MensaToday - Your Dish Recommender in Münster

Leo Giesen
Münster, Germany
leo.giesen@uni-muenster.de

Marten Jostmann
Münster, Germany
marten.jostmann@uni-muenster.de

Polina Kireyeu
Münster, Germany
polina.kireyeu@uni-muenster.de

Marcel Reckmann
Münster, Germany
marcel.reckmann@uni-muenster.de

Erik Zimmermann
Münster, Germany
erik.zimmermann@uni-muenster.de

1 CONTEXT

The objective of the group work in the module *Data Integration* was to conceptualize and develop a recommender system within three months. To achieve this, some goals were defined to build an idea for creating a great product. Mainly, the idea had to be relatable and usable throughout the entire week in perspective of somebody embracing the study lifestyle. Our group, consisting of students who frequently visit the canteen, quickly came up with the idea of a food recommender called *MensaToday*. The search for key elements started with the way the *University of Münster* is distributed. Because the campus is spread all over the city, travel time and other related factors are crucial. This led to the need for information on canteen locations, weather and user flexibility.

Another product goal was to present complex information in an easy-to-use and visually friendly interface which was the main reason for splitting the team into frontend and backend. As an insight behind the scenes, the following report will mostly contain information about the backend comprising data sources, technology stack and recommendations.

2 DATA SOURCES

The success of every recommendation system lies in the combination and integration of different data sources. For this purpose, internal as well as external data sources have to be found and evaluated. One of the most important internal data sources is the explicit feedback of users in the form of ratings. These ratings are, for instance, gathered in the initial survey, which is part of the registration process. In addition to those ratings, the user also specifies their food preferences in the form of categories (vegan, vegetarian, etc.) along with allergies and disliked additives.

The most important external data source for *MensaToday* is the collection of main and side dishes. There are multiple sources which offer the current dish plan for every canteen and bistro in *Münster*. One of them is available in a structured `XML` format and others in semi-structured `HTML` files. As the `XML` data source sometimes appeared to be outdated, it has been disregarded as a data source. The websites *my-mensa*[1] and *iMensa*[2] provide canteen dishes in the exact same way. In addition, *iMensa* also captures an average dish rating, which is why it has been chosen as the central data source. Information about the dishes, their ratings and further data was extracted with the help of the python web scraper `Beautiful Soup`, which is a powerful tool to parse arbitrary `HTML` and `XML` files.

One crucial feature of the recommendation system is the user location awareness because it can be used to fine-tune the final recommendations. The location of students and lecturers is determined by their enrollment in *LearnWeb* courses, which is a *Moodle*-based learning platform to distribute course material at the *University of Münster*. This is achieved by reverse-engineering the *LearnWeb* using the user's *LearnWeb* credentials. Since the data source only provides the courses but no details about the location and time window of them, *QISPOS* as another data source is used. It is a centralized platform of the university providing information about lecture rooms, courses and study programs. Using the already acquired course names from the *LearnWeb*, the schedule as well as the course room address are extracted from *QISPOS*. For this extraction, the same scraping tool as mentioned above was used. Lastly, the *OpenStreetMap* API was used to get the exact longitude and latitude of the room. All together, the distances between every room and all canteens are precalculated once and then stored in the database for a quicker recommendation. In addition to the user location, the weather also plays an important role for the dish recommendation because a student does not want to ride their bike for ten minutes through the pouring rain. Consequently, the *Open-Meteo* API has been contacted to acquire the current weather condition every 15 minutes.

