MÜHENDİSLİK FAKÜLTESİ
YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ
20…-20… AKADEMİK YILI

Öğrenci Adı Soyadı	: Enes Talha Koçyiğit
Öğrenci Numarası	: 2221251023
Tarih	: …/…/20…

Öğrenci Bilgileri		
Öğrenci No	2221251023	Fotoğraf
T.C. Kimlik No	44422381750	
Adı ve Soyadı	Enes Talha Koçyiğit	
Doğum Yeri ve Tarihi	İstanbul/Bakırköy – 23/09/2003	

Staj Yapılan Kuruluş/Firma

Adı	Fatih Sultan Mehmet Vakıf Üniversitesi Personal Daire Başkanlığı Yazılım Birimi
Adresi	Merkezefendi, Mevlevihane Cd. No:25, 34025 Zeytinburnu/İstanbul
Başlangıç Tarihi	07/07/2025
Bitiş Tarihi	18/08/2025

Sorumlu Kurum Yetkilisi

Adı Soyadı	
Unvan	
Onayı	

Bitiş Onayı

Staja Sayılan Gün	
Staj Komisyonu Onayı	
Bölüm Başkanlığı Onayı	

Yaprak No		Kısım	

Tarih	07/07/2025	Yapılan İş	

DAY 1: INTRODUCTION TO C# AND .NET PLATFORM FUNDAMENTALS

Today was the first day of my internship, during which I was introduced to the fundamentals of the C# programming language. At the beginning of the training, I gained detailed knowledge about the history of the C# language, its intended purpose, and its relationship with the .NET Framework. The development environment I used was Visual Studio, and I launched my first "Console Application" project via the project creation screen.

I explored the structure of a C# program through the Program.cs file. In particular, I learned that the static void Main(string[] args) method is the entry point of every C# application. I understood that this method is invoked by the operating system when the program is executed.

Inside the application, I practiced input and output operations using the Console.WriteLine() and Console.ReadLine() commands. For instance, I developed a simple application that prompts the user for a name and displays a welcome message, thereby reinforcing my understanding of basic I/O functions.

Later, I moved on to the topic of variable declarations. I studied data types such as int, string, double, and bool, and learned about their memory footprint and significance in terms of type safety. I also learned best practices in naming variables, such as using camelCase and avoiding conflicts with reserved keywords like int, string, or class.

In addition, I examined type conversions through examples, specifically the differences between methods like Convert.ToInt32() and int.Parse(). For example, I converted a user-provided string input into an integer and performed arithmetic operations on it.

At the end of the day, I implemented a sample project called the "calculator," developed by Sadık Turan. This project involved taking two values from the user, summing them, and displaying the result on the screen. Through this example, I thoroughly understood both the functional and structural logic of a basic C# application.

Overall, this first day was highly productive in helping me grasp the core concepts of the C# programming language. I became familiar with the Visual Studio interface and successfully developed my first application, laying a solid foundation for understanding object-oriented programming, control structures, and file operations in the following days.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	08/07/2025	Yapılan İş	

DAY 2: DEEP DIVE INTO DATA TYPES, VARIABLES, AND OPERATORS IN C#

During today's internship training, I focused extensively on the use of basic data types, variable declarations, and operators in the C# programming language. Through Sadık Turan's explanations, I gained a technical perspective on how each data type occupies memory and affects performance.

To begin with, I learned the usage areas of data types such as int, double, bool, char, string, and decimal, as well as their corresponding types within the .NET framework. I specifically noted that the decimal type should be preferred in financial calculations due to its higher precision in handling decimal values. Furthermore, I experimented with the var keyword, which allows the compiler to infer the type at compile time. While it improves code readability, I observed that improper usage may compromise type safety.

Following data types, I moved on to the topic of operators. I practiced using arithmetic operators (+, -, *, /, %) to perform basic calculations, and explored comparison operators (==, !=, >, <, >=, <=) and logical operators (&&, ||, !) through conditional expressions.

As a practical exercise, I developed a "grade point average calculator" application. In this project, two exam scores entered by the user were averaged, and the program printed whether the user passed or failed. This application integrated data input, type conversion (Convert.ToDouble()), and control flow structures such as if-else.

One of the most significant takeaways of the day was string manipulation. I practiced concatenating first and last names provided by the user using the + operator, and also used string interpolation (e.g., $"Hello {name}") to construct more readable and modern code structures.

By the end of the day, I diversified the examples I had written with various scenarios in order to better understand how variables behave in memory and to reinforce my knowledge of converting between different data types. Since these topics form the foundation of software development, this learning process was both critical and intellectually engaging.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	09/07/2025	Yapılan İş	

DAY 3: CONDITIONAL STRUCTURES AND DECISION-MAKING IN C#

In today's training, I explored one of the most essential concepts in programming: conditional structures and decision-making mechanisms. Through Sadık Turan's lessons, I thoroughly understood how logical flow is established using if, else if, else, and switch blocks.

At the beginning of the lesson, the importance of conditional statements was explained with real-world examples. I observed how real-life decision-making scenarios are expressed in programming—such as determining whether a student has passed based on their average score or whether a number is positive or negative.

First, I developed an application using if-else structures to determine whether a user-provided number is odd or even. With the help of the modulo operator (%), I checked whether the number was divisible by 2; if it was, it was classified as even, otherwise as odd. Next, I implemented a system using else if to categorize numeric grades. For example, a grade over 85 was labeled "Excellent", between 70–85 as "Good", and between 50–70 as "Average".

I then transitioned to the switch-case structure. I worked on a sample project that displayed the day of the week based on a numerical input from the user. I realized that this structure allows for simpler and more readable code blocks compared to long if-else chains.

In addition, I studied the ternary operator (?:), one of the shorthand conditional operators. I practiced expressions like number > 0 ? "Positive" : "Negative", which allow concise and readable evaluations.

At the end of the day, I worked on a mini-project provided by the instructor. I developed a system that calculates movie ticket prices based on user inputs such as age and student status. This project combined if statements, switch cases, and bool type logic to build a dynamic pricing mechanism.

These topics form the foundation of software logic as they enable the program to control its flow and respond dynamically based on external inputs. By modifying the sample exercises with my own variations, I reinforced my understanding of these conditional structures more effectively.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	10/07/2025	Yapılan İş	

DAY 4: LOOPS AND ITERATIVE STRUCTURES IN C#

In today's training, I learned in detail how loop mechanisms work in the C# programming language. With Sadık Turan's guidance, I studied the syntax and logical functionality of loop structures such as for, while, and do-while in depth.

Loops are essential in programming as they allow the repetition of specific operations as long as a condition is met. I first explored the general structure of the for loop. Through the form for (int i = 0; i < 10; i++), I practiced how to create operations that repeat a fixed number of times. For example, I worked on a sample program that calculates the sum of numbers from 1 to 10.

Next, I moved on to the while loop, which is ideal for handling cases where the starting point or number of repetitions is unknown or user-dependent. I wrote an application where the user is prompted to enter a password, and the loop continues until the correct password is provided. This helped me better understand how condition-based control is established.

With the do-while loop, I explored situations where the loop is guaranteed to execute at least once. I developed a program that asks the user whether they want to perform another operation, demonstrating the practical advantage of this loop structure.

Additionally, I learned about the break and continue keywords used within loops. I saw how break can immediately terminate a loop, while continue allows the program to skip the current iteration and move to the next one. Understanding these keywords is especially valuable for building more complex algorithms in the future.

At the end of the session, I developed a project that collects five numbers from the user, stores them in an array, and calculates their average. This allowed me to reinforce both the use of loops and arrays together.

Loops are indispensable in algorithm development as they enable repetitive tasks to be written in a concise and efficient manner. By expanding the examples I created in different ways, I improved my practical understanding. I believe the knowledge I gained today will greatly facilitate the development of my future projects.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	11/07/2025	Yapılan İş	

DAY 5: CLASSES, OBJECTS, AND OOP PRINCIPLES IN C#

Today, I began studying one of the fundamental pillars of software architecture in C#: Object-Oriented Programming (OOP). Under the guidance of Sadık Turan, I learned in detail about structures such as classes, objects, properties, and constructors.

I first grasped the concept and purpose of a class. Classes are templates that represent real-world entities. For example, a Product class can encapsulate all the attributes of a product. Using Visual Studio, I created a class named Product and defined the following properties:

```
public class Product
{
   public int Id { get; set; }
   public string Name { get; set; }
   public double Price { get; set; }
   public int Stock { get; set; }
}
```

Thanks to this class, each product can now store its own Id, Name, Price, and Stock values individually. I then proceeded to the process of object instantiation. Using the new keyword, I created a Product object in memory and assigned values to its properties, learning the logic behind object creation:

```
Product product1 = new Product();
product1.Id = 1;
product1.Name = "Pen";
product1.Price = 9.99;
product1.Stock = 100;
```

Afterwards, I learned how to use constructors to pass these values directly as parameters during object creation. I also realized how get and set accessors play a vital role in encapsulation and data security.

The training briefly introduced the four main principles of OOP: Encapsulation, Inheritance, Polymorphism, and Abstraction. Although the focus today was mainly on encapsulation and class usage, I had the opportunity to observe how these fundamental structures will come together in future projects.

Finally, I practiced using the ToString() method to print objects and display their values in a well-formatted output.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	14/07/2025	Yapılan İş	

DAY 6 – WHAT IS HTML? STRUCTURE, FIRST HTML PAGE, HEADINGS-PARAGRAPHS-FORMATTING TAGS

Today, during my internship, I was introduced to the fundamental language of web design: HTML. HTML is the backbone of any web page and is used to structure elements like text, images, links, and more in a browser.

I started by learning what HTML is and why it's used. An HTML file typically has a .html extension.
In this structure:
<!DOCTYPE html> defines the document as HTML5.
<html> is the root element.
<head> contains metadata such as title and encoding.
<body> holds the actual visible content.

Next, I explored heading tags, ranging from <h1> to <h6>. <h1> is the largest, usually used for the main title, while <h6> is the smallest.

I then learned about paragraphs using the <p> tag to organize text content cleanly:
To format text, I practiced with formatting tags:
<b>: bold text, <i>: italic text, <u>: underlined text

Later in the day, I created a page named "en-iyi-filmler.html" listing the top 3 movies to watch. I used several HTML tags:
<title>: Set the page title as "En iyi Filmler".
<h1>: Used for the main heading: "Muhakkak İzlenmesi Gereken 3 Film".
<p>: Paragraphs containing introductory text and movie descriptions.
<strong>: Applied for bold emphasis, such as the word "Not" and "IMDb 8 and above".
<i>: Italicized text for softer notes or highlights.
<h2>: Subheadings for each movie title: "Baba", "Eşkıya", and "The Lord of the Rings: The Return of the King".
Through this hands-on example, I reinforced my understanding of basic HTML structure and tag functionalities. I also learned how to semantically structure content to improve readability and styling potential. Moreover, I understood the role of <meta charset="UTF-8"> and responsive <meta name="viewport"> elements for better user experience.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	16/07/2025	Yapılan İş	

Day 7 – Developing a Web Page Using Advanced HTML Elements

On the seventh day of my internship, I worked on developing a modern web page by applying advanced elements of the HTML language. Throughout the day, I studied both theoretically and practically how to use elements such as images, links, page structuring, formatting tags, lists, tables, forms, and iframes.

I started by learning how to insert images into a web page using the <img> tag. I placed a total of three images representing movie posters on the page. Each image was positioned next to its corresponding movie description to provide a visual preview to users. Additionally, alt attributes were added to improve accessibility.
Next, I used the <a> tag to create links that direct users either to different parts of the same page or to external websites. These links were primarily used in the category list and navigation menu to enhance user experience.

To structure the page semantically and in a more readable way, I used HTML5 semantic tags such as <header>, <nav>, <main>, <section>, <article>, <aside>, and <footer>. These elements assigned distinct functionalities to each part of the page:
The <header> contained the site title and a short description.
The <nav> tag included internal navigation links.
The <aside> section displayed a categorized list of movie genres.
Each <article> element presented information for a specific movie, including a heading, description paragraph, and a visual.
The <footer> area provided copyright and contact information.

I also created three different tables using the <table> tag. These tables contained movie-related data or example datasets. Within the tables, I used <tr> and <td> tags to define rows and columns. This helped me understand how to structure and display data using tables in HTML.
In addition, I practiced creating unordered lists using <ul> and <li> tags. These lists were useful for displaying categories or grouped information. I also used formatting tags like <strong>, <em>, and <mark> to emphasize certain parts of the text. For example, key words were displayed in bold, italic, or highlighted style.
Finally, I studied the purpose and usage of <form> and <iframe> tags. Although these two elements were not directly used in the en-iyi-filmler.html file, I gained theoretical knowledge about them. I plan to apply these elements in later projects and exercises.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	17/07/2025	Yapılan İş	

DAY 8 – Learning and Practicing CSS Fundamentals

On the eighth day of my internship, I focused on learning the fundamentals of CSS (Cascading Style Sheets), which is used to style and design web pages. I reinforced the theoretical knowledge I gained by applying it in a hands-on project named uygulama-1.

I started by understanding what CSS is and why it is used. CSS allows us to control visual aspects of HTML content such as fonts, colors, spacing, alignment, and layout with ease.

I examined the basic CSS syntax: every style rule includes a selector and one or more declarations. For example: h1 { color: blue; font-size: 20px; }. Here, the h1 element is targeted and styled accordingly.

I learned the three main ways to apply CSS:
Inline CSS: Applied directly to an HTML tag using the style attribute.
Internal CSS: Written within a <style> block inside the <head> of an HTML document.
External CSS: Written in a separate .css file and linked to the HTML document via a <link> tag.

Then I explored id and class selectors. An id targets a unique element (#footer), while a class can be reused for multiple elements (.container). These selectors were used to customize style definitions in the project.

Furthermore, I learned about CSS specificity and inheritance. When multiple styles apply to an element, the browser uses a priority system to determine which rule to apply. ID selectors have a higher priority than class selectors. Some CSS properties are also inherited from parent elements.

Finally, I implemented everything I learned in a project.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	18/07/2025	Yapılan İş	

Day 9 – CSS Box Model and Visual UI Components

Today, I studied one of the most fundamental topics in CSS: the Box Model. Understanding the Box Model helped me see how spacing and borders work around HTML elements. The model consists of layers such as content, padding, border, and margin.

I explored how margin affects the space between elements, and how padding adjusts the space between content and the border.

I used borders to visually enclose elements.

With width and height, I controlled element sizing, and saw how the box-sizing: border-box; rule includes padding and border in the total width calculation.

Next, I learned about display and visibility, experimenting with values like inline-block, block, none, etc., and how they affect layout and visibility of elements.

I then designed a navigation bar using lists, and studied list-style, combinators, pseudo-classes, and pseudo-elements. I implemented effects like :hover, :active, and ::before to enhance interactivity and visual feedback in the menu.

I also worked on styling buttons, using background images, and custom fonts:
I applied background-image, background-position, and background-size properties.
I imported Google Fonts and used them to enhance typography.
I added Font Awesome icons for visual clarity and interaction.

I applied everything I learned in the uygulama-features.html file. The page contains a showcase section, a "How it Works" section with icons, and a featured area describing benefits. The design aimed to be modern, user-friendly, and visually balanced.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	21/07/2025	Yapılan İş	

Day 10 – Creating the General Skeleton of the Pages

Today, I focused on building the general layout of the web application using my HTML/CSS skills. I began by designing the files index.html, login.html, register.html, create.html, and post.html. These pages were built using Bootstrap 5 for a responsive and modern UI design.

On the index.html page, I designed a homepage that displays shared ideas using Bootstrap cards. A navigation bar allows users to access login, registration, and idea submission pages. Each card includes buttons for viewing comments and liking ideas.

The login.html and register.html files include well-structured and styled forms. These forms use Bootstrap form controls and input validation features.

The create.html page provides a structured form for users to share their ideas. The form includes input fields for title and description.

In the post.html file, the detail view of an idea along with its comments is designed.

After finalizing the initial version, I continued improving the layout and created a refined version. In this updated version, I implemented style enhancements, added responsive design improvements, updated the navigation bar layout, and added interactive buttons using JavaScript.

By the end of the day, I had completed the structural framework for a "Share Your Idea" platform. I paid special attention to user interface flow, content hierarchy, and the overall HTML structure while building the application pages.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	22/07/2025	Yapılan İş	

Day 11 – Introduction to .NET Core and MVC Structure

On the eleventh day of my internship, I shifted from the frontend side of web development to the backend. As part of this process, I continued following Sadık Turan's web development course, which provided structured guidance on backend concepts and practical implementations First, I studied the concept of Backend. I learned that the backend is the server-side of an application that is invisible to users but manages the core functionality. Database connections, user authentication, business rules, and APIs are all handled in this layer.

Then, I explored .NET Core technology. I understood that this open-source and cross-platform framework developed by Microsoft provides significant advantages in building modern web applications. Its strengths lie in performance, security, and a vast library ecosystem.

After that, I proceeded with the installation of .NET Core. I installed the SDK and runtime packages, and verified the installation by checking the version with the dotnet --version command.

To start project development, I examined two different IDE options:
With Visual Studio Code, I learned how to create a project through the command line using dotnet new mvc.

