualiza la variable `projectSelected` para que apunte al nuevo proyecto.

3. Se limpia la estructura `tabFiles` y la lista de ficheros pendientes de subida `uploader.clearQueue()`.

4. Y, por último, se activa la pestaña de la consola.

deleteProject La eliminación de un proyecto implica el borrado completo del proyecto y de todos los ficheros asociados a él. Por tanto, antes de realizar la operación se pide confirmación de la operación. La confirmación se realiza llamando a la función `modalConfirm`, a la que se le pasa la pregunta a mostrar y la función en caso de respuesta afirmativa.

Para borrar un proyecto se ejecuta el método `delete` del recurso `project` pasándole el identificador del proyecto, la revisión y las funciones a ejecutar en caso de éxito o error. Si se produce un error, éste será mostrado. Si la operación es exitosa, se realizan las siguientes acciones:

1. Se elimina el proyecto de la estructura `projects`.
2. Se selecciona como proyecto actual el primer proyecto de la lista: `projectSelected=projects[0]`.
3. Se limpia la estructura `tabFiles` y la lista de ficheros pendientes de subir `uploader.clearQueue()`.
4. Y por último, se activa la pestaña de la consola.

changeProject Esta función comprueba primero si se puede realizar un cambio de proyecto buscando en la estructura `tabFiles` si existen ficheros modificados (`modified=true`) y que aun no han sido guardados. En ese caso, se solicita al usuario confirmación para cambiar de proyecto y, eventualmente, perder los cambios sin guardar. Si el usuario confirma la operación, se realiza el cambio de proyecto y si no es así, se restablece el proyecto seleccionado antes de la llamada.

newFile Esta función permite crear un nuevo fichero asociado al proyecto. Antes de crear el nuevo fichero se comprobará que la extensión del fichero es una de las permitidas: *Haskell* (.hs), *literate Haskell* (.lhs) o *Cascading Style Sheets* (.css). Si se trata de un fichero Haskell, el nombre del fichero debe cumplir con la convención de nombres de GHC. Una vez pasadas las validaciones, se ejecuta uno de los métodos del recurso `File` dependiendo del tipo de fichero a crear: `puths`, `putlhs` o `putcss`. Se le pro-

porciona el nombre de la base de datos, el identificador del proyecto, la revisión del proyecto, el nombre del fichero y las funciones a ejecutar en caso de éxito o error. Si la operación tiene éxito, el fichero es añadido a la lista de ficheros de la variable `projectSelected` con los siguientes datos: nombre del fichero y el tipo de fichero. Además, se actualiza el atributo `projectSelected.rev` con el valor de la nueva revisión asociada al documento de proyecto.

saveFile Esta función permite salvar el contenido del fichero. Para ello, se ejecuta uno de los métodos siguientes del recurso `File` (dependiendo del tipo de fichero a crear): `puths`, `putlhs` o `putcss`. Se le proporciona el nombre de la base de datos, el identificador del proyecto, la revisión del proyecto, el nombre del fichero, el contenido del fichero y las funciones a ejecutar en caso de éxito o error. Si la modificación tiene éxito se modifica el atributo `modified` de la estructura `tabFiles` asociada al fichero modificado y se refresca el modelo. Además, se actualiza el atributo `projectSelected.rev` con el valor de la nueva revisión asociada al documento de proyecto.

openFile Esta función es la encargada de obtener el contenido del fichero solicitado. Si el fichero no se encuentra en la estructura `tabFiles`, éste deberá recuperarse de la base de datos. Para ello se ejecuta el método `get` del recurso `File` indicando el identificador del proyecto, la revisión del proyecto, el nombre del fichero a recuperar y las funciones a ejecutar en caso de éxito o error. Si no se puede recuperar el contenido del fichero se muestra el error occurrido. Si se ha recuperado el contenido del fichero con éxito, se añade un elemento más a la estructura `tabFiles`.

