THE EGYPTIAN E-LEARNING UNIVERSITY

Faculty of Computers and Information

Technology (IT)

Graduation project: Career Guidance

BY

- **Mohamed Khaled Abdeltawab (Team Leader)**
- **Abdelrahman Maged Abdeldayem**
- **Abdelrahman Rezk Mohamed**
- **Ziad Elamir Mohamed**
- **Shrouk Ahmed Rashad**
- **Nourhan Khaled Shipa**
- **Rania Rabie Sayed**

Supervised by

Eng. Ahmed Gomaa

2024-2025

Table of content

Introduction

- The field of programming has become one of the most popular fields in the world, and anyone, regardless of their qualifications, can enter it. This field is one of the fields in which a person can learn individually through the huge amount of resources available on the Internet.

Problem

- When starting in programming or college, many people find it hard to choose the right path, which can cause frustration and wasted time. The main problem is that they don't have a clear idea of the different programming tracks available. Without the right guidance, they might depend on advice from others that may be biased, or try to learn everything at once, which can be overwhelming and not effective. Choosing the right track is important for learning efficiently and achieving success in the field.

project idea(Solution)

- Career Guidance, a guidance platform designed to help anyone looking to enter the programming field by providing comprehensive information tailored to their chosen path. Based on the platform's resources, users can access a variety of courses, each accompanied by relevant links and general information about different paths. Additionally, we provide detailed roadmaps for all available paths in the job market, explaining how to learn each one and where to find the necessary resources. All of this is readily available on our platform, ensuring that users have everything they need to successfully navigate their programming career.

Chapter one

Detailed Explanation

Career Guidance: Simplifying Your Journey into Programming

The Career Guidance project is designed to help individuals, particularly those interested in programming or transitioning to a tech career, navigate the overwhelming choices in the field. This project is targeted at students, recent graduates, and professionals who want to make an informed decision about their career path in programming. Here's a clear breakdown of the project's key features and purpose.

Challenges in Exploring Programming Fields

When someone starts exploring programming, they often feel lost or unsure of where to begin. Whether they are new college students or professionals looking to shift their careers, it's common to receive advice that only promotes a single field—often from someone already working in it. This advice can limit their perspective, pushing them toward a specific path, which may not even be the right one for them.

For example, if they hear about a field like Embedded Systems without fully understanding it, they might miss out on discovering whether it's a good fit for them. The same goes for spending time exploring various courses or resources online, often wasting valuable time without achieving any real results. Many people are stuck in this cycle, unsure of what programming field to choose or how to efficiently start learning.

What Career Guidance Offers

This is where Career Guidance steps in. We are here to assist anyone interested in tech or programming, especially those who have been trying hard but haven't seen tangible progress. Our platform provides users with:

- **An Overview of All Available Fields:** We offer insights into various programming tracks and explain each one in detail—what it involves, its demand in the market, and whether companies are still actively using those technologies.
- **Tailored Roadmaps for Every Track:** After helping users gain a clear understanding of different fields, we guide them to select the track that suits their interests. We provide personalized roadmaps that show a structured and organized learning path, complete with recommended courses and resources.
- **Course Recommendations:** Based on whether the user is a beginner or has some prior knowledge, we suggest the best courses to kickstart or further their journey. These courses have been carefully curated, tested, and organized based on real-world experience to ensure users learn in the most efficient way.
- **Real-time Progress Tracking:** Our system allows users to track their learning progress in real-time, with features like a progress bar showing them exactly how much they've completed and what's still ahead.

Additional Support and Features

- **CV Building:** We help users craft strong CVs tailored to their chosen track, increasing their chances of landing a job.
- **Job Opportunities:** Based on the user's learning track, we recommend job opportunities that align with their new skills.
- **Affordable Premium Courses**: We offer paid courses at a lower price than elsewhere, helping users gain access to quality learning materials at a budget-friendly rate.
- **Customization**: Users can choose between dark mode and light mode for a comfortable experience.
- **Quizzes and Checkpoints:** Each stage of the roadmap includes quizzes to ensure that the learner has fully grasped the concepts before moving forward. This helps build confidence that they are ready for the next phase.
- **Detailed Explanations of Every Track:** Before diving into any programming track, users can explore commonly asked questions, details about the programming languages they'll learn, and the frameworks they'll use—so there's no confusion about what they're getting into.

Target Audience

Our project caters to all age groups but is particularly focused on people between the ages of 18 and 40, who are either starting their journey into tech or looking to make a career change.

In Summary

Career Guidance simplifies the process of finding the right programming path by offering personalized, structured guidance, real-time tracking, and support at every step. By helping individuals choose wisely and learn effectively, we ensure they can start their careers in tech without wasting time or effort, all while staying engaged and motivated throughout their journey.

Chapter Two

Diagrams

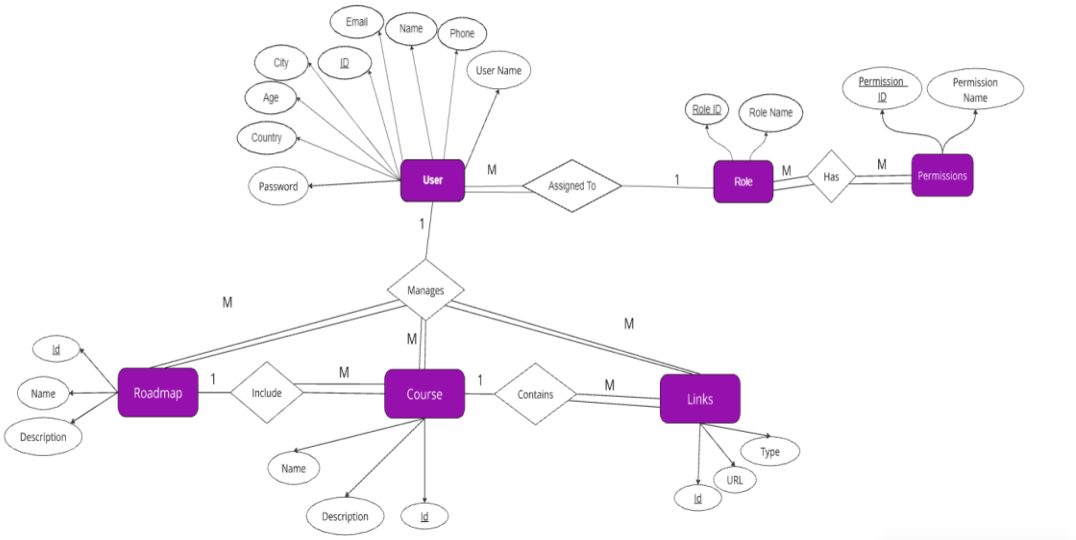

Figure 1:ERD DIAGRAM

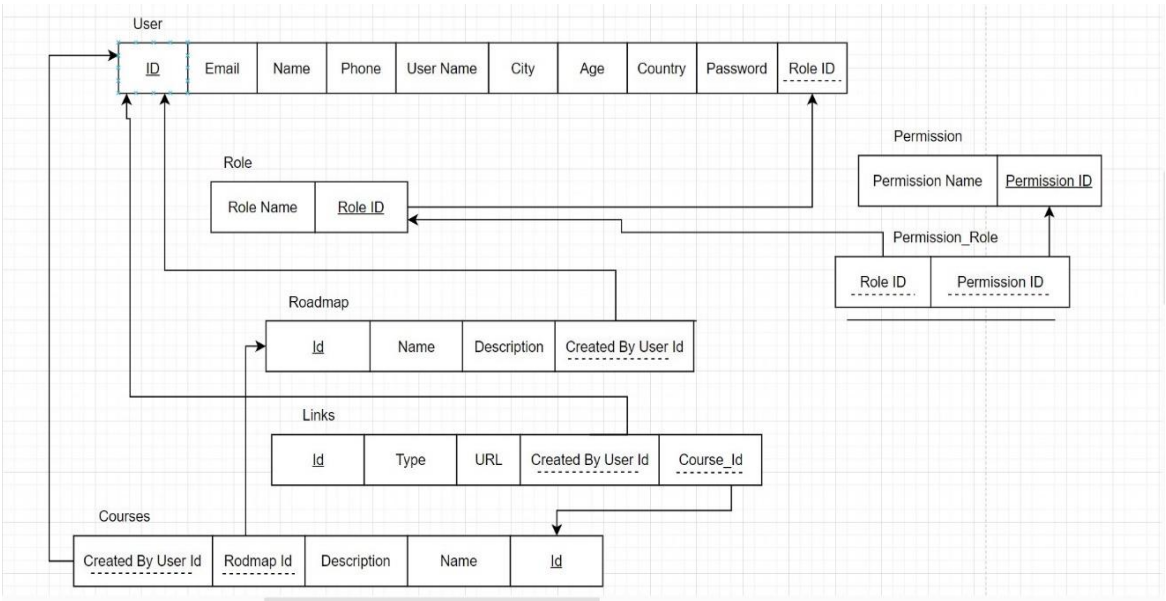

Figure 2:Schema

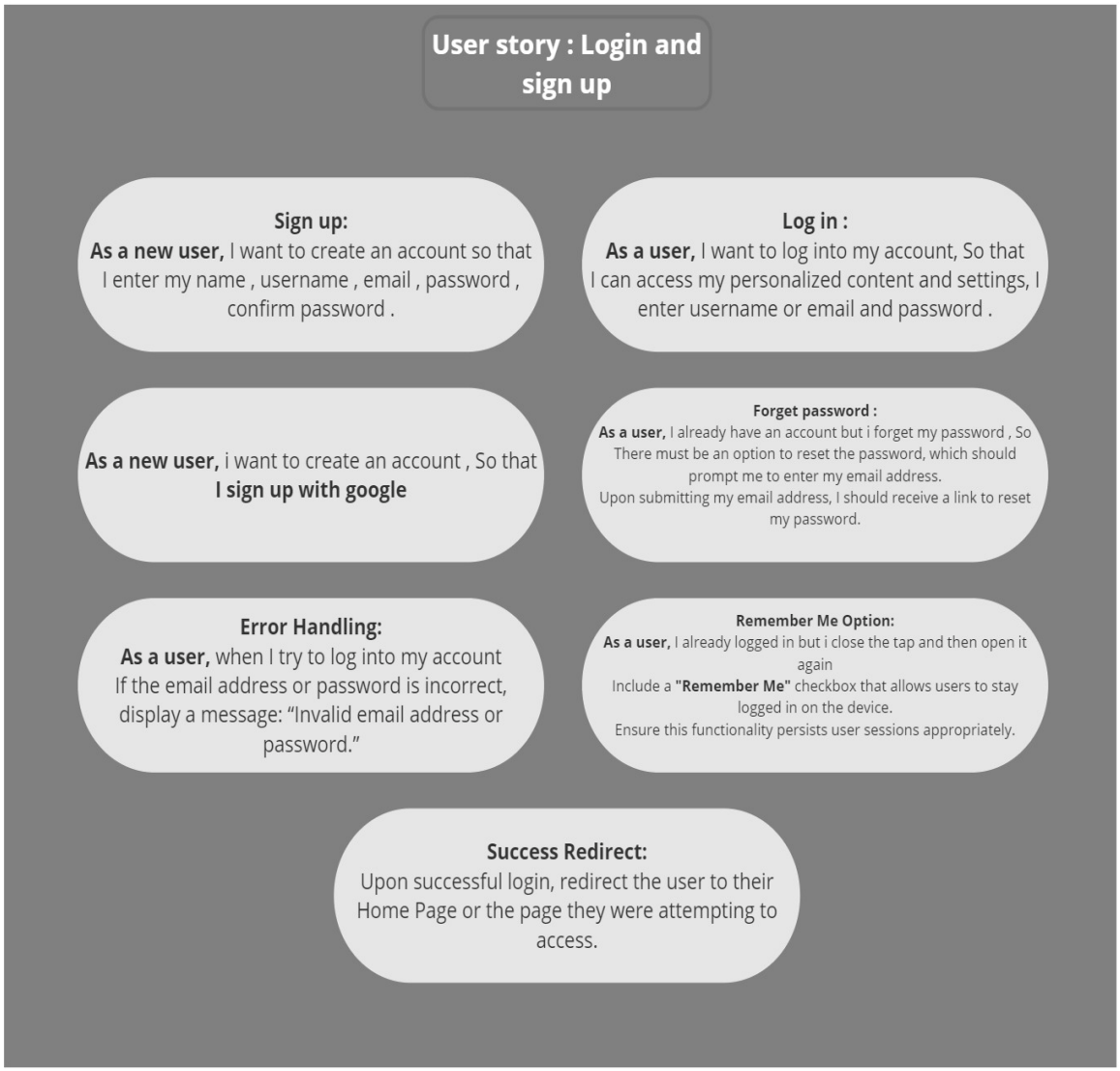

Figure 3:(User Story) Login and Sign Up

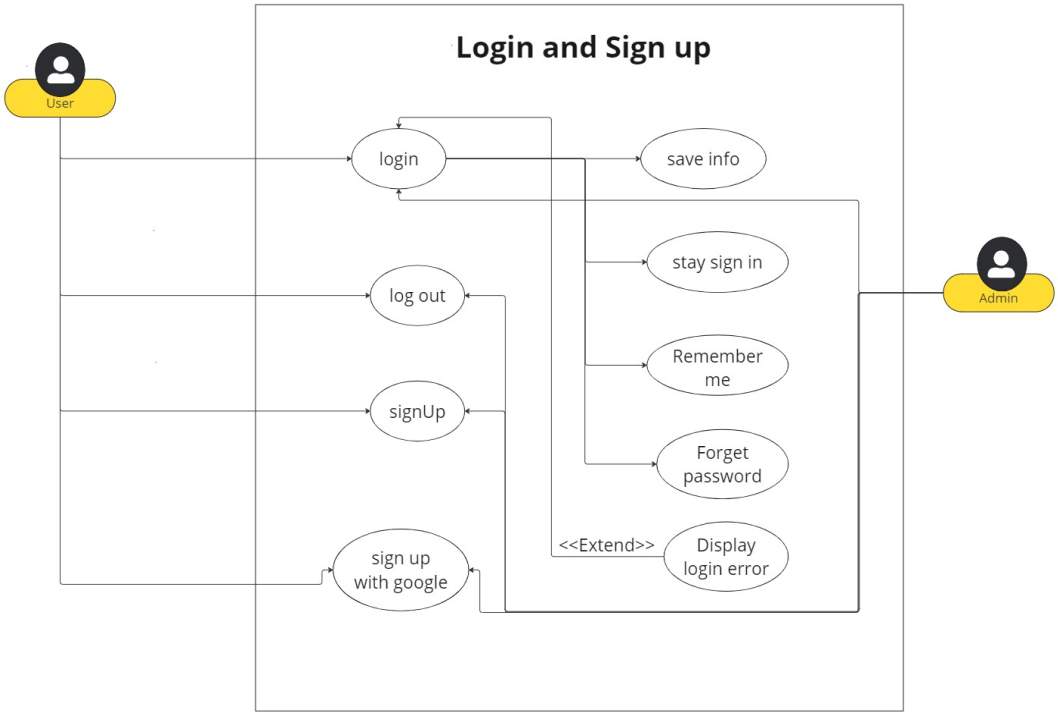

Figure 4:(Use Case) Login and Sign Up

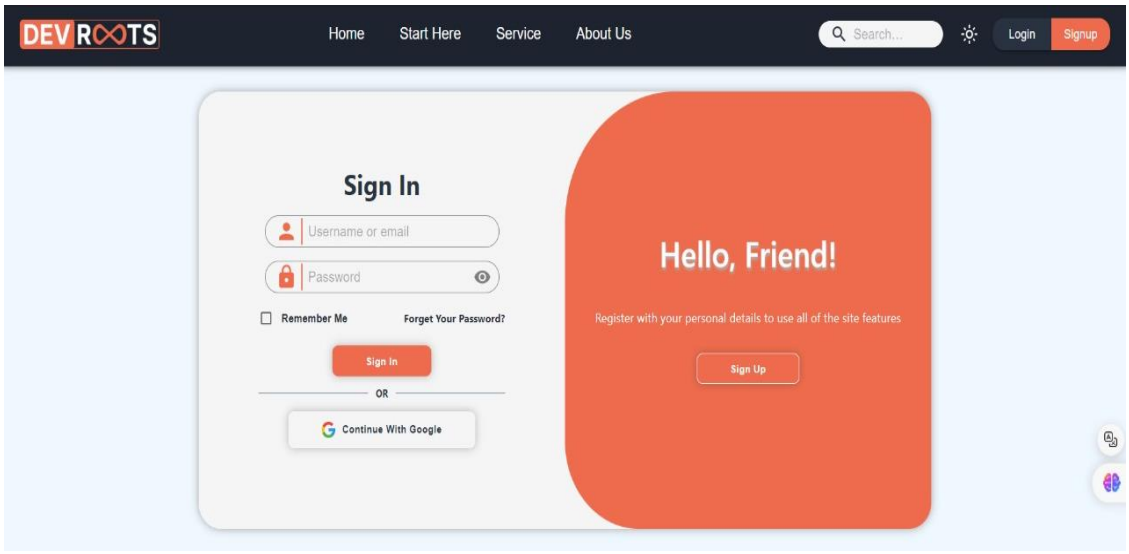

Figure 5: Login Light Mode (UI)

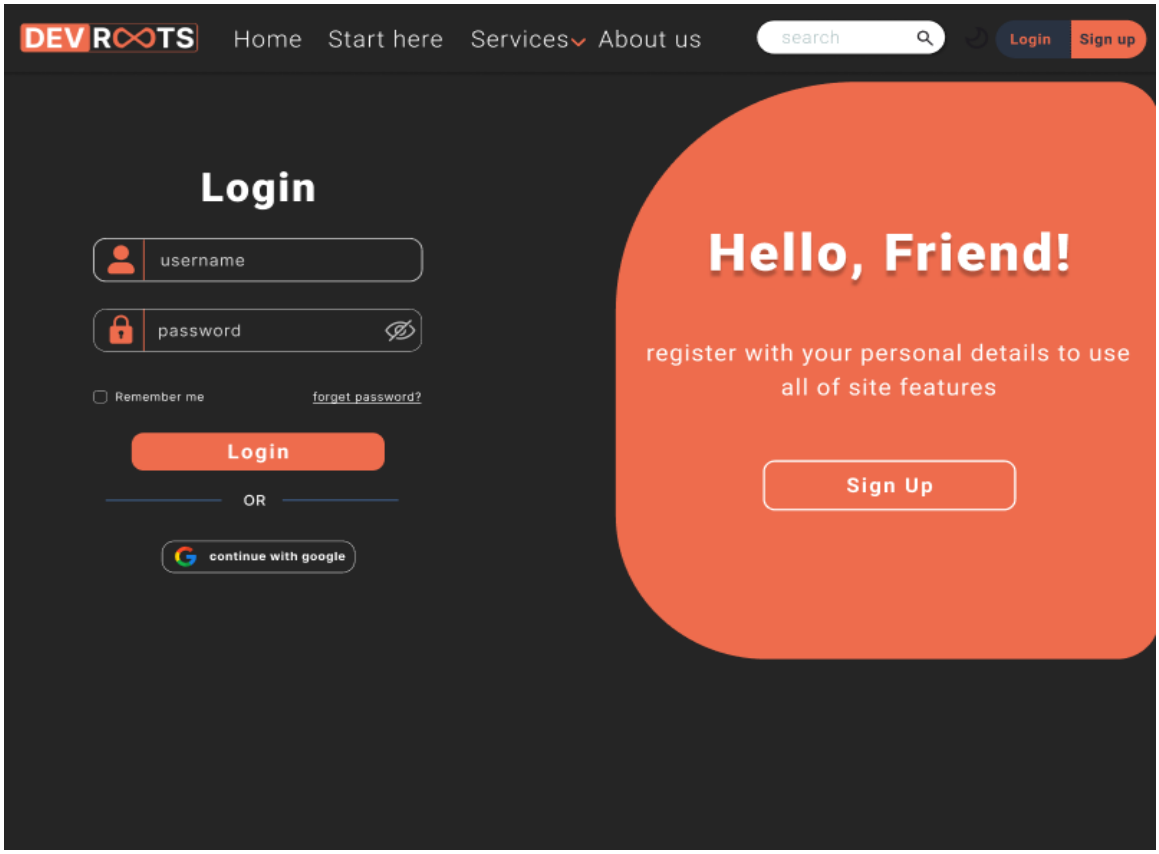

Figure 6:Login Dark Mode (UI)

