Beije Software Developer Internship Project

User Management System with Email Verification

Ahmet Furkan KIZIL

June 5, 2024

Reference Style: APA 7th edition

Table of Contents

Abstract

This project aims to simplify the process of verifying user email addresses during account creation on websites. It utilizes Nest.js framework to implement an efficient method for email verification. The project provides testing features for the control of functionality. Key features include endpoints for user registration, email verification, and checking verification status. The project leverages TypeORM for database management and integrates email sending capabilities using Gmail's free SMTP. The implementation focuses on simplicity, reliability, and scalability, making it suitable for various web applications.

Personal Engagement and ChatGPT Use

As a Sophomore Computer Science major student, I am experienced writing in different coding languages such as Python, Java, C++, Assembly etc. During this internship case, the most difficult part for me to was learning the syntax of JavaScript language and the basics of Nest.js framework. From the different projects which I've been part of, I improved in terms of understanding ChatGPT code and implementing them in my projects. I also developed in terms of handling different errors along the development of the projects and take full advantages of using ChatGPT. Thus, for this purpose, I've used ChatGPT interactively throughout the development of the internship project as shown below:

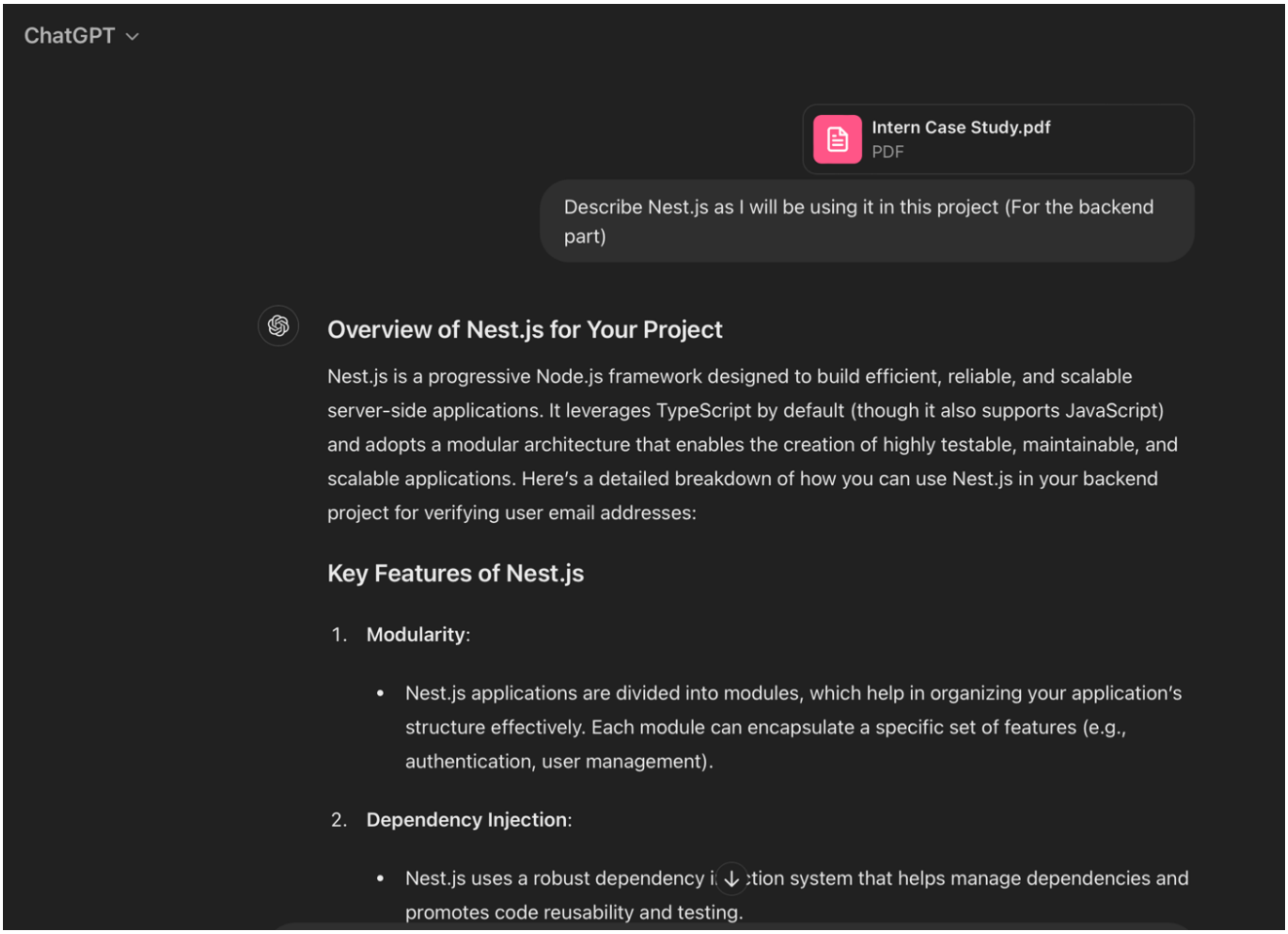

Figure 1: *ChatGPT usage*

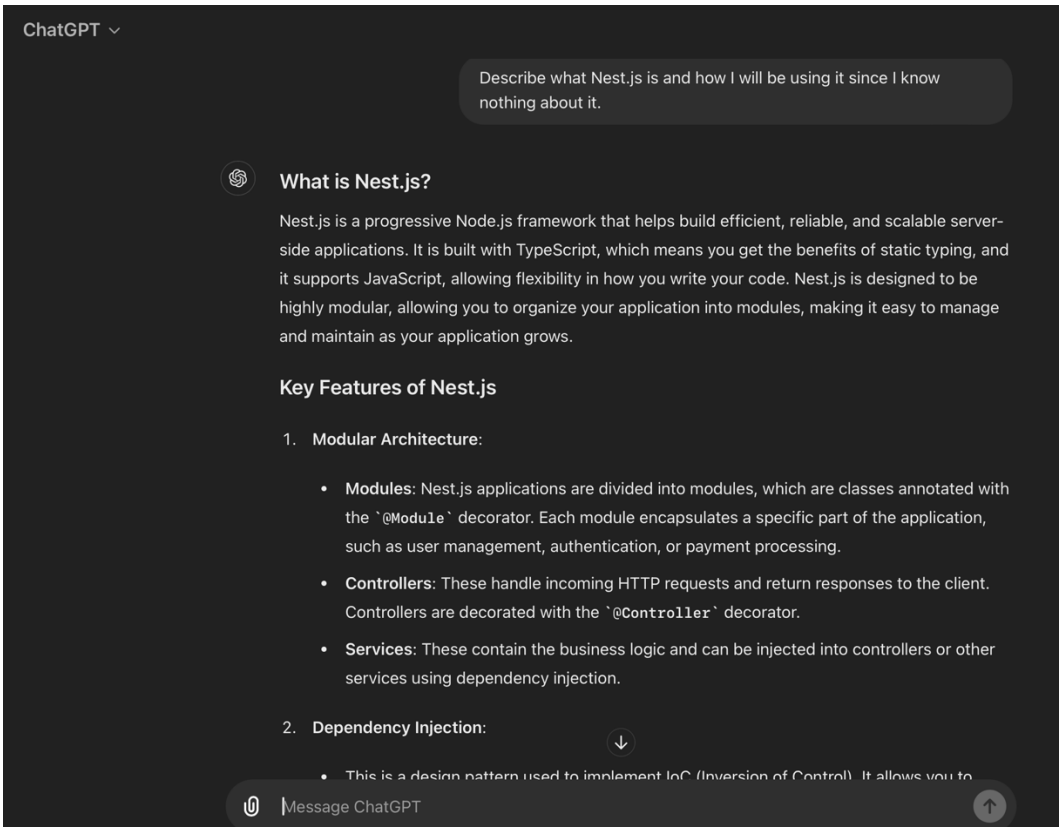

Figure 2: *ChatGPT usage*

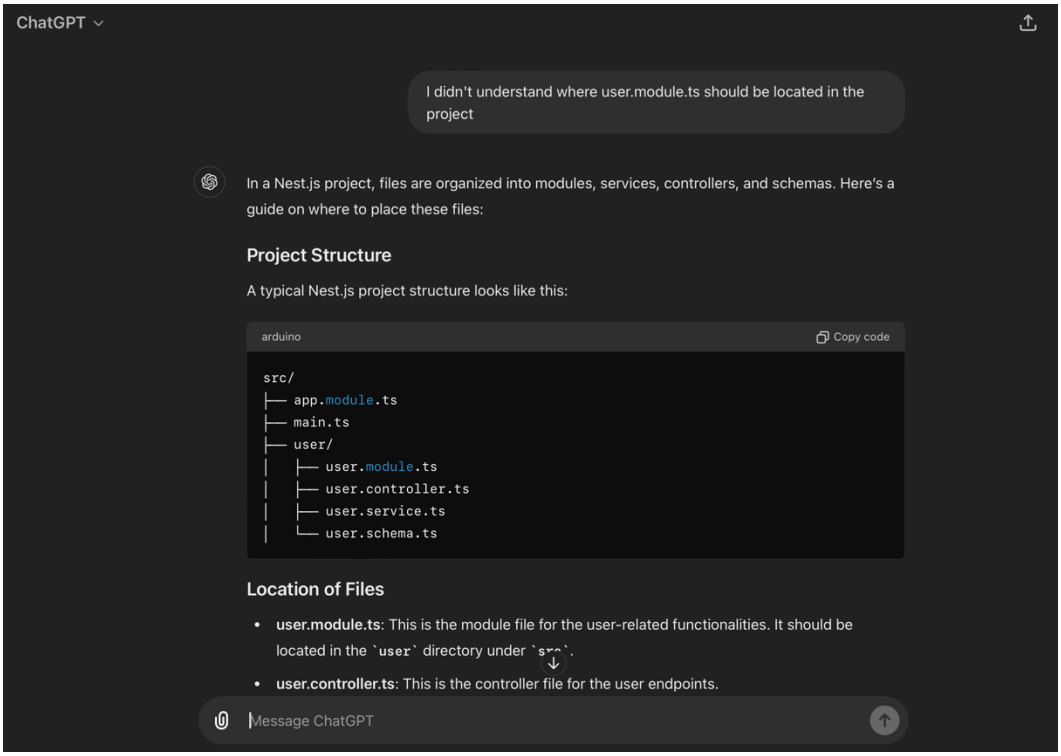

Figure 3*: ChatGPT usage*

