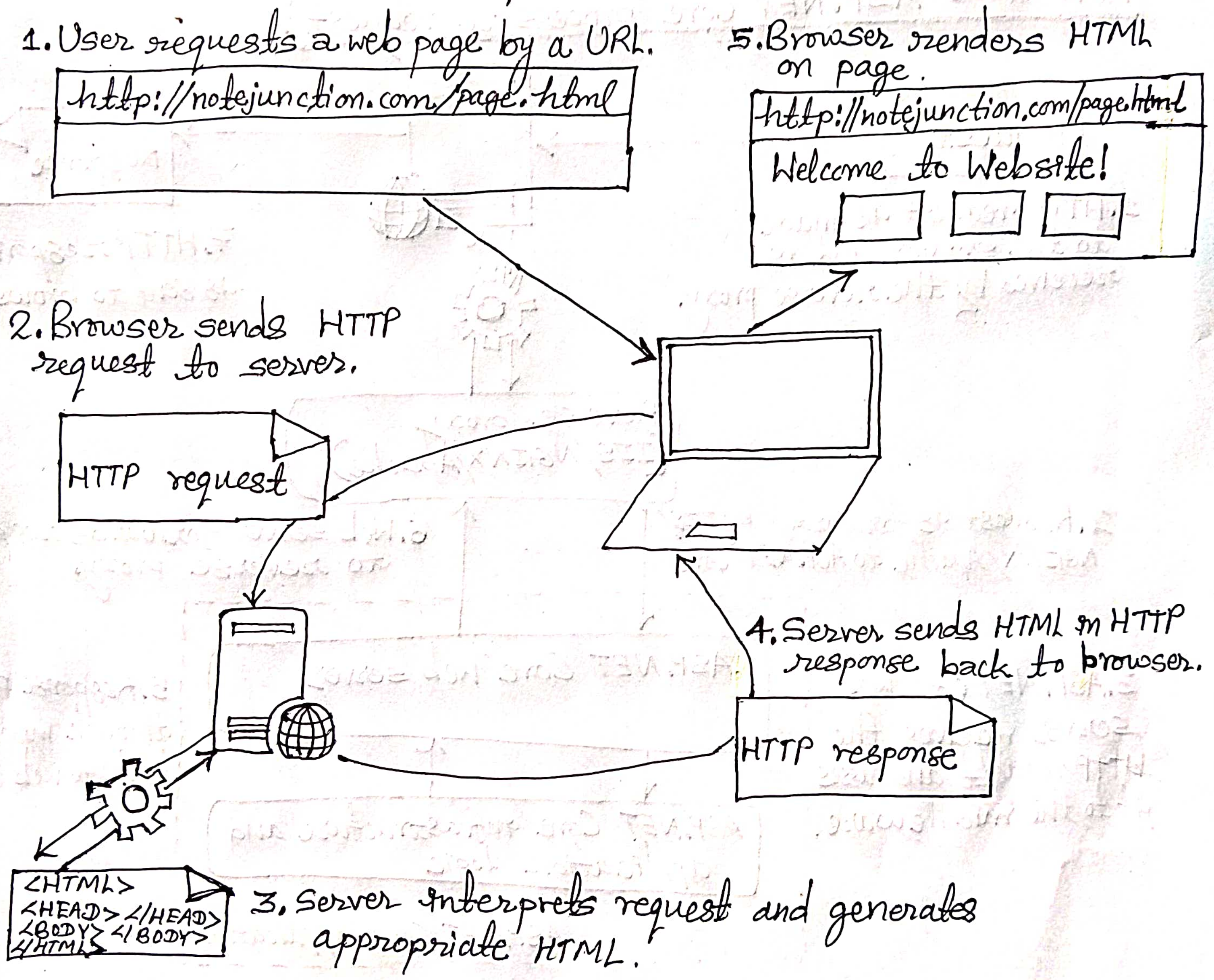
UNIT-3
HTTP and ASP.NET Core

⊛. HTTP: Request and Response Message Format:

To communicate with a web server, the client makes calls over the network using HTTP. A client makes an HTTP request for a resource, and the server sends back an HTTP response.

How does an HTTP web request work?



