# AIR University Islamabad

# HomeToHome

(A Service Exchange Platform)

## Submitted By:

Afia Aziz BSCS-F-23-B-231561

## Instructor

Miss Sara Ibrahim

**Course Name & Code:** Visual Programming (CS-304)
**Semester:** Spring 2025
**Date of Submission:** 3rd June, 2025

# 1. Introduction:

**HomeToHome** is a dynamic, web-based application developed using Blazor and modern Visual Programming tools. The platform acts as a smart digital bridge between homeowners and domestic service providers—including plumbers, cleaners, electricians and other skilled helpers. It simplifies the hiring process by allowing both users and workers to register, log in, and manage their profiles through an intuitive and structured interface.

The application offers a seamless user experience with role-based access, where homeowners can easily search, filter, and connect with available service providers, while workers can showcase their skills and availability. By digitalizing this process, **HomeToHome** enhances accessibility, convenience, and trust for both parties.

## Purpose and Scope:

The primary **purpose** of the HomeToHome application is to **streamline the process of hiring domestic service providers** by connecting homeowners with trusted professionals through a centralized platform. Traditionally, finding reliable home services can be time-consuming and unreliable. This system eliminates the middleman, making the process **faster, transparent, and accessible online**.

The **scope** of the project includes:

- Developing a user-friendly web portal using **Blazor Server**.
- Implementing **secure authentication and authorization** for both users and workers.
- Enabling **search and filtering of service providers** based on category, name, and availability.
- Supporting **service request management**, including creating, reviewing, and tracking requests.
- Maintaining **data consistency** with a fully integrated **relational database** using **ADO.NET**.
- Incorporating session and local storage for better state management in the browser.

This project demonstrates the application of key Visual Programming concepts, ensuring a real-world, production-ready software solution.

## Target Audience / Users:

The HomeToHome application is tailored to meet the needs of two primary user groups:

### i. Homeowners / Clients:

These are individuals who require assistance with home-related tasks such as cleaning, plumbing, electrical work, or general repairs. The application provides them with:

- Easy registration and login process.
- Ability to browse available workers by category.
- A streamlined interface to request services, track status, and view worker details.
- Direct interaction with workers through a secure and structured system.

### ii. Service Providers / Domestic Workers:

These users are individuals offering their services in various domestic categories. The platform empowers them by:

- Allowing them to create and manage their own worker profiles.
- Receiving and responding to service requests from clients.
- Displaying their skills, availability, and contact details in a professional layout.
- Gaining visibility and reaching more potential clients without intermediaries.

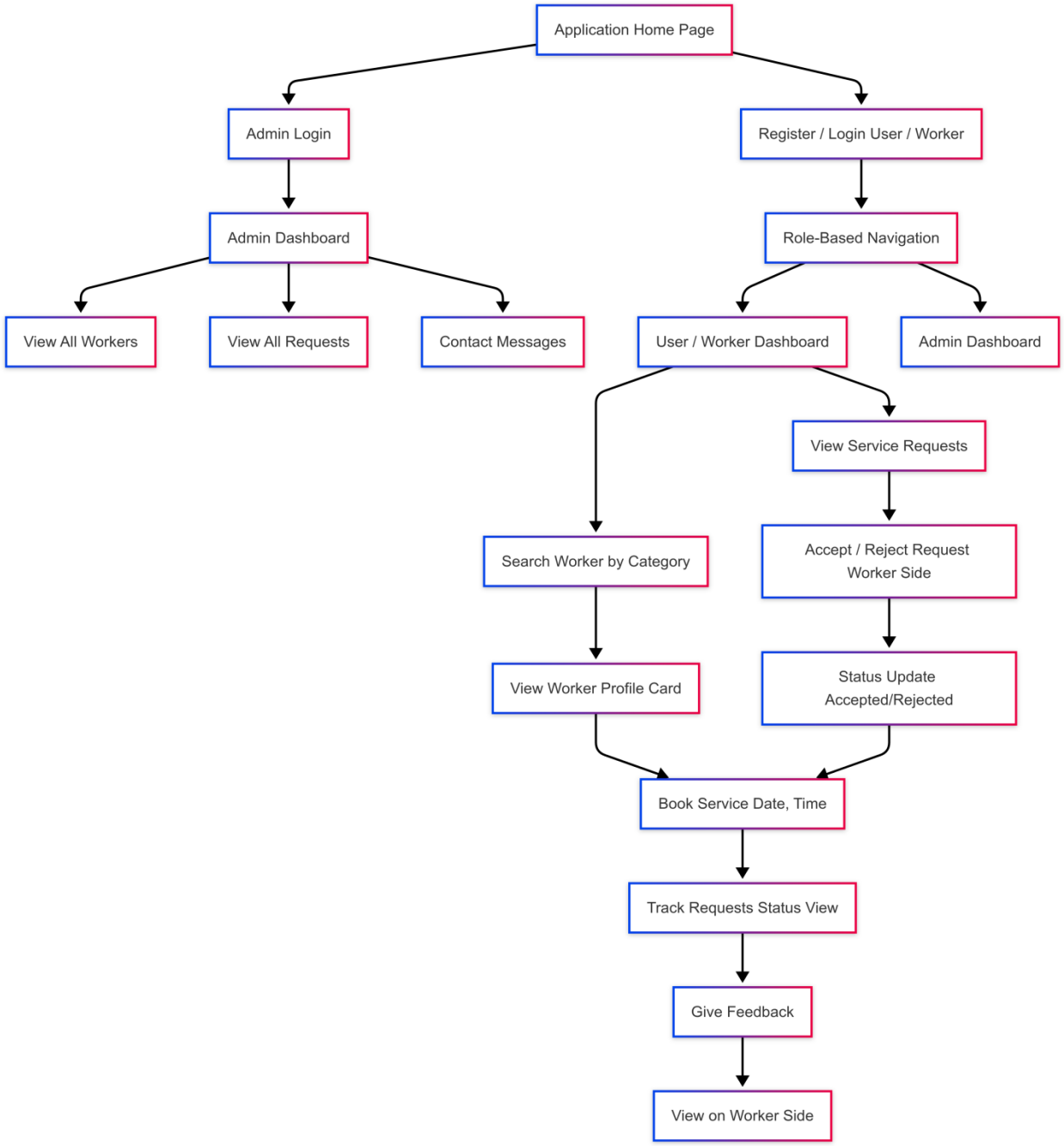
## 2. List of Features:

The HomeToHome web application offers a range of features designed to facilitate seamless interaction between homeowners and domestic workers. Below is a structured overview of the key functionalities:

| Feature | Description |
|---|---|
| **Admin Login and Dashboard** | Admin can log in through a separate interface. Once logged in, they can view all registered **Workers**, **Contact Us messages**, and **all service requests** in a centralized dashboard. |
| **User Registration and Login** | Separate login/registration for **Homeowners (Users)** and **Workers** with form validations and session-based authentication. |
| **Authentication and Security** | Session-based login, password validation, and form-level input validation across all modules. |
| **Role-Based Navigation** | Redirects users to either the worker or user page based on login credentials. |
| **Profile Management** | Users and Workers can update their profile information such as name, email, password, and phone number. |
| **Service Search and Filtering** | Homeowners can search services (e.g., "Plumber") via a **search bar**. The system dynamically displays relevant workers. |

| | |
|---|---|
| **Detailed Worker Profiles** | Workers are listed in cards showing name, email, skill type, and location. Booking buttons are included below each card. |
| **Booking Request Flow** | Users can request services by selecting a worker and filling out a validated **booking form** (with date/time, service type). Past dates are disabled. |
| **Worker Dashboard** | Workers can see **incoming requests**, then **accept** or **reject** them. Their decision updates request status in real-time on the user side. |
| **User Dashboard** | Users can view **all service requests** made to different workers, track their status (Pending, Accepted, Rejected), and **edit or cancel** them. |
| **Feedback System** | Users can provide feedback (Name, Email, Message as well as star ratings) on their experience. Validations ensure meaningful input, and a success message is shown. |
| **Contact Us Form** | Users can send queries through a form validated for Name, Email, and Message. These are viewable by the Admin in the dashboard. |
| **Google Maps Integration** | A **Google Map** is embedded to display the **location**, giving users a spatial reference point on the site. |
| **Reusable Components** | Forms (Login, Registration), Profile Cards, Booking Forms, and Dashboards are developed as modular Blazor components for reusability. |
| **Service Layer Logic** | Application logic is abstracted in services, keeping UI clean and ensuring organized interaction between components and database operations. |
| **Database Integration (ADO.NET)** | Uses **SQL Server** to store data for users, workers, bookings, contact messages, and feedback via **ADO.NET** commands and connections. |
| **Responsive UI Design** | Built with **Bootstrap and custom CSS**, the platform adapts to all screen sizes including desktop, tablet, and mobile. |

# Workflow Diagram:



## 3. Topics Implemented from the Course:

The **HomeToHome** project integrates several key Visual Programming concepts taught during the course. Below is a breakdown of the topics and how they were implemented:

| Concept | Description / Implementation |
|---|---|
| **Services** | The project uses service classes (e.g., UserService, WorkerService) to manage business logic, including user authentication and data retrieval. |
| **Custom Classes** | Custom model classes like User, Worker, and LoginModel are used to represent entities and manage form data. |
| **Blazor Components** | Reusable Blazor components were developed for forms, profile cards, and navigation, promoting modularity and maintainability. |
| **Database Integration** | SQL Server is integrated using ADO.NET |
| **ADO.NET** | Implemented for direct interaction with SQL procedures where needed. |
| **Stored Procedures & Transactions** | Stored procedures were used for optimized data retrieval and transactions for sensitive operations. |
| **SQL Injection Prevention** | Used parameterized queries to safeguard against SQL injection vulnerabilities. |
| **Session Storage** | Session storage is used to maintain login state across pages for users and workers. |
| **Authentication and Authorization** | Session-based login ensures users and admin are authenticated before accessing their respective dashboards. |

# 4. Code Samples:

## AdminService.cs:

```
namespace HomeToHome.Services
{
    3 references
    public class AdminService
    {
        private readonly string _adminEmail = "hometohome@gmail.com";
        private readonly string _adminPasswordHash;

        0 references
        public AdminService()
        {
            // Pre-hashed password for "hometohome123A" using BCrypt
            _adminPasswordHash = BCrypt.Net.BCrypt.HashPassword("hometohome123A");
        }

        1 reference
        public async Task<bool> AuthenticateAdminAsync(string email, string password)
        {
            await Task.CompletedTask; // Simulate async operation
            if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(password))
            {
                return false;
            }

            if (email.Equals(_adminEmail, StringComparison.CurrentCultureIgnoreCase) && BCrypt.Net.BCrypt.Verify(password, _adminPasswordHash))
            {
                return true;
            }

            return false;
        }
    }
}
```

## ContactService.cs:

```
public class ContactService
{
    private readonly string _connectionString;

    1 reference
    public ContactService(string connectionString) => _connectionString = connectionString;

    1 reference
    public async Task SaveContactAsync(Contact contact)
    {
        try
        {
            var query = @"INSERT INTO Contacts (Name, Email, Message)
                VALUES (@Name, @Email, @Message)";

            using SqlConnection conn = new(_connectionString);
            using SqlCommand cmd = new(query, conn);
            cmd.Parameters.AddWithValue("@Name", contact.Name ?? (object)DBNull.Value);
            cmd.Parameters.AddWithValue("@Email", contact.Email ?? (object)DBNull.Value);
            cmd.Parameters.AddWithValue("@Message", contact.Message ?? (object)DBNull.Value);

            await conn.OpenAsync();
            await cmd.ExecuteNonQueryAsync();
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error saving contact: {ex.Message}");
            throw;
        }
    }
```

## FeedbackService.cs:

```
        public async Task<List<Feedback>> GetFeedbackByWorkerEmailAsync(string workerEmail)
        {
            var feedbackList = new List<Feedback>();
            using (var connection = new SqlConnection(_connectionString))
            {
                await connection.OpenAsync();

                var query = @"SELECT Id, Name, Email, Message, WorkerEmail, ServiceRequestId, Rating, SubmittedAt
                              FROM Feedbacks
                              WHERE WorkerEmail = @WorkerEmail
                              ORDER BY SubmittedAt DESC";

                using var command = new SqlCommand(query, connection);
                command.Parameters.AddWithValue("@WorkerEmail", workerEmail);

                using var reader = await command.ExecuteReaderAsync();
                while (await reader.ReadAsync())
                {
                    feedbackList.Add(new Feedback
                    {
                        Id = reader.GetInt32(0),
                        Name = reader.GetString(1),
                        Email = reader.GetString(2),
                        Message = reader.GetString(3),
                        WorkerEmail = reader.GetString(4),
                        ServiceRequestId = reader.GetInt32(5),
                        Rating = reader.GetInt32(6),
                        SubmittedAt = reader.GetDateTime(7)
                    });
                }
            }
            return feedbackList;
        }
    }
}
```

## RequestService.cs:

```
    public async Task<List<ServiceRequest>> GetRequestsByWorkerEmailAsync(string email)
    {
        List<ServiceRequest> requests = new();
        using SqlConnection conn = new(_connectionString);
        await conn.OpenAsync();

        string query = @"
            SELECT Id, UserEmail, WorkerEmail, ServiceType, Description, PreferredDate, PreferredTime, City, FullAddress, RequestStatus, CreatedAt
            FROM ServiceRequests
            WHERE WorkerEmail = @Email";

        using SqlCommand cmd = new(query, conn);
        cmd.Parameters.AddWithValue("@Email", email);

        using SqlDataReader reader = await cmd.ExecuteReaderAsync();
        while (await reader.ReadAsync())
        {
            requests.Add(new ServiceRequest
            {
                Id = reader.GetInt32(0),
                UserEmail = reader.GetString(1),
                WorkerEmail = reader.GetString(2),
                ServiceType = reader.GetString(3),
                Description = reader.IsDBNull(4) ? null : reader.GetString(4),
                PreferredDate = reader.IsDBNull(5) ? null : reader.GetDateTime(5),
                PreferredTime = reader.IsDBNull(6) ? null : reader.GetString(6),
                City = reader.IsDBNull(7) ? null : reader.GetString(7),
                FullAddress = reader.IsDBNull(8) ? null : reader.GetString(8),
                RequestStatus = reader.GetString(9),
                CreatedAt = reader.GetDateTime(10)
            });
        }

        return requests;
    }
}
```

## UserService.cs:

```
public async Task<User?> AuthenticateUser(string email, string password)
{
    using (SqlConnection connection = new(_connectionString))
    {
        await connection.OpenAsync();

        string query = "SELECT * FROM Users WHERE LOWER(Email) = LOWER(@Email)";

        using SqlCommand command = new(query, connection);
        command.Parameters.AddWithValue("@Email", email?.Trim() ?? string.Empty);

        using SqlDataReader reader = await command.ExecuteReaderAsync();
        if (await reader.ReadAsync())
        {
            string storedHash = reader["Password"].ToString();
            // Verify the provided password against the stored hash
            if (BCrypt.Net.BCrypt.Verify(password, storedHash))
            {
                var user = new User
                {
                    FirstName = reader["FirstName"].ToString(),
                    LastName = reader["LastName"].ToString(),
                    Email = reader["Email"].ToString(),
                    City = reader["City"].ToString(),
                    PhoneNumber = reader["PhoneNumber"].ToString(),
                    DateOfBirth = reader.IsDBNull(reader.GetOrdinal("DateOfBirth")) ? null : reader.GetDateTime(reader.GetOrdinal("DateOfBirth")),
                    Gender = reader["Gender"].ToString(),
                    FullAddress = reader["FullAddress"].ToString(),
                    CNIC = reader["CNIC"].ToString()
                };
                return user;
            }
        }
    }
    return null;
}
```