With Visual Studio IDE, I practiced creating a project through the graphical interface, which offers more comprehensive tools for .NET development.

Later, I studied the MVC (Model-View-Controller) architecture. I understood how splitting the project into three layers improves both structure and maintainability. Additionally, I reviewed how the HTTP protocol works, where requests are sent from the browser to the server, and the server responds back.

Finally, I was introduced to Controllers, Action Methods, and Default Action Methods. I learned that controllers handle incoming user requests and action methods generate responses. For example, I created a ProductsController and defined action methods to list products.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	23/07/2025	Yapılan İş	

Day 12 – Entity Framework and Database Operations

On the twelfth day of my internship, I focused on database operations in .NET Core using Entity Framework.

I first reviewed the concept of a database and the basics of SQL, understanding that SQL is the standard language for inserting, updating, deleting, and querying data. Then, I studied ORM (Object Relational Mapping), which acts as a bridge between object-oriented programming and relational databases, allowing developers to manage tables with C# classes instead of raw SQL.

After that, I worked on Entity Framework (EF), the most commonly used ORM tool in .NET Core projects. I installed EF, created Entity classes like Product (with properties such as Id, Name, Price, and ImageUrl), and a DbContext class to connect entities with the database. I configured the Connection String in *appsettings.json* and applied Migrations with Add-Migration and Update-Database to generate the database. I also used Data Seeding to insert initial data.

For querying, I used LINQ expressions to list products, filter by category, and fetch product details. In practice, I built parts of an E-commerce template: a Product listing page, Product detail page, and a filtering function by category. I also created a Category Entity with a category menu, and implemented Navbar and Slider ViewComponents to make the application dynamic. Finally, I added category-based filtering and similar product listing to enhance the project.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	24/07/2025	Yapılan İş	

Day 13 – Identity and User Management

On the thirteenth day of my internship, I studied ASP.NET Core Identity and focused on user management with practical implementations.

I first explored What is Identity?. I learned that Identity is a built-in library in ASP.NET Core that simplifies authentication and authorization processes. It helps developers handle user sign-in/sign-out, password management, roles, and permissions in a standardized way.

Next, I completed the Identity installation. I installed the necessary NuGet packages and configured the services in Program.cs. This prepared the infrastructure required for user-based operations.

Then, I practiced adding users with the UserManager class and created a user list to display all users in the admin panel.

I continued with the Configure Identity step, where I set password policies and email confirmation requirements. This ensured a more secure user experience.

Afterwards, I studied Authentication in detail. For login operations, I implemented Cookie Authentication, allowing the system to track user sessions via cookies.

For better usability, I configured navigation links dynamically: showing “Profile” and “Logout” for logged-in users, and “Register” and “Login” for guests.

Next, I worked on role management, including creating, updating, and deleting roles. For example, “Admin” users could add products, while “User” roles were limited to viewing products only.

In the practical part, I tested user creation and successfully performed login/logout scenarios. I also implemented authorization, restricting access to certain pages based on roles.

Finally, I learned Identity Seed Database, which allows automatic creation of default users and roles when the project starts.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	25/07/2025	Yapılan İş	

Day 14 – Installing Visual Studio and Creating the Layered Architecture

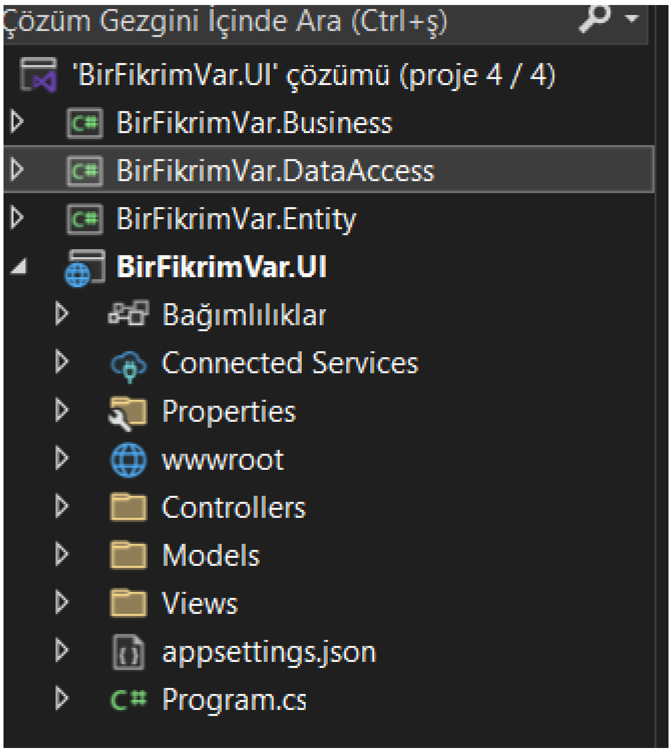
On the first day of the internship, the development environment was prepared and the framework of the software architecture was established. First, Visual Studio 2022 was installed along with all the necessary components for .NET development. The reason for choosing Visual Studio is that it provides the most comprehensive tool support for .NET projects, offers powerful debugging capabilities, and makes project management easier with its graphical interface.
After that, a Layered Architecture approach was adopted to structure the project. Layered architecture increases both readability and maintainability by separating the software into specific responsibilities. With this approach, if any changes are required in the future, only the relevant layer is affected, minimizing dependencies between layers.
The layers created for the project were as follows:

1. BirFikrimVar.Entity (Entity Layer)
This layer contains the classes that represent the database tables. For example: Idea, User, Comment.

2. BirFikrimVar.DataAccess (Data Access Layer)
In this layer, Entity Framework is used to create the DbContext class. DbContext acts as a bridge that directly communicates with the database.

3. BirFikrimVar.Business (Business Layer)
This layer handles the business logic of the application. For example, when adding a new idea, validation is performed, and it is checked whether the data belongs to the correct user.
The purpose here is to centralize all business rules.
This prevents duplication of logic in the DataAccess or UI layers.

4. BirFikrimVar.UI (User Interface Layer)
This is the presentation layer designed according to the MVC pattern. Controllers handle user requests, call the appropriate Service or Repository, and pass the retrieved results to the View.



Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	28/07/2025	Yapılan İş	

Day 15 – Model, DTO, and ViewModel Classes in the Entity Layer

Today, I created the core database building blocks of the project under BirFikrimVar.Entity.

Model Classes (Entity/Models):

User.cs: Contains user information (Id, FullName, Email, Password).
Idea.cs: Stores idea submission details (Title, Content, UserId, CreatedAt).
Comment.cs: Holds the comments made on ideas.
Like.cs: Stores information about which idea a user liked.
CommentLike.cs: Defines the relationship for likes on comments.

Each model was enriched with EF Core compatibility attributes such as [Key], [Required], and navigation properties to establish relationships.

DTO Classes (Entity/DTOs):

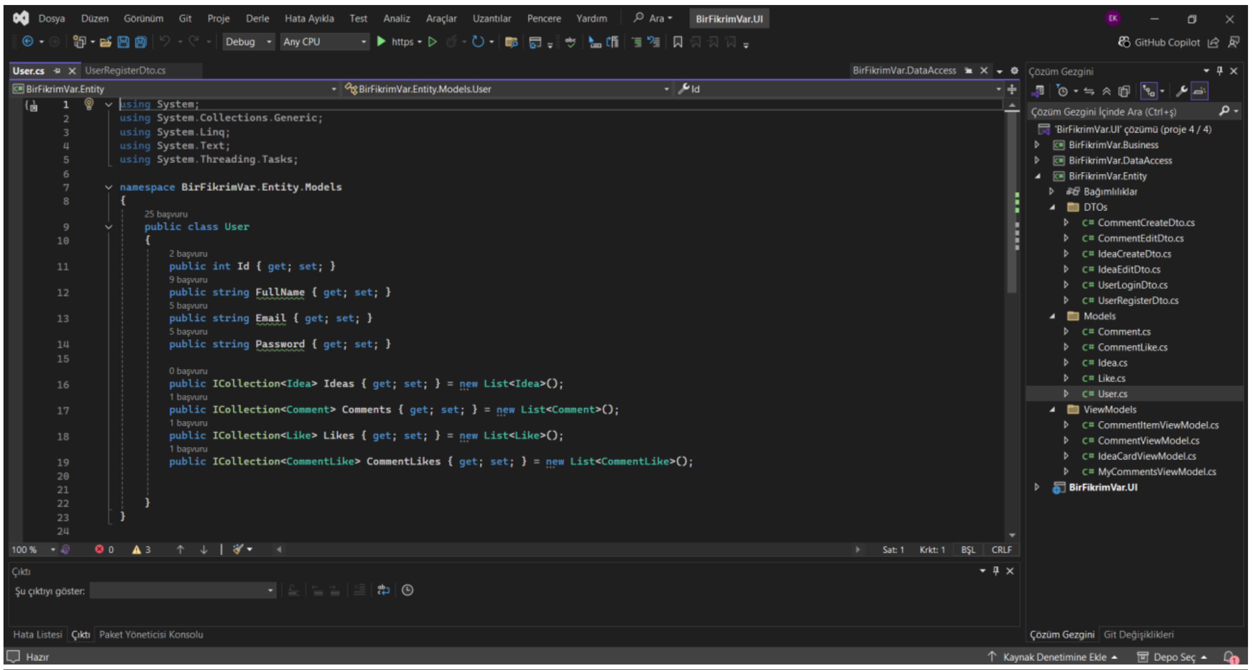
UserRegisterDto, UserLoginDto: Carry data from forms, acting as a filter between the View and the Model.
IdeaCreateDto, IdeaEditDto: Used in idea creation and editing forms.

By using DTOs, sensitive data (such as passwords) is not directly exposed to the Entity. This approach improves security and simplifies data handling.

ViewModel Classes (Entity/ViewModels):

IdeaCardViewModel: Transfers combined data to the View, such as username, idea content, and comment count, coming from different tables.
CommentItemViewModel: Used for displaying each individual comment.

This structure made it easier to transfer data, especially in Razor View files, and allowed me to write cleaner code in the controller layer.

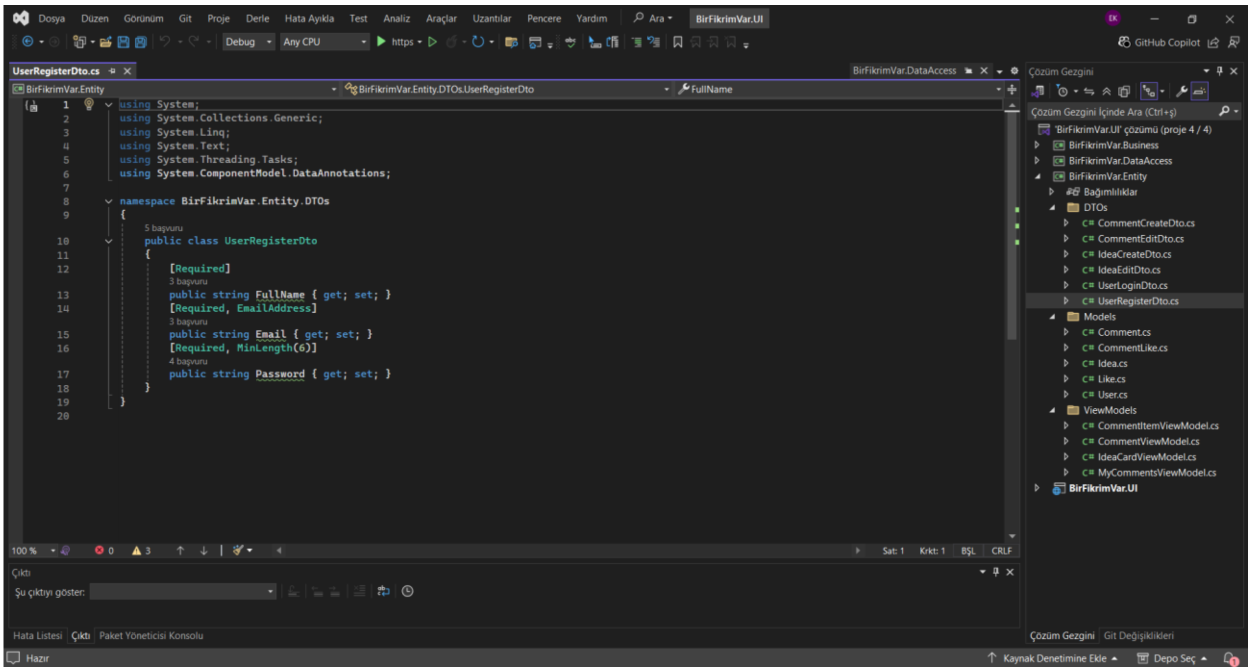


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BirFikrimVar.Entity.Models
{
    25 başvuru
    public class User
    {
        2 başvuru
        public int Id { get; set; }
        9 başvuru
        public string FullName { get; set; }
        5 başvuru
        public string Email { get; set; }
        5 başvuru
        public string Password { get; set; }

        0 başvuru
        public ICollection<Idea> Ideas { get; set; } = new List<Idea>();
        1 başvuru
        public ICollection<Comment> Comments { get; set; } = new List<Comment>();
        1 başvuru
        public ICollection<Like> Likes { get; set; } = new List<Like>();
        1 başvuru
        public ICollection<CommentLike> CommentLikes { get; set; } = new List<CommentLike>();


    }
}
```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace BirFikrimVar.Entity.DTOs
{
    5 başvuru
    public class UserRegisterDto
    {
        [Required]
        3 başvuru
        public string FullName { get; set; }
        [Required, EmailAddress]
        3 başvuru
        public string Email { get; set; }
        [Required, MinLength(6)]
        4 başvuru
        public string Password { get; set; }
    }
}
```

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	29/07/2025	Yapılan İş	

Day 16 – Creating Database Relationships with the AppDbContext Class and Preparing for Migration

In today's work, I developed the AppDbContext class, which constitutes the foundation of the database access layer based on Entity Framework Core. This class serves as the central structure that establishes communication between the application code and the database throughout the project.

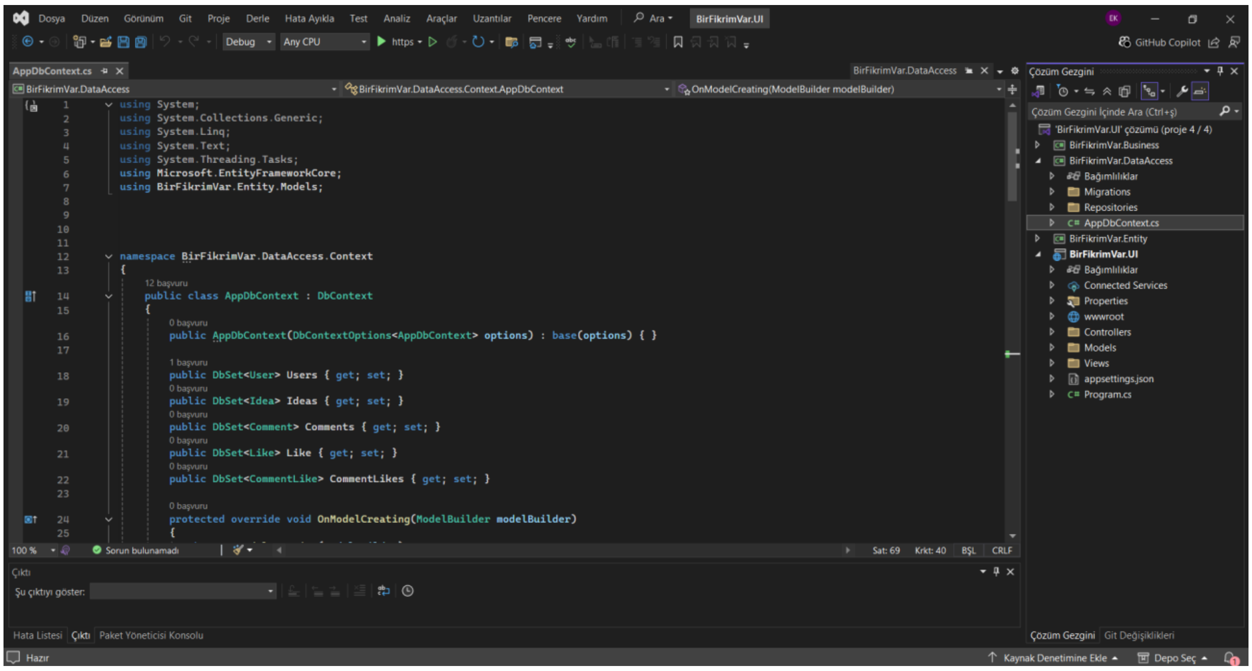
In ASP.NET Core projects, the DbContext class is the core component that manages all database operations. Accordingly, I defined a custom class named AppDbContext, which inherits from EF Core's DbContext. Within this class, the DbSet<T> structures were defined to represent the corresponding database tables for the User, Idea, Comment, Like, and CommentLike models. This mapping ensures that each model in the application is accurately linked to its representation in the database.

The OnModelCreating method was utilized to define specific relationships between tables. Within this method, a composite primary key was defined for the Like table (UserId and IdeaId) and for the CommentLike table (UserId and CommentId). This prevents users from liking the same idea or comment multiple times, thereby preserving database integrity.

