UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL RESISTENCIA

Trabajo Práctico Final.

Asignatura: Desarrollo de Aplicaciones Cliente-Servidor.

Grupo Nº: 4 (Cuatro).

Integrantes:

- Imfeld, Facundo Nicolas (imfeld59@gmail.com)
- Lopez, William Juan Jose (lopezwilliam177@gmail.com)
- Nasir, Khalil Abdul (kanasir28@gmail.com)
- Stride, Eric (strideeric94@gmail.com)
- Troncoso, Clarise (sclariset@gmail.com)

Profesores:

Ing. VILLAVERDE, Jorge Eduardo
Ing. QUEVEDO, Fabricio Ruben

Año: 2021.

Índice

Informe

Escenario

El Ministerio de Desarrollo Productivo, junto con la Secretaría de Comercio Interior han sancionado la Resolución 237/2021 por la cual se crea el Sistema Informático para la Implementación de Políticas de Reactivación Económica (SIPRE).

Este sistema, cuyo objetivo final es contribuir a la reactivación económica del país, tiene por alcance a todas las empresas del sector comercio e industria local. Estas empresas deberán suministrar información de forma mensual, a través del repositorio de información del MINISTERIO DE DESARROLLO PRODUCTIVO, los primeros diez (10) días corridos de cada mes calendario.

La información suministrada deberá contener, como mínimo, los siguientes datos:

CUIT de la empresa.
Denominación del producto.
Código EAN o equivalente sectorial del producto; y
Precio por unidad de peso, cantidad o medida del producto.
Cantidades producidas y vendidas
Para esto los alumnos de la cátedra Desarrollo de Aplicaciones Cliente-Servidor deberá desarrollar un sistema que implemente los siguiente sub-sistemas:

Ministerio de Desarrollo Productivo
Este subsistema deberá exponer las APIs (Application Programing Interface) para que las empresas presenten mensualmente la información de régimen informativo de la resolución. A su vez, deberá exponer servicios para resumir la información que está disponible para la Secretaría de Comercio Interior.

Empresas del Sector Comercio
Se deberá crear librerías para acceder y publicar los regímenes informativos al Ministerio de Desarrollo Productivo. Se deberá proveer una implementación de referencia, que utilice la librería y publique los datos en el Ministerio.

Secretaría de Comercio Interior
La Secretaría de Comercio Interior debe ser capaz de consultar los datos publicados en el repositorio de información del Ministerio de Desarrollo Productivo, y por medio de ciertas reglas de negocios y políticas establecidas, generar alertas de incumplimiento, y en caso de ser necesario, reportar al comercio el incumplimiento de la normativa.

Lenguaje

- ➢ TypeScript

Typescript es un super-conjunto de Javascript, que añade tipado estático y objetos basados en clases, y que finalmente compila en Javascript plano.
Permite establecer el tipo de datos de las variables, métodos, y parámetros, facilitando entender las entradas y las salidas de nuestros componentes de código.
La escritura Estática es una característica que detecta errores de los desarrolladores. Promueve que se escriba un lenguaje más robusto y fácil de mantener. Por lo que es un lenguaje más limpio.

Base de datos

➢ MongoDB

Es una base de datos orientada a documentos. Esto quiere decir que en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.
El usar una base de datos basada en documentos y que sea Schema Less como lo es MongoDB hace que tu base de datos crezca con tu aplicación sin tener que ejecutar scripts que crean campos con valores por defecto cada vez que quieras agregar un campo nuevo en tus registros.

Frameworks

➢ Node.Js

El uso más común de NodeJS es el desarrollo de servicios web que devuelven datos en formato JSON, lo que llamamos habitualmente API REST.
Se destaca por la creación de aplicaciones de red rápidas, ya que es capaz de manejar una gran cantidad de conexiones simultáneas con un alto nivel de rendimiento, lo que equivale a una alta escalabilidad. Tiene la característica de acometer muchas tareas con poco consumo de recursos, lo que lo hace especialmente interesante para el desarrollo de servicios de alta concurrencia.
Podemos expandir nuestro código añadiendo módulos de forma fácil gracias al Node Package Manager (NPM).

