A FULL STACK WEB APPLICATION: IN PRACTICE

Working with a full stack application and applying UX (IPS & GPS)

Author: Maurice Schippers
Class: S3-DB03
Date published: 12th of January 2023
Version: 1.0

CONTENTS

1. INTRODUCTION

This document is structured to feature everything I have done for Learning Outcome 01: Web Application. The major focus will be on my work in the group project, which also contains a section about how I applied UX there and where I got my information and inspiration from. For IPS, I have written about my research and decision regarding the different JavaScript frameworks and my experiments with the different backend technologies & ORM tools. I used footnotes to add any additional information or direct sources where I did not use any paraphrasing or citations. At the end of this document, there is a small section about the references I have used in case of direct citations or paraphrasing.

Bold and underlined text in the paragraphs is clickable and will take you to the referenced section.

2. GPS – IS IT LIVE?

While I spend most of my time working on the frontend, other related components or ‘stacks’ have also been covered during the semester. Additionally, I did quite some research on the topic of UX and I have tried to implement and showcase it in several ways. The project concerns an application that is able to check whether certain content is live on another website.

1.1 Frontend

We are using Vue for our JavaScript framework; this is because the company uses Vue for their projects and specifically asked for scalable solutions. I worked on converting simple wireframes to HTML and CSS during the first two sprints. The following screenshots are an example of what I worked on:

Screenshot of an early high-fidelity wireframe of the retailer – products page.

Screenshot of the conversion process.

1.1.1 Dashboard

I spend the last two sprints working on frontend design, specifically designing the dashboard page for the application: The first page the user sees. Together with the product owner, we identified some of the most important aspects of the application, so we decided to display a couple of these on the dashboard page in a visually pleasing fashion. The **Pie charts** and the **Data table** were the results of this. It took a lot of time for me to make the charts work, mostly because of the data that it needed to display. The data table was also more work than I expected because it required so many functions: Sorting, filtering, searching for products and a user should be able select a product and be taken to the corresponding page

of the selected product. More about these components can be read in the **Don't make them waste time** section.

1.1.2 Asynchronous communication

The **Pie charts** component required asynchronous communication to work correctly. To create the charts, we made use of the Chart.js library. Once the component parts are imported from the library, the application would proceed to 'sketch' the charts. However, these charts are working with certain data that must be processed first. To make that happen, I used the Async and Await pattern to allow the application to gather the data first before displaying the charts. With the help from the build-in function "v-if", I was able to set a simple Boolean determining whether everything is ready to be displayed:

```
<Pie v-if="loaded" :data="retailerChartData" />
```

And the asynchronous function:

```
    async mounted () {
      this.loaded = false

      try {
        await axios.get('http://localhost:3100/retailer/getscore')
          .then(data =>
          {
            console.log(this.convertRetailerData(data.data))
            this.retailerChartData = this.convertRetailerData(data.data)
            this.loaded = true
          })

      } catch (e) {
        console.error(e)
      }
    }
```

1.2 Backend

I spend the fourth sprint working in the two components of the backend. One is built with Express (a Node.js web application framework which communicates with the database and the scraping component) and the other one with Python (used for scraping).

1.2.1 Security checks

After the third sprint review, it became apparent there were not any security checks (checking for duplicate entries or empty values) in the API component of our application. As there are of course multiple ways to get around frontend checks, I took on the task of setting up the security for the CRUD functionalities of the retailer entity. It took more time for me to make it work, because working with TypeScript and an ORM was completely new for me. I learned a lot from creating simple functions with a language that I had never used before.

```
async function doesRetailerExist(retailer:retailer)
{
    if(await repository.findOneBy({name: retailer.name}))
        return true;
}
```

Code snippet of checking for duplicate entries.

