Mohamed Khider University of Biskra
Faculty of Exact Sciences
Department of Computer Science

MASTER THESIS

Speciality : Master - Networks and Information and Communication Technologies - Ref : P2203/2026

TITLE

SmartQueue DZ – Intelligent Queue Management Platform for Public Services

Prepared and defended by

Mabrouk abdelmoumen LAADJAL

Mohamed el mahdi BEN SALAH

Abderraouf CHENNA

Defense date : 09/06/2026

DEFENSE COMMITTEE

Full Name	Grade	Role
Ahmed ALOUI	MCA	**President**
Soheyb AYAD	Prof	**Supervisor**
Imane ALOUI	MCB	**Examiner**
Chahra ADISSA	Prof	**Incubator Representative**

Academic Year : 2025/2026

Dedication

In the Name of Allah, the Most Gracious, the Most Merciful

All praise is due to Allah, who granted me the strength to persevere, the patience to endure challenges, and the opportunity to reach this important milestone in my academic journey. Without His guidance and blessings, none of this would have been possible.

I dedicate this humble work to the two most important people in my life, my beloved parents.

To my dear mother, whose unconditional love, endless prayers, and unwavering support have been my greatest source of comfort and motivation. Your sacrifices and devotion can never be fully repaid, and this achievement is as much yours as it is mine.

To my dear father, whose wisdom, hard work, and dedication have always inspired me to strive for excellence. Thank you for teaching me the values of perseverance, responsibility, and determination, and for believing in me every step of the way.

To my family, thank you for your encouragement, trust, and constant support throughout my journey. Your presence has always been a source of strength and confidence.

To my friends, who shared with me the challenges, memories, and experiences of university life, thank you for your companionship, encouragement, and the unforgettable moments we created together.

My sincere gratitude also goes to my teachers and supervisor, whose guidance, knowledge, and support have contributed greatly to my academic and personal growth.

Finally, I dedicate this work to everyone who believed in me, encouraged me, and supported me throughout this journey. This achievement represents years of effort, perseverance, and hope, and I pray that it marks the beginning of a future filled with success, growth, and meaningful contributions.

Abderraouf Chenna

Dedication

In the Name of Allah, the Most Gracious, the Most Merciful

All praise is due to Allah, who guided my path with knowledge and granted me the strength, patience, and determination to reach this important milestone.

I dedicate this humble work to the most precious people in my life...

To my beloved mother, whose prayers accompanied me through every step of my journey, whose unconditional love, sacrifices, and endless support gave me the courage to pursue my dreams. No words can ever express my gratitude for everything you have done for me.

To my dear father, my role model and greatest source of strength, who taught me that success is achieved through hard work, perseverance, and dedication. Thank you for every sacrifice you made and for always believing in me.

To my parents, whose prayers, support, and unwavering faith in me made it possible for me to become the first graduate in our family, I proudly dedicate this achievement.

I also dedicate this work to my family members, whose encouragement and affection have always motivated me throughout my academic journey.

To my dear friends who shared with me the challenges, memories, and successes of student life, especially Talha, Abdelghani, Djamel, and Firas, thank you for your friendship, support, and the unforgettable moments we shared together.

My sincere appreciation also goes to every teacher, mentor, colleague, and every person who helped, guided, encouraged, or inspired me along the way. Your contribution, no matter how small it may seem, has played a part in this accomplishment.

Finally, I dedicate this work to everyone who believed in me and stood by my side throughout this journey. This achievement is the result of years of effort, perseverance, and hope, and I pray that it marks the beginning of a future filled with success, growth, and contribution.

Abdelmoumen Laadjal

Dedication

In the Name of Allah, the Most Gracious, the Most Merciful

All praise is due to Allah, by whose blessings righteous deeds are completed, and by whose grace and guidance goals are reached and dreams are fulfilled. I praise and thank Him for enabling me to reach this important stage of my academic journey and to complete our graduation project entitled SmartQdz.

I dedicate this humble work to my beloved parents, whose prayers, sacrifices, and constant support have been the foundation of every step I have taken. To my dear mother, for her endless love, care, and encouragement, and to my dear father, for his guidance, trust, and the values of perseverance and responsibility he taught me.

I also dedicate this work to my dear aunt, whose kindness, support, and encouragement have always meant a great deal to me.

To my family, thank you for your love, patience, and belief in me. Your presence has always been a source of strength and motivation.

To my dear friends, especially those who shared with me the challenges and memories of this journey, thank you for your encouragement, support, and unforgettable moments.

I would also like to dedicate this work to my project teammates, with whom I shared effort, responsibility, and determination throughout the development of this project. This achievement is also a reflection of our collaboration and commitment.

Finally, I dedicate this work to every teacher, mentor, colleague, and person who helped, guided, or encouraged me along the way. May this work mark the beginning of a future full of success, learning, and meaningful contribution.

Bensaleh Mahdi

Acknowledgements

I would like to express my sincere gratitude to everyone who contributed to the completion of this work.

First and foremost, I wish to thank my supervisor for his valuable guidance, continuous support, and insightful advice throughout the development of this project. His recommendations and encouragement were essential in strengthening both the academic and practical dimensions of this dissertation.

I would also like to thank the members of the jury for accepting to evaluate this work and for the time they devote to examining it.

My special thanks go to all the teachers of the department for the knowledge and training they provided during my academic journey. Their efforts have greatly contributed to my learning and personal development.

I am deeply grateful to my family for their patience, moral support, and constant encouragement. Their trust and sacrifices have always motivated me to continue and achieve my goals.

Finally, I would like to thank my friends and colleagues who supported me, encouraged me, and shared valuable ideas during the realization of this project.

Abstract

The digital transformation of administrative services has created a strong need for modern, reliable, and citizen-centered queue-management solutions. In many public and service-oriented institutions, traditional queues still lead to overcrowding, long waiting times, poor visibility over service flow, and weak arrival-verification mechanisms. These limitations affect both service quality and the overall user experience.

