IUM-TWEB project report

A. Buscema, L. Ferrero Merlino

1 Introduction

Our project combines different tools to run things smoothly. Angular shapes the front-end, Express in the Main Server manages requests, and Java Spring Boot with PostgreSQL organises and stores data. Another Express server deals with MongoDB, handling lots of data. Jupyter Notebook helps us visualise and understand the data. Each part has a role, and together, they make a practical and well-rounded project.

2 Angular

We used Angular as a frontend framework for developing the user interface, handling the HTML, CSS, and JavaScript aspects. It's beneficial for creating single-page applications due to its property binding. This feature came in handy for implementing a translation feature without the need to refresh the page.

Solution

We chose to use Angular for a better organisation of code across all pages. We have some experience with this framework from our previous use. Additionally, it is easy to integrate open-source libraries like moment.js for date and time manipulation and Chart.js for visualising data using npm.

Issues

The main challenge is that an Angular project needs to be compiled to obtain the HTML, CSS, and JavaScript files for hosting. Therefore, we have uploaded a compiled file of the latest developed version. In case there is a need to make changes, we've included a Windows batch file for a quick build.

Requirements

We completed everything the assignment asked for and created a nice, well-written application.

Limitations

The only limitation is that Angular has an high apprendition curve, it needs a lot of experience to write a clean and optimised code.

3 Main server (express)

The main server handles all frontend requests, relying on other servers to interact with the databases. It needs to be fast to manage potential high loads, so it delegates database requests to other servers. Additionally, it has the function of managing the chat rooms.

Solution

We chose the monolith architecture for our Main application because it's simple and doesn't have too much logic inside. The structure of our monolith architecture is as follows:

- **Routes**: These serve as endpoints for APIs, managing both the logic and data retrieval from the other servers.

An overview of the general request-handling process in the Express application is as follows:

1. The request arrives at the appropriate endpoint based on the route.
2. The function verifies the data for validity.
3. The function establishes a connection with the servers and fetches the data.
4. The data is then sent back to the client.

Issues

The main challenges we encountered with our Main application were connected to the Swagger implementation.

Requirements

We completed everything the assignment asked for and created a working application.

Limitations

The main downsides of the Main server using Express are the single-threaded nature of Node.js and the higher likelihood of making mistakes due to the lack of types and no compiled behaviour in JavaScript.

4 Java Spring Boot

We utilised the Spring Boot server to manage persistent data in our PostgreSQL database, structured with the MVC architecture. We also use swagger to expose documentation for the available APIs.

Solution

We chose to implement the MVC architecture in our Spring Boot application for a specific reason: it is the most common pattern for API solutions. The structure of our MVC architecture is as follows:

- **Controllers**: These act as endpoints for APIs, handling the logic to construct Data Transfer Objects (DTOs).
- **Services**: Controllers utilise services to retrieve raw data from the database.
- **Repositories**: These classes are the only one to interface with the database, facilitating communication between the application and the database.
- **Entities**: Representations of database tables, entities define the structure of the data.
- **DTOs**: Objects returned by the API, encapsulating the data being transferred.

An overview of the general request-handling process in the Spring Boot application is as follows:

1. The request lands at the corresponding controller based on the route.
2. The controller requests the required data from the services.
3. The service, in turn, calls the repository to retrieve the data from the database.
4. The data is cleaned (if needed) by the service before returning it to the controller.
5. Once the controller receives the requested data, it constructs the DTO for the response and sends it back.

Issues

The main challenges we faced with our Spring Boot application were related to the overall structure and how it communicates with the database.

Creating a solid architecture is no easy feat, but with examples from our professor and online resources, we managed to build a flexible and effective application.

We also ran into some issues with repositories, especially when trying to insert data, it wasn't as straightforward as we expected. However, after searching online for solutions, we figured out how to make the insert work.
Another problem popped up with queries that returned a custom entity that is not a database table. The database checker during the build process threw errors, complaining that the entity didn't exist as a table in the database. To solve this issue, we decided to turn off the database checker.
Also we have to specify for each custom entity a new repository.

