Digitaalisen kielentutkimuksen projekti

Turun yliopisto
Tietotekniikan laitos
Projekti
Lokakuu 2023
Iiro Partanen 525720_287358_A

Sisällys

1 Planning

This is a project for my college course DIKI1001-3002. Its aim is to create a project that uses some natural language processing technology and I'm trying to achieve a project that has the scope of five credits.

1.1 Goals

Besides the goals to fulfill the course requirements I have multiple own goals to learn. These have come from previous projects that I have felt myself lacking on or just new things I feel like I would like to learn. Firstly some technologies I want to learn or display my skills in.

I want to create the backend with Java and learn Spring boot to go with it. Here I want to display my skills with OOP and create a well structured backend. I also want to use SQL for the database.

For the models I want to create an API with Python and learn Flask to go with it. I also want to learn AWS and deploy the application and database there. Python is being used in all of the NLP courses so it would be nice to show some skills with it in a project.

I really want to focus on planning and create a good plan for the development of the application. This includes diagrams for the API, backend and the database. I will create a mock of the site before beginning to develop it. I will also create a roadmap of the development and in which order everything will be done.

Some things to focus on the development are first of all structure. I want the application in frontend and backend to have a good structure and really focus on re-usability and modifiability. So basically making small components that are easy to use again and also modify.

I also want to focus on testing and creating good unit tests for all the components that cover the whole application. Maybe even e2e testing, if I have time for it. Documentation and code readability will also be important during this project. While developing I will pay attention to code readability and also try to document everything.

1.2 Plan

The plan of this project is to create a machine learning model from Amazon reviews that rates them and also calculates the most important words from these reviews. Then creating a website around this that mocks a seller admin page. This would then display statistics about the reviews and the items that have these reviews.

For training I will use the amazom_reviews_multi dataset, that includes reviews in six different languages that have a rating from one to five. I will only use the English dataset.

Let's go trough the structure of the project and design of each part. The actual implementation is in chapter 2

1.3 Machine learning models

These models are the linkage to natural learning processing technologies. I'm going to train two models. One that is a basic feed forward neural network and one fine-tuned version of BERT. I chose these because in the end they both relate to each other.

Both of these models are trained on the Amazon_reviews_multi dataset that has 200000 amazon reviews that are labeled from one to five stars. For the application itself I will use other Amazon reviews. I am expecting to get about 70% accuracy on my BERT model.

The better model, probably the BERT model will be hosted at Huggingface, so that I can use it from their API.

1.3.1 Feed forward model

The feed forward model is really basic, but displays the understanding of fundamentals of models that are based on neural networks, so I think it's a great fit for this project. Also I didn't want to use RNN or CNN, because feed forward network ties into the classification layer that I add into BERT so I will get some fun comparison to how the BERT model helps our simple feed forward network to predict correct labels.

1.3.2 BERT fine-tuned model

The fine-tuned BERT model on the other hand is a bit more complex, but this is chosen because it is the best choice for this task (excluding already fine-tuned versions for sentiment analysis). BERT is still state of the art in sentence classification so that's why I chose it. I haven't yet chosen which version of BERT I will use and this will be solved only after I have tried which fits best for me.

I will take the base BERT model and add a classification layer to it that in the end is pretty similar to the feed forward network that I created for this task also. This way I fine-tune it to work with our classification into five labels.

The training for this and also the feed forward network is done in Google Colab. This means I won't reach state of the art results, but it also brings great ways to learn how to manage the models and get the most out of it on lower computing power.

1.3.3 Most important words

The calculation of the most important words will not be done with a machine learning model, but I feel fit to describe it here. I will split the reviews into positive ones, four to five stars, and negative ones, one to two stars. Based on the ratings that the machine learning model created.

Then I will calculate the words with TF-IDF and get the most important words. I don't really know how well this will work so it will be really interesting to see.

This is meant to be a really lightweight and basic addition so I will not be creating a model for this. The point is just to calculate the most important words for the reviews. And logically I use TF-IDF, because if I just calculate the most common words, they will be some of the most common words in English. So with TF-IDF, I will get some importance to the words.

