Informe extensión de proyecto de software

Grupo 23
Integrantes: Cordoba Mariano, Carlos Menendez, Conde Martínez Matías

Resumen

Se realizó una nueva versión del proyecto Hospital Dr. Alejandro Korn manteniendo las mismas características pero utilizando un framework de php para su desarrollo.

Características del sistema

1) Usuarios con roles y permisos con sus respectivos módulos de alta, modificación, asignación y eliminación para poder controlar el acceso y acciones de cada usuario del sistema.
2) Lista de pacientes con su información y CRUD correspondiente.
3) Sistema de consultas que actúa de historial clínico para cada paciente, donde se almacena información sobre derivaciones, diagnósticos y atenciones.
4) Generador de reportes sobre los pacientes y sus atenciones, filtrado por diferentes características y con la posibilidad de exportación en formato pdf.

Características técnicas

1) Aplicación web con PHP (https://php.net/) en su versión 7.2
2) Backend desarrollado con el FrameWork Symfony (https://symfony.com/) en su versión 4.2.
3) ORM Doctrine (https://www.doctrine-project.org/) en su versión 2.6.3 para el manejo (y abstracción) de la base de datos
4) Motor de base de datos en mariadb en su versión 10.1.34 pero podría cambiarse por otras soportadas por Doctrine.
5) Frontend desarrollado con el FrameWork Vue.js (https://vuejs.org/) en su versión 2.6.10.
6) Framework de CSS con Bulma (https://bulma.io/).

Fundamentación sobre el FrameWork elegido

Para el desarrollo de la nueva versión del proyecto quisimos plantear un nuevo modelo de arquitectura, más actual, el cual tiene en cuenta el rápido avance de las tecnologías web y la necesidad de poder adaptar / hacer más flexibles las aplicaciones y servicios para integrar nuevas tecnologías.
En esta nueva arquitectura contamos con una servidor api REST el cual es consumido por los clientes. En este caso la página es el front end, que a su vez es el cliente que consume los servicios de nuestra api REST, que es el backend. Este modelo trae la inmediata ventaja de separar completamente el backend del front, logrando así hacer mucho más sencilla la actualización del front a nuevas tecnologías. También permite integrar múltiples clientes que pueden ser web,

aplicaciones móviles, etc, con un solo backend evitando así tener que escribir un backend para cada aplicación.
Además como las api REST no mantienen estados/sesiones, optamos por utilizar jwt (json web token) que son un estándar de token para dar identidad y privilegios a un usuario, estos son lo suficientemente compactos como para viajar en cada pedido que se hace al servidor.

Para el backend elegimos Symfony que es un open source framework de PHP bajo la licencia MIT Open Source license. Los motivos de nuestra elección fueron su comunidad activa de desarrolladores, una documentación muy detallada, sencilla , con ejemplos (https://symfony.com/doc/current/index.html) y una gran cantidad de soporte y bundles para expandir el proyecto.
Destacamos FOSRestBundle para Symfony ya que es un paquete que integra muchísimas funcionalidades y seguridad para aplicaciones api REST (https://github.com/FriendsOfSymfony/FOSRestBundle).

En cuanto al ORM Symfony tiene una muy buena y desarrollada integración con Doctrine, además también cuenta con un desarrollo activo y una documentación completa (https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/index.html). Además la utilización de un ORM trae acarreada la ventaja de abstraernos del motor de base de datos que estamos utilizando. Esta es la lista de motores que soporta doctrine (https://www.doctrine-project.org/projects/doctrine-dbal/en/2.9/reference/platforms.html).

Pasando al frontend elegimos Vue.js como framework para el desarrollo de la aplicación que consume el back. Vue es relativamente nuevo y ha progresado muy rápidamente en popularidad dentro del mercado gracias a su sencillez (liviano), flexibilidad, y facilidad de integración. Además cuenta con una comunidad muy activa y una documentacion completa (https://vuejs.org/v2/guide/). Vue se auto define como un framework “progresivo” ya que se puede ir integrando nuevos elementos al sistema con facilidad, esto viene de la mano de su arquitectura orientada a componentes que contiene toda la funcionalidad y el html dentro de un mismo conjunto. También sus componentes cuenta con reactividad, esto implica que lo elementos que están en la vista reaccionan en tiempo real a cambios que se realicen en inputs o diferentes eventos que se produzcan en un mismo componente.

