Palestine Polytechnic University

Collage of Information Technology and Computer Engineering

Brozar

Team members:

Mohammad Jawabree

Ibraheem Zeer

Yazan Bhais

Supervisor:

Dr. Ezdehar Jawabreh

Hebron – Palestine

2023/2024

Table of contents

List of figures **5**
Abstract **6**
الملخص **7**
Chapter 1 Introduction **8**
1.1 Overview 9
1.2 Motivation and Importance 9
1.3 Project Scope 9
1.4 Objectives 10
1.5 Project Alternatives 11
Chapter 2 Analysis of Requirements **13**
2.1 Overview 14
2.2 Actors 14
2.3 Functional Requirements 14
2.3.1 User's Side (Owner): 14
2.3.2 Admin's Side: 15
2.3.3 Visitor's Side: 15
2.3.4 Investor's Side: 15
2.4 Non-Functional Requirements 16
2.5 Use-Case Diagram: 17
2.6 User's Functional Requirements Tables 18
2.7 Admin's Functional Requirements Tables 25
2.8 Visitor's Functional Requirements Tables 30
2.9 Investor's Functional Requirements Tables 31
2.10 Context Diagram 35
Chapter 3 Architecture and Design **36**
3.1 Overview 37
3.2 Architecture Design 37
3.2.1 Alternative architecture 38
3.3 Architecture 40
3.3.1 MVCS Architecture's components 41
3.3.2 MVC Architecture' s components Description 41
3.4 Class Diagram 45
3.5 Database Logic Mapping 46
3.6 Database Description 47
1. users 47
2. owners 47
3. event_images 48
5. investors 48
6. projects 49

7. events..... 49
8. comments..... 50
9. contacts.....50
10. project_images.....51
11. requests..... 51
Chapter 4 Implementation.....52
4.1 Overview.....53
4.2 Code..... 55
4.3Interfaces.....57
Chapter 5 Testing.....63
5.1 Overview.....64
5.2 Validation..... 64
5.3 Unit testing.....64
5.4 Endpoint testing.....66
5.5 QA Manual testing..... 68
Conclusion and Future work..... 71
Conclusion.....71
Future work.....71
References..... 72

List of tables

Table 2.6.1 Registration..... 18
Table 2.6.2 Login.....19
Table 2.6.3 Logout.....19
Table 2.6.4 Reset Password..... 20
Table 2.6.5 View Profile.....20
Table 2.6.6 Edit Profile..... 21
Table 2.6.7 Add Project.....21
Table 2.6.8 Edit Project..... 22
Table 2.6.9 Delete Project.....22
Table 2.6.10 View Projects.....23
Table 2.6.11 View Events..... 23
Table 2.6.12 Register for participate in Events.....24
Table 2.6.13 Leave Comments and Rates on projects..... 24
Table 2.7.1 Login.....25
Table 2.7.2 Manage Users..... 25
Table 2.7.3 Edit User.....26
Table 2.7.4 Delete User.....26
Table 2.7.5 Manage Content.....27
Table 2.7.6 Update Project.....27
Table 2.7.7 Delete Project..... 28
Table 2.7.8 Manage Events.....28
Table 2.7.9 Create Event.....29
Table 2.7.10 View Reports..... 29
Table 2.8.1 Browse Projects.....30
Table 2.8.2 View Events.....30
Table 2.9.1 View Projects.....31
Table 2.9.2 Request Detailed Information.....31
Table 2.9.3 View User Profiles..... 32
Table 2.9.4 Registration..... 32
Table 2.9.5 Login.....33
Table 2.9.6 View Events.....33
Table 2.9.7 Leave Comments and Rates on projects..... 34

List of figures

Figure 2.5.1 Use Case Diagram....17
Figure 2.10.1 Context Diagram....35
Figure 3.2.1 MVC Architecture....40
Figure 3.4.1 Class Diagram....45
Figure 3.5.1 : Mapping....46
Figure 4.3.1 : Home page....58
Figure 4.3.2 : Create Account....58
Figure 4.3.3 : Log in....59
Figure 4.3.4 : my profile page....59
Figure 4.3.5 : Projects page....60
Figure 4.3.6 : my projects page....61
Figure 4.3.7 : admin dashboard page....61
Figure 4.3.8 : Add new project page....62

Abstract

In light of the difficult economic conditions in our beloved country, Palestine, many people struggle to find stable job opportunities.

As a result, many individuals start their own small businesses, often relying on handicrafts and personal skills to ensure a steady income.

However, entrepreneurs often face marketing and organizational challenges, which limit the reach of their projects and their ability to make good profits.

This situation inspired our final project, where we aimed to create an online platform designed to help small business owners improve the vision and organization of their projects and present them in a professional way.

On this platform, we provided services such as displaying projects online, organizing real-life bazaars, presenting projects to potential investors, and providing data to help entrepreneurs share experiences and knowledge to improve their efficiency.

الملخص

في ظل الظروف الاقتصادية الصعبة التي تواجهها دولتنا الحبيبة فلسطين، يكافح العديد من الأفراد للعثور على فرص عمل مستقرة .

ونتيجة لذلك ، يلجأ العديد من الأشخاص إلى إنشاء مشاريعهم الصغيرة الخاصة ، وغالبًا ما يعتمدون على الحرف اليدوية والمهارات الشخصية الأخرى لضمان دخل ثابت.

ومع ذلك، غالباً ما يواجه رواد الأعمال عقبات تسويقية وتنظيمية، مما يحد من وصول مشاريعهم وقدرتهم على تحقيق أرباح مجدية.

وبذلك، ألهم هذا الوضع مشروعنا النهائي، حيث كنا نهدف إلى إنشاء منصة عبر الإنترنت مصممة لمساعدة أصحاب الأعمال الصغيرة على تعزيز رؤية وتنظيم مشاريعهم، وتقديمها بطريقة احترافية.

قدمنا في هذا الموقع خدمات تشمل عرض المشاريع إلكترونيا وتنظيم البازارات على أرض الواقع، وعرض المشاريع على المستثمرين المحتملين ، وتوفير البيانات لتسهيل تبادل الخبرات والمعرفة بين رواد الأعمال لتحسين كفاءتهم.

Chapter 1 Introduction

1.1 Overview

This chapter covers the motivation, importance, scope, objectives and requirements for our project, which aims to support small business owners in Palestine by creating a user-friendly online platform.

1.2 Motivation and Importance

The idea for this project arises in response to the urgent need to support small business owners under the challenging economic conditions in Palestine. Many individuals face difficulties in getting stable employment and turn to establishing their own businesses to make a steady income. This project aims to provide the necessary support to these individuals, in order to reduce unemployment rates, improve individual incomes, and boost the local economy.

1.3 Project Scope

The scope of the project includes the development of an online platform that provides a space for small business owners to showcase their projects and some of their products or services, organize both virtual and physical bazaars, connect project owners with investors, and provide analytical data. The project will also involve using technology to enhance the platform’s functionality and increase the success rate of the projects.

1.4 Objectives

- **Develop a Comprehensive Digital Platform:** Developing a user-friendly web application that showcases the projects of handicraft and personal skill owners in a professional way.

- **Improve Accessibility and Marketing:** Making it easier for small business owners to reach a wider audience via both virtual and physical bazaars and utilize digital marketing to increase sales.

- **Enhance Investor Connections:** Provide a direct channel to connect small business owners with potential investors to fund and support their projects.

- **Utilize Data and Analytics:** This is done by generating reports and using analytical tools which would provide administrators and business owners with a basis for making data-driven decisions that improve overall project outcomes and strategic planning.

- **Support Innovation and Technology:** Encourage the use of modern technology and innovation in product and service development, which will make it easier for small and medium business owners to compete in the local market.

1.5 Project Alternatives

To build the project, various tools can be used, among which is the development of a website. This option has a range of advantages and disadvantages:

Advantages:

- **Ease of Use:** This platform is web-based and therefore accessible from any device with an internet connection, making it easy to access the website from anywhere.
- **Device Accessibility:** The website can be accessed from any browser on any operating system.
- **Easy Updates:** The website can be updated at any time with ease.
- **Lower Cost:** Typically, the cost of developing and maintaining websites is lower than that of custom applications.

Disadvantages:

- **Constant Internet Connection Required:** The website will require permanent internet connectivity in order to access and view content.

After evaluating the suggested alternatives, we decided to go with the website for system development because of its reach and flexibility.

We will incorporate responsive web design so that a more optimized viewing across all mobiles is done, thus presenting it optimally to all visitors. Responsive web design enables the content and display of the website to be flexible with the respective size and orientation of the screen, making the website feel like a mobile application when visited on a mobile device.

Mobile vs Website and why website:

First let's discuss the advantages and disadvantages of mobile applications.

Advantages :

- Ease of Use.
- Available to the user through the mobile phone.
- Easy to update.

Disadvantages :

- The need to connect to the internet always.
- People avoid downloading apps due to space limitation on their devices.
- The diversity of phone operating systems.

After looking at the suggested alternatives, we chose the website to build the system due to its accessibility and versatility, but we can add the concept of responsive web design to ensure a better experience when accessed from mobile devices.

Responsive web design allows the website layout and content to adapt and adjust based on the screen size and orientation, providing an optimal user experience on mobile devices. By employing responsive design techniques, we can make the website appear and function similarly to a mobile app when accessed on a mobile device.

Chapter 2 Analysis of Requirements

2.1 Overview

This chapter outlines the functional and non-functional requirements for the Brozar platform. It defines the roles of various actors involved in the system and details the specific functionalities each actor can perform.

2.2 Actors

- User (Owner): business owners and registered users.
- Admin: Website administrators.
- Visitor: Unregistered users who browse the website.
- Investor: Potential investors in projects on the website.

2.3 Functional Requirements

2.3.1 User's Side (Owner):

- Users are able to register to the website by providing necessary information.
- Users are able to log in and log out to the website using their credentials.
- Users are able to view and edit their profile information including reset password, change email and other information.
- Users are able to manage their projects including adding new projects, edit and delete.
- Users are able to browse all public projects on the website.
- Users are able to browse all active owner profiles on the website.
- Users are able to view events and register with upcoming and ongoing events on the website.
- Users can leave comments and rate on projects.

2.3.2 Admin's Side:

- The admin can log in and log out to the website using their credentials.
- The admin is able to manage user accounts and contents on the website, including viewing, editing, and deleting user profiles.
- The admin is able to create new events and manage existing events, including viewing, editing, and deleting events.
- The admin is able to view detailed reports on website usage and performance.

2.3.3 Visitor’s Side:

- Visitors are able to browse available projects on the website.
- Visitors can view events on the website.

2.3.4 Investor’s Side:

- Investors are able to register to the website by providing necessary information.
- Investors are able to log in and log out to the website using their credentials.
- Investors can browse and view all public projects available on the website.
- Investors can request detailed information about users' projects, including sales statistics, profits, and monthly/annual income and performance indicators.
- Investors can view events on the website.
- Investors can leave comments and rate on projects.

2.4 Non-Functional Requirements

- The platform is available for all users so it can be used at all times when the mobile phone/pc and internet connection are available.
- The platform should be easy to use, provide convenient interfaces for the user, and enable users to access the system easily and quickly.
- The platform should deliver fast and responsive performance, ensuring that all user interactions (such as page loads and form submissions) are processed within 2 seconds under normal operating conditions.
- The platform should be consistently its intended functions without failure and able to handle increased load or demand without compromising performance.

