ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

(BLM4531-A) Ağ Tabanlı Teknolojiler ve Uygulamaları

Öznur KANDAKOĞLU

19290250

Öğr. Gör. Enver BAĞCI

01,2023

ÖZET

Bu rapor, Ankara Üniversitesi Bilgisayar Mühendisliği (BLM4531-A) Ağ Tabanlı Teknolojiler Ve Uygulamaları dersi için hazırlanan uygulama hakkında bilgileri kapsamaktadır.

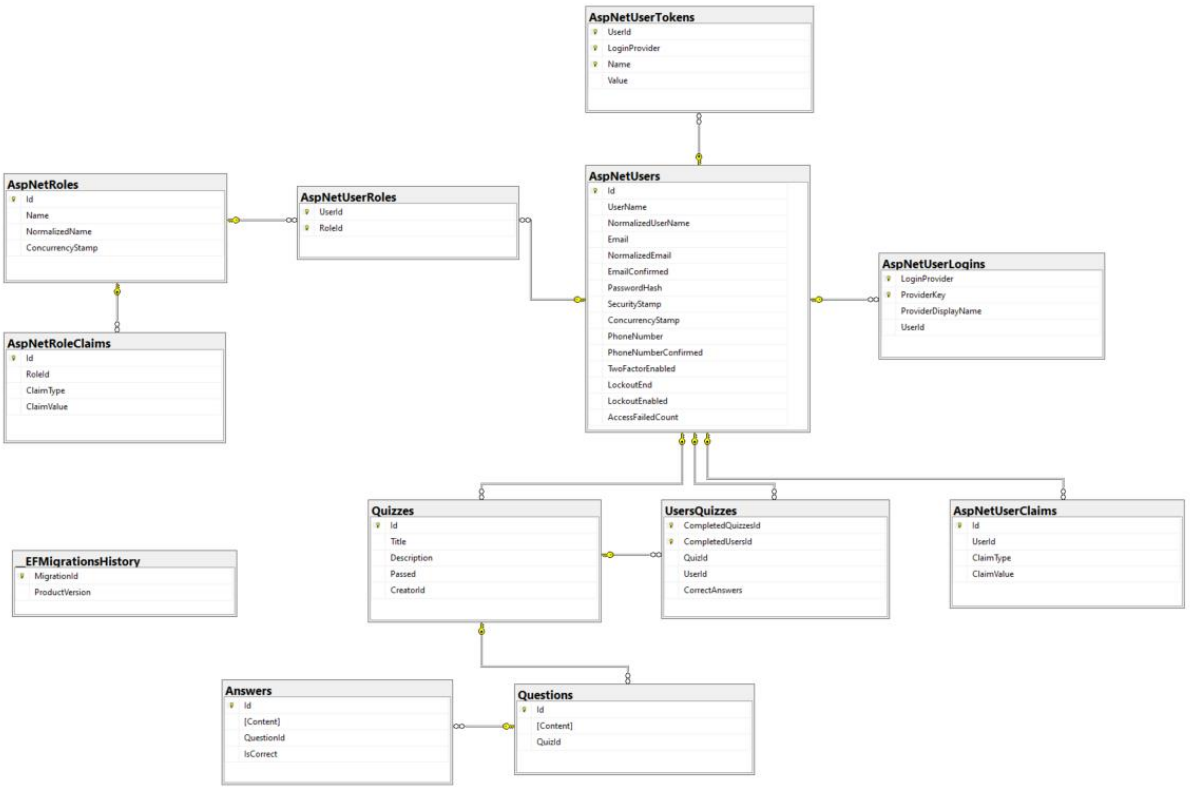
Quiz uygulamaları, kullanıcıların bilgi seviyelerini test etmelerine ve öğrenmelerine olanak tanır. Uygulama içerisinde kullanıcılar sorulara cevap verir ve doğru cevap verdiklerinde puan kazanırlar. Genellikle belirli bir konuda özelleştirilmiş sorular bulunur ve kullanıcılar konuya göre öğrenirler. Örneğin, geliştirmiş olduğum uygulamada kültür, çevre, insan hakları, spor gibi konular içeren soru quizleri oluşturulmuştur.

Uygulamanın amacı; yukarıda yazılmış metin doğrultusunda kullanıcıların bilgi seviyelerini test etmeleri, öğrenirken aynı zamanda eğlenmeleri gibi unsurları içermektedir. Geliştirilen uygulama dört farklı kategoride soru çeşitliği içermektedir. Kullanıcı her kategoride kendini test edebilmektedir.

Uygulamanın kullanımı, arayüzü, tasarımı ve kodlaması raporun devamında detaylı olarak anlatılmıştır.

1. PROJE TANITIMI ve VERİ TABANI

Projede ASP .NET kullanılarak API ve ön yüz kısımları geliştirilmiş, veri tabanında bilgilerin saklanması için Microsoft SQL Server kullanılmıştır. API ile veri tabanı arasındaki iletişim kuralları için Entity Framework kullanılmıştır. Şekil 1.1'de ise bu veri tabanı gösterilmiştir. Veri tabanında Quizzes, Questions, Answers, Users, Roles gibi birçok tablo oluşturulmuştur. Oluşturulan tablolar arasındaki ilişkiler foreign key'lerle sağlanmış ve birbirlerine bağlanmıştır.



