Backend

As backend developer we were responsible for the server-side logic of web applications, connection of endpoints, routing, setup of a database, connection to APIs, integration of other tracks code and retrieving, updating, storing data. We were also in charge of testing and debugging the server-side code and check if everything is working.

The top three tools we have used the most are listed below:

#1 Visual Studio Code

To read, create and edit the source code for our project we used the code editor Visual Studio Code from Microsoft. The reasons why we used Visual Studio Code are that the software is for free, it runs on different operating systems, it has a huge variety of helpful extensions and most of the tutorials and learnings we checked used Visual Studio Code as code editor. In addition, Visual Studio Code has a good working and easy to understand interactive debugging console which helps to identify errors or mistakes in the code to correct them.

#2 Postman

Postman is a platform of several tools like design, test, mock, monitor and document to accelerate the work with APIs. We used Postman mainly to check if responses of requests to our server are returning the right information we need for our project.

#3 MongoDB Atlas

To check and store our results and information we used the multi-cloud database called MongoDB Atlas. That database supports all MongoDB features, supports JSON files and schema, has a live migration and can be used and accessed by several people.

To fulfill the project requirements addressed to the backend track we worked with different libraries as you can see in screenshot “libraries” and used the programming language Java Script.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Before we started coding, we needed to draw the wireframe of the website and the server-side logic behind to get a basic understanding what kind of code and data we need to provide to the frontend team and what we need to receive from the other tracks to make our project code work. Hence a lot of meetings and coordination between the frontend and backend team took place.

After setting up the server and installation of the libraries we created a login with registration. For us it was not so easy to know which route to create and how to retrieve the user ID after the login. After quite a few attempts to make the code work we decided to create a normal login which is just checking the username and password and a login after taking the quiz which we need to process the users answers as you can see in the picture “Login”.

Ein Bild, das Text enthält.

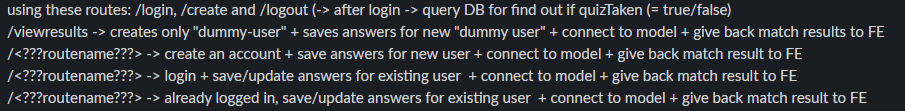
Automatisch generierte Beschreibung

We decided to create user accounts for “non-users” in our database but without saving their email and password information. The idea was to use the objectID as a “user-ID” for identification when we save the users answers in our database. Another obstacle was that we were not able to create a user with “null” as email value after the second try. The reason was that we set the email as unique in our user schema and when we have the “null” value for the email more than one time in the database the code is translating it as a duplicate value. We solved that issue by adding another parameter called “sparse” to the user schema which allows us to have duplicate null value for email. We also needed to coordinate, discuss and think about which information should be part of the user schema which you can see in the picture “User schema”.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

For the answers and the match quiz we also created schemas. While coding and seeing the project grow, we realized that we need more routes, and the possibilities of options and outcomes also became more as you can see in the extract of our working file “Route ideas”.



Beside a lot of trials and errors and moments of enlightenments we had also moments in which the code worked somehow but we could not figure out why. For example, in the screenshot “connect to model” you can see our code we have created to connect to the API-model which was created by the data science track. We faced the problem that the API was working on postman but when we tried to integrate it to our code it was not working on our machine. So somehow the code was working but we could not access the result given back from the API. To solve that issue we needed to add .data after modelOutput to access the result from the API.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We faced a lot of time pressure as the backend track could not really start in the beginning as the whole setup of the website was not clear.

All in all, it was fun to find a solution for all the obstacles we were facing and to get rewarded with a code which is functioning. Also, the meetings with the other tracks were interesting as we got an insight of their work when we needed to create routes or connect endpoints. Most of our problems we were able to solve with the help of our TechLabs mentors, video tutorials, code snippets we found on the internet, tutorials on websites, meetings with the other track members and by just trying.