2.5 Use-Case Diagram:

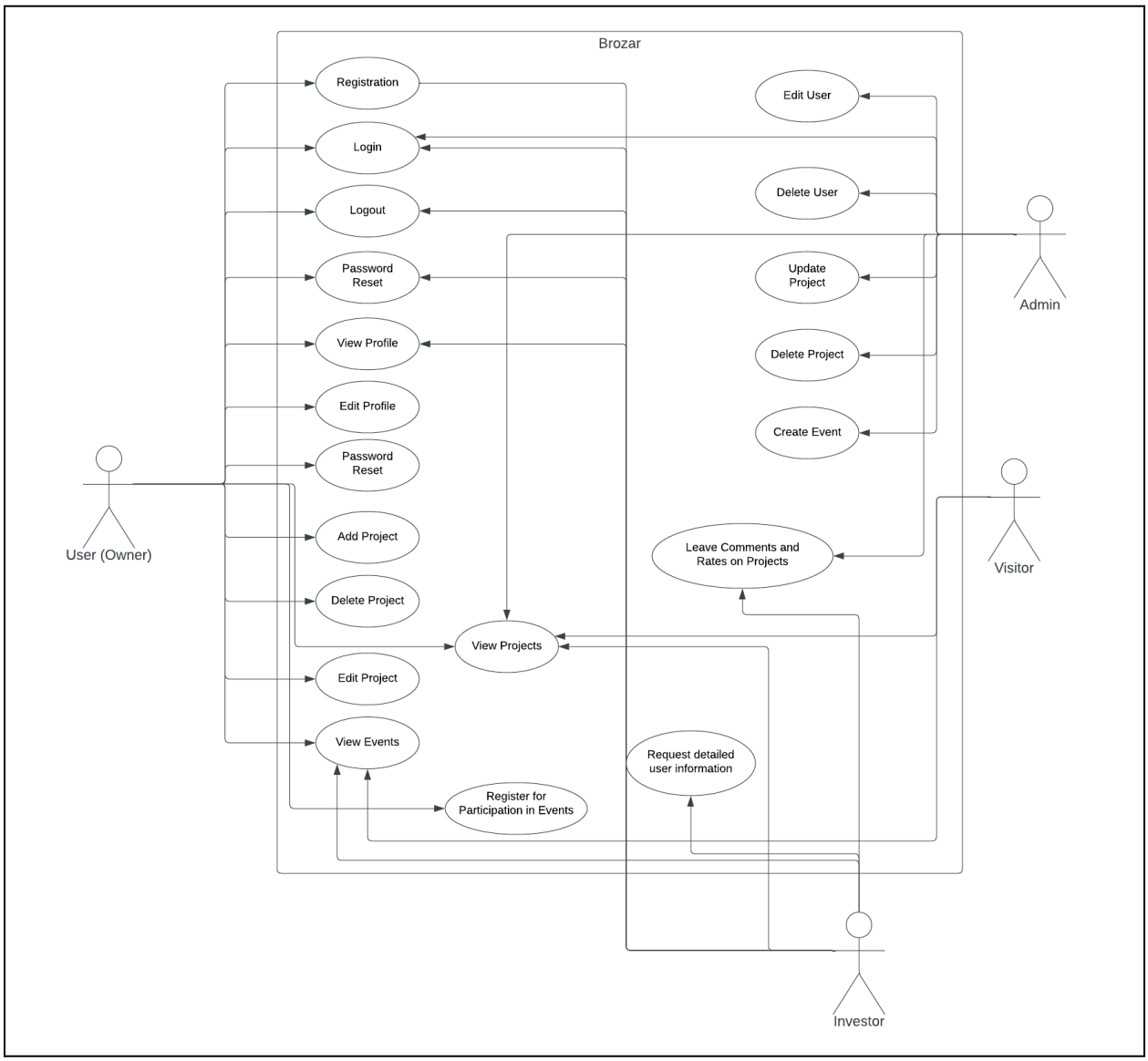


Figure 2.5.1 Use Case Diagram

2.6 User's Functional Requirements Tables

Requirement	Registration
Actor	User
Objectives	Create a new account.
Precondition	The website is being open.
Scenario	1. The user opens the website from any browser. 2. The user chooses the "Sign Up" option. 3. The user fills in the required information: a. Username b. Email c. Password d. Other info 4. The user selects their field of interest. 5. The user presses the "Sign Up" button.
Exception	1 The user enters an email address in a language other than English. 2.The user enters an invalid email. 3.The user enters a password with less than 8 characters. 4.The user does not fill all the required fields. 5.No Internet connection.

Table 2.6.1 Registration

Requirement	Login
Actor	User
Objectives	Access account.
Precondition	User must be registered.
Scenario	1. The user selects 'Login'. 2. The user enters their credentials. 3. The user clicks 'Submit'. 4. The system verifies the credentials and grants access if correct.
Exception	1. Incorrect credentials. 2. The user’s account is locked due to multiple failed login attempts. 3. The user’s account is not activated or has been deactivated. 4. No Internet connection.

Table 2.6.2 Login

Requirement	Logout
Actor	User
Objectives	Log out of account.
Precondition	User must be logged in
Scenario	1. The user selects 'Logout'. 2. The system terminates the session. 3. The user is redirected to the homepage or login screen.
Exception	1. The system fails to process the logout due to a server error. 2. The user’s session has already expired. 3. No Internet connection.

Table 2.6.3 Logout

Requirement	Password Reset
Actor	User
Objectives	Recover account password
Precondition	ser must have a registered email
Scenario	1. The user selects 'Forgot Password'. 2. The user enters their registered email. 3. The user clicks 'Submit'. 4. The system sends an email with password reset instructions. 5. The user follows the instructions to reset their password.
Exception	1. The user enters an email that is not registered in the system. 2. The user enters an invalid or incorrectly formatted email address. 3. The user does not complete the captcha verification (if applicable). 4. No Internet connection.

Table 2.6.4 Reset Password

Requirement	View Profile
Actor	User
Objectives	View own profile
Precondition	User must be logged in
Scenario	1. The user selects 'Profile'. 2. The system retrieves and displays the user’s profile information.
Exception	1. The system fails to retrieve the profile information due to a server error. 2. No Internet connection.

Table 2.6.5 View Profile

Requirement	Edit Profile
Actor	User
Objectives	Update profile information
Precondition	User must be logged in
Scenario	1. The user selects 'Edit Profile'. 2. The user updates their information (e.g., name, email, password). 3. The user clicks 'Save'.
Exception	1. The user enters invalid data (e.g., invalid email format). 2. The system fails to save the updated information due to a server error. 3. No Internet connection.

Table 2.6.6 Edit Profile

Requirement	Add Project
Actor	User
Objectives	Add a new project
Precondition	User must be logged in
Scenario	1. The user selects 'Add Project'. 2. The user enters project details. 3. The user clicks 'Submit'. 4. The system validates the data and adds the project.
Exception	1. Invalid data (e.g., missing required fields). 2. The system fails to save the project due to a server error. 3. No Internet connection.

Table 2.6.7 Add Project

Requirement	Edit Project
Actor	User
Objectives	Edit existing project
Precondition	User must be logged in
Scenario	1. The user selects 'My Projects'. 2. The user chooses a project to edit. 3. The user updates the project details. 4. The user clicks 'Save'. 5. The system validates the data and updates the project.
Exception	1. The system fails to save the updated project due to a server error. 2. The user enters invalid data. 3. No Internet connection.

Table 2.6.8 Edit Project

Requirement	Delete Project
Actor	User
Objectives	Remove a project
Precondition	User must be logged in
Scenario	1. The user selects 'My Projects'. 2. The user chooses a project to delete. 3. The user confirms the deletion. 4. The system deletes the project from the database.
Exception	1. The system fails to delete the project due to a server error. 2. No Internet connection.

Table 2.6.9 Delete Project

Requirement	View Projects
Actor	User
Objectives	Browse and view projects
Precondition	None
Scenario	1. The user navigates to 'Projects'. 2. The user filters and searches for projects. 3. The user views project details.
Exception	1. The system fails to retrieve the project information due to a server error. 2. No Internet connection.

Table 2.6.10 View Projects

Requirement	View Events
Actor	User
Objectives	View upcoming events
Precondition	None
Scenario	1. The user navigates to 'Events'. 2. The user views event details and schedules.
Exception	1. The system fails to retrieve the event information due to a server error. 2. No Internet connection.

Table 2.6.11 View Events

Requirement	Register for Participation in Events
Actor	User
Objectives	Register for events such as bazaars
Precondition	User must be logged in
Scenario	1. The user views event details. 2. The user registers for the event. 3. The system processes the registration and confirms participation.
Exception	1. The system fails to register the user due to a server error. 2. The user enters invalid registration information. 3. No Internet connection.

Table 2.6.12 Register for participate in Events

Requirement	Leave Comments and Rates
Actor	User
Objectives	Leave comments on projects
Precondition	User must be logged in
Scenario	1. The user selects a project. 2. The user writes a comment and selects stars (rating from 1 to 5). 3. The user clicks 'Submit comment'. 4. The system validates and posts the comment.
Exception	1. The user enters invalid data such as an empty comment. 2. The system fails to post the comment due to a server error. 3. No Internet connection.

Table 2.6.13 Leave Comments and Rates on projects

2.7 Admin’s Functional Requirements Tables

Requirement	Login
Actor	Admin
Objectives	Access admin panel
Precondition	Admin must be registered
Scenario	1. Admin selects 'Login'. 2. Admin enters credentials. 3. Admin clicks 'Submit'. 4. The system verifies the credentials and grants access if correct.
Exception	1. Incorrect credentials. 2. No Internet connection.

Table 2.7.1 Login

Requirement	Manage Users
Actor	Admin
Objectives	Manage user accounts
Precondition	Admin must be logged in
Scenario	1. Admin selects 'Manage Users'. 2. Admin views, edits, or deletes user accounts.
Exception	1. The system fails to retrieve user data due to a server error. 2. The system fails to update or delete the user account due to a server error. 3. No Internet connection.

Table 2.7.2 Manage Users

Requirement	Edit User
Actor	Admin
Objectives	Edit user profiles
Precondition	Admin must be logged in
Scenario	1. Admin selects a user. 2. Admin edits user information. 3. Admin clicks 'Save'. 4. The system validates and updates the user profile.
Exception	1. The system fails to update the user profile due to a server error. 2. The admin enters invalid data. 3. No Internet connection.

Table 2.7.3 Edit User

Requirement	Delete User
Actor	Admin
Objectives	Delete user profiles
Precondition	Admin must be logged in
Scenario	1. Admin selects a user. 2. Admin confirms deletion. 3. The system deletes the user profile.
Exception	1. The system fails to delete the user profile due to a server error. 2. No Internet connection.

Table 2.7.4 Delete User

Requirement	Manage Content
Actor	Admin
Objectives	Manage content on the website
Precondition	Admin must be logged in
Scenario	1. Admin selects 'Manage Content'. 2. Admin adds, edits, or deletes content.
Exception	1. The system fails to retrieve, update, or delete content due to a server error. 2. No Internet connection.

Table 2.7.5 Manage Content

Requirement	Update Project
Actor	Admin
Objectives	Update project details
Precondition	Admin must be logged in
Scenario	1. Admin selects a project. 2. Admin edits project information. 3. Admin clicks 'Save'.
Exception	1. The system fails to update the project due to a server error. 2. Admin enters invalid data. 3. No Internet connection.

Table 2.7.6 Update Project

Requirement	Delete Project
Actor	Admin
Objectives	Delete projects
Precondition	Admin must be logged in
Scenario	1. Admin selects a project. 2. Admin confirms deletion. 3. The system deletes the project from the database.
Exception	1. The system fails to delete the project due to a server error. 2. No Internet connection.

Table 2.7.7 Delete Project

Requirement	Manage Events
Actor	Admin
Objectives	Manage existing events
Precondition	Admin must be logged in
Scenario	1. Admin selects 'Manage Events'. 2. Admin views, creates, or deletes events.
Exception	1. The system fails to retrieve, update, or delete events due to a server error. 2. No Internet connection.

Table 2.7.8 Manage Events

Requirement	Create Event
Actor	Admin
Objectives	Create new events
Precondition	Admin must be logged in
Scenario	1. Admin selects 'Create Event'. 2. Admin enters event details. 3. Admin clicks 'Submit'. 4. The system validates and creates the event.
Exception	1. Invalid data entered by the admin. 2. The system fails to create the event due to a server error. 3. No Internet connection.

Table 2.7.9 Create Event

Requirement	View Reports
Actor	Admin
Objectives	View detailed reports
Precondition	Admin must be logged in
Scenario	1. Admin selects 'View Reports'. 2. Admin views detailed usage and performance reports.
Exception	1. The system fails to retrieve the reports due to a server error. 2. No Internet connection.

Table 2.7.10 View Reports

2.8 Visitor’s Functional Requirements Tables

Requirement	Browse Projects
Actor	Visitor
Objectives	Browse available projects
Precondition	None
Scenario	1. Visitor navigates to 'Projects'. 2. Filters and searches projects. 3. Views project details.
Exception	1. System error 2. No Internet connection.

Table 2.8.1 Browse Projects

Requirement	View Events
Actor	Visitor
Objectives	View upcoming events
Precondition	None
Scenario	1. Visitor navigates to 'Events'. 2. Views event details and schedules.
Exception	1. System error 2. No Internet connection.

Table 2.8.2 View Events

2.9 Investor’s Functional Requirements Tables

Requirement	View Projects
Actor	Investor
Objectives	Browse and view projects
Precondition	None
Scenario	1. Investor navigates to 'Projects'. 2. Investor filters and searches for projects. 3. Investor views project details.
Exception	1. The system fails to retrieve the project information due to a server error. 2. No Internet connection.

Table 2.9.1 View Projects

Requirement	Request Detailed Information
Actor	Investor
Objectives	Request detailed information about users and their projects.
Precondition	the investor must be registered and logged in.
Scenario	1. Investor navigates to 'Projects'. 2. Investor filters and searches for projects. 3. Investor views project details.
Exception	1. The system fails to retrieve the detailed information due to a server error. 2. The data requested by the investor is unavailable. 3. No Internet connection.

Table 2.9.2 Request Detailed Information

Requirement	View User Profiles
Actor	Investor
Objectives	View detailed user profiles.
Precondition	the investor must be registered and logged in.
Scenario	1. Investor selects a user profile. 2. The system retrieves and displays detailed user information, including financial metrics, project performance, and historical data.
Exception	1. The system fails to retrieve the user profile due to a server error. 2. No Internet connection.

Table 2.9.3 View User Profiles

Requirement	Registration
Actor	Investor
Objectives	Create a new account.
Precondition	The website is being open.
Scenario	1. The Investor opens the website from any browser. 2. The Investor chooses the "Sign Up" option. 3. The Investor fills in the required information: a. Username b. Email c. Password d. Other info 4. The Investor selects their account type. 5. The user presses the "Sign Up" button.
Exception	1. The Investor enters an email address in a language other than English. 2. The Investor enters an invalid email. 3. The Investor enters a password with less than 8 characters. 4. The Investor does not fill all the required fields. 5. No Internet connection.

Table 2.9.4 Registration

Requirement	Login
Actor	Investor
Objectives	Access account.
Precondition	User must be registered.
Scenario	1. The Investor selects 'Login'. 2. The Investor enters their credentials. 3. The Investor clicks 'Submit'. 4. The system verifies the credentials and grants access if correct.
Exception	1. Incorrect credentials. 2. The Investor’s account is locked due to multiple failed login attempts. 3. The Investor’s account is not activated or has been deactivated. 4. No Internet connection.

Table 2.9.5 Login

Requirement	View Events
Actor	Investor
Objectives	View upcoming events
Precondition	None
Scenario	1. Investor navigates to 'Events'. 2. Investor event details and schedules.
Exception	1. System error 2. No Internet connection.

