Ονοματεπώνυμα και ΑΜ

Παντελεήμων Μαλέκας - 1115201600268
Θεοφάνης Μπιρμπίλης - 1115201600110

Rent-a-Ton

A group project for the class: Web Application Technologies
2019-2020

Table of contents

● Introduction

For the following assignment, we developed all of the following questions, including the bonus question, using React JS for the front-end and Python Django-Rest for the back-end. Also, PostgreSQL was used for the handling of the database, along with PostGIS, an extension that allows the use of geographical data.

Regarding the front-end of our application, we used React, a JavaScript library commonly used for web development. React makes use of (function or class-based) components. A component usually contains a render function that displays JSX elements on the screen. Along with 'classic' CSS, we create the UI needed for the application. React works by rendering one starting component named 'App'. All the other components(for example pages, navigation bars) are rendered through the use of this App component. In order to access data from the server, we used 'axios', an HTTP client that supports the basic HTTP requests.

For the back-end of our application we made use of Django and its Rest framework. Django is a Python framework commonly used for web development and server management, in particular. DjangoRest is also a framework that can be used in Django to use REST capabilities in the application. In Django, an app can be created. In an app, models are defined and through the use of DjangoRest we can create serializers in order to create and get instances of the models we defined, in the format we need for our application. Views and viewsets are defined in order to specify the actions that can be performed on a model and URLs are defined in order for someone to access those views.

Finally, in the Django project we create, there is a settings.py file where we specify the settings we need, such as the definition of our database. The PostgreSQL database we mentioned earlier, is specified in this file. It should be noted that our database is set up locally, so if one wishes to use it, it will have to be created from the ground up.

- ## Question 1

For this question we created the Home page of our website as well as the Navigation bar, with all the necessary work needed in the back-end too.

In React, we defined the App component which renders all the other components and passes some important information to every one of them (such as whether the user is logged in or not, his primary key etc). In order for the App to render multiple pages, we used react-router-dom, in order to specify the URL of each specific page. That way the App can switch from one URL to another when needed. By default, the App renders the Home page, which is needed for this question. The Home page is different for each user-type. For anonymous users, a search form is defined so they may search for rooms. Renters have the same form, in addition to a link to their reserved rooms and a button used to get recommendations. Admins and hosts cannot search for rooms, so the home page includes only a welcome message.

The Navigation bar was also defined in this question, and is used by every page. The navbar includes a link to the home page on the top left. If an admin is logged in, there is also a link to his Admin Page, which is defined in a later question. A link to the Host Page is similarly included for hosts. Every logged in user can also see a Edit Profile button on the right part of the screen. Logged out users instead have a Register button which redirects them to the Register page, so they can create a new user. Finally, on the top right there is a button for a user to log in (or log out if they are already logged in).

If one clicks the Login button, he will be directed to the Login page where he will be instructed to type his username and password. If there is some type of error, the user will be alerted. If the data is correct, the Login Submission is handled in the App component where a post request is made in the API endpoint: ‘authentication/login’. The endpoint was taken from the django-rest-auth (https://django-rest-auth.readthedocs.io/en/latest/introduction.html), which provides all the authentication capabilities we need (only the login endpoint was used, since only that was necessary). After the post request is made, if it was successful, we get the data of the user who logged in, as well as a JSON web token. This information is kept in the state of the App component, so it can be passed in all the other components. Finally we redirect the user to the home page. If a user logs out, we clear the web token, set the App state of the user to a default one (used by visitors), and direct him to the home page.

Last but not least, regarding the SSL encryption, we used a script in order to create a self-signed SSL certificate. Also, HTTPS is set to true in the package.json file and an env.development file was created to define some environment variables needed for the use of the certificate. This certificate is set to be ignored from the browser in order to use the application. If one wishes to use the application, since the certificate is self-signed, he will have to add it in the ‘Ignore’ list of the browser in order to properly use the application.

- ## Question 2

For this question, a Register form was created so a new user can create an account to use the application.

After clicking the Register button in the navigation bar, an anonymous user is directed to the page containing the Register form. If the user clicks the Submit button and there is some kind of error, there will be error messages displayed in the fields that were incorrectly filled. If the submitted username is already used by another user, a message will be displayed prompting the user to enter a new one. If all the data is correct, an axios post is made in order to create a new user and add him in the database. In Django, a user model was defined in order to handle the users in the database. After the successful creation of a new user, an appropriate message will be displayed on the screen and, if someone chose the role of the host, an additional message will appear informing the host that he will have to wait until an admin approves of him. Finally, after the submission is complete the user will be directed to the home page.

