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

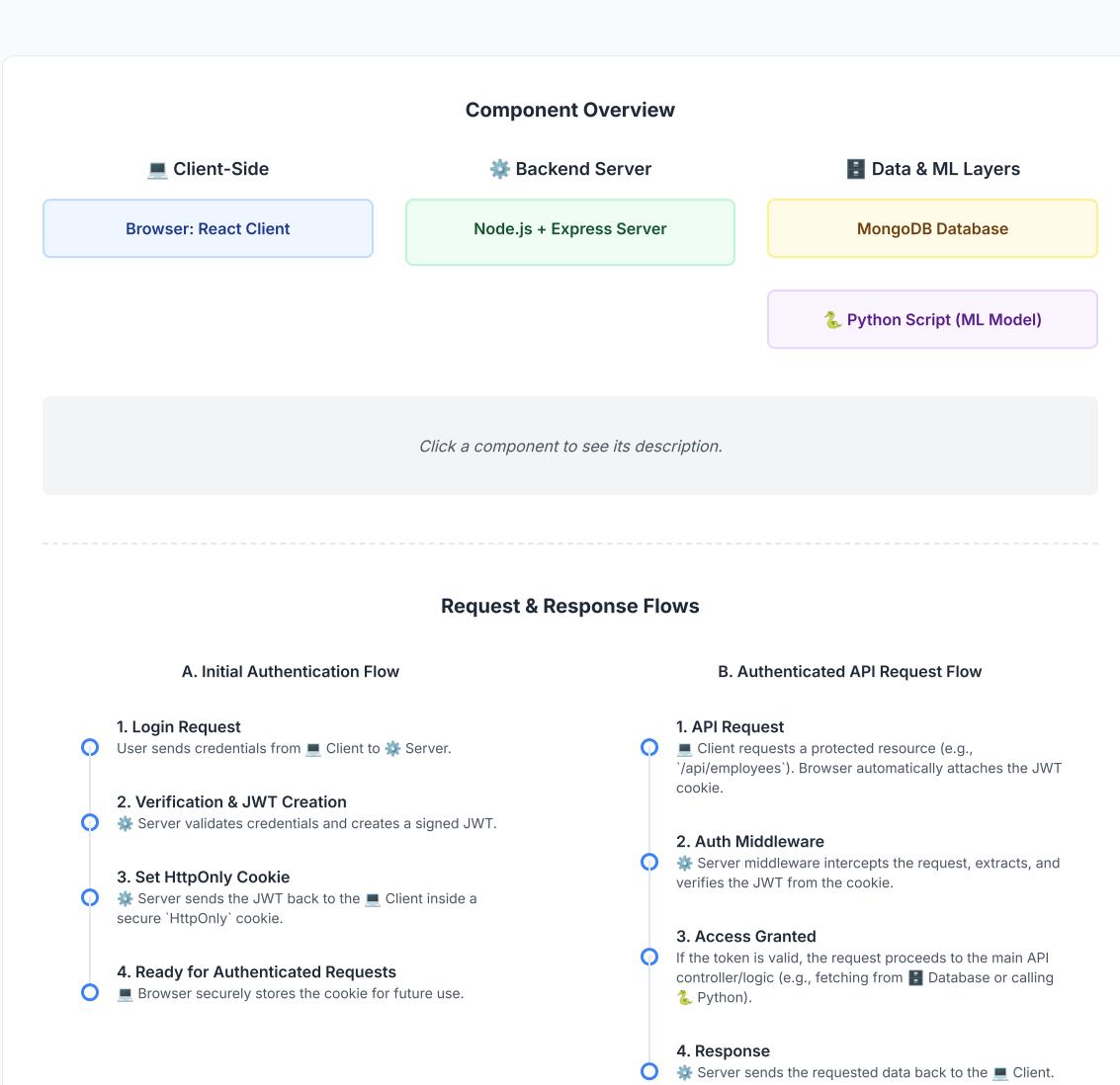
This project is a secure, role-based employee directory. Our primary goal was to build a robust backend API that could handle user authentication, manage complex data relationships, and enforce business rules securely, all while integrating a predictive ML model.

