STech eCommerce Web App

Software Architecture Document

Version 1.2

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 3/12/2023 | 1.0 | Initial version of Software Architecture Document | Đoàn Mạnh Tân |
| 18/12/2023 | 1.1 | Revised version of Software Architecture | Trần Ý Văn |
| 25/12/2023 | 1.2 | Add deployment diagram in section 5 and folder structure of source code in section 6 | Đoàn Mạnh Tân |
|  |  |  |  |

Table of Contents

[1. Introduction 4](#_heading=h.4d34og8)

[2. Architectural Goals and Constraints 4](#_heading=h.2s8eyo1)

[3. Use-Case Model 4](#_heading=h.17dp8vu)

[4. Logical View 5](#_heading=h.3rdcrjn)

[4.1 Package: Entities 5](#_heading=h.26in1rg)

[4.2 Package: DTO 6](#_heading=h.lnxbz9)

[4.3 Package: Repository Contracts 7](#_heading=h.35nkun2)

[4.4 Package: Service Contracts 8](#_heading=h.1ksv4uv)

[4.5 Package: Specification 9](#_heading=h.44sinio)

[4.6 Package: Helpers 10](#_heading=h.2jxsxqh)

[4.7 Package: Repositories 11](#_heading=h.z337ya)

[4.8 Package: Services 12](#_heading=h.3j2qqm3)

[4.9 Package: Specification Evaluator 14](#_heading=h.1y810tw)

[4.10 Package: Controllers 14](#_heading=h.4i7ojhp)

[4.11 Package: Dashboard Controllers 15](#_heading=h.2xcytpi)

[5. Deployment 16](#_heading=h.1ci93xb)

[5.1 Deployment Diagram 16](#_heading=h.b3hi4wldrab3)

[5.2 Diagram Details 16](#_heading=h.w165n9d0a35a)

[6. Implementation View 16](#_heading=h.3whwml4)

