بسم الله الرحمن الرحيم

KINGDOM OF SAUDI ARABIA

NORTHERN BORDER UNIVERSITY

FACULTY OF SCIENCES

COMPUTER SCIENCE DEPARTMENT OF

Aqua Hub: Digital Transformation of Water Services through iOS Mobile Application

A graduation project is submitted to the Computers and IT Department in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science

BY

Bander Talal Al-Anzi	202201940
Zaid Hamoud Al-Anzi	202201882
Abdulrahman Musaed Al-Anzi	202201456
Sultan Mohammed Al-Anzi	202201144
Abdulrahman Abdullah Ahmed	202205463
Abdullah Mustafa Al-Rifai	202204969

SUPERVISOR

Dr. Abdullah Alashjaee

Arar, Kingdom of Saudi Arabia

Second Semester, 2025-2026

Project Course 12350

Acknowledgments

We express our respect and sincere thanks to our supervisor, **Dr. Abdullah Alashjaee**, for honoring us with the opportunity to complete our Bachelor's degree in Computer Science. His guidance and kindness have been invaluable throughout this journey, as he patiently addressed our inquiries and supported our academic growth. He is the cornerstone of this work, and we hold him in the highest regard and deepest gratitude.

We also extend our appreciation to all the project committee members who generously devoted their time to review this work and provided valuable insights and encouragement.

A special note of thanks goes to our families—especially our beloved parents—whose unwavering support and sacrifices have made our education and well-being possible. Their love and encouragement have been a constant source of strength.

Lastly, we acknowledge and thank everyone who has supported us throughout our academic journey and contributed in any way to the success of this project.

Declaration

We hereby declare that this project, titled **"Aqua Hub: iOS Mobile Application for Water Service Management in Arar,"** is our original work and has been completed in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All sources of information used in this document have been duly cited and referenced. No portion of this work has been submitted elsewhere for any other degree or qualification.

Abstract

The **Aqua Hub iOS Application** is a mobile platform designed to streamline and digitize water service management in Arar, Saudi Arabia. The application addresses key challenges in the current system—such as inefficient order processing, limited payment options, and poor communication between users and service providers—by offering an integrated solution for ordering bottled water, scheduling tank cleaning, managing deliveries, and processing payments.

Built using Swift and SwiftUI, the application features a responsive and accessible user interface, while a PostgreSQL backend ensures secure and scalable data management. Key functionalities include user authentication, service selection, order tracking, payment integration (Apple Pay and Cash on Delivery), and feedback collection. An admin dashboard enables service providers to monitor and manage orders efficiently.

The system follows the Agile methodology, allowing iterative development and continuous stakeholder feedback. The project successfully demonstrates how mobile technology can enhance service delivery, improve customer satisfaction, and support sustainable water management in local communities.

الملخص

يعد تطبيق Aqua Hub لنظام iOS منصةً محمولة مصممة لتبسيط وإدارة خدمات المياه رقميًا في مدينة عرعر بالمملكة العربية السعودية. يعالج التطبيق التحديات الرئيسية في النظام الحالي – مثل عدم كفاءة معالجة الطلبات، وخصائص الدفع المحدودة، وضعف التواصل بين المستخدمين ومقدمي الخدمات – من خلال تقديم حل متكامل لطلب قوارير المياه، وجدولة تنظيف الخزانات، وإدارة عمليات التوصيل، ومعالجة المدفوعات.

بني التطبيق باستخدام لغة Swift وواجهة SwiftUI، ويتميز بواجهة مستخدم responsive وسهلة الوصول، بينما يضمن استخدام قاعدة بيانات PostgreSQL في الخلفية إدارة بيانات آمنة وقابلة للتوسع. تشمل الوظائف الرئيسية: المصادقة، واختيار الخدمة، وتتبع الطلبات، وتكامل أنظمة الدفع (Apple Pay والدفع نقداً عند الاستلام)، وجمع التقييمات. كما تتيح لوحة تحكم المسؤول لمقدمي الخدمة مراقبة وإدارة الطلبات بكفاءة.

يتبع النظام منهجية التطوير الرشيقة (Agile)، مما يسمح بالتطوير التكراري وتلقي ملاحظات مستمرة من أصحاب المصلحة. يثبت هذا المشروع بنجاح كيف يمكن للتكنولوجيا المحمولة أن تعزز تقديم الخدمة، وتحسن رضا العملاء، وتدعم الإدارة المستدامة للمياه في المجتمعات المحلية.

Contents

Acknowledgments 1

Declaration 2

Abstract 3

List of Tables 6

List of Figures 6

List of Acronyms and Abbreviations 7

Chapter 1: Planning Phase 8

1.1 Introduction 9

1.2 Overview of the Project 9

1.3 Importance of the Project 9

1.4 Current Business Description 10

1.5 Current System Difficulties 10

1.6 Problem Definition 10

1.6.1 Problem Statement 10

1.6.2 List of Problems Faced by the Current System 11

1.7 Proposed System 11

1.7.1 Scope of the System 11

1.7.2 Advantages of the System 11

1.7.3 System Constraints 12

1.8 Schedule Plan 12

1.8.1 Gantt Chart 12

1.8.2 PERT Chart 13

Chapter 2: Analysis Phase 14

2.1 Introduction to Analysis 15

2.2 System Methodology Used 16

2.3 System Requirements 17

2.3.1 Hardware Requirements 17

2.3.2 Development Environment 17

2.3.3 Non-Functional Requirements 19

2.4 Modeling 20

2.4.1 Process Modeling 20
2.4.2 Data Flow Diagrams (DFD) 21
2.4.3 Context Diagram 22
2.4.3.1 Level 0 Diagram 23
2.4.3.2 Level 1 Diagram 24
2.4.3.3 Level 2 Diagram 26
2.4.4 Use Case Diagrams 27
2.4.5 Sequence Diagrams 29
2.4.6 Activity Diagrams 31
Chapter 3 Design Phase 33
3.1.2 Class Diagram 40
3.1.3 Database Schema 44
Chapter 4 Implementation Phase 50
4.1 System Deployment 51
4.1.1 Development and Testing Environment 51
4.1.2 Production Deployment Plan 52
4.2 Prototype Implementation 53
4.2.1 Front-end Implementation with SwiftUI 53
4.2.2 Back-end Integration Logic 56
4.2.3 Functional Prototype Snapshots 56
4.3 System Maintenance and Support 60
4.3.1 Technical Support and User Guidance 60
4.3.2 Backup and Recovery Strategy 61
4.3.3 Update and Versioning Plan 61
5.1 Conclusion 64
5.2 Future Work 64
References 66
Appendices 66

Keywords: iOS, Swift, SwiftUI, PostgreSQL, Water Service Management, Agile, Mobile Application, Arar.

List of Tables

Table Name	Page Number
Table 1: Comparison Between Agile and Waterfall Methodologies	18
Table 2: Development Environment Tools and Technologies for Aqua Hub	19
Table 3: Use Case Descriptions for Aqua Hub System	30
Table 3.1: users	46
Table 3.2: services	46
Table 3.3: orders	47
Table 3.4: payments	48
Table 3.5: deliveries	48
Table 3.6: feedback	49

List of Figures

Figure Name	Page Number
Figure 1: Process Modeling for Aqua Hub Application	22
Figure 2: DFD Level 0 (Context Diagram) for Aqua Hub	25
Figure 3: Level 1 Data Flow Diagram for Aqua Hub	26
Figure 4: Level 2 Data Flow Diagram for Aqua Hub	28
Figure 5: Use Case Diagram for Aqua Hub	29
Figure 6: Sequence Diagram for Aqua Hub (Order Processing Scenario)	31
Figure 7: Activity Diagram for Tank Cleaning Process	33
Figure 3.1: Entity Relationship Diagram for Aqua Hub iOS Application	40
Figure 3.2: UML Class Diagram for Aqua Hub iOS Application	45
Figure 4.1: Aqua Hub Sign-In Screen (Prototype)	55
Figure 4.2: Home Services Screen with Integrated Options	56

Figure 4.3: In-screen Service Option Selection (Bottom Sheet)	59
Figure 4.4: Admin Dashboard for Managing Orders	60
Figure C.1: Login Screen (User Authentication)	In Appendix C
Figure C.2: Home Screen (Service Overview)	In Appendix C
Figure C.3: Service Details (Order Placement)	In Appendix C
Figure C.4: Payment Screen (Apple Pay Integration)	In Appendix C
Figure C.5: Feedback Screen (Rate and Comment)	In Appendix C
Figure C.6: Admin Dashboard (Manager Control Panel)	In Appendix C
Figure C.5 (a–e): Service Option Selection Screens	In Appendix C

List of Acronyms and Abbreviations

- **SDLC** – Software Development Life Cycle
- **DFD** – Data Flow Diagram
- **ERD** – Entity Relationship Diagram
- **UML** – Unified Modeling Language
- **API** – Application Programming Interface
- **APNs** – Apple Push Notification Service
- **FCM** – Firebase Cloud Messaging
- **COD** – Cash on Delivery
- **HIG** – Human Interface Guidelines
- **RTL** – Right-to-Left
- **SQL** – Structured Query Language
- **UI** – User Interface
- **UX** – User Experience
- **UUID** – Universally Unique Identifier
- **SPM** – Swift Package Manager
- **GCP** – Google Cloud Platform
- **AWS** – Amazon Web Services
- **2FA** – Two-Factor Authentication

Chapter 1: Planning Phase

1.1 Introduction

The planning phase represents the foundational stage in the development of the **Aqua Hub iOS mobile application**, where the project's vision, objectives, scope, and feasibility are formally established. This phase sets the strategic direction for the entire Software Development Life Cycle (SDLC), ensuring alignment with user needs, business goals, and technical capabilities. For Aqua Hub, this phase involved a comprehensive assessment of water service challenges in the Arar community, identifying digital transformation opportunities through a dedicated iOS platform built using Swift and SwiftUI.

1.2 Overview of the Project

Aqua Hub is a native iOS application designed to modernize and streamline water-related services in Arar, Saudi Arabia. The application enables users to order bottled water, schedule tank cleaning, request maintenance services, and manage payments seamlessly. It integrates a cloud-hosted PostgreSQL backend to support real-time order tracking, secure payment processing (via Apple Pay and Cash on Delivery), and efficient service delivery. The system is structured around key entities—User, Service, Order, Payment, Delivery, and Feedback—as defined in the Entity Relationship Diagram (ERD) and implemented via the SQL schema in Appendix A.

1.3 Importance of the Project

The Aqua Hub project addresses critical inefficiencies in the current water service model, including limited access to clean water, inconsistent scheduling, and a lack of digital management tools. By providing a unified mobile platform, Aqua Hub improves service accessibility, ensures timely deliveries, and enhances customer satisfaction through features such as real-time notifications, service rescheduling, and feedback collection. The project also supports community well-being by promoting reliable access to safe drinking water and hygienic tank maintenance, aligning with broader public health and quality-of-life objectives.

1.4 Current Business Description

The existing water service ecosystem in Arar relies heavily on manual, non-integrated processes. Service requests are typically made via phone calls, leading to inconsistent record-keeping, communication gaps, and operational delays. Delivery companies and maintenance teams lack a centralized system to coordinate schedules, track orders, or monitor service quality. Payment collection is often handled in cash, with limited transparency and security. There is no standardized mechanism for customer feedback or service evaluation, hindering continuous improvement.

1.5 Current System Difficulties

- **Inefficient Order Management:** Manual processing of service requests leads to errors, delays, and poor customer experience.
- **Limited Communication Channels:** Absence of real-time updates between users, administrators, and delivery personnel.
- **No Centralized Tracking:** Inability to monitor order status, delivery progress, or service history in a unified manner.
- **Inflexible Payment Options:** Reliance on cash-based transactions with no support for digital or secure payment methods.
- **Lack of Feedback Mechanism:** No structured process for collecting or analyzing user feedback to improve service quality.

