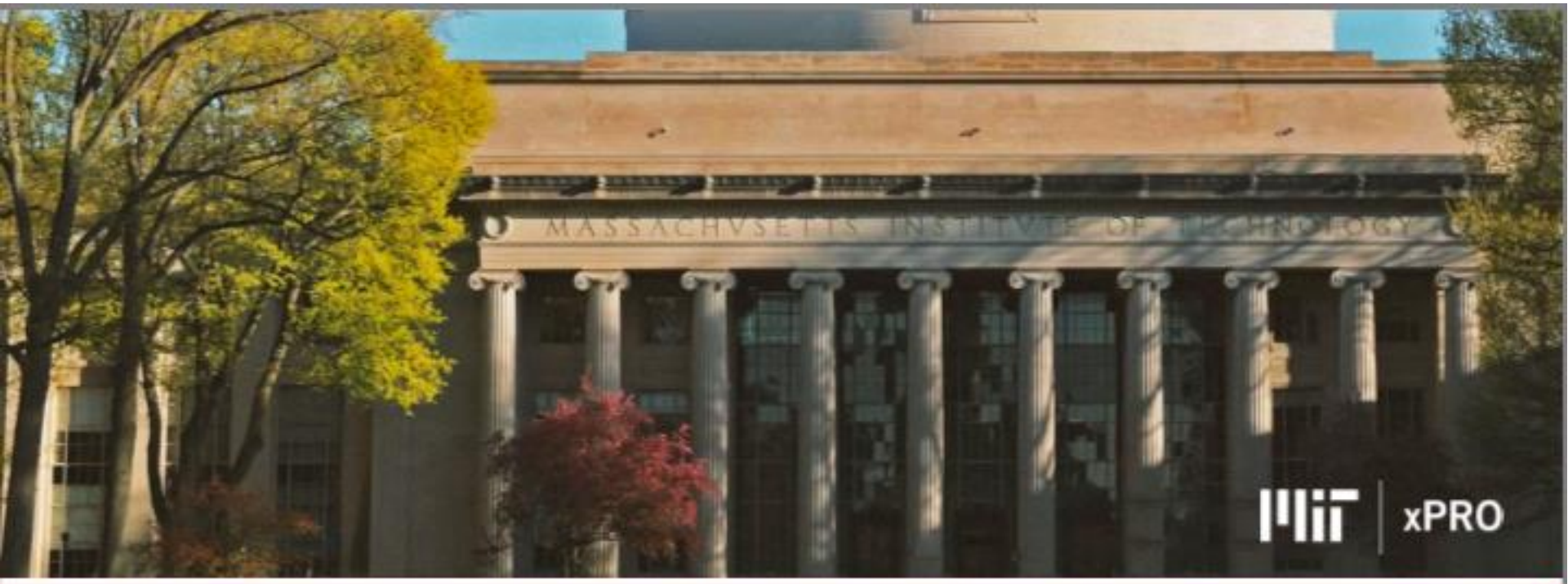
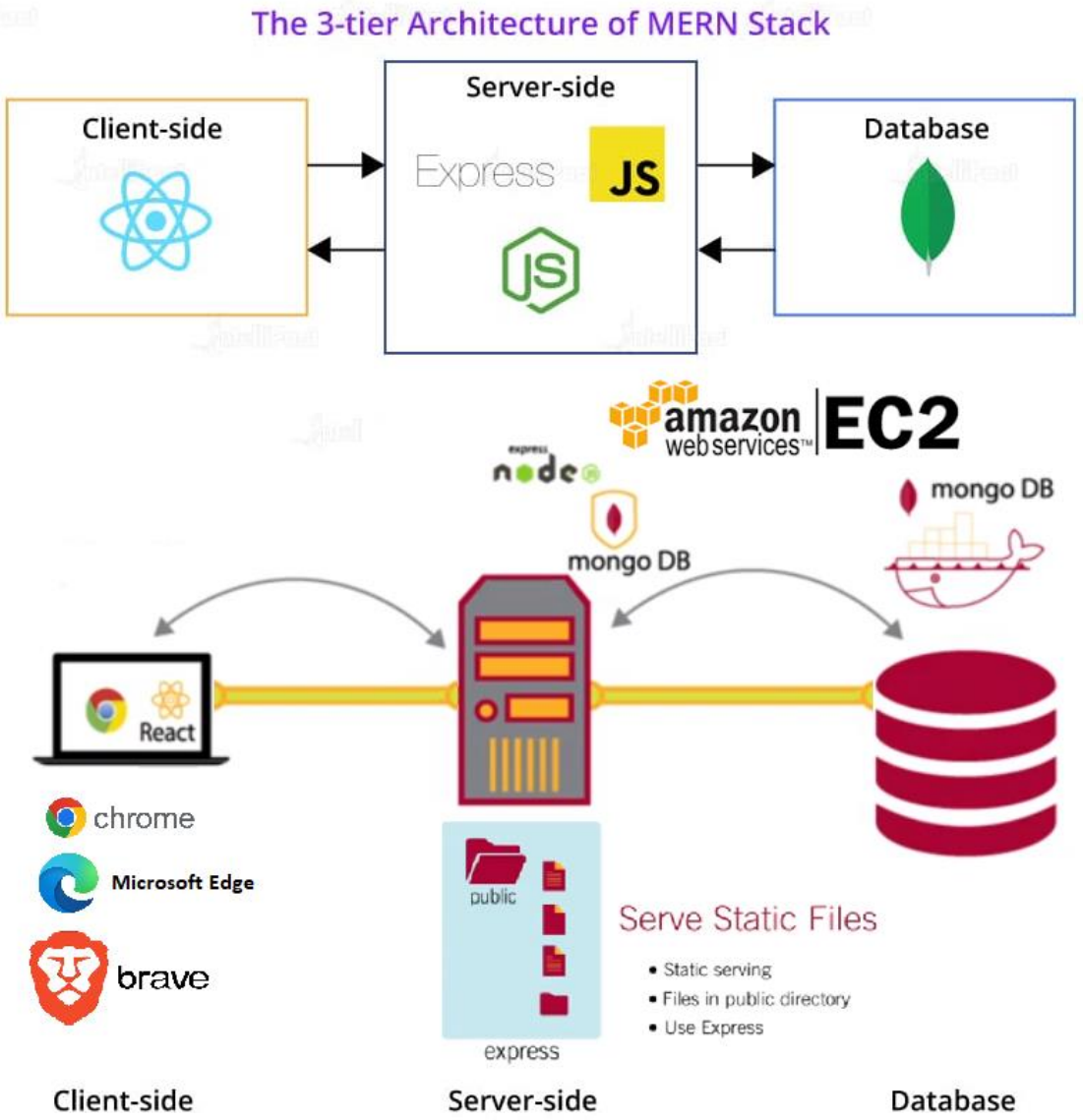