Software Architecture Document

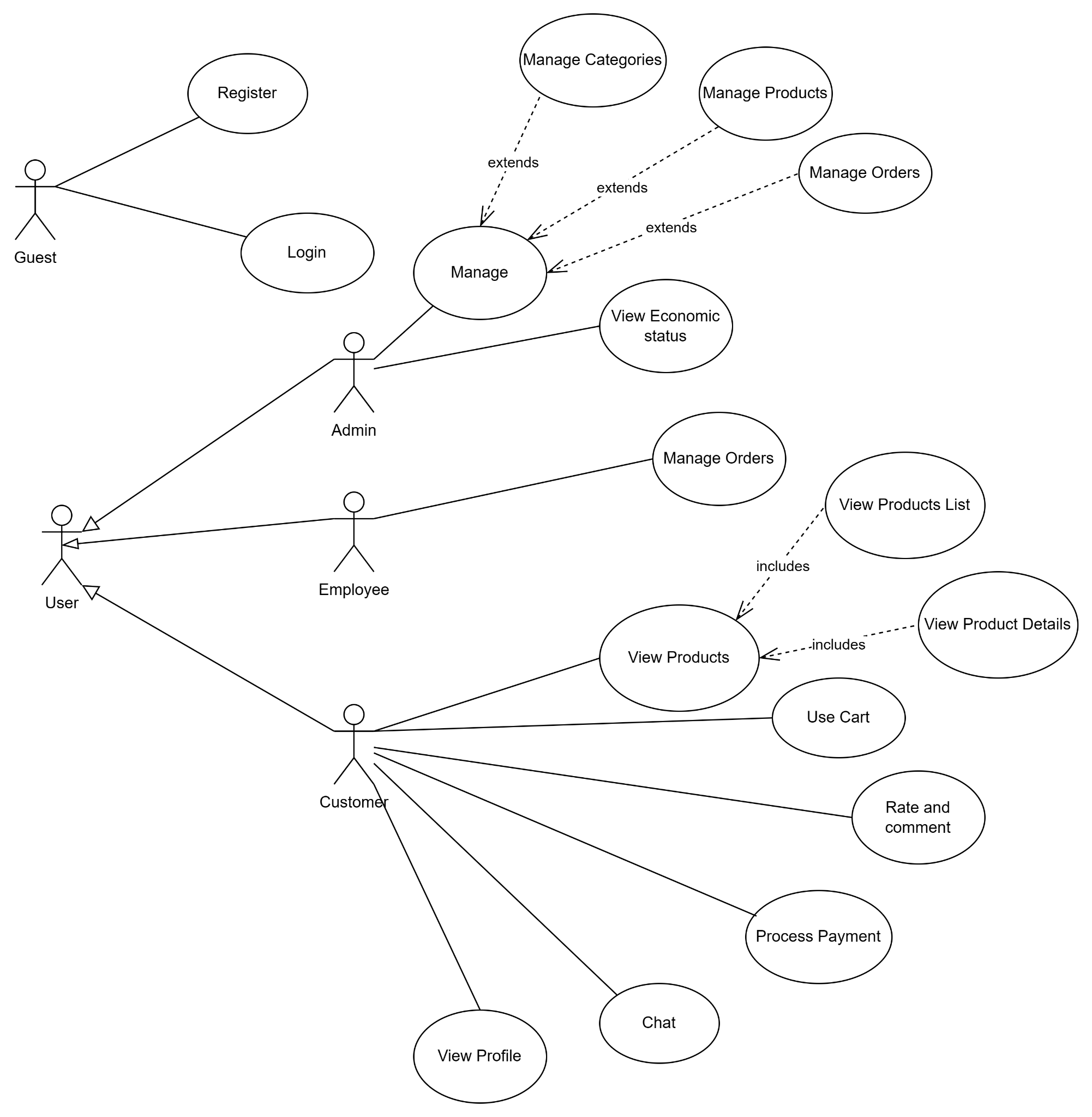
# Introduction

This document will provide details about the architecture of the application. There will be diagrams and further information about the architectures and packages.

# Architectural Goals and Constraints

# Use-Case Model

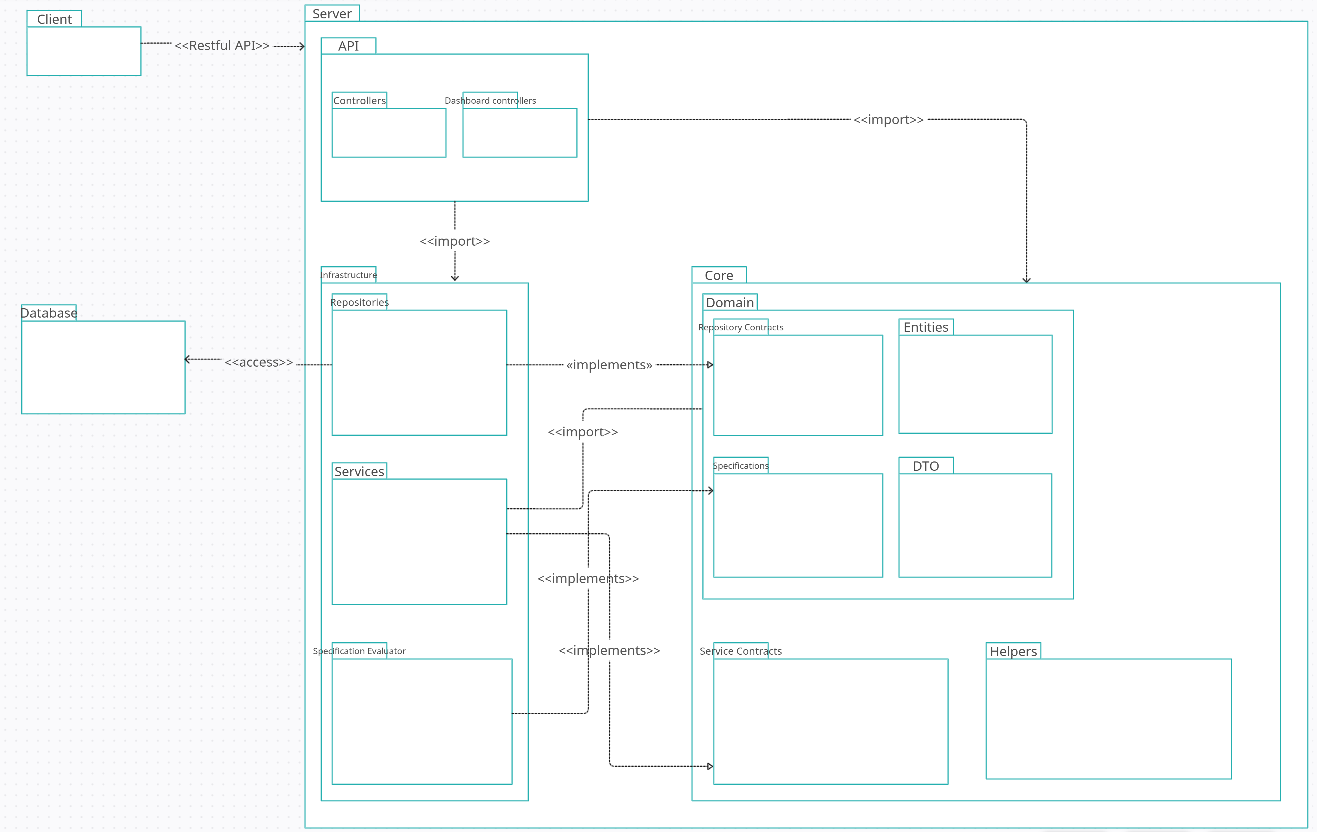
Use case diagram:



# Logical View

\_ STech eCommerce web app contains two main components which are SPA web app and Web API. This document mainly focuses on the Architecture of the Web API.