Una vez que el fichero se encuentra en la estructura `tabFiles`, se activa la propiedad `active` que provocará que en la vista se seleccione la solapa del fichero y se cargue el contenido del fichero en el editor. Antes de mostrar el contenido del fichero en el editor se necesario ejecutar las siguientes órdenes:

```
editor.off("change", $scope.changefun);
editor.setValue(text); // contenido del fichero a
   mostrar en el editor
```

```
editor.on("change", $scope.changefun);
```

que permiten deshabilitar el código asociado al evento `change` del editor.

deleteFile Para la eliminación de un fichero se necesita la confirmación del usuario. Se usa la función `modalConfig` pasándole la pregunta a mostrar y las funciones a ejecutar en caso de respuesta afirmativa y negativa. El método a ejecutar es `delete`, del recurso `File`, pasándole el nombre de la base de datos, el identificador de proyecto, la revisión del proyecto, el nombre del fichero y las funciones a ejecutar en caso de éxito y fracaso. Si el borrado se realiza con éxito, la referencia al fichero se borra de la variables `projectSelected.files` y `tabFiles` (en el caso de que el fichero se haya abierto para edición), se actualiza el atributo `projectSelected.rev` con el valor de la nueva revisión asociada al documento de proyecto en la base de datos y, por último, se activa la solapa de la consola si el fichero borrado estaba en la solapa activa.

closeTab Esta función permite cerrar la solapa asociada al fichero que contiene. Antes de cerrar la solapa se comprueba si el atributo `modified`, asociado al fichero dentro de la estructura `tabFiles`, está a `true`. Si es así, se solicita al usuario confirmación avisando de la eventual pérdida de los cambios realizados al cerrarla. Si el usuario confirma la operación, se eliminan el fichero de la estructura `tabFiles` y la solapa asociada al fichero. Si la solapa está activa (`tabFile.active=true` se llama a la función `activateTabConsole` para activar la consola.

activateTab Esta función es llamada cuando se quiere activar la solapa de un fichero abierto. Para ello se recorre la estructura `tabFiles`, se desactiva la propiedad `active` de todos los ficheros y se activa la propiedad del fichero que contiene dicha solapa.

activateTabConsole Esta función es llamada cuando se quiere activar la solapa de la consola. Para ello se recorre la estructura `tabFiles`, se desactiva la propiedad `active` de todos los ficheros, se activa la solapa de la consola y se hace visible el intérprete de la consola (`console-container`).

changefun Permite guardar el contenido de las modificaciones realizadas en el editor en el atributo `content` y activar la propiedad `modified` de la estructura `tabFiles` asociada al fichero activo. La función está asociada al evento `change` del editor CodeMirror.

sendCommand Esta función permite ejecutar el comando tecleado en la consola. Los parámetros que se han de pasar son el intérprete que se quiere ejecutar (ver 4.2.1.7), el comando a ejecutar por el intérprete y las funciones de éxito y error tras la respuesta. La ejecución del comando se realiza llamando al método `execute` del recurso `Shell`, pasándole el intérprete activo, orden, base de datos del usuario, nombre del proyecto y las funciones de éxito y error a ejecutar tras la respuesta.

B.4. Métodos del servicio `AuthService`

A todos los métodos desarrollados en este servicio se les deben pasar dos parámetros: las dos funciones a ejecutar en caso de éxito o error en la ejecución de la llamada HTTP. A estas funciones se les pasan los datos recibidos de la petición, en el caso de éxito los datos (`data`) y el estado de la petición (`status`) y en caso de error los datos de error (`error`) y el estado de la petición (`status`).

login A esta función se le pasan las credenciales, *email* y contraseña, necesarias para establecer una sesión válida en el servidor de base de datos CouchDB. La petición se realiza mediante el método `post` a la URL `/vizhaskell/db/_session`, pasándole el correo y contraseña. Si la sesión es creada con éxito se ejecuta la función `success`; si falla se ejecuta la función `error`.

logout La eliminación de la sesión en el servidor de base de datos se realiza ejecutando el método `delete` de la petición HTTP sobre la dirección `/vizhaskell/db/_session`. Si la sesión ha sido creada con éxito se ejecuta la función `success`, si no es así, se ejecuta la función `error`.

check Esta función comprueba si hay una sesión válida en el servidor de base de datos. Para ello se ejecuta el método `get` de la petición HTTP

sobre la dirección `/vizhaskell/db/_session`. Si la sesión se crea con éxito, se ejecuta la función `success` y, si no es así, la función `error`.