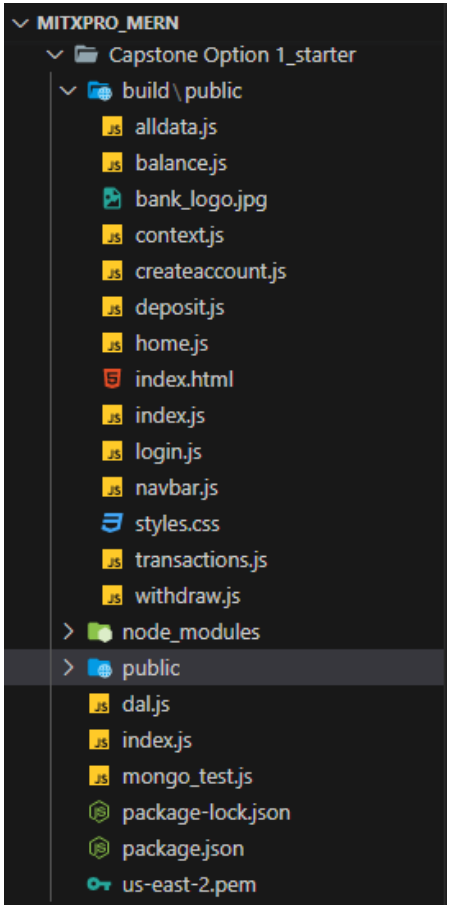
Banking Application Capstone Presentation

