BAYKAR CASE STUDY

Overview
This project allows users the manage IHAs and rent them at will. Users can easily register and login then list, create, update, delete IHAs. Users can also rent the created IHAs with a given date range and do operations like list, update and delete rent records. Behind the scenes, the project is powered by the Django Rest Framework on the backend and React on the frontend. Project data is securely stored in a PostgreSQL database as well.

This document consists of two main sections. These sections are:
- Backend Services
- Frontend Application

This document outlines the construction and purpose of these key sections:

Backend Service: These serve as the primary logic modules responsible for IHA and renting management. They facilitate the storage of IHAs and their renting, retrieval of stored data, and the ability to delete them

Frontend Application: This component furnishes users with a user-friendly interface for seamless interaction with the system. It also plays a pivotal role in dispatching IHA and renting information to the backend services for processing and storage.

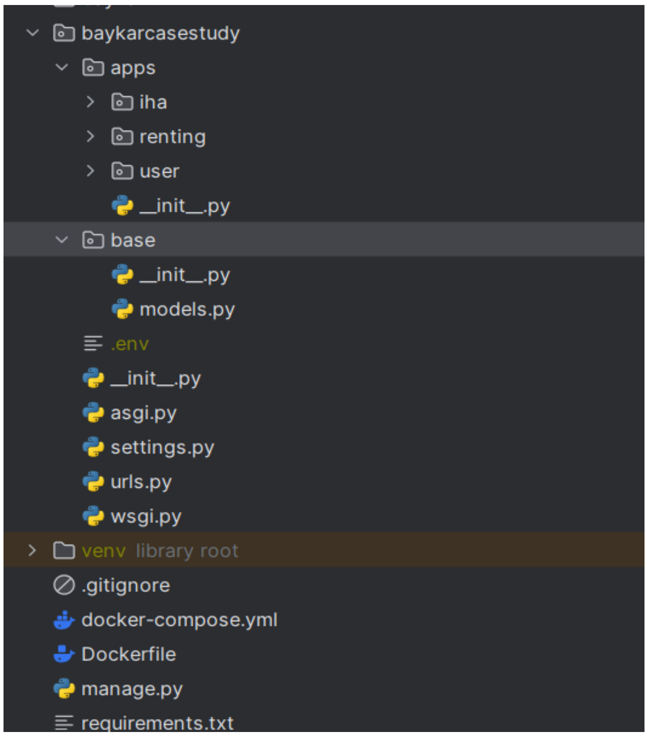
Execution Instructions

1. Execute the **docker compose build** to build the docker container.
2. Execute the **docker compose up -d** to run the docker container.
3. Execute the **docker compose run backend python manage.py makemigrations** to create migrations for tables.
4. Execute the **docker compose run backend python manage.py migrate** to commit migrations to the database.
5. Open **localhost:3000** in your browser to access the user interface.

BACKEND SERVICE

This part serves as the primary logic modules responsible for IHA and renting management. It facilitates the storage of IHAs, retrieval of stored IHAs, and the ability to delete them, rent them etc.

The folder structure of the project can be seen in the Picture 1.1



```
∨ baykarcasestudy
  ∨ apps
    > iha
    > renting
    > user
      __init__.py
  ∨ base
      __init__.py
      models.py
    .env
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
> venv  library root
  .gitignore
  docker-compose.yml
  Dockerfile
  manage.py
  requirements.txt
```

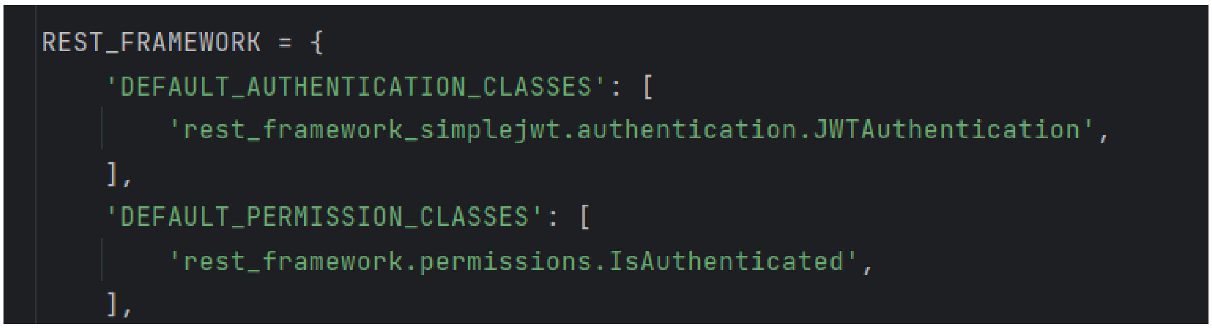
Picture 1.1

This project has a classic DRF folder structure with the apps inside of the project and separated by project domains. It also has an app called “base” which currently holds Mixin classes for other tables to have common fields such as created_at and updated_at fields.