B.5. Métodos del servicio `Session`

create Crea el objeto de sesión a partir del correo electrónico y del perfil del usuario pasados. Establece el contenido de los atributos de la sesión: correo electrónico, perfil y base de datos.

destroy Destruye el objeto de sesión, es decir, limpia los atributos asociados a la sesión.

isAutenticated Indica si tenemos una sesión válida para el usuario conectado.

isAdmin Indica si el perfil del usuario conectado es administrador de aplicación.

isRoot Indica si el perfil del usuario conectado es el administrador de servidor.

getUsername Devuelve el nombre del usuario conectado.

getEmail Devuelve el correo electrónico asociado al usuario.

getDatabase Devuelve el nombre de la base de datos asociada al usuario conectado.

B.6. Detalle de los recursos del modelo

User A través de este recurso, el controlador `UserCtrl` realiza todas la operaciones sobre la base de datos de usuarios (`_users`). También permite recuperar todos los usuarios utilizando los parámetros `id=_all_docs`, `include_docs=true` y `skip=1` para que devuelva todos los documentos excepto el primero, ya que éste es el documento de diseño de autorización sobre la base de datos de usuarios.

Database Este recurso, también utilizado por el controlador `UserCtrl`, es el encargado de crear y borrar la base de datos asociada a cada usuario.

Project Este recurso permite realizar todas las operaciones de creación y borrado de proyectos en la base de datos del usuario. Cada vez que creamos un nuevo proyecto se crea un nuevo documento en la BD, por lo que es necesario recuperar el identificador de revisión del documento para posteriores operaciones con el recurso `File`, que gestiona las operaciones sobre los ficheros. Además, permite recuperar todos los proyectos y sus ficheros adjuntos. Para poder recuperar todos los datos de los documentos existentes en la base de datos es necesario indicar en la cadena de consulta los siguientes parámetros: `id=_all_docs`, `include_docs=true` y `attachments=true`.

File Como hemos dicho anteriormente, y dado que los ficheros son adjuntos al documento creado por el recurso `Project`, es necesario enviar en cada petición el identificador de revisión correcto del proyecto al que esta asociado el fichero. Si esto no se realiza, el servidor de base de datos nos responderá con un mensaje indicando que hay conflictos con el documento a modificar. Tanto para el proceso de alta como para el proceso de modificación se han creado creado tres acciones, una por cada tipo de archivo, ya que es necesario enviar en la cabecera de la petición HTTP el `Content-Type` correcto de cada tipo de fichero. En cuanto a la acción de recuperación (`get`), es necesario realizar una operación de transformación de la respuesta (`transformResponse`), ya que AngularJS interpreta todas las respuestas HTTP de tipo JSON.

Shell Este recurso es el encargado de realizar el proceso de restablecimiento de contraseña. Además, se encarga de realizar las peticiones de ejecución al servidor de las expresiones Haskell introducidas por consola.

B.7. Detalle de configuración de la aplicación

En el directorio `app/main` se encuentran los siguientes ficheros que configuran distintos aspectos de la aplicación.

app En el fichero `app.js` se definen el módulo AngularJS principal de la aplicación y las dependencias con los módulos que son necesarios para el

funcionamiento correcto:

- `ngRoute`: es un módulo proporcionado por AngularJS que permite configurar las rutas parciales para la correcta navegación por la aplicación.
- `dvizControllers`: es un módulo propio donde se encuentran definidos los controladores: `DvizCtrl` (ver 4.2.1.1), `ProjectsCtrl` (ver 4.2.1.3) y `UsersCtrl` (ver 4.2.1.2).
- `dvizServices`: módulo propio que define los servicios disponibles para la aplicación: `AuthService` y `Session` (ver 4.2.1.4).
- `dvizModels`: módulo propio que define todos los recursos de acceso a BD (ver 4.2.1.5).