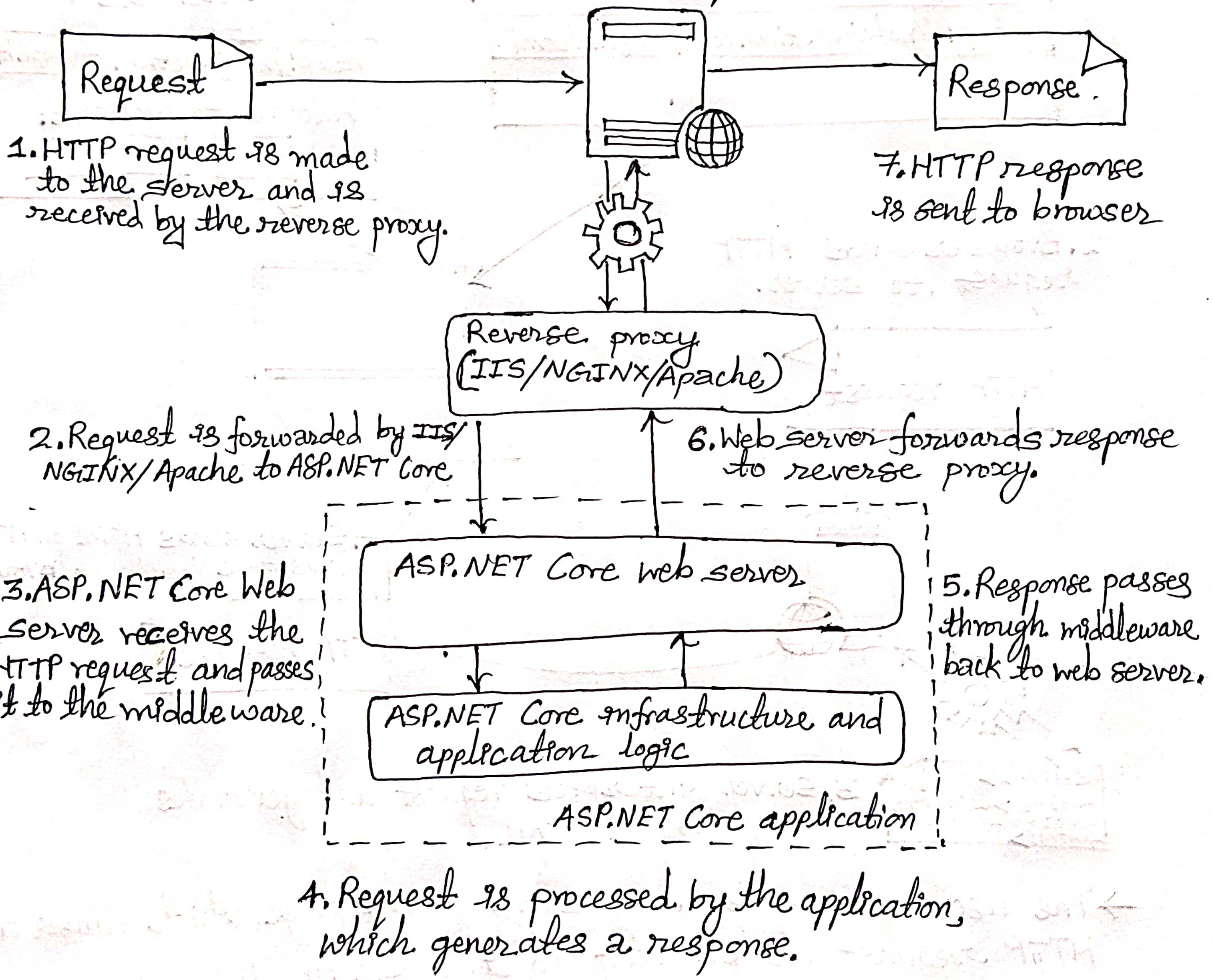
1. User requests a web page by a URL.
http://notejunction.com/page.html
5. Browser renders HTML on page.
http://notejunction.com/page.html
Welcome to Website!
2. Browser sends HTTP request to server.
HTTP request
4. Server sends HTML in HTTP response back to browser.
HTTP response
<HTML>
<HEAD> </HEAD>
<BODY> </BODY>
</HTML>
3. Server interprets request and generates appropriate HTML.

→ 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 refrence other pages and links on the server.

How does ASP.NET Core process a request?



Request
Response
1. HTTP request is made to the server and is received by the reverse proxy.
7. HTTP response is sent to browser
Reverse proxy (IIS/NGINX/Apache)
2. Request is forwarded by IIS/NGINX/Apache to ASP.NET Core
6. Web server forwards response to reverse proxy.
ASP.NET Core web server
3. ASP.NET Core Web server receives the HTTP request and passes it to the middleware.
5. Response passes through middleware back to web server.
ASP.NET Core infrastructure and application logic
ASP.NET Core application
4. Request is processed by the application, which generates a response.

