A Short Write Up

I chose to use fastapi because fastapi is one of the easy-to-use python frameworks and has complete features, such as documentation, dependency injection, unit testing, etc. I created the main folder structure, namely 'app' and 'test' where the 'app' folder contains everything related to the application and 'test' is focused on data that will be used in unit testing using pytest.

I tried to implement clean code by prioritizing separation of concerns, where each folder has its own responsibilities, such as:

1. Router: a place to define all endpoints
2. Schema: defines the data structure that we will use such as response, request, etc.
3. Repositories: contains logic that is directly related to the database (CRUD) where the focus of this is data access logic

The reason why I try to implement separation is so that the code we have becomes structured, making it easier for us to work on large-scale applications, where an application can be worked on by many teams based on modules that have been divided, so that it can reduce conflicts when merging github and make it easier for us during maintenance, for example if there is a bug in a feature, we don't need to read the entire code in the application, but just read the code in the related module. In addition, our code is reusable, where if there is the same logic we can use it elsewhere without having to rewrite the exact same code.

I try to improve application performance by pagination and caching, with pagination we will not display all data, but we only display part of the data, by applying limits and offsets, so that it will not burden the server, besides that I apply in-memory caching using redis, caching techniques need to be done so that our database server is not burdened, so for example there is a request for 10 book data from id 1 to 10, then when the first request we will take it from the database, then the response will be stored in the redis cache for a certain period of time with a unique key, then if there is an exactly the same request we will check the cache first, if the data is in the cache, then that is what we take, if not then we will take it from the database, so that our server will not be burdened.

In addition, if the data already amounts to millions, it is very important to index the database, indexing can be likened to creating a table of contents page in a book, so that it will be faster to search, we need to index the columns that are often searched and columns that are often joined.

Another way that can be done is to create a load balancer on our server, so for an application we can place it on 10 servers simultaneously for example, so that the load is not only handled by one server, we can divide the load evenly to 10 applications using tools such as nginx.