Estructura Código - Trabajo Práctico 6

Este documento describe la disposición del código realizado para el trabajo práctico número 6, indicando:

* Stack Tecnológico
* Convenciones de código
* Estructura de carpetas
* Dependencias utilizadas

Stack Tecnológico

La aplicación tiene un stack con las siguientes tecnologías utilizadas:

* **Backend:** Se utiliza Node.js, plataforma para crear aplicaciones APIs restful.
* **Express:** Framework que maneja rutas y peticiones HTTP.
* **Frontend:** Se utiliza React Native, un framework para desarrollo de aplicaciones móvil nativas.
* **Lenguaje de programación:** Se utiliza JavaScript, tanto en el backend como en el frontend.
* **Base de datos:** Se utiliza MongoDB, una base de datos no relacional seleccionada por el tamaño de la aplicación y su facilidad de uso.

Convenciones de código

* Se utiliza camelCase a la hora de nombrar archivos y carpetas contenedoras.
* Se utiliza PascalCase en elementos funcionales relacionados a la parte visual de la aplicación/frontend de React Native.
* Se utilizan funcioo-‘nes flecha () => {} al usar JavaScript ES6
* Los comentarios se colocan con // comentario

Estructura de carpetas

El proyecto se compone de dos partes, el backend de la aplicación, que maneja la lógica de negocio, y el frontend, que es la parte visual de la aplicación.

Backend

Contiene código fuente necesario para aplicar la lógica de negocio y el manejo de base de datos para realizar consistencia de información.

* Carpeta src: Carpeta contenedora de los elementos de código fuente a realizar para la aplicación, separándose de dependencias utilizadas.9
  + Controllers: Encargados de procesar las solicitudes provenientes del frontend y responder a las mismas, aplicando la lógica de negocio.
    - pedidosController.js: Para manejo de pedidos, tanto su creación como su consulta.
    - soporteController.js: Código que contiene lógica para pasar información simple (un solo campo, formato número y string) del ORM al backend de la aplicación.
    - tiposDeCargaController.js: Para manejo de tipos de carga de pedidos, creación, consulta y uso; así como datos mockeados de tipos predeterminados.
  + Middlewares: Carpeta que contiene interfaz intermediaria para manejo de errores del servidor.
    - errorMiddleware.js: Encargado de manejar los errores que puedan surgir durante la conexión entre un servidor y nuestro sistema.
  + ORM: Módulo encargado de realizar el mapeo de una base de datos no relacional a paradigma de objetos.
    - conexion.js: Encargado de realizar la conexión entre la base de datos Mongo con nuestro código.
    - pedidosOrm.js: Encargado de almacenar y obtener de la base de datos información relacionada a los pedidos creados.
    - soporteOrm.js: Encargado de obtener de la base de datos los datos de un solo campo.
    - transportistasOrm.js: Encargado de obtener de la base de datos la información de los transportistas.
  + Routes: Módulo que maneja las solicitudes a uris concretas, dirigiendo solicitudes a los controladores adecuados, y definiendo qué acción se debería realizar.
    - \_index.js: Define todas las posibles rutas, sus controladores y maneja las excepciones a rutas inexistentes.
    - pedidosRoute.js: Código que contiene verbos y controladores para los pedidos.
    - soporteRoute.js: Código que contiene verbos y controladores para obtener información de un solo campo.
    - tiposDeCargaRoute.js: Código que contiene verbos y controladores para los tipos de carga que se puede tener en un pedido.
  + Services: Carpeta que contiene módulos relacionados a servicios utilizados en la aplicación.
    - emailService.js: Maneja el envío de emails a la hora de notificar creaciones de pedidos a los transportistas correspondientes.
    - notificacionPushService.js: Maneja la creación de un popup a la hora de crear un pedido.
  + index.js: Punto de entrada principal al código, donde se encuentra definición de rutas, la conexión a la base de datos y el middleware utilizado.
* .env: Se almacenan variables de entorno externas al código. En este caso, dirección de base de datos utilizada, token de mail y puerto donde se hostea aplicación.
* package.json: Elemento de configuración de dependencias utilizadas en la aplicación.

Frontend

Contiene código fuente necesario para la parte visual de la aplicación que va a utilizar el usuario final.

* Carpeta src: Carpeta contenedora de los elementos de código fuente a realizar para la aplicación, separándose de dependencias utilizadas.
  + components: Carpeta que contiene elementos visuales propios de la aplicación actual.
    - CustomButton.js: Elemento visual de botón personalizado.
    - CustomDatePicker.js: Elemento visual de selector de fechas para creación de pedidos.
    - CustomImagePicker.js:
    - CustomInput.js: Elemento visual para insertar texto personalizado.
    - CustomSelect.js: Elemento visual para seleccionar opciones personalizado.
    - DomicilioForm.js: Elemento visual que muestra un formulario en el cual se va a cargar el domicilio de retiro y de envío de un pedido.
  + navigation: Carpeta que contiene módulo de navegación entre pantallas
    - AppNavigator.js: Módulo que contiene todas las posibles pantallas de la aplicación y permite la navegación entre estas.
  + screens: Carpeta que contiene pantallas de la aplicación.
    - ConsultarPedidoEnvioScreen.js: Pantalla para consultar todos los pedidos de envío.
    - HomeScreen.js: Pantalla inicial de la aplicación, la cual permite navegar a otras pantallas.
    - RegistrarPedidoEnvioScreen.js: Pantalla para registrar un pedido de envío, cargando toda la información requerida.
  + services: Carpeta que contiene módulo de APIs para comunicarse con el backend de la aplicación.
    - apiClient.js: Código encargado del manejo de solicitudes a APIs y manejo de errores.
    - pedidoEnvioService.js: Encargado de comunicarse con lógica de negocio pertinente a la obtención y creación de pedidos de envío.
    - tipoCargaService.js: Encargado de comunicarse con backend pertinente a obtención de tipos de carga de pedidos.
  + styles: Carpeta contenedora para estilos de fuente y de colores para la aplicación.
    - colores.js: Se encuentra la paleta de colores utilizada en la aplicación.
    - globalStyle.js: Se encuentra el diseño visual de los contenedores de la aplicación.
  + utils: Carpeta contenedora de código de utilidades importadas.
    - dateHelpers.js: Código para selector de fechas
* App.js: Punto de entrada principal del código donde se encuentra el navegador de pantallas. Se encarga de dibujar los distintos componentes.
* app.json
* package.json: Elemento de configuración de dependencias utilizadas en la parte visual de la aplicación.

Dependencias utilizadas

* Expo: Herramientas propias para desarrollo móvil junto con React Native. Proporciona acceso a APIs de los dispositivos móviles.
* MongoDB: Base de datos no relacional, que nos permite un manejo sencillo de altas cantidades de datos sin una configuración de tablas de base de datos rígida. Utilizada junto con Mongoose.
* NodeMailer: Biblioteca de Node.js que nos permite el envío de correos electrónicos.
* Express: Framework utilizado para creación de aplicaciones con Node.js para manejo de solicitudes HTTP.
* Axios: Biblioteca utilizada para manejo de solicitudes HTTP al backend desde el frontend.
* Node.js: Entorno de ejecución de JavaScript para el servidor backend.
* Android Studio: Entorno utilizado para tener un emulador móvil y realizar pruebas desde la PC.