1.6 Problem Definition

1.6.1 Problem Statement

The absence of an integrated digital system for water service management in Arar results in operational inefficiencies, poor resource utilization, and low customer satisfaction. There is a clear need for a mobile application that automates service requests, scheduling, payments, and feedback collection while ensuring reliability, security, and ease of use.

1.6.2 List of Problems Faced by the Current System

- Manual and error-prone order processing.
- Ineffective coordination between customers, admins, and delivery staff.
- No real-time notification system for order status or schedule changes.
- Limited and insecure payment options.
- Inability to collect, analyze, or act on customer feedback.
- Lack of a centralized admin dashboard for service monitoring and management.

1.7 Proposed System

1.7.1 Scope of the System

The Aqua Hub iOS application will encompass the following modules and functionalities:

- **User Management:** Registration, authentication, and profile management for customers, admins, and delivery staff.
- **Service Catalog:** Dynamic listing of services including Safi Water, Drinking/Non-Drinking Water Tanks, Tank Cleaning, and HydroFix Maintenance.
- **Order Management:** End-to-order order placement, scheduling, rescheduling, and status tracking.
- **Payment Processing:** Secure transactions via Apple Pay and Cash on Delivery, integrated with the payments table.
- **Delivery Tracking:** Real-time updates and assignment of delivery staff via the deliveries table.
- **Feedback System:** Rating and review submission linked to completed orders.
- **Admin Dashboard:** Comprehensive interface for order management, user oversight, and service analytics (as implemented in Appendix E).

1.7.2 Advantages of the System

- **Efficiency:** Automated workflows reduce manual effort and processing time.

- **Transparency:** Real-time notifications and status updates keep all stakeholders informed.
- **Security:** Secure authentication, encrypted payments, and data integrity via PostgreSQL and iOS Keychain.
- **Scalability:** Modular architecture supports future expansion of services and user base.
- **User Satisfaction:** Intuitive SwiftUI interface, flexible payment options, and feedback mechanisms enhance the customer experience.

1.7.3 System Constraints

- **Platform Compatibility:** Limited to iOS 15+ devices (iPhone and iPad).
- **Network Dependency:** Requires a stable internet connection for core functionalities such as ordering and payments.
- **Third-Party Services:** Reliance on Apple Pay, APNs, and cloud infrastructure (e.g., AWS, GCP).
- **Geographic Limitation:** Initial rollout restricted to the Arar region.
- **Regulatory Compliance:** Must adhere to Apple App Store Review Guidelines, particularly for payment and data privacy.

1.8 Schedule Plan

1.8.1 Gantt Chart

A Gantt chart will outline the project timeline, highlighting key phases and milestones:

- **Weeks 1-2:** Planning & Requirement Analysis
- **Weeks 3-4:** System Design (ERD, Class Diagram, UI Mockups)
- **Weeks 5-8:** Implementation (SwiftUI Frontend, PostgreSQL Backend)
- **Weeks 9-10:** Testing (Unit, UI, Integration)
- **Week 11:** Deployment (App Store Submission, Database Setup)
- **Week 12:** Post-Deployment Review & Documentation

1.8.2 PERT Chart

A PERT chart will visualize task dependencies and critical paths, ensuring efficient resource allocation and timely project completion. Key dependencies include:

- Completion of UI mockups (Appendix C) before SwiftUI implementation (Appendix D).
- Finalization of database schema (Appendix A) before backend integration.
- Successful testing phases prior to App Store deployment.

Chapter 2: Analysis Phase

2.1 Introduction to Analysis

The analysis phase is a critical stage in the Software Development Life Cycle (SDLC) as it establishes the foundation for building a system that aligns with user expectations and business goals. Its primary purpose is to investigate existing challenges capture user needs and translate them into well-defined system requirements. A structured analysis reduces the risks of misinterpretation prevents unnecessary scope changes and minimizes costly revisions during later phases of development.

Unlike the design and implementation stages which focus on technical solutions the analysis phase emphasizes what the system must achieve. It involves direct interaction with stakeholders to gather requirements define objectives and highlight possible constraints. The deliverables of this phase often include requirement specifications process descriptions and system models all of which serve as reference points for the design stage.

For the Aqua Hub iOS mobile application this phase is especially significant due to the challenges facing water service delivery in Arar Saudi Arabia. Current issues such as limited access to high-quality water inefficiencies in scheduling tank cleaning and increasing demand for customizable bottled water services are carefully studied. By analyzing these problems Aqua Hub can be structured to deliver effective solutions including advanced order management flexible rescheduling timely notifications and water quality assurance.

Through a comprehensive analysis Aqua Hub will not only meet functional requirements but also provide a reliable efficient and community-oriented iOS solution. This ensures that the application contributes to improving the quality of water services while addressing both individual and community needs.

2.2 System Methodology Used

For the Aqua Hub iOS application, the chosen methodology is the Agile approach. Agile provides the flexibility and adaptability required for modern mobile applications, where features evolve continuously based on user needs and real-time feedback. The iterative nature of Agile allows the development team to introduce core functionalities first, such as placing orders and scheduling services, and then gradually expand the system with additional features like tank cleaning services, notifications, payment management, and maintenance tools. This incremental development ensures that the system is functional at every stage and can deliver value to users even before the full set of features is completed.

Moreover, Agile facilitates close collaboration with stakeholders and encourages frequent updates, making it possible to respond quickly to new requirements or challenges. This is critical for Aqua Hub since the application serves a wide range of community needs in Arar, including households, schools, businesses, and mosques. Iterations will also be distributed via TestFlight to collect rapid user feedback from iPhone devices, informing subsequent sprints and ensuring that user expectations are met effectively.

The Waterfall methodology, on the other hand, follows a linear and highly structured process where each phase (requirements, design, implementation, testing, and deployment) must be completed before the next one begins. While this approach provides clarity, discipline, and comprehensive documentation, it lacks flexibility when unexpected changes occur. In the context of Aqua Hub, where service demands may change over time, Waterfall would be less efficient and potentially delay delivery.

Therefore, Agile is considered the most suitable methodology for developing Aqua Hub on iOS, ensuring adaptability, faster delivery, and higher user satisfaction.

Criteria	Agile Methodology	Waterfall Methodology
Flexibility	Highly flexible adapts to changes during development.	Rigid changes are difficult once a phase is completed.
Development Process	Iterative features added in increments (sprints).	Sequential each phase must be completed before the next.
Stakeholder Involvement	Continuous feedback and collaboration throughout.	Limited to requirement gathering and final review.
Delivery Speed	Early and incremental delivery of working features.	Final delivery at the end of the process.
Risk Management	Risks identified early through iterative feedback.	Risks discovered late often after significant investment.

Table 1: Comparison Between Agile and Waterfall Methodologies

2.3 System Requirements

2.3.1 Hardware Requirements

End-user devices include iPhone models running iOS 15 or later (with optional iPad support). A cloud-hosted backend server handles APIs authentication scheduling and notifications. A stable internet connection is required for ordering payments and real-time updates.

2.3.2 Development Environment

The iOS development stack uses Xcode and the iOS SDK with Swift and SwiftUI for building native interfaces. Dependencies are managed via the Swift Package Manager (SPM) or CocoaPods. Data persistence leverages Core Data with optional support for SQLite or Realm while sensitive credentials are

securely stored in the Keychain. Push notifications are delivered through Apple Push Notification service (APNs) (optionally integrated with Firebase Cloud Messaging). Payments are processed via Apple Pay in compliance with App Store policies. Quality assurance is ensured through XCTest and XCUITest while distribution flows through TestFlight for beta testing and the App Store for public release.

In addition to these tools diagramming and modeling are supported using Draw.io and StarUML while GitHub and GitLab provide version control and collaborative workflows. Cloud services such as AWS Google Cloud Platform (GCP) or Microsoft Azure ensure backend scalability and reliability.

Category	Tools / Technologies	Purpose
Language & UI	Swift SwiftUI (UIKit if needed)	Native iOS development & UI
IDE & SDK	Xcode iOS SDK	Coding debugging simulators
Dependencies	Swift Package Manager (SPM) CocoaPods	Manage libraries
Storage	Core Data SQLite Realm	Local data persistence
Security	Keychain	Secure token/credentials storage
Networking	URLSession (or Alamofire)	API communication
Notifications	APNs (optionally FCM → APNs)	Push notifications
Payments	Apple Pay (or compliant gateways)	Payment authorization
Maps/Location	Core Location MapKit	Geo & routing features
Testing	XCTest XCUITest	Unit/UI testing
Distribution	TestFlight App Store Connect	Beta testing & publishing

Table 2: Development Environment Tools and Technologies for Aqua Hub

2.3.3 Non-Functional Requirements

In addition to functional requirements the Aqua Hub iOS application must also satisfy a set of non-functional requirements to ensure reliability usability and compliance with Apple ecosystem standards. These requirements include the following:

- Usability and Accessibility: The application must adhere to Apple's Human Interface Guidelines (HIG) supporting Dynamic Type Dark Mode and Right-to-Left (RTL) layouts for languages such as Arabic. Accessibility features such as VoiceOver and screen reader compatibility must also be enabled.
- Performance: The app should maintain smooth responsiveness across iPhone models running iOS 15 or later. Background processes (such as notifications and scheduling) must minimize battery consumption and memory usage.
- Security and Privacy: All sensitive user information must be securely stored using the iOS Keychain. Network communication should be encrypted via HTTPS (TLS 1.2 or higher). Clear and transparent privacy usage descriptions must be declared in the Info.plist file for all permissions (e.g. notifications location services).
- Scalability: The backend infrastructure must be able to handle increasing numbers of user requests without degrading system performance. Cloud hosting and database indexing will support horizontal and vertical scaling.
- Reliability and Availability: The app must demonstrate high availability with minimal downtime. Push notifications via APNs must deliver alerts reliably to all registered devices.

- Compliance with App Store Guidelines: The Aqua Hub application must align with Apple's App Store Review Guidelines particularly regarding payment integration (Apple Pay) and data privacy. Non-compliance may lead to rejection during the app submission process.

2.4 Modeling

2.4.1 Process Modeling

Process modeling is used to represent the main processes of the Aqua Hub application in a structured and visual way. It provides a high-level overview of how data flows through the system and how different components interact with each other.

The process begins when the user logs in to the application. After authentication the user selects the desired service such as ordering bottled water scheduling tank cleaning or requesting maintenance. The request is then processed by the system where validation and scheduling take place. Once the request is confirmed a delivery or service is scheduled and the necessary stakeholders are notified. Finally the system sends a confirmation and notification back to the user.

This modeling approach ensures smooth coordination between the user the backend system and the service providers. A process diagram is used to visualize these interactions clearly helping stakeholders understand the workflow and dependencies.

