WMC Employee Portal

Introduction

The **Employee Portal** is a web-based platform designed to facilitate memo sharing, communication, and collaboration within an organization. It allows HR and QM departments to post memos, notify employees, and track engagement through comments, replies, and notifications.

This document serves as a **technical guide** for developers and administrators. It explains the **architecture, key components, and implementation details** of the intranet system, ensuring seamless development, maintenance, and future enhancements.

Target Audience and Their Level of Technical Expertise

The intended audience includes:

- **Developers** (familiar with NestJS, Next.js, Prisma, MySQL, and API development)
- **System Administrators** (basic understanding of deployment, database management, and authentication)
- **Project Managers** (high-level understanding of system functionalities)

Overview of the Code and Its Significance

The **intranet system** follows a **MERN-like stack**, using:

- **Frontend:** Next.js (React-based UI, Zustand for state management)
- **Backend:** NestJS (structured API with Prisma ORM and MySQL)
- **Authentication:** JWT-based authentication with access and refresh tokens
- **File Handling:** Multer for file uploads (memos and images)
- **Notifications:** In app notifications every time a user logs in.

The system plays a crucial role in **streamlining internal communication** by providing a secure and efficient platform for memo distribution and feedback.

WMC Employee Portal

Installation and Setup

System Requirements

Before installing the Intranet Portal, ensure your system meets the following requirements:

- Operating System: Linux (Ubuntu 20.04+ recommended)
- CPU & RAM: At least 2 vCPUs and 4GB RAM (recommended for production)
- Storage: Minimum 10GB available space
- Database: PostgreSQL 12 or higher
- Node.js: Version 18 or higher
- Package Manager: npm or yarn

Prerequisites and Dependencies

Before proceeding with the installation, ensure the following dependencies are installed:

1. NodeJS 18 or later - install via:
   ```
   curl -fsSL https://deb.nodesource.com/setup_18.x | sudo
   -E bash -
   sudo apt install -y nodejs
   ```
2. PostgreSQL 12 or later - Install via:
   ```
   sudo apt update
   sudo apt install -y postgresql postgresql-contrib
   ```

   Then create an account for the app to use by running

   ```
   sudo -i -u postgres
   ```

   After entering postgres type **psql** in the bash then it should enter you to the Database.

3. Git - for cloning the repository:
   ```
   sudo apt install -y git
   ```

4. Docker for running containers

```
sudo apt update
sudo apt install -y docker.io
```

5. Install Docker Compose and enable it

```
sudo apt install -y docker-compose
sudo systemctl enable —now docker
```

Step-by-Step Installation Instructions

1. Open your linux terminal and create the main **Employee Portal** directory
   The folder structure should be like:

```
Intranet/
├── client/
└── server/
Dockerfile
docker-compose.yml
```

2. After creating a folder navigate to the client file and clone the frontend repository by running

```
git clone https://github.com/gian-tiqui/intranet_fe.git
```

Then add the **.env** file to at the root of intranet_fe directory, these are the environmental variables required for the frontend:

KEY	**VALUE**
NEXT_PUBLIC_API_URL	Your api url
NEXT_PUBLIC_INTRANET	intranet
NEXT_PUBLIC_API_KEY	Your api key secret
NEXT_PUBLIC_URL_SECRET	Your url secret
NEXT_PUBLIC_PROJECT_VERSION	Employee Portal Version (eg. V1.0)

3. After setting up the frontend, navigate to the backend directory and clone the repository by running:

```
git clone https://github.com/gian-tiqui/intranet_api.git
```

After cloning the api, navigate to the intranet_api directory and add the .env file. These are the following environmental variables needed in the API.

KEY	**VALUE**
DATABASE_URL	Your database url
AT_SECRET	Access token secret
RT_SECRET	Refresh token secret
AT_EXP	Access token expiration (eg. 7m for 7 minutes)

RT_EXP	Refresh Token expiration (eg. 7d for 7 days (minimum)
NODE_ENV	development or production
PORT	Your api port
CLIENT_URL	Your frontend URL (To give access to the frontend app)
API_KEY	Your api key
PROD_CLIENT	Your frontend URL (To give access to the frontend app) must be enclosed by double quotes (“”)
DEV_CLIENT	Your development URL for frontend for debugging and must be enclosed by double quotes (“”)

Now navigate back to the root folder of the project and add the Dockerfile and docker-compose files:

```
Dockerfile

FROM node:20-alpine

WORKDIR /usr/src/app

COPY ./server/intranet_api/package*.json
./server/intranet_api/
COPY ./client/intranet_ui/package*.json
./client/intranet_ui/

WORKDIR /usr/src/app/server/intranet_api
RUN echo
"https://mirror1.hs-esslingen.de/pub/Mirrors/alpine/v3.21/m
ain" > /etc/apk/repositories \
    && echo
```

```
"https://mirror1.hs-esslingen.de/pub/Mirrors/alpine/v3.21/c
ommunity" >> /etc/apk/repositories \
    && apk update && apk add --no-cache openssl
RUN npm install
RUN npx prisma generate
RUN npm run build

WORKDIR /usr/src/app/client/intranet_ui
RUN npm install
RUN npm run build

COPY ./server/intranet_api /usr/src/app/server/intranet_api
COPY ./client/intranet_ui /usr/src/app/client/intranet_ui

WORKDIR /usr/src/app

RUN npm install -g concurrently

EXPOSE 8081 3000

CMD ["concurrently", "--kill-others", "\"npm --prefix
./server/intranet_api run start:prod\"", "\"npm --prefix
./client/intranet_ui run start\""]
```

docker-compose.yml

```
version: "3.8"

services:
  server:
    build:
      context: ./server/intranet_api
      dockerfile: Dockerfile
    container_name: intranet_api
    ports:
      - "8081:8081"
    volumes:
      - ./server/intranet_api/uploads:/usr/src/app/uploads
      - ./server/intranet_api/logs:/usr/src/app/logs

  client:
    build:
      context: ./client/intranet_fe
      dockerfile: Dockerfile
    container_name: intranet_ui
    ports:
      - "3000:3000"
```

