



Bursa Uludağ Üniversitesi

Mühendislik Fakültesi - Bilgisayar
Mühendisliği

BMB3016 - Görsel Programlama Proje
Ödevi

Bilmece Uygulaması (Quiz-App)

032090023 Altuğ AKINCI

032090003 Sefa ÖNDER

032090046 Karahan TÜRKER

032090055 İbrahim YEGEN

032090042 Ahmet Serhat İLGİN

İçindekiler

1.Özet.....	3
2. Database ve Entity Framework (Code-First) Yaklaşımı.....	4
2.1 Model.....	4
2.2 Servisler.....	9
2.3 Sessions (Oturumlar).....	19
2.3.1 UserSession.....	19
2.3.2 RoomSession.....	19
2.3.3 GameSession.....	20
2.4 Helper.....	21
2.5 PanelHandler.....	22
2.6 ThemeHandler.....	23
3. Formlar.....	25
3..1 Login.cs (Form).....	25
3.2 MainMenu.cs (Form).....	30
3.3 LobbyGame.....	40
4. SON SÖZ.....	48

1. Özet

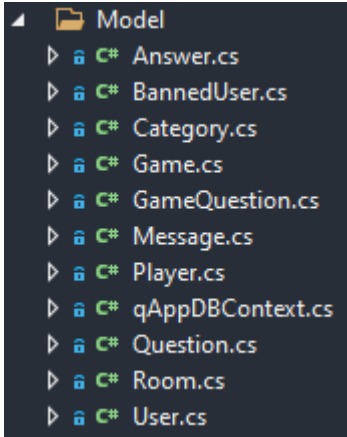
Bilgi yarışması uygulaması, Öğr. Gör. Koray AKİ hocamızın Görsel Programlama dersi içerisinde verdiği proje doğrultusunda, C# Winform ve Entity Framework kullanılarak tasarlanmıştır. Modern görünümlü arayüzü sayesinde kullanıcılara kolay ve eğlenceli bir bilgi yarışması deneyimi sunar. Şu anda belirli bir süre için [bu linkten](#) oyunu indirip siz de arkadaşlarınız ile projeyi deneyimleyebilirsiniz.

Projeyi tüm detayları ile anlattığımız bu rapor, umarız ki uygulama geliştiricilere bir rehber ve ilham olur.

2. Database ve Entity Framework (Code-First) Yaklaşımı

Programın veritabanı ile kuracağı bağlantıların daha kolay yönetilebilmesi için Entity Framework (Code-First) yaklaşımını kullandık. Önce ihtiyaçlarımızı belirledik ve buna göre modellerimizi oluşturduk.

2.1 Model



a. Answer.cs

Oyuncuların verdiği cevapların bir tablo şeklinde depolanması için oluşturulan sınıfımız.

```
public partial class Answer
{
    2 references
    public Guid Id { get; set; }

    [Required]
    1 reference
    public string AnswerText { get; set; }

    12 references
    public int GainedXp { get; set; }

    14 references
    public Guid? UserId { get; set; }

    3 references
    public Guid? QuestionId { get; set; }

    12 references
    public Guid? GameId { get; set; }

    0 references
    public virtual Game Game { get; set; }

    5 references
    public virtual Question Question { get; set; }

    0 references
    public virtual User User { get; set; }
}
```

+ **ID:Guid** Kaydedilen cevabın ID'si.
+ **AnswerText:string** Cevabın içeriği
+ **GainedXP:int** Cevaptan kazanılan XP
+ **UserId:Guid?** Cevabın hangi kullanıcı tarafından verildiği
+ **QuestionId:Guid?** Sorunun ID'si.
+ **GameId:Guid?** Oyunun ID'si.
+ **Game:Game** Cevabın hangi oyunda verildiği
+ **Question:Question** Cevabın hangi soruda verildiği
+ **User:User** Cevabı veren kullanıcı nesnesini tutan alan.

b. BannedUser.cs

```
public partial class BannedUser
{
    1 reference
    public Guid Id { get; set; }

    2 references
    public Guid? UserId { get; set; }

    2 references
    public Guid? RoomId { get; set; }

    0 references
    public virtual Room Room { get; set; }

    0 references
    public virtual User User { get; set; }
}
```

- + **Id:Guid** Atılmış olan ban olayının ID'si
- + **UserId:Guid?** Ban atılan kullanıcının ID'si
- + **RoomId:Guid?** Banın atıldığı odanın ID'si
- + **Room:Room** Banın atıldığı odanın nesnesi
- + **User:User** Banın atıldığı kullanıcının nesne olarak tutulduğu alan.

c. Category.cs

```
public partial class Category
{
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
    0 references
    public Category()
    {
        Questions = new HashSet<Question>();
    }

    6 references
    public Guid Id { get; set; }

    5 references
    public int Index { get; set; }

    [Required]
    [StringLength(255)]
    5 references
    public string Name { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
    1 reference
    public virtual ICollection<Question> Questions { get; set; }
}
```

- + **Id:Guid** Kategorinin ID'si.
- + **Index:int** Kategorinin Index numarası. Böylece kategoriye daha rahat erişim sağlanıyor.
- + **Name:string** Kategorinin ismi.
- + **Questions:ICollection<Question>** Soruların soru listesi olarak tutulduğu alan.

d. Game.cs

```
public partial class Game
{
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
    2 references
    public Game()
    {
        Answers = new HashSet<Answer>();
        GameQuestions = new HashSet<GameQuestion>();
        Rooms = new HashSet<Room>();
    }

    9 references
    public Guid Id { get; set; }

    2 references
    public DateTime StartTime { get; set; }

    3 references
    public DateTime EndTime { get; set; }

    3 references
    public Guid? RoomId { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
    1 reference
    public virtual ICollection<Answer> Answers { get; set; }

    1 reference
    public virtual Room Room { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
    2 references
    public virtual ICollection<GameQuestion> GameQuestions { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
    2 references
    public virtual ICollection<Room> Rooms { get; set; }
}
```

+ **ID:Guid** Oyunun ID'si

+ **StartTime:DateTime** Oyunun başlangıç saatini tutan alan.

+ **EndTime:DateTime** Oyunun bitiş saatini tutan alan.

+ **RoomId:Guid?** Oyunun oynandığı odanın ID'si

+ **Answers:ICollection<Answer>** Oyunda verilen cevapların listesini tutan alan.

+ **Room:Room** Odayı nesne olarak tutan alan.

+ **GameQuestions:ICollection<GameQuestion>** Oyundaki 10 soruluk soru listesini tutan alan.

+ **Rooms:ICollection<Room>** Odaları tutan alan.

e. GameQuestion.cs

```
public partial class GameQuestion
{
    public Guid Id { get; set; }

    public Guid? GameId { get; set; }

    public Guid? QuestionId { get; set; }

    public virtual Game Game { get; set; }

    public virtual Question Question { get; set; }
}
```

+ **ID:Guid** Oyundaki aktif sorunun ID'si

+ **GameId:Guid?** Oyunun ID'sini tutan alan.

+ **QuestionId:Guid?** Sorunun ID'sini tutan alan.

+ **Game:Game** Oyunu nesne olarak tutan alan.

+ **Question:Question** Soruyu nesne olarak tutan alan.

f. Message.cs

```
public partial class Message
{
    public Guid Id { get; set; }

    [Required]
    [StringLength(255)]
    public string MessageText { get; set; }

    public DateTime SentTime { get; set; }

    public Guid? UserId { get; set; }

    public Guid? RoomId { get; set; }

    public virtual Room Room { get; set; }

    public virtual User User { get; set; }
}
```

- + **ID:Guid** Mesajın ID'si
- + **MessageText:string** Mesajın içeriği
- + **SentTime:DateTime** Mesajın gönderildiği saat
- + **UserId:Guid?** Mesajı gönderen kullanıcının ID'si
- + **RoomId:Guid?** Mesajın gönderildiği odanın ID'si
- + **Room:Room** Mesajın odasını nesne olarak tutan alan.
- + **User:User** Mesajın kullanıcıyı nesne olarak tutan alan.

g.Player.cs

- + **Id:Guid** Oyuncunun ID'si
- + **RoomId:Guid?** Oyuncunun odasını tutan alan
- + **UserId:Guid?** Oyuncunun hangi kullanıcı olduğunu tutan alan
- + **Room:Room** Oyuncunun odasını nesne olarak tutan alan.
- + **User:User** Oyuncunun kullanıcıyı nesne olarak tutan alan.

```
public partial class Player
{
    2 references
    public Guid Id { get; set; }

    10 references
    public Guid? RoomId { get; set; }

    9 references
    public Guid? UserId { get; set; }

    0 references
    public virtual Room Room { get; set; }

    1 reference
    public virtual User User { get; set; }
}
```

h.Question.cs

- + **ID:Guid** Sorunun ID'sini tutan alan.
- + **QuestionText:string** Sorunun içeriği
- + **OptionsTest:string** Cevapların bir stringde tutulduğu alan.
- + **CorrectAnswerIndex:int** Doğru cevabın hangi indexte olduğu.
- + **CategoryId:Guid?** Sorunun hangi kategoride olduğu
- + **Category:Category** Sorunun kategorisini nesne olarak tutan alan.

```
public partial class Question
{
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "
    public Question()
    {
        Answers = new HashSet<Answer>();
        GameQuestions = new HashSet<GameQuestion>();
    }

    public Guid Id { get; set; }

    [Required]
    public string QuestionText { get; set; }

    [Required]
    public string OptionsText { get; set; }
    [Required]
    public int CorrectAnswerIndex { get; set; }

    11 references
    public Guid? CategoryId { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "
    1 reference
    public virtual ICollection<Answer> Answers { get; set; }

    4 references
    public virtual Category Category { get; set; }

    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "
    1 reference
    public virtual ICollection<GameQuestion> GameQuestions { get; set; }
}
```

i.Room.cs

Yeni oda oluşturulacağı zaman constructor'a gerekli bilgiler gönderilir.

- + **Id:Guid** Odanın Id'sini tutan alan.
- + **Name:string** Odanın ismini tutan alan.
- + **Password:string** Odanın şifresini tutan alan.
- + **Code:string** Odanın katılım kodunu tutan alan.
- + **CurrentGameId:Guid?** Odanın aktif oyununun ID'si
- + **AdminId:Guid?** Oyununun adminin ID'si

```

public partial class Room
{
    [System.Diagnostics.CodeAnalysis.SuppressMess

    public Room()
    {
        BannedUsers = new HashSet<BannedUser>();
        Games = new HashSet<Game>();
        Messages = new HashSet<Message>();
        Players = new HashSet<Player>();
    }

    public Guid Id { get; set; }

    [Required]
    [StringLength(255)]
    public string Name { get; set; }

    [Required]
    [StringLength(255)]
    public string Password { get; set; }

    [Required]
    [StringLength(255)]
    public string Code { get; set; }

    public Guid? CurrentGameId { get; set; }

    public Guid? AdminId { get; set; }
}

```

j.User.cs

```

public partial class User
{
    [System.Diagnostics.CodeAnalysis.SuppressMess

    public User()
    {
        Answers = new HashSet<Answer>();
        BannedUsers = new HashSet<BannedUser>();
        Messages = new HashSet<Message>();
        Players = new HashSet<Player>();
        Rooms = new HashSet<Room>();
    }

    public Guid Id { get; set; }

    [Required]
    [StringLength(255)]
    public string UserName { get; set; }

    [StringLength(255)]
    public string Email { get; set; }

    [Required]
    public string Password { get; set; }

    [Required]
    public string Salt { get; set; }

    public int Level { get; set; }

    public int Xp { get; set; }
}

```

Kullanıcı oluşturulurken çağırılan constructor.

- + **Id:Guid** Kullanıcının ID'si
- + **UserName:string** Kullanıcı adını tutan alan.
- + **Email:string** Kullanıcının mailini tutan alan.
- + **Password:string** Kullanıcının şifresini tutan alan.
- + **Salt:string** Şifre güvenliğini sağlayan alan.
- + **Level:int** Kullanıcının levelini tutan alan.
- + **Xp:int** Kullanıcının XP değerini tutan alan.

2.2 Servisler

a. GameService

StartGame

```
public static async Task<Game> StartGame(Guid roomId, List<Category> categories, DateTime startTime, DateTime endTime)
```

Oyunun başlatılması için gerekli metottur. **roomId** ile oyunun odasını, **categories** ile oyunda hangi kategorilerin olacağını, **startTime** ile oyunun başlangıç tarihini, **endTime** ile oyunun bitiş tarihini parametre olarak alır.

```
var room = await context.Rooms.FirstOrDefaultAsync(r => r.Id == roomId);
```

Oda, contextten ID'si yardımıyla bulunur.

Eğer buradan gelen room boş değilse,

```
var newGame = new Game
{
    Id = Guid.NewGuid(),
    StartTime = startTime,
    EndTime = endTime,
    RoomId = roomId
};
```

Verilen bilgiler ile yeni bir oyun oluşturulur.

```
var categoryIds = categories.Select(c => c.Id).ToList();
var randomQuestions = await context.Questions
    .Where(q => categoryIds.Contains((Guid)q.CategoryId))
    .OrderBy(q => Guid.NewGuid())
    .Take(10)
    .ToListAsync();
```

Yine context yardımı ile seçili olan kategoriler bulunur ve 10 adet rastgele soru veritabanından çekilir.

```
GameSession.Instance.SetAllQuestions(randomQuestions);
foreach (var question in randomQuestions)
{
    var gameQuestion = new GameQuestion
    {
        Id = Guid.NewGuid(),
        GameId = newGame.Id,
        QuestionId = question.Id
    };

    newGame.GameQuestions.Add(gameQuestion);
}
```

Bu soruların listesi **GameSession**'a gönderilir. Ve her bir soru için ayrı **gameQuestion** nesnesi oluşturulur. Sonra da bu nesneler **GameQuestions** tablosuna eklenir.

```
room.CurrentGameId = newGame.Id;

RoomSession.Instance.SetCurrentRoom(room);

context.Games.Add(newGame);
await context.SaveChangesAsync();

GameSession.Instance.SetCurrentGame(newGame);

return newGame;
```

Oynanan oyun, oluşturulan oyuna eşitlenir

RoomSession içerisindeki Instance, içerisinde bulunulan odaya eşitlenir.

Context ile Games tablosuna oluşturulan oyun işlenir, kaydedilir.

GameSession içindeki instance için tüm bilgiler tamamlanınca yeni oluşturulan oyun aktif oyuna gönderilir ve bu yeni oyun, çağrılan metota geri döndürülür.