Patrón de diseño

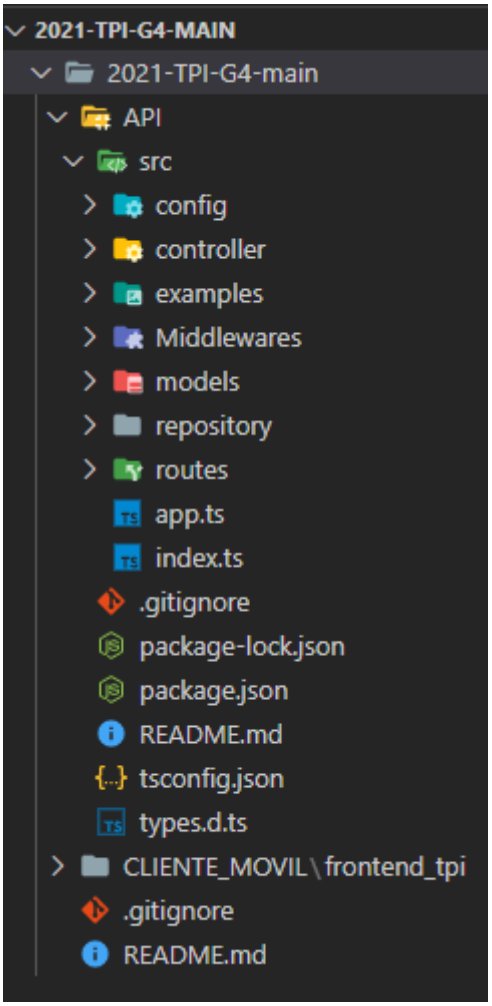
El Patrón Repositorio está destinado a crear una capa de abstracción entre la capa de acceso a datos y la capa empresarial para que pueda ayudar a aislar la aplicación de los cambios en el almacén de datos y facilitar las pruebas unitarias automatizadas para el desarrollo basado en pruebas.
Es un patrón para aislar los tecnicismos de la persistencia de las capas superiores y en especial del modelo de Dominio. Gracias a ello podremos concentrarnos en implementar en el modelo las reglas de negocio. El cliente ahora podrá concentrarse en cómo presentar la información, en cómo aplicar filtros, algoritmos, relaciones y reglas que hagan cumplir las reglas de empresa especificadas.

Ventajas:

- El código de acceso a los datos puede ser reutilizado.

- Es fácil de implementar la lógica del dominio.
- Nos ayuda a desacoplar la lógica.
- La lógica de negocio puede ser probada fácilmente sin acceso a los datos.
- También es una buena manera de implementar la inyección de dependencia que hace que el código sea más testeable.

Estructura del Proyecto



La estructura inicial del proyecto contiene en la raíz un archivo ***package.json*** donde vamos a especificar la información de nuestro paquete con sus dependencias, un archivo ***index.ts*** donde corremos el servidor posterior a las acciones que se encuentran el archivo **app.ts** donde se instancia express, se realizan algunas configuraciones de servidor, se declaran Middlewares incorporados y Rutas de la API, un directorio para nuestros ***middlewares***, un directorio de ***models*** donde se alojan modelos, esquemas e interfaces, un directorio de ***controllers*** en el cual se encuentran los controladores de rutas manejando las Reponse y las Request junto con mínimas operaciones antes de pasar al repositorio correspondiente, y finalmente, un directorio de ***routes*** en el que definimos las rutas (endpoints) a las que responderá nuestra aplicación.

package.json

Es el archivo de configuración principal del proyecto y debe encontrarse en la raíz del mismo. En él debe estar reflejado el nombre del proyecto, versión, descripción, scripts, autor, tipo de licencia y algo muy importante, las dependencias.

index.ts

Es el archivo principal de la aplicación, en este archivo se arranca el servidor posteriormente de ejecutar todas las acciones del archivo app.ts.