[1] https://muenster.my-mensa.de/
[2] https://www.imensa.de/

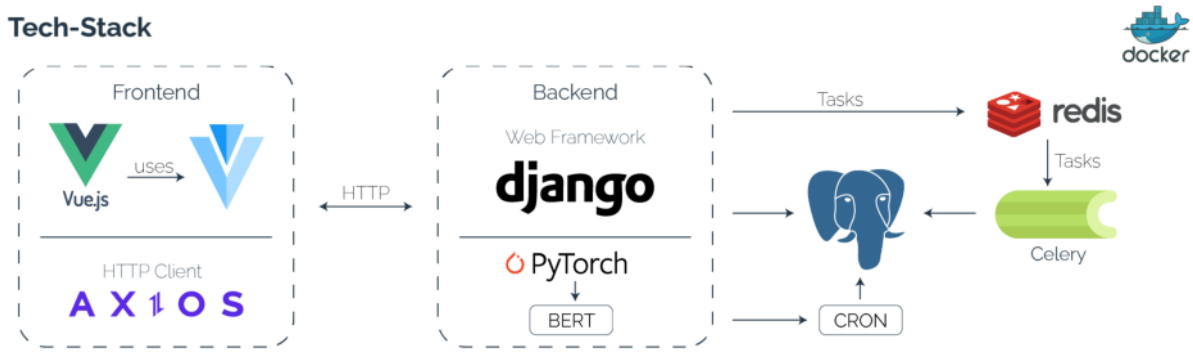

Figure 1: Overview of the Technology Stack

To provide the user with a helpful impression of the dish, a preview image is displayed. Unfortunately, no canteen application on the market displays exemplary images of their dishes. To solve this issue, images were extracted from *DuckDuckGo*. Because many images are served under strict copyright requirements, it is necessary to filter free to use images. Due to this filtering, images could not be found for every dish. For dishes without a preview, the first letter of the dish serves as a placeholder.

MensaToday also tries to make the dining experience more enjoyable. Since many students and employees first check their credit balance at a vending machine when visiting the canteen, it sometimes takes a long time to do so. Therefore, *MensaToday* displays the current card balance in the application eliminating the need for waiting in line. This is implemented with the payment service *Klarna*, which cooperates with the *University of Münster* and *University of Düsseldorf*. Besides checking the balance, *Klarna*'s API enables the user to recharge the card balance by entering the mensa card number and a payment method. As a result, the mensa card can be recharged without the requirement of carrying cash with you.

3 TECHNOLOGY STACK

The technology stack is divided into the two main components frontend and backend. The applied technologies are visualized in Figure 1. The frontend is developed with the performant and versatile framework `Vue.js` that build single-page applications based on `JavaScript`. Additionally, `Vuetify` is used which provides handcrafted `Vue Components` based on the material design guidelines. To communicate with the backend application, `Axios` is used, which is a promise-based `HTTP` Client for `Node.js`.

The backend of the web application is built with the `Django` web framework, which provides a robust set of tools for handling user authentication, URL routing, and database management. The authentication is handled by `JSON` Web Tokens, which provide a stateless, scalable as well as a secure session management. To compare dish titles with each other, a `BERT` Sentence Transformer[3] is used. The framework is based on *PyTorch*, which is a large machine learning library developed by *Meta*. It provides several pretrained models that can be used out of the box. For comparing dishes, the small and performant model *all-MiniLM-L6-v2* is used.

To increase the overall user experience and to reduce the server response time, time-consuming calculations are handled in the background. For this purpose, `Celery` as an asynchronous task queue is used. The Message Broker `Redis` functions as a connection between `Django` and `celery`. With this structure, the user receives a direct response while tasks are executed in the background. To store and manage large amounts of data, the PostgreSQL relational database management system is used as the primary database. Additionally, a *Cron Job* is instantiated to be able to automatically gather every dish regularly. Finally, the entire web application is containerized with `Docker` to allow the application to be easily deployed on multiple machines with different operating systems and hardware specifications.

4 RECOMMENDATION

The project starts without any user data. Hence, it is necessary to gather as much information about the user as possible in the registration phase, which begins with the specification of the user's food preferences. Those are used as hard constraints to directly filter out unsuitable dishes from the item pool. Similarly, user allergies filter out dishes with respective properties because they should not be considered at all. Based on this initial information, three different dishes are proposed to the user that have to be rated. Subsequently, the user profile is created based on the combination of ratings and food preferences, enabling basic recommendations with *content-based filtering*.

4.1 Content-based Filtering

Content-based filtering is a technique that can create recommendations by comparing so-called *user profiles* with *item profiles* using the *cosine similarity* (see equation 1).

$$C(x, y) = \frac{x \cdot y}{\|x\|\|y\|} \tag{1}$$

First, *item profiles* must be generated so that the dish features are captured. This information comprising food categories, additives and allergies are fetched from *iMensa*. These features are translated into binary vectors to determine the similarity between them based

[3] https://www.sbert.net/

on equation 1. Additionally, dish titles are considered to provide significant relations between similar meals. Since BERT also uses the *cosine similarity*, it fits into the construction of the required *item profiles*. The downside of BERT is that the significant number of features introduces the *Curse of Dimensionality*. In order to respect the different variations of feature vector combinations, several tests are conducted by using the *Root Mean Squared Error* (*RMSE*) (see equation 2) to find the optimal balance between generalized recommendations and the *Curse of Dimensionality* (see figure 2).

