Informe de Desarrollo

Lucas Di Cunzolo[1] y Ulises J. Cornejo Fandos[2]

***Abstract*— En este documento se plantean las tecnologías utilizadas para el desarrollo del sistema, así como los fundamentos de su elección. Se busca plantear al lector la razón de cada una de las cuestiones presentes en este desarrollo tales como referencias, descripción de módulos, mecanismos de seguridad y routing, mecanismos provistos para operaciones CRUD, manejo de patrones de arquitecturas y demás cuestiones técnicas del desarrollo.**

I. INTRODUCCIÓN

En las siguientes secciones se desarrollan algunos de los aspectos referidos al desarrollo del sistema para el Hospital de Niños Ricardo Gutierrez. El mismo detalla algunos de los temas fundamentales en relación al desarrollo y elecciones tomadas durante la ejecución del mismo.

A su vez, se pueden encontrar distintos enfoques y mayor documentación sobre ambientes y tecnologías utilizadas, tanto en las wikis del repositorio como en la ***documentación*** del proyecto.

II. ELECCIÓN DE TECNOLOGÍAS

A lo largo de las siguientes subsecciones se debate sobre cada una de las tecnologías utilizadas, fundamentando la elección de cada una de ellas.

Entre los aspectos que se tuvieron en cuenta se destaca la evaluación de benchmarks de los lenguajes a utilizar, el respaldo de la comunidad; diversidad, ventajas y desventajas de frameworks disponibles, comodidad por parte de los desarrolladores, entre otros.

Se sabe que entre los lenguajes de programación destinados al desarrollo web contamos con aquellos que son conocidos por su rapidez al momento de atender pedidos por parte de un cliente, como otros destacados por su potencial al momento de actuar ante una gran cantidad de pedidos en un corto período de tiempo. Para este caso, se busca lograr un punto medio entre los aspectos mencionados, logrando una alta disponibilidad con una cantidad media de accesos.

A. Lenguajes de Programación

El desarrollo dispuesto a lo largo de la cursada tiene lugar a partir de la utilización del lenguaje PHP. Seguramente, y dada la escalabilidad del código resultante, lo más simple sería migrar el mismo a algún framework PHP compatible. Para que esto fuese posible, se desarrolla un framework con la idea de obtener una interfaz similar a muchas de las posibilidades para que la migración resulte rápida y sencilla.

Sin embargo, se sabe que al momento de un desarrollo, el framework a utilizar no debe ser la primer elección respecto de las tecnologías resultantes. Para esto comenzamos evaluando las distintas alternativas disponibles en cuanto a lenguajes de programación.

- ***PHP***, es una de las opciones más viables al momento de pensar en el desarrollo de esta aplicación. La escritura de código en este lenguaje es muy rápida y la gran cantidad de frameworks PHP existentes simplifican mucho el trabajo. Claramente es una excelente opción siempre y cuando se garantice flexibilidad en las técnologías y la posibilidad de utilizar algunas de las últimas versiones estables de la rama de PHP7. Es importante destacar esta necesidad dada la gran diferencia de performance existente entre esta rama del lenguaje y las tecnologias soportadas por la rama PHP5. Teniendo en cuenta esto, PHP pasa a ser una opción viable en un acotado conjunto de posibilidades.

- ***Ruby***, es un lenguaje super amigable para los desarrolladores, con una estadística de benchmarks entre media y lenta, y con un amplio arsenal de frameworks diferentes. Sin embargo, haciendo una comparación rápida entre las últimas versiones estables de PHP7 y ruby, la performance del primero sigue siendo mayor que la de ruby, por lo que se opta por evaluar otras posibilidades.

- ***Python***, es una posibilidad que se ganó mucha fama dentro del desarrollo web en los últimos tiempos. Al igual que ruby, no le hace frente a las posibilidades que PHP provee. De igual forma, ninguno de los integrantes del grupo se sienten cómodos programando en python, por lo que se descarta rapidamente.

