Next Step Project Documentation

Project Overview

Next Step is a productivity application designed to help professionals manage tasks and priorities. The core concept is to intelligently suggest the most logical next task to work on, considering deadlines, priorities, and long-term projects.

Key Features

- Smart task prioritization (initially rule-based, with ML capabilities planned)
- Cross-platform availability (web and mobile)
- Task and project management
- Calendar integration
- Email integration for task extraction (planned)
- Interactive interface with drag-and-drop capabilities

Project Architecture

The project is structured as a monorepo with the following components:

Copy

```
next-step-app/
├── packages/
│   ├── server/       # Node.js backend
│   ├── web/          # React web application
│   ├── mobile/       # React Native mobile app
│   └── common/       # Shared code between apps
├── docs/             # Documentation
└── scripts/          # Build and deployment scripts
```

Technology Stack

Backend

- **Language**: TypeScript
- **Framework**: Express.js
- **Database**: SQLite (changed from MongoDB for simplicity in MVP)

- **ORM**: Sequelize
- **Authentication**: JWT

Web Frontend

- **Framework**: React with TypeScript
- **State Management**: Redux Toolkit
- **Styling**: Styled Components
- **Routing**: React Router

Mobile

- **Framework**: React Native with Expo
- **State Management**: Redux Toolkit

Implementation Steps

1. Initial Setup

1. Created GitHub repository: https://github.com/Fr33Fe77et/next-step-app
2. Set up monorepo structure with packages for server, web, mobile, and common code
3. Configured TypeScript in all packages

2. Backend Implementation

1. Set up Express server with TypeScript
2. Initially planned to use MongoDB with Mongoose, but switched to SQLite with Sequelize for easier development
3. Implemented user authentication with JWT
4. Created data models for:
   - Users (name, email, password)
   - Tasks (title, description, due date, priority, status, etc.)
5. Implemented RESTful API endpoints:
   - User registration and login
   - CRUD operations for tasks
   - Next task recommendation endpoint with rule-based logic

Database Change: MongoDB to SQLite

Decision: Switched from MongoDB to SQLite for the MVP stage.

Rationale:

- Eliminates need for external database server installation
- Simpler setup for development
- Self-contained file-based database
- Adequate for MVP data requirements
- Easier to migrate to PostgreSQL later when needed

Implementation:

- Replaced Mongoose models with Sequelize models
- Updated controllers to use Sequelize query syntax
- Added SQLite database file to project

3. Web Frontend Setup

1. Created React application with TypeScript
2. Set up Redux store with slices for:
   - Authentication state
   - Task management
3. Implemented basic UI components:
   - Button
   - Input
   - Header
4. Created initial pages:
   - Home page
   - Login page
   - Registration page
   - Dashboard page
5. Set up routing with React Router

4. Mobile Setup

1. Initialized React Native application using Expo
2. Set up basic project structure

Current Status

- Backend server running with SQLite database
- Basic API endpoints implemented
- Authentication system working
- Web frontend partially implemented
- Mobile setup initiated

Next Steps

1. Complete controller updates to work with Sequelize
2. Finish web frontend implementation:
   - Task list page
   - Task creation/editing form
   - Calendar integration
3. Implement mobile app screens
4. Add email integration
5. Refine task prioritization algorithm
6. Set up testing
7. Deploy MVP

Architectural Decisions for Future ML Capabilities

- Event sourcing pattern for storing user actions
- Feature flag system for gradual rollout of ML features
- Data collection framework from day one
- Extensible prioritization service that can be replaced with ML

Database Schema

User Model

- id (UUID)
- name (String)
- email (String, unique)
- password (String, hashed)

- timestamps

Task Model

- id (UUID)
- userId (UUID, foreign key)
- title (String)
- description (Text, optional)
- dueDate (Date, optional)
- priority ('low', 'medium', 'high')
- status ('pending', 'in_progress', 'completed')
- category (String, optional)
- estimatedTime (Integer, minutes, optional)
- actualTime (Integer, minutes, optional)
- isRecurring (Boolean)
- recurringPattern (String, optional)
- tags (String, comma-separated)
- timestamps

API Endpoints

Authentication

- POST /api/users - Register new user
- POST /api/users/login - Login user
- GET /api/users/profile - Get user profile (protected)

Tasks

- GET /api/tasks - Get all user tasks (protected)
- POST /api/tasks - Create task (protected)
- GET /api/tasks/
  - Get single task (protected)
- PUT /api/tasks/
  - Update task (protected)
- DELETE /api/tasks/

- Delete task (protected)

- GET /api/tasks/next - Get next task recommendation (protected)