TaskFlow - Project Management Application

A modern, minimal project management application built with FastAPI and Next.js. This application demonstrates full-stack development skills with role-based access control, real-time Kanban boards, and clean architecture.

🚀 Technology Stack

- **Backend**: FastAPI with Python 3.9+
- **Frontend**: Next.js 14+ with React 18+
- **Database**: PostgreSQL with SQLAlchemy (async)
- **Authentication**: JWT tokens
- **Styling**: Tailwind CSS
- **Deployment**: Docker + Docker Compose

👥 User Roles & Permissions

Admin

- Manage all users and projects in the system
- Full access to all functionality

Project Manager

- Create and manage projects
- Invite team members to projects
- Assign and manage tasks

Team Member

- View assigned projects
- Update their tasks and add comments
- Cannot create projects (must be invited)

📋 Core Features

- **User Authentication** with JWT tokens
- **Role-based Access Control**
- **Project Management** with Kanban boards
- **Task Management** with drag & drop
- **Team Collaboration** with comments
- **Responsive Design** for all devices

🔄 User Management Flow

1. **Registration**: Anyone can register as Project Manager or Team Member
2. **Admin Creation**: Admin accounts created manually in database
3. **Project Creation**: Only Project Managers and Admins can create projects

4. **Team Building**: Project Managers invite registered users to their projects
5. **Collaboration**: Team members work on assigned tasks and projects

Database Schema

Users Table

```
users:
- id (UUID, primary key)
- email (unique, not null)
- password_hash (not null)
- first_name (not null)
- last_name (not null)
- role (enum: 'admin', 'project_manager', 'team_member')
- avatar_url (nullable)
- is_active (boolean, default true)
- created_at (timestamp)
- updated_at (timestamp)
```

Projects Table

```
projects:
- id (UUID, primary key)
- name (not null)
- description (text, nullable)
- color (string, default '#3B82F6')
- created_by (UUID, foreign key to users.id)
- status (enum: 'active', 'completed', 'archived', default 'active')
- created_at (timestamp)
- updated_at (timestamp)
```

Project Members Table

```
project_members:
- id (UUID, primary key)
- project_id (UUID, foreign key to projects.id)
- user_id (UUID, foreign key to users.id)
- role (enum: 'manager', 'member', default 'member')
- joined_at (timestamp)
- unique constraint on (project_id, user_id)
```

Tasks Table

```
tasks:
- id (UUID, primary key)
```

```
- title (not null)
- description (text, nullable)
- status (enum: 'todo', 'in_progress', 'review', 'done', default 'todo')
- priority (enum: 'low', 'medium', 'high', default 'medium')
- project_id (UUID, foreign key to projects.id)
- assigned_to (UUID, foreign key to users.id, nullable)
- created_by (UUID, foreign key to users.id)
- due_date (date, nullable)
- created_at (timestamp)
- updated_at (timestamp)
```

Task Comments Table

```
task_comments:
- id (UUID, primary key)
- task_id (UUID, foreign key to tasks.id)
- user_id (UUID, foreign key to users.id)
- content (text, not null)
- created_at (timestamp)
- updated_at (timestamp)
```

🛠 API Endpoints

Authentication

- `POST /api/v1/auth/register` - User registration
- `POST /api/v1/auth/login` - User login
- `POST /api/v1/auth/refresh` - Refresh access token
- `POST /api/v1/auth/logout` - User logout

Users

- `GET /api/v1/users/me` - Get current user
- `PUT /api/v1/users/me` - Update current user
- `GET /api/v1/users` - Get all users (Admin only)
- `GET /api/v1/users/available` - Get available users for project invitation

Projects

- `POST /api/v1/projects` - Create project (PM/Admin only)
- `GET /api/v1/projects` - Get user's projects
- `GET /api/v1/projects/{project_id}` - Get project details
- `PUT /api/v1/projects/{project_id}` - Update project (PM/Admin only)
- `DELETE /api/v1/projects/{project_id}` - Delete project (PM/Admin only)
- `POST /api/v1/projects/{project_id}/invite` - Invite user to project
- `GET /api/v1/projects/{project_id}/members` - Get project members

Tasks

- `POST /api/v1/tasks` - Create task
- `GET /api/v1/tasks` - Get tasks (with filters)
- `PUT /api/v1/tasks/{task_id}` - Update task
- `DELETE /api/v1/tasks/{task_id}` - Delete task
- `POST /api/v1/tasks/{task_id}/comments` - Add task comment
- `GET /api/v1/tasks/{task_id}/comments` - Get task comments

