NIC PREDICTIVE ANALYSIS

GUIDED BY

- **Debajit Ghosh Sir**
- **Subhasish Bhowmik Sir**

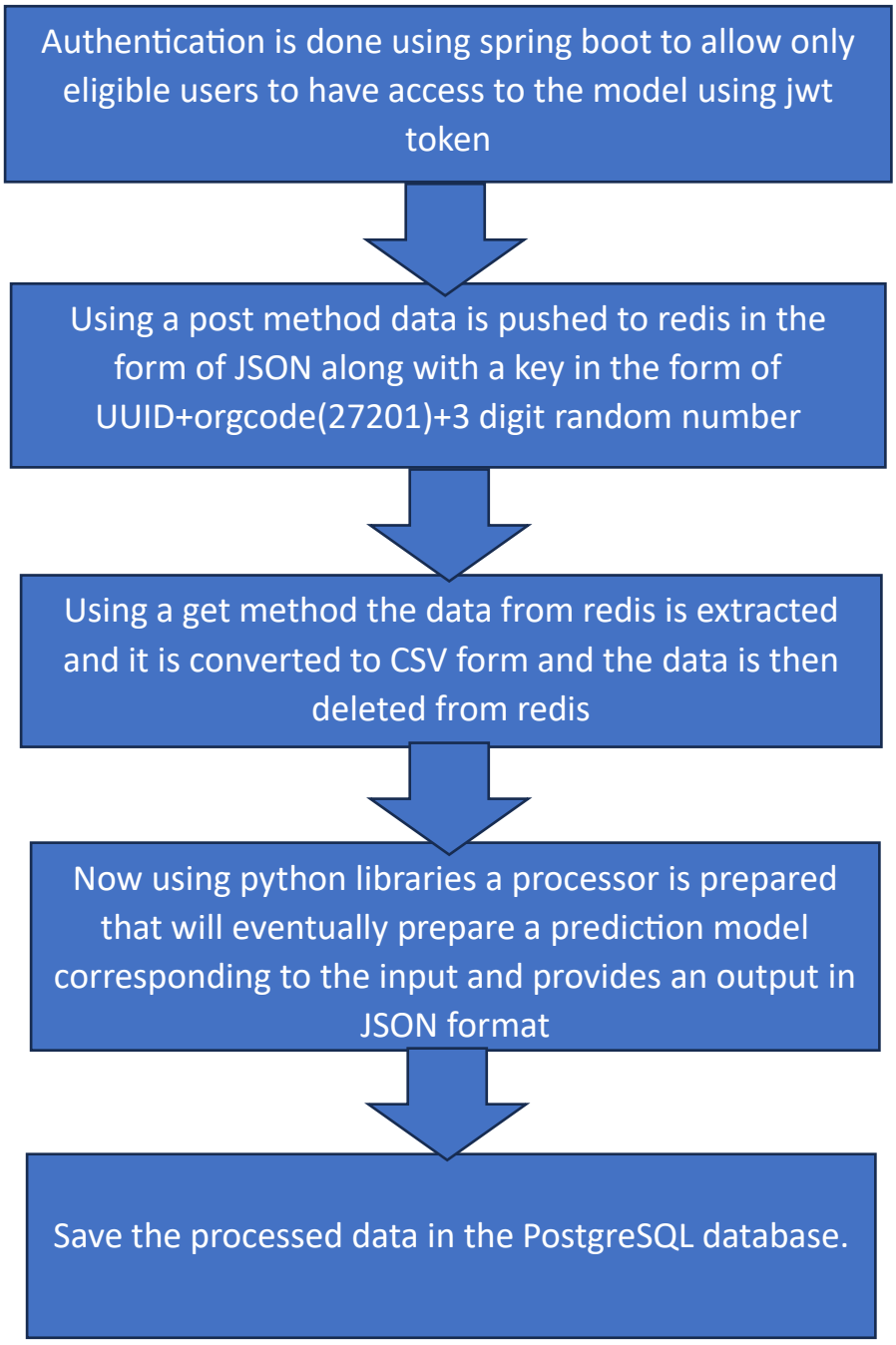
TEAM MEMBERS

- **Abhradip Saha** – 21UCS053 (NIT Agartala)
- **Abhishek Das** – 21UCS190 (NIT Agartala)
- **Abhishek Debnath** – 21UEC007 (NIT Agartala)
- **Soumyadeep Saha** – 21UCS127 (NIT Agartala)
- **Diya Banik** – 21UCS176 (NIT Agartala)
- **Ratnadeep Das** – 21UEC058 (NIT Agartala)
- **Sohel Tripura** – 21UCS152 (NIT Agartala)
- **Umesh Tripura** – 21UEC021 (NIT Agartala)
- **Aiswrang Debbarma** – 21UCS143 (NIT Agartala)
- **Shankar Kumar** – 21UCS065 (NIT Agartala)
- **Sreejit Paul** – 202100443 (SMIT)

Objective:-

The objective of this project is to predict the behaviour of data on the basis of a set of data taken as input which is further processed and plotted. By going through the plotted data we can predict the behaviour of similar form of data in future.

