

Report submitted to the
UNIVERSITY OF MOHAMED BOUDIAF - M'SILA, ALGERIA



FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

in partial fulfillment of the requirements for the degree of

Master in Computer Science

Specialization: SIGL

Presented by

Samir Douadi

Entitled

**Design and Development of an Online Platform to
Connect Algerian Citizens with Local Service
Providers(AlloService DZ)**

Under the supervision of

Abdelbasset Barkat

Jury Members

Prof. first Name Last Name	University of M'sila	President
Prof. first Name Last Name	University of M'sila	Reporter
Dr. first Name Last Name	University of M'sila	Examiner

June, 2026

الملخص

في سياق التحول الرقمي العالمي المتسارع، ومع تزايد اعتماده في الجزائر، يهدف هذا المشروع إلى إنشاء منصة رقمية تعمل كجسر يربط بين مزودي الخدمات والمواطنين. تعاني الفئة المستهدفة من صعوبة في إيجاد مقدمي خدمات موثوقين وسريعين، مما يجعل الحاجة إلى حل رقمي أمراً ضرورياً. توفر منصة AlloServiceDZ إمكانية البحث عن الخدمات، حجزها، تتبعها، والتواصل مباشرة مع مقدميها، بالإضافة إلى نظام تقييم يساهم في رفع مستوى الثقة والجودة. كما تتيح لمقدمي الخدمات عرض مهاراتهم وإدارة الحجوزات بشكل منظم. أظهرت النتائج أن المنصة تساهم في تسهيل الوصول إلى الخدمات وتقليل الوقت والجهد، مع تحسين الشفافية وجودة التعامل بين الطرفين.

الكلمات المفتاحية: منصة خدمات، مزودي خدمات، تطبيق ويب، الثقة الرقمية، AlloServiceDZ.

Abstract

In the context of the accelerating global digital transformation, and with its increasing adoption in Algeria, this project aims to create a digital platform that acts as a bridge between service providers and citizens. The target group faces difficulties in finding reliable and effective service providers, making a digital solution essential.

The AlloServiceDZ platform allows users to search for, book, track, and communicate directly with service providers, in addition to a rating system that contributes to increased trust and quality. It also enables service providers to showcase their skills and manage bookings in an organized manner.

Results have shown that the platform facilitates access to services, reduces time and effort, and improves transparency and the quality of interaction between the two parties.

Keywords: Service Platform, Service providers, Web application, Digital Trust, AlloServiceDZ.

Dedication

*“To those who supported and encouraged me throughout my academic journey,
To those who showered me with love and stood by me every step of the way,
I dedicate this humble work to my dear parents,
who spared no effort in supporting and sacrificing for me, and who were always
my source of strength and inspiration.
And to my dear family,
and to everyone who stood by me and supported me in achieving this accomplishment.
I also dedicate it to my esteemed teachers,
who illuminated my path with knowledge and wisdom
praying that the Almighty may grant you long life and bless you with goodness.”*

Douadi Samir

Acknowledgements

First and foremost, I express our profound gratitude to Allah for His mercy and guidance throughout this journey.

I am deeply indebted to my thesis supervisor **Dr. Abdelbasset Barkat** whose steadfast support and inspiration made this project a success. In a very special way, I thank him for his continuous guidance and encouragement throughout this challenging study.

Special thanks go to my friends and family who endured the hectic moments and supported me during the course of the research.

Finally, I thank my university for giving me the opportunity to complete this work and develop my academic and personal skills.

Table of Contents

List of Tables	vii
List of Figures	viii
General Introduction	1
1 Requirements Analysis	3
1.1 Project Overview	3
1.2 Existing Platforms	3
1.2.1 Thumbtack	3
1.2.2 TaskRabbit	4
1.2.3 Urban Company	4
1.2.4 Ouedkniss	4
1.2.5 Bricoram	4
1.3 Users Categories	4
1.3.1 Client	4
1.3.2 Service Provider	5
1.3.3 Visitor	5
1.3.4 Administrator	5
1.4 User Needs Analysis	5
1.4.1 Client Needs	5
1.4.2 Provider Needs	6
1.4.3 Visitor Needs	6
1.4.4 Administrator Needs	6
1.5 Booking Life Cycle	7
1.6 Functional Requirements	8
1.7 Non-Functional Requirements	9
1.8 System Objectives	10
1.9 System Constraints	10
1.10 Conclusion	10
2 System Design	11
2.1 System Architecture	11

2.1.1	Context	11
2.1.2	Containers	13
2.2	Back-end Architecture	14
2.2.1	API Routing Layer	15
2.2.2	Middleware Layer	19
2.2.3	Security Mechanisms	20
2.2.4	ORM Layer	20
2.3	Database Design	21
2.3.1	Main Entities Description	23
2.3.2	Relationships and Constraints	24
2.4	Client-Side Application	25
2.4.1	User Interface Modules	25
2.4.2	System Use Cases	26
2.4.3	Core Workflows and Activity Diagrams	29
2.4.4	Interaction Sequences	32
2.5	Admin Dashboard	36
2.5.1	Admin Interface Structure	36
2.5.2	Admin Use Cases Diagram	36
2.5.3	Admin Sequence Diagrams	37
2.6	Conclusion	40
3	System Implementation and Deployment	41
3.1	Development Environment and Technologies	41
3.1.1	Software Development Environment	41
3.1.2	Development Technology Stack	42
3.2	Project Structure	44
3.2.1	Frontend Structure	44
3.2.2	Backend Structure	45
3.3	Platform Presentation	45
3.3.1	Visitor Interfaces	46
3.3.2	Client Interfaces	52
3.3.3	Provider Interfaces	56
3.3.4	Admin Interfaces	59
3.4	System Deployment	63
3.4.1	Application Hosting and CI/CD Pipeline	63
3.4.2	Database Hosting	64
3.5	Conclusion	64

General Conclusion	65
References	66

List of Tables

2.1	Authentication Public API Endpoints	15
2.2	Provider Public API Endpoints	15
2.3	Service Public API Endpoints	15
2.4	Location Public API Endpoints	16
2.5	General Public API Endpoints	16
2.6	User Profile API Endpoints	16
2.7	Booking API Endpoints	16
2.8	Review API Endpoints	17
2.9	Notification API Endpoints	17
2.10	Report API Endpoints	17
2.11	Communication API Endpoints	17
2.12	Professional Profile API Endpoints	17
2.13	Gig Management API Endpoints	18
2.14	Media Upload API Endpoints	18
2.15	User Management API Endpoints	18
2.16	Report Moderation API Endpoints	18
2.17	Contact Management API Endpoints	19
2.18	Administrative Statistics API Endpoints	19
2.19	Authentication and Onboarding Pages	25
2.20	Discovery and Search Pages	26
2.21	Transaction and Dashboard Pages	26
2.22	Communication Pages	26
2.23	Administrative Dashboard Pages	36

List of Figures

1.1	Service Life Cycle of AlloServiceDZ Platform	8
2.1	System Context Diagram	12
2.2	System Container Diagram	13
2.3	Backend Components Diagram	14
2.4	Entity-Relationship Diagram	22
2.5	Visitor Use Cases Diagram	27
2.6	Client Use Cases Diagram	28
2.7	Provider Use Cases Diagram	29
2.8	Provider Discovery, Authentication, and Booking Activity Diagram	30
2.9	Service Request Approval, Execution, and Feedback Activity Diagram . .	31
2.10	Provider Registration and Service Offering Activity Diagram	32
2.11	User Registration and Email Verification Sequence Diagram	33
2.12	Service Discovery and Data Obfuscation Sequence Diagram	34
2.13	Booking Lifecycle Sequence Diagram	35
2.14	Admin Use Cases Diagram	37
2.15	Admin Login Sequence Diagram	38
2.16	users management Sequence Diagram	38
2.17	reports management Sequence Diagram	39
2.18	Support Message management Sequence Diagram	40
3.1	Home Page	46
3.2	Services Page	47
3.3	Providers Page	48
3.4	Provider Profile Page	49
3.5	Login Page	50
3.6	Register Page	51
3.7	Client Dashboard Page	52
3.8	Booking Request Page	53
3.9	Messages Page	54
3.10	Profile Settings Page	55
3.11	Provider Dashboard Page	56
3.12	Gig Manager Interface	57
3.13	Provider Profile Setup Page	58

3.14 Admin Dashboard Interface	59
3.15 User Management Interface	60
3.16 Reports Management Interface	61
3.17 Admin Contact Messages Interface	62

General Introduction

The world is undergoing a rapid transformation driven by technological and digital advancements, this transformation has become an integral part of daily life, with information technology in its various branches becoming an essential and indispensable element. With the increasing reliance on these technologies in Algeria, the importance of digital platforms and applications has become paramount, aiming to cover various sectors and keep pace with this development.

Among the problems solved by this transformation is the facilitation of the relationship between clients and service providers, and the growing demand for home services such as plumbing, construction, and painting, coupled with the need for reliable service providers, it has become essential to provide a digital platform that connects service providers and citizens in an efficient and organized manner.

Even with the proliferation of digital platforms offering these types of services, the challenge of finding efficient, reliable, and high-quality providers persist. To address this issue, this project proposes developing a digital platform that acts as a bridge between clients and providers. Clients can search for providers near their location, book and track services, and communicate with them directly with ease. Providers, on the other hand, can showcase their work and demonstrate their expertise through a rating and review system on their profiles, as well as manage incoming orders by accepting or rejecting bookings.

Despite the existence of numerous global and local projects similar to this type of system, such as Thumbtack, TaskRabbit, and Urban Company globally, and Ouedkniss and Bricoram locally in Algeria, they often lack user-friendliness or are not fully tailored to this type of service. This reveals a clear gap between the growing demand for local services and the available digital solutions. This is where the idea of developing an improved platform emerged, combining ease of use with efficient performance, with the aim of providing a better and more convenient user experience.

The proposed solution is AlloServiceDZ, an online marketplace for services that acts as an intermediary between clients and local service providers, specifically designed for the Algerian market. The platform aims to improve access to local services by providing a structured, efficient, and user-friendly digital environment.

For clients, the platform allows them to create an account, view service provider ratings and other users' reviews, and search for nearby service providers based on service type or location using map search. Users can also book services, track the status of their requests, and communicate directly with service providers. For service providers, the platform enables

them to create detailed profiles, showcase images of their past work, and manage incoming service requests by accepting or declining bookings. This helps service providers better organize their activities and increase their visibility and credibility.

The deployment version of the platform is publicly accessible through the following web address: <https://alloservicedz.netlify.app/>

Report structure

This report is structured into three chapters:

- **Chapter 1:** focuses on the project requirements, including the analysis of user needs, functional and non-functional requirements, and the definition of the system objectives for the platform.
- **Chapter 2:** covers the system design, describing the overall architecture of the platform along with its main technical specifications.
- **Chapter 3:** presents the implementation and deployment of the system, detailing the development process, technologies used, and the core functionalities implemented.

Finally, the report concludes with an evaluation of the achieved results and discusses possible improvements and future enhancements for the platform.

Chapter 1

Requirements Analysis

This chapter provides a detailed analysis of the system requirements for the proposed platform. It aims to identify and formalize both functional and non-functional requirements, which define the expected behavior and constraints of the system. This phase is critical as it establishes a solid foundation for the design and implementation stages.

The chapter covers the project overview, stakeholders identification, user needs analysis, booking life cycle, and a comprehensive specification of system requirements.

1.1 Project Overview

The proposed system, AlloServiceDZ [1], is an on-demand home service platform that connects clients with service providers in Algeria. It simplifies the process of discovering, selecting, and booking services.

The platform addresses common issues such as lack of trust, inefficiency in searching for services, and absence of centralized solutions by providing a structured and digital environment.

1.2 Existing Platforms

Several digital platforms have emerged to connect clients with service providers and facilitate service discovery and booking processes. These platforms vary in their business models, geographical coverage, and service specialization. The following examples represent some of the most recognized global and local solutions related to the AlloServiceDZ concept.

1.2.1 Thumbtack

Thumbtack is an American online marketplace that connects customers with local professionals and service providers across multiple categories such as home maintenance, tutoring, photography, and event services. The platform allows users to search for providers, compare profiles and reviews, and request service quotations directly through the system [2].

1.2.2 TaskRabbit

TaskRabbit is a digital service marketplace that enables users to hire freelancers and local workers for everyday tasks including furniture assembly, moving assistance, cleaning, and home repairs. The platform emphasizes fast task matching and location-based service delivery [3].

1.2.3 Urban Company

Urban Company is a large-scale service platform operating primarily in Asia and the Middle East. It connects customers with trained professionals in domains such as beauty services, home maintenance, appliance repair, and cleaning. The platform incorporates provider verification, service scheduling, and customer review mechanisms [4].

1.2.4 Ouedkniss

Ouedkniss is one of the most popular Algerian online classified platforms. Although primarily designed for product advertisements and e-commerce listings, it also includes service-related advertisements where users can publish and search for professional services across different categories [5].

1.2.5 Bricoram

Bricoram is an Algerian online service marketplace that connects clients with local professionals and service providers across various service categories. The platform allows users to publish service requests, receive offers from nearby professionals, compare profiles and reviews, and select suitable providers. It supports multiple service domains including plumbing, maintenance, construction, cleaning, and other professional services [6].

1.3 Users Categories

We categorize the users of the AlloServiceDZ platform into four main types, each with specific roles and interactions within the system:

1.3.1 Client

Clients are users who look for and request services. They can search for service providers, view provider profiles, send booking requests, and follow the status of their reservations. After receiving the service, they can also submit reviews and ratings based on their experience.

1.3.2 Service Provider

Service providers are users who offer services through the platform. They may be individuals or professionals working in various domains such as home repair, cleaning, and other service categories. Their primary role is to create and manage service offerings and respond to client booking requests.

1.3.3 Visitor

Visitors are users who access the platform without creating an account. They can explore available services and view general information about service providers in order to understand the platform and its functionalities before registration.

1.3.4 Administrator

Administrators are privileged users responsible for managing and supervising platform operations. They monitor user activities, manage reported content, moderate platform interactions, and ensure the proper functioning, security, and integrity of the system.

1.4 User Needs Analysis

This section identifies the primary needs and expectations of the different categories of platform users. Understanding these needs is essential for defining system functionalities and ensuring that the proposed platform delivers an effective and user-centered service experience.

1.4.1 Client Needs

The client represents the primary beneficiary of the AlloServiceDZ platform; therefore, the system must address their requirements efficiently and reliably. The main client needs include:

- Ability to easily find trusted service providers
- Quick and efficient booking mechanism
- Transparency in service details and availability
- Ability to manage and track bookings
- Ability to review and provide feedback on service providers

1.4.2 Provider Needs

Service providers rely on the platform to promote their professional activities and manage interactions with potential clients. Therefore, the platform should support their operational and communication requirements. These needs include:

- Platform to showcase services and professional skills
- Efficient booking management system
- Increased visibility and customer acquisition
- Ability to communicate effectively with clients

1.4.3 Visitor Needs

Visitors represent potential future users who access the platform without registration. Consequently, the platform should provide sufficient information and navigation capabilities to encourage engagement and account creation. Their main needs include:

- Ability to explore available services without registration
- Access to general information about the platform
- Ability to view general information about service providers

1.4.4 Administrator Needs

Administrators are responsible for overseeing the overall operation and integrity of the Al-loServiceDZ platform. Therefore, the system must provide them with adequate tools to monitor, manage, and moderate platform activities effectively. Their main needs include:

- Access to a comprehensive dashboard displaying platform statistics and activity
- Ability to manage and moderate registered user accounts
- Tools to review and resolve user-submitted reports and complaints
- Ability to suspend or permanently remove accounts violating platform policies
- Access to all contact messages submitted by platform visitors

1.5 Booking Life Cycle

The booking life cycle defines the set of states through which a service request evolves within the AlloServiceDZ platform, from its initial creation to its final closure. This life cycle provides a structured representation of the booking process, ensuring clear state transitions and improving system traceability as illustrated in Figure 1.1

The main states of the booking life cycle are described as follows:

- **Booking Creation:** The client submits a booking request to a specific service provider.
- **Pending State:** The request is sent to the selected service provider and awaits response.
- **Acceptance / Rejection:** The provider either accepts or rejects the request.
- **Confirmed Booking:** If accepted, the booking is confirmed and scheduled.
- **Service Execution:** The provider performs the requested service.
- **Completion:** The service is marked as completed.
- **Reviewing:** The client gives a review to service provider.
- **Cancellation (Optional):** The client and provider may cancel the booking before completion.

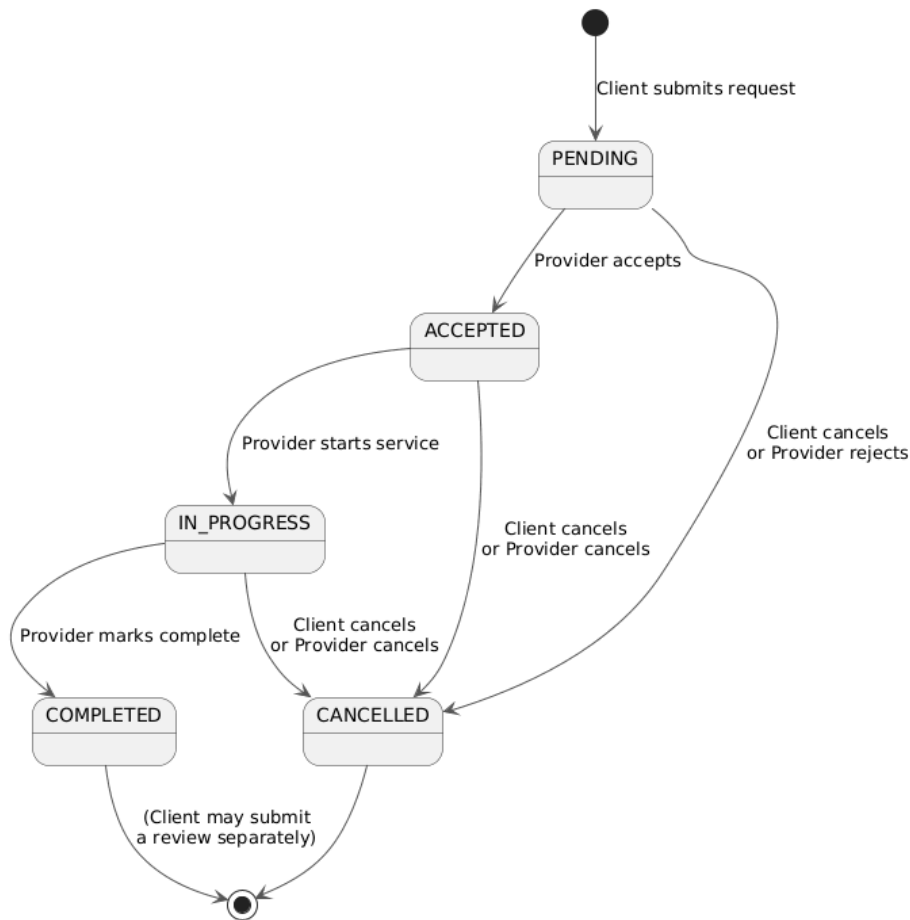


Figure 1.1: Service Life Cycle of AlloServiceDZ Platform

The definition of this life cycle ensures a consistent and well-structured management of service bookings, enabling efficient tracking, better coordination between clients and providers, and improved overall system reliability.

1.6 Functional Requirements

Functional requirements describe the system's behavior and operations.

- **User Registration:** The system shall allow clients and service providers to create accounts by providing required information such as name, email, password and location.
- **User Authentication:** The system shall enable users to securely log in and log out using their credentials.
- **Profile Management:** Users shall be able to view, update, and manage their personal information and profiles.

- **Service Browsing:** Visitors and clients shall be able to browse available services by keyword.
- **Service Search and Filtering:** The system shall provide search and filtering mechanisms to help users find relevant service providers.
- **Booking Creation:** Clients shall be able to create service booking requests by selecting a provider and specifying details.
- **Booking Management:** Users shall be able to view, track, and manage their bookings.
- **Booking Response:** Service providers shall be able to accept or reject booking requests.
- **Booking Cancellation:** Both clients and service providers shall be able to cancel a booking at any stage before completion (PENDING, ACCEPTED, or IN_PROGRESS), provided that the booking has not yet been marked as (COMPLETED).
- **Booking Status Tracking:** The system shall provide real-time updates on booking status (pending, confirmed, InProgress, completed, canceled).

1.7 Non-Functional Requirements

Non-functional requirements define system quality attributes and constraints.

- **Performance:** The system should respond to user actions within an acceptable time (e.g., less than 2 seconds for standard operations), even under moderate load.
- **Security:** The system must ensure data confidentiality and integrity through secure authentication, password encryption, and protection against common attacks.
- **Usability:** The user interface should be intuitive, easy to navigate, and accessible to users with basic digital skills.
- **Availability:** The system should be available 24/7 with minimal downtime and reliable uptime.
- **Scalability:** The platform should be able to handle an increasing number of users and service providers without performance degradation.
- **Maintainability:** The system should be designed in a modular way to facilitate updates, debugging, and future enhancements.
- **Compatibility:** The system should be accessible Across various devices (desktop, mobile) And modern browsers.

