Proyecto de Software

Informe Trabajo Final

Grupo 02

Borrelli Franco Martin
Brost Pedro

Presentación del documento

Este documento se presenta como un informe sobre Trabajo de Final de Promoción del grupo 2 de la materia Proyecto de Software. El mismo se encuentra estructurado de acuerdo al índice que se encuentra a continuación:

Información del grupo

Los dos integrantes del grupo son:

- Borrelli Franco Martín (Legajo: 13726/5, DNI: 39831178)
- Brost Pedro (Legajo: 13775/5, DNI: 40289549)

El grupo no sufrió modificaciones respecto al grupo conformado durante la cursada. Como nota de cursada obtuvimos un 6.

Fundamentación sobre la elección del framework elegido.

Gracias a varios trabajos realizados en otras materias de la facultad ya contábamos con cierta experiencia trabajando con frameworks. En particular, trabajamos con el framework PHP Symfony[1] en Ingeniería de Software 2 y con Ruby on Rails[2] en el taller de Ruby.

Debido a este motivo queríamos optar por hacer algo diferente. Aprovechar la oportunidad que este trabajo proporcionaba para aprender algo en lo que teníamos poco conocimiento.
Fue así como decidimos profundizar nuestro conocimientos en Javascript, y las buenas prácticas de su utilización tanto del lado del cliente, como del lado del servidor (Node.js). Javascript hoy en día ha cobrado gran protagonismo, siendo una de las tecnología más utilizadas en aquellos proyectos que buscan innovar.

Decidimos enfatizar también en un concepto nuevo para nosotros, el de las SPA (Single Page Applications). Si bien este concepto se mencionó brevemente durante la cursada, nos pareció la oportunidad perfecta para ver como es la dinámica de su desarrollo en comparación a la MPA (Multiple Page Application) que desarrollamos durante la cursada. A su vez gracias a esta decisión pudimos poner en práctica otros conceptos vistos tales como request de tipo AJAX y manejo del DOM.

De acuerdo a todo lo antes mencionados, desarrollamos dos proyectos por separado: una para el lado del cliente, utilizando la librería **React** y otra para el servidor, utilizando el framework **Express** para el desarrollo de una API Rest.

React es una librería javascript para la creación de interfaces de usuario desarrollada por Facebook. Hoy en día se posiciona dentro de las tecnologías más utilizadas para el desarrollo de SPA, junto a otras como Vue.js y Angular.

Por el lado del servidor decidimos utilizar Express ya que es uno de los frameworks javascript más utilizados actualmente, contando un amplia comunidad. Express cuenta con una documentación que facilita su aprendizaje de manera rápida y sencilla. Este framework tiene la capacidad de poder utilizar de manera sencilla otras librerías en conjunto que aportan funcionalidades más específicas. Entre estas podemos encontrar librerías que permiten la utilización de cors, de operaciones de log, de debug y de health-check.

[1] http://symfony.es
[2] https://rubyonrails.org

Para la base de datos, se decidió utilizar MongoDB por su facil integración con las tecnologías javascript antes mencionadas. Esto se debe a que Mongo no es una base de datos relacional, sino una no-SQL basada en documentos.

Una gran ventaja de la elección de este stack denominado MERN (Mongo - Express - React - Node) es que requiere el aprendizaje de un único lenguaje de programación.

Descripción de los módulos desarrollados en el trabajo de la cursada que pudieron ser aprovechados para ser usados por el framework.

El módulo de reportes de usuario pudo adaptarse casi totalmente con facilidad. Esto se debe a que ciertas funcionalidades relacionadas con este módulo como por ejemplo la forma en que los gráficos se generan y los mecanismos de exportación en formato pdf, ya se resolvian con Javascript mediante la utilización de la librería HighCharts.

En términos generales también se adaptaron aspectos relacionados con la vista, tales como la disposición de las diferentes pantallas que componen el sitio y los elementos que se presentan en cada una de estas (como mostrar la información, donde poner botones, que información útil es importante agregar para el usuario). El código HTML y CSS utilizado durante el proyecto de la cursada no pudo reutilizarse completamente debido al cambio que implicó pasar de Bootstrap a Ant.Design. Pantallas tales como la pantalla principal, la página del login y las pantallas de error y mantenimiento se mantuvieron completamente.

A lo largo del desarrollo de este proyecto pudimos también observar ciertas similitudes entre Twig y React a la hora de buscar modularización del código. De la misma forma que en Twig usábamos templates base que son utilizados por otros templates más específicos, en React podemos definir componentes genéricos que pueden albergar otros componentes.

Un ejemplo claro de esto, es lo realizamos en el archivo baseLayout.js en la carpeta “containers” en “client”. Este archivo es utilizado en todas las pantallas para los elementos comunes tales como el footer.

Adicionalmente, varias macros de Twig que habíamos realizado en la cursada, como por ejemplo una macro de artículos (para la página de inicio del sitio), pudieron ser adaptadas fácilmente en componentes de React, donde los parámetros que la macro recibe se convierten ahora en propiedades del componente.