- ***Java***, es un lenguaje muy potente y una muy buena opción al momento de plantear el lenguaje a utilizar en el backend de una aplicación. Es una opción muy viable la convención entre la utilización de Java en conjunto con algún framework MVC como spring para el backend y algún framework como AngularJS del lado del cliente para obetener como resultado una aplicación muy potente preparada para resolver un conjunto muy grande de peticiones. Java es un lenguaje que supera en performance a PHP7

[1]Estudiante de la carrera Lic. en Informática, UNLP
[2]Estudiante de la carrera Lic. en Informática, UNLP

en un gran número de casos de prueba, siendo así tambien mucho más potente que todos los casos mencionados anteriormente. Sin embargo, a pesar de esto, Java se muestra lento en casos determinados los cuales complican la performance de una aplicación. La estadística de benchmarks lo muestran como un lenguaje lento. Teniendo en cuenta esto es que se evalúa la presencia de otras tecnologías sin descartar a priori la posibilidad de utilizar java en el desarrollo del backend.

- ***Javascript***, cuenta con ventajas que ninguno de los otros lenguajes podría ofrecer. Por un lado, la posibilidad de una aplicación implementada completamente en JS. Esto es posible gracias a node.js permitiendo así que tanto el cliente como el servidor sean implementados utilizando este lenguaje. Claramente Java es una mejor opción para proyectos mucho más grandes y dado que este no es el caso, las ventajas que provee JS se adaptan mucho más a las necesidades actuales. JS cuenta con una gran variedad de frameworks y un gran soporte por parte de la comunidad. Soporte nativo de JSON para un manejo de datos más simple, permitiendo una buena comunicación entre el cliente y el servidor, y demás detalles que hacen que JS sea la alternativa más viable. A su vez, todos los integrantes del grupo de desarrollo tienen experiencia programando en este lenguaje, por lo que resulta la opción más cómoda.

  Es por esto que Javascript es la opción elegida para el desarrollo de este proyecto.

B. Frameworks y librerías

Una vez determinados los lenguajes de programación a utilizar, se definen los frameworks que permitan acelerar y simplificar el proceso del desarrollo.

Para esto se evalúa distintas tecnologías y, teniendo en cuenta la similitud entre cada una de ellas, no se va a profundizar mucho respecto de aquellas que son descartadas, sinó en destacar las ventajas de aquellas que son seleccionadas.

En primer instancia se piensa en una aplicación MEAN Stack *(Mongo, Express, Angular, Node.js)*. Una aplicación de este estilo abarca absolutamente todos los aspectos de la aplicación logrando una gran compatibilidad entre cada una de las tecnologías, y una gran facilidad de código.

A lo largo de la discusión planteada se evalúa la posibilidad de utilizar alguna otra librería para el desarrollo del cliente, planteando así la utilización de ReactJS. React cuenta con muchas similitudes respecto de Angular, pero con una gran ventaja en su implementación. A diferencia de Angular, React no es un framework sinó una librería, y esto permite mayor flexibilidad al momento de definir la distribución del código así como también mayores libertares al momento de controlar el flujo de los datos. Dado que no se limita a una simple aplicación MVC del lado del cliente, se pueden jugar con otras formar de definir patrones de diseño de la aplicación que permitan un ahorro mayor de almacenamiento de datos, consultas a las APIs, entre otros aspectos. Además, la utilización más eficiente de un DOM Virtual le permite a ReactJS un gran incremento en la performance al momento de trabajar sobre los datos del DOM. React no trabaja directamente sobre el DOM, reduciendo así el tiempo de procesamiento en lectura y escritura, entre otros aspectos. Así mismo, React permite un manejo más eficiente de mayores cantidades de datos a partir de su flojo de ejecución unidireccional.

Posteriormente, se toma en cuenta la posibilidad de utilizar Firebase en lugar de MongoDB. Firebase cuenta con ciertas ventajas para su utilización dada la API que el mismo proveé, pero MongoDB permite un mayor control sobre el manejo de datos, así como un potente lenguaje de consultas.