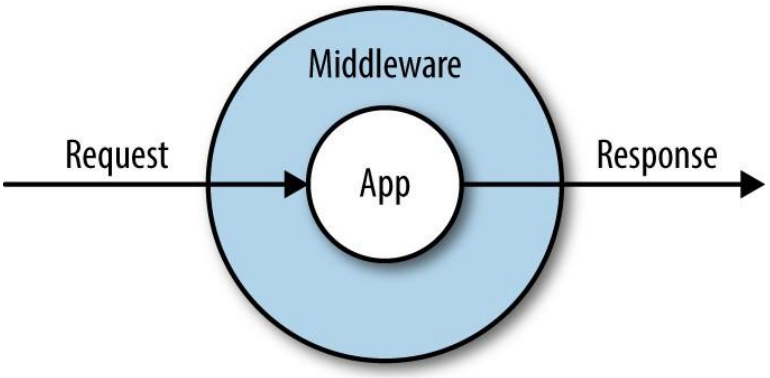
app.ts

Se puede considerar como el archivo secundario de la aplicación, en este archivo se realizan las configuraciones iniciales del proyecto, como instanciar express, realizar algunas configuraciones de servidor, declaramos middlewares generales e invocamos a las rutas de la API.

middleware

En esta carpeta se encuentran los middlewares de la aplicación, o también conocidos como funciones de validacion. Los mismos se pueden ejecutar antes o después del manejo de una ruta. Estas funciones tienen acceso al objeto Request, Response y la función next(). Los middlewares pueden ser clasificados de la siguiente manera:

- Middleware de nivel de aplicación
- Middleware de nivel de direccionador
- Middleware de manejo de errores
- Middleware incorporado
- Middleware de terceros



controller

En esta carpeta se encuentran definidos los controladores de rutas, estos son los encargados de manejar los objetos Request y Response.

routes

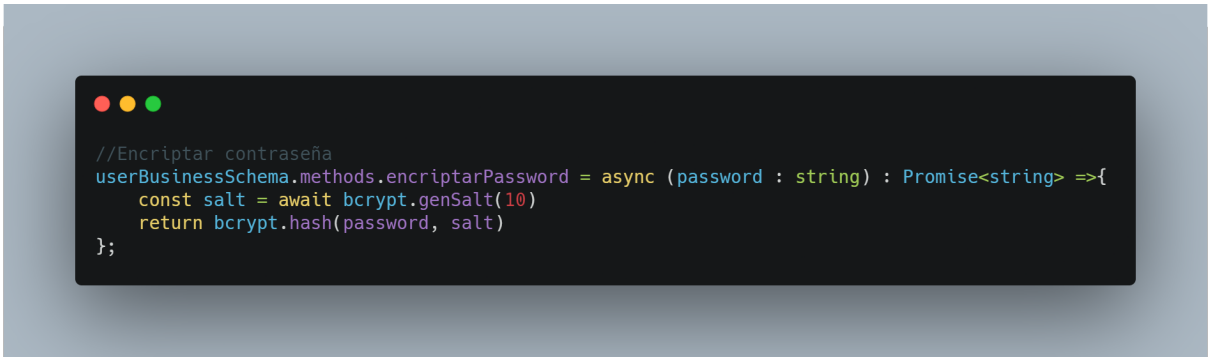
Carpeta que contiene los archivos en el que definimos las rutas (endpoints) a las que responderá nuestra aplicación. Una ruta sería algo así como:

Donde:

- Router es una clase de express para crear manejadores de rutas montables y modulares.
- METHOD es un método de solicitud HTTP
  - GET: el método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
  - HEAD: el método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
  - POST: el método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
  - PUT: el modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
  - DELETE: el método DELETE borra un recurso en específico.
  - CONNECT: el método CONNECT establece un túnel hacia el servidor identificado por el recurso.
  - OPTIONS: el método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.
  - TRACE: el método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.
  - PATCH: el método PATCH es utilizado para aplicar modificaciones parciales a un recurso.
- PATH sería por donde accede al servidor
- HANDLER es la función que se ejecuta

models

Carpeta donde se encuentran las interfaces, modelos y esquemas de datos del proyecto. Cada modelo también puede contener métodos, por ejemplo se muestra a continuación un método para encriptar la contraseña de los usuarios de empresas:



```
//Encriptar contraseña
userBusinessSchema.methods.encriptarPassword = async (password : string) : Promise<string> =>{
    const salt = await bcrypt.genSalt(10)
    return bcrypt.hash(password, salt)
};
```

config

Carpeta contenedora de archivos de configuración donde se hace la conexión a la base de datos y se instancian las variables de entorno.

repository

Carpeta que contiene los repositorios de reportes, empresas, productos y usuario empresa, en estos se definen las operaciones CRUD y otros métodos y funciones para el control y validación de datos.

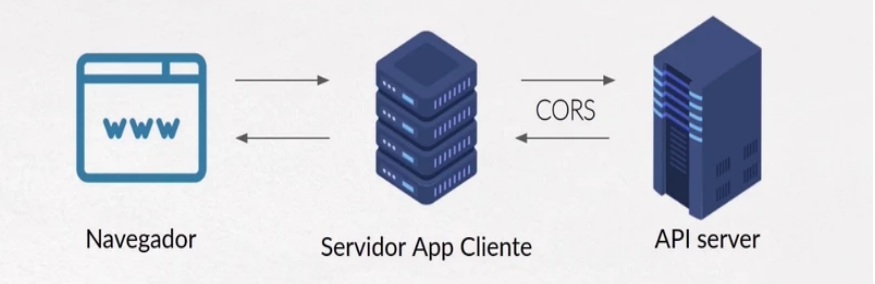
Librerías/Dependencias

- Swagger

  Su objetivo es estandarizar el vocabulario que utilizan las APIs. Es una plataforma que ordena cada uno de nuestros métodos (get, put, post, delete) y categoriza nuestras operaciones. Cada uno de los métodos es expandible, y en ellos podemos encontrar un listado completo de los parámetros con sus respectivos ejemplos.
  El principal objetivo de este framework es enganchar el sistema de documentación con el código del servidor para que esté sincronizado a cada cambio. Casi documentamos al mismo tiempo que creamos la implementación.
  - *swagger-jsdoc*: Genera definiciones de OpenAPI a partir de comentarios JSDoc.
  - *swagger-ui-express*: Crea la página de la interfaz de usuario de Swagger a partir de estas definiciones.

- Cors

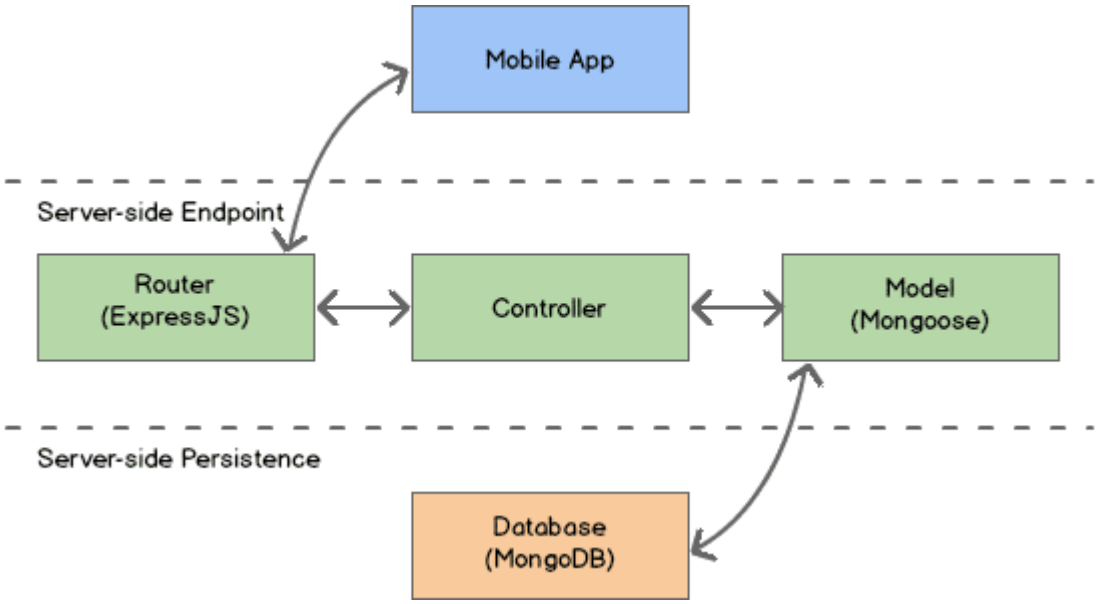
  CORS es una especificación del consorcio W3C que está implementada por casi todos los navegadores, de forma que podemos especificar qué dominios están autorizados y cuáles no, y para qué están autorizados.
  Es una política de seguridad, es decir, una política que establece qué es ser seguro, y que nos va a dar una serie de reglas de control de acceso y de autenticación.



