ABC Corp

The solution I created is based on **Clean** **architecture** and **Repository** **Pattern** to interact with database and **Dependency Injection.**

1. **Domain Layer:**

Consists of Entities (User.cs, Task.cs,ResponseFormatter.cs) and Interfaces(IUserRepo.cs, ITaskRepo.cs)

1. User.cs, Task.cs – defines the model for User and Tasks
2. ResponseFormatter.cs – a custom model for all response
3. IUserRepo.cs, ITaskRepo.cs – defines the CRUD operations on Users and Tasks
4. **Infrastructure Layer:**

This layer has repositories (UserRepository.cs, TaskRepository.cs) and interacts with databases (AppDbContext.cs) and other services (BlobService.cs,RedisPolicies.cs) and a custom middleware(AuthorizationMiddleware.cs).

The **CacheImplementation.cs** has sample business logic for Redis Cache implementation.

GetTaskByIdAsync() – If cache is present returns value from Cache else fetches the value from DB and sets the cache for 10 minutes.

CreateTaskAsync() – Removes the cache on the event of Task creation to avoid inconsistencies.

**Services**

* **RedisPolicies.cs**

Sets 2 policies for Cache implementation.

1. To retry on event of socket failure or timeout exception.
2. To fallback to Database operations on event of other exceptions

* **BlobService.cs**

Contains methods for Upload File and Download File in Azure Blob Storage.

**Middleware (AuthorizationMiddleware.cs)**

Custom middleware which sends a validation message for unauthorized requests.

1. **API Layer**

Contains API controllers (UserController.cs, TaskController.cs)

* UserController.cs - Contains controllers for User actions
* TaskController.cs - Contains controllers for Task actions

**TaskController Endpoints**

* **allTasks –** Returns all the Tasks
* **myTasks** – Returns Tasks of the User
* **createTask** – Creates a Task
* **updateTaskStatus** – Updates Task status
* **getTaskbyID** – Returns the Task of the mentioned ID
* **uploadFile** – Uploads a file in Azure Blob storage
* **download/{blobName}** – downloads the specified file

**UserController Endpoints**

* **createUser –** Creates an User
* **allUsers** – Returns all Users
* **updatePersonalDetails** – Updates the personal details of a User
* **myPersonalDetails** – Returns an User’s personal details