WEB-TECHNOLOGIES PROJECT REPORT

on

E-BANKING SYSTEM

Bachelor of Technology

In

Computer Science and Engineering

Under the esteemed guidance of

Mrs. Venkata Ratnam

Asst. Professor, Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

SANGIVALASA, VISAKHAPATNAM

Bonafide Certificate

This is to certify that this project report ”E-Banking System” is the Bonafide work of D. Chaitanya(A22126510144), P. Meghana(A22126510167), P. Sai Deepak (A22126510168), N. Lokesh(A22126510166), M. Nikitha(A22126510161). This project is carried out and is submitted in the partial fulfillment of the requirements for the award of BACHELOR OF TECHNOLOGY in Computer Science and Engineering, under Anil Neerukonda Institute of Technology and Sciences during the academic year 2024-2025.

Head of the Department	Project Guide	Project Reviewer
Prof. G. Srinivas	Mrs. K.Venkataratnam	G. Pranitha
(Professor & HOD)	(Asst.Professor)	(Asst. Professor)
Department of CSE	Department of CSE	Department of CSE
ANITS	ANITS	ANITS

PROJECT TITLE: ONLINE BANKING SYSTEM

Name of the Student	Roll No
D. Chaitanya	A22126510144
P. Meghana	A22126510167
P. Sai Deepak	A22126510168
N. Lokesh	A22126510166
M. Nikitha	A22126510161

Declaration

This is to certify that the project work entitled ”E-Banking System” is a Bonafide work carried out by as a part of BTech 3rd year 2nd semester of Computer Science and Engineering of Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam during the academic year 2024-2025.

We are D. Chaitanya, P. Meghana, P. Sai Deepak, N. Lokesh, M. Nikitha of 3rd year B.Tech, Department of Computer Science and Engineering from ANITS, Visakhapatnam, hereby declare that the project work entitled ”E-Banking System” is carried out by us and is submitted in the fulfillment of the requirement for the award of Bachelor of Technology in Computer Science and Engineering, under Anil Neerukonda Institute of Technology and Sciences during the academic year 2024-2025 has not been submitted to any other university for the award of any kind of degree.

Name	**Reg.No**
D. Chaitanya	A22126510144
P. Meghana	A22126510167
P. Sai Deepak	A22126510168
N. Lokesh	A22126510166
M. Nikitha	A22126510161

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who made it possible, and whose constant guidance and encouragement always upheld the morale. We take a great pleasure in presenting a project, which is the result of a studied blend of both research and knowledge.

We first take the privilege to thank our Head of the department Prof M. G. SRINIVAS, for permitting us in laying the first stone of success. We feel grateful to thank MRS. K. VENKATA RATNAM Madam, who is our project guide and who shared her valuable knowledge with us and made us understand the real essence of the topic and created interest in us to put our continuous efforts in the project.

Name	**Reg.No**
D. Chaitanya	A22126510144
P. Meghana	A22126510167
P. Sai Deepak	A22126510168
N. Lokesh	A22126510166
M. Nikitha	A22126510161

Contents

Chapter 1

Project Abstract

The proliferation of online banking has revolutionized the way financial transactions are conducted, offering unparalleled convenience and accessibility. However, with this convenience comes the challenge of ensuring robust security measures to protect sensitive financial data and prevent unauthorized access.

This project aims to address these challenges by proposing innovative solutions to enhance both security and user experience in online banking systems.

The project will begin with a comprehensive analysis of existing security protocols and user interface designs in online banking systems. This analysis will identify potential vulnerabilities and areas for improvement. Subsequently, the project will focus on the development and implementation of advanced security measures such as multi-factor authentication, encryption techniques, and real-time fraud detection algorithms.

Additionally, user-centric design principles will be employed to create intuitive and user- friendly interfaces that enhance the overall banking experience.

The effectiveness of the proposed solutions will be evaluated through rigorous testing procedures, including simulated cyber-attacks and user feedback surveys. Furthermore, the project will explore the integration of emerging technologies such as biometrics and blockchain to further enhance security and streamline transactions.

By the conclusion of this project, we anticipate significant advancements in the security and usability of online banking systems. These advancements will not only bolster consumer trust and confidence but also contribute to the continued growth and evolution of digital banking services in an increasingly interconnected world.

Chapter 2

Objectives

Here are a few objectives for your E-Banking System project:

1. To develop a secure and user-friendly online banking platform that allows users to perform various banking transactions, such as account creation, fund transfers, and transaction history viewing.
2. To implement robust security measures to protect user data and prevent unauthorized access to accounts.
3. To provide an intuitive user interface that enhances the overall user experience and simplifies the process of managing finances online.
4. To enable real-time monitoring of transactions and account activities for both users and administrators.
5. To facilitate easy integration with existing banking systems and databases for seamless data management.
6. To ensure compatibility with various devices and platforms, including web and mobile applications, to provide users with flexibility and convenience in accessing their accounts.
7. To conduct thorough testing and validation of the system to ensure its reliability, performance, and security.
8. To gather user feedback and make necessary improvements to enhance the system's functionality and usability.
9. To explore the potential for future enhancements, such as the integration of emerging technologies like artificial intelligence and machine learning for fraud detection and personalized banking experiences.

Chapter 3

Software & Modules Used

1. **Deno:** Deno is a runtime for JavaScript and TypeScript, which is used to run the server-side code. It is a secure runtime that allows you to run JavaScript and TypeScript code outside of a web browser. It also serves a built-in package manager and supports ES modules.

2. **Express.js:** Express.js is a web application framework for Node.js, designed for building web applications and APIs. It provides a robust set of features for web and mobile applications, including routing, middleware support, and template rendering.

3. **MySQL:** MySQL is an open-source relational database management system. It is widely used for web applications and is known for its reliability and performance. MySQL allows you to store and retrieve data efficiently, and it supports SQL (Structured Query Language) for querying and managing databases. In this project, MySQL is used to store user information, transaction details, and other relevant data. This project uses the MariaDB version of MySQL, which is an open-source fork of MySQL that is fully compatible with it.

4. **EJS (Embedded JavaScript):** EJS is a simple templating language that lets you generate HTML markup with plain JavaScript. It is used to render dynamic content on the server side and send it to the client. EJS allows you to include JavaScript code within your HTML, making it easy to create dynamic web pages.

5. **Git:** Git is a distributed version control system that allows you to track changes in your codebase. It is widely used for source code management in software development. Git allows multiple developers to work on the same project simultaneously without conflicts. It provides features like branching, merging, and version history, making it easier to collaborate on projects.

6. **GitHub:** GitHub is a web-based platform that uses Git for version control. It provides a user-friendly interface for managing Git repositories, making it easier to collaborate with other developers. GitHub also offers features like issue tracking, pull requests, and project management tools, making it a popular choice for open-source projects and team collaboration.

7. **Code Editor (Visual Studio Code):** Visual Studio Code (VS Code) is a lightweight but powerful source code editor that runs on your desktop. It is available for Win-

dows, macOS, and Linux. VS Code includes support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. It also has a rich ecosystem of extensions for additional features and functionality.

- Utilize Visual Studio Code as the code editor for project development and editing.

Installation Steps:

1. **Install Deno:** Follow the official Deno installation guide to set up Deno on your system. You can find the installation instructions at `https://deno.land/#installation`.
2. **Install Git:** Download and install Git from the official website: `https://git-scm.com/downloads`. Follow the instructions for your operating system.
3. **Project Setup:** Clone the project repository from GitHub using the following command:

```
        git clone https://github.com/Atan-D-RP4/wt_project
```

Navigate to the project directory:

```
        cd wt_project/express
```

Now, simply run the following command to start the server:

```
        deno run dev
```

This will install all the required dependencies and start the server.

Chapter 4

Structure of the Project

4.1 Overview

The project structure is organized into several directories and files, each serving a specific purpose. Below is a brief overview of the project structure:

- **main.ts:** The main entry point of the application, where the server is initialized and configured.
- **init-db.ts:** A script to initialize the database and create necessary tables.
- **database.ts:** Contains the database connection logic and configuration settings.
- **public/:** Contains static files such as CSS and JavaScript files for the frontend.
- **models/:** Contains TypeScript files defining the data models for the application, such as user, account, and transaction models.
- **controllers/:** Contains TypeScript files that handle the business logic and interact with the models.
- **middlewares/:** Contains middleware functions for authentication and logging.
- **routes/:** Contains route definitions for handling HTTP requests.
- **views/:** Contains HTML templates for rendering dynamic content using EJS.
- **deno.json:** Configuration file for Deno, specifying dependencies and runtime options.
- **deno.lock:** Lock file for Deno, ensuring consistent dependency versions.