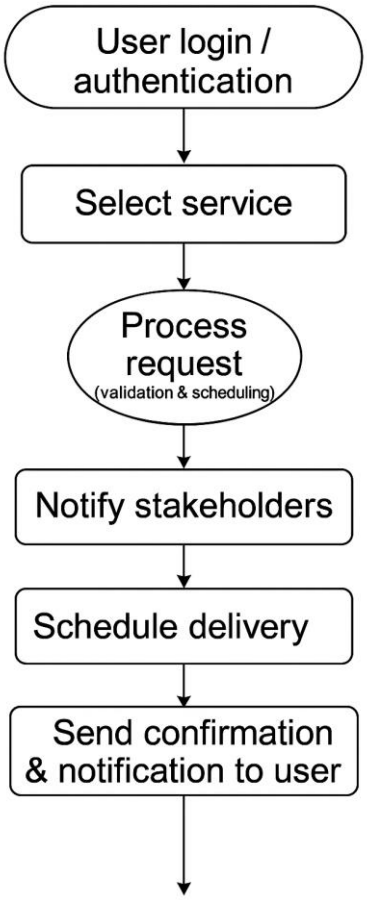

Figure 1: Process Modeling for Aqua Hub Application

2.4.2 Data Flow Diagrams (DFD)

Data Flow Diagrams (DFDs) are a fundamental modeling technique used to illustrate how information flows through a system. They show the interactions between inputs processes outputs and data stores focusing on the movement of data rather than implementation details. This abstraction allows developers and stakeholders to clearly understand how information is processed and transferred within the system.

For the Aqua Hub iOS application DFDs are particularly valuable because they demonstrate how service requests—such as water delivery or tank cleaning—are initiated by users processed by the system and routed to external entities such as the delivery company and payment gateway. The diagrams also highlight how

confirmations and notifications are returned to users and how administrators interact with the system to monitor and manage operations.

By documenting these flows DFDs help identify potential inefficiencies clarify data responsibilities and ensure that all interactions are clearly defined before transitioning to the system design and implementation phase.

2.4.3 Context Diagram

The Context Diagram provides a high-level view of the Aqua Hub application as a single process interacting with its external environment. It defines the system's boundaries and shows how stakeholders exchange information with the application without detailing its internal logic. This makes it an effective tool for illustrating the scope of the system and clarifying its relationships with external actors.

In the case of Aqua Hub the system is represented as a central process labeled **Aqua Hub System** The following external entities surround it:

- **User**: Submits service requests (e.g. bottled water delivery tank cleaning or maintenance) and receives confirmations and notifications.
- **Delivery Company**: Receives delivery details from the system and provides status updates for pending or completed services.
- **Payment Gateway**: Supports Apple Pay for secure digital transactions and also provides Cash on Delivery (COD) as an alternative payment method ensuring flexibility and compliance with iOS platform policies.
- **Admin**: Oversees system management service availability transaction monitoring and reporting for decision-making.

This simplified yet powerful model helps developers administrators and stakeholders understand where the application connects with external parties and how data flows in these interactions.

2.4.3.1 Level 0 Diagram

The Level 0 Data Flow Diagram (Context Diagram) presents Aqua Hub as a single process and shows the main external entities that interact with the system. At this high level the focus is on the flow of information into and out of the system without revealing internal processing details. The primary input is the user's service request (e.g. bottled water orders tank cleaning or maintenance). The system responds with outputs such as confirmations and notifications. Additional interactions include payment authorization with the Payment Gateway and delivery status updates from the Delivery Company while the admin entity provides management inputs and receives monitoring information.

For the iOS migration notifications are delivered using Apple Push Notification Service (APNs) instead of Firebase and secure payments may be integrated with Apple Pay in compliance with App Store Review Guidelines. These platform-specific adjustments ensure that the context diagram remains valid while reflecting the new iOS ecosystem.

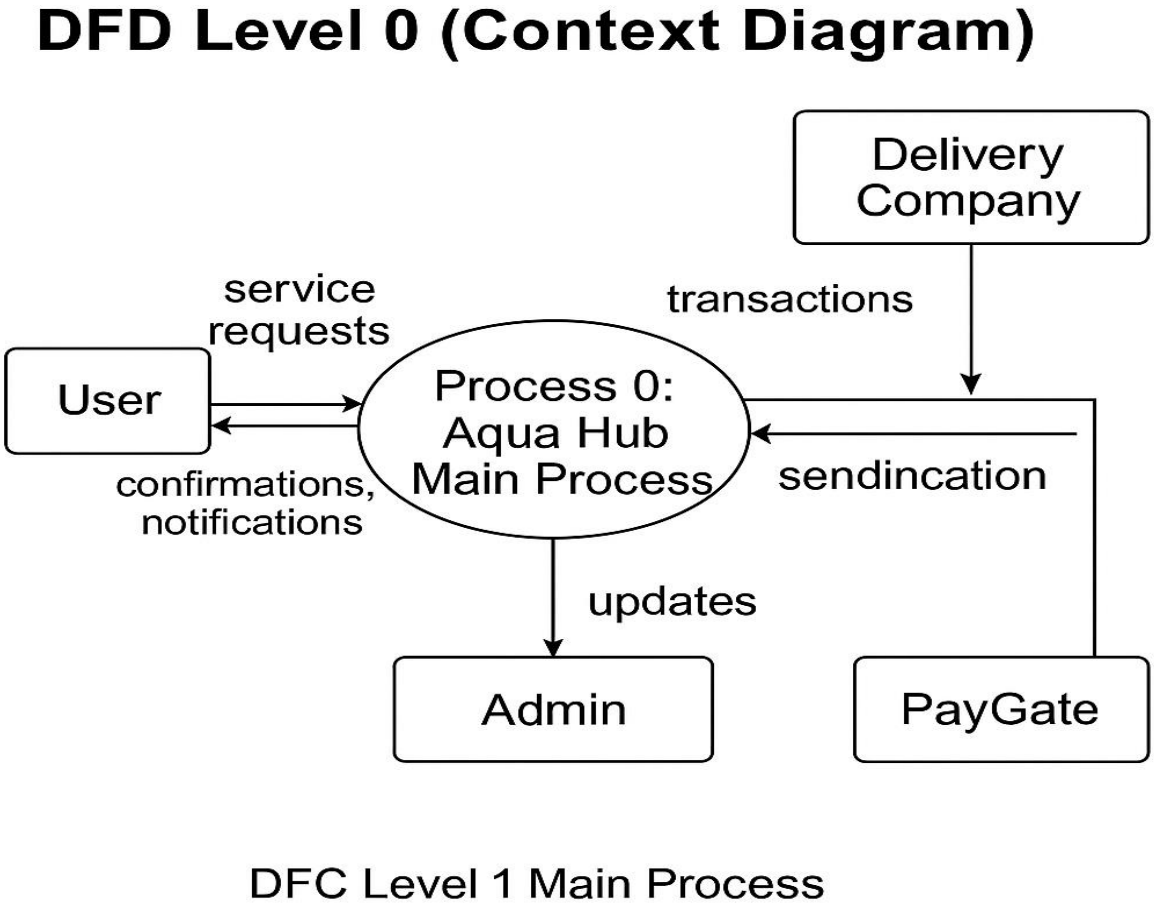

Figure 2: DFD Level 0 (Context Diagram) for Aqua Hub

2.4.3.2 Level 1 Diagram

The Level 1 Data Flow Diagram (DFD) provides a more detailed breakdown of the Aqua Hub system by dividing the main process into several sub-processes. Unlike the Level 0 diagram which shows the system as a single entity the Level 1 diagram highlights the internal operations and how they interact with external entities. This level allows stakeholders to better understand the system's logic and the pathways through which requests are processed.

For Aqua Hub the Level 1 DFD includes the following key processes:

- **Order Management:** Handles requests for bottled water delivery. It validates the order checks availability and schedules delivery.
- **Cleaning Scheduling:** Manages requests for tank cleaning or maintenance services. It checks schedules assigns tasks and provides reminders.
- **Payment Processing:** Coordinates with the payment gateway to authorize and confirm transactions securely.
- **Notification Management:** Sends timely alerts and confirmations to users regarding their requests payments and scheduled services.

External entities such as the User Admin Delivery Company and Payment Gateway remain connected to these processes ensuring smooth flow of information across the system.

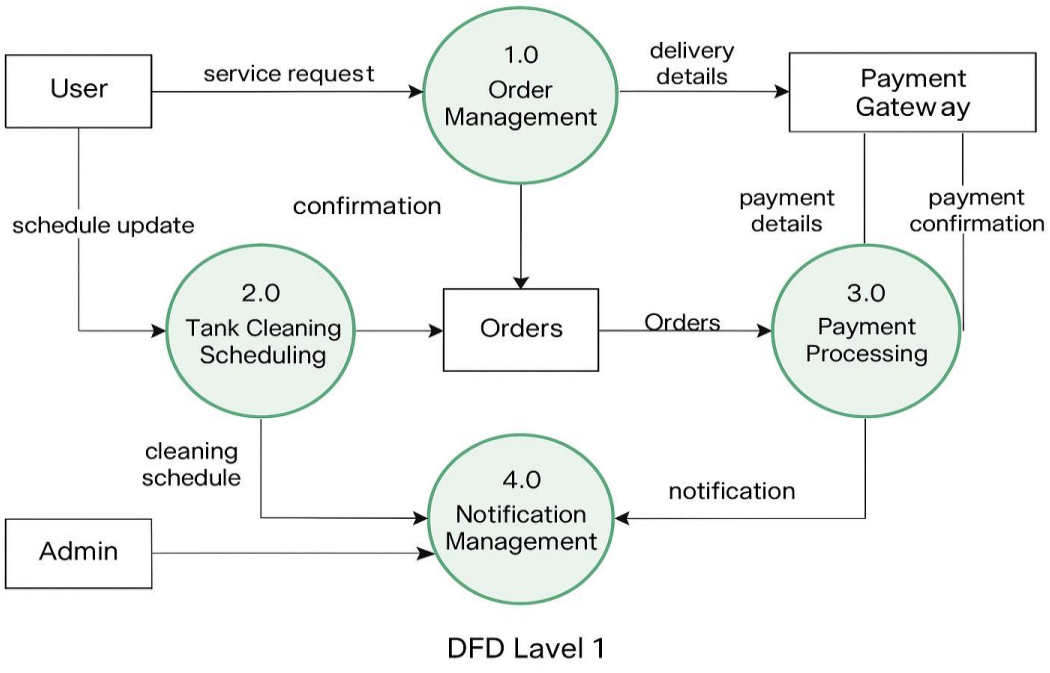

Figure 3: Level 1 Data Flow Diagram for Aqua Hub

As shown in Figure 3 the Level 1 DFD decomposes the Aqua Hub System into four key processes: Order Management Cleaning Scheduling Payment Processing and Notification Management.

2.4.3.3 Level 2 Diagram

The Level 2 Data Flow Diagram (DFD) provides a more detailed view of a single process within the Aqua Hub system. While the Level 1 diagram introduced the main sub-processes the Level 2 diagram expands on one of them to show the internal logic and how specific tasks are executed.

For Aqua Hub the Order Management process is decomposed into the following activities:

- **Authenticate:** The system verifies the user's login credentials to ensure secure access.
- **Manage Request:** After authentication the system processes the user's request (such as water delivery) by validating the details and updating relevant records.
- **User Data Storage (D1):** All request details and user information are securely stored in the User Data repository.
- **Schedule Management (D2):** Validated requests are sent to the schedule repository which is later accessed by delivery personnel for task execution.
- **Delivery Personnel:** Receives the delivery schedule and performs the requested service.

The Level 2 DFD ensures that stakeholders understand the step-by-step internal flow of how a critical process—Order Management—is carried out. This level of detail minimizes ambiguity reduces potential errors and clarifies the interactions between users the system and external delivery staff.