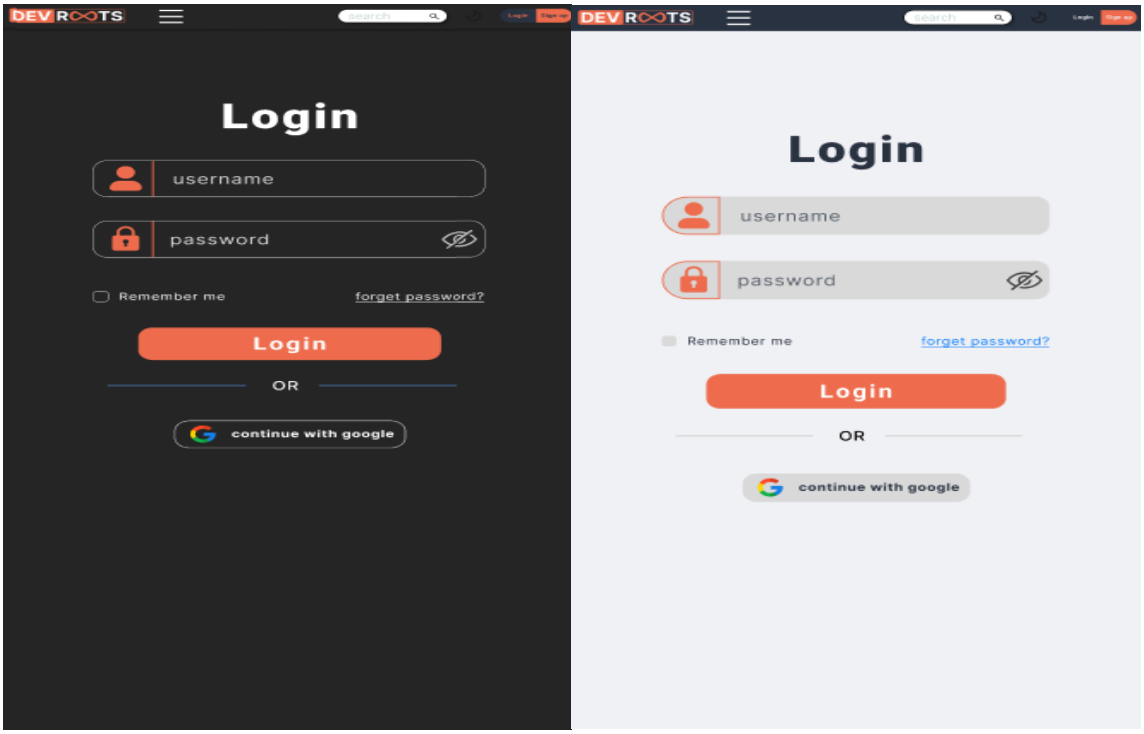

Figure 7:Login Dark and Light Mode For Phone (UI)

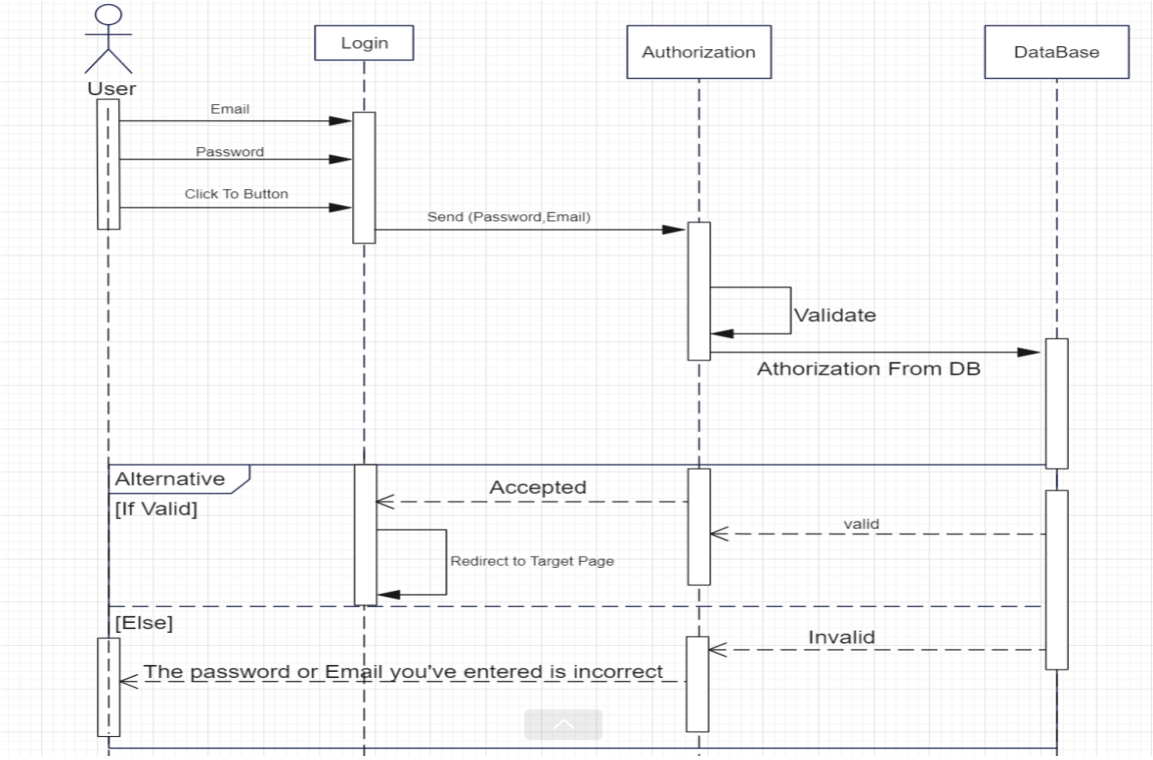

Figure 8: Sequence Diagram For Login

```tsx
// Function to set the email or username
const setEmail: React.Dispatch<React.SetStateAction<string>> = (value: string) => {
  // Implementation to set email or username
};

// Function to set the password for sign-in
const setSignInPassword: React.Dispatch<React.SetStateAction<string>> = (value: string) => {
  // Implementation to set the password
};

// Function to toggle password visibility
const handleToggleShowPassword: () => void = () => {
  // Implementation to toggle show/hide password
};

// Function to set the "Remember Me" checkbox value
const setRememberMe: React.Dispatch<React.SetStateAction<boolean>> = (value: boolean) => {
  // Implementation to set the Remember Me checkbox
};

// Function to handle Google login
const handleGoogleLogin: () => void = () => {
  // Implementation to handle Google login
};

// Function to handle clicking on the "Sign In" button
const handleLoginClick: () => void = () => {
  // Implementation to handle sign-in button click
};

// Function to handle clicking on the "Sign Up" button
const handleRegisterClick: () => void = () => {
  // Implementation to handle register button click
};

// Function to handle closing the snackbar/alert
const handleClose: (event?: React.SyntheticEvent | Event, reason?: string) => void = (event, reason) => {
  // Implementation to handle closing the snackbar
};
```

Figure 9: Login Design Code(Frontend)

```
public record LoginUserDto
(
    /*
     * Property: Email
     * Description: Represents the email address of the user for login purposes.
     * Type: string
     */
    string EmailOrUsername,

    /*
     * Property: Password
     * Description: Represents the password of the user for login purposes.
     * Type: string
     */
    string Password
);
}

public class LoginUserDtoValidator : AbstractValidator<LoginUserDto>
{
    0 references
    public LoginUserDtoValidator()
    {
        /*
    * Rule: EmailOrUsername
    * Description: Validates that the EmailOrUsername property is not empty
    * and is either a valid email address or a valid username.
    */
        RuleFor(x => x.EmailOrUsername)
            .NotEmpty()
            .Must(IsValidEmailOrUsername)
            .WithMessage("The field must be a valid email address or username.");

        /*
         * Rule: Password
         * Description: Validates that the Password property is not empty.
         */
        RuleFor(x => x.Password).NotEmpty();
    }

    1 reference
    private bool IsValidEmailOrUsername(string emailOrUsername)
    {
        return IsValidEmail(emailOrUsername) || IsValidUsername(emailOrUsername);
    }
```

```
        private bool IsValidEmail(string email)
        {
            // استخدم الدالة المدمجة للتحقق من صحة البريد الإلكتروني
            return new EmailAddressAttribute().IsValid(email);
        }

        1 reference
        private bool IsValidUsername(string username)
        {
            // هنا مثلاً يكون على الأقل 3 أحرف، تعريف قواعد التحقق من صحة الـ username
            return username.Length >= 3;
        }
    }
}

// Endpoint for user login
[HttpPost("Login")]
0 references
public async Task<IActionResult> LoginAsync(LoginUserDto request, CancellationToken cancellationToken)
{
    var authResult = await authService.GetTokenAsync(request.EmailOrUsername , request.Password,cancellationToken);

    return authResult.IsSuccess ? Ok(authResult.Value) : authResult.ToProblem();
}

[HttpPost("refresh")]
0 references
public async Task<IActionResult> RefreshAsync(RefreshTokenRequest request, CancellationToken cancellationToken)
{

    var authResult = await authService.GetRefreshTokenAsync(request.Token, request.RefreshToken, cancellationToken);

    return authResult is null
        ? BadRequest("Invalid Token")
        : Ok(authResult);
}
  public record AuthResponse
  (
      string Id, // Unique identifier for the user
      string? Email, // Email address of the user (nullable)
   // string Name, // Full name of the user
      string UserName, // Username of the user
      string Token, // JWT token for authorization
      string Role, // Role assigned to the user (e.g., "Student", "Instructor")
      int ExpiresIn, // Token expiration time in minutes
      string RefreshToken,
      DateTime RefreshTokenExpiration
  );
}
```

Figure 10:Login Design Code(Backend)

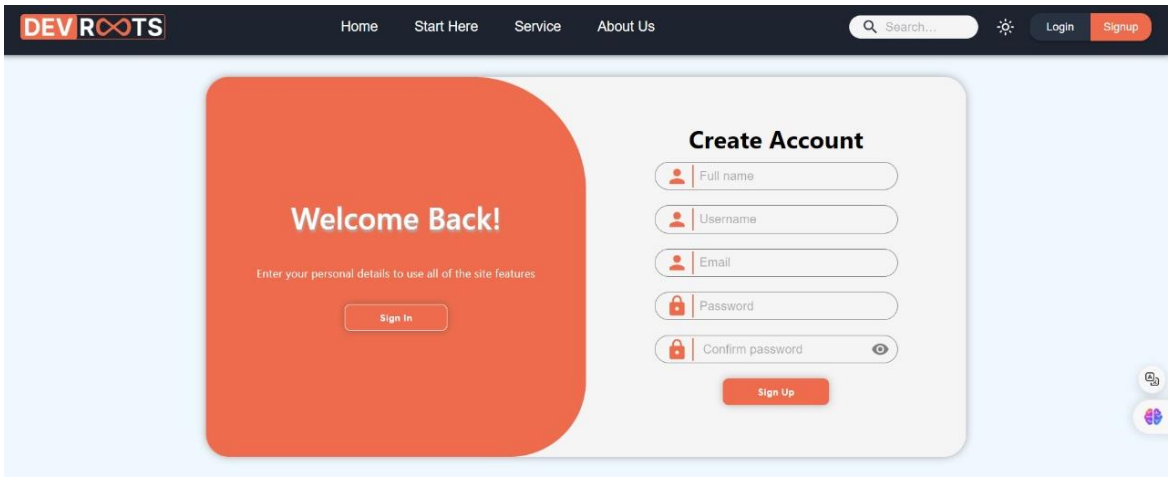

Figure 11:Sign Up Light Mode (UI)

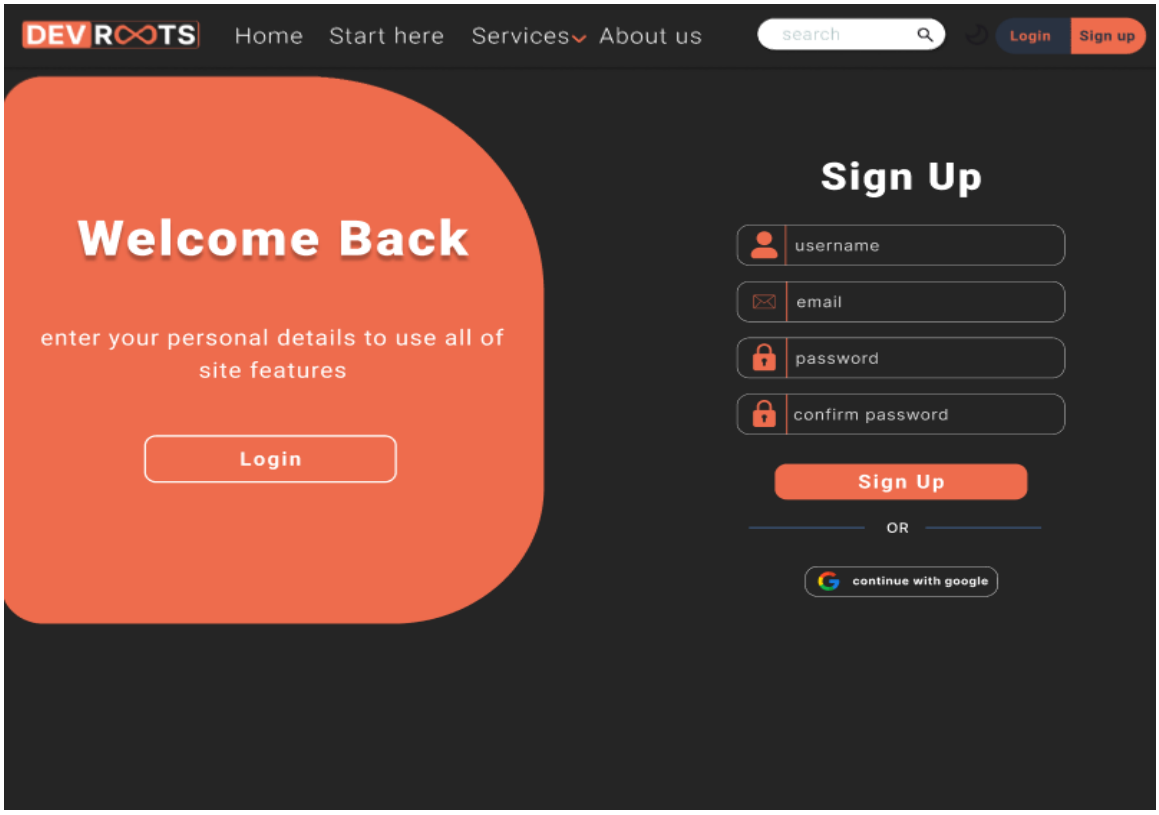

Figure 12:Sign Up Dark Mode (UI)

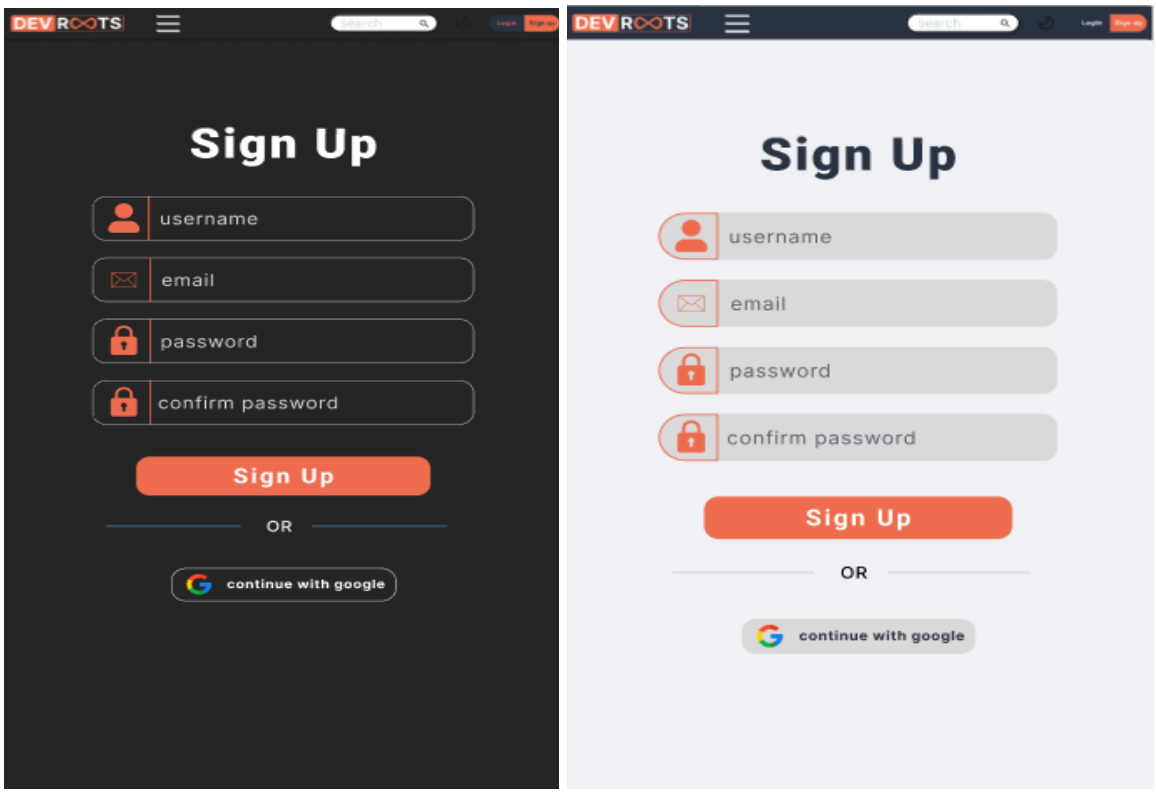

Figure 13:Sign Up Dark and Light Mode For Phone (UI)