Omar Cerda Villalobos

Part 1: Front-End Architecture, Authentication, And App Diagram

Application Overview Diagram

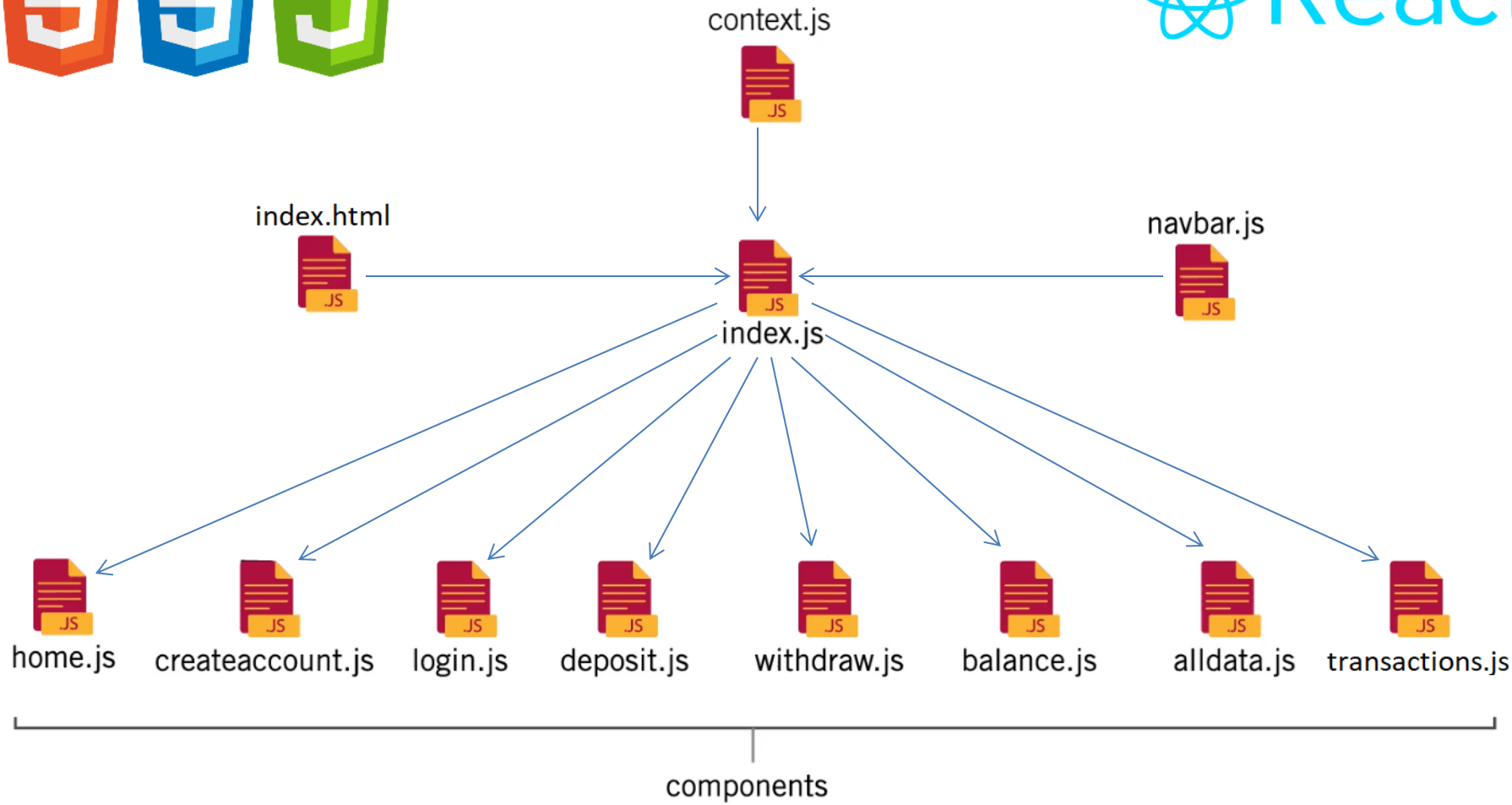




Front-End Architecture

Front-end Architecture

React



Authentication

login.js

```
function Login(){
  const [show, setShow]     = React.useState(true);
  const [status, setStatus] = React.useState('');

  return (
    <Card
      bgcolor="secondary"
      header="Login"
      status={status}
      body={show ?
        <LoginForm setShow={setShow} setStatus={setStatus}/> :
        <LoginMsg setShow={setShow} setStatus={setStatus}/>}
    />
  )
}
```

```
function handle(){
  console.log(email,password);
  if (!validate(email,    'email', props.setStatus))    return;
  if (!validate(password, 'password', props.setStatus)) return;

  fetch(`/account/login/${email}/${password}`)
  .then(response => response.text())
  .then(text => {
      try {
          const data = JSON.parse(text);
          props.setStatus('');
          props.setShow(false);
          console.log('JSON:', data);
      } catch(err) {
          props.setStatus(text)
          console.log('err:', text);
      }
  });
}
```

```
function LoginMsg(props){
  return(<>
    <h5>Success</h5>
    <button type="submit"
      className="btn btn-light"
      onClick={() => props.setShow(true)}>
        Authenticate again
    </button>
  </>);
}
```

{ REST-API }

mongoDB.

index.js

```
// login user
app.get('/account/login/:email/:password', function (req, res) {

    dal.find(req.params.email).
        then((user) => {

            // if user exists, check password
            if(user.length > 0){
                if (user[0].password === req.params.password){
                    res.send(user[0]);
                }
                else{
                    res.send('Login failed: Wrong password!');
                }
            }
            else{
                res.send('Login failed: User not found!');
            }
    });

});
```

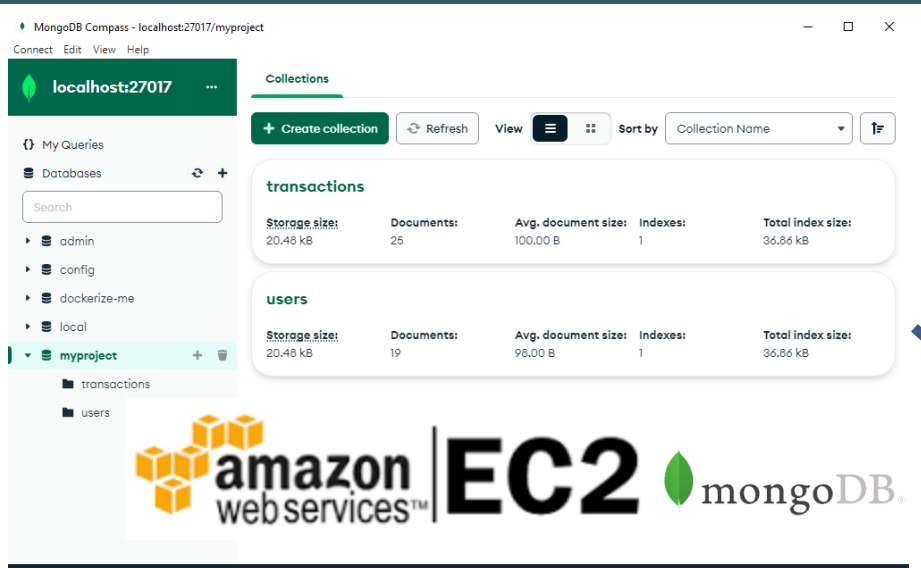
dal.js

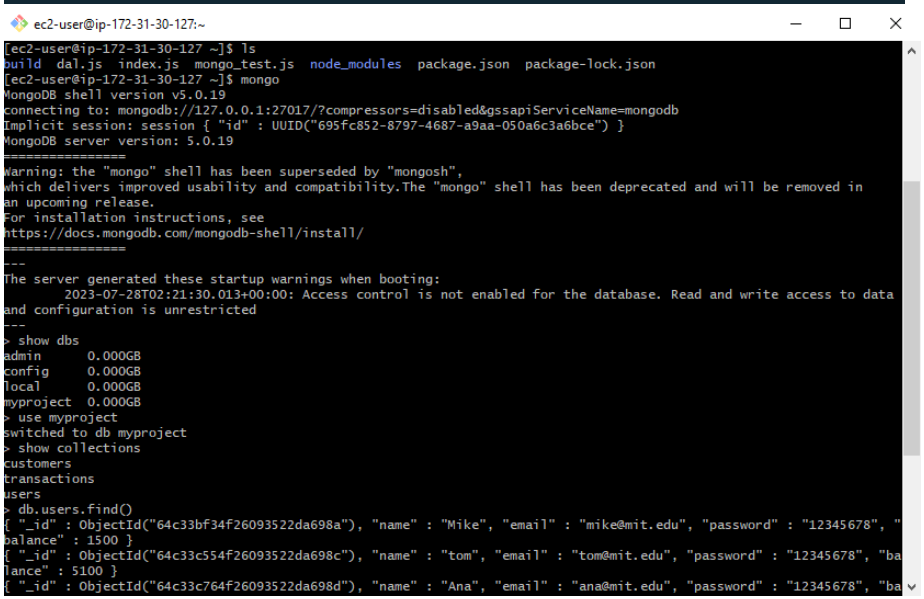
```
// find user account
function find(email){
    return new Promise((resolve, reject) => {
        const customers = db
            .collection('users')
            .find({email: email})
            .toArray(function(err, docs) {
                err ? reject(err) : resolve(docs);
            });
    })
}
```