1.4 API

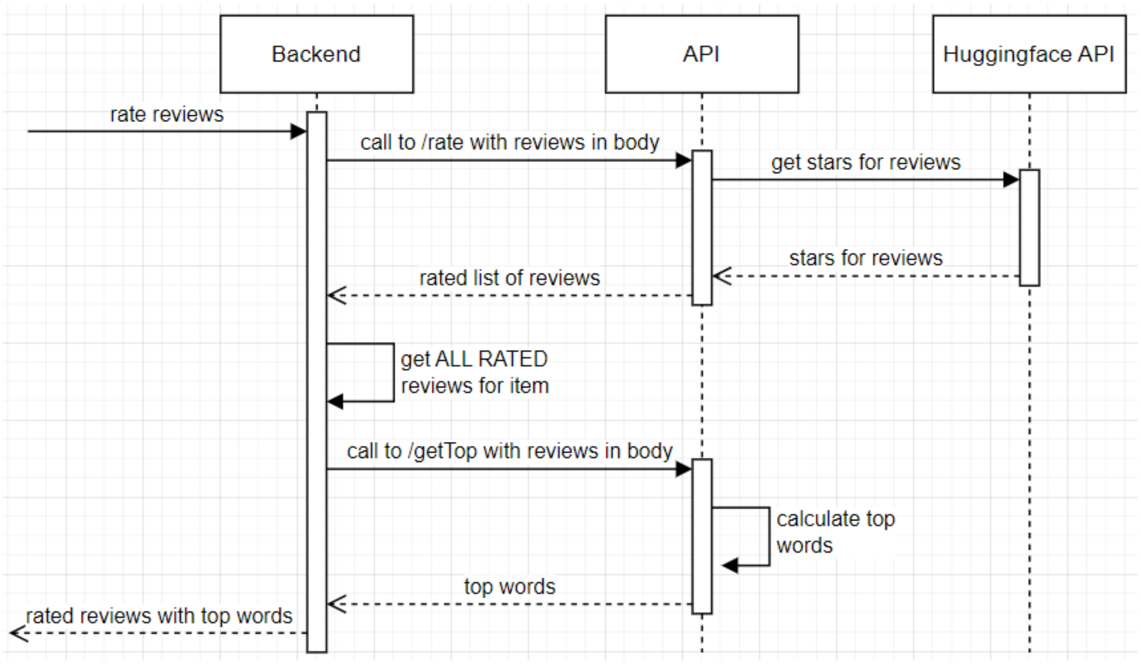
The API will be built to access the Huggingface API and also to calculate the most important words. It's main reason is to rate the reviews or calculate their top words. I will create a separate API for this because it feels more logical than calling it straight from the backend. This way the model can be used in other projects also.

1.4.1 Structure

The API will have two endpoints, /rate and /getTop. Rate will receive a list of reviews in the body and it will return these rated after calling the Huggingface API. GetTop will receive a list of rated reviews in the body and will calculate the top words for positive and negative words.

These are split into two, because it adds to the flexibility. I can then rate some reviews without calculating the top words or get top words without calculating the ratings.

In picture 1.1 is a sequence diagram for displaying how the backend works with the API to get ratings and top words for reviews.


Backend
API
Huggingface API
rate reviews
call to /rate with reviews in body
get stars for reviews
stars for reviews
rated list of reviews
get ALL RATED reviews for item
call to /getTop with reviews in body
calculate top words
top words
rated reviews with top words

Kuva 1.1: API sequence diagram

1.4.2 Technologies

The API will be written in Python with Flask. I will use Python because to calculate the top words I use numpy and sklearn.