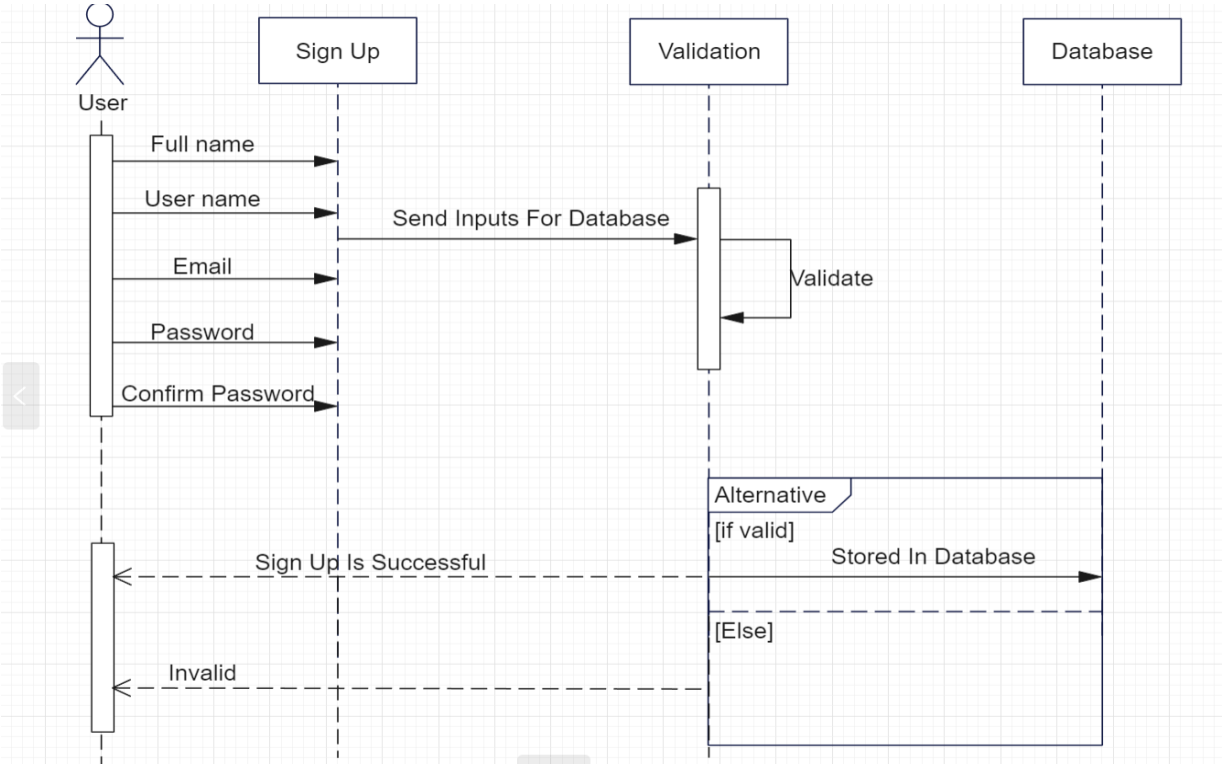

Figure 14:Sequence Diagram For Sign Up

```
  const [username, setUsername] = useState(''); // حالة لإضافة username

  useEffect(() => {
    console.log('Username has changed:', username);
  }, [username]);

  const handleUsernameFocus = () => {
    console.log('Username input focused');
  };

  const handleUsernameBlur = () => {
    console.log('Username input blurred');
  };

  const handleEmailChange = (e) => {
    console.log('Email changed:', e.target.value);
  };

  const handleEmailFocus = () => {
    console.log('Email input focused');
  };

  const handleEmailBlur = () => {
    console.log('Email input blurred');
  };

  const onSubmit = async (data) => {
    console.log('Form submitted with data:', data);
  };

  const handleClick = () => {
    console.log('Button clicked');
  };

  const handleGoogleLogin = () => {
    console.log('Google login clicked');
  };

const handleRegisterClick = () => {
  console.log('Register clicked');
};

const handleLoginClick = () => {
  console.log('Login clicked');
};
```

Figure 15:Sign Up Design Code(Frontend)

```
/*
 * Rule: UserName
 * Description: Validates that the UserName property is 5 to 20 characters long, starts with
 * at least 3 letters, and can include letters, numbers, and underscores.
 */
RuleFor(x => x.UserName)
   .Matches(@"^[a-zA-Z]{3}[a-zA-Z0-9_]{2,17}$")
   .WithMessage("UserName must be 5 to 20 characters long, start with at least 3 letters, and can include letters, numbers, and underscores.");

/*
 * Rule: Password
 * Description: Validates that the Password property meets the complexity requirements:
 * at least 8 characters long, includes at least one uppercase letter, one lowercase letter,
 * one digit, and one special character. Also ensures the password does not contain any part
 * of the name or the phone number.
 */
RuleFor(x => x.Password)
    .Matches(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$")
    .WithMessage("Password must be at least 8 characters long and include at least one uppercase letter, " +
    "one lowercase letter, one digit, and one special character.");
  /*
   * Rule: ConfirmPassword
   * Description: Validates that the ConfirmPassword property matches the Password property.
   */
  RuleFor(x => x.ConfirmPassword)
      .Equal(x => x.Password).WithMessage("Passwords do not match.");

  /*
   * Rule: Email
   * Description: Validates that the Email property is a valid email address and its length is
   * between 16 and 40 characters.
   */
  RuleFor(x => x.Email)
      .EmailAddress().WithMessage("Invalid email address format.")
      .MinimumLength(16).WithMessage("Email must be at least 16 characters long.")
      .MaximumLength(40).WithMessage("Email must not exceed 40 characters.");
```

Figure 16: Sign Up Design Code(Backend)-A

```
public record ForgetPasswordRequest
(
  string Email
);
}
```

```
public class ForgetPasswordRequestValidator : AbstractValidator<ForgetPasswordRequest>
{
    0 references
    public ForgetPasswordRequestValidator()
    {
        RuleFor(x => x.Email)
              .NotEmpty()
              .EmailAddress();
        }
    }
}
```

Figure 17:Sign Up Design Code(Backend)-B

```
    public record ResetPasswordRequest
    (
        string Email,
        string Code,
        string NewPassword
    );
}
```

```
public class ResetPasswordRequestValidator : AbstractValidator<ResetPasswordRequest>
{
    0 references
    public ResetPasswordRequestValidator()
    {
        RuleFor(x => x.Email)
            .NotEmpty()
            .EmailAddress();

        RuleFor(x => x.Code)
            .NotEmpty();

        RuleFor(x => x.NewPassword)
            .NotEmpty()
            .Matches(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$")
            .WithMessage("Password must be at least 8 characters long and include at least one uppercase letter," +
            " one lowercase letter, one digit, and one special character.");
    }
}
```

Figure 18:Sign Up Design Code(Backend)-C

```
17      public class RegisterUserDto
18      {
19          /*
20           * Property: Name
21           * Description: Represents the full name of the user.
22           * Type: string
23           */
            2 references
24          public string Name { get; set; } = string.Empty;
25
26          /*
27           * Property: UserName
28           * Description: Represents the username for the user account.
29           * Type: string
30           */
            2 references
31          public string UserName { get; set; } = string.Empty;
32
33          /*
34           * Property: Password
35           * Description: Represents the password for the user account.
36           * Type: string
37           */
            3 references
38          public string Password { get; set; } = string.Empty;
39
40          /*
41           * Property: ConfirmPassword
42           * Description: Represents the confirmation of the user's password.
43           * Type: string
44           */
```

Figure 19:Sign Up Design Code(Backend)-C

```csharp
public interface IAuthService
{
    /*
     * Method: GetTokenAsync
     * Description: Retrieves an authentication token for a user based on their email and password.
     * Parameters:
     *      - email: User's email address.
     *      - password: User's password.
     *      - cancellationToken: Optional cancellation token for the operation.
     * Returns:
     *      - AuthResponse?: An asynchronous operation that returns an authentication response if successful.
     */
    2 references
    Task<AuthResponse?> GetTokenAsync(string email, string password, CancellationToken cancellationToken = default);

    /*
     * Method: RegisterUser
     * Description: Registers a new user with the provided registration data.
     * Parameters:
     *      - request: Registration details for the user.
     *      - cancellationToken: Optional cancellation token for the operation.
     * Returns:
     *      - (bool IsSuccessful, IEnumerable<string> Errors): A tuple indicating success and a list of errors if any.
     */
    2 references
    Task<(bool IsSuccessful, IEnumerable<string> Errors)> RegisterUser(RegisterUserDto request, CancellationToken cancellationToken);
}
```

```csharp
    2 references
    public async Task<AuthResponse?> GetTokenAsync(string email, string password, CancellationToken cancellationToken = default)
    {
        // Check if the user exists
        var user = await userManager.FindByEmailAsync(email);

        if (user is null)
            return null;

        // Check if the password is valid
        var isValidPassword = await userManager.CheckPasswordAsync(user, password);

        if (!isValidPassword)
            return null;

        // Generate JWT token
        var (token, expiresIn) = jwtProvider.GenerateToken(user);

        return new AuthResponse(user.Id, user.Email, user.Name, user.UserName!, token, user.Role, expiresIn * 60);
    }
/*
 * Method: RegisterUser
 * Description: Registers a new user with the provided registration data.
 * Parameters:
 *      - request: Registration details for the user.
 *      - cancellationToken: Optional cancellation token for the operation.
 * Returns:
 *      - (bool IsSuccessful, IEnumerable<string> Errors): A tuple indicating success and a list of errors if any.
 */
2 references
public async Task<(bool IsSuccessful, IEnumerable<string> Errors)> RegisterUser(RegisterUserDto request, CancellationToken cancellationToken)
{
    var errors = new List<string>();

    // Check if the email is already in use
    var existingEmailUser = await userManager.FindByEmailAsync(request.Email);
    if (existingEmailUser != null)
    {
        errors.Add("Email is already in use.");
    }

    // Check if the username is already in use
    var existingUserNameUser = await userManager.FindByNameAsync(request.UserName);
    if (existingUserNameUser != null)
    {
        errors.Add("UserName is already in use.");
    }
```

```csharp
    // Check if the phone number is already in use
    var existingPhoneNumberUser = await userManager.Users
        .FirstOrDefaultAsync(u => u.PhoneNumber == request.PhoneNumber, cancellationToken);

    if (existingPhoneNumberUser != null)
    {
        errors.Add("Phone number is already in use.");
    }

    if (errors.Any())
    {
        return (false, errors); // Return tuple with failure and list of errors
    }
```

```
        // Map RegisterUserDto to ApplicationUser
        var applicationUser = request.Adapt<ApplicationUser>();
        // Create new user with the provided password
        IdentityResult result = await userManager.CreateAsync(applicationUser, request.Password);

        if (result.Succeeded)
        {
            return (true, Enumerable.Empty<string>()); // Return success with no errors
        }

        // Add errors from IdentityResult if user creation fails
        errors.AddRange(result.Errors.Select(e => e.Description));

        return (false, errors); // Return failure with the list of errors
    }
}
```

Figure 20:Sign Up Design Code(Backend)-D

Figure 21: Home Page Dark Mode

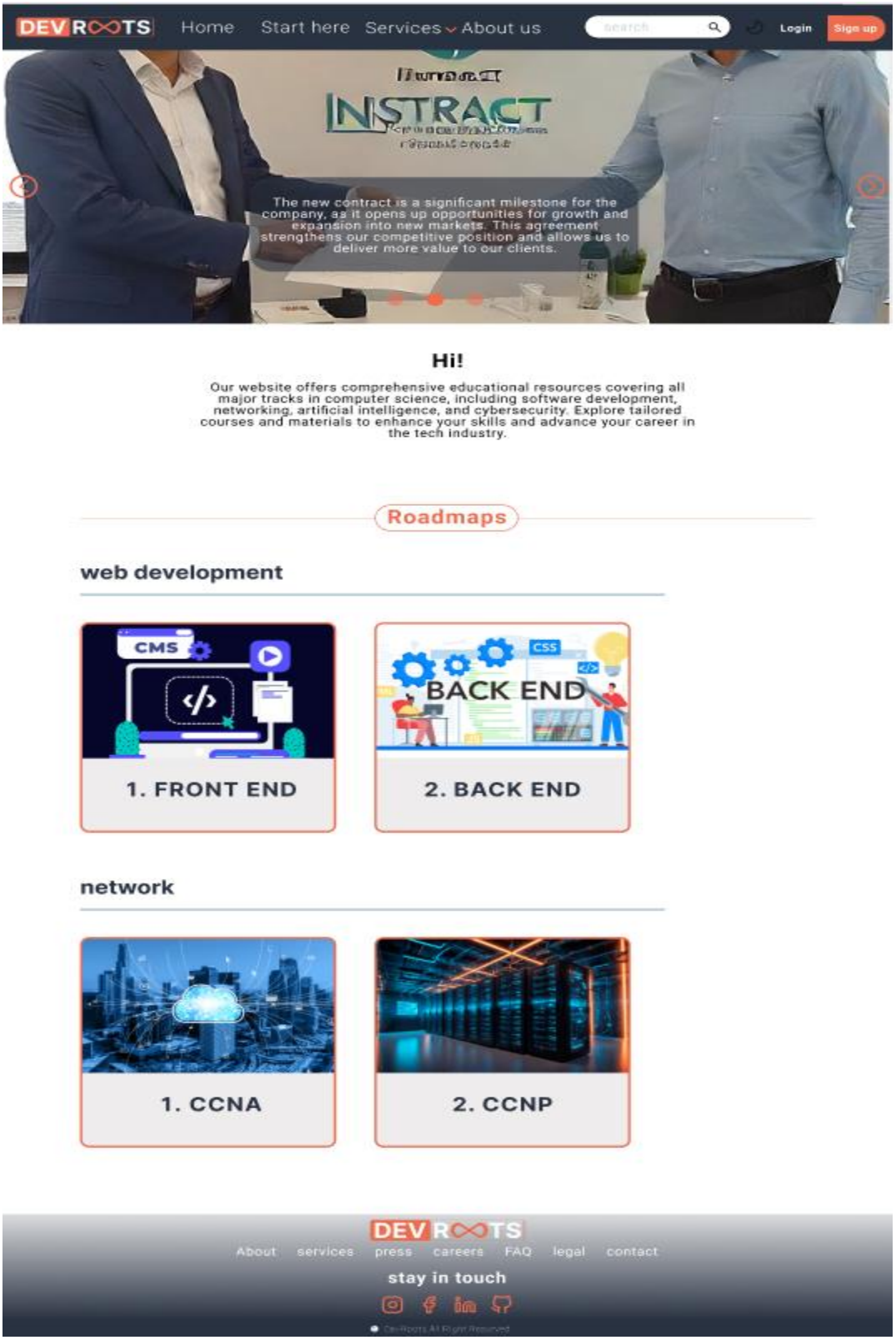

Figure 22: Home Page Light Mode

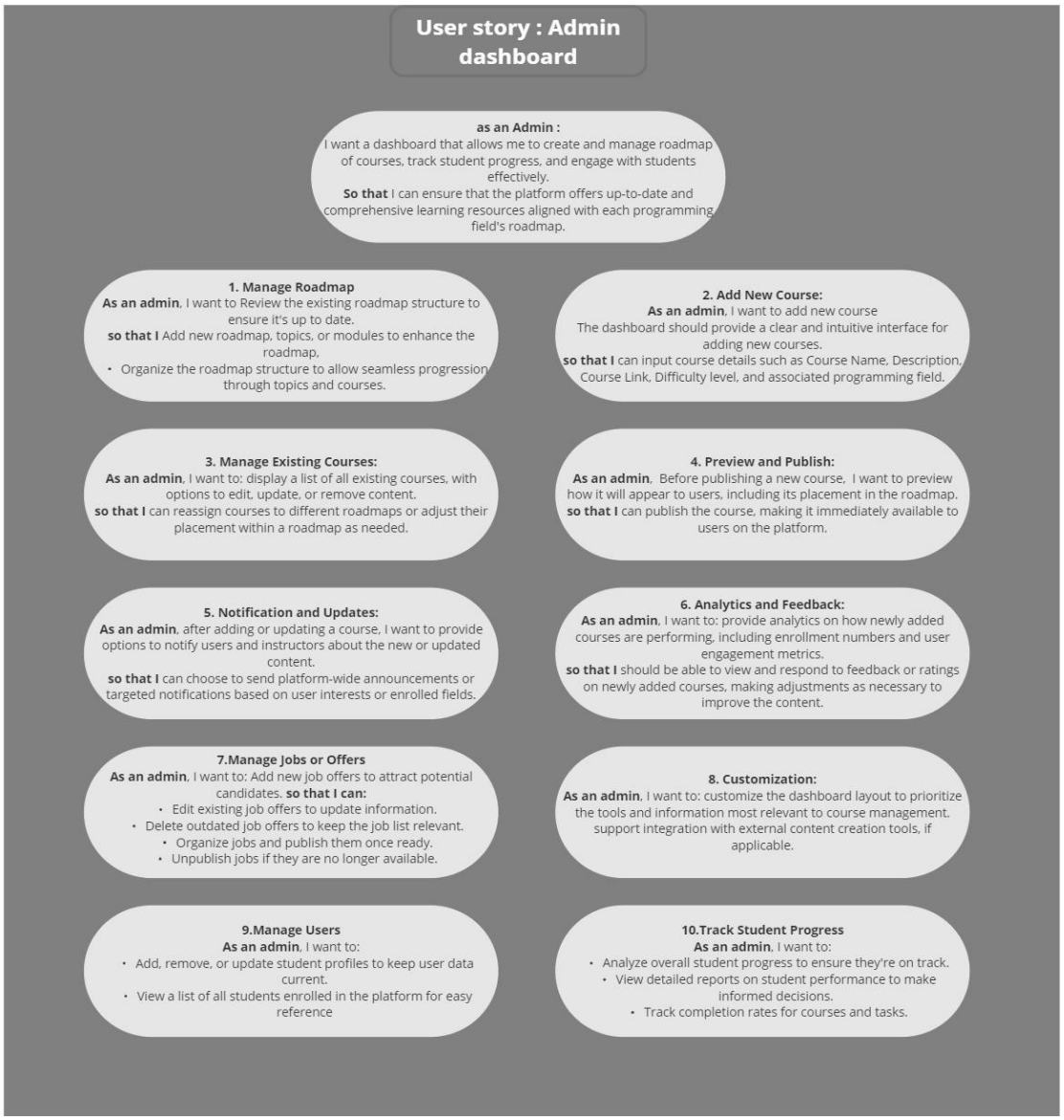

Figure 23:Admin Dashboard (User Story)

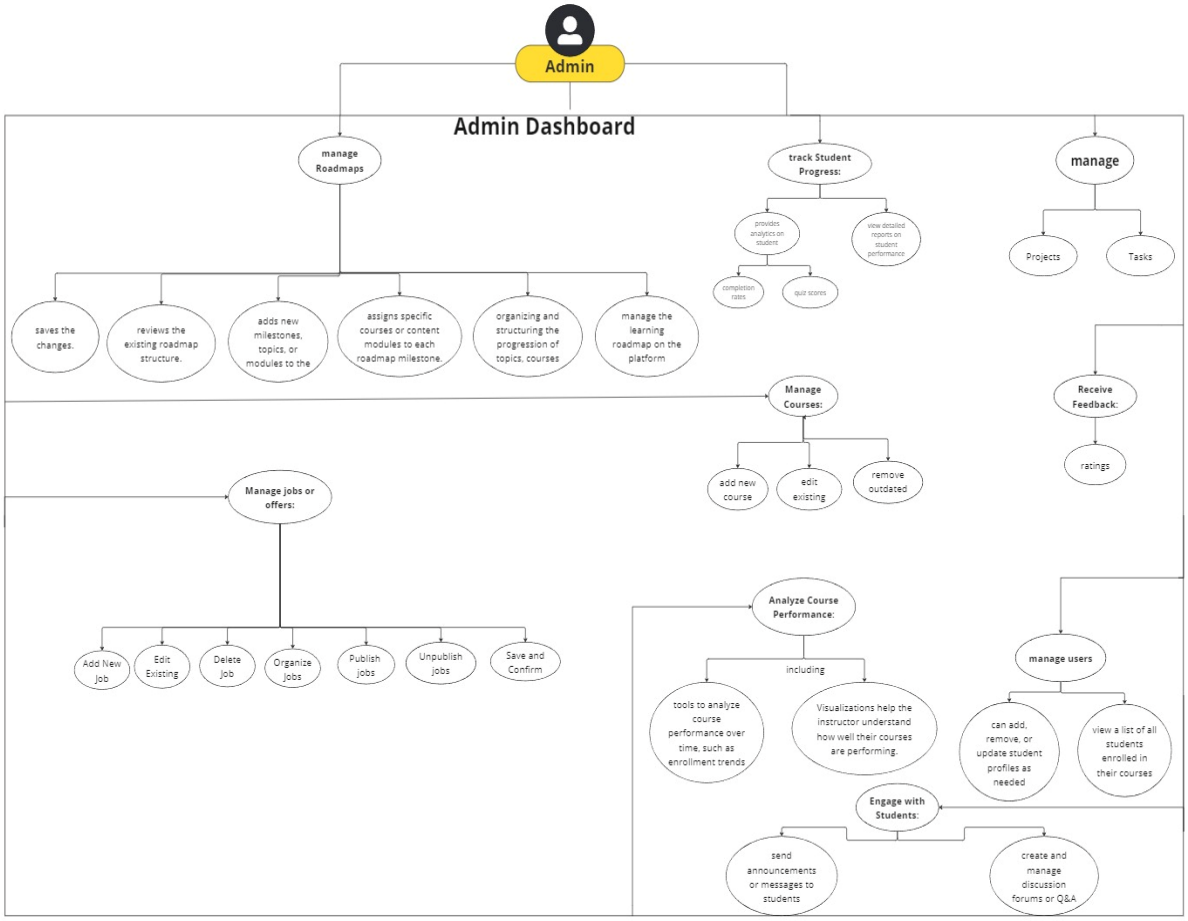

Figure 24:Admin Dashboard (Use Case)