No habiendo discuciones sobre la utilización de Node.js y Express del lado del servidor, se obtiene como resultado final una aplicación MERN Stack *(MongoDB, Express, ReactJS, Redux, NodeJS y Webpack)*.

Por un lado, Redux permite un manejo limpio y cómodo del estado interno de la aplicación. Redux es un contenedor predecible del estado de aplicaciones JavaScript.

Permite a escribir aplicaciones que se comportan de manera consistente, corren en distintos ambientes (cliente, servidor y nativo), y son fáciles de probar. Además de eso, provee una gran experiencia de desarrollo, gracias a edición en vivo combinado con un depurador sobre una línea de tiempo.

Puedes usar Redux combinado con React, o cual cualquier otra librería de vistas. Es muy pequeño (2kB) y no tiene dependencias.

Por otro lado, Webpack es un module bundler (empaquetador de módulos) muy eficiente para aplicaciones grandes que contienen mucho código JavaScript.

Su utilidad reside en la fragmentación de código: no todas las partes de una webapp requieren todo el código JavaScript, por eso se encarga de cargar sólo las partes necesarias en cada petición. Además, funciona con un gran número de lenguajes de plantilla y preprocesadores de JavaScript (TypeScript o CoffeeScript, ReactJS…) y CSS (LESS, SASS…). Para que importar diferentes componentes sea sencillo e intuitivo, Webpack implementa el ya estandarizado RequireJS para la inyección de dependencias de nuestra aplicación.

Otro de los puntos fuertes de Webpack es que dispone de loaders que facilitan el compilado de código de otros

lenguajes a JavaScript nativo, la minificación del mismo o incluso la transpilación de estándares de EcmaScript todavía no disponibles ampliamente (como ES6) a código que todos los navegadores puedan leer. Esto es posible gracias a que existe un loader para Webpack de Babel, el transpilador de código JavaScript por excelencia.

C. Infraestructura

El objetivo principal de este proyecto es brindar un sistema sencillo de gestionar en cada uno de los ambientes, ya sea en desarrollo o en producción. Para esto se proveé un método simple de instalación de dependencias, buscando trabajar siempre sobre las mismas versiones de cada una de ellas en diversos ambientes de trabajo, como también en el servidor.

Para esto se utiliza Docker, un manejador de containers ligeros y portables. Con docker, todos los integrantes del grupo de desarrollo pueden manejar el mismo ambiente replicado.

A su vez, en ambiente de producción, Docker permite desplegar facilmente el sistema y escalarlo para aumentar su disponibilidad, de ser necesario.

Dadas las diferentes opciones de lenguajes de programación, se evalúa la posibilidad de dockerizar un servicio para cada uno de ellos antes de elegir las tecnologías. En principio, dockerizar un servicio en cada uno de los lenguajes evaluados no presenta mayores problemas.

Teniendo en cuenta que la aplicación desarrollada durante la cursada se implementa utilizando un ambiente dockerizado, utilizar las herramientas ya desarrolladas y el lenguaje PHP, parece la opción más simple.

Una vez seleccionadas las tecnologías, se investiga como trabajan cada una de ellas con docker, llegando a la siguiente configuración:

- **Mongo**, cuenta con una imagen oficial de Docker en Dockerhub. Es por esto que se utiliza esta imagen por mayor comodidad.
- **Express, React, Node**, para esto se extiende la imagen oficial de node, agregando express desde la plantilla de *mern-text*.
- **Mongo Express**, se utiliza además la imagen oficial de mongo express para un manejo más amigable de la base de datos.
- **Redis**, se utiliza la imagen oficial de dockerhub para esta herramienta necesaria para el manejo de sesiones del bot de telegram.

Una vez armado el stack a utilizar, se configura utilizando *docker-compose*, para tener de esta manera un ambiente ya acoplado y facil de levantar.