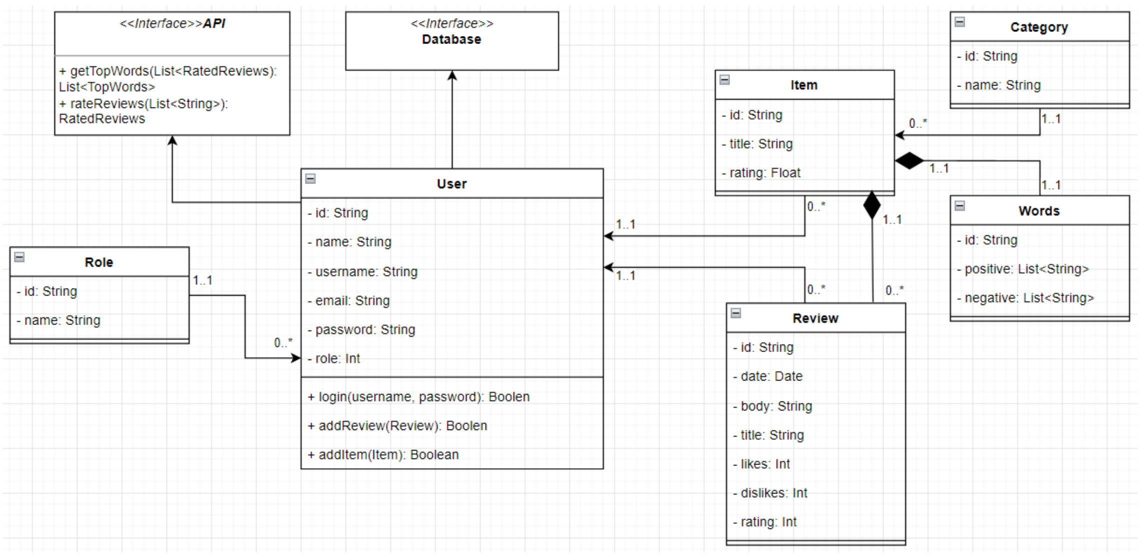
1.5 Backend

The backend for the application will handle all the logic. Everything but calculating the top words and rating reviews will be done in the backend.

1.5.1 Structure

There is an UML diagram in picture 1.2 for the basic structure of the backend. This probably isn't a perfect model of the backend but gives a good reference for its structure and what I should implement. The structure of the code will be split into controllers that handle the calls to the addresses, services that handle the logic and repositories that handle the calls to the database.

The authentication will use JWT and with this I will also use Spring Security to secure the application. I will also implement this so that users can only access their own data.



Kuva 1.2: Backend structure uml

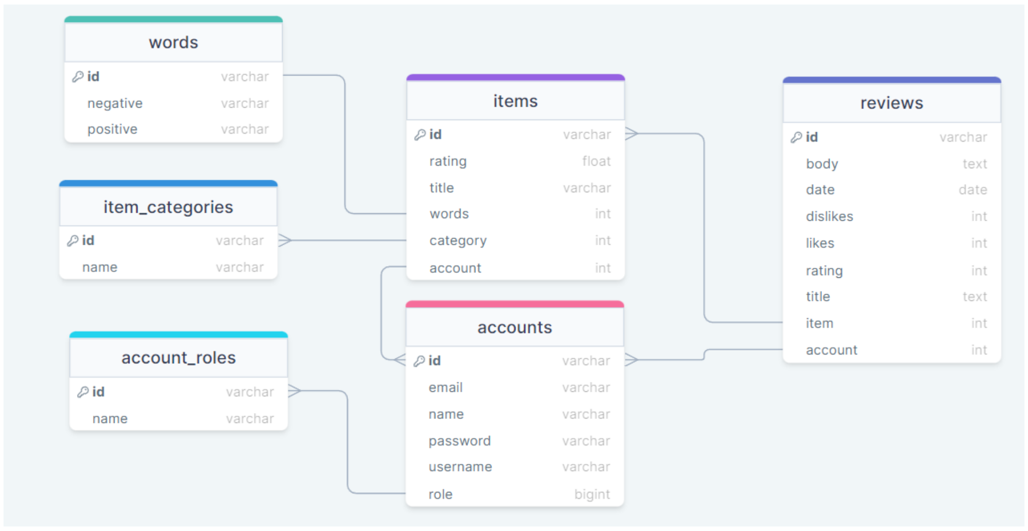
1.5.2 Technologies

The backend will be created with Java using Spring boot. I chose Java because I have some experience using JavaScript and Node for backends, but not any with Java, which I feel is a lot more important for creating structured backends. I also don't have many projects that showcase my skills in Java, so I think this is a great fit to show my OOP skills and specifically applying it to backend development. Spring was chosen because it is the most popular framework for Java. The testing will be done with JUnit and Spring MVC. I will also create a CI/CD pipeline that will test automatically and in the end this will be deployed into AWS.

1.6 Database

1.6.1 Structure

The database's role is to store all the data that is being used in the application. The structure and also the relationships between the tables are represented in picture 1.3