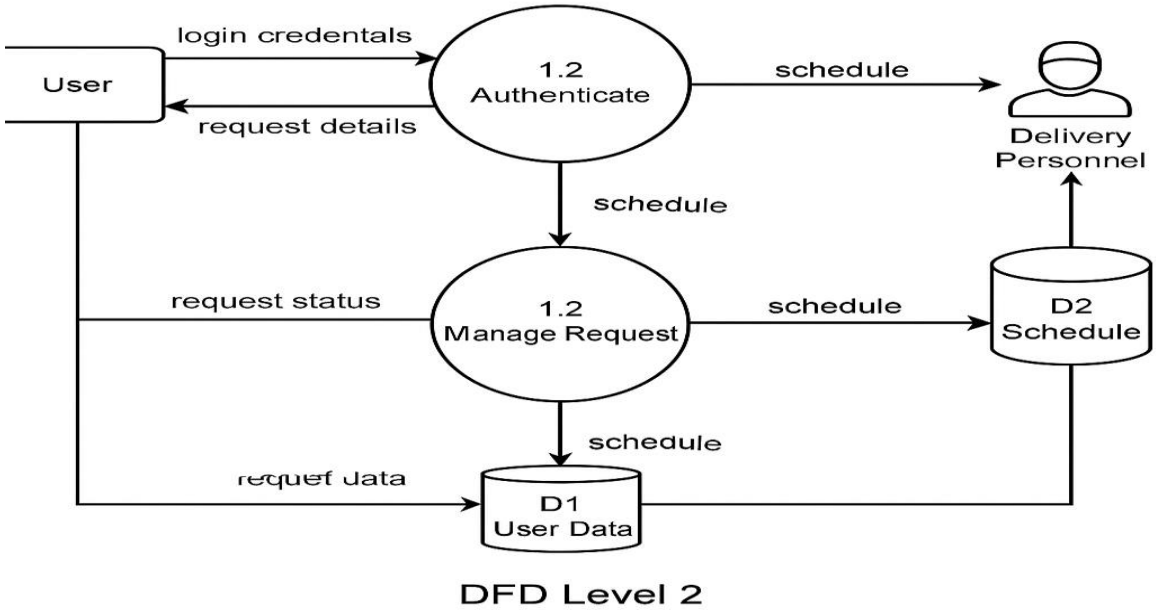

Figure 4: Level 2 Data Flow Diagram for Aqua Hub

2.4.4 Use Case Diagrams

The Use Case Diagram illustrates how different actors interact with the Aqua Hub system. It defines the major functionalities from the user's perspective and identifies which roles (User Admin Delivery Staff) are responsible for each action. This ensures that all behavioral requirements of the application are clearly documented.

For iOS migration the User receives notifications via Apple Push Notification Service (APNs) and payments are processed securely through Apple Pay ensuring compliance with iOS platform guidelines.

The diagram in Figure 5 visualizes these interactions while Table 3 provides a descriptive overview of each use case ensuring both graphical and textual clarity for the Aqua Hub system on iOS.

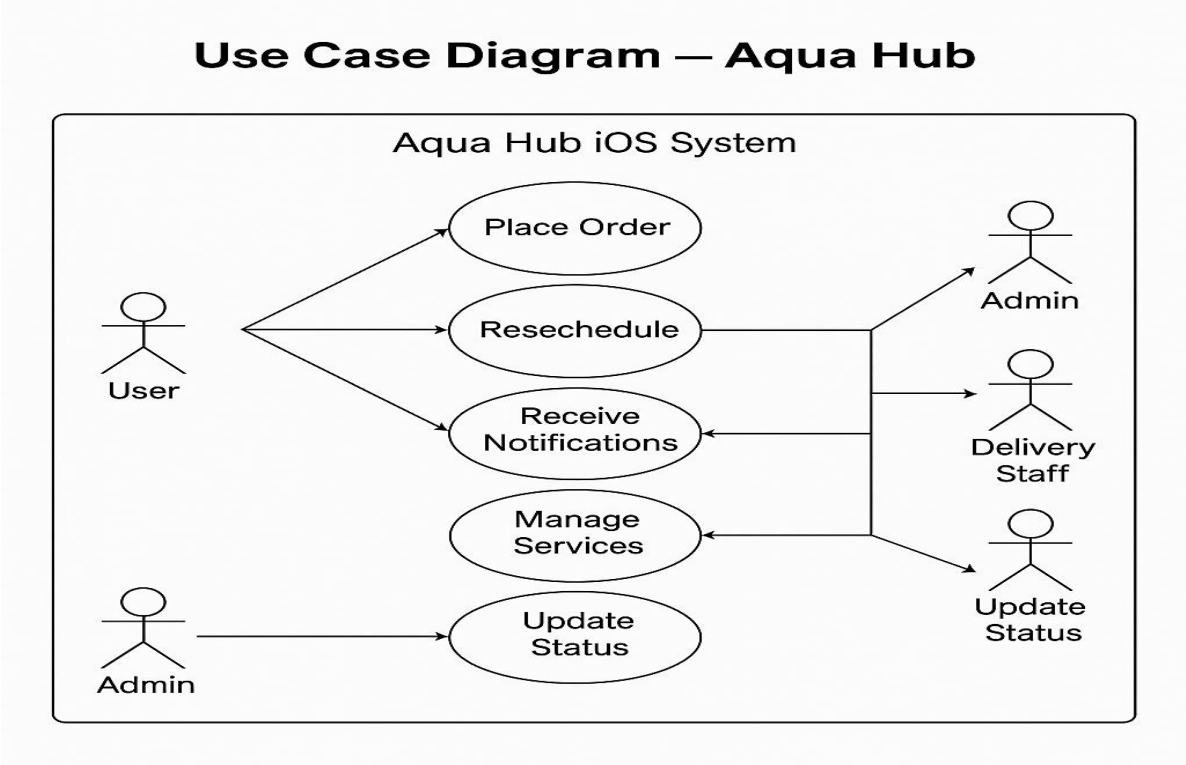

Figure 5: Use Case Diagram for Aqua Hub

Actor	Use Case	Description
User	Place Order	User submits a request for bottled water delivery or a related service.
User	Reschedule Service	User modifies the date/time of an existing order or cleaning request.
User	Receive Notifications	User receives alerts and confirmations about orders payments and schedules.

Admin	Manage Services	Admin adds updates or removes services (delivery cleaning maintenance).
Admin	Manage Users	Admin manages user accounts and access (creation updates activation).
Delivery Staff	Confirm Delivery	Delivery staff marks the assigned order as delivered/completed.
Delivery Staff	Update Status	Delivery staff updates the order/cleaning job status (e.g. assigned in-progress delayed).

Table 3: Use Case Descriptions for Aqua Hub System

2.4.5 Sequence Diagrams

2.4.5 Sequence Diagrams

The Sequence Diagram describes the time-ordered interaction among participants involved in a specific scenario. For Aqua Hub a representative scenario is the order lifecycle: the user places an order the system processes the request the delivery service confirms the delivery and the user receives a notification. This model clarifies the order of messages the responsibilities of each participant and the synchronous nature of the interactions.

Participants in this sequence include the User the Aqua Hub System and the Delivery Service.

The main steps are:

1. The User places an order through the Aqua Hub (iOS).

2. The System validates the request and processes the order.
3. The System forwards the job to the Delivery Service for fulfillment and receives a confirmation.
4. The System sends a final notification to the User indicating the delivery status.

For the iOS migration push notifications are delivered via Apple Push Notification Service (APNs) and payment authorization is securely handled through Apple Pay in compliance with App Store Review Guidelines. This guarantees compliance with iOS standards while maintaining the expected flow of order placement processing delivery confirmation and notification.

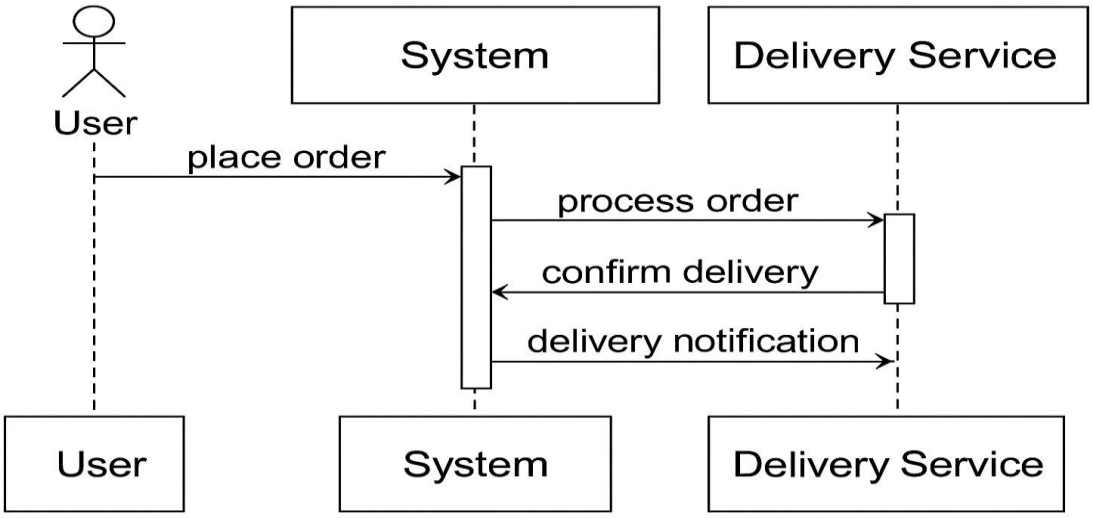

Figure 6: Sequence Diagram for Aqua Hub (Order Processing Scenario)

2.4.6 Activity Diagrams

The Activity Diagram models the workflow of a process by describing the sequence of activities decisions and outcomes. For Aqua Hub a representative workflow is the *Tank Cleaning Request* process. The flow begins when a user submits a request which is then subject to approval. If the request is rejected the process ends. If approved the system proceeds to schedule the cleaning followed by the actual service execution and finally a notification is sent to the user.

Main steps depicted in the diagram:

1. Request tank cleaning.
2. Approve request (Yes/No decision).
3. If approved → Schedule cleaning.
4. Perform cleaning.
5. Send notification to the user.
6. End process.

This representation ensures that both positive and negative outcomes are considered supporting process optimization and clarifying responsibilities at each stage.

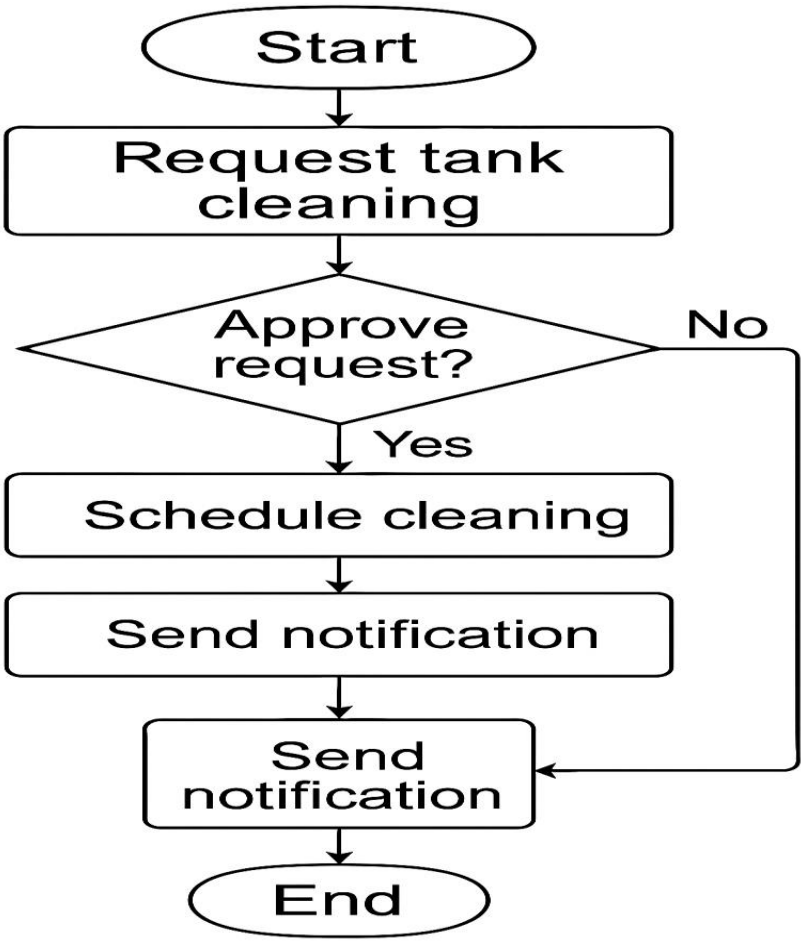

Figure 7: Activity Diagram for Tank Cleaning Process

Chapter 3 Design Phase

This chapter presents the design phase of the Aqua Hub iOS Application, where the system's structure and components are defined in detail. The main objective of this phase is to describe how the application is organized and how its features are designed to meet both functional and non-functional requirements.