Apps

User

This app is responsible for registering users and enabling users to login using JWT as an authentication system.
In order to integrate JWT, the project uses **rest_framework_simplejwt** library to create and validate both access and refresh tokens. It is also set as the default authentication class for the project in the settings.py as can be seen in the Picture 1.2

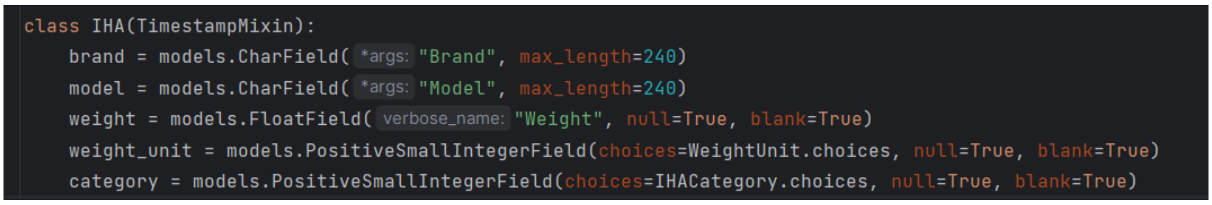


```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
```

Picture 1.2

IHA

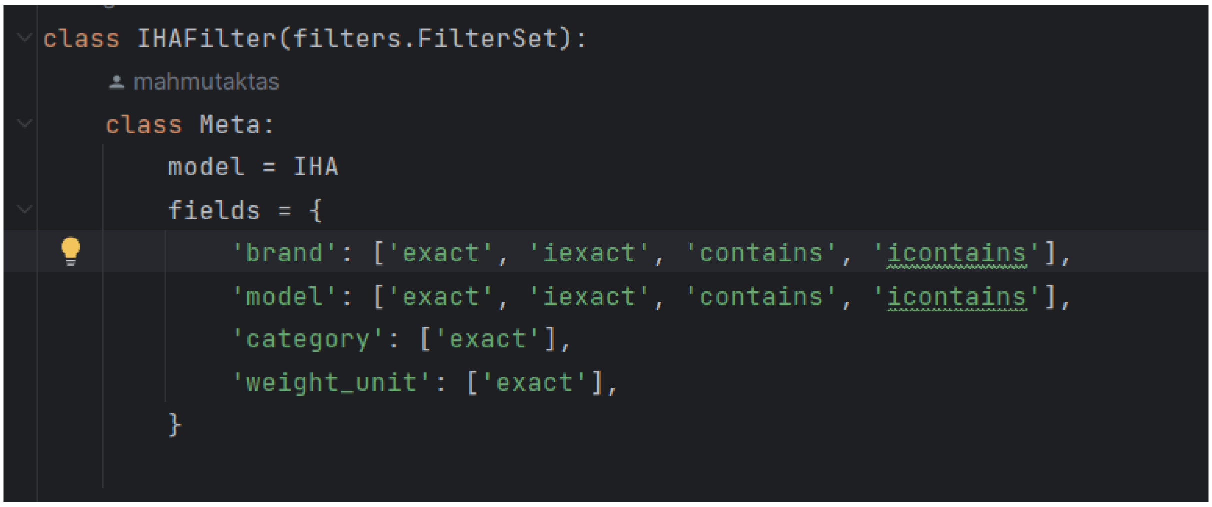
This app is responsible for IHA management. It enables users to create new IHAs, and other operations such as listing, updating and deleting. IHA fields can be seen in the Picture 1.3



```
class IHA(TimestampMixin):
    brand = models.CharField( *args: "Brand", max_length=240)
    model = models.CharField( *args: "Model", max_length=240)
    weight = models.FloatField( verbose_name: "Weight", null=True, blank=True)
    weight_unit = models.PositiveSmallIntegerField(choices=WeightUnit.choices, null=True, blank=True)
    category = models.PositiveSmallIntegerField(choices=IHACategory.choices, null=True, blank=True)
```

