Adnan Kotawala

Kotawala\_Adnan@yahoo.co.in

Task Management app

Architecture Document

Table of Contents

[Background 3](#_Toc169905422)

[Requirements 4](#_Toc169905423)

[Functional Requirements 4](#_Toc169905424)

[Non-Functional Requirements 4](#_Toc169905425)

[Executive Summary 5](#_Toc169905426)

[Clean Architecture Implementation 7](#_Toc169905427)

[Overview 7](#_Toc169905428)

[Principles to Follow 7](#_Toc169905429)

[Layer and Structure Implementation 7](#_Toc169905430)

[Detailed Implementation Instructions 8](#_Toc169905431)

[Overall Architecture 8](#_Toc169905432)

[Services 9](#_Toc169905433)

[Scaling 10](#_Toc169905434)

[Messaging 10](#_Toc169905435)

[Services Drill Down 11](#_Toc169905436)

[UI (Angular Front-End) 11](#_Toc169905437)

[**Role:** 11](#_Toc169905438)

[**Technology Stack** 11](#_Toc169905439)

[**Architecture:** 11](#_Toc169905440)

[**Implementation Instructions** 11](#_Toc169905441)

[Authentication Service 11](#_Toc169905442)

[**Role:** 11](#_Toc169905443)

[**Technology Stack** 11](#_Toc169905444)

[**Architecture:** 11](#_Toc169905445)

[**Implementation Instructions** 12](#_Toc169905446)

[Task Management Service 12](#_Toc169905447)

[**Role:** 12](#_Toc169905448)

[**Technology Stack** 12](#_Toc169905449)

[**Architecture:** 12](#_Toc169905450)

[**Implementation Instructions** 12](#_Toc169905451)

[User Management Service 12](#_Toc169905452)

[**Role:** 12](#_Toc169905453)

[**Technology Stack** 12](#_Toc169905454)

[**Architecture** 13](#_Toc169905455)

[**Implementation Instructions** 13](#_Toc169905456)

[Messaging 13](#_Toc169905457)

# Background

This document describes the Task Management Application’s architecture, a key product of our innovative development team.

The Task Management Application is designed to help users efficiently manage their daily tasks, offering functionalities to create, update, delete, and filter tasks. This application is essential for users seeking an organized approach to handle their to-do lists and project tasks, ensuring they can track progress and prioritize their workload effectively.

In the modern world, individuals and teams often juggle multiple tasks simultaneously, requiring a robust system to manage these activities. The Task Management Application addresses this need by providing a user-friendly interface for task management, including capabilities for user authentication, task prioritization, due dates, reminders, sorting, and searching.

The core functionalities include:

* **Task Creation**: Users can create tasks with a title, description, and status (e.g., "To Do," "In Progress," "Done").
* **Task Management**: Users can update the status of tasks, delete tasks, and view a comprehensive list of tasks.
* **Filtering and Sorting**: Tasks can be filtered by status and sorted to meet the user's preferences.
* **User Authentication**: Ensuring that only authorized users can access and manage their tasks.

Additionally, the application offers advanced features such as task prioritization, due dates, reminders, and user profiles with avatars. These enhancements aim to provide users with a comprehensive tool for managing tasks, improving productivity, and ensuring timely completion of important activities.

The application’s architecture focuses on leveraging modern technologies and best practices, including .Net Core for the backend, Angular for the frontend, and Azure for cloud services. The system is designed to be fast, reliable, and easy to maintain, adhering to SOLID design principles and employing well-established design patterns.

It is crucial for the development team to follow the architecture outlined in this document meticulously. This document will guide the team through the technology choices, design decisions, and implementation strategies that ensure the final product meets the highest standards of quality and performance. Should any uncertainties arise, consulting the Software Architect is recommended.

This document describes the system architecture of the Task Management Application, detailing the considerations and decisions that underpin its design. By adhering to this architecture, we aim to deliver a robust, scalable, and user-friendly task management solution.

# Requirements

## Functional Requirements

1. **User Authentication and Authorization**
   * Implement user registration, login, and role-based access control.
   * Ensure only authenticated users can access and manage tasks.
2. **Task Management**
   * Allow users to create tasks with a title, description, and status (e.g., "To Do," "In Progress," "Done").
   * Enable users to update task details and status.
   * Provide functionality to delete tasks.
   * Ensure tasks cannot be created without a title.
3. **Task Viewing and Filtering**
   * Display a list of tasks for the user.
   * Provide filtering options to view tasks by status (e.g., "All," "To Do," "In Progress," "Done").
4. **Advanced Task Features**
   * Allow users to prioritize tasks.
   * Enable setting due dates and reminders for tasks.
   * Implement sorting and searching capabilities for tasks.
5. **User Profile Management**
   * Allow users to manage their profiles, including updating avatars.

