# MyTasks

## Architecture Notes

### 1. Introduction

This document captures the key decisions that shaped the project: the technologies selected, the code organization, and the reasoning behind design patterns. The goal is not merely to list tools, but to explain why those choices were made so that anyone joining the team can quickly understand how the system works and where it can evolve.

Think of this as a living reference — a practical guide to help maintain, improve, and grow the application while keeping context for future contributors.

### 2. Overall Architecture

The system follows a decoupled Frontend / Backend architecture.

- Frontend
    - React (v19) with Vite.
    - Mobile-first design and responsive UI strategy.
- Backend
    - .NET Isolated Functions exposing a REST API.
    - Entity Framework Core with SQL Server as the primary data store.
    - Authentication and authorization handled by Microsoft Entra ID (Azure AD), with permission validation on both client and server.

### 3. Patterns and Cross-cutting Concerns

- Repository Pattern: abstracts data access for Users, Tasks, and Notifications.
- Factory Pattern: enables flexible creation of notification types.
- Dependency Injection: used for services, repositories, and DB context to improve testability and modularity.
- Middleware:
    - JwtValidationMiddleware: centralizes JWT token validation.
    - GlobalExceptionMiddleware: provides consistent error handling and logging.

### 4. Backend — Key Decisions

- Persistence
    - SQL Server with EF Core using Code First migrations.
    - Automatic migrations and initial DB creation on first run for developer convenience.

- Authentication / Authorization
    - Managed via Microsoft Entra ID with two separate App Registrations in Azure:
        1. Frontend App Registration
            - Represents the React client.
            - Configured to request tokens that allow secure calls to the backend API.
        2. Backend App Registration
            - Represents the protected API.
            - Granted selective Microsoft Graph permissions to safely read user profile information.
    - This separation keeps authentication flows decoupled and secure: the frontend obtains tokens, the backend validates them and enriches user data via Graph where needed.
- Notifications
    - A **stored procedure** (sp_AddNotification) is used to optimize write performance and return the inserted row in a single operation.
    - Current publisher implementation: NoOperationPublisher — designed as a simple placeholder that can be replaced with SignalR, email, or push notification senders in the future.
- API design
    - Pagination and filtering are implemented to ensure efficiency and predictable performance on list endpoints.

**5. Frontend — Key Decisions**

- Framework: React (mobile-first).
- State and data handling:
    - TanStack Query for server state and caching.
    - React Hook Form for form handling and validation.
    - Custom hooks to encapsulate backend calls and prevent code duplication.
- Authentication: integrated with Azure AD via its dedicated App Registration to retrieve tokens and access the backend securely.

**6. Development Workflow**

1. Initial backend scaffold with placeholders for business rules.
2. Configuration of DB, middleware, and dependency graph.

3. Frontend mockups and iterative refinement of business rules.
4. Backend adjustments to match finalized requirements.
5. Integration of authentication and user validation through Entra ID.
6. Automated and manual testing.
7. CI/CD pipeline set up with GitHub Actions.
8. Documentation created and continuously improved.
9. Ongoing refinement of code and documentation as the project matures.

**7. DevOps and Branching Strategy**

- Branching:
    - main: stable, production-ready releases.
    - develop: integration and pre-release testing.
- CI/CD:
    - GitHub Actions run unit tests on each pull request targeting develop.
    - Pipeline configured to promote tested changes through the release process.

**8. Areas for Improvement (Roadmap)**

- Soft delete pattern for persistence to preserve history and enable safer deletes.
- Optimistic UI updates on the frontend to improve perceived responsiveness.
- Infinite scroll or cursor-based pagination for long lists to improve UX.
- Replace polling-based notifications with SignalR / WebSockets for real-time updates.
- Introduce API Management for centralized security, rate limiting, and observability.
- Add telemetry (distributed traces and metrics) to track end-to-end performance.
- Harden Role-Based Access Control policies and provide fine-grained permission checks at the API level.
- Implement integration tests.

**9. Business Rules Definition**

Business rules were developed iteratively, informed by common patterns in task and note management apps and adapted to the specific needs of the project.

Process:

1. Initial research on similar applications to identify common features and UX patterns.

2. Definition of key screens to cover the primary user flow.
3. Refinement of business rules based on those screens and feedback loops between frontend and backend teams.

Key screens and associated behaviors:

- Homepage / Dashboard
    - Provides an overview of tasks and key metrics.
    - Acts as the primary entry point with quick links to important actions.
- All Tasks
    - Full listing of tasks with filters by status and assignment (owner or scope).
    - Future enhancement: add date-range filtering and saved queries.
- Add Task
    - Form with client-side validations using React Hook Form.
    - Includes dynamic assignee selector that queries available users from the backend.
- Notifications
    - Displays notifications when tasks are created or updated.
    - Includes a notification counter and list view.
    - Current limitation: updates rely on polling.
    - Short-term improvement: swap polling for SignalR/WebSockets to deliver real-time updates and reduce load.
- User Profile
    - Allows users to update basic profile information (currently name).
    - Designed to scale: additional fields such as avatar, address, and phone can be added to the user provision table as needed.