Table 2.9.6 View Events

Requirement	Leave Comments and Rates
Actor	Investor
Objectives	Leave comments on projects
Precondition	User must be logged in
Scenario	1. The investor selects a project. 2. The investor writes a comment and selects stars (rating from 1 to 5). 3. The investor clicks 'Submit comment'. 4. The system validates and posts the comment.
Exception	1. The user enters invalid data such as an empty comment. 2. The system fails to post the comment due to a server error. 3. No Internet connection.

Table 2.9.7 Leave Comments and Rates on projects

2.10 Context Diagram

Figure 2.10.1 shows the most important elements surrounding the system or individuals and

their interaction mechanism with the system, they are mainly: admin,user, investor and visitor.

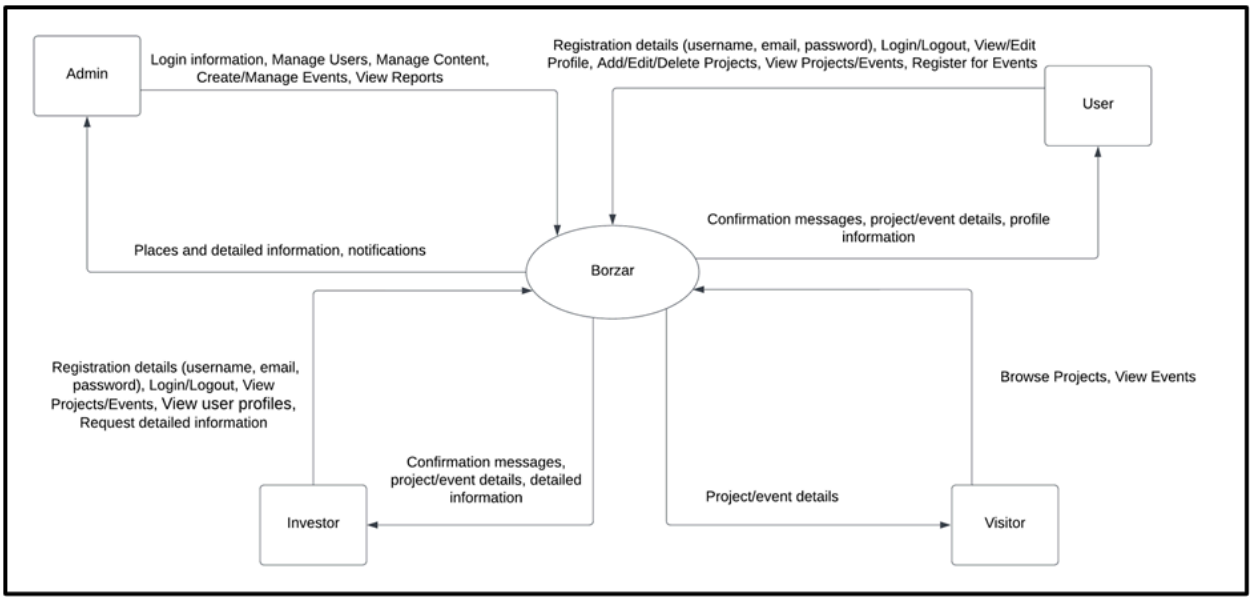


Figure 2.10.1 Context Diagram

Chapter 3 Architecture and Design

3.1 Overview

In this chapter, we will discuss the system's architecture, database tables, database mapping and the interfaces of the application.

3.2 Architecture Design

Model , View , Controller , Service (MVCS)

After studying many architectural patterns, we decided to create a combination of two powerful and important architectures recently, so we decided to combine MVC[1] and Service. Several factors contributed to this decision:

- **Separation of Concerns:** MVC's and Services clear separation of concerns aligns well with our project's requirements.
- **Adoption:** The widespread adoption of MVC and Services across different frameworks and technologies is a significant advantage.
- **Reusability:** The ability to reuse components across different parts of the application will enhance productivity and reduce development time and can be called from multiple places in the application or used with other applications.
- **Ecosystem:** By compound between MVC and Services, we can leverage the existing ecosystem of tools, libraries, and frameworks that support this architectural pattern.

Separates the application into four interconnected parts:

- **Model**: Manages data and business logic.
- **View**: Handles user interface and presentation.
- **Controller**: handle HTTP requests and responses.
- **Service:** contain the core business logic of the application

Pros :

- Improves code reusability and testability.
- Enhances separation of concerns.
- Simplifies UI development and maintenance.

Cons :

- Can become complex for large applications.
- Can involve additional code and complexity when the data model and interactions are simple.

3.2.1 Alternative architecture

There is also an other architecture that known as Layered Architecture (3-tier Architecture)[2] That separates application into three interconnected parts:

● Presentation Layer: Represents the user interface of the application and is responsible for presenting data to the user and receiving user input .

● Application Layer: Contains the business logic of the application and performs the core processing of the data .

● Data Storage Layer: Responsible for managing the data storage and retrieval in the system.

Advantages :

● Promotes modularity and loose coupling.
● Enhances maintainability and testability.
● Simplifies development and deployment.

Disadvantages :

- The communication between layers in a distributed architecture can introduce additional latency and overhead, which may have an impact on the overall performance of the system.
- May require additional layers for complex scenarios.

After studying both the MVC and Three-Tier Architecture, we have chosen MVC as the preferred architectural pattern for our project. Several factors contributed to this decision.

Firstly, MVC's clear separation of concerns aligns well with our project's requirements. Secondly, the widespread adoption of MVC across different frameworks and technologies is another significant advantage. By choosing MVC, we can leverage the existing ecosystem of tools, libraries, and frameworks that support this architectural pattern. Finally, The ability to reuse components across different parts of the application will also enhance productivity and reduce development time, benefiting our project in the long run .

3.3 Architecture

In our web application we choose an MVCS architecture which is an architectural design pattern that organizes an application's logic into distinct components, each of which carries out a specific set of tasks. The components interact with each other to ensure that the application’s functionality is delivered in a coordinated and consistent manner.

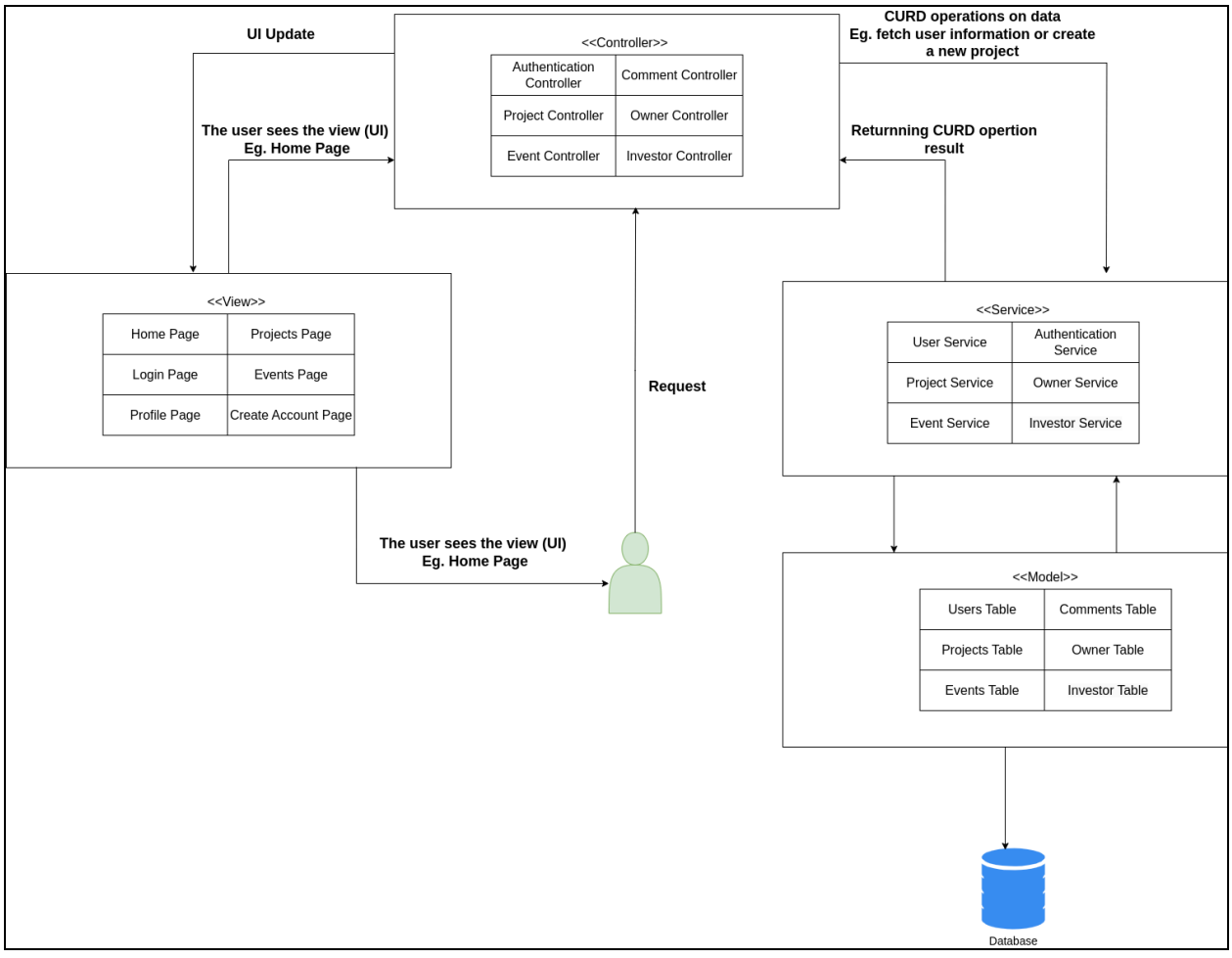


Figure 3.2.1 MVC Architecture

3.3.1 MVCS Architecture’s components

1. **Model**
2. **Service**
3. **Controller**
4. **View**

3.3.2 MVC Architecture’ s components Description

1. Model

The Model represents the application’s data, making it independent of how the data is presented. Class diagram represent:

- Classes representing entities and their relationships.
- Data attributes.

2. Service

The service layer in our architecture encapsulates the business logic and acts as the intermediary between the controller and the model. By implementing all business rules and logic, we can also reuse it across different controllers or parts of the application or even other services so we achieve the DRY principle (Don't Repeat Yourself), increasing maintainability, reusability, and separation of concerns.

4. Controller

The Controller acts as an intermediary between the services and the View. It handles requests originating from the view and invokes the appropriate service methods based on the type of each request. Additionally, it processes the output from service calls when necessary and formats the responses before returning them to the view.[1]

We have several controllers such as :

3.1) Authentication Controller

- **Responsible for:** Managing user authentication-related operations.
- **Some of its methods:**
    - **RegisterUser:** Registers a new user in the system by collecting user details such as name, email, and password.
    - **LoginUser:** Authenticates the user by validating the provided credentials.
    - **ResetPassword:** Sends a password reset link to the user's registered email address.

3.3) Event Controller

- **Responsible for:** Handling operations related to events on the platform.
- **Some of its methods:**
    - **CreateEvent:** Admins can create new events, specifying details such as event type, date, and location.
    - **UpdateEvent:** Modifies the details of an existing event.
    - **DeleteEvent:** Allows the admin to delete an event that is no longer relevant.
    - **ViewEvent:** Retrieves and displays information about a specific event.

3.2) Project Controller

- **Responsible for:** Managing operations related to user projects.
- **Some of its methods:**
    - **AddProject:** Allows users to add a new project by providing details such as project name, description, and associated images.
    - **EditProject:** Enables users to modify the details of an existing project they own.
    - **DeleteProject:** Removes a project from the system that the user no longer wants to maintain.
    - **ViewProject:** Retrieves and displays detailed information about a specific project.

3.4) Comment Controller

- **Responsible for:** Managing user comments on projects.
- **Some of its methods:**
    - **AddComment:** Allows users to add a comment to a specific project to provide feedback or suggestions.
    - **EditComment:** Enables users to modify their previous comments.
    - **DeleteComment:** Allows users to delete their comments from a project.
    - **ViewComments:** Retrieves and displays all comments related to a specific project.

3.5) Owner Controller

- **Responsible for:** Managing operations related to project owners.
- **Some of its methods:**
    - **ViewOwnerProfile:** Displays the profile information of the project owner.
    - **EditOwnerProfile:** Enables the owner to update their profile details.
    - **ViewOwnerProjects:** Retrieves all projects associated with a specific owner.

4. View

The View is responsible for presenting data to the user in a user-friendly format. It represents the user interface (UI) and does not contain any business logic. The View gets updated whenever the underlying data in the Model changes.[1] It often implemented as interfaces, specifying:

- Components for visual elements.
- Methods for displaying data.
- Mechanisms for user interaction.