## WorkerService.cs:

```
1 reference
public async Task<List<Worker>> SearchWorkersBySkill(string skill)
{
    List<Worker> workers = new();

    using SqlConnection conn = new(_connectionString);
    await conn.OpenAsync();

    string query = @"
    SELECT FirstName, LastName, City, Designation, Experience, Skills, Email
    FROM Workers
    WHERE Skills LIKE @Skill";

    using SqlCommand cmd = new(query, conn);
    cmd.Parameters.AddWithValue("@Skill", "%" + skill + "%");

    using SqlDataReader reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        Worker worker = new()
        {
            FirstName = reader["FirstName"].ToString(),
            LastName = reader["LastName"].ToString(),
            City = reader["City"].ToString(),
            Designation = reader["Designation"].ToString(),
            Experience = reader["Experience"] as int?,
            Skills = reader["Skills"].ToString()?.Split(",").Select(s => s.Trim()).ToList(),
            Email = reader["Email"].ToString()
        };
        workers.Add(worker);
    }

    return workers;
}
```

## AdminPanel.razor:

```
@code {
    private string adminEmail = string.Empty;

    protected override async Task OnInitializedAsync()
    {
        try
        {
            // Verify admin session
            adminEmail = await JSRuntime.InvokeAsync<string>("sessionStorage.getItem", "adminEmail");
            if (string.IsNullOrEmpty(adminEmail))
            {
                NavigationManager.NavigateTo("/admin-login");
                return;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error loading admin session: {ex.Message}");
        }
    }

    private void NavigateToServiceRequests()
    {
        NavigationManager.NavigateTo($"/view-requests?email={Uri.EscapeDataString(adminEmail)}");
    }

    private void NavigateToWorkers()
    {
        NavigationManager.NavigateTo($"/view-workers?email={Uri.EscapeDataString(adminEmail)}");
    }

    private void NavigateToContacts()
    {
        NavigationManager.NavigateTo($"/view-contacts?email={Uri.EscapeDataString(adminEmail)}");
    }
```

## ContactUs.razor:

```
<script>
    (g => {
        var h, a, k, p = "The Google Maps JavaScript API", c = "google", l = "importLibrary", q = "_ib", m = document, b = window; b = b[c] || (b[c] = {}); var d = b.maps || (b.maps = {}), r =
            await (a = m.createElement("script")); e.set("libraries", [...r] + ""); for (k in g) e.set(k.replace(/[A-Z]/g, t => "" + t[0].toLowerCase()), g[k]); e.set("callback", c + ".maps."
            ({ key: "AIzaSyBdXmtyD1S1OlF8Ise4y98LLhOT-LeS5EQ", v: "weekly" });
</script>

<GoogleMap ApiKey="@ApiKey"
           Center="new GoogleMapCenter(33.7138, 73.0247)"
           Height="400"
           Width="100"
           Zoom="10"
           Markers="markers" />

@code {
    private string successMessage= string.Empty;

    private Contact formModel = new();

    private async Task HandleSubmit()
    {
        await ContactService.SaveContactAsync(formModel);
        successMessage = "Your message has been sent successfully!";
        formModel = new(); // Reset form
    }

    List<GoogleMapMarker> markers = new()
    {
        new GoogleMapMarker()
        {
            PinElement = new PinElement { Scale = 1.5 },
            Position = new GoogleMapMarkerPosition(33.7138, 73.0247),
            Title = "Single family house with modern design",
        },
    };
}
```

## ReviewRequestPage.razor:

```
    protected override async Task OnInitializedAsync()
    {
        if (string.IsNullOrEmpty(Email))
        {
            var uri = new Uri(NavigationManager.Uri);
            Email = System.Web.HttpUtility.ParseQueryString(uri.Query).Get("email") ?? string.Empty;
        }

        if (!string.IsNullOrEmpty(Email))
        {
            requests = await RequestService.GetRequestsByUserEmailAsync(Email);
        }

        _isInitialized = true;
    }

    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (_isInitialized && firstRender)
        {
            if (string.IsNullOrEmpty(Email))
            {
                Email = await JSRuntime.InvokeAsync<string>("sessionStorage.getItem", "userEmail") ?? string.Empty;
                if (!string.IsNullOrEmpty(Email))
                {
                    requests = await RequestService.GetRequestsByUserEmailAsync(Email);
                    StateHasChanged();
                }
            }

            if (!string.IsNullOrEmpty(Email))
            {
                await JSRuntime.InvokeVoidAsync("sessionStorage.setItem", "userEmail", Email);
            }
        }
    }
```

## WorkerProfileView.razor:

```
@code {
    [Parameter]
    [SupplyParameterFromQuery]
    public string Email { get; set; } = string.Empty;

    private Worker? worker;
    private bool _isInitialized;

    protected override async Task OnInitializedAsync()
    {
        if (string.IsNullOrEmpty(Email))
        {
            var uri = new Uri(NavigationManager.Uri);
            Email = System.Web.HttpUtility.ParseQueryString(uri.Query).Get("email") ?? string.Empty;
        }

        if (!string.IsNullOrEmpty(Email))
        {
            worker = await WorkerService.GetWorkerByEmailAsync(Email);
        }

        _isInitialized = true;
    }

    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (_isInitialized && firstRender)
        {
            if (string.IsNullOrEmpty(Email))
            {
                Email = await JSRuntime.InvokeAsync<string>("sessionStorage.getItem", "userEmail") ?? string.Empty;
                if (!string.IsNullOrEmpty(Email))
                {
                    worker = await WorkerService.GetWorkerByEmailAsync(Email);
                    StateHasChanged();
                }
            }

            if (!string.IsNullOrEmpty(Email))
            {
                await JSRuntime.InvokeVoidAsync("sessionStorage.setItem", "userEmail", Email);
            }
        }
    }
```

## GoogleMapDemoComponents.cs:

```
using Microsoft.AspNetCore.Components;

namespace HomeToHome.Components
{
    1 reference
    public class GoogleMapDemoComponentBase : ComponentBase
    {
        1 reference
        public string ApiKey { get; set; } = "AIzaSyBdXmtyD1S1OlF8Ise4y98LLhOT-Le55EQ";
    }
}
```

## AppSettings.json:

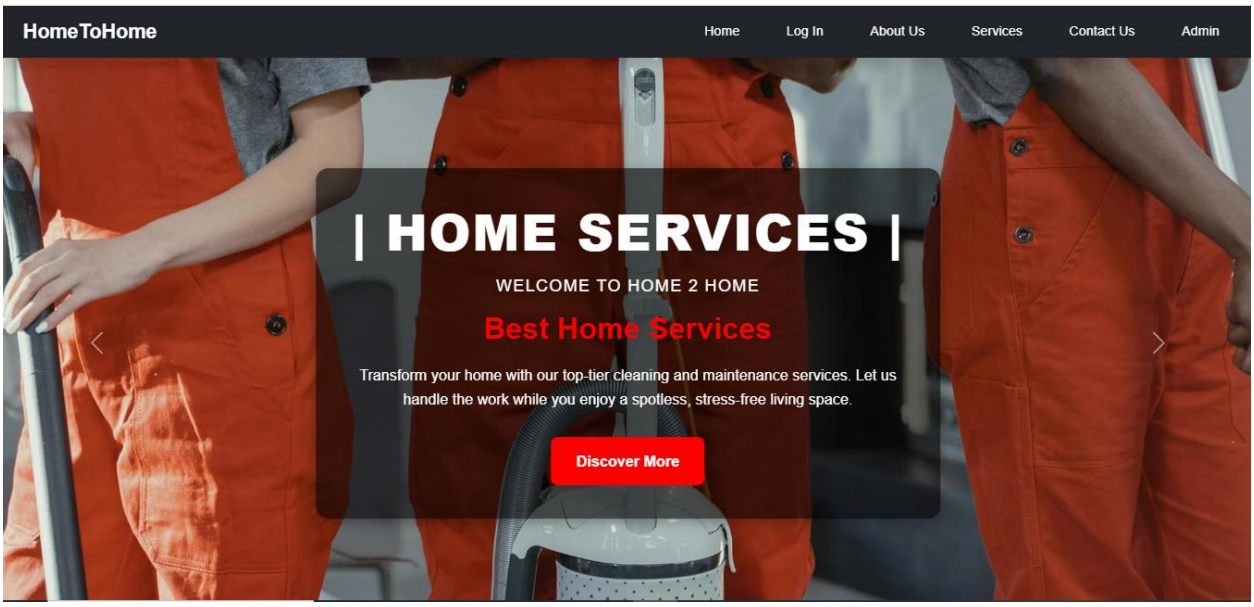
```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=LAPTOP-5A8POKQQ\\SQLEXPRESS;Database=HomeToHome;Trusted_Connection=True;TrustServerCertificate=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

## Program.cs:

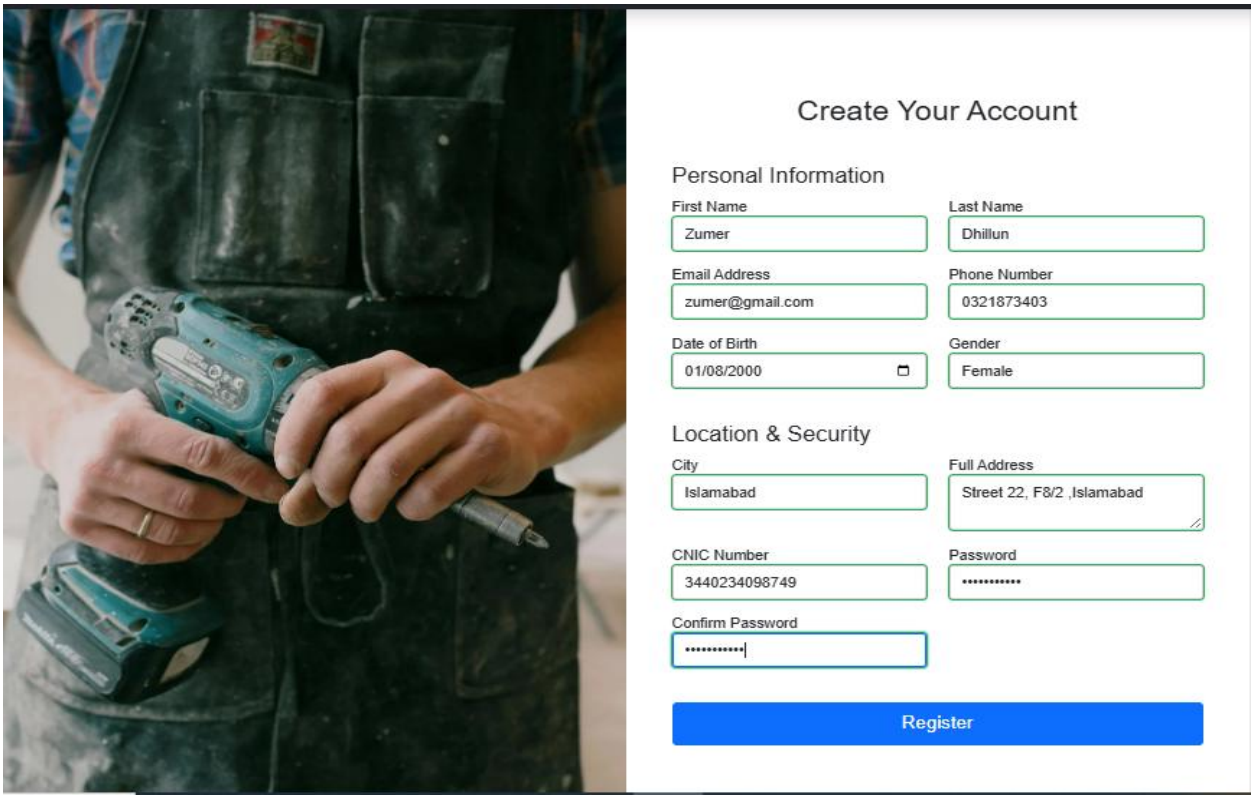
```
builder.Services.AddScoped<UserService>(sp =>
{
    var config = sp.GetRequiredService<IConfiguration>();
    var connectionString = config.GetConnectionString("DefaultConnection");
    return new UserService(connectionString);
});

builder.Services.AddScoped<FeedbackService>();

builder.Services.AddScoped<WorkerService>();
builder.Services.AddScoped<AdminService>();


var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error", createScopeForErrors: true);
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();

app.UseStaticFiles();
app.UseAntiforgery();

app.MapRazorComponents<App>()
    .AddInteractiveServerRenderMode();