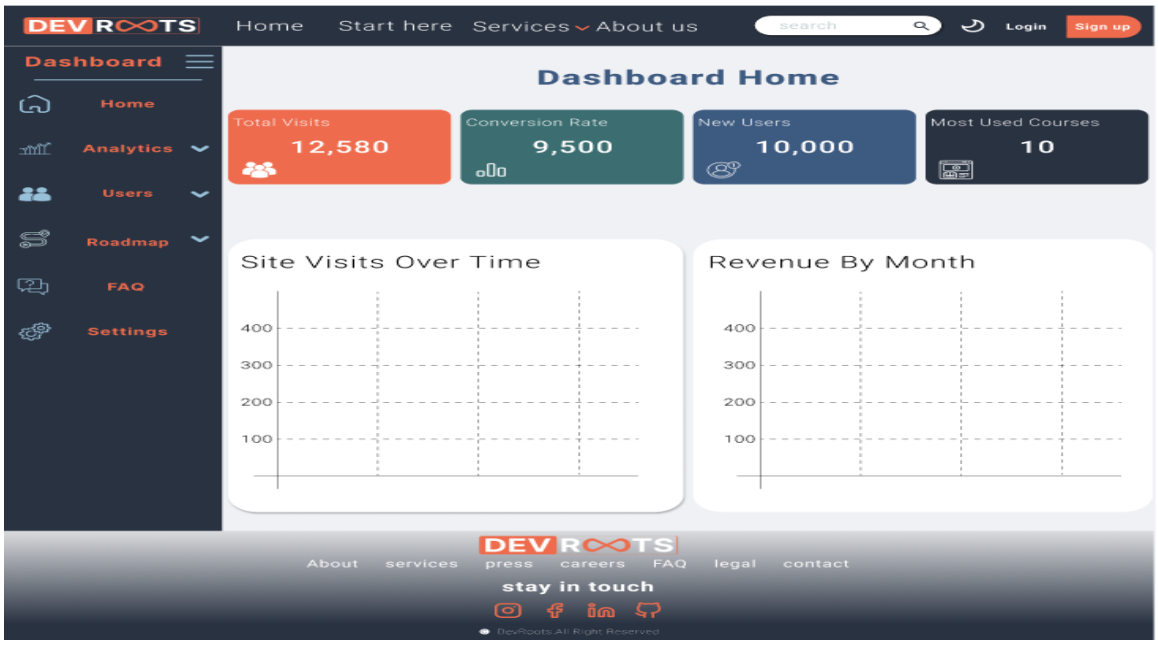

Figure 25:Dashboard Light Mode (UI)

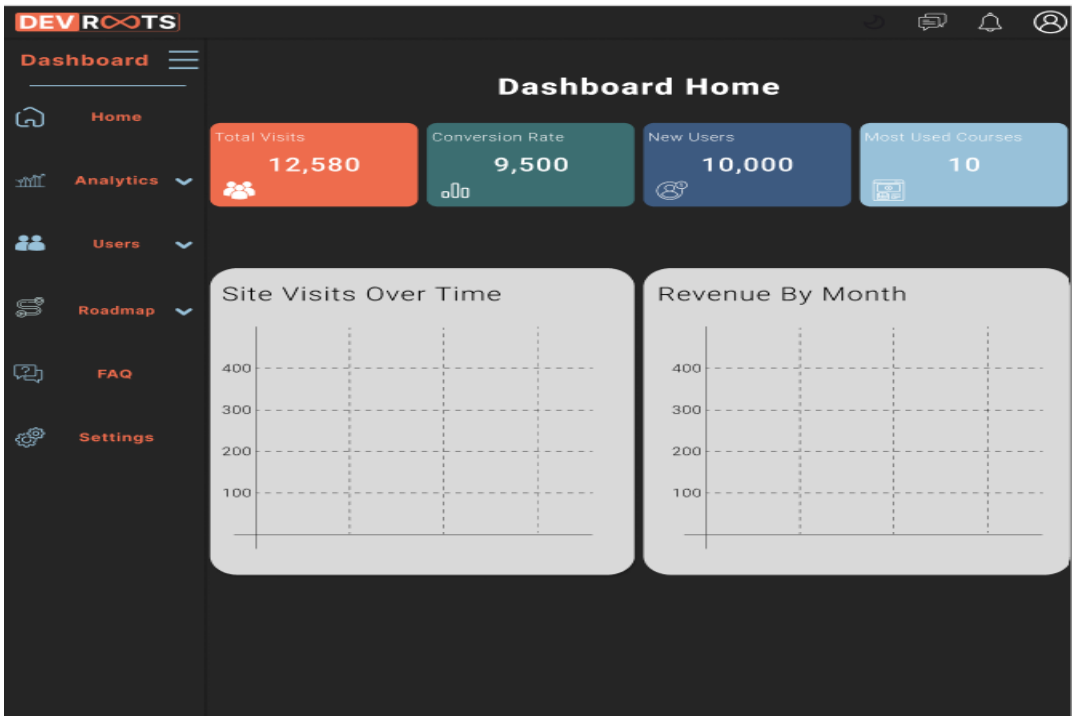

Figure 26:Dashboard Light Mode (UI)

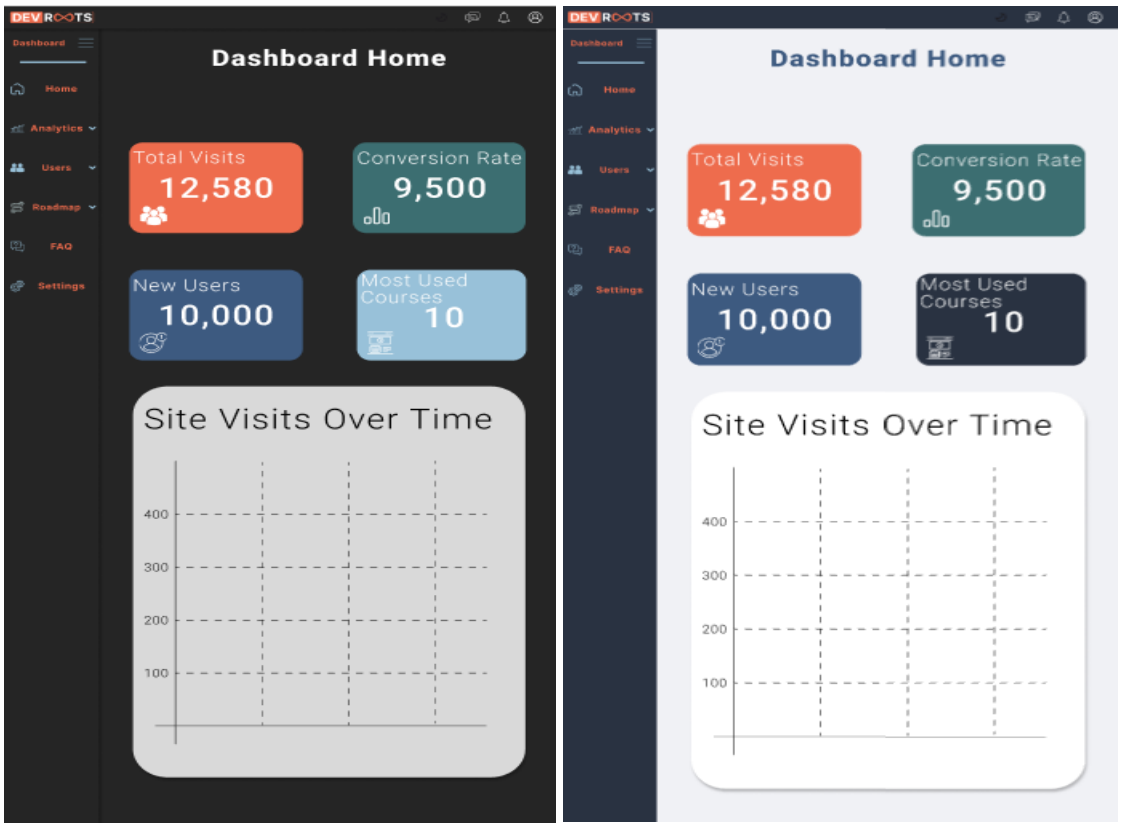

Figure 27:Dashboard home Light and dark Mode For Phone

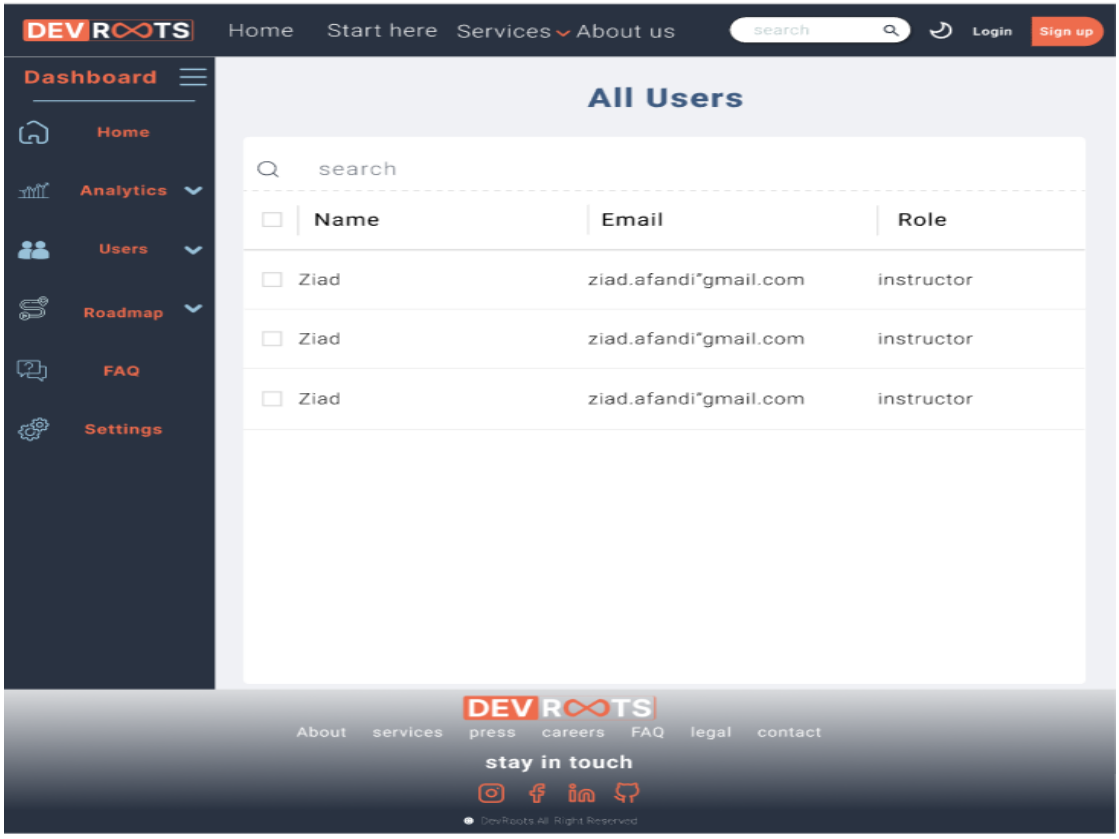

Figure 28:Dashboard All Users (UI)

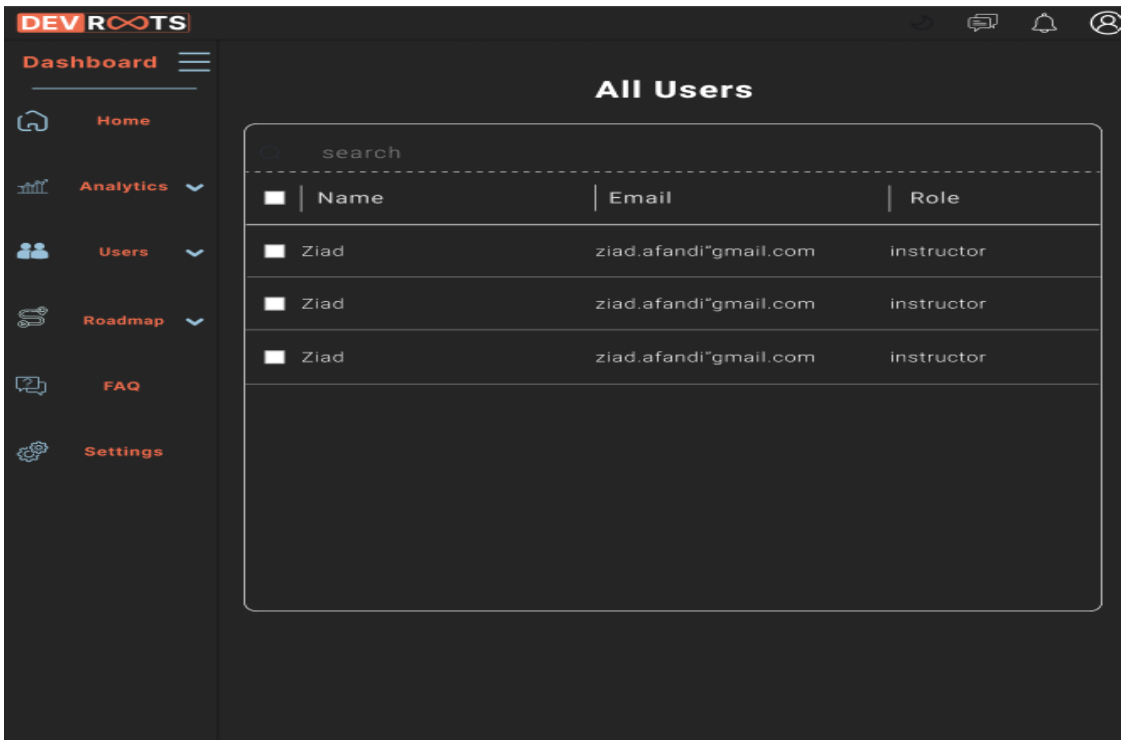

Figure 29:Dashboard All Users Dark mode (UI)

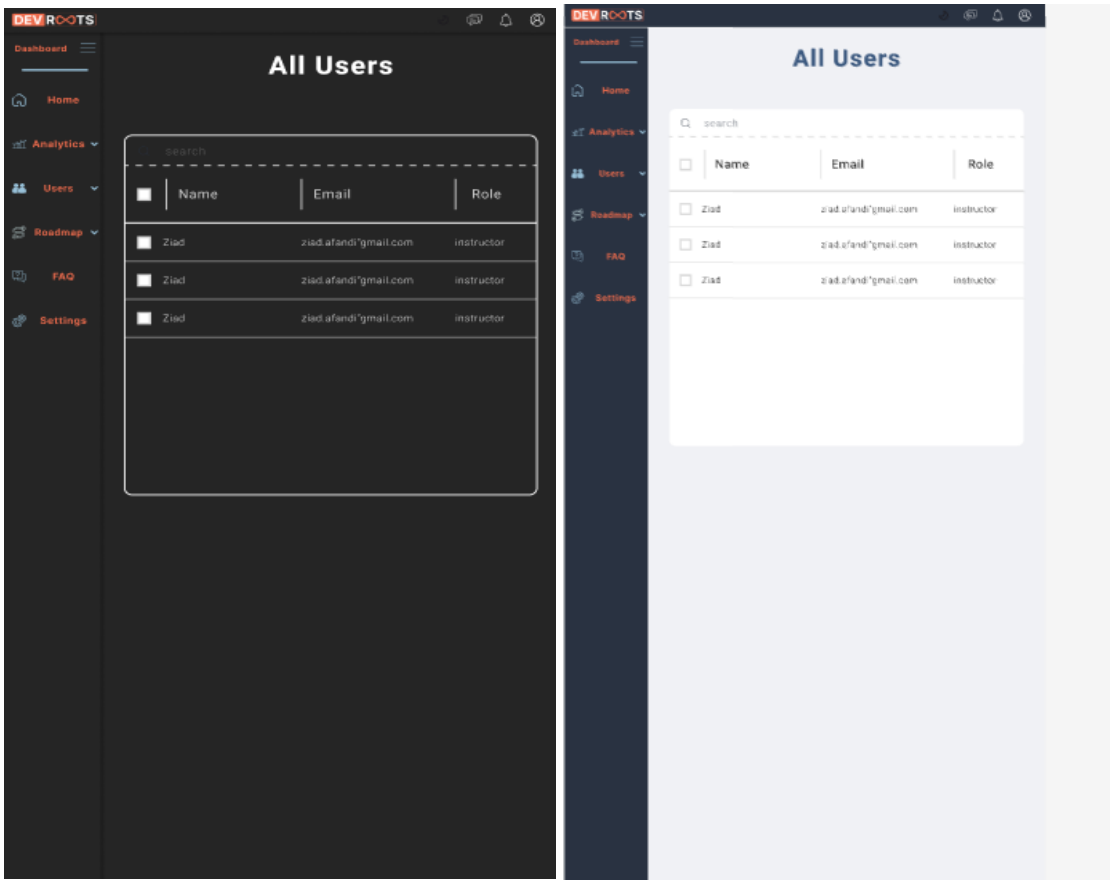

Figure 30:Dashboard All users Dark and light For Phone

```
[HttpGet("GetAllUsers")]
0 references
public async Task<IActionResult> GetAllUsers(CancellationToken cancellationToken)
{
    var result = await dashboardService.GetAllUsers(cancellationToken);
    return result.IsSuccess ? Ok(result.Value)
   : result.ToProblem();
}
```

Figure 31:Dashboard Get All Users Design code (backend)

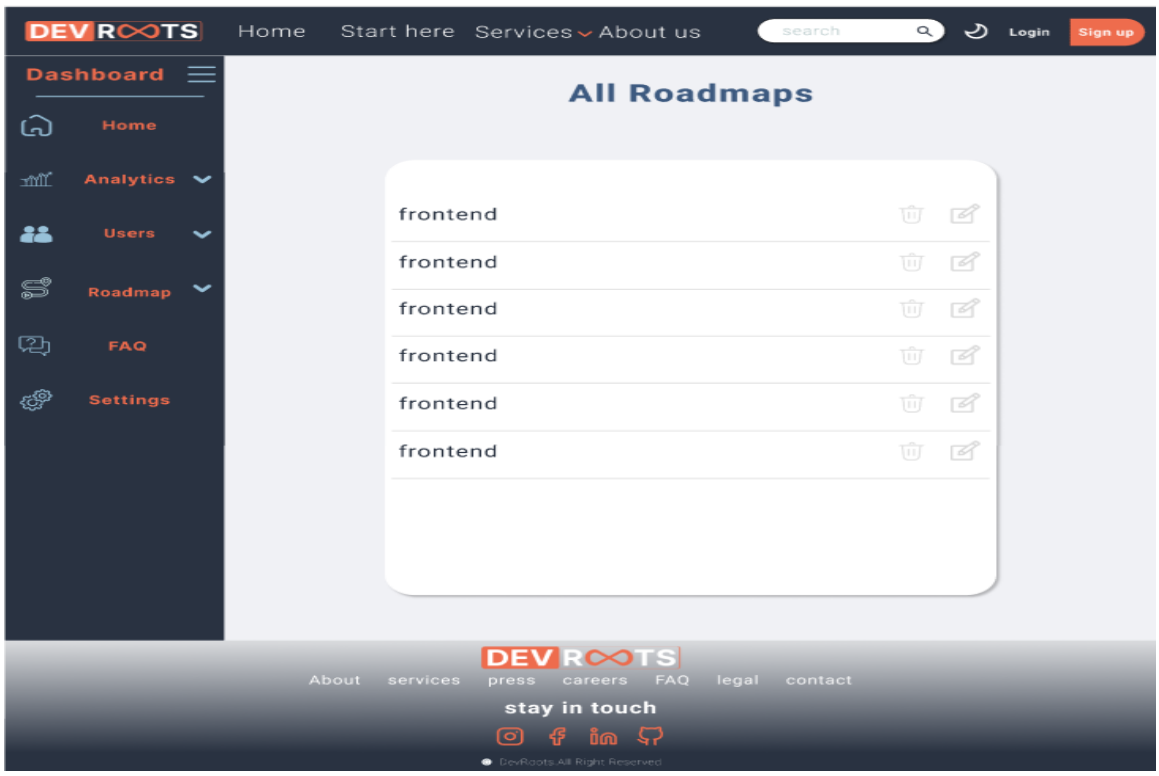

Figure 32:Dashboard All Roadmaps light mode (UI)