This dissertation presents SmartQdz, a smart digital queue-management platform designed to modernize appointment scheduling, walk-in ticket issuance, self check-in, and service handling within administrative organizations. The research question addressed is how IoT-enabled devices, dynamic QR-based authentication, and real-time communication can improve the reliability and operational efficiency of administrative queue systems. The proposed system follows a multi-layer client-server architecture composed of a citizen-facing mobile application, a staff operations portal, an administrative dashboard, a secure backend server, a relational database, and connected devices such as QR kiosks, scanners, and notification services.

Through the platform, users can register, select a branch and service, reserve an appointment, or request a walk-in ticket. The system then generates a dynamic QR token that is used for secure self check-in upon arrival. Staff members can monitor queue activity in real time, call the next client, manage service sessions, and record service outcomes, while administrators maintain centralized control over branches, services, schedules, users, and queue policies. The main contributions include a hybrid appointment/walk-in queue model, counter-based assignment, backend-controlled QR validation, WebSocket synchronization, and role-aware administration.

The proposed solution aims to provide a practical balance between usability, security, scalability, and deployment cost. Prototype-level evaluation focuses on functional correctness, API behavior, QR security checks, real-time update propagation, and concurrency-sensitive operations. By combining IoT connectivity, QR-based verification, secure communication, and real-time synchronization, SmartQdz offers a modern approach for improving administrative service delivery and reducing the inefficiencies associated with conventional queue-management procedures.

Keywords: Queue management, SmartQdz, QR code, self check-in, administrative services, IoT

Résumé

Ce mémoire présente SmartQdz, une plateforme intelligente de gestion des files d'attente conçue pour moderniser les services administratifs et améliorer l'expérience des usagers. Dans de nombreuses institutions publiques, les méthodes traditionnelles reposent encore sur des tickets papier, des espaces d'attente surchargés et des procédures de vérification peu efficaces, ce qui entraîne des retards, une mauvaise visibilité sur l'avancement des files et une baisse de la qualité du service.

La solution proposée s'appuie sur une architecture client-serveur multicouche comprenant une application mobile destinée aux citoyens, un portail opérationnel pour le personnel, un tableau de bord d'administration, un serveur backend sécurisé, une base de données relationnelle ainsi que des dispositifs connectés tels que des bornes QR, des scanners et des services de notification. Les usagers peuvent s'inscrire, choisir une agence et un service, réserver un rendez-vous ou demander un ticket de passage, puis utiliser un code QR dynamique pour l'auto-enregistrement à l'arrivée.

Du côté administratif, la plateforme permet la gestion centralisée des agences, des services, des horaires, des utilisateurs et des règles de file d'attente. Le personnel peut suivre la file en temps réel, appeler l'usager suivant, lancer la prestation et enregistrer le résultat final. L'ensemble est renforcé par des mécanismes de sécurité fondés sur la validation serveur, la limitation temporelle des jetons, le chiffrement des communications et la synchronisation temps réel entre l'environnement physique et le flux numérique.

Ainsi, SmartQdz constitue une solution pertinente pour réduire les temps d'attente, limiter la surcharge des espaces d'accueil et renforcer la transparence du service. Ce travail montre qu'une approche fondée sur l'IoT, les codes QR et des services numériques sécurisés peut contribuer efficacement à la modernisation de la gestion des files d'attente dans les institutions administratives.

Keywords: gestion des files d'attente, SmartQdz, code QR, auto-enregistrement, services administratifs, IoT

Contents

Dedications i

Acknowledgements iv

Abstract v

Résumé vi

General Introduction 1

1 Theoretical and Technological Foundations 5

1.1 Introduction 5

1.2 Software Applications: Mobile and Desktop Environments 5

1.2.1 Introduction to Software Applications 5

1.2.2 Mobile Applications 6

1.2.3 Desktop Applications 6

1.2.4 Integration in the Proposed System 7

1.3 Internet of Things (IoT) 7

1.3.1 Definition of IoT 7

1.3.2 Role of IoT in the Proposed System 8

1.4 Real-Time Communication 8

1.4.1 WebSocket Protocol 8

1.4.2 Real-Time Architecture in Queue Systems 9

1.5 Push Notification Services 10

1.6 Queue Management Theory 11
1.6.1 Basic Queue Models 11
1.6.2 Scheduling and Prioritization 11
1.7 QR Code Technology 12
1.7.1 Definition of QR Code 12
1.7.2 Use of QR Codes in the System 12
1.8 QR-Based Authentication 13
1.8.1 Concept of Digital Authentication 13
1.8.2 QR-Based Authentication Workflow 13
1.8.3 Token Lifecycle 14
1.8.4 Advantages of QR-Based Authentication 15
1.9 Security and Cryptography in the System 15
1.9.1 Importance of Information Security 15
1.9.2 Applied Security Mechanisms 15
1.9.3 Potential Threats and Mitigation 16
1.10 Conclusion 17

2 Existing Solutions and Comparative Study 18

2.1 Introduction 18
2.2 Selection Criteria 18
2.3 Comparative Study of Existing Solutions 19
2.3.1 Qmatic 19
2.3.2 Skiplino 20
2.3.3 QLess 21
2.3.4 Wavetec 22
2.3.5 Lineberty 23
2.4 Comparative Summary Table 24
2.5 Technical and Economic Comparison 25
2.5.1 Deployment Complexity and Cost Considerations 27