En definitiva, CORS (Cross-Origin Resource Sharing) es un mecanismo que, a través de las cabeceras de los encabezados HTTP, va a permitir a un determinado cliente (User-Agent) a acceder a los recursos de un servidor diferente al del servidor actual.

➢ Mongoose

Mongoose es una librería escrita en Node.js para trabajar con MongoDB que me permite crear esquemas muy flexibles con muchas combinaciones diferentes posibles de cómo puedo organizar mis datos, es decir, nos proporciona una solución sencilla basada en esquemas para modelar los datos de la aplicación. Incluye conversión de tipos incorporada, validación, creación de consultas, hooks y más, listos para usar.



➢ Express

Es un framework sobre nodejs que nos permite trabajar con el protocolo http y tener sistemas de rutas.

Básicamente, usaremos esto para manejar el enrutamiento en nuestro servidor backend.

➢ express-validator

Es un conjunto de express.js middlewares que envuelve validator.js funciones de validación y desinfectante.

Es la biblioteca que usaremos para manejar la validación de entradas.

➢ ts-node

Es un motor de ejecución de TypeScript y REPL para Node.js.

JIT transforma TypeScript en JavaScript, lo que le permite ejecutar TypeScript directamente en Node.js sin precompilar. Esto se logra conectando las API de carga del módulo del nodo, lo que permite que se use sin problemas junto con otras herramientas y bibliotecas de Node.js.

➢ Morgan

Es un middleware de registro de solicitudes HTTP para Node.js. Simplifica el proceso de registro de solicitudes en su aplicación.

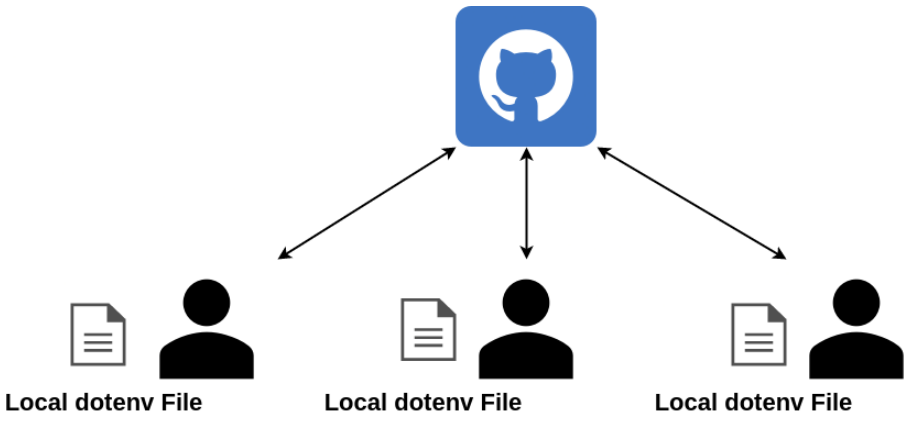
➢ Jsonwebtoken

Usamos un paquete npm llamado jsonwebtoken, que nos ayudará a tratar con JWTs. La idea es recoger las claves públicas y usar este paquete para que valide si es un token correcto o no.

