**UDHNA CITIZEN COMMERCE COLLEGE & S.P.B. COLLEGE OF BUSINESS ADMINISTRATION & SMT. DIWALIBEN HARJIBHAI GONDALIA COLLEGE OF BCA AND I.T.**

Bachelor of Computer Applications

(BCA) Programme

Minor Project Report

Partial Fulfillment of

BCA Sem.-V

A.Y. 2023-24

*Project Title: PAYROLL MANAGEMENT SYSTEM*

*Submitted By:*

|  |  |  |
| --- | --- | --- |
| Exam No. | Roll No. | Name of Student |
| 3603 | 202345005 | *Kishore Ambadas Sunchu* |
| 3493 | 202346002 | *Jyoti Rajkumar Dwivedi* |
| 3476 | 202346030 | *Rupakumari Upendra Chauhan* |

**Project Guided by:**

* *Prof. Dr. Manish Kayasth*
* *Prof. Swapnil Patil*

**Acknowledgement**

We would want to convey my heartfelt gratitude to Prof. Dr. Manish Kayasth and Prof. Swapnil Patil, our mentor, for their invaluable advice and assistance in completing my project. They were there to assist us every step of the way, and his motivation is what enabled us to accomplish my task effectively. We would also like to thank all the other supporting personnel who assisted me by supplying the equipment that was essential and vital, without which we would not have been able to perform efficiently on this project.

We would also want to thank the VNSGU for accepting my project in our desired field of expertise. We had also to thank my friends and parents for their support and encouragement as we worked on this project.







I N D E X

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Description** | **Page No.** |
| 1 | Introduction | 7 |
|  | 1.1 Project description | 7 |
|  | 1.2 Project Profile | 7 |
| 2 | Environment Description | 8 |
|  | 2.1 Hardware and Software Requirements | 8 |
|  | 2.2 Technologies Used | 8 |
| 3 | System Analysis and Planning | 9 |
|  | 3.1 Existing System and its Drawbacks | 9 |
|  | 3.2 Feasibility Study | 9 |
|  | 3.3 Requirement Gathering and Analysis | 9 |
| 4 | Proposed System | 10 |
|  | 4.1 Scope | 10 |
|  | 4.2 Project modules | 10 |
|  | 4.3 Module vise objectives/functionalities Constraints | 10 |
| 5 | Detail Planning | 12 |
|  | 5.1 Data Flow Diagram / UML | 12 |
|  | 5.2 Process Specification / Activity Flow Diagram | 13 |
|  | 5.3 Data Dictionary | 18 |
|  | 5.4 Entity-Relationship Diagram / Class Diagram | 19 |
| 6 | System Design | 20 |
|  | 6.1 Database Design | 20 |
|  | 6.2 Directory Structure | 20 |
|  | 6.3 Input Design | 23 |
|  | 6.4 Output Design | 26 |
|  | 6.5 Development Code | 28 |
| 7 | Software Testing | 41 |
| 8 | Limitations and Future Scope of Enhancements | 42 |
| 9 | References | 43 |

**Payroll-Central**

Payroll Management System

1. **Introduction:**
   1. **Project description:**

A payroll management system is a software application that automates the process of calculating and paying employee salaries. It typically involves keeping track of employee hours worked, calculating taxes and deductions, and generating pay checks. Payroll management systems can save businesses time and money by eliminating the need for manual calculations and processing.

This project will develop a payroll management system for a small business. The system will be designed to be easy to use and affordable. It will also be scalable so that it can be used by businesses of all sizes.

* 1. **Project Profile:**
* Project Title: Payroll-Central
* Project Description: It is a payroll management system web application created in ReactJS and Nodejs.
* Project Duration: 2 months
* Project Team Members: Kishore Sunchu, Jyoti Dwivedi, Rupa Chauhan
* Project Status: completed

1. **Environment Description:**
   1. **Hardware and Software Requirements:**

Follows are the Hardware requirements of the project:

* Processor: Intel Core i5
* SSD: 512GB
* RAM: 8GB

Follows are the Software requirements of the project:

* Windows 7 or higher
* MongoDBCompass
* NodeJs
* Visual Studio Code
* Google Crome Developer Options
  1. **Technologies Used:**

The Technology which are used in the project is as follows:

* **Fronted:**
* ReactJS
* Tailwind CSS
* Material UI
* **Backend:**
  + NodeJs
  + Express
  + MongoDB
  + Mongoose

1. **System Analysis and Planning:**
   1. **Existing System and its Drawbacks:**

**Follows are the existing system for payroll management systems:**

* greytHR
* Keke HR
* HROne
* Workday HCM

**Drawbacks of above system:**

* High cost: The cost of implementing and maintaining can be high, especially for small business
* Inflexibility: Traditional payroll system are often inflexibility and can be difficult to adapt changes in employee information or company policies.
* Difficult to scale: Manual payroll system can be difficult to scale as a company grows.
  1. **Feasibility Study:**
* Technical feasibility: Is the system technically feasible to develop and implement?
* Financial feasibility: Can organization afford to develop and implement the system?
* Market feasibility: Is there a demand for the system in the marketplace?
* Operational feasibility: Can the organization effectively use the system?
  1. **Requirement Gathering and Analysis:**
* Payroll compliance: The system must be able to calculate and deduct all applicable taxes.
* Expense management: The system must be able to track and manage employee expense.
* Dashboard and reporting: The system must provide a dashboard with real-time data on payroll, expenses, and other financial information.

1. **Proposed System:**
   1. **Scope:**

* **Employee Data:** The system should be able to store and manage employee data such as name, address, contact information, job title, salary and deductions.
* **Payroll Calculations:** The system should be able to calculate employee pay accurately, taking into account all relevant factors such as hours worked, overtime, bonuses, and deductions.
* **Payslip generation:** The system should be able to generate payslip for each employee, showing their gross pay, deductions, and net pay.
* **Security:** the system should be secure to protect employee data analytics to help managers track payroll costs, identify trends, and make informed decisions.
* **Self-Service:** The ability for employees to access their pay information and make changes to their personal details.
* **Integration with other systems:** The system can be integrated with other HR systems, such as timekeeping and benefits administration systems.
  1. **Project modules:**

The system comprises of 2 major modules with their sub-modules as follows:

1. Admin
2. Employee
   1. **Module Vise objectives/functionalities Constraints:**

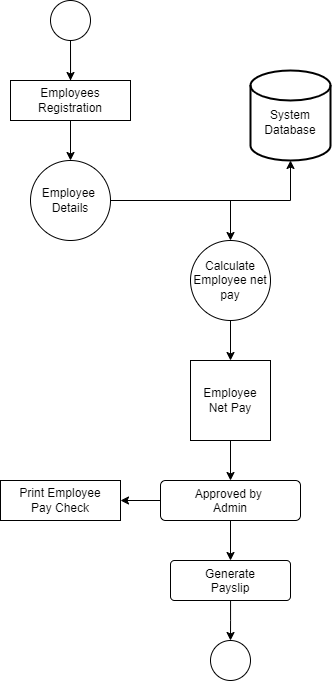
Admin:

* Login: Admin can login into system.
* Add Employee: Admin can add employee.
* Employee Details: Admin can view all employee details.
* Change Details: Admin can change the details of employees.
* Change Status: Admin can change the status of the employee.
* Generate Payroll: Admin can generate payslip for employees
* Logout: Admin can logout.