Picture 1.3

This app also enables IHAs to be filterable, sortable and create pagination structures when fetching them. Filters of this app can be seen in the Picture 1.4

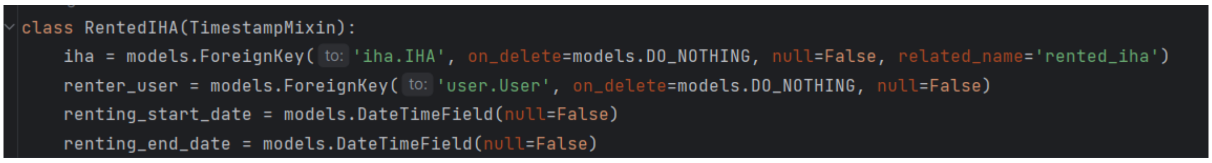


```
class IHAFilter(filters.FilterSet):
    mahmutaktas
    class Meta:
        model = IHA
        fields = {
            'brand': ['exact', 'iexact', 'contains', 'icontains'],
            'model': ['exact', 'iexact', 'contains', 'icontains'],
            'category': ['exact'],
            'weight_unit': ['exact'],
        }
```

Picture 1.4

Renting

This app is responsible for renting logic of the project. It enables users to rent IHAs and update or delete them at their will. RentingIHA model can be seen in the Picture 1.5



```
class RentedIHA(TimestampMixin):
    iha = models.ForeignKey( to: 'iha.IHA', on_delete=models.DO_NOTHING, null=False, related_name='rented_iha')
    renter_user = models.ForeignKey( to: 'user.User', on_delete=models.DO_NOTHING, null=False)
    renting_start_date = models.DateTimeField(null=False)
    renting_end_date = models.DateTimeField(null=False)
```

Picture 1.5

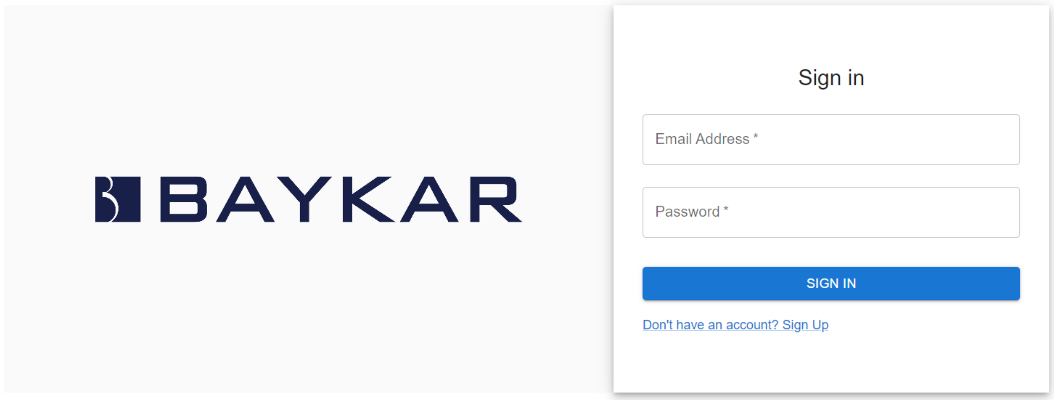
As it can be seen in the picture, the model has a relationship with both user and IHA models. It also has start date and end date fields to set a renting date range.