After these steps you can now run the WMC Employees Portal for production by running:

```
sudo docker-compose up --build
```

You can now view the logs stored at the intranet_api/logs/20xx-xx-xx.log

Getting Started

The **WMC Employees Portal** is built using a **three-tier system architecture**, which consists of:

- **UI Layer (Frontend)**
- **API Layer (Backend)**
- **Data Layer (Database)**

Below are the technologies used in each layer:

UI Layer (Frontend)

- **Next.js** – A popular React framework developed and maintained by Vercel, offering features like server-side rendering and static site generation.
- **Zustand** – A lightweight state management library for global data sharing in the UI.
- **TanStack React Query** – Handles data fetching, caching, and synchronization with the server, including stale data management.
- **Tailwind CSS** – A utility-first CSS framework that provides greater flexibility than Bootstrap by allowing class-based customization.
- **Axios** – A powerful HTTP client with request/response interceptors, commonly used for handling authentication tokens.
- **Tesseract.js** – A JavaScript OCR library that enables text extraction from images, supporting script and orientation detection.
- **PrimeReact** – A UI component library that offers a collection of ready-to-use, customizable components.
- **Recharts** – A charting library for creating visually appealing and responsive data visualizations.

API Layer (Backend)

- **NestJS** – A progressive Node.js framework built on top of Express.js, providing decorators and pipes for structured data conversion and validation.
- **Prisma ORM** – A modern Object-Relational Mapper (ORM) that simplifies database queries while ensuring data integrity and security.

Data Layer (Database)

- **PostgreSQL (v12)** – A powerful relational database system known for its advanced query capabilities and robust data management features.

Features

1. Authentication Module

- User registration and login
- Token-based authentication (JWT)

- Token refresh mechanism
- Logout functionality
- Forgot password functionality by entering user’s secret answer

2. Notification & Unread Posts Module

- In-app notifications for new department posts
- Tracking unread posts per user
- Posts marked as read only when the "Read" button is pressed

3. Posting Module

- HR can create, edit, and delete posts
- Posts can be public or restricted to selected departments

4. Commenting & Replying Module

- Users can comment on posts
- Posters can reply to feedback
- Timestamps for comments and replies

5. User Reads Monitoring & History Module

- Provides read history of a user
- HR and QM can view unread users for department posts

6. Dark Mode Module

- Enables dark mode for better user experience

7. General & Department Bulletin Module

- Displays all public posts
- Department bulletin contains all the posts for a department whether public or private

8. User Settings Module

- Allows users to update profile details

- Enables password change functionality
- Allows user to set secret question and answer for password recovery

9. PDF to Image Conversion & Text Extraction Module

- Upload PDFs and convert them into images
- Extract text from images after conversion

10. Search Module

- Search posts based on extracted text, title, or post description

11. Graphs & Data Management Module (Admin)

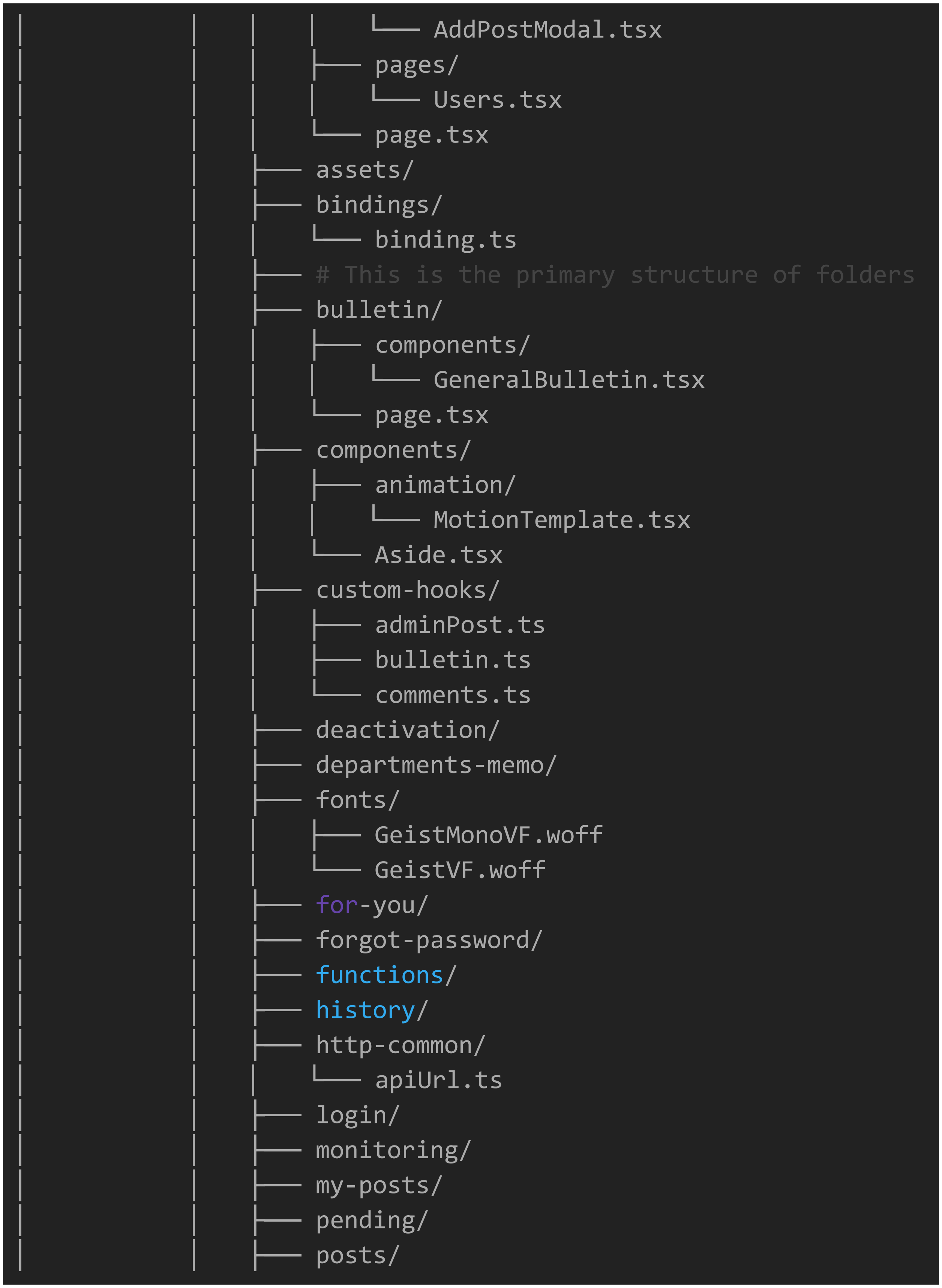
- Provides insights for IT users
- Graph-based data visualization
- Basic usage examples
- Important concepts and terminology

