Asset - x (A mini bank)

In today's fast-paced digital world, managing finances securely and efficiently is more important than ever. At Asset-x, we provide a reliable and secure banking system designed to meet modern financial needs. Whether you're opening an account, making transactions, or managing funds, our platform ensures seamless banking operations with advanced security and scalability.

Asset-x offers a robust backend infrastructure that powers secure account management, real-time transactions, and user authentication. We prioritize data integrity, encryption, and compliance, ensuring that every financial interaction is protected from unauthorized access. With a user-friendly API and seamless integrations, Bank simplifies banking for individuals and businesses alike.

Each user is provided with a secure and personalized banking experience. From tracking transactions to scheduling payments, our system offers intuitive features that enhance financial control and transparency. Whether you're transferring funds, monitoring account activity, or managing multiple accounts, YourBank ensures efficiency and security at every step.

Beyond standard banking operations, Asset-x is built for growth and adaptability. Our system is designed to scale, supporting high transaction volumes while maintaining performance and reliability. With continuous improvements and advanced security measures, Bank is the ideal platform for those seeking a modern, digital-first banking experience.

At Asset-x, we are committed to building a financial ecosystem that is not only secure and efficient but also accessible and adaptable to evolving financial landscapes. Whether you're an individual looking for a safe place to manage your finances or a business requiring a scalable banking solution, Asset-x provides a trusted and seamless banking experience.

Scenario Based Case Study:

Background: Sam is a Frontend developer freelancer?

Problem: Sam got a contract of a full stack bank project?

Solution: Sam got a git repo of our bank api and he integrated with his frontend project and successfully.

Usage:

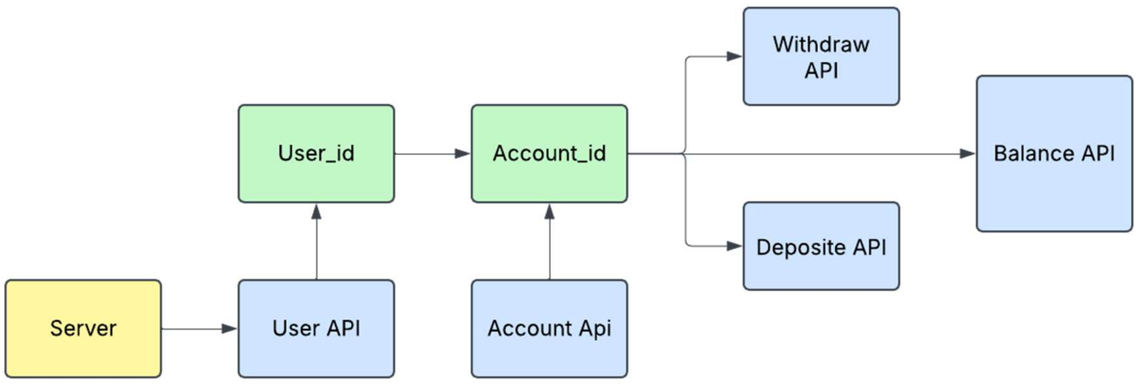
o The user will initially signup and create a account.

o Based on this requirement he will check the balance or he will deposit or withdraw   the Money.

Outcome: With the help of Bank, Sam will seamlessly withdraw and deposit his money.

Technical Architecture:

In this technical architecture,Bank consist of API's:



· User API : User Api will initially register the user and store's his credentials in the database.

· Account API : Account API will create a account for the user on this user_id and also generate the account_id.

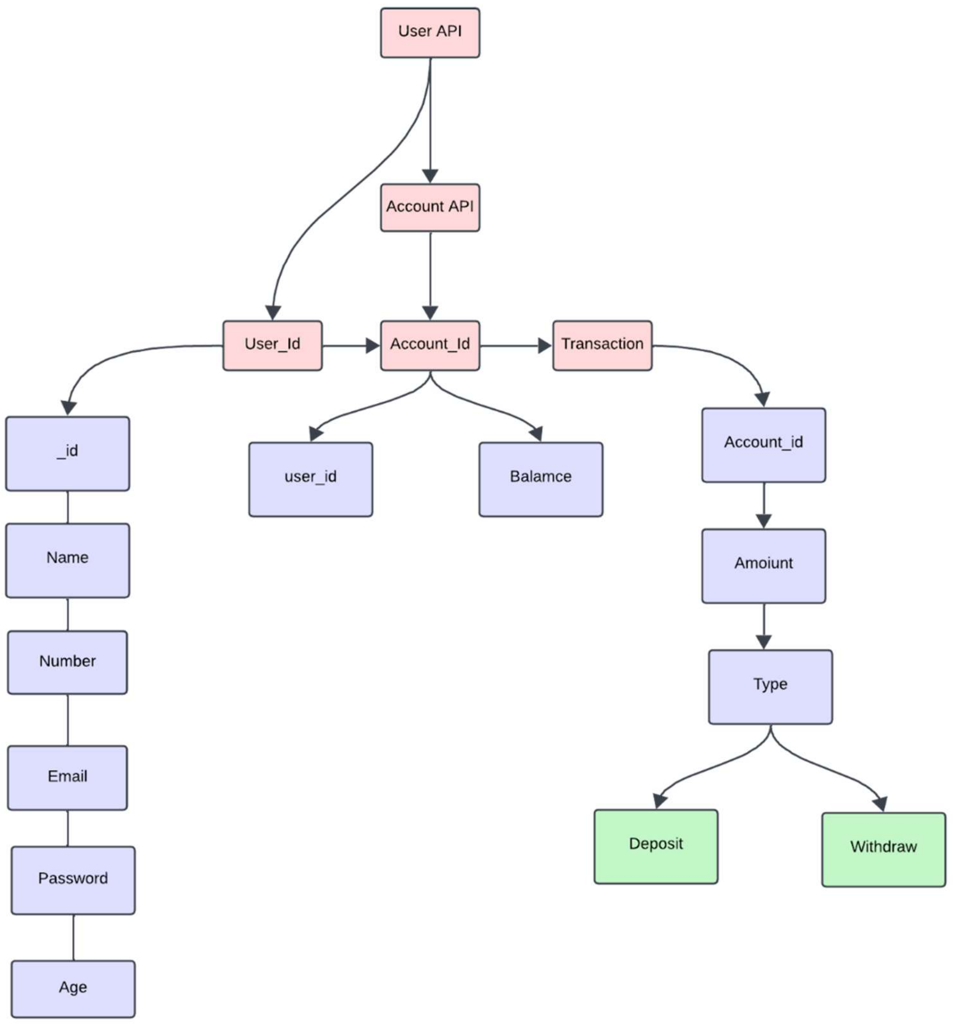
· Balance API: This API will Allow the user to get his account balance.

· Deposit API: This will initially deposit the Money in the user Account based on the user_id .

· Withdraw API : Withdraw API help's in withdraw of money from the specified account.

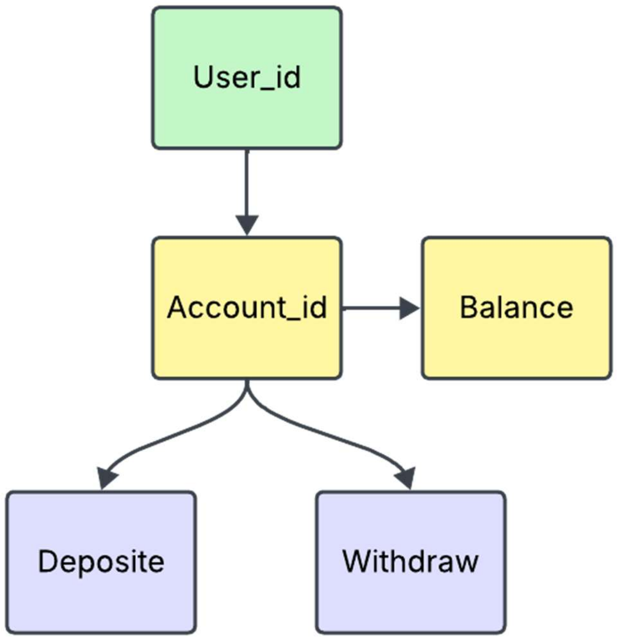
ER-Diagram

ER-Diagram:



ER-Diagram:

User Flow:



Key Features:

1. **User API :** The User will signup with this api.