1.2.2 Scraping using Google Cache

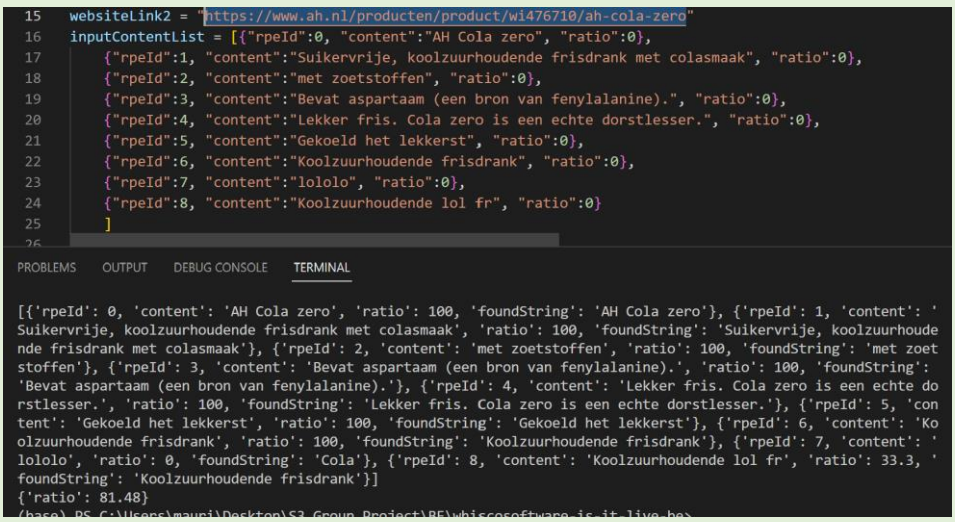
Most of the scraping component was built by other team members. However, when we ran into IP rotation problems and having our requests to scrape information blocked, we came together and investigated different solutions that could still give us the data that we needed. I found one that seemed like one of the most reliable solutions: Google Cache.

Google Cache contains snapshots of webpages and works similar to how cookies save data to increase loading time the next time you visit that page. You can use the 'cached' version of a webpage in case the live version is slow or not responding. However, this solution ended up not foolproof, as the cached version is not an up-to-date version of the website; we cannot report for certain that the data found on the cached version is live on the actual website at the moment of scraping. Furthermore, some websites do not support cached versions of their website. For example, LinkedIn and in our case, the website of Jumbo do not support it. These two reasons are enough to rule out Google Cache as a possible solution.

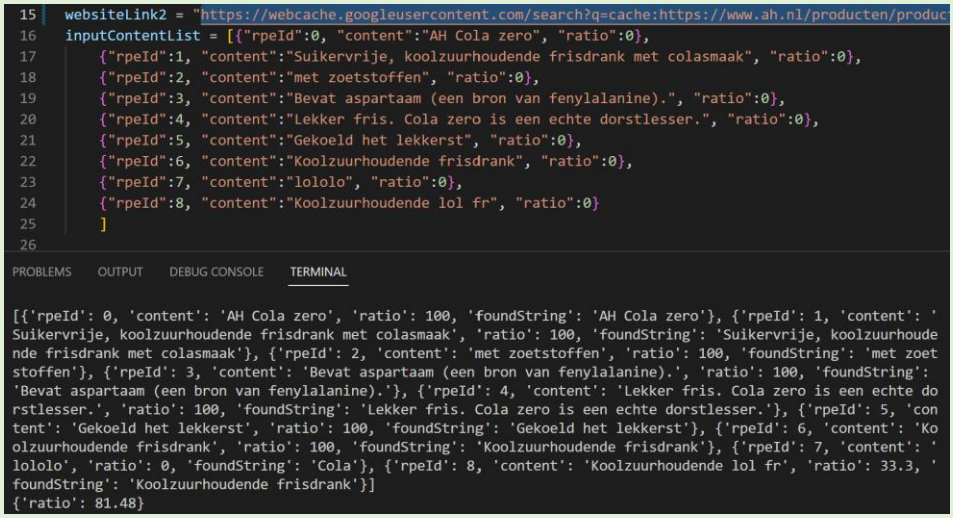
I wanted to test whether my proposed solution could have worked, so I replaced the current way of searching for websites with the following URL:
https://webcache.googleusercontent.com/search?q=cache:

Adding any link of a website after this, like https://www.ah.nl/producten/product/wi476710/ah-cola-zero will take you to the cached version of that website:

https://webcache.googleusercontent.com/search?q=cache:https://www.ah.nl/producten/product/wi476710/ah-cola-zero



Screenshot of the scraper being used on the live webpage of a product.



Screenshot of the scraper being used on the cached version of a product.

It can access all the necessary elements on the page and return the same results. In conclusion, it was worth checking out, but ended up not being the most foolproof solution. It helped me learn a lot about IP rotation, requests and how our scraper application works. Just like TypeScript, this was the first time I have ever worked with the Python language.

1.3 Data persistence

During the first sprint, I worked on the data persistence stack of our project. We discussed whether we would go with a NoSQL or a SQL database and together with another group member, I worked on creating an entity relationship diagram (ERD) for our database.

1.3.1 NoSQL vs SQL

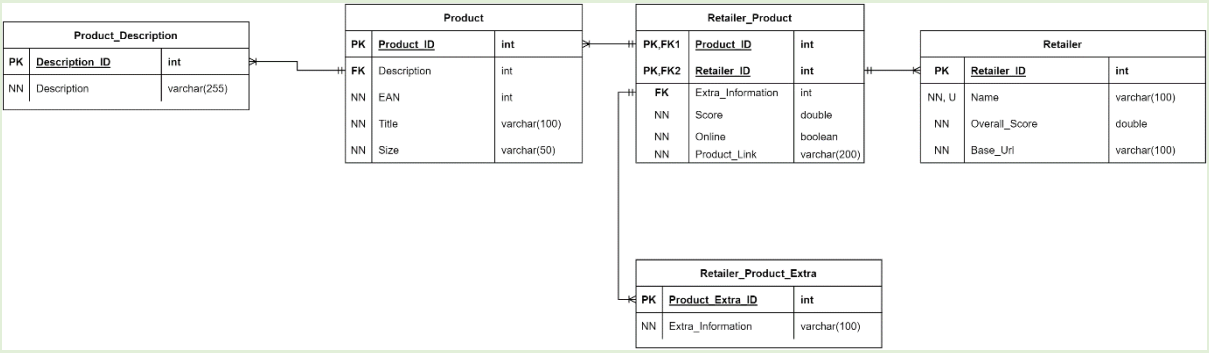
I made sure to identify all the entities before we went into the designing process and especially pay attention to the different excel sheets that were provided by the company. From there on, I figured the keys and ways of interaction with the database were quite clear from the start. Therefore, I proposed to use a NoSQL database for the application. It would have given us more freedom to store different data supplied by the many retailers. In addition, I reminded the team that the product owner urges us to make a scalable product. All these elements seem to hint more at using a NoSQL database. A NoSQL database only requires new servers to be added to the database, since it is 'horizontally scalable'.[1] Furthermore, the company works with cloud-based software, which also promotes using NoSQL over SQL.[2] I discovered later that NoSQL works well with REST APIs, which is also what we are using for this project.[3]

However, as a team we eventually decided to go with a SQL database to support relations between retailers and products while keeping in mind performance when writing more complex queries. We also were not completely sure where the project would take us, so it was the safest option to go with a SQL database in case of new additions or changes.

[1] https://www.bmc.com/blogs/sql-vs-nosql/
[2] https://www.techtarget.com/searchcloudcomputing/tip/Compare-NoSQL-database-types-in-the-cloud
[3] SQL vs NoSQL Explained - https://youtu.be/ruz-vK8IesE

1.3.2 ERD

To allow retailers to add multiple, separate pieces of information, I added a table, Retailer_Product_Extra, with a one-to-many relationship to the Retailer_Product table. The supplied Excel sheets we received had similar products sold by different retailers with different data, so I figured it would be best to allow the user freedom in adding extra information.