app.Run();
```

# 5. Screenshots of Implementation:

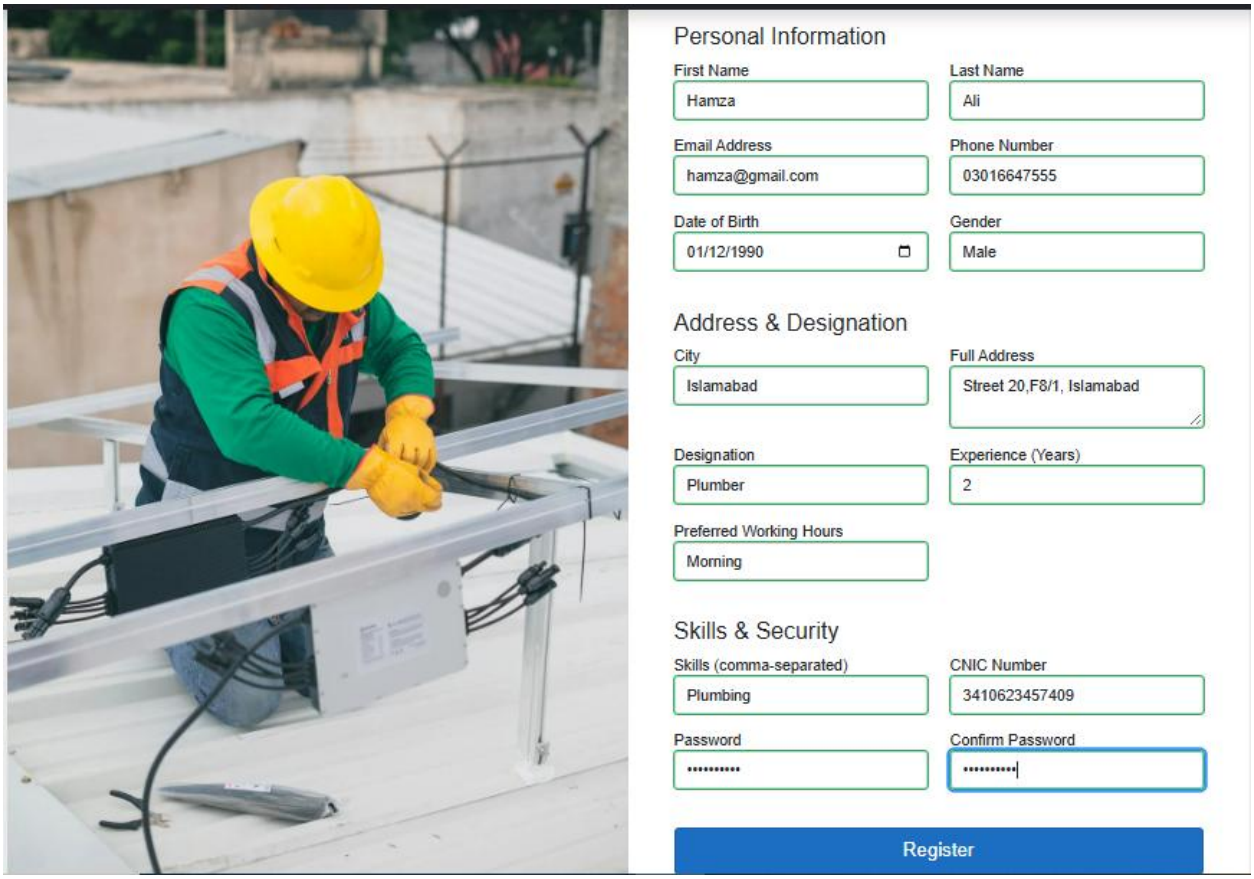
## Home Page:



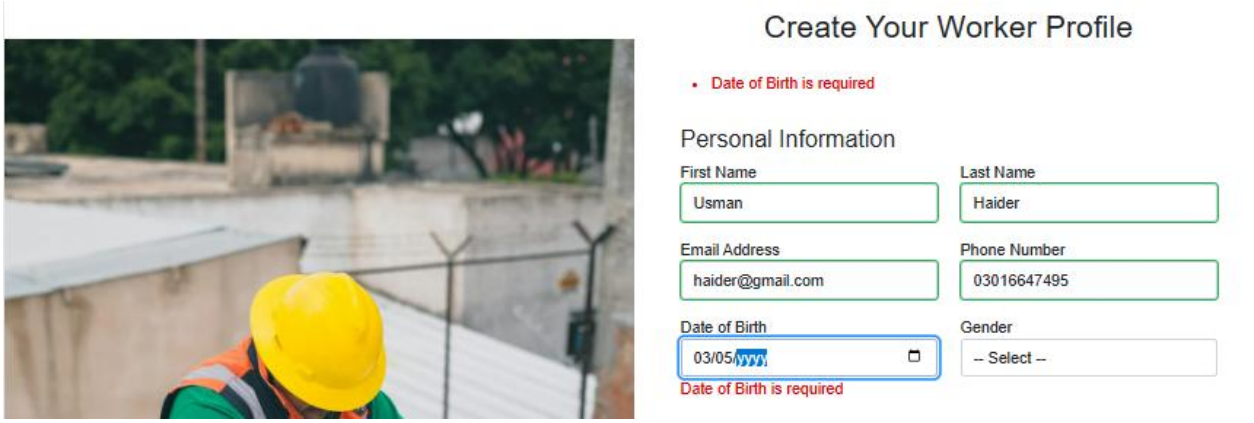

## Register as User:

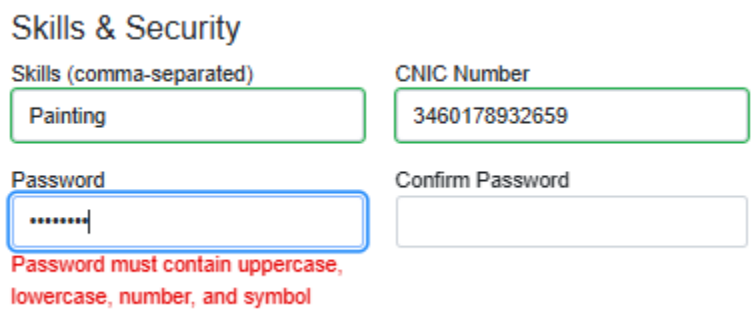


## Register As Worker:

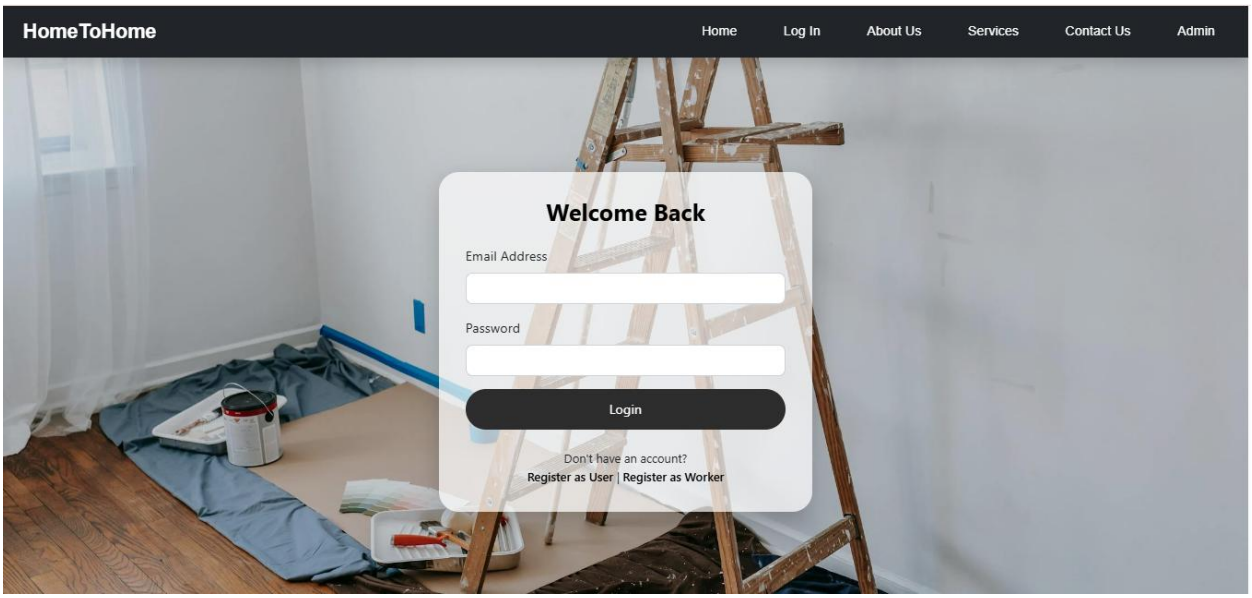



## Validations:

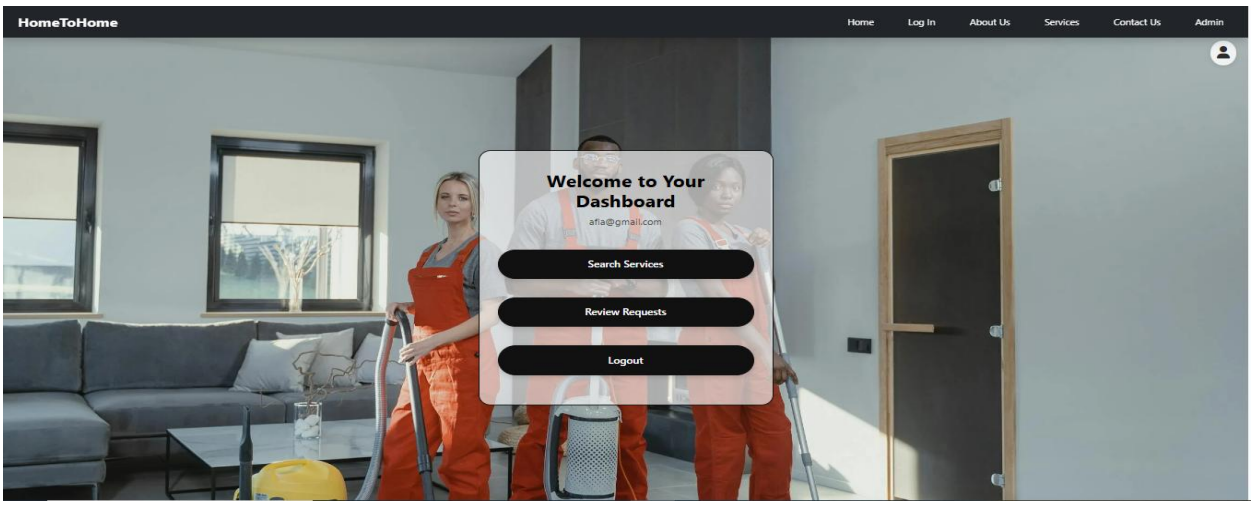




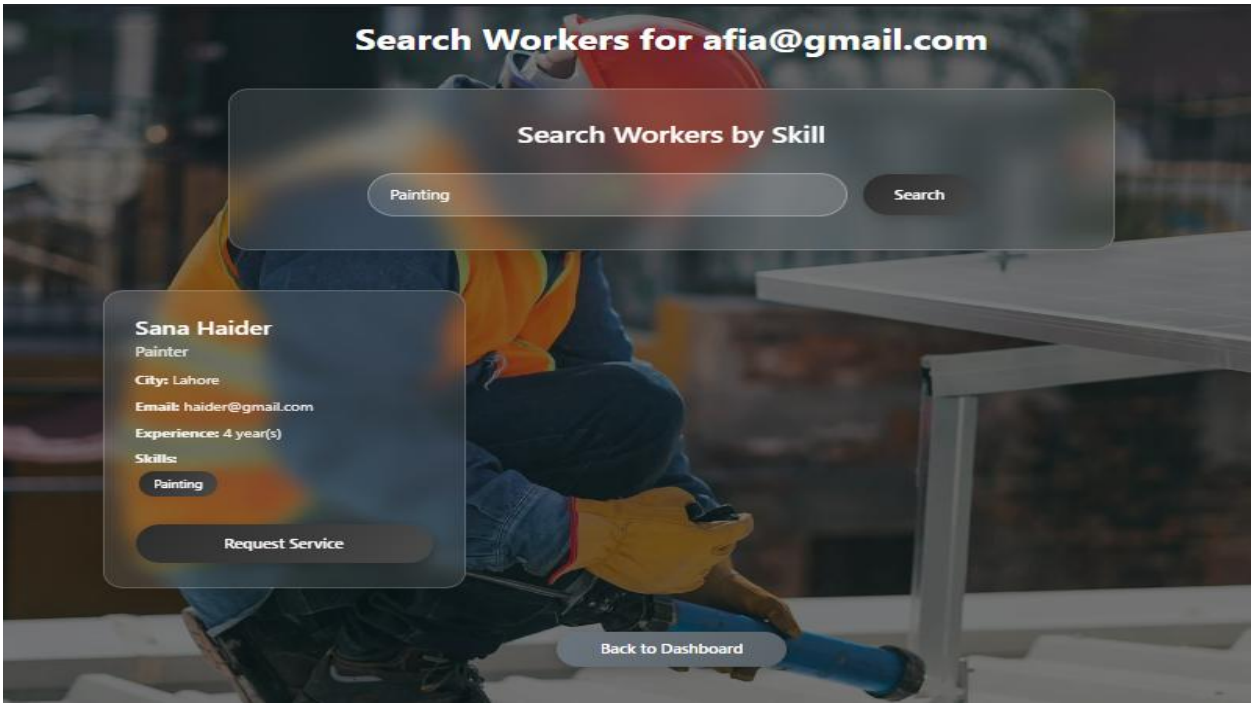

## Log In:



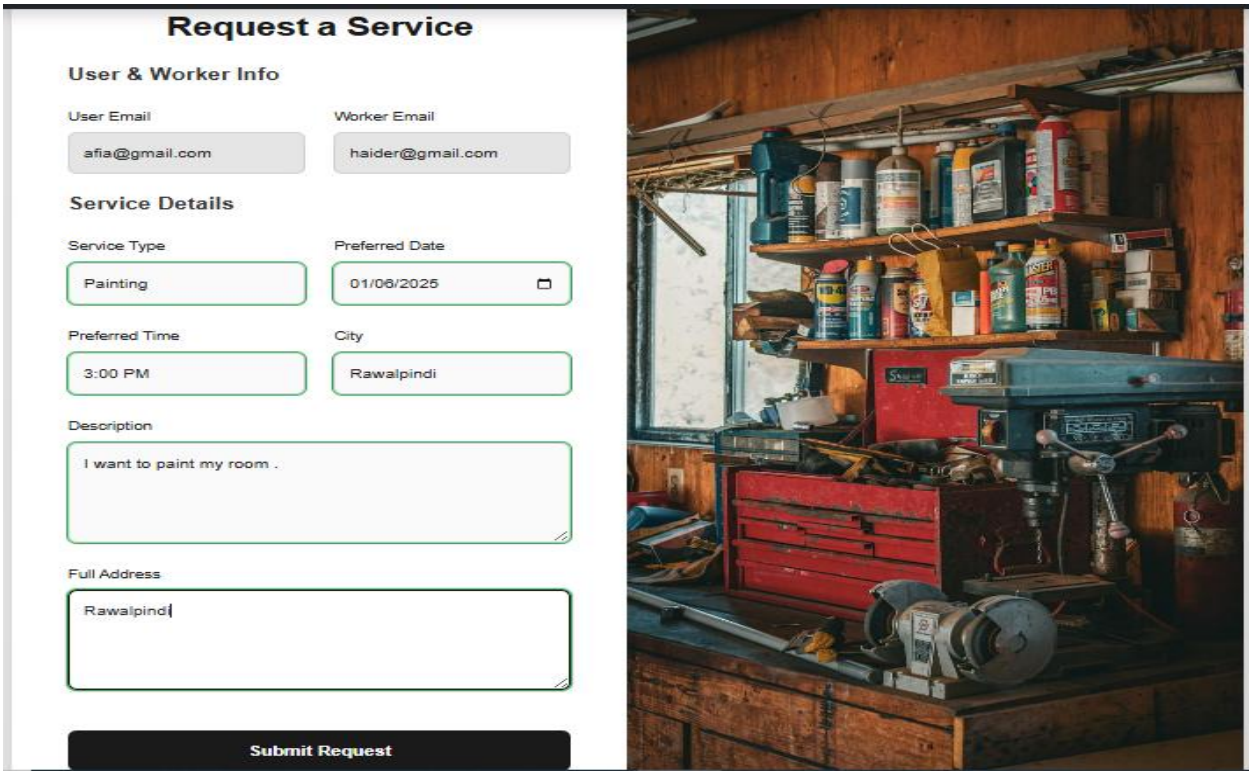

## User Dashboard:



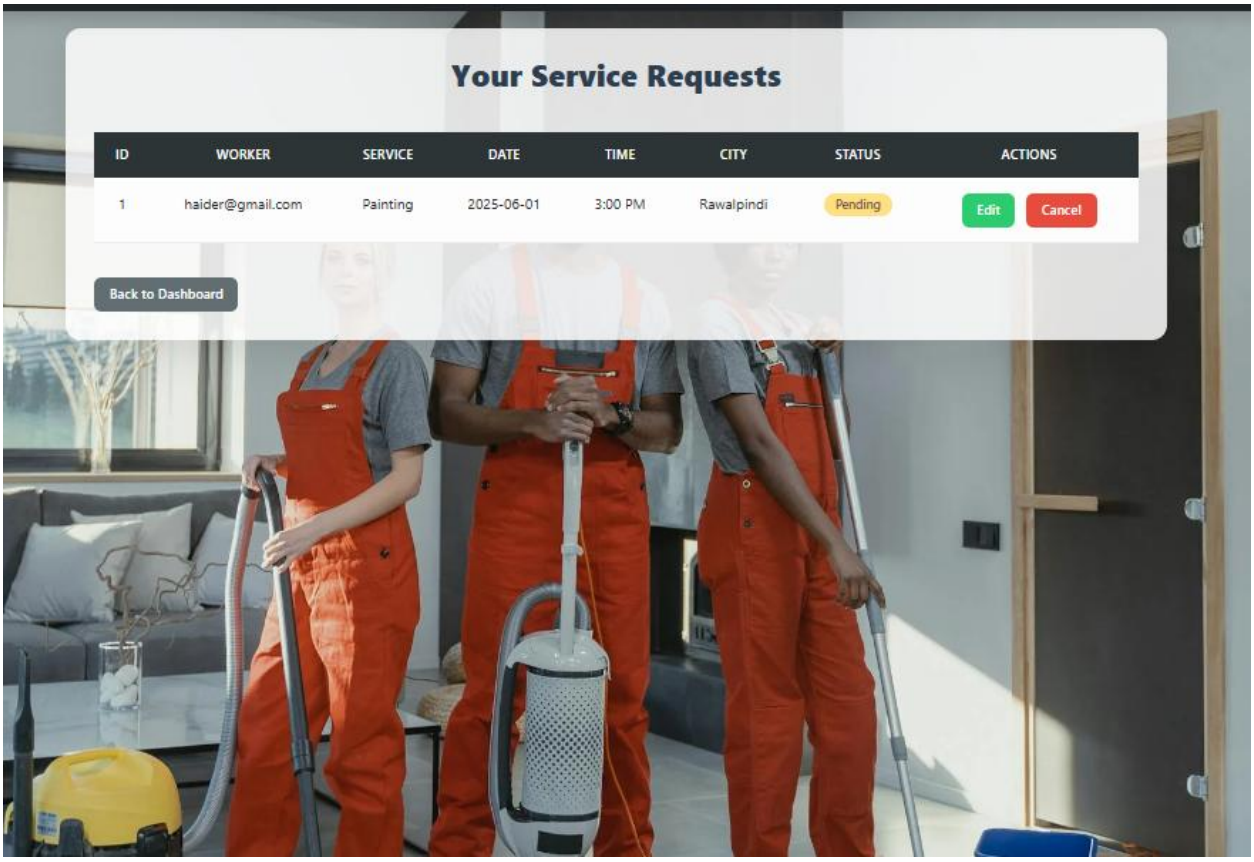
## Search Service To Book a Service:



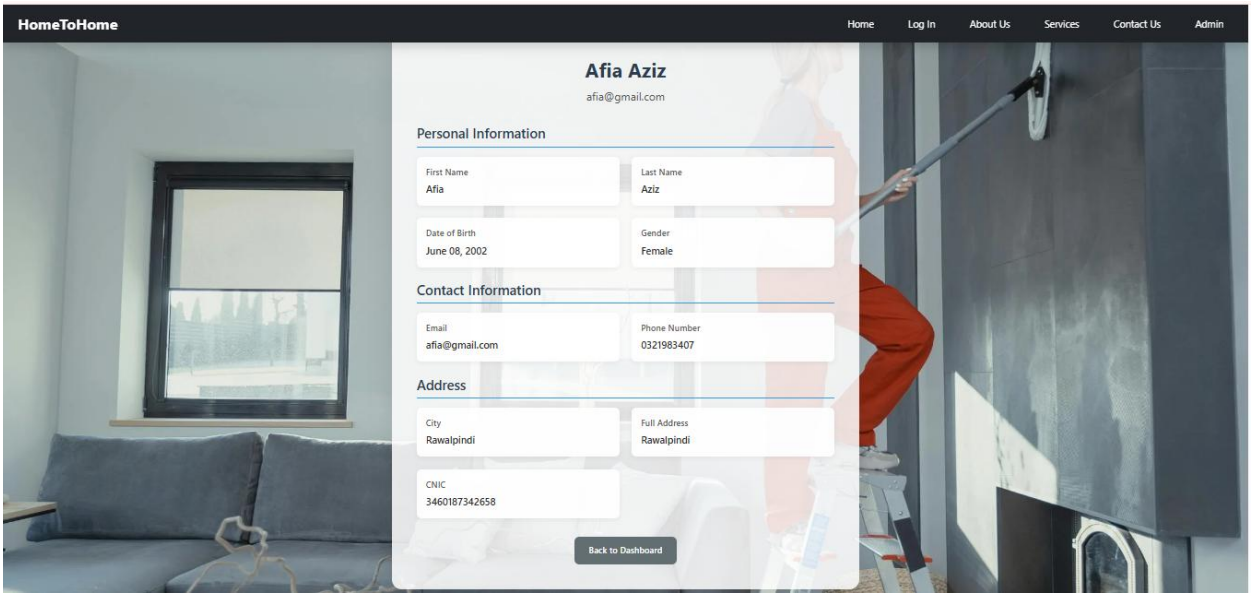

## Request a Service:



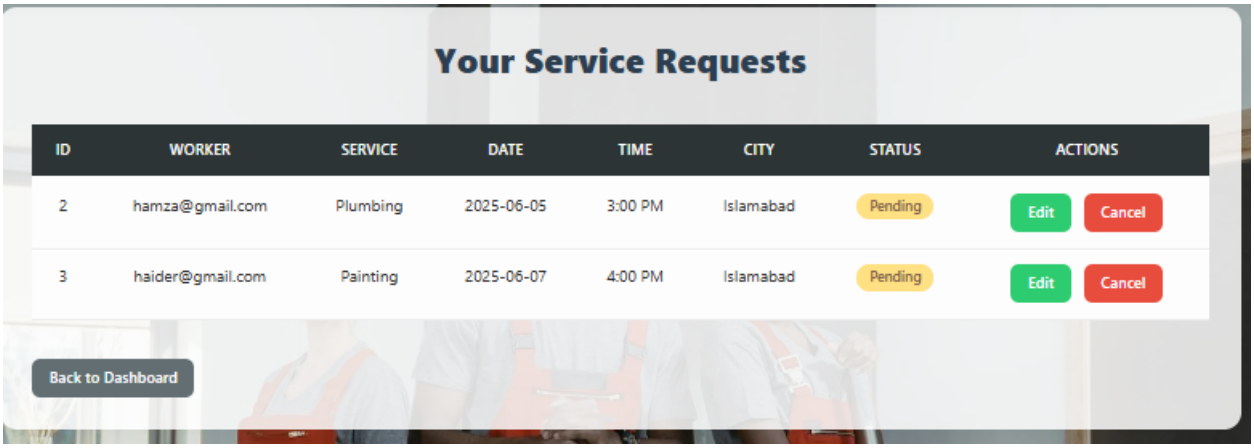
## Review Request – Service Requested:



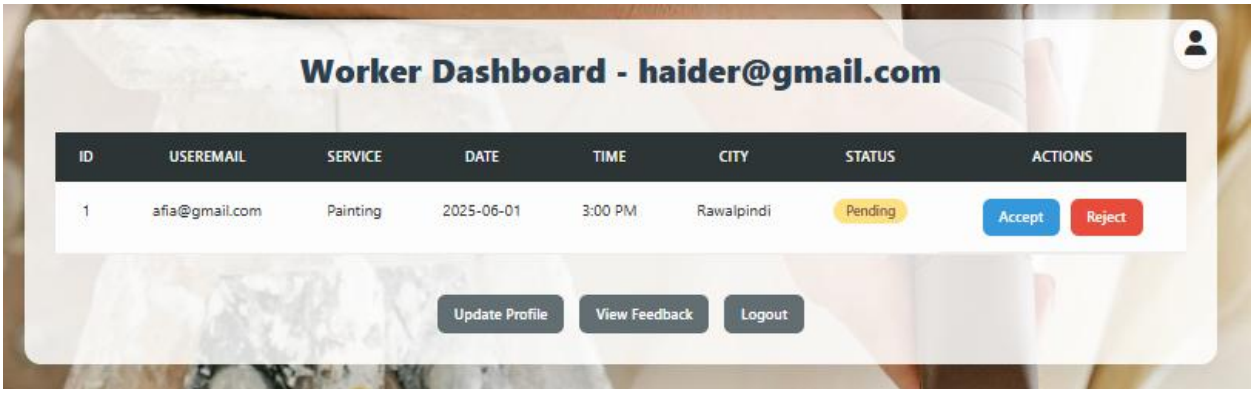

