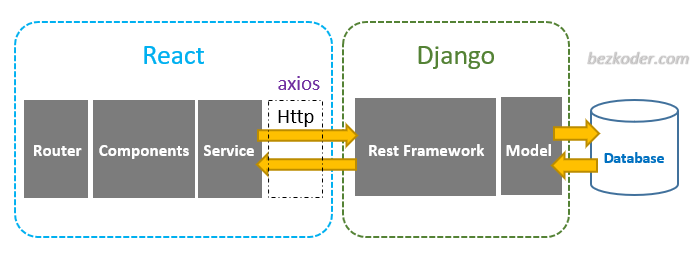
**AMS (Appointment Management System) Architecture**

The application is created using Django(Backend) & React(Frontend). The connection architecture of the project can be explained by the following diagram.

****

***Application Architecture***:

Front-End



Delete Appointment

Update Appointment

Create Appointment

**Patient**



API Call

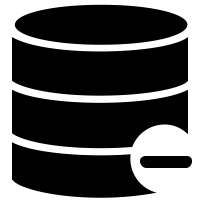
API Call

API Call

Appointment Details

1. Problem
2. Appointment date
3. Details
4. Slug

Backend





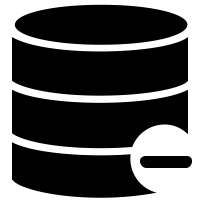
Front-End

Remove Appointment

Update Appointment

View Appointment

**Doctor**



Backend

Appointment Details

1. Problem
2. Appointment date
3. Details
4. Slug

***Description***:

* First user needs to register by entering the details by default every user will be registered as a patient only and will have the following rights-

1. View the created appointment.
2. Create a new appointment.
3. Delete the existing appointment.
4. Update the appointment

* Only **Admin has the right to register users as a** **Doctor.** Also admin can create, view, delete & update any appointment.
* The doctor’s rights include -

1. View daily appointment
2. Update an appointment
3. Remove the appointment (if he is done with the patient)

***Working***:

* **Authentication**:

Authentication is managed using **JWT token management** where every user whoever logs in would be associated with **access & refresh token** once the user logout the token will be send to blacklist so that the session could not be resumed.

* **End-Points and Response**

The application is made using **Django** but to create the end-points **Django-Rest-Framework** is used, since it has predefined **generic** views and **serializers** it’s easy to create end-points using this and it also comes with **coreapi** docs by which we can debug the api faster.

Response generated from the end-points are in the form of **JSON** so that we can consume the response in the front-end easily and use it in web or mobile application accordingly.

To convert the response in **JSON format Serializers** are used which are available in Django-Rest-Framework.

* **Permissions**

The main part of the project is to handle different permissions/rights. The Django-Rest-Framework comes with the **permission class** which can be utilized to make the task easy. The permission can be given in **Project Level, Views Level** or **Object Level** according to our use.

We can also create **custom permission** and allow the user to have **specific rights**

* **Models**

Models are created in the Django framework since it’s easy to manipulate the tables using Django. The ORM helps us to create a class based structure of the data which is converted into SQL internally and saved in the database.

* **Database**

The database used for this project is built-in SQLite3 which comes with Django but we can use any database, it’s easy to integrate with the Django application.

* **Views**

Views are the classes which defines the structure of the data that will be shown in the web application it communicates between the model and template and provide the necessary data as objects. **Generic views** are used in the project which comes with Django-Rest-Framework.

* **Front-End**

Front end is designed using **React Js**. The user will interact with the front end and React will consume the responses which comes from Django in JSON format and display the user.

Since we can design and customize different components of the page using React therefore it’s preferable to use React Js in the front-end.

* **Custom User**

Django comes with default user Model but that was not sufficient for the project therefore BaseUser class was inherited and modified according to the requirements of the project.

Due to the time constraints, the project may contain some bugs since there was less time to perform the testing. Hope, you like the project and give me the opportunity to move forward with the organization.