Data-Flow Diagram

Code Structure

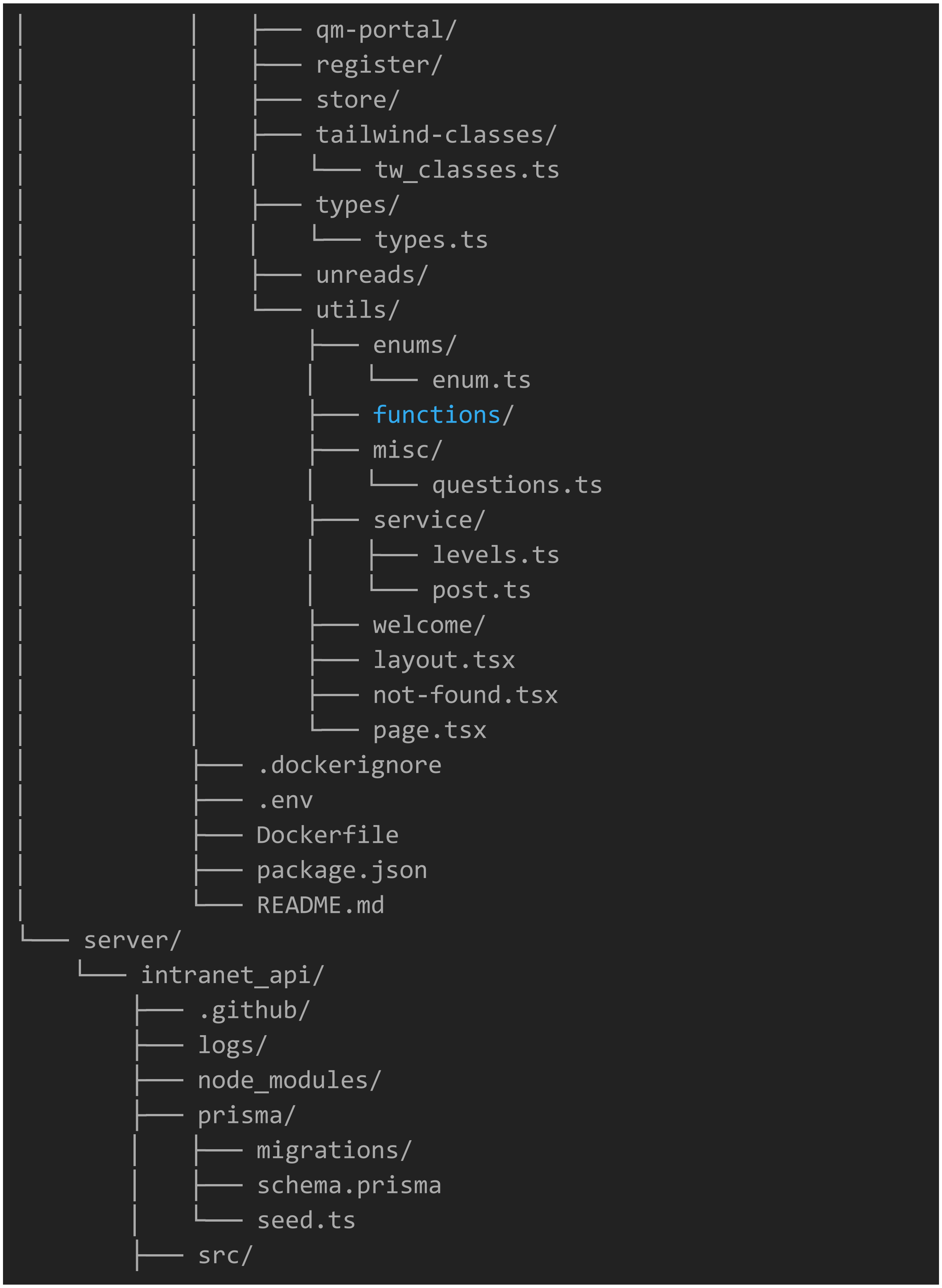
File Structure:

```
Intranet/
├── client/
│   └── intranet_ui/
│       ├── node_modules/
│       └── src/
│           ├── app/
│           │   ├── activate/
│           │   │   ├── ActivateForm.tsx
│           │   │   └── page.tsx
│           │   ├── admin/
│           │   │   ├── components/
```



```
│           │   │   │   └── AddPostModal.tsx
│           │   │   ├── pages/
│           │   │   │   └── Users.tsx
│           │   │   └── page.tsx
│           │   ├── assets/
│           │   ├── bindings/
│           │   │   └── binding.ts
│           │   ├── # This is the primary structure of folders
│           │   ├── bulletin/
│           │   │   ├── components/
│           │   │   │   └── GeneralBulletin.tsx
│           │   │   └── page.tsx
│           │   ├── components/
│           │   │   ├── animation/
│           │   │   │   └── MotionTemplate.tsx
│           │   │   └── Aside.tsx
│           │   ├── custom-hooks/
│           │   │   ├── adminPost.ts
│           │   │   ├── bulletin.ts
│           │   │   └── comments.ts
│           │   ├── deactivation/
│           │   ├── departments-memo/
│           │   ├── fonts/
│           │   │   ├── GeistMonoVF.woff
│           │   │   └── GeistVF.woff
│           │   ├── for-you/
│           │   ├── forgot-password/
│           │   ├── functions/
│           │   ├── history/
│           │   ├── http-common/
│           │   │   └── apiUrl.ts
│           │   ├── login/
│           │   ├── monitoring/
│           │   ├── my-posts/
│           │   ├── pending/
│           │   ├── posts/
```