Şekil 1.1 Proje Veri Tabanı

1.1 Table Oluşturulması

```
migrationBuilder.CreateTable(
    name: "Quizzes",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Title = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Description = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Passed = table.Column<int>(type: "int", nullable: false),
        CreatorId = table.Column<int>(type: "int", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Quizzes", x => x.Id);
        table.ForeignKey(
            name: "FK_Quizzes_AspNetUsers_CreatorId",
            column: x => x.CreatorId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
```

```
migrationBuilder.CreateTable(
    name: "Questions",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Content = table.Column<string>(type: "nvarchar(max)", nullable: false),
        QuizId = table.Column<int>(type: "int", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Questions", x => x.Id);
        table.ForeignKey(
            name: "FK_Questions_Quizzes_QuizId",
            column: x => x.QuizId,
            principalTable: "Quizzes",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
```

```
migrationBuilder.CreateTable(
    name: "Answers",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Content = table.Column<string>(type: "nvarchar(max)", nullable: false),
        QuestionId = table.Column<int>(type: "int", nullable: false),
        IsCorrect = table.Column<bool>(type: "bit", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Answers", x => x.Id);
        table.ForeignKey(
            name: "FK_Answers_Questions_QuestionId",
            column: x => x.QuestionId,
            principalTable: "Questions",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
```

```
migrationBuilder.CreateTable(
    name: "AspNetUserTokens",
    columns: table => new
    {
        UserId = table.Column<int>(type: "int", nullable: false),
        LoginProvider = table.Column<string>(type: "nvarchar(450)", nullable: false),
        Name = table.Column<string>(type: "nvarchar(450)", nullable: false),
        Value = table.Column<string>(type: "nvarchar(max)", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_AspNetUserTokens", x => new { x.UserId, x.LoginProvider, x.Name });
        table.ForeignKey(
            name: "FK_AspNetUserTokens_AspNetUsers_UserId",
            column: x => x.UserId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
```

2. PROJE MİMARİSİ

2.1 Modeller

Projede yedi farklı model kullanılmıştır. Bu modeller sırasıyla LoginModel, RegisterModel, User, UsersQuizzes, Answer, Question ve Quiz olarak aşağıda verilmiştir.

```
3 başvuru
public class LoginModel
{

    2 başvuru
    public string Email { get; set; } = string.Empty;
    1 başvuru
    public string Password { get; set; } = string.Empty;
}
```

```
3 başvuru
public class RegisterModel
{

    2 başvuru
    public string Password { get; set; } = string.Empty;
    1 başvuru
    public string RepeatPassowrd { get; set; } = string.Empty;
    2 başvuru
    public string Email { get; set; } = string.Empty;
}
```

```
16 başvuru
public class User
{
    0 başvuru
    public int Id { get; set; }
    0 başvuru
    public string Username { get; set; } = string.Empty;
    0 başvuru
    public string Email { get; set; } = string.Empty;
}
```

```
7 başvuru
public class UsersQuizzes
{
    0 başvuru
    public int CompletedUsersId { get; set; }
    0 başvuru
    public int CompletedQuizzesId { get; set; }
    0 başvuru
    public int CorrectAnswers { get; set; }
    1 başvuru
    public int MaxScore { get; set; }
}
```

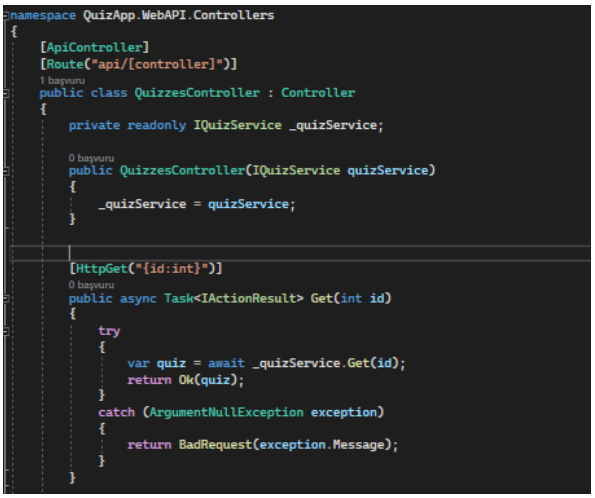
```
7 başvuru
public class Answer
{
    0 başvuru
    public int Id { get; set; }
    0 başvuru
    public string Content { get; set; } = string.Empty;
    0 başvuru
    public int QuestionId { get; set; }
    0 başvuru
    public bool IsCorrect { get; set; } = false;
}
```

```
6 başvuru
public class Question
{
    0 başvuru
    public int Id { get; set; }
    0 başvuru
    public string Content { get; set; } = string.Empty;
    0 başvuru
    public int QuizId { get; set; }
    1 başvuru
    public IEnumerable<Answer> Options { get; set; } = new List<Answer>();
}
```