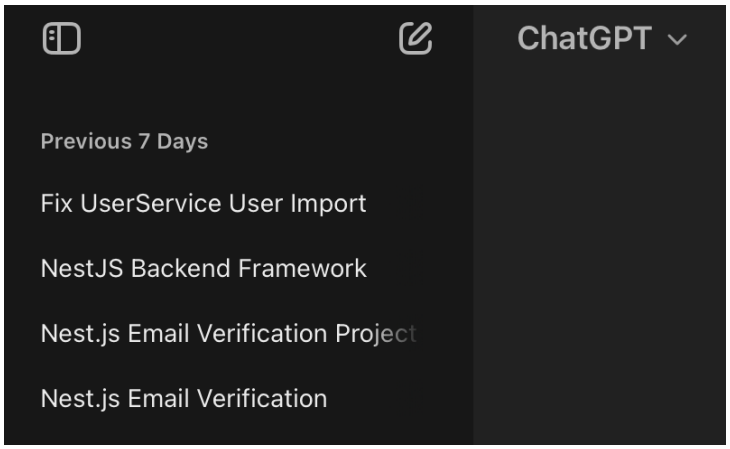

Figure 4: *ChatGPT usage history in Previous 7 Days*

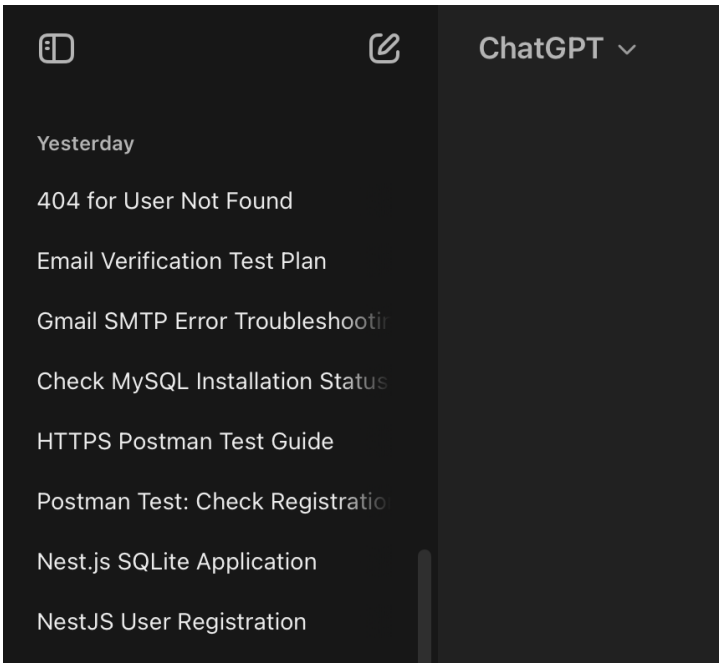

Figure 5: *ChatGPT usage history in Yesterday*

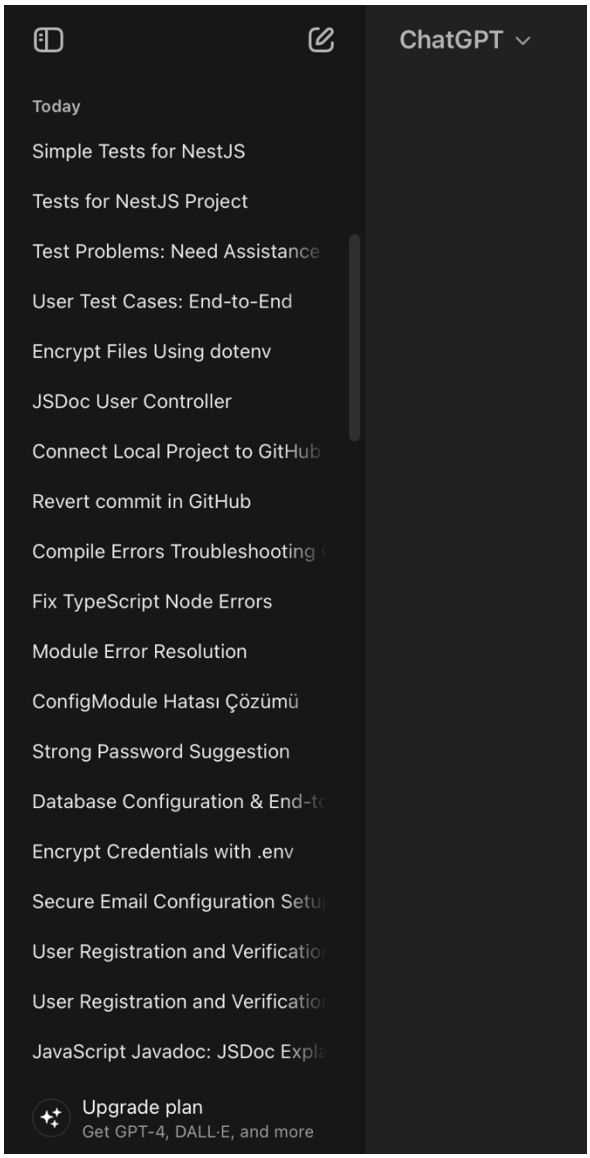

Figure 6: *ChatGPT usage history in Today*

Architecture of the Project

The project's code structure and file organization adhere to clean code principles, making it straightforward to comprehend, efficient, maintainable, scalable, debuggable, and conducive to refactoring. Furthermore, the implementation of NestJS decorators, including @Controller, @Post, and @Get, aligns with the framework's design patterns, promoting declarative route definition and streamlined request handling. Comprehensive type annotations and JSDoc comments provide invaluable documentation, elucidating function signatures and expected behaviors.