Kuva 1.3: Database structure

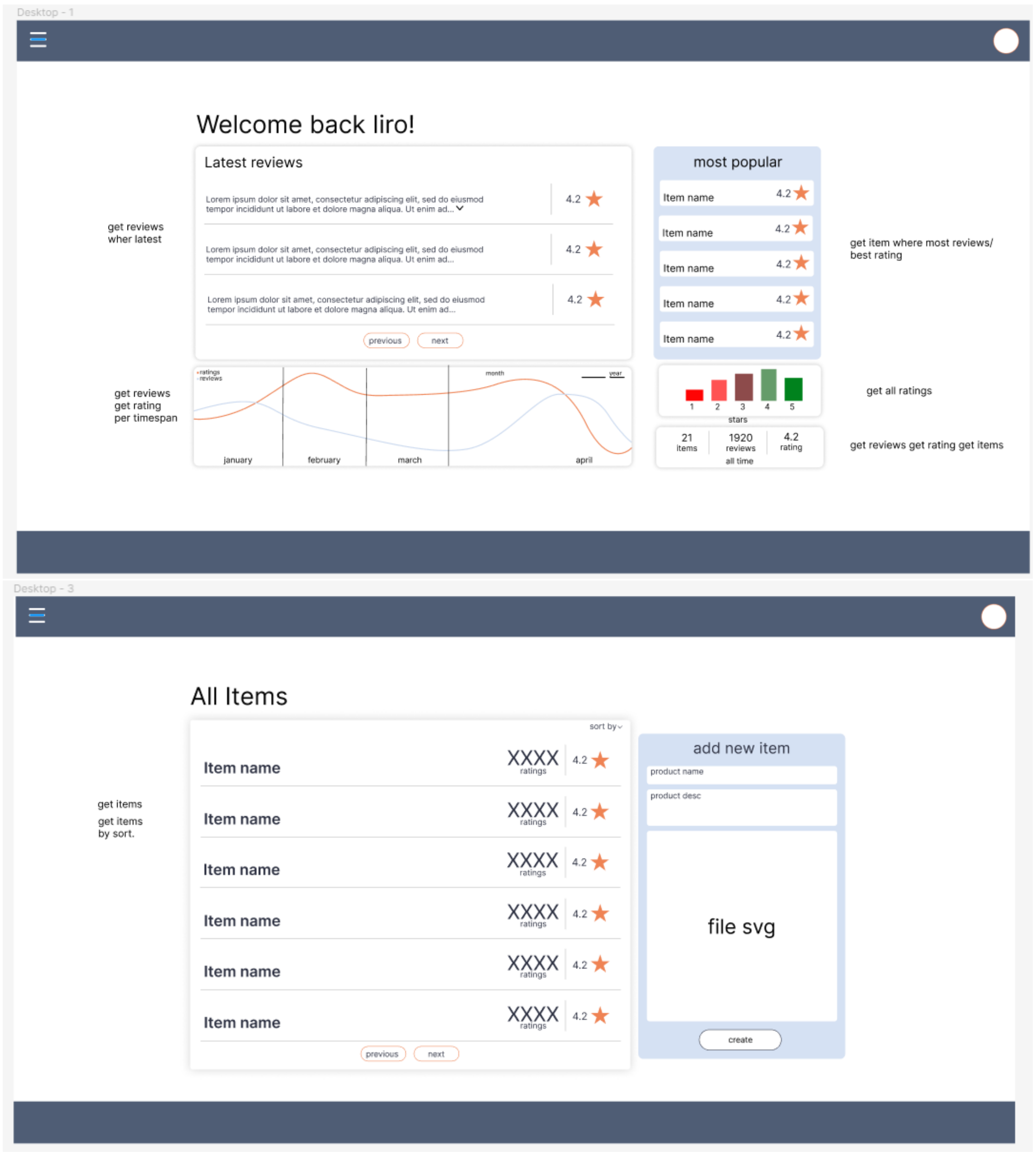
1.6.2 Technologies

The database will be created with MySQL. I chose SQL, because it's something I have learned but don't have any projects where I display my skills. It also is a good fit for this project. MySQL was chosen, because of its popularity. The database will be deployed in AWS and there will be a private connection to secure it. AWS was also chosen because its importance in work life.

1.7 Frontend

1.7.1 Structure

The frontend will be mocking a seller admin page of an online shop. The mockup of the whole application and its pages are in pictures 1.4 and 1.5. The structure of the frontend will be a basic grid layout with a header and footer. The navigation will be done from a navbar on the side of the page. There will also be linkages into the items and so on from their respective lists and reviews. In the overall structure I will try to split the application into as many components as necessary and try to reuse components over the application.