The design process focuses on transforming the identified needs into practical, efficient, and user-friendly solutions. It defines how the data is modeled, how the interfaces interact, and how the overall system maintains consistency and reliability.

This chapter is divided into two main parts: the Conceptual Data Modeling, which defines the structure of the database and its entities, and the System Design, which covers menus, forms, and reports. Together, these sections establish a solid foundation for the implementation phase and ensure that the Aqua Hub iOS Application provides a seamless and effective user experience. Entity Identification

To construct an effective ERD for the Aqua Hub iOS Application it is essential to first identify the key entities that represent the system's core data components. Each entity corresponds to a distinct concept or object within the operational domain of Aqua Hub ranging from users placing water delivery orders to administrators managing service availability.

The following entities were derived from the requirements and process models:

User Entity

User_ID (Primary Key)
Full_Name
Email
Phone_Number
Password_Hash
User_Role (Customer / Admin / Delivery_Staff)
Address

Service Entity

Service_ID (Primary Key)
Service_Name

Description

Service_Type (Delivery / Cleaning / Maintenance)

Price

Availability_Status

Order Entity

Order_ID (Primary Key)

Order_Date

Delivery_Date

Quantity

Total_Amount

Status (Pending / Approved / Completed / Cancelled)

User_ID (Foreign Key → User)

Service_ID (Foreign Key → Service)

Payment Entity

Payment_ID (Primary Key)

Payment_Date

Payment_Method (Apple_Pay / Cash_On_Delivery)

Transaction_Status (Successful / Failed / Pending)

Amount

Order_ID (Foreign Key → Order)

Delivery Entity

Delivery_ID (Primary Key)

Assigned_Staff_ID (Foreign Key → User)

Order_ID (Foreign Key → Order)

Dispatch_Time

Completion_Time

Delivery_Status

Feedback Entity

Feedback_ID (Primary Key)

Rating (1–5)

Comment

Submission_Date

User_ID (Foreign Key → User)

Order_ID (Foreign Key → Order)

Together these six entities provide a complete and normalized data foundation that reflects the business logic of Aqua Hub's ecosystem. Their interconnections—especially the one-to-many relationships between User–Order Order–Payment and Order–Delivery—enable efficient data retrieval and streamlined backend processing.

Relationship Definition and Cardinality

Establishing relationships and cardinalities between entities is a fundamental step in developing a coherent Entity Relationship Diagram (ERD). These relationships define how data elements in one entity are associated with those in another ensuring logical consistency and referential integrity across the database of the Aqua Hub iOS Application. The following relationships were identified based on the functional requirements and workflows described in the analysis phase:

1. User – Order Relationship (1–M)

A single user can place multiple orders but each order is associated with exactly one user. This *one-to-many* relationship ensures that users' historical and active orders can be efficiently tracked retrieved and analyzed for service personalization.

Relationship:

User (1) — (M) Order

Type: One-to-Many

2. User – Feedback Relationship (1–M)

Each user may submit multiple feedback entries for different services or orders while each feedback record belongs to one user only.

This allows the system to collect user insights across multiple transactions for performance monitoring and quality improvement.

Relationship:
User (1) — (M) Feedback
Type: One-to-Many

3. Service – Order Relationship (1–M)

A single service (e.g. Water Delivery or Tank Cleaning) can be requested in multiple orders but each order corresponds to one specific service type.
This relationship facilitates scalable service management and supports dynamic pricing or scheduling models.

Relationship:
Service (1) — (M) Order
Type: One-to-Many

4. Order – Payment Relationship (1–1)

Each order has exactly one payment record and each payment corresponds to a single order. This *one-to-one* relationship ensures financial traceability and enforces transactional consistency in compliance with Apple Pay policies.

Relationship:
Order (1) — (1) Payment
Type: One-to-One

5. Order – Delivery Relationship (1–1)

Every confirmed order is linked to one delivery process and each delivery record refers back to a specific order.
This maintains a clear operational connection between order fulfillment and delivery logistics.

Relationship:

Order (1) — (1) Delivery

Type: One-to-One

6. Delivery Staff (User) – Delivery Relationship (1–M)

Each delivery staff member (classified under the User entity with the Delivery_Staff role) may handle multiple deliveries but each delivery is assigned to only one staff member. This role-based relationship supports accountability and workload management within the application's logistics module.

Relationship:

User (Delivery_Staff) (1) — (M) Delivery

Type: One-to-Many

7. Order – Feedback Relationship (1–1)

Each feedback record corresponds to a single completed order ensuring that reviews are directly tied to the service experience.

Relationship:

Order (1) — (1) Feedback

Type: One-to-One

ERD Figure and Discussion

Figure (3.1) illustrates the overall structure of the Entity Relationship Diagram (ERD) for the Aqua Hub iOS Application. The diagram shows how the main entities of the system User Service Order Payment Delivery and Feedback — are connected and interact with each other to support the core functions of the application.

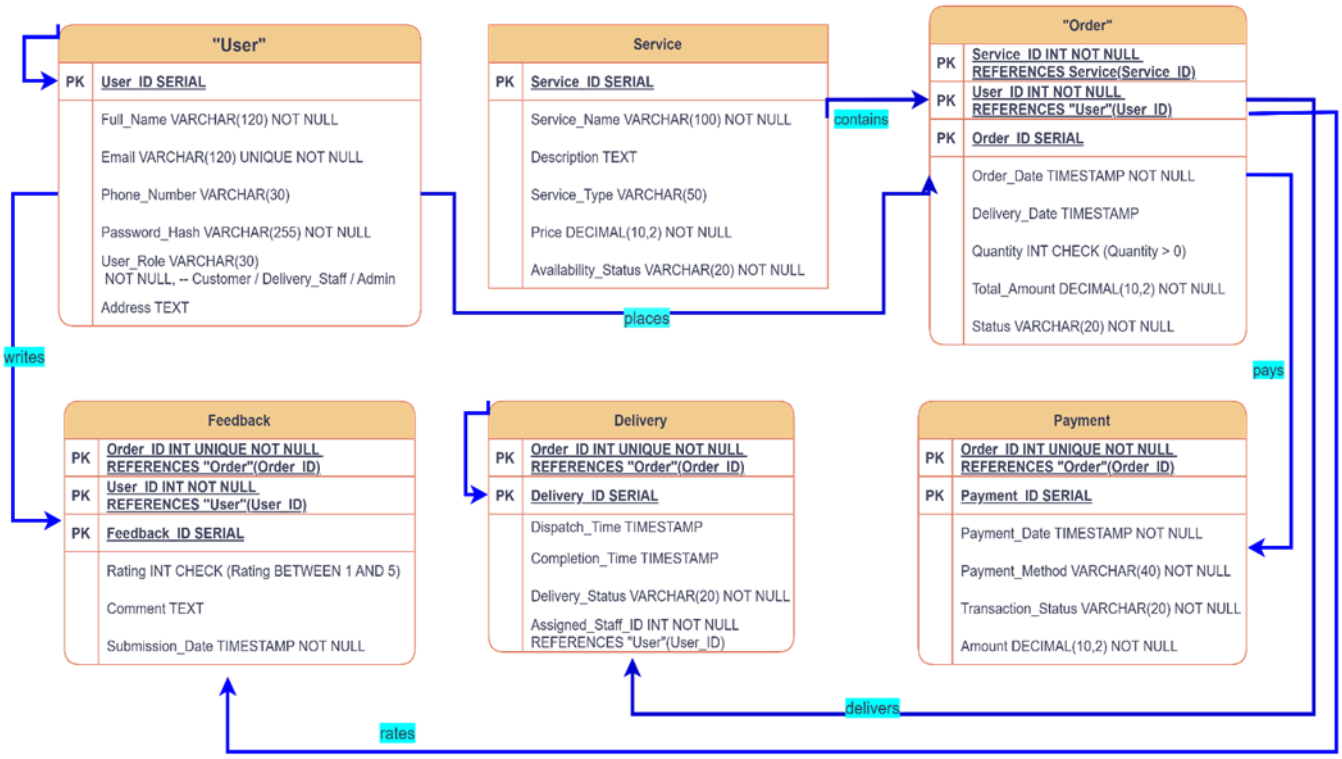

Figure 3.1: Entity Relationship Diagram for Aqua Hub iOS Application

The User entity represents the central element in the system since it is directly linked to orders deliveries and feedback. The Order entity acts as the core of the process by connecting users with the selected services and recording the related payment and delivery details. The Service entity defines the available offerings such as water delivery or tank cleaning which can be associated with multiple orders.

The Payment and Delivery entities describe the financial and operational sides of each order ensuring that every transaction and delivery is properly tracked. Finally the Feedback entity connects each user with a completed order allowing the system to measure customer satisfaction and service quality.

The model follows the Third Normal Form (3NF) to minimize redundancy and maintain data consistency across the database. This normalization improves query performance simplifies updates and reduces the chance of data conflicts. The ERD also provides flexibility for future

updates or feature extensions allowing new service types to be added without redesigning the entire structure.

Overall, this ERD forms the conceptual foundation for the upcoming database design phase. It ensures that the database supports secure scalable and efficient data handling aligned with iOS platform requirements especially in areas such as user data privacy and online payment processing.

3.1.2 Class Diagram

This section presents the UML Class Diagram for the Aqua Hub iOS Application. It models the main classes their attributes and key operations and how they collaborate to support the system's functionality. The diagram follows standard UML notation and is aligned with the entities and relationships introduced in the ERD.

Class Overview

User: Represents customers administrators and delivery staff (distinguished by role).

Service: Describes available services (e.g. bottled water delivery tank cleaning).

Order: Central transactional class that links a user to a service and aggregates payment and delivery details.

Payment: Handles financial transactions associated with an order.

Delivery: Manages the operational process of fulfilling an order including assignment and completion.

Feedback: Captures user evaluations after service completion.

Class Specifications

User

Attributes:

userID: UUID

fullName: String

email: String

phone: String

role: UserRole

Key Methods:

register ()

login ()

updateProfile ()

Notes: A single user can place many orders and submit multiple feedback entries. Users with the Delivery_Staff role can be assigned deliveries.

Service

Attributes:

serviceID: UUID

name: String

type: ServiceType

price: Decimal

available: Bool

Key Methods:

updatePrice ()

checkAvailability ()

Notes: A service can be referenced by many orders.

Order

Attributes:

orderID: UUID

orderDate: Date

deliveryDate: Date?

quantity: Int

totalAmount: Decimal

status: OrderStatus

Key Methods:

place ()

cancel ()

calculateTotal ()

Notes: Each order has exactly one payment one delivery and one feedback after completion.

Payment

Attributes:

paymentID: UUID

date: Date

method: PaymentMethod

status: PaymentStatus

amount: Decimal

Key Methods:

authorize ()

refund ()

Notes: One-to-one association with Order.