2.5.2 Open-Source and Proprietary Considerations 27
2.6 Discussion 28
2.7 Conclusion 29

3 Design of the Proposed System 30
3.1 Introduction 30
3.2 General Architecture of SmartQdz 31
3.3 System Architecture 31
3.3.1 Main Components of the System 32
3.3.2 System Layers 32
3.3.3 Role of the Central Server 34
3.3.4 Why This Architecture is Appropriate 34
3.4 Real-Time Synchronization Architecture 34
3.5 Detailed Functional Architecture 35
3.5.1 Client and Booking Module 36
3.5.2 QR Self Check-in Module 36
3.5.3 Staff Operations Module 37
3.5.4 Administration and Backend Module 38
3.6 Queue Lifecycle 39
3.7 Queue Prioritization Logic 40
3.8 Main Actors and Functionalities 41
3.8.1 Citizens / Clients 41
3.8.2 Receptionists and Service Staff 41
3.8.3 Administrators / Super Administrators 41
3.8.4 Smart Devices and Connected Components 42
3.9 Device Communication 42
3.10 Functional Scope Summary 42
3.11 Sequence Diagrams 43
3.11.1 Booking Process Sequence 43

3.11.2 Authentication and Booking Sequence 44
3.11.3 QR Validation Sequence 45
3.11.4 Serving Process Sequence 46
3.12 Data Model Summary 47
3.13 Security Workflow 48
3.14 Conclusion 48

4 Implementation and Evaluation 50
4.1 Introduction 50
4.2 Development Architecture and Technology Stack 50
4.3 Development Environment and Tools 51
4.4 API Design and Backend Implementation 53
4.4.1 Request and Response Examples 53
4.4.2 Backend Service Responsibilities 55
4.5 Database Implementation 55
4.5.1 Database Integrity Rules 55
4.6 Booking and Queue Logic 56
4.6.1 Main Booking Actors 56
4.6.2 Branch and Service Selection 57
4.6.3 Slot Availability 57
4.6.4 Remote Appointments and Walk-ins 57
4.6.5 Capacity and Remaining Logic 57
4.6.6 Counter Assignment Logic 58
4.6.7 Appointment Creation 59
4.6.8 Walk-in Creation 59
4.6.9 Queue Session and Queue Position Logic 59
4.6.10 Booking Reference Number and Display Code 60
4.6.11 QR Check-in Validation 60
4.6.12 Cancellation and Rebooking 60

4.6.13 Duplicate and Capacity Protection 61
4.6.14 Live Queue and Queue Monitor 61
4.6.15 Special Needs Logic 61
4.6.16 Branch, Counter, and Slot Isolation 62
4.7 Real-Time and Notification Implementation 62
4.7.1 WebSocket Implementation 62
4.7.2 Push Notification Implementation 63
4.8 Security Implementation 63
4.8.1 JWT and Middleware Flow 64
4.8.2 QR Validation Internals 64
4.9 Deployment Architecture 65
4.10 Mobile Application Interfaces 66
4.10.1 Mapping Integration Using OpenStreetMap 66
4.11 Administrative Dashboard Interfaces 69
4.12 QR and IoT Device Integration 71
4.13 Testing and Results 73
4.13.1 Evaluation Method 73
4.13.2 Functional Test Scenarios 74
4.13.3 Security and Error Tests 75
4.13.4 Performance and Real-Time Indicators 75
4.13.5 Load and Concurrency Testing 76
4.14 Conclusion 77

General Conclusion **78**

References **79**

A Additional Technical Documentation **84**

A.1 Introduction 84
A.2 Complete UML Documentation 84

A.2.1 Complete Use Case Diagram 85
A.2.2 UML Activity Diagram – End-to-End Operational Lifecycle 88
A.3 Detailed Implementation 90
A.3.1 Authentication Module 91
A.3.2 Queue Management Module 92
A.3.3 QR Module 95
A.3.4 Real-Time Communication Module 98
A.3.5 Notification Module 100
A.4 Additional Interface Documentation 102
A.4.1 Mobile Application Screens 102
A.4.2 Administrative Dashboard Screens 104
A.4.3 Public Kiosk Screens 105
A.4.4 Hardware Prototype Images 106
A.5 Additional Technical Notes 108
A.5.1 System Scalability Considerations 108
A.5.2 Security Considerations 109
A.5.3 Future Hardware Expansion 109
A.5.4 Multi-Branch Deployment Strategy 109
A.5.5 High Availability Considerations 109
A.6 Conclusion 109

List of Figures

1.1 Mobile and Desktop Application Environments in SmartQdz. 6

1.2 Mobile and desktop integration in SmartQdz. 7

1.3 IoT ecosystem supporting real-time queue synchronization in SmartQdz. . 8

1.4 Real-time communication model for queue-state synchronization. 10

1.5 Lifecycle of the QR-based ticket used by SmartQdz. 13

1.6 QR-based authentication workflow in SmartQdz. 15

1.7 Security layers applied in the SmartQdz platform. 16

2.1 Traditional Queue Management System (Qmatic) 20

2.2 Cloud-Based Queue Management Solution (Skiplino) 21

2.3 Virtual Queue Management Platform (QLess) 22

2.4 Enterprise Queue Management Solution (Wavetec) 23

2.5 Mobile Virtual Queue Platform (Lineberty) 24

3.1 General architecture of the SmartQdz platform. 31

3.2 Client and booking module architecture and workflow. 36

3.3 QR self check-in module architecture and workflow. 37

3.4 Staff operations module architecture and workflow. 38

3.5 Administration and backend module architecture and workflow. 39

3.6 Booking process sequence diagram. 44

3.7 Authentication and booking sequence diagram. 45

3.8 QR validation and self check-in sequence diagram. 46

3.9 Serving process sequence diagram. 47

4.1 Development environment and implementation stack. 53
4.2 Database implementation of the SmartQdz platform. 56
4.3 Counter assignment workflow. 58
4.4 Deployment architecture of the SmartQdz prototype. 65
4.5 Mobile login and home screens of the citizen application. 67
4.6 Booking screens of the mobile application: branch and service selection, appointment scheduling, and walk-in options. 68
4.7 QR ticket display and queue-tracking screens of the mobile application. . . 69
4.8 Administrative dashboard overview. 69
4.9 Queue-management screen. 70
4.10 Service-configuration screen. 70
4.11 User and branch management screen. 70
4.12 Public Kiosk Interfaces of the SmartQdz Platform 72
4.13 SmartQdz Kiosk Hardware Prototype Based on Raspberry Pi 73