2. **Account API :** If the user has registered ,the account api will create the account on the user_id .

3. Deposite API : This api will help the user to deposite the money in his account with the help of account_id.

4. Withdraw API : This api will help the user to withdraw the money form his account with the help of account_id.

5. Balance Api : This api will help in fetching the balance from the account.

PRE REQUISITES

PRE REQUISITES:

To develop a Bank js,Node.js,Express.js and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: https://nodejs.org/en/download/

- Installation :https://nodejs.org/en/download/package-manager/

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: https://www.mongodb.com/try/download/community

• Installation instructions :https://docs.mongodb.com/manual/installation/

Mongoose: Mongoose is mongodb library which is basically an ODM (Object Data Model). It help in establish a connection between the server and a client.

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

• Installation: Open your command prompt or terminal and run the following

command: **npm install express**

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from https://code.visualstudio.com/download

• Sublime Text: Download from

https://www.sublimetext.com/download

• WebStorm: Download from https://www.jetbrains.com/webstorm/download

Roles and Responsibility

Roles and Responsibility :

User:

· **Account:** The user will create his account based on the user_id.

· **Transaction =>**

- **Deposit :** Deposite the amount to the specific account_id.
- **Withdraw :** Withdrawal of amount based on the specific account_id

· **Balance :** User will fetch the balance with the specific account_id.

Project Flow

Let’s start with the project development with the help of the given activities.

Project Setup and Configuration

PROJECT FLOW:-

Before starting to work on this project, let’s see the demo.

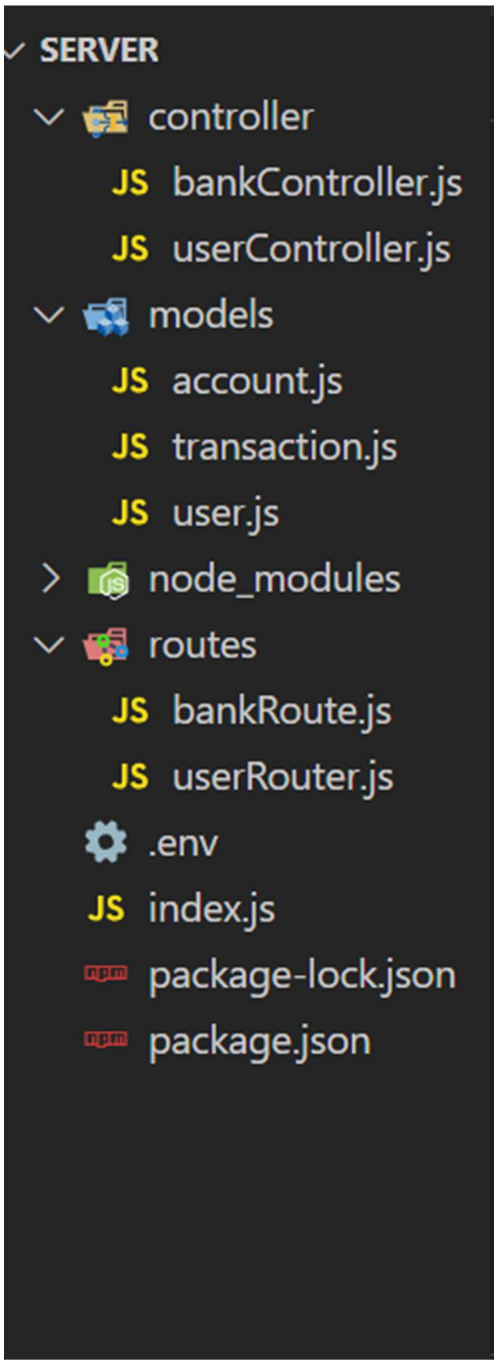
Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

· Node.js.

· MongoDB.

2. Create project folders and files:



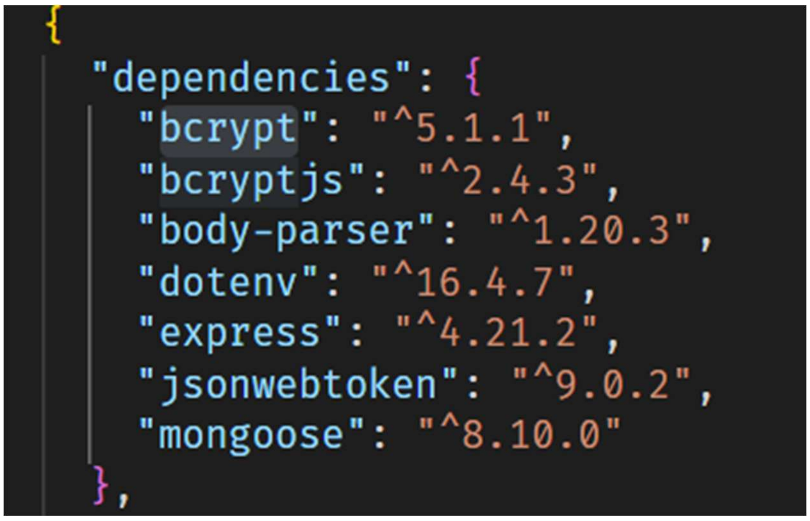
```
SERVER
  controller
    bankController.js
    userController.js
  models
    account.js
    transaction.js
    user.js
  node_modules
  routes
    bankRoute.js
    userRouter.js
  .env
  index.js
  package-lock.json
  package.json
```

- The Folder structure follows the MVC (Model view Controller).
- **Controller :** In the controller we will Create the API.
- **Routes :** In routes We will write the end points of the API's.

- **Model**: The Schema will be designed to take the inputs from the user and to store it in the database.