Finalmente se arman 2 *Dockerfile* y 2 *docker-compose*, uno para desarrollo y otro para producción, con sutiles diferencias.

De contar con gitlab ci/cd, ya se cuenta con un archivo .gitlab-ci, el cual autogenera y pushea las imágenes docker en la registry privada del repositorio, simplemente tageando con una versión semántica, *x.y.z*.

Esto tiene como resultado un simple tag de git, todo un posible ambiente productivo listo para ser descargado desde las diferentes registries, para ser utilizado en producción.

III. Estado previo

Dada la elección de cambiar completamente las tecnologías en el desarrollo de la app, se concidera la migración de todos los módulos desarrollados anteriormente a javascript. Para esto se opta por migrar cada uno de los módulos así como tambien las APIs de referencia, para ofrecer un conjunto centralizado de servicios.

IV. Seguridad y Routing

En esta sección se desarrollan todos los aspectos referidos a las técnicas de seguridad y routing empleadas para el desarrollo del sistema.

A. Seguridad

La aplicación presente emplea express.js del lado del servidor para definir el comportamiento de la app ante una request dada. Luego, se definen distintas estrategias de seguridad utilizando librerias que interactuan exitosamente con express y así obtener una aplicación segura de forma rápida y sencilla.

Para esto se tienen en cuenta los siguiente puntos básicos:

- **No utilizar versiones en desuso o vulnerables de Express**

  Express 2.x y 3.x no se encuentran en mantenimiento. Los problemas de seguridad y rendimiento en estas versiones no se solucionarán por lo que no es recomendable su utilización. Se utiliza entonces la versión 4 de esta libreria.

  Asimismo, se evalúa no estar utilizando ninguna de las versiones vulnerables de Express que se listan en la página Actualizaciones de seguridad.

- **HelmetJS**

  Helmet ayuda a proteger la aplicación de algunas vulnerabilidades web conocidas mediante el establecimiento correcto de cabeceras HTTP.

  Helmet es realmente una colección de nueve funciones de middleware más paquetes que establecen cabeceras HTTP relacionadas con la seguridad:

  - **csp** establece la cabecera Content-Security-Policy para evitar ataques de scripts entre sitios y otras inyecciones entre sitios.
  - **hidePoweredBy** elimina la cabecera X-Powered-By.
  - **hpkp** añade cabeceras Public Key Pinning para evitar ataques de intermediarios con certificados falsos.
  - **hsts** establece la cabecera Strict-Transport-Security que fuerza conexiones seguras (HTTP sobre SSL/TLS) con el servidor.
  - **ieNoOpen** establece X-Download-Options para IE8+.
  - **noCache** establece cabeceras Cache-Control y Pragma para inhabilitar el almacenamiento en memoria caché del lado de cliente.
  - **noSnif**f establece X-Content-Type-Options para evitar que los navegadores rastreen mediante MIME una respuesta del tipo de contenido declarado.
  - **frameguard** establece la cabecera X-Frame-Options para proporcionar protección contra el clickjacking.
  - **xssFilter** establece X-XSS-Protection para habilitar el filtro de scripts entre sitios (XSS) en los navegadores web más recientes.

- **Otras consideraciones**

  Se toman en cuenta otras consideraciones al momento de definir aspectos básicos de seguridad como pueden ser la utilización de nsp y requireSafe para evaluar la seguridad de las dependencias del sistema, buena configuración y utilización de cookies, definiendo siempre flags de configuración de express-session tales como secure, httpOnly, entre otros.

  A su vez, se evalúa que cada una de las expresiones regulares utilizadas no sean susceptibles de ataques de denegación de servicio de expresiones regulares, generando validaciones limpias y consistentes tanto del lado del cliente como del lado del servidor.

B. Routing

Express permite y facilita definir el comportamiento de la aplicación js ante una ruta dada, por lo que resulta sencillo definir distintas funcionalidades asociadas a una ruta. Acelera la definición de puntos finales de aplicación (URI) y cómo responden cada uno de ellos a las solicitudes del cliente.