➢ Dotenv

Permite la lectura sencilla de las variables de entorno en archivos de extensión ".env". Mediante estos archivos podemos almacenar las variables en una notación bastante habitual.

La sintaxis de los .env contiene pares clave (nombre de variable) y valor, separados por un carácter "=". Cada variable en una línea.



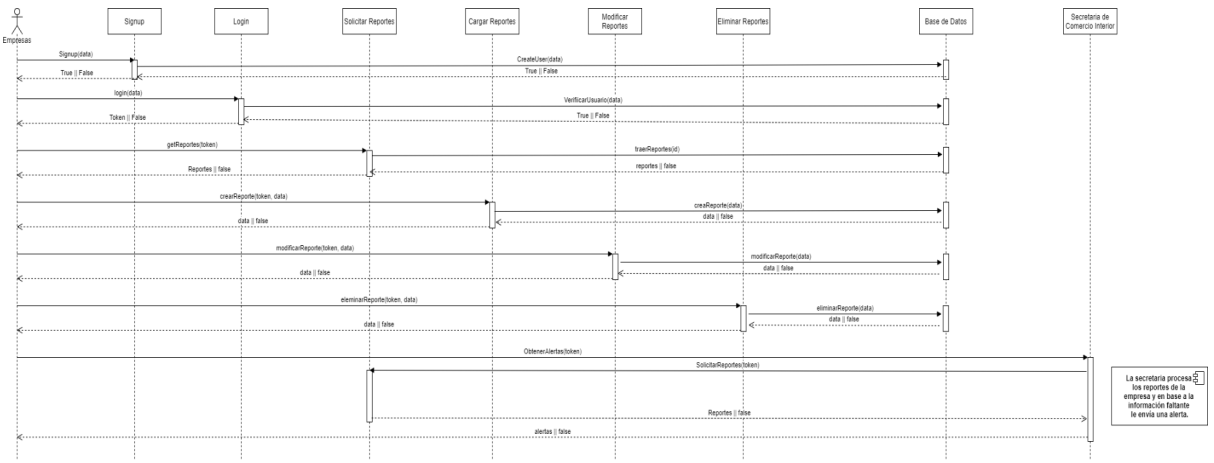
➢ Bcryptjs

Es una biblioteca optimizada escrita en javascript que permite trabajar con hash bcrypt de forma simple y eficiente. Básicamente, se encripta la contraseña de los usuarios utilizando la librería “bcrypt”.

Roles de usuario del sistema

- Usuario Empresa
  - Consultar reportes.
  - Cargar reportes.
  - Modificar reportes.
  - Eliminar reportes.
  - Ver estadísticas totales de cantidades vendidas y producidas (de un reporte a la vez).
- Usuario ministerio
  - Ver estadísticas de cantidades vendidas y producidas por industria, por año. (de todos los reportes juntos).

Casos de usos de integración con otro sistema



Frameworks de frontend

Flutter

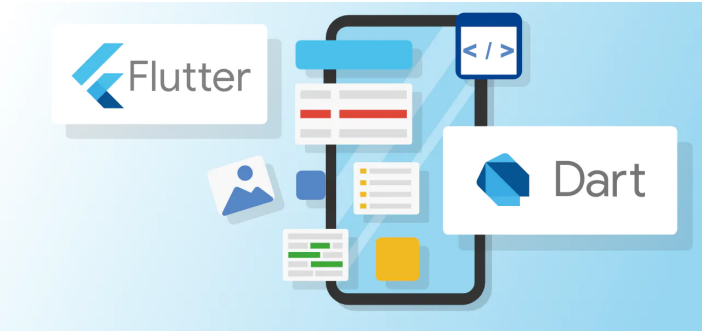
Flutter es un SDK de código fuente abierto de desarrollo de aplicaciones móviles creado por Google. Suele usarse para desarrollar interfaces de usuario para aplicaciones en Android, iOS y Web.