1.8 System Objectives

The objectives of the AlloServiceDZ platform define the main goals that guided the design and development of the system. These objectives focus on improving service accessibility, facilitating interactions between users, and providing a reliable digital environment for service exchange. The system aims to:

- Provide a reliable platform connecting clients and service providers
- Improve service accessibility and efficiency
- Reduce time and effort required to find services
- Enhance trust and transparency between users

1.9 System Constraints

Like any software project, the development of AlloServiceDZ was influenced by a number of technical and organizational limitations. These constraints affected technology selection, implementation decisions, and overall project planning. The main constraints include:

- Limited project development time
- Dependence on selected technologies
- Resource limitations

1.10 Conclusion

This chapter presented a detailed requirements analysis of the AlloServiceDZ platform. It covered the main users Categories involved in the system, the different user needs, and how each type of user interacts with the platform. It also described the booking life cycle, which explains the different stages a booking goes through from creation to completion. In addition, the chapter defined both functional requirements, which describe what the system should do, and non-functional requirements, which describe how the system should perform in terms of quality, security, and usability.

These specifications provide a clear understanding of the system and will serve as a solid foundation for the system design phase presented in the next chapter.

Chapter 2

System Design

Following the requirements analysis presented in the previous chapter, this chapter focuses on the design of the proposed platform, AlloServiceDZ. The system design phase aims to transform the identified functional and non-functional requirements into a structured software architecture that ensures scalability, maintainability, security, and extensibility.

To achieve a clear and standardized architectural representation, the C4 model was adopted as the main approach for documenting the system architecture. This model provides different levels of abstraction that help in understanding complex systems and improving communication between stakeholders and developers [7] [8]. In addition, Unified Modeling Language UML diagrams were used to describe both the structural and behavioral aspects of the system, as UML remains a widely accepted standard for modeling software systems [9] [10].

This chapter presents the overall system architecture, including system context, containers, database design, and the interaction between system components and actors.

2.1 System Architecture

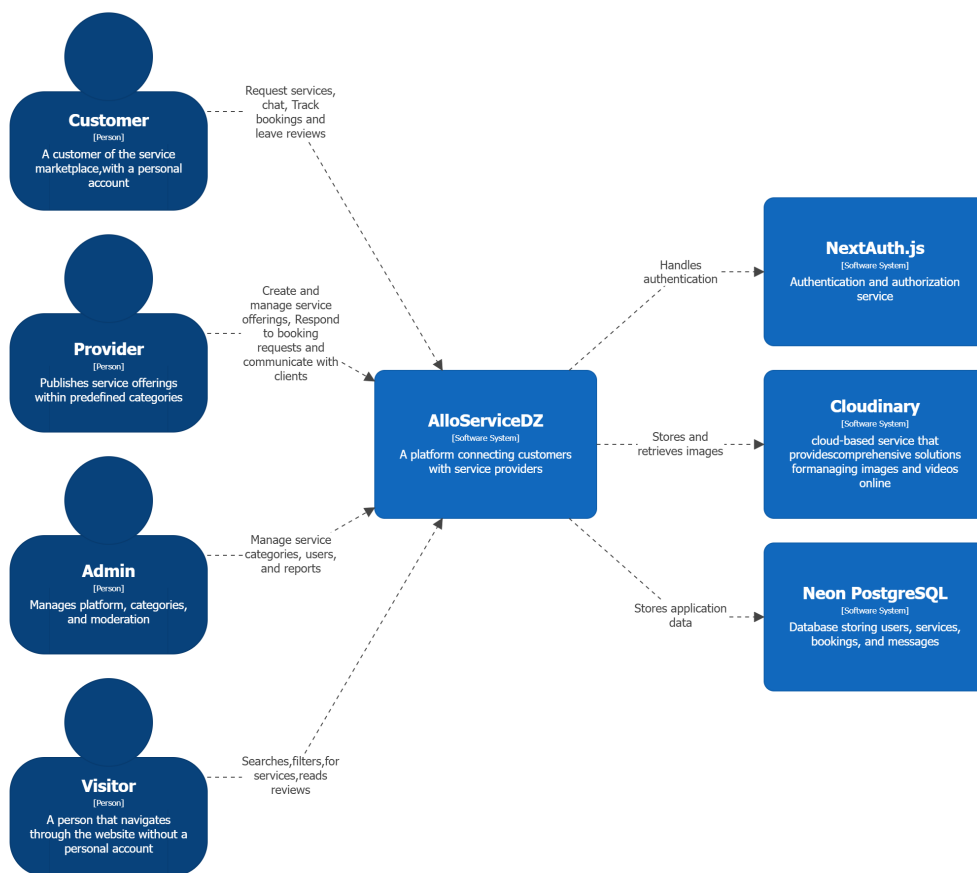
In this section, we describe the architecture of the proposed system using two complementary views: Context and Containers. These views illustrate how the system interacts with external actors and how its internal components are organized to deliver the required functionalities.

2.1.1 Context

Figure 2.1 show the System Context of the AlloServiceDZ platform, showing the main actors and external services interacting with the system.

The system includes four user roles: Visitors who browse services without authentication, Customers who request and manage services, Providers who create and manage their offerings, and an Admin responsible for system supervision and configuration.

The platform also integrates external services, including NextAuth.js for authentication, Cloudinary for image storage, and Neon PostgreSQL for data persistence.



System Context View: AlloServiceDZ

Figure 2.1: System Context Diagram

2.1.2 Containers

Figure 2.2 shows the Container Diagram of the AlloServiceDZ platform. It presents the main parts of the system and how they interact with each other to provide the required functionalities.

The diagram includes the user and admin web applications, the frontend interfaces, the core server that handles business logic and API requests, and the database used to store application data. It also shows the external services integrated into the platform, such as NextAuth.js for authentication and Cloudinary for image storage and management.

This diagram helps provide a clearer understanding of the internal organization of the system and the communication between its different containers.

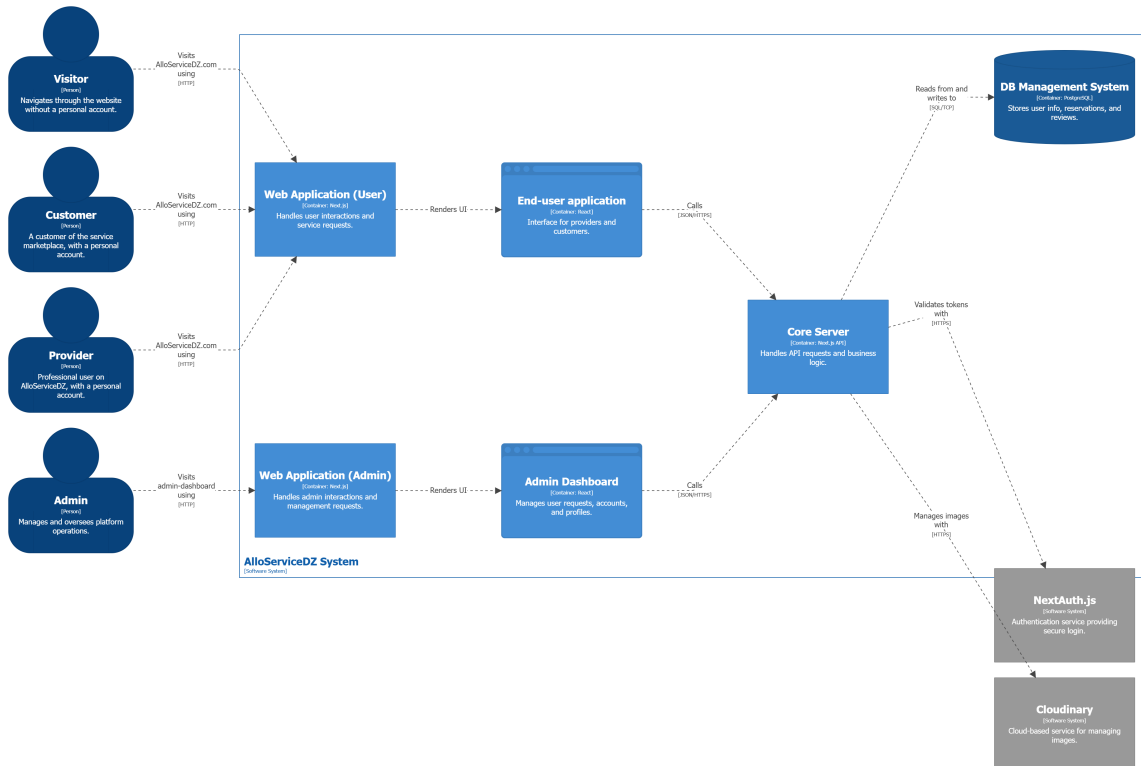


Figure 2.2: System Container Diagram

2.2 Back-end Architecture

The back-end architecture represents the server-side part of the system responsible for processing requests, executing business logic, and managing communication with the database and external services.

Figure 2.3 presents the internal structure of the AlloServiceDZ back-end, highlighting a modular and security-focused design. Client requests are systematically processed through a Middleware layer and secured API Routes, which integrate NextAuth.js for robust authentication and a custom Crypto Utility for data obfuscation. Additionally, the system leverages Cloudinary for optimized image delivery and Prisma ORM for type-safe communication with the Neon PostgreSQL database. This layered approach significantly enhances the platform's scalability, maintainability, and overall data integrity across all transactional services.

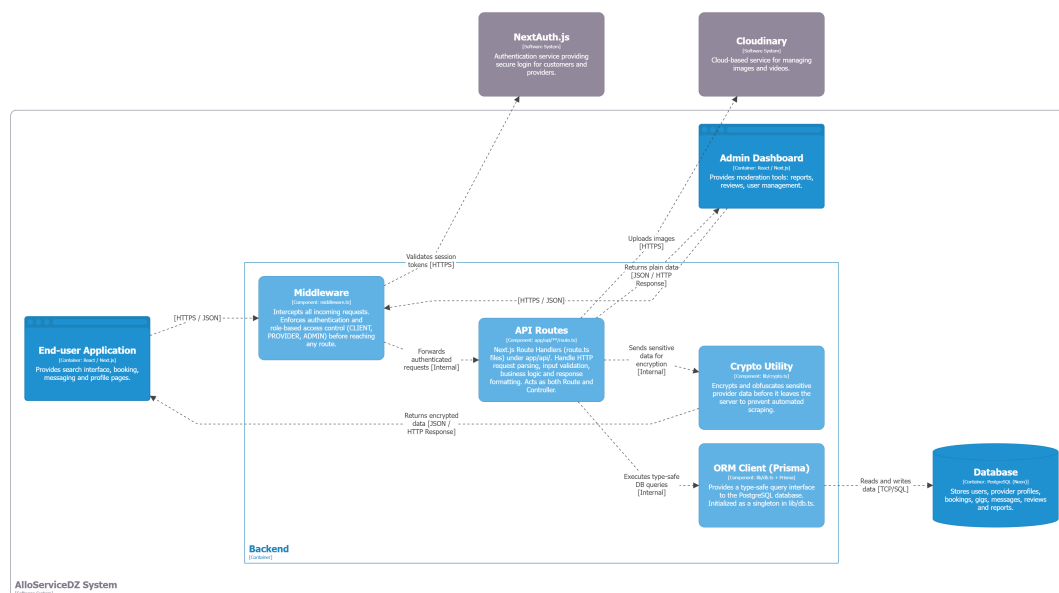


Figure 2.3: Backend Components Diagram

2.2.1 API Routing Layer

The API Routing Layer in AlloServiceDZ is built upon RESTful principles, providing a structured and scalable interface for communication between the frontend applications and the backend services. Leveraging Next.js Route Handlers, this layer manages request parsing, authentication enforcement through middleware, and data protection via a custom obfuscation utility. It ensures that all system resources ranging from provider profiles to booking operations are accessible through standardized HTTP methods.

In the following, the different API endpoints are categorized according to their access level into public, authenticated user, provider-specific, and administrative endpoints.

Public Endpoints: These endpoints are accessible to anyone without authentication.

- **Authentication Route:** /api/auth

Method	Endpoint	Result
POST	/register	Register a new account
POST	/signin	Authenticate a user account
GET	/verify-email	Verify user email address

Table 2.1: Authentication Public API Endpoints

- **Provider Route:** /api/providers

Method	Endpoint	Result
GET	/	Get the list of providers (Encrypted Data)
GET	/:id	Get provider details by ID (Encrypted Data)

Table 2.2: Provider Public API Endpoints

- **Service Route:** /api/services

Method	Endpoint	Result
GET	/	Get the list of available service categories

Table 2.3: Service Public API Endpoints

- **Location Route:** /api/locations

Method	Endpoint	Result
GET	/wilayas	Get the list of wilayas
GET	/dairas	Get the list of dairas
GET	/communes	Get the list of communes

Table 2.4: Location Public API Endpoints

- **General Route**

Method	Endpoint	Result
POST	/api/contact	Submit a contact message
GET	/api/stats	Get public platform statistics

Table 2.5: General Public API Endpoints

Authenticated User Endpoints: These endpoints are accessible only to clients and providers.

- **User Profile Route:** /api/user

Method	Endpoint	Result
GET	/profile	Get current user's profile data
PATCH	/profile	Update profile information
GET	/heartbeat	Update user's last seen status

Table 2.6: User Profile API Endpoints

- **Interaction Route:** /api/bookings

Method	Endpoint	Result
GET	/	Get user bookings
POST	/	Create a new booking request
PATCH	/:id	Update booking status or archive it

Table 2.7: Booking API Endpoints

- **Review Route:** /api/reviews

Method	Endpoint	Result
POST	/	Submit a review for a provider

Table 2.8: Review API Endpoints

- **Notification Route:** /api/notifications

Method	Endpoint	Result
GET	/	Get unread notifications

Table 2.9: Notification API Endpoints

- **Report Route:** /api/reports

Method	Endpoint	Result
POST	/	Submit a report against content or users

Table 2.10: Report API Endpoints

- **Communication Route:** /api/messages

Method	Endpoint	Result
GET	/	List active conversations
GET	/:id	Fetch chat history for a specific booking
POST	/	Send a new message

Table 2.11: Communication API Endpoints

Provider-Specific Endpoints: These endpoints are accessible only to providers.

- **Professional Profile Route:** /api/providers/me

Method	Endpoint	Result
GET	/	Fetch detailed professional profile
PATCH	/	Update bio, price range, and availability

Table 2.12: Professional Profile API Endpoints

- **Gig Management Route:** /api/gigs

Method	Endpoint	Result
GET	/	List all gigs for the current provider
POST	/	Create a new service offering
PATCH	/:id	Update an existing gig
DELETE	/:id	Archive or delete a gig

Table 2.13: Gig Management API Endpoints

- **Media Upload Route:** /api/upload

Method	Endpoint	Result
POST	/	Upload images to Cloudinary
DELETE	/:public_id	Delete image from Cloudinary

Table 2.14: Media Upload API Endpoints

Administrative Endpoints: These endpoints are accessible only to the platform administrators.

- **User Management Route:** /api/admin/users

Method	Endpoint	Result
GET	/	List all users and providers in the system
PATCH	/	Suspend or reactivate user accounts
DELETE	/:id	Permanently delete a user account

Table 2.15: User Management API Endpoints

- **Report Moderation Route:** /api/admin/reports

Method	Endpoint	Result
GET	/	View all moderation reports
PATCH	/	Update report status
DELETE	/:id	Delete reported content and close report

Table 2.16: Report Moderation API Endpoints

- **Contact Management Route:** `/api/admin/contact-messages`

Method	Endpoint	Result
GET	/	View all contact messages
DELETE	/:id	Delete a specific contact message

Table 2.17: Contact Management API Endpoints

- **Administrative Statistics Route:** `/api/admin/stats`

Method	Endpoint	Result
GET	/	View administrative dashboard statistics

Table 2.18: Administrative Statistics API Endpoints

2.2.2 Middleware Layer

In the context of Next.js, middleware consists of functions that intercept requests and responses during the HTTP request-response cycle. Executed before a request reaches the main route handlers, middleware components can access request objects, apply routing logic, modify headers, or terminate requests when necessary. This approach provides a centralized mechanism for enforcing platform-wide security and access control policies.

The middleware mechanisms implemented in AlloServiceDZ include:

- **Route Protection Middleware:** Implemented through `middleware.ts`, this component intercepts incoming requests targeting protected routes such as `/dashboard`, `/messages`, and `/admin`. It verifies the existence of a valid authentication state and redirects unauthenticated users to the login page before the request reaches the server-side logic.
- **Session Authentication Middleware:** Integrated within the API route handlers through `Auth.js`, this mechanism processes secure HTTP-only cookies, validates JSON Web Tokens (JWTs), and extracts user identity and role information to enforce Role-Based Access Control (RBAC).
- **Request Validation Middleware:** Embedded within the API route handlers, it uses schema validation libraries such as `zod` to parse incoming JSON payloads. This layer ensures that required data structures are correctly provided through type-safe validation before any database interaction is performed.

2.2.3 Security Mechanisms

To ensure the confidentiality, integrity, and security of the platform, several protection mechanisms are integrated into the backend architecture.

- **Password Hashing:** User credentials are secured using bcryptjs. Passwords are hashed with salted cryptographic functions before being stored in the database, preventing plaintext password exposure.
- **Data Obfuscation Utility:** A lightweight XOR-based utility is used to obfuscate sensitive provider-related data during API transmission. Although not intended as strong cryptographic encryption, this mechanism helps reduce automated data scraping and unauthorized data extraction.
- **Access Control Mechanisms:** Protected API routes enforce Role-Based Access Control RBAC through Auth.js session validation. This ensures that only authenticated users with authorized roles CLIENT, PROVIDER, or ADMIN can access restricted operations.
- **Input Validation:** Schema-based validation using libraries such as zod is applied to incoming request payloads and parameters. This mechanism helps prevent malformed data and improves the reliability and consistency of backend operations.

2.2.4 ORM Layer

An Object-Relational Mapping ORM layer serves as an intermediary between the backend application and the underlying relational database. It provides an abstraction layer that enables developers to interact with database entities using programmatic data models instead of writing raw database-specific SQL queries.

In the architecture of our system, the ORM layer is utilized for several key reasons:

- **Abstraction:** It decouples the application's business logic from the specific syntax and implementation details of the database engine.
- **Type-Safe Database Operations:** By generating structured database access models from a centralized schema, the ORM enables type-safe interactions between the application and the database, reducing the likelihood of runtime inconsistencies and data-related errors.

- **Reducing Raw SQL Complexity:** The ORM minimizes the need for manually written SQL queries. Complex operations such as multi-table joins, nested writes, and advanced filtering are abstracted into cleaner and more maintainable programmatic methods.

2.3 Database Design

The platform relies on a relational database management system to ensure reliable storage, data consistency, and efficient data management across the system. In this section, we present the Entity Relationship Diagram ERD of the database schema used in AlloServiceDZ, along with the main entities and relationships of the platform.

Figure 2.4 presents the Entity-Relationship Diagram ERD, showing the main database entities and their relationships. The USER entity is the core component of the schema, supporting role-based access for clients, providers, and administrators. Providers are associated with the PROVIDER_PROFILE and GIG entities, while the BOOKING entity connects clients, providers, and service offerings. The schema also includes communication, review, reporting, and geographic entities to support the different functionalities of the platform while maintaining referential integrity.