3. Backend npm Packages

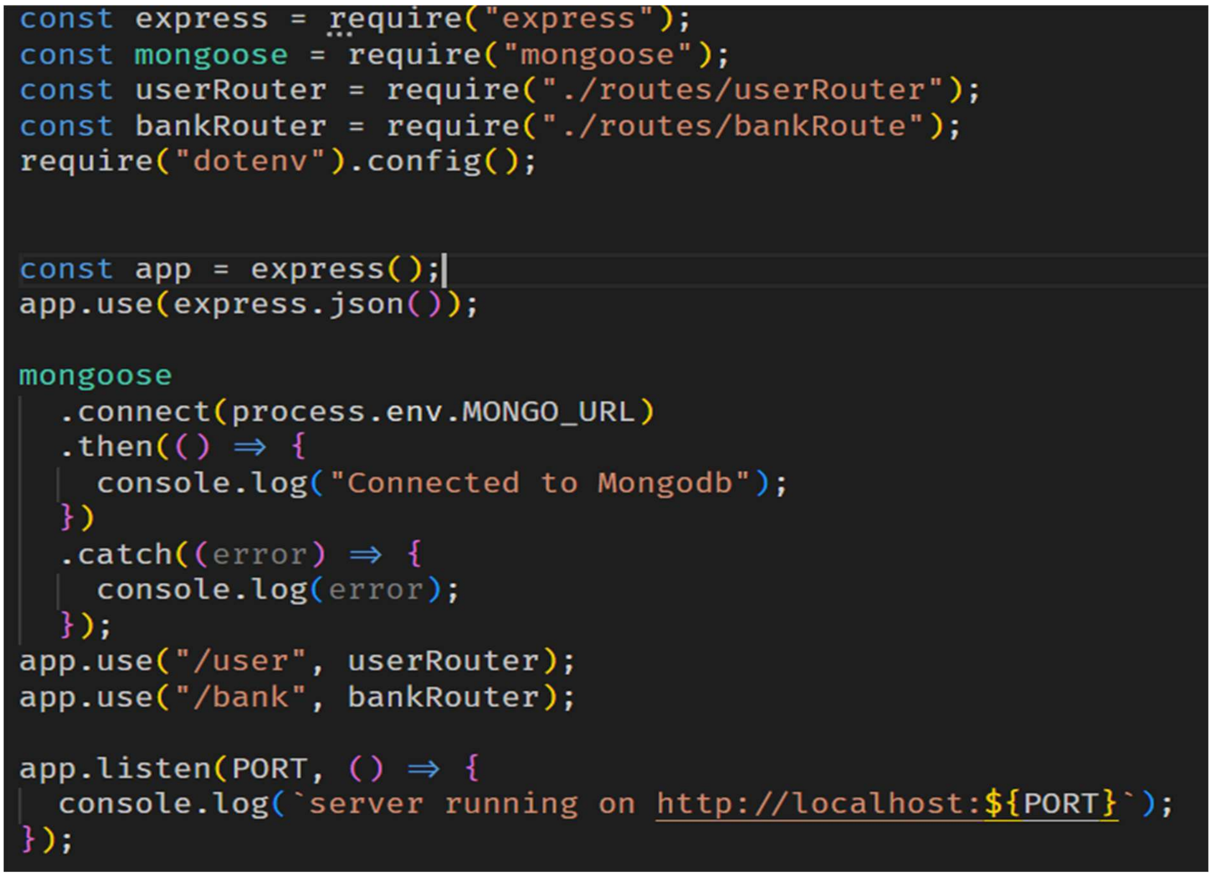
- Download the above dependencies by using the command “npm i <dependencies>“



```
{
  "dependencies": {
    "bcrypt": "^5.1.1",
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.3",
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.10.0"
  },
```

Backend Development

- **Milestone 2: Backend Development:**



```
const express = require("express");
const mongoose = require("mongoose");
const userRouter = require("./routes/userRouter");
const bankRouter = require("./routes/bankRoute");
require("dotenv").config();


const app = express();
app.use(express.json());

mongoose
  .connect(process.env.MONGO_URL)
  .then(() => {
    console.log("Connected to Mongodb");
  })
  .catch((error) => {
    console.log(error);
  });
app.use("/user", userRouter);
app.use("/bank", bankRouter);

app.listen(PORT, () => {
  console.log(`server running on http://localhost:${PORT}`);
});
```

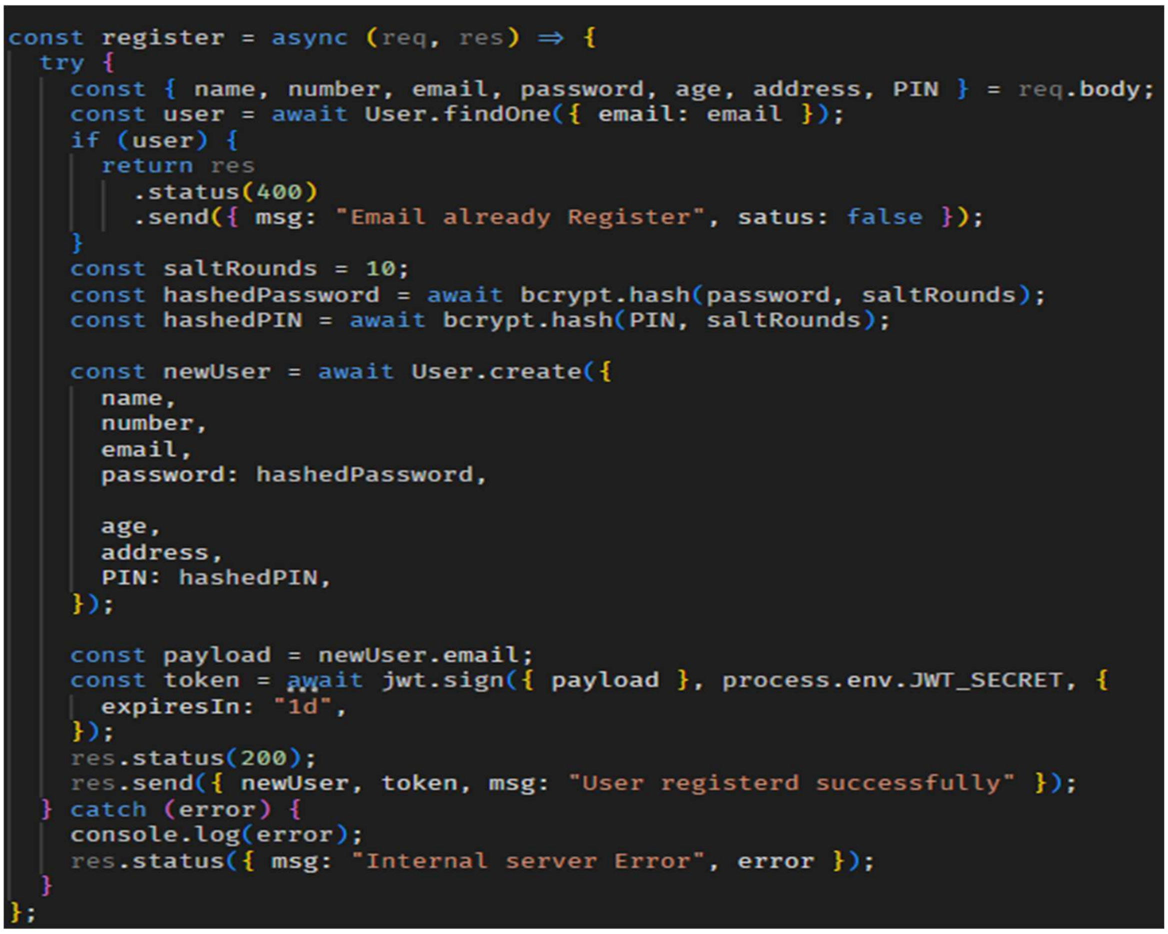
Setup express server :

- Create index.js file in the server (backend folder).

- Install the required dependencies, and require it in the index.js file.
- Create the instance of express.
- Give the mongodb connection, require it from .env file.
- And use the require routes.
- Listen the Server using app.listen.

- Create a dotenv file and initialize the PORT number add the respective non - sharable content.

User Authentication



```
const register = async (req, res) => {
  try {
    const { name, number, email, password, age, address, PIN } = req.body;
    const user = await User.findOne({ email: email });
    if (user) {
      return res
        .status(400)
        .send({ msg: "Email already Register", satus: false });
    }
    const saltRounds = 10;
    const hashedPassword = await bcrypt.hash(password, saltRounds);
    const hashedPIN = await bcrypt.hash(PIN, saltRounds);

    const newUser = await User.create({
      name,
      number,
      email,
      password: hashedPassword,

      age,
      address,
      PIN: hashedPIN,
    });

    const payload = newUser.email;
    const token = await jwt.sign({ payload }, process.env.JWT_SECRET, {
      expiresIn: "1d",
    });
    res.status(200);
    res.send({ newUser, token, msg: "User registerd successfully" });
  } catch (error) {
    console.log(error);
    res.status({ msg: "Internal server Error", error });
  }
};
```

:

- Require the respective dependencies.

- Request the input fields from the body.
- Check if the email is already registered
- Hash the password using hash password.
- Create the new user with the create method.
- Use the Token and give the expiresIn.