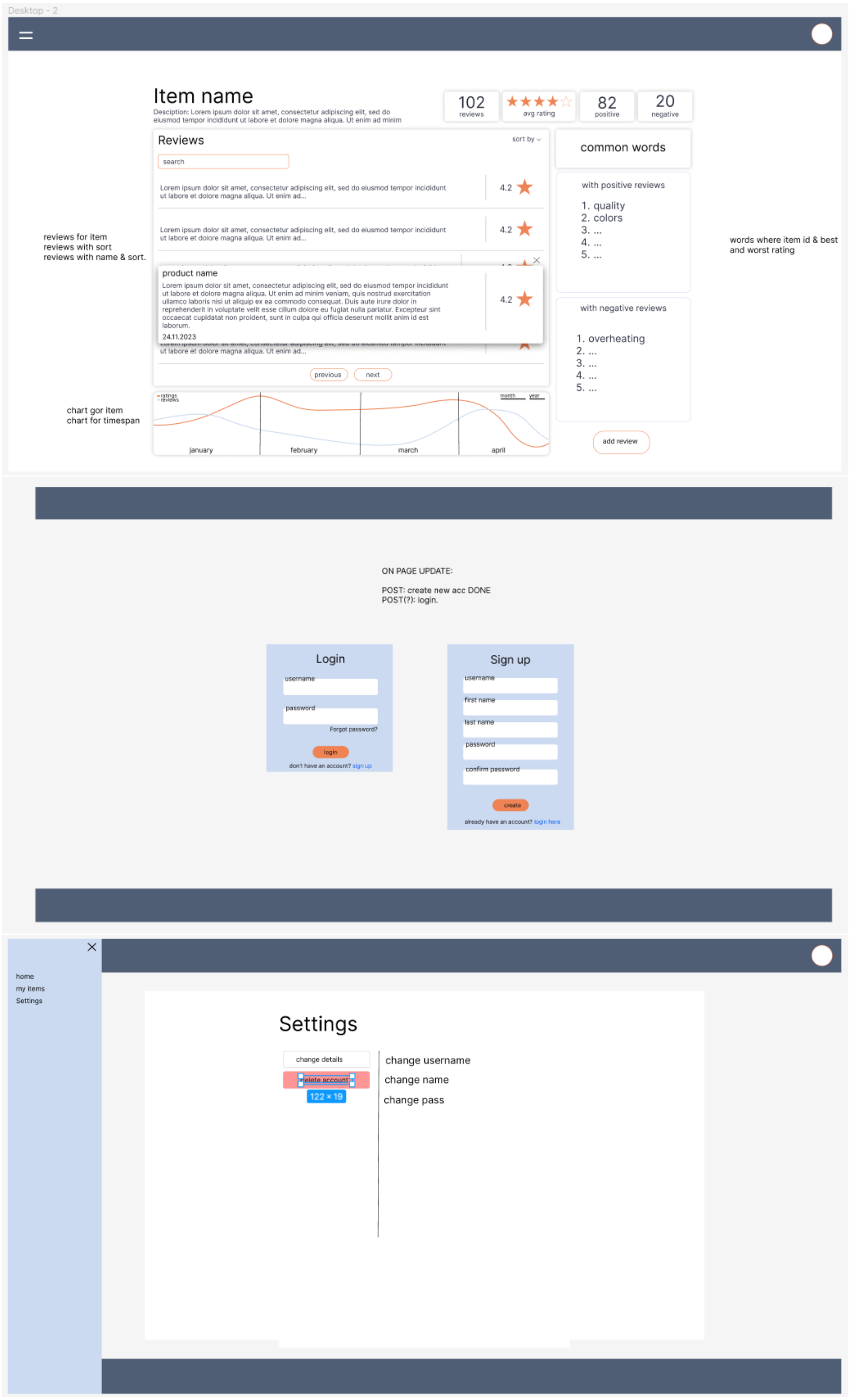


Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad...

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad...

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad...

Kuva 1.4: Frontend mockup pages 1-2


Desktop - 2
Item name
Desciption: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
102 reviews
avg rating
82 positive
20 negative
Reviews
sort by
search
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad...
4.2
product name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
24.11.2023
previous
next
reviews for item
reviews with sort
reviews with name & sort.
common words
with positive reviews
1. quality
2. colors
3. ...
4. ...
5. ...
with negative reviews
1. overheating
2. ...
3. ...
4. ...
5. ...
words where item id & best and worst rating
ratings
reviews
month
year
january
february
march
april
chart gor item
chart for timespan
add review
ON PAGE UPDATE:
POST: create new acc DONE
POST(?): login.
Login
username
password
Forgot password?
login
don't have an account? sign up
Sign up
username
first name
last name
password
confirm password
create
already have an account? login here
home
my items
Settings
Settings
change details
delete account
122 × 19
change username
change name
change pass

1.7.2 Technologies

I will be using React with JavaScript for the frontend. This was chosen because of React's and JavaScript's popularity and also the fact that they are the languages I have been learning. The styling will be written with CSS and SaSS. This is because I feel it is important to learn the fundamentals of CSS before using a library like Bootstrap. Testing will be done with Jest.

1.8 Roadmap

I have created a small plan on the roadmap for this project. As I will be doing this on the side of working and studying I can't really create sprints so the planning is a bit open ended. But I will start with creating the API. This is the most important part of the project and if I can't get it to work the whole project isn't going to work, so I will try to get a version of this running.

