Hardware Haven

Índice

1. Documentación General del Proyecto

1.1 Introducción

Descripción General

La idea de este proyecto surgió de la necesidad de simplificar el proceso de construcción de una PC personalizada. Tomando como base otras plataformas centralizadas que ofrecen una amplia gama de componentes de hardware con información actualizada sobre precios y disponibilidad. Estas ideas nos llevaron a brindar una solución: una plataforma en línea que proporcionara a los usuarios una experiencia simplificada y sin complicaciones para armar una PC personalizada.

Inicialmente, el sistema se ha dividido en dos componentes distintos: el Backend por un lado brindando el servicio de API y el Frontend, cada uno con sus propias subdivisiones y funcionalidades específicas. Siguiendo las directrices de la cátedra, se ha determinado que el sistema debe tener la capacidad de ejecutarse como una API Server. Esta API incluye funcionalidades de persistencia en la base de datos desarrollada en MySQL, lo que permite almacenar y recuperar datos de manera eficiente.

Objetivos: Los objetivos principalmente son educativos aplicar los conocimientos adquiridos a lo largo de la cursada y visualizar el uso de estas tecnologías.

Alcance: Esta definido partiendo desde el manejo de usuarios, componentes y compra hasta la facturación.

1.2 Guía de Inicio Rápido

Guía para el usuario

Acceder a la URL de la página web de “Hardware Haven”

Guía para el Desarrollador

Frontend

Para configurar el Entorno de trabajo basta con posicionarse con una terminal dentro de la carpeta Hardware Haven y luego ejecutar el comando: **ng serve**

Backend

Para configurar el Entorno de trabajo de la API, el primer paso será clonar el proyecto desde el repositorio correspondiente. Una vez clonado, es crucial establecer las variables de entorno necesarias para su correcto funcionamiento. Para ello, sigue estos pasos:

1- Crear Archivo .env: En la raíz del proyecto, crea un archivo llamado .env.

2- Definir el Puerto del Servidor: Dentro del archivo .env, establece el puerto en el cual deseas que el servidor escuche las solicitudes. Por ejemplo, puedes ingresar PORT=3000 para configurar el servidor en el puerto 3000. Este paso es crucial para evitar conflictos de puertos y asegurar la accesibilidad de la API.

3- Ejecutar el Servidor: Una vez que hayas configurado las variables de entorno, abre una terminal y navega hasta el directorio del proyecto. Luego, ejecuta el comando **start:dev** para iniciar el servidor en modo de desarrollo.

Ejecución del servidor en modo de desarrollo
Al seguir estos pasos, será posible ejecutar y trabajar con la API en el entorno local de desarrollo. Recuerde que cualquier cambio en las variables de entorno requerirá reiniciar el servidor para que los cambios surtan efecto

Cadena de conexión a la base de datos

La cadena de conexión de la base de datos se encuentra en el siguiente archivo: .env de variables de entorno si desea modificarla cambie la propiedad DB

2. Documentación Técnica

2.1 Arquitectura del Sistema

El sistema se compone de tres elementos fundamentales que interactúan entre sí para proporcionar una experiencia completa y funcional:

Frontend

Cuando hablamos de esta parte nos referimos a la interfaz de usuario a través de la cual los usuarios interactúan con el sistema. Está diseñado para ser intuitivo y fácil de usar, brindando una experiencia fluida y agradable. Utiliza tecnologías modernas como HTML, CSS y JavaScript, junto con el framework como Angular, para crear interfaces dinámicas y receptivas que se adapten a una variedad de dispositivos y pantallas.

Backend

El Backend es la parte del sistema que se encarga de procesar las solicitudes del cliente, realizar operaciones en la base de datos y devolver los resultados al Frontend. Está construido utilizando la tecnología como Node.js y utiliza frameworks como Express.js para crear la API siendo robusta y escalable. El Backend también gestiona la lógica de negocio y la autenticación de usuarios, garantizando la seguridad y la integridad de los datos.

MikroORM (Nuevo inciso)

MikroORM es una herramienta ORM (Mapeo Objeto-Relacional) que nos simplifica el manejo de la capa de persistencia en aplicaciones como Node.js y TypeScript. Nos proporciona una interfaz fácil de usar para interactuar con la base de datos, permitiendo nos a los desarrolladores definir modelos de datos, realizar consultas y manipular datos de manera eficiente y sin problemas. MikroORM es compatible con una gran variedad de bases de datos, incluyendo PostgreSQL, MySQL y MongoDB, lo que lo convierte en una opción versátil para proyectos que requieren un acceso eficiente a la base de datos con un código limpio y legible.

Base de Datos

La Base de Datos es el repositorio central de datos del sistema, donde se almacenan y gestionan todos los datos relacionados con los usuarios, componentes, compras y cualquier otra información relevante. Nos referimos en este caso a una base de datos relacional como lo es MySQL acorde a los requisitos específicos del sistema. La Base de Datos es accesible desde el Backend a través de consultas estructuradas en un lenguaje como SQL. En este caso específico se utilizó un ORM (MikroORM) el cual se encargará de realizar todas estas consultas

2.2 Detalles del Código

Estructura del Proyecto: Descripción del árbol de directorios y el propósito de cada carpeta.

Lo que primero nos encontramos es con la carpeta **Dist** posee los archivos javaScrip watcheados de TypeScript esta conversión hace que luego podamos ejecutar el código sin problema.

