# Layers

**Presentation layer:** This is the layer with which the user will communicate directly. It will display the information on the client’s browser and allow interaction with it. It will contain Controller classes based on the MVC pattern. Any new user interfaces developed for this application are put in this layer.

**Business Logic layer:** This layer manipulates the data and at the same time ensures their consistency and validity. There is no direct access to the data, but it rather goes through the Data Repository Layer in order to get data. Business Logic can access the data object without having knowledge of the underlying data access architecture. It is meant as a communication bridge between the Presentation and the Data Repository layers.

**Data Repository layer:** The layer provides simplified access to the data stored in the relational database. The business logic methods can be mapped to the Data Repository layer, so instead of making a query into a database to fetch data, the application can call a method from DAL which abstracts those database calls.

# Patterns

**Separation of Concerns:** One of the core principles for developing clean architecture, this pattern states that each distinct type of work that the application is performing should be built separately as a separate component with little to no tight coupling with other components. In our case, we have the Presentation, Business Logic and Data Repository layers as different sections that encapsulate information that can be developed and updated independently.

**Dependency Injection:** Is a design pattern that allows us to develop loosely coupled software components. In other words, we can say that this design pattern is used to reduce the tight coupling between software components. As a result, we can easily manage future changes and other complexity in our application. This provides greater maintainability, testability and re-usability. We will use this when instancing objects from different projects in order to use the same object throughout the solution.

**Repository pattern:** This pattern acts as a middle layer between the rest of the application and the data access logic. That means the repository pattern isolates all the data access code from the rest of the application. The advantage of doing so is that, if you need to do any change then you can do it in one place. We have the interface for the repository inside the contract class library and the repository is implemented inside the data layer.

**Request/Response:** this is the most basic and common of all the client-service interaction patterns. It is used when the client must have an immediate response or wants the service to complete a task without delay. Request/Response begins when the client establishes a connection to the service. Once a connection has been established, the client sends its request and waits for a response. The service processes the request as soon as it is received and returns a response over the same connection. We will implement this for connection the front and back end of the application and also for paging the responses in order to have faster load of searched results.