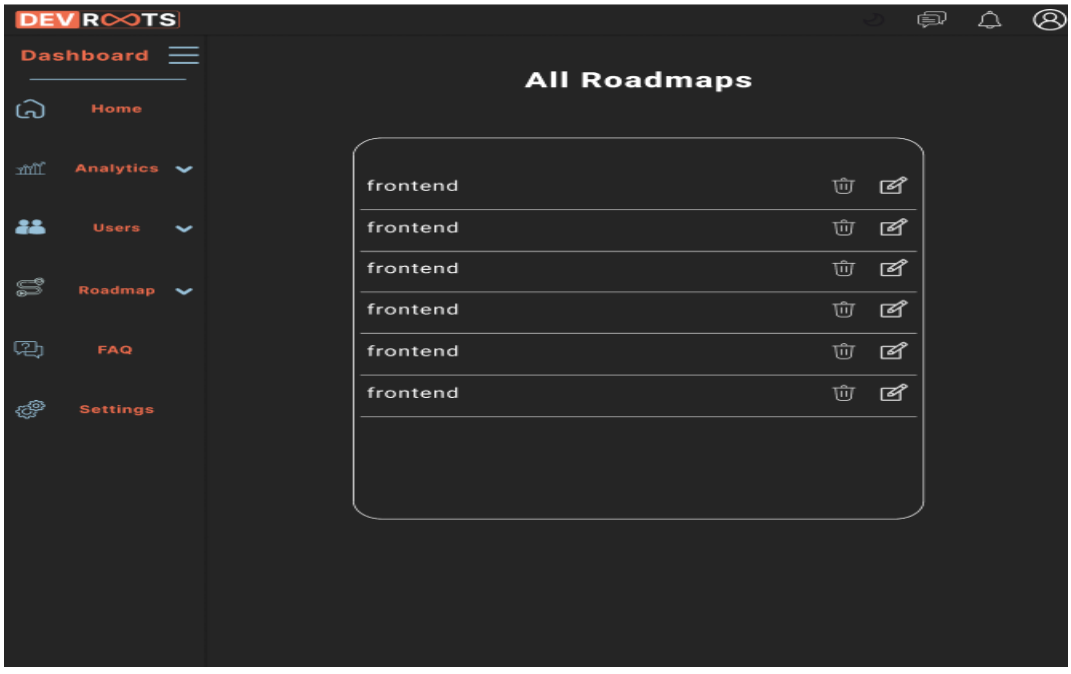

Figure 33: Dashboard All Roadmaps dark mode(UI)

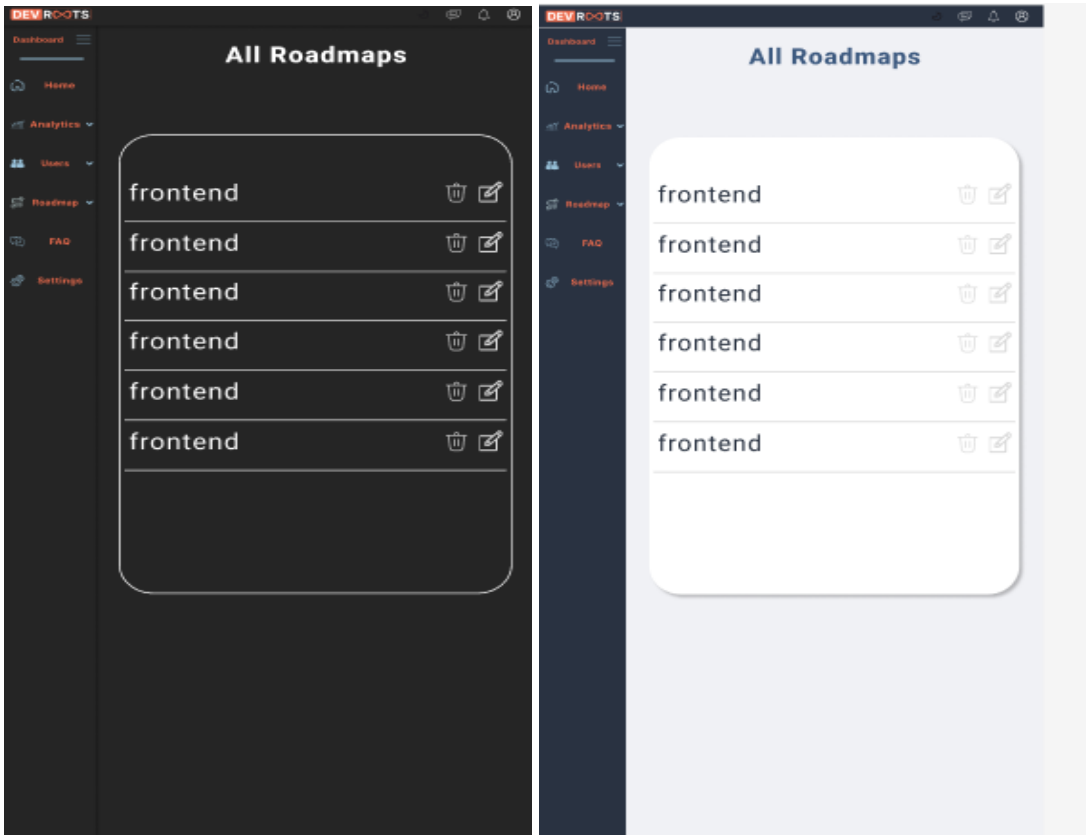

Figure 34: Dashboard All Roadmaps dark and Light mode for phone(UI)

```
[HttpGet("GetALlRoadmapsInDatabase")]
0 references
public async Task<IActionResult> GetAllRoadmapsAsync()
{
    var result = await _roadmapService.GetAllAsync();
    return Ok(result);
}
```

Figure 35:All Roadmaps Design Code (Backend)