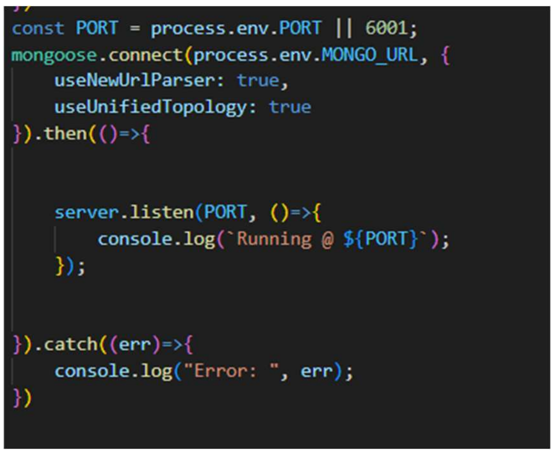
Database

1. Configure MongoDB:

- Install Mongoose.
- Create database connection.
- Create Schemas & Models.

2. Connect database to backend:

Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.

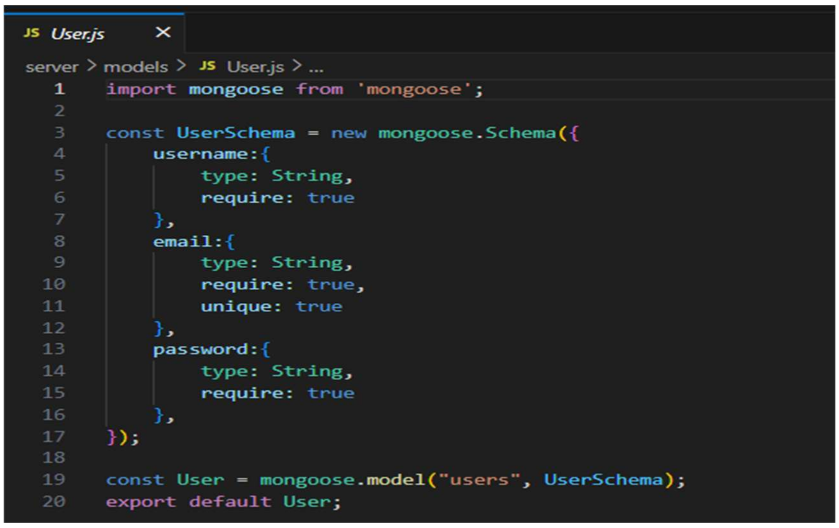


```
const PORT = process.env.PORT || 6001;
mongoose.connect(process.env.MONGO_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(()=>{


    server.listen(PORT, ()=>{
        console.log(`Running @ ${PORT}`);
    });


}).catch((err)=>{
    console.log("Error: ", err);
})
```

3. Configure Schema:

Firstly, configure the Schemas for MongoDB database, to store the data in such pattern. Use the data from the ER diagrams to create the schemas.

The schemas are looks like for the Application.


JS User.js ✕

server > models > JS User.js > ...

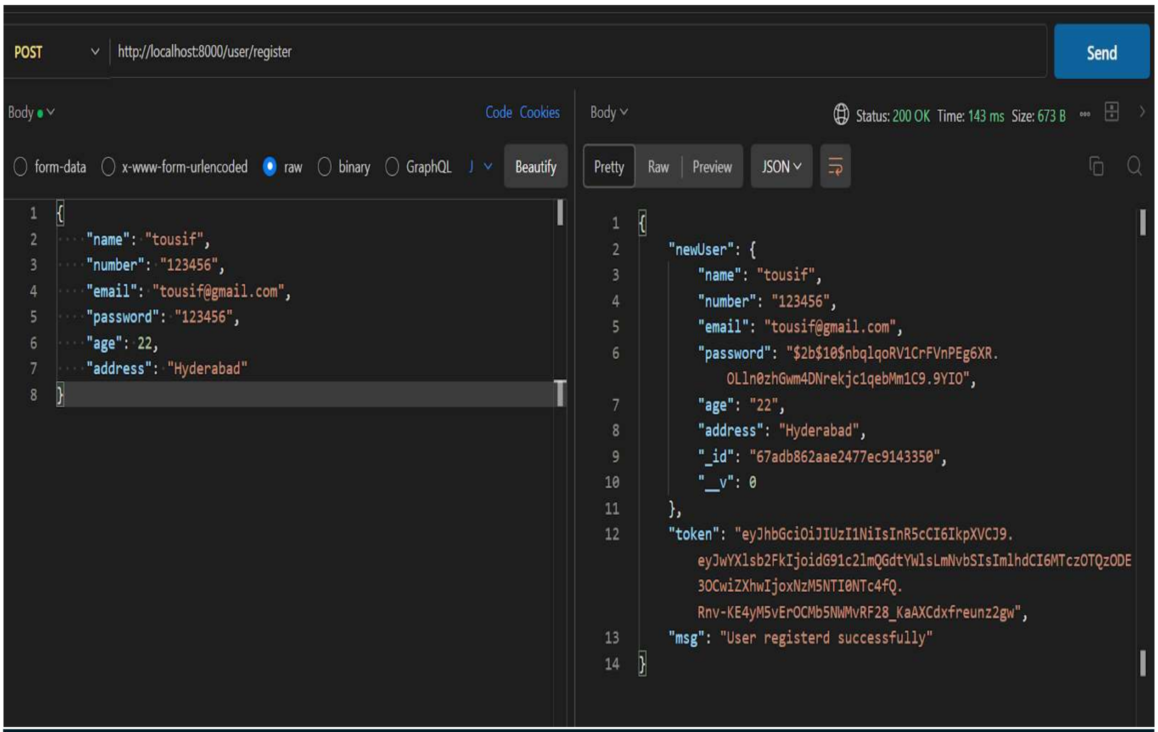
```
import mongoose from 'mongoose';

const UserSchema = new mongoose.Schema({
    username:{
        type: String,
        require: true
    },
    email:{
        type: String,
        require: true,
        unique: true
    },
    password:{
        type: String,
        require: true
    },
});

const User = mongoose.model("users", UserSchema);
export default User;
```

Project Implementation

Milestone 4: API Testing :

- Add Postman Extension in vs Code and Signup, Enter the api endpoint and select the POST method .
- Select the respective method to test the api .
- Select the Body For the Post method in and the select raw give the input in JSON format.

User Creation :

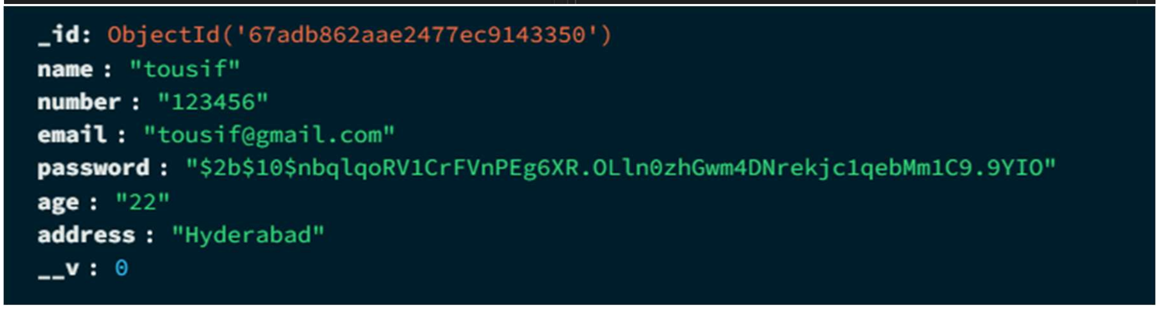


POST http://localhost:8000/user/register

```
{
····"name": "tousif",
····"number": "123456",
····"email": "tousif@gmail.com",
····"password": "123456",
····"age": 22,
····"address": "Hyderabad"
}
```