3.4 Class Diagram

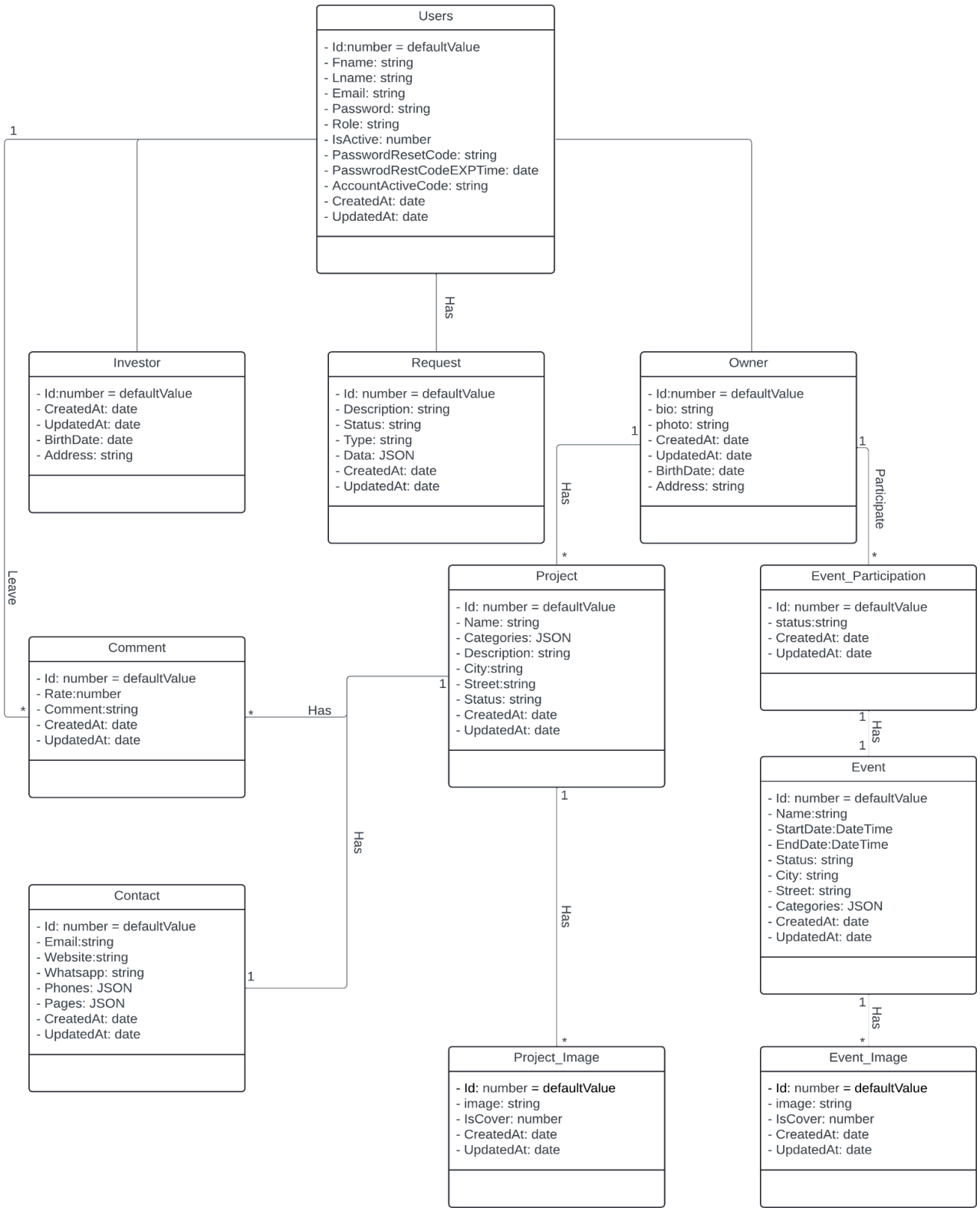


Figure 3.4.1 Class Diagram

3.5 Database Logic Mapping

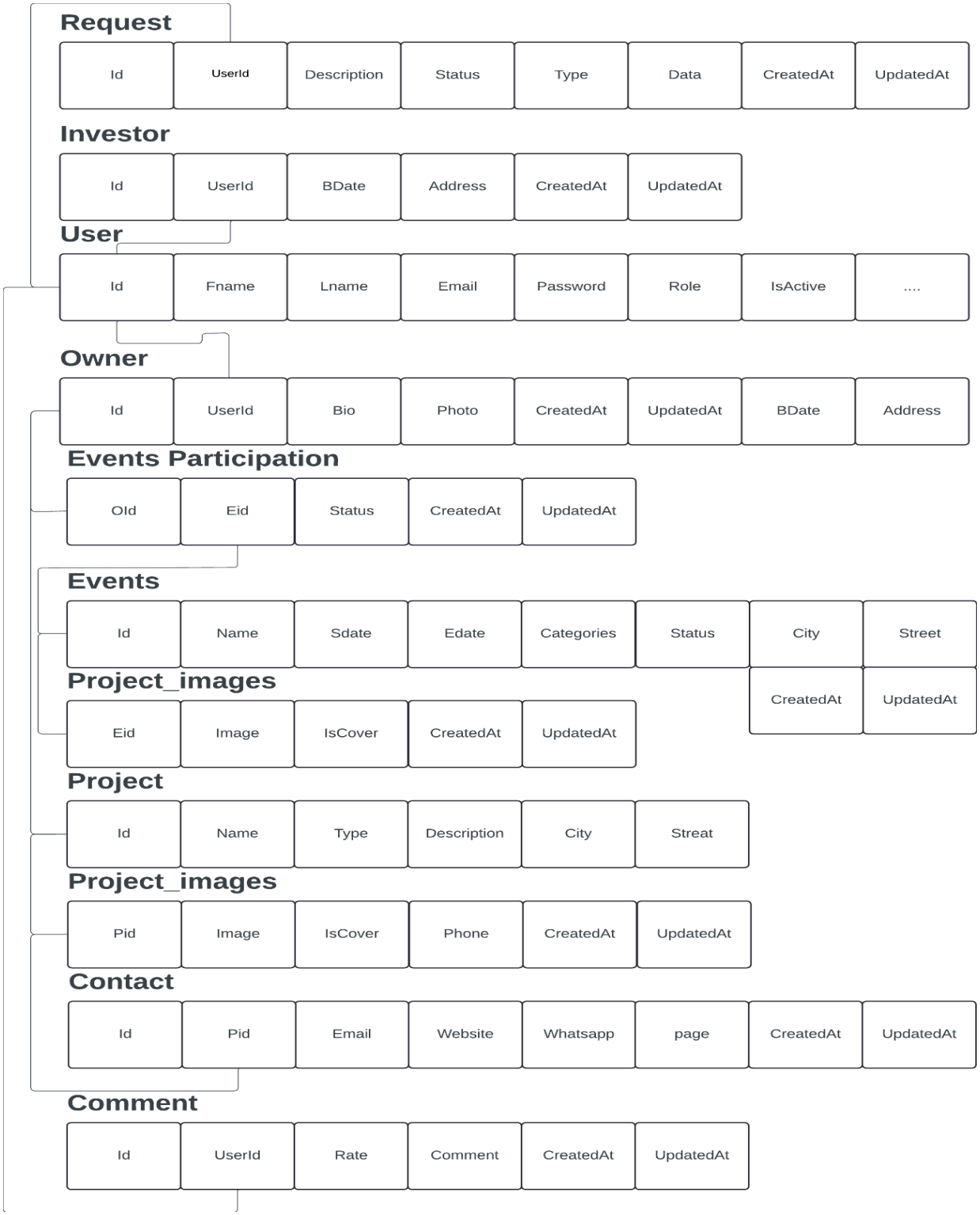


Figure 3.5.1 : Mapping

3.6 Database Description

1. users

- id: unique number for each user (PK)
- first_name: string, not-null, the first name of the user.
- last_name: string, not-null, the last name of the user.
- email: string, unique, not-null, the user's email address.
- password: string, not-null, with a minimum size of 8 characters.
- role: enum('admin','owner','investor'), not-null, the role of the user.
- is_active: number, not-null, check if the account are active or not
- password_reset_code:string, with a minimum of 8 char
- password_reset_code_expiration_time datetime: Datetime
- account_activation_code: string, activation code from email sent via email.
- created_at: Datetime
- updated_at: Datetime

2. owners

- id: unique number for each owner (PK).
- user_id: number, not-null, inherit all User data
- bio: string, not-null, bio and info about the Owner
- photo: string, link of the profile photo
- created_at: Datetime
- updated_at: Datetime
- birth_date: Datetime
- address:string, not-null, the owner address

3. event_images

- id: number for each event image (PK).
- event_id: number, not-null represents the ID of the event (FK)
- image: string, not-null, add image for events
- is_cover: number, chose one of the images to be cover
- created_at: Datetime
- updated_at: Datetime

4. event_participations

- id: unique number for each event participation (PK).
- owner_id: number, represents the ID of the owner (FK).
- event_id: number, represents the ID of the event(FK).
- status: enum('pending','accepted','rejected') the status of event participation
- created_at: Datetime
- updated_at: Datetime

5. investors

- id: unique number for each investor (PK).
- user_id: number, not-null, inherit all User data
- created_at: Datetime
- updated_at: Datetime
- birth_date: Datetime, not-null
- address: string, not-null, the address for the investor

6. **projects**
    - id: unique number for each project (PK).
    - name: string, not-null, the name of the project.
    - categories: JSON, not-null, the type of project.
    - description: string, not-null, a detailed description of the project.
    - city: string, not-null, the city where the project is based.
    - street: string, not-null, the street address of the project.
    - owner_id: number, represents the ID of the owner (FK).
    - status: enum('pending','accepted','rejected') the status of project
    - created_at: Datetime
    - updated_at: Datetime

7. **events**
    - id: unique number for each event (PK).
    - name: string, not-null, the name of the event.
    - start_date: datetime, not-null, the start date and time of the event.
    - end_date: datetime, not-null, the end date and time of the event.
    - status: enum('pending','accepted','rejected') the status of event
    - categories: JSON, not-null, the type of the event.
    - city: string, not-null, the city where the event is held.
    - street: string, not-null, the street address of the event.
    - created_at: Datetime
    - updated_at: Datetime

8. **comments**
    - id: unique number for each comment (PK).
    - user_id: number, represents the ID of the user who made the comment (FK).
    - rate: number, the rating given by the user (e.g., 1-5).
    - comment: string, not-null, the content of the comment.
    - project_id: unique number, represents the ID of the project the comment relates to (FK).
    - created_at: Datetime
    - updated_at: Datetime

9. **contacts**
    - id: unique number for each contact (PK).
    - project_id: number, represents the ID of the project the contact is associated with (FK).
    - email: string, not-null, the email address for contacting the project.
    - website: string, the website of the project.
    - whatsapp: string, the WhatsApp number for contacting the project.
    - phones:JSON, able to add more phone numbers.
    - pages:JSON, able to add more pages.
    - created_at: Datetime
    - updated_at: Datetime

10. **project_images**
    - id: unique number for each project image(PK).
    - project_id: number, represents the ID of the project the image belongs to (FK).
    - image: string, not-null, the name of the image file.
    - Is_cover: number, specify whether it is a cover photo or not.
    - created_at: Datetime
    - updated_at: Datetime

11. **requests**
    - id: unique number for each request(PK).
    - user_id: number, represents the ID of the user the request belongs to (FK).
    - description: string, a detailed description of the request.
    - status: enum('pending','accepted','rejected') the status of requests
    - type: string, the type of request
    - data: JSON, not-null, the data of request
    - created_at: Datetime
    - updated_at: Datetime

Chapter 4 Implementation

4.1 Overview

In this chapter, we will discuss the techniques used to build backend , frontend , deploy and another tools

- Backend :
    - **NodeJs and ExpressJs:** The Runtime environment and backend framework[3],[4].
    - **MySQL:** it is an open-source relational database management system[5].
    - **Sequelize:** It is a library in npm that makes it easy to manage a SQL database and easy-to-use and promise-based Node.js ORM[6].
    - **JWT:** JSON Web Token, it is an open standard used to share security information between two parties - a client and a server[7].
    - **Multer:** it is middleware for handling multipart/form-data , which is primarily used for uploading files.
    - **Bcrypt:** it is a cryptographic hash function designed for password hashing and safe storing in the backend of applications[8].
    - **Jest:** it is a JavaScript testing framework designed to ensure correctness of any JavaScript codebase[9].
    - **Express openapi validator:** is a library that integrates with new and existing API applications, used for validating requests using existing swagger/OpenAPI specifications without the need to modify application code or follow a particular coding convention.