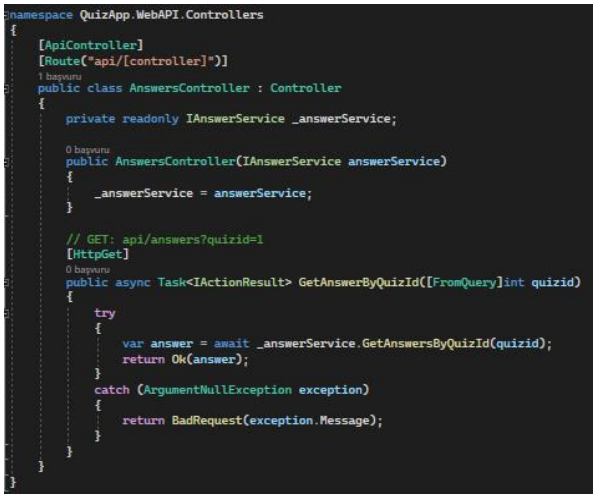
```
17 başvuru
public class Quiz
{
    0 başvuru
    public int Id { get; set; }
    0 başvuru
    public string Title { get; set; } = string.Empty;
    0 başvuru
    public string Description { get; set; } = string.Empty;
    2 başvuru
    public int CreatorId { get; set; }
    4 başvuru
    public User? Author { get; set; }
    0 başvuru
    public int Passed { get; set; } = 0;
    2 başvuru
    public List<Question> Questions { get; set; } = new List<Question>();
}
```

2.2 Controllers

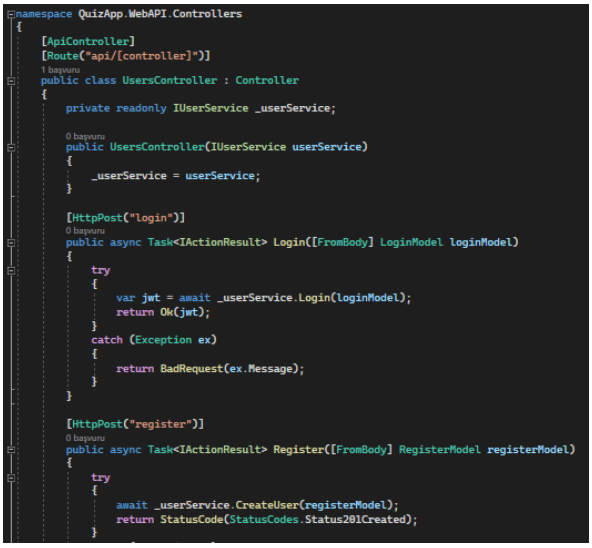
Projede dört farklı controller class'ı vardır. Bu controller'lar sırasıyla; UsersController, AnswersController, QuestionsController ve QuizzesController olarak yer almıştır.



```
namespace QuizApp.WebAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 başvuru
    public class QuizzesController : Controller
    {
        private readonly IQuizService _quizService;

        0 başvuru
        public QuizzesController(IQuizService quizService)
        {
            _quizService = quizService;
        }

        [HttpGet("{id:int}")]
        0 başvuru
        public async Task<IActionResult> Get(int id)
        {
            try
            {
                var quiz = await _quizService.Get(id);
                return Ok(quiz);
            }
            catch (ArgumentNullException exception)
            {
                return BadRequest(exception.Message);
            }
        }
```



```
namespace QuizApp.WebAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 başvuru
    public class AnswersController : Controller
    {
        private readonly IAnswerService _answerService;

        0 başvuru
        public AnswersController(IAnswerService answerService)
        {
            _answerService = answerService;
        }

        // GET: api/answers?quizid=1
        [HttpGet]
        0 başvuru
        public async Task<IActionResult> GetAnswerByQuizId([FromQuery]int quizid)
        {
            try
            {
                var answer = await _answerService.GetAnswersByQuizId(quizid);
                return Ok(answer);
            }
            catch (ArgumentNullException exception)
            {
                return BadRequest(exception.Message);
            }
        }
    }
}
```