Employee:

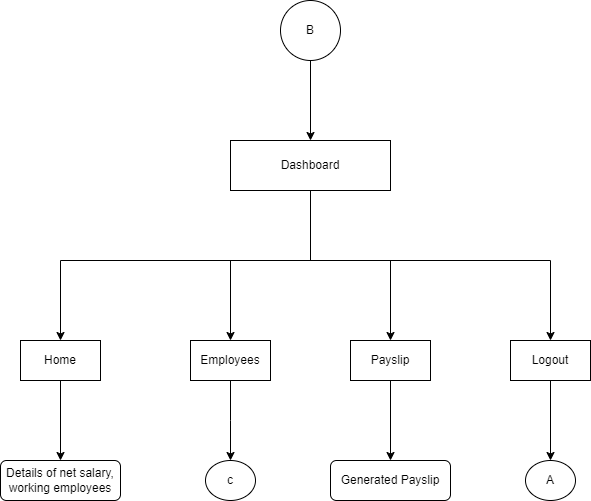
* Login: Employee can login in his profile.
* Profile: Employee view his profile details.
* Change Details: Employee can change his limited details.
* Generate Payroll: Employee can generate payslip for themselves.
* Log out: Employee can logout.

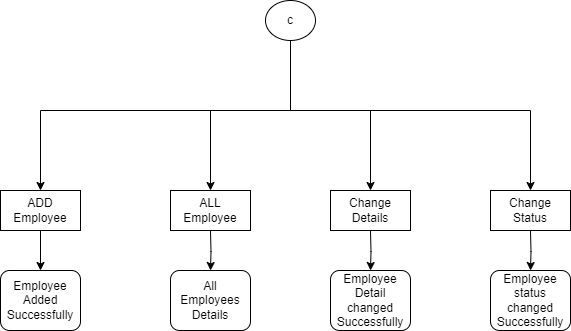
1. **Detail Planning:**
   1. **Data Flow Diagram / UML:**

****

* 1. **Process Specification / Activity Flow Diagram:**
* **Admin:**

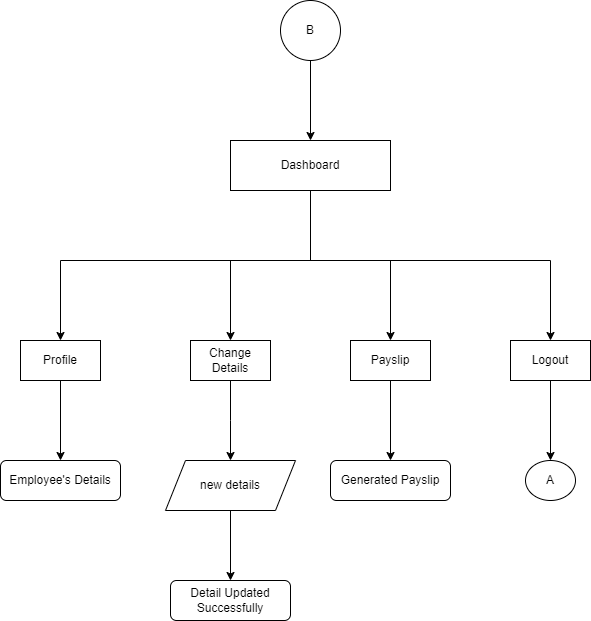






* Employees:





* 1. **Data Dictionary:**

**Table name: Admin**

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| Id | Integer | Unique identifier for admin |
| Username | String | Name of the admin |
| Email | String | Email address of admin |
| Password | String | Pass code to login into the system |

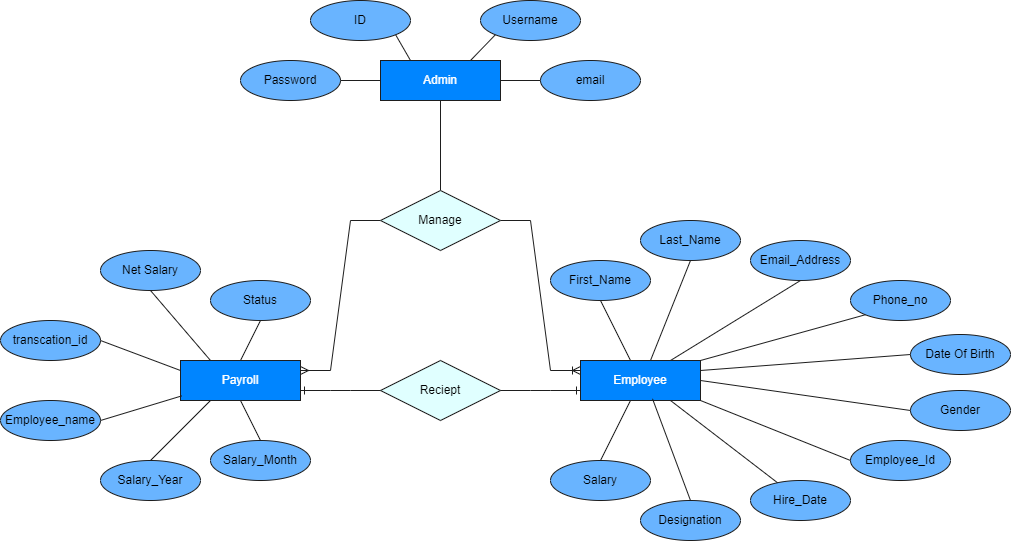
**Table name: Employees**

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| Employee id | Integer | Unique Identifier for each employee |
| First Name | String | First name of the employee |
| Last Name | String | Last name of the employee |
| Email address | String | Email address of the employee |
| Phone No. | Integer | Contact No. of the employee |
| Date of Birth | Date | Birth date of the employee |
| Gender | String | Gender of the employee |
| Hire Date | Date | Hiring date of the employee |
| Designation | String | Designation of the employee |
| Salary | Integer | Salary of the employee |

**Table name: Pay-slips**

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| Transaction id | Integer | Unique id for each transaction |
| Salary month | Date | Month of salary month |
| Salary year | Date | Year of salary year |
| Employee id | Integer | Employee id for references to the employee |
| Net salary | Integer | Total salary of the employee |
| Status | String | Status of the salary (paid/unpaid) |

* 1. **Entity-Relationship Diagram / Class Diagram:**

****

1. **System Desing:**
   1. **Database design:**

* Admin Table:

This table stores the basic information about admin, such as id unique identifier, name for login purpose, email, password for login into the system. The admin can login into the system using name and password.

* Employees Table:

This table stores the basic information about each employee, such as employee id, first and last name, email address, phone number, gender, date of birth, joining and hiring date of the employee, designation of the employee and the salary of the employee.

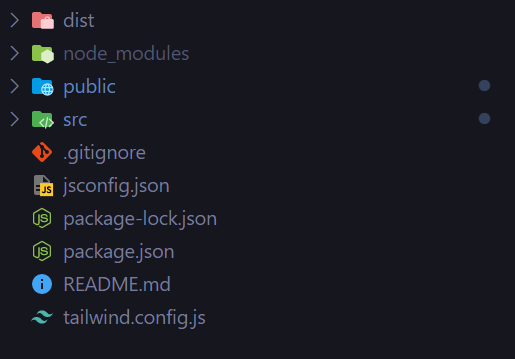
Employee can login into the system using name and password.

* Payslip Table:

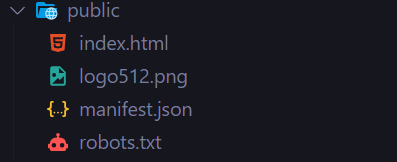
This table stores the information about pay-slips, such as transaction id, month of the salary is given, year of the salary is given, employee name, net salary of employee and status of payment.