## User Profile:



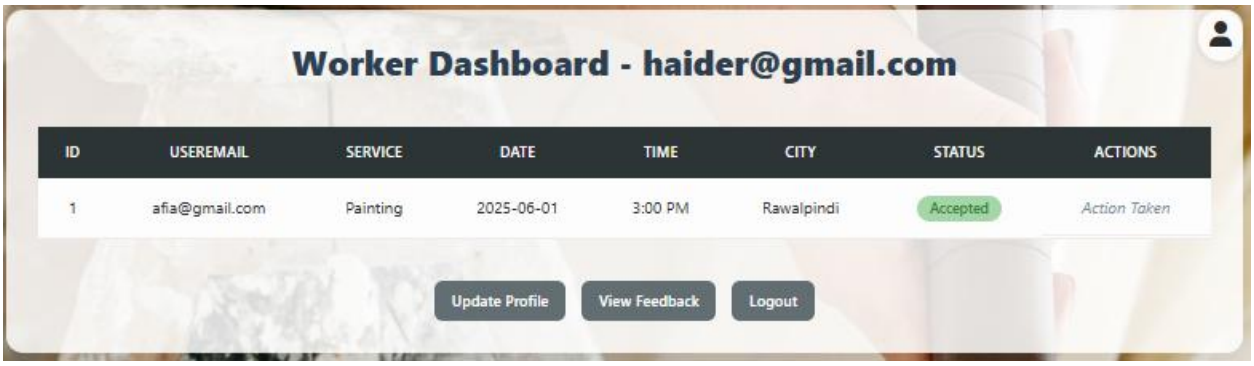
## User Dashboard:



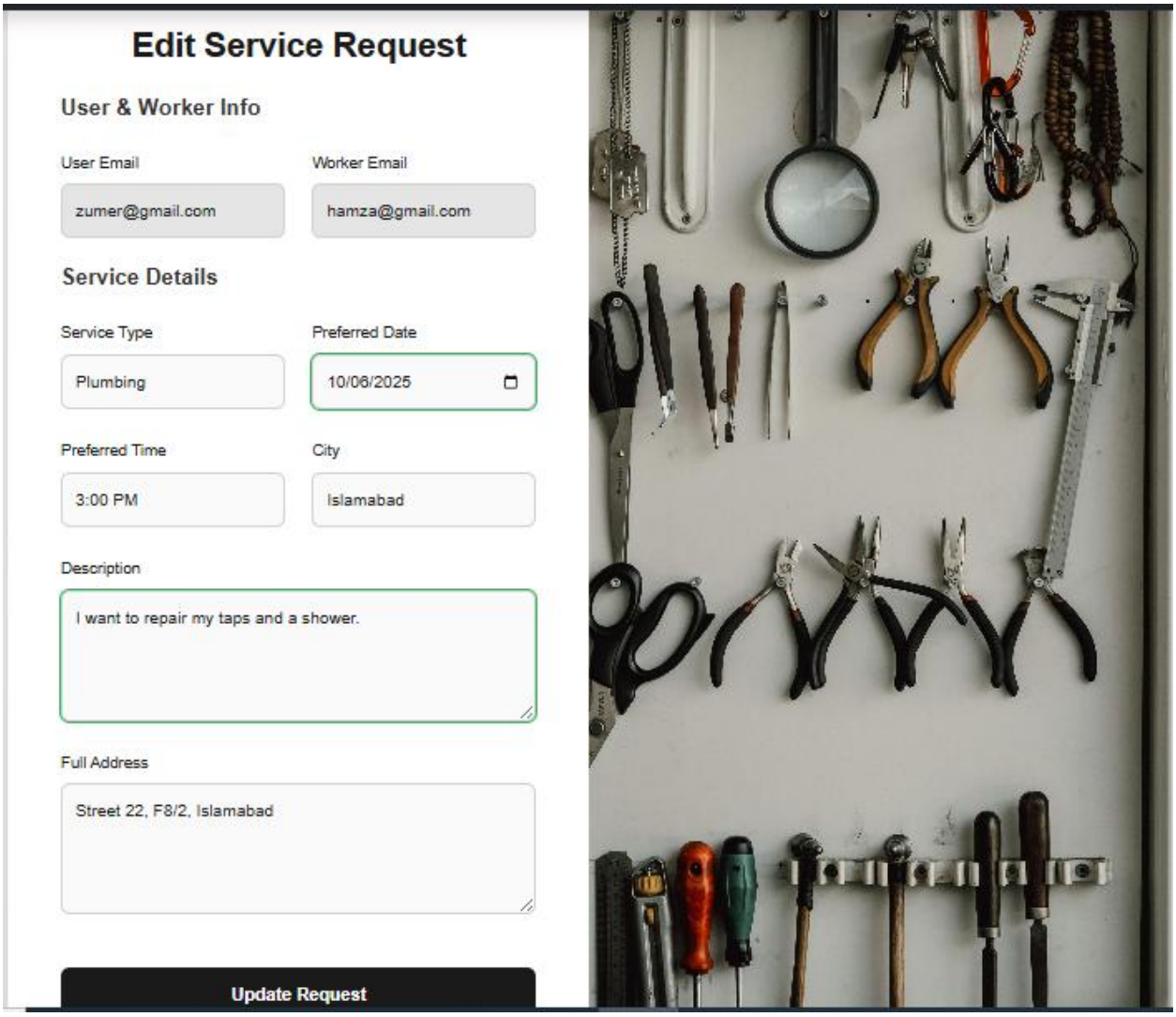

## Worker Dashboard – Received Requested Services:



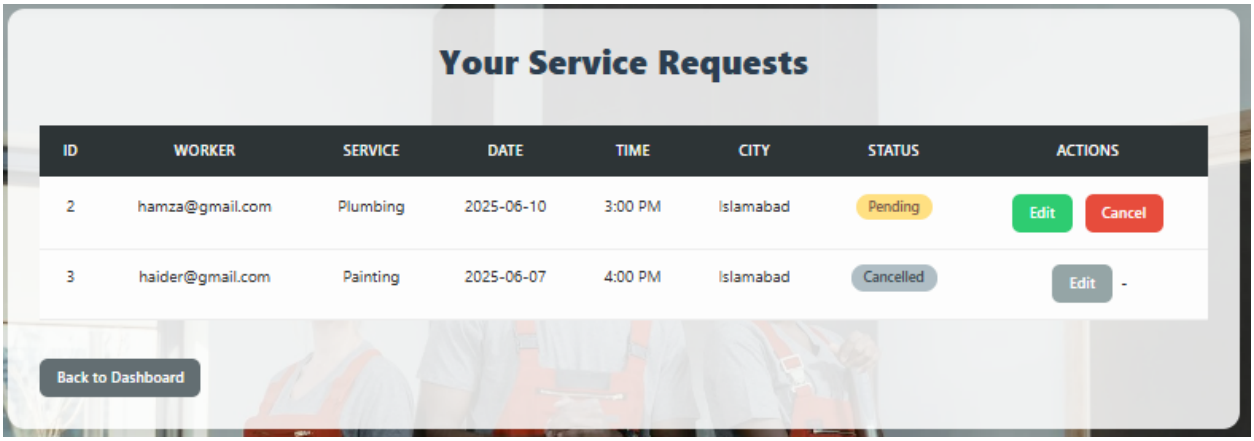

## Accepted – Request Accepted:



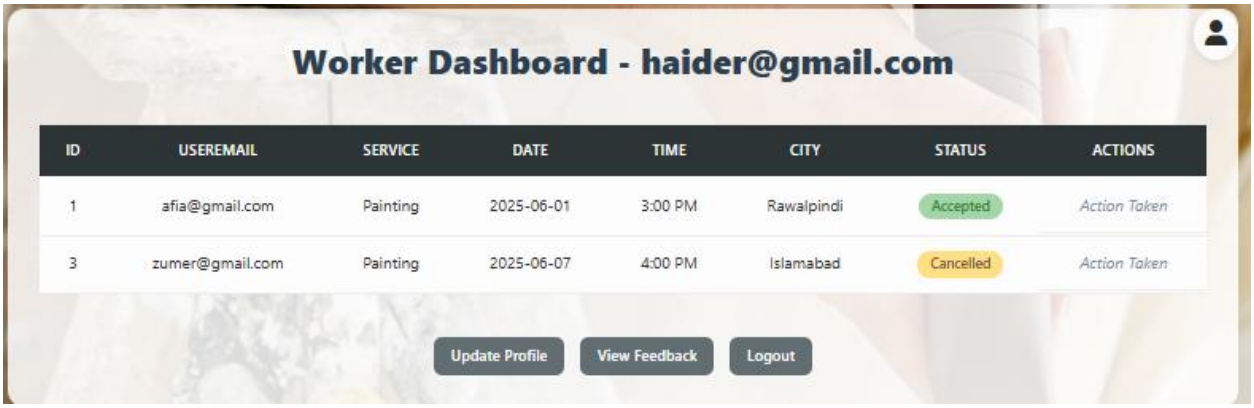
## Edit Service Request – User Side:



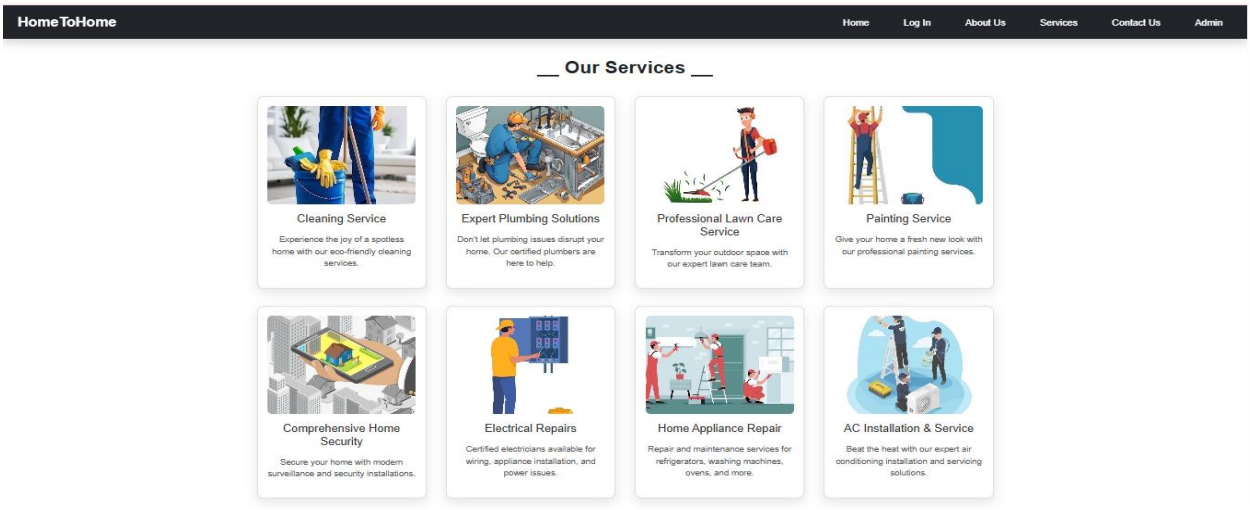

## Request Cancelled – By User:



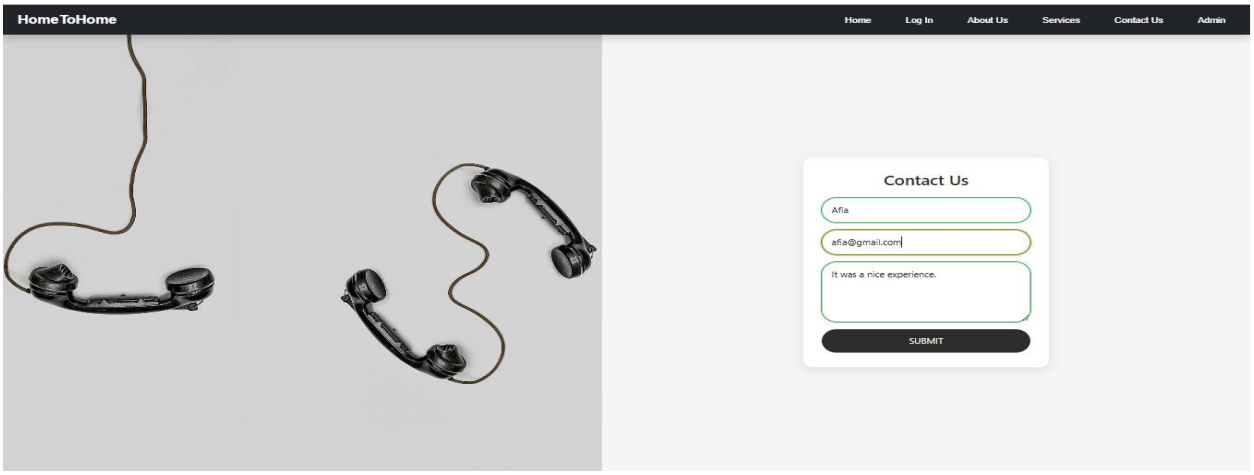
## Updated at the worker side too:



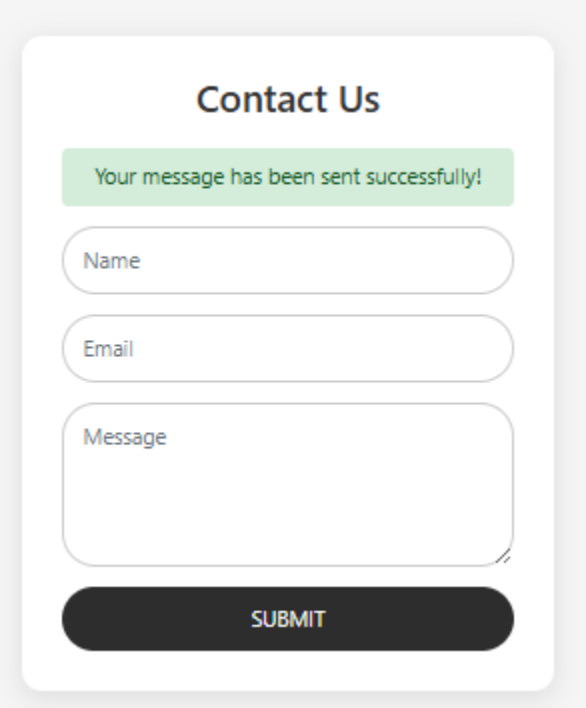

## Our Services Page:



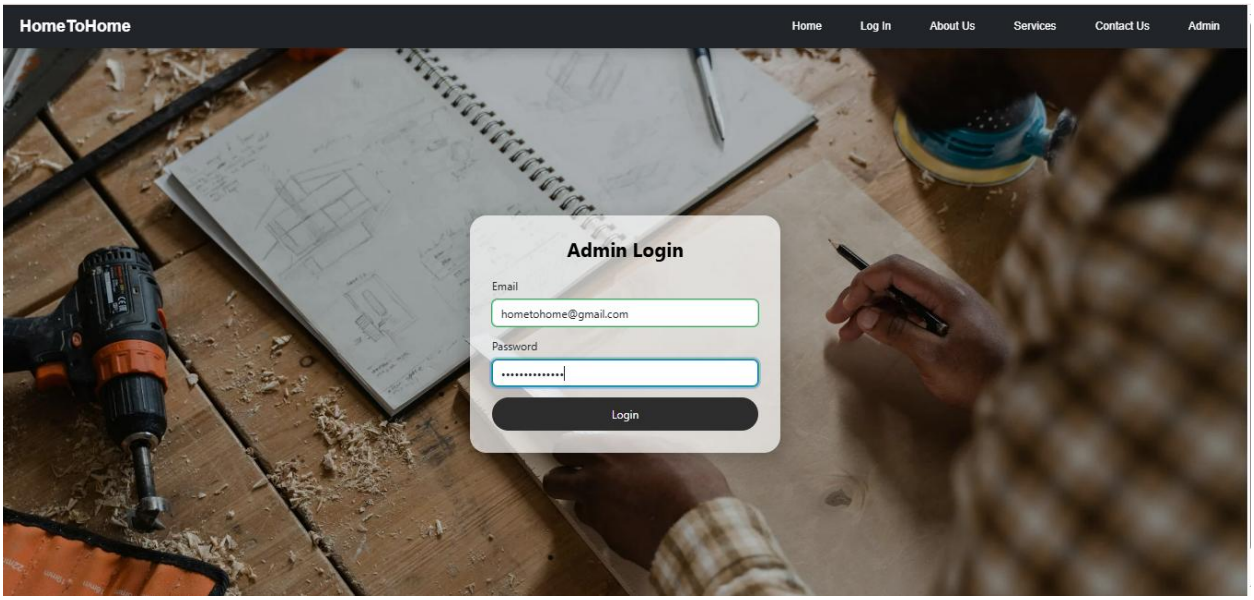

## Contact Us:



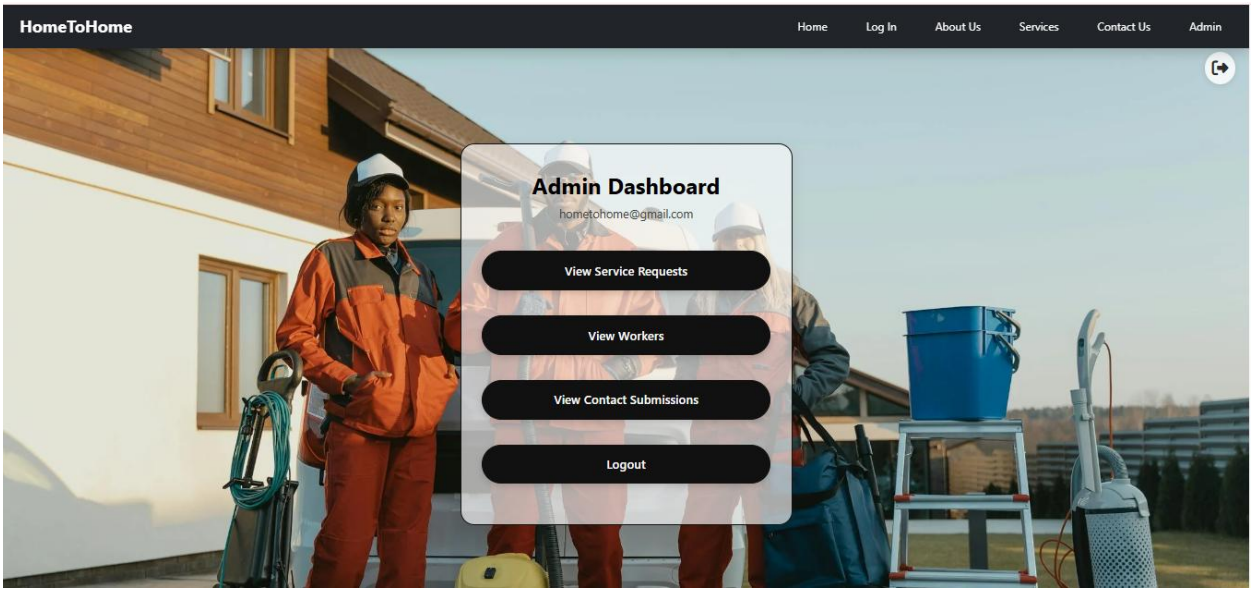
## Success Message:



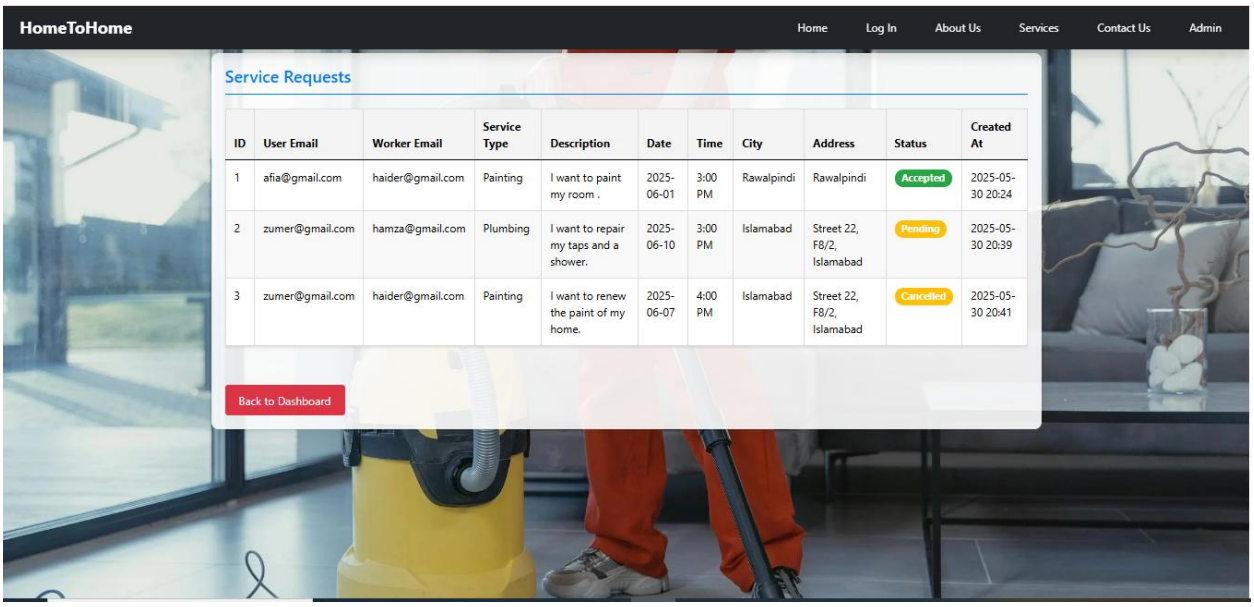