4.2 Project File Structure

Figure 4.1: Project File Structure

```
project-root
├─ main.ts
├─ database.ts
├─ init-db.ts
├─ public
│  ├─ css
│  └─ js
├─ models
│  ├─ transaction.ts
│  ├─ account.ts
│  └─ user.ts
├─ controllers
│  ├─ transactionController.ts
│  ├─ accountController.ts
│  ├─ userController.ts
│  └─ databaseController.ts
├─ middlewares
│  ├─ authMiddleware.ts
│  └─ loggerMiddleware.ts
├─ routes
│  ├─ transactions.ts
│  ├─ accounts.ts
│  └─ auth.ts
├─ views
│  ├─ dashboard.html
│  ├─ login.html
│  ├─ register.html
│  └─ transfer.html
├─ deno.json
└─ deno.lock
```

Chapter 5

Code Listings

5.1 Entry Point: main.ts

```
// File: main.ts
// @ts-types="npm:@types/express"
import express from "express";
import session from "express-session";
import cors from "cors";
// Routes
import accountRoutes from "./routes/accounts.ts";
import authRoutes from "./routes/auth.ts";
import dashboardRoutes from "./routes/dashboard.ts";
import transactionRoutes from "./routes/transactions.ts";
// Auth
import { authMiddleware } from "./middleware/auth.ts";

// Learn more at https://docs.deno.com/runtime/manual/examples/
   module_metadata#concepts
if (import.meta.main) {
  const app = express();
  const PORT = 3000;

  // Middleware
  app.use(express.json());
  app.use(express.urlencoded({ extended: true }));
  app.use(session({
    secret: "yourSecretKey", // Replace with a strong secret key
    resave: false,
    saveUninitialized: true,
    cookie: {
      secure: false,
      httpOnly: true,
      maxAge: 1000 * 60 * 60 * 4, // 4 hours
    }, // Set to true in production with HTTPS
  }));
  app.use(cors({
    origin: "http://localhost:3000", // specific origin or ’*’ for all
    methods: "GET,POST,PUT,DELETE",
    credentials: true,
  }));

  // Static files and view engine
  app.use(express.static("public"));
  app.set("view engine", "ejs");

  // Route setup
  app.use("/auth", authRoutes);
  app.use("/api/accounts", accountRoutes);
  app.use("/api/transactions", transactionRoutes);
  app.use("/api/dashboard", dashboardRoutes);

  // Index page
  app.get("/", (_req: express.Request, res: express.Response) => {
    res.redirect("/dashboard");
  });

  // Login page
  app.get("/login", (_req: express.Request, res: express.Response) => {
    res.render("login");
  });

  app.get("/register", (_req: express.Request, res: express.Response) => {
    res.render("register");
  });

  // Protected routes
  app.use(authMiddleware);
  app.get("/dashboard", (req: express.Request, res: express.Response) => {
    res.render("dashboard", { user: req.user?.username });
  });

  app.get("/transfer", (_req: express.Request, res: express.Response) => {
    res.render("transfer");
  });

  app.listen(PORT, () => {
    console.log("Server is running on http://localhost:3000");
  });
}
```

5.2 Database Connection: database.ts

```

// File: database.ts
import { Client } from "https://deno.land/x/mysql@v2.12.1/mod.ts
  ";

class Database {
  private static instance: Database;
  private client: Client;

  private constructor() {
    this.client = new Client();
  }

  public static async getInstance(): Promise<Database> {
    if (!Database.instance) {
      Database.instance = new Database();
      await Database.instance.connect();
    }
    return Database.instance;
  }

  private async connect() {
    await this.client.connect({
      hostname: "127.0.0.1",
      username: "root",
      password: "password",
      db: "project",
      poolSize: 3,
    });
  }

  public getClient(): Client {
    return this.client;
  }
}

export default await Database.getInstance();
```

5.3 Database Init-Schema: init-db.ts

```

// File: init-db.ts
import db from "./database.ts";

const client = db.getClient();
await client.execute(`CREATE DATABASE IF NOT EXISTS project`);
await client.execute(`USE project`);

await client.execute(
  `CREATE TABLE IF NOT EXISTS users (
  id VARCHAR(36) PRIMARY KEY,
  fullName VARCHAR(100),
  email VARCHAR(100),
  phone VARCHAR(20),
  address VARCHAR(100),
  city VARCHAR(50),
  state VARCHAR(50),
  zipCode VARCHAR(10),
  username VARCHAR(50),
  password VARCHAR(100),
  createdAt DATETIME,
  accountType ENUM('checking', 'savings', 'both')
  )`,
);

await client.execute(
  `CREATE TABLE IF NOT EXISTS accounts (
  id VARCHAR(36) PRIMARY KEY,
  userId VARCHAR(36),
  type ENUM('checking', 'savings', 'both'),
  balance DECIMAL(10, 2),
  createdAt DATETIME,
  FOREIGN KEY (userId) REFERENCES users(id)
  )`,
);

await client.execute(
  `CREATE TABLE IF NOT EXISTS transactions (
  id VARCHAR(36) PRIMARY KEY,
  fromId VARCHAR(36),
  toId VARCHAR(36),
  type ENUM('deposit', 'withdrawal', 'transfer'),
  amount DECIMAL(10, 2),
  description TEXT,
  balance DECIMAL(10, 2),
  createdAt DATETIME,
  FOREIGN KEY (fromId) REFERENCES accounts(id),
  FOREIGN KEY (toId) REFERENCES accounts(id)
  )`,
);

console.log("Database initialized");
client.close();
```

5.4 Auth Middleware: auth.ts

```
// File: middleware/auth.ts
// @ts-types="npm:@types/express-session"
// @ts-types="npm:@types/express"
import { NextFunction, Request, Response } from "express";

// In your auth.ts middleware file
declare global {
  namespace Express {
    interface Request {
      user?: {
        id: string;
        username: string;
        email: string;
      };
    }
  }
}

// Extend express-session
declare module "express-session" {
  interface SessionData {
    user?: {
      id: string;
      username: string;
      email: string;
    };
  }
}

export const authMiddleware = (
  req: Request,
  res: Response,
  next: NextFunction,
) => {

  // Check if user is logged in
  if (!req.session?.user) {
    // Send a message to the user and then redirect to login page
    console.log("Redirect");
    return res.status(300).redirect("/login");
  }

  // Pass the user object to the next middleware
  req.user = req.session.user; // No need for casting now

  console.log("Passed Authentication");
  res.setHeader("X-Content-Type-Options", "nosniff");
  res.setHeader("X-XSS-Protection", "1; mode=block");
  res.setHeader("X-Frame-Options", "DENY");
  next();
};
```

5.5 Auth Controller: authController.ts

```

// File: controllers/authController.ts
// @ts-types="npm:@types/bcryptjs"
// @ts-types="npm:@types/express"
import { Request, Response } from "express";

import { UserModel } from "../models/user.ts";
import { AccountModel } from "../models/account.ts";
import * as bcrypt from "bcryptjs";

export const authController = {
  register: async (req: Request, res: Response) => {
    console.log("Registering user...");
    try {
      const {
        fullName,
        email,
        phone,
        address,
        city,
        state,
        zipCode,
        username,
        password,
        accountType,
      } = req.body;

      // Validate input
      if (!fullName || !email || !password || !username) {
        return res.status().json({ error: "Missing required
           fields" });
      }

      // Check if user already exists
      const existingUser = await UserModel.findByEmail(email);
      if (existingUser) {
        return res.status(400).json({ error: "User already exists
           " });
      }
      const salt = bcrypt.genSaltSync(10);
      const hashedPassword = bcrypt.hashSync(password, salt);

      // Create user
      const user = await UserModel.create({
        fullName,
        email,
        phone,
        address,
        city,
        state,
        zipCode,
        username,
        password: hashedPassword,
        accountType,
      });

      // Create accounts based on accountType
      await AccountModel.create({
        userId: user.id,
        type: accountType,
        balance: 0,
      });

      // Return success response
      res.status(201).json({
        message: "Account created successfully",
        user: { id: user.id, username: user.username, email: user
           .email },
      });
    } catch (error) {
      console.error("Registration error:", error);
      res.status(500).json({ error: "Server error during
         registration" });
    }
  },

  login: async (req: Request, res: Response) => {
    try {
      console.log("Logging in user...");
      const { username, password } = req.body;

      // Find user
      let user = await UserModel.findByUsername(username);
      if (!user) {
        user = await UserModel.findByEmail(username);
        if (!user) {
          console.log("User not found");
          return res.status(401).json({ error: "Invalid
             credentials" });
        }
      }

      // Verify password (should use proper comparison in real
         implementation)
      const isMatch = await bcrypt.compare(password, user.password);
      if (isMatch === false) {
        return res.status(401).json({ error: "Invalid credentials" });
      }

      req.session.user = {
        id: user.id,
        username: user.username,
        email: user.email,
      };

      console.log("Session ID:", req.session.id, "\nUser ID:", user.id);

      res.status(200).json({
        message: "Login successful",
        user: { id: user.id, username: user.username, email: user.email },
      });
    } catch (error) {
      console.error("Login error:", error);
      res.status(500).json({ error: "Server error during login" });
    }
  },

  logout: (req: Request, res: Response) => {
    // Destroy the session
    console.log(req.session);
    req.session.destroy((err: Error) => {
      if (err) {
        console.error("Logout error:", err);
        return res.status(500).json({ error: "Server error during logout" });
      }
      // Clear the session cookie (using the default name "connect.sid")
      res.clearCookie("connect.sid");
      res.status(200).json({ message: "Logged out successfully" });
    });
    console.log("Session destroyed");
  },
};
```

