First iteration of Meal Headcount Planner (MHP)

Niloy Gabriel Gomes

Feb 6, 2026

Overview

Build a web app to track daily meal participation for 100+ employees, replacing Excel-based tracking.

Tech Stack

Frontend: React + Vite

Backend API: Python (FastAPI)

Authentication: Basic login with roles (Employee / Team Lead / Admin/Logistics)

Storage: File-based JSON for the first iteration

Packaging: Run locally

Development Tools:

- Backend: pip + requirements.txt
- Frontend: npm or pnpm
- CORS middleware enabled for local development

Summary

This document contains a comprehensive, development roadmap for building the first iteration of the Meal Headcount Planner (MHP). It outlines the complete technical implementation using Python-FastAPI for the backend and React for the frontend, including detailed guidance on project structure, data models, authentication, API, endpoints, and UI components. The plan breaks down the development process into actionable phases, from initial setup through local deployment, with specific implementation details for each layer of the stack. Additionally, it includes key design decisions, success criteria, and clearly defines what features are in-scope versus deferred to future iterations.

Development Steps

1. Project Setup

Initialize project repository with chosen tech stack; repository name: **Tasks-Repo_CEA-2026-Niloy/Task1_mhp-app**. Proceed with setting up the following folder structure:

```
mhp-app/
├── backend/
│ ├── app/
│ │ └── routers/
│ ├── data/
│ │ ├── users.json
│ │ └── meal_participation.json
│ ├── requirements.txt
│ └── .env
├── frontend/
│ ├── src/
│ │ ├── components/
│ │ ├── pages/
│ │ ├── services/
│ │ ├── context/
│ │ ├── App.jsx
│ │ └── main.jsx
│ ├── package.json
│ └── .env
└── README.md
```

Configure development environment and dependencies and create the .env template for configuration.

2. Data Models and Storage

Define the following models in `models.py` using the Pydantic validation library:

- User - id, name, password_hash, role, department
- MealParticipation - id, user_id, date, meal_type, is_participating
- MealType(enum) - Lunch, Snacks, Iftar, EventDinner, OptionalDinner
- UserRole(enum) - Employee, TeamLead, Admin

Implement file-based JSON storage utilities in storage.py. Use `{load_json(filename)}` to read data, `{save_json(filename, data)}` to write data, and `{get_by_id(); create(); update(); delete()}` as helpers. Create seed data for testing purposes.

3. Authentication and Authorization

- Build the login system in auth.py, using password hashing with passlib(bcrypt). JWT token generation with python-jose, and token verification dependency.
- Implement `{get_current_user()}` dependency and create role-based permission dependencies:
  `{require_employee(); require_team_lead(); require_admin()}`.
- Implement FastAPI `{Depends()}` decorator to protect routes.

4. Backend API Development

Create routers in routers/ directory:

- `auth.py` - Authentication endpoints
- `meals.py` - Meal participation endpoints
- `users.py` - User management endpoints

Core endpoints:

- `POST /api/auth/login` - User authentication (returns JWT)
- `GET /api/meals/today` - Fetch today's meal types and user's participation status
- `PUT /api/meals/participation` - Employee opts in/out of meals
- `POST /api/meals/participation/admin` - Team Lead/Admin updates on behalf of employee

- `GET /api/meals/headcount/today` - Get meal headcount totals (Admin/Logistics only)
- `GET /api/users/me` - Current user profile

FastAPI features to be used:

- Automatic OpenAPI docs at `/docs`
- Request/response validation with Pydantic
- Dependency injection for auth
- CORS middleware configuration

5. Frontend Development

Initialize a React app with Vite and install dependencies using react-router-dom, axios. Create API service layer in `services/api.js.` Set up the auth context for global user state.

Pages/Views:

- `LoginPage.jsx` - Login form
- `EmployeeDashboard.jsx`
    - Today's date display
    - List of meals with opt-in/out toggles
    - Confirmation feedback on updates
- `AdminDashboard.jsx`
    - Headcount summary view (totals per meal type)
    - Option to update employee participation
    - Employee search/filter functionality

Components:

1. `PrivateRoute.jsx` - Protected route wrapper
2. `MealCard.jsx` - Individual meal participation card/toggle
3. `HeadcountTable.jsx` - Summary table for admin
4. `Navbar.jsx` - Navigation with user role indicator
5. `Loading.jsx`, `ErrorMessage.jsx` - UI states

Implement auth context for user session and token for state management. Local state for meal participation data and API calls wrapped in try-catch with loading states.

6. Other Logic Implementation

Default all employees to "opted-in" for all meals, while also allowing employees to toggle participation status. Prevent duplicate entries for the same user/date/meal. Calculate real-time headcount aggregations, and validate role permissions for admin actions.

7. Documentation, Testing and Validation

Test authentication flow for all roles and feature verifications.

- Verify employee can opt-in/out correctly
- Verify admin can update on behalf of employees
- Test headcount calculations accuracy

Add a README with setup instructions and API documentation (endpoints, payloads, responses). Add an user guide for each role with a sample `.env` configuration.

8. Local Deployment

- Create startup scripts:
  - Backend: `uvicorn app.main:app --reload --port 8000`
  - Frontend: `npm run dev` (Vite dev server on port 5173)
- Document environment variables needed:
  - Backend `.env`: `SECRET_KEY`, `ALGORITHM`, `ACCESS_TOKEN_EXPIRE_MINUTES`
  - Frontend `.env`: `VITE_API_URL=http://localhost:8000`
- Provide instructions for first-time setup:

```
# Backend
cd backend
python -m venv venv
source venv/bin/activate  # or venv\Scripts\activate on Windows
pip install -r requirements.txt
uvicorn app.main:app --reload

# Frontend (new terminal)
cd frontend
npm install
npm run dev
```

- Test end-to-end on fresh installation

Key Design Decisions for Iteration 1

1. **Default opt-in approach** - All employees assumed participating unless they opt out
2. **Same-day focus** - Only "today's" meals visible and editable (no historical/future view yet)
3. **Simple storage** - File-based JSON (easy to migrate to DB later)
4. **Minimal validation** - Basic role checks; advanced cutoff logic deferred to later iterations
5. **Real-time counts** - Headcount calculated on-demand from participation records

Success Criteria

- ↳ Employees can log in and opt-out of today's meals
- ↳ Team Leads/Admins can update participation for any employee
- ↳ Logistics can view accurate headcount totals per meal type
- ↳ System handles 100+ concurrent users
- ↳ All three user roles work correctly
- ↳ Data persists across app restarts