Más allá de estas cuestiones mencionadas, pasar de PHP a Javascript conllevo tener que arrancar casi desde cero en lo referido a las funcionalidades realizadas del lado del backend.

El mecanismo provisto para el manejo de seguridad y routing

Routing

Cliente

La manera de realizar routing dentro de una SPA es muy distinto a como se trabaja en una página web tradicional (MPA). Mientras que estas últimas suponen un recargo de página ante cada cambio de ruta, en una SPA esto se maneja desde el lado del cliente, siempre actualizando la página actual. En particular, para React existe una librería independiente denominada React Router que facilita los mecanismos de ruteo. Lo que destaca a React Router es que provee un mecanismo de ruteo dinámico, es decir que el ruteo se lleva a cabo mientras la aplicación se renderiza en lugar de utilizar por ejemplo un archivo de configuración.

Servidor

Para la realización de los distintos endpoints que conforman a la API REST desarrollada, se utilizaron las herramientas que provee Express para la realización de rutas haciendo uso de distintos middlewares los cuales son uno de los principales conceptos dentro de Express.

Seguridad

Autenticación

Para la implementación de la autenticación no se utilizaron los mecanismos tradicionales, es decir mediante el uso de cookies y sesiones. Por lo contrario, se utilizó una tecnología denominada JWT (JSON Web Tokens) la cuál es la que se suele elegir a la hora de realizar una SPA. Esta técnica permite que ante el logueo de un usuario se cree un token firmado del lado del servidor el cual le servirá al cliente como evidencia de que es quien realmente dice ser. Esto se logra enviando el JWT al cliente, el cual lo almacenará y enviará a través de headers en todos los requests que requieran autenticación.

Autorización

Una de las particularidades de los JWT es que no pueden ser modificados por el cliente. Esto permite almacenar dentro de ellos cierta información relacionada con el usuario logueado. Dentro de esta podemos encontrar los distintos roles y permisos que el usuario posee. De esta manera, es fácil determinar para la aplicación si el usuario puede realizar ciertas operaciones. Hay casos en los que igualmente es necesario acceder a la base de datos para verificar los permisos que tiene un usuario en un determinado momento.

XSS (cross-site-scripting)

Uno de los principales beneficios de utilizar React es que provee mecanismos que permiten evitar la ocurrencia de XSS. React garantiza que nunca se podrá inyectar nada

que no esté escrito explícitamente en la aplicación. Todo valor pasado a un elemento JSX se convierte en un string antes de ser renderizado.

El mecanismo provisto para operaciones CRUD

Express al ser un framework muy simple, en sí mismo no posee ningún mecanismo para la automatización de la generación de operaciones CRUD. Existen ciertas librerías independientes complementarias que proveen esta funcionalidad pero optamos por no utilizar ninguna de ellas. Más allá de esto, se decidió utilizar una denominada “boilerplate application”, la cual es una aplicación de referencia que sirve para comenzar a desarrollar de manera fácil y sencilla siguiendo buenas prácticas.

La forma de manejar el MVC

La página web no fue desarrollada utilizando un framework tradicional como puede ser Symfony, Laravel, Ruby on Rails, que permiten al desarrollador seguir con el patrón arquitectural MVC con facilidad. Estos frameworks normalmente abarcan las tres partes de MVC, y proveen una serie de convenciones y buenas prácticas para aplicar bien estos conceptos.

A diferencia de estos frameworks, el stack de tecnologías que se elegidos para la realización de este proyecto fue muy distinto. Al tratarse de una SPA desarrollada principalmente con React y Express, no es tan fácil explicar una manera en la que se “manejo” el MVC.

Por el lado de React, la misma documentación[3] oficial de React explica que este no es un framework MVC sino una librería para la construcción de interfaces de usuario. Vale aclarar que en sus principios se denominaba a React como la V en MVC. Esto fue cambiando aunque hoy en día sigue habiendo ciertos debates sobre si esto es cierto o no.

Por otro lado, en el desarrollo de la API REST desarrollada con Express tal vez es más sencillo ver la M y la C de MVC. Se definieron muy explícitamente cada modelo y sus respectivos controladores. Vale aclarar que al utilizarse una base de datos documental no sql como lo es MongoDB, tal vez no era necesario la definición explícita de modelos, debido a que este tipo de base de datos carecen de un esquema estático y bien definido. Para facilitar algunos aspectos del desarrollo, en cambio, se decidió utilizar una librería llamada Mongoose que justamente permite definir unos “esquemas” que nos brindaran ciertas facilidades.

Árbol de directorios

Se presenta a continuación el árbol de directorio dentro de la carpeta “final”, donde se encuentra el código desarrollado durante esta última etapa.

[3] https://reactjs.org/blog/2013/06/05/why-react.html

Client

Si bien partimos de la estructura provista por create-react-app (ver en material de referencia), definir la estructura en que queríamos organizar nuestro código no fue fácil. La documentación oficial de React no ofrece referencia alguna sobre la forma correcta en que se debe estructurar el código.