WMC Employee Portal



```
│           │   ├── qm-portal/
│           │   ├── register/
│           │   ├── store/
│           │   ├── tailwind-classes/
│           │   │   └── tw_classes.ts
│           │   ├── types/
│           │   │   └── types.ts
│           │   ├── unreads/
│           │   └── utils/
│           │       ├── enums/
│           │       │   └── enum.ts
│           │       ├── functions/
│           │       ├── misc/
│           │       │   └── questions.ts
│           │       ├── service/
│           │       │   ├── levels.ts
│           │       │   └── post.ts
│           │       ├── welcome/
│           │       ├── layout.tsx
│           │       ├── not-found.tsx
│           │       └── page.tsx
│           ├── .dockerignore
│           ├── .env
│           ├── Dockerfile
│           ├── package.json
│           └── README.md
└── server/
    └── intranet_api/
        ├── .github/
        ├── logs/
        ├── node_modules/
        ├── prisma/
        │   ├── migrations/
        │   ├── schema.prisma
        │   └── seed.ts
        ├── src/
```

```
│   ├── auth/
│   │   ├── dto/
│   │   │   ├── login.dto.ts
│   │   │   ├── logout.dto.ts
│   │   │   ├── refresh-token.ts
│   │   │   └── register.dto.ts
│   │   ├── guards/
│   │   │   └── auth.guard.ts
│   │   ├── strategy/
│   │   │   └── jwt.strategy.ts
│   │   ├── auth.controller.ts
│   │   ├── auth.module.ts
│   │   └── auth.service.ts
│   ├── # This is the primary structure of all modules
│   ├── comment/
│   │   ├── dto/
│   │   │   ├── create-comment.dto.ts
│   │   │   └── update-comment.dto.ts
│   │   ├── comment.controller.ts
│   │   ├── comment.module.ts
│   │   └── comment.service.ts
│   ├── department/
│   ├── division/
│   ├── edit-logs/
│   ├── folder/
│   ├── level/
│   ├── logger/
│   ├── middleware/
│   ├── monitoring/
│   ├── notification/
│   ├── post/
│   ├── post-department/
│   ├── post-reader/
│   ├── prisma/
│   ├── templates/
│   ├── user/
```

WMC Employee Portal

```
         │   ├── utils/
         │   │   └── functions/
         │   │       ├── extractAccessToken.ts
         │   │       ├── extractUserId.ts
         │   │       ├── sanitize.ts
         │   │       └── sanitizeSQL.ts
         │   ├── app.controller.ts
         │   ├── app.module.ts
         │   └── main.ts
         ├── test/
         ├── uploads/
         ├── .env
         ├── .dockerignore
         ├── Dockerfile
         ├── package.json
         └── README.md
docker-compose.yml
DockerFile
```

Description of Each Major Component

1. Frontend (client/intranet_fe)

- **Framework:** Next.js (React-based)
- **Purpose:** Provides the **user interface** for employees to interact with the system.
- **Key Features:**
  - User authentication (Login/Logout)
  - Dashboard displaying memos and announcements
  - Memo creation and file uploads
  - Notifications for comments and replies
  - Department-based memo filtering
  - Profile management

2. Backend (server/intranet_api)

- **Framework:** NestJS with Prisma ORM
- **Purpose:** Handles business logic, authentication, and database operations.
- **Key Features:**
  - User authentication (JWT-based)
  - Memo posting and department filtering
  - Comments and replies system
  - Notifications (new memo, new comment, new reply)
  - API key protection for secure API access
  - File upload and retrieval via Multer
  - Prisma ORM for database management

3. Database (PostgreSQL)

- **Managed via:** Prisma ORM
- **Purpose:** Stores all system data such as users, memos, comments, and notifications.
- **Key Tables:**
  - User – Stores employee details, credentials, and roles.
  - Post – Stores memos and announcements.
  - Comment – Stores comments and replies on posts.
  - Department – Defines departments for memo filtering.
  - Notification – Stores notification data for users.

4. Authentication System

- **Method:** JWT (JSON Web Token)
- **Tokens Used:**
  - **Access Token** (Short-lived, for authentication)
  - **Refresh Token** (Long-lived, for re-authentication)
- **API Endpoints:**
  - /login – Generates tokens
  - /refresh – Issues a new access token

  - /logout – Revokes refresh tokens

5. Docker & Deployment

- **Docker:** Containers for frontend and backend for easy deployment.
- **Docker Compose:** Manages multi-container setup.
- **Volumes:** Stores uploads and logs persistently.

Flow of Interaction

1. **User logs in** → Frontend sends credentials to Backend (/login).
2. **Backend verifies credentials** → Returns JWT access/refresh tokens.
3. **Frontend requests data** → Calls Backend APIs (/posts, /comments).
4. **Backend fetches from database** → Returns data to Frontend.
5. **User interacts (posts memo, comments, etc.)** → Frontend sends API requests.
6. **Backend processes requests** → Updates database and sends notifications.
7. **Notifications are delivered** → Frontend displays alerts for new memos or replies

API Documentation

This section includes the API endpoints with their parameters if applicable in the backend of the application. In addition to that it will show which headers, body, response, and errors of each endpoint.

Note that there are endpoints that require a bearer token in order to access data from the server.

