ΑΜ ΚΑΙ ΟΝΟΜΑΤΕΠΩΝΥΜΑ:

1115201900067 ΚΑΖΑΚΟΣ ΠΑΝΑΓΙΩΤΗΣ
1115201900048 ΔΗΜΑΚΟΠΟΥΛΟΣ ΘΕΟΔΩΡΟΣ

AUCTIONS

Web Applications & Technologies class (2021-2022)

Contents

INTRODUCTION

For the development of this project we decided to use python Django as the back-end and React.js for the front-end. The relational database management system we are using is PostgreSQL, since it is recommended from the Django documentation. The purpose of this project was to create an auction site which allows to users to create their own auctions or make bids to other users' auctions. The web application supports admin, user and guest roles. It should be noted that we managed to complete almost every requested functionalities except for displaying a map to determine the location of the seller, having a messaging system between the winner and seller of an auction and giving the ability to the admin to export data of users in both xml and json formats. The name of the web application is kept as simple as "Auctions" but some suggestions would be "Golden Auctions" or "SeeAuctions".

INSTALLATIONS AND SETUP

To begin using this web application on localhost domain you will need to install quite a few python pip packages to get the Django functionalities. These packages are listed in the requirements.txt file at the root folder of this project. To install you can execute "pip install -r requirements.txt". Then, after making sure you have installed PostgreSQL on your system you need to create a new database called "auction". The name of the database as long as the password and other information for accessing it can be set in settings.py file. Before running the server you can create a new admin by running "python3 manage.py createadmin". In order to achieve ssl connection on localhost you will need to create you own certificate on your machine. This can be done by using the "mkcert" tool. After having installed mkcert, run "mkcert -install" to create a local Certificate Authority CA and then at the root folder of the project run "mkcert -cert-file cert.pem -key-file key.pem localhost 127.0.0.1" to create a certificate called cert.pem and its key called key.pem.

After the above configurations, you can go ahead and run the server over https by executing "python3 manage.py runsslserver –certificate cert.pem –key key.pem". Note: in order to login the users you have registered you have to approve them first, from the admin page. This is further discussed in 'Login Users' and 'Admin Page' sections. There are also two additional scripts provided, one for populating the database with data from the given dataset and one for calculating the top 3

recommended items for each user. We have tested that our scripts work for every item of items-0.xml file. You can run “python3 manage.py db 0 0 0 10” which gives zero as an entry number for phone and TIN counters and starts filling database from 0 until 10 items of the dataset of items-0.xml. To calculate recommendations run “python3 manage.py calc_recom”. If you need to make any changes to the front end you should install additional npm packages and then build the react project again. Otherwise there is no need to install anything for the front end.

REGISTER USERS

From the index page there is a button that leads to the page that registers a new user by completing a form. The register button sends a post request to the Django rest api. All our requests from the front end are handled with “axios” library. Also, all the requests are handled in the back end from django/auction/api/views.py file. Anyone has permission to the service of the server that creates a new user, so in this case there are no credentials given in the request. With conditional rendering we inform the user with appropriate messages if request fails depending on the situation (“username already exists” etc.). If the request is successful, a new user is created in the database with the data given to the form, but he is stilled unapproved which is implied by a boolean column. Then the user is notified from another page that he has to wait for his application to be approved in order to login as a user, so for now he is prompted to continue as a guest. Unapproved users cannot have access to any of the services of the back end server that restrict to only authenticated users, because they are a not allowed to get any access tokens. More about access tokens in Login Users section.

ADMIN PAGE

The admin can navigate to the admin page under /*MyAdmin*/. From there he can navigate to ListUsers page or ApproveUsers page. Any other user who is not an admin member will be redirected from the front end in case he tries to navigate to above pages. This is done in react/auction/src/Views.js file. If however someone

manages to access these pages he would not get a response from the server because the permission is set to admin users only. From ListUsers page the admin can see all the users by their usernames. He can then click to each user and see more details about them as well as approving them in case they are not approved yet. Notice that the link in the front end is changing accordingly to user's username. From ApproveUsers page, admin can view directly all the unnaproved users' information and click on a button for each user to approve them. In both pages, pagination would be essential for the scalability of the project, but we ignored it for the time being.