- Frontend :
    - **Figma:** to create user interface designs
    - **ReactJs:** it is a front-end library that aims to build user interfaces based on components[10]
    - **CSS:** a web language for designing and styling
    - **Bootstrap:** CSS Framework
    - JWT[7]
    - **Axios:** it is a library to fetch API's
- Deploy and Tools :
    - **Hetzenr:** Host server where we deploy and upload all files and the database on it
    - **Cloudflare:** The place we take our domain name (Brozar.net) , caching and DNS management of the domain and all its subdomains.[11]
    - **Nginx:** it is an HTTP web server, reverse proxy, content cache, load balancer, TCP/UDP proxy server, and mail proxy server.[12]
    - **MailJet**: It is a transactional email service provider that helps businesses create, send, and track email[13].
    - **Swagger and Postman:** test , try and make documentation for all endpoints[14]
    - **GitHub:** To collaborate easily between team members [15]
    - **Docker:** To containerize the project and assist in the deployment[16]
    - **Notion and Trello:** For manage and distribute tasks among the team and track progress
    - **SonarQube:** it is for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs and code smells on 29 programming languages.[17]

4.2 Code

Backend Code , ExpressJs framework

```
router.post('/auth/register', authController.createAccount);
router.get('/auth/activate-account', authController.activateAccount);
router.post('/auth/login', authController.login);
router.post(
  '/auth/change-password',
  authenticate,
  authorizeRoles(['owner', 'investor']),
  authController.changePassword,
);
router.get('/auth/logout', authenticate, authorizeRoles(['owner', 'investor']), authController.logout);
router.post('/auth/forgot-password', authController.forgotPassword);
router.post('/auth/reset-password/verify-code', authController.verifyPasswordResetCode);
router.post('/auth/reset-password', authController.resetPassword);
```

Route: to chose the HTTP request method and and link it with controllers

```
async createAccount(req, resp, next)
    {
        const accountData = req.body;

        try {

            const {password, passwordVerify} = accountData;

            if (password =/= passwordVerify ) {
                throw new passwordNotMatch('Passwords do not match');
            }

            const account = await authService.createAccount(accountData);
            return resp.json(account);
        } catch (error) {
            console.log(error);
            next(error);
        }
    }
```

AuthController: to create handle HTTP response and link to the services

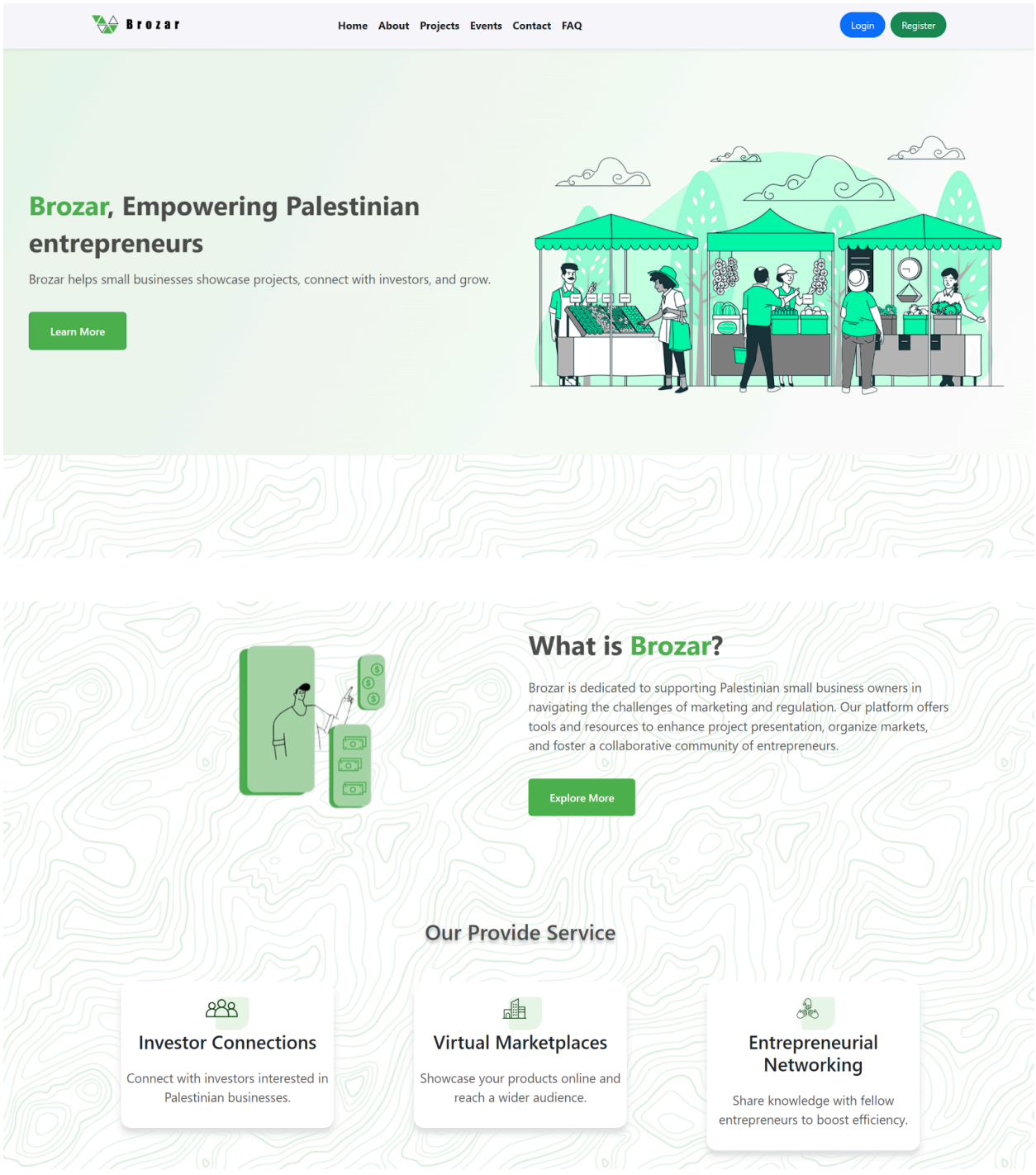
```
async createAccount(data) {
        let account;
        if (data.role === "owner")
            account = await ownerService.createOwner(data);
        else if (data.role === "investor")
            account = await investorService.createInvestor(data);
        else
            throw new Error("Invalid role provided. Must be 'owner' or 'investor'.");

        const user = await userService.getUserByEmail(data.email);
        const code = crypto.randomBytes(64).toString('hex');
        user.account_activation_code = code;
        await user.save();

        const activationLink = `${config.app.AppApiBaseUrl}/api/app/auth/activate-account/?code=${code}
        &email=${encodeURIComponent(user.email)}`;

        await sendMail({
            to: user.email,
            subject: config.mail.mailData.accountActivationEmail.subject,
            templateName: config.mail.mailData.accountActivationEmail.template,
            templateData: {
                firstName: user.firstName,
                activationLink,
            }
        });
        return account;
    }
```

authService: here we handle the logic

```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: brozar_api
    ports:
      - "3000:3000"

    volumes:
      - "/root/brozar_uploads/images:/app/public/images"

    extra_hosts:
      - "host.docker.internal:host-gateway"
```