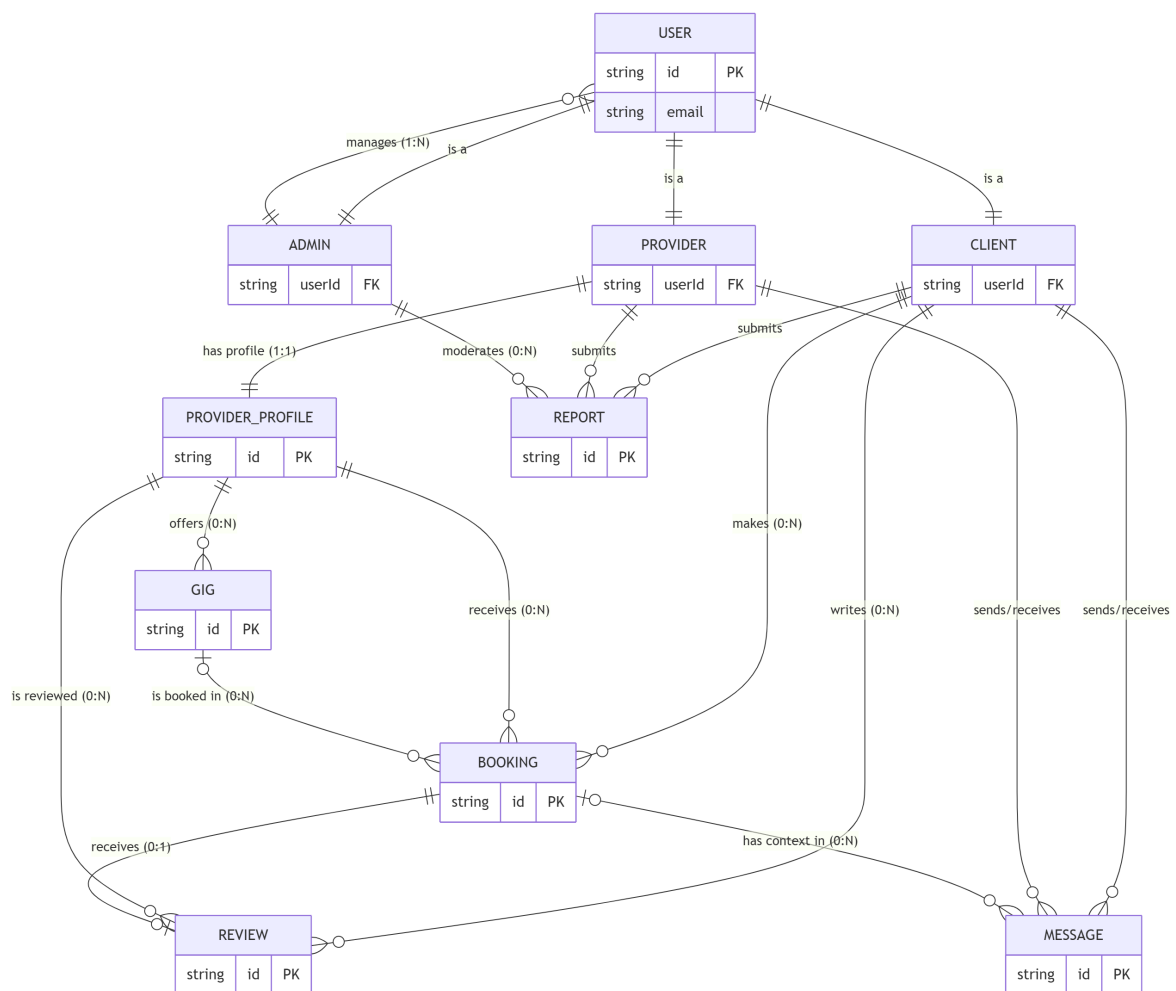


Figure 2.4: Entity-Relationship Diagram

2.3.1 Main Entities Description

To better understand the flow of data within the system, the following are the most critical entities in the relational schema:

- **USER:** The central authentication and identity entity acting as the root for almost all relations.
 - `id` (PK): Unique identifier for the user account.
 - `name`, `email`, `phone`: Basic contact and personal identification details.
 - `hashedPassword`: Cryptographically secured string for safe authentication.
 - `role`: Defines system access level (CLIENT, PROVIDER, or ADMIN).
 - `wilaya`, `commune`, `latitude`, `longitude`: Geographic data used for location-based filtering.
 - `lastSeen`: Timestamp updated periodically for real-time presence tracking.
 - `isSuspended`: Boolean flag indicating if the account has been restricted by admin.
- **PROVIDER_PROFILE:** A specialized extension of the USER entity (one-to-one). It houses professional attributes essential for discovery.
 - `id` (PK): Unique profile identifier.
 - `userId` (FK): Links back to the base USER account.
 - `bio`, `serviceTypes`: Professional description and a list of categorized skills.
 - `priceRange`, `radius`: Service pricing tier and maximum travel distance for work.
 - `rating`, `reviewCount`: Aggregated performance metrics for platform reputation.
 - `isAvailable`: Toggles the provider's visibility in public search results.
 - `images`: An array of URLs showcasing the provider's past work (portfolio).
- **BOOKING:** The primary transactional entity logically linking a client with a professional for a specific task.
 - `id` (PK): Unique transaction reference.
 - `clientId`, `providerId`, `gigId` (FKs): Relational links to the involved parties and the specific service requested.

- **status**: State machine field tracking lifecycle (PENDING, ACCEPTED, COMPLETED, CANCELLED).
- **scheduledAt, address**: The exact agreed-upon time and physical location for the service.
- **description**: The client's detailed explanation of the requested work.
- **REVIEW**: A feedback mechanism tied to a completed booking to ensure accountability.
 - **id (PK)**: Unique review identifier.
 - **bookingId, clientId, providerId (FKs)**: Traces the feedback back to the exact transaction and participants.
 - **rating**: A numeric score ranging from 1 to 5.
 - **comment**: Detailed textual feedback evaluating the provider's service.
- **MESSAGE**: The entity responsible for managing communication between users and maintaining booking-related conversations.
 - **id (PK)**: Unique message identifier.
 - **senderId, receiverId, bookingId (FKs)**: Defines the participants and the conversational context.
 - **content**: The actual text payload of the message.
 - **isRead**: Boolean flag used to support read receipts.
 - **archivedBySender, deletedByReceiver**: Independent state flags allowing users to manage their inbox visibility without affecting the other participant.

2.3.2 Relationships and Constraints

The integrity of the relational model is enforced through strict cardinalities and constraints managed by the ORM layer. This ensures that no orphaned records or data anomalies exist within the system.

- **One-to-One Relationships**: A strict 1:1 mapping exists between a **USER** and a **PROVIDER_PROFILE**. A user can only register as a provider once, ensuring that the **userId** acts as a unique foreign key identifier for the profile.
- **One-to-Many / Many-to-One**: This is the most common cardinality in the platform. For example, a single **USER (Client)** can initiate multiple **BOOKINGS** over time (One-to-Many). Similarly, multiple **BOOKINGS** can be associated with the same provider account (Many-to-One).

- **Foreign Keys:** Relational constraints are applied universally. Entities like `BOOKING` hold foreign keys (`clientId`, `providerId`) that explicitly require the existence of valid parent records in the `USER` and `PROVIDER_PROFILE` tables.
- **Cascade Deletion:** To maintain referential integrity and comply with data privacy standards, cascade deletion is implemented. If a `USER` account is deleted, the database automatically removes their associated `PROVIDER_PROFILE`, `BOOKINGS`, `REVIEWS`, and `MESSAGES`, preventing orphaned data accumulation.
- **Uniqueness Constraints (@unique):** Data duplication is prevented at the schema level. For instance, the `REVIEW` entity applies a uniqueness constraint on the [`clientId`, `providerId`] combination, ensuring that a client can only review a specific provider once, regardless of how many times they have booked them. Additionally, fields like user email are enforced as universally unique.

2.4 Client-Side Application

The client-side application serves as the primary gateway for all front-facing interactions, allowing visitors, clients, and providers to seamlessly engage with the AlloServiceDZ platform. In this section, we detail the modular architecture of the user interface, the core behavioral use cases, and the logical workflows that govern the user experience for each specific role.

2.4.1 User Interface Modules

The client-side interface is organized into four main functional modules. Each module groups a set of related pages/routes designed to provide a structured user experience.

- **Authentication and Onboarding Module**

This module handles user authentication and initial profile setup operations.

Route	Description
/login	Secure user authentication and sign-in page
/register	New account creation form
/profile	Core user profile details and settings management

Table 2.19: Authentication and Onboarding Pages

- **Discovery and Search Module**

These public-facing pages allow visitors and authenticated users to explore the platform without requiring authentication.

Route	Description
/	Main home/landing page
/services	Main catalog page listing available service categories
/providers	Provider listing, search, and filtering page
/providers/[id]	Detailed profile page of a specific provider
/about	Platform information and presentation page
/contact	Customer support and contact form page

Table 2.20: Discovery and Search Pages

- **Transaction and Dashboard Module**

This protected module serves as the main management interface for users to manage their platform activities.

Route	Description
/dashboard	Client dashboard used to track booking requests and statuses
/provider/dashboard	Provider dashboard used to manage services, booking requests, and job completion statistics
/book/[providerId]	Interactive booking page for creating a service request

Table 2.21: Transaction and Dashboard Pages

- **Communication Module**

This module supports messaging and interaction between users.

Route	Description
/messages	Unified near-real-time chat interface using smart HTTP polling with presence indicators

Table 2.22: Communication Pages

2.4.2 System Use Cases

The behavior of the client-side application is represented using UML Use Case diagrams. In this part, we present the main functionalities available for visitors, clients, and providers within the platform.

Visitor Use Cases: Figure 2.5 represents the interactions of any unauthenticated user (Visitor) within the platform, including browsing services, searching providers, viewing provider profiles, and accessing authentication features.

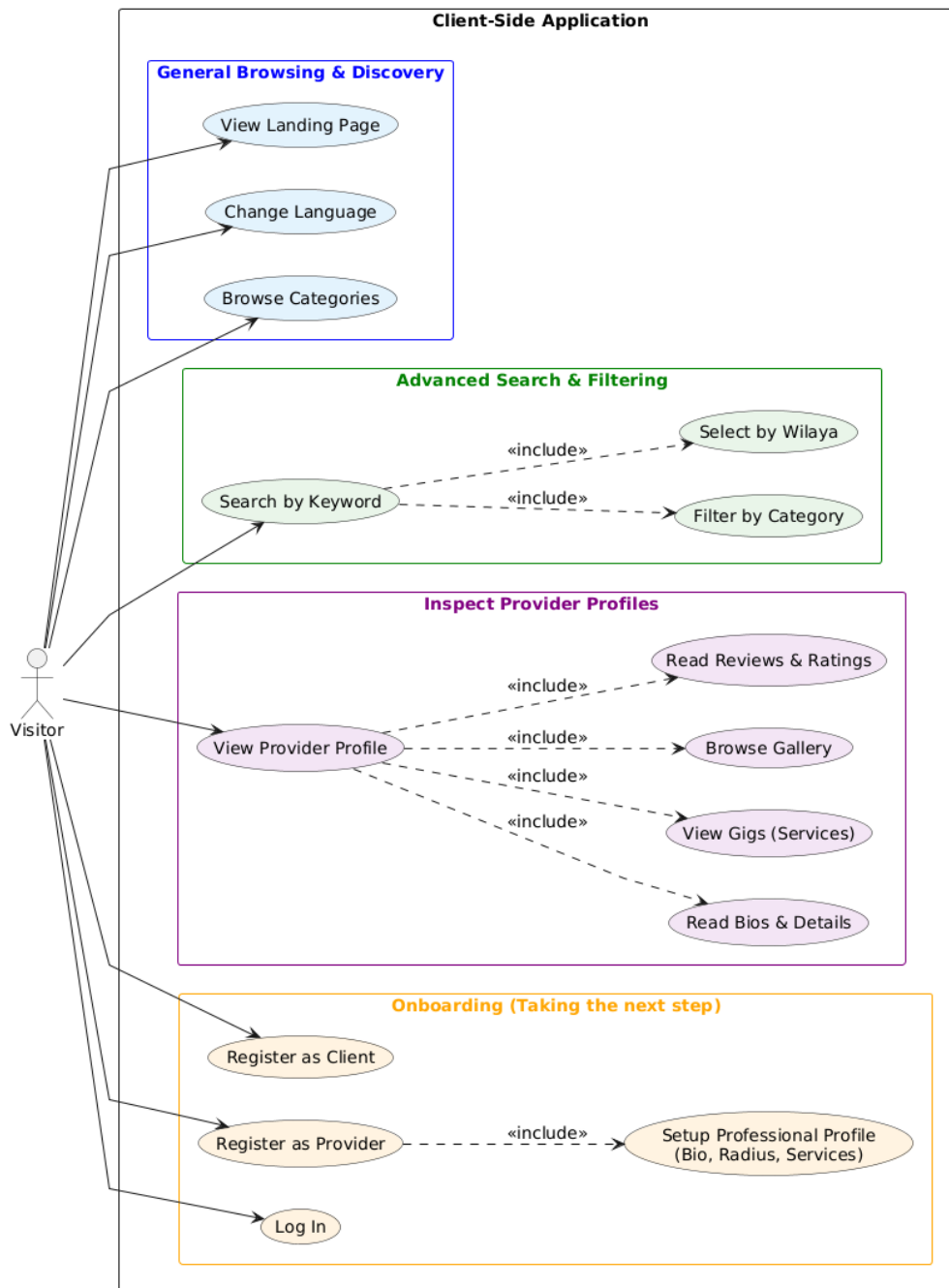


Figure 2.5: Visitor Use Cases Diagram

Client Use Cases: Figure 2.6 represents the interactions of authenticated client within the platform, including booking services, managing booking requests, communicating with providers, and submitting reviews or reports.

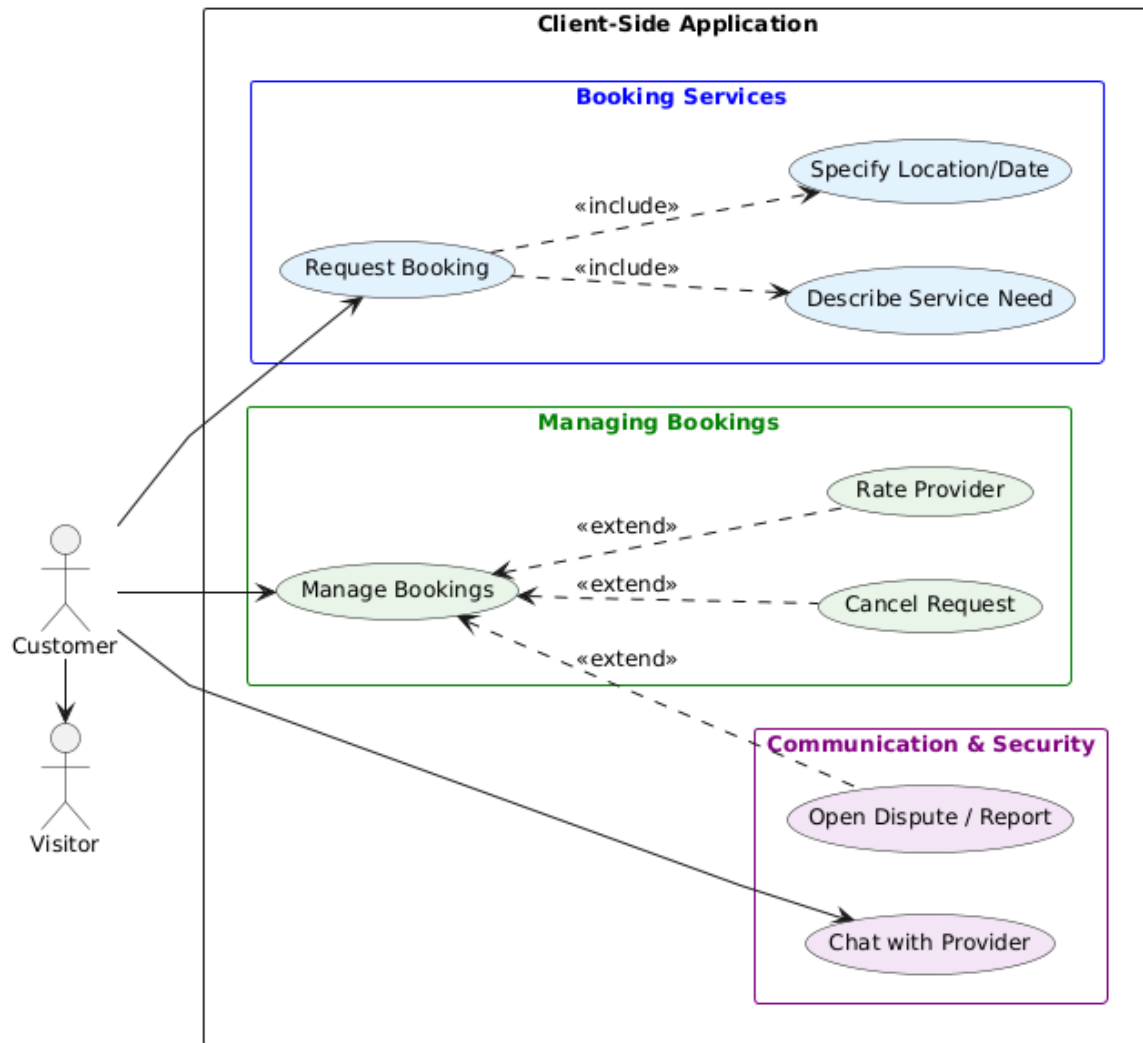


Figure 2.6: Client Use Cases Diagram

Provider Use Cases: Figure 2.7 represents the interactions of registered service providers within the platform, including profile management, booking handling, workflow tracking, and service management operations.

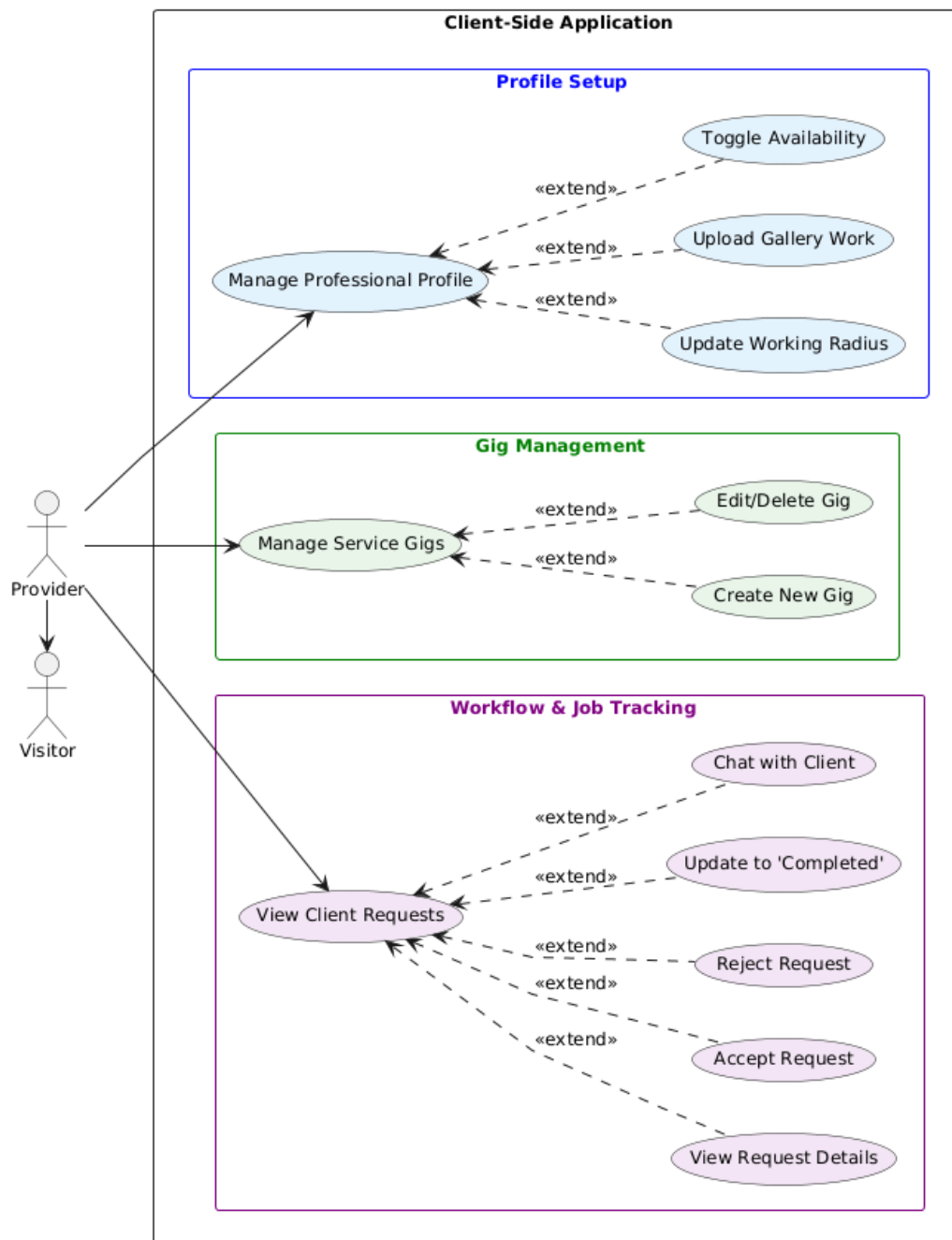


Figure 2.7: Provider Use Cases Diagram

2.4.3 Core Workflows and Activity Diagrams

To represent the dynamic behavior of the platform, this section presents the core workflows using UML Activity Diagrams. These diagrams illustrate the sequence of actions, decision flows, and interactions involved in the main functionalities of the system.

Provider Discovery, Authentication, and Booking Workflow: Figure 2.8 represents the workflow of provider discovery, user authentication, and booking request submission within the platform.