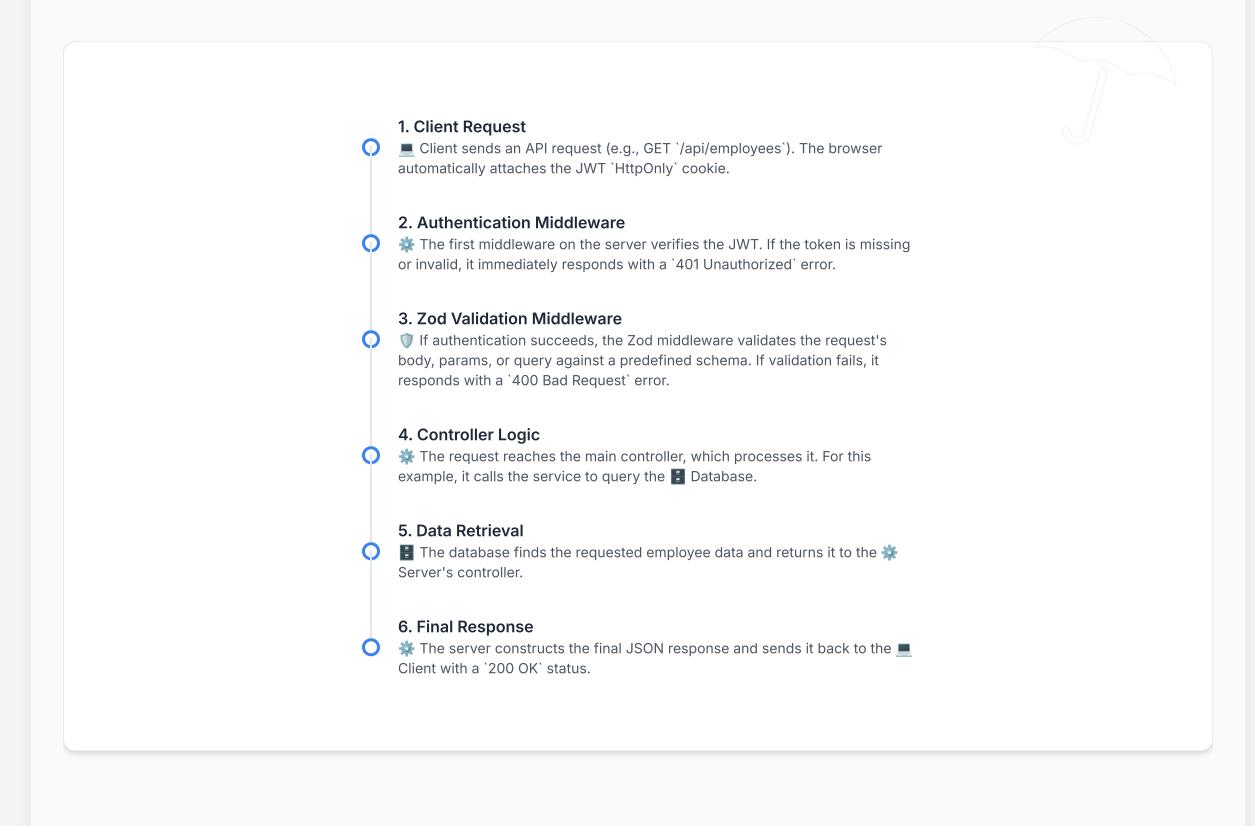
Core Requirements

- 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.





Testing

4. Tools & Software Environment

• Node.js & Express.js • React, Vite, React Router Tailwind CSS, Daisy UI

X Development

- TypeScript
- Mongoose (ODM)
- Postman (API Testing)
- Pandas & NumPy Scikit-learn

Python

■ Data Analysis & ML

Problem: Integrating Python & Node.js Problem: Enforcing Security on the Server

5. Challenges & Solutions

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

"We needed a reliable way for our Node.js backend to run the

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."

"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."

After This Project

6. What We Learned

• Basic understanding of individual components (React, Node.js). • Limited practical experience with full-stack integration.

Before This Project

- Theoretical knowledge of authentication and database design.

• Successfully built and connected a full-stack application from • Implemented a complete, secure authentication and authorization

- system. • Gained practical experience in cross-language integration (JS &
- Python).

Enhanced API Security Implement rate limiting on login

endpoints to protect against brute-force

attacks.

Robust Logging

8. Future Work & Improvements

detailed, structured logs for easier debugging and monitoring.

Production Deployment

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