Luego nos encontramos con la carpeta llamada **src**

Listado de rutas de sus subcarpetas:

App.ts: archivo principal de nuestra APP de TypeScript

config: carpeta que posee todas las configuraciones del server tanto express y de http

controllers: Controladores que manejan la lógica de negocio y las respuestas como las requisiciones del cliente al api

model: Estructura que posee cada entidad del modelo. (Esto es requerido principalmente por el **ORM**)

repository: Funciones que se encargan de acceder a la base de datos atreves del **ORM**

routers: Serie de routers divididos por cada entidad

security: Todo lo referido con la seguridad de la aplicación. Posee principalmente las funciones de sanitización

shared:

Serie de funcionalidades que comparten el conjunto de las entidades como el acceso a la base de datos, interfaces, o características generales.

requests.http: Archivo que le permitirá evaluar la api. Este posee todas las req que se le pueden hacer todos los métodos y demás

Patrones de Diseño Utilizados: Es parecido a la modelo vista controlador, pero utilizando una capa más de repositorio, routers, y de seguridad si se deberían hacer validaciones o impedir ciertos accesos

Convenciones de Código:

Convenciones:	Descripción
Controladores	**<Nombre de la entidad>-<Método>.Controllers.ts**

Routers	**<Nombre de la entidad>Router.ts**
Repository	**<Nombre de la entidad>Respository.ts**
Model	**<Nombre de la entidad>**.entity.**ts**

Nombre de la entidad: Representa a la entidad

Método: Descripción de que se busca hacer con dicha entidad

'**-**' y '**.**' Son separadores

3. Documentación de la API del Servidor

3.1 Visión General de la API

La API es un servicio que provee el Backend, su propósito se integra con el resto del sistema mediante solicitudes http comunicando así el cliente con el servidor

Especificación de Versiones:

Aquí se encuentran las versiones actuales de la API y notas de cada versión.

Fecha	Versión	descripción
2024-04-16	1.00	FIRST CRUD FINISHED
2024-05-03	1.01	TESTING FINISHED
2024-09-22	1.02	JWT IMPLEMENTED

3.2 Autenticación y Autorización

Métodos de Autenticación:

Los usuarios hasta el momento poseen usuario único y contraseña

Los roles implementados son:

Administrador: El rol de administrador posee la potestad de poder alterar cualquiera de las entidades del sistema. Es el más importante de los roles existente.

Cliente: El rol de cliente solo posee la capacidad de ver los productos. Ver su detalle, comprar, editar, ver o eliminar su propia cuenta.

JWT, DTOs y validaciones extras: De las últimas modificaciones que se ha buscado implementar son Json Web Token para que cada vez que se realiza un "login" la API responda con un Token a los usuarios autorizados. El mismo deberá utilizarse por parte del cliente en el "Beare" encabezado de autorización "Authorization" en el "Header" cuando sea el caso dependiendo de los roles antes mencionados. Por otra parte, los DTOs son validaciones previas que se realiza por parte de la API. A pesar de no haberse implementado con profundidad son herramientas super potentes para realizar validaciones.

3.3 Endpoints

Detalles principales

Descripción: Breve descripción del propósito del endpoint.

URL: Ruta del endpoint.

Método HTTP: GET, POST, PUT, DELETE, etc.

Parámetros de Entrada: Detalles sobre los parámetros aceptados, incluyendo tipo de datos, si son obligatorios u opcionales, y ejemplos.

Cuerpo de la Solicitud (si aplica): Estructura y ejemplos del JSON u otro formato de entrada.

Tabla de EndPoints
Excel de EndPoints

Códigos de respuesta HTTP implementados
Estos son los códigos de respuesta implementados

Código	Respuesta	Descripción
200	OK	La solicitud ha tenido éxito
201	Created	La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello
204	No Content	La petición se ha completado con éxito pero su respuesta no tiene ningún contenido
404	Not Found	El servidor no pudo encontrar el contenido solicitado
501	Not Implemented	El método solicitado no está soportado por el servidor y no puede ser manejado
503	Service Unavailable	El servidor no está listo para manejar la petición. Causas comunes (Mantenimiento)
500	Internal Server Error	El servidor ha encontrado una situación que no sabe cómo manejarla.

(Esta en evaluación agregar 400, 403, 406)

4. Guías y Ejemplos de Uso

4.1 Casos de Uso Comunes

Ejemplos Prácticos: Casos de uso específicos y ejemplos detallados de cómo interactuar con la API.

4.2 Tutoriales y Walkthroughs

Tutoriales Paso a Paso: Guías detalladas para realizar tareas específicas utilizando la API.

5. Mantenimiento y Soporte

5.1 Gestión de Incidentes y Problemas

Reporte de Errores: Si encuentras algún error o problema mientras utilizas la API, te animamos a que nos lo hagas saber. Puedes reportar errores o problemas a través de nuestro sistema de seguimiento de problemas en GitHub o mediante el contacto directo con nuestro email.

Política de Mantenimiento: Nos comprometemos a proporcionar actualizaciones y mantenimiento regulares para el proyecto. Mantenemos informados a nuestros usuarios sobre cualquier actualización planificada a través de nuestros canales de comunicación habituales. Hasta el fin de la cursada

5.2 Contribución

Guía de Contribución: Si deseas contribuir al proyecto ponte en contacto, te damos la bienvenida luego de la cursada. Consulta nuestra guía de contribución en el repositorio de GitHub para obtener información detallada sobre cómo puedes contribuir al proyecto. Esto incluye normas de codificación, pautas de prueba y cómo enviar pull requests de manera efectiva.

6. Referencias y Recursos Adicionales

Enlaces Útiles: Documentación de terceros, herramientas recomendadas, tutoriales adicionales, etc.

Como utilizar una API: https://appmaster.io/es/blog/apis-para-principiantes-como-utilizar-una-api-una-guia-completa