**.NET Core Pathway**

**HTML, CSS, and JavaScript Basics**

* **HTML:** Elements, attributes, forms, semantic HTML, tables
* **CSS:** Styling, selectors, layout techniques (Flexbox, Grid), animations, media queries, responsive design
* **JavaScript:** Variables, data types, operators, functions, DOM manipulation, events, AJAX basics, ES6 features

**2. .NET Core and C# Fundamentals**

* **Introduction to .NET Core** and **C#** syntax
* **C# Fundamentals:** Data types, control structures (if, switch, loops), error handling (try-catch), debugging
* **OOP Principles in C#:** Classes, objects, inheritance, polymorphism, abstraction, encapsulation
* **Data Structures:** Collections (List, Dictionary, Array), LINQ basics
* **Asynchronous Programming:** async and await, Task, Task-based Asynchronous Pattern (TAP)

**3. Entity Framework Core**

* **Overview of Entity Framework:** Code-first and database-first approaches
* **Model Creation:** Defining entities, configuring relationships, migrations
* **Querying Data:** LINQ queries, async operations, CRUD operations
* **Database Connections:** Working with MS SQL Server, connection strings
* **Advanced Topics:** Transactions, eager/lazy loading, navigation properties, change tracking

**4. ASP.NET Core MVC**

* **Introduction to MVC Architecture:** Model, View, Controller pattern
* **Controllers and Actions:** Routing, URL routing conventions, Action results
* **Views:** Razor syntax, layout views, partial views, view components
* **Models and Data Binding:** Data annotations, form validation, model binding
* **Dependency Injection:** Service registration, middleware pipeline
* **Session and State Management:** Working with sessions, cookies
* **Security Basics:** Authentication and authorization (JWT, cookies-based authentication)

**5. ASP.NET Core Web API**

* **RESTful API Design:** Principles, HTTP methods (GET, POST, PUT, DELETE)
* **Creating Web API Controllers:** Routing, attribute routing, custom responses
* **Data Transfer Objects (DTOs):** Mapping entities to DTOs for API
* **Dependency Injection in Web API:** Services and repositories pattern
* **Error Handling in APIs:** Exception filters, global exception handling
* **Securing Web API:** JWT, OAuth basics
* **Versioning and Documentation:** API versioning strategies, Swagger/OpenAPI integration

**6. MS SQL Server**

* **Introduction to SQL and Relational Databases:** Data modeling, ER diagrams
* **Writing SQL Queries:** SELECT, INSERT, UPDATE, DELETE
* **Joins and Subqueries:** INNER JOIN, LEFT JOIN, RIGHT JOIN, CROSS JOIN, subqueries
* **Stored Procedures and Views:** Creating and using stored procedures and views
* **Indexes and Optimization:** Basics of indexing, query optimization techniques
* **Database Administration Basics:** Backup, restore, and data integrity

**7. Unit Testing in .NET Core**

* **Introduction to Unit Testing:** Importance and purpose
* **Testing Frameworks:** Using xUnit or NUnit for writing tests
* **Moq Framework:** Mocking dependencies, setting up test data
* **Writing Tests:** Unit tests for methods, controllers, and business logic
* **Code Coverage:** Analyzing code coverage, understanding test reports
* **Best Practices:** Test-driven development (TDD), clean and maintainable tests, avoiding anti-patterns

This syllabus provides a solid foundation for .NET Core development, covering both frontend and backend essentials, with a focus on real-world skills for software development in the .NET ecosystem. Let me know if you’d like additional detail on any specific section!

Here's a breakdown of which components fall under **frontend, backend, and server-side** in the given syllabus:

**Frontend (Client-Side)**

The **frontend** is the part of the application users interact with directly. It includes:

1. **HTML:** The structure of web pages.
2. **CSS:** The styling and layout of web pages.
3. **JavaScript:** Adds interactivity and dynamic behavior to the UI.

**Backend (Server-Side Logic)**

The **backend** handles the business logic, database operations, and APIs. It processes requests from the frontend and sends back responses. Backend components include:

1. **C# Fundamentals:** Programming logic, OOP principles, and asynchronous tasks.
2. **Entity Framework Core:** Database interaction layer for CRUD operations.
3. **ASP.NET Core MVC:** For developing dynamic web applications (models, controllers, server-side views).
4. **ASP.NET Core Web API:** For building RESTful APIs to handle client requests (data exchange).

**Server (Infrastructure and Database)**

The **server** is where the application runs and stores data. It includes:

1. **MS SQL Server:** A relational database system for storing and querying data.
2. **ASP.NET Core Runtime:** Hosting the backend application logic.
3. **Web Server (e.g., IIS, Kestrel):** The environment where the application is deployed and serves client requests.

**How They Work Together:**

1. **Frontend:** HTML/CSS/JS is sent to the user's browser.
2. **Backend:** When a user interacts with the frontend (e.g., submitting a form), requests are sent to the backend (ASP.NET Core MVC or Web API).
3. **Server:** The backend accesses data from the MS SQL Server database and processes the request on the server.
4. **Response:** The backend sends the processed data (HTML, JSON, etc.) back to the frontend for display to the user.

This structure ensures smooth communication between the user, application logic, and database.

**Client: Roche** *(Switzerland-based pharmaceutical company)*

* **Duration:** Oct 2023 - Ongoing
* **Details:** Roche Reactive Web | **Team** **Size:** 7 | **Tools:** OutSystems Service Studio (11), Service Center, Visual Studio, Postman, Jira
* **Skills**: OutSystems Reactive Development, JavaScript, SQL Server, C#

**Role and Responsibilities:**

* Extended the features and functionality of the pre-existing Roche business application to meet the requirements of growing business needs.
* Collaborated as part of a team to develop new screens, focusing on various forms and UI elements to enhance application functionality and user experience.
* Integrated REST APIs within OutSystems' three-layer architecture, ensuring seamless data interaction and system scalability.
* Managed server-side and client-side aggregates, implementing advanced filtering, sorting, and data action techniques to optimize data retrieval processes and enhance overall application performance.
* Debugged and resolved issues within the web application, improving stability and user satisfaction.
* Focused on optimizing performance and expanding features to support Roche's healthcare operations.

**Client: Eneco***(Netherlands-based energy company)*

* **Duration:** 10 months
* **Details:** Module-Eneco MVS Run Phase | **Team** **Size:** 4 | **Tools:** OutSystems Service Studio (11), Integration Studio, Oracle SQL Developer 19c, Postman
* **Skills:** OutSystems Traditional Web Development, Oracle PLSQL

**Role and Responsibilities:**

* Designed and implemented user interfaces across multiple screens, ensuring consistency and usability.
* Managed data integration through extension modules, effectively mapping entities to facilitate seamless data flow.
* Developed and executed PLSQL code for backend processes and data exports within OutSystems applications.
* Delivered reliable and efficient web-based solutions tailored to the energy sector.

HTML/CSS/JS github link: https://github.com/jonasschmedtmann/html-css-course/tree/master

https://capgemini.udemy.com/course/design-and-develop-a-killer-website-with-html5-and-css3

HTML:Different tag elements head,title,body,img(src), anchor, hyperlink-href, paragraph, orderlist, unorder list

Inline CSS,