dialog En el fichero `dialog.js` se han agrupado las funciones de acceso a los diálogos de aviso y confirmación. También gestiona la visualización del *spinner* para operaciones que pueden demorarse un tiempo considerable, como son la carga inicial de las páginas de la consola y la gestión de usuarios.

waiting Esta función es la encargada de mostrar u ocultar el objeto *spinner* en la página dependiendo del valor del parámetro `wait`. Cuando lo muestra, la página es bloqueada hasta que termina el procesamiento que invocó la función. Una vez finalizado el proceso se debe llamar a la función para ocultarla. Como medida de precaución, se ha establecido un tiempo de espera máximo, que permite liberar la aplicación en el caso de que no se llame a la función para ocultarla. Antes de mostrarse, establece los valores de posición para que aparezca centrada en el dispositivo sobre el que se está ejecutando la aplicación.

addNotify El objetivo de esta función es mostrar en la página el mensaje indicado en la llamada. Tiene tres parámetros: el mensaje a mostrar (`message`), el tipo de mensajes (`type`) y un tercer parámetro opcional que indica el tipo de error que se ha producido en una llamada al servidor (`r`).

El mensaje que se muestra depende de la disponibilidad de información en la causa del error, que es extraída heurísticamente desde la variable `r`. Configurado el mensaje, se muestra una ventana *modal* no bloqueante informando del error. Esta ventana puede ser cerrada, bien mediante la botón de cierre a la derecha del mensaje, o bien pulsando la tecla `escape` o pulsando fuera del área del mensaje.

modalConfirm Esta función mostrará una ventana *modal* mostrando el mensaje de confirmación y, dependiendo del botón pulsado, ejecutará una de las dos funciones pasadas por parámetro.

directives Para una mejor experiencia del usuario con la aplicación se han creado dos directivas[2] HTML5 a medida, que están definidas en el fichero `directives.js`.

vizAutofocus Se define como un atributo asociado a un objeto de formulario que reclamará el foco de forma automática cuando la página sea mostrada por primera vez.

vizTooltip Se trata de un atributo asociado a cualquier objeto del DOM (*Document Object Model*) que hará que aparezca un mensaje emergente al pasar el ratón por encima del mismo. Esta directiva ejecuta la función `tooltip` (definida en Bootstrap) asociada al elemento. Esta directiva es necesaria dado que los tiempos de refresco de AngularJS y Bootstrap no son síncronos. La facilidad de Boostrap no funciona correctamente sobre objetos que se muestran dentro de una directiva `ng-repeat` de AngularJS.

routes En el fichero `routes.js` se encuentran definidas todas las rutas necesarias para la navegación correcta por las páginas de la aplicación. Para ello, se configura el objeto `$routeProvider` y estableciendo la correspondencia entre rutas y páginas parciales que se han de mostrar, el título que debe aparecer en la cabecera del navegador al cambiar la ruta y el nombre del controlador en cada ruta, si lo utiliza. A continuación mostramos un ejemplo de ruta extraída de la aplicación:

[2]La directivas están pensadas para modificar o mejorar el comportamiento de los objetos HTML.

```
/* Configure app for routerProvider */
app.config(["$routeProvider",
  function($routeProvider) {
    $routeProvider.
    ...
     when("/console", {
        templateUrl: "view/console.html",
        controller: "ProjectsCtrl",
        title: "Consola"
      }).
    ...
      otherwise({
        redirectTo: "/about"
      });
}]);
```

Para que el título de cada una de las página aparezca en la cabecera del navegador es necesario supervisar el cambio de ruta y cambiarlo entonces:

```
app.run(['$rootScope', '$route', function($rootScope,
   $route) {
  $rootScope.$on('$routeChangeSuccess', function(rcs) {
    $rootScope.title = $route.current.title;
  });
}]);
```

console En el fichero `console.js` se encuentra la configuración del editor CodeMirror y de la consola Wterm.