A.1 Exhaustive use case boundary diagram showcasing multi-actor interactions, administrative overrides, and automated device validation scopes. 85
A.2 End-to-End SmartQdz Appointment, Check-In, and Queue Service Lifecycle Activity Diagram 89
A.3 Architectural block diagram of the authentication pipeline, illustrating token generation, middleware verification, and RBAC enforcement. 91
A.4 System workflow illustrating daily session generation, capacity validation, load calculation, and counter routing operations. 93
A.5 Block diagram detailing the cryptographic verification flow, anti-replay validation, and WebSocket trigger sequence of the QR module. 96
A.6 Real-time network architecture diagram demonstrating event-driven broadcasts, private channel subscriptions, and client synchronization loops. . . . 98
A.7 Notification dispatch workflow detailing event listener captures, payload packaging, and asynchronous delivery. 101
A.8 Layout representations of the mobile application, showcasing registration, branch selection, scheduling, and active ticket views. 104

A.9 Visual layout of the administrative control center, showcasing real-time queue states, counter controls, and analytics widgets. 105

A.10 Layout representation of the lobby kiosk interface, showcasing language selection, service categories, and confirmation views. 106

A.11 Visual references of the Raspberry Pi 4 hardware prototype, showing enclosure details and touchscreen integration. 107

A.12 Photographic views of the completed kiosk housing, thermal printer integrations, and scanner mounts. 108

List of Tables

2.1 Qualitative comparison of existing solutions and the SmartQdz proposal . . 24

2.2 Technical comparison of queue-management solutions 26

2.3 Qualitative cost factors affecting deployment 27

3.1 Main real-time events in the proposed design 35

3.2 Queue-entry lifecycle states 40

3.3 Device communication responsibilities 42

3.4 Functional scope summary 43

3.5 Main entities of the proposed database design 48

4.1 Selected technologies supporting the system architecture 51

4.2 Development environment and implementation tools 52

4.3 Implemented WebSocket channel model 62

4.4 Notification events and payload purpose 63

4.5 Security controls in the implementation 64

4.6 Deployment components 66

4.7 Functional test scenarios 74

4.8 Security validation tests 75

4.9 Quantitative indicators for prototype evaluation 76

4.10 Concurrency tests 77

General Introduction

Context and Motivation

The modernization of administrative services is increasingly linked to broader digital-government strategies that seek to improve accessibility, transparency, and operational efficiency. Public institutions are expected to offer services that are not only secure, but also convenient and responsive to the needs of citizens. In this context, queue management is no longer a secondary operational detail; it directly affects service quality, user satisfaction, staff productivity, and the perceived fairness of the institution itself.

Traditional queue-management methods still rely heavily on on-site presence, paper tickets, and manual verification. Such practices often generate congestion, uncertainty about waiting times, and weak control over service flow. At the same time, the expansion of secure digital identity practices has increased expectations regarding authentication, accountability, and the trustworthy use of digital public services.

Problem Statement

Many administrative offices continue to operate through first-come, first-served physical lines in which users must arrive early, wait in crowded areas, and depend on manual calling procedures. This model creates several limitations:

- long and unpredictable waiting times.
- overcrowding in service areas.
- difficulty in balancing appointments and walk-in requests.
- limited visibility for users regarding queue progression.
- weak arrival verification, which may enable misuse of reservations or tickets.

These limitations reduce service quality and make operational control more difficult for staff and administrators. A modern solution should therefore support remote booking, real-time queue monitoring, secure self check-in, and centralized administrative supervision within the same digital platform.

Research Question

The central research question addressed by this dissertation is:

> How can IoT-enabled devices, dynamic QR-based authentication, and real-time communication improve the reliability and operational efficiency of administrative queue-management systems?

This question is supported by three secondary questions:

- how can remote appointments and same-day walk-ins be integrated into one coherent queue model?
- how can the system verify physical arrival without relying on manual ticket inspection?
- how can queue-state changes be synchronized between citizens, staff members, administrators, and branch devices?

Objectives

The main objective of this dissertation is to design and present a smart queue-management system suitable for administrative institutions. More specifically, the project aims to:

1. provide citizens with a digital channel for registration, appointment booking, walk-in ticketing, and queue tracking.
2. strengthen arrival verification through dynamic QR-code authentication and backend validation.
3. support staff members with a real-time operational interface for queue handling and service execution.
4. offer administrators centralized control over branches, services, schedules, users, and queue rules.

5. establish a secure and scalable technical architecture integrating mobile clients, back-end services, databases, and connected devices.

Main Contributions

The main contributions of this work are:

- a hybrid appointment and walk-in queue model governed by branch, counter, date, and time-slot constraints.
- a dynamic QR-based self check-in mechanism that links digital reservations to verified physical arrival.
- a real-time synchronization model for updating mobile applications, staff dashboards, administrative views, and connected branch devices.
- a counter-based assignment strategy that distributes bookings according to capacity and operational eligibility.
- a role-aware administrative architecture that separates citizen, staff, administrator, device, and backend responsibilities.

Methodology

The project was developed using an iterative and prototype-driven methodology. The work began with the identification of operational problems in traditional administrative queues, followed by the analysis of existing queue-management platforms. The proposed architecture was then modeled through functional modules, actors, workflows, and data entities. After that, a prototype implementation was organized around a mobile client, an administrative dashboard, a Laravel backend API, a relational database, and connected QR-scanning devices.

Each iteration refined one part of the system: booking logic, QR token generation, check-in validation, queue-state transitions, dashboard synchronization, and administrative control. The evaluation phase focused on functional correctness, integration consistency, security checks, and prototype-level performance indicators rather than a full production-scale deployment.