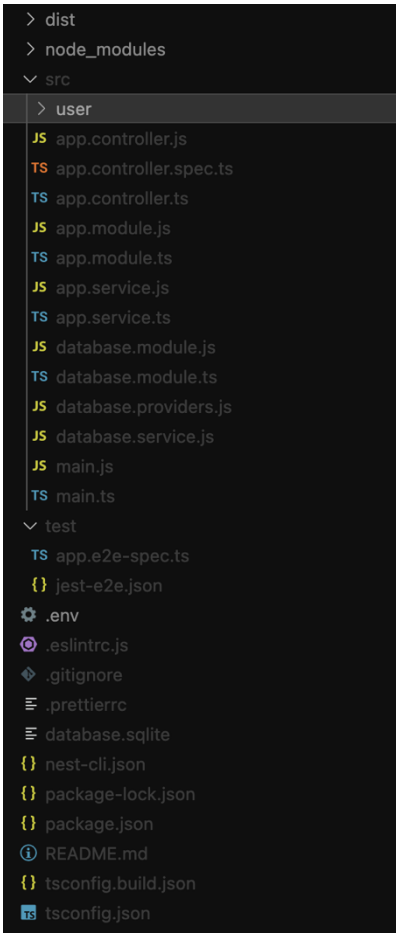

***Figure 7**: Project Architecture*

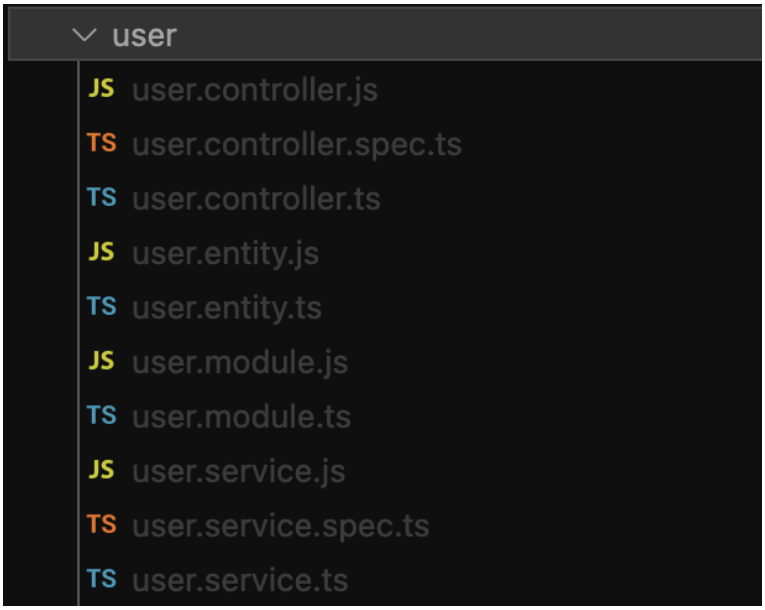

***Figure 8:** Project Architecture (User Module in detail)*

Selection of Technologies and Tools

The Software Developer Internship Case required Nest.js for the framework and due to the fact that it is a versatile and powerful framework in APIs and web development applications, Nest.js separates from other frameworks such as Express.js, Koa.js and Fastify etc. For the development IDE, Visual Studio Code is used due its simplicity, easy-to-use design, and lightweight. The project is made public and distributed to other developers through GitHub, employing GitKraken as a substitute for the Git command line interface.

In this project, the Nest.js framework serves as the backbone, providing a robust foundation for building scalable and maintainable applications. Leveraging various libraries and packages, including '@nestjs-modules/mailer' for email functionality, '@nestjs/typeorm' for seamless integration with TypeORM for database operations, and '@nestjs/microservices' for efficient microservices architecture, the project embodies modern development practices. Additionally, '@nestjs/common' and '@nestjs/platform-express' contribute to the framework's versatility and ease of use. The inclusion of 'dotenv' ensures secure and flexible configuration handling, while 'typeorm' and 'sqlite3' facilitate database interactions with efficiency and reliability. Development and testing are streamlined with the aid of '@nestjs/cli', '@nestjs/testing', and '@types/jest', ensuring code quality and robustness. The project embraces modern JavaScript/TypeScript development paradigms with 'typescript' and 'ts-loader', while 'jest' and 'ts-jest' enhance testing capabilities. Furthermore, 'supertest' enables comprehensive API testing, ensuring functional correctness. Overall, this project embodies the ethos of modern development, leveraging cutting-edge technologies and best practices to deliver scalable, maintainable, and efficient solutions."

Methodology

1. Nest.js framework and required extensions are installed to the local computer
2. Project architecture created
3. A User folder is created and **user.service.ts**, **user.controller.ts**, **user.entity.ts** and **user.module.ts** files are created. The implementation of those classes are done by utilizing different documentations provided and using ChatGPT.
4. A temporary Gmail account is created for verification emails. Gmail free SMTP is used and app password is taken. The authentication information is entered in **user.module.ts** file.
5. For testing purposes, MySQL server and MySQL Workbench is installed. A temporary database is created.
6. Postman is downloaded and the test cases are created as requests to check
7. Based on the test cases, the necessary debugging is done, and several errors are handled.
8. A public GitHub repository is opened and the local files are uploaded into the repository
9. After the project is completed, the following files are uploaded with the initial commit named: "The last commit is reverted with the original files"
10. For a better documentation, several classes are documented using JSDoc.
11. The authentication information is encapsulated using dotenv package of the Nest.js framework and the necessity information is kept in an .env file.
12. Lastly, the test cases are added. (I was unable to run the test cases due to version differences. If it happens for the person who checks my code, I will provide the .env file for a manual test if necessary)

Example Run & Result

Following screenshot displays the initial version of the database table:

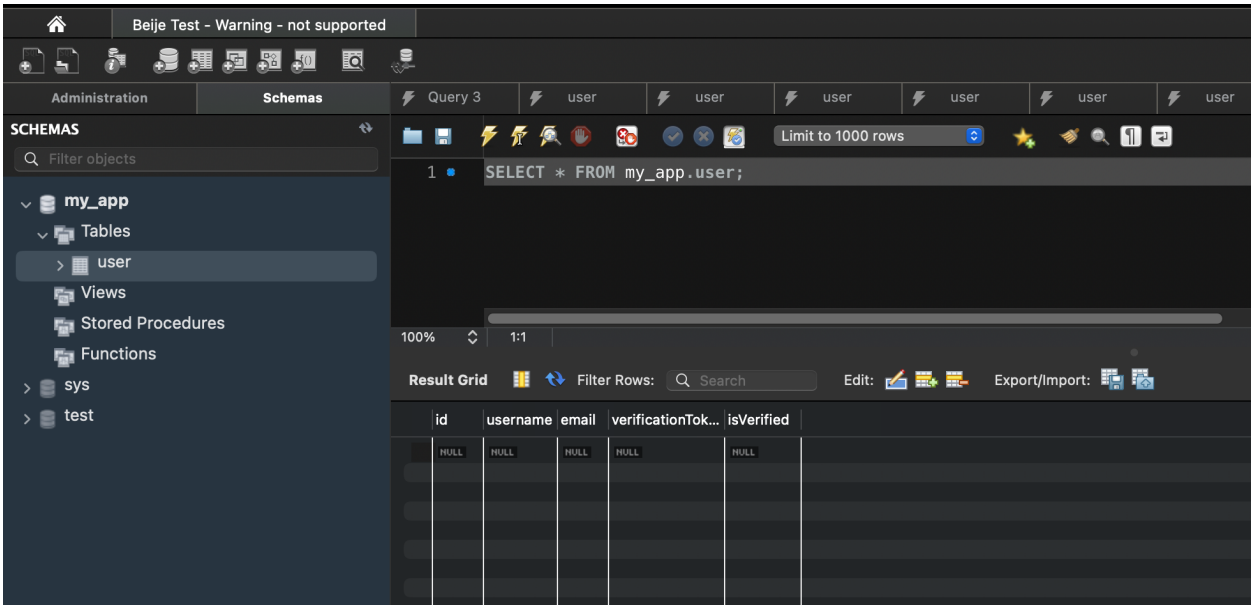

Figure 9: *Initial Database table*

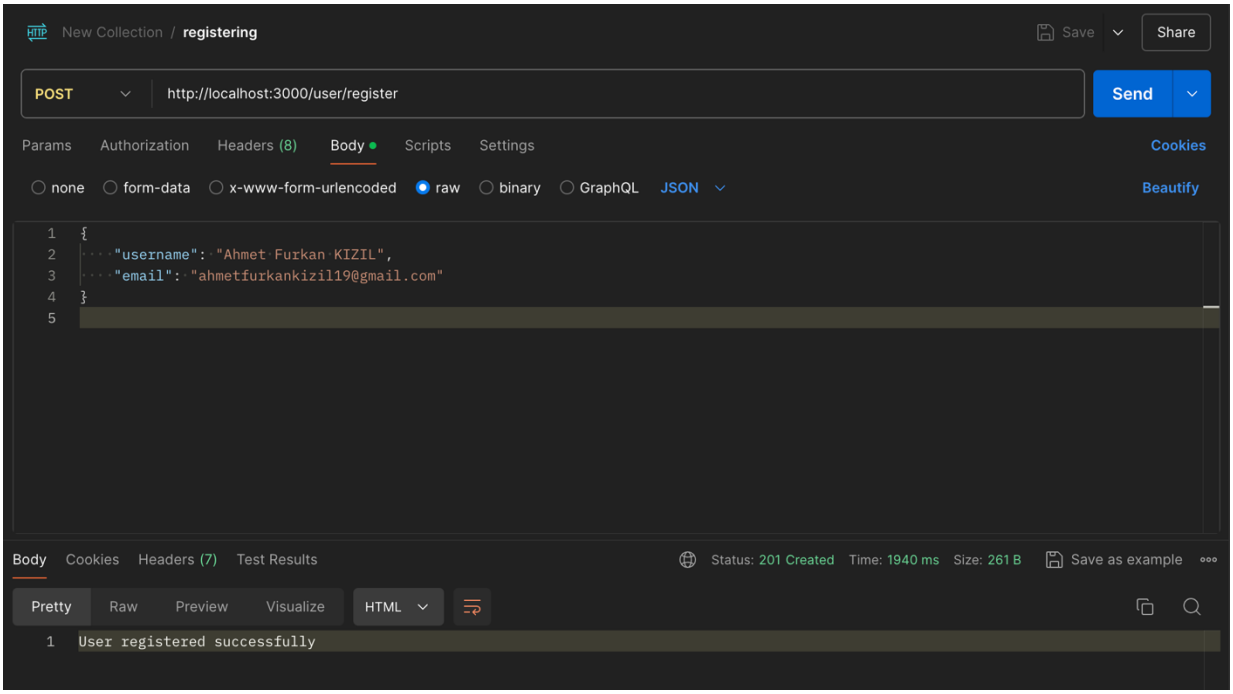

Figure 10: *Sending post request*

The screenshot above shows the post request created by using Postman for the following instruction in the case file:

- **POST /user/register:**
  - - POST parameters: { username, email }
  - - Action:
    - - Create a random alphanumeric value as verificationToken
    - - Save **{ name, email, verificationToken, isVerified }** in database
      - - Any database can be chosen
      - - isVerified is always false when the user is newly registered
    - - Send verificationToken to user's email address

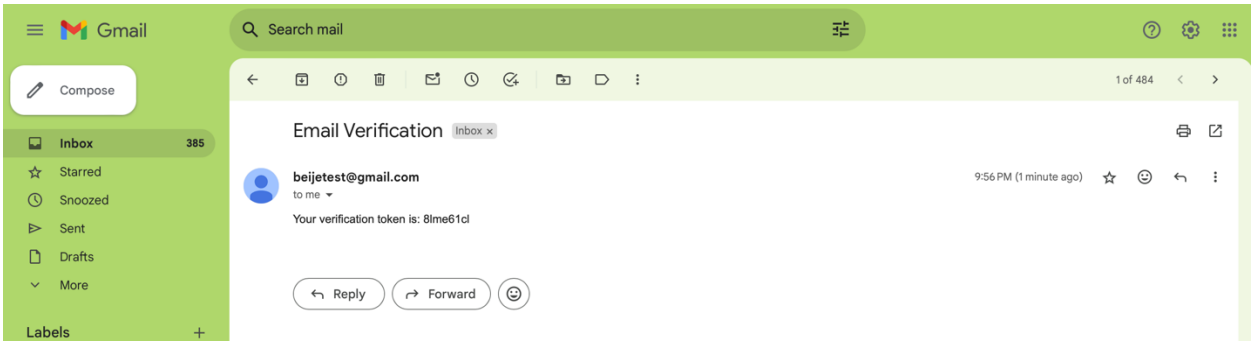

Figure 11: *Verification Token is sent to the Email adress*

The verification token is successfully sent to the entered email as seen from the screenshot above. The following screenshot shows the change in database:

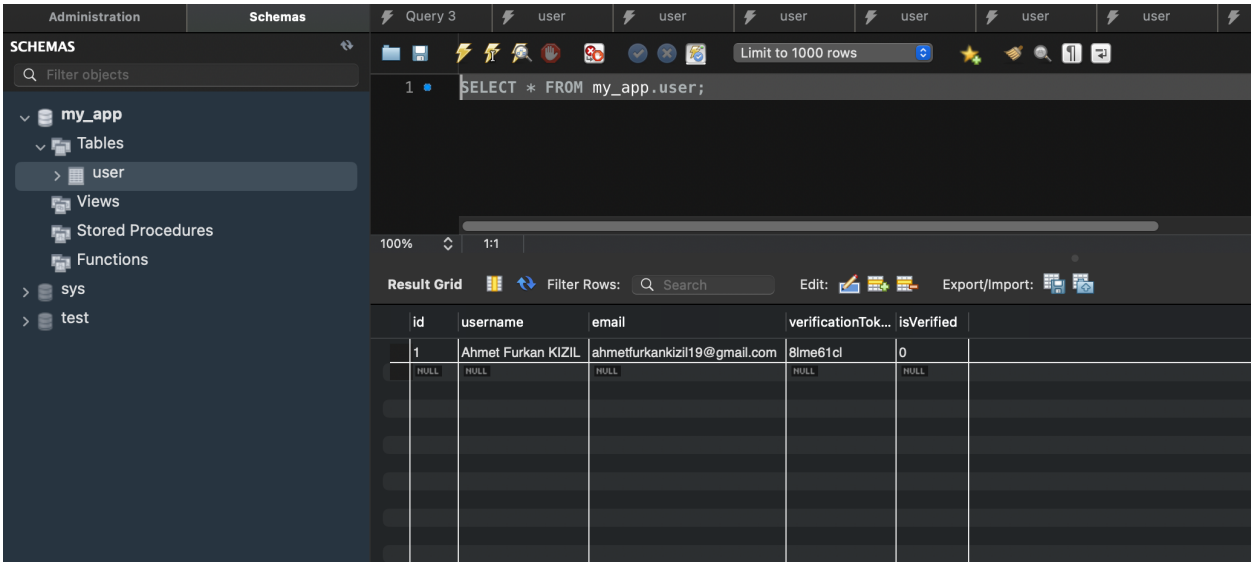

Figure 12: *The database after the post request*

The following screenshots gives the application of the given requirements:

GET /user/verify-email/{username}/{verificationToken}

- GET URL parameters: { **username**, **verificationToken** }
- Find the record with the **username** from database
- If record didn't exist, return **404 not found** response

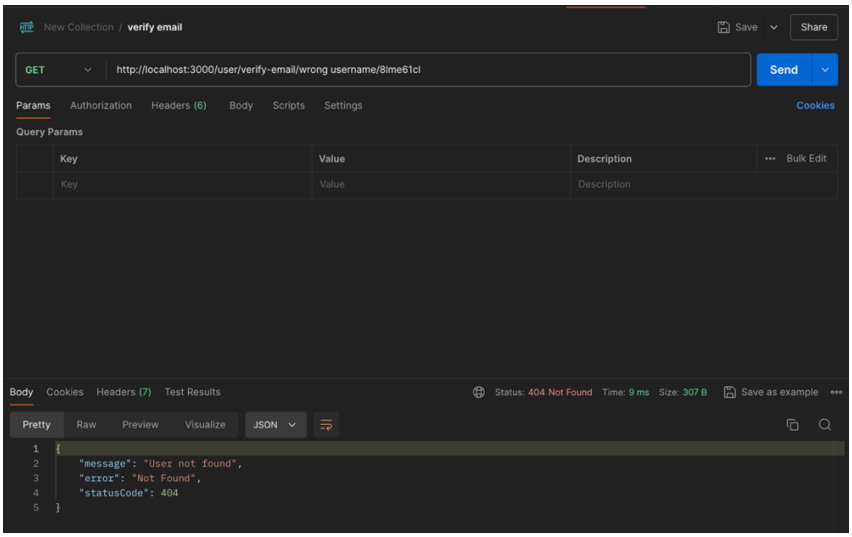

***Figure 13**: Sending GET /user/verify-email request (User not found)*

- Check **verificationToken** in url parameter is equal to the database record
- If it wasn't equal, return **400 bad request** response