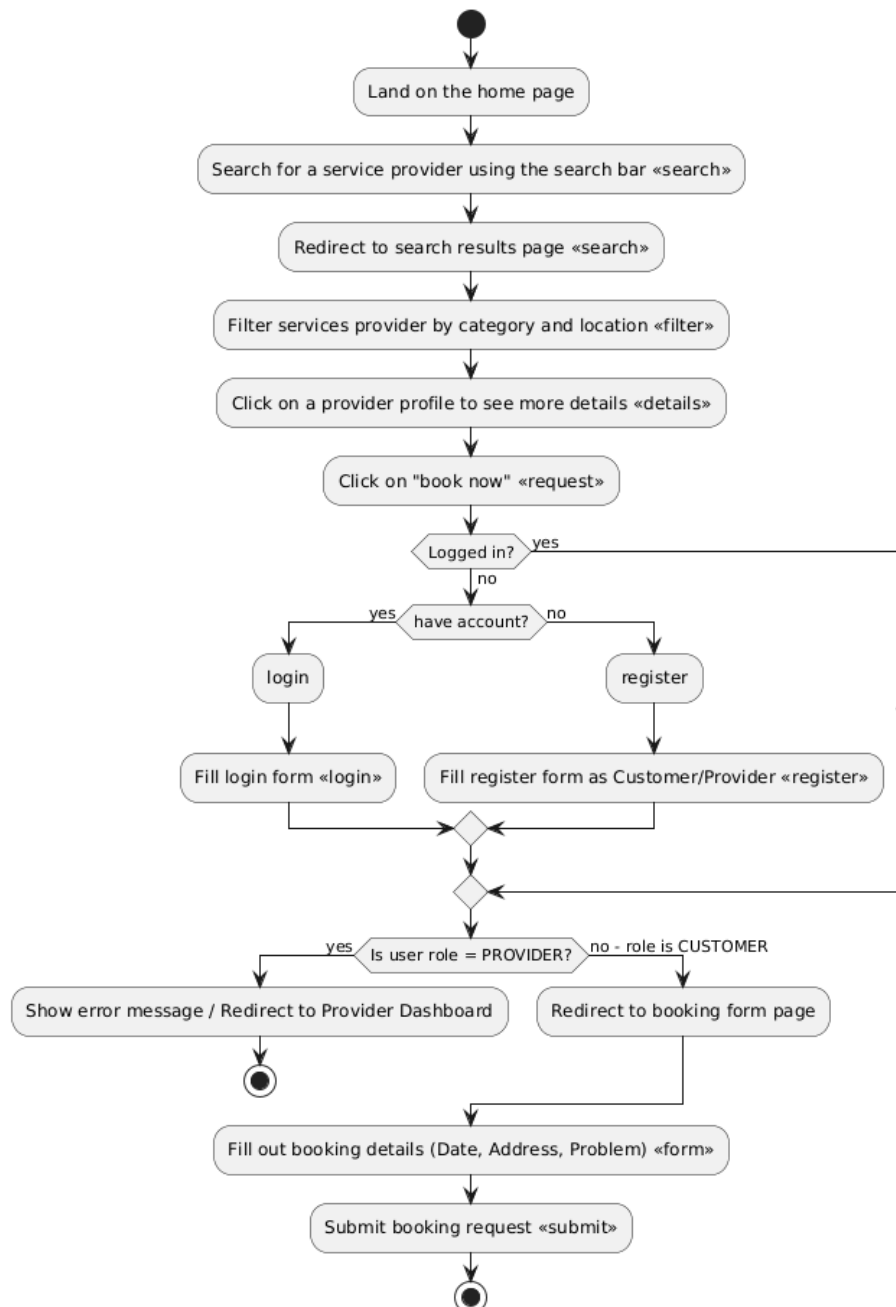


Figure 2.8: Provider Discovery, Authentication, and Booking Activity Diagram

Service Request Approval, Execution, and Feedback Lifecycle: Figure 2.9 represents the service request approval, service execution, and client feedback stages of the workflow.

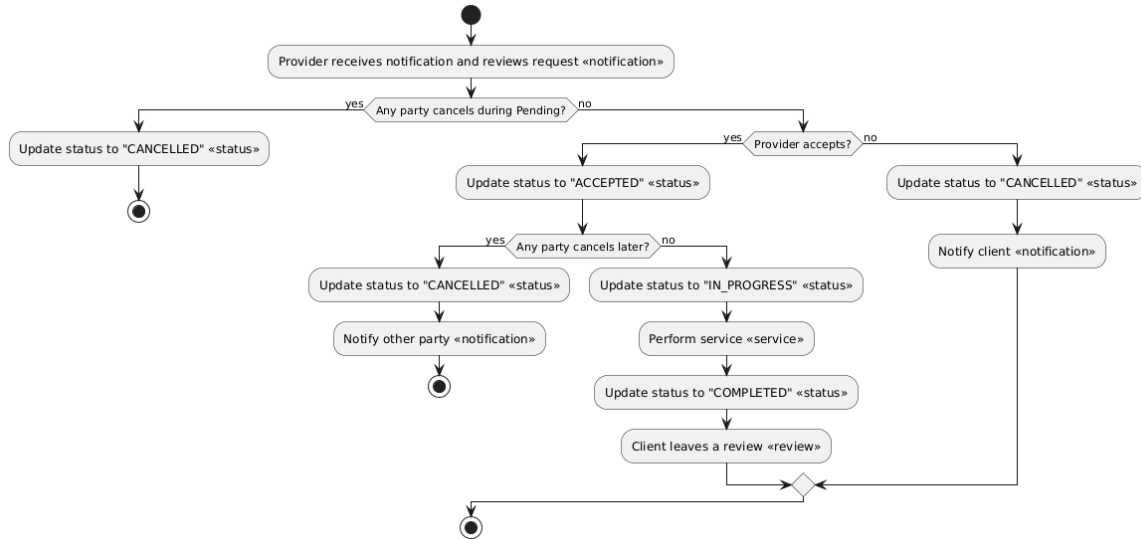


Figure 2.9: Service Request Approval, Execution, and Feedback Activity Diagram

Provider Registration and Service Offering: Figure 2.10 represents the registration and service offering creation workflow for service providers.

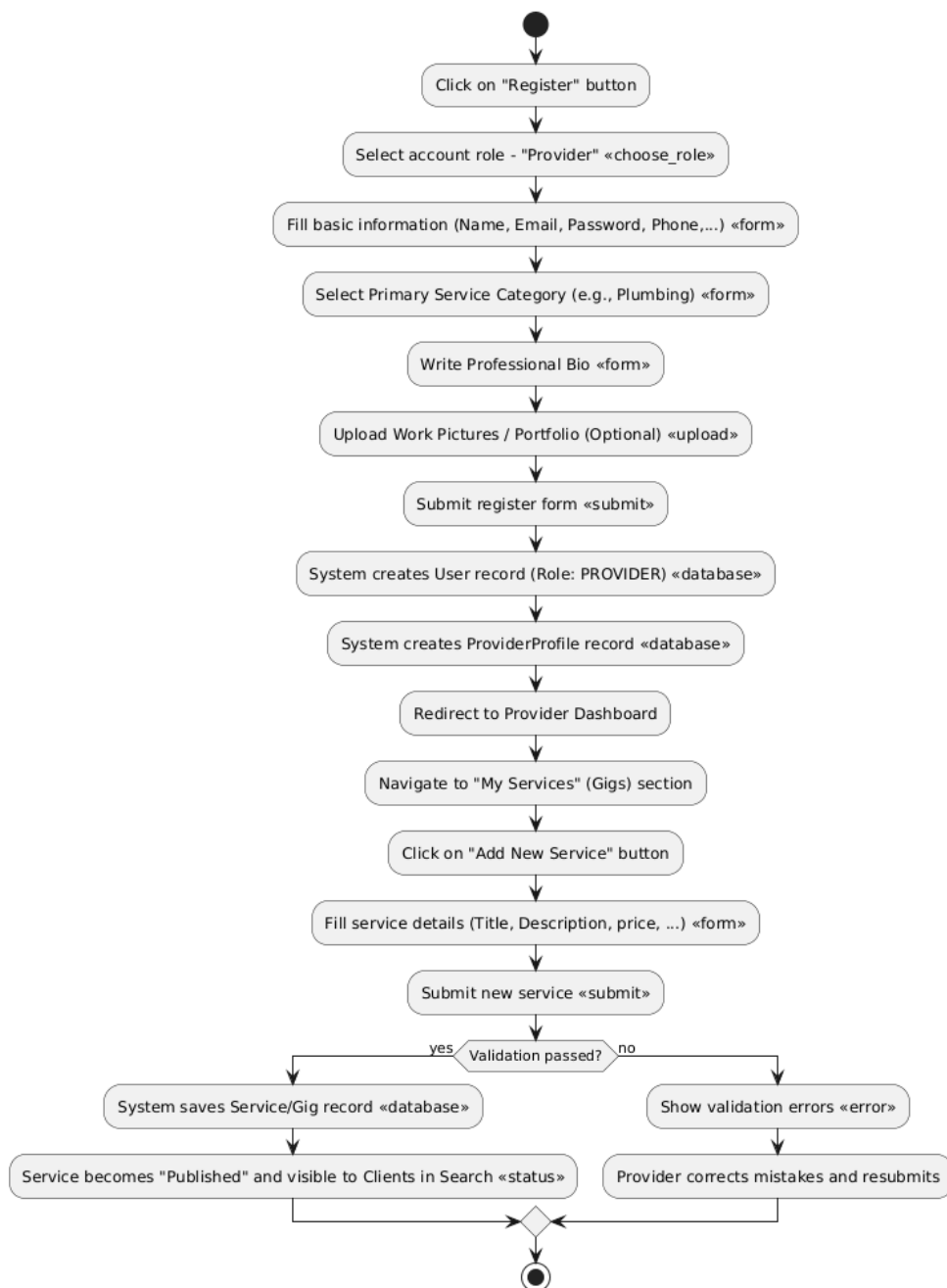


Figure 2.10: Provider Registration and Service Offering Activity Diagram

2.4.4 Interaction Sequences

This section details the runtime interaction sequences between the platform actors Visitors, Clients, and Providers, the front-end application, the back-end API server, database storage, and third-party integrations. These models illustrate the flow of requests, responses, and validation steps involved in the platform's core operations.

User Registration and Identity Verification: Figure 2.11 represents the sequence of

interactions involved in user registration and email verification within the platform.

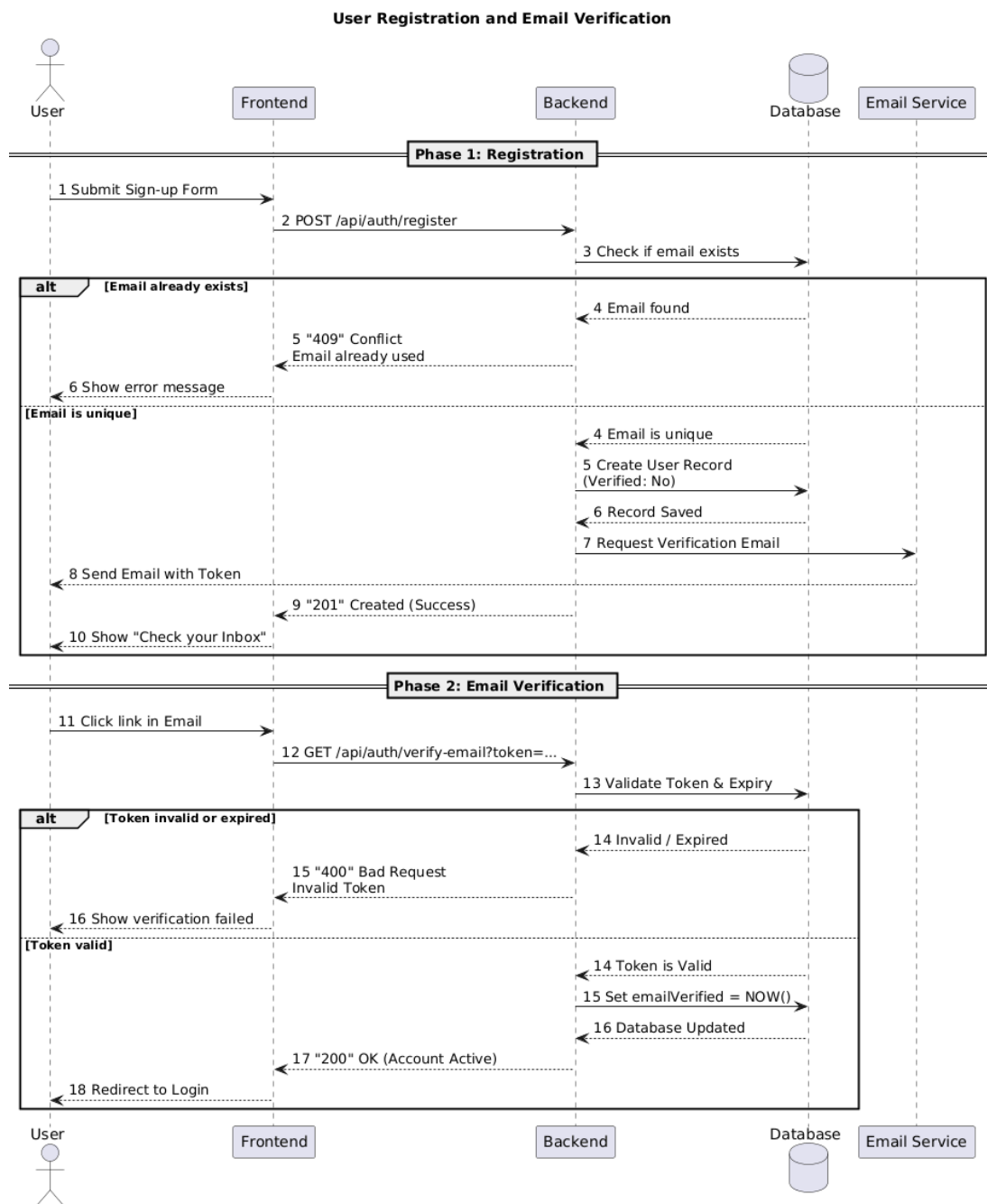


Figure 2.11: User Registration and Email Verification Sequence Diagram

Service Discovery and Data Obfuscation: Figure 2.12 represents the interaction sequence for provider discovery, including backend data obfuscation and frontend decryption operations used to protect sensitive provider information.

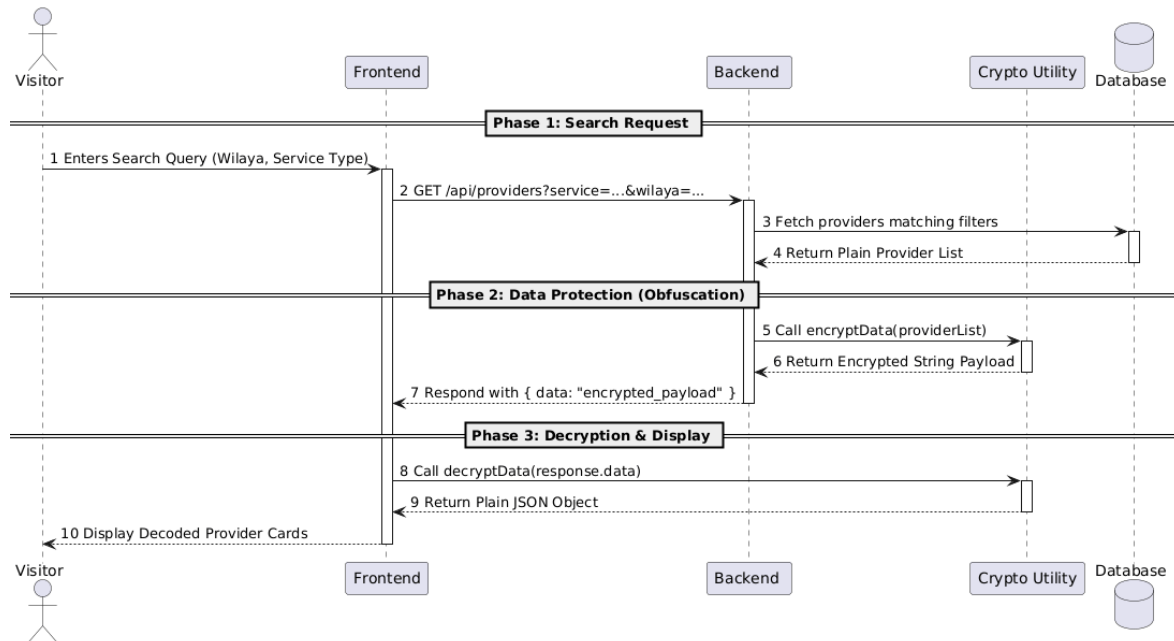


Figure 2.12: Service Discovery and Data Obfuscation Sequence Diagram

Booking Lifecycle Sequence: Figure 2.13 represents the interaction sequence of the service booking lifecycle, including request submission, provider approval, service completion, and client feedback operations.

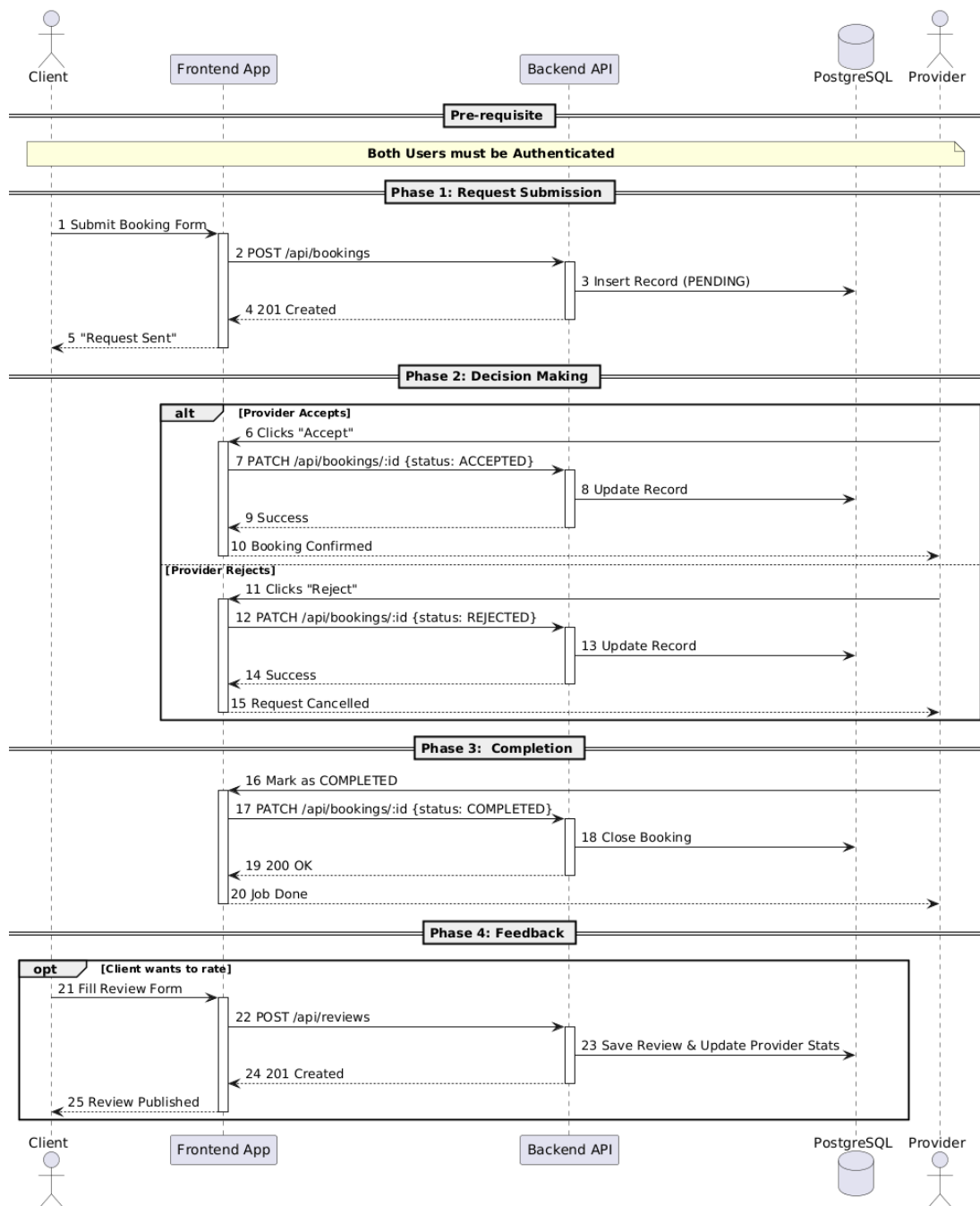


Figure 2.13: Booking Lifecycle Sequence Diagram

2.5 Admin Dashboard

The Admin Dashboard serves as the central administrative interface for platform management and moderation. It provides administrators with tools for monitoring platform statistics, managing user accounts, reviewing reports, and handling support requests to maintain service quality and platform security. This section presents the administrative interface structure, the core admin use cases, and the interaction sequences associated with authentication, moderation, reporting, and support management.

2.5.1 Admin Interface Structure

The administrative dashboard is composed of several dedicated pages for monitoring, moderation, and support management. Table 2.23 outlines the core administrative routes and their primary functionalities.

Route	Description
/admin	Platform monitoring and statistics overview
/admin/reports	Review and resolve user reports
/admin/users	Manage and moderate user accounts
/admin/messages	Handle contact and support requests

Table 2.23: Administrative Dashboard Pages

2.5.2 Admin Use Cases Diagram

The administrative functionalities of the platform are modeled using a UML Use Case Diagram. The administrator plays a critical role in maintaining platform integrity, enforcing moderation policies, and supervising the overall system activity. In this section, we present the main administrative use cases, including user management, report moderation, support request handling, and platform monitoring operations. Figure 2.14 illustrates the interactions between the administrator and the different functional modules of the administration workspace.