## Admin Log In:



# Admin Dashboard:

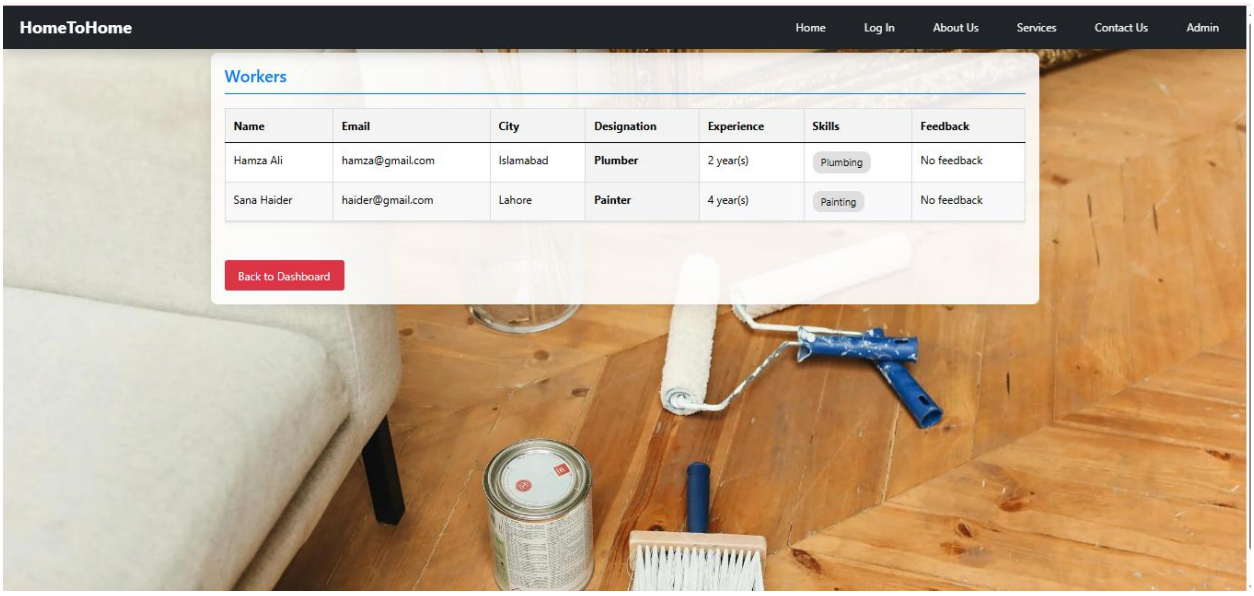



# All Service Requests:

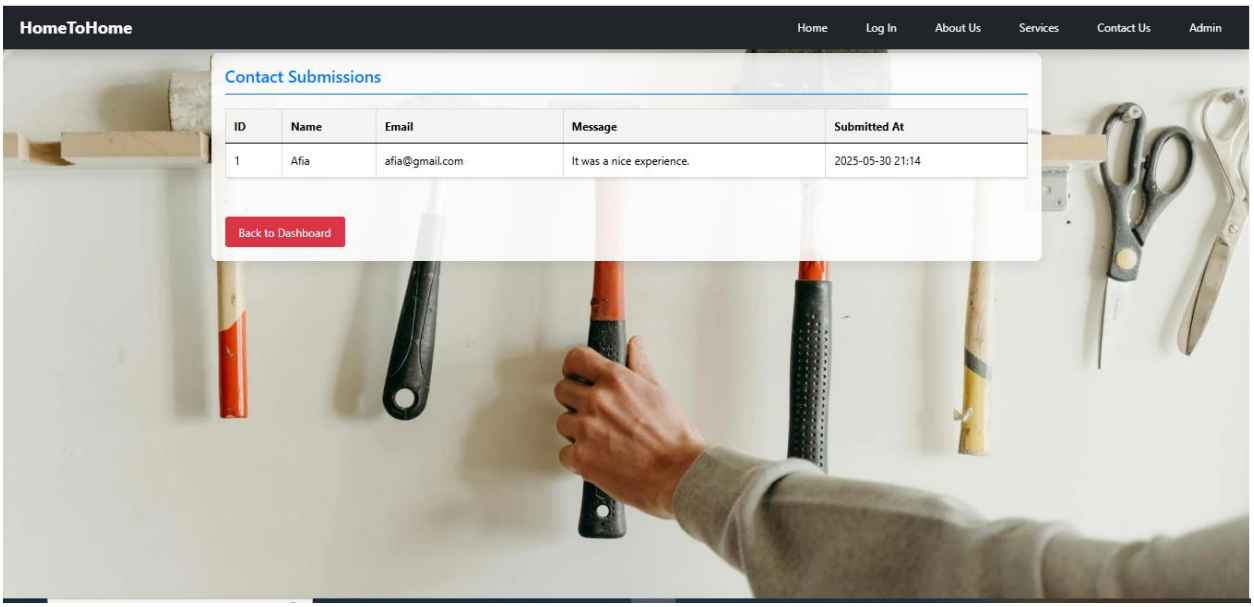



| ID | User Email | Worker Email | Service Type | Description | Date | Time | City | Address | Status | Created At |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | afia@gmail.com | haider@gmail.com | Painting | I want to paint my room . | 2025-06-01 | 3:00 PM | Rawalpindi | Rawalpindi | Accepted | 2025-05-30 20:24 |
| 2 | zumer@gmail.com | hamza@gmail.com | Plumbing | I want to repair my taps and a shower. | 2025-06-10 | 3:00 PM | Islamabad | Street 22, F8/2, Islamabad | Pending | 2025-05-30 20:39 |