Dissertation Structure

This dissertation is organized into four core chapters, followed by a general conclusion:

- Chapter 1 presents the theoretical and technological foundations of the proposed system, including application environments, IoT, QR technology, authentication, and security.
- Chapter 2 reviews representative queue-management solutions and develops a comparative analysis that motivates the SmartQdz proposal.
- Chapter 3 describes the design of the proposed system, including its architecture, actors, workflows, UML-oriented models, and security logic.
- Chapter 4 presents the implementation environment, interfaces, database realization, QR and device integration, and the evaluation approach.

The dissertation concludes by summarizing the main contributions of the work, identifying its limitations, and presenting possible directions for future improvement.

Chapter 1

Theoretical and Technological Foundations

1.1 Introduction

The rapid evolution of digital technologies and intelligent information systems has significantly transformed the way institutions deliver services. One of the most visible operational challenges in administrative environments remains queue congestion, which affects waiting times, service quality, and user satisfaction.

To address this challenge, several technological building blocks can be combined into an integrated digital solution. Among the most relevant are mobile and desktop applications, Internet of Things (IoT) connectivity, QR-code technology, digital authentication mechanisms, and information-security techniques. Together, these technologies support remote interaction, secure identification, real-time synchronization, and operational control.

This chapter presents the theoretical and technological foundations of SmartQdz by explaining the role of each of these components and their relevance to a secure and efficient queue-management platform.

1.2 Software Applications: Mobile and Desktop Environments

1.2.1 Introduction to Software Applications

Software applications form the user-facing layer of information systems. In modern service platforms, they rarely operate as isolated programs; they coordinate with backend services,

databases, and external devices to support structured workflows [1, 2]. This principle is important for SmartQdz, where mobile, dashboard, and device interfaces must all follow the same queue rules.

1.2.2 Mobile Applications

Mobile applications provide the citizen access point for digital services. They support registration, appointment booking, notifications, queue tracking, and QR ticket display while using device capabilities such as camera access and connectivity [3, 4]. In SmartQdz, the mobile application is mainly responsible for remote interaction and for presenting the dynamic QR token used during self check-in.

1.2.3 Desktop Applications

Desktop or web-based dashboards are better suited to staff and administrative work because they provide larger displays, easier multitasking, and efficient data-management interfaces [5, 6]. In SmartQdz, staff use the dashboard to consult queues, call users, and record service outcomes, while administrators manage branches, services, users, schedules, and reports.

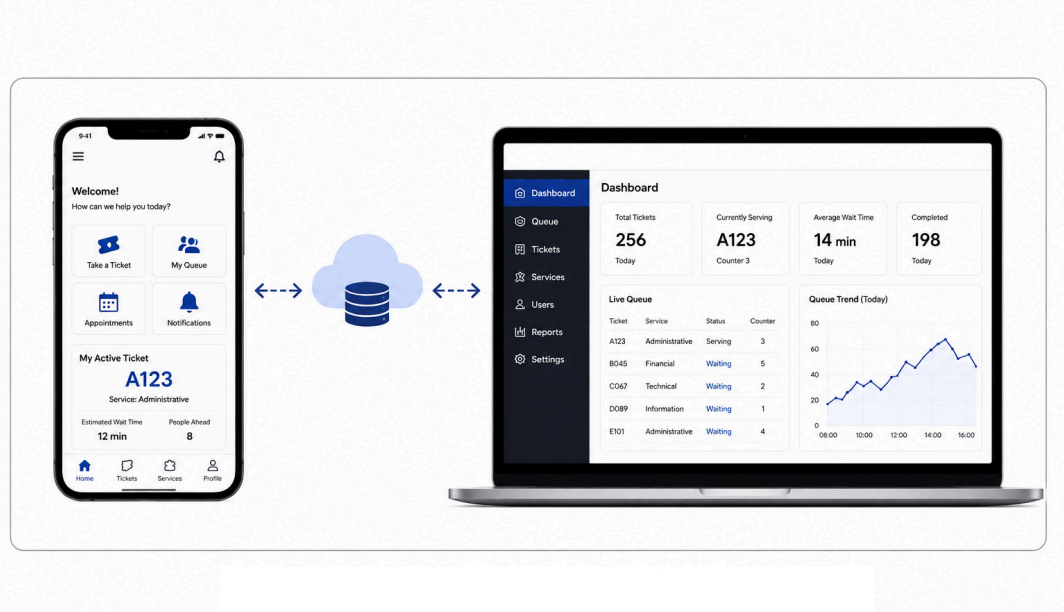


Figure 1.1: Mobile and Desktop Application Environments in SmartQdz.

1.2.4 Integration in the Proposed System

The proposed system relies on the complementary use of both environments. The mobile side provides accessibility and citizen autonomy, while the dashboard side ensures operational control and administrative supervision. The effectiveness of SmartQdz depends on the secure synchronization between these two application environments through backend services and shared data persistence.



Figure 1.2: Mobile and desktop integration in SmartQdz.

1.3 Internet of Things (IoT)

1.3.1 Definition of IoT

The Internet of Things refers to a network of interconnected physical devices capable of collecting, processing, and exchanging data through communication networks with limited direct human intervention. IoT systems commonly combine sensors, embedded devices, communication protocols, and cloud or server-side services in order to extend digital logic into the physical environment [7, 8].

1.3.2 Role of IoT in the Proposed System

In SmartQdz, IoT technology connects the digital platform to the physical service environment. It enables QR scanners, self check-in kiosks, smart displays, and notification gateways to communicate with the backend in real time. This integration supports fast arrival verification, immediate queue updates, synchronized display of service progression, and better visibility over on-site activity.

By linking physical devices to the queue-management logic, IoT contributes to reducing manual intervention and strengthening the consistency between what happens at the service counter and what users observe through the application.



Figure 1.3: IoT ecosystem supporting real-time queue synchronization in SmartQdz.