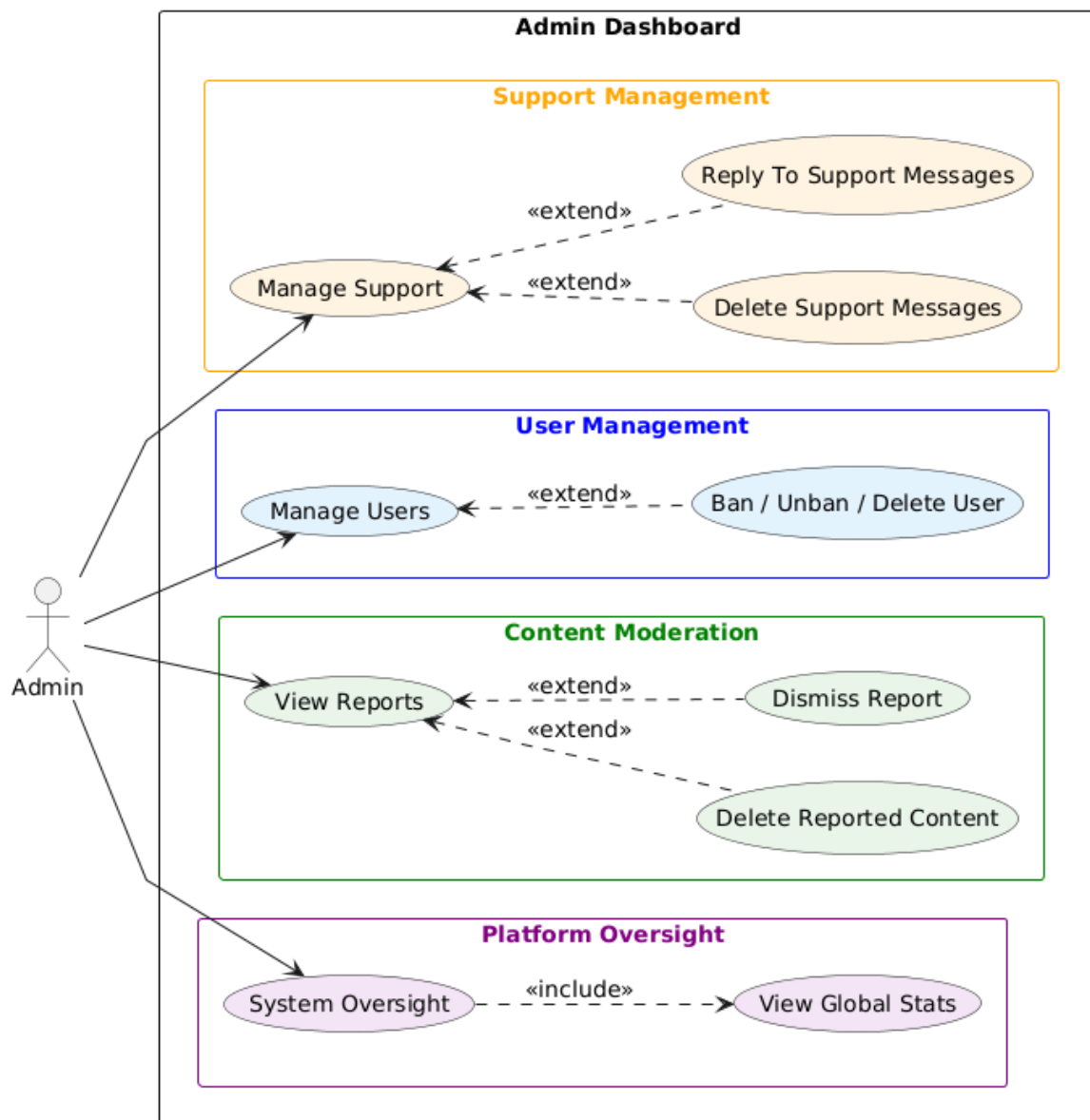


Figure 2.14: Admin Use Cases Diagram

2.5.3 Admin Sequence Diagrams

This section presents the interaction sequences involved in the platform's administrative operations using UML Sequence Diagrams. These models illustrate the flow of requests, validations, and system responses related to administrator authentication, user moderation, report management, and support request handling.

Admin Authentication Sequence: Figure 2.15 represents the runtime interaction sequence for secure administrator authentication and dashboard access within the platform.

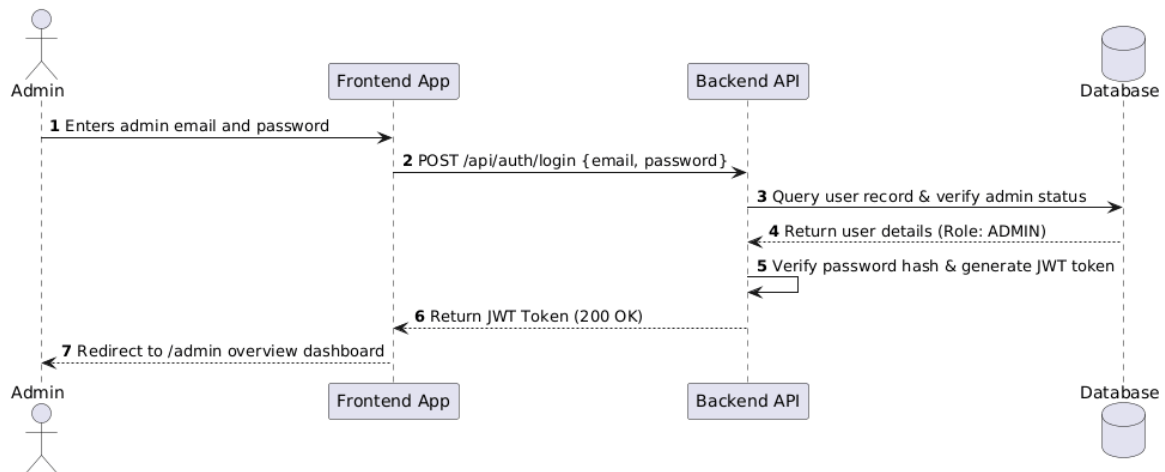


Figure 2.15: Admin Login Sequence Diagram

User Account Moderation Sequence: Figure 2.16 represents the interaction sequence for user account moderation, including account querying, suspension, restoration, and permanent deletion operations.

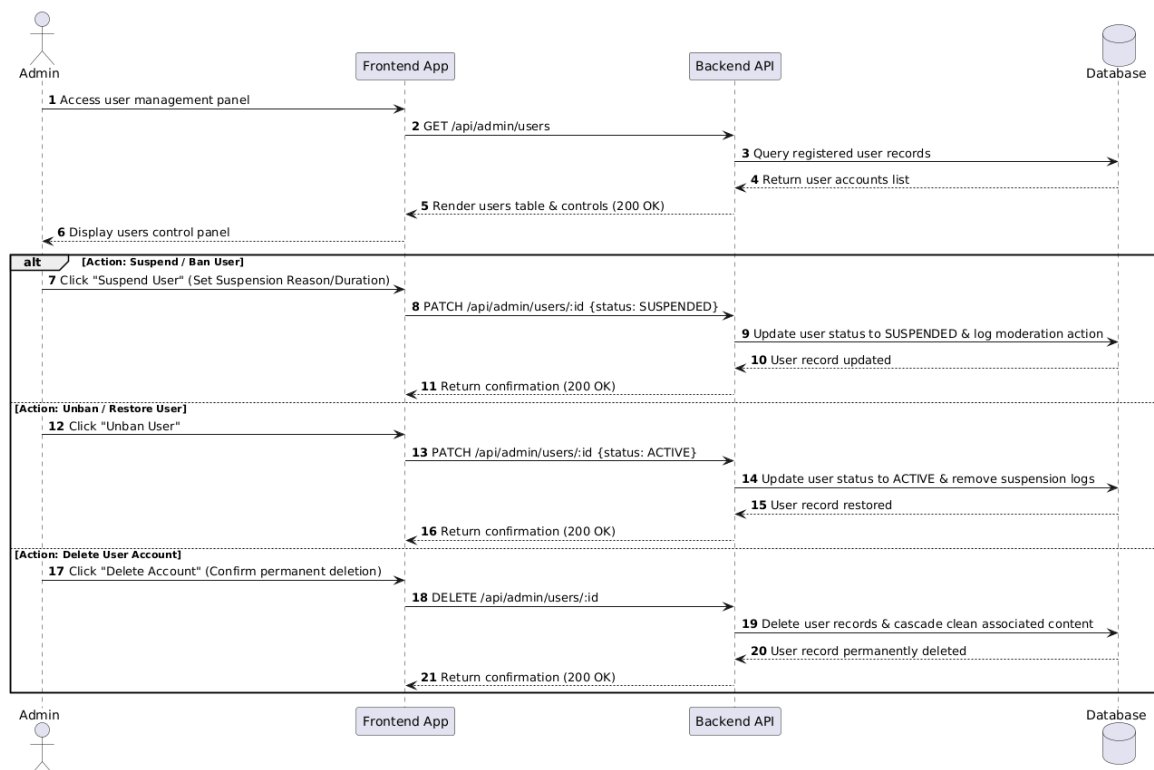


Figure 2.16: users management Sequence Diagram

Reports Resolution Sequence: Figure 2.17 represents the interaction sequence involved in administrative review, moderation, and resolution of submitted user reports.

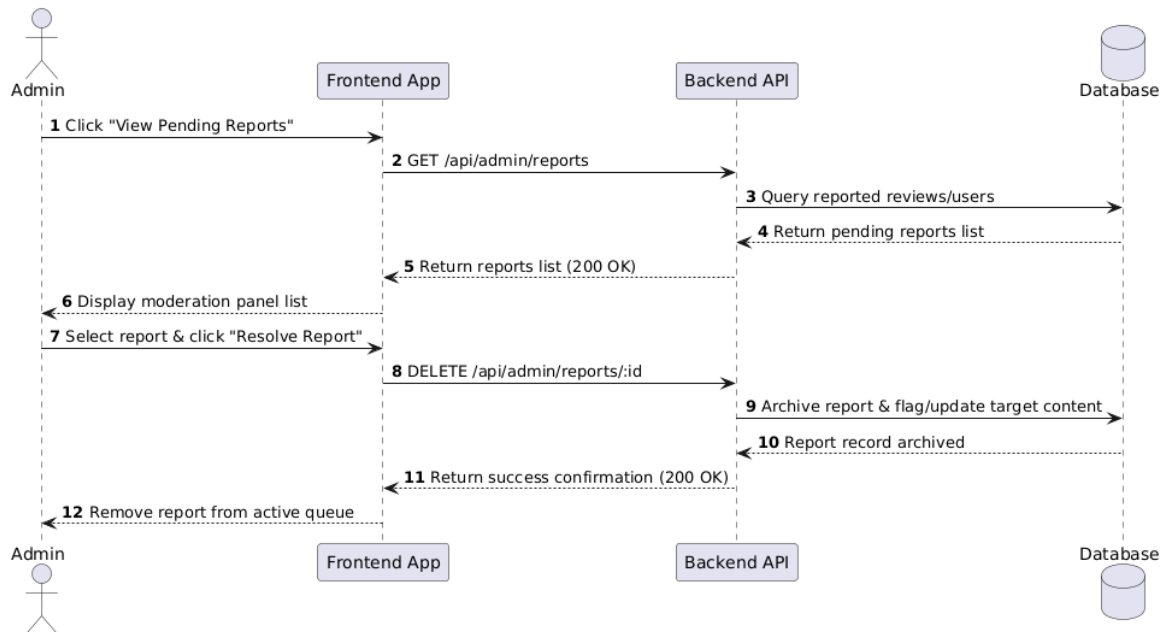


Figure 2.17: reports management Sequence Diagram

Support Message Management Sequence: Figure 2.18 represents the interaction sequence involved in reviewing public contact messages, replying via email, and managing message deletion operations.

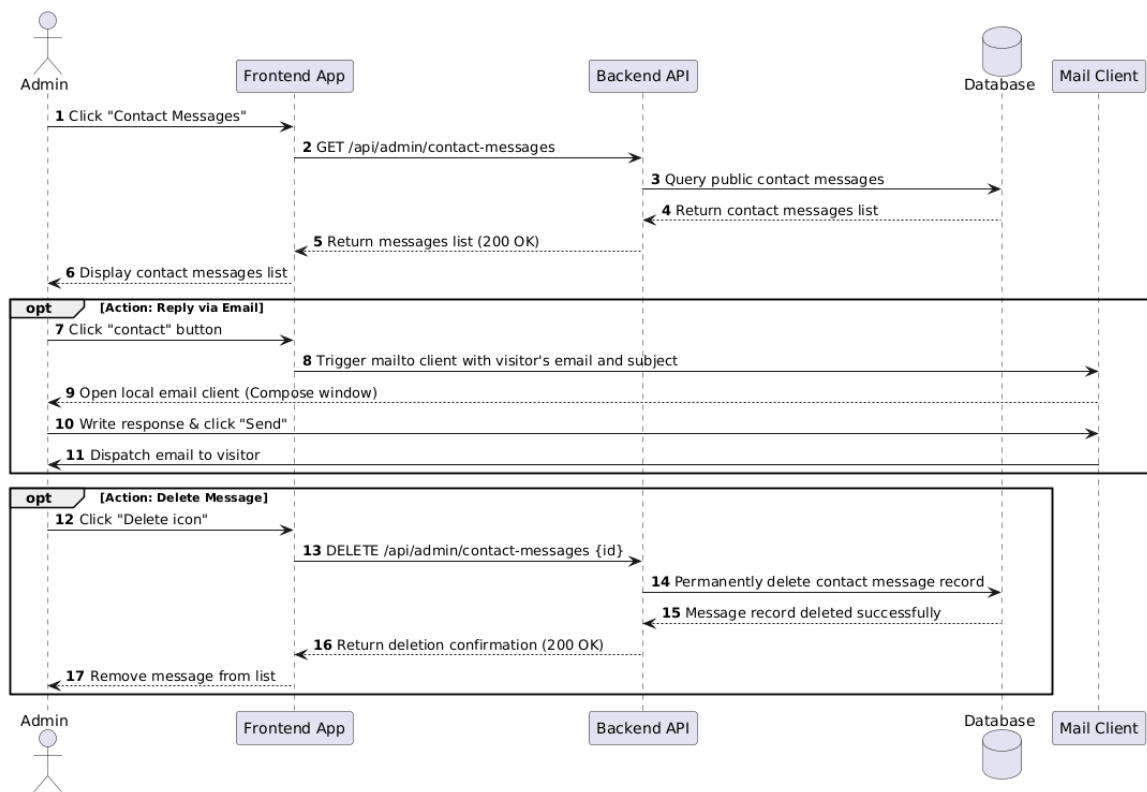


Figure 2.18: Support Message management Sequence Diagram

2.6 Conclusion

The architectural models and design specifications presented in this chapter provide a structured foundation for the implementation of the AlloServiceDZ platform. Through the use of the C4 model, the system architecture was defined across multiple abstraction levels, covering the overall system structure, container organization, and internal component interactions.

In addition, several UML models were introduced to describe both the static and dynamic aspects of the platform. Use Case, Activity, and Sequence Diagrams were used to illustrate the main user workflows, booking operations, administrative functionalities, and communication processes. These models also highlighted important security and validation mechanisms integrated within the system architecture.

Finally, the database design and administrative dashboard specifications established the core operational structure required for platform management, moderation, and service coordination. Together, these design elements provide a clear and coherent blueprint for the development and deployment phases of the platform.

Chapter 3

System Implementation and Deployment

This chapter presents the practical implementation phase of the AlloServiceDZ platform [1], covering the technical decisions, architectural structures, and deployment strategies adopted throughout the development process. The chapter is organized into four main sections. Section 3.1 describes the development environment and technology stack used to build the system. Section 3.2 outlines the physical and logical organization of the project codebase. Section 3.3 presents the platform interfaces for different user roles. Finally, Section 3.4 discusses the cloud deployment infrastructure and the continuous delivery pipeline used to operate the platform in a production environment.

3.1 Development Environment and Technologies

The implementation of AlloServiceDZ was carried out using a modern software engineering approach emphasizing type safety, modular architecture, reproducible environments, and efficient build performance. This section presents the software development environment, programming frameworks, package management tools, database technologies, and external cloud services utilized to build and deploy the platform.

3.1.1 Software Development Environment

To ensure consistency, maintain developer productivity, and provide compatibility between local and production environments, the software development environment was configured using a standardized set of engineering tools and utilities.

- **Integrated Development Environment (VS Code):** A lightweight and highly extensible source-code editor developed by Microsoft [11]. It provides built-in support for debugging, syntax highlighting, and a large ecosystem of modular extensions:
 - *ESLint*: A static code analysis tool used to detect problematic patterns in JavaScript and TypeScript code [12].
 - *Prettier*: An opinionated code formatter that automatically enforces consistent layout, spacing, and styling conventions across all files [13].

- *TypeScript IntelliSense*: A development service that offers real-time type checking, parameter hints, member suggestions, and intelligent code autocompletion for TypeScript projects [14].
- **Version Control System (Git)**: A distributed version control system used to track and manage source code changes during software development through commits, branching, and version history management [15].
- **Repository Hosting Platform (GitHub)**: A cloud-based platform used to host Git repositories and facilitate collaborative development workflows, code reviews, issue tracking, and automated deployment integrations [16].
- **API Testing Platform (Postman)**: An API client used for testing and debugging RESTful APIs. It enables developers to send HTTP requests, inspect JSON payloads and headers, and validate server responses during backend development [17].

3.1.2 Development Technology Stack

The AlloServiceDZ technology stack was selected to build a scalable and optimized full-stack web application using modern web and serverless technologies. The following sections present the main technologies used across the different architectural layers of the system.

Frontend Technologies: The frontend layer of AlloServiceDZ was developed using modern web technologies and UI libraries to provide responsive interfaces, interactive components, and a smooth user experience.

- **HTML5 (HyperText Markup Language)**: The standard markup language used to define the structure and semantic layout of web pages [18].
- **CSS3 (Cascading Style Sheets)**: A stylesheet language used to control the visual appearance, layouts, responsiveness, and animations of web interfaces [19].
- **JavaScript (ES6+)**: A high-level scripting language used to implement dynamic client-side interactions and asynchronous operations [20].
- **React 18**: A component-based JavaScript library developed by Meta for building interactive and reusable user interfaces efficiently [21].
- **TypeScript**: A strongly typed superset of JavaScript that improves code maintainability, type safety, and development reliability [22].
- **Material-UI (MUI Core)**: A React UI framework implementing Google's Material Design principles, providing reusable components and responsive layouts [23].

- **Leaflet & React Leaflet:** Libraries used to integrate interactive geographical maps and location-based functionalities into the platform [24].

Backend Technologies: The backend layer of AlloServiceDZ was developed using modern server-side technologies to support API management, authentication, and business logic execution.

- **Next.js 15 (App Router & Route Handlers):** A full-stack React framework used to implement server-side rendering, API route handling, middleware processing, and application routing within a unified architecture [25].
- **Node.js:** An open-source JavaScript runtime environment built on Chrome's V8 engine, used to execute server-side operations and support the backend infrastructure of the platform [26].
- **NextAuth.js (Auth.js):** An authentication framework for Next.js used to manage user authentication, session handling, credential validation, and social login integrations within the platform [27].

Database Technologies: The database layer of AlloServiceDZ relies on relational database technologies and ORM tools to ensure reliable data storage, consistency, and efficient database interaction.

- **PostgreSQL 16:** An open-source relational database management system RDBMS used for storing and managing structured application data with strong reliability and transactional support [28].
- **Prisma ORM:** An Object-Relational Mapping ORM tool used to simplify database operations through type-safe queries and schema-based database modeling [29].

External Services: The system integrates several external services to support media management, email communication, and geographic functionalities.

- **Cloudinary:** A cloud-based media management service used for storing, optimizing, and delivering uploaded images within the platform [30].
- **Google SMTP Service:** An email service used to send automated emails such as account verification and notification messages [31].
- **OpenStreetMap (OSM):** An open-source geographic data service used to provide map tiles and location visualization features through Leaflet integration [32].

3.2 Project Structure

This section presents the physical and logical organization of the AlloServiceDZ codebase, highlighting the separation between frontend interfaces, backend services, database schemas, and shared application resources.

3.2.1 Frontend Structure

The frontend architecture of AlloServiceDZ is based on the Next.js 15 App Router structure, which provides file-system-based routing and a modular organization of application pages and components. The frontend is divided into three main architectural layers.

Routing and Page Composition Layer (`app/` Directory): The routing layer defines the main application pages and layouts using nested directory structures provided by Next.js.

- **Root Layout (`app/layout.tsx`):** Defines the global application layout, shared styles, metadata configuration, and global providers used across the platform.
- **Page Routes (e.g., `app/page.tsx`, `app/providers/page.tsx`):** Represent the main entry points for different application pages and organize page content and layouts.
- **Server-Side Rendering Support:** Pages are rendered using Next.js server-side rendering mechanisms to improve application performance and initial loading speed.

Component and Presentation Layer (`components/` Directory): Reusable user interface elements are separated from routing logic to improve maintainability and code reusability.

- **Domain-Specific Components (`components/home/`, `components/map/`):** Contain reusable interface modules related to specific application sections such as homepage sections and map visualization features.
- **Shared UI Components (`components/ui/`):** Include reusable visual helpers and animation components used throughout the platform interface.
- **Interactive Client Components (`"use client"`):** Components requiring browser-side interactivity, such as maps, search inputs, and dynamic forms, are implemented as client components to support local state management and interactive behaviors.

State and Localization Layer (`context/` Directory): Global application states and localization settings are managed separately to ensure centralized state handling across the platform.

- **Language Context (`context/language-context.tsx`):** Handles multilingual support, language switching, translation mappings, and RTL/LTR layout management across the application interface.

3.2.2 Backend Structure

The backend architecture of AlloServiceDZ is responsible for handling business logic, request processing, security mechanisms, and communication with external services. The backend structure is organized into three main layers.