Editor La configuración del editor CodeMirror se realiza con la variable `editor`:

```
1 var editor = CodeMirror.fromTextArea(
2   document.getElementById("txtCode"), {
3   lineNumbers: true,
4   matchBrackets: true,
5   autoCloseBrackets: true,
6   extraKeys : {
7     "Ctrl-S": function(instance) {
```

```
 8         var index = null;
 9         $.map($('#divProjects').scope().tabFiles, function(
              t,i){
10           if(t.active){
11             index = i;
12             t.content = instance.getValue();
13           }
14         });
15
16         if(index !== null)
17           $('#divProjects').scope().saveFile(index);
18         return false;
19       }
20     }
21 })
```

En la línea 2 se indica el nombre del elemento de la vista que gestiona el editor, en este caso el `textarea` `txtcode`. En las líneas 3 a la 5 se configura: que el editor muestre los números de línea (`lineNumbers`); que se marquen las correspondencias de aperturas y cierres de paréntesis, corchetes y llaves (`matchBrackets`, para esta opción es necesario incluir el fichero `matchbrackets.js`); y que cuando se abra un paréntesis, corchete o llave, también se cierre automáticamente (`autoCloseBrackets`, para esta opción es necesario incluir el fichero `closebrackets.js`). En las líneas 6 a 20 se configura el comportamiento de ciertas combinaciones de teclas. En este caso hemo incluido la combinación `Ctrl+S` para guardar el contenido del fichero (ver 4.2.1.3: `saveFile`).

Consola La configuración de la consola (personalización del componente *Wterm*) se realiza con la variable `ghcConsole`. Definimos el aspecto de la consola, el *prompt*, las clases CSS que se aplicarán a la consola y el tamaño.

También se definen aquí las órdenes que pueden ser ejecutadas dentro de la consola. Todas la expresiones Hashell son ejecutadas mediante la invocación de la función `ghcConsole.ajaxCommand`, que permite invocar al intérprete GHCi y mostrar el resultado en la consola.

Esta función recibe como parámetros la orden a ejecutar y los parámetros necesarios para su ejecución en el intérprete. La llamada al intérprete se realiza mediante la función `sendCommand` del controlador `dvizProjects` (ver 4.2.1.3: sendCommand). A esta función se le pasa el nombre del intérprete a ejecutar (`gheval`), la expresión a evaluar y las funciones de éxito o error. Si la ejecución ha producido algún error, éste es mostrado en la consola. En caso de éxito, se evalúa el JSON recibido:

- Si se recibe un `cmdResult`, su contenido es mostrado en forma textual.
- Si se recibe un `cmdGraph`, la representación de la expresión es gráfica. Dentro de esta respuesta pueden venir datos que permiten configurar el tamaño y estilo del gráfico a mostrar, todas estas características vienen definidas en el elemento `csss`. En este tipo de representación es importante que se declare el atributo `repType`, ya que indica el tipo de representación. Si no es indicado se mostrará un mensaje de error. Actualmente el único tipo soportado es `tree`.

Bibliografía

ANDERSON, C. J., LEHNARDT, J. y SLATER, N. *CouchDB: The Definitive Guide*. O'Reilly Media, Beijing; Cambridge [Mass.], 2010. ISBN 9780596155896.

BOINTON, M., PREVOST, A. y MATZELLE, B. R. PHPMailer: A full-Featured email creation and transfer class for PHP. http://github.com/PHPMailer/PHPMailer/, 2009. Accedido 08-06-2014.

BOSTOCK, M., OGIEVETSKY, V. y HEER, J. D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

COCHRAN, D. *Twitter Bootstrap Web Development How-To*. Packt Publishing, Limited, 2012. ISBN 9781849518833.

FELSING, D. *Visualization of Lazy Evaluation and Sharing*. Bachelor's thesis, Karlsruhe Institute of Technology, Germany, 2012.

GILL, A. Debugging Haskell by Observing Intermediate Data Structures. En *Proceedings of the 2000 ACM SIGPLAN Workshop on Haskell*. 2000.

HUDAK, P., HUGHES, J., PEYTON JONES, S. y WADLER, P. A History of Haskell: Being Lazy with Class. En *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, páginas 12–1–12–55. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-766-7.

IBORRA, J. y MARLOW, S. Examine your laziness. A lightweight procedural debugging technique for Haskell. Informe técnico, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2007.