Express da soporte a los siguientes métodos de direccionamiento que se corresponden con los métodos HTTP: get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search y connect.

- **Via de acceso de ruta**

  Las vías de acceso de ruta, en combinación con un método de solicitud, definen los puntos finales en los que pueden realizarse las solicitudes. Las vías de acceso de ruta pueden ser series, patrones de serie o expresiones regulares.

  Express utiliza path-to-regexp para correlacionar las vías de acceso de ruta; consulte la documentación de path-to-regexp para ver todas las posibilidades para definir vías de acceso de ruta. Express Route Tester es una herramienta muy útil para probar rutas básicas de Express, aunque no da soporte a la coincidencia de patrones.

  Las series de consulta no forman parte de la vía de acceso de ruta.

- **Manejadores de rutas**

  Puede proporcionar varias funciones de devolución de llamada que se comportan como middleware para manejar una solicitud. La única excepción es que estas devoluciones de llamada pueden invocar next('route') para omitir el resto de las devoluciones de llamada de ruta. Se utiliza este mecanismo para imponer condiciones previas en una ruta y, a continuación, pasar el control a las rutas posteriores si no hay motivo para continuar con la ruta actual.

  Los manejadores de rutas pueden tener la forma de una función, una matriz de funciones o combinaciones de ambas, como se muestra en los siguientes ejemplos.

- **express.Router**

  Se utiliza la clase express.Router para crear manejadores de rutas montables y modulares. Una instancia Router es un sistema de middleware y direccionamiento completo; por este motivo, a menudo se conoce como una "miniaplicación".

V. CRUD

Se probeé una API REST para facilitar acceso al sistema de CRUD de cada una de las entidades presentes en el sistema. Para esto se utiliza express, mencionado anteriormente el sistema de *routing*, y mongoose para la definición de modelos.

VI. Arquitectura de la aplicación

El sistema en cuestión se plantea como una aplicación cliente-servidor, definiendo mediante una API REST el acceso de cada uno de los recursos para desacoplar casi completamente la aplicación del lado del cliente y del servidor. Si bien se utiliza *server side rendering* para el renderizado del frontend, se busca que todo lo referido a control y persistencia de datos quede completamente del lado del servidor.

A. Servidor

Del lado del servidor se definen unicamente *Controllers* y *Models*, con el fin de definir de forma sencilla las URIs a utilizar en la definición de la API.

Como se menciona anteriormente, se utiliza *mongoose* para definir los esquemas de las entidades del modelo de base de datos no relacional, y express para la definición de las rutas. Luego, los controladores pasan a ser funciones exportadas en modulos a partir de la utilización de nuevos estandares de ECMAScript, haciendo uso de la porgramación basada en módulos.

Respecto del servidor, el código del mismo se encuentra en */server* y los directorios que contiene son los siguientes.

- */config*, en este directio se define todo lo referido a configuración del servidor.
- */controllers*, en este directorio se definen los controladores de la aplicación. Son archivos de la forma *<resource>-controller.js*. En cada uno de ellos se exportan funcionalidades especificas para ser utilizadas en la definición de las rutas de la API.
- */models*, en este directorio se definen los modelos de la aplicación. Los mismos son de la forma $<resource>\ .js$ y exportan la definición de los esquemas generados utilizando *mongoose*.
- */modules*, en este directorio se definen los distintos módulos de la aplicación como middlewares de configuración y validación de datos, autentificación de usuarios, chequeo de permisos, entre otros.
- */routes*, en este directorio se definen las distintas rutas así como el comportamiento de la API sobre cada una de ellas.
- ...

B. Cliente

Respecto del cliete de la aplicación, es natural pensar que todo lo definido en este lado del desarrollo corresponda a la V en MVC. Sin embargo, el lado del cliente se aleja de lo que es el flujo de ejecución de una aplicación MVC, y es esto mismo lo que la hace distinta.