Server-Side Configuration and Utility Layer (`lib/` Directory): This layer contains configuration files, authentication settings, utility functions, and external service integrations used across the backend system.

- **Authentication Configuration (`lib/auth.ts`):** Manages authentication settings, session handling, user credential validation, and JSON Web Token (JWT) processing.
- **Mail Configuration (`lib/mail.ts`):** Configures the email service used to send automated verification and notification emails through Nodemailer.
- **Application Utilities:** Contain reusable helper functions, localization utilities, and shared backend operations used across different API routes and services.

Request Protection and Middleware Layer (`middleware.ts`): This layer is responsible for handling route protection and access control before requests reach the application routes.

- **Route Protection Middleware (`middleware.ts`):** Intercepts incoming HTTP requests, validates authentication sessions, extracts user roles, and redirects unauthorized users away from protected routes such as (`/admin`) and (`/dashboard`).

API Routing and Handler Layer (`app/api/` Directory): This layer acts as the server-side gateway of the platform and hosts the API route handlers responsible for processing client requests.

- **API Route Handlers (`app/api/.../route.ts`):** API endpoints are organized into modular directories where HTTP requests (GET, POST, PATCH, DELETE) are processed through dedicated route handlers. These handlers validate incoming data, interact with the Prisma database layer, execute backend operations, and return standardized JSON responses to the frontend application.

3.3 Platform Presentation

This section presents the main interfaces of the AlloServiceDZ platform through screenshots and interface descriptions. It highlights the visual design, navigation structure, and the core functionalities available to different platform users, including visitors, clients, providers, and administrators.

3.3.1 Visitor Interfaces

The Visitor Interface includes all publicly accessible pages that can be accessed by guest users without requiring user authentication.

Home Page: Figure 3.1 shows the Home Page interface of the AlloServiceDZ platform. It serves as the main entry point for visitors and provides quick access to service discovery functionalities.

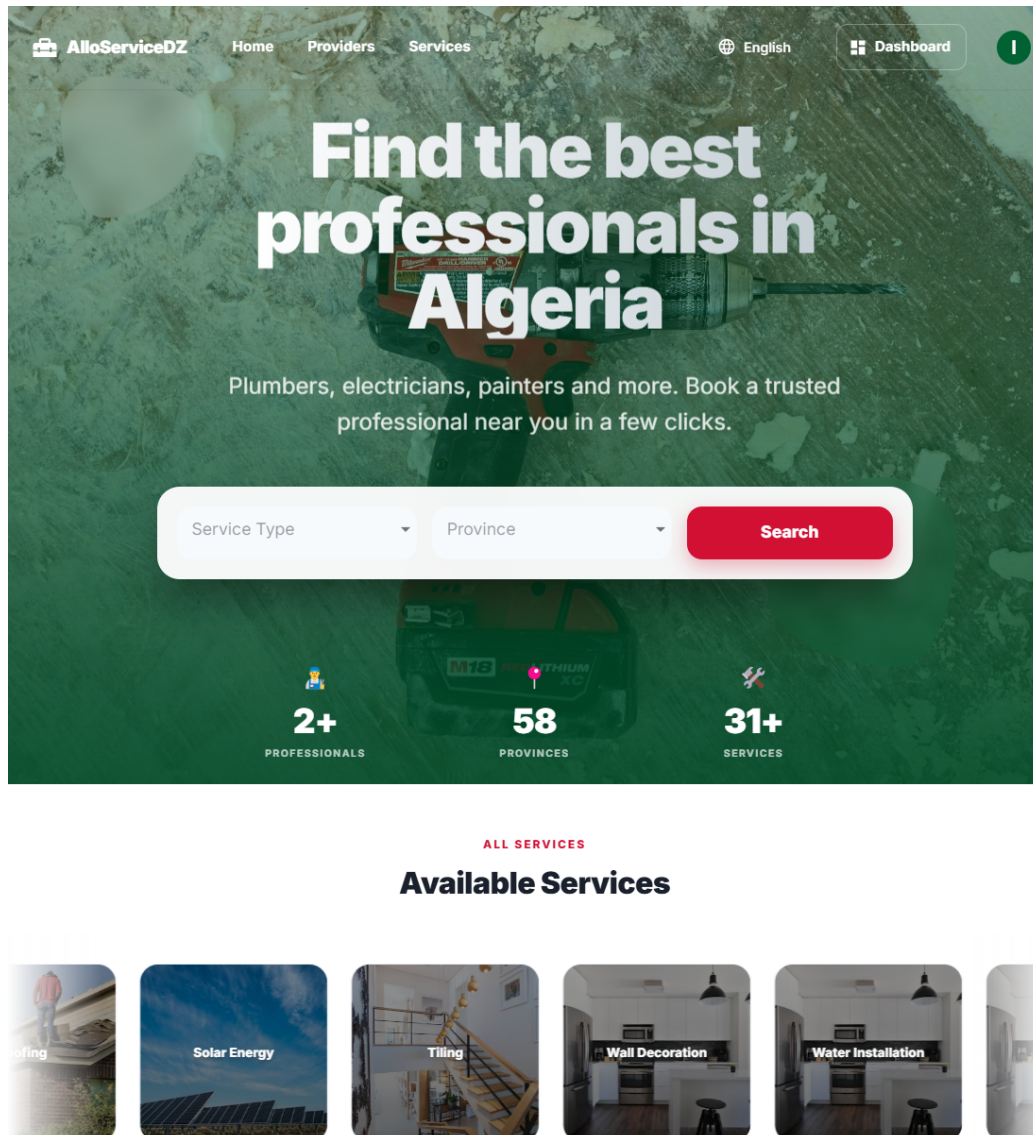


Figure 3.1: Home Page

Services Page: Figure 3.2 shows the Services Page interface of the AlloServiceDZ platform. This page presents the different service categories available within the platform in a card-based layout.

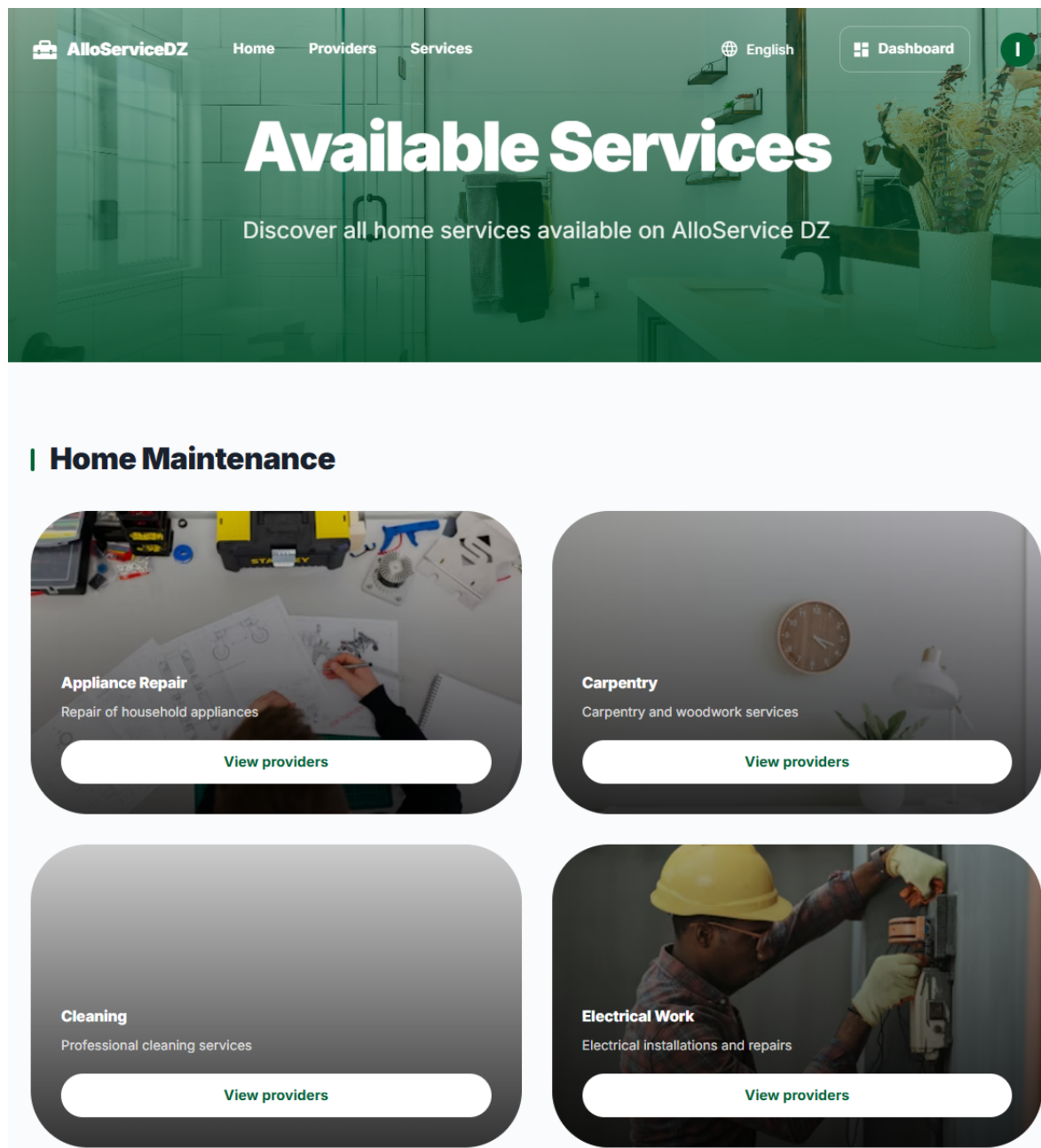


Figure 3.2: Services Page

Provider Search: Figure 3.3 shows the Provider Search interface of the AlloServiceDZ platform. This page allows users to locate service providers through multi-criteria filtering options including Wilaya geographic selectors and service type filters.

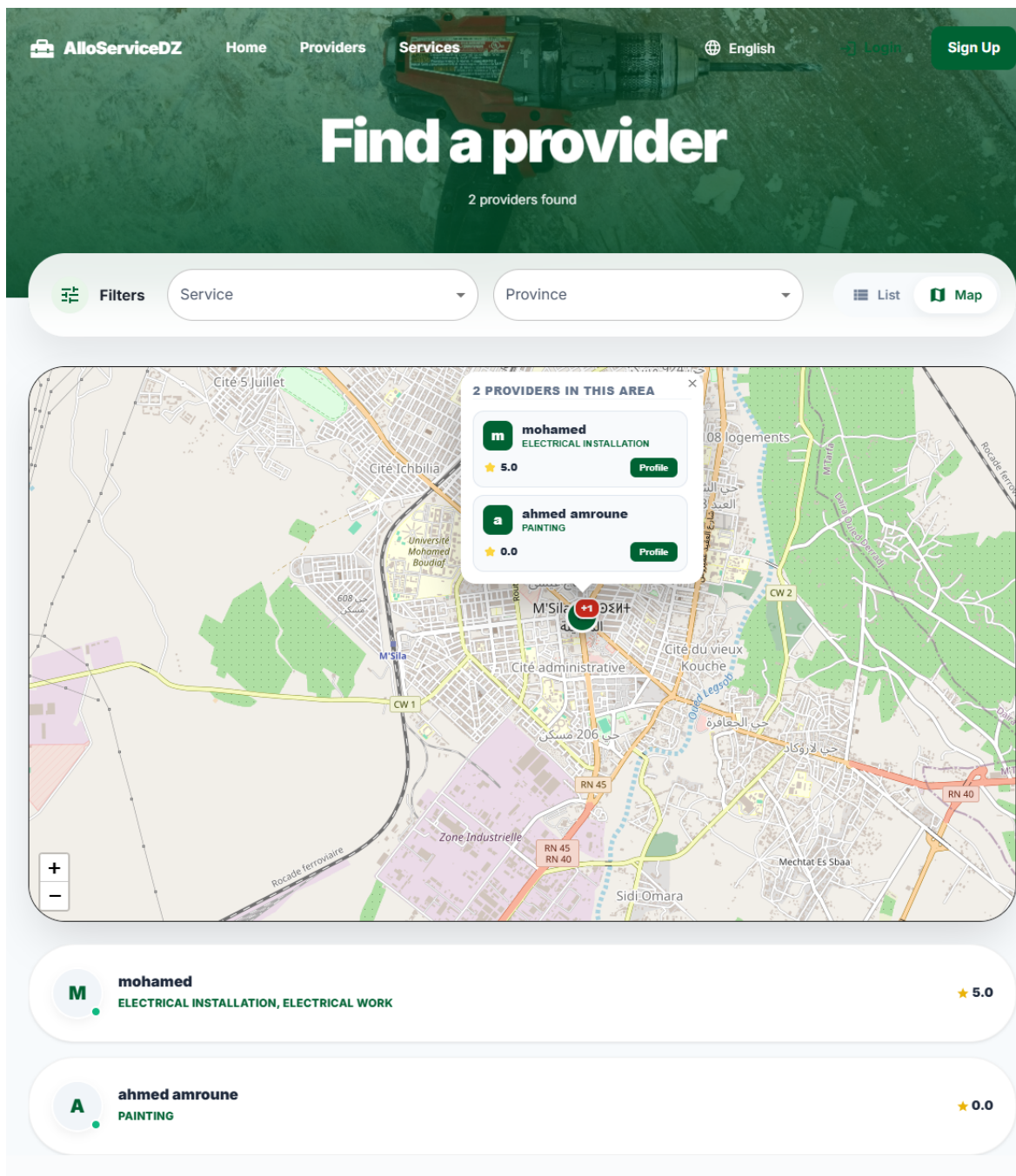


Figure 3.3: Providers Page

Provider Profile: Figure 3.4 shows the Provider Profile page of the AlloServiceDZ platform. This page presents the detailed professional profile of a service provider, including their biography, average rating, price range, geographic location, and portfolio images.

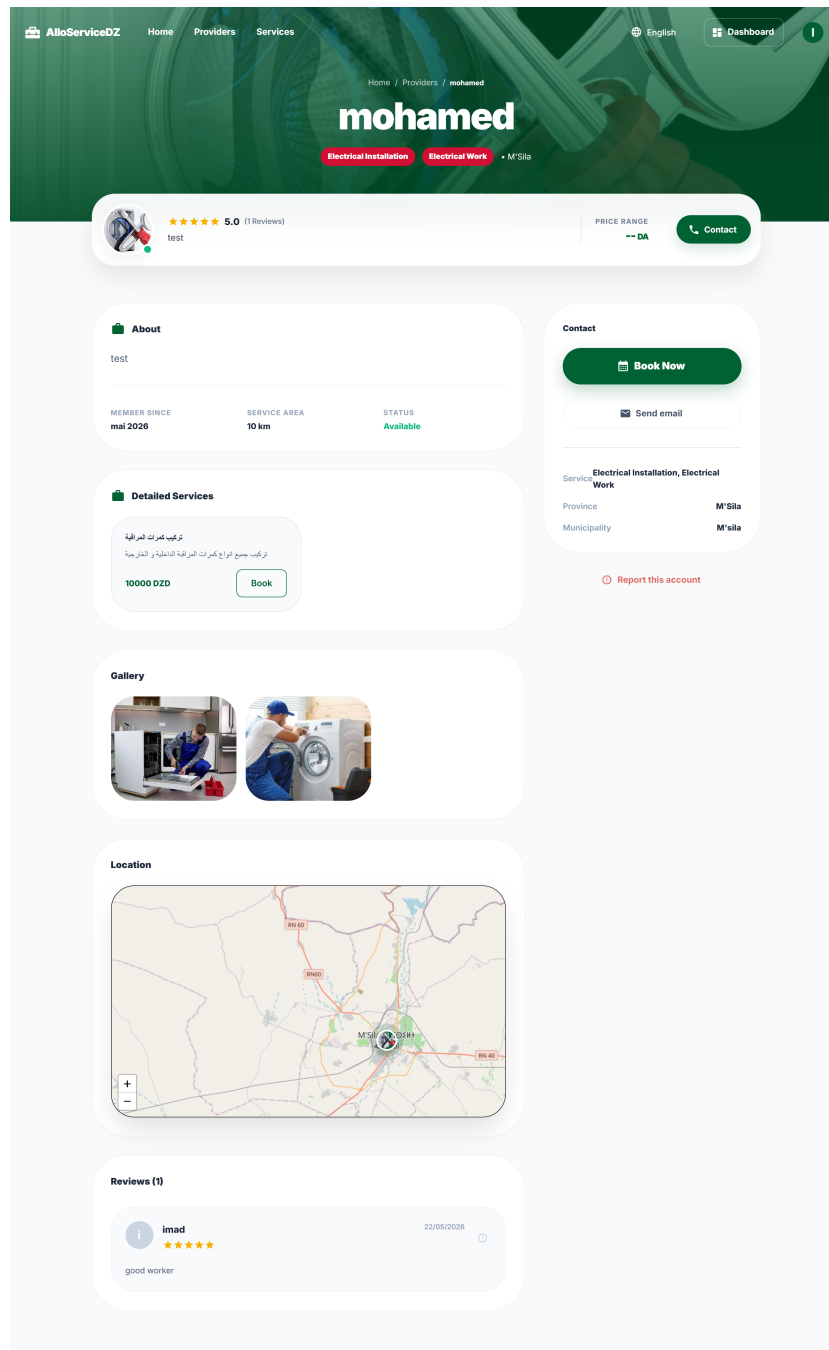
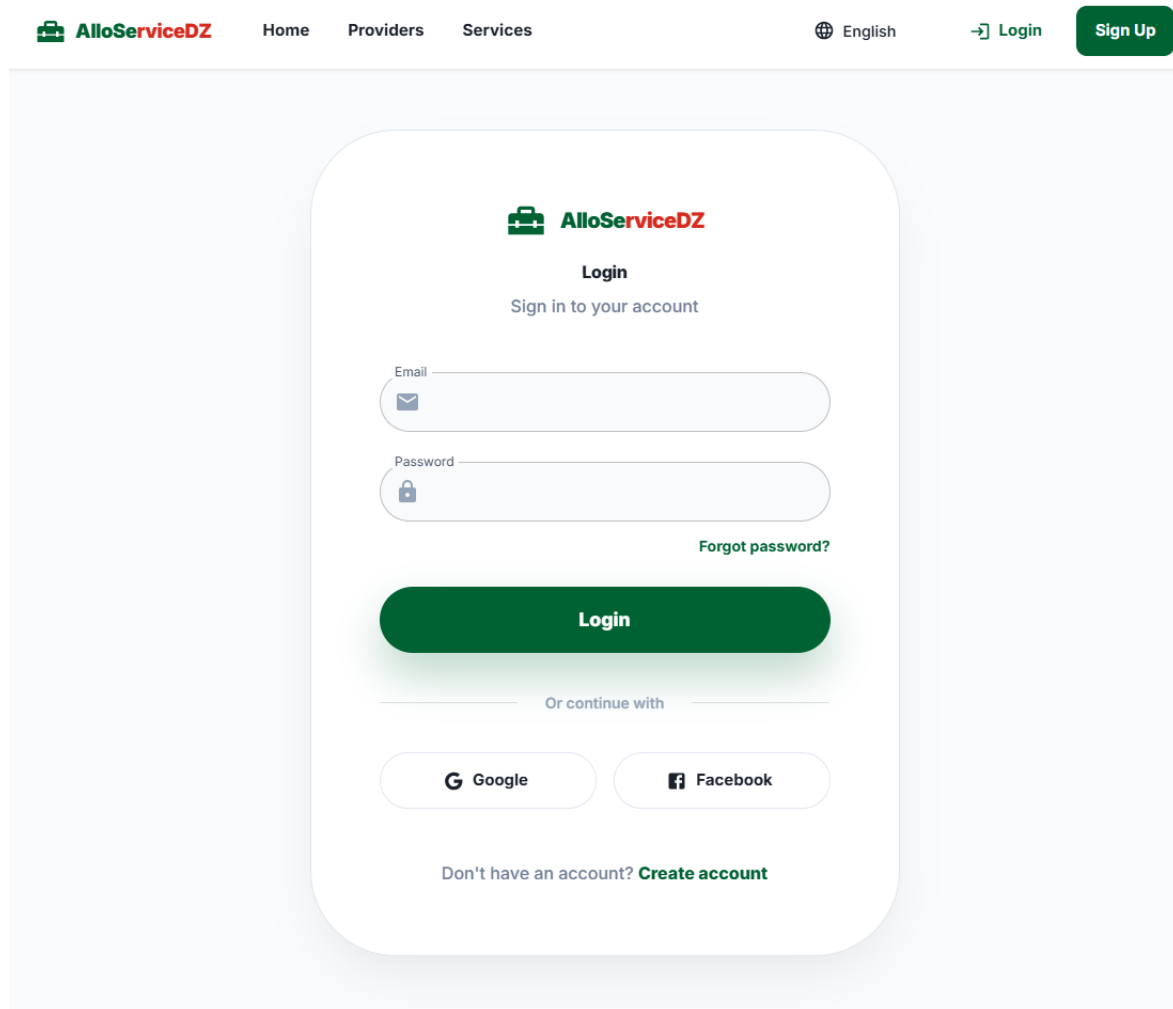