Ventajas que ofrece Flutter al desarrollo multiplataforma:

- Compila en nativo, tanto en Android como en iOS.
- La creación de interfaces gráficas es muy flexible, puedes combinar diferentes Widgets (elementos gráficos) para crear las vistas.
- El desarrollo es muy rápido, permite ver el resultado de forma instantánea mientras se escribe el código.

Desventajas de Flutter

- Para poder usar Flutter es necesario aprender el lenguaje de programación Dart.
- Actualmente está enfocado en especial para uso Móvil. Si nuestra aplicación va a tener un sitio web tendremos que desarrollarla paralelamente a la versión de móvil.
- Otra desventaja de usar Flutter es el tamaño de la aplicación. Las aplicaciones creadas con Flutter son pesadas y tardan mucho en iniciarse o cargarse. Por lo que esto puede arruinar la experiencia del usuario y es un indicador de bajo rendimiento.



Vue js

Vue es un framework progresivo para construir interfaces de usuario.
La librería central está enfocada solo en la capa de visualización, y es fácil de utilizar e integrar con otras librerías o proyectos existentes. Por otro lado, Vue también es perfectamente capaz de impulsar sofisticadas Single-Page Applications cuando se utiliza en combinación con herramientas modernas y librerías de apoyo.

Librerías

Pie Chart

Pie Chart es un plugin que hace uso del elemento canvas para renderizar gráficos circulares para valores unitarios. Los gráficos tienen opción de personalización, son sencillos de implementar e incluso se puede modificar su escala para optimizarlos en pantallas de retina.

http

Librería utilizada para realizar las peticiones hacia el servidor instalado en Heroku.

dart:async

Librería utilizada para trabajar con programación asincrónica en el código

dart:convert

Librería utilizada para codificar y decodificar entre distintos tipos de datos. En nuestro caso, usada para decodificar los datos que llegan en formato JSON desde el servidor y usada también para codificar a formato JSON los datos que querramos enviar al servidor.

material.dart

Librería utilizada para trabajar y mostrar en la UI los datos que obtengamos desde el servidor.

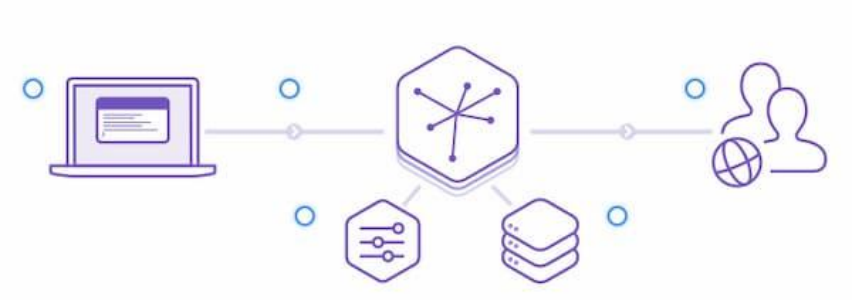
tailwind CSS

Framework de CSS que permite aplicar estilos al sitio web de manera ágil y optimizada.