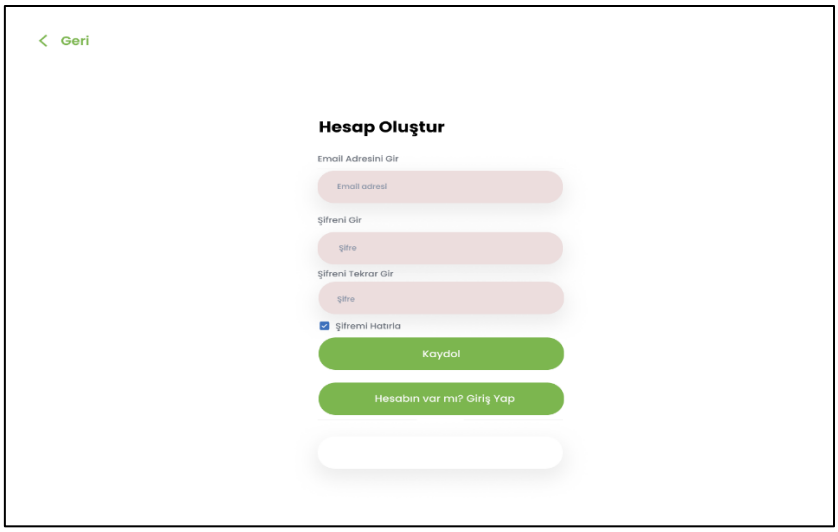
```
namespace QuizApp.WebAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 başvuru
    public class UsersController : Controller
    {
        private readonly IUserService _userService;

        0 başvuru
        public UsersController(IUserService userService)
        {
            _userService = userService;
        }

        [HttpPost("login")]
        0 başvuru
        public async Task<IActionResult> Login([FromBody] LoginModel loginModel)
        {
            try
            {
                var jwt = await _userService.Login(loginModel);
                return Ok(jwt);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpPost("register")]
        0 başvuru
        public async Task<IActionResult> Register([FromBody] RegisterModel registerModel)
        {
            try
            {
                await _userService.CreateUser(registerModel);
                return StatusCode(StatusCodes.Status201Created);
            }
```

```
namespace QuizApp.WebAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 başvuru
    public class QuestionsController : Controller
    {
        private readonly IQuestionService _questionService;

        0 başvuru
        public QuestionsController(IQuestionService questionService)
        {
            _questionService = questionService;
        }

        // GET: api/questions?quizid=1
        [HttpGet]
        0 başvuru
        public async Task<IActionResult> GetByQuizId([FromQuery]int quizid)
        {
            try
            {
                var question = await _questionService.GetQuestionsByQuizId(quizid);
                return Ok(question);
            }
            catch (ArgumentNullException excepion)
            {
                return BadRequest(excepion.Message);
            }
        }
    }
}
```

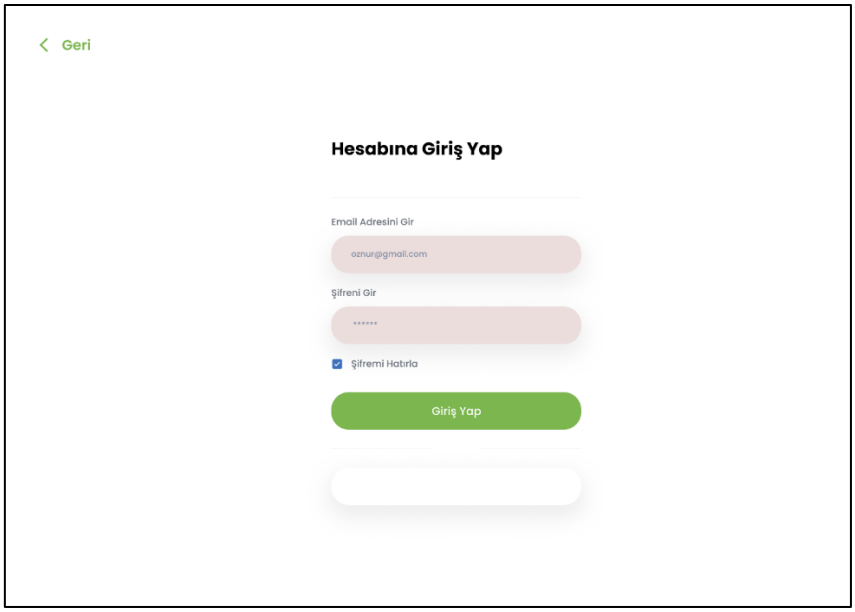
3. PROJE ARA YÜZÜ VE KODLAMALARI

3.1 Giriş ve Kayıt Ekranları

Projeyi açtığımızda kullanıcı ilk olarak kayıt ekranı ile karşılanır (Şekil 3.1). Bu ekranda kullanıcı hesap oluşturabilir ya da hesabı varsa giriş yapabilir. Kullanıcıdan email adresi, şifre bilgileri istenerek kayıt işlemi tamamlanır. Kayıt işlemini tamamlayan kullanıcı ise Şekil 3.2'de gösterilen giriş ekranı ile karşılanır.



Şekil 3.1 Hesap Oluştur Ekranı



Şekil 3.1 Giriş Yap Ekranı

```
2 başvuru
public async Task<string> Login(LoginModel loginModel)
{
    if (loginModel is null)
        throw new ArgumentNullException(nameof(loginModel));

    ApplicationUser user;
    if (string.IsNullOrEmpty(loginModel.Username) == false)
        user = await _userManager.FindByNameAsync(loginModel.Username);
    else if (string.IsNullOrEmpty(loginModel.Email) == false)
        user = await _userManager.FindByEmailAsync(loginModel.Email);
    else throw new ArgumentNullException(nameof(loginModel), "username and email are empty");

    bool isPasswordCorrect = await _userManager.CheckPasswordAsync(user, loginModel.Password);
    if (user is not null && isPasswordCorrect)
    {
        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
            new Claim(ClaimTypes.Name, user.UserName),
            new Claim(ClaimTypes.Email, user.Email),
            new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        };

        var signinKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:SecretKey"]));
        var token = new JwtSecurityToken(
            issuer: _configuration["Jwt:ValidIssuer"],
            audience: _configuration["Jwt:ValidAudience"],
            expires: DateTime.Now.AddMinutes(120),
            claims: claims,
            signingCredentials: new SigningCredentials(signinKey, SecurityAlgorithms.HmacSha256)
            );

        var jwt = new JwtSecurityTokenHandler().WriteToken(token);
```