Figure 3.4: Provider Profile Page

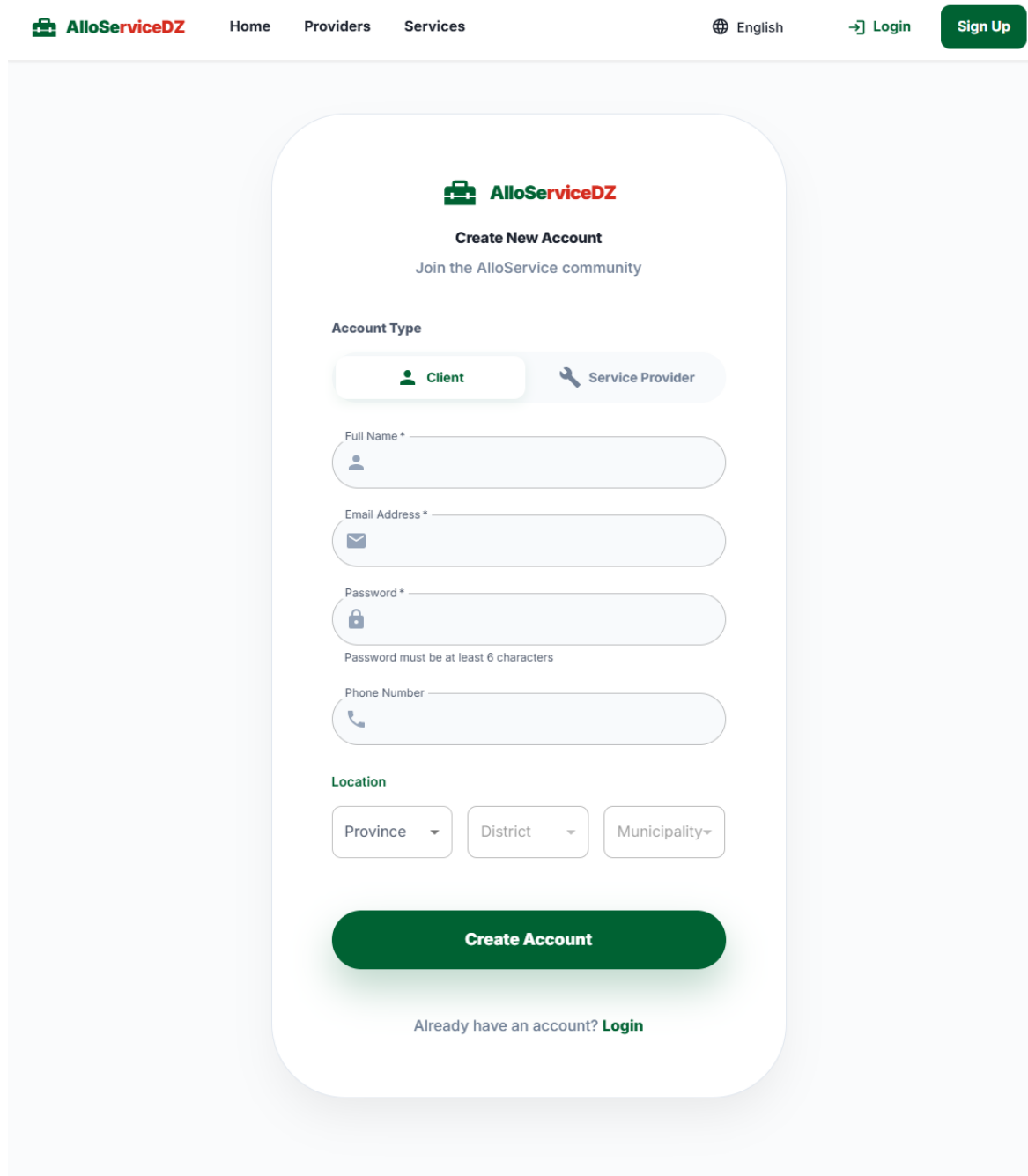
Login: Figure 3.5 shows the Login interface of the AlloServiceDZ platform. This page provides a credential-based authentication form.



The image shows the login page of the AlloServiceDZ platform. At the top, there is a navigation bar with the AlloServiceDZ logo, links for Home, Providers, and Services, a language selector set to English, and buttons for Login and Sign Up. The main content area features a white rounded rectangle with the AlloServiceDZ logo and the title 'Login'. Below the title is the instruction 'Sign in to your account'. There are two input fields: 'Email' with an envelope icon and 'Password' with a lock icon. A 'Forgot password?' link is positioned to the right of the password field. A large green 'Login' button is centered below the fields. Below this button is a horizontal line with the text 'Or continue with'. Underneath are two buttons for 'Google' and 'Facebook'. At the bottom, there is a link that says 'Don't have an account? Create account'.

Figure 3.5: Login Page

Register: Figure 3.6 shows the Registration interface of the AlloServiceDZ platform. this page offers two distinct registration paths for clients and service providers.



The registration interface of the AlloServiceDZ platform is displayed. The header includes the AlloServiceDZ logo, navigation links for Home, Providers, and Services, a language selector for English, and buttons for Login and Sign Up. The main content area features a 'Create New Account' section with the subtext 'Join the AlloService community'. Under 'Account Type', there are two options: 'Client' (selected) and 'Service Provider'. The form includes input fields for Full Name, Email Address, Password (with a note that it must be at least 6 characters), and Phone Number. Below these is a 'Location' section with dropdown menus for Province, District, and Municipality. A large green 'Create Account' button is at the bottom, followed by a link for users who already have an account to 'Login'.

Figure 3.6: Register Page

3.3.2 Client Interfaces

The Client Interface is accessible to authenticated users registered under the (CLIENT) role. It provides booking management, near-real-time messaging, and profile configuration tools.

Client Dashboard: Figure 3.7 shows the Client Dashboard interface of the AlloServiceDZ platform. This page provides clients with a centralized view of all their booking requests organized by status (PENDING, ACCEPTED, IN_PROGRESS, COMPLETED, CANCELLED, ARCHIVED).

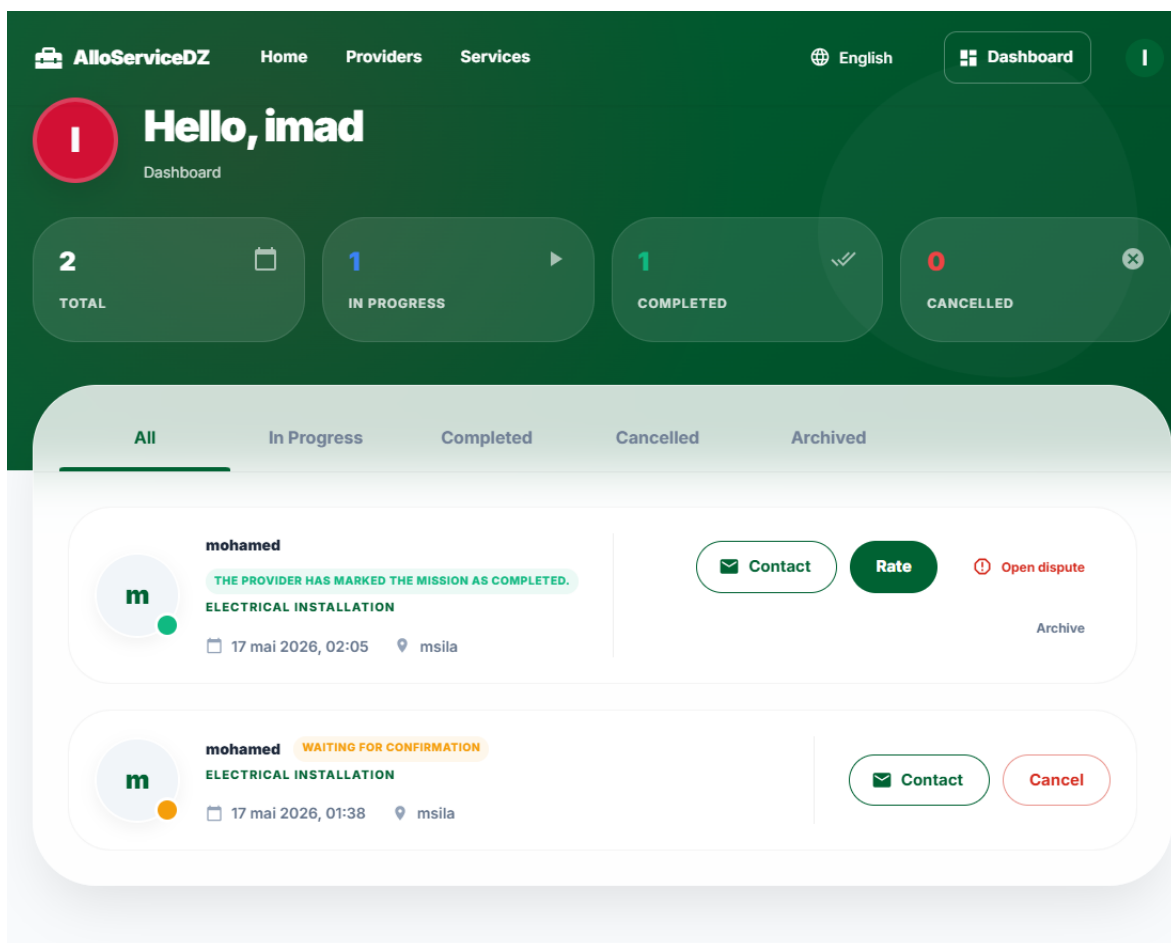


Figure 3.7: Client Dashboard Page

Booking Request: Figure 3.8 shows the Booking Request interface of the AlloServiceDZ platform. This page allows clients to submit a new service request by specifying the service type, preferred scheduling date, service address, and a service description.

AlloServiceDZ Home Providers Services English Dashboard

New Booking

Confirm providers • mohamed

☒ **Description**

Specify your problem or other details...

☒ **Confirm booking**

Provider

 **mohamed**
ELECTRICAL INSTALLATION, ELECTRICAL WORK

★★★★★ 5.0 (1 Reviews)

M'Sila, M'sila

PRICE RANGE -- DA

☒ Verified professionals. Quality service guaranteed for every booking.

Figure 3.8: Booking Request Page

Messaging Interface: Figure 3.9 shows the Messaging interface of the AlloServiceDZ platform. This page provides a dual-panel chat interface that allows clients to communicate with service providers.

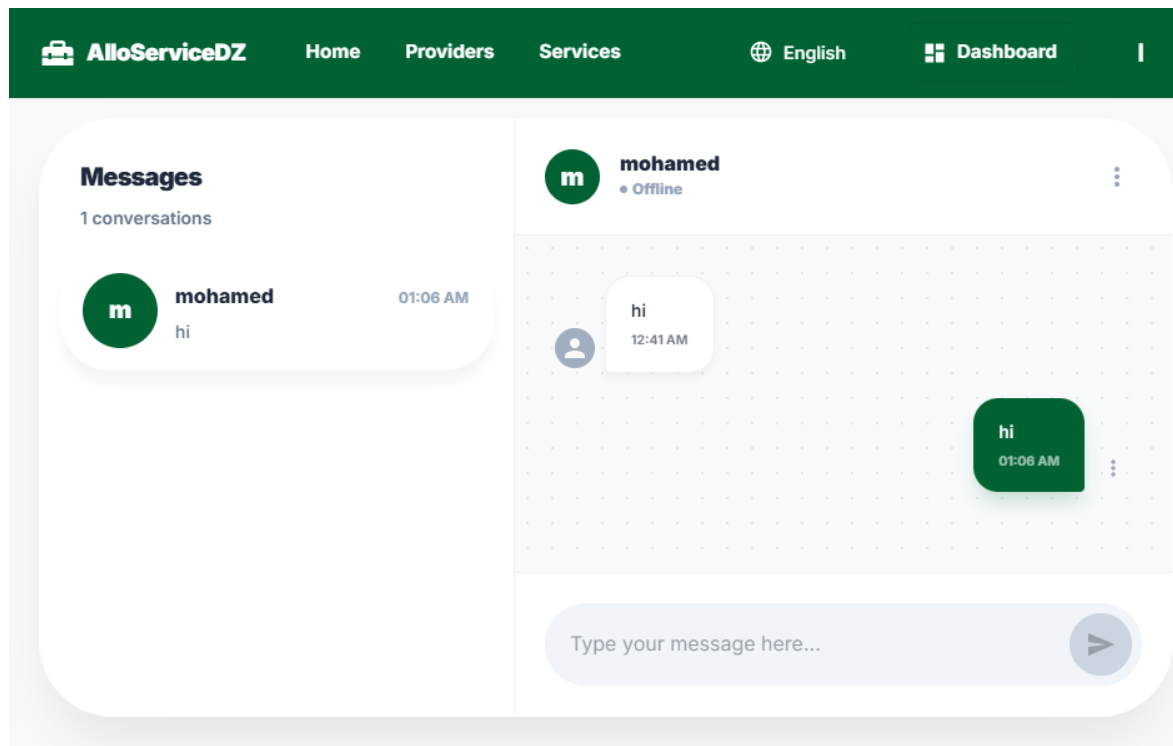


Figure 3.9: Messages Page

Profile Settings: Figure 3.10 shows the Profile Settings interface of the AlloServiceDZ platform. This page allows authenticated users to update their personal information including display name, phone number, profile image, and geographic location.

The screenshot displays the 'Profile Settings' interface of the AlloServiceDZ platform. The top navigation bar includes the AlloServiceDZ logo, links to Home, Providers, and Services, a language selector for English, and a Dashboard button. The user's profile is shown with a circular placeholder for a profile picture, the name 'imad', and the role 'CLIENT'. Below this, a modal form titled 'Personal Information' is open, showing fields for Full Name, Email, and Phone. The Email field is marked as 'Not editable'. An 'Edit' button is located in the top right corner of the form.

Field	Value	Editable
Full Name	imad	Yes
Email	faviw69733@justnapa.com	No (Not editable)
Phone	+213 ...	Yes

Figure 3.10: Profile Settings Page

3.3.3 Provider Interfaces

The Provider Interface is accessible to authenticated users registered under the (PROVIDER) role. It extends the standard user dashboard with professional profile management, service offering tools, and booking acceptance workflows.

Provider Dashboard: Figure 3.11 shows the Provider Dashboard interface of the AlloServiceDZ platform. This page serves as the main operational panel for service providers, allowing them to manage incoming booking requests.

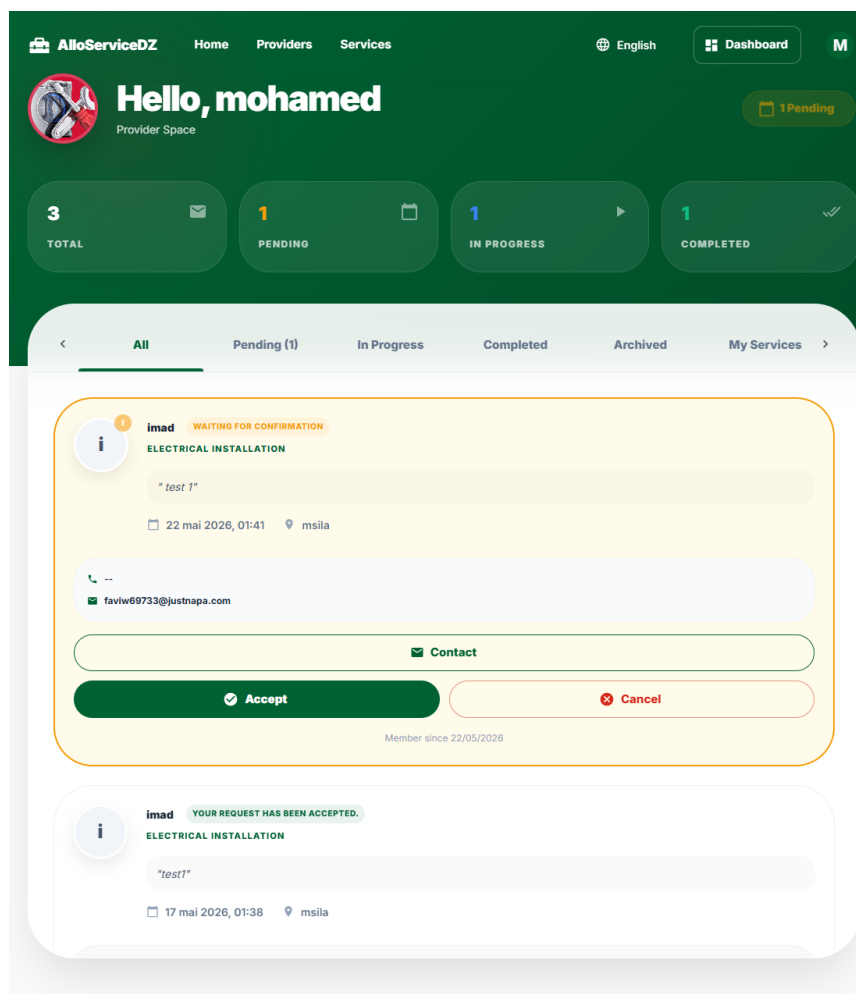


Figure 3.11: Provider Dashboard Page

Gig Manager: Figure 3.12 shows the Gig Manager interface of the AlloServiceDZ platform, accessible within the provider dashboard. This section allows service providers to create, edit, and manage their individual service offerings.

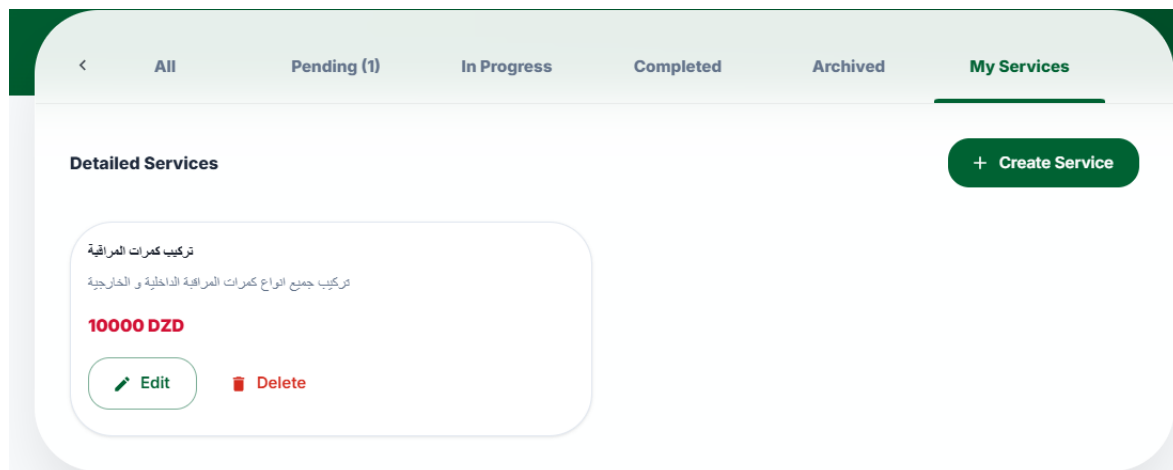


Figure 3.12: Gig Manager Interface

Professional Profile Setup: Figure 3.13 shows the Professional Profile Setup interface of the AlloServiceDZ platform. This page allows service providers to configure their public-facing profile, including professional biography, price range, geographic service area, service radius in kilometers, and portfolio image gallery.

The screenshot displays the 'Professional Profile Setup' interface for a user named 'mohamed', identified as a 'SERVICE PROVIDER'. The page has a dark green header with the 'AlloServiceDZ' logo and navigation links: 'Home', 'Providers', and 'Services'. On the right, there are links for 'English', 'Dashboard', and a user icon 'M'.

Below the header, a circular profile picture placeholder shows tools like a wrench and a screwdriver. The name 'mohamed' is prominently displayed in white, with 'SERVICE PROVIDER' in a smaller green box below it.

The main content area is a light gray rounded rectangle with four tabs: 'Personal Information', 'Location Information', 'Professional Information' (which is active and underlined), and 'Gallery'.

Under the 'Professional Information' tab, there is a section titled 'Professional Information' with a subtext 'Define your service details, pricing, and availability.' and an 'Edit' button. Below this is a 'Biography / Description' section with a text input field containing the word 'test'.

Next are two input fields: 'Price Range (e.g. 1000 - 5000 DA)' with a placeholder 'ex: 1000 - 5000 DA', and 'Service Radius (km) (km)' with a placeholder '10'.

At the bottom, there is a green box for 'Availability' with a bullet point. It contains the text 'I am currently available' and a checked checkbox.

Figure 3.13: Provider Profile Setup Page

3.3.4 Admin Interfaces

The Admin Interface is exclusively accessible to platform administrators under the (ADMIN) role, providing full oversight and moderation controls for all platform entities.

Admin Dashboard: Figure 3.14 shows the Admin Dashboard interface of the AlloServiceDZ platform. This page provides administrators with a high-level overview of key platform metrics.