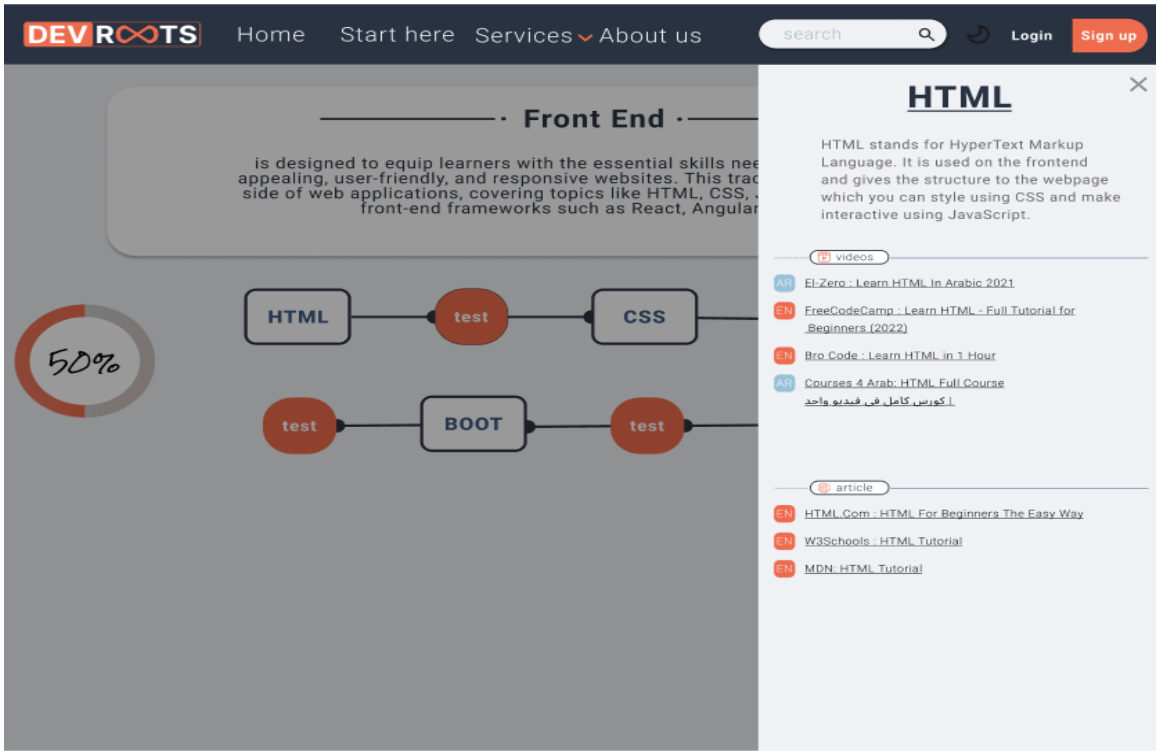

Figure 36:Links for courses in roadmap light mode

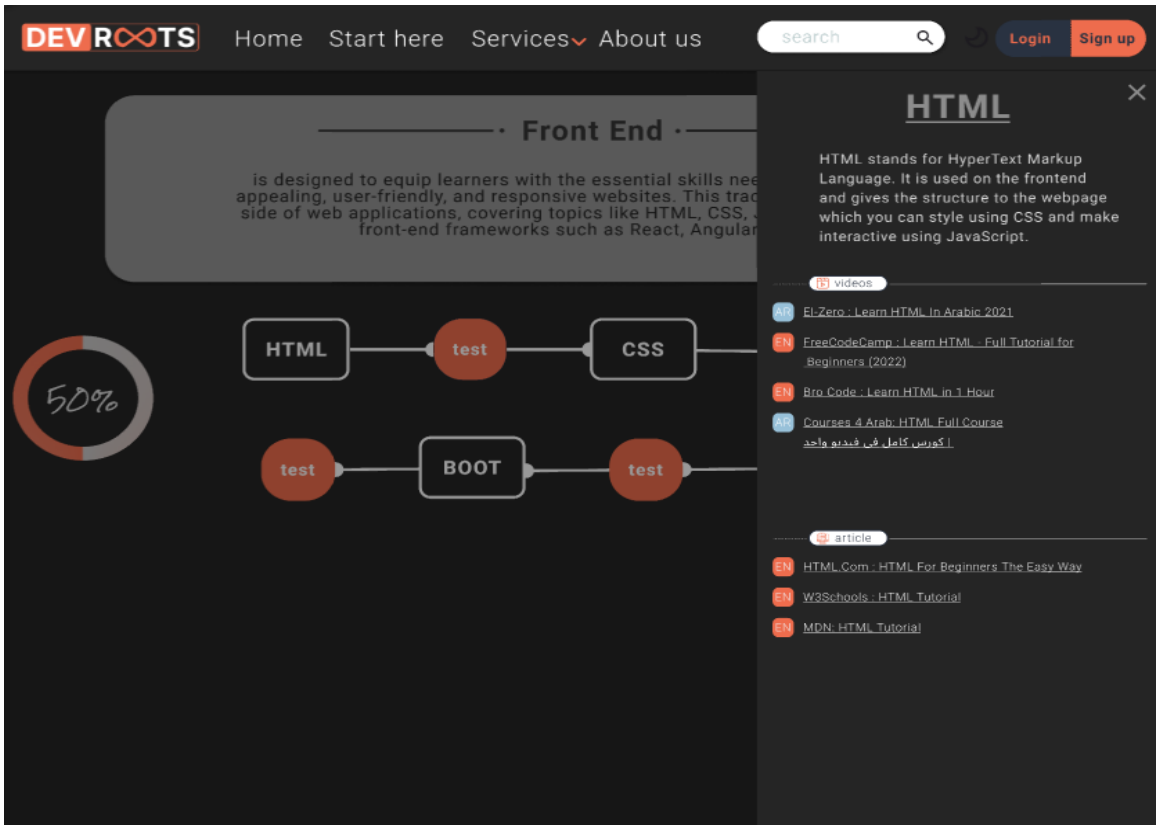

Figure 37:Links for courses in roadmap dark mode

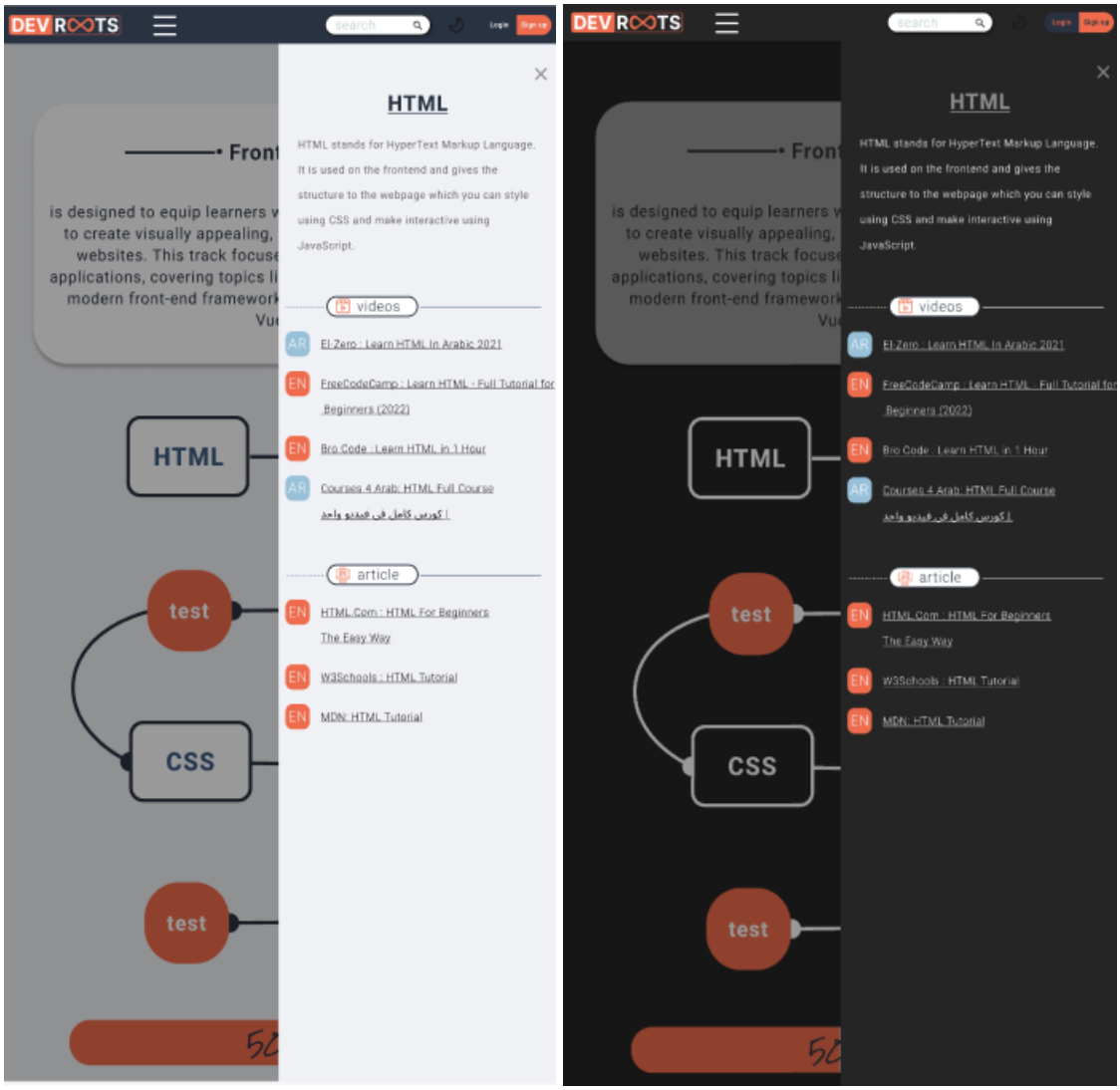

Figure 38:Links for courses in roadmap dark and light mode for phone

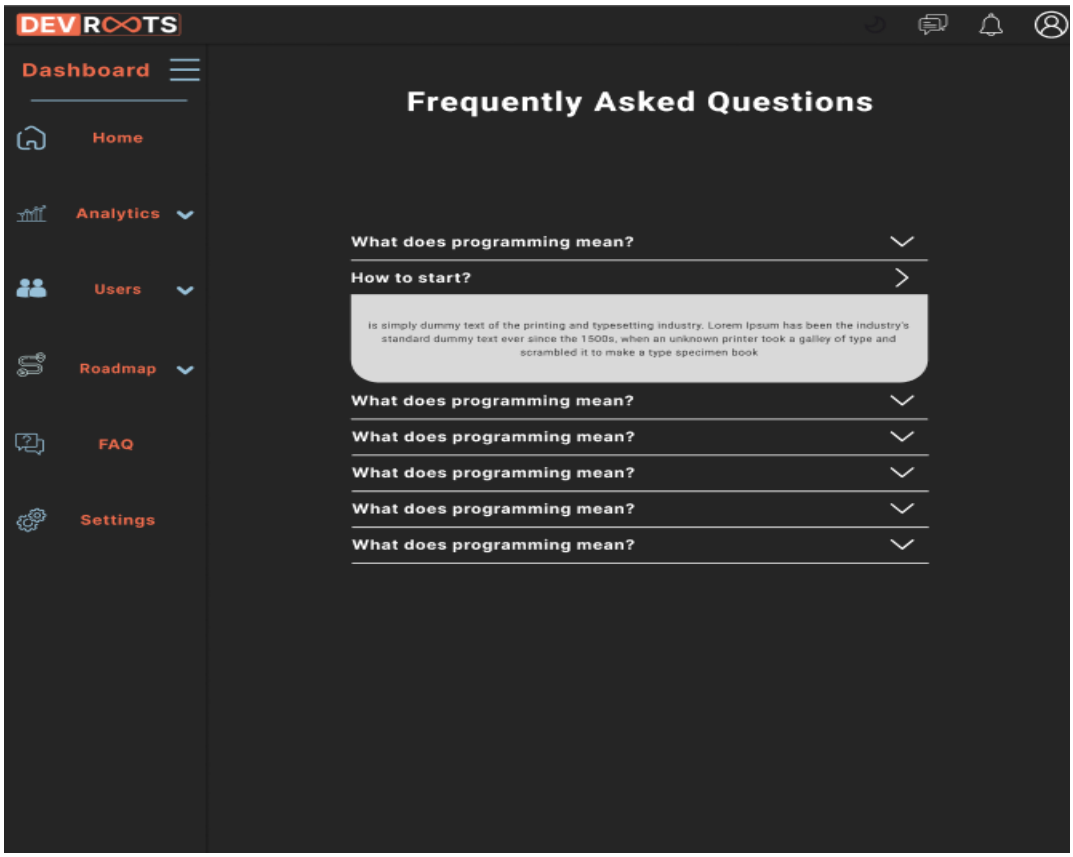

Figure 39:Dashboard for Questions Dark mode

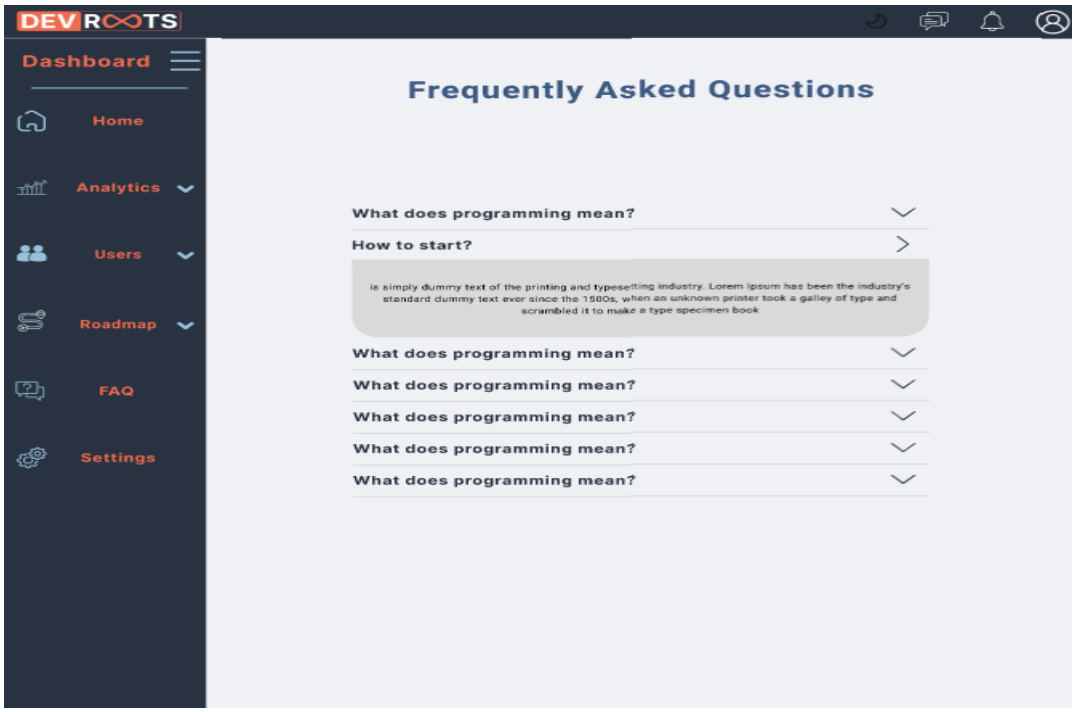

Figure 40:Dashboard for Questions Dark mode

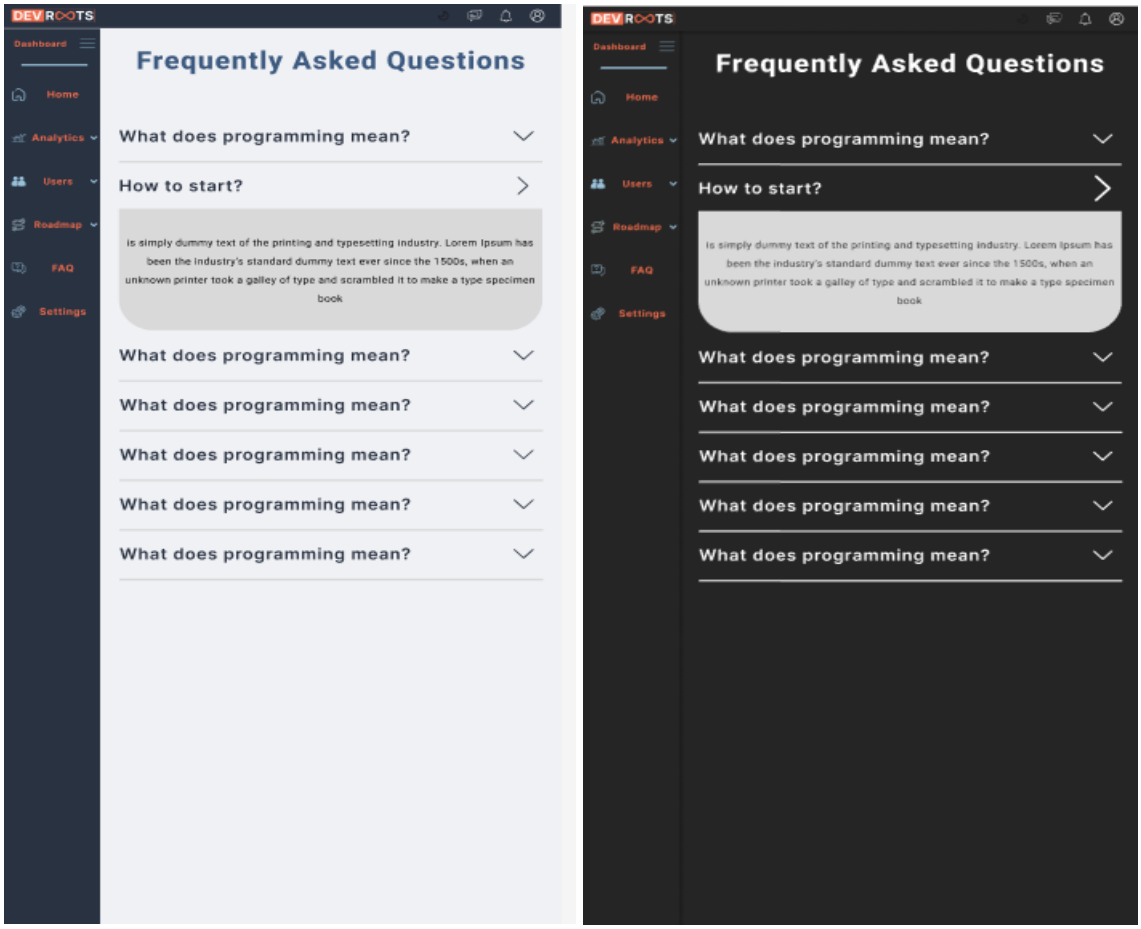

Figure 41:Dashboard for Questions Dark mode and light mode for phone

```
[HttpPost("AddQuestion")]
0 references
public async Task<IActionResult> AddQuestion(QuestionRequest request, CancellationToken cancellationToken)
{
    var result = await dashboardService.AddQuestion(request, cancellationToken);
    return result.IsSuccess ? Ok()
    : result.ToProblem();
}
[HttpGet("GetAllQuestions")]
0 references
public async Task<IActionResult> GetAllQuestions(CancellationToken cancellationToken)
{
    var result = await dashboardService.GetAllQuestions(cancellationToken);
    return result.IsSuccess ? Ok(result.Value)
    : result.ToProblem();
}
[HttpPut("UpdateQuestion/{id}")]
0 references
public async Task<IActionResult> UpdateQuestion([FromRoute] int id, [FromBody] QuestionRequest request, CancellationToken cancellationToken)
{
    var result = await dashboardService.UpdateQuestion(id, request, cancellationToken);
    return result.IsSuccess
        ? NoContent()
        : result.ToProblem();
}
```

```
[HttpDelete("DeleteQuestion/{id}")]
0 references
public async Task<IActionResult> DeleteQuestion([FromRoute] int id, CancellationToken cancellationToken)
{
    var result = await dashboardService.DeleteQuestion(id, cancellationToken);
    return result.IsSuccess
        ? NoContent()
        : result.ToProblem();
}
```

Figure 42:Questions in Dashboard Design Code (Backend)

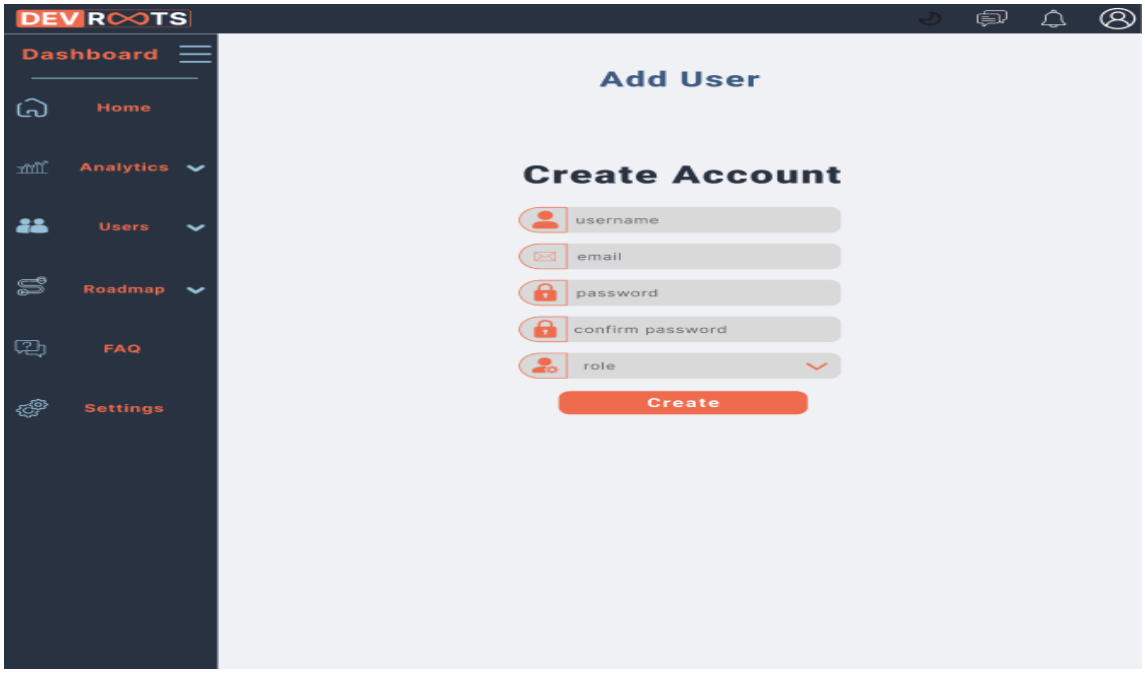

Figure 43: Dashboard Add User Light mode (UI)

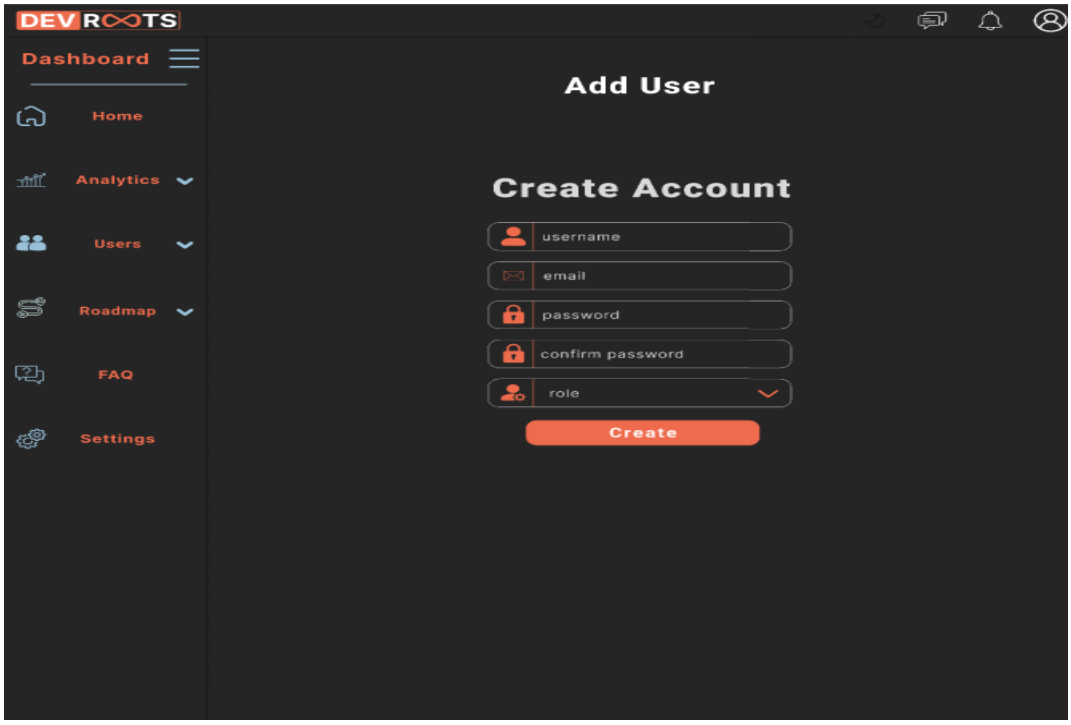

Figure 44: Dashboard Add User Light mode (UI)

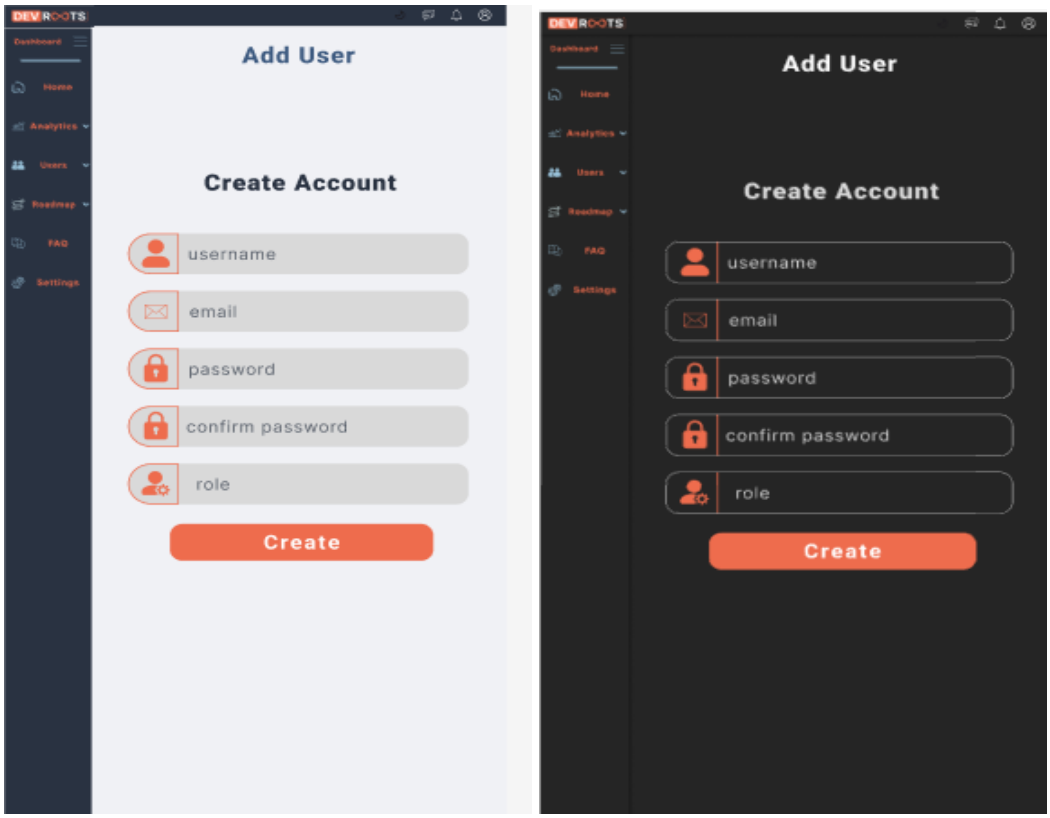

Figure 45: Dashboard Add User Light mode and dark mode for phone (UI)

```
[HttpPost("AddUser")]
0 references
public async Task<IActionResult> AddUser(AddNewUserRequest request, CancellationToken cancellationToken)
{
    var result = await dashboardService.AddNewUserAsync(request, cancellationToken);
    return result.IsSuccess ? Ok(): result.ToProblem();
}
```

Figure 46:Dashboard Add User Design Code

```jsx
import React from "react";
import {
  Grid,
  Typography,
  Card,
  CardContent,
  Box,
  Divider,
} from "@mui/material";
import { styled } from "@mui/system";
import { motion } from "framer-motion";
import {
  People,
  BarChart,
  AccessTime,
  ShoppingCart,
} from "@mui/icons-material";
import {
  LineChart,
  Line,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  BarChart as ReBarChart,
  Bar,
  PieChart,
  Pie,
  Cell,
} from "recharts";
import SourceIcon from "@mui/icons-material/Source";
import "./Dashboard.css"; // Import the CSS file
import { green, purple } from "@mui/material/colors";
// Responsive Grid container style
const GridContainerStyle = styled(Grid)(({ theme }) => ({
  marginTop: theme.spacing(2),
  marginBottom: theme.spacing(2),
}));

// InfoCard component with hover effect
const InfoCard = ({ title, value, icon, style }) => (
  <motion.div
    initial={{ opacity: 0 }}
    animate={{ opacity: 1 }}
    whileHover={{ scale: 1.05 }}
    transition={{ duration: 0.5 }}
  >
    <Card className="card" style={style}>
      {" "}
      {/* Apply the inline style here */}
      <CardContent>
        <Typography variant="h6" style={{ color: "white" }}>
          {title}
        </Typography>
        <Typography variant="h4" style={{ color: "white" }}>
          {value}
        </Typography>
```

Figure 47: Dashboard Code (Frontend)-A

```
            {icon && <Box>{icon}</Box>}
          </CardContent>
        </Card>
      </motion.div>
    );

    // Data for the charts
    const visitsData = [
      { name: "Jan", visits: 4000 },
      { name: "Feb", visits: 3000 },
      { name: "Mar", visits: 2000 },
      { name: "Apr", visits: 2780 },
      { name: "May", visits: 1890 },
    ];

    const revenueData = [
      { name: "Jan", revenue: 2400 },
      { name: "Feb", revenue: 3000 },
      { name: "Mar", revenue: 2800 },
      { name: "Apr", revenue: 3500 },
    ];

    const userDistributionData = [
      { name: "New Users", value: 400 },
      { name: "Main Users", value: 300 },
      { name: "Guest Users", value: 300 },
    ];
```

Figure 48:Dashboard Code (Frontend)-B

```
86
87    // Dashboard component
88    const Dashboard = () => (
89      <>
90        <Grid item xs={12} className="infoCardsText">
91          <Typography
92            className="dashboard-title"
93            sx={{
94              fontSize: 40,
95              fontWeight: "bold",
96              color: "#293241",
97              mt: 8,
98              textAlign: "center",
99            }}
100         >
101           Dashboard Home
102         </Typography>
103       </Grid>
104       <GridContainerStyle container spacing={2}>
105         {/* Info Cards with Icons */}
106         <Grid item xs={12} sm={6} md={4} lg={3}>
107           <InfoCard
108             title="Total Visits"
109             value="12,345"
110             icon={<People />}
111             style={{ backgroundColor: "#ee6c4d", color: "white" }}
112           />
113         </Grid>
114         <Grid item xs={12} sm={6} md={4} lg={3}>
115           <InfoCard
116             title="Conversion Rate"
117             value="30%"
118             icon={<BarChart />}
119             style={{ backgroundColor: "#3c6e71", color: "white" }}
120           />
121         </Grid>
122         <Grid item xs={12} sm={6} md={4} lg={3}>
123           <InfoCard
124             title="New Users"
125             value="789"
126             icon={<AccessTime />}
127             style={{ backgroundColor: "#284b63", color: "white" }}
128           />
129         </Grid>
130         <Grid item xs={12} sm={6} md={4} lg={3}>
131           <InfoCard
132             title="Most Used Courses"
133             value="5"
134             icon={<SourceIcon />}
135             style={{ backgroundColor: "#293241", color: "white" }}
136           />
137         </Grid>
138       </GridContainerStyle>
```

