# Enterprise Employee Directory

## 1. Team Introduction

### Adam Smith
**Frontend Developer**

Built the user interface using React, Tailwind CSS, Daisy UI and React Router. Handled user interaction and API integration on the client side.

### Sakhawat Hossain
**Backend Developer**

Designed and developed a secure and scalable RESTful API using Node.js, Express, and MongoDB. Used Mongoose for data modeling and schema validation.

### Kameron Dear
**ML Model**

Processed Data and built and trained the ML model to predict salaries based on job role, title, and location. Integrated the ML model into the backend using python-shell to provide a salary prediction API.

## 2. Project Overview

### Brief Description

This project is a secure, role-based employee directory. Our primary goal was to build a robust backend API that could handle user authentication, manage complex data relationships, and enforce business rules securely, all while integrating a predictive ML model.

### Core Requirements

- **Secure Authentication:** Implement a JWT-based login system using HttpOnly cookies.
- **Role-Based Access Control (RBAC):** Ensure API endpoints are protected based on user roles (Employee, Manager, HR).
- **Data Integrity:** Design a MongoDB schema to effectively manage hierarchical employee data.
- **ML Integration:** Predict employee salaries based on their data using a Python ML model.

## 3. System Architecture

This section details the project's components and data flows. Click a component to learn more.

### Component Overview

| 💻 Client-Side | ⚙️ Backend Server | 🗄️ Data & ML Layers |
|---|---|---|
| Browser: React Client | Node.js + Express Server | MongoDB Database |
| | | 🐍 Python Script (ML Model) |

*Click a component to see its description.*

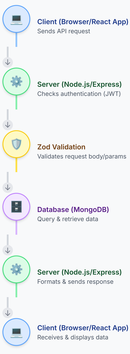
### Request & Response Flows

**A. Initial Authentication Flow**

1. **Login Request**
   User sends credentials from 💻 Client to ⚙️ Server.
2. **Verification & JWT Creation**
   ⚙️ Server validates credentials and creates a signed *JWT*.
3. **Set HttpOnly Cookie**
   ⚙️ Server sends the *JWT* back to the 💻 Client inside a secure `HttpOnly` cookie.
4. **Ready for Authenticated Requests**
   💻 Browser securely stores the cookie for future use.

**B. Authenticated API Request Flow**

1. **API Request**
   💻 Client requests a protected resource (e.g., `/api/employees`). Browser automatically attaches the *JWT* cookie.
2. **Auth Middleware**
   ⚙️ Server middleware intercepts the request, extracts, and verifies the *JWT* from the cookie.
3. **Access Granted**
   If the token is valid, the request proceeds to the main API controller/logic (e.g., fetching from 🗄️ Database or calling 🐍 Python).
4. **Response**
   ⚙️ Server sends the requested data back to the 💻 Client.




## 4. Tools & Software Environment

### 🛠️ Development

- Node.js & Express.js
- React, Vite, React Router
- Tailwind CSS, Daisy UI
- TypeScript
- Mongoose (ODM)

### 🧪 Testing

- Postman (API Testing)

### 📊 Data Analysis & ML

- Python
- Pandas & NumPy
- Scikit-learn

## 5. Challenges & Solutions

### Problem: Integrating Python & Node.js

"We needed a reliable way for our Node.js backend to run the Python ML model for predictions without creating a complex microservice architecture."

### Solution: Using `python-shell`

"We used the `python-shell` library, which allowed the Node.js server to execute the Python script as a child process and communicate via standard I/O. This provided a simple yet effective bridge between the two environments."

### Problem: Enforcing Security on the Server

"A key requirement was that a manager could see their team's salaries, but not others. We had to ensure this logic was enforced securely on the backend, not the client."

### Solution: Backend Service Layer Logic

"The backend service layer intercepts requests for employee data. It checks the logged-in user's role and ID, then filters the data *before* sending the response, nullifying sensitive fields. The client never receives unauthorized data."

## 6. What We Learned

### Before This Project

- Basic understanding of individual components (React, Node.js).
- Limited practical experience with full-stack integration.
- Theoretical knowledge of authentication and database design.

### After This Project

- Successfully built and connected a full-stack application from scratch.
- Implemented a complete, secure authentication and authorization system.
- Gained practical experience in cross-language integration (JS & Python).

## 8. Future Work & Improvements

### Enhanced API Security

Implement rate limiting on login endpoints to protect against brute-force attacks.

### Robust Logging

Integrate a library like Winston to create detailed, structured logs for easier debugging and monitoring.

### Production Deployment

Containerize the Node.js server and Python script using Docker and deploy to a cloud service like AWS with a CI/CD pipeline.

## 9. Q & A

Thank you for your time. We are now ready for your questions.