Docker-compse: to run an instance the image

4.3Interfaces



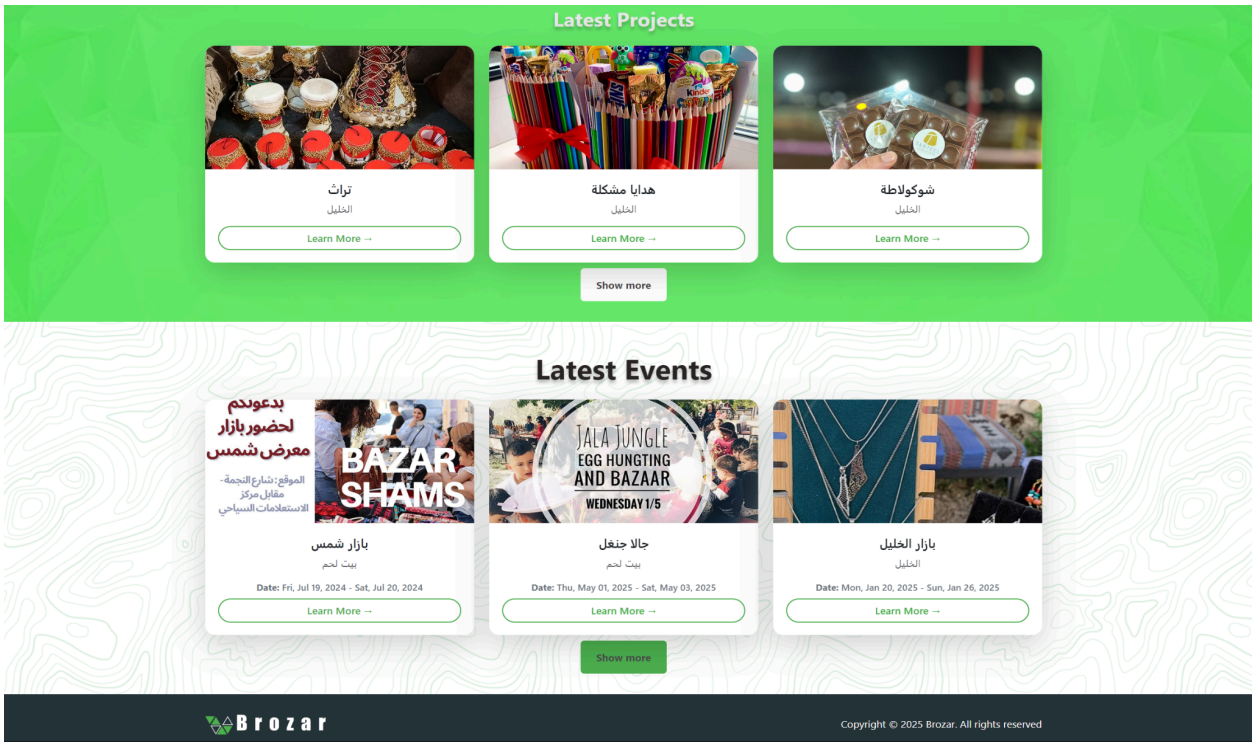


Figure 4.3.1 : Home page

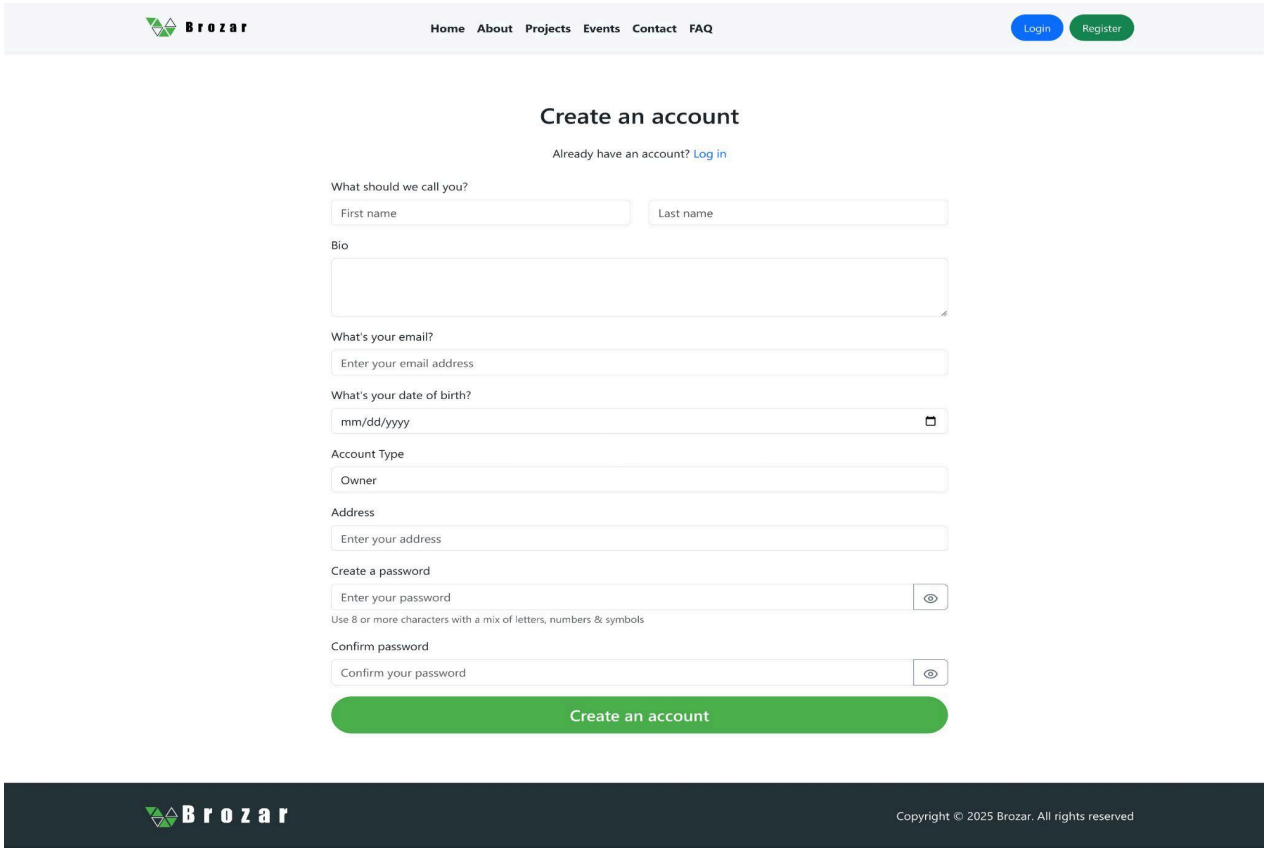


Figure 4.3.2 : Create Account

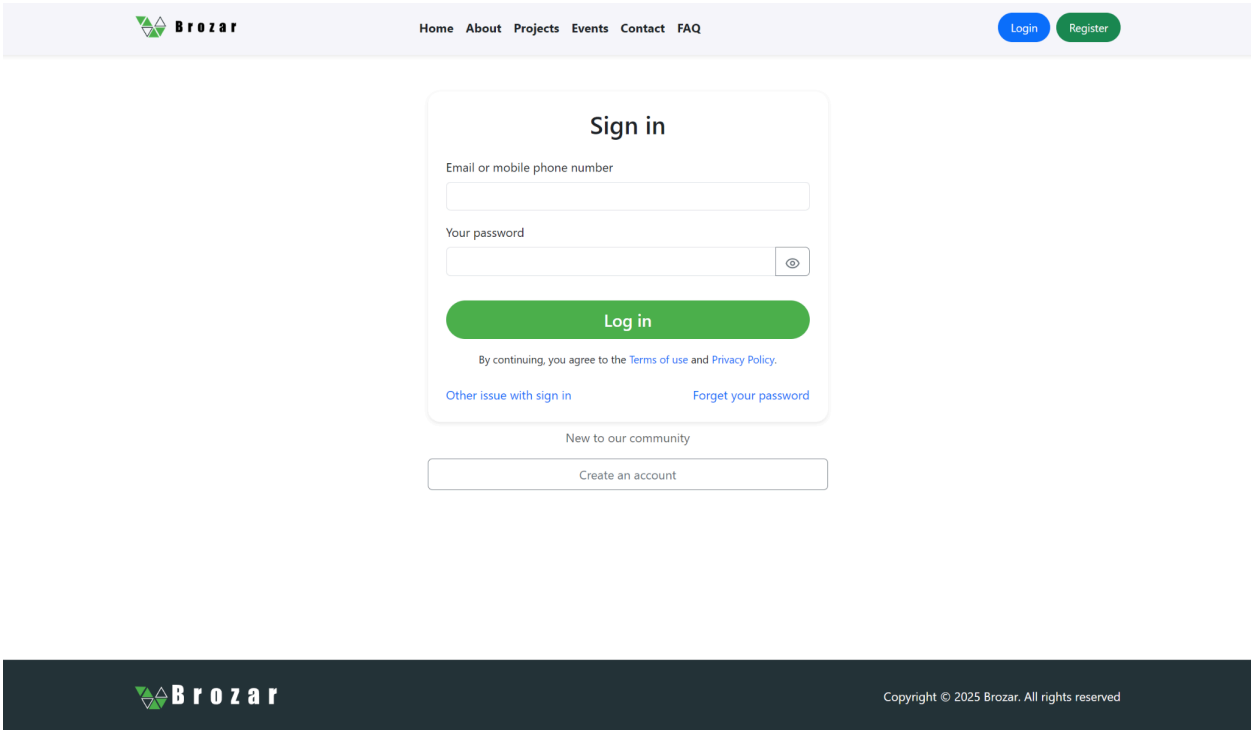


Figure 4.3.3 : Log in

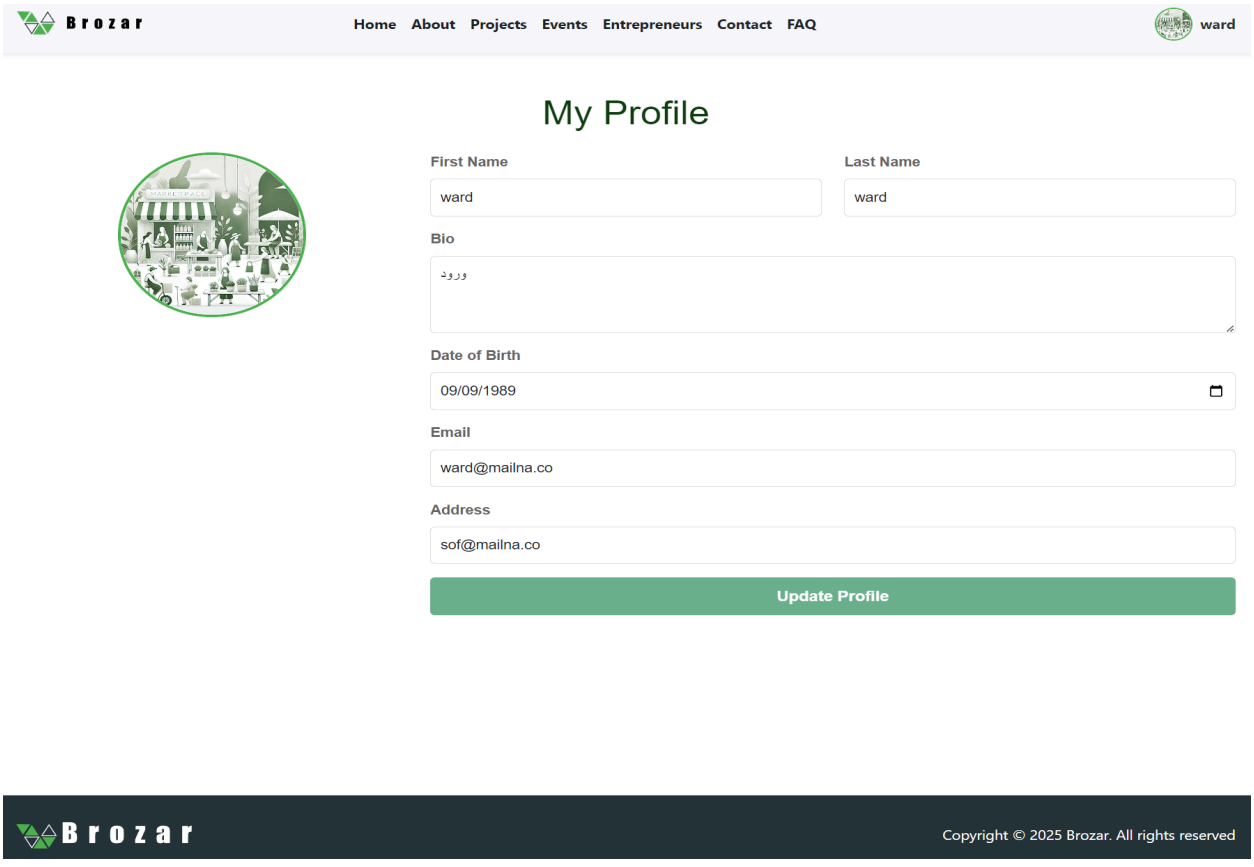


Figure 4.3.4 : my profile page

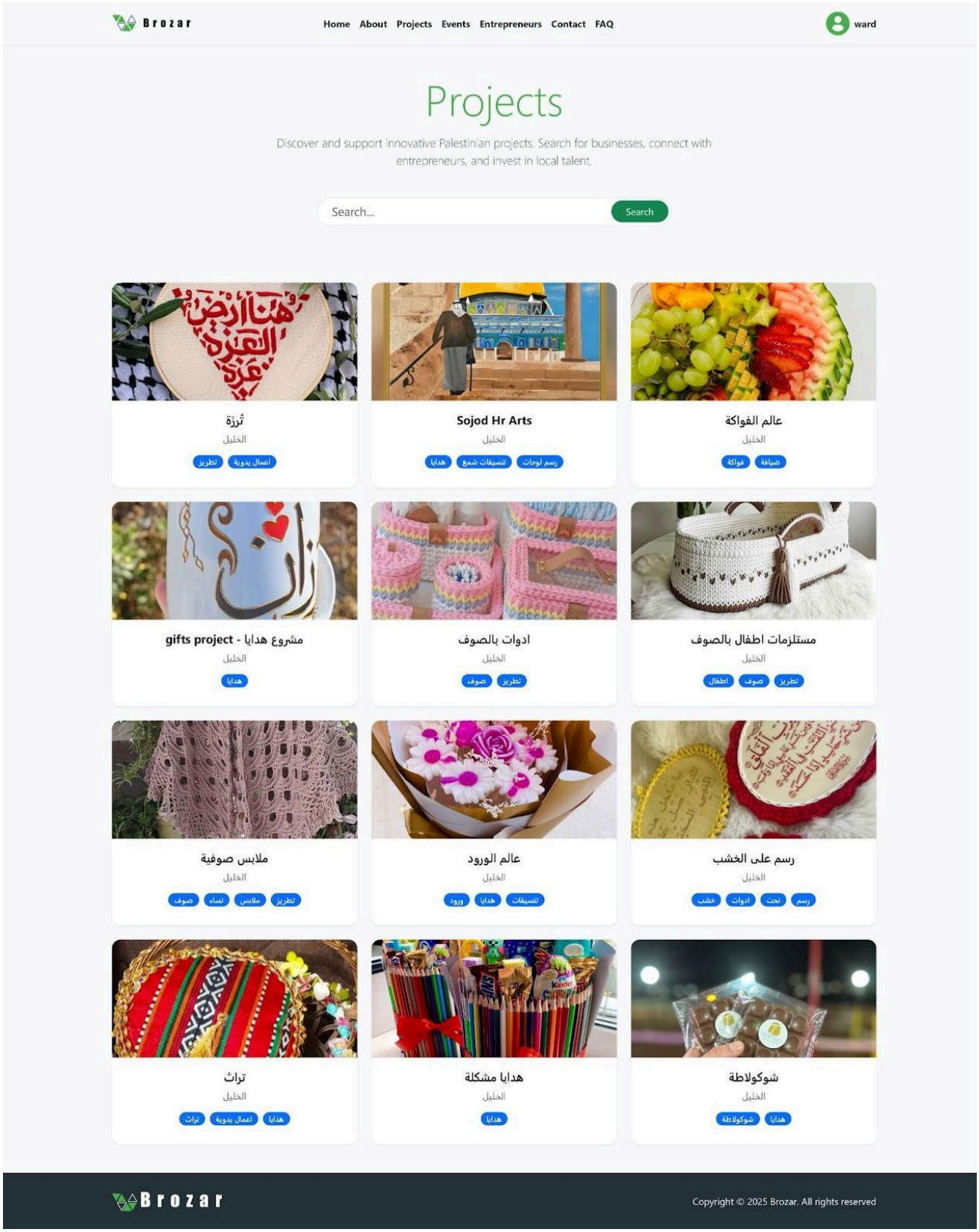


Figure 4.3.5 : Projects page

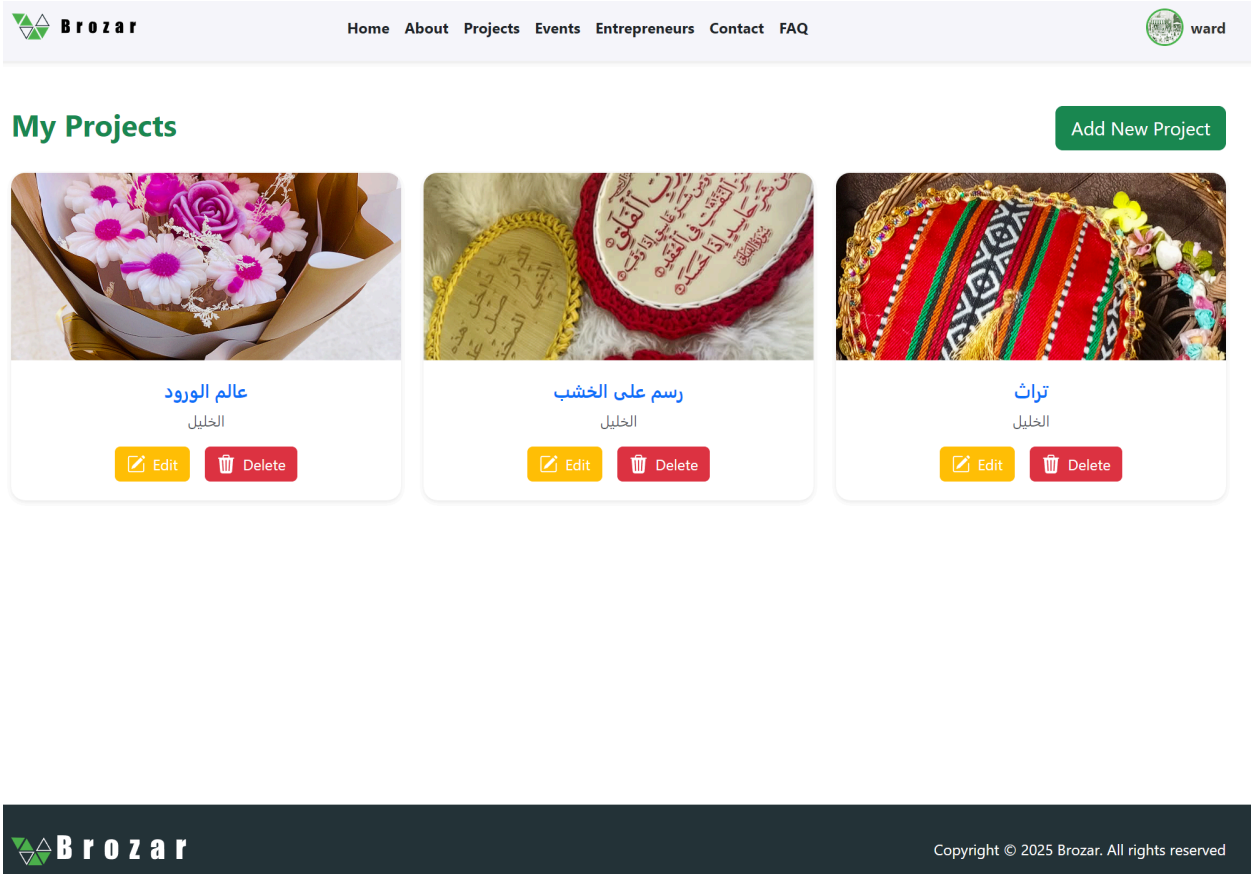


Figure 4.3.6 : my projects page

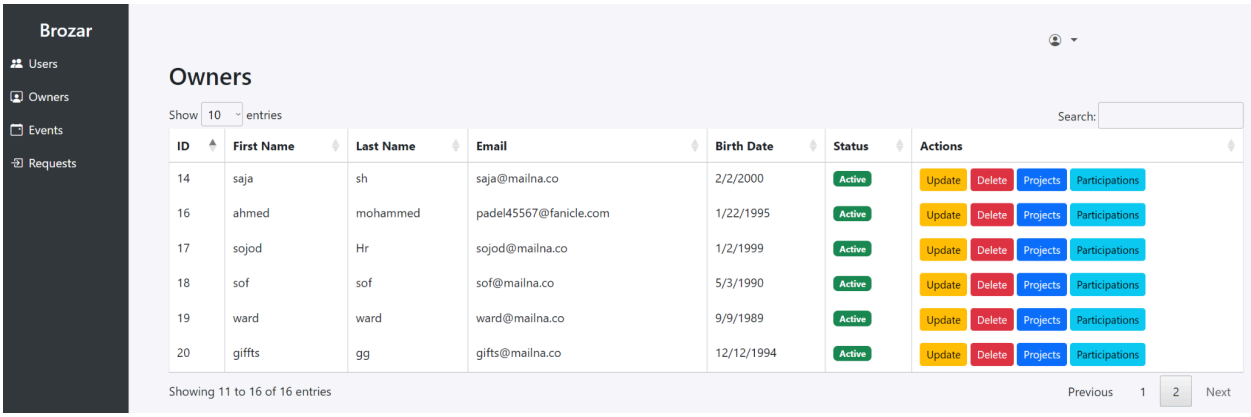


Figure 4.3.7 : admin dashboard page

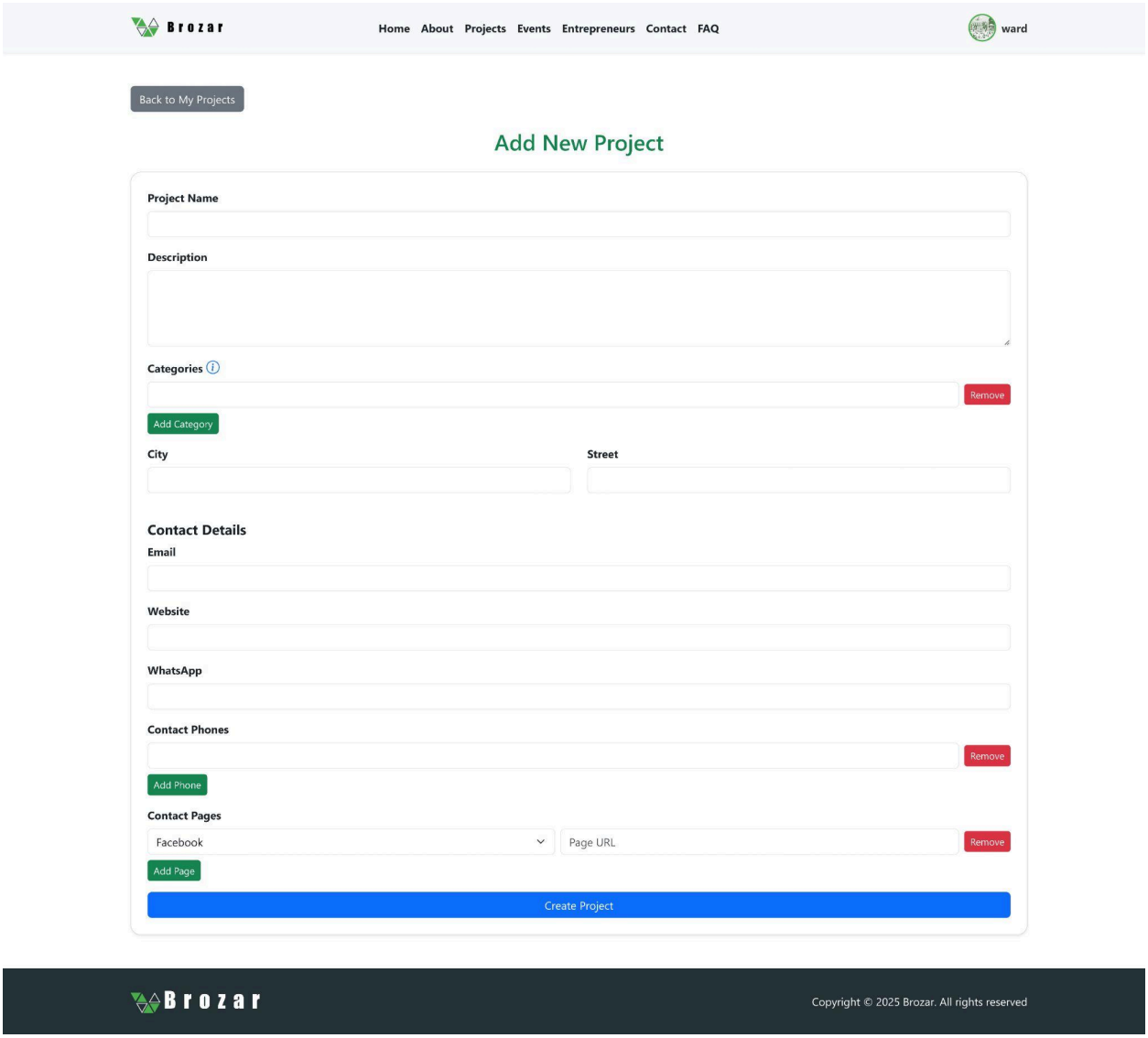


Figure 4.3.8 : Add new project page

Chapter 5 Testing

5.1 Overview

In the stage of testing the system, we make sure that the system works correctly without any problems, and we also make sure that the functional and non-functional requirements of the project are completed, and that the system works accurately and quickly in completing the tasks.The stage of checking the system comes after the design and implementation of the system. The method of checking the system will be explained in this chapter.

5.2 Validation

All information entered in all fields in the website is checked to ensure that the data entered by the user matches all conditions as follows:

- Customize the field to match the entry
- The process will not be executed if wrong data is entered.
- The operation is not performed if the fields are empty
- Ensure that there are actual users in the database.

5.3 Unit testing

```
it('should return 400 if passwords do not match', async () => {
  req.body = {
    password: 'password123',
    passwordVerify: 'password456',
  };
  await createAccount(req, res, next);
  expect(next).toHaveBeenCalledWith(new passwordNotMatch('Passwords do not match'));
});
```

Will test if the password match or not , in this case the password not match and return value was “Passwords do not match”

```
it('should handle errors and call next with the error', async () => {