* 1. **Directory Structure:**

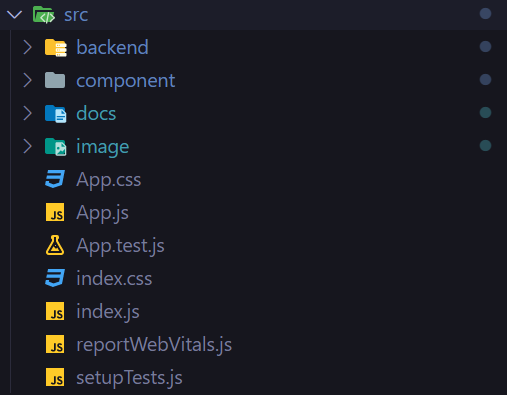
Project folders:

****

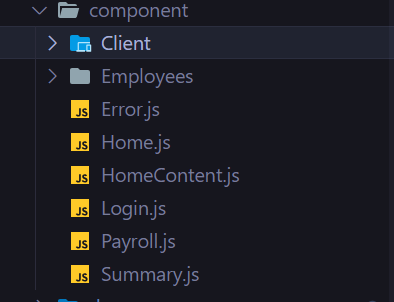
Public folders:



Src folders:

****

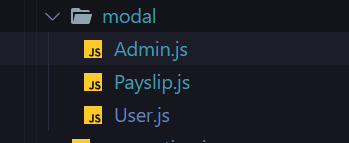
Components folder:



Employees Folder:



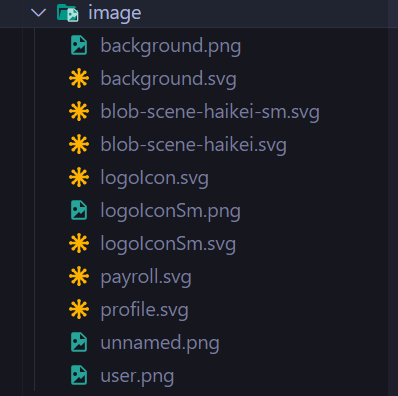
Model Folder:



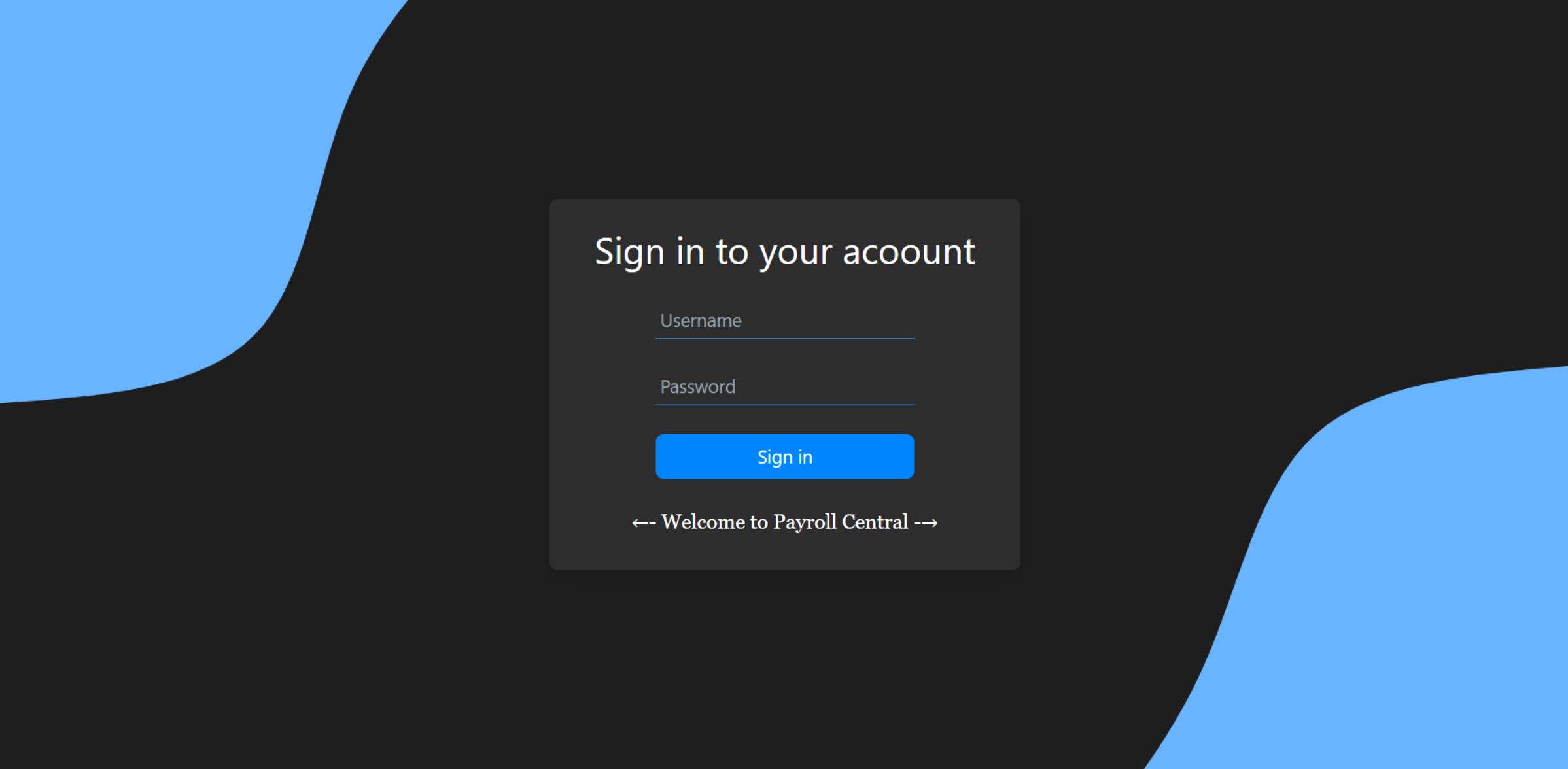
Backend folders:



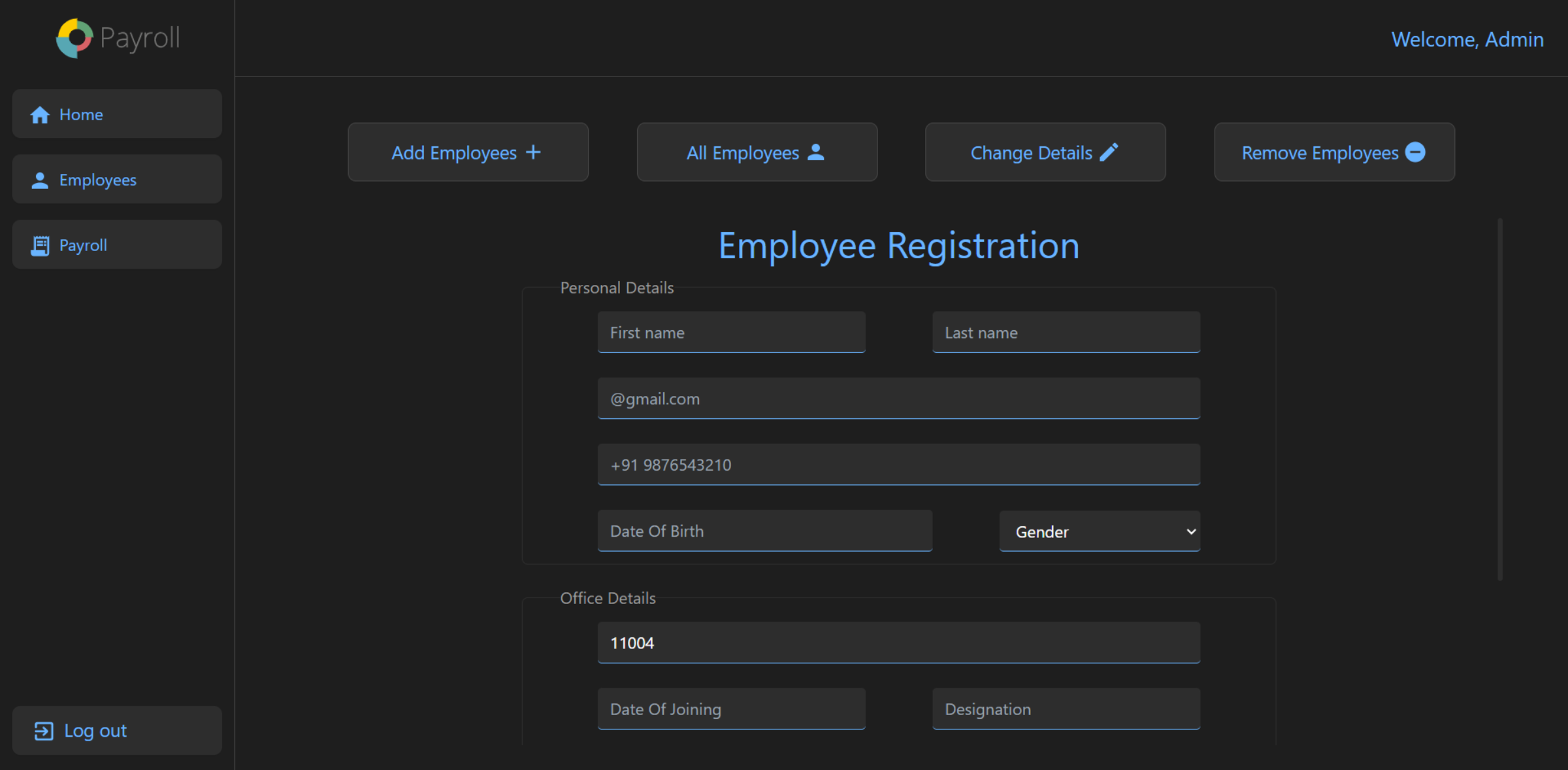
Image folder:



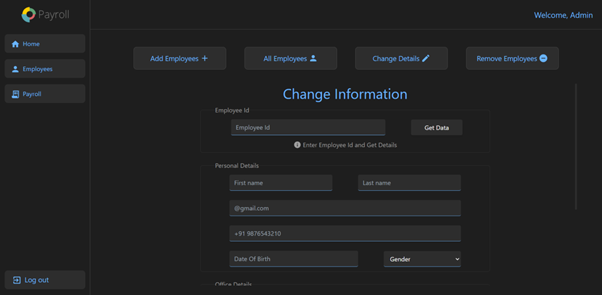
* 1. **Input Design:**
* Login page:

****

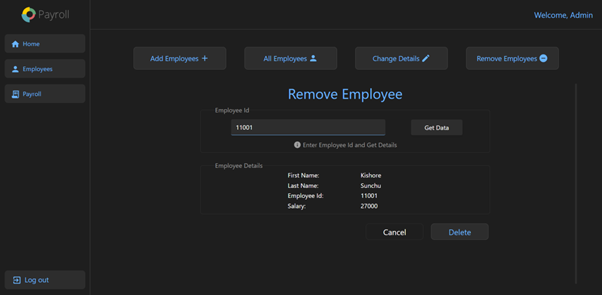
* Employee Registration Form:



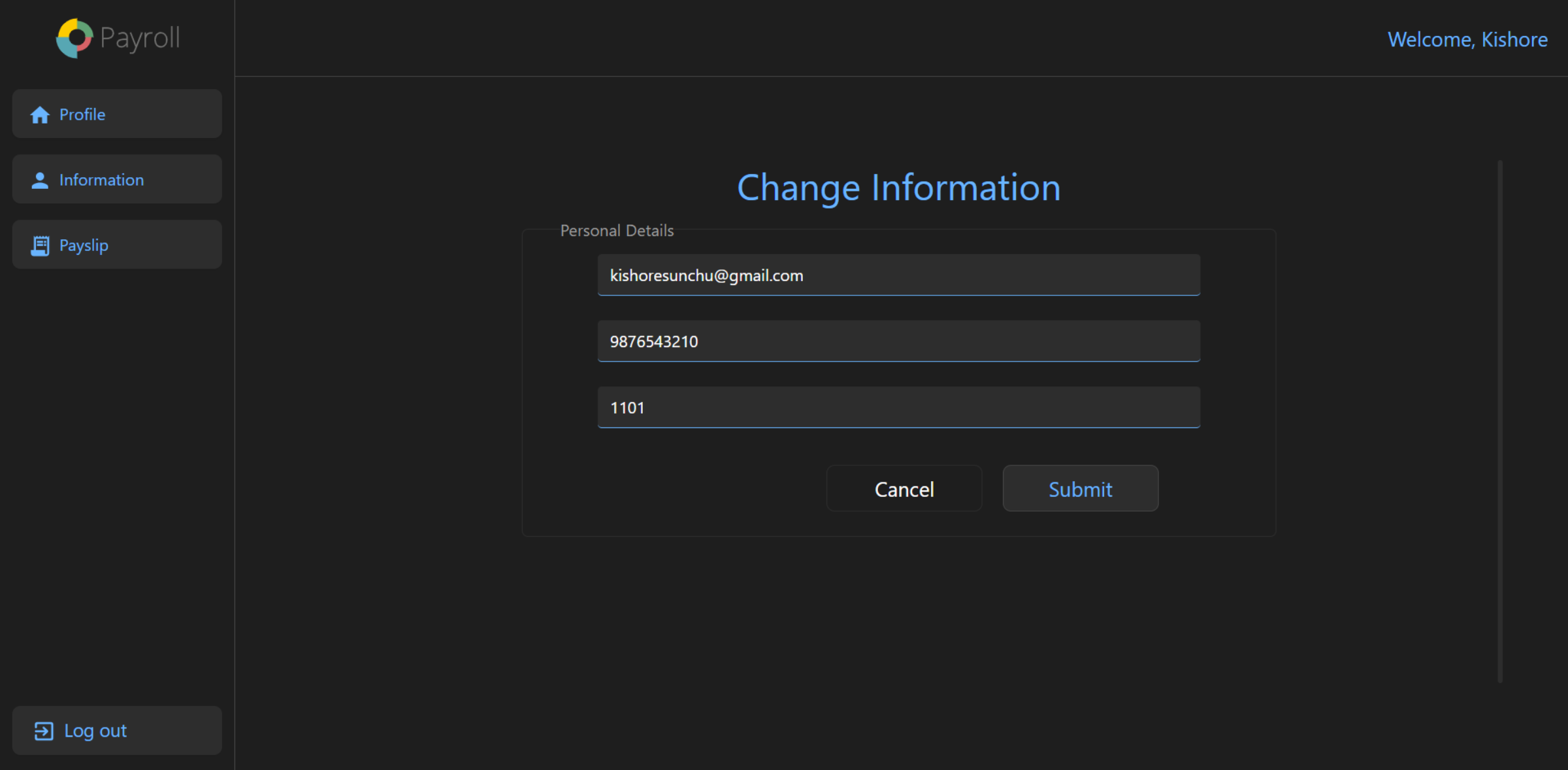
* Employee Change Information Form (Admin Side):



* Employee Active or Inactive Status Change Form:

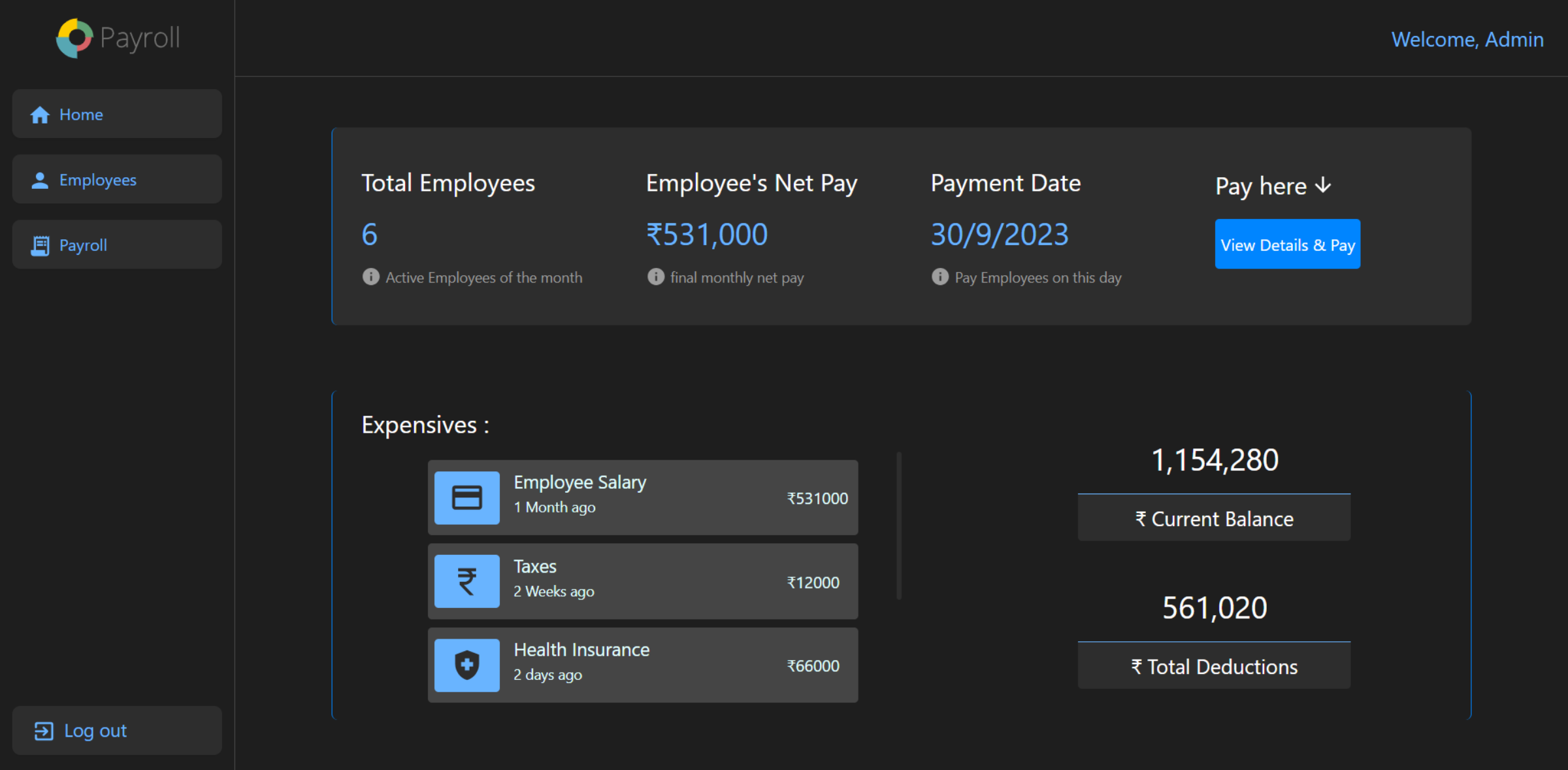


* Change Information Form (Employee Side):