Entity Relationship Diagram which follows the current database structure.

1.4 User Experience

To get a clear idea of usability and learn more from a professional, I decided to read the (Dutch translated) book “Don’t Make Me Think” by Steve Krug. Considering our application is not a public website, I only picked a couple of chapters that I figured were relevant to the design of our project.

1.4.1 Don’t make them waste time

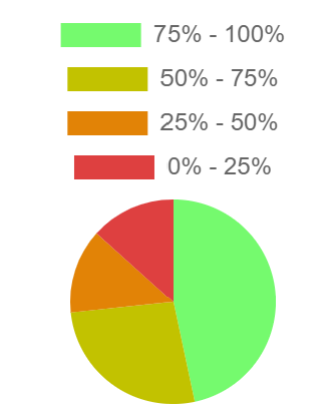
To quickly gain insight into the different sets of data handled by our application, I worked on two different components that show an overview of the most important data.

One of which are the charts mentioned in the **Asynchronous communication** section and the other one is a data table. Why are these components part of UX? Because it adheres to the 8th principle on the list of Nielsen’s 10 Usability Heuristics[4]: Aesthetic and minimalist design.

1.4.1.1 Pie charts

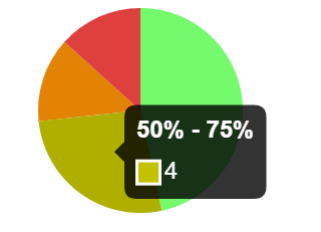
The idea behind the pie chart below is showing the number of entities that have a score percentage within a certain range. Using a gradient to give a familiar feeling to whether an entity is in the higher or lower scoring range, there is a lot of self-explanatory going on to make the data easily visible while it does not feel out of place with the rest of the design. As of the moment of writing, the design for the charts is not yet finalized and is still up for changes.

[4] All 10 principles are discussed here: https://www.nngroup.com/articles/ten-usability-heuristics/



Screenshot of the overall design of the pie chart.

When a user hovers over the different colour sections, a tooltip appears with additional information.



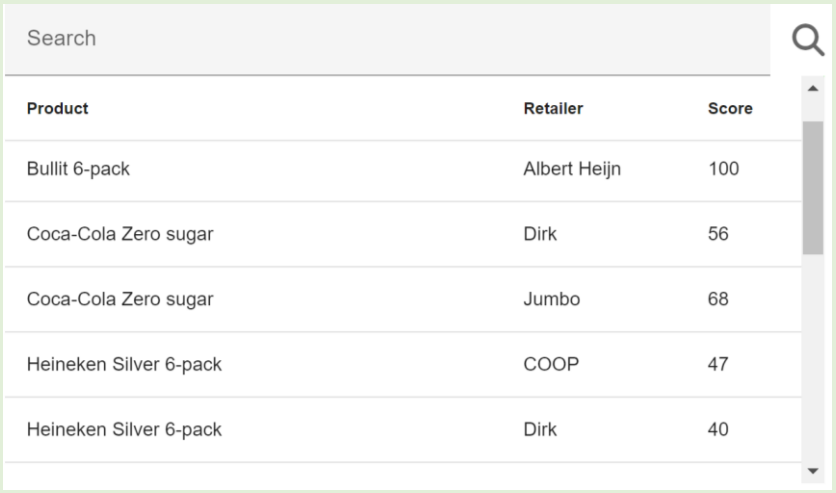
Screenshot of a tooltip showing over the yellow (50% - 75%) area, where 4 entities are count be in that range.