1.4 Real-Time Communication

1.4.1 WebSocket Protocol

Real-time queue management requires more than periodic data refresh. When a client is called, served, cancelled, or checked in, the change must be reflected quickly on staff dashboards, mobile applications, and branch displays. WebSocket is a communication protocol that establishes a persistent full-duplex channel between a client and a server, allowing both sides to exchange messages without creating a new HTTP request for each update [44].

In contrast with traditional polling, WebSocket reduces unnecessary repeated requests and improves the responsiveness of event-driven interfaces. This makes it suitable for queue dashboards, live position tracking, counter-status updates, and branch display screens.

1.4.2 Real-Time Architecture in Queue Systems

In a real-time queue architecture, the backend remains the source of truth. Clients send operational actions through authenticated API requests, while the backend validates the action, updates the database, and publishes an event to subscribed clients. Typical events include:

- appointment created.
- QR check-in accepted or rejected.
- queue entry added, called, serving, completed, cancelled, or expired.
- counter status changed.
- branch capacity updated.

For SmartQdz, this model avoids inconsistent local decisions. The mobile application and dashboard do not calculate queue state independently; they receive synchronized updates from the backend after the business rules have been applied.

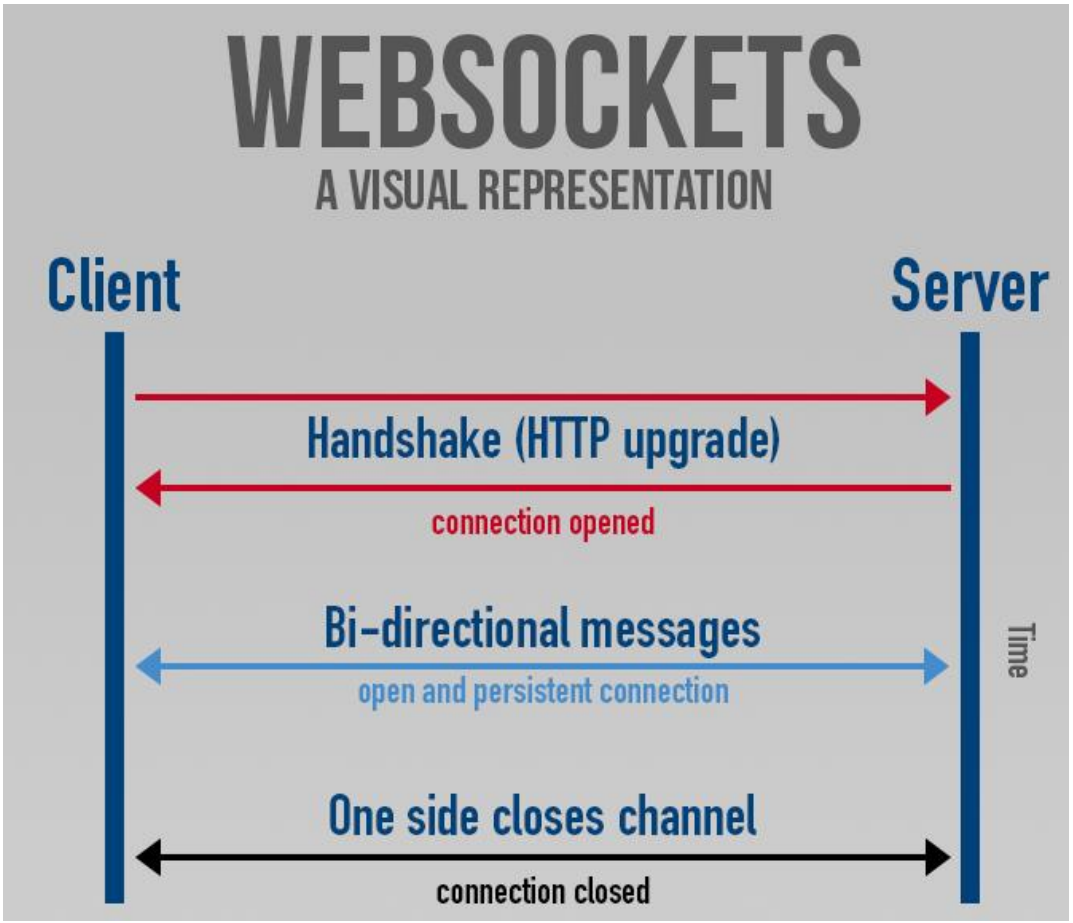


Figure 1.4: Real-time communication model for queue-state synchronization.

1.5 Push Notification Services

Push notifications complement WebSocket communication. WebSocket is useful while the application is open and connected, whereas push notifications can reach users even when the mobile application is in the background. Services such as Firebase Cloud Messaging (FCM) provide infrastructure for delivering notification messages to mobile devices and web clients [45].

In a smart queue-management system, push notifications can be used for:

- booking confirmation.
- appointment reminders.
- successful QR check-in confirmation.
- approaching-turn alerts.
- cancellation or schedule-change notifications.

The notification workflow should be event-driven. After a validated backend state change, the notification service receives a message payload containing the user identifier, event type, and short user-facing content. This separation keeps notification delivery independent from the main queue transaction while preserving traceability through notification logs.

1.6 Queue Management Theory

1.6.1 Basic Queue Models

Queue-management systems organize the order in which service requests are processed. The most common model is First-In First-Out (FIFO), where users are served according to arrival order. Administrative services, however, may require additional rules such as appointment priority, special-needs handling, counter eligibility, and service-specific capacity limits [46].

The proposed system therefore combines FIFO ordering with controlled priority rules. Normal entries follow their queue position inside a branch, counter, date, and time slot. Special cases, such as users with priority status, are handled through explicit rules rather than through informal manual intervention.