Şekil 3.3 Giriş Yapma Kodu

```
2 başvuru
public async Task CreateUser(RegisterModel registerModel)
{
    if (registerModel is null)
        throw new ArgumentNullException(nameof(registerModel));

    var usernameExist = await _context.Users.AnyAsync(x => x.UserName == registerModel.Username);
    if (usernameExist == true)
        throw new DuplicateNameException("user with the given username already exists");

    var emailExist = await _context.Users.AnyAsync(x => x.Email == registerModel.Email);
    if (emailExist == true)
        throw new DuplicateNameException("user with the given email already exists");

    if (registerModel.Password != registerModel.RepeatPassowrd)
        throw new PasswordMatchException("passwords do not match");

    var user = new ApplicationUser
    {
        UserName = registerModel.Username,
        Email = registerModel.Email,
        SecurityStamp = Guid.NewGuid().ToString()
    };

    var result = await _userManager.CreateAsync(user, registerModel.Password);
    if (result.Succeeded == false)
        throw new InvalidOperationException("user was not created");
}
```

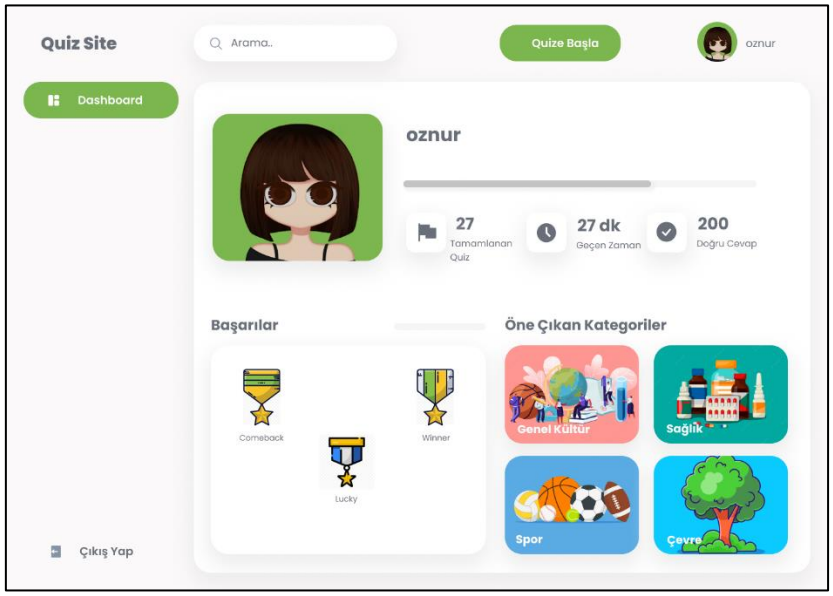
Şekil 3.3 Kayıt Olma Kodu

```
2 başvuru
public void Logout()
{
    bool authExist = _httpContext.HttpContext
        .Request.Headers.TryGetValue("Authorization", out var auth);

    if (StringValues.IsNullOrEmpty(auth) || string.IsNullOrEmpty(auth) || authExist == false)
        throw new ArgumentNullException(nameof(auth), "auth header is empty");
    string jwt = auth[0]["Bearer ".Length..];

    if (string.IsNullOrEmpty(jwt))
        throw new UnauthorizedAccessException();

    _httpContext.HttpContext.Response.Cookies.Delete("jwt");
}
```