AnswerQuestion

```
public static async Task<bool> AnswerQuestion(Guid userId, Guid questionId, Guid gameId, string answerText)
```

Kullanıcı bir seçim yaptığı zaman, **userId** ile kullanıcının ID'sini, **questionId** ile soru ID'sini, **gameId** ile oyunun ID'sini, verilen cevabın içeriği için ise **answerText** parametrelerini alan bir metot çağırılır.

```
var user = await context.Users.FirstOrDefaultAsync(u => u.Id == userId);
var question = await context.Questions.FirstOrDefaultAsync(q => q.Id == questionId);
var game = await context.Games.FirstOrDefaultAsync(g => g.Id == gameId);
```

Önce tüm bu parametreler veritabanından bulunur ve değişkenlere atanır.

```
string[] questionAnswers = Helper.SplitString(question.OptionsText);
string correctAnswer = questionAnswers[question.CorrectAnswerIndex];
```

Daha sonra sorunun şıklarını bir listeye **Helper.SplitString** metodu sayesinde bölüyoruz. Bu sınıfa daha sonradan değineceğiz. Doğru şık ise, bu listenin kaçınıcı indexi olduğu ile bulunuyor. Karşılaştırma yapabilmek için bir değişkene atanıyor.

```
var isCorrectAnswer = correctAnswer.Equals(answerText, StringComparison.OrdinalIgnoreCase) ? 50 : 0;
```

Şık doğru ise, **isCorrectAnswer** true olarak dönüyor ve 50 ile 0 arasında bir değer veriyor.

```
if (user != null && question != null && game != null)
{
    var answer = new Answer
    {
        Id = Guid.NewGuid(),
        AnswerText = answerText,
        GainedXp = isCorrectAnswer,
        UserId = userId,
        QuestionId = questionId,
        GameId = gameId
    };

    context.Answers.Add(answer);
    await context.SaveChangesAsync();
    return true;
}

public static async Task<SummaryGame> GetSummaryGame(Guid gameId, Guid userId, Guid roomId)
{
    return false;
}
```

Tüm her şey doğru ve hiçbir **null** değilse yeni bir **answer** nesnesi oluşturup bunu database'e kaydediyoruz. Ardından da çağırıldığı metoda bunu **true** olarak dönüyoruz.

GetSummaryGame

Oyun bittiğinde, sonuç ekranının görüntülenmesinde, bize bilgileri sağlayan

```
var summaryGame = new SummaryGame();
```

metottur. **gameId** ile oyunun ID'sini, **userId** ile kullanıcı ID'sini, **roomId** ile odanın ID'sini parametre olarak alır.

Önce yeni bir **summaryGame** nesnesi oluşturulur.

```
var topUsers = await context.Answers
    .Where(a => a.GameId == gameId)
    .GroupBy(a => a.UserId)
    .Select(g => new { UserId = g.Key, SumXP = g.Sum(a => a.GainedXp) })
    .OrderByDescending(g => g.SumXP)
    .Take(3)
    .ToListAsync();

if (topUsers.Count >= 1)
{
    Guid? firstUser = topUsers[0].UserId;
    summaryGame.FirstUser = await context.Users.FirstOrDefaultAsync(u => u.Id == firstUser);
}

if (topUsers.Count >= 2)
{
    Guid? secondUser = topUsers[1].UserId;
    summaryGame.SecondUser = await context.Users.FirstOrDefaultAsync(u => u.Id == secondUser);
}

if (topUsers.Count >= 3)
{
    Guid? thirdUser = topUsers[2].UserId;
    summaryGame.ThirdUser = await context.Users.FirstOrDefaultAsync(u => u.Id == thirdUser);
}
```

Sonrasında context ile, ilgili özelliklerdeki en çok puan alan 3 kullanıcı, **firstUser**, **secondUser** ve **thirdUser** olarak **summaryGame** nesnesinin ilgili property'lerine yerleştirilir.

```
var categoryIds = await context.Categories.OrderBy(c=>c.Index).Select(c => c.Id).ToListAsync();

Guid? Category_1 = categoryIds[0];
Guid? Category_2 = categoryIds[1];
Guid? Category_3 = categoryIds[2];
Guid? Category_4 = categoryIds[3];
Guid? Category_5 = categoryIds[4];
```

Oyuncunun hangi kategoride kaç doğru yaptığını gösterebilmek için, önce kategorileri değişkenlere atıyoruz.

```
summaryGame.Category1Correct = await context.Answers
    .CountAsync(a => a.UserId == userId && a.GameId == gameId && a.Question.CategoryId == Category_1 && a.GainedXp > 0);
summaryGame.Category2Correct = await context.Answers
    .CountAsync(a => a.UserId == userId && a.GameId == gameId && a.Question.CategoryId == Category_2 && a.GainedXp > 0);
summaryGame.Category3Correct = await context.Answers
    .CountAsync(a => a.UserId == userId && a.GameId == gameId && a.Question.CategoryId == Category_3 && a.GainedXp > 0);
summaryGame.Category4Correct = await context.Answers
    .CountAsync(a => a.UserId == userId && a.GameId == gameId && a.Question.CategoryId == Category_4 && a.GainedXp > 0);
summaryGame.Category5Correct = await context.Answers
    .CountAsync(a => a.UserId == userId && a.GameId == gameId && a.Question.CategoryId == Category_5 && a.GainedXp > 0);
```

Daha sonra da **summaryGame** nesnesinin ilgili alanlarına, context yardımı ile çektiğimiz doğru sayılarını yerleştiriyoruz.

```
summaryGame.SumXP = await context.Answers
    .Where(a => a.UserId == userId && a.GameId == gameId)
    .SumAsync(a => (int?)a.GainedXp) ?? 0;
```

summaryGame nesnesinin **sumXP** alanına, contextten aldığımız kazanılmış XP yi yerleştiriyoruz.

```
var user = await context.Users.FirstOrDefaultAsync(u => u.Id == userId);
if (user != null)
{
    int result = (summaryGame.SumXP + user.Xp) / 500;
    if (result > 0)
        summaryGame.isLevelUp = true;
    else
        summaryGame.isLevelUp = false;

    user.Level += result;
    user.Xp = ((summaryGame.SumXP + user.Xp) % 500);

    summaryGame.Level = user.Level;
    summaryGame.SumXP = user.Xp;

    // set the current user to session
    UserSession.Instance.SetCurrentUser(user);
    await context.SaveChangesAsync();
}
```

Parametre olarak aldığımız **userId** ile context içerisinde bir arama yapıyoruz. Bu **user** başarılı bir şekilde dönerse, sonuçları hesaplıyoruz. Burada 500xp karşılığında 1 level artacak şekilde tasarladık. İsteğe göre bu değiştirilebilir. Sonrasında mevcut xp ile kazanılanı topladığımızda olan artışa göre de leveli güncelledik ve **“Level atladın!”** yazısını çıkarabilmek için **isLevelUp** değişkenini **true** atadık.

En son da **summaryGame** nesnesinin ilgili alanlarına Xp ve Level değerlerimizi yerleştirdik. **UserSession** içindeki instance yardımı ile, bu metotun çağırıldığı yere, bu **summaryGame**’e hangi kullanıcının sahip olduğu bilgisini de dönmüş

```
var room = await context.Rooms.FirstOrDefaultAsync(r => r.Id == roomId);
if (room != null)
{
    room.CurrentGameId = null;
}
```

olduk.

Verilen **roomId** ile eşleşen bir room varsa atanıyor, eğer yoksa **null** atanıyor.

```
RoomSession.Instance.SetCurrentRoom(room);
GameSession.Instance.SetCurrentGame(null);
GameSession.Instance.SetAllQuestions(null);
await context.SaveChangesAsync();
return summaryGame;
```

İlgili alanlar ayarlanıyor ve current game ile questions listesi temizleniyor. Database’e kaydediliyor.

b. MessageService

SendMessageAsync

```
public static async Task SendMessageAsync(Guid userId, string messageText, Guid roomId)
```

Herhangi bir kullanıcı odaya mesaj gönderdiğinde çalışan metod. Hangi oyuncunun gönderdiğini, mesajın içeriğini ve hangi odada gönderildiği bilgilerini parametre olarak alır.

```
var message = new Message
{
    Id = Guid.NewGuid(),
    MessageText = messageText,
    SentTime = DateTime.Now,
    UserId = userId,
    RoomId = roomId
};

context.Messages.Add(message);
await context.SaveChangesAsync();
```

Basitçe, yeni bir message listesi oluşturup, ilgili bilgileri parametreler ile dolduruyoruz. Daha sonradan database kaydını sağlıyoruz.

GetMessageByRoomId

```
public static async Task<List<Message>> GetMessagesByRoomId(Guid roomId)
```

O odada atılan mesajların listesini dönen bir fonksiyon. Loglamada kullanılabilir. Oda bilgisini parametre olarak alır.

```
using (var context = new qAppDbContext())
{
    var messages = await context.Messages
        .Where(m => m.RoomId == roomId)
        .OrderBy(m => m.SentTime)
        .Include(m => m.User)
        .ToListAsync();
    return messages;
}
```

Context yardımıyla yeni bir **messages** nesnesi oluşturup, ilgili bilgiler ile veritabanından mesajları çeker ve bir değişkene aktarır. Daha sonra bu nesneyi döner.

c. QuestionService

GetAllQuestions

```
public static async Task<List<Question>> GetAllQuestions()
```

Veritabanındaki tüm soruları liste şeklinde dönen bir fonksiyondur.

```
using (var db = new qAppDbContext())
{
    var questions = await db.Questions.Include(q => q.Category).ToListAsync();
    return questions;
}
```

Context yardımı ile tüm soruları bir değişkene atıp bu değişkeni dönüyoruz.

GetQuestionsById

```
public static async Task<Question> GetQuestionById(Guid id)
```

Parametre olarak ID alıp bize o ID de bir soru varsa onu getiren metod.

```
using (var db = new qAppDbContext())
{
    var question = await db.Questions.FirstOrDefaultAsync(q => q.Id == id);
    return question;
}
```

Context yardımı ile parametre olarak girilen ID ile eşleşen soruyu döndürür.

AddQuestion

```
public static async Task<bool> AddQuestion(Question question)
```

Parametre olarak soru nesnesini alır ve database kaydeder. Question nesnesinin daha önceden tanımlanmış olması gerekir. Bu metod sadece ekleme işlemini yapar.

```
using (var db = new qAppDbContext())
{
    db.Questions.Add(question);
    await db.SaveChangesAsync();
    return true;
}
```

Context yardımı ile question, questions listesine eklenir ve kaydedilir.

UpdateQuestion

```
public static async Task<bool> UpdateQuestion(Question question)
```

Parametre olarak bir soru nesnesi alır ve veritabanında ilgili yerdeki soruyu günceller.

```
using (var db = new qAppDbContext())
{
    db.Entry(question).State = EntityState.Modified;
    await db.SaveChangesAsync();
    return true;
}
```

Context yardımı ile verilen soruyu veritabanında günceller.
Daha sonrasında yine kaydetme işlemini yapar.

DeleteQuestion

```
public static async Task<bool> DeleteQuestion(Guid id)
```

Parametre olarak soru ID alır ve veritabanında eşleşen soruyu siler.

Önce veritabanında böyle bir soru var mı bakıyoruz.

```
var question = await db.Questions.FirstOrDefaultAsync(q => q.Id == id);
```

Answer tablosu ile de bağlantılı olduğu için silme işlemlerini Answers tablosunda da yapmamız gerekiyor. Yoksa silmez ya da hata alırız.

```
var answerIds = await db.Answers
    .Where(a => a.QuestionId == id)
    .Select(a => a.Id)
    .ToListAsync();

foreach (var answerId in answerIds)
{
    var answer = await db.Answers.FindAsync(answerId);
    if (answer != null)
    {
        db.Answers.Remove(answer);
    }
}
```

soruyu gameQuestionsIds tablosundan da silmemiz gerekiyor. Answer'da silmemiz ile aynı sebep.

```
var gameQuestionIds = await db.GameQuestions
    .Where(gq => gq.QuestionId == id)
    .Select(gq => gq.Id)
    .ToListAsync();

foreach (var gameQuestionId in gameQuestionIds)
{
    var gameQuestion = await db.GameQuestions.FindAsync(gameQuestionId);
    if (gameQuestion != null)
    {
        db.GameQuestions.Remove(gameQuestion);
    }
}
```

Tablodan siliyoruz ve değişiklikleri veri tabanında kaydediyoruz

```
db.Questions.Remove(question);

await db.SaveChangesAsync();
```

d. RoomService

CreateRoom

```
public static async Task<bool> CreateRoom(Room newRoom)
```

Yeni oda nesnesini parametre olarak alır ve yeni bir oda oluşturur. Bu oda nesnesini metotun çağırıldığı yerde oluşturulup hazır olarak gönderilmesi gerekir.

```
context.Rooms.Add(newRoom);
await context.SaveChangesAsync();

RoomSession.Instance.SetCurrentRoom(newRoom);
// set all categories to session
var categories = await context.Categories.ToListAsync();
RoomSession.Instance.SetAllCategories(categories);
var player = new Player
{
    Id = Guid.NewGuid(),
    RoomId = newRoom.Id,
    UserId = newRoom.AdminId
};
```

Context yardımı ile yeni odayı ekliyoruz ve kaydediyoruz.

Odayı kuran kişinin **GameSession** bilgisini güncelliyoruz.

Veritabanından kategorileri çekiyoruz.

RoomSession instance içerisindeki kategorilere bu kategorileri gönderiyoruz.

Yeni bir oyuncu oluşturup, bilgilerini odayı kuran kişinin bilgileri ile dolduruyoruz.

Sonrasında bu oyuncuyu Players tablosuna ekleyip veritabanına kaydediyoruz.

```
context.Players.Add(player);  
await context.SaveChangesAsync();
```

GetPlayers

```
public static async Task<List<User>> GetPlayers(Guid roomId)
```

Parametre olarak oda ID'si alıp odadaki oyuncuları liste halinde dönen metot.

```
using (var context = new qAppDbContext())  
{  
    var players = await context.Players  
        .Where(p => p.RoomId == roomId)  
        .Select(p => p.User)  
        .ToListAsync();  
  
    return players;  
}
```

Context yardımı ile players değişkenine veritabanındaki ilgili kayıtları yükleyip bu değişkeni döndürüyoruz.

JoinRoom

```
public static async Task<bool> JoinRoom(string code, string password, User user)
```

Odaya katılmayı sağlayan bu metot, parametre olarak bir katılım kodu, bir şifre ve katılan kişinin nesnesini alır.

Bu parametreler ile eşleşen bir oda varsa bunu değişkene atıyoruz.

```
var room = await context.Rooms.FirstOrDefaultAsync(r => r.Code == code && r.Password == password);
```

```
var bannedPlayer = await context.BannedUsers.FirstOrDefaultAsync(p => p.UserId == user.Id && p.RoomId == room.Id);  
if (bannedPlayer != null)  
{  
    return false;  
}  
RoomSession.Instance.SetCurrentRoom(room);
```

Önce katılmaya çalışan oyuncu banlı listesinde var mı bunu kontrol ediyoruz. Eğer yoksa return bloğuna girmeyip, oyuncunun **RoomSession** bilgisini girmeye çalıştığı room olarak güncelliyoruz.

```
var existingPlayer = await context.Players.FirstOrDefaultAsync(p => p.RoomId == room.Id && p.UserId == user.Id);  
if (existingPlayer == null)  
{  
    var player = new Player  
    {  
        Id = Guid.NewGuid(),  
        RoomId = room.Id,  
        UserId = user.Id  
    };  
  
    context.Players.Add(player);  
    await context.SaveChangesAsync();  
}
```

Banlı mı diye kontrol ettikten sonra bu kullanıcı, bir hata sonrası olası bir kopyalamanın önüne geçmek için bir de odada bulunuyor mu diye kontrol edilir. Eğer halihazırda odada bulunmuyorsa, yeni bir **player** nesnesi oluşturulur ve bu nesne **players** tablosuna kaydedilir. Daha sonra da veritabanına kaydedilir.