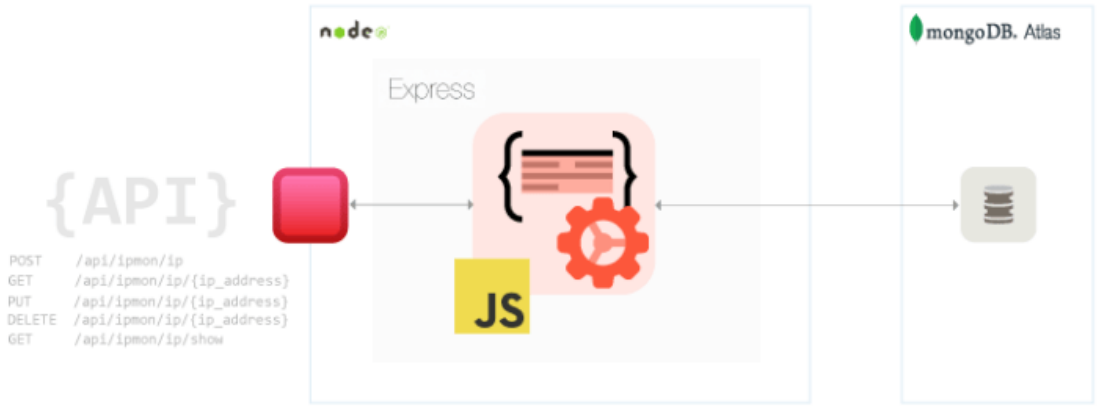
Deployment en Cloud o Servidor remoto

- **DEL BACKEND**
  Aplicación: HEROKU
  Heroku es una plataforma basada en la nube como servicio diseñada para que los desarrolladores y equipos creen, envíen, supervisen y escalen aplicaciones modernas. Esta plataforma basada en contenedores brinda a los desarrolladores

más tiempo para centrarse en el producto principal sin tener que preocuparse de mantener la infraestructura de la aplicación.
El servicio tiene una versión limitada pero gratuita con la posibilidad de actualizar a un plan de pago dependiendo de las necesidades. Para usarlo basta con crear una cuenta en la plataforma, proceder a crear una aplicación y proceder a deployar la misma, para esto heroku nos brinda su propio entorno de trabajo llamado Heroku Git o bien la posibilidad de vincular la aplicación con un repositorio de GitHub. El deploy se puede realizar de manera manual o automática, esto último sucede cada vez que empujemos un cambio a nuestro repositorio remoto.



Base de Datos: MongoDB Atlas
MongoDB Atlas es un servicio de Cloud Database (o Base de Datos en la Nube), que te permite crear y administrar tu Base de Datos Mongo desde cualquier lugar, a través de su plataforma. No solo está orientado a ser accesible desde el navegador, sino que, fue desarrollado con el objetivo de aliviar el trabajo de los desarrolladores, al quitarles la necesidad de instalar y administrar entornos de BBDD, los que a veces pueden ser lentos y engorrosos.
MongoDB Atlas cuenta con una versión limitada pero gratuita, con posibilidad de actualizar a un plan de pago dependiendo de las necesidades. Para utilizarlo basta con crear una cuenta en la plataforma (“Organización”), para luego proceder a crear un Clúster (Lugar donde se almacenan los datos en MongoDB). Terminado esto debemos proporcionar acceso a nuestra Base de Datos, ya que MongoDB Atlas tiene bastantes medidas de seguridad integradas para evitar accesos no deseados a la BBDD. Una de ellas, es el bloqueo de IP, que permite restringir las direcciones desde las que se puede acceder al BBDD. Para permitir una dirección IP en MongoDB Atlas, desde nuestro dashboard debemos hacer clic en “Network Access”, y luego en “Agregar Dirección IP”.



Conclusión

En cuanto a cuestiones técnicas pudimos aprender el concepto básico de servicios REST, logramos comprender los requerimientos del usuario y plasmarlo en funcionalidad a través de TypeScript, NodeJs, Express y MongoDB, en el transcurso del trabajo tuvimos muchos altibajos técnicamente hablando ya que al ser typescript un lenguaje tipado tuvimos repetidos problemas con las entradas y salidas de datos, también pudimos comprender el concepto de Middlewares y como se clasifican estos, logrando con éxito su implementación. A nivel de equipo logramos tener buena comunicación y coordinación a la hora de dividir tareas y cumplir con los plazos. Finalmente podemos decir que aprendimos conceptos básicos de TypeScript como ser métodos y funciones, así como también conceptos como rutas, peticiones HTTP, controladores y otros derivados de express, consecuentemente, logramos adquirir conocimientos de colecciones y consultas a base de datos no relacionales ya que usamos MongoDB como motor principal.

Bibliografía

- https://es.vuejs.org/v2/guide/
- https://pub.dev/
- https://nodejs.org/es/
- https://docs.mongodb.com/
- https://mongoosejs.com/docs/guide.html