→ 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.

⊛ Common web application architectures:

1) **Monolithic Application:** A monolithic application is one that is entirely self-contained, in terms of it's behaviour. It may interact with other services or data stores in the course of performing its operations, but the core of its behaviour runs within its own process and the entire application is typically deployed as a single unit.

2) **All-in-one application:** The smallest possible number of projects for an application architecture is one. In this architecture, the entire logic of the application is contained in a single project, compiled to a single assembly, and deployed as a single unit.

3) **Layered architecture:** As applications grow in complexity, one way to manage that complexity is to break up the application into different layers. By organizing code into layers, common low-level functionality can be reused throughout the application. With layered architecture, applications can enforce restrictions on which layers can communicate with other layers.

4) **Traditional "N-layer" architecture applications:**
Using this architecture users make requests through the User Interface (UI) layer, which interacts only with the Business Logic Layer (BLL). The BLL in turn can call the Data Access Layer (DAL) for data access requests. 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.

5> **Clean Architecture:** Applications that follow the Dependency Inversion Principle as well as the Domain-Driven Design (DDD) principles are known as clean architecture. Clean architecture puts the business logic and application model at the center of the application.

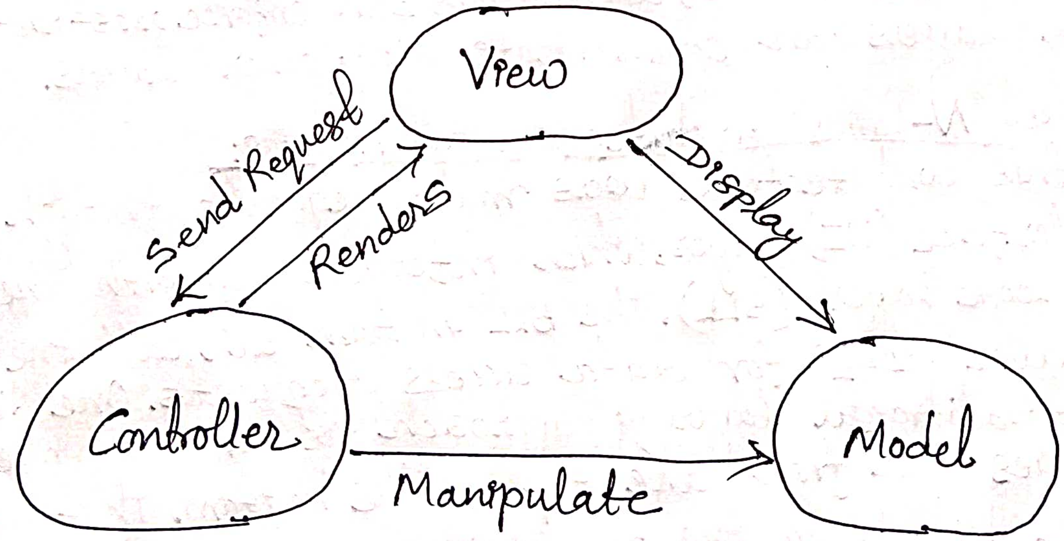
⊛. MVC Pattern/Architecture:

MVC stands for Model, View, and Controller. MVC seperates an application into three components — Model, View, and Controller.

Model: It represents the shape of the data. A class in C# is used to describe a model. Model objects store data retrived from the database. Model represents the data.

View: A view in MVC is a user interface. View display model data to the user and also enables them to modify them. View in ASP.NET MVC is HTML, CSS, and some special syntax (Razor syntax) that makes it easy to communicate with the model and the controller.

Controller: It handles the user request. Typically the user uses the view and raises an HTTP request. Controller processes request and returns the appropriate view as a response. Controller is the request handler.



⊗ ASP.NET Core Architecture Overview :

The ideology behind ASP.Net Core in general is to lay out web logic, infrastructure, and core components from each other in order to provide a more development-friendly environment. In ASP.NET Core the main business logic and UI logic are encapsulated in ASP.NET Core Web App Layer, while the database access layer, cache services, and web API services are encapsulated in infrastructure layer and common utilities objects, interfaces and reusable business services are encapsulated as micro-services in application core layer.