- ## Question 3

After an admin logs in the application, there will be an Admin Page button in the navigation bar so he can navigate to said page. We should note that an anonymous user cannot choose the role of the admin, so we simply created some admin users while setting up the application. In the Admin Page there will be a message containing a link to the page with the users list and two buttons which can be used to export data, one button for XML format and one for JSON.

- ## Question 4

In the user list, the admin can see a paginated list of all the users who exist in the website, with some basic information about each one. By clicking on one of the users, he will be directed to the detailed view of a user. Here he may see some additional information about the user, and if the user is a host there will be a button to approve this user (or disable him, if he wishes). When an admin clicks on that button, a post request is made on the 'approveUser' endpoint of the Django server. In the view of this endpoint, the user is located and his status is updated and saved in the database.

Regarding the export of the data, by clicking on one of the two buttons, a message will be displayed informing the admin that export has begun and will be notified upon completion. Accordingly, a message is displayed when all of the data has been exported. When a button is clicked a post request is made on the 'ExportData' endpoint of the Django server where the server collects all of the following data: Rooms, Reservations, Room Ratings, Host Ratings. These data are then exported in XML or JSON formats, depending on what the admin chose (Note: the files are stored in the Django project folder).

- ## Question 5

An anonymous user or a user with the role of the renter can search for rooms using the search form in the Home page. The form contains the following fields: Neighborhood, City, Country, Starting and Ending Dates and Number of People. After clicking on the search button, a post request is made on the 'SearchRooms' endpoint of the Django server. In this endpoint, we get the rooms that meet the criteria specified in the search form. The rooms are then serialized and returned to the front-end (or a 'not found' message if no rooms were found).

After the data is returned to the front-end, the rooms are paginated and sorted according to their price so they can be displayed to the user. The user can see some basic information about each room and click on it to get a detailed view. Also, on the top of the page there are some additional search filters where a user can narrow down his search results. After specifying the additional filters and clicking on the Search button an axios post request is made on the same endpoint and the page is reloaded to display the new results.

(Note: Each room included in a search being made is added to a SearchedItem model defined in the Django server, that is later used to make recommendations)

- ## Question 6

After clicking on a room, a user can see all the necessary information about a room and perform the actions he wants (if he is a renter). Specifically, anonymous users can only view the data of the room. Renters can also make reservations, if the room is available for the dates specified in the starting search. Otherwise, the ‘Make a reservation button’ won’t appear. If a renter clicks on the said button, a ‘reservation’ item is created and added in the database. Also a message is displayed informing the user that the reservation was successfully made. After that, the page is reloaded.

After making at least one reservation, if the renter navigates to the detailed page of the room a check is performed regarding whether the user can leave a rating for the room and/or the host of the room. This check is performed using the ‘RatingCheck’ endpoint of the Django server. Here, we check whether the current date is after the end date of at least one reservation made by the user. If so, the renter may leave ratings about the room and the host.

Also, by clicking on the host information in the page, the user is directed to a page where he can create a message and send it to the host. The ‘Message’ model and all of its related views are defined in the Django users app.

A renter may see all of his reserved rooms, by clicking the specified link on the Home page. The page containing the reserved rooms also includes a link where a renter can access his messages. In this page, he has the option to create a new message, or to view his sent and received messages by clicking on the buttons specified in the page. By clicking on a sent message the user will be directed to a detailed view where he can update or delete his message. Likewise, on a received message, the user has the option to leave a reply.
(Note: Whenever a renter clicks on a room it is added to a ClickedItem model defined in the Django server, that is later used to make recommendations)

- ## Question 7

A host may navigate to his Host Page by clicking on the related button on the navigation bar. If the host is not approved yet the page will inform him that he is not able to access the page and that he will have to wait until further notice. Otherwise, the Host Page will include a link where the host may access his existing rooms as well as a form where he can add a new one. If the host clicks on the link to access his existing rooms, he will see a paginated list of his rooms and by clicking on one, he can navigate to the detail view of the room. In the detail view, there will be a link to the host’s messages (all the message-related handling is done same as for the renters), as well as the option to edit all of the data of this specific room.

If a host wishes to edit the additional images of one room, he will be directed to a page where he can handle them. One image ‘slot’ is reserved for the photo that represents the room. All the other images are regarded as ‘additional’ images. So, in the Room Images page, a host has the option to add new images, or to update and delete the existing images.

