Chapter 3 sample question and answer

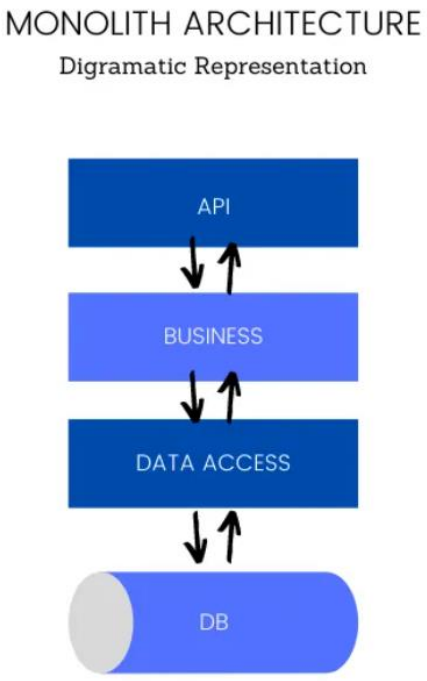
1. What are the common web architecture available in web application? Explain any 3 common web architecture.
   - **monolithic application**
   - **Layered Architecture**
   - **Clean architecture**

Monolithic Application:

These are the applications developed as single component, a single deployable unit. In other words we can say an application which can be treated as a single executable unit.

Some e.g. of monolithic applications are MS word, MS excel, Notepad etc. These are the single tier applications

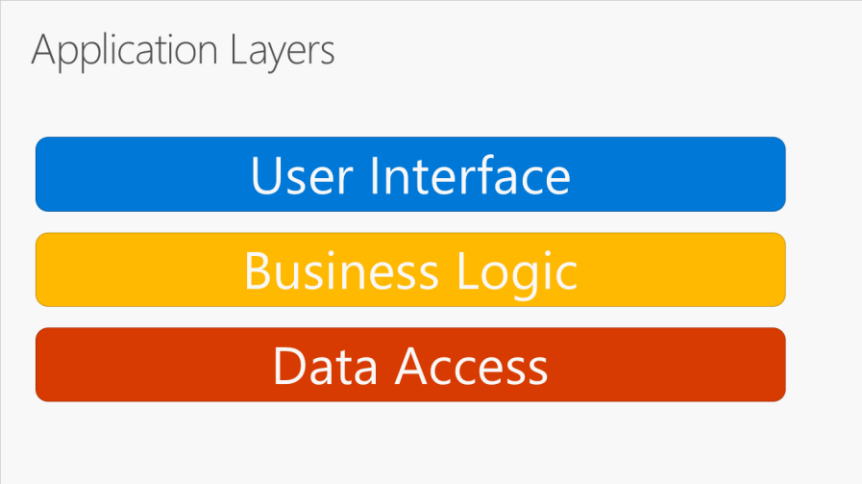
You can call it 3-Layer Architecture or N-Layer Architecture, it's all the same. Ultimately this is a Monolith. It's a single application code block without dependencies on other applications (exes).



Benefits Of Modular Architecture In ASP.NET Core

1. Clear Separation of Concerns
2. Easily Scalable
3. Lower complexity compared to Microservices
4. Low operational / deployment costs.
5. Reusability
6. Organized Dependencies

Layered Architecture



These layers are frequently abbreviated as UI, BLL (Business Logic Layer), and DAL (Data Access Layer). Using this architecture, users make requests through the UI layer, which interacts only with the BLL. The BLL, in turn, can call the DAL for data access requests. The UI layer shouldn't make any requests to the DAL directly, nor should it interact with persistence directly through other means. Likewise, the BLL should only interact with persistence by going through the DAL. In this way, each layer has its own well-known responsibility.

One disadvantage of this traditional layering approach is that compile-time dependencies run from the top to the bottom. That is, the UI layer depends on the BLL, which depends on the DAL. This means that the BLL, which usually holds the most important logic in the application, is dependent on data access implementation details (and often on the existence of a database). Testing business logic in such an architecture is often difficult, requiring a test database.