5.6 Account Controller: accountController.ts

```

// File: controllers/accountController.ts
// @ts-tpes="npm:@types/node"
import { Request, Response } from "npm:express@^4.21.2";
import { AccountModel } from "../models/account.ts";

export const accountController = {
  createAccount: async (req: Request, res: Response) => {
    try {
      const userId = (req as any).user.id; // From auth
         middleware
      const { type } = req.body;

      console.log("Creating an account");
      if (!type) {
        console.log("Type are required");
        return res.status(400).json({ error: "Name and type are
           required" });
      }

      const account = await AccountModel.create({
        userId,
        type,
        balance: 1000.0,
      });
      console.log("Account created:", account);

      res.status(201).json({ account });
    } catch (error) {
      console.error("Create account error:", error);
      res.status(500).json({
        error: "Server error creating account",
        details: (error as Error).message,
      });
    }
  },

  getAllAccounts: async (req: Request, res: Response) => {
    try {
      const userId = (req as any).user.id; // From auth
         middleware
      const accounts = await AccountModel.findByUserId(userId);

      if (!accounts || accounts.length === 0) {
        return res.status(200).json({ accounts: [] });
      }

      res.status(200).json({ accounts });
    } catch (error) {
      console.error("Get accounts error:", error);
      res.status(500).json({
        error: "Server error retrieving accounts",
        details: (error as Error).message,
      });
    }
  },

  getAccount: async (req: Request, res: Response) => {
    try {
      const { accountId } = req.params;
      const userId = (req as any).user.id; // From auth
         middleware

      if (!accountId) {
        return res.status(400).json({ error: "Account ID is
           required" });
      }

      const account = await AccountModel.findById(accountId);

      // Check if account exists and belongs to user
      if (!account || account.userId !== userId) {
        return res.status(404).json({ error: "Account not found"
           });
      }

      res.status(200).json({ account });
    } catch (error) {
      console.error("Get account error:", error);
      res.status(500).json({
        error: "Server error retrieving account",
        details: (error as Error).message,
      });
    }
  },

  getTransactions: async (req: Request, res: Response) => {
    try {
      const { accountId } = req.params;
      const userId = (req as any).user.id; // From auth
         middleware

      if (!accountId) {
        return res.status(400).json({ error: "Account ID is
           required" });
      }

      const account = await AccountModel.findById(accountId);

      // Check if account exists and belongs to user
      if (!account || account.userId !== userId) {
        return res.status(404).json({ error: "Account not found" });
      }

      const transactions = await AccountModel.findTransactions(accountId);

      res.status(200).json({ transactions });
    } catch (error) {
      console.error("Get account transactions error:", error);
      res.status(500).json({
        error: "Server error retrieving account transactions",
        details: (error as Error).message,
      });
    }
  },
};
```

5.7 Dashboard Controller: dashboardController.ts

```

// File: controllers/dashboardController.ts

import { Request, Response } from "express";
import { AccountModel } from "../models/account.ts";
import { TransactionModel } from "../models/transaction.ts";

export const dashboardController = {
  getDashboardData: async (req: Request, res: Response) => {
    console.log("Getting dashboard data...");
    try {
      // deno-lint-ignore no-explicit-any
      const userId = (req as any).user.id; // from auth middleware

      // Get all accounts for the user
      const accounts = await AccountModel.findByUserId(userId);

      // For each account, get recent transactions (limit to 5 for demonstration)
      const accountsWithTransactions = await Promise.all(
        accounts.map(async (account) => {
          const transactions = await TransactionModel.findByAccountId(
            account.id,
          );
          return {
            ...account,
            transactions: transactions.slice(0, 5), // recent 5
              transactions
          };
        }),
      );

      // Calculate a simple financial summary (this can be made
        more detailed)
      let totalBalance = 0;
      accounts.forEach((account) => {
        totalBalance += account.balance;
      });

      res.status(200).json({
        accounts: accountsWithTransactions,
        summary: {
          totalBalance,
          monthlySpending: 1245.62, // Dummy values; compute as
            needed
          monthlyIncome: 3850.0,
          savingsGoal: 10000.0,
          savingsProgress: Math.round((totalBalance / 10000.0) *
            100),
          creditScore: 760,
        },
      });
    } catch (error) {
      console.error("Dashboard data error:", error);
      res.status(500).json({ error: "Server error retrieving
        dashboard data" });
    }
  },
};
```

5.8 Dashboard Routes: dashboard.ts

```

// File: routes/dashboard.ts
import express from "express";
import { dashboardController } from "../controllers/
  dashboardController.ts";
import { authMiddleware } from "../middleware/auth.ts";

const router = express.Router();

router.use((req, res, next) => {
  console.log("Authenticating dashboard");
  authMiddleware(req, res, next);
});
```

```

// Dashboard data endpoint (dynamic data for UI)
router.get("/", async (req, res) => {
  await dashboardController.getDashboardData(req, res);
  if (res.statusCode === 500) {
    res.redirect("/login");
  }
});

export default router;
```

5.9 Auth Routes: auth.ts

```

// File: routes/auth.ts
// @ts-types="npm:@types/express"

import express from "express";
import { authController } from "../controllers/authController.ts
    ";

const router = express.Router();

// Registration endpoint
router.post("/register", async (req: express.Request, res:
    express.Response) => {
     await authController.register(req, res);
});

// Login endpoint
router.post("/login", async (req: express.Request, res: express.
    Response) => {
  await authController.login(req, res);
});

// Logout endpoint
router.post("/logout", authController.logout);

export default router;
```

5.10 Transaction Controller: transactionController.ts

```

// File: controllers/transactionController.ts
// @ts-tpes="npm:@types/node"
import { Request, Response } from "npm:express@^4.21.2";
import { AccountModel } from "../models/account.ts";
import { TransactionModel } from "../models/transaction.ts";
import db from "../database.ts";

export const transactionController = {
  getTransactions: async (req: Request, res: Response) => {
    try {
      const { accountId } = req.params;
      // deno-lint-ignore no-explicit-any
      const userId = (req as any).user.id; // From auth middleware

      if (!accountId) {
        return res.status(400).json({ error: "Account ID is required" });
      }

      const account = await AccountModel.findById(accountId);

      // Check if account exists and belongs to user
      if (!account || account.userId !== userId) {
        return res.status(404).json({ error: "Account not found" });
      }

      const transactions = await TransactionModel.findByAccountId(accountId);

      res.status(200).json({ transactions });
    } catch (error) {
      console.error("Get transactions error:", error);
      res.status(500).json({
        error: "Server error retrieving transactions",
        details: (error as Error).message,
      });
    }
  },

  getTransactionHistory: async (req: Request, res: Response) => {
    const client = db.getClient();
    try {
      const userId = (req as any).user.id;
      const transactions = await client.execute(
        `SELECT * FROM transactions
       WHERE accountId IN (SELECT id FROM accounts WHERE userId = ?)
       AND type = 'transfer'`,
        [userId],
      );
      res.json({ transactions });
    } catch (error) {
      // Handle error
      console.error("Transaction history error:", error);
    }
  },

  createTransaction: async (req: Request, res: Response) => {
    const client = db.getClient();
    try {
      const { fromAccountId, toAccountId, amount, description } = req.body;
      // deno-lint-ignore no-explicit-any
      const userId = (req as any).user.id; // From auth middleware
      const amountNum = Number(amount);


      // Validate input
      if (!fromAccountId || !toAccountId || !amount || amount <= 0) {
        return res.status(400).json({ error: "Invalid transfer details" });
      }

      if (fromAccountId === toAccountId) {
        return res.status(400).json({ error: "Same Sender and Receiver accounts" });
      }

      // Get sender account and check ownership
      const senderAccount = await AccountModel.findById(fromAccountId);
      if (!senderAccount || senderAccount.userId !== userId) {
        return res.status(404).json({ error: "Sender account not found" });
      }

      // Get receiver account
      const receiverAccount = await AccountModel.findById(toAccountId);
      if (!receiverAccount) {
        return res.status(404).json({ error: "Receiver account not found" });
      }

      // Check for sufficient funds
      if (senderAccount.balance < amount) {
        return res.status(400).json({
          error: "Insufficient funds for transfer",
        });
      }

      console.log("Creating transaction...");
      console.log("From Account ID:", fromAccountId);
      console.log("To Account ID:", toAccountId);
      console.log("Amount:", amountNum);
      console.log("Description:", description);
      console.log("User ID:", userId);

      client.execute("START TRANSACTION");
      const newReceiverBalance = Number(receiverAccount.balance) + amountNum;
      const newSenderBalance = Number(senderAccount.balance) - amountNum;
      console.log(newSenderBalance);
      console.log(newReceiverBalance);

      // Update sender balance
      await AccountModel.updateBalance(fromAccountId, newSenderBalance);

      // Update receiver balance
      await AccountModel.updateBalance(toAccountId, newReceiverBalance);

      // Create a new transaction
      const Transaction = await TransactionModel.create({
        fromId: fromAccountId,
        toId: toAccountId,
        type: "transfer",
        amount,
        description,
        balance: newSenderBalance,
      });
      client.execute("COMMIT");

      res.status(201).json({
        message: "Transfer completed successfully",
        Transaction,
        newSenderBalance,
        newReceiverBalance,
      });
    } catch (error) {
      client.execute("ROLLBACK");
      console.error("Transfer error:", error);
      res.status(500).json({ error: "Server error processing transfer" });
    }
  },
};
```