\_ Package Diagram of the application:



\_ The Web API is designed using Clean Architecture which divides the server into three main packages: Core, Infrastructure and API.

\_ The Core package contains child packages like Domain Models (Entities, DTO), Abstractions (Repo, Service contracts) and necessary helper classes.

\_ The Infrastructure package contains packages with concrete implementation of the abstractions in the Core package and other classes, so this package will reference the Core package. This package will be responsible for communicating with the database.

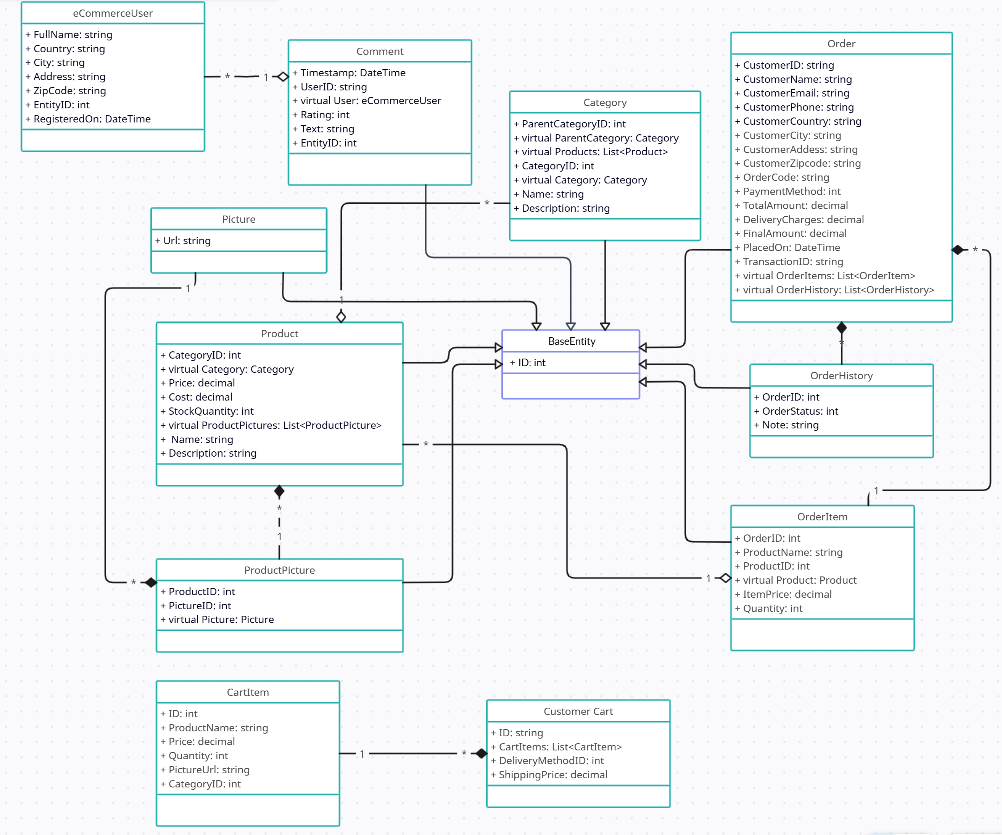
\_ The API package, which is responsible for direct communication with Client App, contains packages with API endpoints and necessary API configuration. This package will reference both the Core and Infrastructure package to use the utilities from these two.

\_ Below will provide details about the packages in the diagram.

## Package: Entities

\_ Entities package contains classes representing the objects used in the ecommerce scenario.

\_ Below is the class UML diagram of this package:



## Package: DTO

\_ DTO package contains classes which are used as Data Transfer Objects in the communication between Client and Server. In certain scenarios, server will use Data Transfer Objects to response data to Client as there are some properties hold the information that we do not want to expose to the Client.

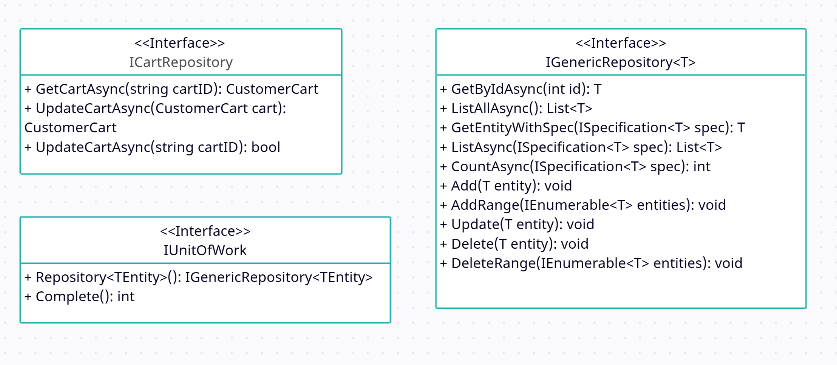
\_ Below is the class UML diagram of the package:



## Package: Repository Contracts

\_ Repository Contracts contains the interfaces which are the abstractions of the repositories.