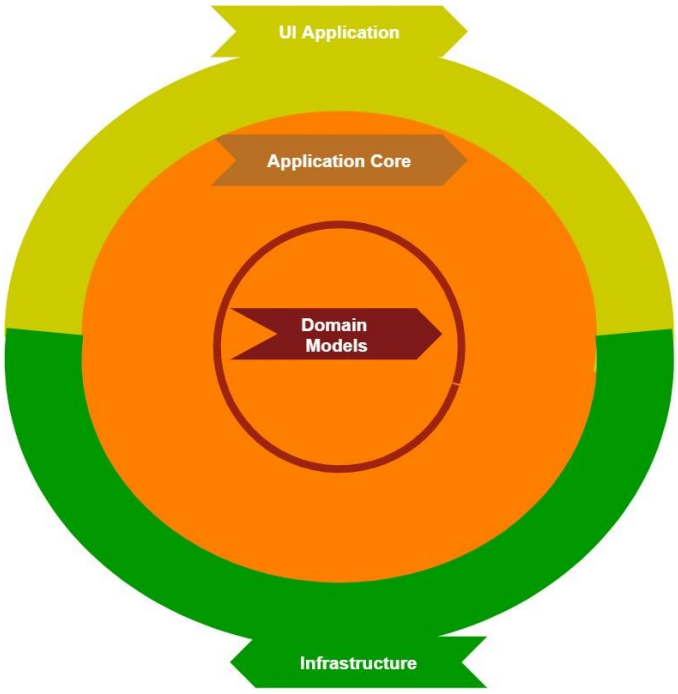
Clean architecture:

Clean Architecture core building blocks are:

- Application Core
- Infrastructure
- UI Application

Clean Architecture lives on the dependency inversion principle. In general business, logic depends on the data access layer or infrastructure layer. But in clean architecture, it is inverted, which means data

access layers or infrastructure layers depend on the business logic layer(which means Application Core). So with the dependency inversion technique it easy to configure 'Unit Test' or 'Integrating Test'.



Application Core:

- Application Core is a top layer or parent layer which will not depend on any other layer. So other layers like Infrastructure or UI depend on the 'Application' core.
- Application Core contains 'Entites', 'DTOs', 'Interfaces', 'BusinessLogics', etc.

Infrastructure:

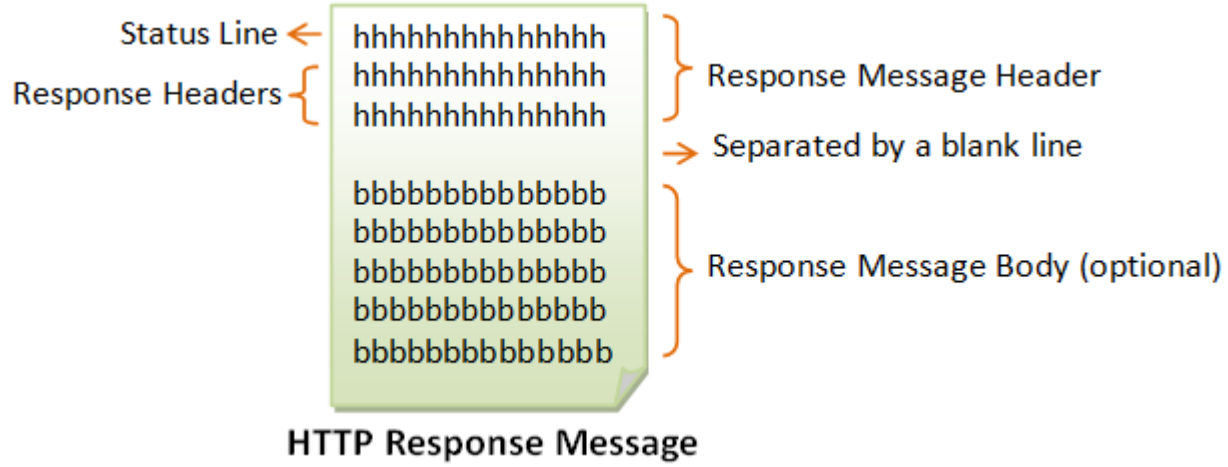
Infrastructure deals with 'DataBases', 'External API Calls', 'Cache', etc. Basically, infrastructure deals with all external resources. Infrastructure depends on the 'Interface' inside of the 'Application Core'. Because of the dependency inversion, our 'Application Core' will be loosely coupled which is easy to test.