  req.body = {
    password: 'password123',
    passwordVerify: 'password123',
    role: 'invalidRole',
    email: 'invalid@example.com',
    firstName: 'Invalid',
    lastName: 'User',
  };
  const mockError = new Error("Invalid role provided. Must be 'owner' or 'investor'.");
  authService.createAccount.mockRejectedValue(mockError);

  await createAccount(req, res, next);

  expect(authService.createAccount).toHaveBeenCalledWith(req.body);
  expect(next).toHaveBeenCalledWith(mockError);
});
```

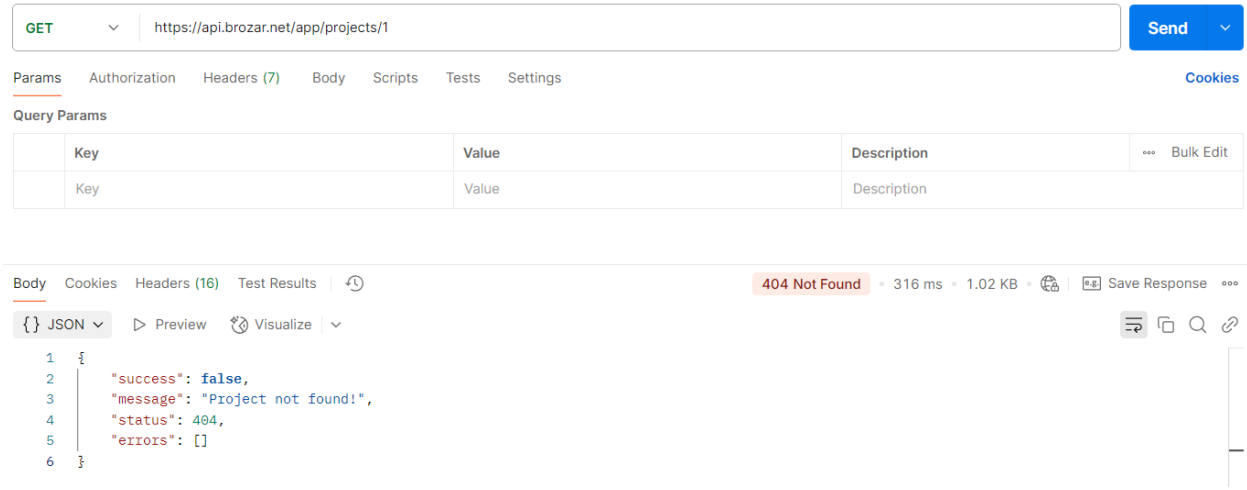
Will test if the create account will be correct or not , in this case the account created successfully and the return value was “req.body”

```
it('should create an account for owner role and return 200', async () => {
  req.body = {
    password: 'password123',
    passwordVerify: 'password123',
    role: 'owner',
    email: 'owner@example.com',
    firstName: 'John',
    lastName: 'Doe',
  };
  const mockOwner = { id: 1, userId: 1, ...req.body };
  authService.createAccount.mockResolvedValue(mockOwner);
  await createAccount(req, res, next);
  expect(authService.createAccount).toHaveBeenCalledWith(req.body);
  expect(res.statusCode).toBe(200);
  expect(res._getJSONData()).toEqual(mockOwner);
});
```

Will test if there are invalid data or not , in this case the role is invalid data and the return value was “Invalid role provided. Must be 'owner' or 'investor'.”

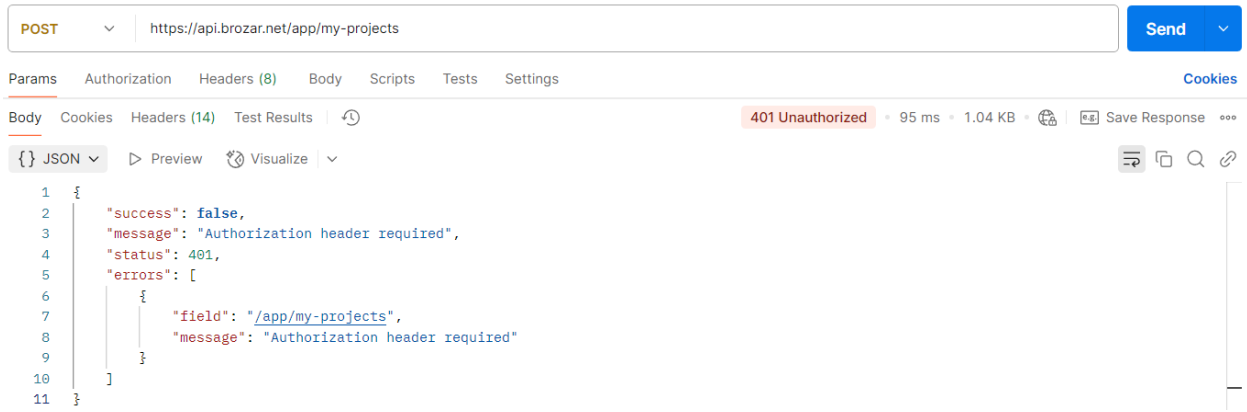
5.4 Endpoint testing

- Projects :
  - Search for a non-existent project



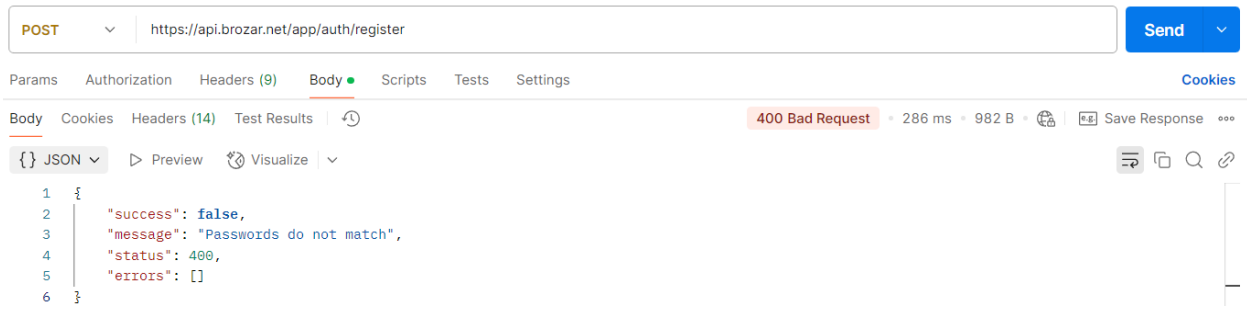
Expected output is : project not found! , 404 Not Found
Actual output is : project not found! , 404 Not Found

  - Create project without the owner loggnig in



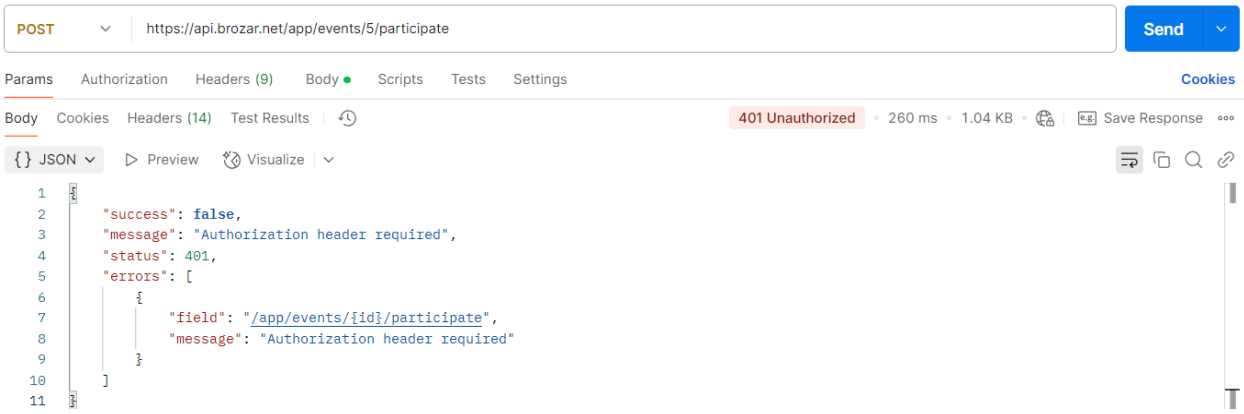
Expected output and status is : Authorization header required , 401 Unauthorized
Actula output and status is : Authorization header required , 401 Unauthorized

