BACKEND

Sprint # 02 - Team 33

23 March 2021

Ishwor Giri, Hamza Bouhelal

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/

DEPENDENCIES

```
django
djangorestframework
django-cors-headers
djangorestframework_simplejwt
drf-yasg
```

https://www.django-rest-framework.org/
https://pypi.org/project/django-cors-headers/
https://django-rest-framework-simplejwt.readthedocs.io/en/latest/
https://drf-yasg.readthedocs.io/en/stable/ (yet another swagger generator)

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 -r requirements.txt
```

Change Directory to the project

```
cd backend
```

Database Migration

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

Start Server

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.
.

IMPORTANT:
https://www.django-rest-framework.org/ please follow the documentation related to rest_framework .

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). Also make sure to configure settings to whitelist your frontend url

```
CORS_ORIGIN_WHITELIST = 'http://localhost:3000',
```

```
REST_FRAMEWORK = {
   'DEFAULT_PERMISSION_CLASSES': [
       'rest_framework.permissions.IsAuthenticated'
   ],

   'DEFAULT_AUTHENTICATION_CLASSES': [
```

```
        'rest_framework_simplejwt.authentication.JWTAuthentication',
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
}
```

Settings also contain REST_FRAMEWORK settings .By default, We have added IsAuthenticated and only authenticated users can access the backend API.

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

```
path('', include('Home.urls')),
```

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

```
urlpatterns = [
    path('', include('Home.urls')),
    path('admin/', admin.site.urls),
    path('game/', include('Game.urls')),
    path('user/', include('User.urls')),
    path('apiauth/', include('rest_framework.urls')),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
]
```

2. GAME

This will be the backend application that will handle all the requests related to the game such as fetching game data, database models, API views with serializers and Views Permissions .

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

```
urlpatterns = [
path('<int:pk>/', GameDetail.as_view(),name='detailgame'),

path('', GameList.as_view(),name='listgame')

]
```

Will call the function **GameDetail** inside views.py which will handle api view for each given game

Models.py contains all the models related to the game . For now only the basic model is present.

```
class game(models.Model):
   session_length = models.IntegerField()
   distributer_present = models.BooleanField()
   wholesaler_present = models.BooleanField()
   holding_cost = models.FloatField(default=1)
   backlog_cost = models.FloatField(default=1)
   instructor = models.ForeignKey(User,limit_choices_to={'is_instructor': True},
on_delete=models.CASCADE)
   rounds_completed = models.IntegerField(default=0)
   starting_inventory = models.IntegerField(default=0)
```

Instructor is a foreign key representing a user who has the attribute is_instructor as true. This layout needs more change and addition .

Serializers.py

This file contains serializers from rest_framework that are useful for validation and using models output for API views and responses.

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

```
#View to get specific games with game id /game/gameid created by loggedin instructor
class GameDetail(generics.RetrieveUpdateDestroyAPIView,GameUserWritePermission):
       permission_classes=[IsAuthenticated,GameUserWritePermission]
       queryset = game.objects.all()
       serializer_class = gameserializer
```

One should render this as GameDetail.as_view()

Above function must be imported from the library before using

```
from rest_framework import viewsets
from Game.models import game
from Game.serializers import gameserializer
from rest_framework import generics
from rest_framework.permissions import IsAdminUser,IsAuthenticated, SAFE_METHODS,
DjangoModelPermissions, BasePermission
from rest_framework_simplejwt.models import TokenUser
```

3. **USER**

All the basic functionality is the same compared to the game .

Models.py

In this file we have created a new user model which is self explanatory but extends from BaseUSerManager which will allow us to have django user management features. We also added a function to create a super user .

One can create superuser as **python manage.py createsuperuser**

Serializers.py

This is also the same to Game but has some additions for signup validation and security to prevent password being seen from API view.

Urls.py

```
urlpatterns = [
   path('create/', CustomUserCreate.as_view(), name="create_user"),
   path('info/', CustomUserInfo.as_view(), name="user_info"),

   path('logout/blacklist/', BlacklistTokenUpdateView.as_view(),
        name='blacklist')
]
```

Views.py

User views.py contains CreateAPIViews allowing any visitors without authentication to create an account. It also features CustomView to get current logged in user email.

4. **HOME**

Home is a basic app that normally handles backend homepage url .

It has basic url configs pointing to views which returns a simple html view displaying useful links related to the backend.

Urls.py

It also has One view linking to url /swagger/ and /redoc/ which are useful for API documentation.

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

API OVERVIEW:
One can also goto /redoc/
/swagger/ to test and check api overview and the parameters it takes.

Little Information about JWT TOKEN.
This project uses jwt tokens facilitated by
https://django-rest-framework-simplejwt.readthedocs.io/en/latest/

The configuration has already been set inside settings.py file.

```
SIMPLE_JWT = {
   'ACCESS_TOKEN_LIFETIME': timedelta(minutes=1),
   'REFRESH_TOKEN_LIFETIME': timedelta(days=10),
   'ROTATE_REFRESH_TOKENS': True,
   'BLACKLIST_AFTER_ROTATION': True,
   'ALGORITHM': 'HS256',
   'SIGNING_KEY': SECRET_KEY,
   'VERIFYING_KEY': None,
   'AUTH_HEADER_TYPES': ('JWT',),
   'USER_ID_FIELD': 'id',
   'USER_ID_CLAIM': 'user_id',
   'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),
   'TOKEN_TYPE_CLAIM': 'token_type',
}
```

The endpoints related to token generation are as follows.
path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),

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