xaver

Full Stack Intern - Challenge

April / May 2024

Documentation Report

Assignee: Ömer Onat Postacı / onatpostac@gmail.com

TABLE OF CONTENTS

Introduction.....2
General Architecture.....2
Possible Scalings.....3
How to Run the Application.....3
References Used In the Implementation Phase.....6

Introduction

First of all, I would like to start this report by thanking you for considering me as a candidate for this position and giving me the opportunity to provide my skills with a use-case. This challenge was amazing for not only demonstrating our coding skills, but also demonstrating quick-learner and quick-implementing skills because even though it seems easy, it consists of certain important phases. I allocated 4 hours 20 minutes for developing this application. Even though it could have been developed in a shorter time, providing a scalable and understandable code was important for me.

General Architecture

In this challenge, there are three layers communicating in an order. Basically, we have a UI Layer (Client Side), a Logic Layer (Backend Side), and a Data Layer (MongoDB Cloud). UI Layer communicates with the Logic Layer, then Logic Layer has different services inside of it and it communicates with the Data Layer. Therefore, we can state that the most basic architectural pattern that we adopt for this challenge is Three-Tier Architecture. However, there are internal functionalities inside of the Logic Layer. As mentioned in the Challenge document, we have a Web Socket service that enables real-time communication between the client side and the server side. Therefore, in terms of this specific interaction, we have a Publish-Subscribe Architectural Pattern even though this pattern mostly relates with distributed systems because both services subscribe to themselves and publish messages to each other.

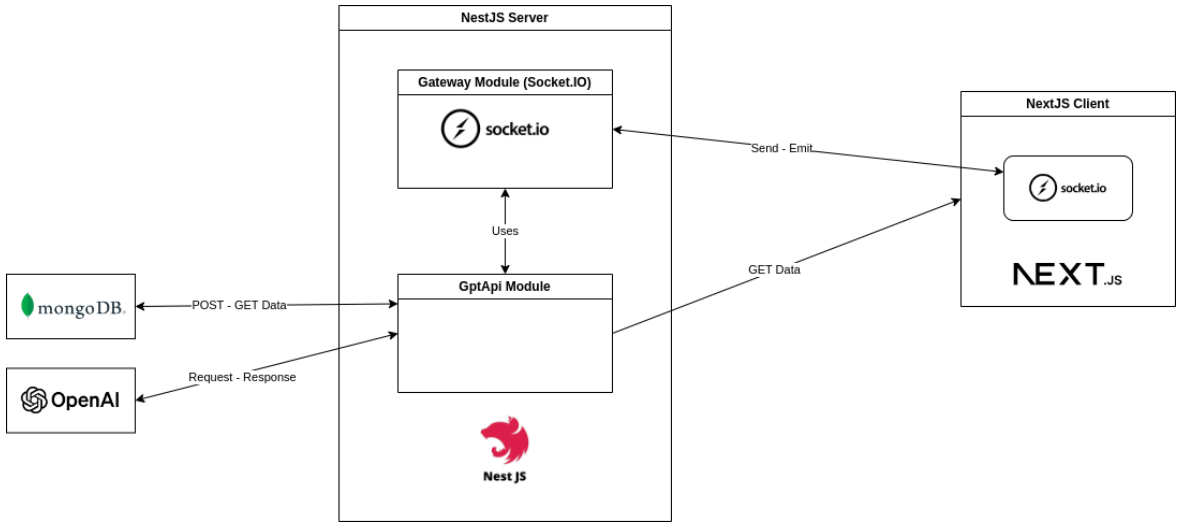


Fig 1: System Decomposition UML Diagram of the System

As demonstrated in Fig 1, we have a 3-tier architecture as well as a pub/sub architecture to ensure communication between the two layers (UI Layer and Logic Layer). Therefore, the latency was

decreased. Instead of waiting the whole request response lifecycle, the client side received the necessary data through real-time communication.

Possible Scalings

Undoubtedly, this application can be scaled since it currently only compensates for basic requirements. There are certain possible scalings that can be implemented to have more cumulative and complex application. For example, in this application, we are using the OpenAI API key to communicate with the GPT API. However, for such costly actions, authentications would be used to filter the users that only authenticates themselves to use our product. Hereby, we can eliminate the bots and unnecessary API requests through an authentication middleware. Another possible scaling might apply to access and recovery of recently deleted implementations. Using a second collection for deleted items and recovering implementation could have been developed. In terms of Frontend side, a Storybook could be formed for generic components of Xaver for providing a modular roadmap for the frontend. Furthermore, Last but not least, we could implement an infrastructure to deploy the application. For this purpose we could use AWS Lambda Functions with a Layerization to our backend service, and then use AWS API Gateway REST and WebSocket configuration to trigger the Lambda function when needed. Hence, we could get a costly optimized live backend service.