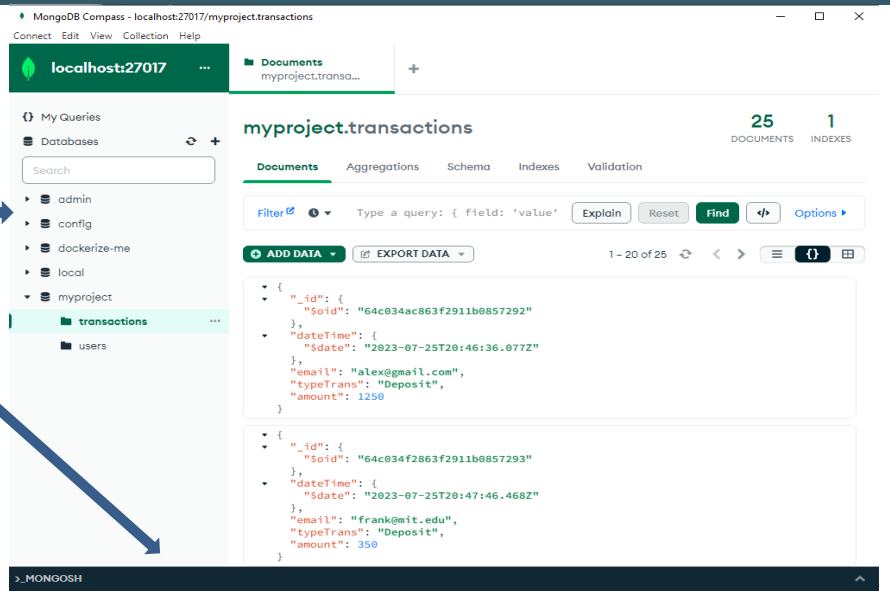
MIT xPRO

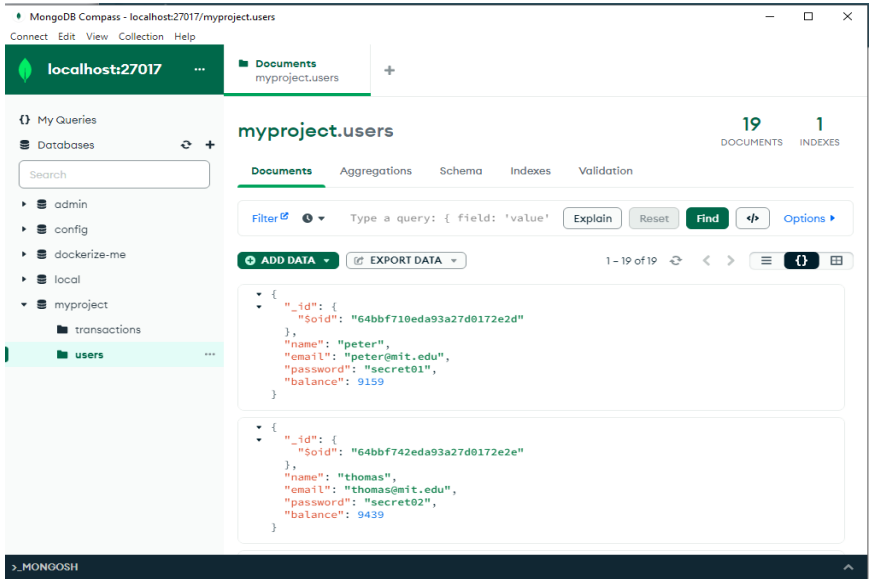
Part 2: Database And API

Database









MIT xPRO

API

index.js

```
var express = require('express');
var app     = express();
var cors    = require('cors');
var dal     = require('./dal.js');


// used to serve static files from public directory
app.use(express.static('build/public'));
// app.use(express.static('public'));

app.use(cors());

// create user account
app.get('/account/create/:name/:email/:password', function

    // check if account exists
    dal.find(req.params.email).
        then((users) => {

            // if user exists, return error message
            if(users.length > 0){
                console.log('Error: User already exists');
                res.send({'Error': 1});
            }
            else{
                // else create user
                dal.create(req.params.name,req.params.email
                    then((user) => {
                        console.log(user);
                        res.send(user);
                    });
            }

        });
});


// login user
app.get('/account/login/:email/:password', function (req, res) {
```