After this I will create the frontend from the mock. This way I know what data in what format from what calls I want from the backend. It will also be easier to develop the frontend with just mock data at the start. After this I will move onto the backend and create it. This will include combining the API and backend. Lastly I will combine the frontend and backend and thus the project should be ready.

2 Results

The project is now finished. In the end the application reached the goals that I set, but it became a bit larger than I anticipated and thus also took a bit longer than I hoped, but I'm happy with the results. Let's look at the pros and cons of each part of the application in depth.

2.1 Models

In the end I reached about 59% with the feed forward model and almost the 70% witht the BERT model. The results in percentage might not seem that impressive, but when calculating the average of ratings given the model achieves really close results to the original values. This indicates that the model has really high accuracy if we would measure ratings with a plus minus one rating.

I will dive deeper into the BERT model and tell about all the steps, but lets quickly look at the feed forward model first.

2.1.1 Feed forward model

The basic feed forward model achieved about 59% accuracy. This was surprisingly good in my opinion and I didn't expect this. As this wasn't a finetuned BERT model and also as this is the focus point of the model, I decided to not use all the out of the package functions for this model. For example the collator is my own and the vectorizer is just a TFIDF vectorizer. This shows some understanding of the model's steps. Some things to take from the preprocessing is that I decided to add the title of the review to the body by just using ":". In the end with the BERT model the dataset was removed from Huggingface and I had to use a copy of the dataset that included the titles with "\n\n"which was smarter thinking back on it.

The model itself is really basic with just one hidden layer with the hidden size of 25 being the best in the end. For non-linearity I used tanh which worked best here. And lastly for optimization I automated it with Optuna. I added a lot of comments describing what the model does at each step and I think the best way to understand it is by reading the Google Colab file in github for the model.

2.1.2 BERT finetuned model

There is good documentation line by line on the Google Colab file, but let's go through why I used what I did. First of all the dataset, for which I decided to use amazon_reviews_multi, because it had a nice size of 200k for English reviews. Because of Goole Colab I wouldn't really benefit from a larger dataset. And the main reason of using the dataset is the labeling. Most of the review datasets are just positive and negative, but I needed the ratings from 1 to 5 stars. The dataset didn't luckily need any preprocessing in the BERT model as it had the newer dataset. The model used on the other hand is "bert-base-cased". This is because the BERT model performs better than the distil version and for ratings cased is more suitable than using uncased.

For the tokenizer I used AutoTokenizer for BERT. I chose this over the BertTokenizer, because it is significally faster and doesn't lower the performance as I tried.

Now the actual model. I just run through the BERT model and then add a simple one layer feed forward neural network to the output labels. Just for simplicity and because this model already is on the limit of Google Colab, I decided to use FFN isntead of RNN. I ended up using 24 size for the network as it worked best with my own feed forward model. Otherwise the models training is pretty straightforward. For optimization I used Optuna again. The data collator is also just loaded and not implemented myself.

The model reached 66% in the end. This was almost the 70% that I tried to reach and I am a little bit disappointed in this result, but when evaluating the model on large amounts of data it reaches the average value of ratings really well which I am proud of. Also the model very rarely misses the rating by over one rating which is nice to see. So in the end the models single rating percentage doesn't tell the whole story and I am happy with how well the model works in practice.

2.1.3 Top words

The top words are a part of the API, but I will go through it here, because it also belongs to the course work. The top words calculation was done with just basic TF-IDF vectors and calculating the most important words. The function itself can be found in the APIs github in the file getTopWords.py. There is a lot of comments to go through the calculation line by line.

Evaluating the success of this is difficult as I cannot just run this through a function and get a percentage on how good it is, but I have tried it with a lot of reviews and have found some pros and cons. Firstly, the calculating does work. It finds the words that show the item's positive and negative attributes, so the TF-IDF was a good choice. It is easy to search with the negative words to straight away see what are the common worries people have with the item or what is positive about it.

However there are some downfalls. This regards a lot on the reviews being inputted into the model, but sometimes the most common words are the same on the positive and negative reviews, as they are talking about the same item. I thought about some solutions and I could have somehow removed all the words that are in the name of the item or just about the item in general, but in the end this deemed a bit too difficult and effected the results negatively. Another problem was similar words, for example "did not work"and "didn't work"were counted separately. Unfortunately I didn't have a solution for this either.