Subsequently, preparations were made for the physical database creation process using Entity Framework Core's Migration system. By executing the command dotnet ef migrations add InitialCreate, migration classes were generated, and with the command dotnet ef database update, these definitions were applied to SQL Server to create the actual tables.

During the process, the error "No project was found" was encountered. This issue was resolved by switching to the correct directory containing the .csproj file. Additionally, the packages Microsoft.EntityFrameworkCore.Design and Microsoft.EntityFrameworkCore.Tools were installed to enable migration support.

In conclusion, the AppDbContext class established itself as the core of the project's data access layer, providing a solid foundation for the Repository and Service layers that will interact with the database in the subsequent stages of development.


Dosya Düzen Görünüm Git Proje Derle Hata Ayıkla Test Analiz Araçlar Uzantılar Pencere Yardım Ara BirFikrimVar.UI
Debug Any CPU https
GitHub Copilot
AppDbContext.cs
BirFikrimVar.DataAccess
BirFikrimVar.DataAccess.Context.AppDbContext
OnModelCreating(ModelBuilder modelBuilder)
Çözüm Gezgini
Çözüm Gezgini İçinde Ara (Ctrl+ş)
'BirFikrimVar.UI' çözümü (proje 4 / 4)
BirFikrimVar.Business
BirFikrimVar.DataAccess
Bağımlılıklar
Migrations
Repositories
AppDbContext.cs
BirFikrimVar.Entity
BirFikrimVar.UI
Bağımlılıklar
Connected Services
Properties
wwwroot
Controllers
Models
Views
appsettings.json
Program.cs
100 % Sorun bulunamadı Sat: 69 Krkt: 40 BSL CRLF
Çıktı
Şu çıktıyı göster:
Hata Listesi Çıktı Paket Yöneticisi Konsolu
Çözüm Gezgini Git Değişiklikleri
Hazır Kaynak Denetimine Ekle Depo Seç

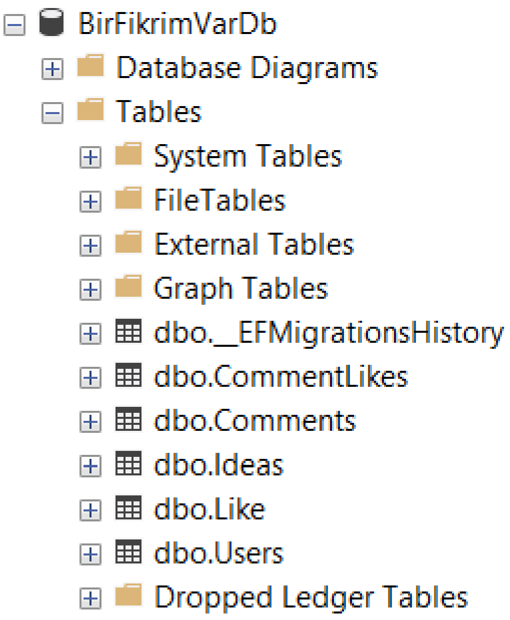
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using BirFikrimVar.Entity.Models;



namespace BirFikrimVar.DataAccess.Context
{
    12 başvuru
    public class AppDbContext : DbContext
    {
        0 başvuru
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

        1 başvuru
        public DbSet<User> Users { get; set; }
        0 başvuru
        public DbSet<Idea> Ideas { get; set; }
        0 başvuru
        public DbSet<Comment> Comments { get; set; }
        0 başvuru
        public DbSet<Like> Like { get; set; }
        0 başvuru
        public DbSet<CommentLike> CommentLikes { get; set; }

        0 başvuru
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
```



- BirFikrimVarDb
  - Database Diagrams
  - Tables
    - System Tables
    - FileTables
    - External Tables
    - Graph Tables
    - dbo.__EFMigrationsHistory
    - dbo.CommentLikes
    - dbo.Comments
    - dbo.Ideas
    - dbo.Like
    - dbo.Users
    - Dropped Ledger Tables

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	30/07/2025	Yapılan İş	

Day 17 – Implementation of the Generic Repository Pattern and Transition to the Service Layer

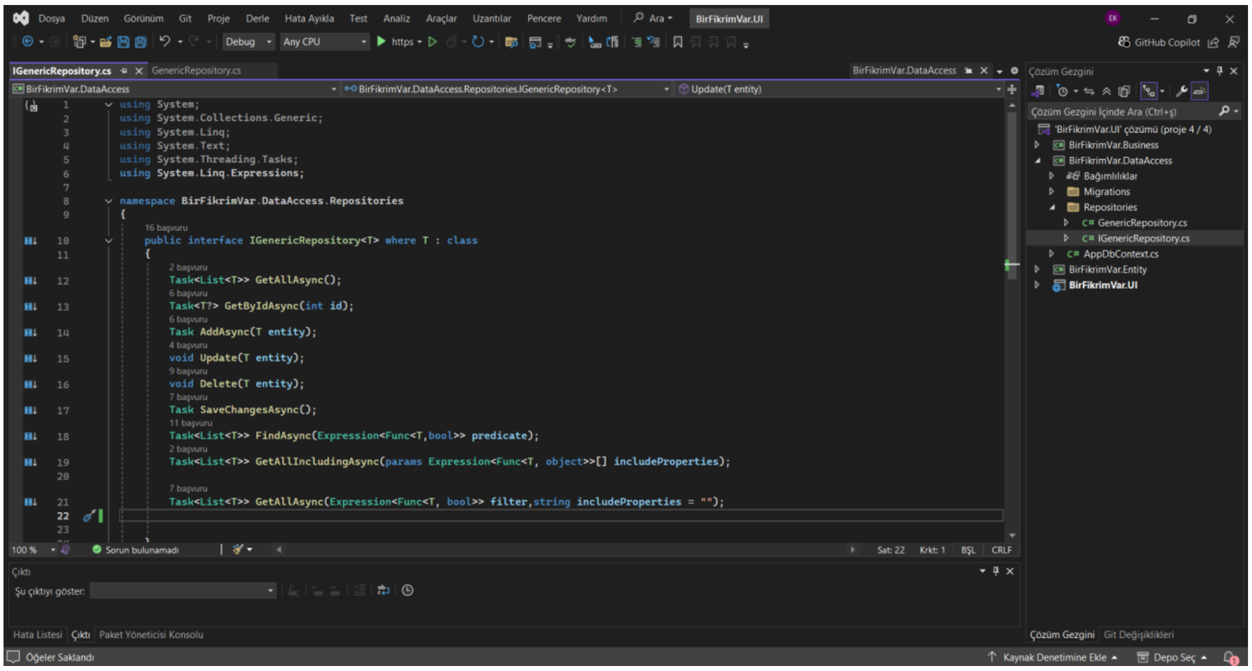
In today's work, I applied the Generic Repository pattern to strengthen the data access infrastructure of the project. This allowed me to create a structure that eliminates code repetition and can be reused across all entity models. Following this, I developed the first service interfaces that would utilize this repository structure within the Service layer.

Initially, an interface named IGenericRepository<T> was defined, containing common methods (GetAllAsync, GetByIdAsync, AddAsync, Update, Delete) that can be applied to any entity class. Subsequently, its implementation GenericRepository<T> was created, leveraging EF Core's DbSet<T> to dynamically interact with database tables. This means that depending on the entity type parameter, the repository automatically connects to the corresponding table.

After establishing this infrastructure, the focus shifted to the Service layer, where business logic is maintained. As part of this, the IUserService interface was introduced to handle user-specific operations. Then, the UserManager class was implemented to realize these operations by communicating with the repository. Within this class, methods were written to retrieve users by ID or email, as well as to list or add new users. Furthermore, this layer provides a suitable foundation for adding validation, logging, or additional business rules in the future.

Finally, the entire structure was integrated into the application through Dependency Injection in the Program.cs file. As a result, the Controller layer now communicates exclusively with the Service layer rather than accessing the database directly. This approach reduces inter-layer dependencies (Loose Coupling) and improves testability.

At the end of the day, the structure was tested via the HomeController, and it was confirmed that data retrieval operations functioned correctly. This step represents a significant milestone in ensuring both sustainability and extensibility within the project architecture.

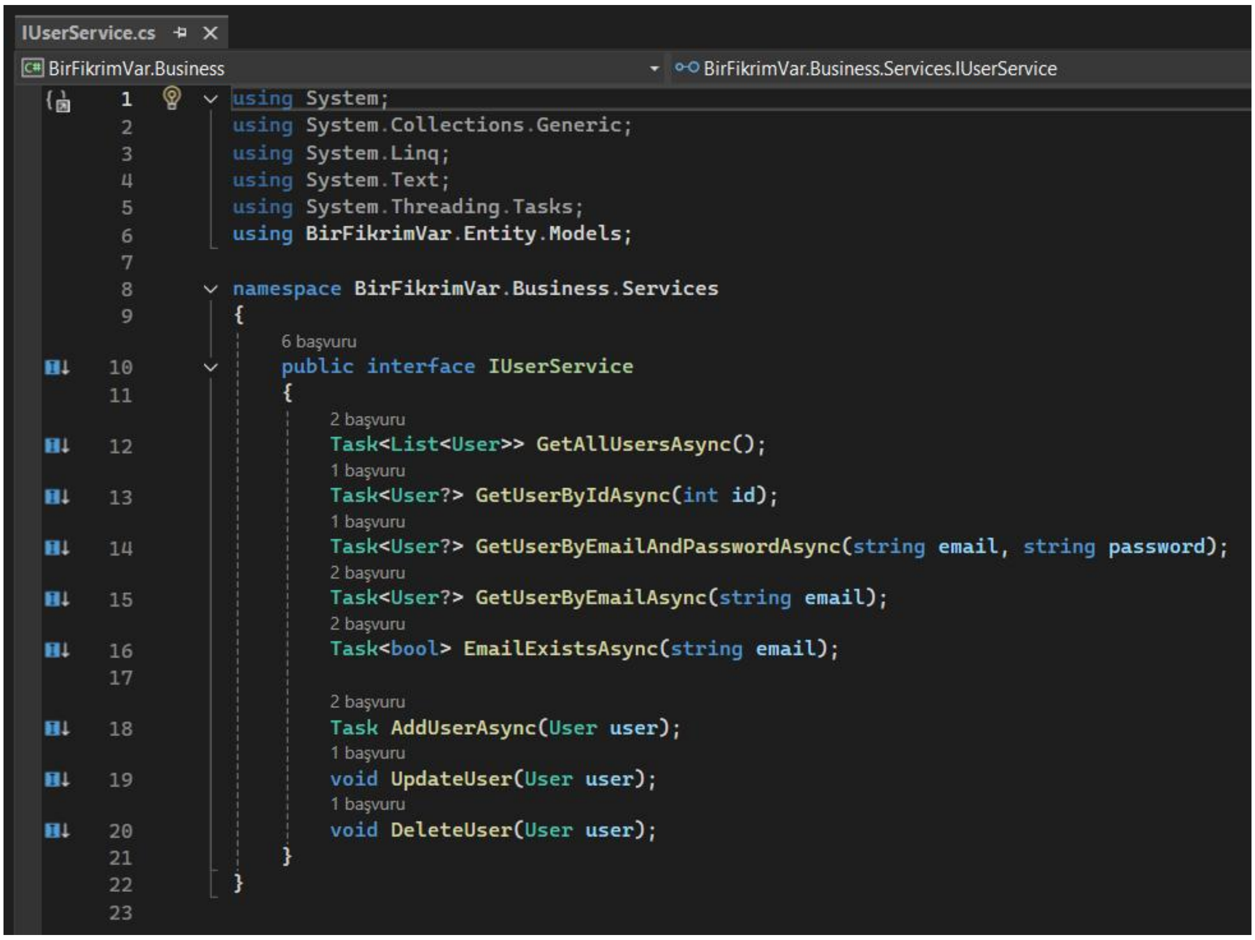


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Linq.Expressions;

namespace BirFikrimVar.DataAccess.Repositories
{
    16 başvuru
    public interface IGenericRepository<T> where T : class
    {
        2 başvuru
        Task<List<T>> GetAllAsync();
        6 başvuru
        Task<T?> GetByIdAsync(int id);
        6 başvuru
        Task AddAsync(T entity);
        4 başvuru
        void Update(T entity);
        9 başvuru
        void Delete(T entity);
        7 başvuru
        Task SaveChangesAsync();
        11 başvuru
        Task<List<T>> FindAsync(Expression<Func<T,bool>> predicate);
        2 başvuru
        Task<List<T>> GetAllIncludingAsync(params Expression<Func<T, object>>[] includeProperties);

        7 başvuru
        Task<List<T>> GetAllAsync(Expression<Func<T, bool>> filter,string includeProperties = "");

```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using BirFikrimVar.Entity.Models;

namespace BirFikrimVar.Business.Services
{
    6 başvuru
    public interface IUserService
    {
        2 başvuru
        Task<List<User>> GetAllUsersAsync();
        1 başvuru
        Task<User?> GetUserByIdAsync(int id);
        1 başvuru
        Task<User?> GetUserByEmailAndPasswordAsync(string email, string password);
        2 başvuru
        Task<User?> GetUserByEmailAsync(string email);
        2 başvuru
        Task<bool> EmailExistsAsync(string email);