5.11 Account Routes: accounts.ts

```

// File: routes/accounts.ts
import express from "express";
import { authMiddleware } from "../middleware/auth.ts";
import { accountController } from "../controllers/
   accountController.ts";

const router = express.Router();

// Apply auth middleware to all account routes
router.use((req, res, next) => {
  authMiddleware(req, res, next);
});

router.get("/", async (req, res) => {
  await accountController.getAllAccounts(req, res);
});

router.get("/create", async (req, res) => {
  await accountController.createAccount(req, res);
});

// Get account details
router.get("/:accountId", async (req, res) => {
  await accountController.getAccount(req, res);
});

// Get account transactions
router.get("/:accountId/transactions", async (req, res) => {
  await accountController.getTransactions(req, res);
});

export default router;
```

5.12 Transaction Routes: transactions.ts

```
// File: routes/transaction.ts
import express from "express";
import { authMiddleware } from "../middleware/auth.ts";
import { transactionController } from "../controllers/
   transactionController.ts";

const router = express.Router();

// Apply auth middleware to all transaction routes
router.use((req, res, next) => {
  console.log("Authenticating new transaction");
  authMiddleware(req, res, next);
});

router.post("/", async (req, res) => {
  await transactionController.createTransaction(req, res);
});

router.post("/history", async (req, res) => {
  await transactionController.getTransactionHistory(req, res);
});

router.post("/:accountId", async (req, res) => {
  await transactionController.getTransactions(req, res);
});

export default router;
```

5.13 User Model: user.ts

```

// File: models/user.ts
// This is a simple in-memory model for demonstration
// In a real application, you'd use a database

import { AccountModel, AccountType } from "./account.ts";
import db from "../database.ts";

interface User {
  id: string;
  fullName: string;
  email: string;
  phone: string;
  address: string;
  city: string;
  state: string;
  zipCode: string;
  username: string;
  password: string;
  createdAt: Date;
  accountType: AccountType;
}

const client = db.getClient();

export const UserModel = {
  create: async (userData: Omit<User, "id" | "createdAt">):
     Promise<User> => {
    // Check if user already exists
    const users = await client.execute(
      "SELECT * FROM users WHERE email = ? or username = ?",
      [
        userData.email,
        userData.username,
      ],
    );

    if (users.rows == undefined) {
      console.log("SQL Query Error");
      throw new Error("SQL Query Error");
    }
    if (users.rows.length > 0) {
      console.log("User already exists");
      return users.rows[0] as User;
    }

    let id = 0;
    const count = await client.query(
      "SELECT COUNT(*) FROM users",
    );
    if (count.rows !== undefined) {
      id = count.rows[0]["COUNT(*)"] + 1;
    }
    const newUser: User = {
      ...userData,
      accountType: userData.accountType as AccountType,
      id: String(id),
      createdAt: new Date(),
    };

    await client.execute(
      "INSERT INTO users (id, fullName, email, phone, address,
         city, state, zipCode, username, password, createdAt,
         accountType) \
      VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
      [
        newUser.id,
        newUser.fullName,
        newUser.email,
        newUser.phone,
        newUser.address,
        newUser.city,
        newUser.state,
        newUser.zipCode,
        newUser.username,
        newUser.password,
        newUser.createdAt,
        newUser.accountType,
      ],
    );

    // Add an account for the user
    await AccountModel.create({
      userId: newUser.id,
      type: newUser.accountType,
      balance: 1000.0,
    });

    return newUser;
  },

  findByEmail: async (email: string): Promise<User | undefined> => {
    const users = await client.execute("SELECT * FROM users WHERE email = ?", [
      email,
    ]);
    if (users.rows == undefined) {
      return undefined;
    }

    return users.rows.length > 0 ? users.rows[0] : undefined;
  },

  findByUsername: async (username: string): Promise<User | undefined> => {
    const users = await client.execute(
      "SELECT * FROM users WHERE username = ?",
      [
        username,
      ],
    );
    if (users.rows == undefined) {
      return undefined;
    }
    return users.rows[0] as User;
  },

  findById: async (id: string): Promise<User | undefined> => {
    const users = await client.execute("SELECT * FROM users WHERE id = ?", [
      id,
    ]);
    if (users.rows == undefined) {
      return undefined;
    }
    return users.rows[0] as User;
  },
};
```

5.14 Transaction Model: transaction.ts

```

// File: models/transaction.ts

import db from "../database.ts";

export enum TransactionType {
  Deposit = "deposit",
  Withdrawal = "withdrawal",
  Transfer = "transfer",
}

export interface Transaction {
  id: string;
  fromId: string;
  toId: string;
  type: "deposit" | "withdrawal" | "transfer";
  amount: number;
  description: string;
  balance: number;
  createdAt: Date;
}

// In-memory storage
const client = db.getClient();

export const TransactionModel = {
  create: async (
    transactionData: Omit<Transaction, "id" | "createdAt">,
  ): Promise<Transaction> => {
    let id = 0;
    const count = await client.execute(
      "SELECT COUNT(*) FROM transactions",
    );
    if (count.rows !== undefined) {
      id = count.rows[0]["COUNT(*)"] + 1;
    }

    const newTransaction: Transaction = {
      ...transactionData,
      createdAt: new Date(),
      id: String(id),
    };

    await client.execute(
      "INSERT INTO transactions (id, toId, fromId, type, amount,
         description, balance, createdAt) VALUES (?, ?, ?, ?, ?,
         ?, ?, ?)",
      [
        newTransaction.id,
        newTransaction.fromId,
        newTransaction.toId,
        newTransaction.type,
        newTransaction.amount,
        newTransaction.description,
        newTransaction.balance,
        newTransaction.createdAt,
      ],
    );
    return newTransaction;
  },

  findById: async (id: string): Promise<Transaction | undefined>
     => {
    const transactions = await client.execute(
      "SELECT * FROM transactions WHERE id = ?",
      [id],
    );
    if (transactions.rows == undefined) {
      throw new Error("Account not found");
    }

    return transactions.rows.length > 0 ? transactions.rows[0] :
       undefined;
  },

  findByUserId: async (userId: string): Promise<Transaction[]> =>
     {
    const transactions = await client.execute(
      "SELECT * FROM transactions WHERE userId = ?",
      [userId],
    );

    if (transactions.rows == undefined) {
      throw new Error("Account not found");
    }

    return transactions.rows as Transaction[];
  },

  findByAccountId: async (accountId: string): Promise<Transaction
     []> => {
    const transactions = await client.execute(
      "SELECT * FROM transactions WHERE fromId = ?",
      [accountId],
    );
    if (transactions.rows == undefined) {
      throw new Error("Account not found");
    }

    return transactions.rows as Transaction[];
  },
};
```

5.15 Account Model: account.ts

```

// File: models/account.ts

import db from "../database.ts";
import { Transaction } from "./transaction.ts";

export enum AccountType {
  Checking = "checking",
  Savings = "savings",
  Both = "both",
}

export interface Account {
  id: string;
  userId: string;
  type: AccountType;
  balance: number;
  createdAt: Date;
}

const client = db.getClient();

export const AccountModel = {
  create: async (
    accountData: Omit<Account, "id" | "createdAt">,
  ): Promise<Account | undefined> => {
    try {
      let id = 0;
      const count = await client.execute(
        "SELECT COUNT(*) FROM accounts",
      );
      if (count.rows !== undefined) {
        id = count.rows[0]["COUNT(*)"] + 1;
      }

      const newAccount: Account = {
        ...accountData,
        id: String(id),
        createdAt: new Date(),
      };

      // Check if user already exists
      const accounts = await client.execute(
        "SELECT * FROM accounts WHERE id = ?",
        [newAccount.id],
      );

      if (accounts.rows == undefined) {
        console.log("SQL Query Error");
        return newAccount;
      }
      if (accounts.rows.length > 0) {
        console.log("Account already exists");
        return accounts.rows[0] as Account;
      }

      await client.execute(
        "INSERT INTO accounts (id, userId, type, balance,
           createdAt) VALUES (?, ?, ?, ?, ?)",
        [
          newAccount.id,
          newAccount.userId,
          newAccount.type,
          newAccount.balance,
          newAccount.createdAt,
        ],
      );
      console.log("Account created");

      return newAccount;
    } catch (error) {
      console.error("Account Creation Error: ", (error as Error).
         message);
      throw new Error("Server error creating account");
    }
  },

  listAll: async (): Promise<Account[]> => {
    const accounts = await client.execute("SELECT * FROM accounts
       ");

    if (accounts.rows == undefined) {
      return [];
    }

    return accounts.rows as Account[];
  },

  findById: async (id: string): Promise<Account | undefined> => {
    const accounts = await client.execute(
      "SELECT * FROM accounts WHERE id = ?",
      [id],
    );

    if (accounts.rows == undefined) {
      throw new Error("Account not found");
    }

    return accounts.rows.length > 0 ? accounts.rows[0] :
       undefined;
  },

  findByUserId: async (userId: string): Promise<Account[]> => {
    const accounts = await client.execute(
      "SELECT * FROM accounts WHERE userId = ?",
      [userId],
    );

    if (accounts.rows == undefined) {
      return [];
    }

    return accounts.rows as Account[];
  },

  findTransactions: async (id: string): Promise<Transaction[]> =>
      {
    const transactions = await client.execute(
      "SELECT * FROM transactions WHERE fromId = ?",
      [id],
    );

    if (transactions.rows == undefined) {
      return [];
    }

    return transactions.rows as Transaction[];
  },

  updateBalance: async (
    id: string,
    newBalance: number,
  ): Promise<Account | undefined> => {
    console.log(
      "UPDATE accounts SET balance = " + newBalance + " WHERE id
         = " + id,
    );
    await client.execute(
      "UPDATE accounts SET balance = ? WHERE id = ?",
      [newBalance, id],
    );

    return AccountModel.findById(id);
  },
};
```