ExitRoom

```
public static async Task<bool> ExitRoom(Guid roomId, User user)
```

Odadan çıkmayı sağlayan bu metot, parametre olarak oda ID'sini ve kullanıcı nesnesini alır.

```
var player = await context.Players.FirstOrDefaultAsync(p => p.RoomId == roomId && p.UserId == user.Id);
```

Önce veri tabanında eşleşen kullanıcı bulunur ve değişkene atılır.

```
context.Players.Remove(player);
await context.SaveChangesAsync();

// Admin ise adminliği devret
if (isPlayerAdmin)
{
    var room = await context.Rooms.FirstOrDefaultAsync(r => r.Id == roomId);
    var players = await context.Players.Where(p => p.RoomId == roomId).ToListAsync();
    // Delete the room when admin exit
    foreach (Player p in players)
    {
        context.Players.Remove(p);
    }

    room.AdminId = null;
    await context.SaveChangesAsync();
}
```

Oyuncuyu **players** listesinden kaldırır ve kaydeder. Eğer çıkan oyuncu bir admin ise, **room** ve **players** bilgisi değişkenlere alınır ve her bir player **players** tablosundan silinir. **Room** içindeki adminId özelliği de **null**lanır. Böylece admin odadan çıktığında diğer oyuncular da lobi gider.

```
RoomSession.Instance.SetCurrentRoom(null);
RoomSession.Instance.SetAllCategories(null);
GameSession.Instance.SetCurrentGame(null);
```

Tüm **Session** değerleri **null**lanır.

BanUser

```
public static async Task<bool> BanUser(Guid userId, Guid roomId, Guid adminId)
```

Parametre olarak hangi kullanıcının, hangi odadan, hangi adminin banladığını parametre alır.

```
var room = await context.Rooms.FirstOrDefaultAsync(r => r.Id == roomId && r.AdminId == adminId);
```

Önce veritabanında böyle bir oda varsa bunu değişkene alıyoruz.

```
var player = await context.Players.FirstOrDefaultAsync(p => p.UserId == userId && p.RoomId == roomId);

if (player != null)
{
    var bannedUser = new BannedUser
    {
        Id = Guid.NewGuid(),
        UserId = player.UserId,
        RoomId = player.RoomId
    };

    context.Players.Remove(player);
    context.BannedUsers.Add(bannedUser);

    await context.SaveChangesAsync();
    return true;
}
```

Player'ı da veritabanından çekip değişkene atıyoruz. Yeni bir **bannedUser** nesnesi oluşturup bu tabloya eklenir. Aynı zamanda **players** tablosundan da çıkarılır. En son veritabanına kaydedilir.

KickUser

```
public static async Task<bool> KickUser(Guid roomId, Guid userId)
```

Parametre olarak hangi kullanıcının hangi odadan kickleneceği bilgilerini alır.

```
using (var context = new qAppDbContext())
{
    var player = await context.Players.FirstOrDefaultAsync(p => p.RoomId == roomId && p.UserId == userId);

    if (player != null)
    {
        context.Players.Remove(player);
        await context.SaveChangesAsync();
    }

    return true;
}
```

Context yardımı ile player bulunur ve

players listesinden silinir. Database'e kaydedilir.

CheckCurrentGame

```
public static async Task<bool> CheckCurrentGame(Guid roomId)
```

Parametre olarak oda ID'si alır ve oyuncular odada oynanan aktif bir oyun var mı onu kontrol eder.

e. UserService

AddUser

```
public static async Task<bool> AddUser(User user)
```

Parametre olarak kullanıcı nesnesi alır.

```
using (var db = new qAppDbContext())
{
    var newUser = new User
    {
        Id = Guid.NewGuid(),
        UserName = user.UserName,
        Email = user.Email,
        Password = passSalt[1],
        Salt = passSalt[0],
        Level = 1,
    };
    var isDuplicate = await db.Users.FirstOrDefaultAsync(u => u.Email == user.Email);
    if (isDuplicate != null)
    {
        return false;
    }
    db.Users.Add(newUser);
    await db.SaveChangesAsync();
}
```

Context yardımı ile yeni bi kullanıcı oluşturup bilgilerini atıyoruz. Eğer kopyalama varsa, eklemeyi yapmıyor.

Daha sonrasında database'e kaydediyor.

LoginUser

```
public static async Task<bool> LoginUser(string email, string password)
```

Parametre olarak kullanıcının maili ve şifresini alır. Kullanıcının giriş yapmasını sağlar.

```
var user = await context.Users.FirstOrDefaultAsync(u => u.Email == email);

if (user == null)
{
    isValid = false;
    //return false; // Kullanıcı adı yanlış
}
else
{
    byte[] saltValue = Convert.FromBase64String(user.Salt);
    var keyGenerator = new Rfc2898DeriveBytes(password, saltValue, 10000);
    byte[] encryptionKey = keyGenerator.GetBytes(32); // 256 bit = 32 byte
    string encryptionKeyString = Convert.ToBase64String(encryptionKey);

    if (encryptionKeyString.Equals(user.Password))
    {
        isValid = true;
        //return true; // Şifre doğru
        UserSession.Instance.SetCurrentUser(user);
    }
    else
    {
        isValid = false;
        //return false; // Şifre yanlış
    }
}
```

Context yardımı ile kullanıcı varsa değişkene atılır. Eğer kullanıcı **null** gelmezse, ilgili **encryption** işlemleri yapıldıktan sonra şifre kontrol edilir ve eğer şifre doğru ise **UserSession** bilgisi güncellenir.

Sonrasında giriş yapıldığına dair **true** bilgisi bize dönmüş olur.

UpdateUser

```
public static async Task<bool> UpdateUser(User user, string currentPassword, Guid currentId)
```

Parametre olarak bir kullanıcı nesnesi, bir mevcut şifre, bir de mevcut ID alan bu parametre, profil kısmından güncelleme yapıldığında kullanıcı bilgilerini güncellemeye yarıyor.

```
using (var db = new qAppDbContext())
{
    var existingUser = await db.Users.FirstOrDefaultAsync(u => u.Id == currentId);

    if (existingUser != null)
    {
        byte[] saltValue = Convert.FromBase64String(existingUser.Salt);
        var keyGenerator = new Rfc2898DeriveBytes(currentPassword, saltValue, 10000);
        byte[] encryptionKey = keyGenerator.GetBytes(32); // 256 bit = 32 byte
        string encryptionKeyString = Convert.ToBase64String(encryptionKey);

        if (encryptionKeyString.Equals(existingUser.Password))
        {
            //return true; // Şifre doğru
            string[] passSalt = PassSaltGenerator(user.Password);

            existingUser.UserName = user.UserName;
            existingUser.Email = user.Email;
            existingUser.Password = passSalt[1];
            existingUser.Salt = passSalt[0];

            UserSession.Instance.SetCurrentUser(existingUser);
        }
    }

    public static void LogoutUser()
    {
        return true;
    }

    UserSession.Instance.SetCurrentUser(null);
}
```

Login ile çok benzer bir mantık kullanan UpdateUser metodu, gereken **encryption** işlemlerini yaptıktan sonra eğer şifreler doğru ise ilgili yerlerdeki bilgiler ile mevcut kullanıcıyı günceller.

LogoutUser

Kullanıcının çıkış yapmasını sağlar.

UserSession bilgisini null olarak günceller.

GetUserGamesSummary

```
public static async Task<UserGamesSummary> GetUserGamesSummary(Guid userId)
```

Parametre olarak userID alan bu metot, kullanıcının profil sonuçlarını getirmeye yarar.

```
var userGamesSummary = new UserGamesSummary();

userGamesSummary.TotalGamesPlayed = await context.Answers
    .Where(a => a.UserId == userId)
    .Select(a => a.GameId)
    .Distinct()
    .CountAsync();
```

Context yardımıyla **userGamesSummary** nesnesindeki toplam oynanan oyunu bulur.

```
var distinctGameIds = await context.Answers
    .Where(a => a.UserId == userId)
    .Select(a => a.GameId)
    .Distinct()
    .ToListAsync();

userGamesSummary.WonGames = distinctGameIds.Count(g => context.Answers
    .Where(a => a.UserId == userId && a.GameId == g)
    .Sum(a => a.GainedXp) == distinctGameIds.Max(g2 => context.Answers
    .Where(a => a.UserId == userId && a.GameId == g2)
    .Sum(a => a.GainedXp)));

userGamesSummary.CorrectAnswers = await context.Answers
    .CountAsync(a => a.UserId == userId && a.GainedXp >= 50);
```

Yine context yardımı ile aynı nesnenin bu sefer **wonGames** özelliğini dolduruyoruz.

Aynı zamanda doğru cevapların sayısını da bize dönüyor.

```
var categoryIds = await context.Categories.OrderBy(c => c.Index).Select(c => c.Id).ToListAsync();

Guid? Category_1 = categoryIds[0];
Guid? Category_2 = categoryIds[1];
Guid? Category_3 = categoryIds[2];
Guid? Category_4 = categoryIds[3];
Guid? Category_5 = categoryIds[4];

userGamesSummary.Category1Correct = await GetCategoryCorrectCount(userId, (Guid)Category_1);
userGamesSummary.Category2Correct = await GetCategoryCorrectCount(userId, (Guid)Category_2);
userGamesSummary.Category3Correct = await GetCategoryCorrectCount(userId, (Guid)Category_3);
userGamesSummary.Category4Correct = await GetCategoryCorrectCount(userId, (Guid)Category_4);
userGamesSummary.Category5Correct = await GetCategoryCorrectCount(userId, (Guid)Category_5);
HideLoadingIndicator();
return userGamesSummary;
```

Context yardımı ile kategoriye göre doğru cevap sayısını getiriyor. Ve gerektiği yerde direkt

nesnenin bir özelliği olarak görülebiliyor

2.3 Sessions (Oturumlar)

Oturumlar genel anlamıyla, kullanıcının giriş yaptığından itibaren anlık olarak yaptığı hareketlerin bir nesne ya da değişkene atandığı yerlerdir. Bu projede üç adet oturum kullandık.

```
class UserSession
{
    private static UserSession _instance;
    private User _currentUser;

    1 reference
    private UserSession() {
        _currentUser = new User();
    }

    18 references
    public static UserSession Instance
    {
        get
        {
            if (_instance == null)
            {
                _instance = new UserSession();
            }
            return _instance;
        }
    }

    4 references
    public void SetCurrentUser(User user)
    {
        _currentUser = user;
    }

    14 references
    public User GetCurrentUser()
    {
        return _currentUser;
    }
}
```

2.3.1 UserSession

Oturumlar genel olarak Singleton modelini kullanır çünkü ayrı ayrı nesne oluşturmaya izin vermeden, içerisindeki statik metotlardan erişim sağlarlar.

Bu oturum tasarımında bir instance vardır ve kodun birçok yerinde aslında UserSession.Instance olarak çağırılır. Eğer herhangi bir Instance bulunamazsa yeni oluşturur ve döner, eğer varsa mevcut Instance döner.

SetCurrentUser, parametre olarak bir User nesnesi alır ve genellikle login işleminden sonra çağırılır.

GetCurrentUser, kullanım sıklığı 14 referans olduğundan görüldüğü üzere, çok işe yarayan bir metottur. Instance ulaşmak için her seferinde çağırılması gereken bir metottur.

```
class RoomSession
{
    private static RoomSession _instance;
    private List<Category> _allCategories;
    private List<Category> _selectedCategories;
    private Room _currentRoom;

    1 reference
    private RoomSession() { }

    36 references
    public static RoomSession Instance
    {
        get
        {
            if (_instance == null)
            {
                _instance = new RoomSession();
            }
            return _instance;
        }
    }

    // DB den çekilen kategorileri set etmek
    3 references
    public void SetAllCategories(List<Category> categories)
    {
        _allCategories = categories;
    }

    1 reference
    public void AddSelectedCategory(Category category)
    {
        _selectedCategories.Add(category);
    }

    1 reference
    public void RemoveSelectedCategory(Category category)
    {
        _selectedCategories.Remove(category);
    }
}
```

2.3.2 RoomSession

Singleton yapısını kullanan oturumdur.

SetAllCategories, bir kategori listesi alır ve kendi içerisindeki **_allCategories** değişkenini doldurur.

AddSelectedCategory, bir kategori nesnesi alır ve lider tarafından seçilen kategorinin **_selectedCategories** listesine eklenmesini sağlar.

RemoveSelectedCategory, bir kategori nesnesi alır ve yine lider tarafından seçimi kaldırılan kategorinin listeden silinmesini sağlar.

```

public List<Category> GetSelectedCategories()
{
    return _selectedCategories;
}
9 references
public void SetCurrentRoom(Room room)
{
    _currentRoom = room;
}
18 references
public Room GetCurrentRoom()
{
    return _currentRoom;
}

```

GetSelectedCategories, çağırıldığı yere listesini dönen fonksiyondur.

SetCurrentRoom, bir room nesnesi alır ve güncel odayı bu nesne ile doldurur. Genelde oda oluşturulduğu ya da mevcut bir odaya katılanacağı zaman çağırılır.

GetCurrentRoom, mevcut oda oturumunun nesnesini döner.

```

class GameSession
{
    private static GameSession _instance;
    3 references
    private Game _currentGame { get; set; }
    3 references
    public List<Question> _gameQuestions { get; set; }

    1 reference
    private GameSession()...

    17 references
    public static GameSession Instance...

    // DB den çekilen kategorileri set etmek
    5 references
    public void SetAllQuestions(List<Question> questions)
    {
        _gameQuestions = questions;
    }
    3 references
    public List<Question> GetAllQuestions()
    {
        return _gameQuestions;
    }
    6 references
    public void SetCurrentGame(Game game)
    {
        _currentGame = game;
    }
    3 references
    public Game GetCurrentGame()
    {
        return _currentGame;
    }
}

```

2.3.3 GameSession

Oyun oturumu, genel olarak oyun başladığında doldurulan ve çağırılan oturumdur.

SetAllQuestions, bir soru listesi alır ve o oyunun listesini günceller.

GetAllQuestions, o oyundaki soruların listesini döndürür.

SetCurrentGame, bir oyun nesnesi alır ve mevcut oyunu günceller.

GetCurrentGame, mevcut oyunu bir nesne olarak döner.

2.4 Helper

Helper.cs, çoğu gruba uymayan ama bize yardımcı olacak metotları yazdığımız yer olarak tanımlanabilir. İçerisindeki metotlara bakacak olursak

ConcatenateStrings

```
public static string ConcatenateStrings(params string[] strings)
{
    return string.Join("\n", strings);
}
```

Parametre olarak bir dizi string alan bu metot, bunların arasına belli bir karakter koyarak birleştirip tek bir stringe çevirir. Burada biz bu string'i "**\n**" olarak belirledik. Hem cevap kısmında kullanılması zor hem de güzel bir araç olduğu için bunu seçtik.

SplitString

```
public static string[] SplitString(string input)
{
    return input.Split('\n');
}
```

Parametre olarak tek bir string alır ve belirlenen bir **char** karakteri gördüğü yerlerde stringi ayırıp bir diziye depolar. Sonrasında bu listeyi bize döner.