        2 başvuru
        Task AddUserAsync(User user);
        1 başvuru
        void UpdateUser(User user);
        1 başvuru
        void DeleteUser(User user);
    }
}

```

Kontrol Eden		Onay	
Yaprak No		Kısım	

Tarih	31/07/2025	Yapılan İş	

Day 18–AutoMapper, FluentValidation, and User Registration/Login (AccountController)

In today's work, I developed the user management module of the project. I implemented the necessary Controller, View, Validation, and Service components for user registration and login processes. To simplify business logic, I integrated the AutoMapper and FluentValidation libraries into the project.

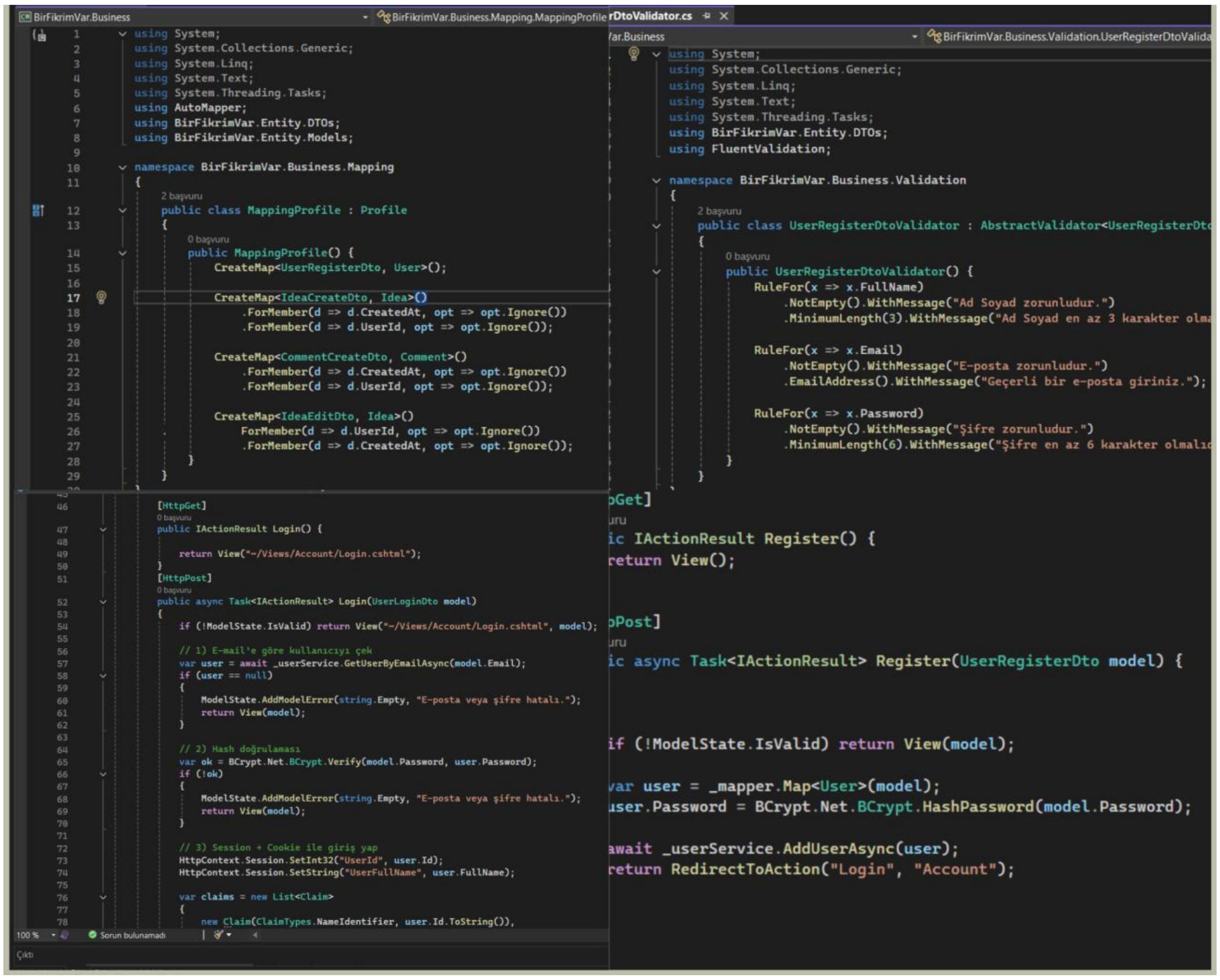
By using AutoMapper, data mappings between DTO classes and Entity classes were automated, eliminating the need for manual property assignments. In the MappingProfile class, the mappings between DTOs and Entities were defined, and these mappings were automatically applied when the application started.

Additionally, FluentValidation was employed to define flexible validation rules for user input. For instance, within the UserRegisterDtoValidator class, it was enforced that the FullName field cannot be empty and must not exceed 50 characters, the Email field must be in a valid format, and the Password field must contain at least 6 characters. As a result, users receive instant feedback when submitting incomplete or invalid data.

User registration and login functionalities were implemented in the AccountController. In the Register method, user input was validated, passwords were secured through BCrypt hashing, and new accounts were persisted in the database through the service layer. In the Login method, the user's email and password were verified, and upon successful authentication, Session and Cookie-based authentication were configured. Furthermore, the Claims structure was utilized to securely store user identity information.

On the View side, Razor-based forms were created for user interaction. These forms were automatically validated against FluentValidation rules, ensuring robust data entry control.

In conclusion, today's implementation successfully completed a secure user management module, where validation and data transformation processes were standardized through the integration of professional libraries such as AutoMapper and FluentValidation.


BirFikrimVar.Business
BirFikrimVar.Business.Mapping.MappingProfile
rDtoValidator.cs
/ar.Business
BirFikrimVar.Business.Validation.UserRegisterDtoValida
100 %
Sorun bulunamadı
Çıktı

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using AutoMapper;
using BirFikrimVar.Entity.DTOs;
using BirFikrimVar.Entity.Models;

namespace BirFikrimVar.Business.Mapping
{
    2 başvuru
    public class MappingProfile : Profile
    {
        0 başvuru
        public MappingProfile() {
            CreateMap<UserRegisterDto, User>();

            CreateMap<IdeaCreateDto, Idea>()
                .ForMember(d => d.CreatedAt, opt => opt.Ignore())
                .ForMember(d => d.UserId, opt => opt.Ignore());

            CreateMap<CommentCreateDto, Comment>()
                .ForMember(d => d.CreatedAt, opt => opt.Ignore())
                .ForMember(d => d.UserId, opt => opt.Ignore());

            CreateMap<IdeaEditDto, Idea>()
                ForMember(d => d.UserId, opt => opt.Ignore())
                .ForMember(d => d.CreatedAt, opt => opt.Ignore());
        }
    }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using BirFikrimVar.Entity.DTOs;
using FluentValidation;

namespace BirFikrimVar.Business.Validation
{
    2 başvuru
    public class UserRegisterDtoValidator : AbstractValidator<UserRegisterDto
    {
        0 başvuru
        public UserRegisterDtoValidator() {
            RuleFor(x => x.FullName)
                .NotEmpty().WithMessage("Ad Soyad zorunludur.")
                .MinimumLength(3).WithMessage("Ad Soyad en az 3 karakter olma

            RuleFor(x => x.Email)
                .NotEmpty().WithMessage("E-posta zorunludur.")
                .EmailAddress().WithMessage("Geçerli bir e-posta giriniz.");

            RuleFor(x => x.Password)
                .NotEmpty().WithMessage("Şifre zorunludur.")
                .MinimumLength(6).WithMessage("Şifre en az 6 karakter olmalıd
        }
    }
```

```
        [HttpGet]
        0 başvuru
        public IActionResult Login() {

            return View("~/Views/Account/Login.cshtml");
        }
        [HttpPost]
        0 başvuru
        public async Task<IActionResult> Login(UserLoginDto model)
        {
            if (!ModelState.IsValid) return View("~/Views/Account/Login.cshtml", model);

            // 1) E-mail'e göre kullanıcıyı çek
            var user = await _userService.GetUserByEmailAsync(model.Email);
            if (user == null)
            {
                ModelState.AddModelError(string.Empty, "E-posta veya şifre hatalı.");
                return View(model);
            }

            // 2) Hash doğrulaması
            var ok = BCrypt.Net.BCrypt.Verify(model.Password, user.Password);
            if (!ok)
            {
                ModelState.AddModelError(string.Empty, "E-posta veya şifre hatalı.");
                return View(model);
            }

            // 3) Session + Cookie ile giriş yap
            HttpContext.Session.SetInt32("UserId", user.Id);
            HttpContext.Session.SetString("UserFullName", user.FullName);

            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
```

```
pGet]
uru
ic IActionResult Register() {
return View();


pPost]
uru
ic async Task<IActionResult> Register(UserRegisterDto model) {


if (!ModelState.IsValid) return View(model);

var user = _mapper.Map<User>(model);
user.Password = BCrypt.Net.BCrypt.HashPassword(model.Password);

await _userService.AddUserAsync(user);
return RedirectToAction("Login", "Account");
```

Kontrol Eden		Onay	
Yaprak No		Kısım	

Tarih	01/08/2025	Yapılan İş	

Day 19 – Idea Sharing System: IdeaController, IdeaCreateDto, ViewModel, and Listing

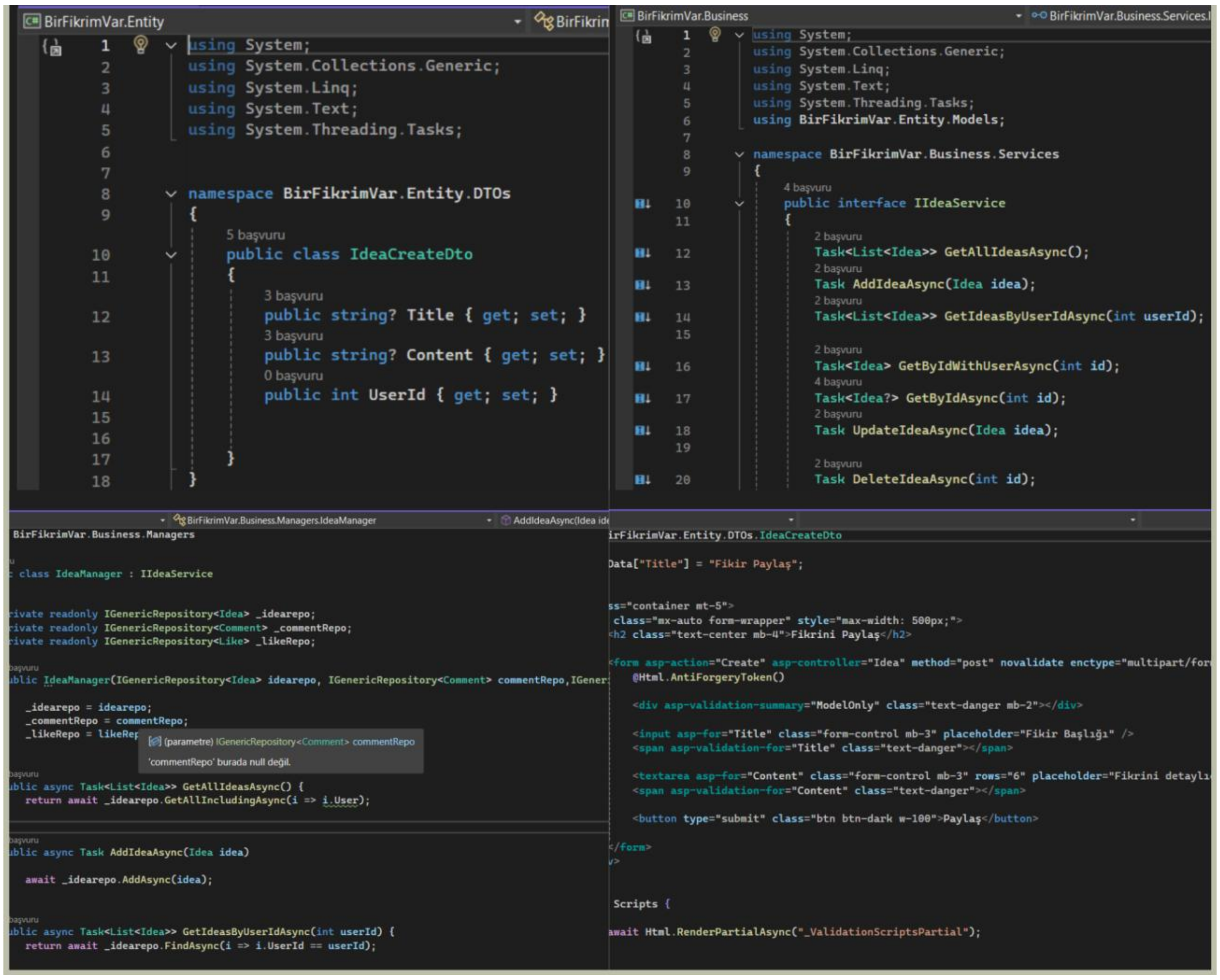
In today's work, I developed the module that enables users to share ideas within the system. Following the MVC principles, I implemented the processes of adding ideas through a form, saving them into the database, and listing them on the homepage by using DTOs, ViewModels**,** Controllers, Services, and Razor Views.

To begin with, the IdeaCreateDto class was introduced to establish a secure data transfer mechanism instead of directly using the Entity. This DTO contained only the Title and Content fields, thereby preventing unsafe practices such as requesting UserId or CreatedAt from the user.
Next, in the Business layer, the IIdeaService interface and its implementation, IdeaManager, were created. These components handled the addition of new ideas as well as retrieving all ideas along with user information. At later stages, EF Core's Include method will be employed to fetch related user details within the same query.
On the Controller side, the IdeaController was implemented to manage idea creation operations. The GET Create() method returned an empty form, while the POST Create() method validated user input, performed the DTO → Entity mapping through AutoMapper, and saved the idea into the database by assigning the user information from the active session.

In the View layer, the Create.cshtml Razor file was designed to provide the user interface for submitting ideas. Users entered the idea's title and content, and the form data was submitted to the server via a POST request.

For listing ideas on the homepage, the IdeaCardViewModel class was introduced. This ViewModel aggregated the idea's title, content, and the author's full name, and the data was passed through the HomeController to the Index page, allowing the display of ideas in a structured card layout.

During development, the most significant issue encountered was a NullReferenceException, which occurred when an unauthenticated user attempted to submit an idea. This was resolved by applying the [Authorize] attribute to the Create method, ensuring that only logged-in users could add ideas.
In conclusion, today's implementation successfully established a secure and structured idea sharing system, allowing ideas to be created and displayed on the homepage while adhering to a layered and maintainable architecture.



BirFikrimVar.Entity

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace BirFikrimVar.Entity.DTOs
{
    5 başvuru
    public class IdeaCreateDto
    {
        3 başvuru
        public string? Title { get; set; }
        3 başvuru
        public string? Content { get; set; }
        0 başvuru
        public int UserId { get; set; }


    }
}
```

BirFikrimVar.Business — BirFikrimVar.Business.Services.I

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using BirFikrimVar.Entity.Models;

namespace BirFikrimVar.Business.Services
{
    4 başvuru
    public interface IIdeaService
    {
        2 başvuru
        Task<List<Idea>> GetAllIdeasAsync();
        2 başvuru
        Task AddIdeaAsync(Idea idea);
        2 başvuru
        Task<List<Idea>> GetIdeasByUserIdAsync(int userId);

        2 başvuru
        Task<Idea> GetByIdWithUserAsync(int id);
        4 başvuru
        Task<Idea?> GetByIdAsync(int id);
        2 başvuru
        Task UpdateIdeaAsync(Idea idea);

        2 başvuru
        Task DeleteIdeaAsync(int id);
```

BirFikrimVar.Business.Managers.IdeaManager — AddIdeaAsync(Idea ide

```
 BirFikrimVar.Business.Managers

u
 class IdeaManager : IIdeaService


rivate readonly IGenericRepository<Idea> _idearepo;
rivate readonly IGenericRepository<Comment> _commentRepo;
rivate readonly IGenericRepository<Like> _likeRepo;

başvuru
ublic IdeaManager(IGenericRepository<Idea> idearepo, IGenericRepository<Comment> commentRepo,IGener

  _idearepo = idearepo;
  _commentRepo = commentRepo;
  _likeRepo = likeRep   (parametre) IGenericRepository<Comment> commentRepo
                        'commentRepo' burada null değil.
başvuru
ublic async Task<List<Idea>> GetAllIdeasAsync() {
  return await _idearepo.GetAllIncludingAsync(i => i.User);

başvuru
ublic async Task AddIdeaAsync(Idea idea)

  await _idearepo.AddAsync(idea);


başvuru
ublic async Task<List<Idea>> GetIdeasByUserIdAsync(int userId) {
  return await _idearepo.FindAsync(i => i.UserId == userId);
```

```
irFikrimVar.Entity.DTOs.IdeaCreateDto

Data["Title"] = "Fikir Paylaş";


ss="container mt-5">
 class="mx-auto form-wrapper" style="max-width: 500px;">
<h2 class="text-center mb-4">Fikrini Paylaş</h2>

<form asp-action="Create" asp-controller="Idea" method="post" novalidate enctype="multipart/for
    @Html.AntiForgeryToken()

    <div asp-validation-summary="ModelOnly" class="text-danger mb-2"></div>

    <input asp-for="Title" class="form-control mb-3" placeholder="Fikir Başlığı" />
    <span asp-validation-for="Title" class="text-danger"></span>

    <textarea asp-for="Content" class="form-control mb-3" rows="6" placeholder="Fikrini detaylı
    <span asp-validation-for="Content" class="text-danger"></span>

    <button type="submit" class="btn btn-dark w-100">Paylaş</button>

</form>
v>


 Scripts {

await Html.RenderPartialAsync("_ValidationScriptsPartial");
```

Kontrol Eden		Onay	
Yaprak No		Kısım	

Tarih	04/08/2025	Yapılan İş	

Day 20 – Comment System: CommentController, CommentCreateDto, ViewModel, and Service Layer

In today's work, I developed the comment system that enables users to leave comments on shared ideas. For this purpose, an end-to-end structure was built using DTOs, ViewModels, Controllers, Services, and Razor Views.

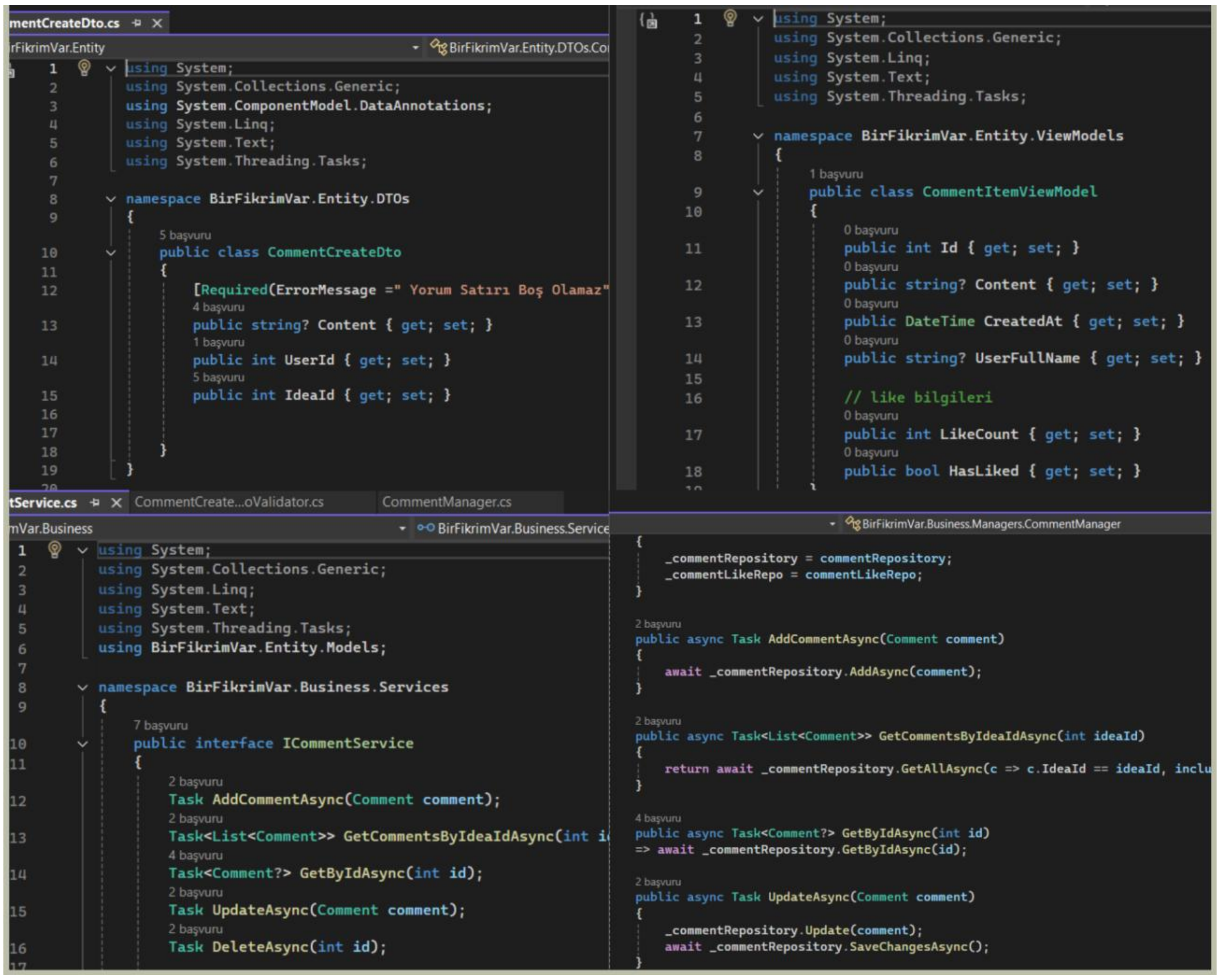
To begin with, the CommentCreateDto class was introduced to collect only the comment text and the related idea identifier from the user. This approach ensured that sensitive fields such as UserId were not exposed, and only the necessary data was transmitted from the form.
For displaying comments, the CommentItemViewModel class was created. This ViewModel combined the commenter's full name, comment content, and creation date into a single structure, making it directly usable within Razor Views.

