Documentación Técnica de [Prueba Tecnica node.js Backend]

1. Introducción

1.1 Descripción del Proyecto

Desarrollar un API Restful para un administrador de tareas básico.

1.2 Tecnologías Utilizadas

Node.js v18.16.0.

Framework Express.js

Base de datos: MySQL Workbench.

Visual Studio Code

2. Configuración del Entorno de Desarrollo

2.1 Requisitos Previos

Antes de comenzar con el desarrollo en este proyecto, se debe tener los siguientes requisitos instalados en el ambiente de desarrollo:

- Node.js
- Express.js
- MySQL Workbench
- Visual Studio Code

2.2 Configuración del Entorno Local

Este proyecto utiliza Express.js como framework web

Se ejecuta el siguiente comando para instalar Express.js

```
npm install express
```

- Nodemon (Opcional, pero recomendado para desarrollo):

Nodemon es una herramienta que ayuda a desarrollar aplicaciones basadas en Node.js reiniciando automáticamente la aplicación cuando se detectan cambios en el código.

Se ejecuta el siguiente comando para instalar Nodemon.js

```
npm install -g nodemon
```

- MySQL y express-myconnection:

El proyecto utiliza una base de datos MySQL y express-myconnection como middleware de conexión.

Se ejecuta el siguiente comando para instalar Express.js

```
npm install express-myconnection
```

- Faker (Opcional, solo si se utiliza para generar datos de prueba):

Faker es una biblioteca que puede ser útil para generar datos de prueba.

Se ejecuta el siguiente comando para instalar Express.js

```
npm install faker
```

3. Estructura del Proyecto

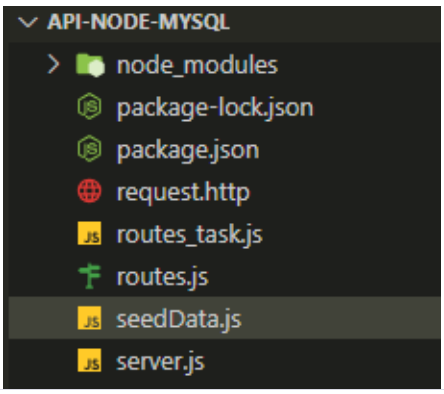
3.1 Estructura de Carpetas y Archivos

La organización de archivos en este proyecto sigue una estructura clara que simplifica el proceso de desarrollo y mantenimiento. A continuación, se describe la estructura de carpetas y los archivos más relevantes:

- server.js:
  Este archivo sirve como punto de entrada principal para la aplicación. Aquí se configuran e inicializan los elementos esenciales, como el servidor Express
- routes.js y routes_task.js:
  Estos archivos contienen las definiciones de las rutas de la aplicación. `routes.js` puede manejar rutas de los usuarios, mientras que `routes_task.js` podría estar específicamente relacionado con las rutas relacionadas con las tareas, además aquí se manejan los querys de la conexión a la base de datos.
- seedData.js:
  En este archivo, se encuentra el script relacionado con la generación de datos de prueba o iniciales para la base de datos. Es decir aquí se encuentra la generación de los 50 datos aleatorios que se pidieron desde un inicio.
- request.http:
  Este archivo puede contener ejemplos de solicitudes HTTP que pueden ser utilizadas para probar las rutas de la API. Es especialmente útil para probar y documentar las API de forma interactiva utilizando herramientas como VS Code y sus extensiones.

.

3.2 Arquitectura del Proyecto



El presente proyecto se divide en 2 rutas diferentes, una para los usuarios y otra ruta para las tareas, así no se usara una ruta para los 2 modelos y no se interferiría una con el otro, en request.http se hacen las 2 pruebas con cada petición.