\_ Below is the class UML diagram of the package:



\_The ICartRepository interface will communicate with Redis to process customer cart data.

\_ The IGenericRepository is a generic interface and the type T will be any of the classes that inherits from BaseEntity class in the Entities package. This repository can be used for most of the entities and has necessary data access methods to use in most of the scenarios. By using generic repository, as the amount of entities increases, such repetitive implementations of new repositories can be avoided.

\_ The IUnitOfWork is responsible for handling concurrency issues for database transactions as it uses only one dbContext instance (this instance is used by Entity Framework for data transactions) and use it for every generic repositories it contains. As a result, when transactions is finished, all of the actions conducted will be acknowledged by the context and make the transactions stable and consistent.

## Package: Service Contracts

\_ The Service Contracts contains the abstractions of the services which will be used by API endpoints.

\_ Below is the class UML diagram of the package:



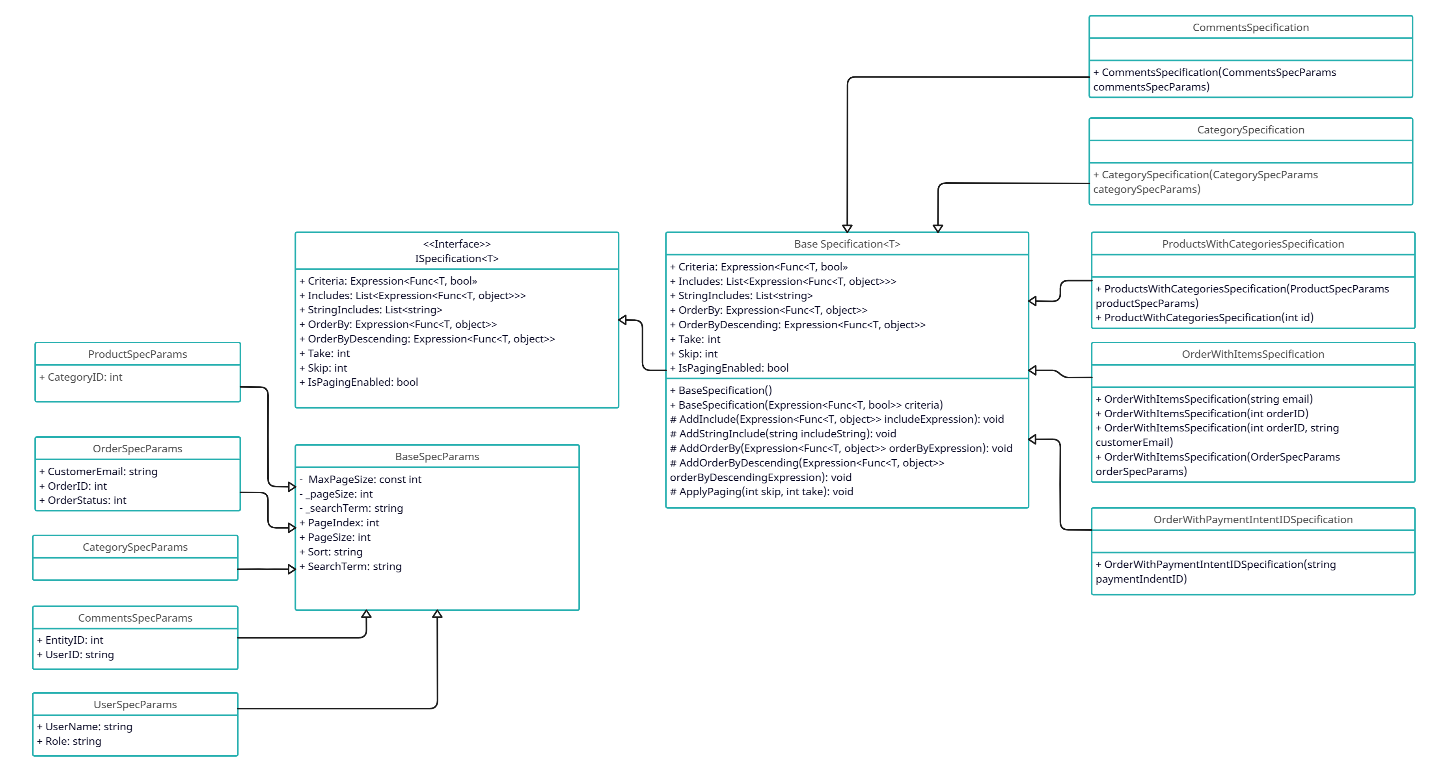
## Package: Specification

\_ The Specification package contains the utilities for specifications when using generic repositories for data accessing.

\_ One of the draw back of the generic repositories is they are quite limited to handle queries with specific requirements in certain scenarios.