On the Controller side, the CommentController was implemented. Within the Add method, form data received as DTO was mapped to the Entity type using AutoMapper. The user's identity was retrieved from the active session and assigned to the comment, while the CreatedAt property was automatically set by the system. Once the comment was created, the user was redirected to the homepage.

In the View layer, the Index.cshtml file was modified to include a comment form below each idea card. A hidden IdeaId field ensured that the controller correctly identified the related idea for each submitted comment.

In the Service layer, the ICommentService interface and its implementation, CommentManager, were developed. This layer facilitated both saving new comments to the database and retrieving comments associated with a specific idea. By doing so, it acted as a bridge between the Controller and Repository layers in accordance with layered architecture principles.
During testing, comments made by different users were correctly recorded with the appropriate user information. The primary issue encountered was a NullReferenceException, which occurred when unauthenticated users attempted to submit comments. This was resolved by applying the [Authorize] attribute, restricting comment functionality to logged-in users only.

In conclusion, the comment system was designed around the flow View → DTO → Controller → AutoMapper → Entity → Service → Repository → DbContext. This structure ensures that layers can be developed independently while maintaining flexibility for future extensions, such as adding likes, deletion, or editing functionalities for comments.



```
mentCreateDto.cs
rFikrimVar.Entity                    BirFikrimVar.Entity.DTOs.Co

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BirFikrimVar.Entity.DTOs
{
    5 başvuru
    public class CommentCreateDto
    {
        [Required(ErrorMessage =" Yorum Satırı Boş Olamaz"
        4 başvuru
        public string? Content { get; set; }
        1 başvuru
        public int UserId { get; set; }
        5 başvuru
        public int IdeaId { get; set; }


    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BirFikrimVar.Entity.ViewModels
{
    1 başvuru
    public class CommentItemViewModel
    {
        0 başvuru
        public int Id { get; set; }
        0 başvuru
        public string? Content { get; set; }
        0 başvuru
        public DateTime CreatedAt { get; set; }
        0 başvuru
        public string? UserFullName { get; set; }

        // like bilgileri
        0 başvuru
        public int LikeCount { get; set; }
        0 başvuru
        public bool HasLiked { get; set; }
```

```
tService.cs    CommentCreate...oValidator.cs    CommentManager.cs
mVar.Business                    BirFikrimVar.Business.Service

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using BirFikrimVar.Entity.Models;

namespace BirFikrimVar.Business.Services
{
    7 başvuru
    public interface ICommentService
    {
        2 başvuru
        Task AddCommentAsync(Comment comment);
        2 başvuru
        Task<List<Comment>> GetCommentsByIdeaIdAsync(int i
        4 başvuru
        Task<Comment?> GetByIdAsync(int id);
        2 başvuru
        Task UpdateAsync(Comment comment);
        2 başvuru
        Task DeleteAsync(int id);
```

```
                    BirFikrimVar.Business.Managers.CommentManager
{
    _commentRepository = commentRepository;
    _commentLikeRepo = commentLikeRepo;
}

2 başvuru
public async Task AddCommentAsync(Comment comment)
{
    await _commentRepository.AddAsync(comment);
}

2 başvuru
public async Task<List<Comment>> GetCommentsByIdeaIdAsync(int ideaId)
{
    return await _commentRepository.GetAllAsync(c => c.IdeaId == ideaId, inclu
}

4 başvuru
public async Task<Comment?> GetByIdAsync(int id)
=> await _commentRepository.GetByIdAsync(id);

2 başvuru
public async Task UpdateAsync(Comment comment)
{
    _commentRepository.Update(comment);
    await _commentRepository.SaveChangesAsync();
}
```

Kontrol Eden		Onay	
Yaprak No		Kısım	

Tarih	05/08/2025	Yapılan İş	

Day 21– Comment Validation (FluentValidation), User-Specific Comment Listing

In today's work, I implemented validation for user comments and developed a structure that allows each user to view only their own comments. During this process, I integrated FluentValidation, the Service Layer, Controller, ViewModel, and Razor View components to establish a complete architecture.

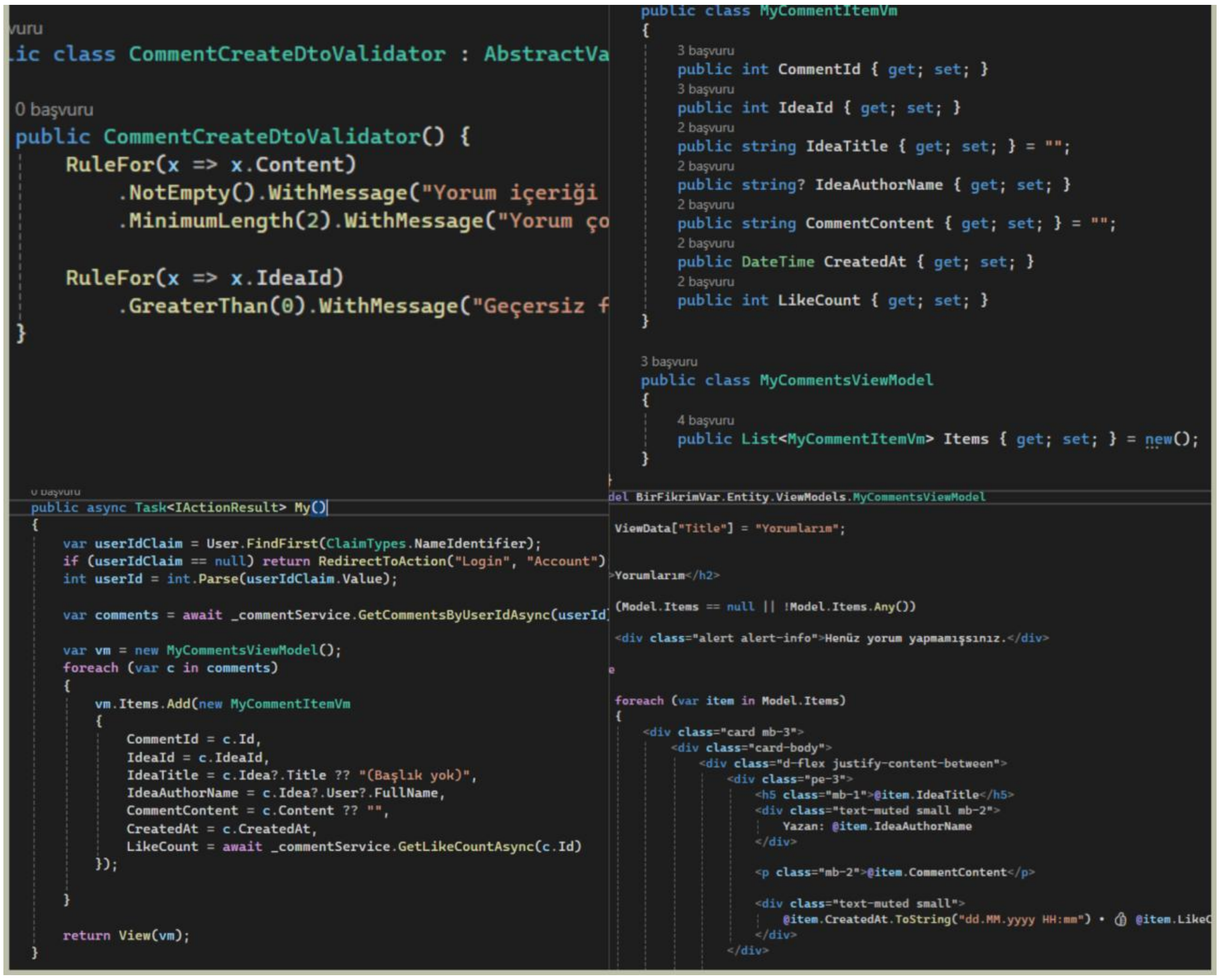
Firstly, a CommentCreateDtoValidator class was created using FluentValidation. Within this validator, rules were defined to ensure that the comment field cannot be empty, must not exceed 500 characters, and must always be associated with a specific idea (IdeaId). This way, when the form was submitted with invalid data, error messages were automatically displayed without requiring additional controller logic.

Next, a MyCommentsViewModel was designed to allow users to view their own comments. This ViewModel contained the comment content, the related idea title, and the creation date. The ICommentService interface was extended with the method GetCommentsByUserIdAsync. In its implementation within CommentManager, all comments were retrieved from the repository, filtered by the current user's ID, and then transformed into the ViewModel format.

A new MyComments action method was added to the CommentController, where the logged-in user's ID was retrieved from the session and used to fetch the relevant comments via the service layer. These comments were then passed to a Razor View, which displayed them in a structured card layout including the idea title, comment content, and creation date.

During testing, different users logged into the system and posted comments. Each user was able to see only their own comments, validation rules worked properly, and invalid inputs were immediately flagged with error messages.

In conclusion, the comment system was significantly enhanced by enabling user-specific comment listing and robust validation. This ensured that each user has access only to their own data, while maintaining compliance with layered architecture principles and improving overall user experience and security.



```
vuru
ic class CommentCreateDtoValidator : AbstractVa

0 başvuru
public CommentCreateDtoValidator() {
    RuleFor(x => x.Content)
        .NotEmpty().WithMessage("Yorum içeriği
        .MinimumLength(2).WithMessage("Yorum ço

    RuleFor(x => x.IdeaId)
        .GreaterThan(0).WithMessage("Geçersiz f
}
```

```
public class MyCommentItemVm
{
    3 başvuru
    public int CommentId { get; set; }
    3 başvuru
    public int IdeaId { get; set; }
    2 başvuru
    public string IdeaTitle { get; set; } = "";
    2 başvuru
    public string? IdeaAuthorName { get; set; }
    2 başvuru
    public string CommentContent { get; set; } = "";
    2 başvuru
    public DateTime CreatedAt { get; set; }
    2 başvuru
    public int LikeCount { get; set; }
}

3 başvuru
public class MyCommentsViewModel
{
    4 başvuru
    public List<MyCommentItemVm> Items { get; set; } = new();
}
```

```
0 başvuru
public async Task<IActionResult> My()
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier);
    if (userIdClaim == null) return RedirectToAction("Login", "Account")
    int userId = int.Parse(userIdClaim.Value);

    var comments = await _commentService.GetCommentsByUserIdAsync(userId

    var vm = new MyCommentsViewModel();
    foreach (var c in comments)
    {
        vm.Items.Add(new MyCommentItemVm
        {
            CommentId = c.Id,
            IdeaId = c.IdeaId,
            IdeaTitle = c.Idea?.Title ?? "(Başlık yok)",
            IdeaAuthorName = c.Idea?.User?.FullName,
            CommentContent = c.Content ?? "",
            CreatedAt = c.CreatedAt,
            LikeCount = await _commentService.GetLikeCountAsync(c.Id)
        });

    }

    return View(vm);
}
```

```
del BirFikrimVar.Entity.ViewModels.MyCommentsViewModel

ViewData["Title"] = "Yorumlarım";

>Yorumlarım</h2>

(Model.Items == null || !Model.Items.Any())

<div class="alert alert-info">Henüz yorum yapmamışsınız.</div>

e

foreach (var item in Model.Items)
{
    <div class="card mb-3">
        <div class="card-body">
            <div class="d-flex justify-content-between">
                <div class="pe-3">
                    <h5 class="mb-1">@item.IdeaTitle</h5>
                    <div class="text-muted small mb-2">
                        Yazan: @item.IdeaAuthorName
                    </div>

                    <p class="mb-2">@item.CommentContent</p>

                    <div class="text-muted small">
                        @item.CreatedAt.ToString("dd.MM.yyyy HH:mm") • 👍 @item.LikeC
                    </div>
                </div>
```

Kontrol Eden		Onay	
Yaprak No		Kısım	

Tarih	06/08/2025	Yapılan İş	

Day 22 – Comment Deletion, Authorization, and Secure User Interaction with Dynamic Buttons

In today's work, I implemented a secure comment deletion system that allows users to delete only their own comments. This required modifications across the service layer, controller, and Razor View to prevent users from tampering with or deleting other users' comments.

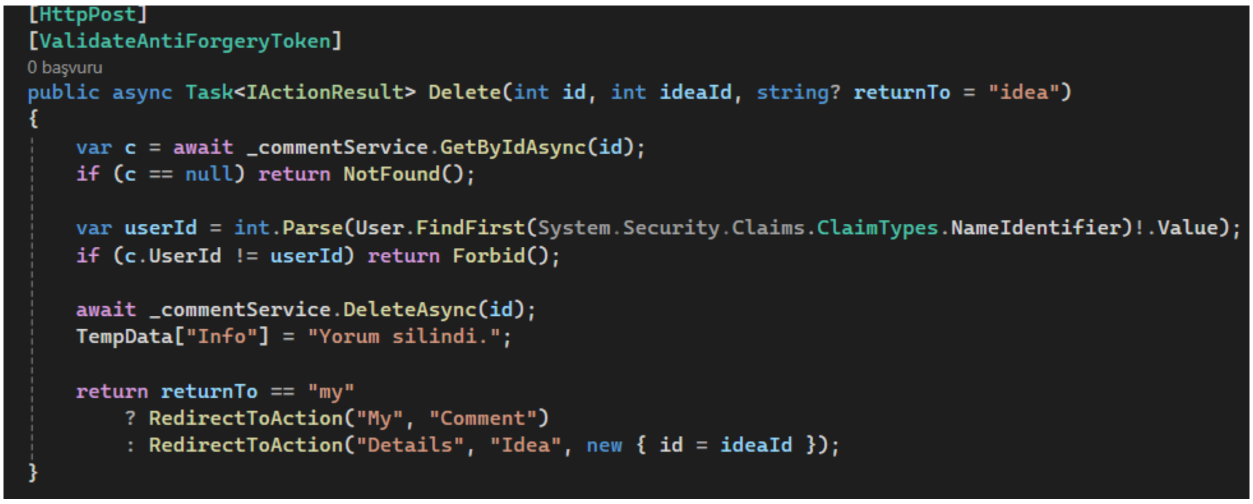
In the service layer, the method DeleteCommentAsync was defined to ensure that a comment could only be deleted by its rightful owner. This method retrieves all comments from the database and applies a dual filter using both commentId and userId. If a match is found, the comment is deleted; otherwise, no action is taken. This guarantees that users cannot remove comments that do not belong to them.

On the controller side, a Delete action method was added to the CommentController. Here, the logged-in user's ID was retrieved from the session and passed into the service layer. Once the deletion was complete, the user was redirected back to the MyComments page, maintaining a seamless user experience.

In the Razor View, the MyComments.cshtml file was enhanced with a delete button under each comment card. This button submits a POST request to the Delete action, sending the comment's ID through a hidden input field. Using POST requests and hidden fields prevents URL manipulation and strengthens the security of the process.

Testing confirmed that logged-in users could successfully delete their own comments, while attempts to delete another user's comments failed as expected. Additionally, the page automatically refreshed after deletion, improving overall usability.

In conclusion, today's implementation established a secure and user-friendly deletion mechanism, fully aligned with layered architecture principles. The system now ensures both data security and an improved user experience, marking an important step in the project's development.