LOGIN USERS

The page that logs users in is located at the index page path. So whenever a new user visits our website who is not already logged in will be greeted by the login page, which has a form for logging in as well as button to register and a button to proceed as a guest. Similarly with register page, in this page the user is notified if the login has failed and why. Unapproved users who try to login get an error "invalid credentials". When submitted, the form sends a post request to the view that responds with tokens. The authorization process is handled by the simple-jwt package. If a user is allowed to get a token he gets back a JSON web token as an access token and another one as a refresh token. From the settings.py file the tokens are set to rotate, meaning that every time a user gets a new set of access and refresh tokens the previous tokens are blacklisted. Also, the duration of the access and refresh tokens is set to 10 minutes and 30 days respectively.

From the front end if a user is logged in there is a time interval that will trigger a post request to the refresh token view every 9 minutes. The refresh token request should not be triggered right at the time expiry of the access token because there might be some delay to get a response. The tokens are stored in LocalStorage and the web application does not use cookies at all. If the user logouts from the logout button the tokens get removed from the LocalStorage so the login form will be rendered. Otherwise the home page is rendered. Also, if the user leaves the page without logging out the tokens will remain inside LocalSorage when he comes back. If the access token has expired then there will be an attempt to refresh the tokens and the user will be directed instantly to the home page logged in. If the refresh token has also expired, then the tokens will be removed and the user will have to provide credentials again.

HOME PAGE

The home page path is the root of the URL, same as the login page. The home page will be displayed only if a user has logged in or if a user has proceeded as a guest. The home page consists of a header which acts as a navigation bar and a sticky footer. The header and the footer are also included in the Manage and Browse pages. The home page is filled with random text generated by Lorem Ipsum text generator because we didn't have any content to display. Some suggestions would be to display the most popular auctions by number of visits or by number of bids etc. In the navigation bar for the links we used the <Link> tag to navigate to the rest of the pages and <a> tag when a click occurs to the same page in order to refresh it. This happens because React.js is a single page application so we need to use <a> tag in this case.

GUESTS

When someone proceeds to the home page as a guest, a key value pair is created in SessionStorage to remember that the user is a guest. This is important because if the user refreshes the page, without the SessionStorage item the login form would have been displayed instead of the home page, because all the temorary information such as javascript variables are lost. SessionStorage removes items if the tab is closed so the user will be greeted with the login form if he tries to open the web page from another tab. Guests also can only see Home page and Browse on navigation bar and two buttons to Sign in/ Sing up instead of a logout button. With same pattern as admin page, guests will be redirected from the front end if they try to access Manage page.

MANAGE ITEMS

In this page a user can create a new item, list his items that are available for auction and list all the items on which he has at least one bid. For creating a new item the front end provides a button and a modal with the suitable form. Then the item is temporary saved and the user can either edit, delete or start an auction for each item. To start the a new auction the user has to provide a date and time. Before making an item available on auction the item's Started and Ends datetimes are set to null. The Started datetime is set to the current date and time of the server (set it UTC) and the Ends datetime provided by the user should be past to Started datetime. Before creating a new auction the user will have to confirm his option. In the items that are available for auction and those that a user has placed a bid, the user can view a list of all bids with their values in us dollars. For the items on which the user has placed at least one bid he can view his last (biggest) bid on each item.