Flowchart :-



Project Summary:-

This project is divided into **five parts**:-

1) Authentication
2) Data in the form of JSON being pushed to redis
3) Data being obtained from redis and send to the processor in the form of JSON followed by deletion of data from redis.
4) Processing of data.
5) Save the processed data in PostgreSQL.

Techstack :-

To prepare the API microservices are prepared where **Spring Boot** is used and databases used are **Redis** and **PostgreSQL**.

To prepare the prediction model **ARIMA algorithm** is used where **Flask** has been used as the main techstack along with few other **Python libraries**.

Contributions :-

1) **Authentication:-**
   - Soumyadeep Saha

2) **Data in the form of JSON being pushed to redis:-**
   - Abhradip Saha

3) **Data being obtained from redis and send to the processor in the form of JSON followed by deletion of data from redis :-**
   - Abhradip Saha

4) **Processing of data:-**
   - Abhishek Debnath
   - Ratnadeep Das
   - Abhradip Saha
   - Soumyadeep Saha
   - Umesh Tripura
   - Sreejit Paul
   - Sohel Tripura
   - Aiswrang Debbarma

5) **Save the processed data in PostgreSQL**:-
   - Abhishek Das
   - Diya Banik
   - Sohel Tripura
   - Ratnadeep Das

Format of JSON data to be pushed to redis:-

```
[
   {
        "Category":"ABC",
       "Sub Category":"A",
       "Expected Period in month":12,
        "Out_data_pattern":3,
        "demand_date":"2024-05-27",
       "demand_Qty":15,
       "analysis year":2024
   }
]
```

Microservices:-

Following are the microservices used in this project:-

a) **Authentication:-**
   This is for the purpose of authenticating the project using jwt token. It contains the following API endpoints :
   - **(/auth/create-user) :** A post method to create a new user.
   - **(/auth/login) :** A post method to get access to the project resources.

b) **Push to redis :-**
   It allows to push data to redis in the form of JSON. It contains the following API endpoint :
   - **(/auth/nic) :** A post method to push JSON data to redis.

c) **Get data from redis and delete from redis :-**
   It fetches the data in the form of json and immediately deletes it from the redis database. It contains the following API endpoint :

- **(/auth/fetch-nic) :** A get method to extract JSON data from redis so that it can be processed and the data gets deleted from redis immediately.

Integration of ARIMA Forecasting :-

This application integrates ARIMA (AutoRegressive Integrated Moving Average) forecasting to predict future demand quantities based on historical data. The process involves several key steps:

1. Data Loading and Preprocessing:

a) Data is loaded from a API endpoint using Pandas.

b) Dates in the Demand_date column are parsed into datetime format ('%d-%m-%Y'), ensuring consistency for time-series analysis.

c) The DataFrame is indexed by Demand_date and sorted chronologically to maintain temporal order.

2. ARIMA Model Building:

a) An ARIMA model is constructed using the statsmodels library, tailored to analyze and forecast time-series data.

b) The model is trained on historical demand quantities (Demand_qty column), capturing patterns and trends over time.

3. Forecasting:

a) Using the fitted ARIMA model, future demand quantities are forecasted for a predefined horizon (default: 12 periods).

b) These forecasts provide insights into expected demand trends, aiding in proactive decision-making and resource allocation.

4. Visualization:

a) The forecasted values are visually represented using matplotlib and seaborn libraries, enhancing interpretability through graphical analysis.

b) Plots display historical demand trends alongside forecasted quantities, facilitating comparisons and trend identification.

5. Database Integration:

a) Forecasted demand quantities are persisted in a PostgreSQL database (forecast_results table) using psycopg2.

b) This integration ensures data persistence and accessibility for future analysis and reporting needs.

6. **Flask API Deployment:**
   a) A Flask API endpoint (/api/forecast) is configured to deliver forecasted quantities in JSON format, accessible via HTTP requests.
   b) This API enhances the application's usability and facilitates seamless integration with other systems and applications. This approach combines advanced forecasting techniques with robust data management and visualization capabilities, empowering organizations to make informed decisions based on predictive analytics.
   c) The API endpoint (/api/forecast) also saves the processed data to the PostgreSQL database.

Database Description:-

A database is created in PostgreSQL and it is named as **Nic_Predictive_Analysis.** Inside this following schemas are created : –

a) **user_table :**
   This contains the details of authenticated users. It contains the following fields :-
   - user_id
   - name
   - email
   - password'
   - about

b) **forecast_results :**
   This contains the processed data. The fields in it are :-
   - id
   - forecast_date
   - forecast_qty

Conclusion:-

On execution of this project we successfully obtained the input data and pushed it into redis on port 6380 and executed the Spring Boot microservice on port 8080 followed by processing of the data using flask on port 5000 which is finally saved to PostgreSQL.