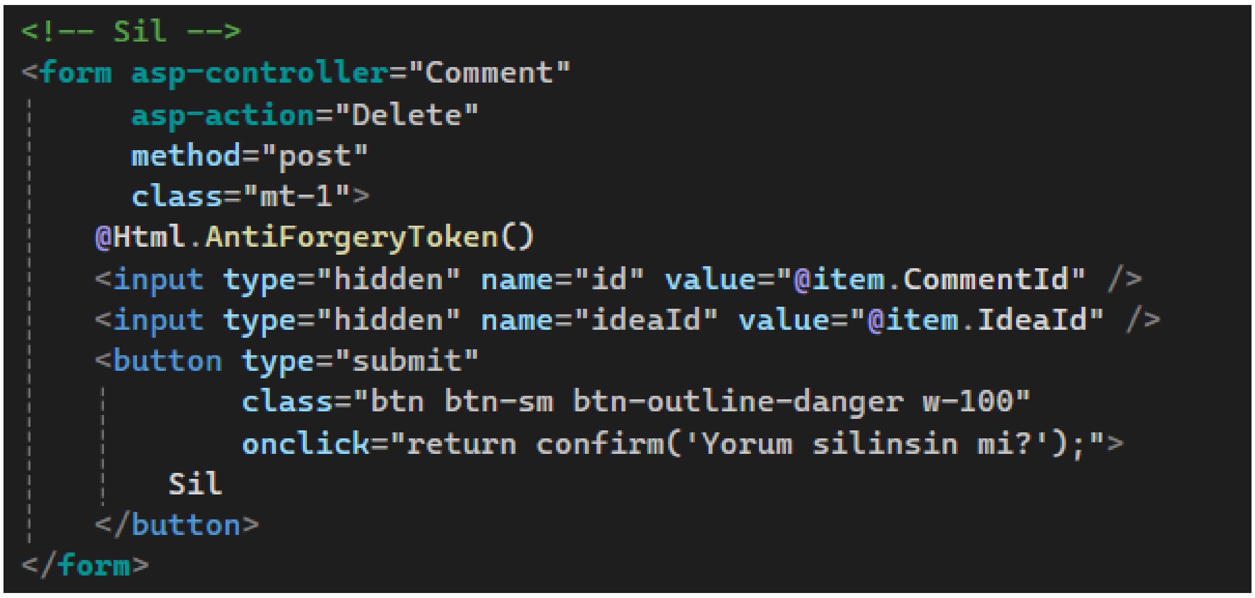


```
[HttpPost]
[ValidateAntiForgeryToken]
0 başvuru
public async Task<IActionResult> Delete(int id, int ideaId, string? returnTo = "idea")
{
    var c = await _commentService.GetByIdAsync(id);
    if (c == null) return NotFound();

    var userId = int.Parse(User.FindFirst(System.Security.Claims.ClaimTypes.NameIdentifier)!.Value);
    if (c.UserId != userId) return Forbid();

    await _commentService.DeleteAsync(id);
    TempData["Info"] = "Yorum silindi.";

    return returnTo == "my"
        ? RedirectToAction("My", "Comment")
        : RedirectToAction("Details", "Idea", new { id = ideaId });
}
```



```
<!-- Sil -->
<form asp-controller="Comment"
      asp-action="Delete"
      method="post"
      class="mt-1">
    @Html.AntiForgeryToken()
    <input type="hidden" name="id" value="@item.CommentId" />
    <input type="hidden" name="ideaId" value="@item.IdeaId" />
    <button type="submit"
            class="btn btn-sm btn-outline-danger w-100"
            onclick="return confirm('Yorum silinsin mi?');">
        Sil
    </button>
</form>
```

Kontrol Eden		Onay	
Yaprak No		Kısım	

Tarih	07/08/2025	Yapılan İş	

Day 23– Like System: LikeController, Many-to-Many Relationship, Security, and Total Like Counter

In today's work, I developed a like system for ideas. The goal was to ensure that each user could like an idea only once, display the total number of likes dynamically, and enforce security checks at both the database and service levels.

The first step was creating the Like model with UserId and IdeaId as a composite key. This structure prevented a user from liking the same idea more than once. Navigation properties (User, Idea) were also included to allow Entity Framework Core to recognize the relationships.
Subsequently, the AppDbContext was configured with modelBuilder.Entity<Like>().HasKey(x => new { x.UserId, x.IdeaId });, ensuring that the UserId + IdeaId combination remained unique in the database, thereby maintaining data integrity.

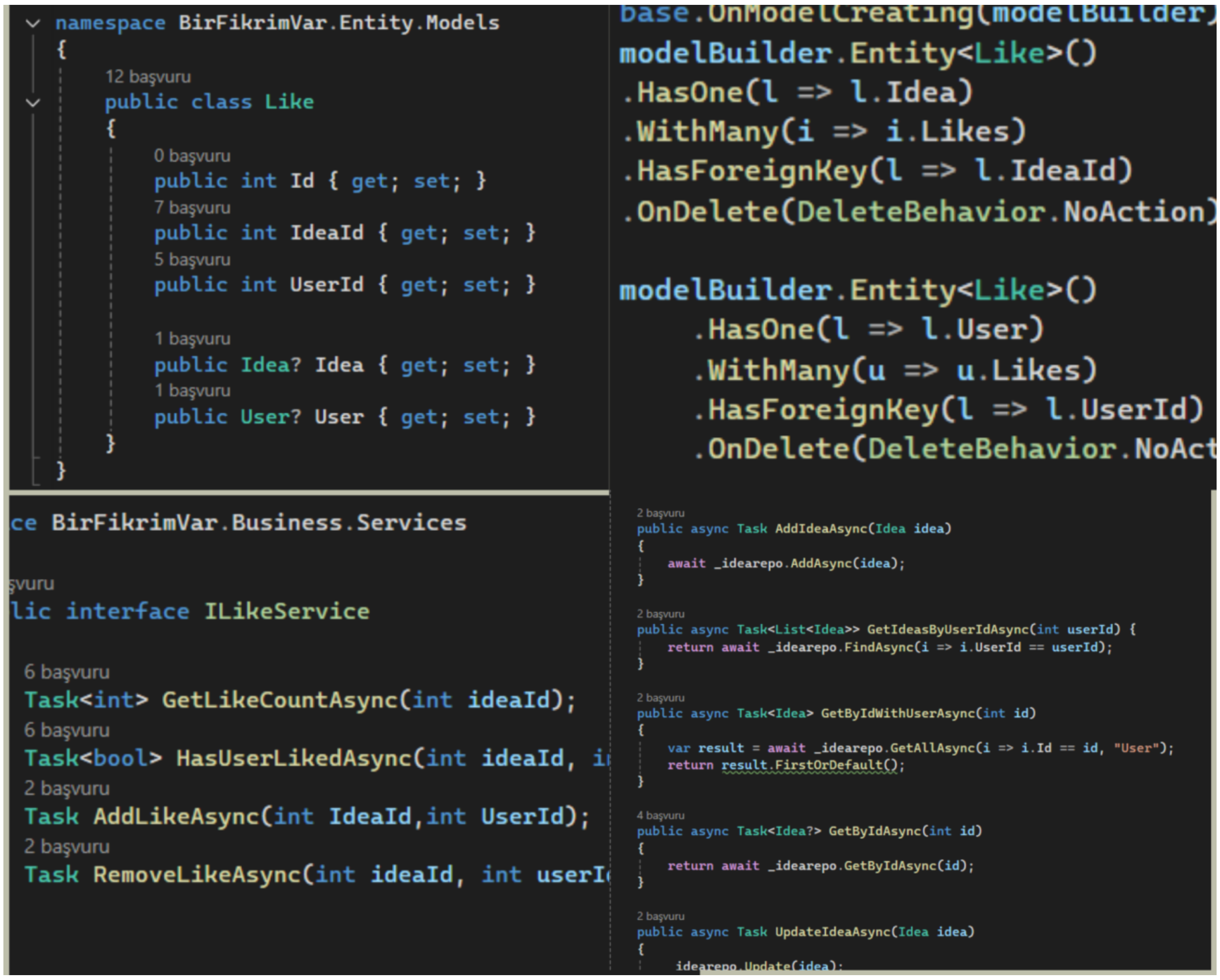
In the service layer, the ILikeService interface and its implementation LikeManager were created. These provided methods for adding a like (AddLikeAsync), retrieving the total number of likes (GetLikeCountAsync), and checking whether a user had already liked a specific idea (HasUserLikedAsync). By enforcing validation in both the database and the service layer, a double-layered security mechanism was achieved.

On the controller side, the LikeController was implemented. Through the Add action method, the user's ID was retrieved from the session, and a like was added to the corresponding idea. After completion, the user was redirected to the homepage, where the updated like count was displayed.

In the View layer, the Index.cshtml file was enhanced with a like button. This button sent a POST request to LikeController.Add and displayed the total like count dynamically for each idea. To support this, a LikeCount property was added to the IdeaCardViewModel, and the like count was fetched from the service in the HomeController before being passed to the View.

Testing confirmed that authenticated users were able to like ideas successfully, while repeated likes on the same idea were prevented. The total number of likes was displayed correctly for each idea, and the like button was hidden for unauthenticated users, ensuring stronger security.

In conclusion, today's implementation successfully delivered a secure, dynamic, and layered architecture–compliant like system for ideas.



```
namespace BirFikrimVar.Entity.Models
{
    12 başvuru
    public class Like
    {
        0 başvuru
        public int Id { get; set; }
        7 başvuru
        public int IdeaId { get; set; }
        5 başvuru
        public int UserId { get; set; }

        1 başvuru
        public Idea? Idea { get; set; }
        1 başvuru
        public User? User { get; set; }
    }
}
```

```
base.OnModelCreating(modelBuilder)
modelBuilder.Entity<Like>()
.HasOne(l => l.Idea)
.WithMany(i => i.Likes)
.HasForeignKey(l => l.IdeaId)
.OnDelete(DeleteBehavior.NoAction)

modelBuilder.Entity<Like>()
    .HasOne(l => l.User)
    .WithMany(u => u.Likes)
    .HasForeignKey(l => l.UserId)
    .OnDelete(DeleteBehavior.NoAct
```

```
ce BirFikrimVar.Business.Services

şvuru
lic interface ILikeService

 6 başvuru
 Task<int> GetLikeCountAsync(int ideaId);
 6 başvuru
 Task<bool> HasUserLikedAsync(int ideaId, i
 2 başvuru
 Task AddLikeAsync(int IdeaId,int UserId);
 2 başvuru
 Task RemoveLikeAsync(int ideaId, int userI
```

```
2 başvuru
public async Task AddIdeaAsync(Idea idea)
{
    await _idearepo.AddAsync(idea);
}

2 başvuru
public async Task<List<Idea>> GetIdeasByUserIdAsync(int userId) {
    return await _idearepo.FindAsync(i => i.UserId == userId);
}

2 başvuru
public async Task<Idea> GetByIdWithUserAsync(int id)
{
    var result = await _idearepo.GetAllAsync(i => i.Id == id, "User");
    return result.FirstOrDefault();
}

4 başvuru
public async Task<Idea?> GetByIdAsync(int id)
{
    return await _idearepo.GetByIdAsync(id);
}

2 başvuru
public async Task UpdateIdeaAsync(Idea idea)
{
     idearepo.Update(idea);
```

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	08/08/2025	Yapılan İş	

Day 24 – User Session Management, Authorize Usage, Layout Design, and Logout System

In today’s work, I focused on user session management. First, I integrated the Authorize attribute to restrict access for users who had not logged in. This ensured that only authenticated users could perform operations such as adding ideas, commenting, and liking.

After login, the user’s name was stored in the Session and dynamically displayed in the Layout file as “Welcome, [User Name]” on the navigation bar. Additionally, a “Logout” button was implemented, and for security reasons, it was configured to work through the POST method rather than GET.

The Logout process was defined inside the AccountController. This process terminates cookie-based authentication using SignOutAsync and clears session data via Session.Clear(). Once completed, the user is redirected back to the Login page.

To maintain a consistent design across the application, the _Layout.cshtml file was structured. The navigation bar, Bootstrap integration, and content rendering with @RenderBody() were managed within this file. As a result, repetitive code across different views was eliminated.

Finally, session and authentication services were configured in the Program.cs file. By adding app.UseAuthentication() and app.UseAuthorization() to the middleware pipeline, the Authorize attribute became fully functional.

The conducted tests confirmed that users who attempted to access restricted pages without logging in were redirected to the login page, authenticated users’ names were correctly displayed in the navigation bar, and session data was cleared successfully after logging out.

Through these implementations, user-specific operations were secured, dynamic session management was established, and a consistent interface structure was achieved across the project.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	11/08/2025	Yapılan İş	

Day 25 – Content Deletion Authorization, User Validation, and ViewModel Enhancements

In today's work, the content management system was strengthened from a security perspective. The main focus was ensuring that users could only delete their own ideas and comments. For this purpose, security layers were implemented both on the server side (controller level) and on the client side (Razor View).

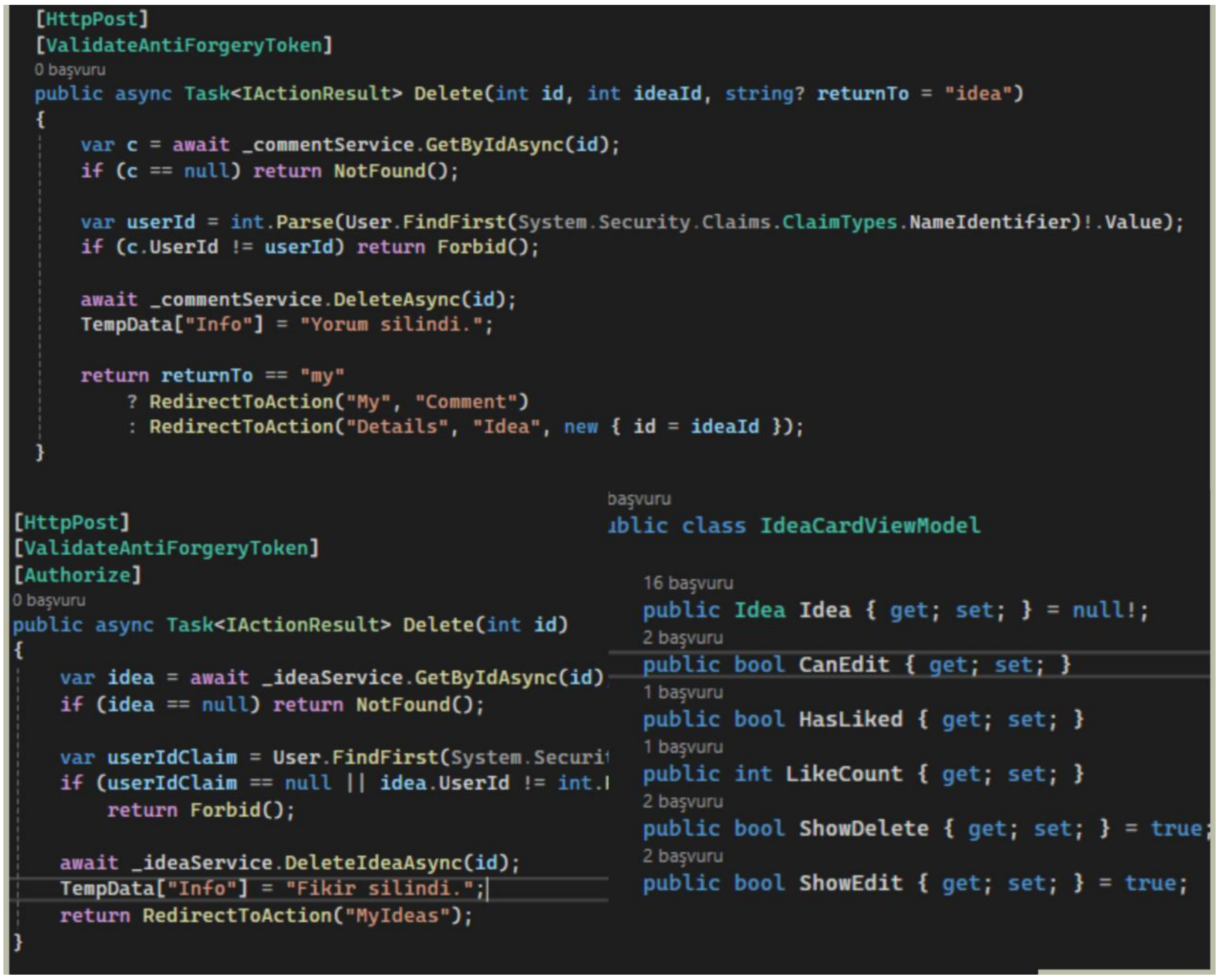
At the controller level, the user's identity was compared with the content owner's identity, and deletion was permitted only when both matched. As a result, when a user attempted to delete content belonging to someone else, the system returned an "Unauthorized access" response. This significantly improved the security standards of the application.

In parallel, the ViewModel structures were extended. For instance, a user identity field was added to the IdeaCardViewModel, enabling the visibility of action buttons to be managed dynamically on the Razor View. As a result, users only saw the "Delete" button for content they owned. This not only enhanced user experience but also introduced an additional client-side security layer.

The same logic was applied to comment management. A user identity field was also added to the CommentItemViewModel, ensuring that the "Delete" button appeared only for comments created by the logged-in user. This approach provided a consistent security model across both ideas and comments.

Another important aspect of the work was identifying and solving errors. In certain cases, the user identity field within the ViewModel returned null, which triggered a NullReferenceException. Upon investigation, the issue was traced to the failure of loading related user data from the database. The solution involved extending database queries with an Include statement, ensuring that the associated user data was eagerly loaded. This modification eliminated the null reference issue and stabilized the ViewModel population process.