4. Funcionalidades Principales

4.1 Descripción de Funcionalidades

1. **Gestión de Tareas:**
    - El núcleo del proyecto se centra en la gestión de tareas. Los usuarios pueden crear, editar y eliminar tareas a través de llamadas API.
2. **API RESTful:**
    - Se ha implementado una API utilizando Express.js para permitir la comunicación eficiente entre el cliente y el servidor. Las operaciones CRUD (Crear, Ver, Asignar, Eliminar) están disponibles para las tareas, lo que facilita la integración con aplicaciones front-end y otros servicios.
3. **Generación de Datos de Prueba:**
    - Para facilitar el desarrollo y las pruebas, se ha implementado una funcionalidad de generación de datos de prueba. El archivo `seedData.js` contiene scripts para poblar la base de datos con datos ficticios que simulan escenarios del mundo real para correr este script se ejecuta **npm seeData.js**.
4. **Manejo de Rutas:**
    - Las rutas de la aplicación están organizadas de manera modular en los archivos **`routes.js`** y **`routes_task.js`**. Esto permite una fácil expansión y mantenimiento a medida que se agregan nuevas funcionalidades a la aplicación.

4.2 Endpoints de API

A continuación, se presenta una lista de los endpoints disponibles en la API junto con sus métodos HTTP y descripciones de funcionalidad, cabe señalar que se divide tanto para las tareas como para los usuarios:

Usuarios

1. **`GET /api`**
    - **Descripción:** Obtiene la lista completa de los usuarios.

Ejemplo: http://localhost:9000/api

2. `POST /api`
    - **Descripción:** Crea una nuevo usuario utilizando los datos proporcionados en el cuerpo de la solicitud.

   Ejemplo: http://localhost:9000/api
3. `DELETE /api/:id`
    - **Descripción:** Elimina un usuario específico según el ID proporcionado

   Ejemplo: http://localhost:9000/api/32
4. `PUT /tasks/:id`:
    - **Descripción:** Actualiza los detalles de una usuario existente identificado por el ID proporcionado.

   Ejemplo: http://localhost:9000/api/1

Tareas

5. `GET /tarea`
    - **Descripción:** Obtiene la lista completa de las tareas.

   Ejemplo: http://localhost:9000/tarea
6. `GET /tarea/:id`
    - **Descripción:** Obtiene las tareas que tiene asignadas el usuario John Doe en estatus “En bandeja” (puede ser para este usuario o cualquier otro).

   Ejemplo: http://localhost:9000/tarea/1
7. `GET /tarea/buscar/:palabra_abuscar_en_cadena`
    - **Descripción:** Obtiene la coincidencia de las tareas proporcionando una cadena de búsqueda.

   Ejemplo: http://localhost:9000/tarea/buscar/comida
8. `POST /tarea`
    - **Descripción:** Crea una nueva tarea utilizando los datos proporcionados en el cuerpo de la solicitud.

   Ejemplo: http://localhost:9000/tarea
9. `DELETE /tarea/:id`
    - **Descripción:** Elimina una tarea específica según el ID proporcionado

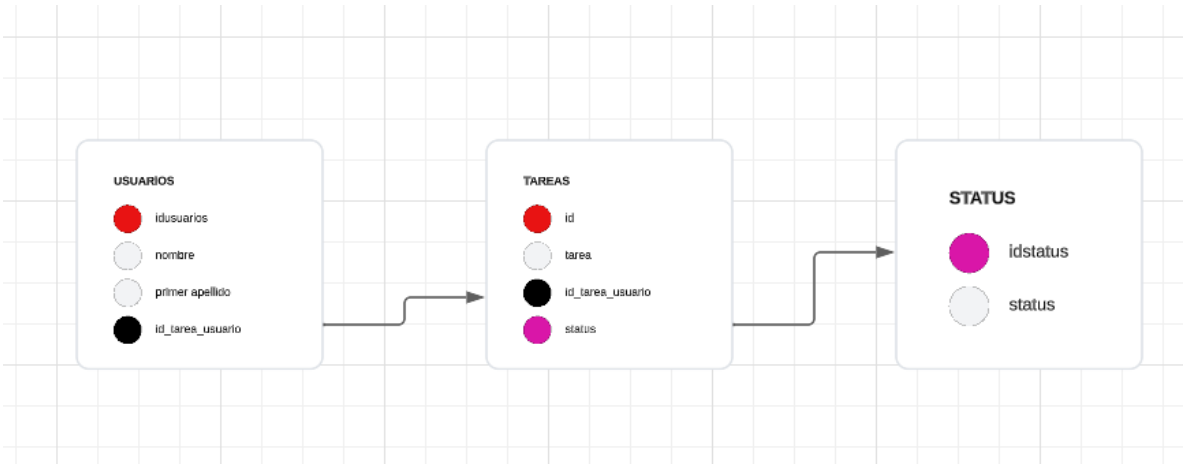
   Ejemplo: http://localhost:9000/tarea/4
10. `PUT /tarea/:id`:
    - **Descripción:** Asigna una tarea específica según el ID proporcionado a un usuario según los datos proporcionados en el cuerpo de la solicitud.