## Non-Functional Requirements

The following non-functional requirements were defined after discussions with the customer and are agreed upon by all the team members.

1. **Scalability:**
   * Support up to 100,000 users concurrently.
   * Handle up to 1,000,000 tasks in the system.
2. **Performance:**
   * Ensure average response time for task-related operations is under 200 milliseconds.
   * Ensure the application remains responsive under peak load conditions.
3. **Availability:**
   * Maintain an uptime of 99.9% (SLA: Gold).
4. **Data Integrity:**
   * Ensure zero data loss during operations.
   * Implement data backup and recovery mechanisms.
5. **Security:**
   * Implement strong encryption for sensitive data.
   * Adhere to best practices for securing APIs and user data.
   * Conduct regular security audits and vulnerability assessments.
6. **Usability:**
   * Ensure the application is intuitive and user-friendly.
   * Provide a responsive design that works seamlessly on both desktop and mobile devices.
7. **Maintainability:**
   * Write clean, well-documented, and maintainable code.
   * Follow coding best practices and conventions.
   * Implement automated testing and continuous integration to ensure code quality.
8. **Compliance:**
   * Adhere to data protection regulations (e.g., GDPR).

# Executive Summary

This document outlines the architecture of the Task Management Application, designed to provide users with an efficient and intuitive platform for managing tasks. The application allows users to create, update, delete, and filter tasks, ensuring that they can keep track of their work effortlessly. For example, users can prioritize their tasks, set due dates, and receive reminders, all within a responsive and user-friendly interface.

When designing the architecture, strong emphasis was placed on several key features:

* The application should be reliable
* The application should be highly scalable and performant
* The user interface should be intuitive and responsive

To achieve these features, the architecture leverages modern best practices and methodologies, ensuring high availability, performance, and maintainability. Here is a high-level overview of the architecture:

User

User

UI

User

**REST API / HTTP**

Azure Data Store

**Azure Cloud**

Authentication Service (JWT)

Task Management Service (CRUD)

**Azure App Service**

As illustrated in the diagram, the application comprises several independent, loosely coupled components, each responsible for specific tasks and communicating with each other through RESTful APIs. The components include:

* **UI (Angular Front-End)**: Provides a responsive and intuitive interface for users, hosted on Azure App Service.
* **Authentication Service**: Manages user authentication and authorization using JWT tokens, ensuring secure access to the application.
* **Task Management Service**: Handles all CRUD operations for tasks, ensuring data integrity and consistency.
* **Azure Data Store**: Stores task and user data securely, leveraging Azure SQL Database for high availability and performance.

All services are built as stateless services, meaning no data is lost if a service shuts down unexpectedly. The only place where data is stored persistently is the Azure Data Store, which serializes data to disk to protect it from service interruptions.

This architecture, combined with a modern development platform (.NET Core), will help create a robust, easy-to-maintain, and reliable system. This system is designed to scale with user demand and support the company in achieving its operational and financial goals for years to come.

# Clean Architecture Implementation

## Overview

Implement the Task Management Application using Clean Architecture principles to ensure a separation of concerns and high maintainability. This approach will make the system easy to understand, develop, and test, while also allowing flexibility for future changes.

## Principles to Follow

1. **Ensure Independence of Frameworks**: Design the architecture so it does not depend on any particular framework. This will allow flexibility in choosing tools and frameworks for future updates.
2. **Focus on Testability**: Build the application to facilitate testing of individual components in isolation, enhancing code reliability and quality.
3. **Maintain UI Independence**: Structure the application so that the UI can be changed without affecting the business rules, and vice versa. This will allow the UI to evolve independently.
4. **Ensure Database Independence**: Design the business rules to be independent of the database. This will make it easy to swap out the database or make changes to the data access layer without affecting the core logic.
5. **Ensure Independence of External Agencies**: Build the architecture so the business rules can be tested without the UI, database, web server, or any other external elements.