In conclusion, the improvements made on both the backend and frontend strengthened content security and provided a dynamic interface adapted to the logged-in user.



```
[HttpPost]
[ValidateAntiForgeryToken]
0 başvuru
public async Task<IActionResult> Delete(int id, int ideaId, string? returnTo = "idea")
{
    var c = await _commentService.GetByIdAsync(id);
    if (c == null) return NotFound();

    var userId = int.Parse(User.FindFirst(System.Security.Claims.ClaimTypes.NameIdentifier)!.Value);
    if (c.UserId != userId) return Forbid();

    await _commentService.DeleteAsync(id);
    TempData["Info"] = "Yorum silindi.";

    return returnTo == "my"
        ? RedirectToAction("My", "Comment")
        : RedirectToAction("Details", "Idea", new { id = ideaId });
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize]
0 başvuru
public async Task<IActionResult> Delete(int id)
{
    var idea = await _ideaService.GetByIdAsync(id)
    if (idea == null) return NotFound();

    var userIdClaim = User.FindFirst(System.Securi
    if (userIdClaim == null || idea.UserId != int.
        return Forbid();

    await _ideaService.DeleteIdeaAsync(id);
    TempData["Info"] = "Fikir silindi.";
    return RedirectToAction("MyIdeas");
}
```

```
başvuru
ublic class IdeaCardViewModel

    16 başvuru
    public Idea Idea { get; set; } = null!;
    2 başvuru
    public bool CanEdit { get; set; }
    1 başvuru
    public bool HasLiked { get; set; }
    1 başvuru
    public int LikeCount { get; set; }
    2 başvuru
    public bool ShowDelete { get; set; } = true
    2 başvuru
    public bool ShowEdit { get; set; } = true;
```

Kontrol Eden		Onay	

Yaprak No			Kısım	

Tarih	12/08/2025		Yapılan İş	

Day 26 – Detail Page, Dynamic Routing, and Data Enrichment

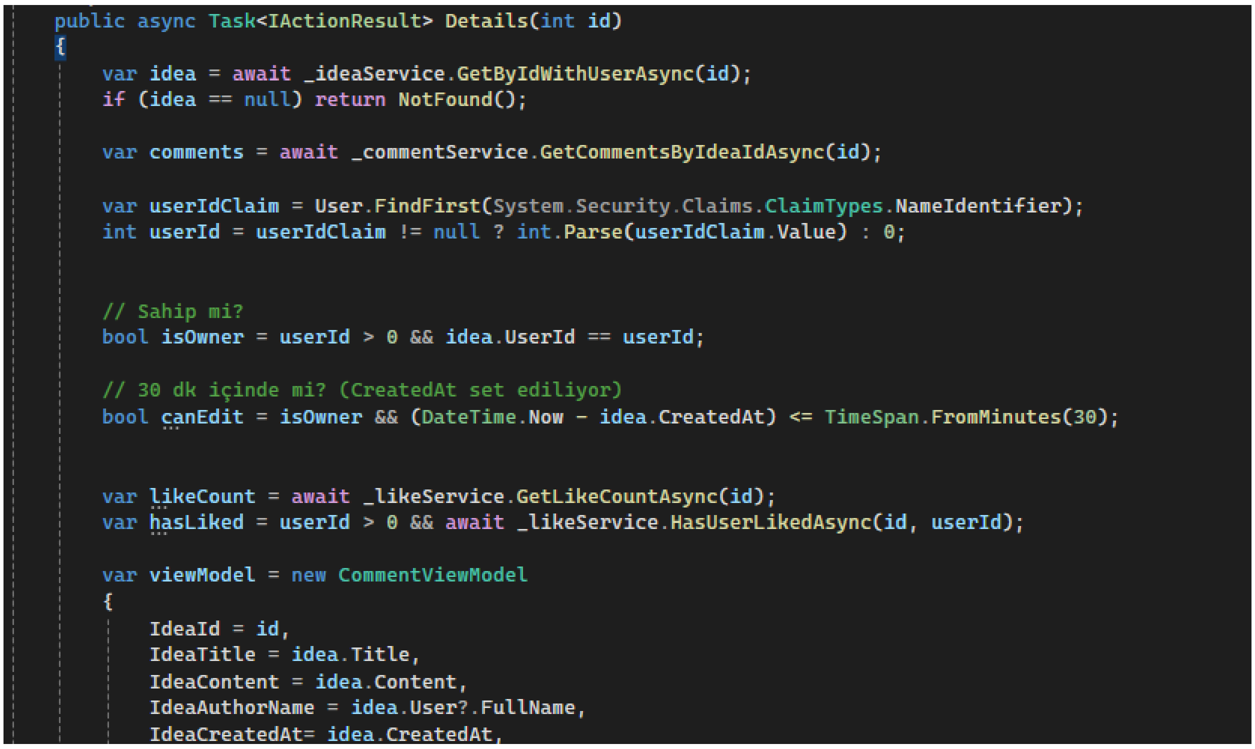
In today's work, a dedicated "detail page" was developed for ideas to improve user experience. A dynamic URL structure was created so that each idea could be accessed individually. By clicking on an idea title, the user was directed to an address in the format /Idea/Details/{id}, where all the information specific to that idea was displayed.

To ensure comprehensive data management, a dedicated IdeaDetailsViewModel was designed. This model consolidated multiple data points, including the idea's title, content, author, creation date, associated comments, and the total number of likes. By unifying data from multiple tables into a single structure, the ViewModel provided a clean and efficient way of passing information to the View.

On the frontend, the idea title and content were displayed prominently at the top of the page, while the number of likes and the list of comments were shown below. Each comment entry displayed its content, author, and creation date. This allowed users not only to view the idea itself but also to follow the discussion surrounding it within the same interface.
Additionally, on the home page, idea titles were updated to be clickable links that redirected users to the corresponding detail page. This improvement created a smoother navigation flow across the system.

During testing, it was verified that when a valid idea ID was entered, the detail page loaded all relevant data correctly, while an invalid ID triggered a "NotFound" response. These scenarios confirmed that the system was both functional and resilient against incorrect input.

In conclusion, the addition of the detail page allowed users to evaluate ideas more comprehensively. Furthermore, the integration of dynamic routing and consolidated data handling via the ViewModel improved the system architecture, making it more maintainable and sustainable.



```
public async Task<IActionResult> Details(int id)
{
    var idea = await _ideaService.GetByIdWithUserAsync(id);
    if (idea == null) return NotFound();

    var comments = await _commentService.GetCommentsByIdeaIdAsync(id);

    var userIdClaim = User.FindFirst(System.Security.Claims.ClaimTypes.NameIdentifier);
    int userId = userIdClaim != null ? int.Parse(userIdClaim.Value) : 0;


    // Sahip mi?
    bool isOwner = userId > 0 && idea.UserId == userId;

    // 30 dk içinde mi? (CreatedAt set ediliyor)
    bool canEdit = isOwner && (DateTime.Now - idea.CreatedAt) <= TimeSpan.FromMinutes(30);


    var likeCount = await _likeService.GetLikeCountAsync(id);
    var hasLiked = userId > 0 && await _likeService.HasUserLikedAsync(id, userId);

    var viewModel = new CommentViewModel
    {
        IdeaId = id,
        IdeaTitle = idea.Title,
        IdeaContent = idea.Content,
        IdeaAuthorName = idea.User?.FullName,
        IdeaCreatedAt= idea.CreatedAt,
```

Kontrol Eden			Onay	

Yaprak No		Kısım	

Tarih	13/08/2025	Yapılan İş	

Day 27 – Search System in Ideas: LINQ Filtering, ViewBag Feedback, and Razor Form Integration

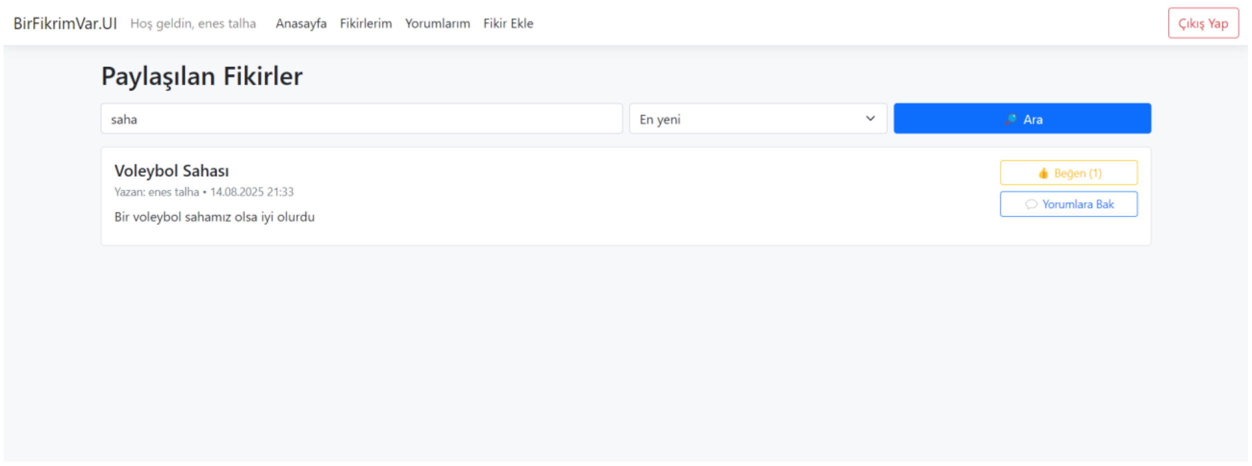
In today’s work, a search functionality was integrated into the idea management system to enhance accessibility and user experience. The search system was implemented through a combination of a Razor form, LINQ-based filtering on the controller side, and feedback mechanisms on the view side.

A search form was added to the home page (Index.cshtml). The form utilized the GET method, ensuring that search queries were reflected directly in the URL (e.g., /Home/Index?search=robot). This approach provided benefits such as shareable search results and automatic saving of queries in the browser history.

On the controller side, the Index action was updated to accept a string search parameter. If the parameter was not null or empty, the LINQ query filtered ideas by checking whether their titles contained the search term. To ensure case-insensitivity, both the search text and the titles were converted to lowercase before comparison. The filtered results, along with feedback data such as the search term and result count, were passed to the view using ViewBag.

On the frontend, an alert box was dynamically displayed whenever a search was performed, showing both the searched keyword and the total number of matching results. For instance, a query for *“artificial intelligence”* returned an informational message: *“2 results found for ‘artificial intelligence’.”* Furthermore, the search input retained the previously entered text, improving usability.

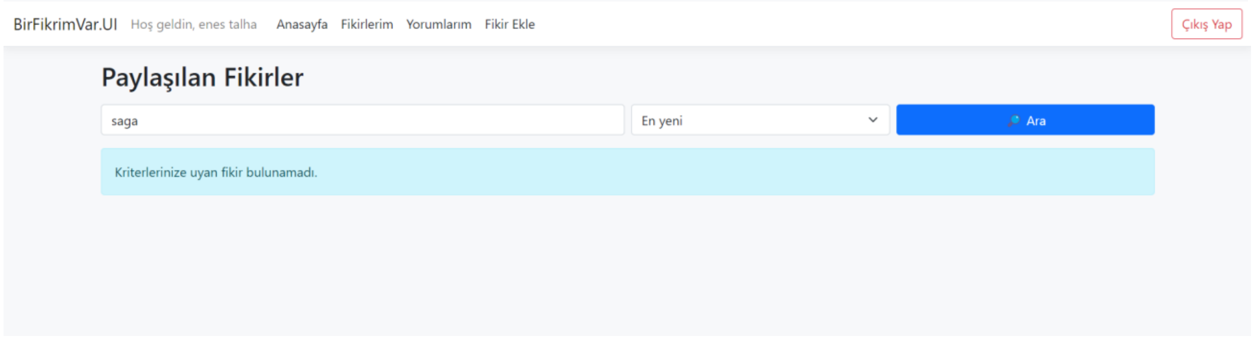
The IdeaCardViewModel already contained the necessary fields (ID, title, content, user information, like count, and owner ID), so no structural modifications were required. Each result continued to be rendered through the existing card design, ensuring consistency.
Extensive testing was carried out to validate the implementation. Searches with varying capitalization (e.g., “ROBOT” vs. “robot”) consistently returned the same results, confirming that the system was properly case-insensitive. Empty queries displayed all ideas, while specific keywords correctly filtered the results.
In conclusion, the integration of the search feature successfully improved the system’s functionality. It provided a user-friendly, URL-based filtering mechanism that enhanced accessibility, improved navigation, and laid the groundwork for potential future enhancements, such as content-based searches, sorting by likes, or category-based filtering.



Paylaşılan Fikirler

saha

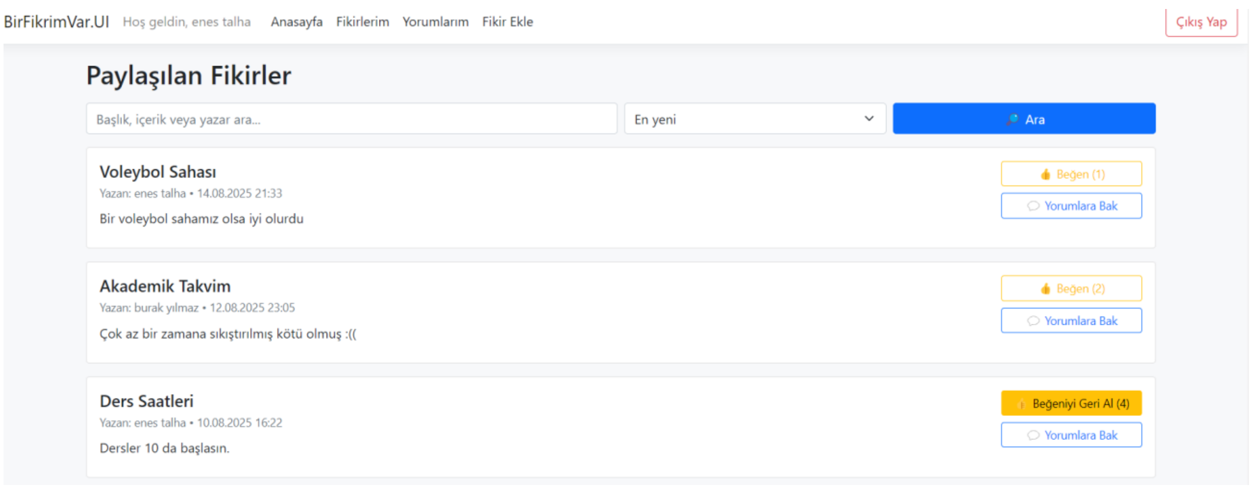
Voyleybol Sahası
Yazan: enes talha • 14.08.2025 21:33
Bir voleybol sahamız olsa iyi olurdu



Paylaşılan Fikirler

saga

Kriterlerinize uyan fikir bulunamadı.



Paylaşılan Fikirler

Voleybol Sahası
Yazan: enes talha • 14.08.2025 21:33
Bir voleybol sahamız olsa iyi olurdu

Akademik Takvim
Yazan: burak yılmaz • 12.08.2025 23:05
Çok az bir zamana sıkıştırılmış kötü olmuş :((

Ders Saatleri
Yazan: enes talha • 10.08.2025 16:22
Dersler 10 da başlasın.

Kontrol Eden		Onay	
Yaprak No		Kısım	

Tarih	14/08/2025	Yapılan İş	

Day 28 – Comment System: Comment Entity, ViewModel, Service, and Add Operation

In today’s work, I extended the comment system that I initially developed on Day 20. While the first version focused mainly on creating comments through the Controller, DTO, and ViewModel, today’s work expanded the system by adding deeper Entity relationships, repository support, and enriched data presentation. This extension ensured better maintainability and a more scalable architecture in accordance with layered design principles.

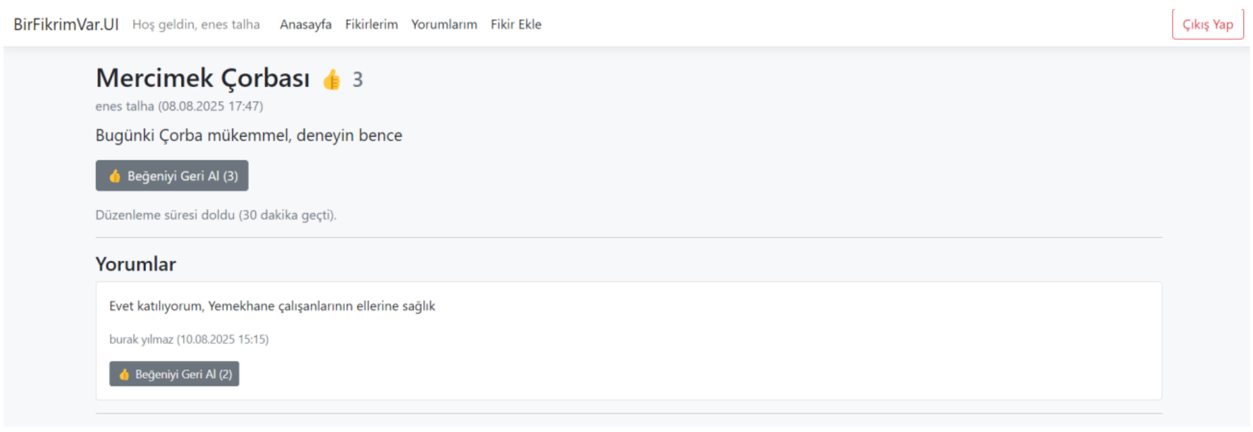
Firstly, the Comment Entity was defined to store the content of the comment (Content), its creation time (CreatedAt), the user who posted it (UserId), and the related idea (IdeaId). Relationships were established through navigation properties linking comments to both users and ideas.