By using pie charts to display data, the user can quickly view relevant information in a compact format. I made sure to pay attention to importing only the necessary component parts from the Chart.js library, to not interfere with the performance of our application. After all, nowadays some of the biggest causes of slow response time are proven to be the “overly fancy widgets”. (Nielsen, 2010)

```
import { Pie } from 'vue-chartjs'
import {Chart, ArcElement, PieController, Decimation, Filler, Legend, Title,
Tooltip, SubTitle}  from 'chart.js';
Chart.register(ArcElement, PieController, Decimation, Filler, Legend, Title,
Tooltip, SubTitle);
//Only importing the necessary attributes to optimalize performance.
```

1.4.1.2 Data table

The relation between retailer and product has a certain score, based on how much of the data is live. All these three elements are shown below in the table, to be able to see all important aspects in just one glance. The user can sort by selecting one of the column names, highlighted in bold text. The search bar on top is able to filter by the given search criteria and will return any product or retailer that matches the given input.



Screenshot of the data table. Currently sorted by product.

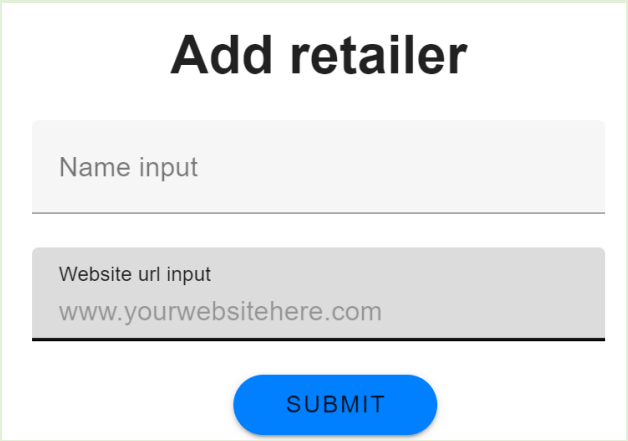
1.4.2 Don't make them think

To allow users to successfully complete their objectives within our website, we do not want them to waste any time on tasks that would otherwise be very easy to handle. A few tricks I learned on this topic are the power of simplicity and to avoid making users 'follow any rules'.

1.4.2.1 Formatting

During the third sprint, I spend some time working on the first CRUD functionalities for our application. I used this opportunity to apply some of the new things I learned about UX: When creating a new retailer, users are asked to insert a 'base' URL to their website. However, they should not have to think about formatting this URL or whether they added certain elements like http. While this is an application only accessible to certain users, it is important to help the user in any given way to make their tasks easier and to keep their "reservoir of goodwill" (Krug, 2011, p. 149). I made an attempt at helping the user in two ways:

1. Using a placeholder in the HTML to give the user a hint of how to format the URL:



When collapsed, the input fields show a relevant placeholder to help the user.

2. I wrote a function in the backend to format the URL:

```
function modifyURL(baseUrl: string)
```

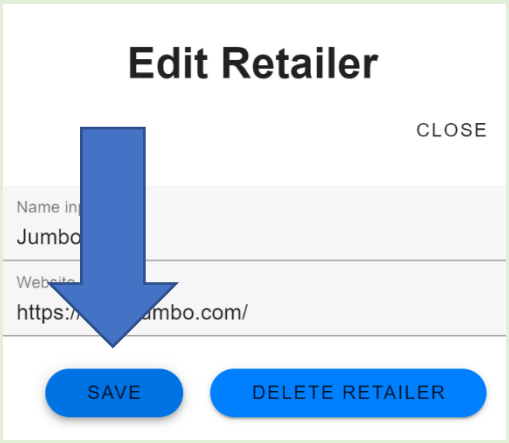
```
{
    let url = baseUrl;
    let PREFIX = "http";
    if (url.startsWith(PREFIX)) {
        // PREFIX is exactly at the beginning
        url = url.slice(PREFIX.length);
        if(url.startsWith("s"))
        {
            url = url.slice(1);
        }
    }

    let i = url.length;
    while (i--) {
    if(url.charAt(i) == "/")
    {
        url = url.replace('/', '')
    }
    else if(url.charAt(i) == ":")
    {
        url = url.replace(':', '')
    }
    }

    return url;
}
```

1.4.2.2 Clear choice of words

A more subtle addition is the confirmation button present when an entity is being edited. In the initial release of our application, the button said “OK”, which was later determined to be somewhat confusing: Whether it meant the changes were to be saved or that it would cancel the process of editing an entity, was not clear to the users.