BROWSE AND SEARCH ITEMS

Users and guests can list all the items with pagination in the Browse page. The items are displayed in a box with border to make it easy to distinguish from the rest. At the top of the Browse page there is a form that receives the name of an item as text. Additional values can be added to the form from the modal that is displayed when clicking the "Add filters" button. This form sends a get request to the server with the values of the form as parameters. By default the browse page sends an asynchronous get request to the SearchItems view. The SearchItems view checks if there are any parameters given and if so the items will be filtered. Otherwise the Browse page will display all the items. Also, SearchItems takes into consideration the top 3 recommended items, which are stored in the VisitsAndRecom table for each user, and displays them first. It should be noted that the recommended items as well have to pass through the filters before being returned at the response. Also the link on the front end has the values of the form as parameters in order to make it possible navigate to this specific link and instantly make the same request. Each item's name is clickable and from there a user can see more details about the item, like description and the list of bids. Furthermore the user can place a new bid which has to be bigger than the previous ones. The front end link changes depending on the id of the item. As far as the pagination there is appropriate conditional rendering done in order to display the correct number of pages as well as Previous and Next button. If a user tries to access a number of page that does not exist he will get as a response Bad Request.

RECOMMENDATION SYSTEM

In order to make a recommendation system work firstly we needed to implement the algorithm Matrix factorization as asked in the assignment.

Matrix factorization implementation

Abstract implementation

The input is a 2d array and the output is a 2d array again. recom_src.py contains the mathematical logic abstractly.

Automatic constructor

The constructor of Factors, by default, prepares all that's necessary to make recommendations, and will take a while. Whenever data updates, the caller is expected to create new
Factors from scratch.

Database memory useage

The latent factors are 3 at minimum, to help with testing tiny datasets. On a large scale they are about 1/3 of the width, specifically, they're 1/3 of the smallest of
the 2 dimentions of the table. This can be memory inefficient for square input tables and it can certainly be suboptimal, but testing these scenarios wouldn't be easy.

Unused functions

You may have noticed functions in recom_src.py that are actually not used anywhere. They were used for testing. The expectations were quite low and most of the functions were used for trying out scenarios and values. Another reason is we were uncertain about the implementation. There was a file that didn't make it to the final zip and it was a wrapper or interface, with embedded unit tests and a function that calls all of them optionally. But the provided functions were not helpful for the final implementation.

Testing

Testing was done with groups of people, scattered. The challenge was to make the algorithm recommend items to people of similar general interests. For excentric people or newcomers, we'd want to recommend generally popular items. However there were limits to what results we can comprehend. Function.error did help a little for testing. However that's not its purpose, we tested by eye using Function.print

Function.print_order_of_recomm_per_user so that we could tell if the popular items were recommended to too many people, as well as to tell what would be the first recommended items for each user. Long vertical lines on the printed tables are a bad sign.

Training the data

Preparing the recommendation data consists of counter weighting popular items, so they'd be recommended less, using Function.de_popularise and then training. The order of the called functions can be seen in __init__. Training is done using randomness and natural selection. Function.error judges the quality of a Factors object. Function.squared_distance_derivatives returns instructions for tweaking a Factors object for a hopefully better result. It's the derivative, but the local minimum is unknown. There are two phases in training, one where many generations are tweaked a few times draftly to prove potential, called fat and one where the winners are trained carefully, called grinding. In the end the best one is returned. To avoid memory leaks, memory is freed periodically during he first phase. So for every few consecutive generations dthe best one would be kept and the others would be freed.

Then in order to display the recommended items we had to create a new table called ‘VisitsAndRecom’. It is called like that because it stores the visits of a user for an item as well as the top 3 recommended items of each user. The recommended items will change each time the python script calc_recom is run. This script gathers all the visits and the bids of a user to each item and increases the his rating for each item depending visits and bids (0.1 for each visit and 0.3 is added for each bid). The max rating is 1. For unrated items we decided to give a rating of 0. Latsly, SearchItems view called by Browse page returns firstly the top 3 recommended items for the users and then the rest of the items.

EPILOGUE

During our development we figured that there might be a lot of ways for a user to make directly malicious requests and get unexpected behavior from the web application. Having said and some missing functionalities that would be essential, the code needs a lot of work before deployment. Also, in some cases the code could have been simplified or be more organized. For example, with the use of nested serializers, the categories of an item could have been returned with much simpler code and probably more efficient.