Subsequently, CommentCreateDto was introduced to model the data received from users, including the comment text, the user ID, and the related idea ID. The UserId field is not displayed in the View but is retrieved from the session within the Controller.

For database operations, ICommentRepository and CommentRepository were implemented. Within these repositories, the method GetCommentsByIdeaIdAsync was developed to fetch comments related to a specific idea and to order them in reverse chronological order so that the most recent comments appear first.

In the business logic layer, ICommentService and CommentService were created to facilitate interaction with the repository. AutoMapper was utilized to transform DTOs into Entities, and the CreatedAt property was automatically assigned during the addition of comments.
On the user interface side, the CommentController contained an Add method to manage the process of submitting comments. The user identity was extracted via ClaimTypes.NameIdentifier and injected into the DTO. After submission, the user was redirected back to the relevant idea detail page.

In the Idea Detail Page (Detail.cshtml), existing comments were displayed, showing the author’s name, timestamp, and content. Furthermore, a submission form was provided, ensuring a user-friendly interaction experience.
In conclusion, the comment system enables users to post feedback under ideas, with data stored through user and idea associations. AutoMapper simplified DTO–Entity mapping, and the layered architecture improved maintainability.



Mercimek Çorbası 👍 3

enes talha (08.08.2025 17:47)

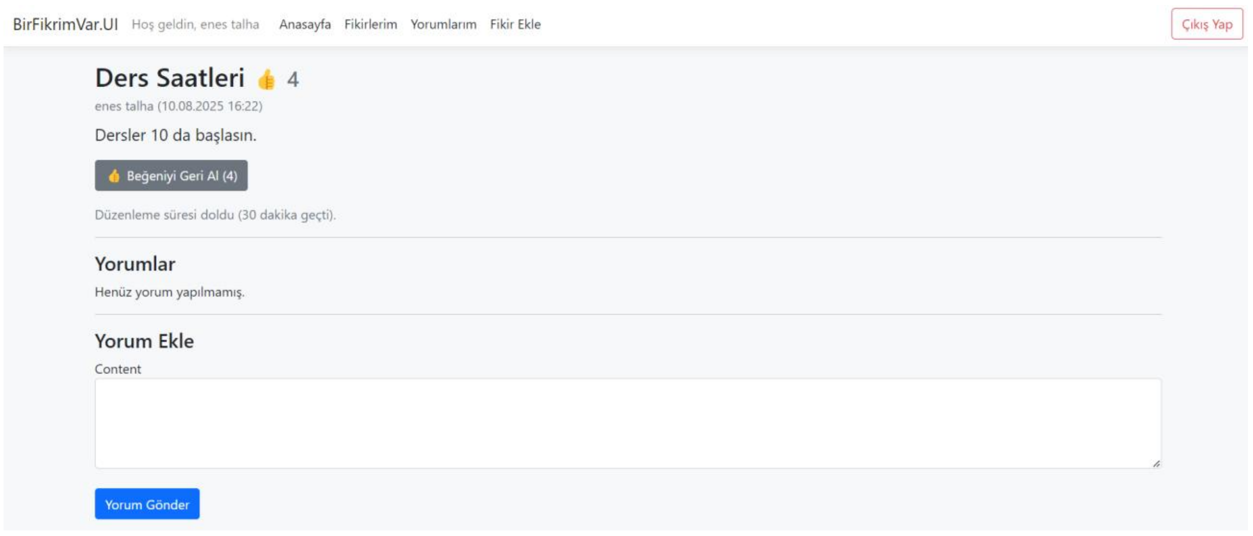
Bugünki Çorba mükemmel, deneyin bence

Düzenleme süresi doldu (30 dakika geçti).

Yorumlar

Evet katılıyorum, Yemekhane çalışanlarının ellerine sağlık

burak yılmaz (10.08.2025 15:15)



Ders Saatleri 👍 4

enes talha (10.08.2025 16:22)

Dersler 10 da başlasın.

Düzenleme süresi doldu (30 dakika geçti).

Yorumlar

Henüz yorum yapılmamış.

Yorum Ekle

Content

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	15/08/2025	Yapılan İş	

Day 29 – Code Refactoring and General Review

Today, instead of adding new features, I mainly focused on reviewing and refactoring the existing structure of the project. First, I checked whether the classes created in the Entity layer, as well as the DTOs and ViewModels, were used consistently throughout the project. I verified that the getters and setters functioned correctly and ensured that there was no unnecessary code duplication.

In the DataAccess layer, I confirmed that the Repository structure was being properly applied and that database operations were invoked through the services. Since there were no redundant methods, no additional modifications were necessary. However, I also noticed that the infrastructure was suitable for implementing more complex queries in the future if needed.

On the UI side, I tested whether the Controllers were performing their duties correctly and working in harmony with the Views. For example, I observed that the flows such as adding an idea, listing ideas, and displaying idea details through the IdeaController were functioning without errors.

Overall, today’s work focused on reviewing the code structure, evaluating architectural consistency, and checking for any missing or faulty components. As a result, the project has become more organized, maintainable, and sustainable.

Kontrol Eden		Onay	

Yaprak No		Kısım	

Tarih	18/08/2025	Yapılan İş	

Day 30– Final Testing, UI Enhancements, and Delivery Preparation

On the final day, all project modules were systematically tested to assess the overall functionality and stability of the system. The tests confirmed that the user registration and login processes operated correctly with validation mechanisms, while idea creation and listing modules successfully recorded data with accurate user associations. Furthermore, the idea detail page displayed comments and likes without any inconsistencies.

In terms of user interface, several improvements were implemented to enhance accessibility and readability. The comment box was enlarged, title and description sections were styled with larger fonts, and the login and registration pages were optimized for mobile responsiveness. Additionally, the date format of comments was updated to improve clarity for end users.

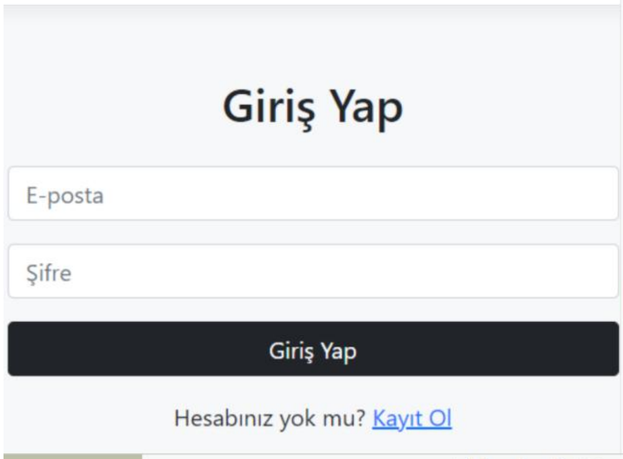
The AutoMapper configuration was thoroughly tested, verifying that DTO → Entity conversions functioned accurately. Through the MappingProfile, data was transferred securely and consistently across different layers of the application.

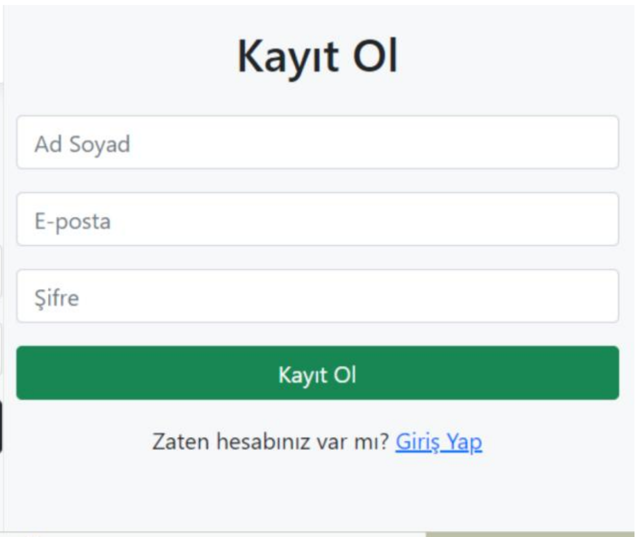
During the testing phase, several minor issues were detected and resolved. For instance, the missing redirection after user registration was addressed by adding a RedirectToAction, the comment ordering issue was corrected using OrderByDescending, and structural errors in HTML were fixed by completing missing <div> tags.

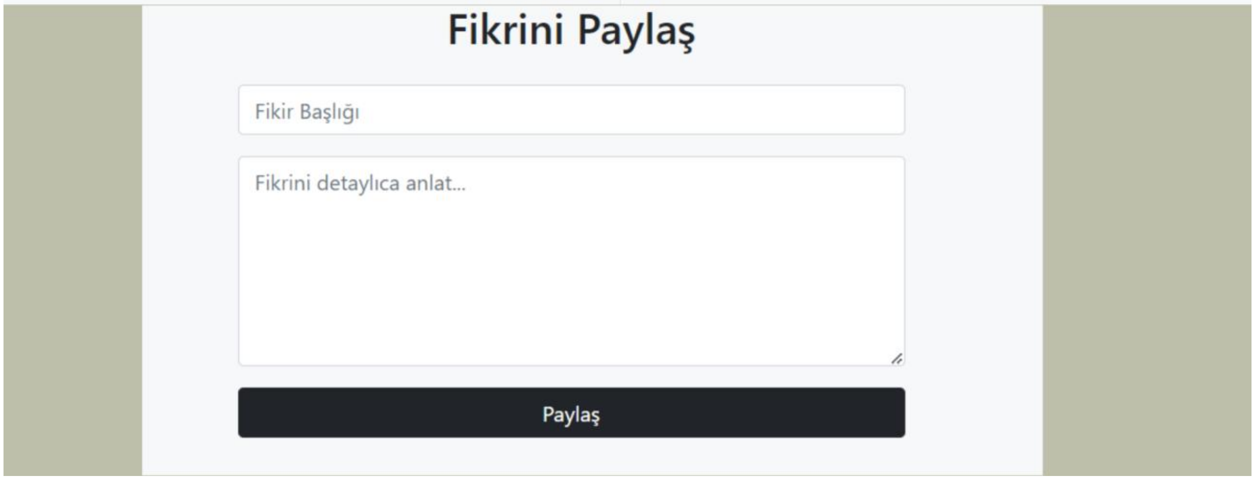
Finally, the project's folder structure was organized in accordance with layered architecture principles, preparing the system for delivery.

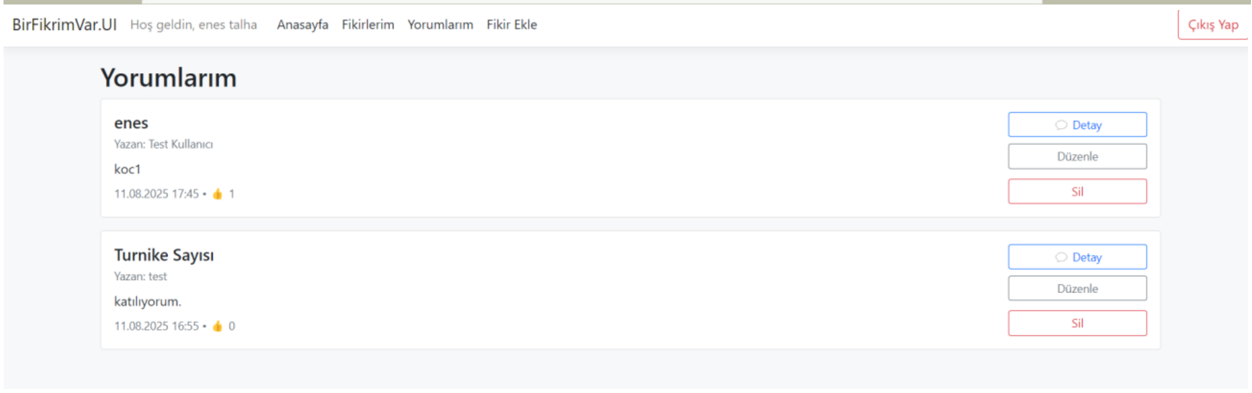
Overall Evaluation:

Over the course of 30 days, valuable experience was gained in developing a professional .NET MVC project using layered architecture. Tools such as AutoMapper, FluentValidation, and Entity Framework were effectively applied, while debugging and testing processes enhanced an analytical perspective toward software development. As a result, the project reached a deliverable, sustainable, and extensible state.





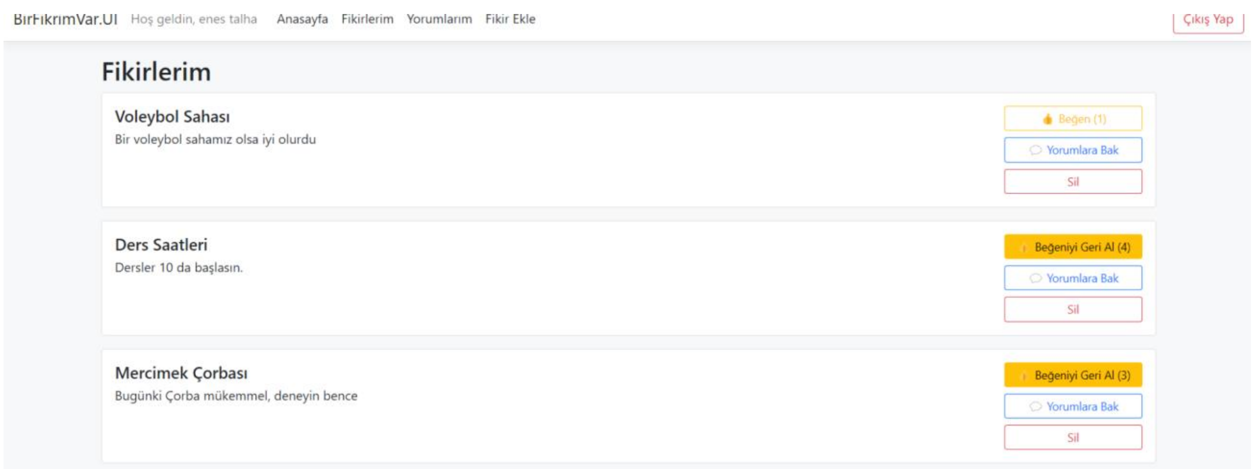




Yorumlarım

enes
Yazan: Test Kullanıcı
koc1
11.08.2025 17:45 • 👍 1

Turnike Sayısı
Yazan: test
katılıyorum.
11.08.2025 16:55 • 👍 0

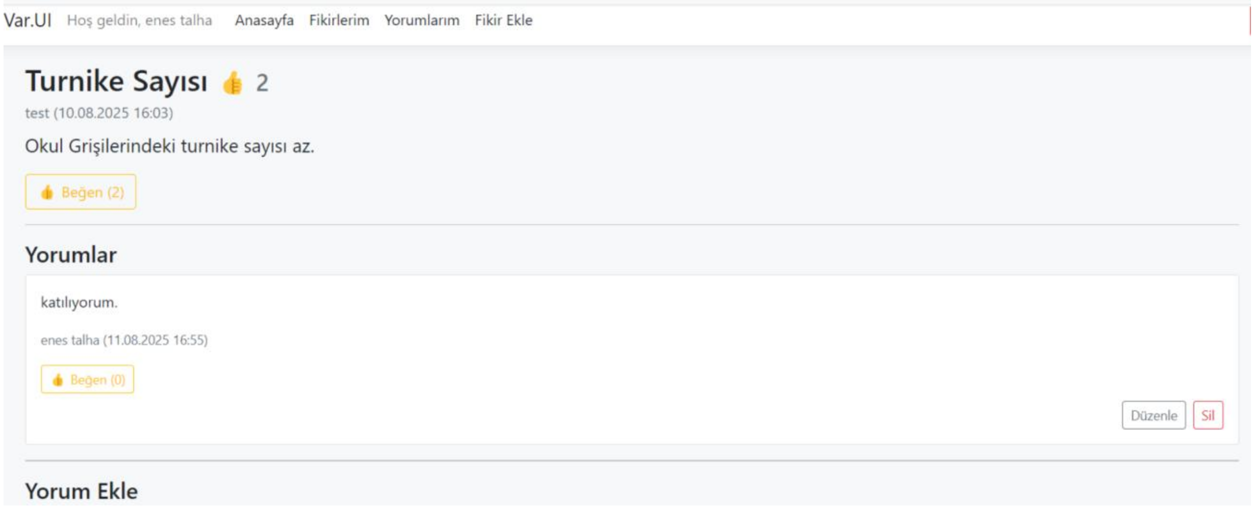


Fikirlerim

Voleybol Sahası
Bir voleybol sahamız olsa iyi olurdu

Ders Saatleri
Dersler 10 da başlasın.

Mercimek Çorbası
Bugünki Çorba mükemmel, deneyin bence



Turnike Sayısı 2
test (10.08.2025 16:03)
Okul Grişilerindeki turnike sayısı az.

Yorumlar

katılıyorum.
enes talha (11.08.2025 16:55)

Yorum Ekle

Kontrol Eden		Onay	