Old screenshot of the editing screen where the button text is “OK”.

So, I simply changed the word “OK” to “SAVE” instead and made sure to pay more attention to our choice of words. It is important to avoid any small obstacles like these that require thought to succeed.[5]



Screenshot of the editing screen where the button text has changed to “SAVE”.

2. IPS – VOICECHECK

Because I focussed more on the GPS part during this semester, this project unfortunately never got to take off. VoiceCheck was supposed to be a project where users, fan of the voice acting genre, could gather to find information and audition for vacancies. Additionally, recruiters could post about these vacancies and get in touch with users that submitted an audition.

I chose Vue as my framework of choice for this project, the **Frontend** section shows how I got to that decision. I chose to expand my knowledge on .NET Core for the backend, more about it in the **Backend** section. The work I did in Vue and .NET Core both have their respective repositories on GitLab and I added a zip containing the source code for each to this folder.

2.1 Frontend

I did manage to do some research on JavaScript frameworks and experiment with them. One of the most helpful sources I could find was a video by the instructor Maximilian Schwarzmüller (Academind on Udemy & YouTube).[6] The key factor I considered when making a decision is my very limited knowledge of JavaScript (at the start of the semester). I must mention that the fact that we are using Vue in the group project did not necessarily affect my decision. It did, however, make it easier to learn and understand its perks.

[5] See Krug (2011), chapter 3, for a more detailed analysis of this approach.
[6] Angular vs React vs Vue – https://youtu.be/lYWYWyX04JI

2.1.1 Framework research

I started out with looking into similar applications and their stacks. Some similar applications include Facebook and Reddit. React is developed by Facebook and Reddit also uses React, according to Stackshare.[7] So, I chose this as the first framework I wanted to look into.

2.1.1.1 React

React is the most popular framework (or library, as it is called by the developers[8]) among employers. When visiting jobsites like Indeed.com, most vacancies ask for experience with React. With simplistic and minimalistic as its end goals, it makes perfect sense it is so popular among employers. Possible extensions like routing are not included and thus urges developers to look elsewhere for routing libraries. This also adheres to the minimalistic approach.

However, working with React and becoming comfortable with the way of things is different. I found out that React's simplistic approach can be either a pro or a con, depending on one's skill level. First of all, when creating a new project in React, it requires quite the complex setup before any code can be written. This was quite overwhelming. Secondly, its mixture of JavaScript and HTML, JSX, can be very confusing to get adjusted to when you are not comfortable with JavaScript yet.

I figured there were too many elements that required my attention to make something happen, so for this semester, I decided not to go with React.

2.1.1.2 Angular

Angular is a framework developed by Google and has, unlike React, most features like routing included in its framework. It is considered to be 'very rich' when it comes to features, having mostly everything you need and this does not impact the performance time of the application[9]. In addition, Angular makes a clear separation between HTML and JavaScript, which is in my opinion easier to read. A fascinating aspect about this framework is that it still 'holds up' after all these years and is at a stable level of popularity.[10] Developers mention that its popularity depends on its releases and the changes in every update; it is a case of 'either make it or break it'.

Angular seemed to fit my project well and a good place to start. However, like React, Angular comes with a complex setup when creating an application. This was quite overwhelming. A further deciding factor for me was the use of TypeScript within Angular, which was even more advanced to learn for people with little JavaScript experience. Most developers therefore do not recommend starting out with Angular in this case.