5.16 Dashboard View: dashboard.ejs

```

<!--// File: views/dashboard.ejs-->
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0">
    <title>E(xpress)-Bank Dashboard</title>
    <!-- Bootstrap CSS from CDN -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap
       /5.3.0/css/bootstrap.min.css" rel="stylesheet">
    <!-- Font Awesome for icons -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
       awesome/6.4.0/css/all.min.css" rel="stylesheet">
    <!-- Custom CSS -->
    <link rel="stylesheet" href="/css/dashboard.css" id='
       dashboard.css'>
    <link rel="stylesheet" href="/css/styles.css" id='dashboard.
       css'>
</head>

<body>
    <!-- Sidebar -->
    <div class="sidebar d-none d-lg-block">
        <div class="bank-logo">E(xpress)-Bank</div>
        <ul class="nav flex-column">
            <li class="nav-item">
                <a class="nav-link active" href="#"><i class="fas
                    fa-home"></i> Dashboard</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/transfer"><i class="
                   fas fa-exchange-alt"></i> Transfers</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#"><i class="fas fa-cog
                   "></i> Settings</a>
            </li>
            <li class="nav-item mt-5">
                <a class="nav-link text-danger" id="logoutButton"
                   > <i class="fas fa-sign-out-alt" onclick="
                   logout()"></i> Logout</a>
            </li>
        </ul>
    </div>

    <!-- Main Content -->
    <div class="main-content">
        <!-- Header with user greeting -->
        <div class="d-flex justify-content-between align-items-
           center mb-4">
            <div class="user-greeting">
                    Welcome back, <span class="user-name" id="
                       userName"><%- user %></span>!
            </div>
            <a href="/details" class="profile-button" id="
               profileInitials">U</a>
        </div>

        <!-- Accounts Overview Section -->
        <h5 class="mb-4">Your Accounts</h5>
        <div class="row" id="accountsContainer"> </div>

        <!-- Quick Stats Section -->
    <!--     <h5 class="mb-4 mt-4">Financial Summary</h5> -->
    <!--     <div class="row"> -->
            <!-- Monthly Spending -->
    <!--         <div class="col-md-6 col-lg-3 mb-4"> -->
    <!--             <div class="stats-card"> -->
    <!--                 <div class="stat-label">Monthly Spending
       </div> -->
    <!--                 <div class="stat-value">$1,245.62</div>
       -->
    <!--                 <div class="text-muted small"> -->
    <!--                     <i class="fas fa-arrow-down text-
       success"></i> 12% from last month -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--         </div> -->
    <!---->
            <!-- Income -->
    <!--         <div class="col-md-6 col-lg-3 mb-4"> -->
    <!--             <div class="stats-card"> -->
    <!--                 <div class="stat-label">Monthly Income</
       div> -->
    <!--                 <div class="stat-value">$3,850.00</div>
       -->
    <!--                 <div class="text-muted small"> -->
    <!--                     <i class="fas fa-arrow-up text-
       success"></i> 5% from last month -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--         </div> -->
    <!---->
```

```
            <!-- Savings Goal -->
    <!--         <div class="col-md-6 col-lg-3 mb-4"> -->
    <!--             <div class="stats-card"> -->
    <!--                 <div class="stat-label">Savings Goal</div> -->
    <!--                 <div class="stat-value">$10,000.00</div> -->
    <!--                 <div class="text-muted small">78% complete</div> -->
    <!--                 <div class="progress savings-progress"> -->
    <!--                     <div class="progress-bar bg-success" role="progressbar" style="width: 78%" aria-valuenow="78" -->
    <!--                         aria-valuemin="0" aria-valuemax="100"></div> -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--         </div> -->
    <!---->
            <!-- Credit Score -->
    <!--         <div class="col-md-6 col-lg-3 mb-4"> -->
    <!--             <div class="stats-card"> -->
    <!--                 <div class="stat-label">Credit Score</div> -->
    <!--                 <div class="stat-value">760</div> -->
    <!--                 <div class="text-muted small"> -->
    <!--                     <i class="fas fa-arrow-up text-success"></i> Excellent -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--         </div> -->
    <!--     </div> -->
    <!---->
        <!-- Recent Transactions Section -->
    <!--     <h5 class="mb-4">Recent Transactions</h5> -->
    <!--     <div class="transaction-list"> -->
    <!--         <div class="list-group list-group-flush" id="transactionsList"> -->
                <!-- Transaction items will be populated dynamically -->
    <!--             <div class="transaction-item deposit"> -->
    <!--                 <div class="d-flex justify-content-between align-items-center"> -->
    <!--                     <div> -->
    <!--                         <div class="fw-bold">Deposit: Payroll</div> -->
    <!--                         <div class="text-muted small">Feb 25, 2025</div> -->
    <!--                     </div> -->
    <!--                     <div class="transaction-amount deposit">+$1,850.00</div> -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--             <div class="transaction-item withdrawal"> -->
    <!--                 <div class="d-flex justify-content-between align-items-center"> -->
    <!--                     <div> -->
    <!--                         <div class="fw-bold">Walmart</div> -->
    <!--                         <div class="text-muted small">Feb 23, 2025</div> -->
    <!--                     </div> -->
    <!--                     <div class="transaction-amount withdrawal">-$87.45</div> -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--             <div class="transaction-item withdrawal"> -->
    <!--                 <div class="d-flex justify-content-between align-items-center"> -->
    <!--                     <div> -->
    <!--                         <div class="fw-bold">Amazon Prime</div> -->
    <!--                         <div class="text-muted small">Feb 21, 2025</div> -->
    <!--                     </div> -->
    <!--                     <div class="transaction-amount withdrawal">-$14.99</div> -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--             <div class="transaction-item withdrawal"> -->
    <!--                 <div class="d-flex justify-content-between align-items-center"> -->
    <!--                     <div> -->
    <!--                         <div class="fw-bold">Uber Ride</div> -->
    <!--                         <div class="text-muted small">Feb 20, 2025</div> -->
    <!--                     </div> -->
    <!--                     <div class="transaction-amount withdrawal">-$24.50</div> -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--             <div class="transaction-item deposit"> -->
    <!--                 <div class="d-flex justify-content-between align-items-center"> -->
    <!--                     <div> -->
    <!--                         <div class="fw-bold">Transfer from Savings</div> -->
    <!--                         <div class="text-muted small">Feb 18, 2025</div> -->
    <!--                     </div> -->
    <!--                     <div class="transaction-amount deposit">+$500.00</div> -->
    <!--                 </div> -->
    <!--             </div> -->
    <!--         </div> -->
    <!--         <div class="p-3 text-center"> -->
    <!--             <button class="btn btn-link text-decoration-none">View All Transactions</button> -->
    <!--         </div> -->
    <!--     </div> -->
    <!-- </div> -->

    <!-- Quick Action Button -->
    <div class="quick-actions">
        <div class="dropdown">
            <button class="quick-action-btn" type="button" id="quickActionsDropdown" data-bs-toggle="dropdown"
                aria-expanded="false">
                <i class="fas fa-plus"></i>
            </button>
            <ul class="dropdown-menu dropdown-menu-end" aria-labelledby="quickActionsDropdown">
                <li><a class="dropdown-item" href="#"><i class="fas fa-exchange-alt me-2"></i> New Transfer</a></li>
                <li><a class="dropdown-item" href="#"><i class="fas fa-credit-card me-2"></i> Pay Bill</a></li>
                <li><a class="dropdown-item" href="#"><i class="fas fa-mobile-alt me-2"></i> Mobile Deposit</a></li>
                <li><a class="dropdown-item" href="#"><i class="fas fa-user-friends me-2"></i> Send to Friend</a></li>
            </ul>
        </div>
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.min.js"></script>

    <!-- Dashboard Script -->
    <script src="/js/dashboard.cjs"></script>
    <script>
        // Logout functionality
        async function logout() {
            console.log("Logging out...");
            try {
                const response = await fetch("/auth/logout", {
                    method: "POST",
                    headers: {
                        "Content-Type": "application/json",
                    },
                });
                console.log("Logout response:", response);

                if (response.ok) {
                    // Redirect to the login page after
                       successful logout
                    globalThis.window.location.href = "/login";
                } else {
                    console.error("Logout failed");
                }
            } catch (error) {
                console.error("Error during logout:", error);
                globalThis.window.location.href = "/login";
            }
        }
    </script>

</body>

</html>
```