If you have too few meaningful features, the dish is not captured sufficiently. For instance, only having the feature *additives* or *food categories* would be too broad and would not describe the dishes properly. Against all expectations, combining all the available data into a single vector worked best to define *item profiles* as portrayed in figure 2.

$$RMSE(x) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x - \hat{x})^2} \tag{2}$$

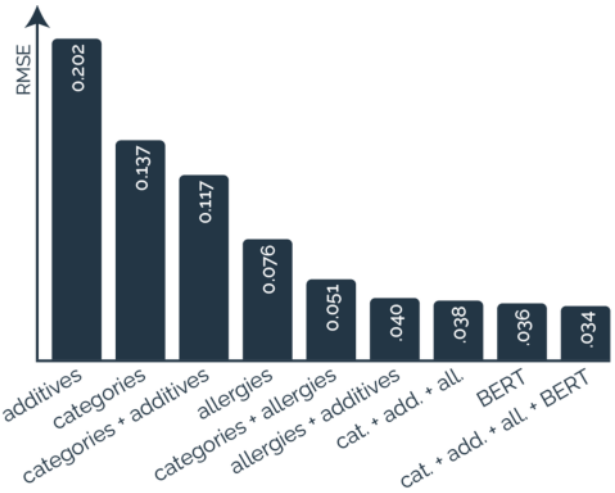

Figure 2: Evaluation of Different Item Profiles

Having the *item profiles* in place, everything is set for generating basic recommendations. Once a request reaches the project backend, the core computes the *user profile* by multiplying every available rating given by the requesting user with their corresponding *item profile* (see equation 3).

$$user = \sum_{i}^{N} rating_i \cdot item_i \quad rating \in [0.2, 1] \tag{3}$$

Finally, in a last step, the *cosine similarity* between the computed *user profile* and every available *item profile* for the given weekday is calculated. By sorting the result list, the most suitable recommendations can be presented to the user. As already mentioned, this is only a basic recommendation. As an example, some dishes could be the healthiest and most delicious, but a user would just not consider them, if they have to cycle across the entire city during a storm. Therefore, we apply some fine-tuning in order to ensure satisfactory results.

4.2 Fine-Tuning

For fine-tuning, two main aspects were considered. First, the distance in kilometers between the last known location of the user (based on the university class schedule) and a given canteen is computed. Second, the API *Open-Meteo* is used to derive a score that describes the weather behavior during the given lunchtime, which is set to noon. Both constraints were transformed into some $score \in [0.1, 1]$ which is then multiplied with the similarity of one user-item tuple (see equation 4).

$$score_{distance} = 1 - min\left(\frac{distance}{flexibility}, 0.9\right) \tag{4}$$

The lower bound of the interval of $[0.1, 1]$ is chosen to be able to still recommend a dish considering the user preferences despite extraordinarily bad fine-tuning constraints. In this case, it is ensured to select top recommendations between an interval of $[0, 0.1]$. The *flexibility* variable in equation 4 is used as a hyper-parameter that includes the individual maximum range that a user would travel to a canteen. Initially, it was set to three kilometers and is planned to be configurable by the user in the future.

Computing a score for the given weather is handled differently. The weather score is calculated by temperature, wind, rain and snow values, which are summed up indifferently with regard to their severity. Once summed up, the score is calculated according to equation 5, which is manually tested and iteratively improved.

$$score_{weather} = 1 - min\left(\frac{points^{1.3}}{max\_points}, 0.9\right) \tag{5}$$

Due to the lack of dish ratings, some recommendations appeared as duplicates in some testing iterations. To address this issue, an additional feature has been implemented to detect duplicate suggestions which improved the user experience, especially in the beginning when only a few (user-specific) ratings are available.

4.3 Side Dishes

In the feedback of the intermediate project demonstration, side dish suggestions were proposed to contextually complete a main dish recommendation. Based on that idea, an additional subsystem has been implemented. Due to the nature of the user-interface, main and side dish combinations are saved in the database which can be counted as an aggregation on main dishes. So, by having users in the system who frequently select side dishes, this functionality can suggest popular combinations based on active selections. Those suggestions appear as most-popular in the main and side dish selections.

5 USER-INTERFACE

The central functionalities of *MensaToday* are captured on two main pages, namely the default recommendation page and the discover dishes page. The recommendation page allows the user to view the recommendations in a weekly overview, where the user receives one individual recommendation per weekday (see figure 3). If the user wants to evaluate multiple dishes for a single day,