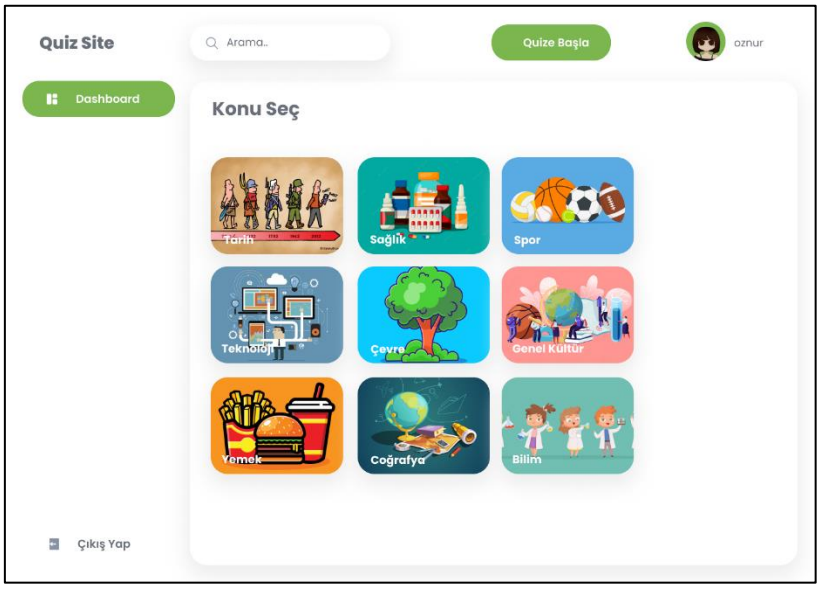
Şekil 3.3 Çıkış Yapma Kodu

3.2 Dashboard ve Konu Seçme Ekranı

Kullanıcı giriş yaptıktan sonra her kullanıcı bir dashboard'a sahip oluyor. Dashboard şekil 3.4'te gösterilmiştir. Bu sayfada kullanıcı tamamladığı quizleri, toplam geçen zamanı ve doğru cevapladığı soru sayısını görebilmektedir. Aynı zamanda "Öne Çıkan Kategoriler" ve "Başarılar" adı altında iki bölme daha oluşturulmuştur. Öne çıkan kategoriler başlığı altında kullanıcının çözdüğü son dört kategori gösteriler. Başarılar kısmında ise kullanıcının çözdüğü testler sonucu aldığı rozetler sergilenmektedir. Kullanıcının quiz çözebilmesi için "Quize Başla" butonuna tıklaması gerekmektedir.



Şekil 3.4 Dashboard Ekranı



Şekil 3.5 Konu Seç Ekranı

Quize Başla butonuna bastıktan sonra kullanıcı şekil 3.5'te gösterilen "Konu Seç" ekranı ile karşılaşır. Bu ekranda kullanıcı istediği farklı kategorilerdeki konulardan oluşan quizlerden çözebilir. Bunlar tarih, sağlık, spor, teknoloji, çevre, genel kültür, yemek, coğrafya ve bilim olarak sınırlandırılmıştır. Projenin şu anki aşamasında her kategori yalnızca bir testten oluşmaktadır. Eğer kullanıcı herhangi bir kategoride teste tıklarsa Şekil 3.6'da gösterilen ekran ile karşılaşmaktadır.

```
2 başvuru
public async Task<IEnumerable<User>> GetPassedUsersByQuiz(int quizid)
{
    var userQuizzes = await _context.UsersQuizzes.Where(x => x.CompletedQuizzesId == quizid).ToListAsync();
    List<User> users = new();
    foreach (var userQuiz in userQuizzes)
        users.Add(_mapper.Map<User>(await _context.Users.FirstOrDefaultAsync(x => x.Id == userQuiz.CompletedUsersId)));
    return users;
}

2 başvuru
public async Task<UsersQuizzes> GetResultByUsername(string username, int quizid)
{
    var userid = (await _context.Users.FirstOrDefaultAsync(x => x.UserName == username)).Id;
    var userQuiz = await _context.UsersQuizzes.FirstOrDefaultAsync(x => x.CompletedQuizzesId == quizid && x.CompletedUsersId == userid);
    var userQuizDto = _mapper.Map<UsersQuizzes>(userQuiz);
    return userQuizDto;
}
```

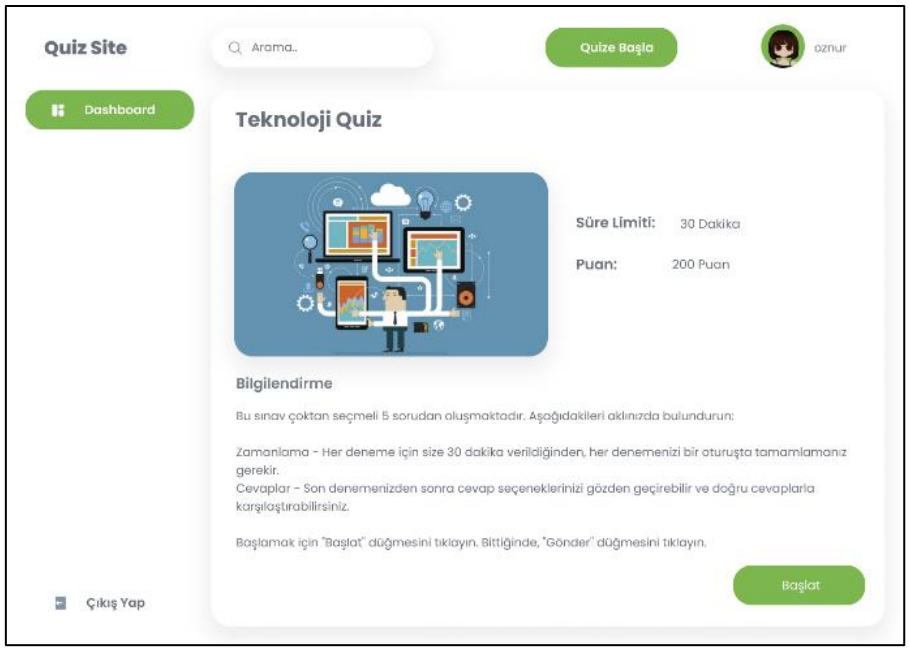
Şekil 3.6 Tamalanan Quiz ve Skor Alma İçin Yazılan Kod

```
2 başvuru
public async Task<IEnumerable<Question>> GetQuestionsByQuizId(int quizId)
{
    var questions = await _context.Questions.Where(x => x.QuizId == quizId).ToListAsync();
    var questionDto = _mapper.Map<IEnumerable<Question>>(questions);
    return questionDto;
}
```