2.1.1.3 Vue

Unlike the before mentioned frameworks, Vue is developed by a standalone team of collaborators. It is the newest among all three and in comparison the least popular when it comes to job opportunities.

---

[7] https://stackshare.io/reddit/reddit
[8] https://reactjs.org/
[9] This depends on the business case. In general, the difference with other frameworks should not be too noticeable.
[10] https://trends.google.com/trends/

Vue is the final candidate. It is the framework of choice for the group project, which helped me learn the framework much faster. Just like Angular, Vue separates HTML and JavaScript. It even has the feature to add a “scoped” style; CSS code that only applies to that particular file. This helps a lot when learning to work with a component-based framework. Furthermore, unlike the other two frameworks, Vue does not require a complex setup and it is up to the developer what is included when starting a new project.

It is safe to conclude that Vue has the lowest learning curve among all the other listed frameworks, making it the best option when a developer has limited JavaScript experience. According to Google, it seems this conclusion is very common among developers.[11]

2.2 Backend

During my frontend framework research, I mostly spend my time reading and watching informative video’s. For the backend framework, I took on a more practical approach and set up sample projects to try and connect with them from the frontend. As there was a workshop about Quarkus before, I could use that as my starting point for working with Java. The results of that are more related to **Data persistence**, which is why there is no separate section covering my work with Quarkus.

2.2.1 Experimenting with backend frameworks

It took a lot of time to get Java to work on my laptop. Once it was configured, I took the sample project from the workshop as my starting point and adjusted small things to understand the process. After getting stuck on small elements, it turned out that I was more in a C# mindset and that in order to save time, it might be a better idea to continue in .NET Core.

2.2.1.1 .NET Core

I was able to make the connection from Vue to .NET Core work.

ID	Content	Date created	Author
1	derde test met nieuwe database structuur	2022-10-21T18:28:06.275	maurice
2	vierde test met nieuwe database structuur	2022-10-21T18:28:27.758	maurice

Screenshot of a table in Vue.

[{"id":1,"content":"derde test met nieuwe database structuur","creationDate":"2022-10-21T18:28:06.275","author":"maurice"},{"id":2,"content":"vierde test met nieuwe database structuur","creationDate":"2022-10-21T18:28:27.758","author":"maurice"}]

Screenshot of the JSON data returned with a GET request by the API in .NET Core.

	ID	author	content	creation_Date
Edit Copy Delete	1	maurice	derde test met nieuwe database structuur	2022-10-21 18:28:06.275000
Edit Copy Delete	2	maurice	vierde test met nieuwe database structuur	2022-10-21 18:28:27.758000

Screenshot of PHPMyAdmin. The endpoint of the data returned by the API.

The downside to this is that I still made use of handwritten queries instead of an ORM, as is required for the individual project. As stated in the **Data persistence** section, I had plans of using Entity Framework to

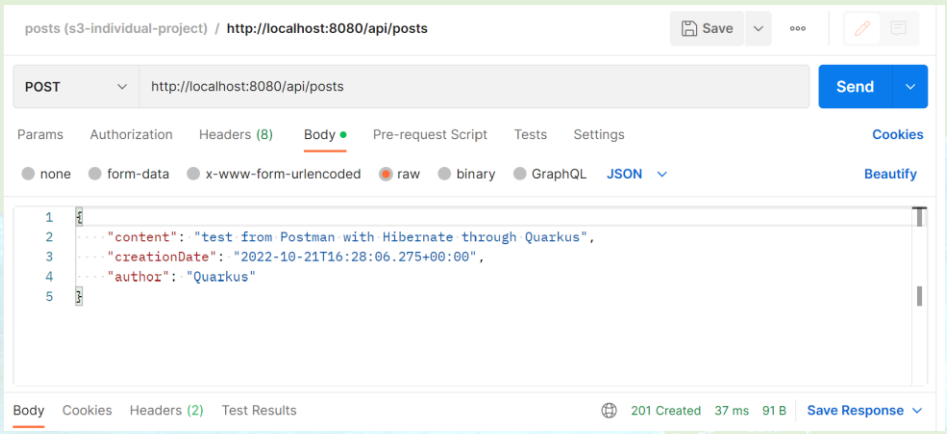
[11] https://www.google.com/search?q=react+vs+angular+vs+vue+learning+curve

cover this problem, but never got the queries to work. I have, however, shown some proficiency towards this because of my contribution in the group project, as can be read here: **Security checks**.

