My approach involved in using DI and Method Injection as much as possible throughout the API to allow for a larger modularity of the system.

I looked at the requirements and created the Database Entities, and their relations, based on what I would need.

I picked SQLite for the DB as it allows for an easy and simple database solution, plus it is a RDBS which fits my design as my entities would have many relations.

I followed the controller-service pattern, where most of the logic is done in the service classes, leaving the controllers as free of logic as possible. That paired with DI patterns, allows for easier maintenance and/or switch of the services that do the logic.

A requirement was Authentication and Authorization of the users. For authentication, I decided to go with JWT tokens that are generated on Login with TTL and internally supported by Microsoft Identity. For Authorization I decided to with Rule Based Authentication as it fits the Authorization requirement, its simple for the brief time of the challenge and it is natively supported by Microsoft Identity.

Every interaction, Incoming and Outgoing, are “packaged” in Data Transfer Objects, to hide and control the information that the client receives. The entities and DTOs mapping were done via Extension methods, given more time I would’ve used the Mapper package that does this in a more automatic way, but I haven’t had enough experience with the package to commit.