Şekil 3.7 Seçilen Quiz'i Algılama Kodu

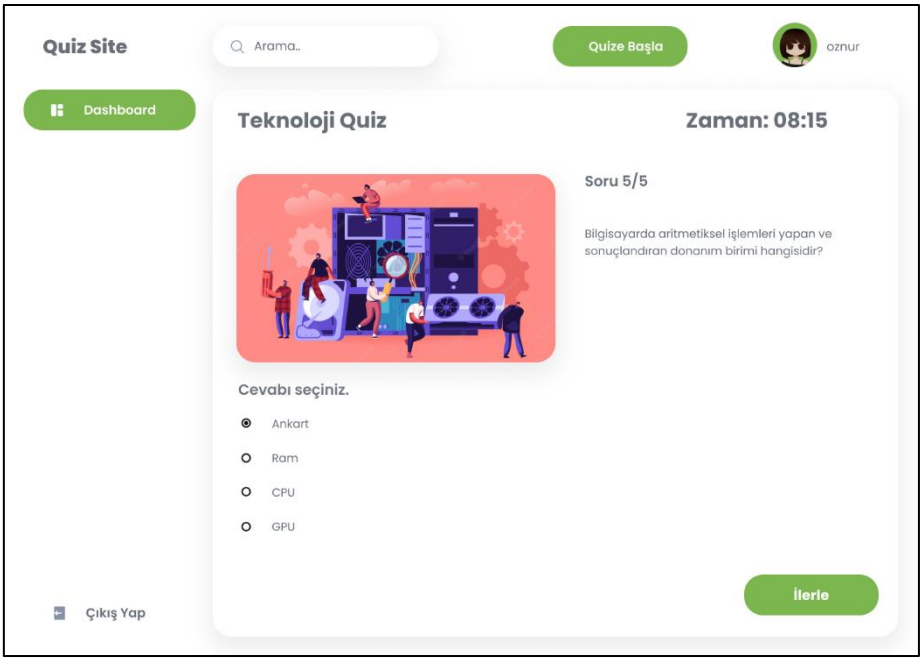
3.3 Quiz Ekranı

Konu seçildikten sonra açılan ekranda test hakkında bilgilendirme ekranı açılır. Bu ekranda test için verilen süre ve puan gösterilir. Aynı zamanda her test içinde aynı bilgilendirme ekranı yer alır. Burada test hakkında ufak bir bilgilendirme yer alır.