UI Application:

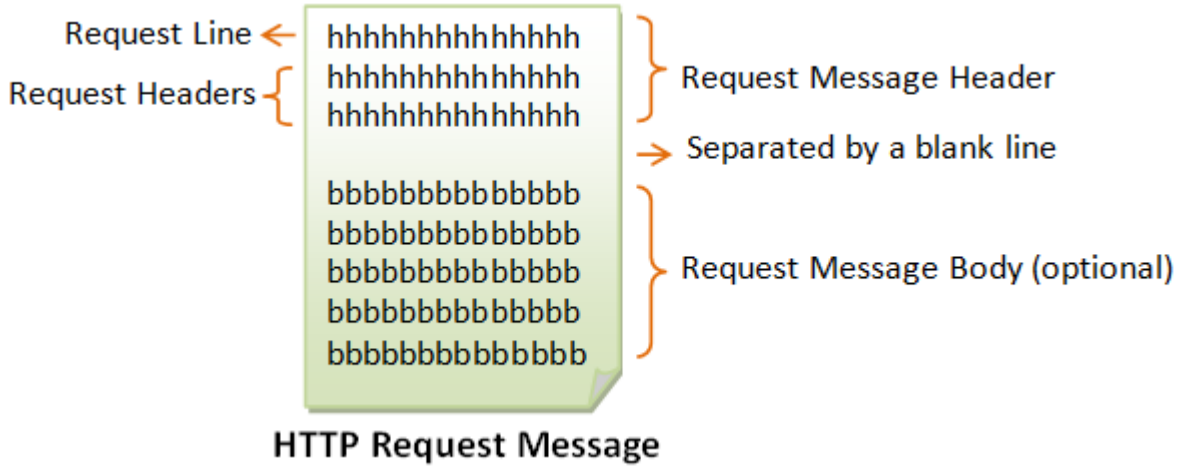
UI Application consumes the 'Application Core' to produce the results. In a real-time scenario UI Application never depends on the infrastructure layer, but we have to reference the infrastructure layer into the UI project in the case to register the services dependency injection.

2. **Explain HTTP Request and Response Message Format.**

HTTP Response Message:

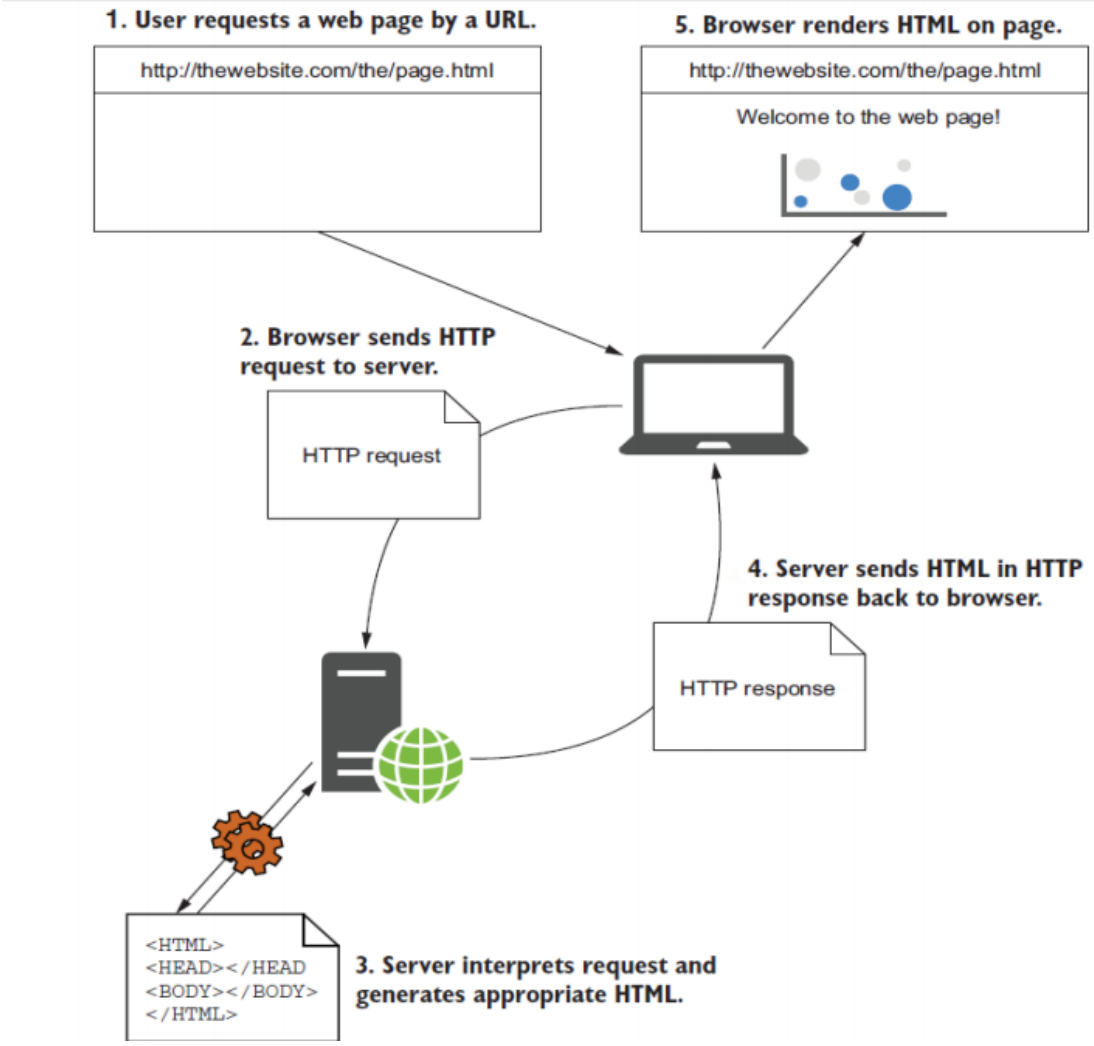


HTTP Request Message:



The first line of the header is called the *request line*, followed by optional *request headers*.