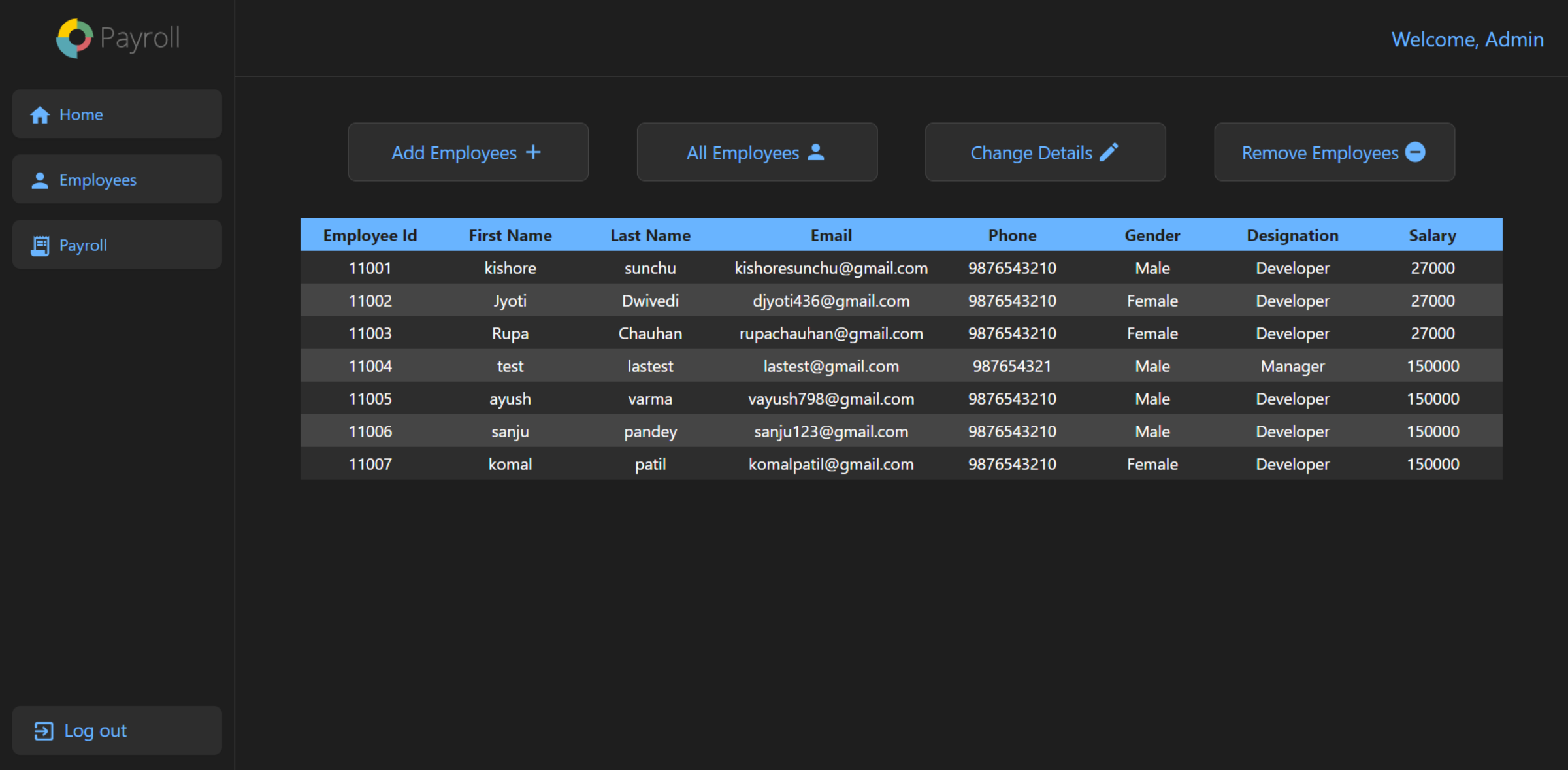


* 1. **Output design:**

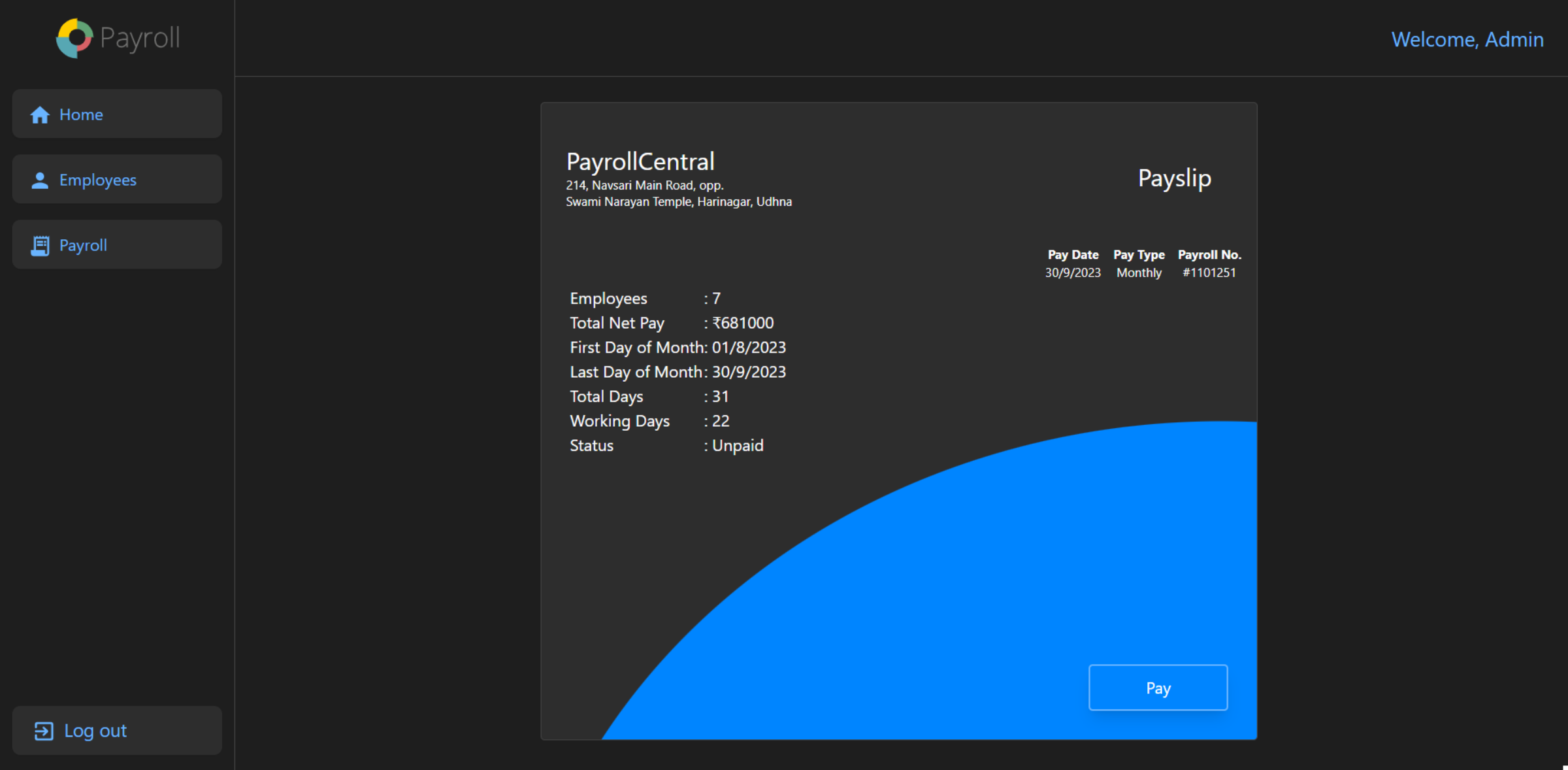
Admin Dashboard – Home page:

****

Admin Dashboard – Employees



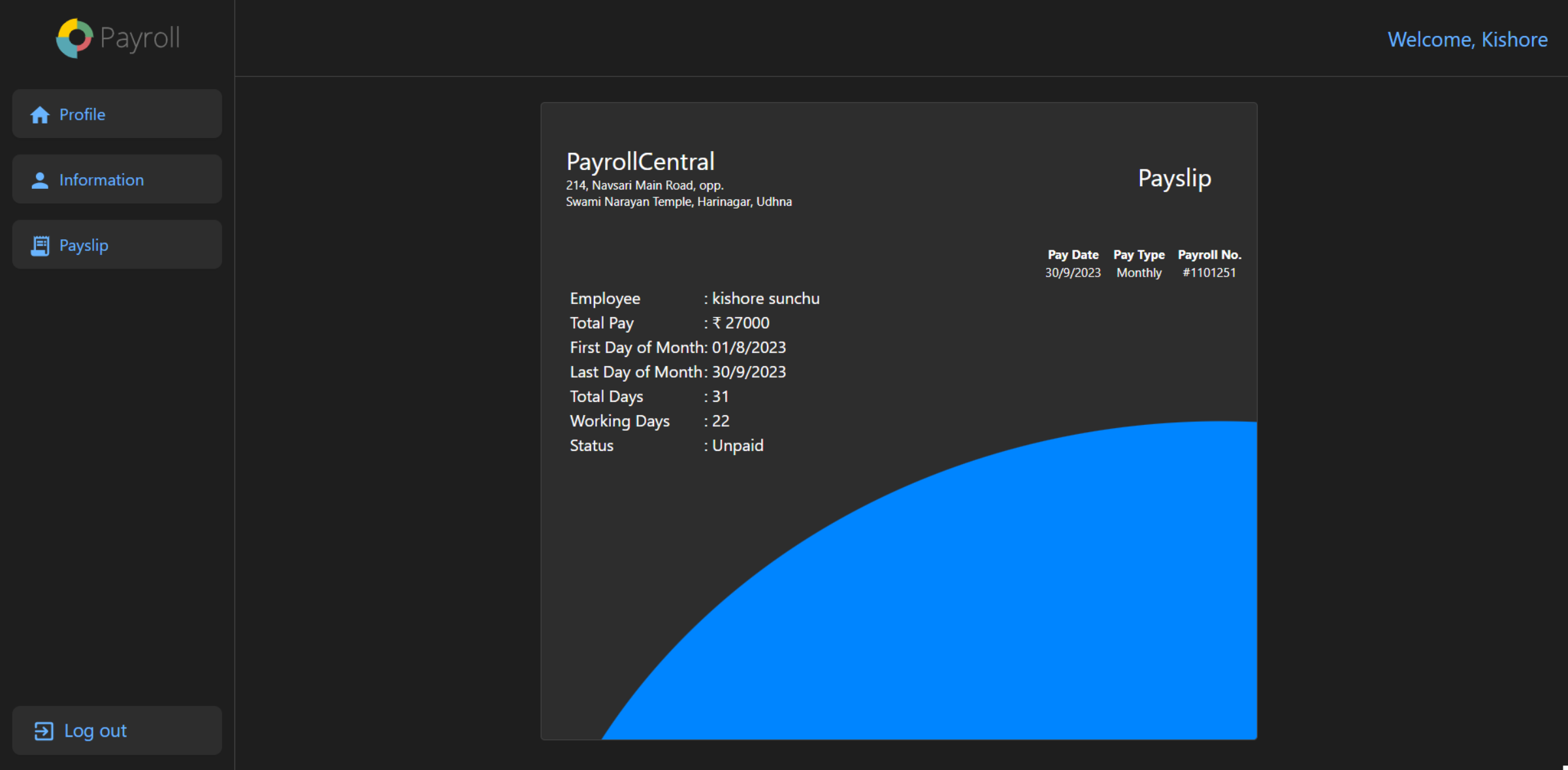
Admin Dashboard – Payroll



Employee Dashboard – Profile



Employee Dashboard – Pay-slip



* 1. **Development Code:**

Index.html:

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="utf-8" />

  <link rel="icon" type="image/x-icon" href="/src/image/payroll.svg">

  <meta name="viewport" content="width=device-width, initial-scale=1" />

  <link rel="stylesheet"

    href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@20..48,100..700,0..1,-50..200" />