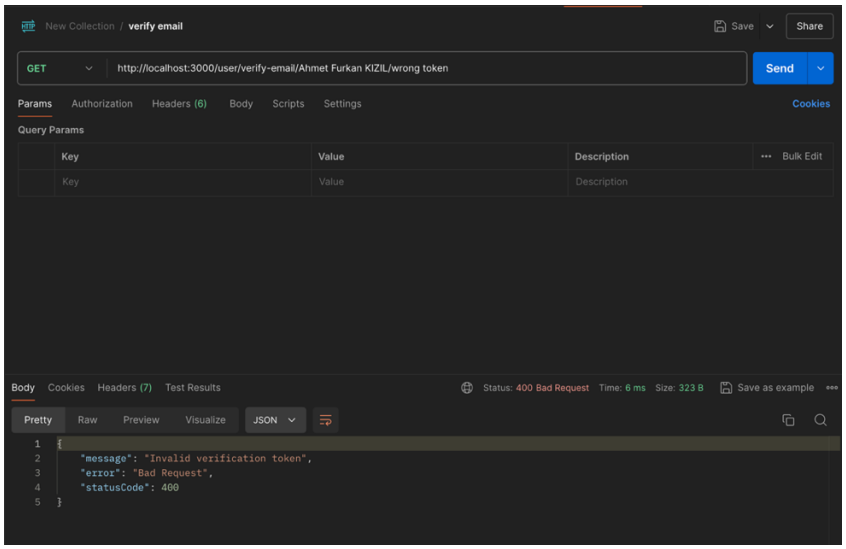

***Figure 14:** Sending GET /user/verify-email request (Invalid verification token)*

- - Set **isVerified=true** in database record of the user
- - return **success** response

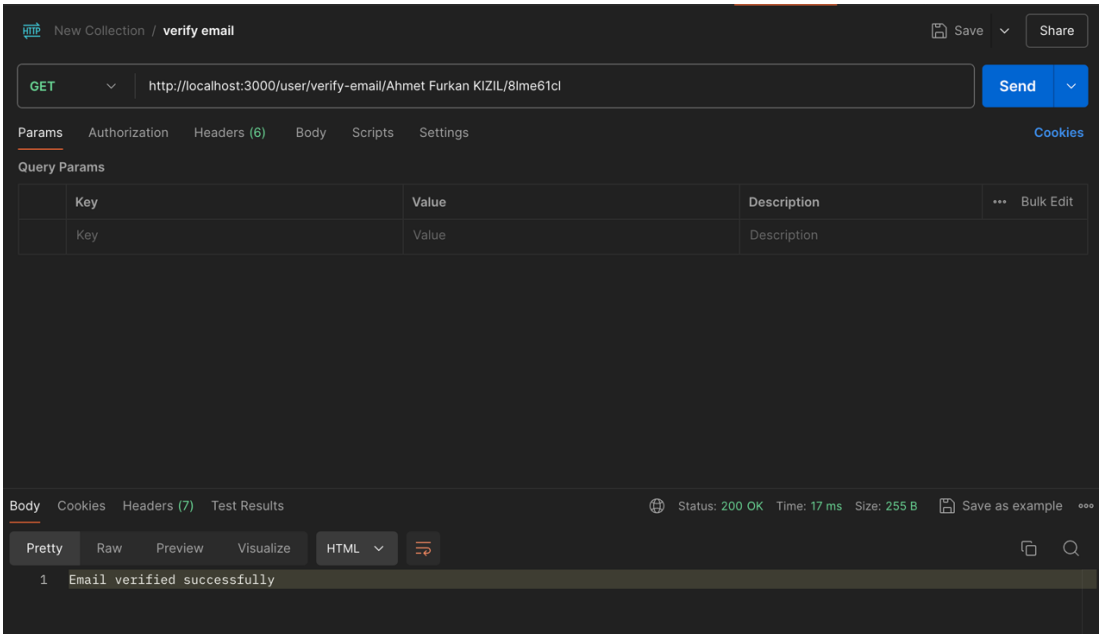

***Figure 15**: Sending GET /user/verify-email request (Success response)*

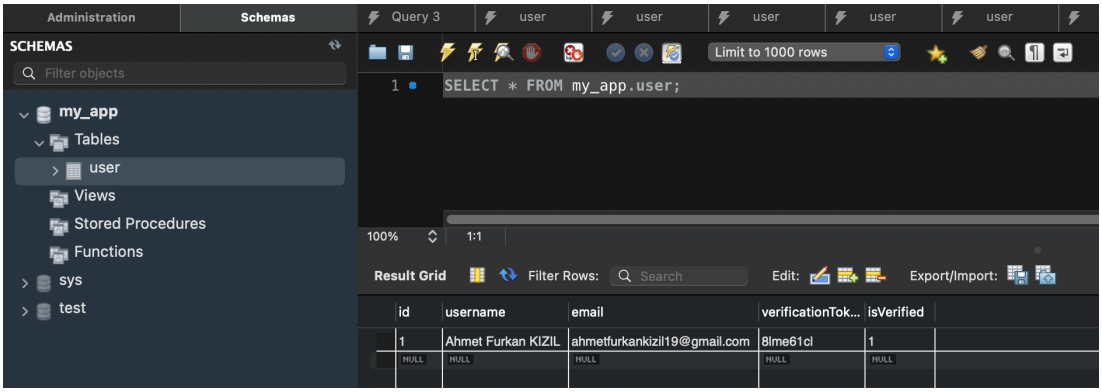

***Figure 16:** Database table after success response (isVerified is set to true {1})*

- GET /user/check-verification/{username}:

- - GET URL parameter: { **username** }
- - Find the record with the **username** from database