FindFirstTrueIndex

```
public static int FindFirstTrueIndex(bool[] array)
{
    for (int i = 0; i < array.Length; i++)
    {
        if (array[i])
        {
            return i;
        }
    }
    return -1; // Eğer true eleman bulunamazsa -1 döndürülür
}
```

Bu da bize, bir örnek olarak soru ekleme menüsünde kategori seçerken hangi kategorinin **true** olarak işaretlendiğini bulmaya yarıyor. Bir boolean dizisi alıp bunun ilk **true** değerini bize index olarak dönüyor.

```
public static void AddSelectedCategory(int buttonIndex)
{
    List<Category> allCategories = RoomSession.Instance.GetAllCategories();

    Category selectedCategory = allCategories.FirstOrDefault(c => c.Index == buttonIndex);

    RoomSession.Instance.AddSelectedCategory(selectedCategory);
}

2 references
public static void RemoveSelectedCategory(int buttonIndex)
{
    List<Category> allCategories = RoomSession.Instance.GetAllCategories();

    Category selectedCategory = allCategories.FirstOrDefault(c => c.Index == buttonIndex);

    RoomSession.Instance.RemoveSelectedCategory(selectedCategory);
}
```

2.5 PanelHandler

Proje genel anlamda formların içinde fazla sayıda panel yardımı ile routing ve diğer işlemleri gerçekleştiriyor. Bunun için de panellerin çok iyi ve kolay bir şekilde kontrol edilmesi gerekmekte. Biz de birnevi kendi metodlarımızı yazıp bunların kontrolünü sağladık.

setPanelMiddle

```
public static void setPanelMiddle(Form form, Panel inactive_panel, Panel active_panel)
{
    inactive_panel.Visible = false;
    active_panel.Size = new Size(600, 600);
    active_panel.Left = (form.Width - active_panel.Width) / 2;
    active_panel.Top = (form.Height - active_panel.Height) / 2;
    active_panel.Visible = true;
}
```

Parametre olarak üzerinde çalışılan formun referansını, kapanacak ve açılacak olan formların referanslarını alır.

Kapanacak olan panelin **.Visible** özelliğini **false** yaptıktan sonra, açılacak olan panelin önce boyutunu, sonra konumunu belirler ve sonrasında görünürlüğü değiştirir. Formun içerisinde aynı anda 1 aktif panel bulunduğu için bu paneli daha sonra da anlatacağımız üzere **active_panel** adlı bir değişkende tutuyoruz. Böylece aktif paneli kapatıp yeni paneli açacağımız zaman, yeni active_panel değişkenini o panele atıyoruz ve böylece kolay bir kontrol sağlanmış oluyor. Aynı anda birden fazla panel açılıyor durumda olsaydı bu durum tabi daha farklı dizayn edilebilirdi.

setPanelFill

```
public static void setPanelFill(Panel inactive_panel, Panel active_panel)
{
    inactive_panel.Visible = false;
    active_panel.Visible = true;
    active_panel.Dock = DockStyle.Fill;
}
```

Parametre olarak kapanacak ve açılacak olan panelin referanslarını alan bu metod, kapanacak paneli kapattıktan ve açılacak paneli açtıktan sonra, açılan paneli ana kapsayıcıya yerleştiriyor. Neden hem ortalama hem de genişletme olarak iki metod var diye bir soru sorulabilir, örneğin **LobbyGame** yani oyunun oynandığı kısımdaki paneller **FullScreen** modda başlatıldığı için yani ortalananacak herhangi bir içerik olmadığından **Fill** metodunu çağırıyoruz. Fakat ana menü yani **MainMenu.cs** de olduğu gibi panelleri 600px – 600px genişlikte açtığımızda takdir edersiniz ki ortalananması gerekmekte.

centralizePanel

```
public static void centralizePanel(Form form, Panel panel)
{
    panel.Left = (form.Width - panel.Width) / 2;
    panel.Top = (form.Height - panel.Height) / 2;
}
```

Parametre olarak üzerinde çalışılan formu ve ortalananacak paneli alan bu metod, projenin belli kısımlarında kullanıldı ve daha çok bir yardımcı araç gibiydi. Öncelik olarak, formun üst bar kısmından tutup örneğin **Full Screen** modundan küçültülmüş moda alındığında, bunu handle etmezsek, panel ekranın alakasız bir yerinde çalışmasını sürdürmüş olacaktı. Bu metod sayesinde **Form_OnResizing** eventi tetiklendiğinde bu metodu çağırıp panelin o anki form ortasına gelmesini sağlayabiliyoruz.

2.6 ThemeHandler

Bu kısım aslında işin biraz daha UI ve tasarım kısmını ilgilendiriyor. Projede belirli bir renk şablonu ve uyumu olması göze daha hoş gelmekle birlikte projeyi amatörükten biraz olsun uzaklaştırıyor.

```
static public Color color_texts;  
static public Color color_background;  
static public Color color_textboxes;  
static public Color color_alt1;  
static public Color color_alt2;
```

Statik olarak tanımladığımız renkler ile hem projenin her yerinden bu renklere erişebiliyoruz hem de renkler üzerinde daha düzenli kontroller yapmış oluyoruz. Yazılar için ayrı, arkaplan için ayrı, metin kutuları ve kullanılacak alternatif renkler gibi renkler tanımlayıp, tek yapmamız gereken o fonksiyonu çağırıp bu renklerin atamasını yapmak oluyor.

```
public static void loadColors()  
{  
    theme3();  
}
```

Proje ilk yüklendiğinde takdir edersiniz ki bir teması olmalı ki biz bu temayı **Koyu Tema** yani **theme3** ismiyle tanımladık.

```
public static void changeFormsColor(Form form)  
{  
    form.BackColor = color_background;  
}
```

Parametre olarak form ögesinin referansını alan **changeFormsColor** metodu, üzerinde çalıştığımız formun arkaplanını değiştirmemize yarıyor. Buna neden ihtiyaç duyuyoruz sorusuna gelecek olursak da, formun dizaynı ve editör kısmında her şeyin daha rahat görünebilmesi açısından renkler genelde uygulama başlatıldıktan sonraki halinden çok çok farklı oluyor. Uygulama başladığında sanki bambaşka bir program çalışıyor havası veriyor. Bu açıdan biz genel olarak geliştirme yaptığımız renkler ile uygulama renklerini birbirinden ayrı tuttuk. Bu sayede de **Visual Studio'nun** tüm nimetlerinden aslında faydalanmaya çalıştık.

```
public static void changeAllControlsColor(Control c)  
{  
    foreach(Control ctrl in c.Controls)  
    {  
        if (ctrl.HasChildren)  
            changeAllControlsColor(ctrl);  
        else  
        {  
            if(ctrl is TextBox)  
            {  
                ctrl.BackColor = color_textboxes;  
                ctrl.ForeColor = color_texts;  
            }else if(ctrl is Label)  
            {  
                ctrl.ForeColor = color_texts;  
            }else if(ctrl is MaskedTextBox)  
            {  
                ctrl.BackColor = color_textboxes;  
                ctrl.ForeColor = color_texts;  
            }else if(ctrl is Button)  
            {  
                ctrl.ForeColor = color_texts;  
                ctrl.BackColor = Color.Transparent;  
                Button b = (Button)ctrl;  
                b.FlatAppearance.MouseOverBackColor = ControlPaint.Light(color_background);  
                b.FlatAppearance.MouseDownBackColor = Color.Transparent;  
            }else if(ctrl is GroupBox)  
            {  
                ctrl.ForeColor = color_texts;  
            }else if(ctrl is RadioButton)  
            {  
                ctrl.ForeColor = color_texts;  
            }else if(ctrl is ListView)  
            {  
                ctrl.BackColor = color_background;  
                ctrl.ForeColor = color_texts;  
            }  
        }  
    }  
}
```

Asıl temayı yükleyen ve tüm kontrollerin rengini düzenleyen metodumuz **changeAllControlsColor** oldu. İçine bir Control türünden nesne alıyor ki bu nesne bir form da olabilir, bir buton veya label. Çünkü bu componentlerin tümü **Control** sınıfından türemiş nesnelerdir. Bizim burada yaptığımız işlem, bu metodun çağırıldığı formda, parametre olarak **this** göndermek ve gerisini buraya bırakmak. Bu metod, önce ona gönderilen kontrolde bir çocuk var mı diye bakıyor. Çünkü takdir edeceğimiz üzere iç içe panellere projede sıkça yer verdik. Eğer varsa, o kontrol ile birlikte bu metod tekrardan çağırılıyor ve rekürsif bir şekilde tüm kontrollere erişim sağlanmış oluyor. Eğer çocuğu yoksa işte o zaman renk işlemlerine

Giriyoruz. Örneğin **ctrl** ismini verdiğimiz kontrol bir **TextBox** nesnesi ise, renklerini ThemeHandler içindeki statik değişkenlerden alıyor. Bu sayede forma ne eklersek ekleyelim, rengi ile oynamaya gerek kalmadan yalnızca **Form_Load** eventi içerisine yazılan **ThemeHandler.changeAllControlsColor(this)** komutu, uygulama çalıştırıldığında tüm kontrollerin renklerini temaya uygun şekilde yüklüyor.

```
public static void lightTheme() //Default light theme
{
    color_texts = ColorTranslator.FromHtml("#392E3C");
    color_background = ColorTranslator.FromHtml("#BAB4E1");
    color_textboxes = ColorTranslator.FromHtml("#9891C1");
    color_alt1 = ColorTranslator.FromHtml("#3C3A49");
    color_alt2 = ColorTranslator.FromHtml("#1F1531");
    //color_buttons = Color.White;
}
```

```
public static void theme3()
{
    color_texts = ColorTranslator.FromHtml("#BBE1FA");
    color_background = ColorTranslator.FromHtml("#1B262C");
    color_textboxes = ColorTranslator.FromHtml("#294749");
    color_alt1 = ColorTranslator.FromHtml("#0F4C75");
    color_alt2 = ColorTranslator.FromHtml("#FF8341");
}
```

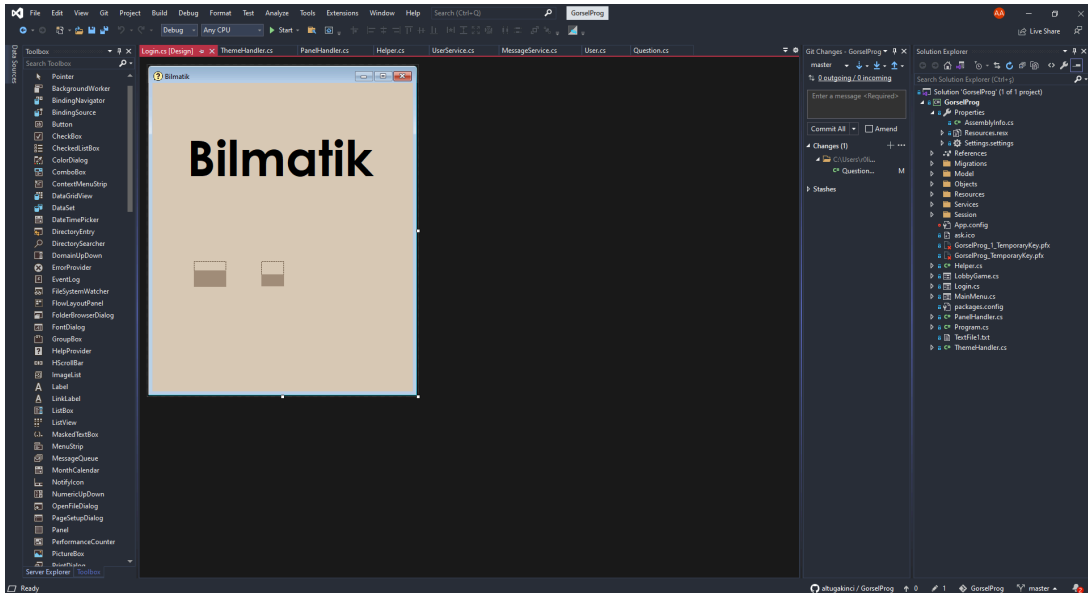
Aynı zamanda bu kısma eklenecek yeni tema metodları ile çok kolay bir biçimde renkler arası geçiş yapabiliyoruz. Açık ve koyu temamızı değiştirebiliyoruz ki bu da modülerlik açısından oldukça rahat bir özellik.

3. Formlar

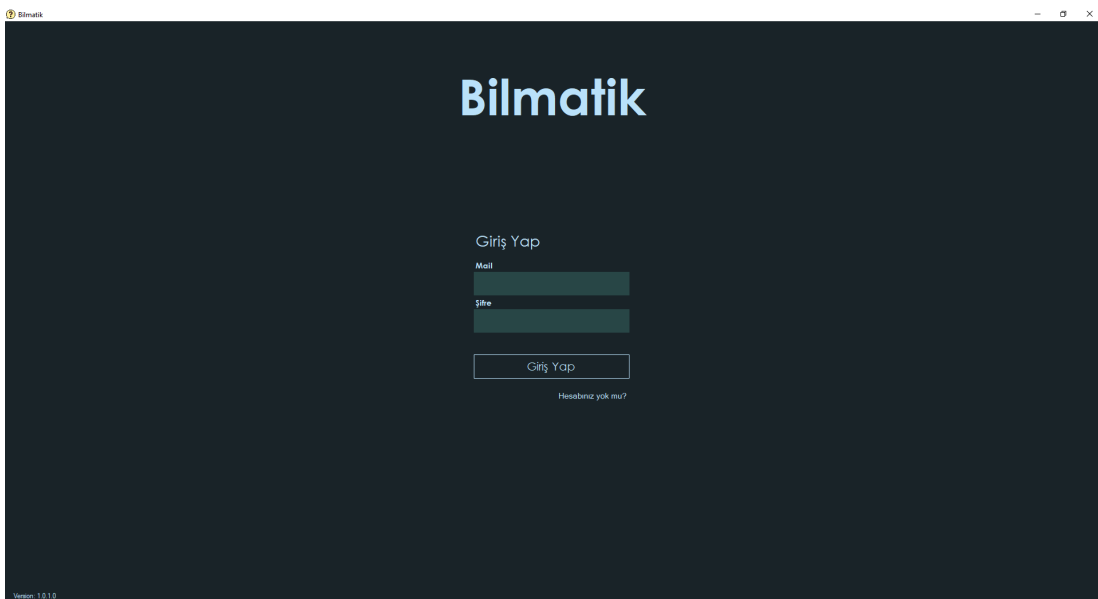
Daha önce de bahsettiğimiz üzere projede yalnızca 3 form ve buna bağlı olan onlarca panelimiz var. Örneğin ana menüde Profil kısmı bir panelde tutulurken, seçenekler kısmı ayrı bir panelde tutulmakta. Uygulama başlatıldığından itibaren her şey otomatize bir şekilde ortaladığı ve renkleri ayarlandığı için panellerin boyutlarında yapılan hiçbir değişiklik uygulamanın kendisinde herhangi bir soruna yol açmıyor.

3.1 Login.cs (Form)

Kullanıcıların giriş yapıp kayıt olabilecekleri form olarak nitelendirebiliriz.

