**Software Design and Implementation of an Employee Clocking System**

**Software Design and Management Project Assignment**

**1. Introduction**

**1.1 Project Overview**

This project aims to design and implement a **software-based Employee Clocking System** using **Next.js, PostgreSQL, Prisma, and GitHub**. The system will allow employees to **clock in and out securely** while providing **real-time attendance tracking** for HR and management.

**1.2 Objectives**

✅ Develop a web-based **clocking system** for employees.  
✅ Implement **secure authentication** (Login via email/password).  
✅ Track **clock-in and clock-out timestamps** in a database.  
✅ Provide an **admin dashboard** for HR to monitor attendance.  
✅ Generate **attendance reports** for payroll processing.

**2. Software Development Life Cycle (SDLC) Implementation**

We follow the **Agile SDLC model** to ensure flexibility and continuous improvement.

**2.1 Planning & Requirement Analysis**

**Functional Requirements**

✔ Employees can **log in securely** using email and password.  
✔ Employees can **clock in and out** with automatic timestamps.  
✔ Admins can **view and export** attendance logs.  
✔ The system provides **real-time monitoring**.

**Non-Functional Requirements**

✔ **Security**: Password encryption and JWT authentication.  
✔ **Scalability**: Support for multiple employees.  
✔ **Performance**: Optimized database queries for fast response.

📌 **Deliverable:** **Software Requirements Specification (SRS)**

**2.2 System Design**

The system is built using a **three-tier architecture**:

1️⃣ **Frontend**: Next.js (React-based web UI).  
2️⃣ **Backend**: Node.js with Prisma ORM.  
3️⃣ **Database**: PostgreSQL for storing employee records and timestamps.

📌 **[System Architecture Diagram Here]**

**Entity-Relationship (ER) Diagram**

📌 **[ER Diagram Here]**

**2.3 Implementation**

**Technologies Used:**  
✔ **Frontend**: Next.js (React, Tailwind CSS).  
✔ **Backend**: Node.js, Express.js.  
✔ **Database**: PostgreSQL (with Prisma ORM).  
✔ **Version Control**: GitHub (CI/CD for deployment).

**Step 1: Setup Next.js Project**

npx create-next-app employee-clocking-system

cd employee-clocking-system

npm install

**Step 2: Configure Prisma & PostgreSQL**

npm install @prisma/client @prisma/cli

npx prisma init

**Step 3: Define Prisma Schema (Database Model)**

model Employee {

id Int @id @default(autoincrement())

name String

email String @unique

password String

role String @default("employee")

logs ClockLog[]

}

model ClockLog {

id Int @id @default(autoincrement())

employeeId Int

clockIn DateTime

clockOut DateTime?

employee Employee @relation(fields: [employeeId], references: [id])

}

📌 **[Database Schema Diagram Here]**

**Step 4: Implement Authentication (JWT-based login)**

import jwt from 'jsonwebtoken';

export const authenticateUser = (user) => {

return jwt.sign({ id: user.id, email: user.email }, process.env.JWT\_SECRET, { expiresIn: '1h' });

};

**Step 5: Implement Clocking Feature**

import prisma from '../lib/prisma';

export async function clockIn(employeeId) {

return await prisma.clockLog.create({

data: { employeeId, clockIn: new Date() }

});

}

export async function clockOut(employeeId) {

return await prisma.clockLog.updateMany({

where: { employeeId, clockOut: null },

data: { clockOut: new Date() }

});

}

📌 **[System Flow Diagram Here]**

**2.4 Testing**

**Testing Strategy:**  
✅ **Unit Testing**: Validate login, clock-in, and clock-out functionality.  
✅ **Integration Testing**: Ensure frontend and backend communicate correctly.  
✅ **User Acceptance Testing (UAT)**: Employees and HR test the system.

📌 **Deliverable:** **Test Case Document**

**2.5 Deployment**

**Frontend Deployment:** Vercel

vercel deploy

**Backend Deployment:** Heroku

git push heroku main

📌 **Deliverable:** **Deployment Guide**

**2.6 Maintenance Plan**

✔ Regular security updates (password hashing, JWT refresh tokens).  
✔ Performance monitoring and database optimization.  
✔ User feedback integration for continuous improvement.

📌 **Deliverable:** **Maintenance Report**

**3. UI Wireframes & Screenshots**

📌 **[Login Page Wireframe Here]**  
📌 **[Employee Dashboard Wireframe Here]**  
📌 **[Admin Dashboard Wireframe Here]**

**4. GitHub Repository & Workflow**

1️⃣ **Create a GitHub Repo**: github.com/your-username/employee-clocking-system  
2️⃣ **Push Initial Code**

git init

git add .

git commit -m "Initial commit"

git branch -M main

git remote add origin https://github.com/your-username/employee-clocking-system.git

git push -u origin main

3️⃣ **Set Up CI/CD (GitHub Actions for Auto Deployment)**

name: Deploy

on: [push]

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout Repository

uses: actions/checkout@v2

- name: Install Dependencies

run: npm install

- name: Deploy to Vercel

run: vercel deploy --prod

📌 **Deliverable:** **GitHub Workflow Guide**

**5. Expected Outcomes**

✔ A **fully functional, software-based Employee Clocking System**.  
✔ Secure **authentication, clock-in/out, and real-time attendance tracking**.  
✔ Scalable, responsive **web application** for employees and HR.  
✔ **Admin dashboard for monitoring attendance and generating reports**.

**6. Conclusion & Future Improvements**

**Future Enhancements:**

🔹 Mobile app version for easier accessibility.  
🔹 Integration with **biometric authentication** for added security.  
🔹 AI-powered **anomaly detection** to identify fraudulent clock-ins.

**Final Deliverables**

📌 **Software Requirements Specification (SRS)**  
📌 **System Architecture Diagram**  
📌 **ER Diagram**  
📌 **System Flow Diagram**  
📌 **Database Schema (PostgreSQL & Prisma Models)**  
📌 **UI Wireframes & Screenshots**  
📌 **Step-by-Step Implementation Guide**  
📌 **GitHub Repository & CI/CD Workflow**  
📌 **Testing Plan & Maintenance Guide**

**Next Steps**

Would you like me to:  
🔹 **Generate a final formatted PDF version** of this document?  
🔹 **Provide more code samples** for implementation?  
🔹 **Set up a demo GitHub repository** with the working project?

Let me know what you need!