2.3 Data persistence

As I experimented with a .NET and a Java backend framework, I figured I could use that opportunity to try out working with ORM tools in different languages. I experimented with a demo to use Entity Framework for .NET Core, I was able to establish a connection, but unfortunately never got the queries to work. For Java, I used Hibernate in my small Quarkus application after learning how a "transactional" works: Making sure it is either all or nothing that gets to the other end; similar to a bank transaction, to determine whether the deal was successful or not.

Through the application Postman, which served as a possible substitute for frontend interaction at the time, I could set up a new database or access an existing one and query it through Hibernate.



Screenshot of the application Postman. The POST JSON data returns a successful response (201).

	ID	author	content	creation_Date
Edit Copy Delete	1	maurice	derde test met nieuwe database structuur	2022-10-21 18:28:06.275000
Edit Copy Delete	2	maurice	vierde test met nieuwe database structuur	2022-10-21 18:28:27.758000
Edit Copy Delete	1002	Quarkus	test from Postman with Hibernate through Quarkus	2023-01-12 11:46:47.829000

Screenshot of PHPMyAdmin. The new record is added to the table.

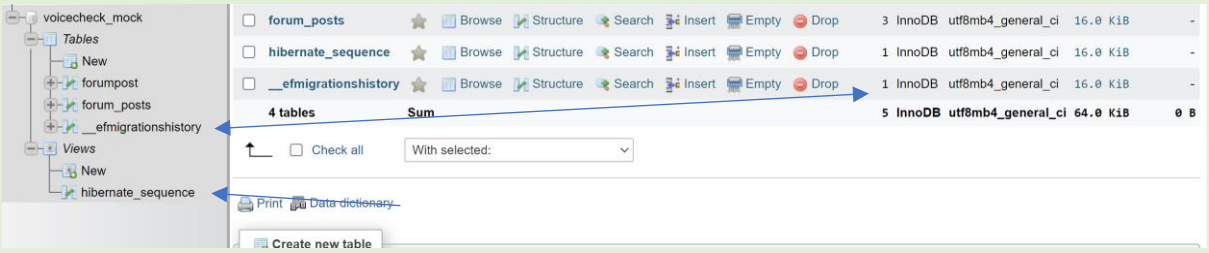
The code below is a snippet of the class that was 'persisted' through Hibernate. A lot of things are specified; like the creation_Date column and data.

```
@Entity
@Table(name = "forum_posts")
public class ForumPost {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "ID", updatable = false, nullable = false)
    private int id; //ID is auto incremented by db
    private String content;
    @CreationTimestamp
```

```
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "creation_Date")
```

Code snippet of what the class that was persisted looks like.

After receiving a successful response, a new “hibernate_sequence” was added to the Views of the database VoiceCheck_Mock. There is also a table called “_EFMigrationHistory”, this was created once I connected to the database with Entity Framework.



Screenshot of PHPMyAdmin. The arrows highlight the new records created by both ORM tools.

3. REFERENCES

Krug, S. (2011). Don't make me think (Nederlandstalige editie). In S. Krug, *Don't Make Me Think* (p. 187). Uitgeverij Thema. Opgeroepen op 2022

Nielsen, J. (2010, June 20). *Website Response Times*. Opgehaald van Nielsen Norman Group: https://www.nngroup.com/articles/website-response-times/