| 3 | zumer@gmail.com | haider@gmail.com | Painting | I want to renew the paint of my home. | 2025-06-07 | 4:00 PM | Islamabad | Street 22, F8/2, Islamabad | Cancelled | 2025-05-30 20:41 |

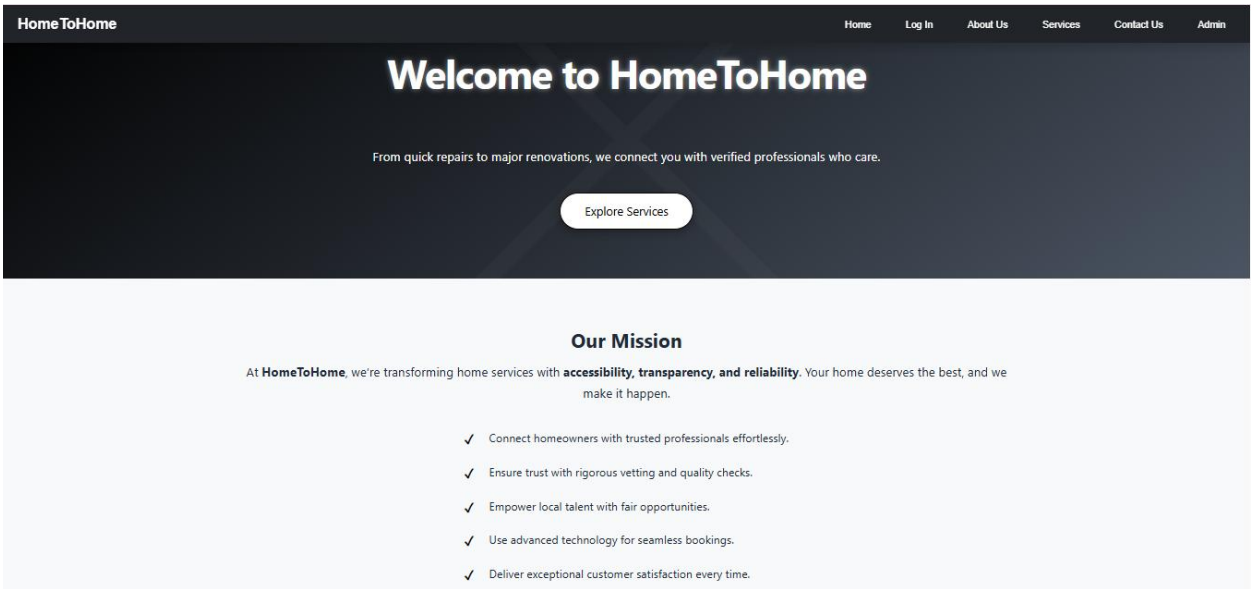
# All Workers:




# Contact Submissions:

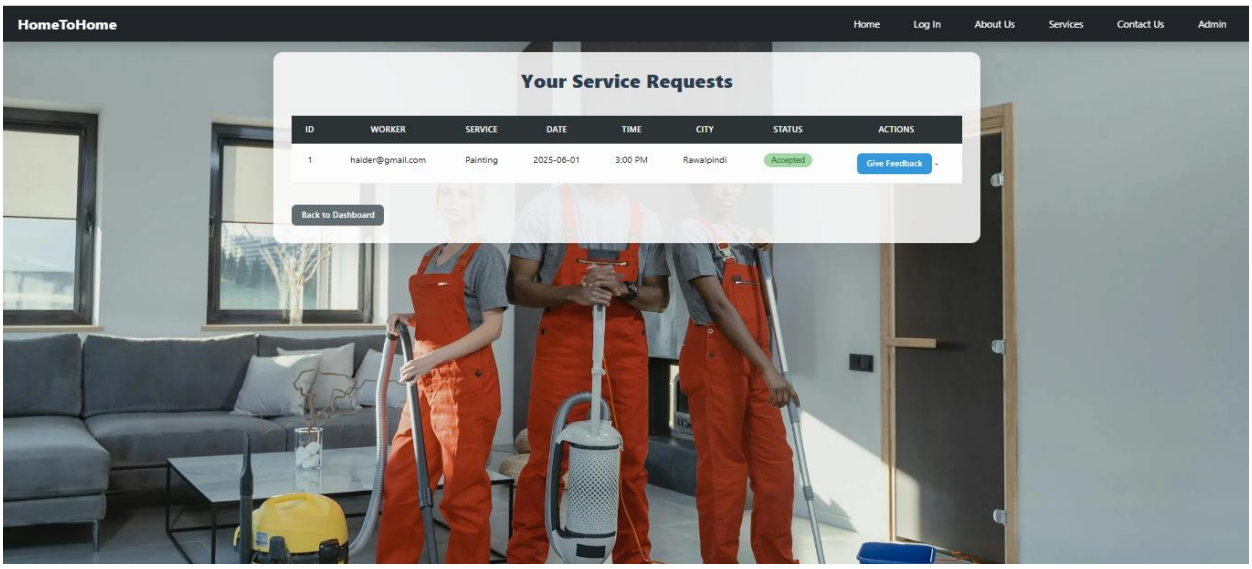


## About us Page:

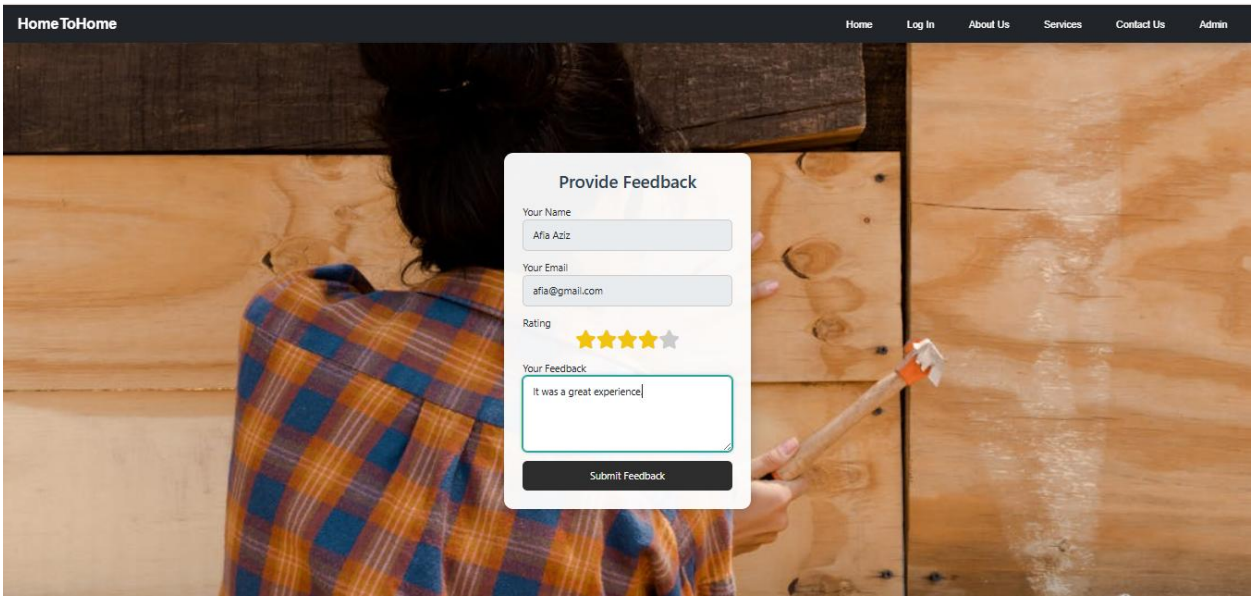



## Google Map Integration:

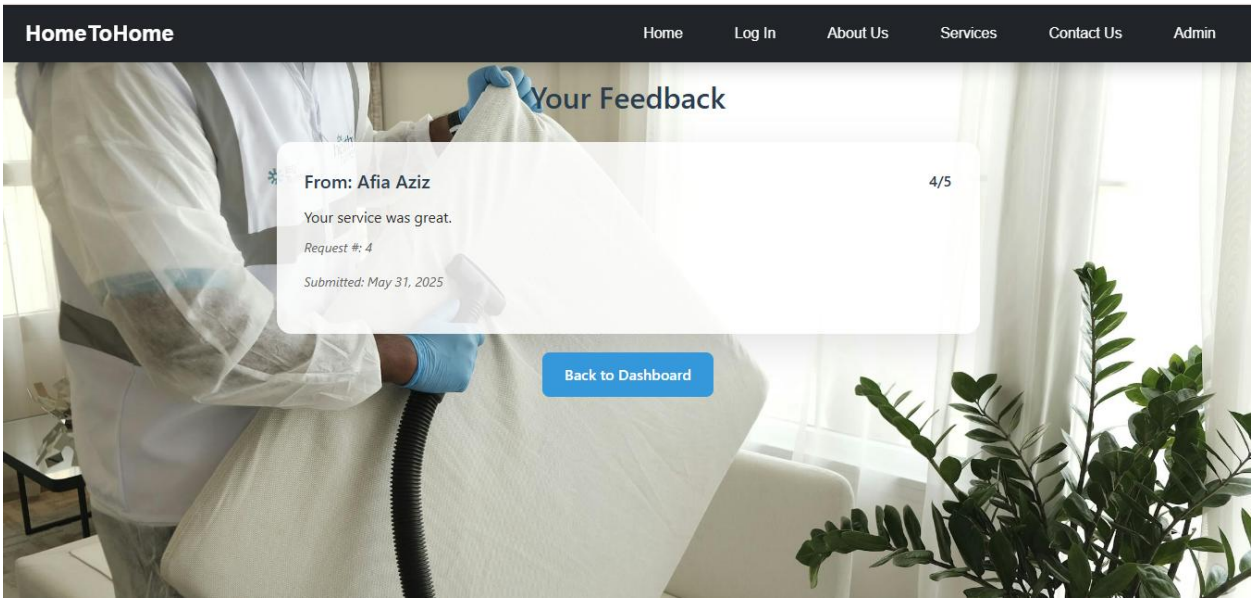

## Give Feedback – Enables after request accepted:



## Provide Feedback:

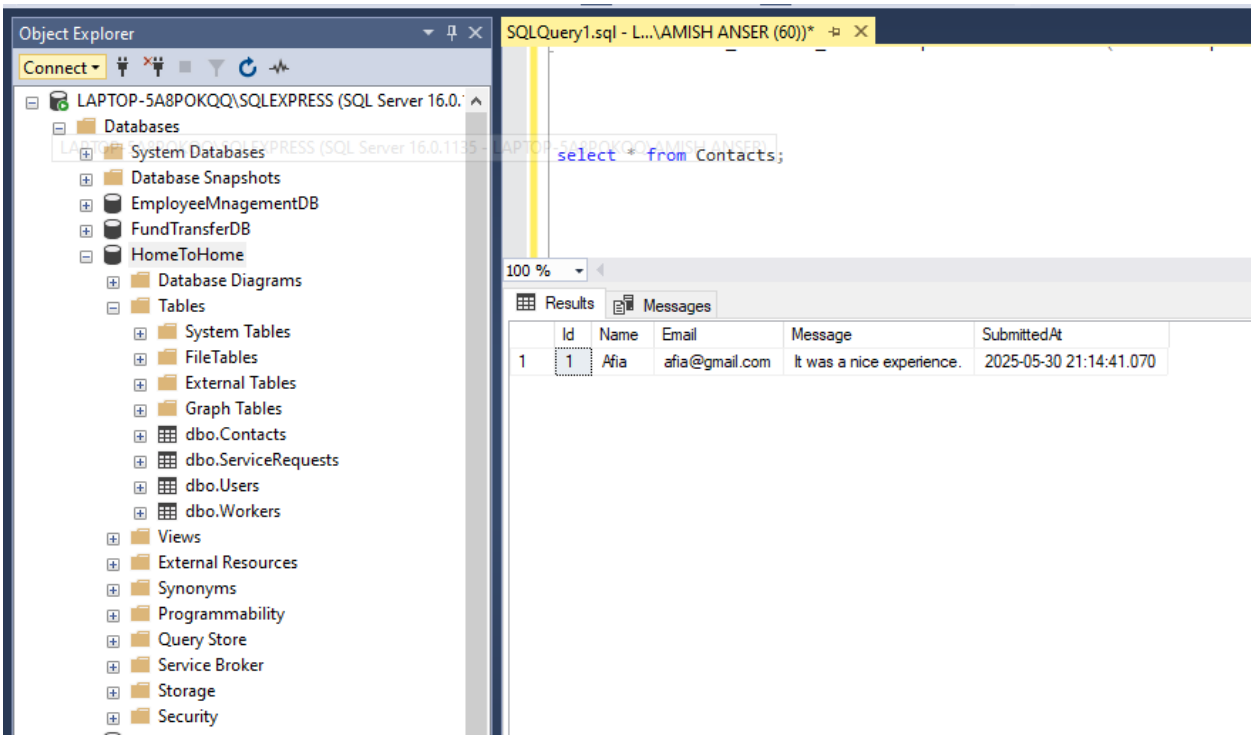



## Feedback Received at Worker Side:

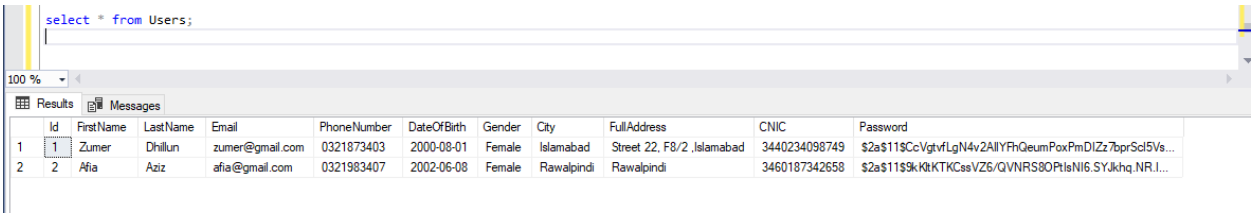


# SSMS SETUP:

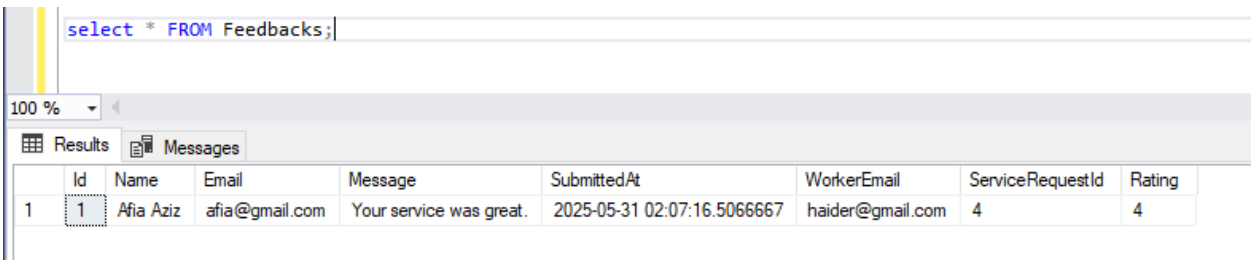
## Contact Table:




## Users Table:




## Feedbacks Table:



## Workers Table:

```
select * from Workers;
```

100 %

Results | Messages

| | WorkerId | FirstName | LastName | Email | PhoneNumber | DateOfBirth | Gender | City | FullAddress | Designation | Experience | PreferredWorkingHours | CNIC | Password |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Hamza | Ali | hamza@gmail.com | 03016647555 | 1990-12-01 | Male | Islamabad | Street 20,F8/1, Islamabad | Plumber | 2 | Morning | 3410623457409 | $2a$11$OZ.zjdvAo0WvwYooqkK40OM4, |
| 2 | 2 | Sana | Haider | haider@gmail.com | 03016647495 | 1998-05-03 | Female | Lahore | Johar Town, Lahore | Painter | 4 | Full-time | 3460178932659 | $2a$11$OAyRHYpelpjGITvGd8SDOa8CI |

# 6. Testing

## Applied Testing Techniques:

- **Manual Testing:**
  All functional modules (Login, Booking, Feedback, Contact Form, Request Flow) were tested manually using different input combinations and user roles (Customer & Worker).
- **UI Testing:**
  We validated layout behavior, button clicks, visibility toggles, and conditional UI elements such as the dynamic request list, date pickers, and feedback form.
- **Form Validation Testing:**
  Tested form components by submitting:
  - Empty fields
  - Invalid formats (e.g., incorrect email)
  - Disallowed inputs (e.g., past dates in date pickers)
  - Confirmed visual cues like success or error messages

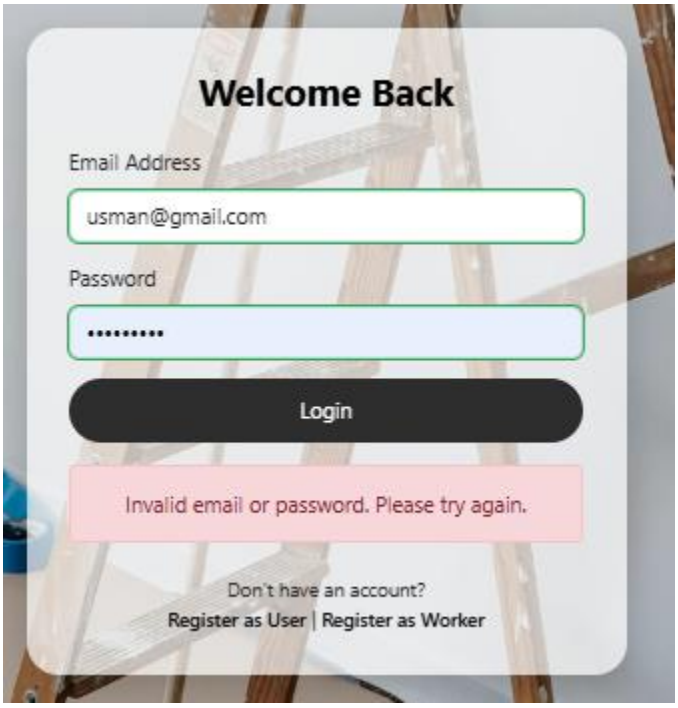
## Tools Used:

- **Browser Console:** For real-time debugging.
- **Visual Studio Debugger:** For backend routing and event validation.
- **Blazor Hot Reload:** Helped test UI changes quickly.
- **SSMS:** For database integration

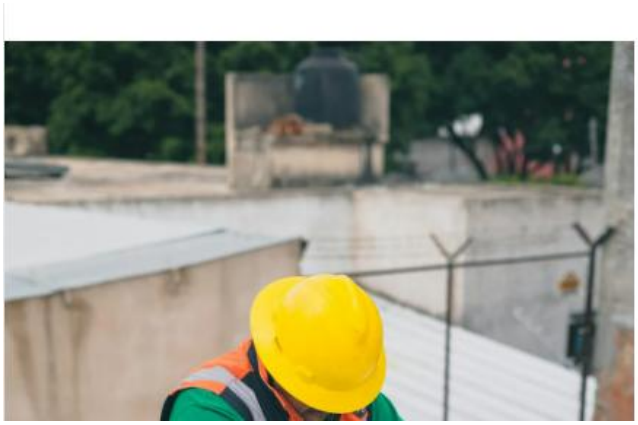
## Input & Form Validations:

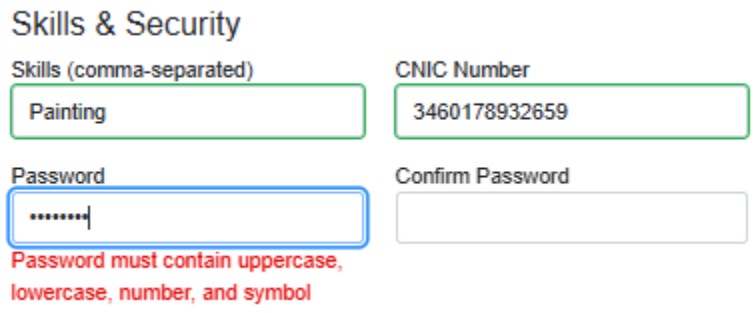
- **Login Form:** Validates presence of email and password.
- **Search Booking:** Requires a service name; prevents blank submissions.
- **Booking Form:**
  - Date Picker: Only allows today or future dates.
  - Required fields check (name, phone, etc.)
- **Feedback & Contact Forms:**
  - Validates email format
  - Required fields cannot be blank
  - Shows confirmation on successful submission

## Login Form Validation:

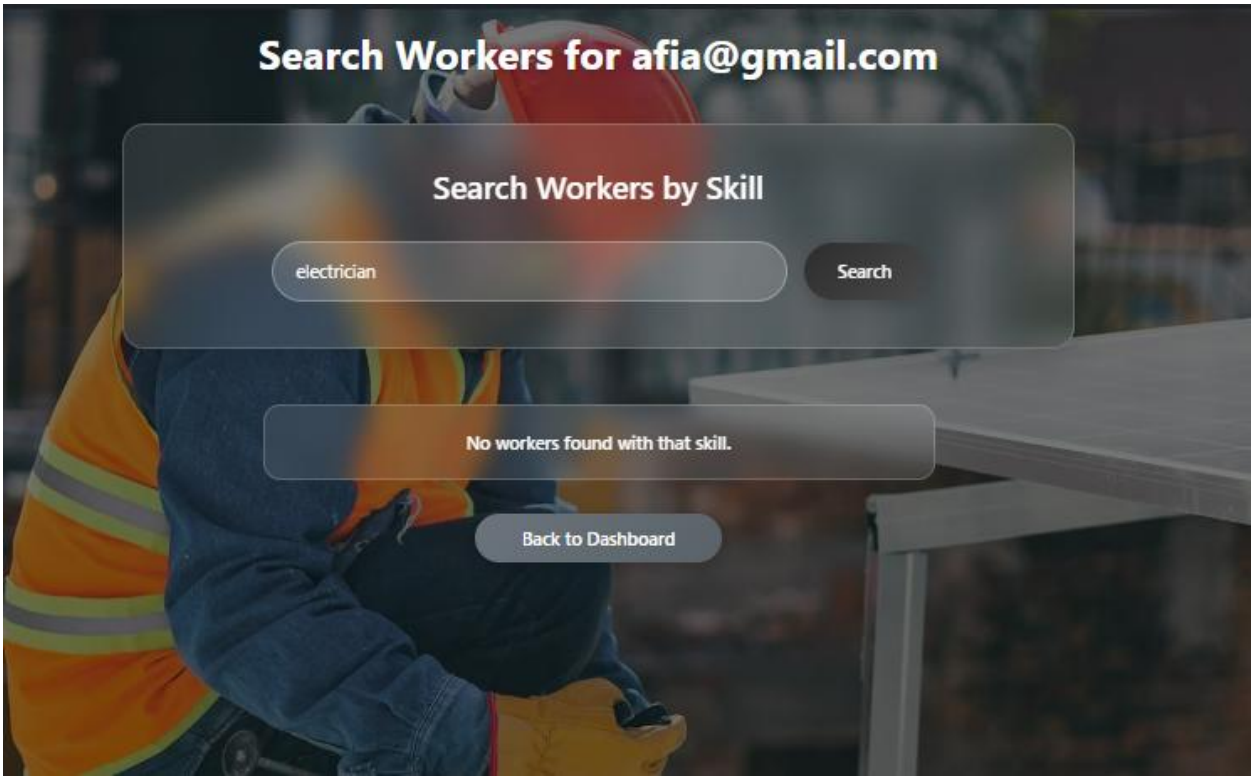



## Validations:

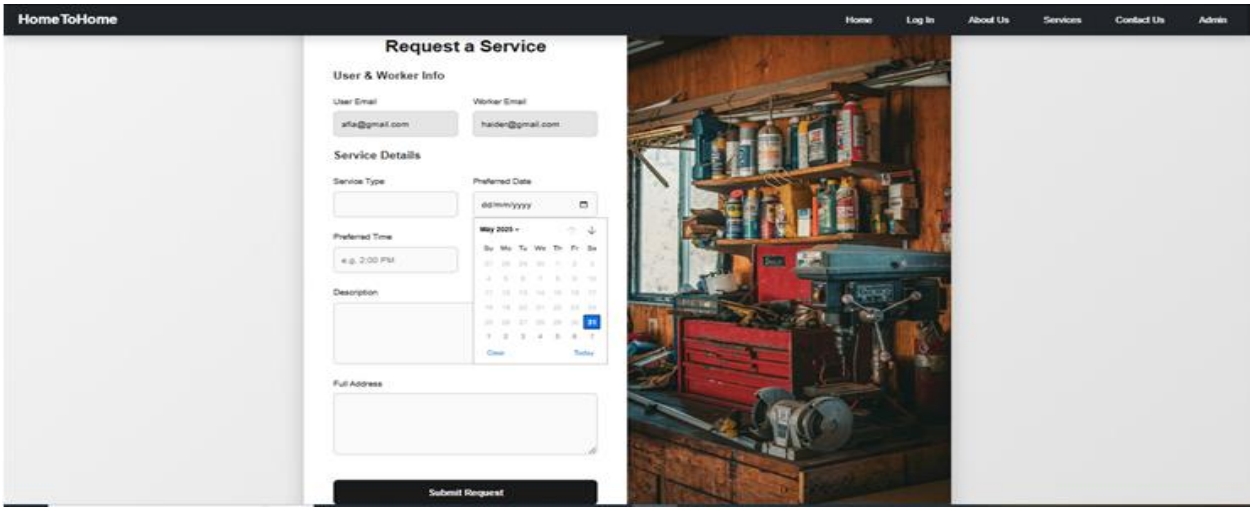




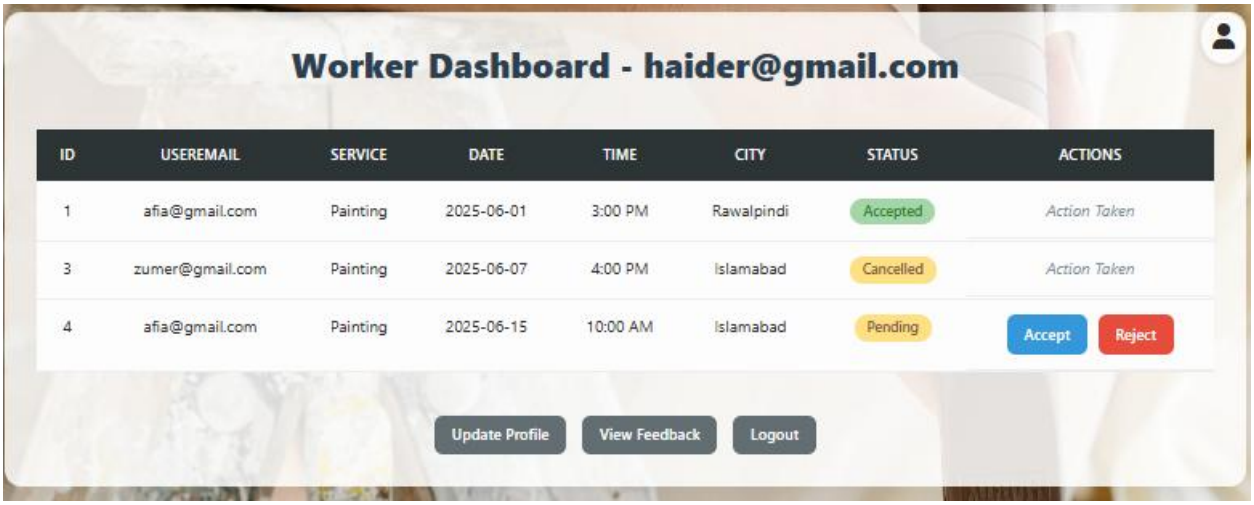

## Service Search:



## Booking Form – Past Date Disabled:

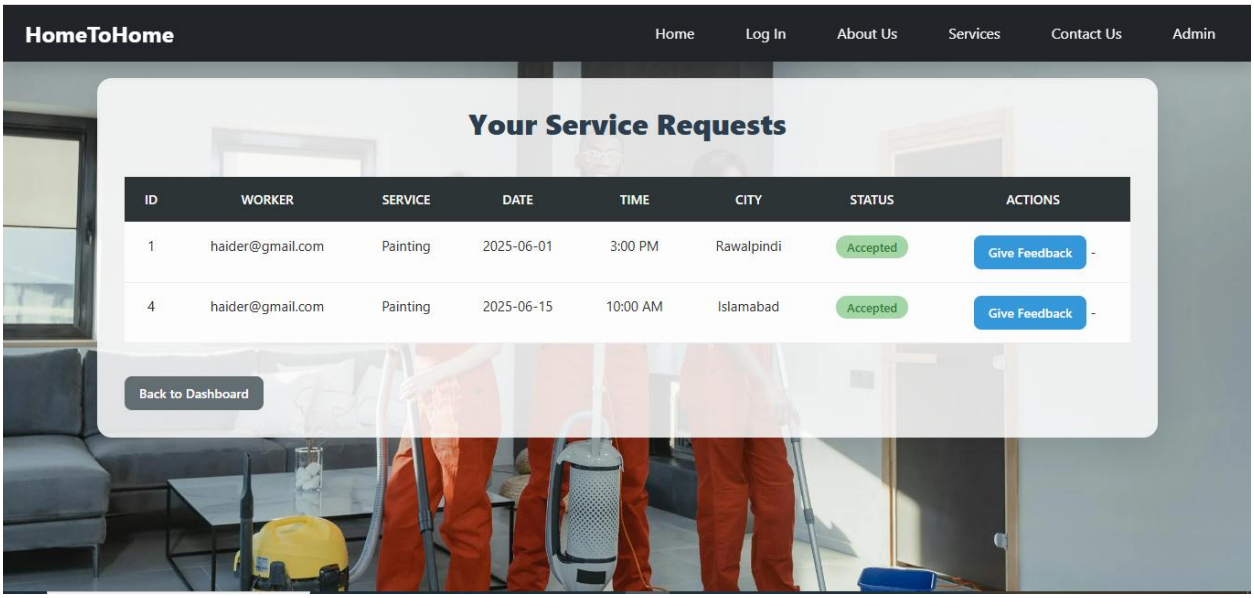



## Worker Dashboard – Request Received:

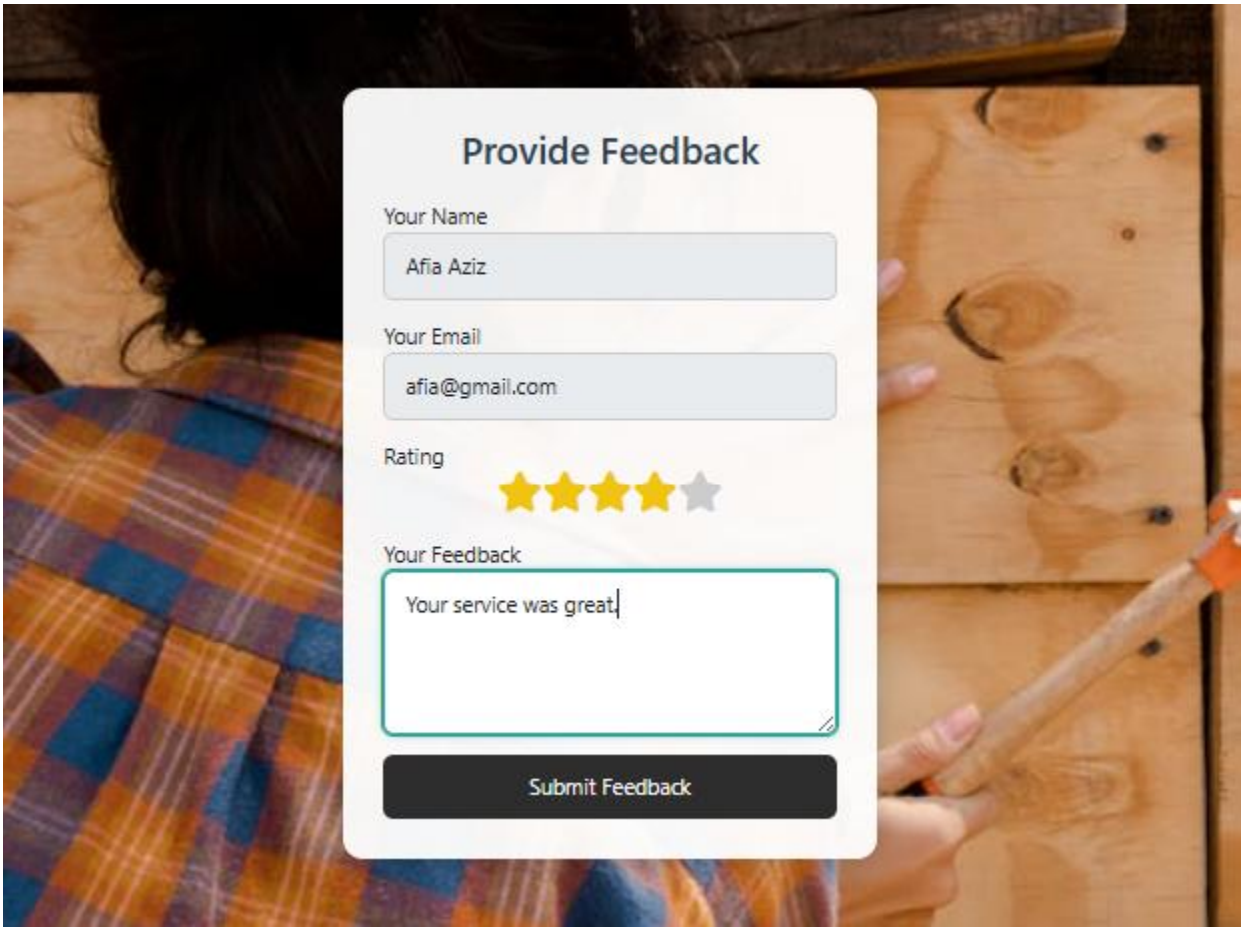



## Worker Accepts/Rejects Request:

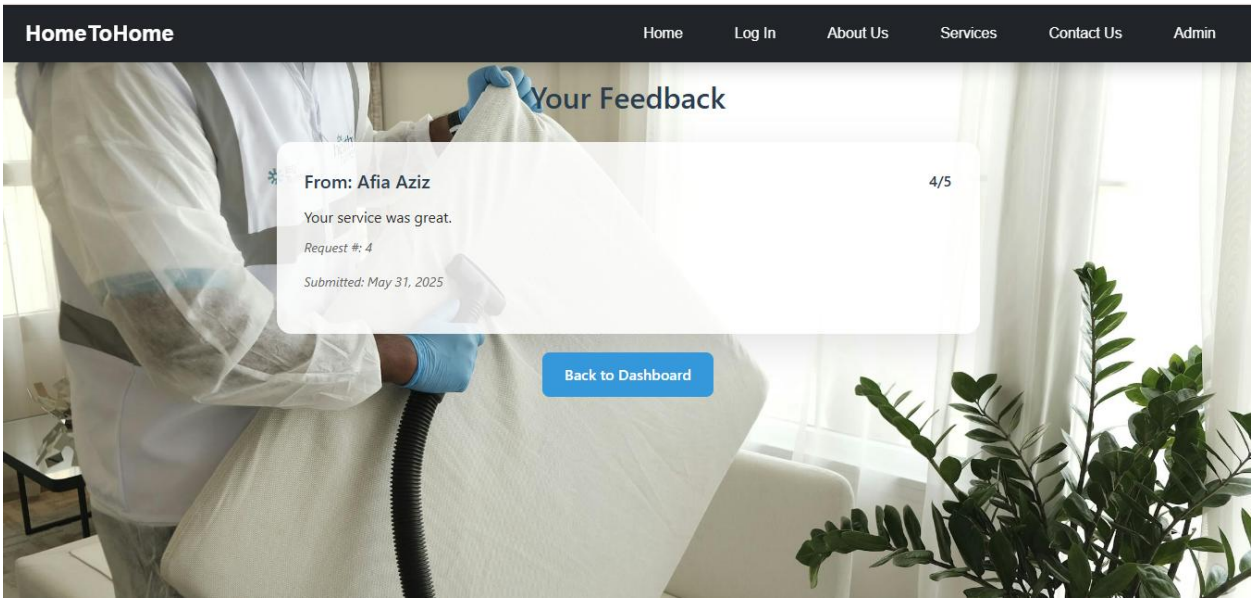
## User Dashboard – Request Tracking:




## Provide Feedback:



## Feedback Received at Worker Side:




# 7. Traceability Matrix:

| REQUIREMENT TRACEABILITY | BUSINESS REQUIREMENTS | | | | |
|---|---|---|---|---|---|
| **TEST CASES** | **BRD001** (Login) | **BRD002** (Search & Book) | **BRD003** (Contact Us) | **BRD004** (Booking Workflow) | **BRD005** (Review Requests) |
| TC01-01 | ✔ | | | | |
| TC01-02 | ✔ | | | | |
| TC01-03 | ✔ | | | | |
| TC02-01 | | ✔ | | | |
| TC02-02 | | ✔ | | | |
| TC02-03 | | ✔ | | | |
| TC03-01 | | | ✔ | | |
| TC03-02 | | | ✔ | | |
| TC03-03 | | | ✔ | | |
| TC04-01 | | | | ✔ | |
| TC04-02 | | | | ✔ | |

| | | | | | |
|---|---|---|---|---|---|
| TC04-03 | | | | ✔ | |
| TC05-01 | | | | | ✔ |
| TC05-02 | | | | | ✔ |
| TC05-03 | | | | | ✔ |

# 8. Risk Analysis:

## Risk Analysis

During the development of the *HomeToHome* project, we encountered several risks related to time, technology, scope, and deployment strategy. Below is an analysis of each key risk and how we managed or planned to handle it.

## R1 – Organizational Risk

**Risk Description:** Limited team capacity or delays in distributing tasks among team members affected the pace of feature development.
**Management Strategy:**

- We adopted a **priority-based development approach** — starting with core modules like login, booking, and feedback.