    Ejemplo: http://localhost:9000/tarea/5

5. Base de Datos

5.1 Modelo de Datos

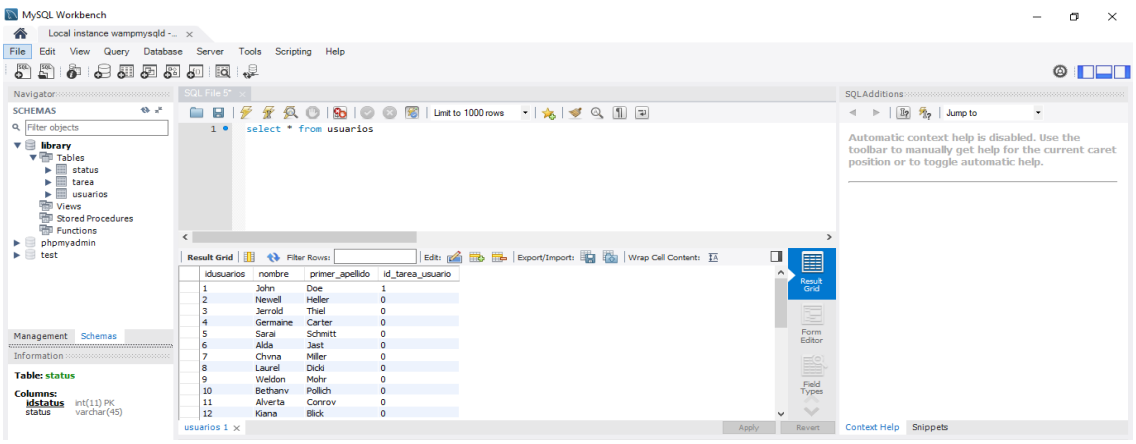
El presente proyecto contiene 3 tablas creadas en Mysql Workbench, que son la tabla de Usuarios, la tabla de Tareas y la tabla de Status a continuación se comparte el esquema del modelo de datos con sus relaciones



1.1 Imagen del diagrama entidad relación

En el diagrama se pueden observar las relaciones que tienen las tablas con cada una creando una relación 1 a 1 con cada tabla.

A continuación se podrá ver el ambiente de trabajo ya en función con las tablas y un query simple para ver la funcionalidad ya plasmada del diagrama entidad relación