WMC Employee Portal

Base URL

```
http://localhost:8081
```

Endpoints:

Authentication

1. Login

```
POST /auth/login
```

```
    Request Body:
```

```
{
     employeeId: "00002616",
     password: "abcd_123",
}
```

2. Register

```
POST /auth/register
```

```
    Request Body:
```

```
{
     email: "gian.tiqui.dev@gmail.com",
     password: "abcd_123",
     firstName: "Michael Gian",
     middleName: "Magsino",
     lastName: "Tiqui",
     lastNamePrefix: "",
     preferredName: "",
     suffix: "",
     address: "184 Independece, GSIS",
     city: "City of San Pedro",
     state: "Laguna",
     zipCode: 4023,
     dob: "05/07/2021",
     gender: "male",
      deptId: 3,
      employeeId: "00002616",
      lid: 1,
      divisionId: 2,
}
```

3. Logout

```
POST /auth/logout
```

`Request Body:`

```
{
      userId: 123,
}
```

4. Refresh Token

```
POST /auth/refresh
```

`Request Body:`

```
{
      refreshToken: "asdsad.sdadsadsa.asdsdaawerwersda.asdsadsadsda"
}
```

5. Forgot Password

```
POST /auth/forgot-password
```

`Query Parameters:`

```
{
      employeeId: "00002616",
      answer: "Dog",
      newPassword: "newPassword123",
      secretQuestion: "What was your first pet?",
}
```

WMC Employee Portal

Users

1. Fetch Users

```
GET /users
```

```
Query Parameters:
```

```
{
      search: "search query",
      skip: 0,
      take: 10,
      deptId: 3,
      confirm: "true"
}
```

2. Get User by ID

```
GET /users/<id>
```

3. Update User by ID

```
PUT /users/<id>
```

```
Request Body:
```

```
{
      employeeId: "00002616",
      password: "secrethulaanmo",
      firstName: "Michael Gian",
      middleName: "Magsino",
      lastName: "Tiqui",
      lastNamePrefix: "",
      preferredName: "Gi",
      suffix: "",
      address: "184 Independence, GSIS",
      city: "San Pedro",
      state: "Laguna",
      zipCode: 4023,
      dob: "05/07/2001",
      gender: "male",
      deptId: 3,
      employeeId: "00002616",
      lid: 1,
      divisionId: 1,
}
```

4. Change User Password

```
POST /users/password
```

Request Body:

```
{
      userId: 123,
      oldPassword: "abcd_123",
      newPassword: "Abcd_123",
}
```

5. Get User History

```
GET users/history/<id>
```

Parameters:

```
{
      search: "HR Advisory",
}
```

6. Deactivate a User

```
POST /user/deactivate
```

Parameters:

```
{
      password: "abcd_123",
      employeeId: "00002616",
      deactivatorId: 26,
}
```

7. Set User's Secret Question

```
POST /users/secret-question
```

Parameters:

```
{
     question: "what was the name of your first dog?",
     answer: "dog",
     userId: 123,
}
```

Post

1. Get Admin Posts

```
GET /post/admin
```

2. Get User's Posts

```
GET post/my-posts
```

`Parameters:`

```
{
     userId: 123,
     direction: "asc",
     offset: 0,
     limit: 10,
}
```

3. Get Posts

```
GET /post
```

Parameters:

```
{
  "userId": 123,
  "imageLocation": "uploads/images/sample.jpg",
  "search": "sample query",
  "public": "true",
  "userIdComment": 456,
  "lid": 2,
  "offset": 0,
  "limit": 100,
  "direction": "asc",
  "deptId": 10
}
```

4. Create Post (Also accepts file upload)

```
POST /post
```

Request Body:

```
{
  "userId": 123,
  "title": "Sample Post Title",
  "deptIds": "1,2,3",
  "message": "This is a sample message.",
  "public": "true",
  "lid": 2,
  "extractedText": "Extracted content from the post file/s.",
  "folderId": 10,
  "downloadable": 1
}
```

5. Get Post By ID

```
POST /post/<postId>
```

6. Update Post by ID (Also accepts file upload)

```
PUT /post/<postId>
```

Request Body:

```
{
  "message": "Updated post message.",
  "title": "Updated Post Title",
  "public": "true",
  "deptIds": "1,2,3",
  "lid": 2,
  "extractedText": "Updated extracted text from the post.",
  "updatedBy": 123,
  "addPhoto": "path/to/new/photo.jpg",
  "downloadable": 1,
  "folderId": 10
}
```

7. Delete Post by ID

```
DELETE /post/<postId>
```

Department

1. Get Departments

```
GET /department
```

Parameters:

```
{
  "search": "example query",
  "skip": 0,
  "take": 50,
  "includeSubfolders": 1,
  "depth": 2,
  "deptId": 10,
  "confirm": "true"
}
```

2. Get Department by ID

```
GET /department/<deptId>
```

3. Create Department

```
POST /department
```

Request Body:

```
{
  "departmentName": "Human Resources",
  "departmentCode": "HR",
  "divisionId": 1
}
```

4. Update Department by ID

```
PUT /department/<deptId>
```

`Request Body:`

```
{
  "departmentName": "Updated Department Name",
  "departmentCode": "UPDATED_CODE",
  "divisionId": 2
}
```

5. Delete Department by ID

```
DELETE /department/<deptId>
```

Comment

1. Get Comments

```
GET /comment/
```

`Parameters:`

```
{
     userId: 123
}
```

2. Get Comment Replies

```
POST /comment/replies
```

`Parameters:`

```
{
     parentId: 123
}
```

3. Get Comment by ID

```
GET /comment/<commentId>
```

4. Create Comment

```
POST /comment
```

```
Request Body:
```

```
{
  "userId": 123,
  "postId": 456,
  "message": "This is a sample comment.",
  "parentId": 789
}
```

5. Update Comment by ID

```
PUT /comment/<commentId>
```

```
Request Body:
```

```
{
  "message": "This is an updated comment.",
  "parentId": 789,
  "updatedBy": 123
}
```

6. Delete Comment by ID

```
DELETE /comment/<commentId>
```

Notification

1. Get Unread Posts of a User by ID

```
GET /notification/undreads/<userId>
```

```
Parameters:
```

```
{
     deptId: 1
}
```

2. Get Posts that has been read by the User by ID

```
POST /notification/user-reads
```

```
Request Body:
```

```
{
     userId: 123,
     deptId: 1,
}
```