- ## Question 8

Any logged in user can navigate to the Edit Profile form, by clicking the related button in the navigation bar. The said form contains all the data the user has submitted and he can edit all of them (he cannot change his role however, so this wasn't included on the form). If the user clicks the Submit button and there is some kind of error, there will be error messages displayed in the fields that were incorrectly filled. If the submitted username is already used by another user, a message will be displayed prompting the user to enter a new one. If all the data is correct, an axios patch request is made on the server, and the existing user is updated on the database.

- ## Question 9

In order to develop the bonus question, we used the AirBnbDataset, provided in the e-class folder. Before we can make use of it, we made some cleanup on the data in order to suit our needs. This work is done on the Cleanup.py. In this file, we kept only the data fields that were important to us, meaning the ones to be passed in the models we defined. Plus we filled some missing data using their average values etc. After making the changes we need, we exported the data to two new .csv files, new_listings.csv and new_reviews.csv. Those new files are then used to populate our database with more entries. By using those files, we added more users (host, renters and both), rooms and room ratings. This work is done on the Database_Populate.py. The population, apart from the bonus question was also used in the previous questions to test the speed during searches for rooms and accessing the user list.

After adding the data we needed, all we had to do was to create a Matrix_Factorization.py file in order to create recommendations. First, we take a subset of our data. Since we have way too many users (30000+) we used a subset of the users as well as one for the rooms. After getting the rooms and users we need, we create an array containing the 'real' values. The script makes use of the Ratings, Reservations, Searches and Clicks. For the latter three, we added a 3 as their value on the array since this is the middle value of a rating (meaning a 'neutral' one). For ratings we add the average of all the ratings the user has made on the room. Unknown values are set to 0 and are saved to a list so we can know their positions in the array. Then, we get the rows and columns of the 'real' array and set a number for the latent features. With those numbers, we initialize two random P and Q arrays. Now we finally call the Matrix_Factorization function in order to execute the algorithm and get results.

The algorithm works as follows: We execute the algorithm for 'steps' times (given as an argument). In each step: First, we calculate an error variable by taking the difference between each (known) value in the real matrix and its corresponding value in a predicted P*Q array. Then, we update the values in P and Q using K latent features. The idea for gradient descent formula used for the updating was inspired from the following link: http://proceedings.mlr.press/v36/li14.pdf.

After the error is calculated for each known value, we create the updated predicted array with updated values of P and Q. Finally we calculate the Mean Squared Error and save it in a variable, to evaluate the accuracy of the algorithm. Ideas for the calculation of the Mean Squared Error used by: https://www.diva-portal.org/smash/get/diva2:927190/FULLTEXT01.pdf
Also, we set a boundary for the Mean Squared Error. We may want to stop the execution of the algorithm if its value is good enough. Finally, we return the final version of P, Q and the Mean Squared Error. Using P and Q we get the final predicted array. In this array we get the values that were predicted, and we sort them getting the highest ratings first. Using the predicted values, we start creating 'Recommendation' items (defined in the Django rooms app) and we stop creating recommendations if they reach a certain 'top' number.
In the front-end, a renter may search for recommendations by clicking the related button on the home page. Then, the user gets a paginated list of the recommended rooms, and he can navigate to their detail view by clicking on them. We should note that since no dates/people etc are specified when getting recommendations, a renter can only view the room's data, so he can't make reservations, leave ratings etc.

● Epilogue

In conclusion, we developed all of the required questions as accurately as possible and without deviating too much from the presentation of this assignment. There were however a few cases where we experienced a few difficulties, but we tried to solve them the best way we could. For example, one such case was regarding the data of the AirBnbDataset. The given data had rooms with dates dating back to 2015-2016. Normally, an actual web application like the one we made wouldn't allow for a user to reserve rooms 'back in time'. However in order to make use of the dataset we decided to allow for a user to reserve a room, even if it's in a past date. The only 'side effect' of this decision is that whenever a renter reserves one of these rooms, he will be able to leave a rating immediately after making a reservation, but we didn't consider this a huge issue. Also, we should note that in calendar.csv there are no indications as to whom users reserved the specified rooms. For this reason, we only used the starting and ending dates from the calendar.csv and we decided not to add any 'new' reservations from this dataset. Finally, the presentation didn't mention anything as to how a renter can leave ratings. Nevertheless we decided to come up with our own ideas for this feature.

Note: Details regarding the setup and installation of the project can be seen at the 'Setup and Installation' files of each folder (one on the front-end folder and one on the back-end folder).