Delivery

Attributes:

deliveryID: UUID

dispatchTime: Date?

completionTime: Date?

status: DeliveryStatus

Key Methods:

assignStaff (user: User)

markCompleted ()

Notes: One-to-one association with Order; many deliveries can be assigned to a delivery staff user.

Feedback

Attributes:

feedbackID: UUID

rating: Int

comment: String

date: Date

Key Methods:

submit ()

Notes: One-to-one association with Order and many-to-one with User.

Relationships

User — Order: 1...* (a user can place multiple orders).

Service — Order: 1...* (a service can appear in many orders).

Order — Payment: 1...1 (each order has one payment).

Order — Delivery: 1...1 (each order has one delivery).

User (Delivery_Staff) — Delivery: 1...* (a staff user can handle multiple deliveries).

User — Feedback: 1...* (a user can submit multiple feedback records).

Order — Feedback: 1...1 (each order has one feedback after completion).

iOS/Swift Design Notes

The classes map naturally to Swift models. Value types (e.g. identifiers amount) can use Swift primitives while domain objects are modeled as structs or classes depending on mutability needs. For persistence Core Data entities can mirror the domain classes (e.g. Order Payment Delivery) with relationships enforced via NSManagedObject associations. Sensitive data (e.g. authentication tokens) should be stored in the Keychain.

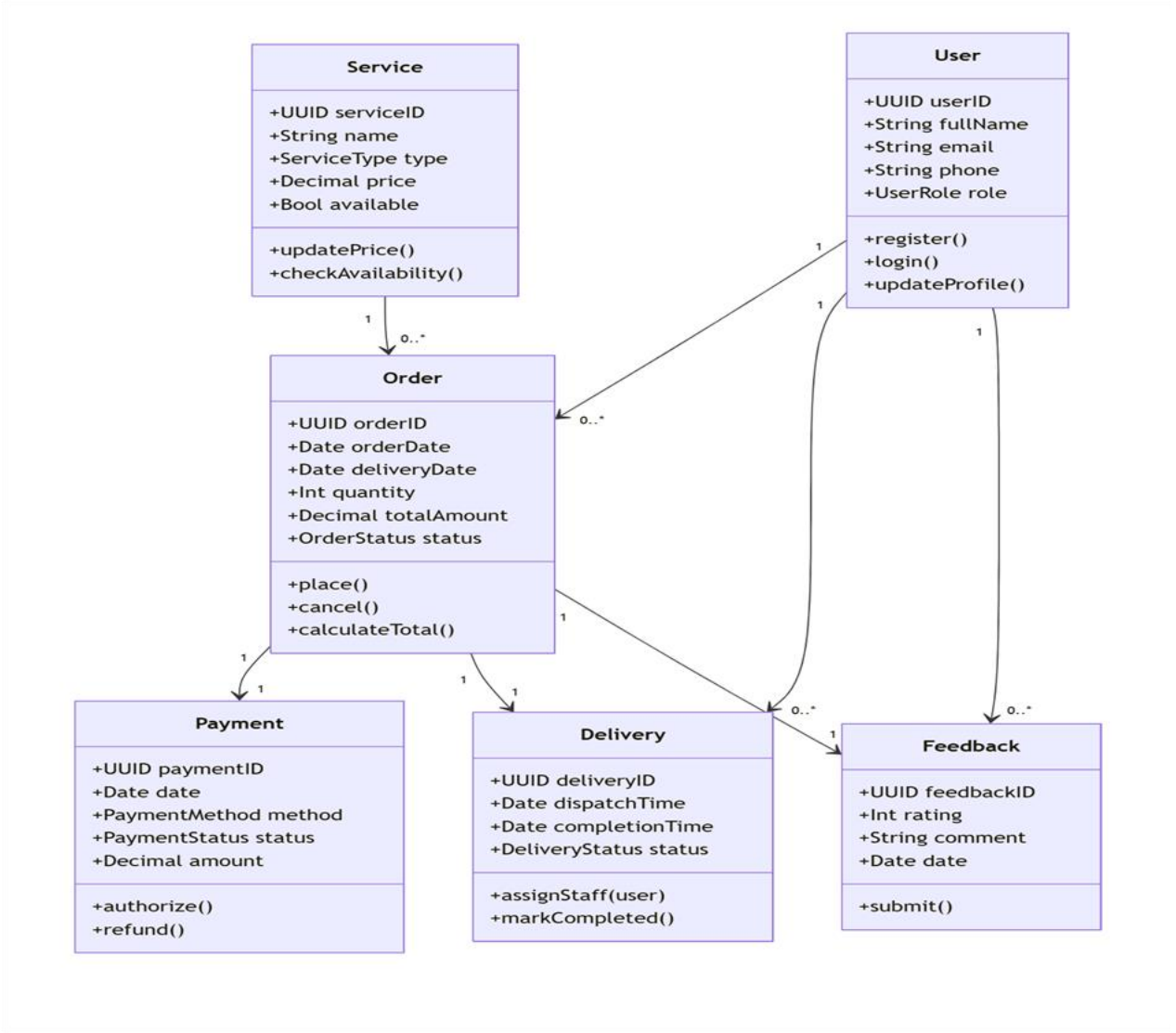

Figure 3.2: UML Class Diagram for Aqua Hub iOS Application

3.1.3 Database Schema

This section presents the logical-to-physical mapping of the ERD into a relational database schema for the Aqua Hub iOS Application. The schema follows third normal form (3NF) defines primary and foreign keys and specifies datatypes and constraints suitable for a cloud-hosted relational database (e.g. PostgreSQL). Identifiers use UUID to ensure global uniqueness across distributed services; timestamps are stored in UTC.

Table 3.1: users

Column	SQL Type	Constraints	Description
user_id	UUID	PRIMARY KEY NOT NULL	Unique identifier for the user.
full_name	VARCHAR (120)	NOT NULL	User full name.
email	VARCHAR (160)	NOT NULL UNIQUE	Login identifier; unique per user.
phone	VARCHAR (25)	NOT NULL	E.164 formatted phone number.
password_hash	VARCHAR (255)	NOT NULL	Hashed credential (never store plaintext).
user_role	VARCHAR (20)	NOT NULL	One of {Customer Admin Delivery_Staff}.
address	TEXT	NULL	Primary address used for deliveries.
created_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Creation timestamp (UTC).
updated_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Last update timestamp (UTC).

Table 3.2: services

Column	SQL Type	Constraints	Description
service_id	UUID	PRIMARY KEY NOT NULL	Unique service identifier.
service_name	VARCHAR (120)	NOT NULL	Name of the service.
description	TEXT	NULL	Service description.

service_type	VARCHAR (20)	NOT NULL	Delivery / Cleaning / Maintenance.
price	DECIMAL (102)	NOT NULL CHECK (price >= 0)	Base price for the service.
availability_status	VARCHAR (20)	NOT NULL	Available / Unavailable.
created_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Creation timestamp (UTC).
updated_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Last update timestamp (UTC).

Table 3.3: orders

Column	SQL Type	Constraints	Description
order_id	UUID	PRIMARY KEY NOT NULL	Unique order identifier.
order_date	DATE	NOT NULL	Date when the order was placed.
delivery_date	DATE	NULL	Scheduled delivery date.
quantity	INTEGER	NOT NULL CHECK (quantity > 0)	Requested quantity (e.g. number of bottles).
total_amount	DECIMAL (102)	NOT NULL CHECK (total_amount >= 0)	Final amount including taxes/fees.
status	VARCHAR (20)	NOT NULL	Pending / Approved / Completed / Cancelled.
user_id	UUID	NOT NULL REFERENCES users(user_id)	Customer who placed the order.
service_id	UUID	NOT NULL REFERENCES services(service_id)	Requested service.

created_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Creation timestamp (UTC).
updated_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Last update timestamp (UTC).

Table 3.4: payments

Column	SQL Type	Constraints	Description
payment_id	UUID	PRIMARY KEY NOT NULL	Unique payment identifier.
payment_date	TIMESTAMP	NOT NULL	Payment timestamp (UTC).
payment_method	VARCHAR (20)	NOT NULL	Apple_Pay / Cash_On_Delivery.
transaction_status	VARCHAR (20)	NOT NULL	Successful / Failed / Pending.
amount	DECIMAL (102)	NOT NULL CHECK (amount >= 0)	Paid amount.
order_id	UUID	NOT NULL UNIQUE REFERENCES orders(order_id)	One-to-one with orders.
created_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Creation timestamp (UTC).
updated_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Last update timestamp (UTC).

Table 3.5: deliveries

Column	SQL Type	Constraints	Description
delivery_id	UUID	PRIMARY KEY NOT NULL	Unique delivery identifier.

assigned_staff_id	UUID	NOT NULL REFERENCES users(user_id)	Delivery staff (user with Delivery_Staff role).
order_id	UUID	NOT NULL UNIQUE REFERENCES orders(order_id)	One-to-one with orders.
dispatch_time	TIMESTAMP	NULL	Time when delivery was dispatched (UTC).
completion_time	TIMESTAMP	NULL	Time when delivery was completed (UTC).
delivery_status	VARCHAR (20)	NOT NULL	Assigned / In_Progress / Completed / Delayed / Cancelled.
created_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Creation timestamp (UTC).
updated_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Last update timestamp (UTC).

Table 3.6: feedback

Column	SQL Type	Constraints	Description
feedback_id	UUID	PRIMARY KEY NOT NULL	Unique feedback identifier.
rating	INTEGER	NOT NULL CHECK (rating BETWEEN 1 AND 5)	User rating (1–5).
comment	TEXT	NULL	Optional textual feedback.
submission_date	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Feedback submission timestamp (UTC).

user_id	UUID	NOT NULL REFERENCES users(user_id)	User who submitted the feedback.
order_id	UUID	NOT NULL UNIQUE REFERENCES orders(order_id)	One-to-one with orders.
created_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Creation timestamp (UTC).
updated_at	TIMESTAMP	NOT NULL DEFAULT CURRENT_TIMESTAMP	Last update timestamp (UTC).

Indexes and Constraints

- UNIQUE (email) on users enforces one account per email.
- UNIQUE (order_id) on payments deliveries and feedback preserves one-to-one relations with orders.
- Foreign keys (user_id service_id order_id assigned_staff_id) enforce referential integrity.
- CHECK constraints validate domain values (e.g. non-negative amounts 1–5 rating).
- Recommended indexes: (user_id) (service_id) and (status) on orders; (transaction_status) on payments; (delivery_status) on deliveries.

iOS Persistence Notes

For iOS implementation Core Data entities can mirror these tables. UUIDs map to NSUUID/UUID. Sensitive credentials are not stored in Core Data; they should be kept in the Keychain. Server-side validation should mirror database constraints to provide early error feedback in the app.

Chapter 4 Implementation Phase

4.1 System Deployment

This section describes the environment used during the development and testing stages of the Aqua Hub iOS Application as well as the planned deployment process for production. The goal of this phase is to ensure that the system can smoothly transition from a development environment to a stable production-ready state.

4.1.1 Development and Testing Environment

The Aqua Hub application was developed using a modern iOS development environment designed to ensure reliability maintainability and performance.

- **Development Tools:** The application was built using Xcode 15 Apple's official Integrated Development Environment (IDE) for iOS applications.
- **Programming Language:** Swift 5 and SwiftUI were used to create a clean interactive and responsive user interface.
- **Database:** The system's data structure was implemented in PostgreSQL 15 based on the schema described in Appendix A (SQL Implementation).
- **Back-end Simulation:** During development mock APIs were used to simulate interactions between the front-end (SwiftUI) and the database as defined in **Appendix B (Swift Sample Code)**.
- **Operating Systems:** Testing was conducted on macOS Sonoma using iPhone 14 and iPhone 15 Pro simulators within Xcode.
- **Version Control:** The source code was maintained using GitHub ensuring version tracking and team collaboration.
- **Testing Approach:** Both functional and interface testing were performed through the Xcode Simulator to validate user interactions data flow and layout consistency.