1.6.2 Scheduling and Prioritization

Scheduling determines how requests are assigned to service resources. In SmartQdz, a service does not own a physical queue by itself. Instead, services are mapped to eligible counters, and each counter manages its queue for a defined time slot. This design supports several operational requirements:

- balancing load between counters that can process the same service.
- preventing overbooking by applying capacity at slot level.
- preserving branch and counter isolation.
- integrating remote appointments and walk-ins into one operational flow.

1.7 QR Code Technology

1.7.1 Definition of QR Code

A QR code (Quick Response code) is a two-dimensional barcode capable of storing digital information in a compact and machine-readable format. Compared with traditional one-dimensional barcodes, QR codes offer higher data density, faster scanning, and greater flexibility across authentication, ticketing, access control, and payment scenarios [9].

1.7.2 Use of QR Codes in the System

In the proposed system, QR codes are used as digital carriers for queue-related tokens. After an appointment is confirmed or a walk-in ticket is issued, the backend generates a unique token and provides it to the user through the mobile interface. At the branch, the token is scanned and verified against server-side rules before the queue status is updated.

This approach offers several operational benefits:

- fast and contactless identification at the point of service.
- reduced dependence on paper-based tickets.
- better control over reservation ownership and arrival status.
- improved traceability of queue events.

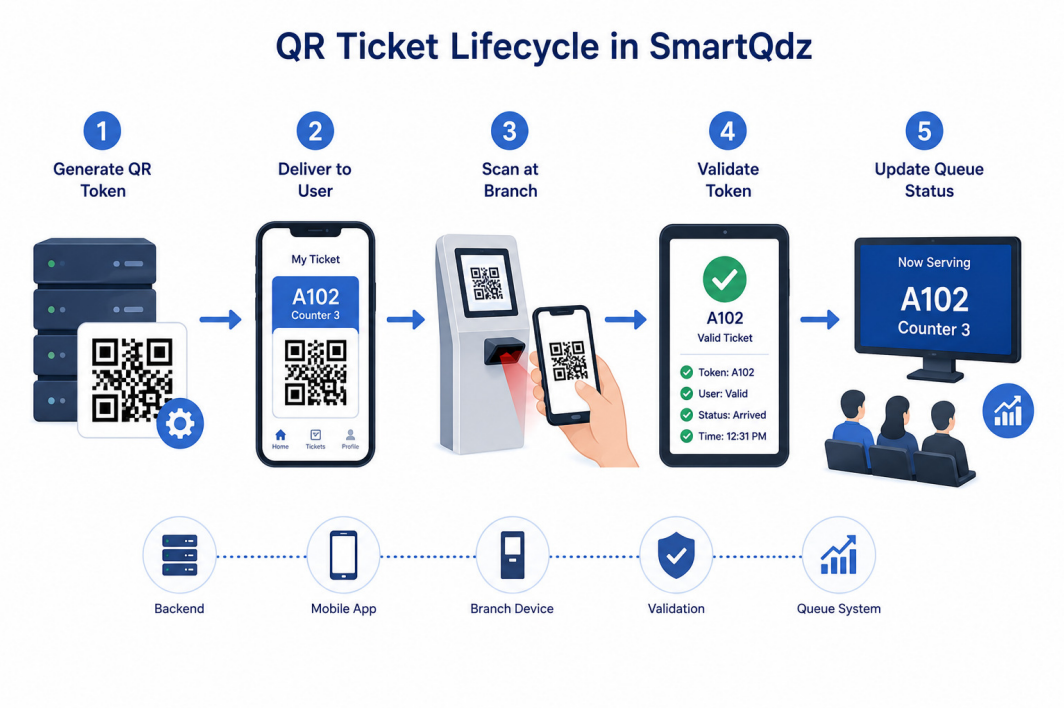


Figure 1.5: Lifecycle of the QR-based ticket used by SmartQdz.

1.8 QR-Based Authentication

1.8.1 Concept of Digital Authentication

Authentication is the process of verifying the identity of a user or device before granting access to a service or resource. In digital systems, authentication may rely on passwords, one-time passwords, biometric factors, tokens, certificates, or federated identity mechanisms [15, 12].

In SmartQdz, authentication is required at two levels. First, users authenticate themselves to access the platform and manage their reservations. Second, the system authenticates the QR-based arrival token in order to confirm that the person presenting the reservation is associated with a valid and active queue entry.

1.8.2 QR-Based Authentication Workflow

The QR-based workflow proposed for SmartQdz can be summarized as follows:

1. the backend generates a unique token linked to a reservation or ticket.
2. the token is stored securely with metadata such as creation time, expiration time, and usage status.

3. the user presents the QR code at the service location.
4. a scanner or kiosk sends the token to the backend for validation.
5. the backend checks authenticity, validity period, and usage state before updating the queue.

This workflow strengthens the relation between digital reservation and physical presence, which is a central requirement for administrative queue management.

1.8.3 Token Lifecycle

The QR image displayed to the user should not be treated as the security mechanism itself. The important element is the backend token encoded inside the QR code. Its lifecycle includes the following stages:

1. **Generation:** the backend creates a high-entropy token after a booking is accepted.
2. **Association:** the token is linked to one appointment, user, branch, service, counter, date, and slot.
3. **Activation:** the token becomes usable only during the authorized check-in window.
4. **Validation:** the scanner sends the token to the backend, which verifies status, expiration, ownership context, and queue eligibility.
5. **Consumption:** after successful check-in, the token is marked as used and cannot be submitted again.
6. **Invalidation:** cancellation, expiration, completion, or administrative rejection makes the token unusable.

This lifecycle limits replay attempts and ensures that a QR code cannot be reused outside its intended operational context.



Figure 1.6: QR-based authentication workflow in SmartQdz.

1.8.4 Advantages of QR-Based Authentication

QR-based authentication enables rapid verification, reduces manual errors, limits person-to-person contact during check-in, and supports better traceability than purely paper-based procedures. When combined with backend validation and expiration control, it also helps prevent replay, duplication, and misuse of queue tickets.