Status: 200 OK Time: 143 ms Size: 673 B

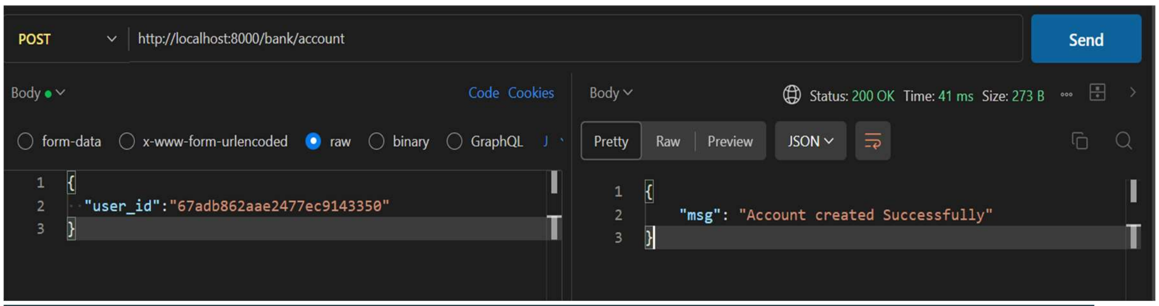
```
{
    "newUser": {
        "name": "tousif",
        "number": "123456",
        "email": "tousif@gmail.com",
        "password": "$2b$10$nbqlqoRV1CrFVnPEg6XR.
            OLln0zhGwm4DNrekjc1qebMm1C9.9YIO",
        "age": "22",
        "address": "Hyderabad",
        "_id": "67adb862aae2477ec9143350",
        "__v": 0
    },
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJwYXlsb2FkIjoidG91c2lmQGdtYWlsLmNvbSIsImlhdCI6MTczOTQzODE
        3OCwiZXhwIjoxNzM5NTI0NTc4fQ.
        Rnv-KE4yM5vErOCMb5NWMvRF28_KaAXCdxfreunz2gw",
    "msg": "User registerd successfully"
}
```



```
_id: ObjectId('67adb862aae2477ec9143350')
name : "tousif"
number : "123456"
email : "tousif@gmail.com"
password : "$2b$10$nbqlqoRV1CrFVnPEg6XR.OLln0zhGwm4DNrekjc1qebMm1C9.9YIO"
age : "22"
address : "Hyderabad"
__v : 0
```

- **METHOD : POST**
- **URL : http://localhost/user/register**
- **This API is for user creation.**

Account Creation :

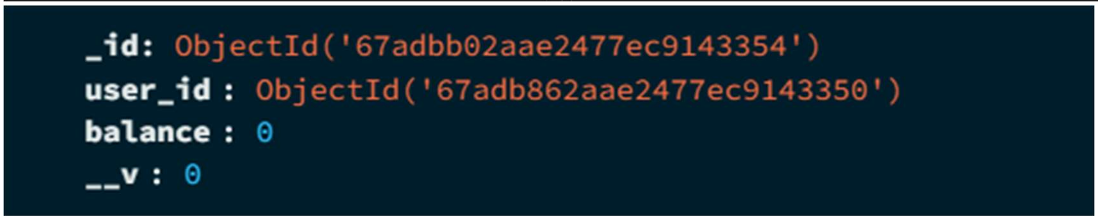


```
POST http://localhost:8000/bank/account
```

```
{
  "user_id":"67adb862aae2477ec9143350"
}
```

Status: 200 OK Time: 41 ms Size: 273 B

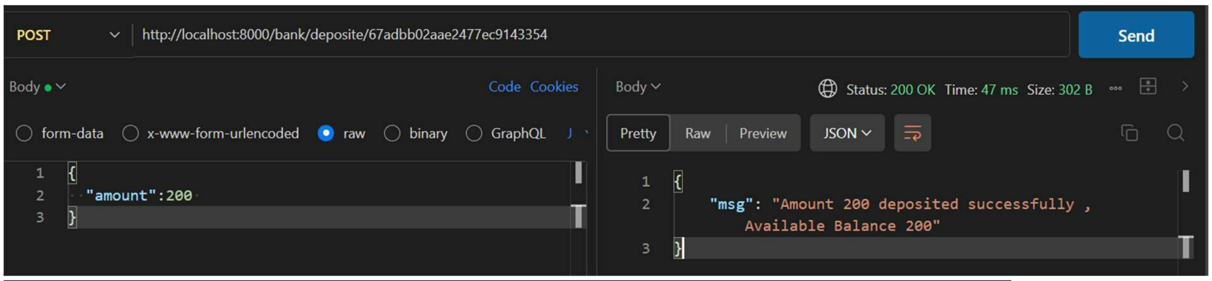
```
{
    "msg": "Account created Successfully"
}
```



```
_id: ObjectId('67adbb02aae2477ec9143354')
user_id : ObjectId('67adb862aae2477ec9143350')
balance : 0
__v : 0
```

- **METHOD : POST**
- **URL : http://localhost/bank/account**
- **This API is for Account creation.**

Money Deposit :

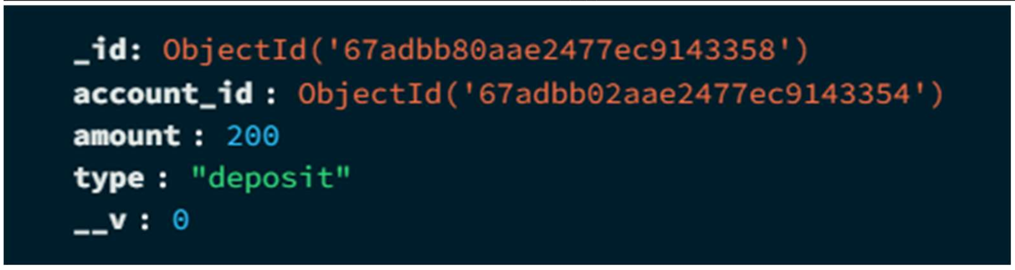


```
POST http://localhost:8000/bank/deposite/67adbb02aae2477ec9143354
```

```
{
  "amount":200
}
```

Status: 200 OK Time: 47 ms Size: 302 B

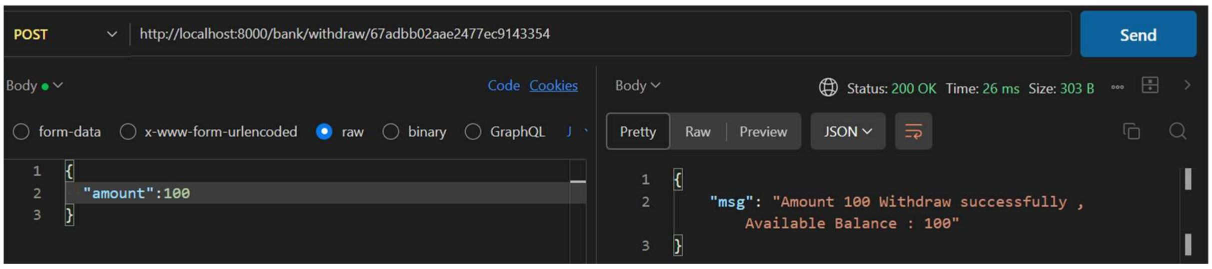
```
{
    "msg": "Amount 200 deposited successfully ,
     Available Balance 200"
}
```



```
_id: ObjectId('67adbb80aae2477ec9143358')
account_id : ObjectId('67adbb02aae2477ec9143354')
amount : 200
type : "deposit"
__v : 0
```

- **METHOD : POST**
- **URL : http://localhost/bank/deposite/:id**
- **This API is for Depositing money.**

Money Withdraw:

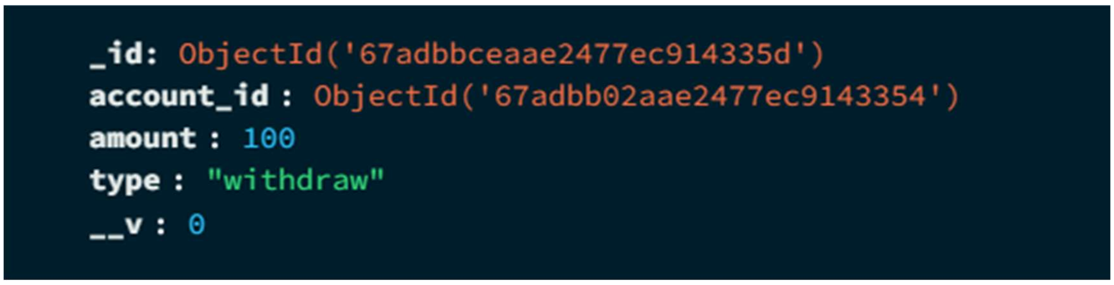


```
http://localhost:8000/bank/withdraw/67adbb02aae2477ec9143354
```

```
{
  "amount":100
}
```

Status: 200 OK Time: 26 ms Size: 303 B

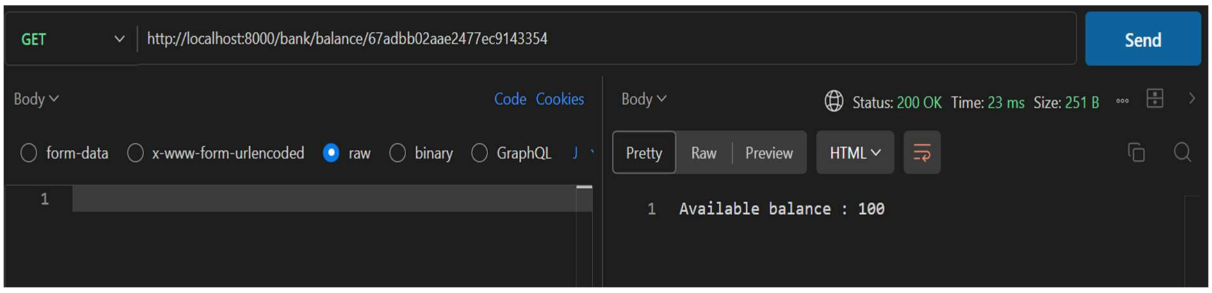
```
{
    "msg": "Amount 100 Withdraw successfully ,
        Available Balance : 100"
}
```



```
_id: ObjectId('67adbbceaae2477ec914335d')
account_id : ObjectId('67adbb02aae2477ec9143354')
amount : 100
type : "withdraw"
__v : 0
```

- **METHOD : POST**
- **URL : http://localhost/bank/withdraw/:id**
- **This API is for withdrawing money.**

Balance Check:



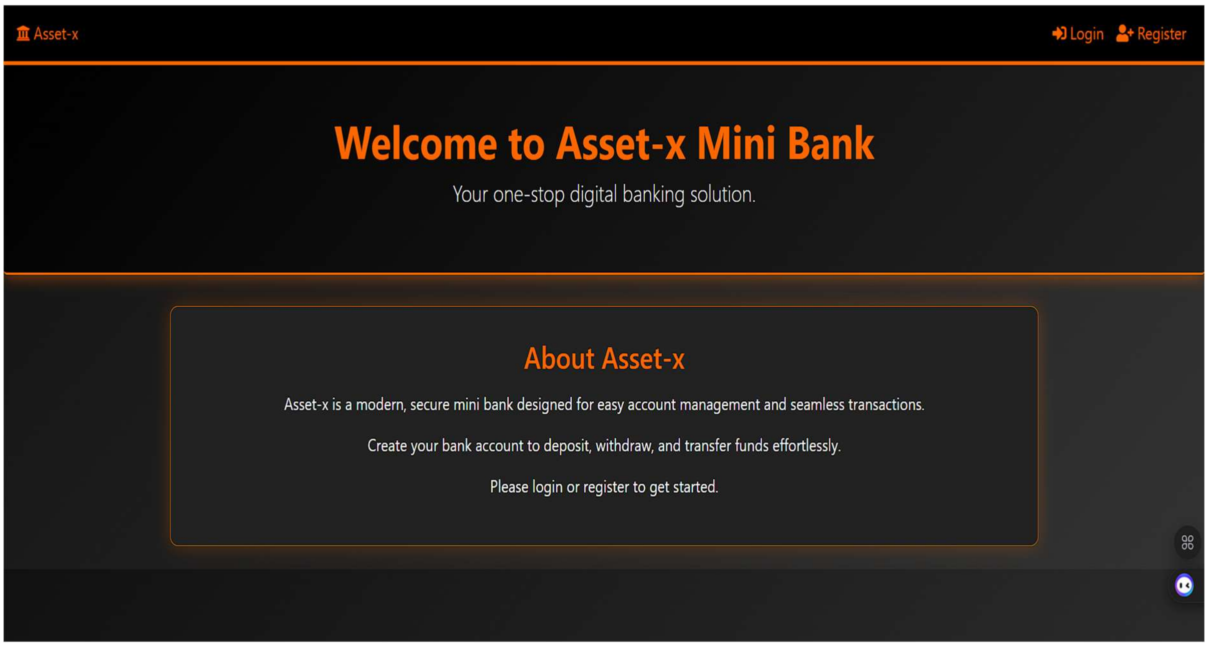
```
http://localhost:8000/bank/balance/67adbb02aae2477ec9143354
```

Status: 200 OK Time: 23 ms Size: 251 B

```
Available balance : 100
```

- **METHOD : GET**
- **URL : http://localhost/bank/balance/:id**
- **This API is for Checking balance.**

Screenshots of Implemented project:



Asset-x

Login Register

Welcome to Asset-x Mini Bank

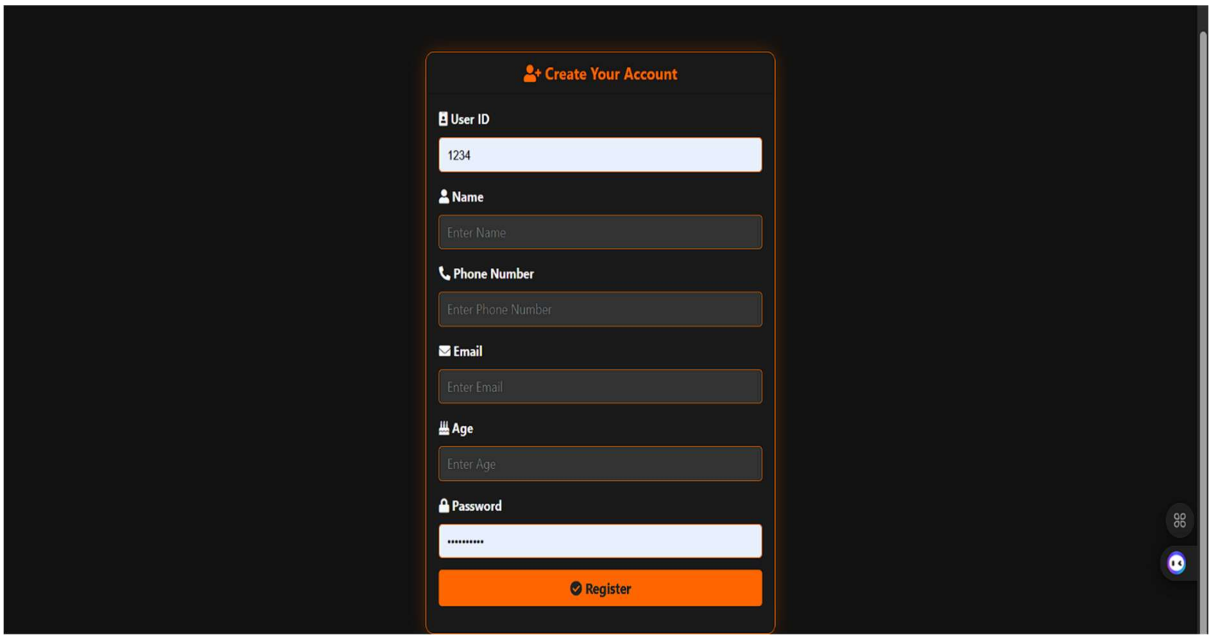
Your one-stop digital banking solution.

