EULA = end user license agreement

DLL = Dynamic Link Library – they are created when you build the project

What I was missing to do :

* Username should not be duplicated in db
* Nu am prevazut cazul de redirect pentru cand utilizatorul se delogheaza si incearca sa acceseze o pagina la care nu are acces
* Mobile side – css content is not rendering due to size of screen

Dependency Injection pattern is used heavely in ASP.NET Core architecture

SOLID stands for:

* [**S** - Single-responsiblity Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#single-responsibility-principle)
* [**O** - Open-closed Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#open-closed-principle)
* [**L** - Liskov Substitution Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#liskov-substitution-principle)
* [**I** - Interface Segregation Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#interface-segregation-principle)
* [**D** - Dependency Inversion Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#dependency-inversion-principle)

**Single-responsibility Principle (SRP) states**:

A class should have one and only one reason to change, meaning that a class should have only one job.

**Open-closed Principle (OCP) states:**

Objects or entities should be open for extension but closed for modification.

**Liskov Substitution Principle states:**

Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.

This means that every subclass or derived class should be substitutable for their base or parent class.

**Interface segregation principle states:**

A client should never be forced to implement an interface that it doesn’t use, or clients shouldn’t be forced to depend on methods they do not use.

**Dependency inversion principle states:**

Entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.

This principle allows for decoupling.

IoC (Inversion of Control) – is a principle

DIP = Dependency Inversion Principle

Dependency injection pattern solves this problem by injecting dependent objects via a constructor, a property, or an interface.

The IoC container creates an object of the specified class and also injects all the dependency objects through a constructor, a property or a method at run time and disposes it at the appropriate time. This is done so that we don't have to create and manage objects manually.

Asynchronous programming is the default mode for ASP.NET Core and EF Core.

A web server has a limited number of threads available, and in high load situations all of the available threads might be in use. When that happens, the server can't process new requests until the threads are freed up. With synchronous code, many threads may be tied up while they aren't actually doing any work because they're waiting for I/O to complete. With asynchronous code, when a process is waiting for I/O to complete, its thread is freed up for the server to use for processing other requests. As a result, asynchronous code enables server resources to be used more efficiently, and the server is enabled to handle more traffic without delays.

If you're writing a new ASP.NET Core application that needs to work with relational data, then Entity Framework Core (EF Core) is the recommended way for your application to access its data. EF Core is an object-relational mapper (O/RM) that enables .NET developers to persist objects to and from a data source. It eliminates the need for most of the data access code developers would typically need to write. Like ASP.NET Core, EF Core has been rewritten from the ground up to support modular cross-platform applications. You add it to your application as a NuGet package, configure it in Startup, and request it through dependency injection wherever you need it.

DLL = cod compilat

Once you have installed EF Core, created a DbContext child type, and configured it in ConfigureServices, you are ready to use EF Core. You can request an instance of your DbContext type in any service that needs it, and start working with your persisted entities using LINQ as if they were simply in a collection. EF Core does the work of translating your LINQ expressions into SQL queries to store and retrieve your data.

The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

A transitive [functional dependency](https://www.guru99.com/dbms-functional-dependency.html) is when changing a non-key column, might cause any of the other non-key columns to change

Read about IIS Express Server