3. Get Notifications

```
GET /notification
```

```
Parameters:
```

```
{
     userId: 123,
     isRead: true
}
```

4. Get Notification by ID

```
POST /notification/<notificationId>
```

5. Create Post Reply Notification

```
POST /notification/post-reply
```

Parameters:

```
{
      userId: 123,
      postId: 1234,
      Cid: 12345,
}
```

6. Create Comment Reply Notification

```
POST /notification/comment-reply
```

Parameters:

```
{
      userId: 123
      commentId: 1234
}
```

7. Create New Post Notification

```
POST /notification/new-post
```

Parameters:

```
{
      deptId: 1,
      postId: 123,
      lid: 1,
}
```

8. Check if the User has read all of the post for his/her Department

```
POST /notification/read/<userId>
```

9. Delete Notification by ID

```
DELETE notification/<notificationId>
```

Post Reader

1. Get Post Readers

```
GET /post-reader
```

2. Get Post Reader by ID

```
GET /post-reader/<postReaderId>
```

3. Create Post Reader

```
POST /post-reader
```

`Request Body:`

```
{
     postId: 123,
     userId: 1234,
}
```

Monitoring

1. Get Users Read and Unread Count

```
GET /monitoring/users
```

2. Get Read Status

```
POST /monitoring
```

`Parameters:`

```
{
     userId: 123,
     postId: 1234,
}
```

Level

1. Get Levels

```
GET /level
```

`Parameters:`

```
{
    lid: 1,
}
```

Post Department

1. Get Post Department by Department IDS

```
POST /post-department/deptIds
```

    Parameters:

```
{
     postId: 123,
}
```

Edit Logs

1. Get Edit Logs

```
GET /edit-logs
```

    Parameters:

```
{
     editTypeId: 1234
}
```

Folder

1. Get Folders

```
GET /folders
```

    Parameters:

```
{
  "search": "example query",
  "skip": 0,
  "take": 50,
  "includeSubfolders": 1,
  "depth": 2
}
```

2. Create Folder

```
POST /folders
```

`Request Body:`

```
{
  "name": "New Folder",
  "textColor": "#000000",
  "folderColor": "#FFFFFF"
}
```

3. Get Folder’s Subfolder by ID

```
GET /folder/<folderId>/subfolder
```

`Query:`

```
{
  "search": "example query",
  "skip": 0,
  "take": 50,
  "includeSubfolders": 1,
  "depth": 2
}
```

4. Get Folder’s Post by ID

```
GET /folder/<folderId>/post
```

Request Body:

```
{
  "search": "example query",
  "skip": 0,
  "take": 50,
}
```

5. Get Folder by ID

```
GET /folder/<folderId>
```

6. Update Folder by ID

```
PUT /folder<folderID>
```

`Request Body:`

```
{
  "name": "New Folder",
  "textColor": "#000000",
  "folderColor": "#FFFFFF"
}
```

7. Delete Folder by ID

```
DELETE /folder/<folderId>
```

Division

1. Get Divisions

```
GET /divisions
```

`Request Body:`

```
{
  "search": "example query",
  "skip": 0,
  "take": 50,
}
```

WMC Employee Portal

Configuration and Customisation

Configuration and Customization
1. Explanation of Configuration Files or Settings

Several configuration files manage the behavior of the **Intranet Employee Portal**. Below are the key files and their purposes:

Frontend Configuration (client/intranet_fe/.env)

- Defines environment variables for the frontend.
- Controls API URLs, security keys, and project settings.

```
NEXT_PUBLIC_API_URL=http://localhost:8081/
NEXT_PUBLIC_INTRANET=intranet
NEXT_PUBLIC_API_KEY=your-secret-api-key
NEXT_PUBLIC_URL_SECRET=your-url-secret
NEXT_PUBLIC_PROJECT_VERSION=V1.0
```

Customizable Options:

- NEXT_PUBLIC_API_URL - Change this if deploying to a different backend.
- NEXT_PUBLIC_PROJECT_VERSION - Update the version for tracking.

Backend Configuration (server/intranet_api/.env)

- Stores authentication secrets, database connections, and API settings.

```
DATABASE_URL=postgresql://<username>:<password>@localhost:5432/db_intranet
AT_SECRET=your-access-token-secret
RT_SECRET=your-refresh-token-secret
AT_EXP=7m
RT_EXP=7d
NODE_ENV=development
PORT=8081
```

WMC Employee Portal

```
CLIENT_URL=http://localhost:3000
API_KEY=your-api-key
PROD_CLIENT="https://your-production-url.com"
DEV_CLIENT="http://localhost:3000"
```

Customizable Options:

- DATABASE_URL - Update to match the production database.
- NODE_ENV - Set to production when deploying.
- PROD_CLIENT & DEV_CLIENT - Set frontend URLs accordingly.

Docker Configuration (Dockerfile & docker-compose.yml)

- Handles containerized deployment for the frontend and backend.

Dockerfile

- Defines how the **frontend** and **backend** are built inside a container.

docker-compose.yml

- Orchestrates multiple services (client & server).
- Manages ports, volumes, and dependencies.

```yaml
version: "3.8"

services:
  server:
    build:
      context: ./server/intranet_api
      dockerfile: Dockerfile
    container_name: intranet_api
    ports:
      - "8081:8081"
    volumes:
      - ./server/intranet_api/uploads:/usr/src/app/uploads
      - ./server/intranet_api/logs:/usr/src/app/logs
```

```
 client:
   build:
     context: ./client/intranet_fe
     dockerfile: Dockerfile
   container_name: intranet_ui
   ports:
     - "3000:3000"
```

Customizable Options:

- Change **ports** if running multiple apps on the same machine.
- Modify **volumes** to map persistent storage locations.

2. How to Customize the Code for Specific Use Cases

1. Changing the Database

- Modify prisma/schema.prisma to add new fields or tables.

Run the following command to apply changes:

```
npx prisma migrate dev --name _your_update_description
```

2. Modifying API Endpoints

- The backend routes are located in server/intranet_api/src/.
- To add a new API, run:

```
nest g resource endpoint_name
```

3. Customizing Frontend UI

- The frontend UI components are inside client/intranet_fe/src/app/.