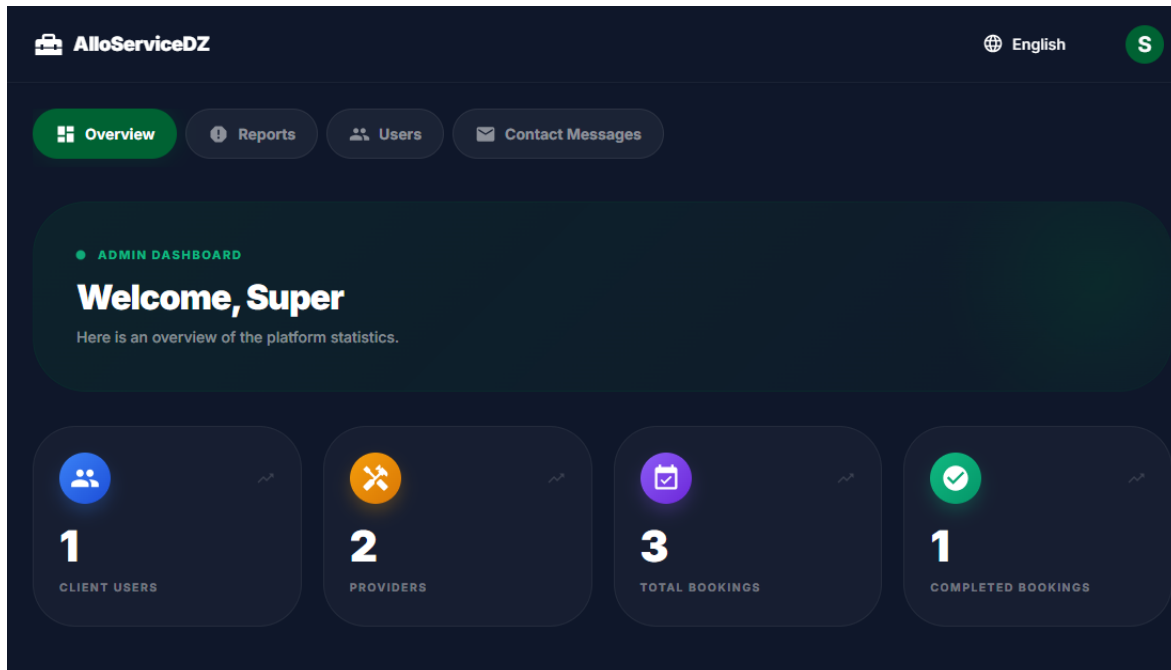
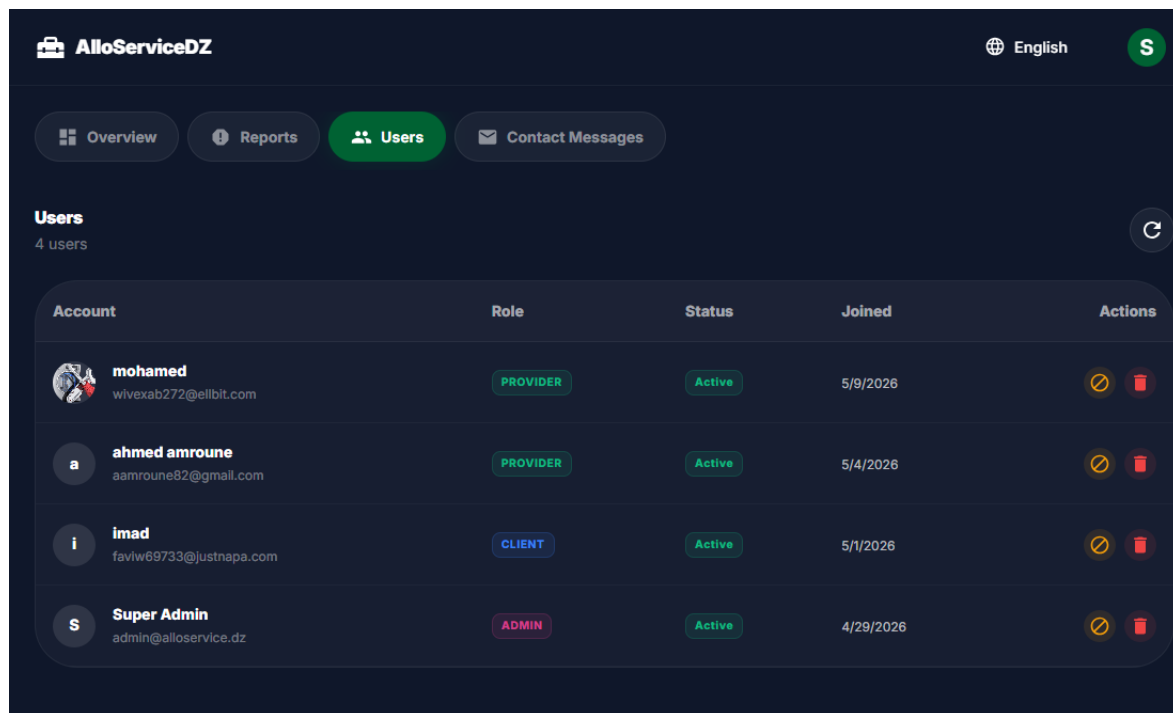


Figure 3.14: Admin Dashboard Interface

User Management: Figure 3.15 shows the User Management interface of the AlloServiceDZ platform. This page allows administrators to view and manage all registered user accounts through a structured data table.



Account	Role	Status	Joined	Actions
 mohamed wivexab272@ellbit.com	PROVIDER	Active	5/9/2026	 
 ahmed amroune aamroune82@gmail.com	PROVIDER	Active	5/4/2026	 
 imad faviw69733@justnapa.com	CLIENT	Active	5/1/2026	 
 Super Admin admin@alloservice.dz	ADMIN	Active	4/29/2026	 

Figure 3.15: User Management Interface

Reports and Moderation: Figure 3.16 shows the Reports and Moderation interface of the AlloServiceDZ platform. This page enables administrators to review all user-submitted reports categorized by type (USER, REVIEW, MESSAGE, BOOKING).

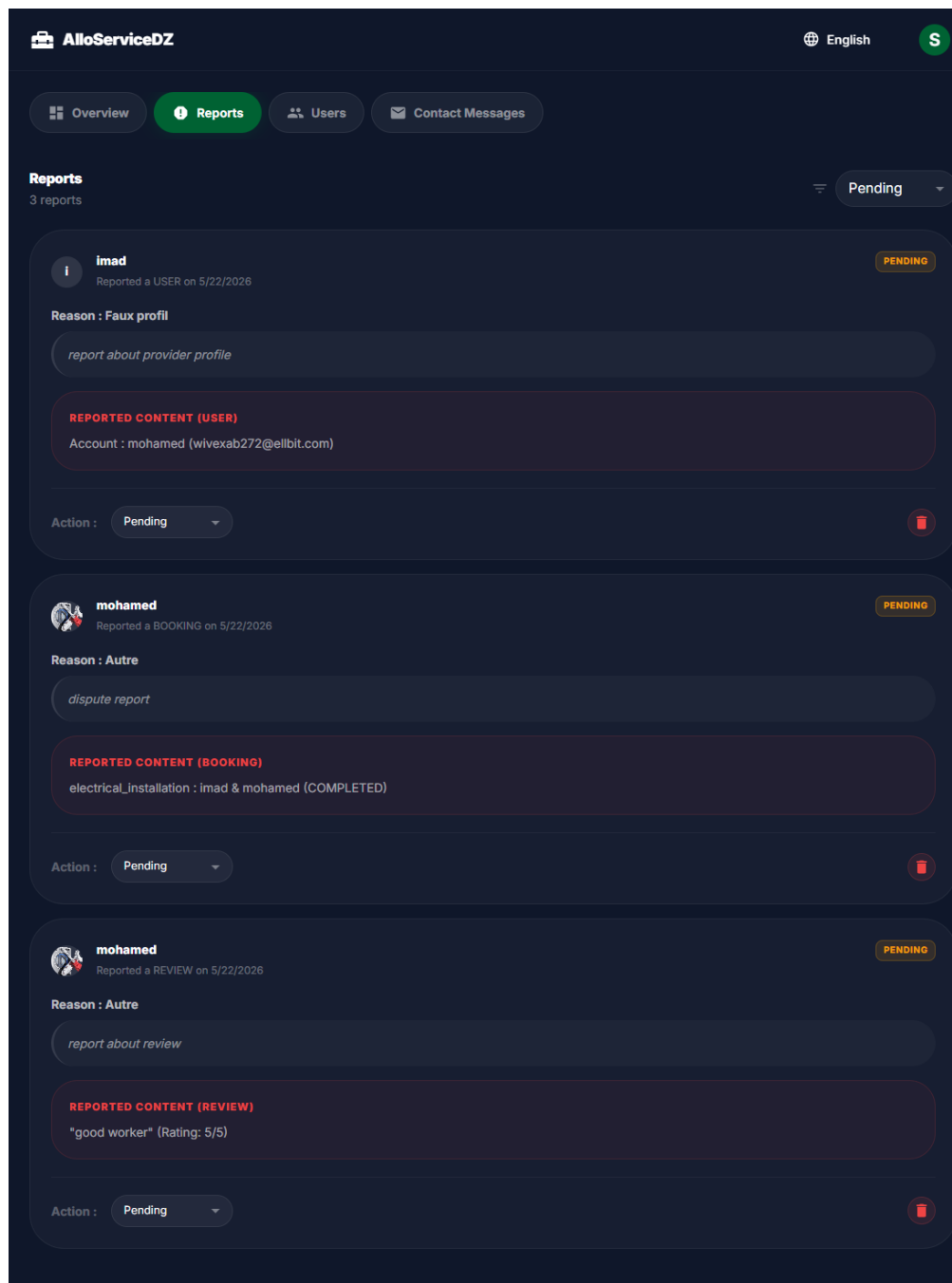


Figure 3.16: Reports Management Interface

Contact Messages: Figure 3.17 shows the Contact Messages interface of the AlloServiceDZ platform. This page provides administrators with an inbox view of all contact form submissions received from platform visitors, displaying sender name, email, subject, and submission date.

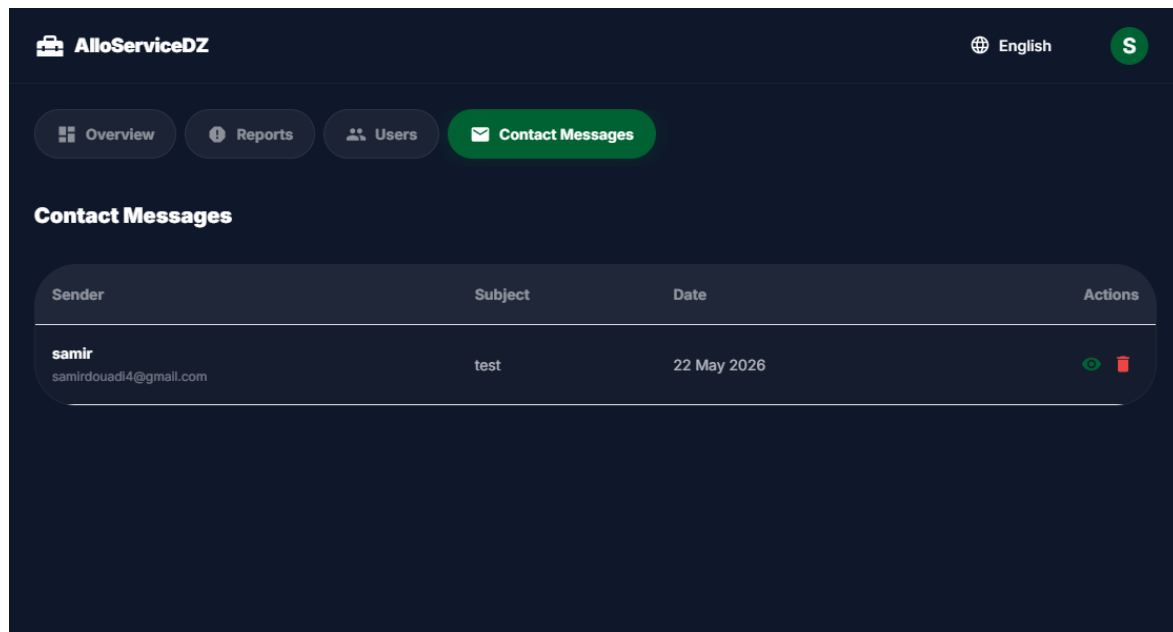


Figure 3.17: Admin Contact Messages Interface

3.4 System Deployment

This section describes the cloud infrastructure and deployment pipeline used to host and operate the AlloServiceDZ platform in a production environment. The system follows a fully serverless deployment architecture, eliminating the need for dedicated physical servers and enabling automatic scaling based on request demand.

3.4.1 Application Hosting and CI/CD Pipeline

The AlloServiceDZ frontend and backend are deployed and hosted on Netlify, a cloud platform specialized in hosting modern full-stack web applications built on serverless and static site generation paradigms [33].

Hosting Architecture: Netlify handles two distinct layers of the application simultaneously:

- **Static Frontend Delivery:** Next.js pages that are pre-rendered at build time (Static Generation) are compiled into optimized static HTML assets and distributed globally across Netlify's CDN edge network, ensuring fast initial page loads regardless of the user's geographic location.
- **Serverless API Functions:** All API Route Handlers defined under (app/api/) are automatically compiled and deployed as independent Netlify Serverless Functions. Each function is invoked on demand upon receiving an HTTP request, executes in an isolated runtime container, and terminates immediately after returning the response.

CI/CD Pipeline: The deployment pipeline is integrated directly with the project's GitHub repository, enabling a fully automated Continuous Integration and Continuous Deployment (CI/CD) workflow.

- **Trigger:** Every git push to the main branch automatically initiates a new deployment build on Netlify.
- **Build Process:** Netlify pulls the latest codebase, installs dependencies, runs the Next.js production build, and packages both static assets and serverless function bundles.
- **Atomic Deployment:** Upon successful build completion, Netlify replaces the previous deployment with the new version instantly with zero downtime.
- **Rollback Support:** Previous deployments remain accessible, allowing rollback to earlier stable versions when necessary.

3.4.2 Database Hosting

The production PostgreSQL database is hosted on Neon, a cloud-native serverless database platform designed specifically for PostgreSQL workloads in serverless environments [34].

Serverless Database Architecture: Neon separates storage from compute within its database infrastructure.

- **Persistent Storage Layer:** Database records, schemas, and relational data are stored persistently on distributed cloud storage infrastructure.
- **On-Demand Compute Layer:** Database compute resources scale dynamically according to incoming query load and suspend automatically during inactivity.

Connection Pooling: Since serverless runtimes create frequent database connections, Neon integrates a built-in PgBouncer connection pooler to optimize connection management.

- **PgBouncer:** Acts as a proxy layer between the application and PostgreSQL engine by maintaining reusable pooled database connections and distributing incoming requests efficiently.
- **Database connection:** The application connects to the pooled database endpoint through the (DATABASE_URL) environment variable.

3.5 Conclusion

This chapter presented the practical implementation and deployment aspects of the AlloServiceDZ platform. It described the development environment and technology stack adopted to build the system, including the frontend, backend, database, and supporting cloud services. The chapter also outlined the internal organization of the project codebase and highlighted the separation between frontend interfaces, backend services, and API handlers.

In addition, the platform interfaces for visitors, clients, providers, and administrators were presented to illustrate the implemented functionalities and user interactions. Finally, the deployment architecture based on Netlify and Neon was discussed, emphasizing the use of serverless hosting, automated deployment workflows, and scalable database infrastructure.

Overall, the implementation phase successfully transformed the architectural and design specifications presented in the previous chapter into a functional and deployable platform, providing a solid foundation for future improvements and feature extensions.

General Conclusion

This report presented the design, development, and deployment of AlloServiceDZ, a digital platform dedicated to connecting clients with local service providers in Algeria. The project was motivated by the growing need for accessible, organized, and trustworthy service discovery solutions capable of simplifying interactions between customers and professionals.

The study began with an analysis of the problem domain and existing solutions, highlighting the limitations of current platforms and identifying the functional and non-functional requirements necessary for the proposed system. Based on this analysis, a complete system design was developed using modern software engineering practices and architectural modeling techniques.

The implementation phase transformed these conceptual designs into a functional full-stack web application using technologies such as Next.js, TypeScript, Prisma ORM, and PostgreSQL. The platform was structured around a modular architecture supporting multiple user roles, including visitors, clients, providers, and administrators, each benefiting from dedicated functionalities and interfaces. Features such as provider discovery, booking management, messaging, review systems, and administrative moderation were successfully integrated to support the service marketplace workflow.

Furthermore, the deployment strategy relied on modern serverless infrastructure through Netlify and Neon, enabling simplified hosting, automated deployment pipelines, and scalable resource management. This infrastructure contributes to maintaining system reliability and operational efficiency while reducing deployment complexity.

Overall, the developed platform demonstrates the feasibility of building a localized digital marketplace for home services using contemporary web technologies. While the current implementation provides a solid operational foundation, several future improvements remain possible. These may include integrating secure online payment systems, enhancing recommendation and matching algorithms, introducing mobile applications, and expanding intelligent analytics capabilities to further improve user experience and platform performance.

In conclusion, AlloServiceDZ represents both a practical software engineering achievement and a promising foundation for future development toward a more connected and efficient local service ecosystem.

References

- [1] D. Samir, *Allo servicedz - platform for on-demand home services in algeria*, <https://alloservicedz.netlify.app>, 2026.
- [2] Thumbtack, *Thumbtack - find local service professionals*, <https://www.thumbtack.com>, Online service marketplace, 2026.
- [3] TaskRabbit, *Taskrabbit - on-demand local services*, <https://www.taskrabbit.com>, Home services platform, 2026.
- [4] Urban Company, *Urban company - home services platform*, <https://www.urbancompany.com>, Home service marketplace, 2026.
- [5] Ouedkniss, *Ouedkniss - classifieds and services in algeria*, <https://www.ouedkniss.com>, Algerian marketplace platform, 2026.
- [6] Bricoram, *Bricoram - the services market in algeria*, <https://bricoram.com/>, Algerian marketplace platform, 2026.
- [7] Simon Brown, *The c4 model for visualising software architecture*, <https://c4model.com/>, Accessed on may 14, 2026.
- [8] S. Brown, *The C4 Model: Visualizing Software Architecture*. O'Reilly Media, Inc., 2026.
- [9] Object Management Group, *Unified modeling language (uml)*, <https://www.omg.org/spec/UML/>, Accessed on may 14, 2026.
- [10] C. Larman, *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India, 2012.
- [11] Microsoft, *Vscode - source code editor developed by microsoft*, <https://code.visualstudio.com>, Accessed on May 23, 2026.
- [12] OpenJS Foundation, *Eslint - pluggable javascript linter*, <https://eslint.org>, Accessed on May 23, 2026.
- [13] Prettier Team, *Prettier - opinionated code formatter*, <https://prettier.io>, Accessed on May 23, 2026.
- [14] Microsoft, *Typescript intellisense in visual studio code*, <https://code.visualstudio.com/docs/editor/intellisense>, Accessed on May 23, 2026.

- [15] Software Freedom Conservancy, *Git - distributed version control system*, <https://git-scm.com>, Accessed on May 23, 2026.
- [16] GitHub Inc., *Github - code hosting and collaboration platform*, <https://github.com>, Accessed on May 23, 2026.
- [17] Postman, Inc., *Postman - api development environment*, <https://www.postman.com>, Accessed on May 23, 2026.
- [18] WHATWG, *Html - markup language standard for web documents*, <https://html.spec.whatwg.org>, Accessed on May 23, 2026.
- [19] W3C, *Css - style sheet language for web presentation*, <https://www.w3.org/Style/CSS/>, Accessed on May 23, 2026.
- [20] ECMA International, *Javascript - programming language standard for the web*, <https://developer.mozilla.org/en-US/docs/Web/JavaScript>, Accessed on May 23, 2026.
- [21] Meta, *React - a javascript library for building user interfaces*, <https://react.dev>, Accessed on May 23, 2026.
- [22] Microsoft, *Typescript - typed superset of javascript for scalable applications*, <https://www.typescriptlang.org>, Accessed on May 23, 2026.
- [23] MUI Team, *Material ui - react component library*, <https://mui.com>, Accessed on May 23, 2026.
- [24] Vladimir Agafonkin, *Leaflet - javascript library for interactive maps*, <https://leafletjs.com>, Accessed on May 23, 2026.
- [25] Vercel, *Next.js - react framework for production*, <https://nextjs.org>, Accessed on May 23, 2026.
- [26] OpenJS Foundation, *Node.js javascript runtime*, <https://nodejs.org>, Accessed on May 23, 2026.
- [27] Auth.js Team, *Nextauth.js - authentication for next.js applications*, <https://next-auth.js.org>, Accessed on May 23, 2026.
- [28] PostgreSQL Global Development Group, *Postgresql - advanced open source relational database*, <https://www.postgresql.org>, Accessed on May 23, 2026.
- [29] Prisma Data, *Prisma orm - next generation database toolkit*, <https://www.prisma.io>, Accessed on May 23, 2026.
- [30] Cloudinary, *Cloudinary - cloud media management platform*, <https://cloudinary.com>, Accessed on May 23, 2026.

- [31] Google, *Gmail smtp service for sending emails*, <https://developers.google.com/workspace/gmail/imap/imap-smtp>, Accessed on May 23, 2026.
- [32] OpenStreetMap Foundation, *Openstreetmap - open map data platform*, <https://www.openstreetmap.org>, Accessed on May 23, 2026.
- [33] Netlify, Inc., *Netlify - web development and hosting platform for modern web applications*, <https://www.netlify.com>, Accessed on May 23, 2026.
- [34] Neon, Inc., *Neon - serverless postgresql database platform*, <https://neon.tech>, Accessed on May 23, 2026.