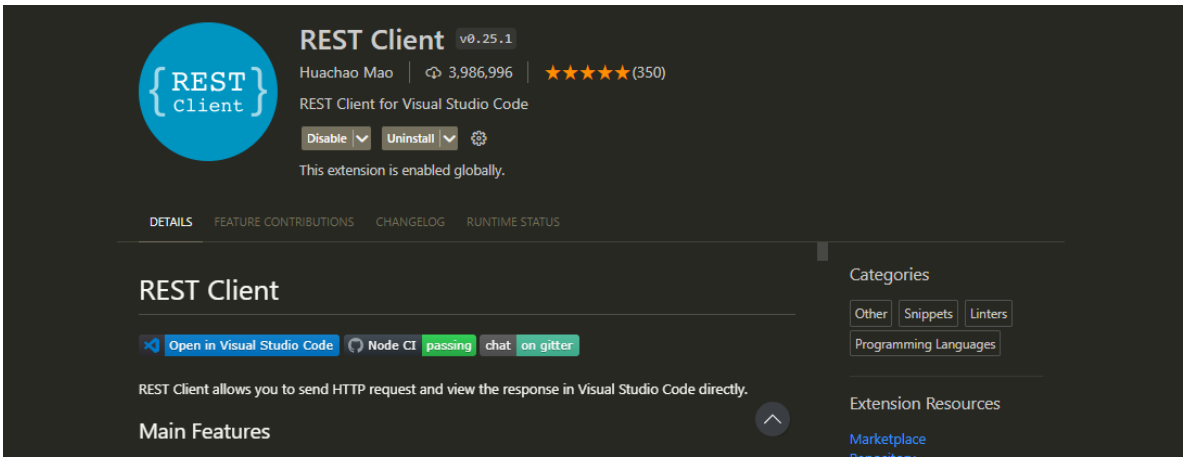


1.2 Mysql Workbench ya funcionando con las tablas ya creadas y un query funcionando

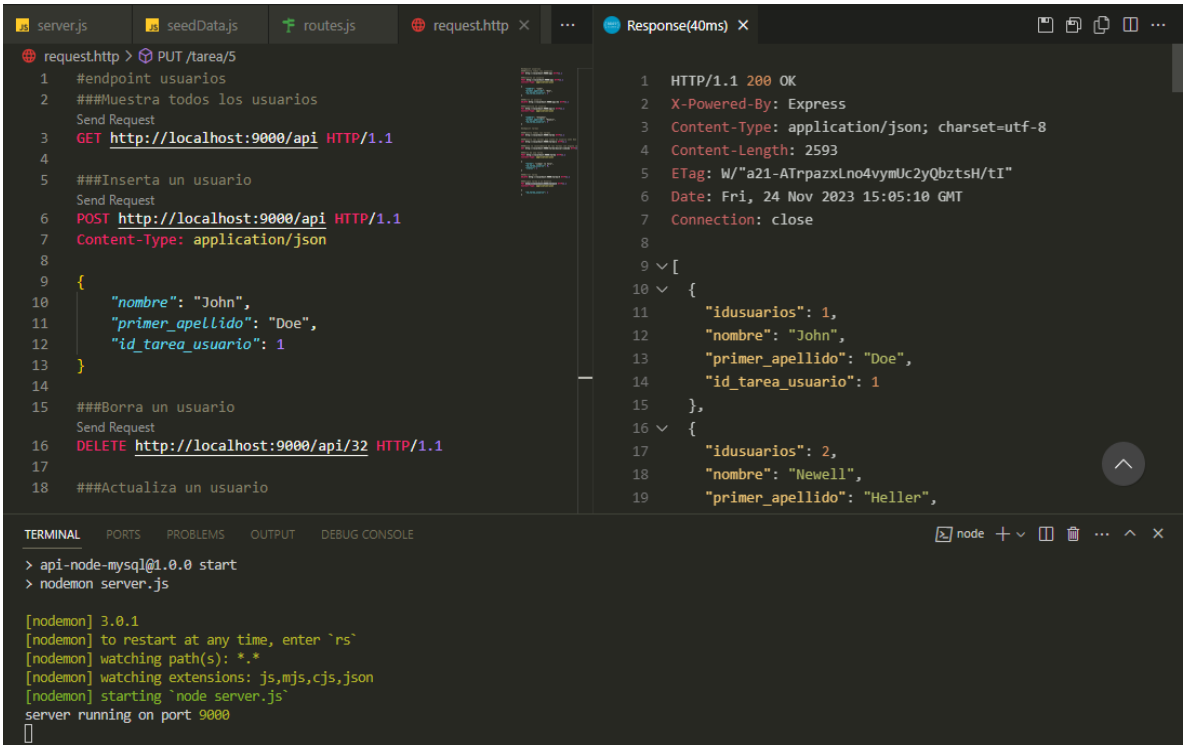
6. Pruebas

6.1 Extensión para aplicación de las Pruebas

Se instaló la extensión “REST client” de visual studio code para crear las pruebas, ese es el porqué de la creación del archivo request.http a continuación se dejan las imágenes de la extensión y de cómo se pueden ver las pruebas corriendo



1.3 Imagen de la extensión Rest Client de Visual Studio Code



1.4 Ejemplo de una llamada Get desde el archivo request.http corriendo con la extensión Rest Client de Visual Studio Code

7. Recursos Adicionales

7.1 Enlaces Útiles

MonkeyWit. (2023). Como crear una API REST con Node js y MySQL | CRUD (Create, Read, Update, Delete). MonkeyWit. https://www.youtube.com/watch?v=OWukxSRtr-A&t=1454s

Fazt. (2023). Nodejs & MySQL Deploy gratuito en Railway. Fazt. https://www.youtube.com/watch?v=C3NhmT__Mn4&t=415s.