- Authentication
  - Register with password not match



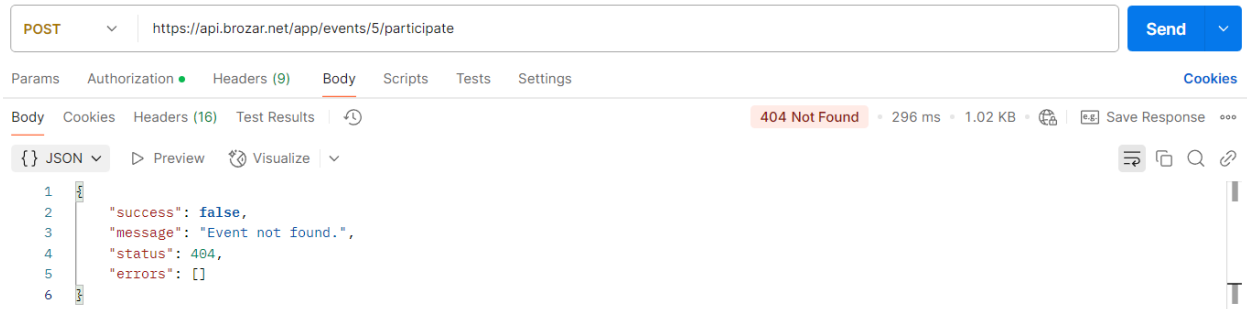
Expected output and status is : Passwords do not match , 400 Bad Request
Actula output and status is : Passwords do not match , 400 Bad Request
Send an Email to activate the account

- Events
  - Participate without the owner logging in



Expected output and status is : Authorization header required , 401 Unauthorized
Actula output and status is : Authorization header required , 401 Unauthorized

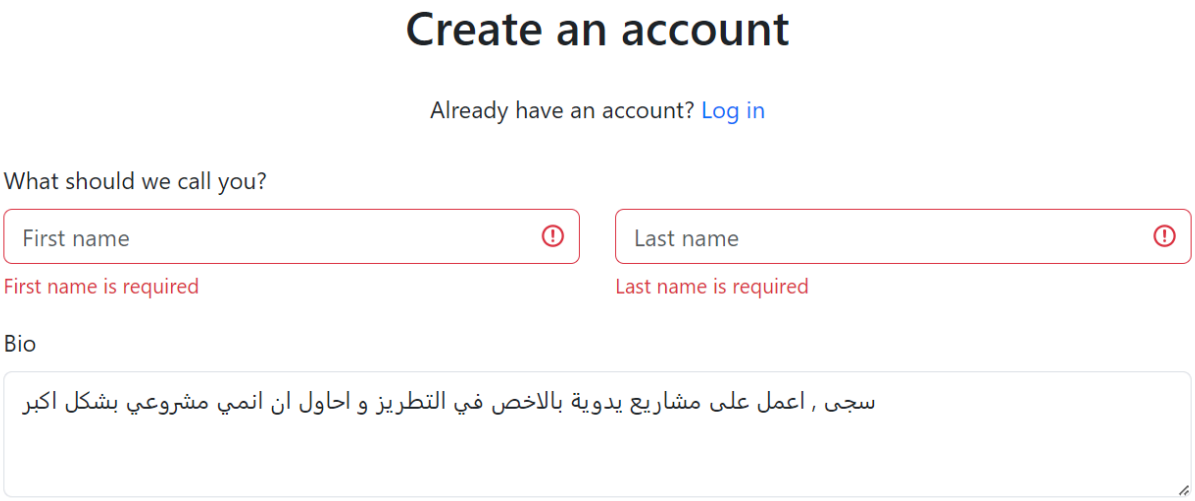
- Participate for a non-existent Event



Expected output and status is : Event not found , 404 Not Found
Actula output and status is : Event not found , 404 Not Found

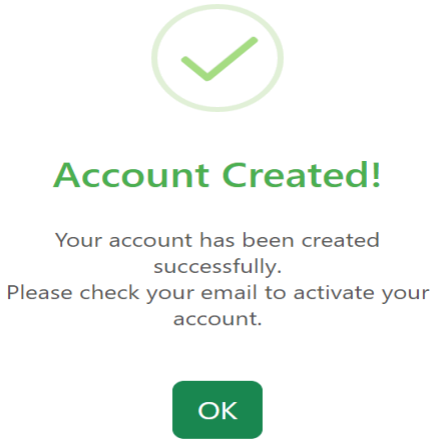
5.5 QA Manual testing

- Register without first name or last name



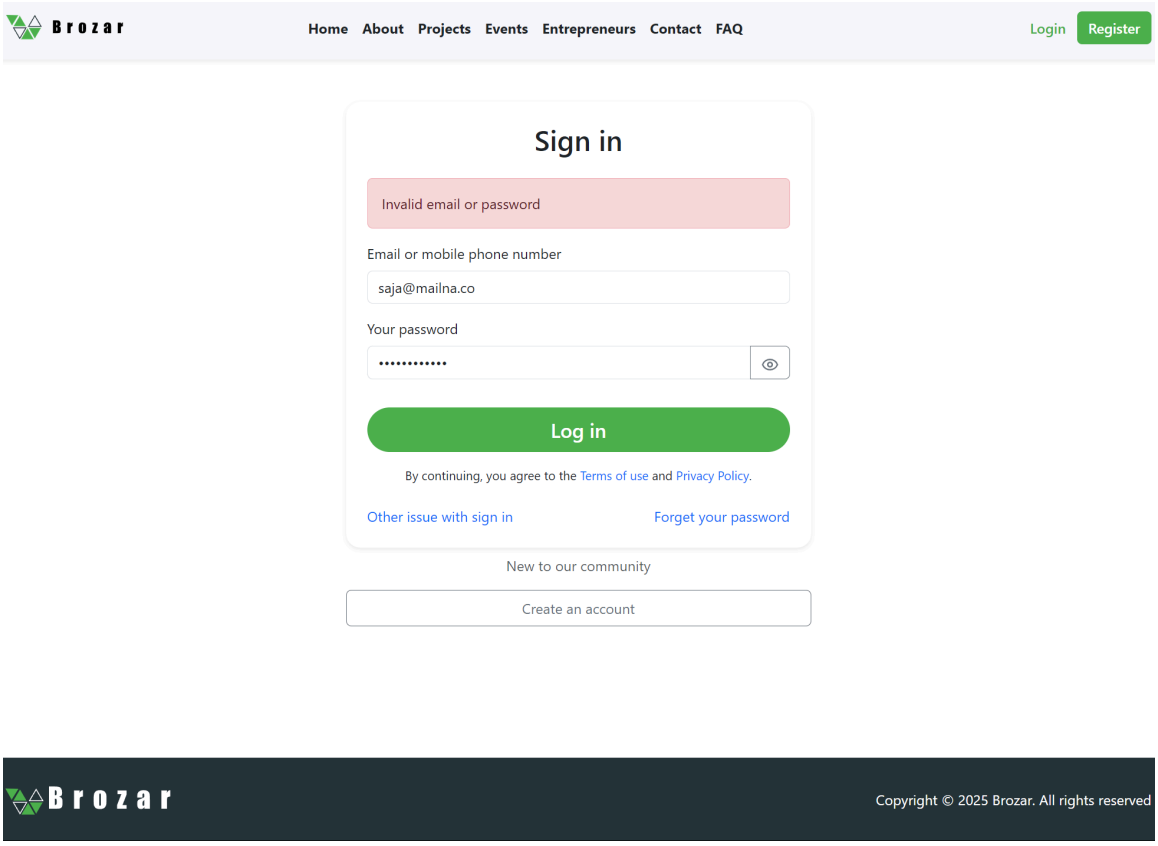
Expected : can not register first name is required, last name is required
Actual : can not register first name is required, last name is required

- Register



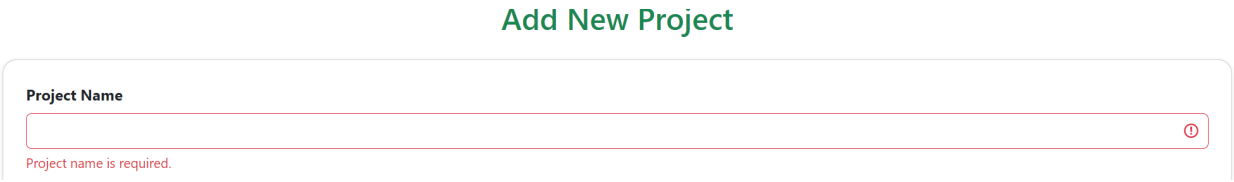
Expected : Account created
Actual : Account created

- Singh in with wrong email or password



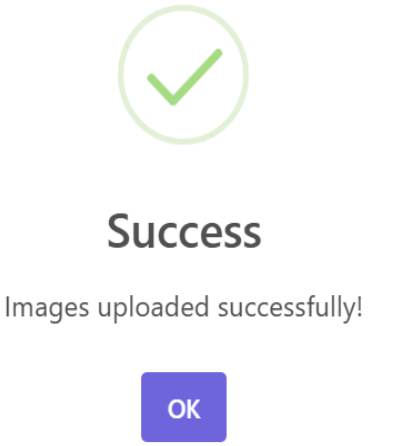
Expected : Invalid email or passwrod
Actual : Invalid email or password

- Create project without name



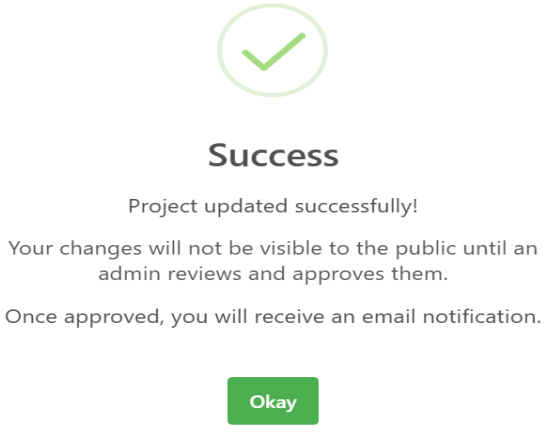
Expected : project name is required
Actual : project name is required

- Add Image



Expected : Success
Actual : Success

- Update Project



Expected : Success
Actual : Success , then pass to the admin

Conclusion and Future work

Conclusion

At the end of our project, we highlight the key aspects of Brozar, a platform designed to support Palestinian entrepreneurs. In this project, we covered the importance and motivation behind its creation, explained its goals and how to achieve them, and outlined the system requirements, both functional and non-functional. We built the project using modern technologies like Express.js for the backend, React.js for the frontend, and other excellent tools to achieve all the requirements we aimed for. Finally, we designed the system models, which demonstrate the structure of the platform, the relationship between users and its components, and how the system works.

Future work

In the future, we plan to add several improvements to the project, focusing on several key areas. First, we aim to improve the user interface, making it easier, more convenient, and more user-friendly, as user experience is the most important point in any project. Second, we plan to develop a mobile application. With the increasing reliance on smartphones, having a dedicated application has become a necessity, not just a mobile-friendly website. Third, we will integrate AI features to help project owners and investors predict important factors, such as product pricing and project growth potential, based on live data for each project. Finally, we will provide more advanced features for investors, making it easier for them to search for valuable projects to invest in, especially with the integration of AI into the platform.

References

[1] C. Subramanya, “Application architecture — Understanding MVC (Model-View-Controller),” *Medium*, Jul. 28, 2023. [Online]. Available: https://subramanya-c.medium.com/application-architecture-understanding-mvc-model-view-controller-fda0879839e1.

[2] S. Matina, “What is the 3-Tier Architecture?,” *Medium*, May 1, 2023. [Online]. Available: https://medium.com/@shrestha.matina.20/what-is-the-3-tier-architecture-4520522e0720.

[3] Node.js, “Index | Node.js v20.2.0 documentation,” Node.js, [Online]. Available: https://nodejs.org/docs/latest/api/.

[4] OpenJS Foundation, “Express - Node.js web application framework,” *Expressjs.com*, 2017. [Online]. Available: https://expressjs.com/.

[5] MySQL, “MySQL documentation,” *Mysql.com*, 2019. [Online]. Available: https://dev.mysql.com/doc/.

[6] Sequelize ORM, “Sequelize,” *Sequelize.org*, 2019. [Online]. Available: https://sequelize.org/.

[7] Auth0, “JWT.IO - JSON Web Tokens introduction,” *Jwt.io*, 2019. [Online]. Available: https://jwt.io/introduction/.

[8] npm, “bcrypt,” [Online]. Available: https://www.npmjs.com/package/bcrypt.

[9] Jest, “Getting started,” *Jestjs.io*, [Online]. Available: https://jestjs.io/docs/getting-started.

[10] React, “Getting started,” *Reactjs.org*, 2024. [Online]. Available: https://legacy.reactjs.org/docs/getting-started.html.

[11] Cloudflare, “Home,” *Cloudflare Docs*, [Online]. Available: https://developers.cloudflare.com/.

[12] Nginx, “Nginx documentation,” *Nginx.org*, [Online]. Available: https://nginx.org/en/docs/.

[13] Mailjet, “Mailjet Help Center,” *Mailjet.com*, 2025. [Online]. Available: https://www.mailjet.com/docs/.

[14] Swagger, “Swagger documentation,” *Swagger.io*, 2021. [Online]. Available: https://swagger.io/docs/.

[15] GitHub, “GitHub.com help documentation,” *Docs.github.com*, 2024. [Online]. Available: https://docs.github.com/en.

[16] Docker, “Docker documentation,” *Docker Documentation*, Oct. 31, 2019. [Online]. Available: https://docs.docker.com/.

[17] SonarSource, “SonarQube 10.7 | Documentation,” *Sonarsource.com*, 2024. [Online]. Available: https://docs.sonarsource.com/sonarqube-server/latest/.