About Asset-x

Asset-x is a modern, secure mini bank designed for easy account management and seamless transactions.

Create your bank account to deposit, withdraw, and transfer funds effortlessly.

Please login or register to get started.



Create Your Account

User ID: 1234

Name: Enter Name

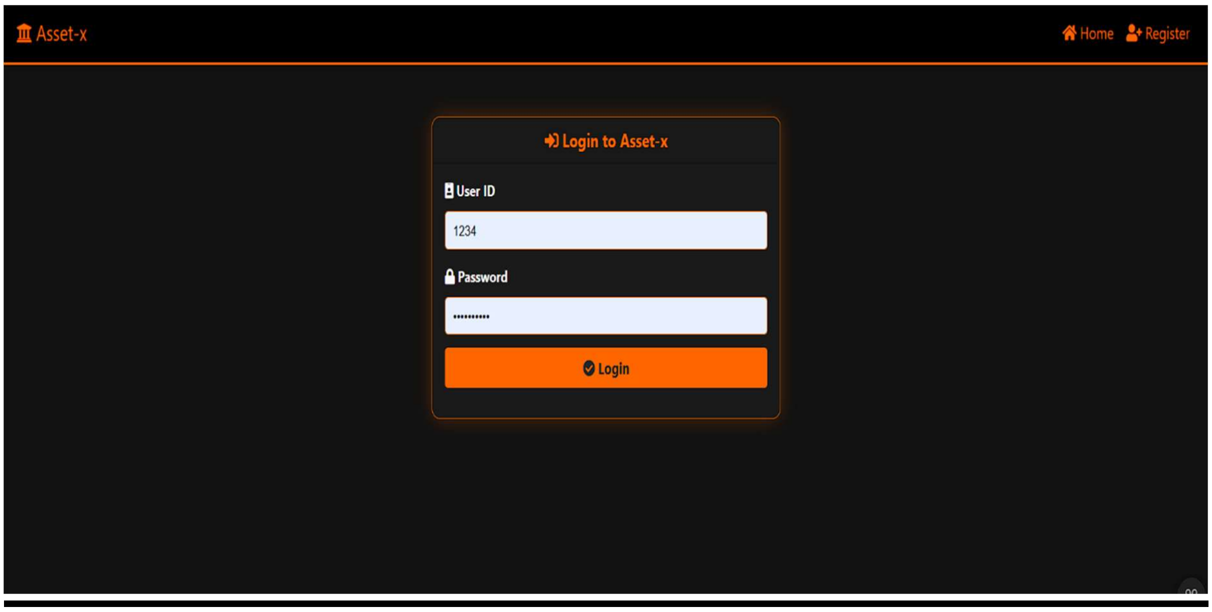
Phone Number: Enter Phone Number

Email: Enter Email

Age: Enter Age

Password: ••••••••••

Register


Asset-x

Home Register

Login to Asset-x

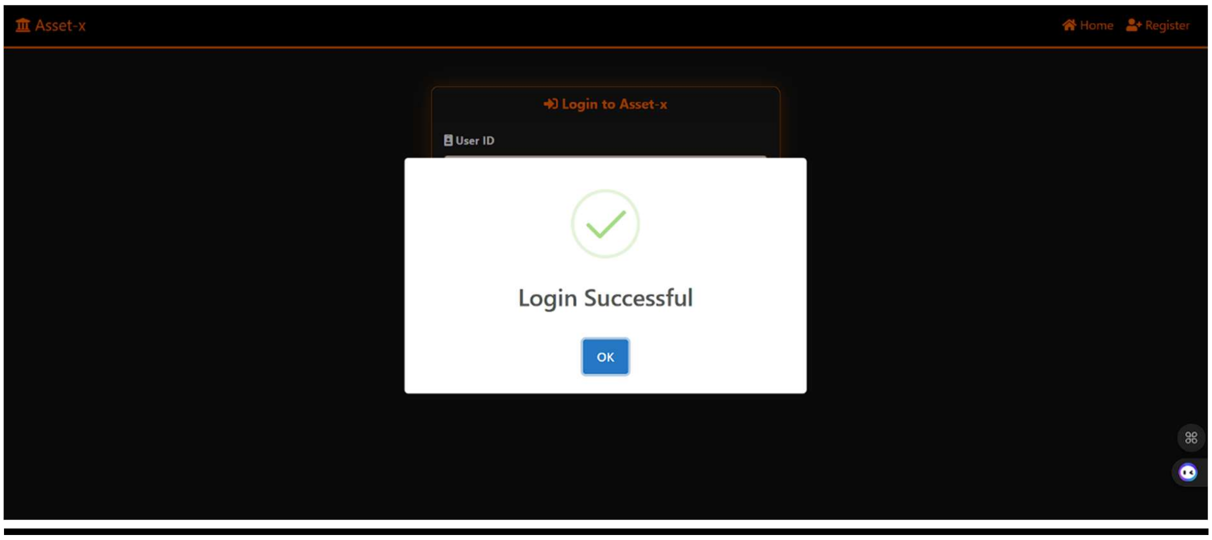
User ID

1234

Password

••••••••••

Login


Asset-x

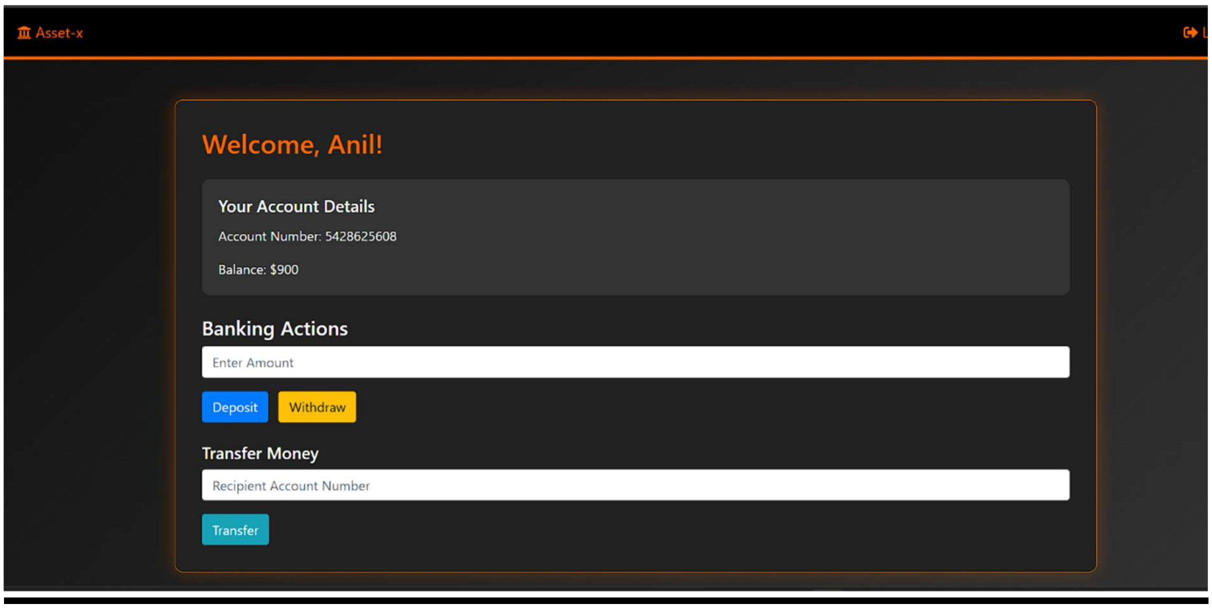
Home Register

Login to Asset-x

User ID

Login Successful

OK


Asset-x

Welcome, Anil!