\_ The Specification package provides classes with necessary properties to satisfy the query requirements such as Criteria (for filtering data), Includes (to include child entity), OrderBy (access data in on order base on property), … . With specification, the process of querying data can be more flexible in certain scenarios.

\_ Below is the class UML diagram of the package:

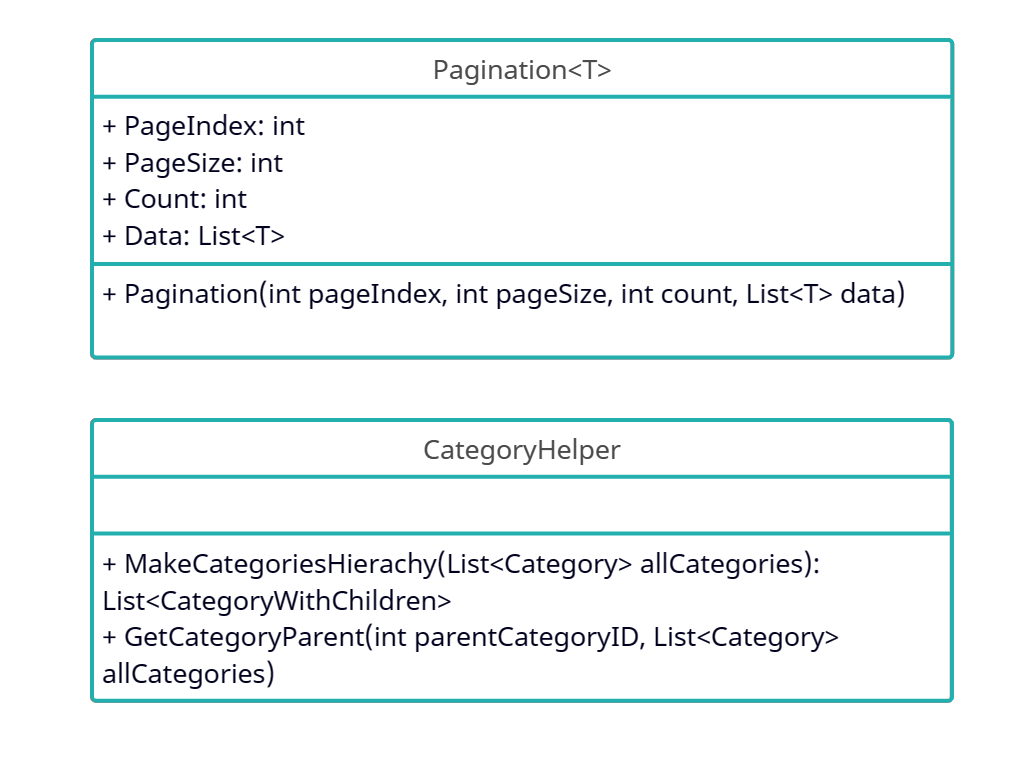


\_ The spec params classes will contain the properties for paging and criterias. The specification classes will receive the spec params as parameters and apply them as well as add necessary specs to form the complete specification.

## Package: Helpers

\_ The Helpers package contains helper classes.

\_ Below is the class UML diagram of the package:



\_ The Pagination class is a generic class which help to split data into pages to response to Client. This

makes paging at Client side way more easier.

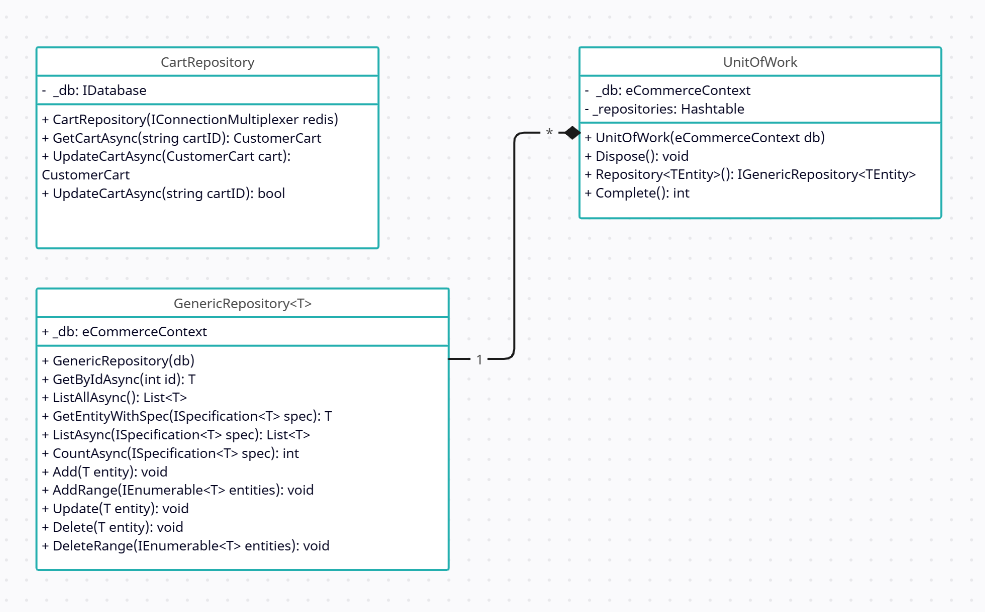
\_ The CategoryHelper class helps to organize the categories in a proper hierarchy as there are categories

that have parent category.