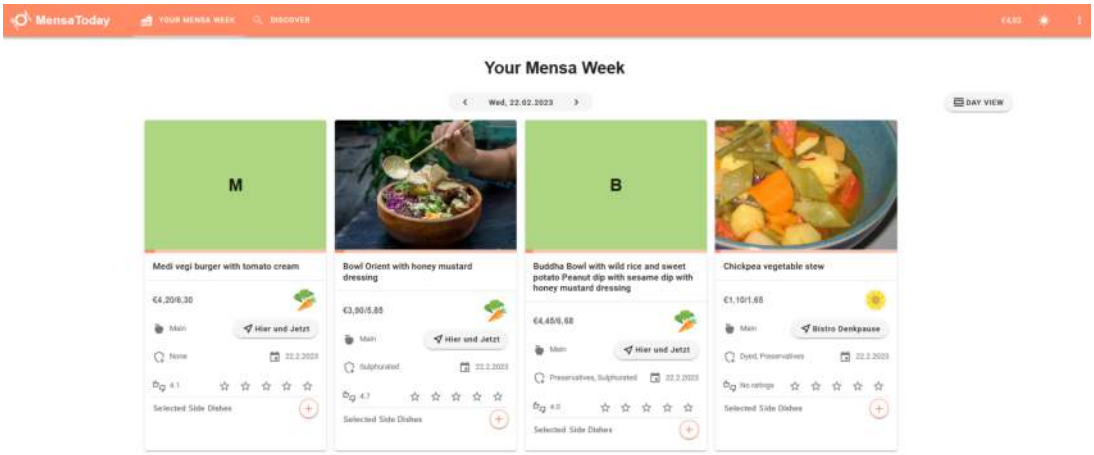

Figure 3: Excerpt from the User Interface

the day-specific view can be selected. Here, the user is presented with four suggested meals for each day providing the user with a variety of dishes and meal-specific side dishes to choose from. A sleek card design offers a rich informational and vivid overview about information regarding the meals. The information about additives and food categories is supplemented by a *Google Maps* link directing the user to the canteen and a photo of the meal to get a great impression of the dish. Additionally, the meal-specific side dishes and the most popular side dish are displayed in a user dialog. The selected side dishes are saved in the overview.

The discover dishes page comprises the complete week plan and helps the user to search through all possible dishes. For instance, if a friend is going to the *Aasee Mensa* on Tuesday, the user can filter respectively to receive all relevant dishes. Besides the filter of date and canteen, the user can apply further filters to limit categories, allergies and main dishes, search for individual dish names and filter out dishes that exceed the user's card balance.

6 CHALLENGES AND OUTLOOK

Every software development project has its own challenges and shortcomings which should be addressed in the following. A major challenge in this project was the evaluation of the recommendation system's output. Thus, a test set with exemplary dish ratings as labels is used to evaluate the model accuracy by using the *RMSE* as a loss function (see equation 2 and figure 2).

Another challenge was the `JSON` serialization and database access performance. In general, `Django` serializes one object after the other and therefore conducts a separate `SQL` query for every object, joining all relevant tables. This serialization process takes a long time for a large number of objects. To solve this challenge, a bulk load with the *prefetch_related* property was used. With these properties, all the required objects are loaded at once and serialized afterwards. This reduced the total number of `SQL` queries drastically and further improved the response time. The same technique is applied to improve the performance of the recommendation process. Besides that, the request from the weather API call is cached and only refreshed every 15 minutes. In the first place, the sentence embeddings produced by `BERT` were computed at every request which reduced the response time of the recommendation system. In the current version of *MensaToday*, the sentence embeddings are cached and only recomputed when there are any changes to the dishes.

MensaToday is a web application that can recommend main as well as side dishes to potential 50 thousand students and employees in *Münster*. To increase the usability and overall user experience, more effort has to be invested to improve the response times. Additionally, the software heavily relies on *iMensa* at this point. In the future, this data source should be replaced by an own dish pool directly served from the canteens in *Münster*. In order to improve the recommendation sustainably, further techniques should be elaborated. In particular, *Collaborative Filtering* and machine learning algorithms should be considered.

7 INDIVIDUAL CONTRIBUTION

Every group member participated in every planned and spontaneous jour fixe, contributed their own ideas and gave food for thought. Nevertheless, few members had a crucial contribution to the project. Please note that no task has been completed alone but rather with the help of other participants. Thus, only the central contributors for each task are listed here.

Marten J.	Project Structure, Data Structure, Recommender Conception, Data Collection, API-Endpoints (Backend), Serialization, Authentication, User Location Calculation
Leo G.	Vue.js Setup, Frontend Implementation and Design, connection to backend with API-calls
Marcel R.	Frontend (Recommendation and User Page)
Erik Z.	Recommender Conception, Data Collection, API-Endpoints (Backend), Recommender Implementation, Poster and Designs

The entire project is GPL-3.0 licensed and can be publicly revisited on GitHub[4].

[4]https://github.com/MensaToday/mensa-today