Your Account Details

Account Number: 5428625608

Balance: $900

Banking Actions

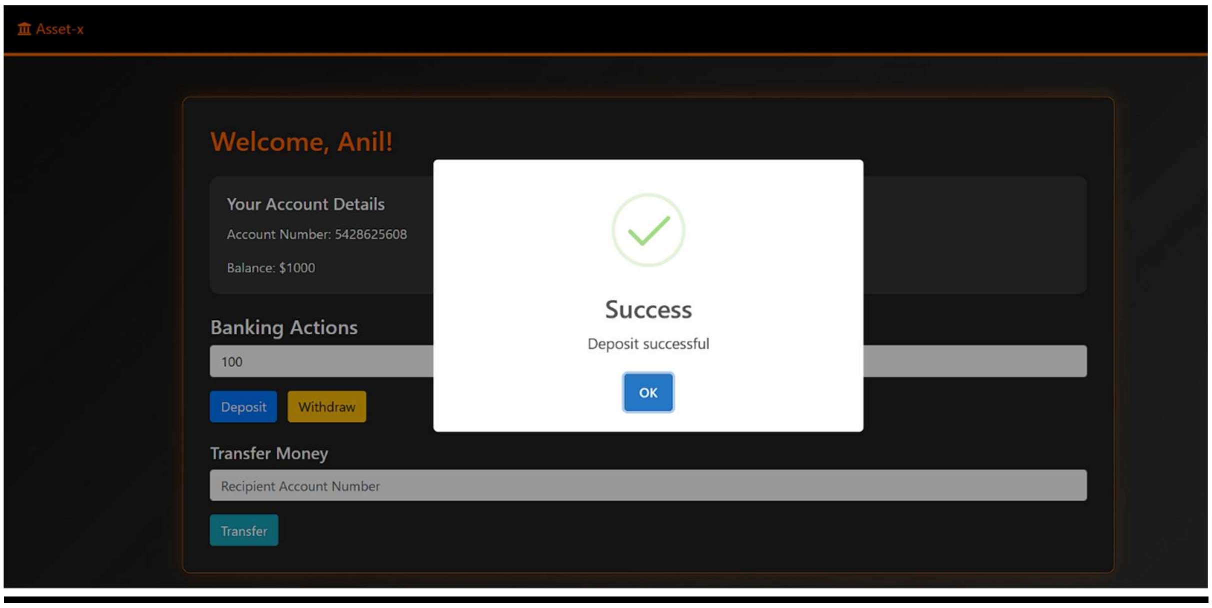
Enter Amount

Deposit Withdraw

Transfer Money

Recipient Account Number

Transfer


Asset-x

Welcome, Anil!

Your Account Details

Account Number: 5428625608

Balance: $1000

Banking Actions

100

Deposit Withdraw

Transfer Money

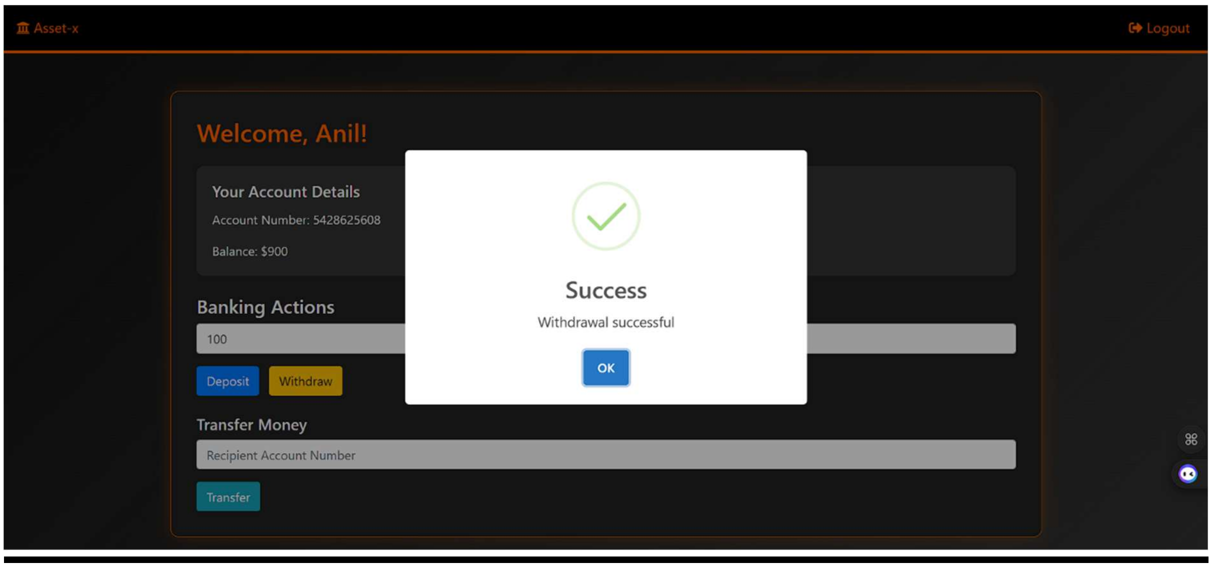
Recipient Account Number

Transfer

Success

Deposit successful

OK



Asset-x

Logout

Welcome, Anil!

Your Account Details

Account Number: 5428625608

Balance: $900

Banking Actions

100

Deposit Withdraw

Transfer Money

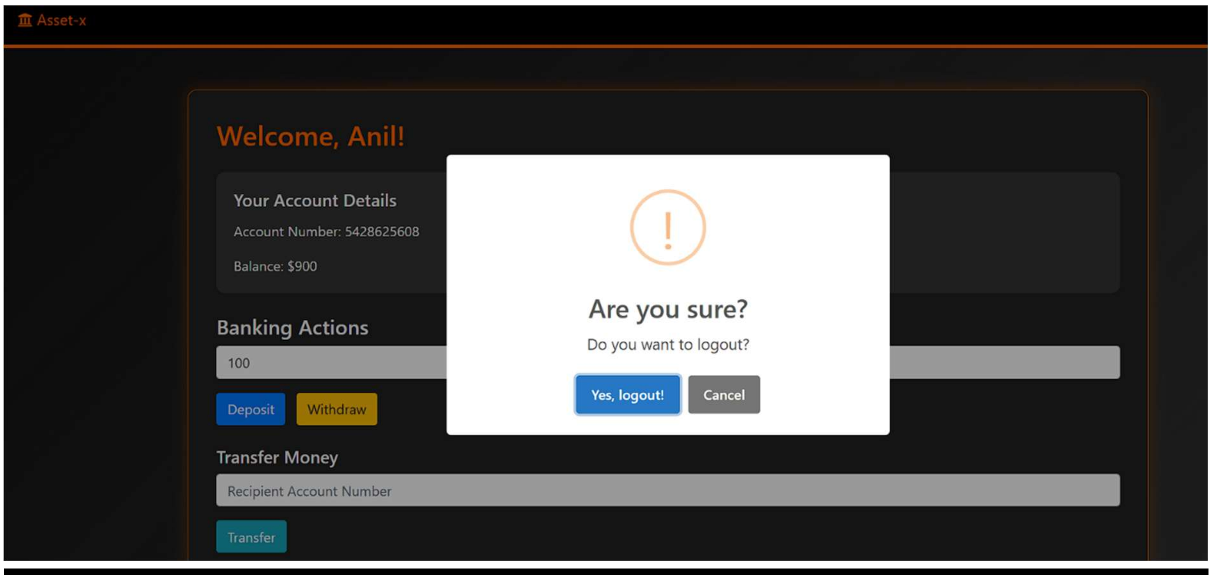
Recipient Account Number

Transfer

Success

Withdrawal successful

OK



Asset-x

Welcome, Anil!

Your Account Details

Account Number: 5428625608

Balance: $900

Banking Actions

100

Deposit Withdraw

Transfer Money

Recipient Account Number

Transfer

Are you sure?

Do you want to logout?

Yes, logout! Cancel