documentation

SE Sprint 01, Team 34

Ishwor Giri & Muhammad Shahzaib Tahir Awan

DATE: 9 March 2021

Software Requirements:

Python,Pip,mysql

Django please go through this tutorial once before starting working on this project

https://docs.djangoproject.com/en/3.1/intro/tutorial01/

Node.js, npm , all the dependencies inside frontend/package.json

Make sure you have python and pip installed first

```
#Install Virtual Environment first
python3 -m pip install --user virtualenv

# Creating a virtual environment
python3 -m venv env
#Activating Virtual Environment
source env/bin/activate
```

If everything works well then you should be inside the virtual environment where you can install packages such as django,

```
python -m pip install Django

#or

python -m pip install -r requirements.txt
```

Change Directory to the project

```
cd beergame
```

Database Migration

```
#make Migration
python manage.py makemigrations
Apply Migration
python manage.py migrate

Start Server
./manage.py runserver
```

Please follow readme.md which can be found on the github repo of this project.

PROJECT STRUCTURE

There are three folders inside beer game project api, beergame, frontend and one main manage.py python file that is useful for running all django commands such as runserver, making migrations and many more.

1. **BEERGAME**

This will be the main folder which contains settings.py, urls.py

Settings.py will be the settings for the project. It contains configurations like INSTALLED_APPS which contains applications installed for the project (frontend,api) and also database configurations(sqlite3 by default).

Urls.py contains all the configurations related to routing. For example

```
re_path(r'api/?', include('api.urls')),
```

Will include another urls.py from the app api, so that all the endpoints after api/ will be handled by urls.py file inside api folder.

2. **API**

This will be the backend application that will handle all the requests related to the game such as login, fetching data , database models.

Urls.py contains all the endpoints after /api/ and displays or calls the views(functions) from views.py . For example

```
path('login',api.views.loginvalidate),
```

Will call the function **loginvalidate** inside views.py

Models.py contains all the models that will later on translate to database tables using django migration features.

Views.py contains all the functions that return some response which can be seen by the end user if views are being rendered/called using url endpoints . Normally all the models import and user request using GET,POST methods and validation , database should be done in this file and response can be returned using httpresponse, jsonresponse with respective response_code. For example

```
        def loginvalidate(request):
    # TODO VALIDATE AND RETURN
                return JsonResponse({'error': 'not authorized'},status=401)
```

Above function must be imported from the library before using
from django.http import JsonResponse

NOTE: For now urls.py contains few basic endpoints just to check if it works or not , models.py contains nothing and Views.py contains functions used for urls routing. We haven't done any backend configurations for the api endpoints.

3. **FRONTEND**

This will be the main application where all the frontend related files will be present.

Urls.py contains only one routing that will point every frontend url requests to one function inside views.py i.e **home_view.**

Views.py contains a home_view function that will render our frontend content which is inside the template folder.

Explaining the index.html file,

It loads static files such as css/ javascript contents and includes it in the html code. Everything related to frontend is driven by this javascript file inside static file **/frontend/main.js** Also, some of the resources are loaded from the internet and one can also serve it from locally if one prefers to do so.

The main thing here is to understand how **main.js** is being created since we have many components made up of javascript files and external packages with 100s of imports/includes.

We have used webpack and babel to configure react workflow for this django + react integration. Babel is used for easy coding that allows us to convert JSX format files including html tags to actual js functions which then are used to render the actual website.
Webpack takes care of combining all the related javascript codes that are necessary for building the website. It usually starts to check for one entry point (index.js file) and then keeps on following all the codes such as imports and includes all new javascript files and builds it. One can see the setup for webpack viewing **webpack.config.js** file. Also for babel **babel.config.js** There is a very less chance that you need to modify these config files. But you should make sure that all the packages are properly installed before this setup can work seamlessly.

Before working with Files . Execute this command to install all required packages
Npm install
One can output the main.js file using this command inside the frontend directory.

Npm run dev

Just be clear that this watches for file changes and continuously runs in the background to generate new main.js file. You don't have to run this everytime you want to run a server. Only one server has to run, that is Django server just serving this exported main.js file through index.html

If everything is final with frontend and you are ready for production , you can simply run **npm run build** to build the final main.js file and you never have to look back to frontend if everything works well.

For the testing, We are using a react testing library along with jest . There is one **jest.config.filem** inside this app folder that will load the setup test javascript file which basically imports the jest-dom/extend-expect package.

There is also one command **npm run doc** to generate react documentation but it doesn't do the job well.
All the includes and required packages, scripts , entry points can be seen inside **package.json**

The folders where you'll spend most of your time for the frontend will be the src folder . src contains one main **index.js** file linked to webpack for compiling codes to one file.

Index.js gets element containing id “app” and renders/puts all the output inside that div element

Inside index.html

```
<div id="app"></div>
```

Inside index.js

```
const appDiv = document.getElementById("app");
render(<App />, appDiv);
```

For the rest , one can view frontend documentation .

There is also file db.sqlite3 where one can do all the operating related to database using SQL commands but for Django every changes can be done using models and making migrations

This is all one needs to know about the backend for now. Thank you

Ishwor Giri & Muhammad Shahzaib Tahir Awan

DATE: 9 March 2021