dal.js

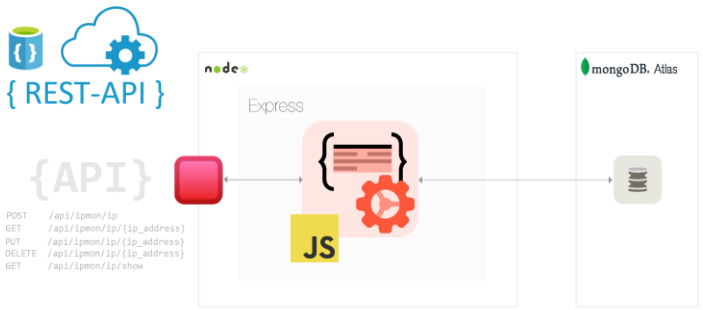
```
const MongoClient = require('mongodb').MongoClient;
const url         = 'mongodb://localhost:27017';
//const url           = 'mongodb://doadmin:H0p4h9L3T2mo7Z56@myproject-109eca8f.mongo.ondigitalocean.com:270
let db            = null;

// connect to mongo
MongoClient.connect(url, {useUnifiedTopology: true}, function(err, client) {
    console.log("Connected successfully to db server");

    // connect to myproject database
    db = client.db('myproject');
});

// create user account
function create(name, email, password){
    return new Promise((resolve, reject) => {
        const collection = db.collection('users');
        const doc = {name, email, password, balance: 0};
        collection.insertOne(doc, {w:1}, function(err, result) {
            err ? reject(err) : resolve(doc);
        });
    })
}

// create transaction
function transaction(email, typeTrans, amount){
    return new Promise((resolve, reject) => {
        const collection = db.collection('transactions');
        const dateTime   = new Date();
        const doc = {dateTime, email, typeTrans, amount};
        collection.insertOne(d // all users
            err ? reject(err) function all(){
        });
    })
}
```

dal.js

```
// find user account
function find(email){
    return new Promise((resolve, reject) => {
        const customers = db
            .collection('users')
            .find({email: email})
            .toArray(function(err, docs) {
                err ? reject(err) : resolve(docs);
            });
    })
}

// find user account
function findOne(email){
    return new Promise((resolve, reject) => {
        const customers = db
            .collection('users')
            .findOne({email: email})
            .then((doc) => resolve(doc))
            .catch((err) => reject(err));
    })
}

// update - deposit/withdraw amount
ion update(email, amount){
                                                    t) => {
```

dal.js

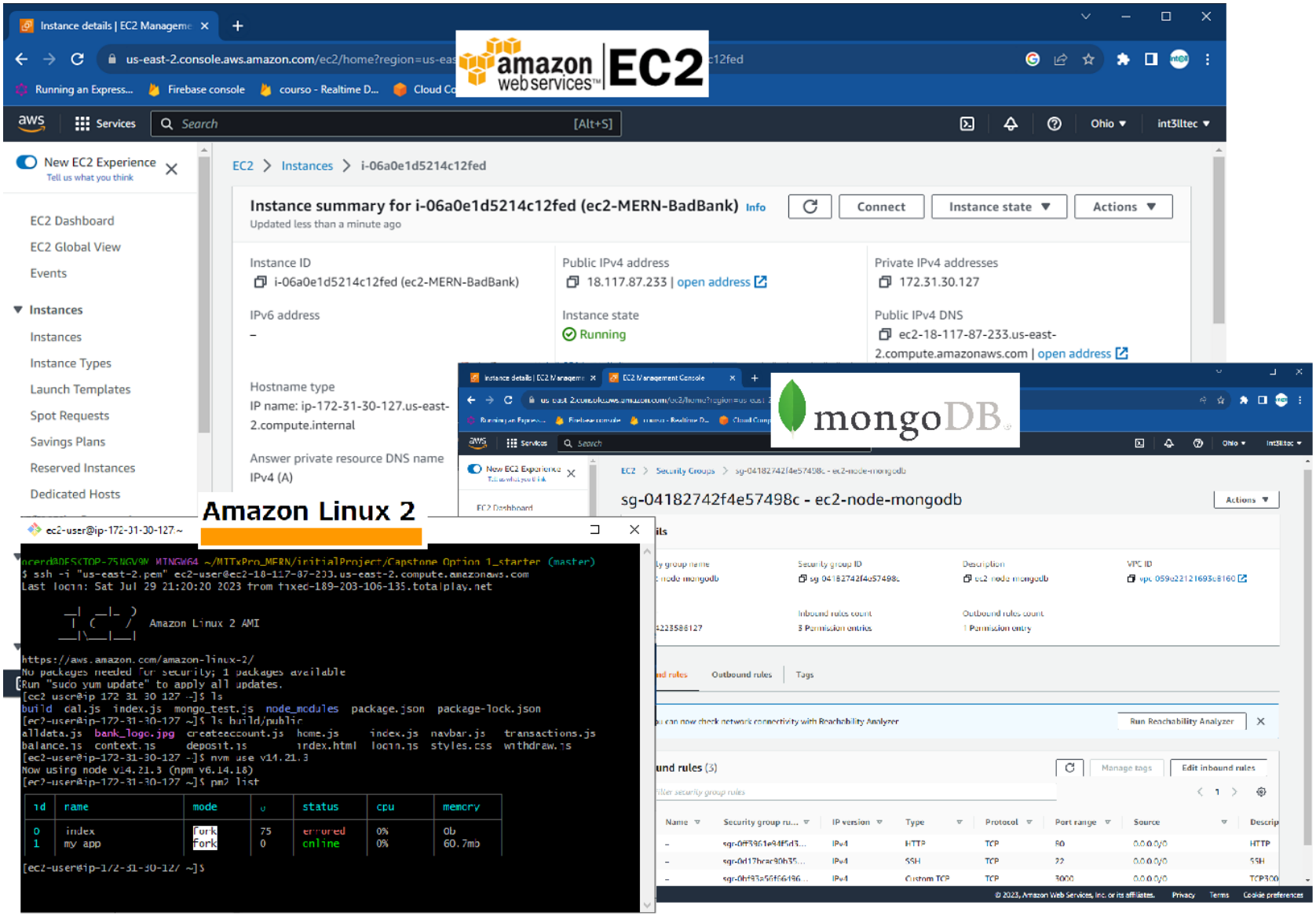
```
    return new Promise((resolve, reject) => {
        const customers = db
            .collection('users')
            .find({})
            .toArray(function(err, docs) {
                err ? reject(err) : resolve(docs);
            });
    })
}

// transactions
function transactions(){
    return new Promise((resolve, reject) => {
        const customers = db
            .collection('transactions')
            .find({})
            .toArray(function(err, docs) {
                err ? reject(err) : resolve(docs);
            });
    })
}

module.exports = {create, transaction, findOne, find, update, all, transactions};
```