## Layer and Structure Implementation

Divide the Task Management Application into several layers, each with distinct responsibilities:

1. **Domain Layer**:
   * Implement the enterprise logic and types in this layer.
   * Ensure this layer is independent of any other layer.
   * This is the core of the application where business rules and entities reside.
2. **Application Layer**:
   * Define the operations the application can perform in this layer.
   * Interact with the domain layer to handle use cases and business logic.
3. **Infrastructure Layer**:
   * Implement interactions with external systems such as databases, file systems, and web services in this layer.
   * Ensure it contains implementations of repository interfaces defined in the application layer.
4. **Presentation Layer**:
   * This is the UI layer which users interact with.
   * Build it using Angular and ensure it communicates with the application layer via API calls.

## Detailed Implementation Instructions

* **Repositories**:
  + Start with in-memory repositories (e.g., InMemoryUserRepository, InMemoryTaskRepository).
  + Replace these with actual database repositories for persistent storage.
* **Dependency Injection**:
  + Use ASP.NET Core’s built-in DI container to manage dependencies.
  + Ensure loose coupling between components.
* **Entity Framework Core**:
  + Use EF Core for database operations.
  + Leverage its tools to interact efficiently with the Azure SQL Database.
* **Authentication**:
  + Manage authentication through a separate Authentication Service.
  + Use JWT tokens for secure communication between the UI and backend services.
* **Task and User Management**:
  + Segregate into distinct services to ensure a clear separation of concerns and maintainability.

By adhering to these Clean Architecture principles and following these detailed instructions, you will ensure the Task Management Application remains modular, testable, and maintainable. This approach will result in a robust solution that can adapt to future requirements with minimal friction.

# Overall Architecture

Here is the architecture diagram for the Task Management Application:

User

User

UI

User

**REST API / HTTP**

Azure Data Store

**Azure Cloud**

Authentication Service (JWT)

Task Management Service (CRUD)

**Azure App Service**

## Services

The architecture is comprised of the following services:

1. **UI**: The user interface is the front-end part of the application that users interact with. Built using Angular, it provides a responsive and intuitive experience. Users can create, update, delete, and filter tasks. It also allows users to set task priorities, due dates, and reminders. The UI handles user authentication, ensuring that only authorized users can access and manage their tasks.
2. **Authentication Service**: This service handles user login and authentication. It verifies user credentials and issues JSON Web Tokens (JWT) to authenticated users. These tokens are then used to secure communication between the user interface and other backend services, ensuring that only authorized users can perform actions.
3. **User Management Service:** The User Management Service handles all user-related operations except authentication. This includes user registration, profile updates, and retrieving user information. By offloading these tasks from the Authentication Service, we ensure a clear separation of concerns, making the system more modular and easier to maintain.
4. **Task Management Service**: This service is responsible for managing tasks. It performs all the necessary operations such as creating, reading, updating, and deleting tasks. When a user interacts with the UI to manage their tasks, the Task Management Service processes these requests and ensures the tasks are handled correctly and efficiently.
5. **Azure Data Store**: This is where all task and user data is stored. The Azure SQL Database ensures that data is persistently saved and can be efficiently retrieved or modified as needed. It supports the operations performed by the Task Management Service by providing reliable and scalable data storage.

## Scaling

This architecture allows easy scaling of the services as needed. Since each service is focused on specific tasks, they can be scaled independently, either automatically (by Azure App Service scaling features) or manually. The services are designed to be stateless, enabling scaling without downtime or code changes.

## Messaging

The various services communicate with each other using straightforward and efficient messaging methods to handle user requests:

1. **UI to Backend Services**: The UI sends requests to the backend services using REST API over HTTP. For example, when a user logs in, the UI sends their credentials to the Authentication Service. When the user wants to manage tasks, the UI sends requests to the Task Management Service.
2. **Authentication Service**: When the Authentication Service receives a login request, it verifies the credentials. If successful, it issues a JWT token back to the UI. This token is then used for all subsequent requests to ensure secure communication.
3. **Task Management Service**: The Task Management Service receives task management requests (like creating or updating a task) from the UI via REST API. It processes these requests and interacts with the Azure Data Store to store or retrieve task data as needed.
4. **Azure Data Store**: The Task Management Service communicates directly with the Azure Data Store to perform CRUD operations on task data. It ensures that all task data is correctly saved and can be retrieved when requested by the UI.

By using these methods, the Task Management Application ensures smooth and secure interactions between the user interface and backend services, providing a seamless experience for users. This architecture supports advanced task management features, including prioritization, due dates, and reminders, enhancing the application's usability and functionality.

# Services Drill Down

## UI (Angular Front-End)

### **Role:**

The UI is the front-end of the application that users interact with. It provides a responsive and intuitive experience, allowing users to manage their tasks, set priorities, due dates, and reminders, and authenticate their identity.