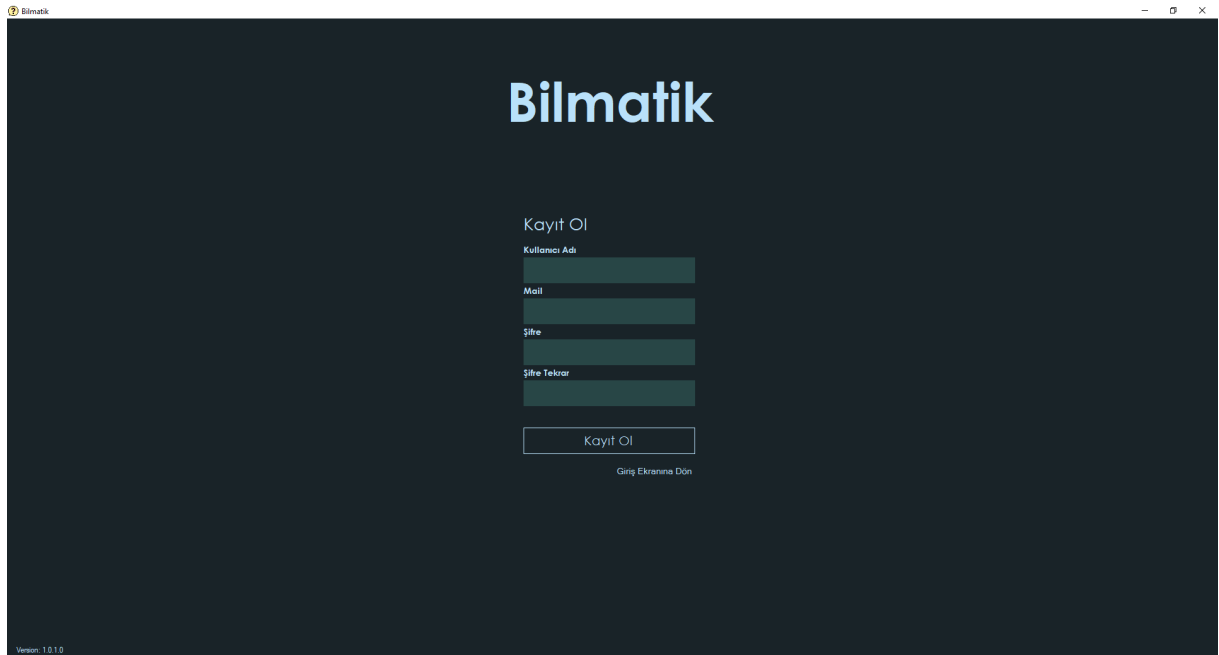


Editör görüntüsünü gördüğümüze göre şimdi de uygulama içindeki görüntüsüne bakalım.



Hesabımız varsa bu ekranda **login** olabiliyoruz. Eğer yoksa da;

Hesabınız Yok Mu? Labelına tıkladığımız zaman bizi bir **Register** paneli karşılıyor.



Bu kısımdaki kodları inceleyecek olursak, çoğu alan handle edilmiş ve boş girdilere karşı veritabanında hataya sebep olmayacak şekilde tasarlandı.

```
private async void btnLogin_Click(object sender, EventArgs e)
{
    if (txtLoginEmail.Text == "")
    {
        lblLoginWarning.Text = "Email alanı boş bırakılamaz!";
        return;
    }
    if (txtLoginPassword.Text == "")
    {
        lblLoginWarning.Text = "Şifre alanı boş bırakılamaz!";
        return;
    }

    bool isValid = await UserService.LoginUser(txtLoginEmail.Text, txtLoginPassword.Text);

    // TODO: Loading işlemleri buraya eklenebilir

    if (!isValid) {
        MessageBox.Show("Kullanıcı adı veya şifre yanlıştır.", "Alert", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    } else
    {
        formMainMenu game = new formMainMenu();
        this.Hide();
        game.Show();
        game.WindowState = FormWindowState.Maximized;
    }
}
```

Önce alanların boş olup olmadığının kontrolünü yaptık. Daha sonrasında kullanıcı **login** edebilmek için **UserService** servisimizin **LoginUser** metotunu kullandık. Parametre olarak da **textBox**' larımızdan aldığımız verileri kullandık. Eğer bize geçerli bir sonuç dönerse, ana menü formumuzu yükledik ve tam ekran yaptık. Eğer hatalı bir dönüş sağlanır ve kullanıcı veritabanında bulunamazsa, o halde bir uyarı mesajı döndürdük.

Register Butonu

```
private async void btnRegister_Click(object sender, EventArgs e)
{
    if (txtRegUsername.Text == "")
    {
        lblRegWarning.Text = "Kullanıcı adı boş bırakılamaz!";
        return;
    }
    if (txtRegMail.Text == "")
    {
        lblRegWarning.Text = "Email alanı boş bırakılamaz!";
        return;
    }
    if (txtRegPassword.Text == "")
    {
        lblRegWarning.Text = "Şifre alanı boş bırakılamaz!";
        return;
    }
    if (txtRegPassword2.Text == "")
    {
        lblRegWarning.Text = "Şifreyi tekrar girmelisiniz!";
        return;
    }
    if (!txtRegPassword.Text.Equals(txtRegPassword2.Text))
    {
        lblRegWarning.Text = "Şifreler Uyuşmuyor!";
        return;
    }

    User user = new User { UserName = txtRegUsername.Text, Email = txtRegMail.Text, Password = txtRegPassword.Text };

    bool isSuccess = await UserService.AddUser(user);

    if (isSuccess)
    {
        MessageBox.Show("Başarılı bir şekilde kayıt oluşturulmuştur.", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("Böyle bir kullanıcı vardır", "Alert", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    // TODO: buraya belki bi loading gibi birşey gelebilir

    txtRegUsername.Text = "";
    txtRegMail.Text = "";
    txtRegPassword.Text = "";
    txtRegPassword2.Text = "";

    PanelHandler.setPanelFill(active_panel, pnlLogin);
    active_panel = pnlLogin;
}
```

Kayıt ol butonuna basıldığı anda yine boş alan kontrolü yapılıyor ve iki defa girilmesini istediğimiz şifrelerin birbiri ile uyuşup uyuşmadığına bakıyoruz. Sonrasında bu bilgiler ile yeni bir **User** nesnesi oluşturup bunu yine **UserService** içindeki **AddUser** metotuna gönderiyoruz. Bu metot bizim için veritabanı ile bağlantı sağlayıp ilgili tabloya bunu ekleyecektir. Eğer başarılı dönüş olursa kayıt oluşturuldu bildirimi geldikten sonra ilgili alanları temizleyip kullanıcıyı **login** ekranına döndürüyoruz.

Buradaki kullanımdan görmüş olacağınız üzere **PanelHandler.setPanelFill** metodu kullanıldı. **Login.cs** formumuzda fill işlemi yapıyoruz ve açılan panellerden sonra da **active_panel** değişkenini güncellemeyi asla unutmuyoruz.

Burada **login** ve **register** işlemlerini yaparken kullanıcı herhangi bir **textbox** içerisinde “**Enter**” tuşuna basarsa, sanki ilgili butonlara tıklamış gibi bir yapı kuruyoruz. Böylece kullanıcının direkt olarak mouse ile butona basmasına gerek kalmıyor.

```
private void txtLoginPassword_KeyDown(object sender, KeyEventArgs e)
{
    if(e.KeyCode == Keys.Enter)
    {
        btnLogin_Click(null,null);
    }
}

private void txtLoginEmail_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        btnLogin_Click(null, null);
    }
}

private void txtRegPassword2_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        btnRegister_Click(null, null);
    }
}

private void txtRegUsername_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        btnRegister_Click(null, null);
    }
}

private void txtRegMail_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        btnRegister_Click(null, null);
    }
}

private void txtRegPassword_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        btnRegister_Click(null, null);
    }
}
```

```
private void Form1_Load(object sender, EventArgs e)
{
    active_panel = pnlLogin;
    PanelHandler.setPanelFill(active_panel, pnlLogin);

    ThemeHandler.loadColors();
    ThemeHandler.changeAllControlsColor(this);
    ThemeHandler.changeFormsColor(this);
    this.WindowState = FormWindowState.Maximized;

    lblVersion.Text = $"Version: {Application.ProductVersion}";
}
```

Form yüklendiği zaman da yine aktif panelimizi güncelliyoruz ve PanelHandler yardımı ile açıyoruz. Daha sonradan temayı yüklüyoruz ve tam ekran yapıyoruz. Versiyon bilgisini de sol alt kısımdaki label'a yazdırarak **versiyon** takibini de yapmış oluyoruz.

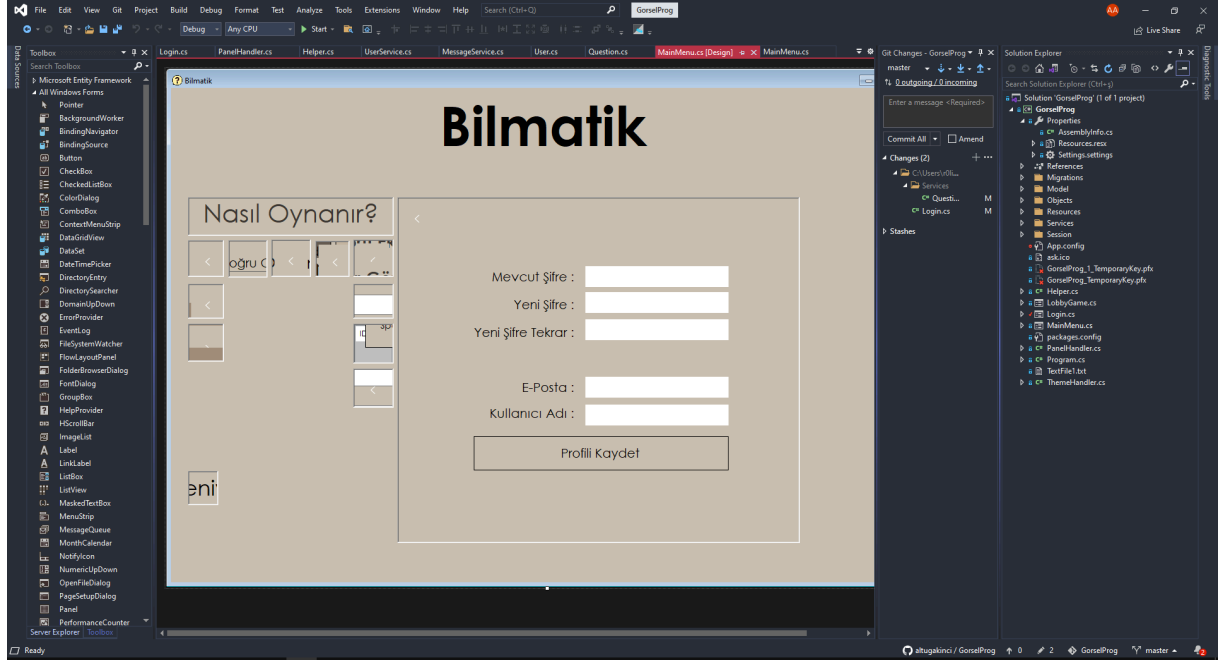
Bu formun son kodları olan **Hesabınız Yok Mu?** Veya **Giriş Ekranına Dön** label'larına tıklandığında gerçekleşen **routing** işlemlerini de aşağıda görebilirsiniz.

```
private void lblNoAcc_click(object sender, EventArgs e)
{
    PanelHandler.setPanelFill(active_panel, pnlRegister);
    active_panel = pnlRegister;
}

1 reference
private void lblRetLogin_MouseClick(object sender, MouseEventArgs e)
{
    PanelHandler.setPanelFill(active_panel, pnlLogin);
    active_panel = pnlLogin;
}
```

3.2 MainMenu.cs (Form)

Genel olarak tüm ana menü panellerinin içinde bulunduğu form olarak nitelendirilebilir. Editör içinde karmaşık bir şey olarak görünse de uygulama çalıştığında bu görüntüden eser kalmıyor.



Form yüklendiğinde olan olaylar öncekilere çok benzer.

```
private void Game_Load(object sender, EventArgs e)
{
    active_panel = pnlMainMenu;
    ThemeHandler.changeAllControlsColor(this);
    ThemeHandler.changeFormsColor(this);
    this.WindowState = FormWindowState.Maximized;
    PanelHandler.setPanelMiddle(this, pnlMainMenu);

    this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.formMainMenu_FormClosing);
    this.SizeChanged += new EventHandler(this.form_resize);
}
```

Farklı olarak **FormClosing** eventine bağlı olarak ilgili metodu çalıştırıyoruz. Aynı zamanda formun boyutlarında bir değişiklik olduğunda da **Resize** fonksiyonunu çağırıyoruz ve paneli ortalıyoruz.

Routing kısımlarında yaptığımız şey hep aynı, PanelHandler metodunu çağırıp aktif paneli güncelledik.



Ana menünün görüntüsü bu şekilde.

Profil kısmına tıkladığımız zaman, veritabanından ilgili alanların bilgilerini çekmesi için servisi çağırıyoruz.



```
private async void btnProfile_Click(object sender, EventArgs e)
{
    User current = UserSession.Instance.GetCurrentUser();
    Objects.UserGamesSummary sum = await UserService.GetUserGamesSummary(current.Id);
    lblProfileUsername.Text = current.UserName;
    lblProfileMail.Text = current.Email;
    lblProfileLevel.Text = $"{current.Level}. Level";
    lblProfileXP.Text = $"{current.Xp} / 500";
    prgProfileXP.Value = current.Xp;

    lblProfilePlayedGames.Text = sum.TotalGamesPlayed.ToString();
    lblProfileWins.Text = sum.WonGames.ToString();

    //spor tarih sanat bilim eđl
    lblProfileSpor.Text = sum.Category1Correct.ToString();
    lblProfileTarih.Text = sum.Category2Correct.ToString();
    lblProfileSanat.Text = sum.Category3Correct.ToString();
    lblProfileBilim.Text = sum.Category4Correct.ToString();
    lblProfileEglence.Text = sum.Category5Correct.ToString();

    PanelHandler.setPanelMiddle(this, active_panel, pnlProfile);
    active_panel = pnlProfile;
}
}
```

Önce mevcut kullanıcıyı **UserSession** dan alıyoruz. Sonrasında **UserGamesSummary** sınıfını çağırıp oyuncunun özet bilgilerine erişiyoruz. Ardından ilgili bilgileri ilgili alanlara eşleyip görüntülenmesini sağlıyoruz. En son da aktif paneli güncellemeyi ihmal etmiyoruz.

Nasıl oynanır kısmında ise oyuncuların birbiri ile nasıl oynayabileceğini anlatan küçük bir bilgilendirme mesajı bulunmaktadır.



Seenekler kısmında oyuncu tercihine göre Açık veya Koyu temadan birini seçip oynayabilir.



Açık temanın görüntüsü de şu şekildedir:



Sorular kısmında bizi iki seçenek karşılıyor. Buradan soru ekleyip görüntüleyebiliyoruz.



Soruları görüntüleme kısmında, listeye veritabanındaki mevcut sorular yüklenir, sayısı görüntülenir. Ardındaki kod bloklarına geçmeden evvel, kontrollerden bahsedelim, soruları tutan bir listview, soruyu silip güncellememize yarayan butonlar, ve kategoriye göre soruları görüntülememizi sağlayan butonlar bulunuyor. Silmek için herhangi bir sorunun üzerine tıklayıp silmemiz yeterli.



```
private async void viewQuestions(string category)
{
    // ListView temizle
    lvSorular.Items.Clear();

    // Soruları veritabanından çek
    List<Question> questions = await QuestionService.GetAllQuestions();
    int count = 0;
    // Soruları ListView'e ekle
    if (category.Equals("all"))
    {
        foreach (Question question in questions)
        {
            ListViewItem item = new ListViewItem(question.Id.ToString());
            string[] options = Helper.SplitString(question.OptionsText);
            //item.SubItems.Add(question.Id.ToString());
            item.SubItems.Add(question.Category.Name);
            item.SubItems.Add(question.QuestionText);
            item.SubItems.Add(options[0]);
            item.SubItems.Add(options[1]);
            item.SubItems.Add(options[2]);
            item.SubItems.Add(options[3]);

            item.SubItems.Add(question.CorrectAnswerIndex.ToString());
            count++;
            lvSorular.Items.Add(item);
        }
    }
    else
    {
        foreach (Question question in questions)
        {
            if (question.Category.Name.Equals(category))
            {
                ListViewItem item = new ListViewItem(question.Id.ToString());
                string[] options = Helper.SplitString(question.OptionsText);
                //item.SubItems.Add(question.Id.ToString());
                item.SubItems.Add(question.Category.Name);
                item.SubItems.Add(question.QuestionText);
                item.SubItems.Add(options[0]);
                item.SubItems.Add(options[1]);
                item.SubItems.Add(options[2]);
                item.SubItems.Add(options[3]);

                item.SubItems.Add(question.CorrectAnswerIndex.ToString());
                count++;
            }
        }
    }
}
```

Soruları görüntüleme kısmında, metodumuz bir kategori parametresi alıyor. Bu parametre **"all"** olarak gönderilirse tüm soruları, eğer **"all"** değil de örneğin **"Sanat"** şeklinde gönderilirse, veritabanından çekilen sorulardan yalnızca **"Sanat"** kategorisine ait olan soruları görüntülüyoruz.