- Modify pages inside pages/ and components inside components/.

4. Adjusting Authentication Rules

- Located in server/intranet_api/src/auth/
- Modify jwt.strategy.ts to change access control policies.

3. Best Practices and Recommended Configurations

1. Security Best Practices

Use strong JWT secrets

- Replace default secrets in .env with randomly generated values.
- Use a password manager or a secret vault for storage.

Restrict API Access with API Keys

- Ensure only authenticated users can call the API.

Sanitize User Inputs

- Use **class-validators** from nest js and create functions for input sanitations

2. Performance Optimization

Enable Database Indexing

Indexes are created by adding relationships in schema.prisma and migrating the updates to the database

Use Lazy Loading in the Frontend

Load images and components only when needed:

```
import dynamic from 'next/dynamic';
const YourComponent= dynamic(() => import('../components/YourComponent));
```

Optimize Docker Images

- Use **Alpine Linux** base images for smaller container sizes.

Troubleshooting and FAQS

Common issues and their solutions

The most common issue is that when a user has logged in to two devices and the other device logs out, it will remove the refresh token from the database which will cause issues to the other device where the user is logged in. To prevent this, always check the refresh token when the user jumps back to the web app. If the refresh token does not exist in the user's row, logout the user and remove access token and refresh token in the cookies and local storage.

Performance Optimization

This section includes the way on how to improve the performance of the application. Which might require changes to the codebase of the particular layer of the app.

- **Backend Caching:** By adding caching to API in the backend which is frequently called, it will improve performance and data load times
- **Frontend Enhancement:** Instead of fetching posts one by one inside the list of the Post Container Component, insert all of the posts to the post container and make the comments and replies centralized using react query.
- **Database Indexing:** There might be missing indices in the database. Which might require being indexed as the data grows.
- **Map Usage:** Currently, the objects are stored in a list. Replacing some of the lists with Maps might improve lookups.
- **Signal Separation:** Currently, the app has only one signal. That signal is the indicator if react-query will re-fetch the data when something changes in the database. Creating different signals will reduce API calls and not do unnecessary updates.

Security Considerations

Potential security vulnerabilities and their mitigation

The WMC Employee Portal's Security Vulnerability is that it does not encrypt data before sending the data to the server. So when a user is capable of using chrome's developer tools, the user will be able to see the payload of a request whenever the frontend sends data to the server.

Always use .env to store secrets. Therefore only the app can access them in production. Also prevent pushing .env files to the git repository. Enforcing this method will secure the app since the bindings of the app are enabled by the use of the secrets which are stored in the .env file.

There are different permissions based on the user's role, ensuring that QM and HR departments are the only departments that can create, edit, delete posts and folders while other departments are for viewing only. IT Users are granted access to use the Dashboard of the application which can be used to modify data in case of emergency (eg. deactivating a user).

In the backend, the CORS Policy must be enabled properly by allowing only the WMC Employee Frontend to make API calls in the backend server. This prevents other frontends from gaining access to the data of the WMC Employee Portal.

When changing the configuration for Authentication and Authorization, always use a randomly generated AT_SECRET and RT_SECRET.

WMC Employee Portal

Limitations and Future Enhancements

Known Limitations

- **Role Management:** Posting is currently restricted to QM, HR, and IT.
- **Feedback System:** Feedback updates are not processed in real-time.
- **Notifications:** Users do not receive notifications in real-time.
- **Post Viewing Control:** Access to posts is determined by the user's employee level and department.
- **IT Dashboard:** Needs additional graphs and UI enhancements.
- **User Read Monitoring:** Cannot be exported or printed.
- **Idle Logout:** Users are not automatically logged out after being idle for a set period.
- **Update Logs:** Only visible in the database, lacking a UI display.
- **Post Loading:** Posts are loaded individually on initial load, causing multiple API calls.
- **Dashboard Responsiveness:** The dashboard is not fully optimized for different screen sizes.
- **IT Folder & Division Controls:** No dedicated controls for folder and division management.
- **Caching:** Implemented only on the frontend, not in the backend.
- **Data Parsing:** Some endpoints lack proper data type parsing and require a dedicated parsing service or method.
- **Image Processing:** During post creation and updates, image organization is handled on the frontend instead of the backend.