ASP.NET Core creates necessary pre-defined "N" ~~tier~~ architecture for developers automatically which saves time and effort. It has benefit of a pre-built architectural framework that eases out tier deployment of the project along with providing pre-build Single Page Application (SPA) design pattern, Razor Pages design and traditional MVC design pattern.

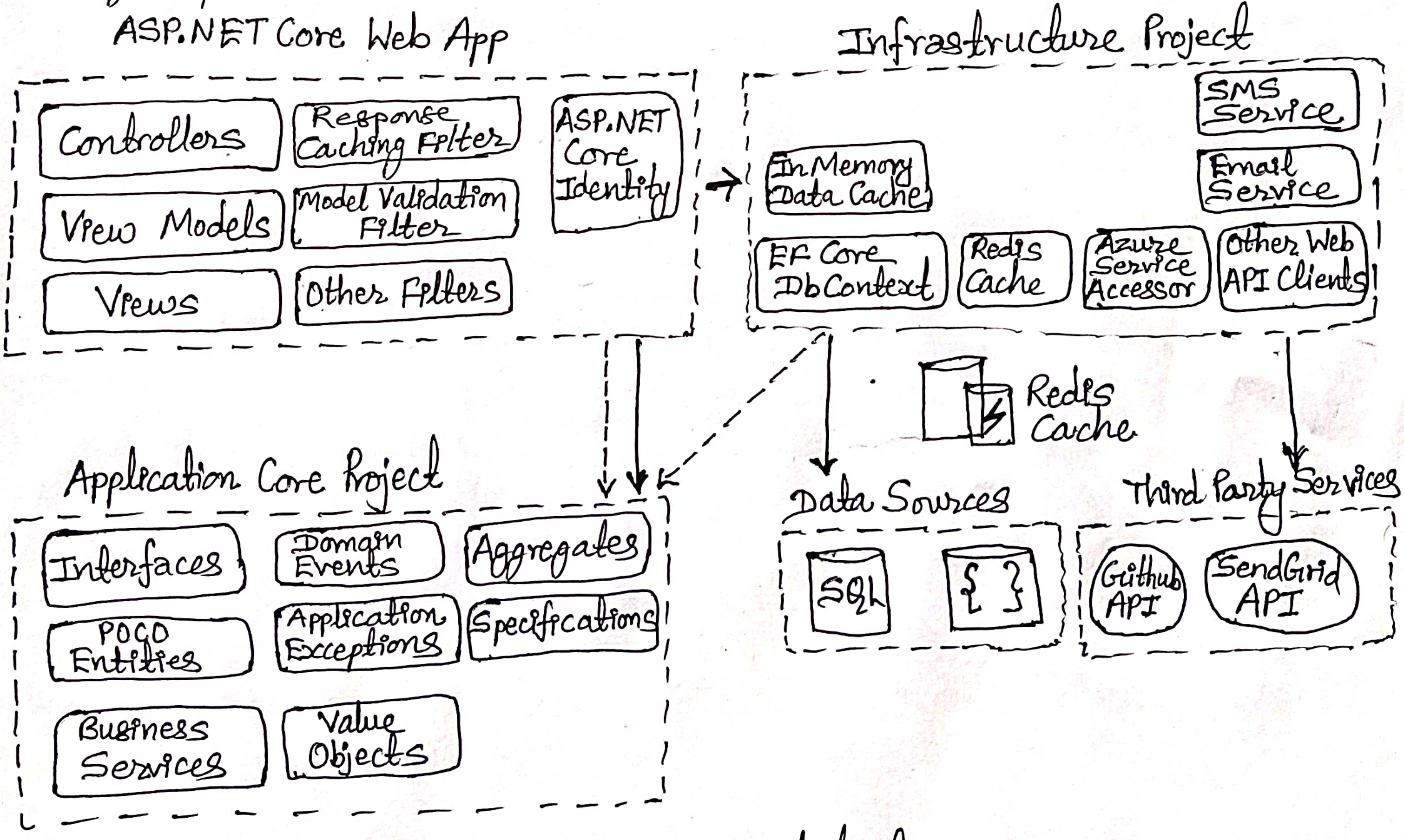


Fig: ASP.NET Core Architecture Overview.

⊛. Projects and Conventions:

less important can be escaped. It is practical work rather than theory

Visual Studio now uses .csproj file to manage projects. We can edit the .csproj settings by:

→ right click on the project.

→ select Edit <project-name>.csproj

The csproj file includes settings related to targeted .NET frameworks, project folders, NutGet package refrences etc.