Luego de leer varias fuentes de información decidimos optar por seguir el enfoque presentado en un artículo de Medium. El mismo se encuentra más abajo con el material de referencia.

De esta forma definimos entonces:

- **public**: almacena el único archivo html del sitio junto con el favicon.

- **src:** Es el directorio más importante dentro de cliente. Es el lugar donde se encuentra todo nuestro código (archivos js y scss). El mismo se encuentra subdividido en distintas carpetas:
    - **components:** Se almacenan archivos que definen diferentes componentes reutilizables que pueden ser usados globalmente en las diferentes pantallas que componen la aplicación.

    - **containers:** Los contenedores son un tipo particular de componentes en React. Estos generalmente se encargan de la obtención de información mediante request o algún otro tipo de lógica para luego renderizar sus diferentes componentes hijos. Justamente se llaman así, porque contienen otros componentes. En esta carpeta se puede encontrar por ejemplo el Layout base para todas las páginas del sitio.

    - **hoc:** Almacena los hoc o “High-Order Components” comunes. Un hoc no es más que una técnica para la reutilización de lógica de componentes. Se lo puede, en su forma más general, como una función que recibe un componente para retornar otro. En nuestro caso entre los hocs que definimos se encuentra uno que se encarga del chequeo de permisos de usuario, devolviendo el componente que representa la pantalla o el componente de pantalla de error, de acuerdo a lo que corresponda.

    - **scenes**: Un “escenario” se presenta como una página de la aplicación. Nosotros por agrupar estos escenarios de acuerdo al módulo correspondiente. Así, contamos con la carpeta tales como “users” (que engloba todas las pantallas relacionadas a estos), “errors” (para las pantallas de error), “configuration”, “login” y “patients”.
      Dentro de muchas de estas carpetas definimos también las carpetas “components”, “hoc” y “containers” para almacenar el código reutilizable entre los componentes comunes a un mismo módulo.

- **store:** Posee el código asociado al manejo de Redux, librería que sirve para el desacoplamiento del estado de los distintos componentes. La carpeta de divide en “actions” y “reducers”. Una acción es simplemente un objeto JavaScript que incluye al menos un atributo type que indica el tipo de acción que estamos emitiendo. Las acciones describen que algo pasó, pero no especifican cómo cambiar el estado de la aplicación. Esto es trabajo de los reducers.

- **index.js:** Almacena nuestra llamada principal de Render de ReactDOM. Importa nuestro componente App.js con el que comenzamos y le dice a React dónde renderizarlo.

- **axios-api.js y axios-apiReferencias.js**: Archivos de configuración de axios para lograr la comunicación con la api desarrollada y con la api de referencias provista por la cátedra.

- **package.json:** Para gestionar nuestras dependencias, también usado para las reglas babel.

Server

Al ser Express un framework muy sencillo, este no provee un árbol de directorios estricto que se debe respetar. En cambio, el código que representa por la aplicación deberá ser estructurado según sea conveniente para los desarrolladores. Más allá de esto, existen buenas prácticas que se pueden seguir tal como se implementó en el desarrollo de la página.

- **config**: Distintos archivos de configuración para la base de datos, servidor, logger, validador, etc.

- **seeds**: Archivos de seeds para la base de datos.

- **src**: Código fuente de la aplicación. Se optó por agrupar en carpetas los modelos y controladores que corresponden a una misma entidad, así como también las rutas que definen los endpoints. En este directorio también se encuentra el archivo principal de la aplicación (“app.js”) y el index de las rutas. También se puede encontrar un directorio denominado “helpers” en el cual se encuentran algunas utilidades que comparten los distintos controladores.

- **package.json:** definición de dependencias, scripts y metadatos.

- **index.js:** punto de entrada de la aplicación. Aquí se inicia el servidor web.

En el directorio raíz además se pueden encontrar distintos archivos para la configuración de linters, editores de texto, gitignore y seeds. Además se encuentran los archivos correspondientes para iniciar la aplicación utilizando Docker.

Material de Referencia (libros, tutoriales, cursos, etc).

React:

- Documentación oficial de React : www.reactjs.org/docs
- Curso de React de Udemy: www.udemy.com/react-the-complete-guide-incl-redux
- Doc. de React Router: www.reacttraining.com/react-router/core/guides/philosophy
- Documentación oficial de Ant.Design: www.ant.design
- Documentación de gráficos HighChart: www.github.com/whawker/react-jsx-highcharts
- Documentación de Redux: www.redux.js.org/introduction
- Documentación de Axios: www.github.com/axios/axios
- Create-react-app: www.github.com/facebook/create-react-app
- How to better organize your React applications: www.medium.com/@alexmngn/how-to-better-organize-your-react-applications-2fd3ea1920f1

Express:

- Página Oficial de Express: www.expressjs.com
- Boilerplate Application: www.github.com/kunalkapadia/express-mongoose-es6-rest-api
- Documentación de Mongoose: www.mongoosejs.com
- Documentación de Docker: www.docs.docker.com