  <link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

  <link href="/dist/output.css" rel="stylesheet">

  <title>React App</title>

</head>

<body>

  <noscript>You need to enable JavaScript to run this app.</noscript>

  <div id="root" class="h-screen"></div>

</body>

</html>

App.js:

import "./App.css";

import { useEffect, useState } from "react";

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import Login from "./component/Login";

import Home from "./component/Home";

import HomeContent from "./component/HomeContent";

import Payroll from "./component/Payroll";

import Error from "./component/Error";

import Employee from "./component/Employees/Employee";

import AddEmployee from "./component/Employees/AddEmployee";

import ModifyEmployee from "./component/Employees/ModifyEmployee";

import RemoveEmployee from "./component/Employees/RemoveEmployee";

import ClientDashboard from "./component/Client/ClientDashboard";

import Profile from "./component/Client/Profile";

import Information from "./component/Client/Information";

import Payslip from "./component/Client/Payslip";

function App() {

*const* [Session, setSession] = useState(0);

*const* [Employees, setEmployees] = useState();

*const* [User, setUser] = useState("");

*const* [lastDate, setLastDate] = useState();

*const* [userCount, setUserCount] = useState(0);

*const* [salaryCount, setSalaryCount] = useState(0);

*const* [Status, setStatus] = useState("Unpaid");

  useEffect(() => {

*var* date = new Date();

*var* year = date.getFullYear();

*var* month = date.getMonth();

*var* day = new Date(year, month + 1, 0);

    setLastDate(day.getDate() + "/" + (month + 1) + "/" + year);

  }, []);

*return* (

    <>

      <Router>

        <Routes>

          <Route

            index

            element={

              <Login

                setSession={setSession}

                setUser={setUser}

                setEmployees={setEmployees}

                Employees={Employees}

              />

            }

          />

          <Route

            path="/home"

            element={

              Session !== 0 ? (

                <Home

                  Session={Session}

                  setEmployees={setEmployees}

                  setSession={setSession}

                  User={User}

                />

              ) : (

                <Error />

              )

            }>

            <Route

              path=""

              element={

                <HomeContent

                  userCount={userCount}

                  setUserCount={setUserCount}

                  lastDate={lastDate}

                  setLastDate={setLastDate}

                  salaryCount={salaryCount}

                  setSalaryCount={setSalaryCount}

                />

              }

            />

            <Route

              path="employee"

              element={

                <AddEmployee

                  setEmployees={setEmployees}

                  userCount={userCount}

                />

              }

            />

            <Route

              path="employee/all"

              element={

                <Employee Employees={Employees} setEmployees={setEmployees} />

              }

            />

            <Route

              path="employee/modify"

              element={<ModifyEmployee setEmployees={setEmployees} />}

            />

            <Route

              path="employee/remove"

              element={<RemoveEmployee setEmployees={setEmployees} />}

            />

            <Route

              path="payroll"

              element={

                <Payroll

                  userCount={userCount}

                  salaryCount={salaryCount}

                  lastDate={lastDate}

                  Status={Status}

                  setStatus={setStatus}

                />

              }

            />

          </Route>

          <Route

            path="/Dashboard"

            element={

              Session !== 0 ? (

                <ClientDashboard

                  Session={Session}

                  setSession={setSession}

                  User={User}

                />

              ) : (

                <Error />

              )

            }>

            <Route

              path=""

              element={

                <Profile

                  User={User}

                  setEmployees={setEmployees}

                  Employees={Employees}

                />

              }

            />

            <Route

              path="information"

              element={

                <Information

                  User={User}

                  setEmployees={setEmployees}

                  Employees={Employees}

                />

              }

            />

            <Route

              path="payslip"

              element={

                <Payslip

                  Employees={Employees}

                  lastDate={lastDate}

                  Status={Status}

                  setStatus={setStatus}

                />

              }

            />

          </Route>

        </Routes>

      </Router>

    </>

  );

}

export default App;

login.js:

import React, { useState, useEffect } from "react";

import { useNavigate } from "react-router-dom";

export default function Login({

  setSession,

  setUser,

  setid,

  setEmployees,

  Employees,

}) {

  useEffect(() => {

    document.title = `PayrollCentral | SignIn`;

  });

*const* navigate = useNavigate();

*const* [Form, setForm] = useState("");

*const* handleForm = (e) => {

    setForm({

**...**Form,

      [e.target.name]: e.target.value,

    });

  };

*const* handleSubmit = *async* (e) => {

    e.preventDefault();

*const* response = *await* fetch("http://localhost:5000/auth", {

      headers: {

        "Content-Type": "application/json",

      },

      method: "POST",

      body: JSON.stringify(Form),

    });

*const* data = *await* response.json();

    if (data.status === "true" && data.user === "Admin") {

      setSession(2);

      setUser(data.fname);

      navigate("/home");

    } else if (data.status === "true" && data.user === "Employee") {

      setSession(2);

      setUser(data.fname);

*// setid(data.id);*

*const* response = *await* fetch("http://localhost:5000/employeeData", {

        headers: {

          "Content-Type": "application/json",

        },

        method: "POST",

        body: JSON.stringify({ emp\_id: data.id }),

      });

*const* UserData = *await* response.json();

      setEmployees(UserData);

      navigate("/Dashboard");

    } else {

      alert("Invalid Username and Password");

    }

  };

*return* (

    <>

      <div className="bg-bgColor-100 w-screen h-screen lg:flex flex flex-wrap-reverse justify-center items-center lg:bg-backgroundDesign bg-backgroundDesignSm bg-center bg-cover bg-no-repeat">

        <div className="lg:h-4/5 lg:w-1/2 w-full h-1/2 flex justify-center items-center">

          <form

            className="bg-bgColor-200 lg:h-3/5 p-5 text-textColor-100 lg:w-3/5 w-4/5 rounded-lg lg:shadow-lg shadow-sm"

            onSubmit={handleSubmit}>

            <h1 className="text-center lg:text-4xl lg:mb-8 lg:mt-2 font-sans text-2xl">

              Sign in to your acoount

            </h1>

            <div className="flex justify-center lg:my-7 my-7">

              <input

                type="text"

                name="fname"

                onChange={handleForm}

                className="bg-transparent border-b border-primary-200 outline-none lg:p-1 lg:w-3/5 w-4/5 mx-auto lg:text-lg"

                placeholder="Username"

              />

            </div>

            <div className="flex justify-center lg:my-7 my-7">

              <input

                type="password"

                name="password"

                onChange={handleForm}

                className="bg-transparent border-b border-primary-200 outline-none lg:p-1 lg:w-3/5 w-4/5 mx-auto lg:text-lg"

                placeholder="Password"

              />

            </div>

            <div className="flex justify-center lg:my-7 my-7">

              <button

                type="submit"

                className="bg-primary-100 lg:w-3/5 w-4/5 text-lg lg:p-2 p-1 rounded-lg hover:bg-accent-100 hover:text-primary-300 outline-accent-100">

                Sign in

              </button>

            </div>

            <div className="flex justify-center lg:my-7 my-7">

              <h1 className="lg:text-xl font-serif">

                ←- Welcome to Payroll Central -→

              </h1>

            </div>

          </form>

        </div>

      </div>

    </>

  );

}

connection.js

*const* mongoose = require("mongoose");

*const* URL = "mongodb://127.0.0.1:27017";

*const* connectToMongo = () => {

  mongoose

    .connect(URL)

    .then(() => {

      console.log("Connected to mongo");

    })

    .catch(() => {

      throw new Error("Could not connect");

    });

};

module.exports = connectToMongo;

user.js:

*const* mongoose = require("mongoose");

*const* Schema = mongoose.Schema;