FRONTEND APPLICATION

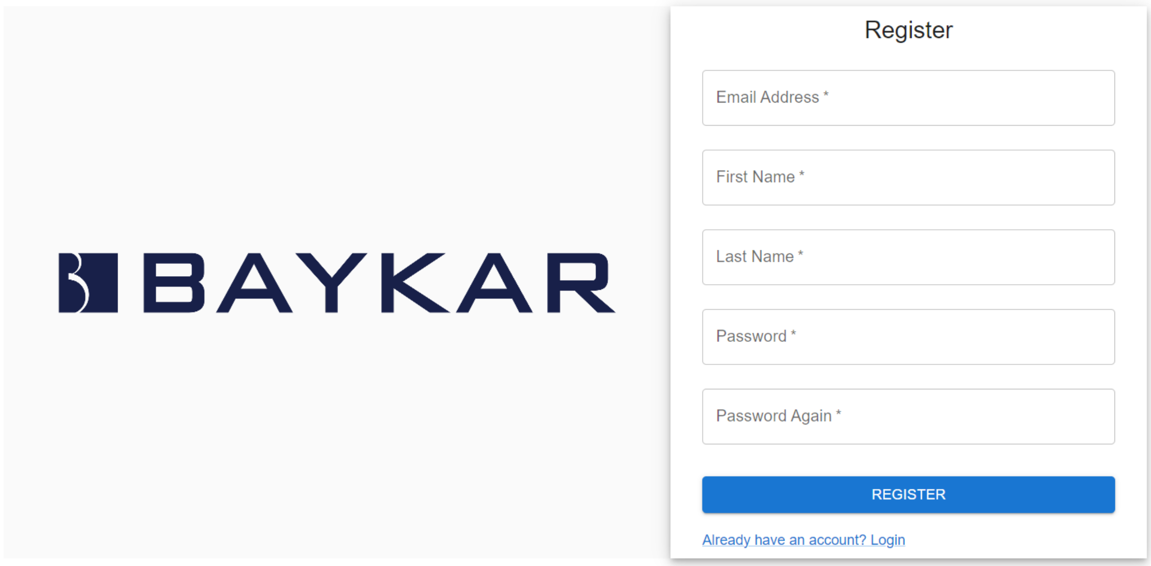
This part serves as the user interface of the project. It enables users to sign up, sign in, manage IHAs and manage rentings. Frontend application can be separated into 3 parts. These are:

Authentication

In this part users can either sign up or sign in as it can be seen in the Picture 2.1 and Picture 2.2

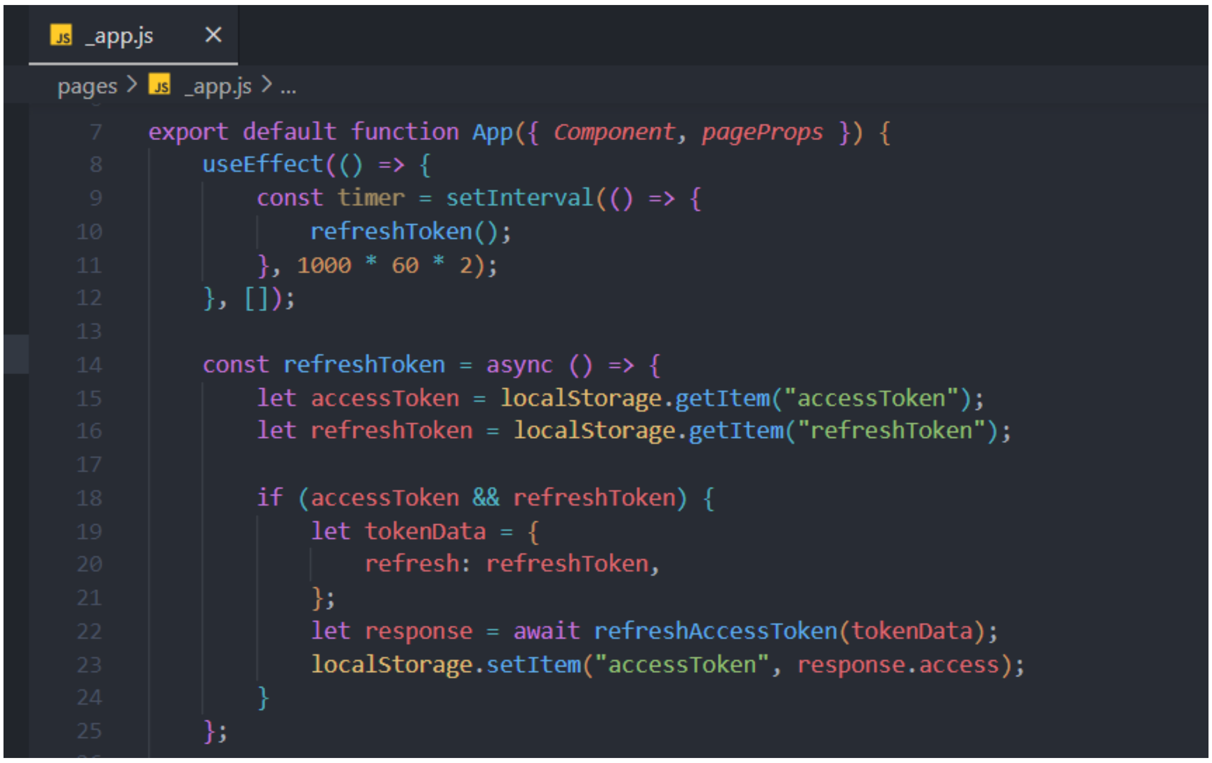


Picture 2.1



Picture 2.2

After a user signs in, the application stores its access token and refresh token. Due to short lifetime nature of the access token, application gets a new access token for every minute as can be seen in the Picture 2.3