Como se menciona anteriormente, ReactJs tiene una forma particular de manejar los datos, particularmente una forma poco común de manejar el estado interno de cada una de sus componentes así como también el flujo de los datos y los cambios efectuados al estado interno de la aplicación.

El problema con una estructura MVC es la comunicación bidireccional. No sería viable para una aplicación de este estilo pensar en la actualización dinámica del estado interno de las componentes si tuviese que actualizarse en cascada cada cambio efectuado a una entidad en el servidor. Es por esto que Facebook crea una arquitectura denominada *Flux*.

Flux es una arquitectura compuesta de 4 componentes o herramientas,

- *Actions*, objetos con propiedades y datos.
- *Stores*, contenedores del estado y la lógica de la aplicación.
- *The Dispatcher*, procesa *actions* y *callbacks* registradas.
- *Views*, escuchan los cambios de los *stores* y se re-renderizan a si mismos.

Ahora las diferencias con un MVC son que el flujo de procesamiento es unidireccional en lugar de bidireccional, las stores pueden almacenar cualquier estado relacionado con la aplicación, mientras que el modelo en MVC fue diseñado para almacenar objetos individuales, y el punto de inicio *Dispatcher* hace que la depuración sea mucho más sencilla.

Nuevamente, no conformes con este cambio de arquitectura, llega otra variante. Redux es un predecible contenedor para el estado interno de las aplicaciones JS. En conjunto con React permite crear una aplicación que aproyeche todas las ventajas que permite Flux y a su vez tenga un buen manejo del estado interno de la app. Redux es una librería creada para trabajar sobre Flux y que se apoya en 3 principios:

- ***Solo una fuente de verdad***, el estado completo de la aplicación se almacena en una unica store.

- ***El estado es de solo lectura***, la única forma de cambiar el estado es emitir una acción (un objeto que describe lo que sucedió).

- ***Los cambios se realizan unicamente con funciones***, se especifica la transformación por acciones con reductores, que permiten navegar por los estados.

Luego, las diferencias con Flux son las siguiente:

- ***Redux no tiene el concepto de dispatcher*** porque depende de funciones puras en lugar de emisores de eventos.

- ***Redux supone que nunca muta sus datos.*** No los mutas en un reductor sino que devuelves un nuevo objeto.

Finalmente, como se puede ver, del lado del cliente se opta por una variante de Flux utilizando Redux, quedando la siguiente distribución de directorios,

- */components*, en este directorio se definen las distintas componentes de la aplicación utilizando react.

- */containers*, en este directorio se definen distintos contenedores. Los mismos son componentes que interactuan con redux para acceder al estado interno de la aplicación así como también a las acciones definidas.

- */helpers*, en este directorio se definen distintas funciones que puedan ser de utilidad para el desarrollo de la aplicación.

- */reducers*, en este directorio se define todo lo referido a redux,

  - */actions*, se definen las acciones redux

  - */constants*, se definen los nombres de las distintas estrategias que determinan el estado dado de una aplicación.

  - */reducers*, define los reducers de la aplicación.

  - *states*, define el estado inicial de cada una de las entidades en el store de la aplicación.

  - ...

- ...

REFERENCES

[1] Documentación oficial de docker
https://www.docker.com/what-docker

[2] Documentación oficial de webpack
https://webpack.js.org/concepts/

[3] Documentación oficial de React
https://reactjs.org/docs/getting-started.html

[4] Documentación oficial de Semantic UI
https://react.semantic-ui.com/introduction

[5] Documentación oficial de React Semantic UI
https://semantic-ui.com/introduction/getting-started.html

[6] Documentación de Routing de Express
http://expressjs.com/en/guide/routing.html

[7] Guía de Seguridad de Express
http://expressjs.com/en/advanced/best-practice-security.html

[8] Documentación de Redux
https://redux.js.org/

[9] Documentación de Angular
https://angular.io/docs

[10] Mongoose Docs
http://mongoosejs.com/docs/guide.html