Figure 49:Dashboard Code (Frontend)-C

```
      {/* charts */}
      <Grid container spacing={2}>
        {/* Line Chart */}
        <Grid item xs={12} md={6}>
          <Card className="chart-card">
            <CardContent>
              <Typography variant="h6">Site Visits Over Time</Typography>
              <Divider />
              <LineChart width={500} height={300} data={visitsData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis dataKey="name" />
                <YAxis />
                <Tooltip />
                <Legend />
                <Line
                  type="monotone"
                  dataKey="visits"
                  stroke="#8884d8"
                  activeDot={{ r: 8 }}
                />
              </LineChart>
            </CardContent>
          </Card>
        </Grid>

        {/* New Bar Chart - Revenue by Month */}
        <Grid item xs={12} md={6}>
          <Card className="chart-card">
            <CardContent>
              <Typography variant="h5">Revenue by Month</Typography>
              <Divider />
              <ReBarChart width={300} height={300} data={revenueData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis dataKey="name" />
                <YAxis />
                <Tooltip />
                <Legend />
                <Bar dataKey="revenue" fill="#ee6c4d" />
              </ReBarChart>
            </CardContent>
          </Card>
        </Grid>
      </Grid>
    </>
  );

export default Dashboard;
```

Figure 50:Dashboard Code (Frontend)-D

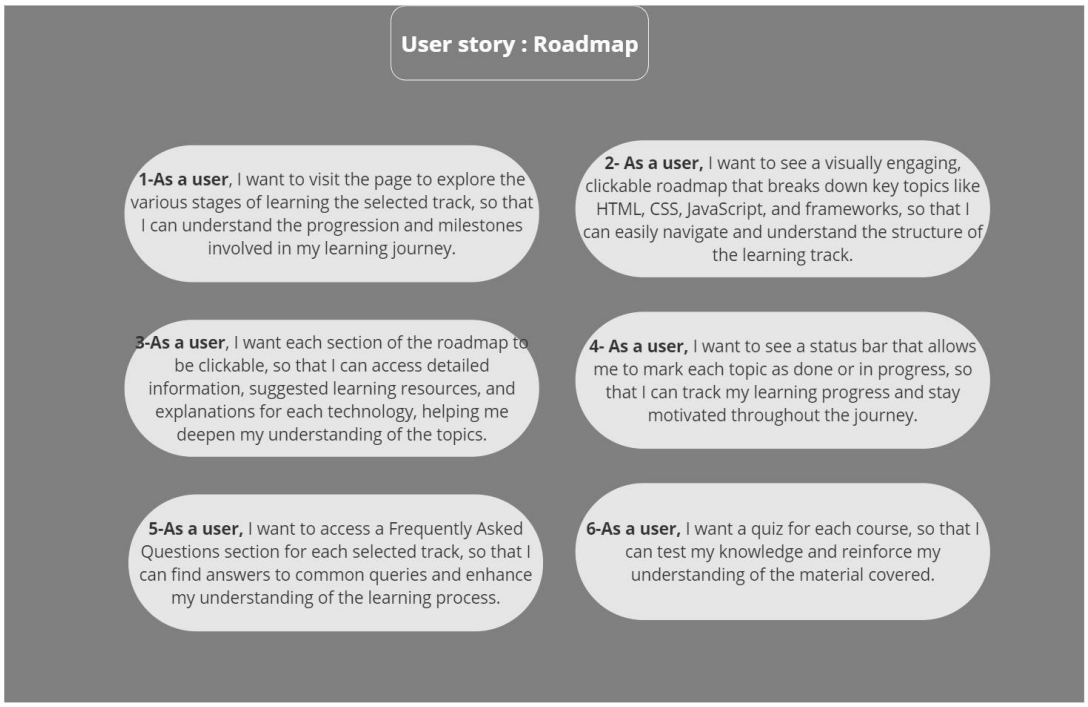

Figure 51:Roadmap Design (User Story)

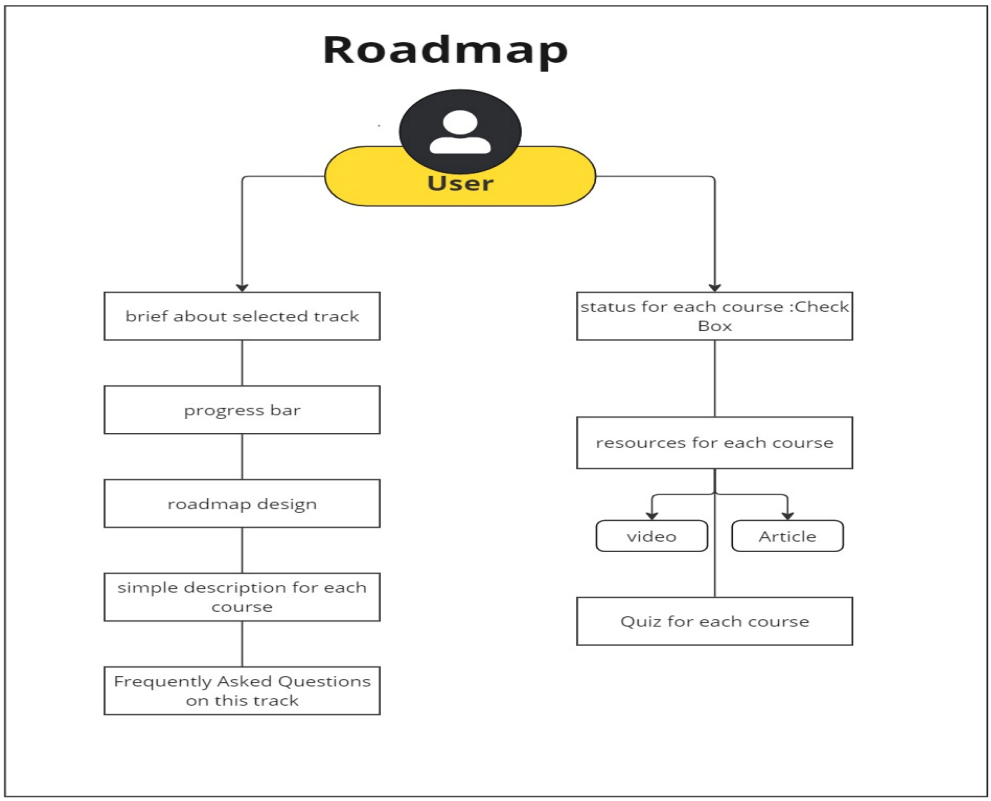

Figure 52:Roadmap Design (Use Case)

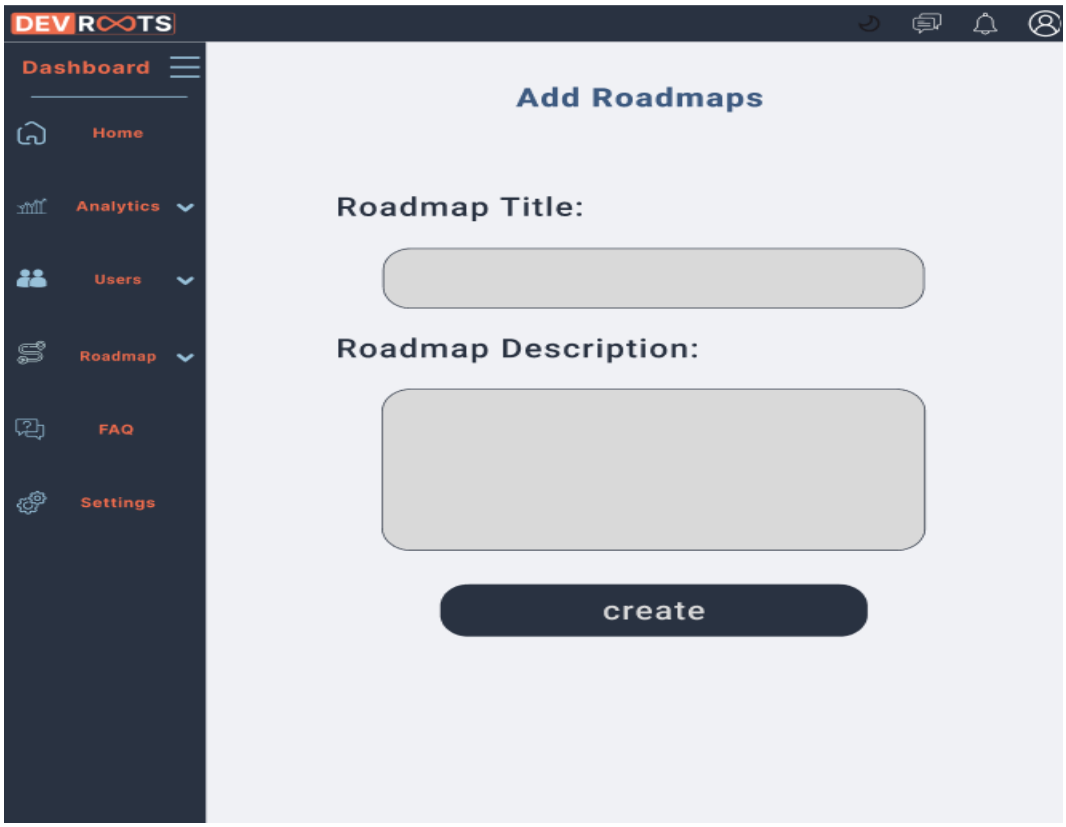

Figure 53:Add Roadmap Light Mode(UI)

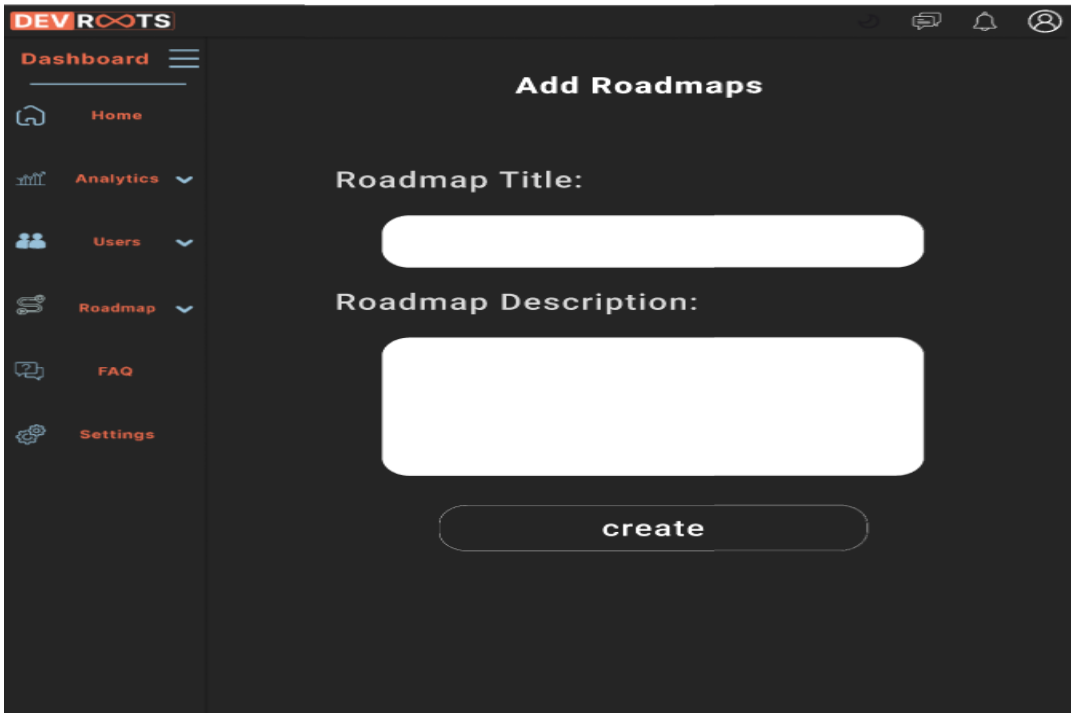

Figure 54:Add Roadmap dark Mode(UI)

```
[HttpPost("AddDataForRoadmap")]
0 references
public async Task<IActionResult> AddDataForRoadmap([FromBody] AllRoadmapInsertedDto roadmapDto, CancellationToken cancellationToken)
{

    // التأكد من صحة البيانات
    if (!ModelState.IsValid)
    {
        return BadRequest("Invalid data.");
    }

    try
    {
        //var filteredData = await _roadmapService.AddFilteredRoadmapAsync(roadmapDto, cancellationToken);

        var allDataRoadmap = await _roadmapService.AddAllRoadmapAsync(roadmapDto, cancellationToken);

        return Ok(allDataRoadmap);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error: {ex.Message}");
    }

}
```

Figure 55:Add Roadmap Design code(backend)

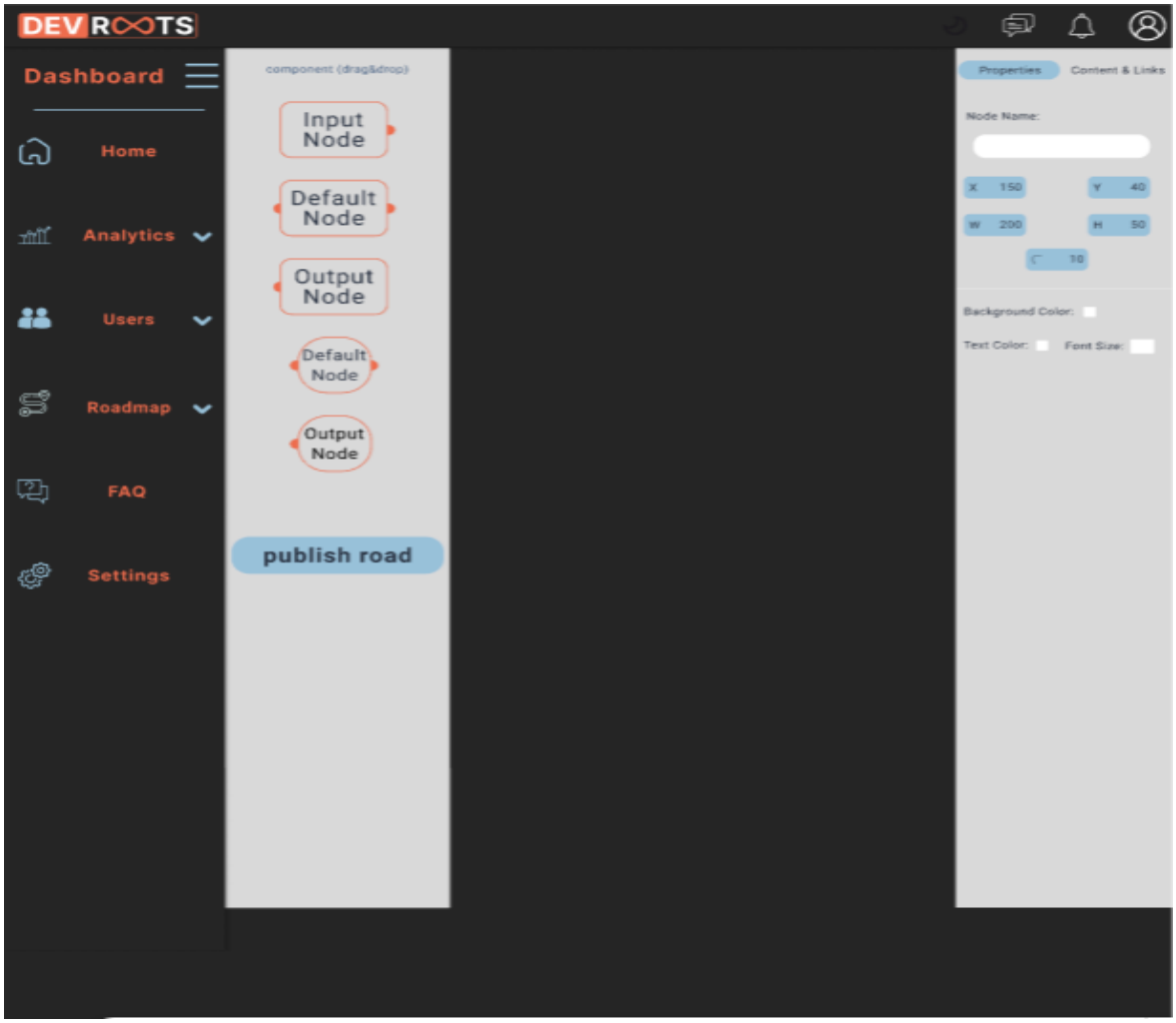

Figure 56:Dashboard Add Roadmaps properties Dark Mode

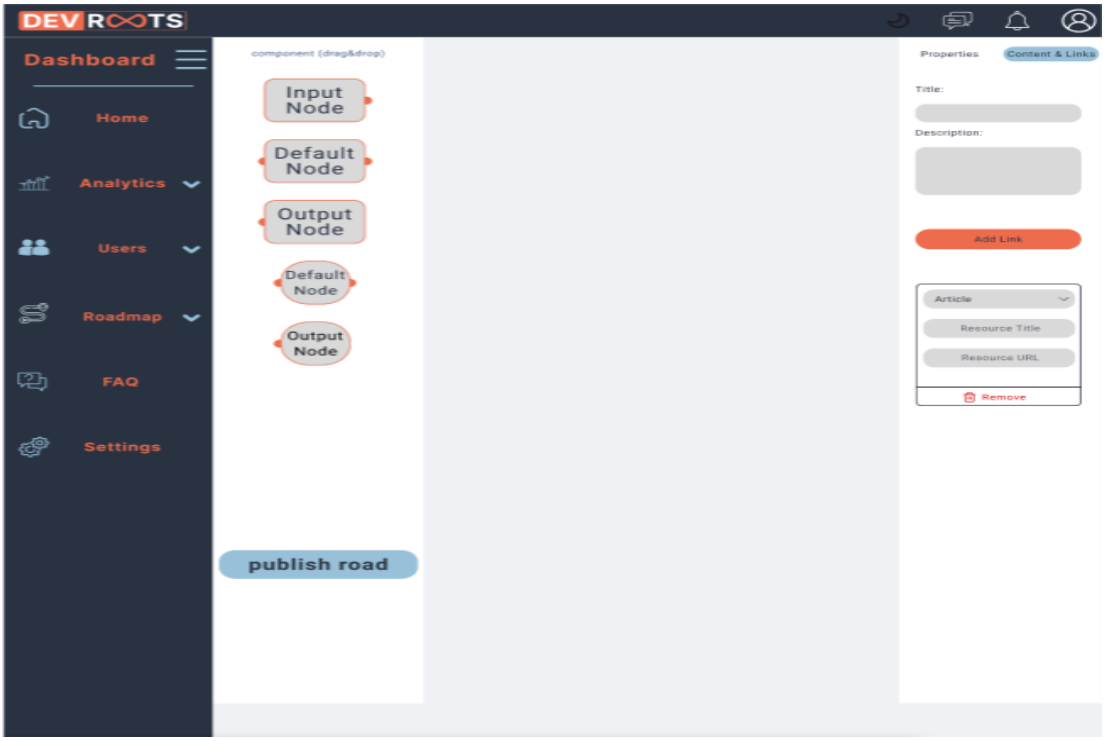

Figure 57:Dashboard Add Roadmaps content & links Light Mode

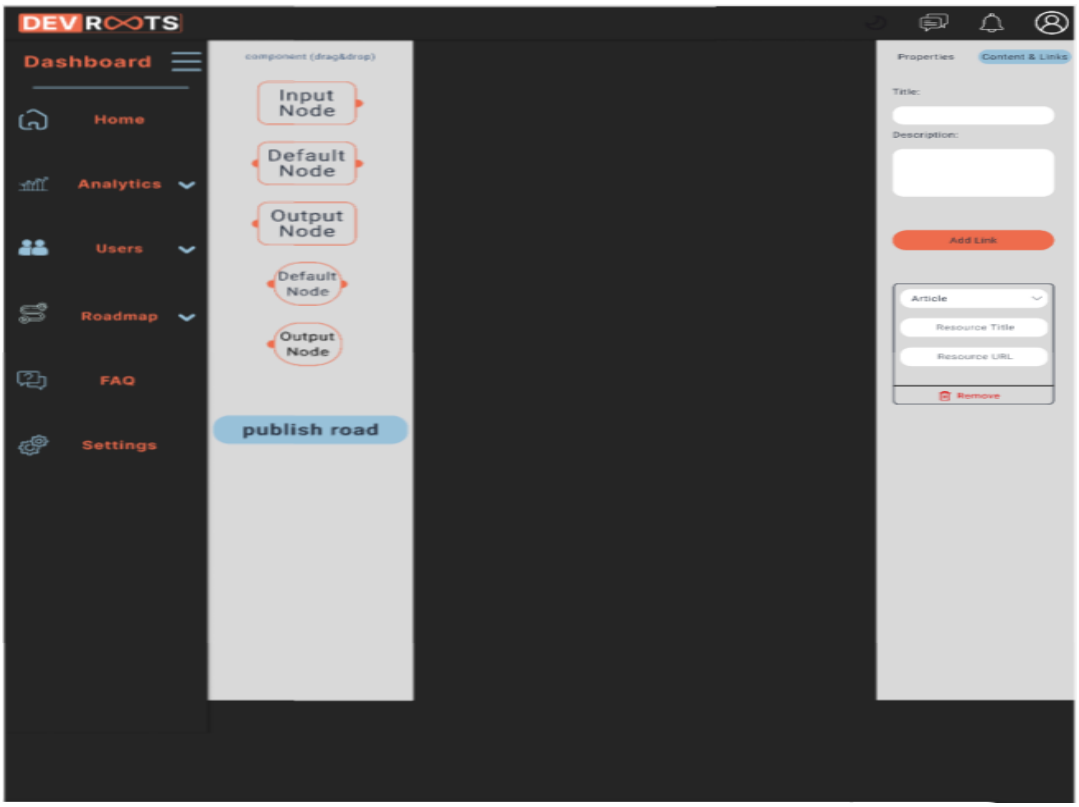

Figure 58:Dashboard Add Roadmaps content & links Dark Mode

```jsx
import React, { useState } from "react";
import TextField from "@mui/material/TextField";
import { Box, Button } from "@mui/material";
import { useNavigate } from "react-router-dom";

export default function RoadmapDetails() {
  const [roadmapName, setRoadmapName] = useState("");
  const [roadmapDescription, setRoadmapDescription] = useState("");
  const navigate = useNavigate();

  const handleCreate = () => {
    // Save values to local storage
    localStorage.setItem("roadmapName", roadmapName);
    localStorage.setItem("roadmapDescription", roadmapDescription);

    // Navigate to /create
    navigate("/create");
  };

  return (
    <Box sx={{ mt: 2 }}>
      <TextField
        id="outlined-basic"
        label="Roadmap Name"
        variant="outlined"
        value={roadmapName}
        onChange={(e) => setRoadmapName(e.target.value)}
      />
```