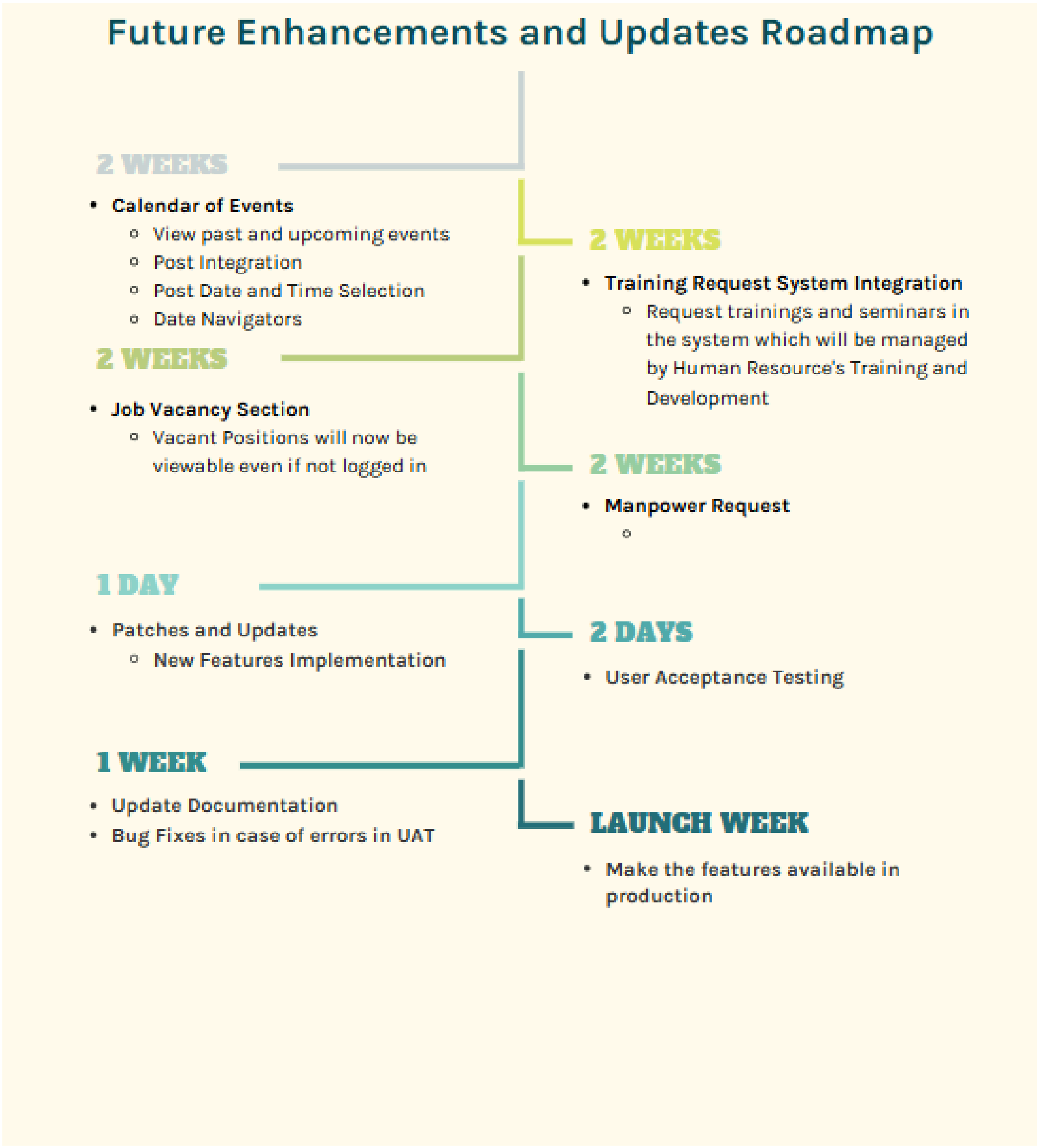
Possible Areas for Improvement

- **Optimize API Calls:** Implement post injection from findAll endpoints to minimize unnecessary API requests.
- **API Pagination:** Add pagination to endpoints that currently lack it.
- **Improve Dashboard Responsiveness:** Ensure the IT dashboard is fully responsive.

WMC Employee Portal

- **Enhance Data Parsing:** Apply data type parse pipes to all endpoints to reduce processing overhead in the service layer.
- **Backend Caching:** Implement caching for endpoints that return large datasets.
- **Image Handling:** Shift image organization from the frontend to the backend during post creation and updates.
- **UI Enhancements:** Improve overall design and user experience.
- **Flexible Post Permissions:** Enhance post access rules by allowing custom department-based permissions (e.g., an HR user can view private QM posts if granted access).

WMC Employee Portal



Future Enhancements and Updates Roadmap

2 WEEKS

- Calendar of Events
    - View past and upcoming events
    - Post Integration
    - Post Date and Time Selection
    - Date Navigators

2 WEEKS

- Training Request System Integration
    - Request trainings and seminars in the system which will be managed by Human Resource's Training and Development

2 WEEKS

- Job Vacancy Section
    - Vacant Positions will now be viewable even if not logged in

2 WEEKS

- Manpower Request
    -

1 DAY

- Patches and Updates
    - New Features Implementation

2 DAYS

- User Acceptance Testing

1 WEEK

- Update Documentation
- Bug Fixes in case of errors in UAT

LAUNCH WEEK

- Make the features available in production

WMC Employee Portal

FAQS and Troubleshooting

Question: How do we backup the database and when do we do backups of the database?

Answer: To backup the database, use the command
pg_dump -U westlake -W -h localhost -F c -b -v -f /home/superuser/db_intranet_3_14_25.sql db_intranet daily. You can also set a cron job for the daily database backup at the specified time.

Question: When the site goes down, what are the steps that you will do in order to restore it?

Answer: First I will ask help from <insert name here> to announce that the WMC Employee Portal is currently down and is undergoing fixes and will give an estimated time of recovery. Then I will open the Putty for SSH to check the status of the container if it is still running. If it has stopped and has not appeared in the running containers, I will navigate to the directory of the backend and check the logs to see what has caused the errors. If data updates were the reason for the server to stop, I will adjust the code based on the error and update my git repository. After pushing my updates to the git repository, if the checks have passed, I will now merge the code from the development branch to the main branch and will update the production and make the WMC Employee Portal go live again.

Question: When applying updates what steps would you do?

Answer: I will ask for help from <insert name here> to announce that the WMC Employee Portal will undergo maintenance for a set period of time and will give an estimated time for the website to go back. After passing the health checks of my frontend and unit tests in my backend, I will now merge the updates to the main branch from the development branch. After merging the branches, I will now use Putty to pull the updates from my git repository and stop the containers and rebuild the containers to push the updates to the web application.

Question: Since there are photos included in here, how do you pull the image files from the server?

WMC Employee Portal

Answer: I am using a command to pull the images from the server using the command:

pscp -r superuser@10.10.10.30:/home/superuser/intranet/server/intranet_api/uploads/post
C:\Users\Michael.Tiqui\Documents\test then to upload pictures run the command,
pscp C:\Users\Michael.Tiqui\Documents\test\your_image.jpg
superuser@10.10.10.30:/home/superuser/intranet/server/intranet_api/uploads/post/ in
your windows terminal

Question: If the data from the database disappeared and the pictures disappeared in the server, what steps would you do to retrieve it?

Answer: Using the backups I made, I will insert the backup to the live database and push the files back to the server using my terminal. To push the images you can run
pscp -r C:\Users\Michael.Tiqui\Documents\test\
superuser@10.10.10.30:/home/superuser/intranet/server/intranet_api/uploads/post/test