How to Run the Application - Server Side

The process to guide tho executing the application is also delivered in the README.md file; however, it is also rational to mention it in this report.

1- Cloning the Repository

The following command will be executed to clone the server repository:

```
git clone https://github.com/onatpostaci/casestudy-backend.git
```

2- Install the necessary NPM packages

The following command will be executed to install the necessary NPM packages:

```
npm install
```

Now, we have one more last step to getting ready before running the server:

3- Create a new file in the server folder called “.env” and the content of the “.env” file can be (API Key is the provided key):

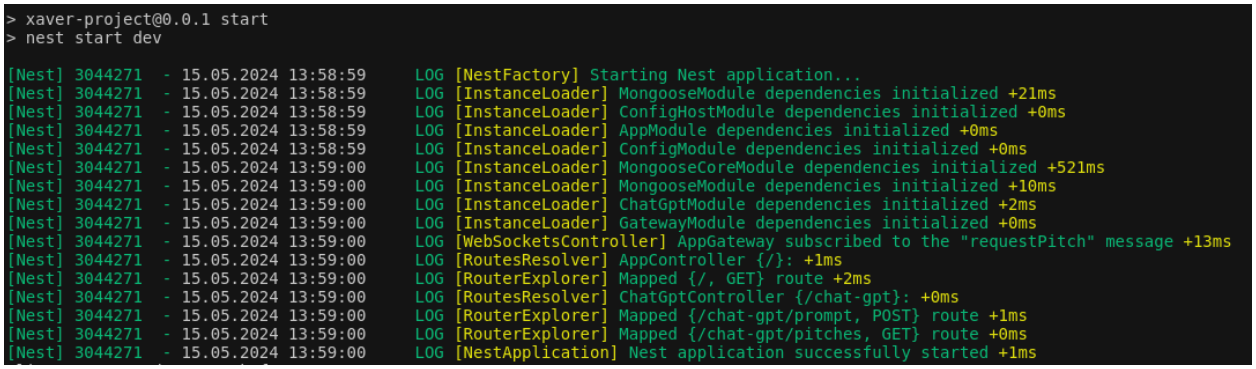
```
OPENAI_API_KEY="sk-proj-GLEFe12Qvp1NgY6sLeR2T3BlbkFJ45hnZu0td23vYreq5KBO"
PORT_NUMBER=3000
MONGODB_CONNECTION_STRING="mongodb+srv://onatpostaci:Xaver1243@xaverdatabase.ohennli.mongodb.net/?retryWrites=true&w=majority&appName=XaverDatabase"
```

4- Ready to run the server:

Now, we are ready to run the server. The necessary command is:

npm start dev

After, executing the command, the following should appear in the terminal:



Checkpoint: Server started on the specified port in the .env file successfully.

How to Run the Application - Client Side

The process to guide tho executing the application is also delivered in the README.md file; however, it is also rational to mention it in this report.

1- Cloning the Repository

The following command will be executed to clone the client repository:

https://github.com/onatpostaci/casestudy-frontend.git

2- Install the necessary NPM packages

The following command will be executed to install the necessary NPM packages:

```
npm install
```

Now, we have one more last step to getting ready before running the client:

3- Create a new file in the client folder called ".env.local" and the content of the ".env.local" file can be (SAME PORT WITH SERVER):

```
NEXT_PUBLIC_API_URL=http://localhost:3000
```

4- Ready to run the client:

Now, we are ready to run the client. The necessary command is:

```
npm run dev
```

After, executing the command, the following should appear in the terminal:

Checkpoint: Server started on the demonstrated port in the terminal successfully. Now, using the local URL in the above Figure, the application can be reached.

Once again, thank you for your time, consideration, and interest. I hope I developed a comprehensive application and it meets your needs. I am looking forward to meeting you in-person.

References Used In the Implementation Phase

[1] “Documentation: Nestjs - a progressive node.js framework,” NestJS, https://docs.nestjs.com/ (accessed May 16, 2024).

[2] “How to use with next.js,” SocketIO RSS, https://socket.io/how-to/use-with-nextjs (accessed May 16, 2024).