Figure 59:Add Roadmap Design Code (Frontend)-A

```jsx
      <TextField
        id="outlined-multiline-flexible"
        label="Roadmap Description"
        multiline
        value={roadmapDescription}
        onChange={(e) => setRoadmapDescription(e.target.value)}
      />
      <Button variant="contained" onClick={handleCreate}>
        Create
      </Button>
    </Box>
  );
}
```

Figure 60:Add Roadmap Design Code (Frontend)-B

```csharp
    [HttpDelete(template: "Delete {Id}")]
    0 references
    public async Task<IActionResult> DeleteAsync([FromRoute] int Id, CancellationToken cancellationToken)
    {
        var isDlt = await _Services.DeleteAsync(Id, cancellationToken);

        if (isDlt == null)
            return NotFound();
        return NoContent();
    }
}
```

Figure 61:Roadmap Design Code (Backend)

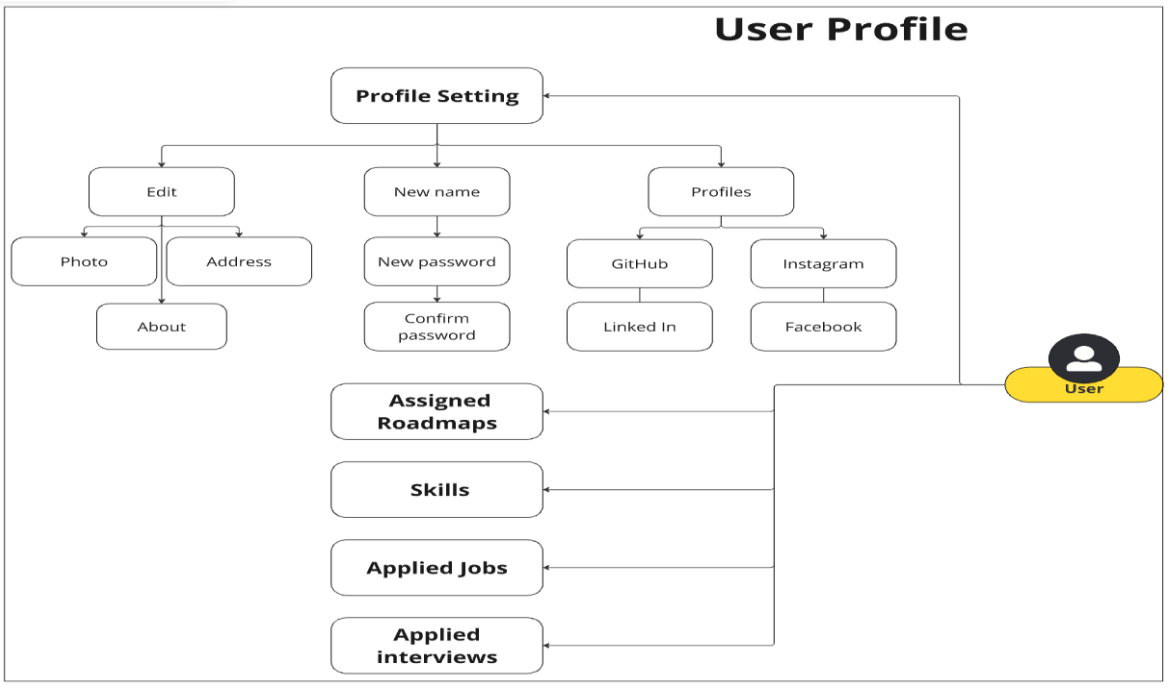

Figure 62:User Profile (Use Case)

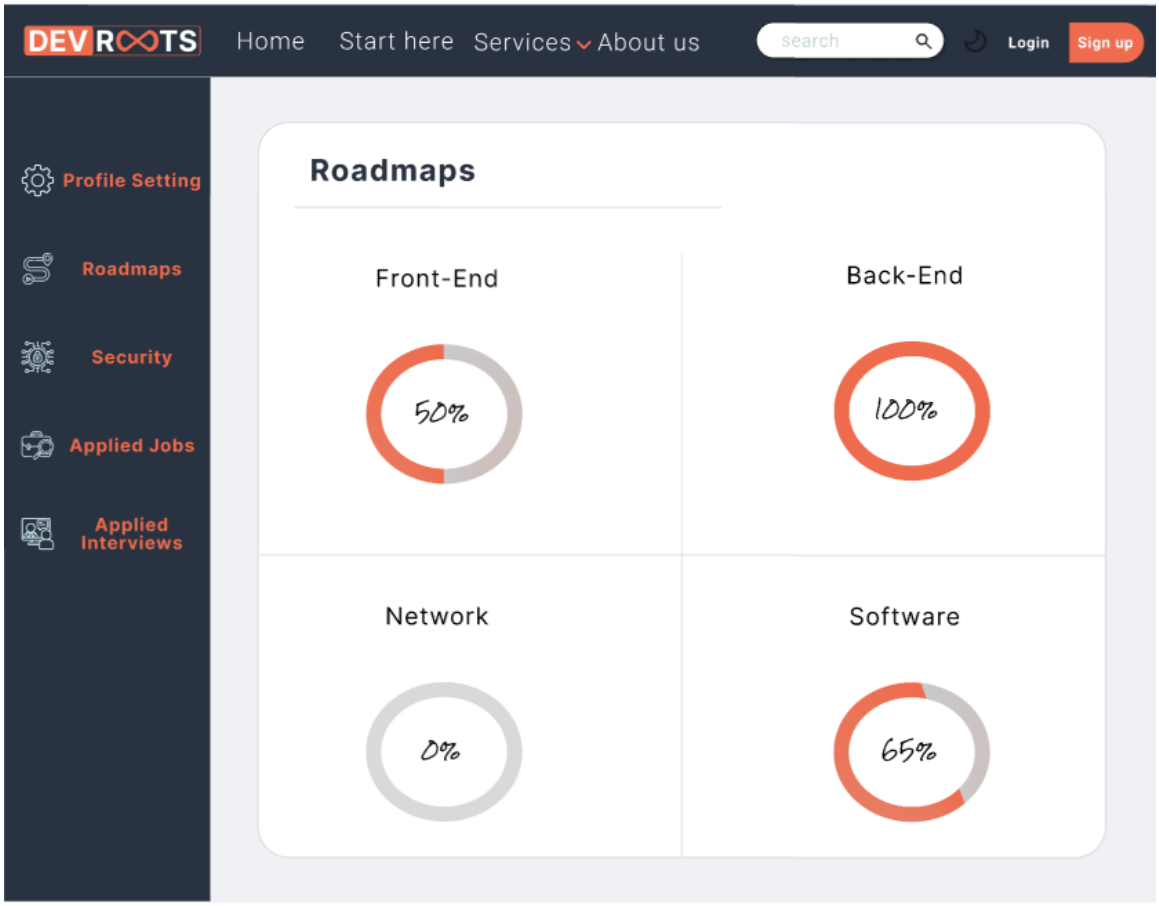

Figure 63:Profile Roadmap (UI)

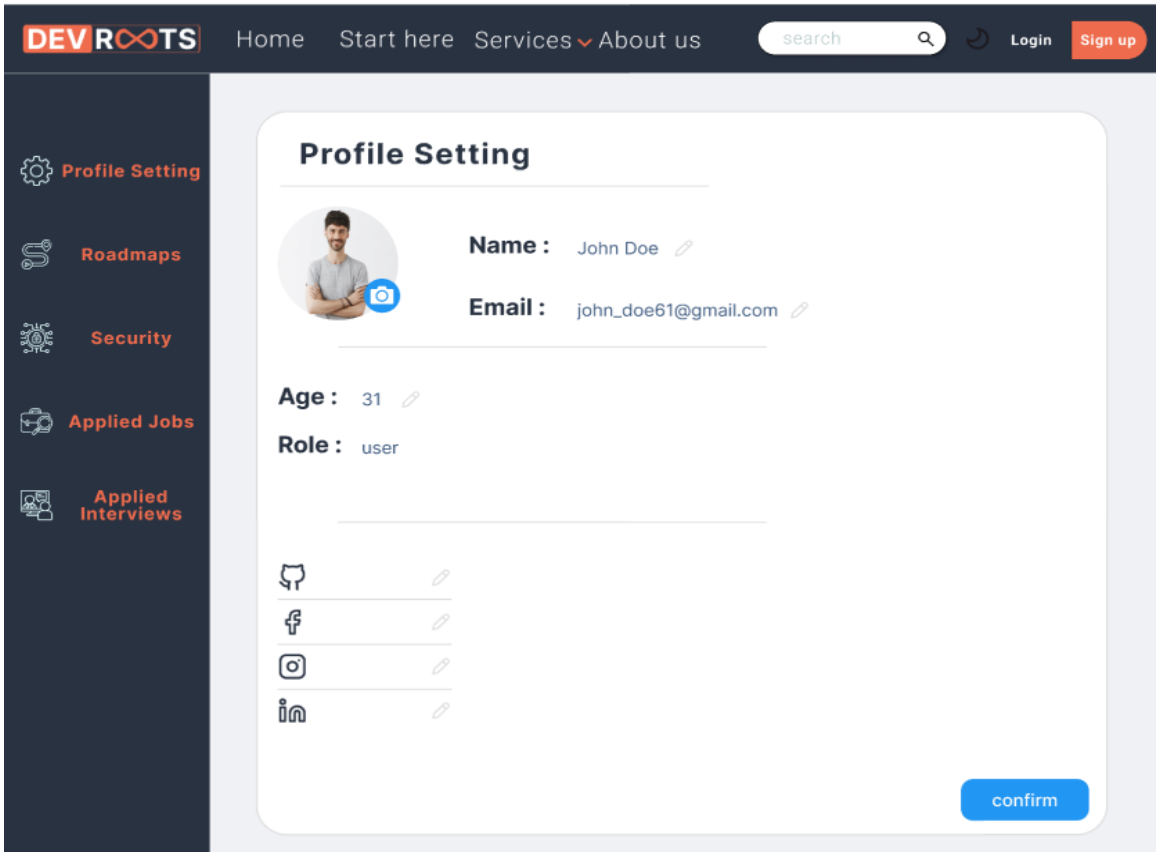

Figure 64:Profile Setting Design (UI)

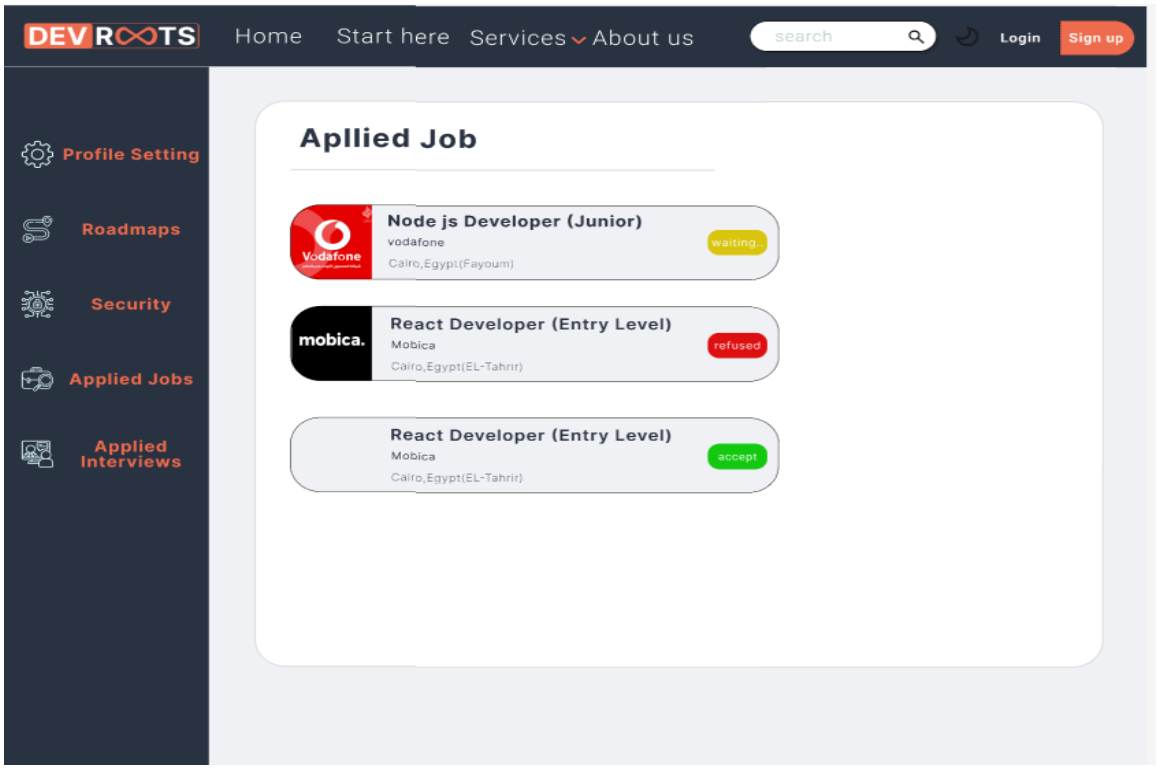

Figure 65:Applied Job(UI)

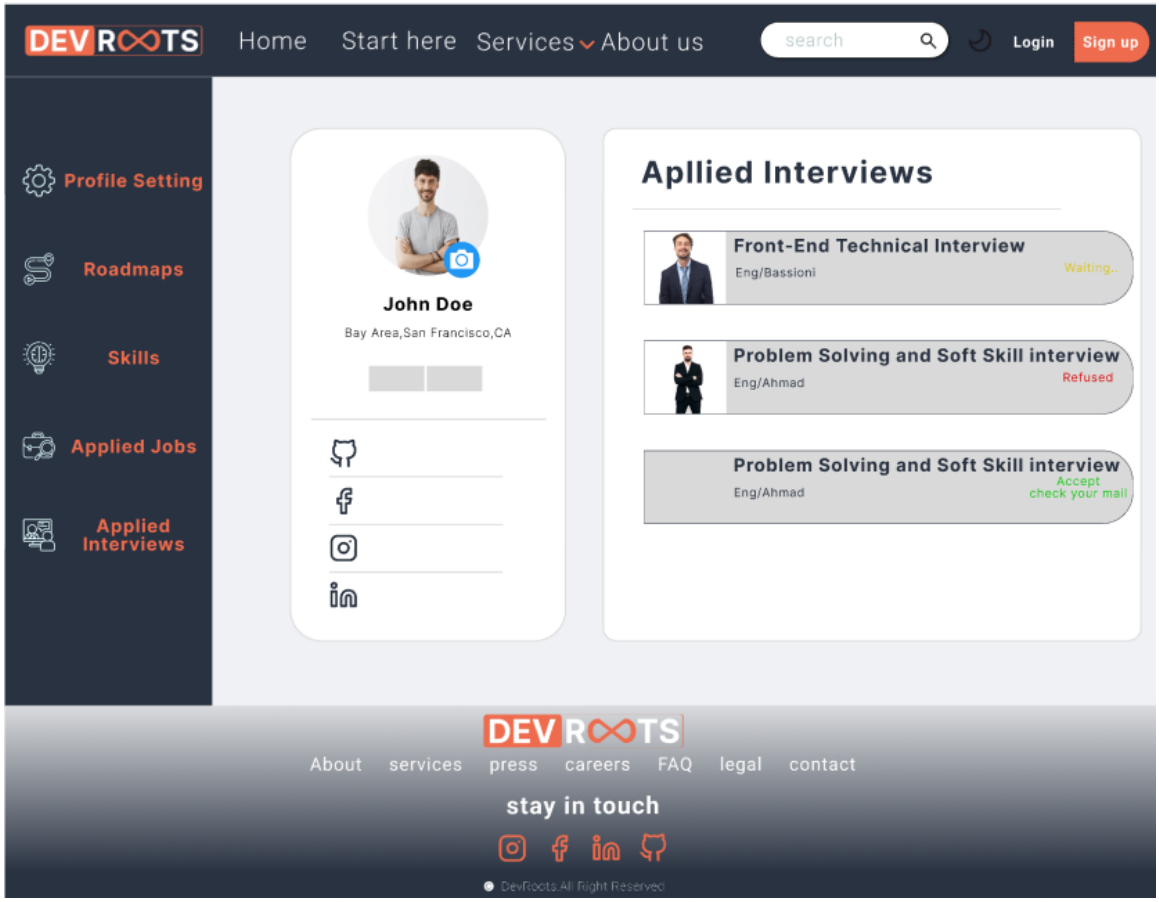

Figure 66:InterView(UI)

Chapter Four

Tools & Dependences

```
"dependencies":
 {

    "@emotion/react": "^11.13.3",
    "@emotion/styled": "^11.13.0",
    "@mui/material": "^6.1.1",
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "@xyflow/react": "^12.3.1",
    "axios": "^1.7.7",
    "json-server": "^1.0.0-beta.2",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-flow-renderer": "^10.3.17",
    "react-resizable": "^3.0.5",
    "react-router-dom": "^6.26.2",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
 },
```

```
 Libraries and Tools:

- HTML, CSS, JS
- ElZero Web School
- SEF online React fronent course
- ITI One Month angular
- React full course youtube "Courses 4 Arab" Channel
- IDE visual studio code (Frontend)
- IDE visual studio community (Backend)
```