*const* UserSchema = new Schema({

  fname: {

    type: String,

    required: true,

  },

  lname: {

    type: String,

    required: true,

  },

  email: {

    type: String,

    required: true,

    unique: [true, "Email already in used"],

  },

  phone: {

    type: Number,

    required: true,

  },

  dob: {

    type: Date,

    required: true,

  },

  gender: {

    type: String,

    required: true,

  },

  emp\_id:{

    type:Number,

    required:true,

  },

  hire\_date: {

    type: Date,

    required: true,

  },

  job\_title: {

    type: String,

    required: true,

  },

  salary: {

    type: Number,

    required: true,

  },

  password: {

    type: String,

    required: true,

  },

  date: {

    type: Date,

    default: Date.now,

  },

});

*const* User = mongoose.model("employees", UserSchema);

module.exports = User;

server.js:

*const* express = require("express");

*const* cors = require("cors");

*const* bodyParser = require("body-parser");

*const* connectToMongo = require("./connection");

*const* User = require("./modal/User");

*const* port = 5000;

*const* server = express();

server.use(cors());

server.use(bodyParser());

connectToMongo();

*//  to start the localhost*

server.listen(port, () => {

  console.log(`Server is running on port ${port}`);

});

*// for login purpose*

server.post("/auth", *async* (req, res) => {

*const* userId = req.body.fname;

*const* userPassword = req.body.password;

  if (userId === "admin" && userPassword === "admin") {

    res.json({ status: "true", fname: "Admin", user: "Admin" });

  } else {

*const* user = *await* User.findOne({

      $and: [{ fname: userId }, { password: userPassword }],

    });

    if (user !== null) {

      res

        .status(200)

        .json({

          status: "true",

          fname: user.fname,

          user: "Employee",

          id: user.emp\_id,

        });

    } else {

      console.log("sended false");

      res.send(false);

    }

  }

});

*// for count user and total salary*

server.post("/countUser", *async* (req, res) => {

*const* count = *await* User.countDocuments();

*const* totalSalary = *await* User.aggregate([

    { $group: { \_id: null, total\_salary: { $sum: "$salary" } } },

  ]);

*const* salaryCount = totalSalary[0].total\_salary;

  res.status(200).json({ count: count, salary: salaryCount });

});

*// for find all employees  data*

server.post("/employees", *async* (req, res) => {

*const* employees = *await* User.find();

  res.send(employees);

});

*// add new employee*

server.post("/add", *async* (req, res) => {

*const* user = new User();

  user.fname = req.body.fname;

  user.lname = req.body.lname;

  user.email = req.body.email;

  user.phone = req.body.phone;

  user.dob = req.body.dob;

  user.gender = req.body.gender;

  user.emp\_id = req.body.emp\_id;

  user.hire\_date = req.body.hire\_date;

  user.job\_title = req.body.job\_title;

  user.salary = req.body.salary;

  user.password = req.body.password;

*const* data = *await* user.save();

  if (data.\_id !== "") {

    res.json({ status: true });

  } else {

    res.json({ status: false });

  }

});

*// for employee data  by using id*

server.post("/employeeData", *async* (req, res) => {

*const* emp\_id = req.body.emp\_id;

*const* user = *await* User.findOne({ emp\_id: emp\_id });

  if (user !== null) {

    res.send(user);

  } else {

    res.send(false);

  }

});

*// for update employee information*

server.post("/update", *async* (req, res) => {

*const* emp\_id = req.body.emp\_id;

*const* user = *await* User.findOne({ emp\_id: emp\_id });

  if (user !== null) {

*const* data = *await* user.updateOne({

      email: req.body.email,

      phone: req.body.phone,

      password: req.body.password,

    });

    if (data !== null) {

      res.json({ status: true });

    } else {

      res.json({ status: false, reason: "something went wrong" });

    }

  } else {

    res.json({ status: false, reason: "Employee Does Not Exist" });

  }

});

*// for delete employee information*

server.post("/delete", *async* (req, res) => {

*const* emp\_id = req.body.emp\_id;

*const* user = *await* User.findOne({ emp\_id: emp\_id });

*const* data = *await* user.deleteOne();

  if (data.\_id !== "") {

    res.json({ status: true });

  } else {

    res.json({ status: false, reason: "Employee Does Not Exist" });

  }

});

tailwind.config.js:

*/\*\* @type {import('tailwindcss').Config} \*/*

module.exports = {

  content: ["./src/\*\*/\*.{html,js}"],

  theme: {

    extend: {

      colors:{

        'primary' : {

          100:'#0085ff',

          200:'#69b4ff',

          300:'#e0ffff',

        },

        'accent':{

          100:'#006fff',

          200:'#e1ffff',

        },

        'textColor':{

          100:'#FFFFFF',

          200:'#9e9e9e',

        },

        'bgColor':{

          100:'#1E1E1E',

          200:'#2d2d2d',

          300:'#454545',

        },

      },

      backgroundImage:{

        'backgroundDesign':"url(/src/image/blob-scene-haikei.svg)",

        'backgroundDesignSm':"url(/src/image/blob-scene-haikei-sm.svg)",

        'logoIcon':"url(/src/image/logoIcon.svg)",

        'logoIconSm':"url(/src/image/logoIconSm.png)",

        'payroll':"url(/src/image/payroll.svg)",

        'profile':"url(/src/image/profile.svg)",

        'user':"url(/src/image/user.png)",

      },

      spacing: {

        '5%': '5%',

        '10%': '10%',

        '15%': '15%',

        '20%': '20%',

        '25%': '25%',

        '30%': '30%',

        '40%': '40%',

        '50%': '50%',

        '60%': '60%',

        '70%': '70%',

        '80%': '80%',

        '85%': '85%',

        '90%': '90%',

        '95%': '95%',

      },

    },

  },

  plugins: [],

};

1. **Software Testing**
2. **Data Entry:**

Test the data entry fields to make sure they are properly formatted and validated. For example, the phone no field should only accept numeric values and 10-digit number and hire date field should only accept date values.

1. **Calculations:**

Test the payroll calculations to make sure they are accurate. For example, the system should correctly calculate all employee salary accurately.

1. **Security:**

Test the security of the payroll system to make sure it is protected form unauthorized access. For example, the system should require users to authenticate themselves before they can access sensitive data.

1. **Reports:**

Test the payroll reports to make sure they are accurate and easy to read. For example, the system should generate a report that list all employees and their pay stubs for a given period.

1. **Limitations and Future Scope of Enhancements:**
2. **Limitations:**
   1. Internet is required
   2. Proper data is needed.
3. **Future scope of Enhancements:**
   1. Make the system more user-friendly.
   2. Use AI and machine learning.
   3. Automate more tasks
   4. Make the system more secure.
   5. Offer mobile access from website.
   6. Make app for the system.
4. **References:**
   * [**https://tailwindcss.com/**](https://tailwindcss.com/)
   * [**https://www.npmjs.com/**](https://www.npmjs.com/)
   * [**https://legacy.reactjs.org/**](https://legacy.reactjs.org/)
   * [**https://www.mongodb.com/**](https://www.mongodb.com/)
   * [**https://mongoosejs.com/**](https://mongoosejs.com/)
   * [**https://expressjs.com/**](https://expressjs.com/)
   * [**https://nodejs.org/en**](https://nodejs.org/en)
   * [**https://mui.com/**](https://mui.com/)
   * [**https://aicolors.co/**](https://aicolors.co/)
   * [**https://app.haikei.app/**](https://app.haikei.app/)
   * [**https://www.npmjs.com/package/nodemon**](https://www.npmjs.com/package/nodemon)
   * [**https://www.npmjs.com/package/react-countup**](https://www.npmjs.com/package/react-countup)