One thing that I did a lot of experimenting on was the length of the ngrams. In the end I ended up on calculating only two length ngrams. If I used just one, I ran into problems with for example "heart rate", which came up as "heart"and "rate". And if I used one and two it came up as "heart rate"and also "heart"and "rate". In the end two length ngrams fixed this, but also added some good context to words, like "stopped working"instead of "working".

Overall I'm happy with the implementation and it seems to be working fairly well. The words add context to the positive and negative words and mostly describe what is broken and what works with the item.

2.2 API

The API itself was pretty easy to create. Knowing REST APIs and this being a really basic API, the structure was fairly easy. Learning Flask for Python was also rather easy as I didn't need to learn more than the basic stuff. In the end I'm happy with the API. It works well and I achieved what I set out to achieve with the API and the structure was how I planned it. I only needed the two calls /rate and /getTop for the application to work.

In /rate I take in the reviews in the body of the request and then call the Huggingface API with these reviews. I did try to implement the model in the backend, but in the end none of the free deployment options didn't allow enough memory for this, but luckily the Huggingface API worked well. The API returns just the stars for the ratings, so I also created a response where I have a JSON with the reviews and ratings together for all the reviews that were inputted.

The /getTop on the other hand takes reviews that are already rated in and splits them into positive and negative. Positives are ratings 4-5 and negatives are 1-2. Then I calculate the top words, which was explained in the previous subsection.

I deployed this into Fly.io. It was the easiest option as I had deployed there before. This could have been deployed into AWS, but I couldn't because of the free tier there, but more on this later.

Overall I'm happy with the API. The structure is, in my opinion, good. It works well and achieved everything it needed to do. If I would do something better in the future, it would be the tests. Currently the API doesn't have any tests, because in the end I didn't have time to implement them. This is something I would have done differently and at least tested the functionality of the API calls.

2.3 Backend

Overall I'm happy with the backend. The UML-diagram was also really helpful as the finished backend did feature a similar structure as what the UML diagram depicted. Some things I'm really happy with the applications are the structure, testing, error handling and overall execution.

I did end up learning Spring to use with Java for the backend and I'm pleased with this decision. Learning Spring was really helpful and I'm satisfied with the structure of the backend that came with it. There was a lot to learn about Spring and also the structure that the backend should follow. In the end I followed these well, I have separate controllers for each calls that just take in the call and call a function to handle it. All the logic is done in the services classes and all the calls to the backend are separately in the repository files. This helped me a lot when I needed to modify or add calls in the late stages of development as it was easy to modify.

Testing was also done with JUnit as I planned. I ended up reaching 100% coverage on the tests and learnt plenty about testing and especially testing Java applications as I was creating the tests. This was pretty difficult at times and also this went most over my estimated time, but also felt that I learnt a lot about testing and also found bugs and improved the application as a whole.

One thing that I had difficulty with was security. The implementation of JWT and other security measures weren't that difficult, but holes in the returned data did cause some problems. At first I just returned the whole objects, for example get items just returned the item straight from the backend, but this caused sensitive information to be returned also. To fix this I first tried to modify the SQL calls, but in the end I needed to create custom returnable object that the results from the backend can be casted to. It is nice to know that these are things that need to be planned better before implementing them in the future, so I won't waste so much time fixing it.

Other than this the SQL calls to the backend were pretty easy to implement as I knew SQL from previous courses. It was nice to be able to use SQL in an application as it hasn't been used in implementation in courses. I also wanted to do most of the SQL calls myself even tough this could have been done automatically to get some practice with SQL.

Deploying the application wasn't that easy. The original plan was to deploy it into AWS with the database, but in the end AWS only allowed one EC2 instance for the free tier, so sadly this wasn't possible. I then decided to deploy the application to Render.com. This did have one pro as I needed to make a dockerfile for the application. This way I got to learn a little bit about Docker and making dockerfiles and in the end this didn't turn out to be that big of a disappointment. Sadly with the Render the application will wind down in between uses and need a few minutes to get back up.

The backend of course isn't perfect and there are some things I would do differently if it had to be done again. One of the biggest mistakes was some of the coding. Especially the pages service is in my opinion pretty messy code and could be written a lot better and cleaner. This was because I ran into some problems with the implementation of the backend to the pages and I had to fix up the logic in the backend. This made some of the logic really messy and I should have planned this better.

Otherwise the coding was mostly good. Especially the error handling I'm happy with. I caught most of the common errors, for example with checking passwords and so on and returned corresponding errors that caused them. This helped to display some errors in the frontend and also debugging the application was a lot easier, so it was also helpful in the end. For documentation I ended up using JavaDoc for it all with also writing a README, so I think this does help a little bit on the messy code. In the end I'm happy how this displays my skills with Java and backend development.