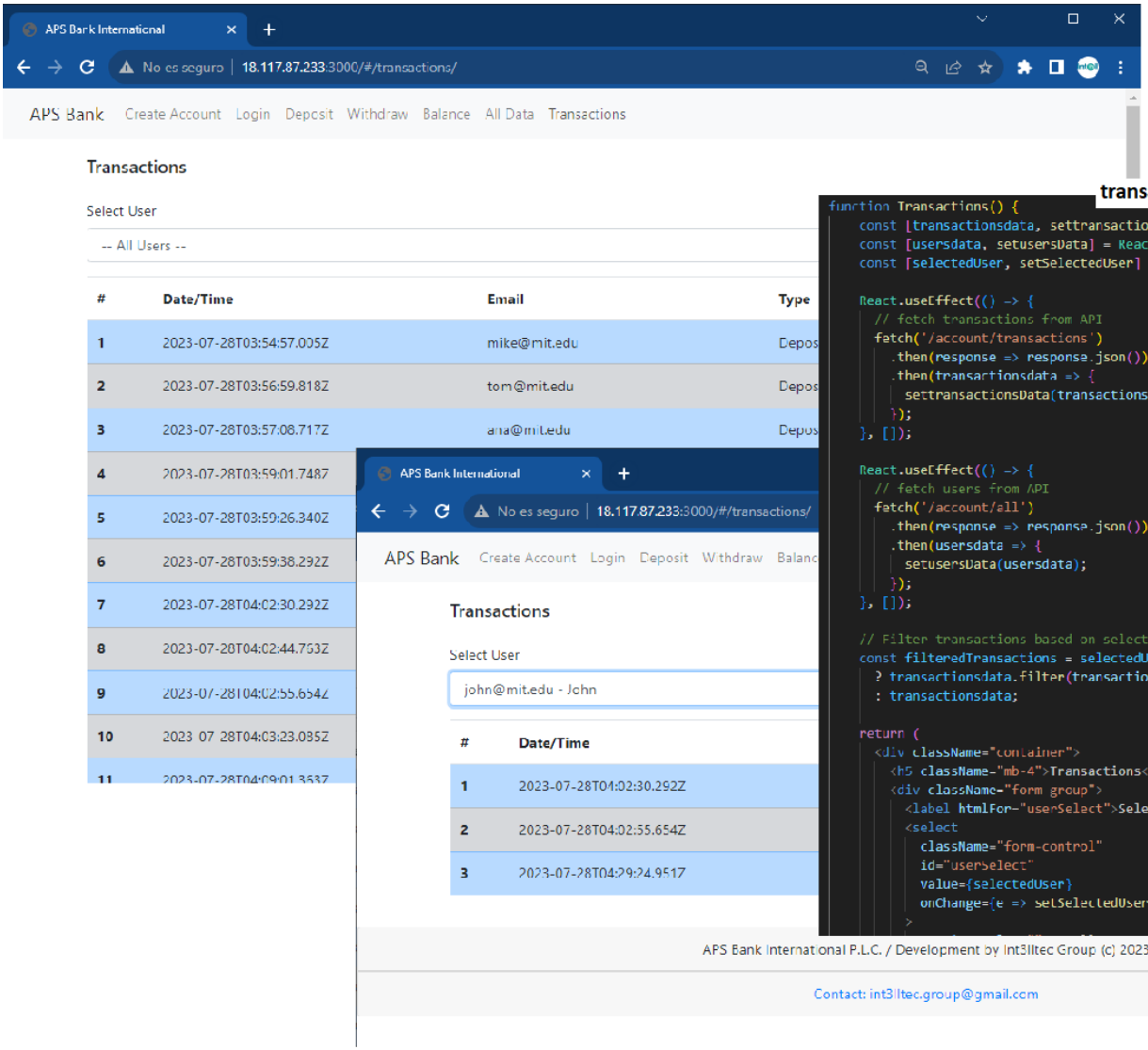
MIT xPRO

Part 3: Deployment, Additional Features, App Demonstration, Reflection

Deployment


amazon web services EC2
mongoDB.
Amazon Linux 2
New EC2 Experience
EC2 Dashboard
EC2 Global View
Events
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Instance summary for i-06a0e1d5214c12fed (ec2-MERN-BadBank)
Connect
Instance state
Actions
Instance ID
i-06a0e1d5214c12fed (ec2-MERN-BadBank)
Public IPv4 address
18.117.87.233 | open address
Private IPv4 addresses
172.31.30.127
IPv6 address
Instance state
Running
Public IPv4 DNS
ec2-18-117-87-233.us-east-2.compute.amazonaws.com | open address
Hostname type
IP name: ip-172-31-30-127.us-east-2.compute.internal
Answer private resource DNS name
IPv4 (A)
sg-04182742f4e57498c - ec2-node-mongodb
Actions
Outbound rules
Tags
Run Reachability Analyzer
Edit inbound rules
Manage tags
ec2-user@ip-172-31-30-127:~
$ ssh -i "us-east-2.pem" ec2-user@ec2-18-117-87-233.us-east-2.compute.amazonaws.com
Last login: Sat Jul 29 21:20:20 2023 from fixed-189-203-106-135.totalplay.net
Amazon Linux 2 AMI
https://aws.amazon.com/amazon-linux-2/
No packages needed for security; 1 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-30-127 ~]$ ls
build cal.js index.js mongo_test.js node_modules package.json package-lock.json
[ec2-user@ip-172-31-30-127 ~]$ ls build/public
alldata.js bank_logo.jpg createaccount.js home.js index.js navbar.js transactions.js
balance.js context.js deposit.js index.html login.js styles.css withdraw.js
[ec2-user@ip-172-31-30-127 ~]$ nvm use v14.21.3
Now using node v14.21.3 (npm v6.14.18)
[ec2-user@ip-172-31-30-127 ~]$ pm2 list
id name mode ↺ status cpu memory
0 index fork 75 errored 0% 0b
1 my app fork 0 online 0% 60.7mb
[ec2-user@ip-172-31-30-127 ~]$

Additional Features



Record each transaction (Deposit/Withdraw) in DB.

transactions.js

```
function Transactions() {
  const [transactionsdata, settransactionsData] = React.useState([]);
  const [usersdata, setusersData] = React.useState([]);