_app.js

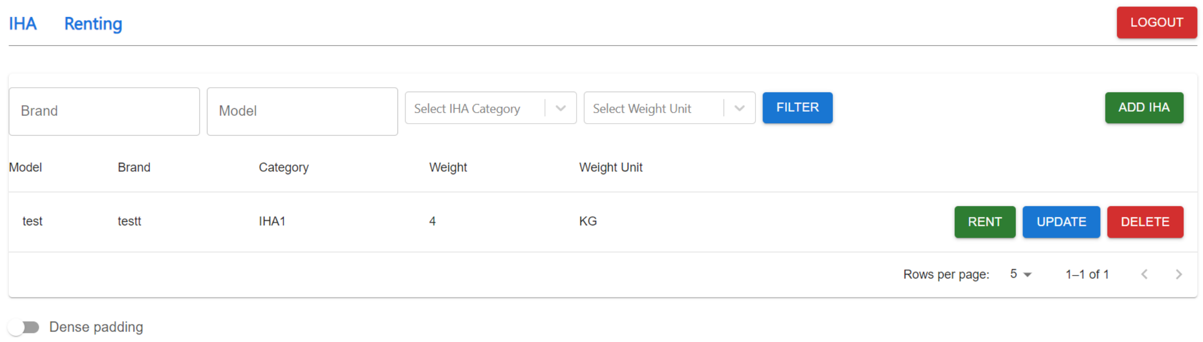
pages > _app.js > ...

```
export default function App({ Component, pageProps }) {
    useEffect(() => {
        const timer = setInterval(() => {
            refreshToken();
        }, 1000 * 60 * 2);
    }, []);

    const refreshToken = async () => {
        let accessToken = localStorage.getItem("accessToken");
        let refreshToken = localStorage.getItem("refreshToken");

        if (accessToken && refreshToken) {
            let tokenData = {
                refresh: refreshToken,
            };
            let response = await refreshAccessToken(tokenData);
            localStorage.setItem("accessToken", response.access);
        }
    };
```

Picture 2.3

IHA

This part provides a table and multiple modals to manage IHAs with the integration of the backend. The IHA Datatable can be seen in Picture 2.4.



Model	Brand	Category	Weight	Weight Unit	
test	testt	IHA1	4	KG	RENT UPDATE DELETE

Rows per page: 5 1–1 of 1

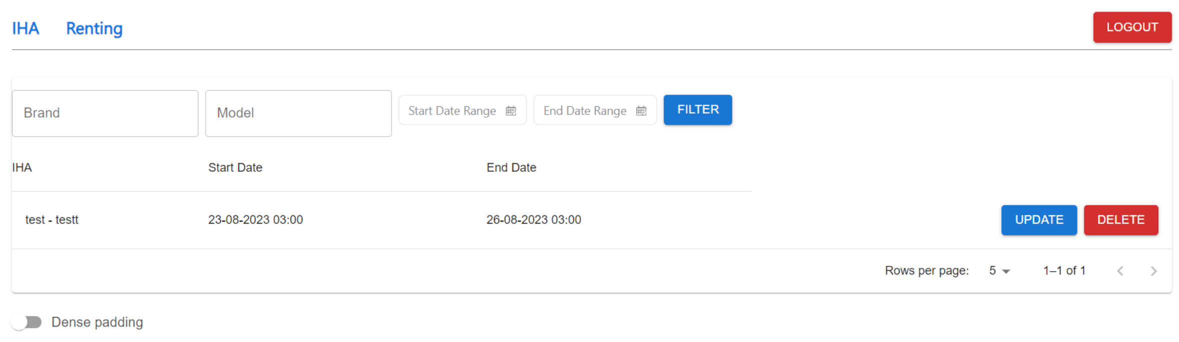
Dense padding

Picture 2.4

Users can filter IHAs, rent them, update them, delete them or create new IHAs with this screen.

Renting

This part provides a table and multiple modals to manage renting data with the integration of the backend. The Renting Datatable can be seen in Picture 2.5.



IHA	Start Date	End Date
test - testt	23-08-2023 03:00	26-08-2023 03:00

Picture 2.5

Users can filter renting records, update them or delete them with this screen.