Daha detaylı incelersek, önce **QuestionService** yardımı ile tüm soruları getiriyoruz. Sonrasında ise kategori bilgisine bağlı olarak if else bloklarına girdikten sonra bir döngü ile tüm soruların üzerinde geziyoruz. Her bir soru için bir liste elemanı oluşturuyoruz ve ilk ID bilgisini soru ID si yapıyoruz. Sonrasında **Helper** yardımı ile cevaplar stringini ayırıp listenin cevap kısmına subitem olarak

ekliyoruz ve en son olarak da bu nesneyi listeye ekliyoruz.

En sonda da soru sayısını label'a basmayı unutmuyoruz.

Soru ekleme kısmında ise, sorunun içeriğini, şıkları, kategorisini ve doğru cevabı işaretledikten sonra **Soruyu Ekle** butonuna bastığımızda, soru veritabanına eklenmiş oluyor. Kodlarına bakarsak;



Önce aktif butonumuz hangisi bunu ayarlamamız gerekiyor.

```
#region Soru Ekleme Modülleri

int se_cat_index;
Button ekleme_aktif_buton;
5 references
private void selectButtons_Add(object sender, int index) {...}

1 reference
private async void btnSoruEkle_Click(object sender, EventArgs e) {...}

1 reference
private void ClearQuestionFields() {...}

#endregion
```

Herhangi bir butona tıklandığında çağrılan bu metot, önce Buton nesnesine cast yapıyor. Daha sonra da her kategori için atanan Index numarasına göre indexi güncelliyor. Bu kısımda her butonun gönderdiği Index farklı. Basılan butonun rengini değiştirip kullanıcıya geri bildirim sağlıyoruz ve aktif butonu da güncelliyoruz.

```
private void selectButtons_Add(object sender, int index)
{
    Button basilan_buton = (Button)sender;
    se_cat_index = index;

    if(ekleme_aktif_buton != null)
    {
        ekleme_aktif_buton.ForeColor = ThemeHandler.color_text;
    }
    basilan_buton.ForeColor = Color.Green;
    ekleme_aktif_buton = basilan_buton;
}
```

Soru ekleme kısmındaki kategori butonlarının gönderdiği Indexlerin örneklerini şu şekilde

```
#region Soru Eklemede Kategori Butonları

2 references
private void btnSporSE_Click(object sender, EventArgs e)
{
    selectButtons_Add(sender, 0);
}

1 reference
private void btnTarihSE_Click(object sender, EventArgs e)
{
    selectButtons_Add(sender, 1);
}

1 reference
private void btnSanatSE_Click(object sender, EventArgs e)
{
    selectButtons_Add(sender, 2);
}

1 reference
private void btnBilimSE_Click(object sender, EventArgs e)
{
    selectButtons_Add(sender, 3);
}

1 reference
private void btnEglenceSE_Click(object sender, EventArgs e)
{
    selectButtons_Add(sender, 4);
}

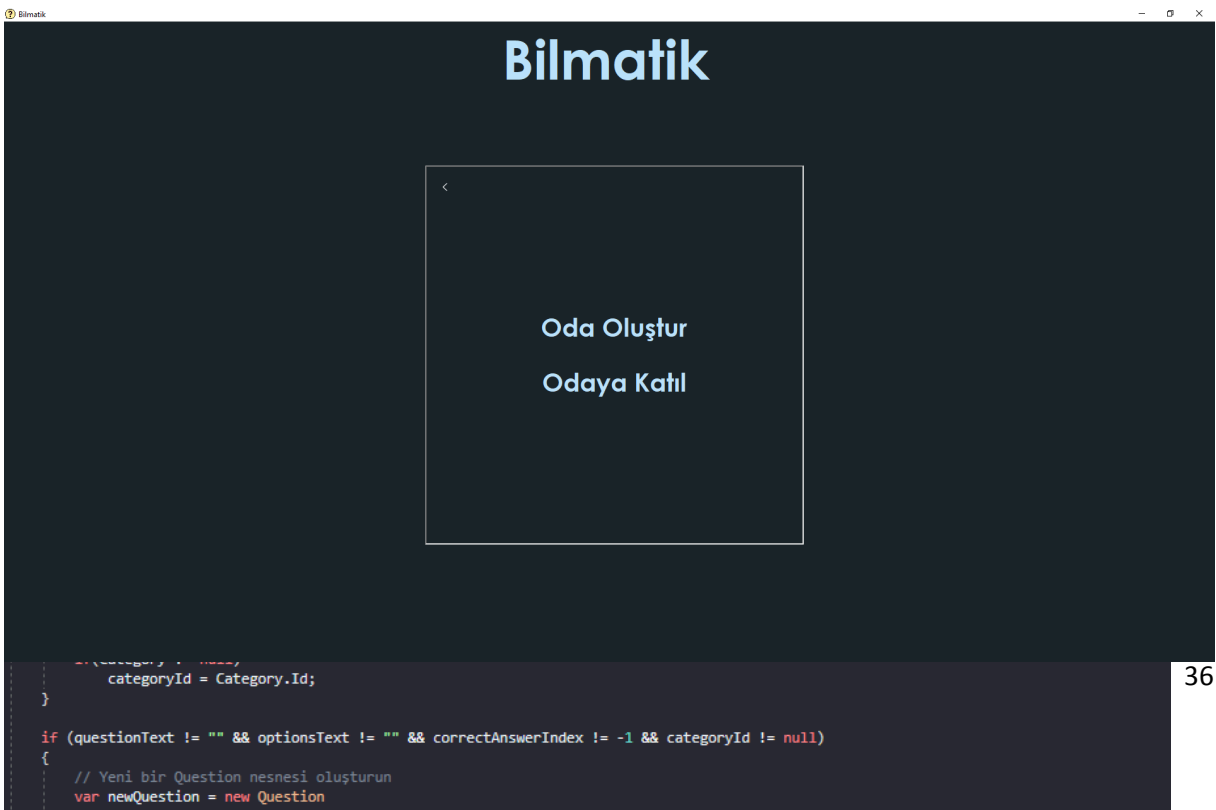
#endregion
```

görebilirsiniz:

Bu kısımda soru ekleme tuşuna tıklandığında olan olaylar görülmektedir.

Öncelikle sorunun veritabanına kaydedilmesi için gerekli olan alanları kullanıcıdan alıp bir değişkene atıyoruz. Daha sonra **context** yardımı ile girilen Index ile veritabanında eşleşen kategorinin ismini buluyoruz. Eğer ilgili alanlar boş ve geçersiz değilse, yeni bir soru nesnesi oluşturuyoruz. Sonrasında **QuestionService** içinde bulunan **AddQuestion** metodu ile soruyu veritabanına kaydediyoruz. Eğer başarılı ise olumlu, değilse olumsuz geri dönüş sağlıyoruz.

OYNA Butonu



Karşımıza oda kurup katılmak için iki seçenek çıkıyor. Oda oluşturmak istersek

Açılan panelde bizden bir **Oda ismi** ve **şifre** istiyor. İlgili alanları doldurduktan sonra olanlara bakalım.

```
private async void btnCAGCreateRoom_ClickAsync(object sender, EventArgs e)
{
    //Oda ismi ve şifrenin boş olup olmadığının kontrolü
    if (txtCAGRoomName.Text == "" || txtCAGRoomPassword.Text == "")
    {
        MessageBox.Show("Oda ismi veya şifreyi eksiksiz girdiğinize emin olun!", "Oda Oluşturma", MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }

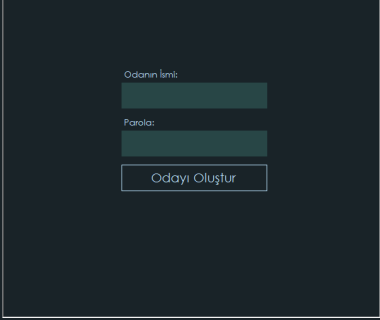
    User admin = UserSession.Instance.GetCurrentUser();

    //List<Category> categories = RoomSession.Instance.GetAllCategories();

    // Oda oluşturma işlemini yap
    var newRoom = new Room
    {
        Id = Guid.NewGuid(),
        Name = txtCAGRoomName.Text,
        Password = txtCAGRoomPassword.Text,
        Code = Helper.GenerateRoomCode(),
        AdminId = admin.Id,
    };

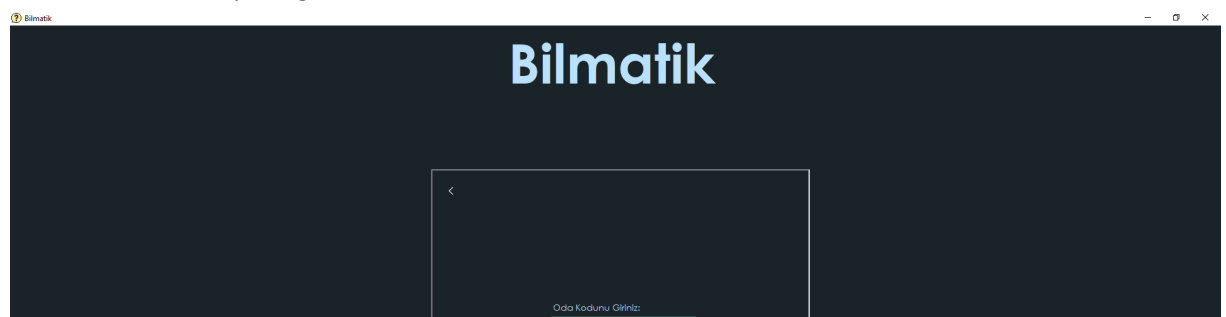
    //Servis başarılı sonuç döndüğünde kullanıcıyı bilgilendiriyoruz.
    bool isRoomCreated = await RoomService.CreateRoom(newRoom);

    if (isRoomCreated)
    {
        // Room oluşturulduğunda yapılacak işlemler
        isLeaving = true;
        LobbyGame game = new LobbyGame();
        game.Show();
        this.Close();
    }
    else
    {
        MessageBox.Show("Oda oluşturulurken bir hata oluştu.");
    }
}
```



İlk olarak alanların boş olup olmadığını handle ediyoruz. Daha sonra admini, odayı kuran kişi ayarlayabilmek için mevcut kullanıcının ID bilgisini alıyoruz. Sonrasında yeni bir oda nesnesi oluşturup, **RoomService** içerisindeki **CreateRoom** modülüne bu yeni oda nesnesini gönderiyoruz. Başarılı olursa bu formu kapatıp Lobi formunu açıyoruz, başarısız olursa da kullanıcıya bir geri bildirim sağlıyoruz. Bu kısımdaki **isLeaving** değişkeni, formun bilinçli bir şekilde mi yoksa sağ üstteki kapat butonundan mı kapatıldığını handle etmek için kullanılıyor.

Odaya katılma panelinde, kullanıcılardan bir oda kodu ve parolası girmesini istiyoruz. **Odaya Katıl** butonu ile olan olaylara göz atalım.



```

private async void btnJAGJoinRoom_ClickAsync(object sender, EventArgs e)
{
    string roomCode = txtJoinCode.Text;
    string roomPassword = txtJoinPassword.Text;
    User currentUser = UserSession.Instance.GetCurrentUser();
    Room curr_room = RoomSession.Instance.GetCurrentRoom();

    using (var context = new qAppDBContext())
    {
        var room = await context.Rooms.FirstOrDefaultAsync(r => r.Code == roomCode && r.Password == roomPassword);

        if(room.CurrentGameId != null)
        {
            MessageBox.Show("Oda da bir oyun oynanıyor", "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        bool joined = await RoomService.JoinRoom(roomCode, roomPassword, currentUser);

        if (joined)
        {
            isLeaving = true;
            //MessageBox.Show("Odaya katılma işlemi başarılı.", "Bilgi", MessageBoxButtons.OK, MessageBoxIcon.Information);
            LobbyGame game = new LobbyGame();
            game.Show();
            this.Close();
        }
        else
        {
            MessageBox.Show("Odaya katılma işlemi başarısız.", "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);
            // Odaya katılma işlemi başarısız olduğunda yapılacak işlemler
        }
    }
}

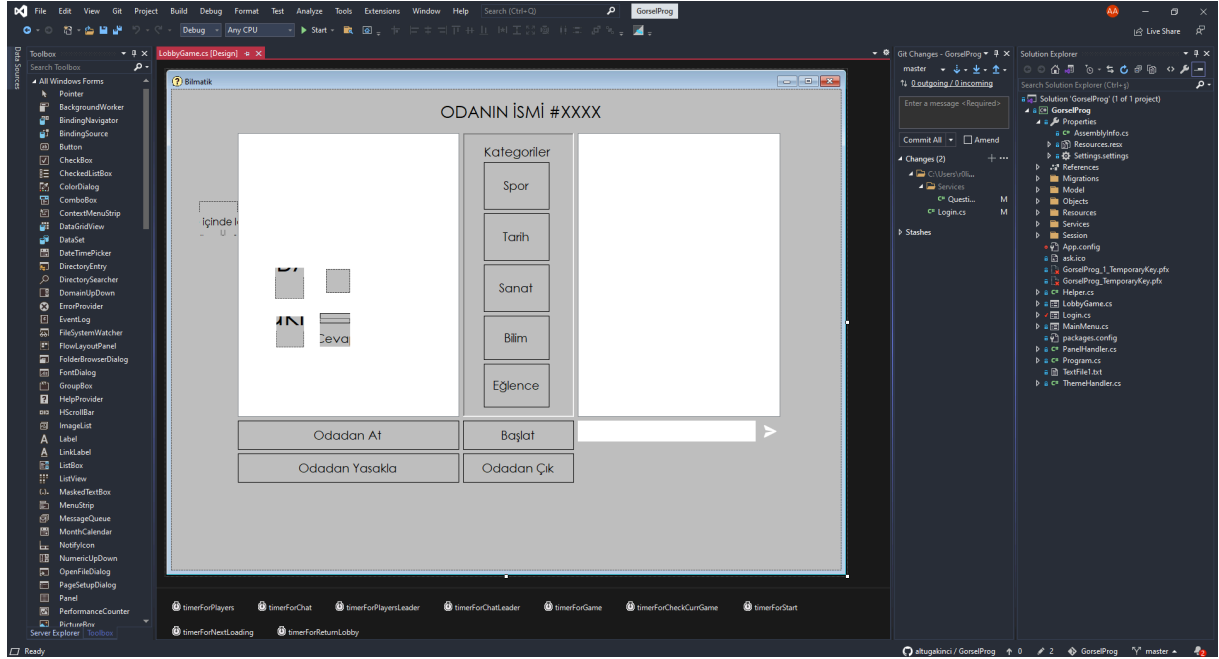
```

İlgili alanları değişkenlere atıyoruz. Daha sonrasında mevcut kullanıcı ve oda bilgisini de aynı şekilde çekiyoruz. **Context** yardımı ile girilen kod ve şifreye uygun bir oda var mı diye veritabanından kontrol ediyoruz. Eğer odada mevcut bir oyun oynanıyorsa kullanıcıyı odaya almıyoruz. Daha sonrasında **RoomService** servisinin **JoinRoom** modülü sayesinde odaya katılmasını sağlıyoruz. Başarılı olursa Lobi formunu yüklüyoruz. Eğer bir sorun olursa da ona göre bir geri dönüş sağlıyoruz.