  const [selectedUser, setSelectedUser] = React.useState("");

  React.useEffect(() => {
    // fetch transactions from API
    fetch('/account/transactions')
      .then(response => response.json())
      .then(transactionsdata => {
        settransactionsData(transactionsdata);
      });
  }, []);

  React.useEffect(() => {
    // fetch users from API
    fetch('/account/all')
      .then(response => response.json())
      .then(usersdata => {
        setusersData(usersdata);
      });
  }, []);

  // Filter transactions based on selected user
  const filteredTransactions = selectedUser
    ? transactionsdata.filter(transaction => transac
    : transactionsdata;

  return (
    <div className="container">
      <h5 className="mb-4">Transactions</h5>
      <div className="form-group">
        <label htmlFor="userSelect">Select User</lab
        <select
          className="form-control"
          id="userSelect"
          value={selectedUser}
          onChange={e => setSelectedUser(e.currentTar
        >
```

index.js

```
// create transaction
app.get('/account/transaction/:email/:amount', function (req, res) {

    var amount          = Number(req.params.amount);
    var typeTransaction = "Deposit";
    // check the transaction is a withdraw or deposit
    if  (amount <0) {
        typeTransaction = "Withdraw";
    };

    dal.transaction(req.params.email, typeTransaction, amount).
        then((response) => {
            console.log(response);
            res.send(response);
        });


});
```

dal.js

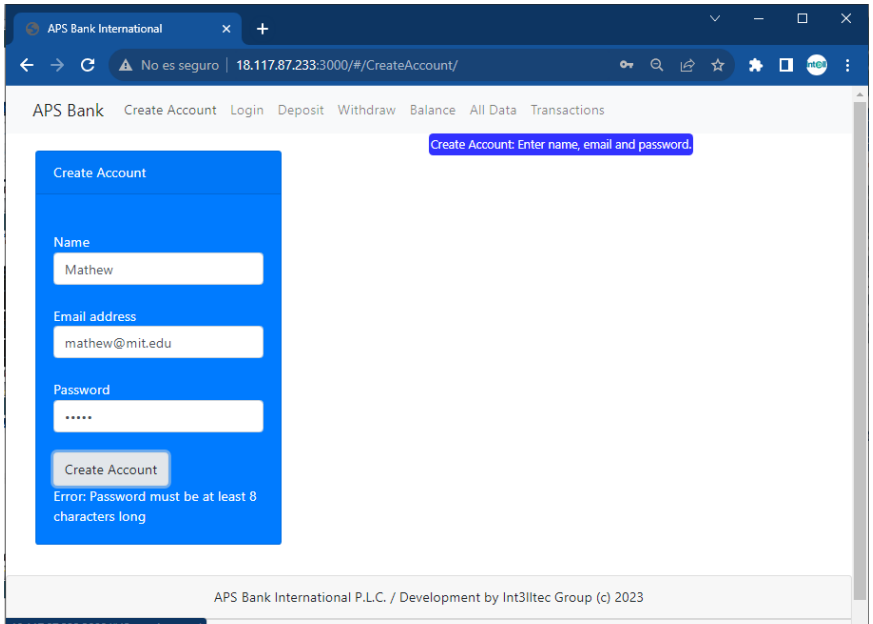
```
// create transaction
function transaction(email, typeTrans, amount){
    return new Promise((resolve, reject) => {
        const collection = db.collection('transactions');
        const dateTime   = new Date();
        const doc = {dateTime, email, typeTrans, amount};
        collection.insertOne(doc, {w:1}, function(err, result) {
            err ? reject(err) : resolve(doc);
        });
    })
}
```