Vue cuenta con una gran cantidad de bundles para expandir sus funcionalidad, entre los que más detacamos se encuentra axios (https://vuejs.org/v2/cookbook/using-axios-to-consume-apis.html) el cual es escencial para que nuesta aplicacion consuma los servicios del backend ya que integra todas las funcionalidad para realizar pedidos AJAX y todo el manejo de errores, headers y request.

Por último en cuanto al framework de css nos quedamos con la misma elección que hicimos en la primera versión del proyecto, Bulma (https://bulma.io/) ya que es extremadamente sencillo, puro css y sin javascript para su funcionalidad.

Módulos reutilizados

Al realizarse la nueva versión de la aplicación respetando el esquema MVC, hemos podido reutilizar un gran porcentaje de los módulos de la vista. Una parte de los controladores y los modelos se hicieron desde cero.

El manejo de la vista realizado con twig en la primer versión de la aplicación no difería mucho de lo que se puede hacer con Vue. Esto nos resultó beneficioso y nos permitió realizar una transición simple.

En cuanto a los controladores: cambiar la lógica de una aplicación integrada con el front a una lógica útil a una api REST separada del front, causó que no pudiéramos reutilizar todo. Sí se pudo aprovechar una gran parte de la lógica del procesamiento de la información. No obstante, al usar un ORM, agregamos anotaciones para que Doctrine se encargue por nosotros de los chequeos/validaciones de los campos.

Por último, el modelo o entidades se tuvieron que rehacer, esto es debido a que pasamos del uso de PDO a ORM, por lo cual todas las entidades se hicieron desde cero para ser compatibles con esta tecnología. Aprovechamos muchísimo la potencia del ORM y sus funciones predefinidas y prácticamente no tuvimos que escribir código para el modelo.

Implementación del sistema

API REST

En la implementación del backend realizamos una servicio API REST con Symfony + FOSRestBundle, este conjunto nos brinda una estructura de proyecto lista para este propósito. Comenzando por el árbol de directorios del proyecto:

- ☐ Final
  - ☐ Assets
  - ☐ Config
    - ☐ jwt
    - ☐ packages
  - ☐ Public
    - ☐ build
    - ☐ bundles
    - ☐ index.php
  - ☐ src
    - ☐ Controller
    - ☐ Entity
    - ☐ Repository
  - ☐ templates
  - .env
  - webpack.config.js

En la carpeta src tenemos nuestro subdirectorio /Controller ,aquí se encuentran todos los controladores y gran parte de la lógica de nuestra API REST. En /Entity tenemos las entidades de nuestro modelo, son los objetos de nuestro ORM que utilizamos para representar los elementos de nuestra base de datos. Por último en /Repository están los objetos que se encargan de buscar y procesar los datos que se necesitan de la BD para los controladores.

En la carpeta Config se encuentra la mayor parte de la configuración de la página, dentro de /packages esta la config individual para cada uno de los bundles y componentes instalados y en /jwt están las llaves privada y pública para generar los jwt.

Public es la unica carpeta que se puede acceder desde afuera al sistema y donde están todos los recursos de la página. Aquí se encuentra el index.php que es el punto de acceso al sistema.

El archivo .env están las variables de ambiente del sistema y en webpack.config.js la configuración del webpack.

En cuanto a la implementación del servicio se encuentran los siguientes bundles:

FOSRestBundle - Entre sus tantas funcionalidades nos facilita la posibilidad de rutear métodos de nuestros controladores a rutas específicas que especifiquemos para ellos por request http, también con la utilización de annotations nos permite definir reglas que se deben cumplir para poder consumir esos métodos, por ejemplo que el request sea específicamente POST, GET, DELETE etc, o que estén presentes ciertos parámetros en el body de la request y a su vez que cumplan con cierto conjunto de reglas. Esto nos da la inmediata ventaja de ahorrarnos definir un montón de reglas que usualmente definimos dentro de cada método y generamos una respuesta. FOSRestBundle genera respuesta automáticas a cualquier violación de estas reglas.

LexikJWTAuthenticationBundle - Este bundle junto a SecurityBundle nos permiten generar (al logearnos) tokens jwt para autenticarnos dentro del sistema. Al ser una aplicación que no guarda sesiones se le da al usuario este token para que en cada request lo coloque en el header de autenticación, así el sistema sabe que usuario es. Este toquen tiene varias particularidades que se detallan en el apartado de seguridad. Además el SecurityBundle nos permite definir firewalls que son reglas de autenticación que tiene que cumplir un usuario antes de poder invocar una ruta específica. Las reglas de este firewall se definen dentro de el archivo config/packages/security.yaml

WebpackEncoreBundle - Permite la integración de webpack en el sistema

NelmioApiDocBundle - Es un pequeño extra a modo de documentación, este bundle genera una página (poner link) que permite ver todos los métodos disponibles para invocar de nuestra api y especifica los parámetros y métodos que se necesita.