5.17 Login View: login.ejs

```

<!--// File: views/login.ejs-->
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0">
    <title>E(xpress)-Bank Login</title>
    <!-- Bootstrap CSS from CDN -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap
       /5.3.0/css/bootstrap.min.css" rel="stylesheet">
    <link href="/css/styles.css" rel="stylesheet">
</head>

<body class="bg-light">
    <div class="container py-5">
```

```
        <div class="card shadow form-container">
            <div class="bank-header">
                <div class="bank-logo">E(xpress)-Bank</div>
                <h2 class="mt-2">Log In</h2>
                <p class="text-muted">Access your secure banking account</p>
            </div>

            <form id="loginForm">
                <div class="mb-3">
                    <label for="username" class="form-label">Username</label>
                    <input type="text" class="form-control" id="username" required>
                </div>

                <div class="mb-3">
                    <label for="password" class="form-label">Password</label>
                    <input type="password" class="form-control" id="password" required>
                </div>

                <div class="mb-3 form-check">
                    <input type="checkbox" class="form-check-input" id="rememberMe">
                    <label class="form-check-label" for="rememberMe">Remember me</label>
                </div>

                <button type="submit" class="btn btn-primary w-100">Log In</button>

                <div class="text-center mt-3">
                    <!-- <p><a href="#" class="text-decoration-none">Forgot password?</a></p> -->
                    <p>Don't have an account? <a href="/register" class="text-decoration-none">Sign up</a></p>
                </div>
            </form>
        </div>
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.min.js"></script>

    <!-- Login Script -->
    <script lang="javascript">
```

```
        document.getElementById('loginForm').addEventListener('submit', async function (e) {
            e.preventDefault();

            const username = document.getElementById('username').value;
            const password = document.getElementById('password').value;

            try {
                const response = await fetch('/auth/login', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({username, password})
                });

                const data = await response.json();

                if (response.ok) {
                    // Session cookie is set automatically, so simply redirect to dashboard
                    window.location.href = '/dashboard';
                } else {
                    alert(data.error || 'Login failed');
                }
            } catch (error) {
                console.error('Login error:', error);
                alert('An error occurred during login');
            }
        });
    </script>

</body>

</html>
```

5.18 Register View: register.ejs

```

<!--// File: views/register.ejs-->
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E(xpress)-Bank</title>
    <!-- Bootstrap CSS from CDN -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap
       /5.3.0/css/bootstrap.min.css" rel="stylesheet">
    <link href="/css/styles.css" rel="stylesheet">
</head>

<body class="bg-light">
    <div class="container py-5">
        <div class="card shadow form-container">
            <div class="bank-header text-center">
                <div class="bank-logo">E(xpress)-Bank</div>
                <h2 class="mt-2">Create an Account</h2>
                <p class="text-muted">Join our secure banking
                   platform</p>
            </div>
            <form id="registrationForm">
                <!-- Personal Information -->
                <fieldset class="mb-4">
                    <legend>Personal Information</legend>
                    <div class="mb-3">
                        <label for="fullName" class="form-label">
                           Full Name</label>
                        <input type="text" class="form-control"
                           id="fullName" required>
                    </div>
                    <div class="row mb-3">
                        <div class="col">
                            <label for="email" class="form-label"
                               >Email</label>
                            <input type="email" class="form-
                               control" id="email" required>
                        </div>
                        <div class="col">
                            <label for="phone" class="form-label"
                               >Phone Number</label>
                            <input type="tel" class="form-control
                               " id="phone" required>
                        </div>
                    </div>
                </fieldset>
                <!-- Address Information -->
                <fieldset class="mb-4">
                    <legend>Address Information</legend>
                    <div class="mb-3">
                        <label for="address" class="form-label">
                           Address</label>
                        <input type="text" class="form-control"
                           id="address" required>
                    </div>
                    <div class="row mb-3">
```

```html
                        <div class="col">
                            <label for="city" class="form-label">City</label>
                            <input type="text" class="form-control" id="city" required>
                        </div>
                        <div class="col">
                            <label for="state" class="form-label">State</label>
                            <input type="text" class="form-control" id="state" required>
                        </div>
                        <div class="col">
                            <label for="zipCode" class="form-label">Zip Code</label>
                            <input type="text" class="form-control" id="zipCode" required>
                        </div>
                    </div>
                </fieldset>
                <!-- Account Security -->
                <fieldset class="mb-4">
                    <legend>Account Security</legend>
                    <div class="mb-3">
                        <label for="username" class="form-label">Username</label>
                        <input type="text" class="form-control" id="username" required>
                    </div>
                    <div class="mb-3">
                        <label for="password" class="form-label">Password</label>
                        <input type="password" class="form-control" id="password" required>
                        <div class="password-requirements mt-1">
                            Password must be at least 8 characters and include uppercase, lowercase, number, and special character.
                        </div>
                    </div>
                    <div class="mb-3">
                        <label for="confirmPassword" class="form-label">Confirm Password</label>
                        <input type="password" class="form-control" id="confirmPassword" required>
                    </div>
                </fieldset>
                <!-- Account Type Selection -->
                <fieldset class="mb-4">
```

```html
                    <legend>Account Type</legend>
                    <div class="form-check">
                        <input class="form-check-input" type="radio" name="accountType" id="checkingAccount" value="checking" checked>
                        <label class="form-check-label" for="checkingAccount">Checking Account</label>
                    </div>
                    <div class="form-check">
                        <input class="form-check-input" type="radio" name="accountType" id="savingsAccount" value="savings">
                        <label class="form-check-label" for="savingsAccount">Savings Account</label>
                    </div>
                    <div class="form-check">
                        <input class="form-check-input" type="radio" name="accountType" id="bothAccounts" value="both">
                        <label class="form-check-label" for="bothAccounts">Both Checking & Savings</label>
                    </div>
                </fieldset>
                <!-- Terms and Conditions -->
                <div class="mb-4">
                    <div class="form-check">
                        <input class="form-check-input" type="checkbox" id="termsAgreement" required>
                        <label class="form-check-label" for="termsAgreement">
                            I agree to the <a href="#" data-bs-toggle="modal" data-bs-target="#termsModal">Terms and Conditions</a>
                        </label>
                    </div>
                </div>
                <!-- Submit Button -->
                <button type="submit" class="btn btn-primary w-100">Create Account</button>
                <div class="text-center mt-3">
                    <p>Already have an account? <a href="/login">Log in</a></p>
                </div>
            </form>
        </div>
```

```
    </div>
    <!-- Bootstrap JS from CDN -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap
       /5.3.0/js/bootstrap.bundle.min.js"></script>
    <!-- Custom JS -->
    <script src="/js/registration.cjs"></script>
</body>

</html>
```

5.19 Transfers View: transfers.ejs

```

<!--// File: views/transfer.ejs-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0">
    <title>Transfer Funds - E-Bank</title>
    <!-- Bootstrap CSS from CDN -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap
       /5.3.0/css/bootstrap.min.css" rel="stylesheet">
    <!-- Font Awesome for icons -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
       awesome/6.4.0/css/all.min.css" rel="stylesheet">
    <link rel="stylesheet" href="/css/dashboard.css">
    <link rel="stylesheet" href="/css/styles.css">
</head>
<body>
    <!-- Sidebar -->
    <div class="sidebar d-none d-lg-block">
        <div class="bank-logo">E(xpress)-Bank</div>
        <ul class="nav flex-column">
            <li class="nav-item">
                <a class="nav-link" href="/dashboard"><i class="
                   fas fa-home"></i> Dashboard</a>
            </li>
            <li class="nav-item">
                <a class="nav-link active" href="/transfer"><i
                   class="fas fa-exchange-alt"></i> Transfers</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#"><i class="fas fa-
                   credit-card"></i> Cards</a>
            </li>
            <li class="nav-item">
```