4.1.2 Production Deployment Plan

The deployment plan ensures that the Aqua Hub system can operate reliably once released to actual users.
It consists of two main components: application distribution and database deployment.

- **App Store Deployment:**
  The iOS application is prepared for release through Apple's App Store Connect platform.
  This includes the following steps:
  1. Building the final archive of the app using *Xcode's Archive and Distribution* tools.
  2. Uploading the build to App Store Connect.
  3. Performing internal testing through TestFlight before public release.
  4. Publishing the app with proper metadata (icon description screenshots and category).
- **Database Deployment:**
  The PostgreSQL database is hosted on a remote production server configured with access control and user authentication.
  Database tables constraints and relationships defined in Appendix A are executed via SQL scripts to ensure full integrity.
  Backups and maintenance procedures are set to run automatically to preserve data consistency and availability.
- **Post-Deployment Testing:**
  After deployment both the client application and the database are validated for proper communication data synchronization and API response accuracy.

4.2 Prototype Implementation

This section describes the practical implementation of the **Aqua Hub prototype** highlighting how the user interface and functional components were developed integrated and validated during the system's initial build.
The focus of this phase was to transform the conceptual and design models described in earlier chapters into a working interactive system that accurately represents the intended features and workflows of the final product.

4.2.1 Front-end Implementation with SwiftUI

The front-end of the Aqua Hub application was implemented entirely using SwiftUI Apple's modern declarative framework for building iOS interfaces.
This approach enabled the creation of responsive scalable and reusable components for all key screens including:

- **Sign In / Registration** – for secure user authentication and onboarding.
- **Home Screen** – displaying available services such as Safi Water Water Tank (Drinking / Non-Drinking) Tank Cleaning and HydroFix Maintenance.
- **Ordering Flow** – allowing users to select service type specify quantities or capacities and confirm orders directly through intuitive bottom-sheet modals.
- **Payment Interface** – integrating mock options for Apple Pay and Cash on Delivery.
- **Feedback Form** – for collecting user ratings and service comments after order completion.

The SwiftUI code implemented in Appendix D demonstrates how the views state management and navigation logic were constructed. Custom SwiftUI views such as Primary Button Service Card and Capacity Selector were created to ensure design consistency and reduce code redundancy.

Each screen was optimized for light and dark modes ensuring full accessibility across different devices and orientations.

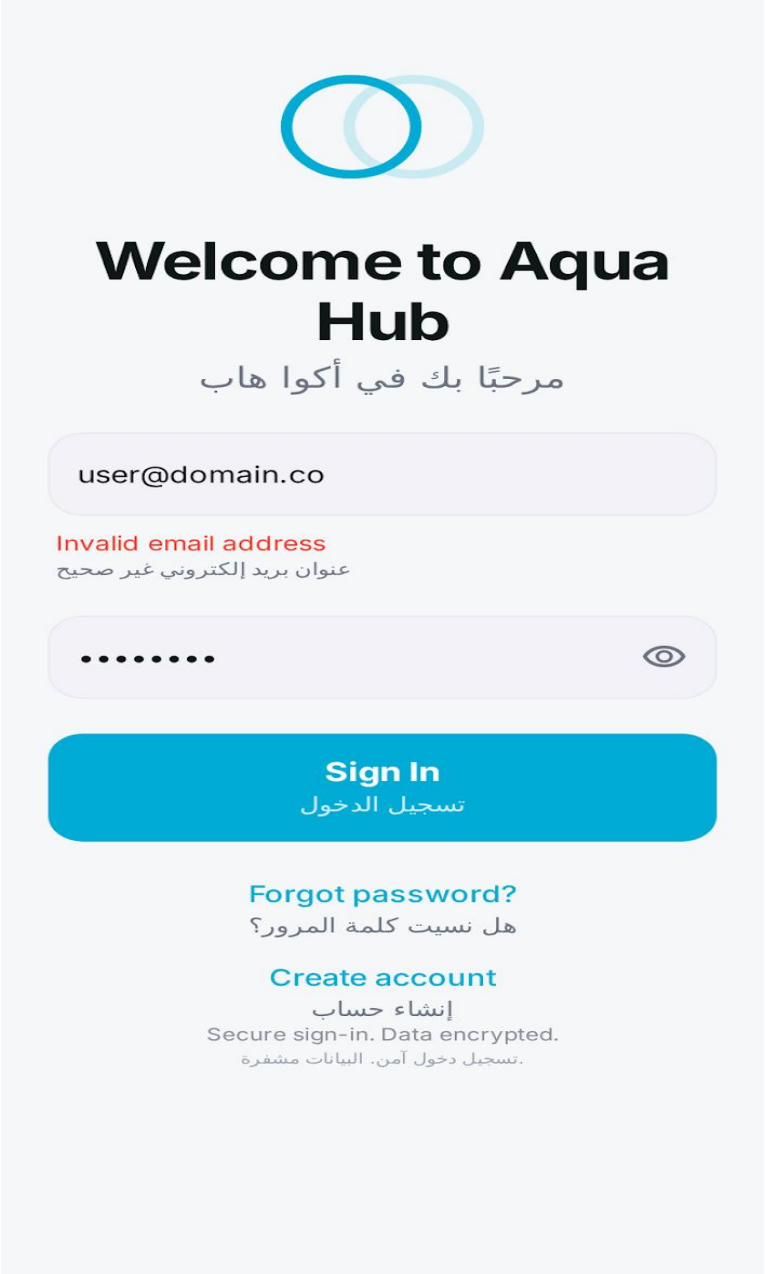

Figure 4.1: Aqua Hub Sign-In Screen (Prototype)

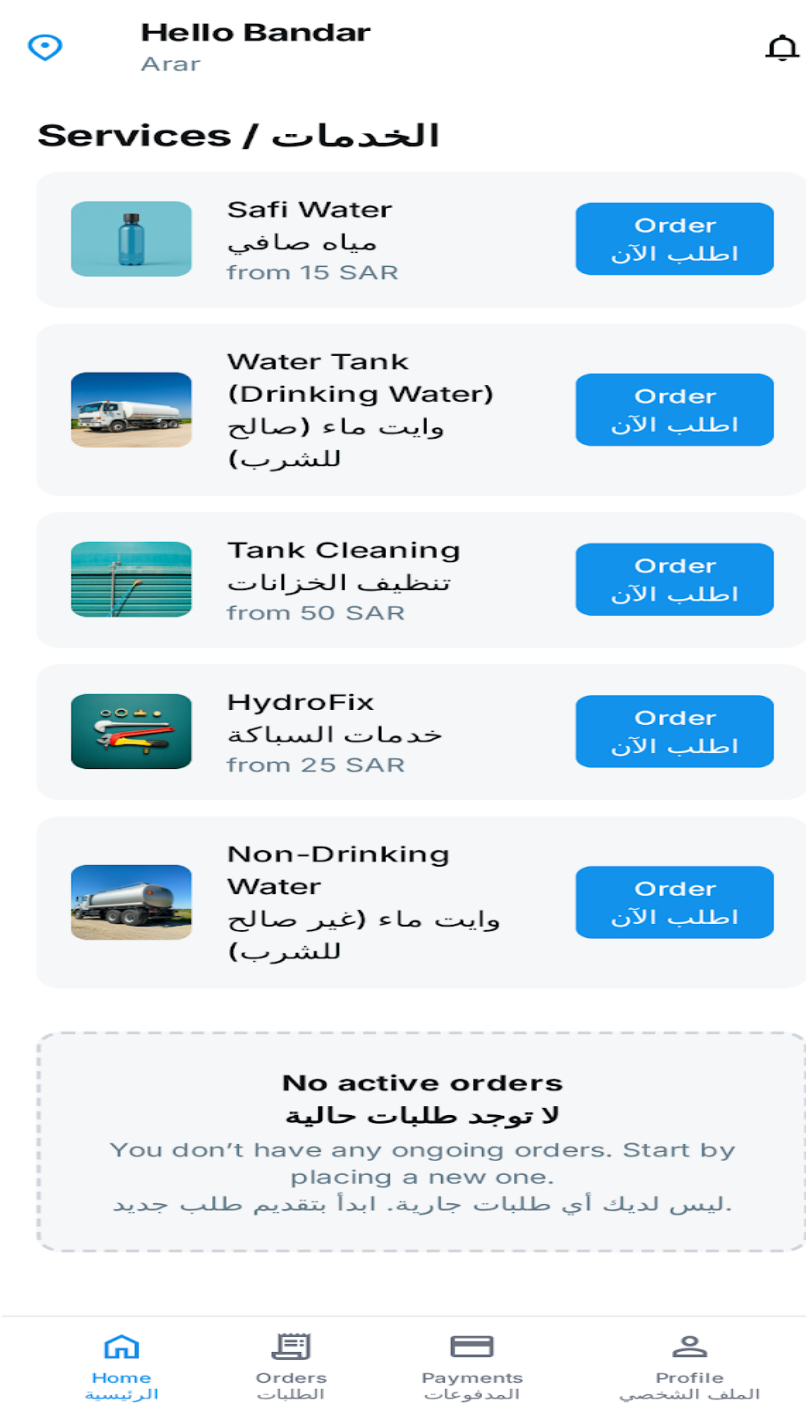

Figure 4.2: Home Services Screen with Integrated Options

4.2.2 Back-end Integration Logic

Although the Aqua Hub prototype primarily focuses on the front-end layer the design includes a clear structure for future back-end integration.
The data flow between the front-end and database is modeled based on the PostgreSQL schema presented in Appendix A and the Swift network layer shown in Appendix B.

Key integration principles defined include:

- **User Authentication:** Conceptual endpoints were defined for login and registration to validate credentials against encrypted records stored in the *user's* table.
- **Order Management:** Logical interaction was designed between the *orders* and *services* tables to enable dynamic status updates and maintain referential integrity.
- **Payment Transactions:** Prototype-level logic was implemented to simulate posting payment records to the *payments* table and synchronize payment status between the app and the database.
- **Feedback Handling:** A direct linkage between the *feedback* table and the user interface designed to store ratings and comments for each completed order.

Even though live server APIs were not implemented in this prototype stage the structure ensures seamless future integration with a RESTful back-end or cloud service (e.g. Firebase AWS or a custom Node.js API).

4.2.3 Functional Prototype Snapshots

The functional prototype was tested extensively in the Xcode Simulator and on physical devices (iPhone 14/15) to ensure all interactions animations and navigation flows operated as expected.

The visual design and interactive output of the implemented prototype as represented in the mockups of Appendix C and the executable SwiftUI code in Appendix D demonstrate the complete workflow

1. **User Login and Registration** – Secure entry point for end users.
2. **Home Services Display** – Organized cards showing available services.
3. **Order Placement Process** – Step-by-step selection via interactive modals.
4. **Payment Confirmation** – Simulated Apple Pay and Cash confirmation screens.
5. **Feedback Submission** – Simple form to evaluate service quality.
6. **Admin Dashboard** – Control panel for administrators to manage orders and delivery assignments.

These functional screens validate the application's core usability, navigation consistency, and aesthetic coherence.
They demonstrate that the implementation phase successfully achieved the intended design objectives.