Requirements

We completed everything the assignment asked for and created a nice, well-written application.

Limitations

The main drawbacks of Spring Boot aren't related to flexibility, a delicate structure, or slow request handling. Instead, it's the fact that it's quite heavy. For simple APIs like these, it's not necessary. An Express server, for this type of task, is lighter, quicker to write, and overall easier.

5 PostgreSQL

We utilised the PostgreSQL database as SQL database.

Solution

We chose to use an SQL database to manage long-term data and information that requires accuracy without loss.

Issues

The main challenge we encountered was with the import/export of the database when we first installed it. We resolved this issue by reading the online documentation, finding out that pgAdmin 4 required a manual entry of the bin path for the SQL server.

Requirements

We completed everything the assignment asked for.

Limitations

If the database accumulates a lot of data and lacks the correct indexes, data querying can become slow.

6 Second server (express)

We utilised the Express server to manage our MongoDB database, structured with the monolith architecture. We also use swagger to expose documentation for the available APIs.

Solution

We opted to go with the monolith architecture for our Express application, choosing it simply for the sake of using something different from MVC and for the no brain approach (also because it is easy and fast to write). The structure of our Monolith architecture is as follows:

- **Routes**: These serve as endpoints for APIs, managing both the logic and data retrieval from the database.

An overview of the general request-handling process in the Express application is as follows:

1. The request arrives at the appropriate endpoint based on the route.
2. The function verifies the data for validity.
3. The function establishes a connection with the database and fetches the data.
4. The data is then sent back.

Issues

The main challenges we encountered with our Express application were connected to the Swagger implementation and the distinct features of NoSQL.

Comparing the Swagger implementation to Spring Boot, it was very different. In Spring Boot, it was easy, just installing the package. In Express, you have to write everything by hand.

Understanding the unique syntax for communicating with the NoSQL database posed a significant challenge. However, after reading a lot of documentation and searching online, we were able to complete the task.

Requirements

We completed everything the assignment asked for and created a working application.

Limitations

The main downsides of Express are the single-threaded nature of Node.js and the higher likelihood of making mistakes due to the lack of types and no compiled behaviour in JavaScript.

The primary limitation of the monolith architecture is that it's not easily scalable, and if it becomes too big, managing it can become quite challenging. However, it's straightforward to implement and is a perfect fit for these kinds of APIs.

7 MongoDB

We utilised the MongoDB database to manage high volume data.

Solution

We chose to use a NoSQL database to optimise storage and handle large collections of documents more efficiently.

Issues

The primary challenge we faced was with the import/export of the database. We managed to overcome this issue by reading the online documentation.

Requirements

We completed everything the assignment asked for.

Limitations

We didn't explore too deeply into the NoSQL world to form opinions about limitations with this technology.

8 Jupyter Notebook

We used Jupyter to collect and present data in plots using various Python libraries.

Solution

Using Jupyter is great because it allows for very fast iteration among the data.

Issues

We haven’t got any particular issue with Jupyter

Requirements

We didn't encounter any specific issues with Jupyter.

Limitations

We didn't explore Jupyter too much to notice any limitations.

9 Conclusions

Having good partners and some experience makes creating a nice website quite straightforward. A capable team and shared knowledge help things move quickly and efficiently. With experienced collaborators, you can navigate challenges smoothly, and familiarity with the process speeds up development. In short, a solid team and past experience are key in delivering a well-crafted website in a reasonable time.

Division of work

Antonio Buscema (Yotsumi) and **Leonardo Ferrero Merlino** (leonardo-fm) split the work as evenly as they could.

Antonio Buscema worked on:

- Angular
- Main server (express)
- PostgreSQL
- Jupyter Notebook

Leonardo Ferrero Merlino worked on:

- UX/UI
- Spring Boot server
- Second express server
- MongoDB
- Jupyter Notebook

The most time-consuming part of the project was Angular, followed by Spring Boot, the second express server, the main server and the databases.

Extra information

In the repository, we have a Readme.md file that provides all the information needed to run the project.