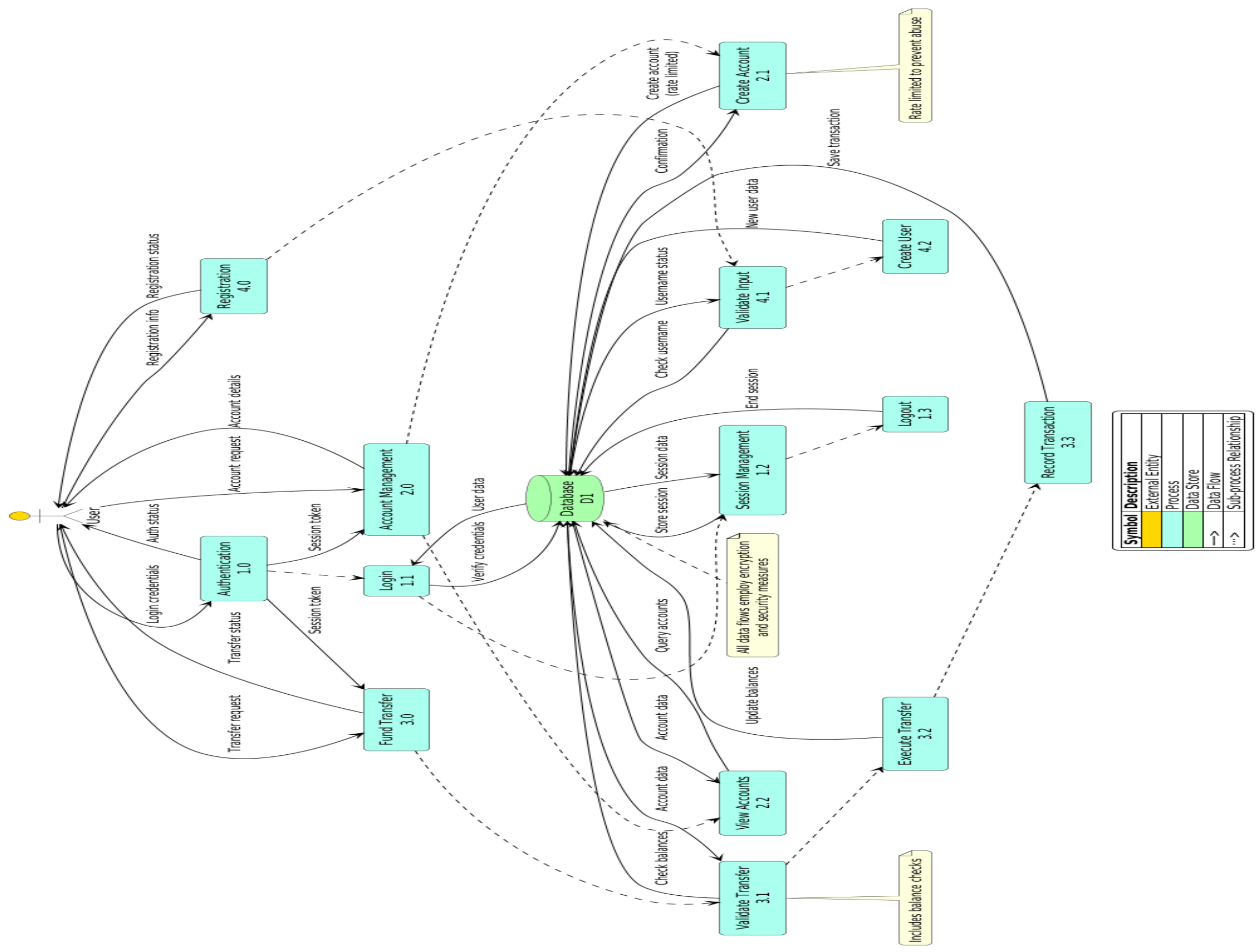
```html
                <a class="nav-link" href="#"><i class="fas fa-chart-pie"></i> Investments</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#"><i class="fas fa-cog"></i> Settings</a>
            </li>
            <li class="nav-item mt-5">
                <a class="nav-link text-danger" id="logoutButton"> <i class="fas fa-sign-out-alt"></i> Logout</a>
            </li>
        </ul>
    </div>

    <!-- Main Content -->
    <div class="main-content">
        <!-- Header with title -->
        <div class="d-flex justify-content-between align-items-center mb-4">
            <h4>Transfer Funds</h4>
        </div>

        <div class="row">
            <div class="col-md-8">
                <div class="card shadow-sm mb-4">
                    <div class="card-body">
                        <h5 class="card-title mb-3">New Transfer</h5>
                        <form id="transfer-form">
                            <div class="mb-3">
                                <label for="fromAccount" class="form-label">From Account</label>
                                <select class="form-select" id="fromAccount" name="fromAccount" required>
                                    <option value="">Select account</option>
                                </select>
                            </div>

                            <div class="mb-3">
                                <label for="toAccount" class="form-label">To Account</label>
                                <input type="text" class="form-select" id="toAccount" name="fromAccount" required>
                            </div>

                            <div class="mb-3">
```

```
                                <label for="amount" class="form-
                                   label">Amount</label>
                                <div class="input-group">
                                    <span class="input-group-text
                                       ">$</span>
                                    <input type="number" class="
                                       form-control" id="amount"
                                       name="amount" min="0.01"
                                       step="0.01" required>
                                </div>
                            </div>

                            <div class="mb-3">
                                <label for="note" class="form-
                                   label">Note (optional)</label>
                                <input type="text" class="form-
                                   control" id="note" name="note"
                                    placeholder="What's this
                                   transfer for?">
                            </div>

                            <button type="submit" class="btn btn-
                               primary">Transfer Funds</button>
                        </form>
                    </div>
                </div>
            </div>

            <div class="col-md-4">
                <div class="card shadow-sm">
                    <div class="card-body">
                        <h5 class="card-title mb-3">Transfer Tips
                           </h5>
                        <ul class="small text-muted">
                            <li>Transfers between your accounts
                               are instant</li>
                            <li>You can transfer up to $10,000
                               per day</li>
                            <li>Adding a note helps you track
                               transfers later</li>
                            <li>Transfers to external accounts
                               may take 1-3 business days</li>
                        </ul>
                    </div>
                </div>
            </div>
        </div>

        <div class="row mt-4">
            <div class="col-12">
                <div class="card shadow-sm">
```

```
                    <div class="card-body">
                        <h5 class="card-title mb-3">Recent Transfers</h5>
                        <div id="transfer-history" class="transaction-list">
                            <!-- Transfer history will be loaded here -->
                            <p class="text-center text-muted">Loading transfer history...</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- Bootstrap JS -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.min.js"></script>
    <!-- Transfer Script -->
    <script src="/js/transfers.cjs"></script>

    <script>
        // Logout functionality
        document.getElementById('logoutButton').addEventListener('click', async function() {
            try {
                const response = await fetch("/auth/logout", {
                    method: "POST",
                    headers: {
                        "Content-Type": "application/json",
                    },
                });

                if (response.ok) {
                    window.location.href = "/login";
                } else {
                    console.error("Logout failed");
                }
            } catch (error) {
                console.error("Error during logout:", error);
                window.location.href = "/login";
            }
        });
    </script>
</body>
</html>
```

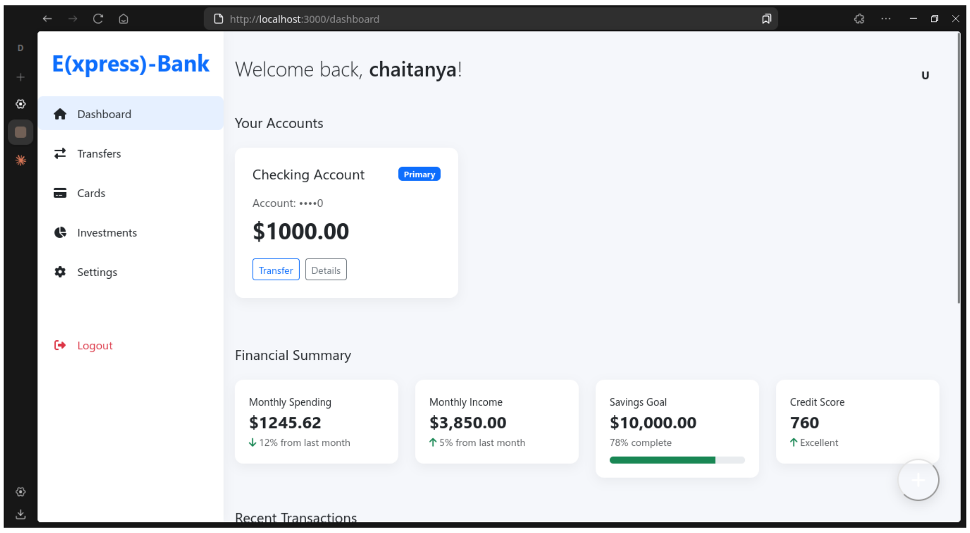
Chapter 6

Data Flow Diagram



Chapter 7

Outputs



http://localhost:3000/dashboard

E(xpress)-Bank

Welcome back, **chaitanya**!