Frontend Vue.js

- Final
  - Assets
    - css
    - img
    - js
      - components
      - app.js

El front es una aplicación con su propia lógica que consume el Api rest antes mencionado.
La estructura del proyecto se encuentra dentro de la carpeta /Assets. Los estilos, en /css. Las imágenes en /img y los js que conforman todo el front se alojan en /js.

Dentro de /js esta el app.js que seria el script principal. Aquí importamos todas las librerías a usar, bundles y estilo. Además se realiza toda la configuración principal y se definen los componentes a utilizar, métodos y eventos globales.

En la carpeta components se alojan todos los componentes de Vue. Un componentes de Vue está conformado por una parte de vista (Html) luego la lógica (Scripts) y por ultimo estilos (Style), los componentes a su vez pueden alojar otros componentes.
Esto le brinda muchísimo dinamismo a la página, y la gran ventaja de poder re utilizar un solo componente en cualquier parte que se lo sea requerido.

Dentro de los bundles que utilizamos tenemos:

Axios - Nos brinda herramientas para hacer consultas Ajax a la Api, de esta forma podemos tanto obtener toda la información necesaria para la vista como concretar acciones en el sistema.

VueRouter: enrutador oficial de Vue. Se integra profundamente con el núcleo de Vue para facilitar la creación de aplicaciones. Algunas de las características que incluye son: mapeo de rutas/vistas anidadas, configuración basada en componentes/módulos, parámetros de ruta, etc. (https://router.vuejs.org/)

VueGoodTable: plugin para creación de tablas. Posee funcionalidades tales como clasificación, filtrado de columnas, paginación, etc. (https://xaksis.github.io/vue-good-table/)

VeeValidate: librería para validación de inputs HTML con gran variedad de reglas listas para utilizar y soporte para reglas personalizadas. También permite la validación de componentes personalizados de Vue. (https://baianat.github.io/vee-validate/)

Vue2Leaflet: librería JavaScript para Vue que empaqueta Leaflet (librería para mapas interactivos) (https://korigan.github.io/Vue2Leaflet/#/)

VueSweetAlert: reemplazo responsive, accesible y personalizable para las cajas de alerta de JavaScript. (https://sweetalert2.github.io/)

VueChartJs: es un empaquetador de ChartJs para Vue. Nos permite crear gráficos rápidamente. Simples, flexibles, y fáciles de utilizar.(https://vue-chartjs.org/)

Seguridad

Protección CSRF

El sistema se protege intrínsecamente de este tipo de ataque, ya que para realizar cualquier acción al back es necesario enviar un JSON Web Token el cual solo tiene la aplicación en su espacio de memoria, por lo cual no es accesible u obtenible desde otra aplicación para realizar este tipo de ataque.

Inyección de SQL

Con la utilización de el ORM Doctrine nos garantiza la proteccion e integridad de la base de datos contra todos esos tipos de ataques (https://www.doctrine-project.org/projects/doctrine-orm/en/latest/reference/security.html#security) .

Proteccion XSS

Aquí tenemos dos protecciones contra esto, Doctrine nos cubre contra la inserción de código XSS en la base de datos y Vue al procesar o analizar datos en el front, antes de mostrarlos, los sanitiza para evitar cualquier ejecución de XSS.

Conclusiones

Luego de muchas idas y vueltas con cómo encarar el proyecto y probar otras maneras de realizarlo (ya que en un punto tiramos lo que habíamos hecho por algo distinto). Nos dimos cuenta de el gran potencial y facilidades que dan tanto los frameworks como este modelo de Api Rest con clientes que lo consumen.

Cuando nos ponemos a pensar, que pasaria si tengo que actualizar el sistema? cambiar el back o el front o portear el trabajo a otro tipo de tecnologías, es evidente que la gran independencia que hay entre ellos facilita muchísimo la tarea, tanto en esfuerzo como en tiempo.

Por otra parte la importancia de las comunidades y el soporte que dan los mismos desarrolladores de los frameworks que utilizamos. Es elemental tanto a la hora del desarrollo, como en agregar nuevas funcionalidades que estén presentes esas cosas.