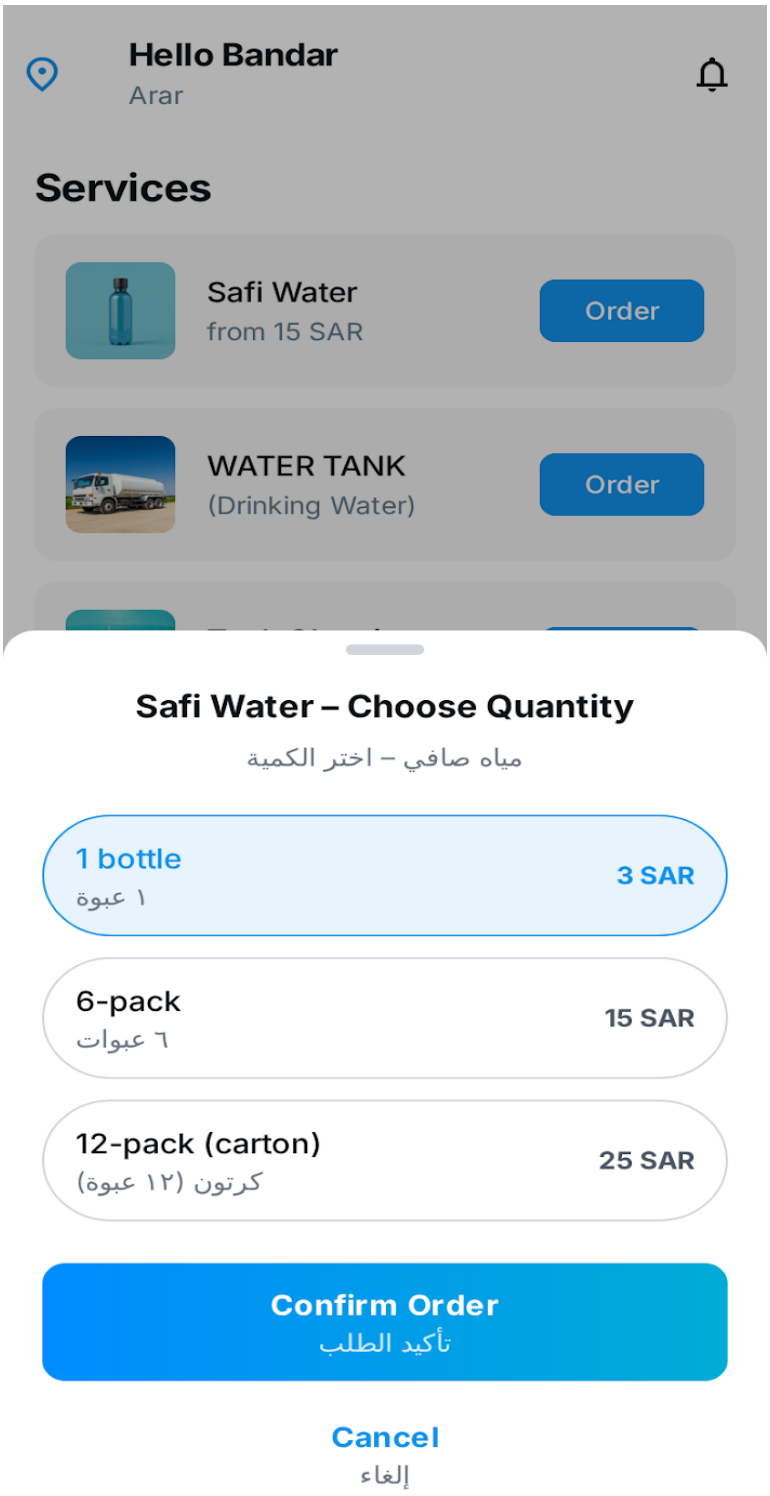

Figure 4.3: In-screen Service Option Selection (Bottom Sheet)

(Refer to Appendix C for the interface mockups and Appendix D for the implemented SwiftUI code and corresponding prototype snapshots.)

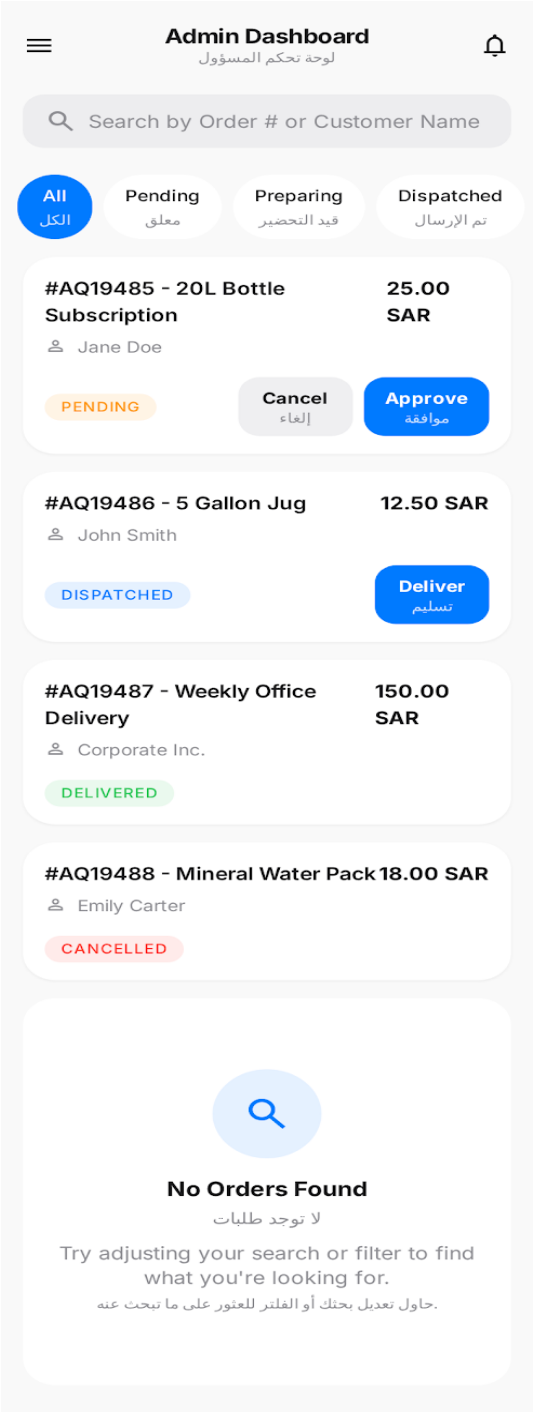

Figure 4.4: Admin Dashboard for Managing Orders

4.3 System Maintenance and Support

This section explains how the Aqua Hub system is maintained supported and continuously improved after deployment.
Proper maintenance ensures long-term reliability data integrity and user satisfaction while support mechanisms help end users resolve issues efficiently.
The system maintenance plan focuses on three main aspects: technical support backup and recovery and version management.

4.3.1 Technical Support and User Guidance

The Aqua Hub application includes a structured framework for technical support and user assistance to ensure operational continuity:

- **User Support:** A dedicated support channel (via email or in-app “Help & Support” form) is available to assist customers with login issues payment errors or order inquiries.
- **Admin Monitoring:** The Admin Dashboard (see Appendix E) provides administrators with tools to track incoming orders monitor system performance and respond to user feedback in real time.
- **User Documentation:** A brief in-app guide and FAQ section were designed to help users navigate core features such as ordering payment and feedback submission without external help.
- **Bug Tracking:** Identified issues are logged and prioritized using a task management system (e.g. GitHub Issues or Jira) to ensure they are resolved systematically during maintenance cycles.

This structured approach minimizes downtime and improves the reliability of user interactions across the platform.

4.3.2 Backup and Recovery Strategy

To maintain **data integrity and security** the system implements a proactive backup and disaster recovery strategy as follows:

- **Automated Backups:** The PostgreSQL database is configured for daily incremental backups and weekly full backups stored securely on a remote cloud storage service.
- **Redundancy:** Backup copies are maintained in multiple geographic regions to prevent data loss in case of hardware failure or regional outages.
- **Recovery Procedures:** Database restoration procedures are documented and periodically tested to ensure quick recovery in case of corruption or accidental deletion.
- **Access Control:** Only authorized administrators can perform backup retrieval and restoration following strict authentication protocols.

This approach guarantees the availability and resilience of Aqua Hub's core data assets under all operational conditions.

4.3.3 Update and Versioning Plan

To ensure continued improvement and compatibility with evolving user needs and iOS updates Aqua Hub follows a version-controlled update plan:

- **Version Control System:** All source code is maintained under Git allowing proper tracking of changes branches and releases.
- **Update Cycle:** Regular updates are planned every quarter focusing on minor bug fixes interface refinements and performance optimization.
- **Major Releases:** Larger feature updates (e.g. new service modules admin analytics or additional languages) are rolled out biannually following regression testing.
- **User Notifications:** Users are notified via App Store updates and in-app prompts whenever a new version is available ensuring a smooth upgrade experience.

- **Testing Before Release:** Each new version undergoes internal testing through **TestFlight** to validate stability before public deployment.

Through these maintenance practices Aqua Hub maintains a consistent secure and user-focused service experience over time.

Chapter 5 Conclusion and Future work

5.1 Conclusion

The development of the **Aqua Hub iOS Application** successfully demonstrated the process of transforming a conceptual design into a functional prototype that supports core water service management operations.
Throughout the project, emphasis was placed on achieving a user-friendly interface, structured database design, and scalable architecture capable of supporting real-time interactions between users, service providers, and administrators.

The project fulfilled its main objectives by:

- Providing a clear structure for service ordering, payment, and feedback.
- Implementing a responsive and intuitive SwiftUI-based interface.
- Ensuring data integrity and organization through a PostgreSQL relational schema.
- Presenting a cohesive model that can easily evolve into a full production-level system in the future.

Overall, Aqua Hub has proven the feasibility of integrating multiple service functions — from order placement to delivery tracking — in a unified, interactive mobile platform.

5.2 Future Work

While the current prototype fulfills the primary functional and design goals, several enhancements can be introduced in future iterations to expand its capabilities and improve scalability.
Recommended areas of future development include:

1. **Full Back-end Integration:**
   Developing live API endpoints and a secure authentication mechanism to connect the SwiftUI interface with a real-time PostgreSQL or cloud-hosted database.

2. **Advanced Analytics Dashboard:**
   Expanding the Admin Dashboard (Appendix E) with graphical reports, usage statistics, and predictive insights for service demand and delivery performance.
3. **Multilingual Support:**
   Adding additional language options beyond English and Arabic to improve accessibility for non-native users.
4. **Enhanced Security and Authentication:**
   Implementing two-factor authentication (2FA) and data encryption for sensitive user transactions.
5. **Cross-Platform Expansion:**
   Developing Android and web versions of the Aqua Hub application to increase reach and user adoption.
6. **AI-driven Optimization:**
   Integrating AI-based recommendations for delivery routes, pricing, and resource allocation to further improve efficiency.

Through these future enhancements, Aqua Hub can evolve from a functional prototype into a comprehensive, production-grade digital platform for water service management.

References

1. Pressman, R. S., & Maxim, B. R. (2019). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.
2. Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education.
3. Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide* (2nd ed.). Addison-Wesley.
4. Larman, C. (2004). *Applying UML and Patterns* (3rd ed.). Prentice Hall.
5. Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson Education.
6. Connolly, T. M., & Begg, C. E. (2020). *Database Systems: A Practical Approach to Design, Implementation and Management* (7th ed.). Pearson Education.
7. Apple Inc. (2024). *SwiftUI Documentation.* Retrieved from https://developer.apple.com/documentation/swiftui
8. PostgreSQL Global Development Group. (2024). *PostgreSQL 15 Official Documentation.*

Appendices

Appendix A – SQL Implementation Script

Appendix B – Swift Sample Code

Appendix C – User Interface Mockups

Appendix D – Prototype SwiftUI Screens

Appendix E – Aqua Hub Admin Dashboard