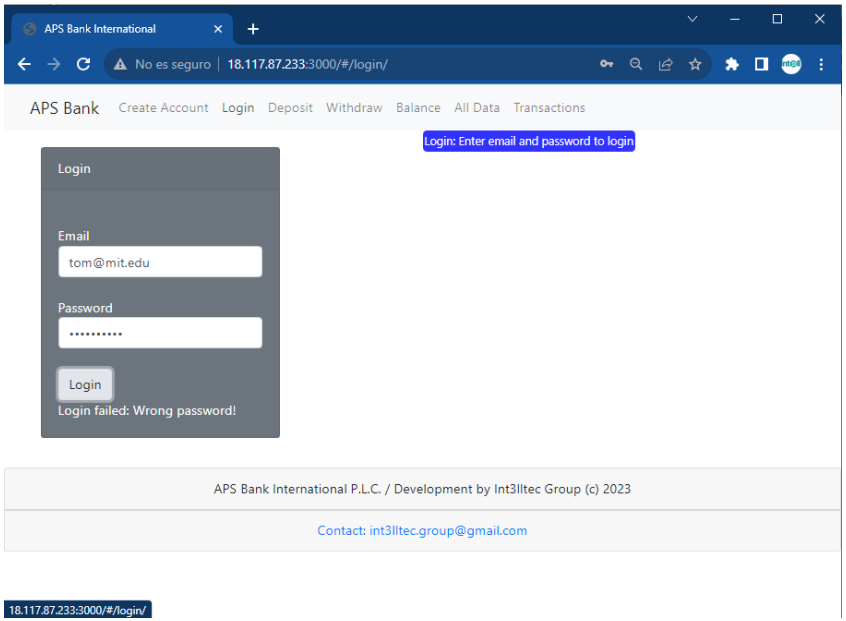
dal.js

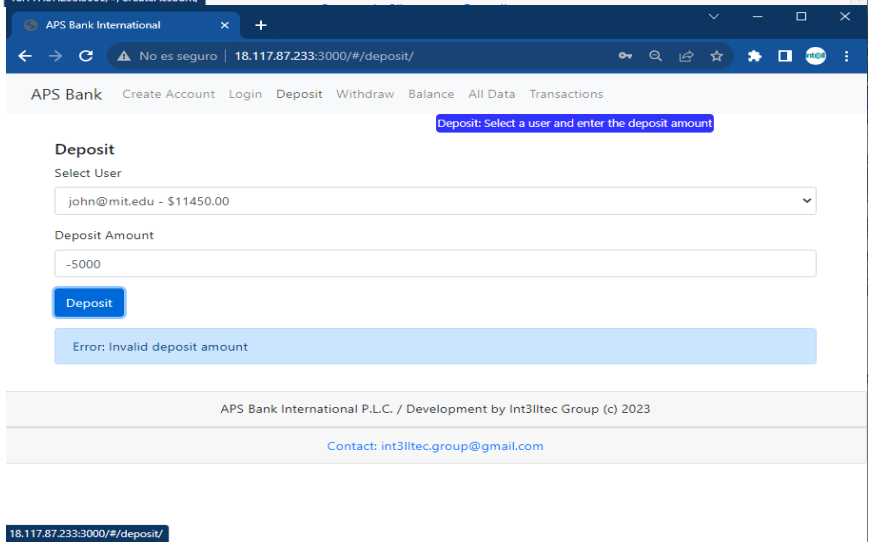
```
// transactions
function transactions(){
    return new Promise((resolve, reject) => {
        const customers = db
            .collection('transactions')
            .find({})
            .toArray(function(err, docs) {
                err ? reject(err) : resolve(docs);
            });
    })
}
```

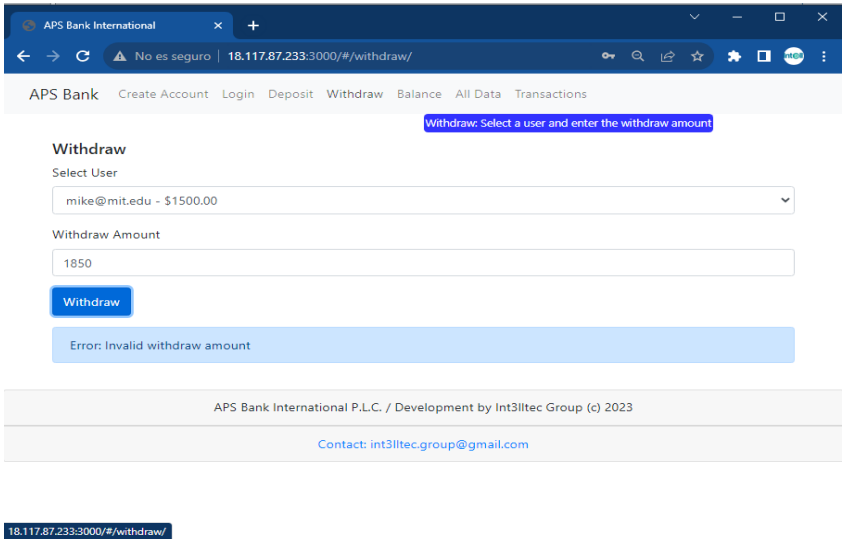
Show the transactions of All Users or Selected user

Application Demonstration – Create Account, Log In, Deposit, Withdraw









Reflection

- If you started the project today, how would you structure your code differently?

If I started the project today, I would structure the code differently by following a more modular and scalable architecture. I would use design patterns such as **MVC (Model-View-Controller)** to clearly separate **business logic**, **views**, and **interactions with the database**. Additionally, I would implement the use of **controllers to handle different server routes and actions**. I would also use **TypeScript** to add static typing and improve code maintainability and robustness. Moreover, I would implement **unit and integration tests** to ensure more reliable code.

- If you started the project today, what additional features would you build?

If I started the project today, I would add features such as **two-factor authentication (2FA)** to enhance the security of user accounts. I would also implement real-time notifications so that users receive instant alerts about their transactions. **Another interesting feature would be to allow users to set savings goals and receive notifications when they are close to achieving them**. Additionally, I would consider adding support for **transfers between accounts** and the ability to link external bank accounts for comprehensive financial management.


xPRO