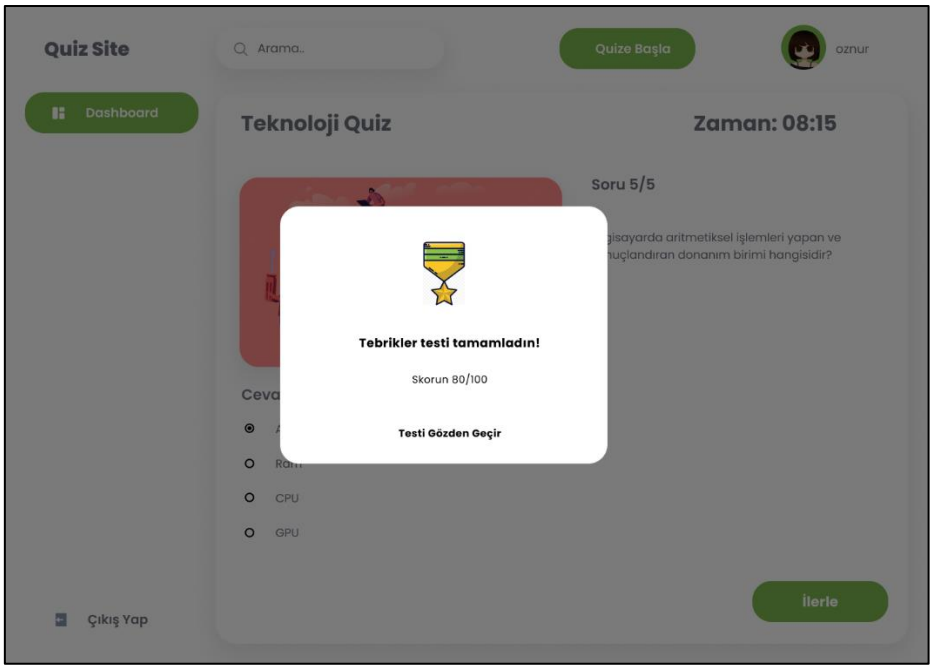


Şekil 3.8 Quiz Bilgilendirme Ekranı

Bilgilendirme ekranında yer alan “Başlat” butonuna bastıktan sonra quiz başlar ve bu sayfada kaçıncı soruda olduğumuzu gösteren ibare, kronometre, soruyla ilişkilendirilmiş görsel, soru ve cevaplar yer almaktadır. Bu ögeler Şekil 3.7’de gösterilmiştir. Cevabı işaretleyen kullanıcı ise ilerle butonuna basarak bir sonraki soruya geçebilir. Eğer test tamamlanırsa bizi Şekil 3.9’da yer alan ekran karşılamaktadır.



Şekil 3.9 Quiz Ekranı



Şekil 3.10 Tamalanan Test Ekranı

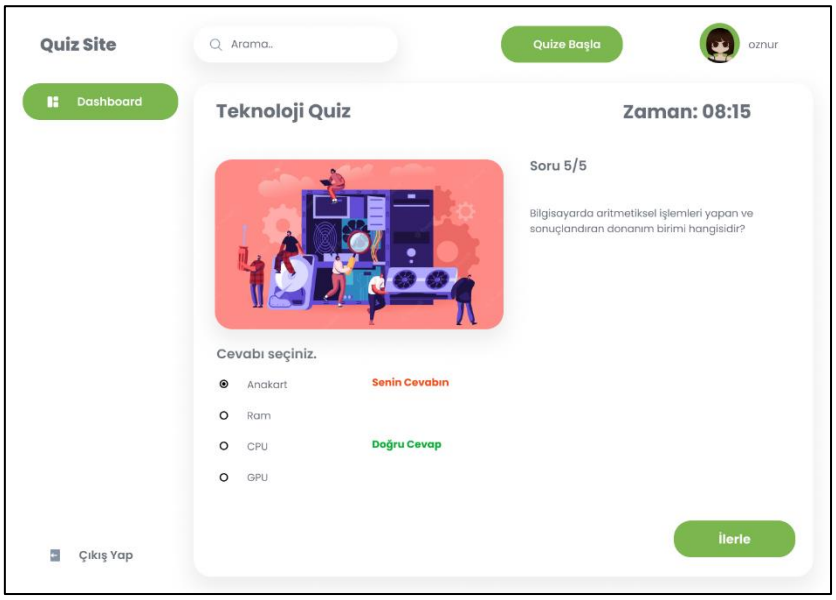
```
2 başvuru
public async Task<IEnumerable<Answer>> GetAnswersByQuizId(int quizId)
{
    var questions = await _context.Questions.Where(x => x.QuizId == quizId).ToListAsync();
    var answersDto = new List<Answer>();
    foreach (var question in questions)
    {
        var answers = await _context.Answers
            .Where(x => x.QuestionId == question.Id)
            .ToListAsync();
        answersDto.AddRange(_mapper.Map<List<Answer>>(answers));
    }
    return answersDto;
}
```

Şekil 3.11 Kullanıcıdan cevap alma kodu

```
2 başvuru
public async Task<bool> IsQuizCompletedByCurrentUser(int quizId)
{
    var user = await GetOriginalCurrentUser();
    bool isCompleted = await _context.UsersQuizzes
        .AnyAsync(x => x.CompletedUsersId == user.Id && x.CompletedQuizzesId == quizId);
    return isCompleted;
}
```

Şekil 3.12 Tamamlanan Quiz Kodu

Şekil 3.10’ da gösterilen ekranda kullanıcıya testi tamamladığı için bir rozet verilir ve testi tamamladın yazısı bildirilir. Bu ekranda aynı zamanda kullanıcının skoru doğru ve yanlış sayısı oranlanarak hesaplanır. Kullanıcının bu ekranda yapabileceği bir diğer özellik ise testi gözden geçirmek olacaktır. Testi gözden geçirmek isteyen kullanıcı Şekil 3.13’de örneği gösterilen ekran ile karşılaşır. Bu ekran kullanıcı cevapladığı tüm soruları görebilir ve yaptığı yanlışları doğrusu ile görebilir.



Şekil 3.13 Sonuç Ekranı

```dart
class AfterGameScreen extends StatefulWidget {
  final int score;

  const AfterGameScreen({Key? key, required this.score}) : super(key: key);

  @override
  State<AfterGameScreen> createState() => _AfterGameScreenState();
}

class _AfterGameScreenState extends State<AfterGameScreen> {
  late double correctNumber;
  bool isPlaying = false;
  late ConfettiController _controllerTopCenter;

  @override
  void dispose() {
    _controllerTopCenter.dispose();
    super.dispose();
  }

  @override
  void initState() {
    correctNumber = widget.score / 10;
    _controllerTopCenter =
        ConfettiController(duration: const Duration(seconds: 5));

    super.initState();
  }
```

Şekil 3.14 Test Tamamlandı Ekranı