JONES, P. y MARLOW, S. The Glorious Glasgow Haskell Compilation System User's Guide, Version 7.8.2. http://www.haskell.org/ghc/docs/latest/html/users_guide/index.html, 2007. Accedido 08-06-2014.

KNOL, A. *Dependency Injection with AngularJS*. Packt Publishing, 2013. ISBN 9781782166573.

MARLOW, S. y JONES, P. The Haskell 98 Report. The Revised Report (December de 2002). http://www.haskell.org/onlinereport/preface-jfp.html, 2002. Accedido 08-06-2014.

MILJENOVIC, I. y SEIPP, A. Vacuum-graphviz: A library for transforming vacuum graphs into GraphViz output. http://hackage.haskell.org/package/vacuum-graphviz, 2009. Accedido 08-06-2014.

MORROW, M. y SEIPP, A. Vacuum: visualising the GHC heap. http://thoughtpolice.github.io/vacuum/, 2009. Accedido 08-06-2014.

REINKE, C. GHood. Graphical Visualisation and Animation of Haskell Object Observations. En *Proceedings of the 2001 ACM SIGPLAN Workshop on Haskell*. 2001.

SCHMUECKER, R. D3.js Drag and Drop, Zoomable, Panning, Collapsible Tree with auto-sizing. https://gist.github.com/robschmuecker/7880033, 2013. Accedido 08-06-2014.

STEWART, D. Vacuum-cairo: Visualize live Haskell data structures using vacuum, graphviz and cairo. http://hackage.haskell.org/package/vacuum-cairo, 2009. Accedido 08-06-2014.

TEREI, D., MARLOW, S., PEYTON JONES, S. y MAZIÈRES, D. Safe Haskell. *SIGPLAN Not.*, vol. 47(12), páginas 137–148, 2012. ISSN 0362-1340.

Lista de acrónimos

AJAX . . . . . *Asynchronous JavaScript And XML*

API . . . . . *Application Programming Interface*, Interfaz de programación de aplicaciones

BD . . . . . Base de datos

BSD . . . . . *Berkeley Software Distribution*

CRUD . . . . . *Create-Read-Update-Delete*, Creación-Lectura-Actualización-Eliminación

CSS . . . . . *Cascading Style Sheets*

D3 . . . . . *Data-Driven Documents*

DOM . . . . . *Document Object Model*

GHC . . . . . *Glasgow Haskell Compiler*

GHCi . . . . . *GHC's interpreter*, Intérprete de GHC

HTML . . . . . *HyperText Markup Language*, Lenguaje de marcas de hipertexto

HTTP . . . . . *Hypertext Transfer Protocol*, Protocolo de transferencia de hipertexto

HTTPS . . . . . *Hypertext Transfer Protocol Secure*, Protocolo seguro de transferencia de hipertexto

HW . . . . . *hardware*, componentes electrónicos del ordenador

IDE . . . . . *Integrated Development Environment*, Entorno de desarrollo integrado

JSON . . . . . *JavaScript Object Notation*, Notación de objetos de JavaScript

LGPL2 . . . . . *Lesser General Public License Version 2*

LGPL3 . . . . . *Lesser General Public License Version 3*

MIT . . . . . *MIT, Massachusetts Institute of Technology*

MVC . . . . . *Model-View-Controller*, Modelo-Vista-Controlador

MVW . . . . . *Model-View-Whatever*, Modelo Vista y "lo que sea"

PHP . . . . . *PHP Hypertext Pre-processor*, recursivo de Preprocesador de Hipertexto PHP

PNG . . . . . *Portable Network Graphics*

REST . . . . . *REpresentational State Transfer*

SMTP . . . . . *Simple Mail Transfer Protocol*, Protocolo Simple de Transferencia de Correo

SO . . . . . *Sistema Operativo*

SOAP . . . . . *Simple Object Access Protocol*

SVG . . . . . *Scalable Vector Graphics*

SW . . . . . *software*, programas

URL . . . . . *Uniform Resource Locator*

XML . . . . . *Extensible Markup Language*, Lenguaje de marcas extensible