- If record didn't exist, return **404 not found** response

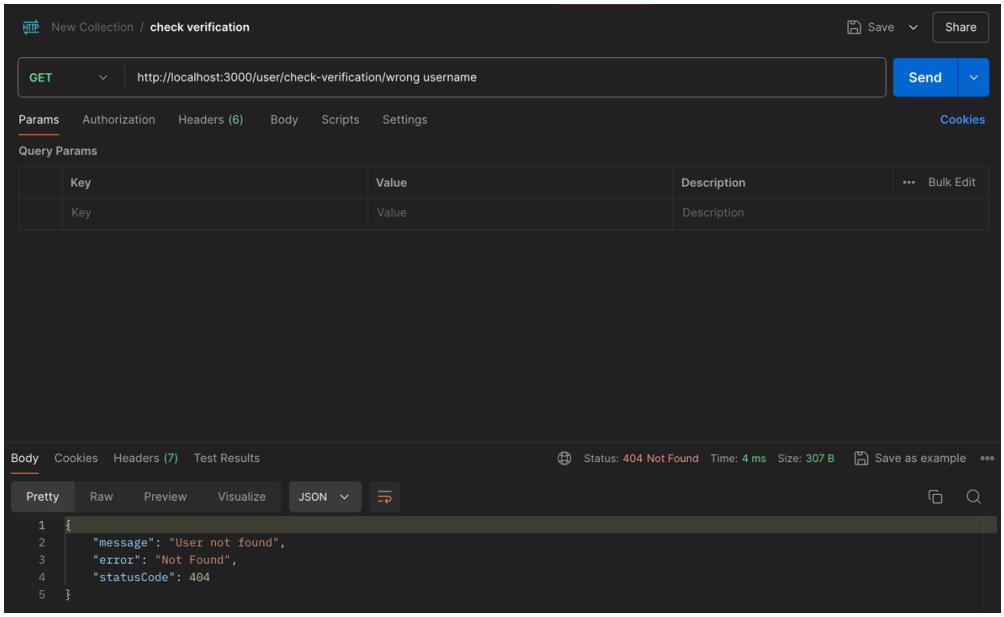

***Figure 17**: Sending GET /user/check-verification request (User not found)*

- Check **isVerified** equals true in the database: If it wasn't return "user is not verified"

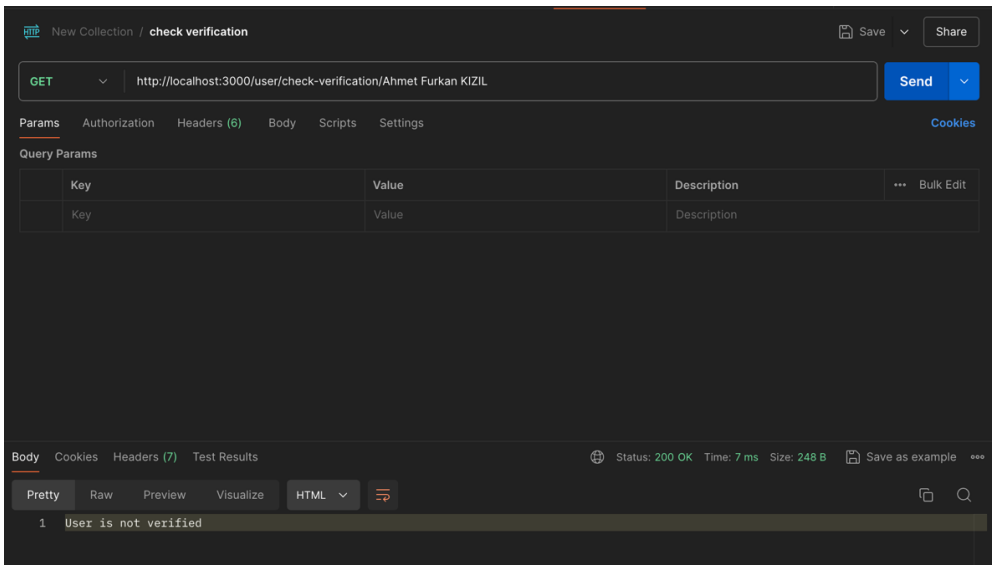

***Figure 18:** GET /user/check-verification (User not verified)*

- Check **isVerified** equals true in the database: If it was, return “user is verified”

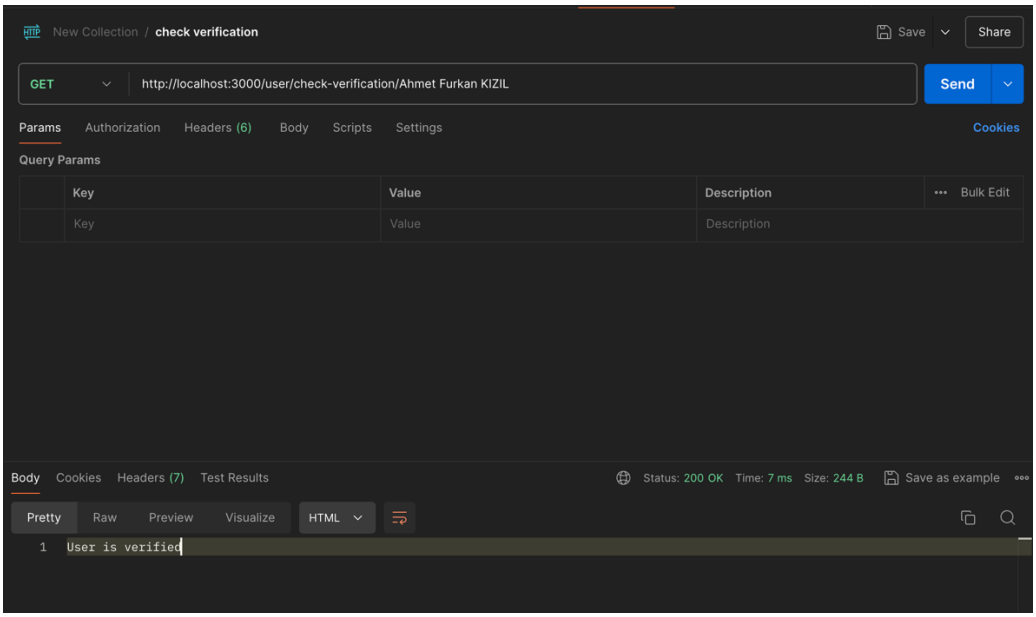

Figure 19: *GET /user/check-verification (User is verified)*