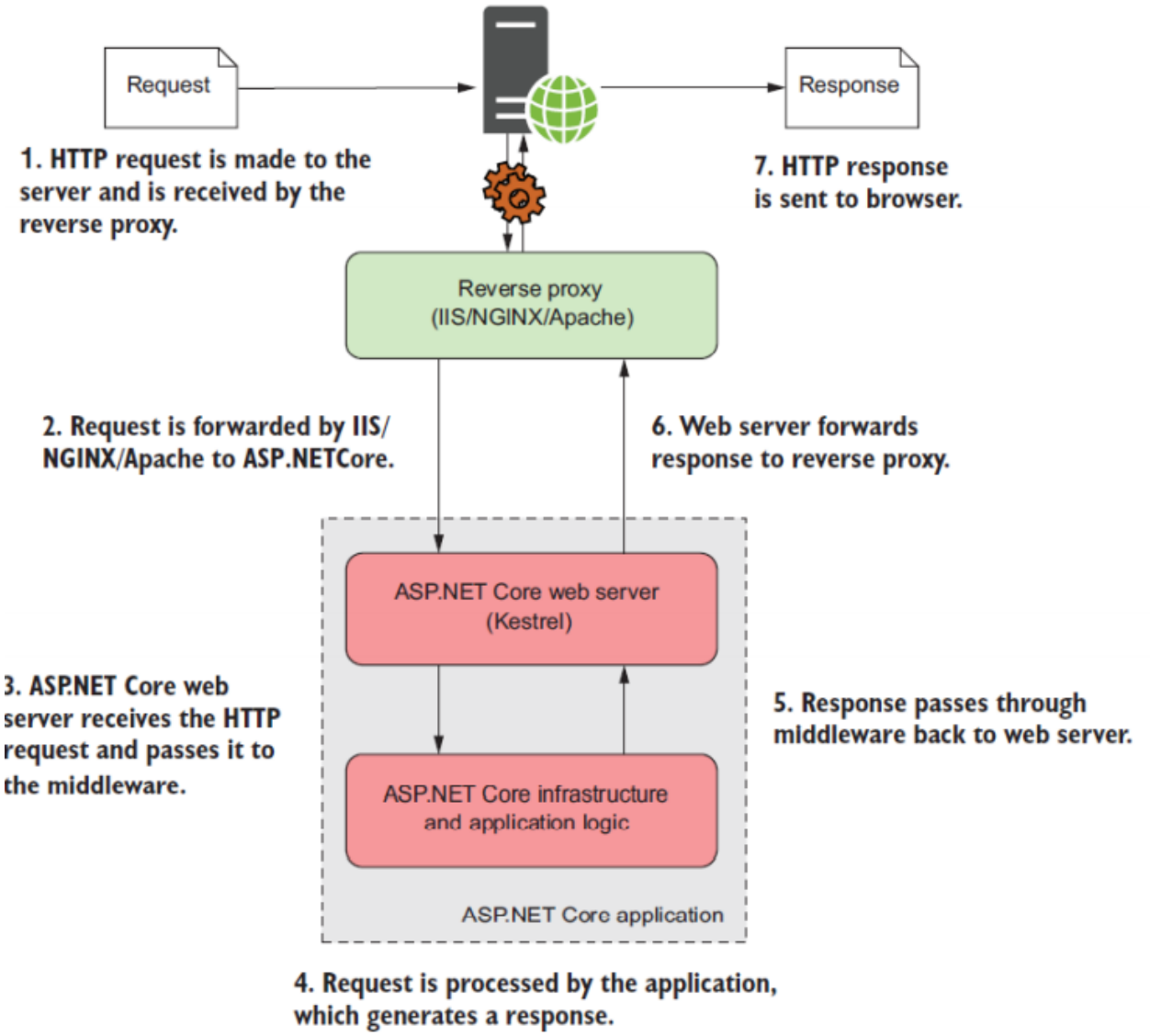
3. How does HTTP Response and HTTP Request Works?



the user starts by requesting a web page, which causes an HTTP request to be sent to the server. The server interprets the request, generates the necessary HTML, and sends it back in an HTTP response. The browser can then display the web page.

 Once the server receives the request, it will check that it makes sense, and if it does, will generate an HTTP response. Depending on the request, this response could be a web page, an image, a JavaScript file, or a simple acknowledgment.

 As soon as the user's browser begins receiving the HTTP response, it can start displaying content on the screen, but the HTML page may also reference other pages and links on the server.



A request is received from a browser at the reverse proxy, which passes the request to the ASP.NET Core application, which runs a self-hosted web server.

 The web server processes the request and passes it to the body of the application, which generates a response and returns it to the web server. The web server relays this to the reverse proxy, which sends the response to the browser.

 benefit of a reverse proxy is that it can be hardened against potential threats from the public internet. They're often responsible for additional aspects, such as restarting a process that has crashed. Kestrel can stay as a simple HTTP server. Think of it as a simple separation of concerns: Kestrel is concerned with generating HTTP responses; a reverse proxy is concerned with handling the connection to the internet.