### **Technology Stack**

* **Framework**: Angular
* **Styling**: Bootstrap
* **Deployment**: Azure App Service

### **Architecture:**

The UI follows a component-based architecture, where each component is responsible for a specific part of the application. The architecture leverages Angular’s modularity to ensure maintainability and scalability.

### **Implementation Instructions**

* **Setup**: Use the Angular CLI to scaffold the project.
* **Styling**: Use Bootstrap for consistent UI elements and responsiveness.
* **Routing**: Implement Angular Router for navigation.
* **State Management**: Utilize Angular Services and RxJS for state management.
* **Deployment**: Deploy to Azure App Service using GitHub Actions for CI/CD pipelines.

## Authentication Service

### **Role:**

The Authentication Service is responsible for verifying user credentials and issuing JSON Web Tokens (JWT) for authenticated sessions. It ensures that only authorized users can access the system.

### **Technology Stack**

* **Framework**: ASP.NET Core
* **Authentication**: JWT
* **Deployment**: Azure App Service

### **Architecture:**

The Authentication Service uses a layered architecture:

* **API Layer**: Exposes REST endpoints for login and token issuance.
* **Business Logic Layer**: Handles the authentication logic and token generation.
* **Data Access Layer**: Interacts with the database to verify user credentials.

### **Implementation Instructions**

* **Dependency Injection**: Utilize ASP.NET Core’s built-in DI container.
* **Authentication**: Implement JWT authentication using Microsoft.AspNetCore.Authentication.JwtBearer.
* **Data Access**: Use Entity Framework Core for database operations.
* **Security**: Ensure all endpoints are secured and use HTTPS.

## Task Management Service

### **Role:**

The Task Management Service handles all operations related to tasks. It allows users to create, read, update, and delete tasks, as well as set priorities, due dates, and reminders.

### **Technology Stack**

* **Framework**: ASP.NET Core
* **Deployment**: Azure App Service
* **Database**: Azure SQL Database

### **Architecture:**

The Task Management Service employs a layered architecture:

* **API Layer**: Exposes REST endpoints for task operations.
* **Business Logic Layer**: Contains the logic for task management, including validation and processing.
* **Data Access Layer**: Interacts with the Azure SQL Database for CRUD operations.

### **Implementation Instructions**

* **Dependency Injection**: Use ASP.NET Core’s DI container.
* **Data Access**: Implement Entity Framework Core for database operations.
* **Logging**: Use built-in logging to track operations and errors.
* **Validation**: Ensure all task data is validated before processing.

## User Management Service

### **Role:**

The User Management Service handles all user-related operations, including user registration, profile updates, and retrieving user information. This service is essential for managing user data separately from authentication logic, improving modularity and maintainability.

### **Technology Stack**

* **Framework**: ASP.NET Core
* **Deployment**: Azure App Service
* **Database**: Azure SQL Database

### **Architecture**

The User Management Service follows a layered architecture:

* **API Layer**: Exposes REST endpoints for user registration, profile updates, and retrieval.
* **Business Logic Layer**: Manages user-related logic, including validation and processing.
* **Data Access Layer**: Interacts with the Azure SQL Database for user data operations.

### **Implementation Instructions**

* **Dependency Injection**: Utilize ASP.NET Core’s built-in DI container.
* **Data Access**: Implement Entity Framework Core for database operations.
* **Validation**: Ensure all user data is validated before processing.
* **Security**: Secure all endpoints and use HTTPS.

## Messaging

The services communicate using REST APIs over HTTP to handle user requests and ensure secure data exchange.

1. **UI to Authentication Service**:
   * The UI sends login requests to the Authentication Service via REST API. The Authentication Service verifies credentials and issues a JWT token, which is then used for all subsequent requests to secure communication.
2. **UI to Task Management Service**:
   * The UI sends task-related requests (create, read, update, delete) to the Task Management Service via REST API. The Task Management Service processes these requests and interacts with the Azure SQL Database to manage task data.
3. **UI to User Management Service**:
   * The UI sends user-related requests (registration, profile updates) to the User Management Service via REST API. The User Management Service processes these requests and interacts with the Azure SQL Database to manage user data.
4. **Authentication Service to Task Management Service**:
   * While direct communication between the Authentication Service and Task Management Service is minimal, JWT tokens issued by the Authentication Service are used by the Task Management Service to authenticate requests and ensure that only authorized users can perform operations.
5. **Authentication Service to User Management Service**:
   * The Authentication Service verifies credentials and manages tokens, while the User Management Service handles user-specific data. They work together to ensure that user operations are secure and authorized.