MainMenu formu da genel hatlarıyla bu şekildeydi. Şimdi de asıl lobiye ve oyuna bakalım.

3.3 LobbyGame

LobbyGame, genel olarak lobinin oluşturulduğu, oyunun oynandığı form olarak tanımlanabilir.



Bu kısımda odayı oluşturan mı yoksa katılan bir oyuncu mu olduğunuza göre bir panel yükleniyor. Oyuncu ve mesaj listesini güncelleyen sayaçlar da buna göre başlatılıyor. Buradaki timerlardan kısaca bahsedecek olursak, **timerForPlayers** oyuncuların oyuncu listesini, **timerForChat** oyuncuların mesaj listesini, **timerForPlayersLeader** liderin oyuncu listesini, **timerForChatLeader** liderin mesaj listesini yeniler. Aslında hepsini tek timer üzerinde toplayabilirdik fakat o şekilde daha fazla sorun yaşayacağımızı düşündüğümüz için böyle bir çözüme gittik. **timerForCheckCurrGame**, adminin oyunu başlatıp başlatmadığının anlık olarak oyuncular tarafında kontrolünü yapan timer. **timerForStart**, oyun başlamadan hemen önce “Yarışma Başlıyor” panelinin sayacı, **timerForGame**, oyundaki her bir soru için verilen süreyi tutan sayaç, **timerForNextLoading**, her sorudan sonra bir sonraki soru yüklenirken görünen panelin sayacı, **timerForReturnLobby**, oyun bittikten sonra lobiye dönüş için başlatılan sayaç olarak tanımlayabiliriz.

```

private void LobbyGame_Load(object sender, EventArgs e)
{
    this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.Form1_FormClosing);

    //Lobinin yukarisinda oda ismi ve kodun görüntülenmesini sağlıyor.
    string room_code = RoomSession.Instance.GetCurrentRoom().Code;
    string room_name = RoomSession.Instance.GetCurrentRoom().Name;
    lblPlayerRoomName.Text = $"{room_name} #{room_code}";
    lblLeaderRoomName.Text = $"{room_name} #{room_code}";

    //Adminin veya kullanıcının lobiye oluşturmaya göre hangi refresh timerlarının başlatılacağı.
    User currentuser = UserSession.Instance.GetCurrentUser();
    Room room = RoomSession.Instance.GetCurrentRoom();
    if (currentuser.Id.Equals(room.AdminId))
    {
        active_panel = pnlLobbyLeader;
        PanelHandler.setPanelFill(active_panel, pnlLobbyLeader);
        active_panel = pnlLobbyLeader;

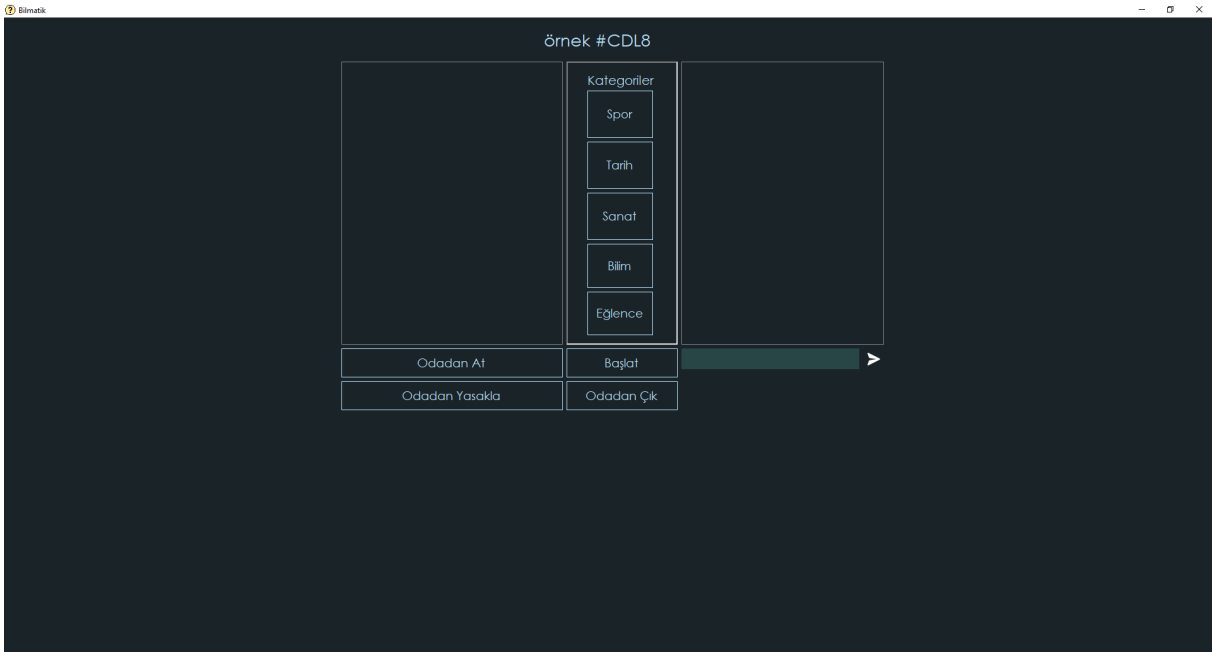
        timerForChatLeader.Start();
        timerForPlayersLeader.Start();
    }
    else
    {
        active_panel = pnlLobbyPlayer;
        PanelHandler.setPanelFill(active_panel, pnlLobbyPlayer);
        active_panel = pnlLobbyPlayer;

        timerForChat.Start();
        timerForPlayers.Start();
        timerForCheckCurrGame.Start();
    }

    //Tüm forma geçerli temanın uygulanmasını sağlıyor.
    ThemeHandler.changeFormsColor(this);
    ThemeHandler.changeAllControlsColor(this);
    this.WindowState = FormWindowState.Maximized;
}

```

Form yüklendiği zaman olan olaylara bakacak olursak, önce oda ismi ve kodu bir değişkene atılıp bunları ekrana basıyoruz. Sonrasında **Session**'lar yardımı ile kullanıcı ve oda bilgisini alıyoruz. Eğer formu yükleyen kişi bir admin ise bu kişinin önüne lider paneli geliyor. Eğer değilse, oyuncu paneli geliyor. Liderse kendi timerları, oyuncu ise de kendi timerları aktif hale getiriliyor. Bu yeni bir form olduğu için temanın yüklenmesi de önemli bir detay bizim için.



Liderin gözünden oda bu şekilde, ortada kategorileri seçebiliyoruz. Kategoriler tıklandığında toggle oluyor ve kategori indexini ilgili servise gönderiyoruz. Onun dışında birini atmak için oyuncu listesinden herhangi birinin ismine tıklayıp odadan at veya yasakla komutlarını kullanabiliyoruz. Öncelikle bunların kod kısmını gösterelim. Daha sonradan oyuna geçebiliriz.


```
private void btnLeaderSpor_Click(object sender, EventArgs e)
{
    toggleButtons(sender, 0);
}

//Bilim kategorisi seçildiğinde 1 indexini togglebuttons'a gönderiyor.
1 reference
private void btnLeaderBilim_Click(object sender, EventArgs e)
{
    toggleButtons(sender, 1);
}

//Tarih kategorisi seçildiğinde 2 indexini togglebuttons'a gönderiyor.
1 reference
private void btnLeaderTarih_Click(object sender, EventArgs e)
{
    toggleButtons(sender, 2);
}

//Sanat kategorisi seçildiğinde 3 indexini togglebuttons'a gönderiyor.
1 reference
private void btnLeaderSanat_Click(object sender, EventArgs e)
{
    toggleButtons(sender, 3);
}

//Eğlence kategorisi seçildiğinde 4 indexini togglebuttons'a gönderiyor.
1 reference
private void btnLeaderEglence_Click(object sender, EventArgs e)
{
    toggleButtons(sender, 4);
}
```

Butonlara tıklandığında **toggleButtons** isimli modülümüze butonun referansı ve indexini gönderiyoruz. Bu metod da bizim için gerekli işlemleri yapıyor. Metodu görelim.

```
private void toggleButtons(object sender, int buttonIndex)
{
    Button button = (Button)sender;

    if (button.ForeColor != Color.Green)
    {
        button.ForeColor = Color.Green;
        Helper.AddSelectedCategory(buttonIndex);
    }
    else
    {
        button.ForeColor = ThemeHandler.color_texts;
        Helper.RemoveSelectedCategory(buttonIndex);
    }
}
```

Helper kısmında bahsettiğimiz **AddSelectedCategory** metodu burada işimize yaramakta.

Mesaj kısmına bakacak olursak

```
private async void sendMsgLeader()
{
    if (txtLeaderMsg.Text != "")
    {
        User current = UserSession.Instance.GetCurrentUser();
        Room room = RoomSession.Instance.GetCurrentRoom();
        string message = txtLeaderMsg.Text;
        txtLeaderMsg.Clear();
        await MessageService.SendMessageAsync(current.Id, message, room.Id);
    }
}
```

Liderin mesaj gönderme butonuna basıldığında eğer mesaj kutucuğu boş değilse, mevcut kullanıcı ve oda oturum bilgilerinden çekilir, içerik değişkene atılır ve **MessageService** içerisindeki mesaj gönderme metoduna ilgili bilgiler gönderilir.

```
private async void btnLeaderKick_Click(object sender, EventArgs e)
{
    // kick the player
    Room curr_room = RoomSession.Instance.GetCurrentRoom();
    Guid userId = Guid.Parse(lvLeaderPlayers.SelectedItems[0].SubItems[0].Text);
    await RoomService.KickUser(curr_room.Id, userId);
}
```

Oyuncu atma işleminde, yine oturum bilgilerinden odanın bilgisi alınır ve atılacak kullanıcının ID'si, ListView kontrolünde seçili olan elemanların ilkinin, ilk sütun bilgisinden çekilir. Bu da tam olarak string tipindeki ID'ye denk geliyor. Bunu Guid castleyip göndermemiz gerekiyor. Bundan sonra da daha önceden konuştuğumuz **KickUser** metodunu çağırıyoruz.

```
private async void btnLeaderBan_Click(object sender, EventArgs e)
{
    // Ban the player
    Room curr_room = RoomSession.Instance.GetCurrentRoom();
    Guid userId = Guid.Parse(lvLeaderPlayers.SelectedItems[0].SubItems[0].Text);
    User admin = UserSession.Instance.GetCurrentUser();
    await RoomService.BanUser(userId, curr_room.Id, admin.Id);
}
```

Yasaklamada da benzer bir mantıkla **BanUser** metodunu çağırıyoruz.

Bu bölümün en önemli yapı taşlarından olan sayaçların da arkasındaki mantığa baktıktan sonra oyuna geçebiliriz.

```
private async void timerForPlayersLeader_Tick(object sender, EventArgs e)
{
    lvLeaderPlayers.Items.Clear();
    Room room = RoomSession.Instance.GetCurrentRoom();
    if (room == null)
    {
        stopAllTimers();
        closeCounter++;
        this.Close();
        return;
    }
    List<User> players = await RoomService.GetPlayers(room.Id);
    foreach (User u in players)
    {
        string guid = u.Id.ToString();
        string username = u.UserName;
        ListViewItem item = new ListViewItem(guid);
        item.SubItems.Add(username);
        lvLeaderPlayers.Items.Add(item);
    }
    lvLeaderPlayers.AutoResizeColumn(1, ColumnHeaderAutoResizeStyle.ColumnContent);
}
```

Burada örnek olarak liderin oyuncular listesinin yenilenmesini ele alalım. Her şeyden önce veri kopyalanmasının önüne geçmek amaçlı liste temizleniyor. Sonra da oda oturumu bilgisi çekildikten sonra önce bir room var mı diye kontrol ediyoruz. Neden bunu yapıyoruz? Çünkü eğer herhangi bir biçimde oda kapanırsa, timerların arka planda kaynakları kullanmasını istemeyiz. Aynı zamanda lider odadan çıktığında da diğer oyuncuları da ana menüye döndürebilmemiz için bu kontrolü yapmalıyız. Sonrasında oyuncu listesini servis yardımıyla çekiyoruz ve her bir oyuncuyu listeye bir item olarak ekliyoruz. En sonda da en uzun isimli kullanıcıya göre bir otomatik genişletme işlemi yapıyoruz.

Oyun başlangıcı

```
int game_timer = 0;
int time_for_question = 10;
int[] remaining_seconds = new int[10];
int question_index;
List<Question> question_list;
Question current_question;
string[] options;
```

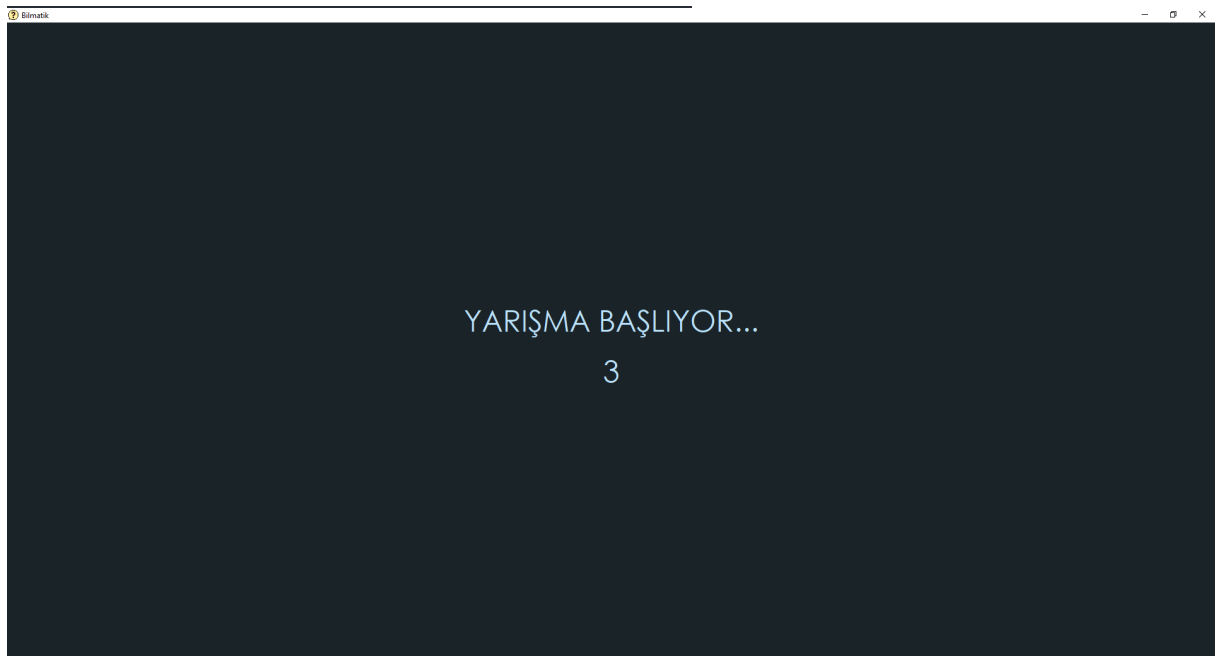
Oyun için genel olarak kullanmamız gereken değişkenlerden **game_timer** oyunu başlatan sayacın ekrana basılmasını, **time_for_question** soru başına olan sürenin ekrana basılmasını, **remaining_seconds** isimli dizi de şu anda bir modül olarak duran ve oyuncuların doğru cevap verdiği sorularda artırdıkları süreye göre bonus puan kazanabilecekleri bir sisteme dönüşecek bir yapı, **question_index** kaçınıcı soruda olduğumuzu tutan değer, **question_list** soru listesi, bu liste oyun başladığında doldurulacak. **current_question** değişkeni, o sorunun cevap, doğru cevap indexi gibi bilgilerine erişebilmemiz için tutuluyor. **Options** ise cevap şıklarını ekrana basabilmemizi sağlamak için bulunmakta.

```
private async void startGame()
{
    //Roomsessionda tutulan önbellekteki seçilen kategorileri getiriyoruz.
    List<Category> categories = RoomSession.Instance.GetSelectedCategories();
    Room room = RoomSession.Instance.GetCurrentRoom();

    //İlgili servisin oyunu başlatması ve database'e kaydetmesi.
    var result = await GameService.StartGame(room.Id, categories, DateTime.Now, DateTime.Now.AddMinutes(10));

    if (result != null) //Oyunu başlatma başarılı
    {
        //MessageBox.Show("Oyun başladı!");
        question_list = GameSession.Instance.GetAllQuestions();
        question_index = 0;
    }
    else //Oyun başlatılamadı.
    {
        MessageBox.Show("Oyun başlatılamadı.");
    }
}
}
```

Oyunu başlatan modül, kategorileri getirir, odayı çeker ve ilgili servisin ilgili metodunu çağırır. Eğer başarılı bir şekilde başlarsa, liste doldurulur ve index=0 a ayarlanır. Oyun başlamış olur.