## Package: Repositories

\_ The Repositories package contains the concrete implementations of the repository interfaces in the Repository Contracts package.

\_ Below is the class UML diagram of the package:



\_ In the diagram, the CartRepository class holds the \_db which is of type IDatabase to access Redis. The UnitOfWork holds the instance of the eCommerceContext and provide that one for every repositories it has. The repositories in UnitOfWork will be organized as a Hashtable with the name of entity to be the key and the generic repository instance to be the value.

## Package: Services

\_ The Services package contains the concrete implementations of the service interfaces in the Service Contracts package.

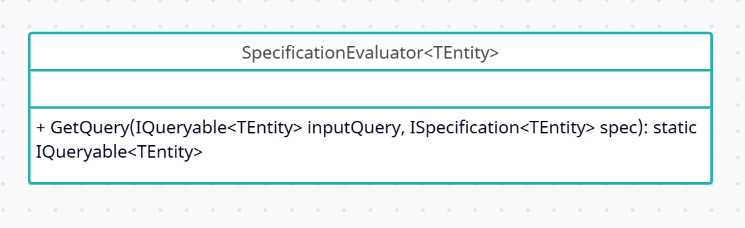
\_ Below is the class UML diagram of the package:



## Package: Specification Evaluator

\_ The Specification Evaluator contains the class that evaluate given query base on the provided specification. The GetQuery method will apply all of the specifications like Criterias, Includes, OrderBy, Paging, … and return the final query to access data.

\_ Below is the class UML of the package:

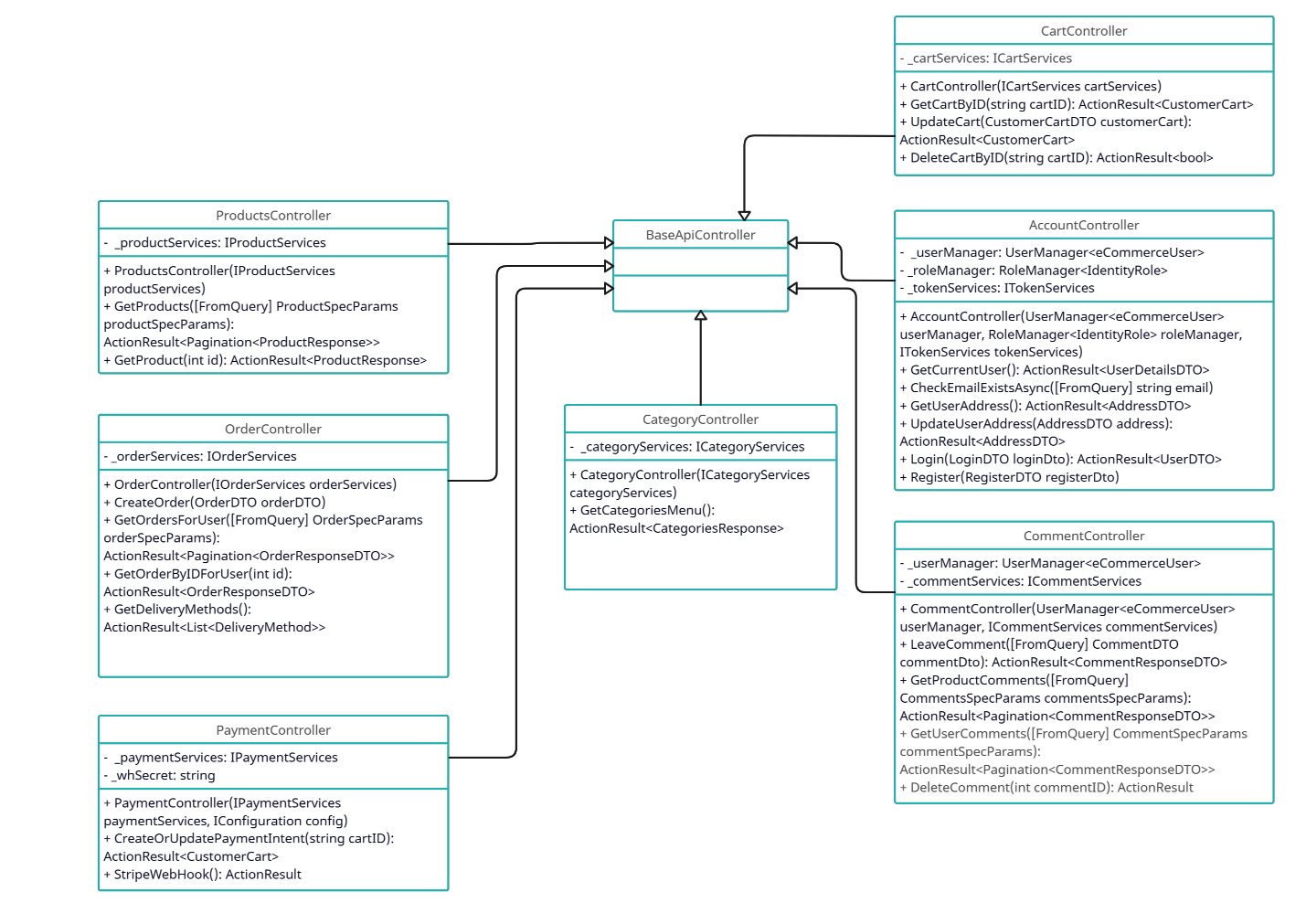


\_ This class will be used be the generic repository when querying data base on the provided specification.

## Package: Controllers

\_ The controllers package contains the API endpoints for regular requests from Client which mostly apply for customer operations.

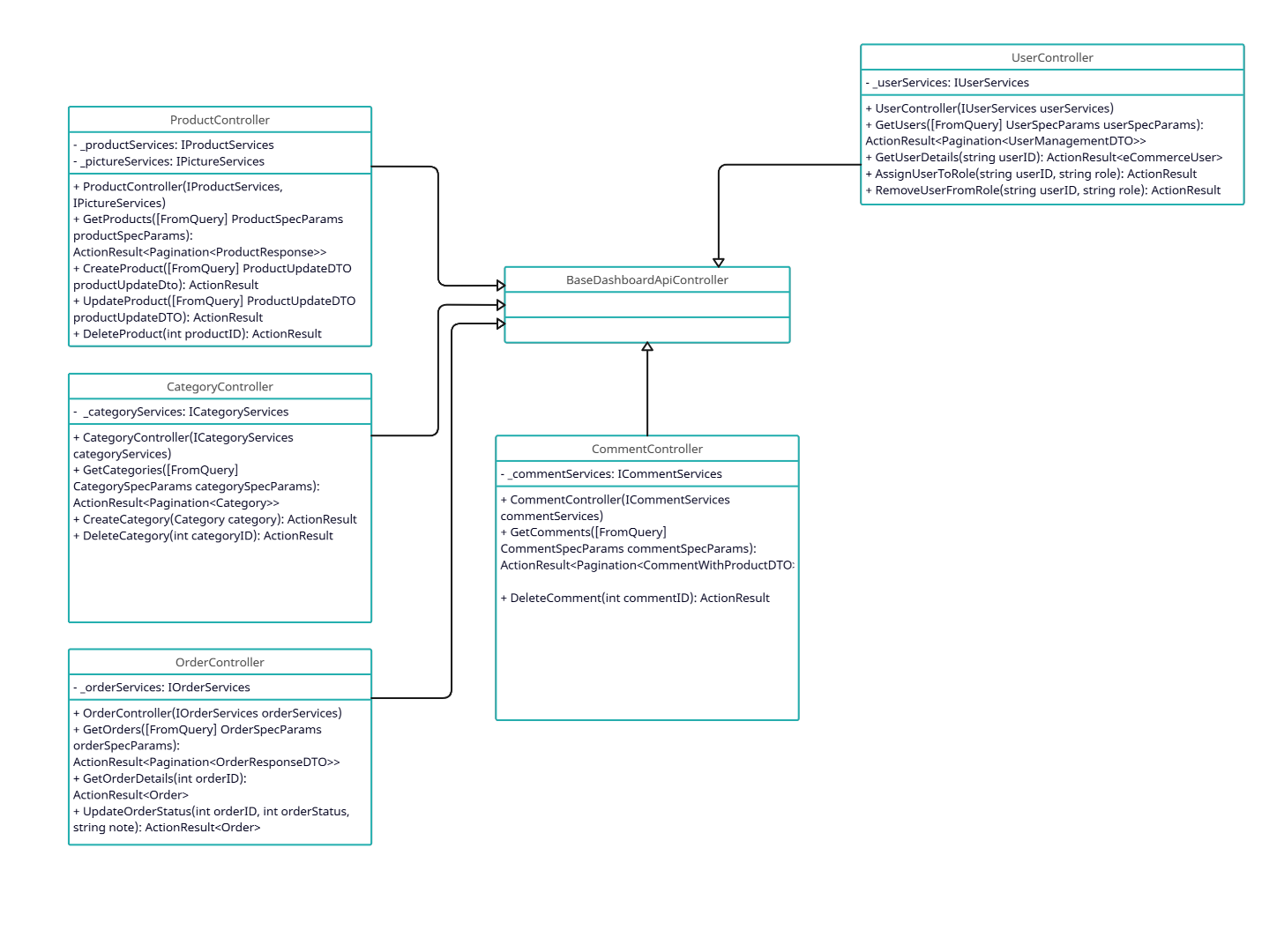
\_ Below is the class UML diagram of the package:



## Package: Dashboard Controllers

\_ The Dashboard Controllers contains API endpoints to handle specific requests from administrator or employee to perform management operations to run their business.

