GITHUB

BEFORE PUSHING CHANGES  
Need to fetch the latest changed of the repository

*Git fetch origin*

*Git pull origin main*

Resolve Any Merge Conflicts (if necessary):

 Open the conflicted files, look for conflict markers (e.g., <<<<<<<, =======, >>>>>>>), and edit the files to resolve the conflicts.

 After resolving conflicts, stage the changes:

*git add <conflicted-file>*

*git commit -m "Resolved merge conflicts"*

Initializing Server

Create a folder for the API and initialize typescript and Npm.

C:\Users\Tiago\Documents\CoverMatch\API>tsc –init

C:\Users\Tiago\Documents\CoverMatch\API>npm init -y

In the tsconfig file that was created for typescript, uncomment the lines: and add “src” as the root and “dist” in output directory

    "rootDir": "./src",

    "moduleResolution": "node10",

    "outDir": "./dist",

Now under API create 2 folders “dist” and “src”

Add a file “.gitignore” under the main folder CoverMatch:

.env

node\_modules

Under “src” create “index.ts”

We will be coding typescript in “src” and the javascript code will be generated in “dist”.

First install needed packages.

Npm i express cors json dotenv mssql

And then under “src” create a “Helpers” folder to store the functions and user validation.

And also a “DB” folder for DB documentation and stored procedures.

And also a “Config” folder to connect to our database.

And also a “routes” folder with a file “index.ts” to import and export the routes…

And also a “Controllers” folder for holding all the logic. We are developing in the way of MVC (Model View Controller).

And create “.env” file under API and add PORT = 5000;

Start coding on index.ts under “src” to initialize the server.

import express, { json, NextFunction, Request, request, Response } from 'express';

import dotenv from 'dotenv';

import cors from 'cors';

const app = express();

dotenv.config();

app.use(cors());

app.use(json());

const PORT = process.env.PORT;

const DB = process.env.DB;

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}`)

})

app.use((err: Error, req: Request, res: Response, next: NextFunction) => {

    if (err.message) {

        res.status(500).json({ error: err.message })

    }

})

Now to run the code in API folder:

*Tsc –W*

In another terminal in API folder:  
*nodemon dist/index.js*

Setting up Controllers

Create files needed for handling the logic. In this case I only need Users and maybe Resume (Under Controllers folder)

users.ts

resume.ts

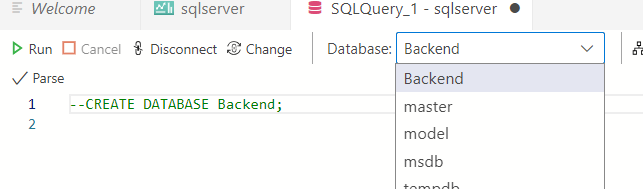
Database

To connect to SQL server, docker container needs to be running, then open Azure data studio and connect to the server.

Now we can create a new query and create the database.

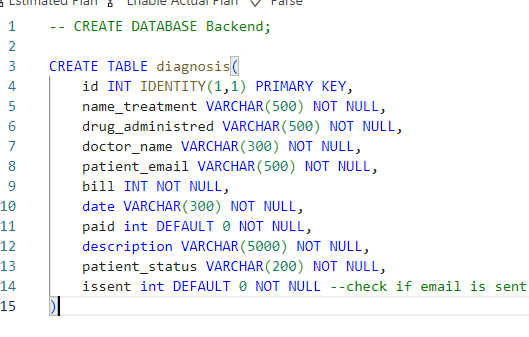
CREATE DATABASE Backend

Running this command will create a database and we can select the database to create the tables



We should also comment the query after its created.

Create tables



Now create a file “table.sql” in DB folder in vsCode

And then document the database on the new file, by copying all tables to the new file. This will help other programmers understand the database.

Create a new sql file ind DB folder “storeprocedure”:

API is done.

Background Services

We are going to use background services to send email to users.

When the user register, it will have a value of 0, meaning no email was sent to him. This services will run every minute or so, and will check if there is a new user to send email.

Our background services are only operating with the database.

Create “src”, “dist” and “templates” folder

Create .env file and copy everything from the other .env, except the key. Also change the port to 5001

Now initialize the typescript config.

*Cd background services*

*Tsc –init*

Uncomment the lines: and add “src” as the root and “dist” in output directory

    "rootDir": "./src",

    "moduleResolution": "node10",

    "outDir": "./dist",

We don’t run typescript, it is only for development state. In production, we use javascript

Inside background services run

*Npm init –y*

Then create an index.ts file in src

And a Config folder inside src with a config.ts file for connecting the database.

Create a folder EmailService for creating the emails

And a Helpers folder for holding the functions.

All those folders inside “src”

Now we install the packages we need.

*npm i express dotenv node-cron mssql ejs*