U

- Dashboard
- Transfers
- Cards
- Investments
- Settings
- Logout

Your Accounts

Checking Account Primary

Account: ••••0

$1000.00

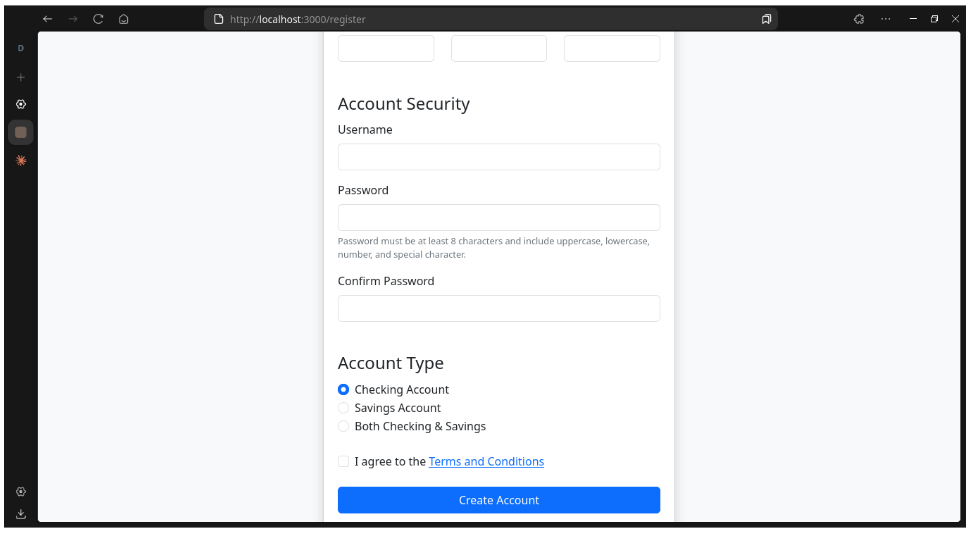
Transfer Details

Financial Summary

Monthly Spending	Monthly Income	Savings Goal	Credit Score
$1245.62	$3,850.00	$10,000.00	760
↓ 12% from last month	↑ 5% from last month	78% complete	↑ Excellent

Recent Transactions

Figure 7.1: Dashboard Interface



http://localhost:3000/register

Account Security

Username

Password

Password must be at least 8 characters and include uppercase, lowercase, number, and special character.

Confirm Password

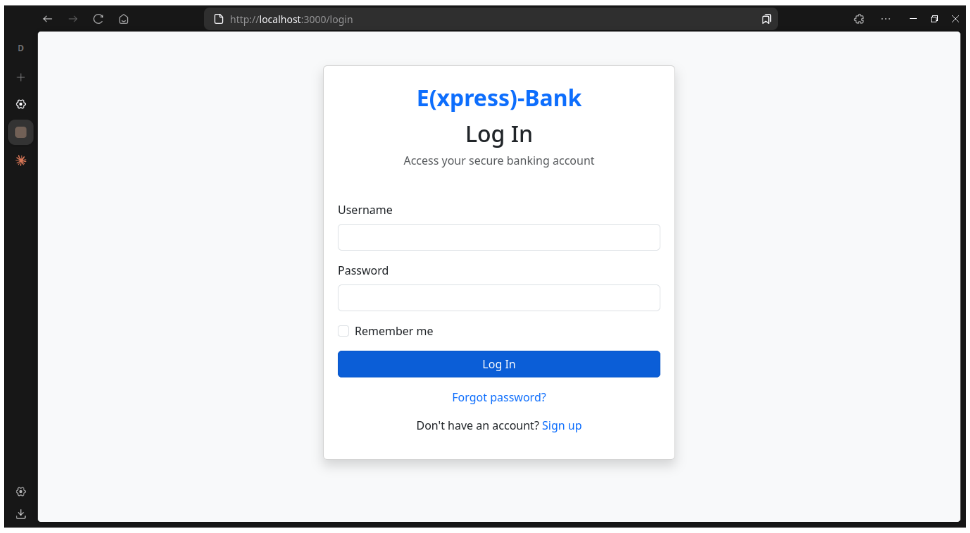
Account Type

- [✓ Checking Account]
- Savings Account
- Both Checking & Savings

☐ I agree to the Terms and Conditions

Create Account

Figure 7.2: User Sign Up Interface


http://localhost:3000/login

E(xpress)-Bank

Log In

Access your secure banking account

Username

Password

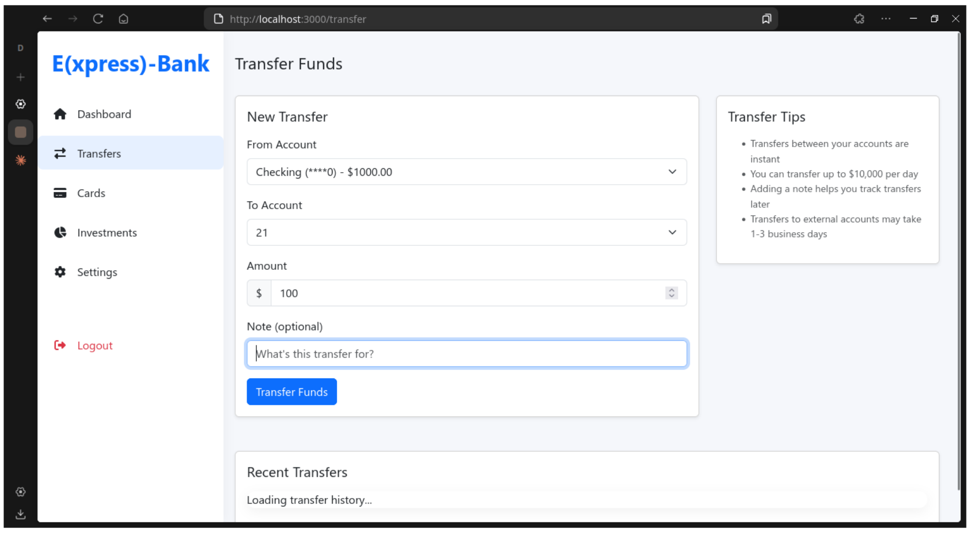
Remember me

Log In

Forgot password?

Don't have an account? Sign up

Figure 7.3: User Account Login Interface


http://localhost:3000/transfer

E(xpress)-Bank

Dashboard

Transfers

Cards

Investments

Settings

Logout

Transfer Funds

New Transfer

From Account

Checking (****0) - $1000.00

To Account

21

Amount

$ 100

Note (optional)

What's this transfer for?

Transfer Funds

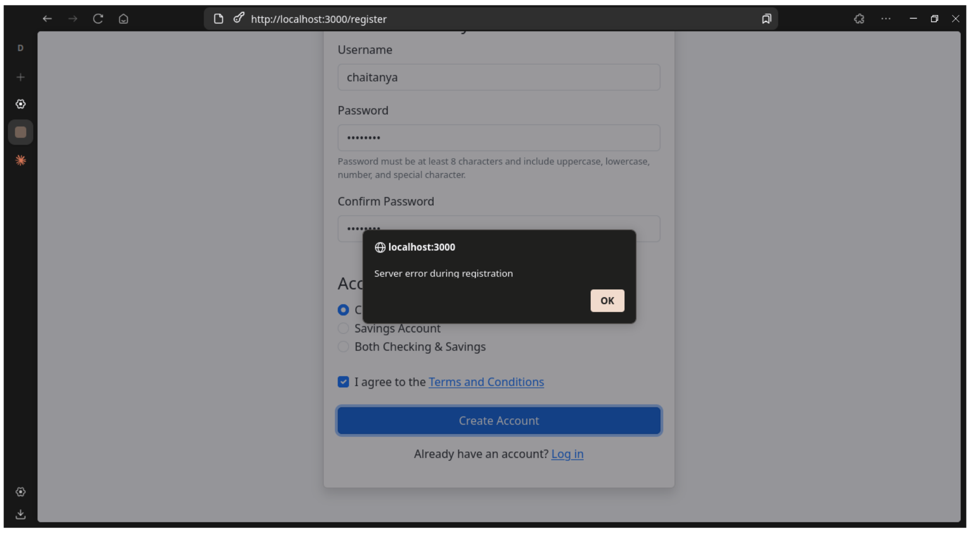
Transfer Tips

- Transfers between your accounts are instant
- You can transfer up to $10,000 per day
- Adding a note helps you track transfers later
- Transfers to external accounts may take 1-3 business days

Recent Transfers

Loading transfer history...

Figure 7.4: Transaction Interface


http://localhost:3000/register

Username

chaitanya

Password

••••••••

Password must be at least 8 characters and include uppercase, lowercase, number, and special character.

Confirm Password

••••••••

localhost:3000

Server error during registration

OK

Savings Account

Both Checking & Savings

I agree to the Terms and Conditions

Create Account

Already have an account? Log in

Figure 7.5: Error Messages

Chapter 8

References

1. https://stackoverflow.com/questions/5132923/how-to-add-a-primary-key-to-a-mysql-table
2. https://stackoverflow.com/questions/9070764/insert-auto-increment-primary-key-to-existing-table
3. https://stackoverflow.com/questions/5813771/in-ejs-template-engine-how-do-i-include-a-footer
4. https://stackoverflow.com/questions/22212440/using-math-class-to-convert-cosine-into-degrees
5. https://www.geeksforgeeks.org/how-to-connect-node-js-application-to-mysql/
6. https://stackoverflow.com/questions/13396119/how-to-add-a-value-to-existing-value-using-sql-query
7. https://stackoverflow.blog/css/

Bibliography

[1] Full Project Source Code: *https://github/Atan-D-RP4/wt_project*

[2] Git: *https://git-scm.com/*

[3] GitHub: *htps://github.com/*

[4] Deno: *https://deno.com/*

[5] Node.js: *https://nodejs.org/en/*

[6] Bootstrap: *https://getbootstrap.com/*

[7] Express.js: *https://expressjs.com/*

[8] EJS; *https://ejs.co/*

[9] MySQL: *https://www.mysql.com/*

[10] Visual Studio Code. *https://code.visualstudio.com/*