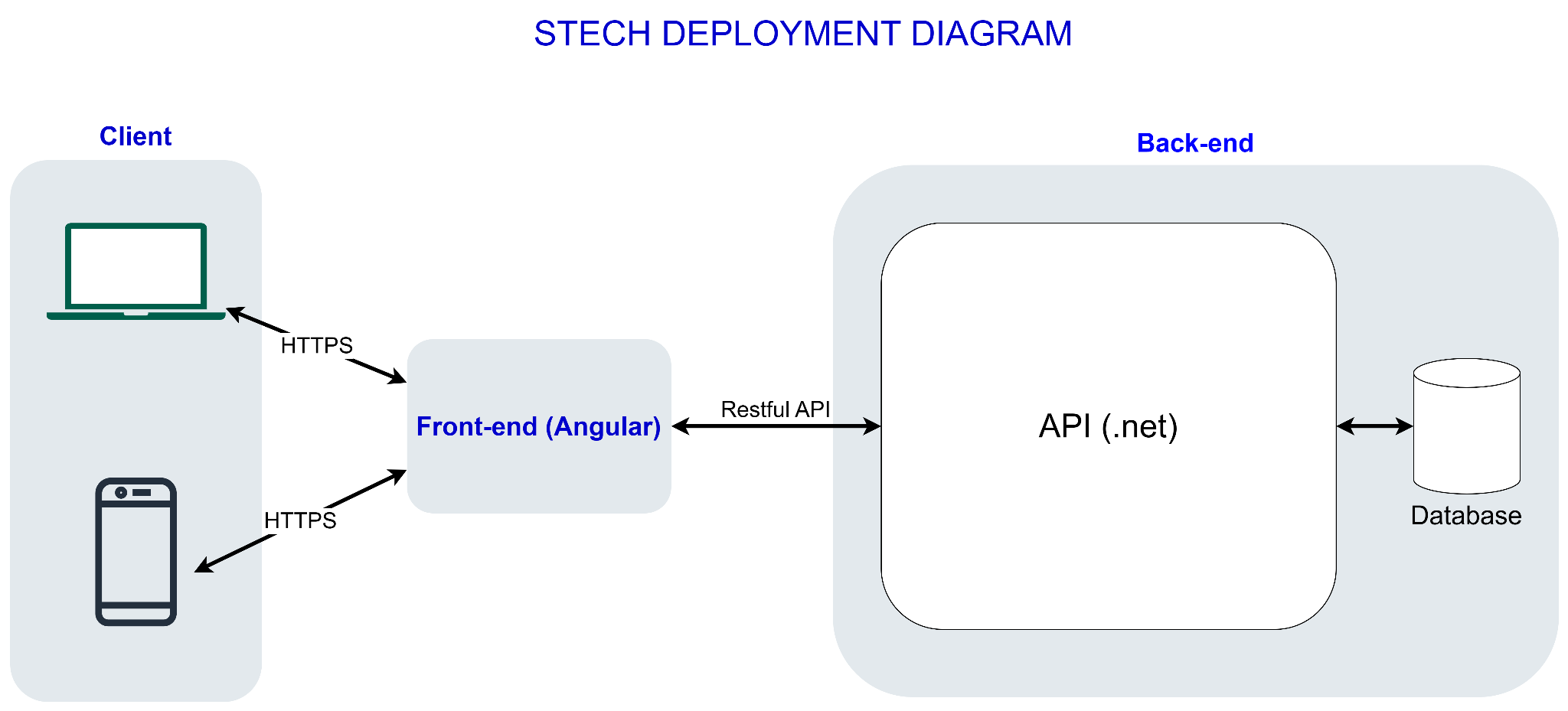
\_ Below is the class UML diagram of the package:



\_ These controllers will require authorization and verify the role of the user (admin and employee only) before processing further actions.

# Deployment

## Deployment Diagram



## Diagram Details

\_ Client: users can access the web app on PC or mobile phone from browser. Browser will make requests to front-end app through HTTPS protocol.

\_ Front-end: front-end app is built with Angular and is responsible for making requests to back-end server and render user interface with responses from server.

\_ Back-end: back-end server will contain web API (built with .net) and database (MSSQL or PostgreSQL). API will handle requests from front-end app, process come logic and then communicate with database (through Entity Framework Core) to retrieve data. API will then send data back to front-end app for it to render for users.

# Implementation View

\_ Folder structure(web api):