- Used a **shared task board** to assign responsibilities clearly and track progress.
- Some optional features like Admin panel were deferred for later phases to stay within deadlines.

## R2 – Technology Risk

**Risk Description:** The lack of two-factor authentication could expose the login system to security threats.
**Management Strategy:**

- We implemented **basic validation**, error handling, and hashed password storage (using database).
- Two-factor authentication was logged as a **future improvement**, with UI considerations already kept modular for easy addition.

## R3 – Requirements Risk

**Risk Description:** There were moments where we considered adding extra features like **real-time chat**, which would have required significant backend rework.
**Management Strategy:**

- We created a **feature freeze list** during development and any non-critical features were moved to a **"Phase 2 backlog."**
- Ensured that the current features were working smoothly instead of overloading the scope.

### R4 – Estimation Risk

**Risk Description:** We initially underestimated the time needed for implementing and testing the **booking flow**.
**Management Strategy:**

- We split the booking module into multiple small components (search → request → status).
- Prioritized **form validations and user-worker sync logic** early so UI and logic bugs could be resolved before deadline crunch.

### R5 – Usability Risk

**Risk Description:**
Because the system is interacting between user and worker and system is being updating in real time, which may cause confusion during workflows like:

- Booking services
- Viewing request statuses

**Management Strategy:**

- We added **confirmation and status messages** (e.g., "Request submitted", "Status: Accepted") where possible.
- Ensured **form validations show clear, immediate feedback** to guide users.

## 9. Conclusion

The development of the **HomeToHome** project provided us with a valuable opportunity to apply our theoretical knowledge of software engineering and visual programming in a practical setting. This semester project simulated a real-world scenario where users and service providers interact through an online platform, and it required us to think critically about both the frontend user experience and backend data flow.

### What We Learned:

- We gained hands-on experience in designing and developing a **role-based application** involving **Customers** and **Workers**.
- We learned to use **Blazor components** to create dynamic, interactive UI elements such as service search, feedback forms, and request dashboards.

- We understood the importance of **form validation**, **status updates**, and **real-time UI feedback** to ensure smooth user interaction.

## Understanding of VP Concepts:

This project deepened our understanding of **Visual Programming (VP)** by:

- Helping us design a **component-based UI** .
- Demonstrating the interaction between **UI components and data models**, such as dynamically binding request status or pre-filling forms for editing.
- Allowing us to simulate **real-time dashboard updates** through data binding and state management, which aligns directly with VP principles of visual interactivity.

## Future Improvements:

While the project fulfills its current requirements, we have identified several areas for potential enhancement:

- Real-time notifications for booking updates
- Advanced search filters (rating, city, availability)
- Two-factor authentication for secure login
- In-app chat system between users and workers
- Booking calendar with worker availability slots
- Recurring service subscriptions and payment plans