Bu sayaç işini bitirdikten sonra, dizideki ilk soru çekilir ve ekrana basılır.

```
private async void printQuestion()
{
    resetAllOptionButtons();
    if (question_index == GameSession.Instance.GetAllQuestions().Count)
    {
        getSummary();
        return;
    }
    PanelHandler.setPanelFill(active_panel, pnlGame);
    active_panel = pnlGame;

    //Soruyu çekiyoruz.
    current_question = question_list[question_index];

    string question_text = current_question.QuestionText;
    options = Helper.SplitString(current_question.OptionsText);
    int correct_ans_index = current_question.CorrectAnswerIndex;

    lblSoru.Text = question_text;
    btnOption1.Text = options[0];
    btnOption2.Text = options[1];
    btnOption3.Text = options[2];
    btnOption4.Text = options[3];

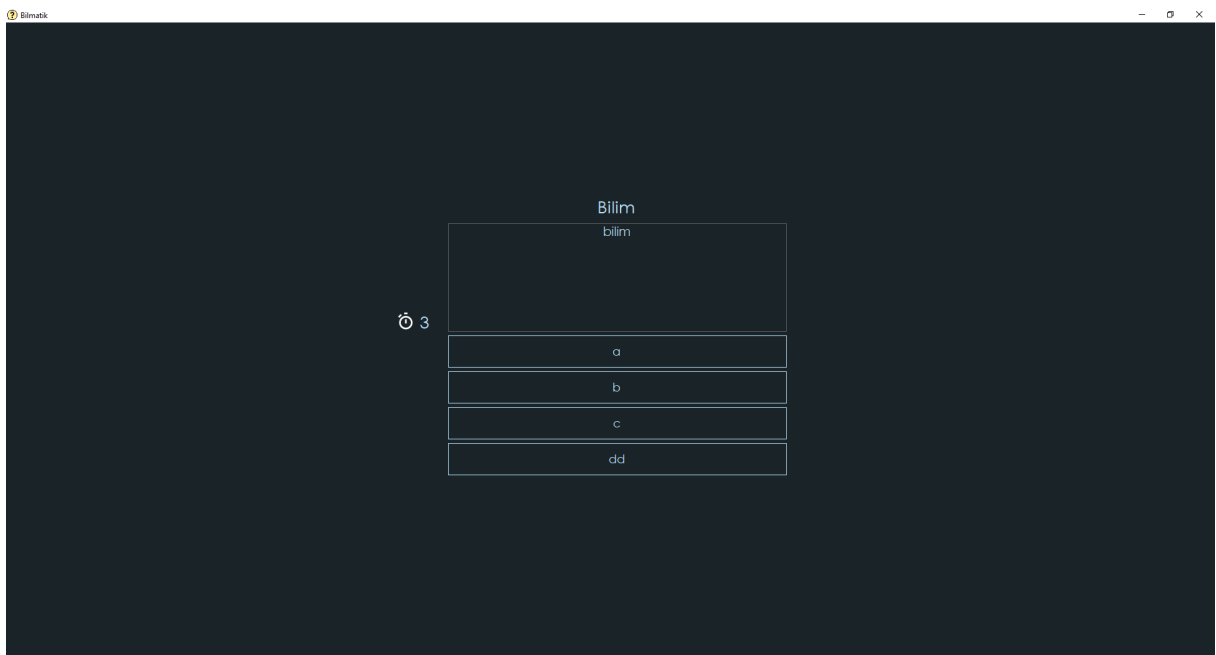
    Category curr_question_category = null;

    using (var db = new qAppDBContext())
    {
        var Category = await db.Categories.FirstOrDefaultAsync(c => c.Id == current_question.CategoryId);
        if (Category != null)
            curr_question_category = Category;
    }

    lblCategory.Text = curr_question_category.Name.ToString();

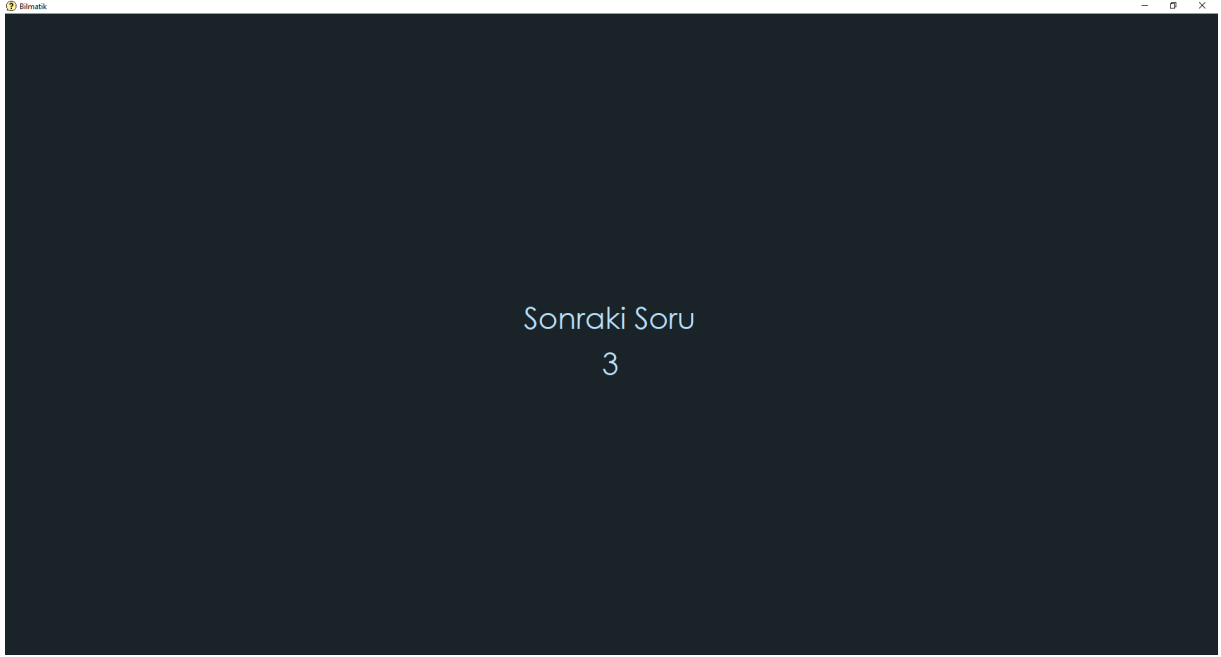
    question_index++;
    timerForGame.Start();
}
```

Soruyu ekrana basan metodumuz, önce tüm seçili şıkları temizler. Sonrasında oyunun bitebilmesi için bir if bloğuna girer. Biz burada **.take(10)** diyerek **StartGame** metodunda 10 soru çektiğimiz için, doğal olarak bu if de 10 sorudan sonra içeri girip **summary** kısmını önümüze getirecektir. Soru basıldığında panel önümüze gelir, mevcut soru, diziden çekilir ve ilgili alanları da değişkenlere alınır, sonrasında labellar güncellenir. Sorunun kategorisini bulup basmak için de geçici bir context kullanıyoruz. Daha sonrasında indexi artırıp sayacı başlatıyoruz.



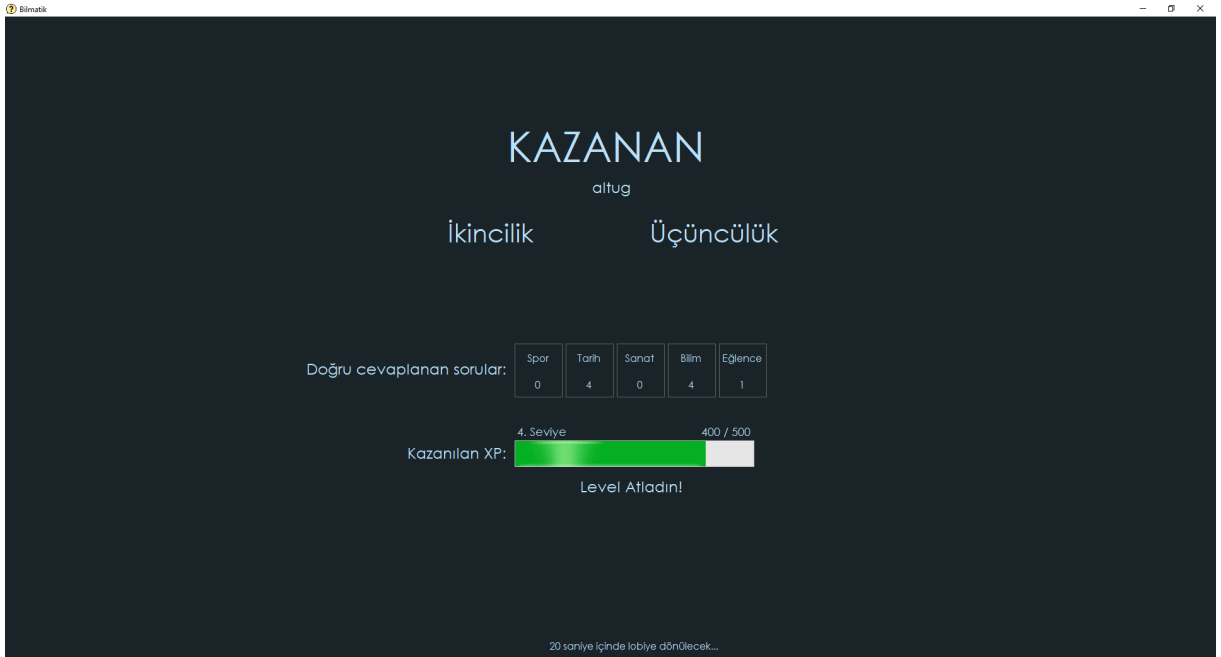
Soru ekranı da görüldüğü gibidir.

Sayaç bittiğinde;



Sonraki soru paneli bizi karşılıyor. 3' den geriye sayıyor ve bir sonraki soruyu basıyor. Bu döngü index, çekilen soru sayısına eşitlenene kadar devam ediyor.

Son olarak da özet bölümü geliyor.



Bu kısımda kazananı, ikinci ve üçüncü olan kişileri, hangi kategoriden kaç doğru cevapladığımızı, kazandığımız xp yi ve level atlayıp atlamadığımızı görüntüleyebiliyoruz. En altta da lobiye dönmek için kaç saniyemiz kaldığını ekrana basıyoruz.

```

private async void getSummary()
{
    User curr_user = UserSession.Instance.GetCurrentUser();
    Game curr_game = GameSession.Instance.GetCurrentGame();
    Room curr_room = RoomSession.Instance.GetCurrentRoom();

    //db ile bağlantı kurup özeti önbelleğe alıyoruz.
    var summary = await GameService.GetSummaryGame(curr_game.Id, curr_user.Id, curr_room.Id);

    if(summary != null)
    {
        if(summary.FirstUser != null)
            lblSumWinnerName.Text = summary.FirstUser.UserName;

        if(summary.SecondUser != null)
            lblSumSecondName.Text = summary.SecondUser.UserName;

        if(summary.ThirdUser != null)
            lblSumThirdName.Text = summary.ThirdUser.UserName;

        lblSumSpor.Text = summary.Category1Correct.ToString();
        lblSumTarih.Text = summary.Category2Correct.ToString();
        lblSumSanat.Text = summary.Category3Correct.ToString();
        lblSumBilim.Text = summary.Category4Correct.ToString();
        lblSumEglenme.Text = summary.Category5Correct.ToString();

        lblSumLevel.Text = $"{summary.Level}. Seviye";
        lblSumXP.Text = $"{summary.SumXP} / 500";
        prgSumXP.Value = summary.SumXP;

        if (summary.isLevelUp)
            lblSumLevelUp.Visible = true;
        else
            lblSumLevelUp.Visible = false;
    }
    else
    {
        MessageBox.Show("Bir hata oluştu.");
    }

    PanelHandler.setPanelFill(active_panel, pnlSum);
    active_panel = pnlSum;
    //Oynanan oyunu kapatıyor, diğer oyuncular otomatik yeni bir oyuna başlamasın.
    Helper.ClearGameSession();
    //Özetten belli bir süre sonra lobiye dönülmesi gerekiyor.
    timerForReturnLobby.Start();
}

```

Özeti getirme kısmında, oturum bilgilerinin hepsi alınıyor. Özet bilgisini de çekip bir nesneye atıyoruz. İlk oyuncu, ikinci oyuncu, üçüncü oyuncu bilgisini alıp ilgili labellara basıyoruz. Doğru yanıt verilen kategorileri de aynı şekilde level ve xp ile basıyoruz. Eğer level artışı varsa labelı gösteriyoruz. Daha sonra da paneli getiriyoruz ve lobiye dönüş sayacını başlatıyoruz. En son da tekrar oyun oynamaya hazır bir şekilde lobi önümüze geliyor.

4. SON SÖZ

Proje detay hatlarıyla bu şekildeydi. Yaklaşık 3 ay süren bir geliştirme süreci geçirdik. Her modülü teker teker araştırıp öğrenip yazmaya ve birbiri ile uyumlu bir şekilde birleştirmeye çalıştık. Projede emeği geçen ekip arkadaşlarımıza ve hocamız Koray Aki'ye teşekkür ediyoruz.

CV'LER

Ahmet Serhat İLGİN

İbrahim YEGEN

Altuğ AKINCI

Sefa ÖNDER

Karahan TÜRKER