2.4 Database

The database was pretty simple to implement. Knowing SQL from before I just had to learn how to use MySQL, but this wasn't hard. Creating the backend was quick and easy, the structure of the database was also easy to create as I had planned it in advance. In the end the structure didn't really change at all and the structure in the planned section is the same what it is currently.

The implementation was a bit more difficult as I had an SSH connection to use the database. This taught me a lot about security and using SSHs. The most difficult part that came with the database was the deployment of the backend as I had to create the SSH to be able to deploy it, but in the end I was able to do this in the dockerfile of the backend.

The database was the first to be deployed so it was the one that was deployed into AWS. This and also the attempt of trying to deploy the backend into AWS did teach me a bit of the usage of AWS and it did help me learn it. Overall I'm pleased with the database and in using SQL it filled all the goals I wanted from it, I wouldn't really do anything differently if needed to be done again.

2.5 Frontend

The frontend pretty much followed the mockups that I had created. My plan to develop the frontend based on the mocks with mock data was smart and really helped with developing the frontend and to see what kind of data was needed for it. Some things I'm happy with the frontend are its structure, overall coding, the achieved goals with the application and testing.

I had a clear plan on what I wanted to implement for the application and it was nice that I was able to follow the mocks and implement everything I had planned at the start. There wasn't really anything that I wasn't able to implement.

I did use only React and JavaScript for the coding and from previous projects the one thing I wanted to improve on was using React and also structuring the applications better. I tried to utilize React better now than in previous applications that I had created. I feel like I achieved this and especially used more custom hooks and also React's own hooks. This taught a lot about React.

This time I did some good structuring and split the application well into smaller components that were able to be used in different parts of the application. This really sped up the development when a list of reviews only needed to be coded once and then could be used again in different parts. On top of this I split the application well into their own pages and these into their own parts, which included the components that could be reused in different parts. I also split the service calls to the backend and every pages hooks into their own parts.

With the structuring I also tried to keep every component, page and the coding as clear as possible. Other than few parts where I found the coding was a bit clumsy I feel like I achieved this. None of the files are extremely long nor include too much code that is difficult to understand. The documentation I did also helps with this as I documented all of the code with JavaDoc. I also like how the code turned out when I implemented all the functions in the main elements for example Item.js, but the calls are in child components. This really helped to reuse the child components, but also made testing a lot easier.

Other than some of the sloppy coding that I used there are a few things I would like to do better in the future. I would like to utilize HTML better. It feels like sometimes the code is a lot of just divs inside divs and this is something I could become better at and utilize HTML better. Some of the component could also be done better. In the end of the development I just saw myself adding more and more to the already done components and not creating new components which is something I will try to do better in the future.

The testing on the other hand I did well with. I only implemented unit tests and not any e2e tests as I didn't have any time for it. The testing is done with Jest and same as in the backend this took way more time than I anticipated, but I learnt a lot during this and I was able to achieve almost 100% coverage on the application. I learnt about Jest and testing a lot, but also this helped me find a lot of bugs and overall improve the code. I also implemented a CI/CD pipeline for this application.

As a whole the development went really well in my opinion. Other than testing I didn't have extreme difficulties with any task, but something I didn't realize to plan was all the loading states. This was something I realized after implementing the backend to the frontend. And after this I had to add all the states of loading, which could have been easier by doing this in the mocking stage. So this is something I need to remember in the future, to think about all the states of the application while mocking it.

Styling is something I'm pretty happy with. Being able to produce what I planned in the start is nice, but this is something I feel like I do need some more practice with as there is still a lot to learn. I will hopefully create a more styling based application in the future.

This also wasn't deployed into AWS as it wasn't possible, so this is deployed into Vercel.com, which I had used before. In the end I am happy with the end result of the frontend.

2.6 Summary

Thinking of the application I'm very proud of it. I feel like I achieved all the plans I had created and also the implementation went well. The machine learning models that this application was built around exceeded my expectations when implemented on the application and also taught me a lot about Transformer models as I was creating them.

The application itself came out also really good. All the parts are well written and have good structure that is easily expanded or to be modified in the future, which was really important as it was something I tried to achieve with this application.

The one lesson to learn from this, is the importance of planning the applications. Even tough I tried to invest my time into this it still felt like it could have been done better and that I could have saved a lot of time with it. Also something negative was the time that I spent on this project. Looking back at it, the applications plan was pretty big, but it took a lot longer than I anticipated. This is nice to learn now tough.

Overall I'm happy with the application. All my goals feel met and I learnt a ton on the way of creating the application, so it feels like a success.