Frontend Pages

Public Pages

- `/` - Landing page
- `/login` - User login
- `/register` - User registration

Protected Pages

- `/dashboard` - User dashboard overview
- `/projects` - Projects list
- `/projects/new` - Create new project (PM/Admin only)
- `/projects/[id]` - Project Kanban board
- `/projects/[id]/settings` - Project settings (PM/Admin only)
- `/tasks/[id]` - Task details
- `/profile` - User profile

Key UI Components

Kanban Board

- 4 columns: Todo, In Progress, Review, Done
- Drag & drop task management
- Color-coded priorities
- Task cards with assignee and due date

Task Management

- Quick task creation
- Detailed task modal
- Assignment to team members
- Priority and status management
- Comment system

User Interface

- Clean, modern design with Tailwind CSS
- Fully responsive (mobile, tablet, desktop)
- Loading states and error handling
- Form validation

🚀 Getting Started

Prerequisites

- Docker and Docker Compose
- Node.js 18+ (for local development)
- Python 3.9+ (for local development)

Quick Start with Docker

```
# Clone the repository
git clone <repository-url>
cd taskflow

# Start all services
docker-compose up -d

# Access the application
# Frontend: http://localhost:3000
# Backend API: http://localhost:8000
# API Documentation: http://localhost:8000/docs
```

Local Development Setup

Backend Setup

```
cd backend

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Set environment variables
cp .env.example .env
# Edit .env with your settings

# Run database migrations
alembic upgrade head

# Seed database with sample data
python scripts/seed_db.py

# Start development server
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

Frontend Setup

```
cd frontend

# Install dependencies
npm install

# Set environment variables
cp .env.local.example .env.local
# Edit .env.local with your settings

# Start development server
npm run dev
```

🔧 Environment Variables

Backend (.env)

```
DATABASE_URL=postgresql+asyncpg://user:password@localhost:5432/taskflow
SECRET_KEY=your-secret-key-here
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=15
REFRESH_TOKEN_EXPIRE_DAYS=7
CORS_ORIGINS=http://localhost:3000
```

Frontend (.env.local)

```
NEXT_PUBLIC_API_URL=http://localhost:8000
NEXTAUTH_SECRET=your-nextauth-secret
NEXTAUTH_URL=http://localhost:3000
```

🎯 Default Users (Seeded Data)

After running the database seed script:

```
Admin:
- Email: admin@taskflow.com
- Password: Admin123!

Project Manager:
- Email: pm@taskflow.com
- Password: Manager123!

Team Member:
```

```
- Email: member@taskflow.com
- Password: Member123!
```

Project Structure

```
taskflow/
├── backend/
│   ├── app/
│   │   ├── api/
│   │   ├── core/
│   │   ├── models/
│   │   ├── schemas/
│   │   ├── services/
│   │   └── main.py
│   ├── alembic/
│   ├── scripts/
│   ├── requirements.txt
│   └── Dockerfile
├── frontend/
│   ├── components/
│   ├── pages/
│   ├── styles/
│   ├── utils/
│   ├── package.json
│   └── Dockerfile
├── docker-compose.yml
└── README.md
```

Testing

Backend Tests

```
cd backend
pytest
```

Frontend Tests

```
cd frontend
npm test
```

Deployment

Production Build

```
# Build all services
docker-compose -f docker-compose.prod.yml build

# Start production services
docker-compose -f docker-compose.prod.yml up -d
```

Environment Setup

1. Set production environment variables
2. Configure database with proper credentials
3. Set up reverse proxy (nginx recommended)
4. Enable HTTPS with SSL certificates

Security Features

- JWT authentication with refresh tokens
- Password hashing with bcrypt
- CORS protection
- Input validation and sanitization
- Role-based access control
- SQL injection prevention with SQLAlchemy

Performance Considerations

- Async/await throughout the backend
- Database indexing on frequently queried fields
- Optimistic UI updates on frontend
- Lazy loading and code splitting
- Image optimization
- API response caching

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests if applicable
5. Submit a pull request

License

This project is for educational and portfolio purposes.

Learning Objectives Demonstrated

This project showcases proficiency in:

- **Full-stack Development**: FastAPI + Next.js integration

- **Database Design**: Relational data modeling with proper constraints
- **Authentication**: JWT implementation with refresh tokens
- **Authorization**: Role-based access control
- **Modern React**: Hooks, Context API, and TypeScript
- **API Design**: RESTful endpoints with proper HTTP methods
- **UI/UX**: Responsive design with modern interactions
- **DevOps**: Docker containerization and development workflows
- **Security**: Input validation, password hashing, and CORS
- **Testing**: Unit and integration testing strategies

Perfect for demonstrating modern web development skills in technical interviews!