1.9 Security and Cryptography in the System

1.9.1 Importance of Information Security

Because SmartQdz processes personal identifiers, booking data, queue events, and administrative actions, information security is a core design requirement. The platform must preserve the confidentiality, integrity, and availability of data while protecting operational trust in the queue itself [10, 11].

1.9.2 Applied Security Mechanisms

The proposed system incorporates several complementary security mechanisms:

- secure transport channels such as HTTPS/TLS for API communication.
- protected credential storage using secure password-hashing practices.
- JSON Web Token (JWT)-based authentication for secure session management and user identity verification.
- time-limited tokens to reduce the risk of reuse.
- server-side validation of every scanned token.
- role-based access control for staff and administrators.
- activity logging to preserve operational traceability.

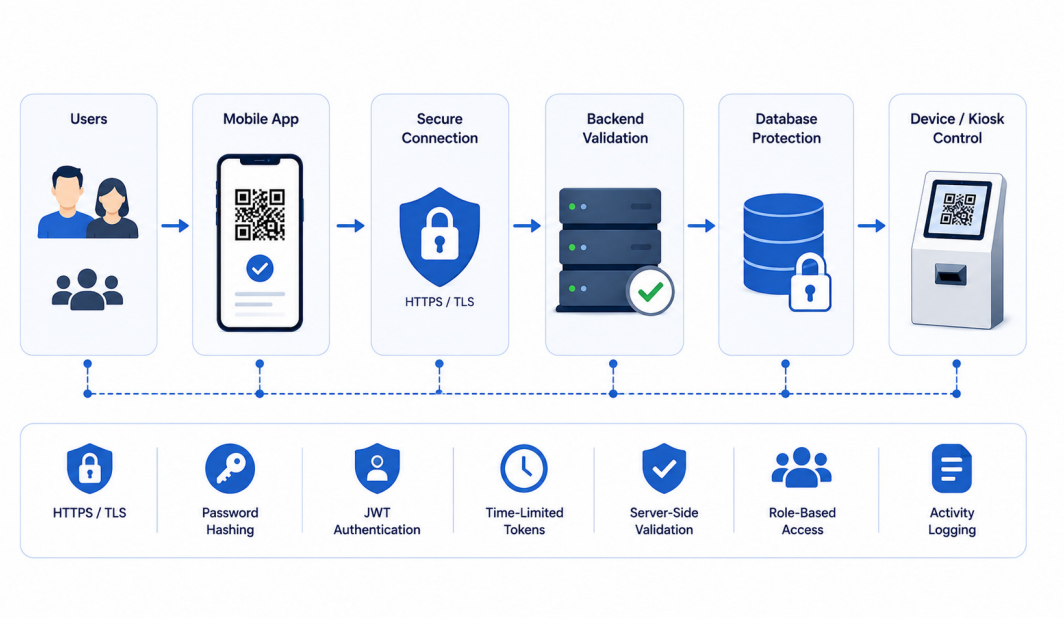


Figure 1.7: Security layers applied in the SmartQdz platform.

1.9.3 Potential Threats and Mitigation

Several risks may affect a queue-management platform, including credential theft, token duplication, replay attempts, unauthorized access to administrative features, and exposure of data during transmission. These threats can be mitigated through encrypted communication, short-lived tokens, strict authorization rules, secure backend checks, and controlled device integration [15, 12].

1.10 Conclusion

This chapter introduced the main technological foundations required for the conception of SmartQdz. It showed that mobile and desktop applications provide the interaction layer, IoT extends the system into the physical service environment, QR codes support practical and fast identity linking, and modern security mechanisms protect the integrity of queue operations.

These elements together establish a coherent technical basis for a secure and efficient smart queue-management platform. The next chapter examines existing solutions in order to position SmartQdz within the broader landscape of digital queue-management systems.

Chapter 2

Existing Solutions and Comparative Study

2.1 Introduction

Before presenting the proposed solution, it is necessary to review representative queue-management platforms already available on the market. Such a review helps identify mature design patterns, common functional capabilities, and the architectural trade-offs adopted by existing vendors. It also clarifies where SmartQdz should differentiate itself in order to better fit the needs of administrative institutions.

The comparative study developed in this chapter focuses on publicly documented features related to remote booking, arrival verification, device integration, user communication, operational visibility, and deployment complexity. Unless explicitly stated otherwise, the analysis below is based on official public product materials consulted on May 9, 2026 [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26].

2.2 Selection Criteria

The platforms selected for this comparison were chosen because they represent different orientations in the queue-management market. The selection was guided by the following criteria:

- **functional relevance:** the solution must address queue management, appointment scheduling, digital ticketing, or virtual waiting.
- **public documentation:** enough public product information must be available to

support a fair academic comparison.

- **architectural diversity:** the selected platforms should include software-centric, cloud-oriented, hardware-integrated, and API-enabled approaches.
- **administrative applicability:** the solution should be relevant to public institutions, service branches, reception desks, or high-volume front-office operations.
- **comparison value:** each platform should highlight a design trade-off that helps position the proposed SmartQdz architecture.

The objective of the comparison is not to rank vendors commercially, but to identify engineering decisions that matter for the proposed system: deployment model, real-time support, device integration, arrival verification, administrative control, and expected operational complexity.

2.3 Comparative Study of Existing Solutions

2.3.1 Qmatic

Qmatic positions itself as a customer-journey and queue-management platform combining appointment management, queue orchestration, virtual queuing, and operational analytics. Its official product materials emphasize omnichannel service delivery, enterprise-grade orchestration, and cloud-based operational visibility [16, 17].

Strengths.

- strong integration of queue management, appointments, and operational analytics.
- mature enterprise orientation with centralized supervisory capabilities.
- publicly visible attention to security and certification posture [18].

Limitations in relation to the present project.

- the platform is oriented toward large-scale and often hardware-rich deployments.
- this level of enterprise orchestration may be excessive for smaller local administrative branches with moderate budgets.

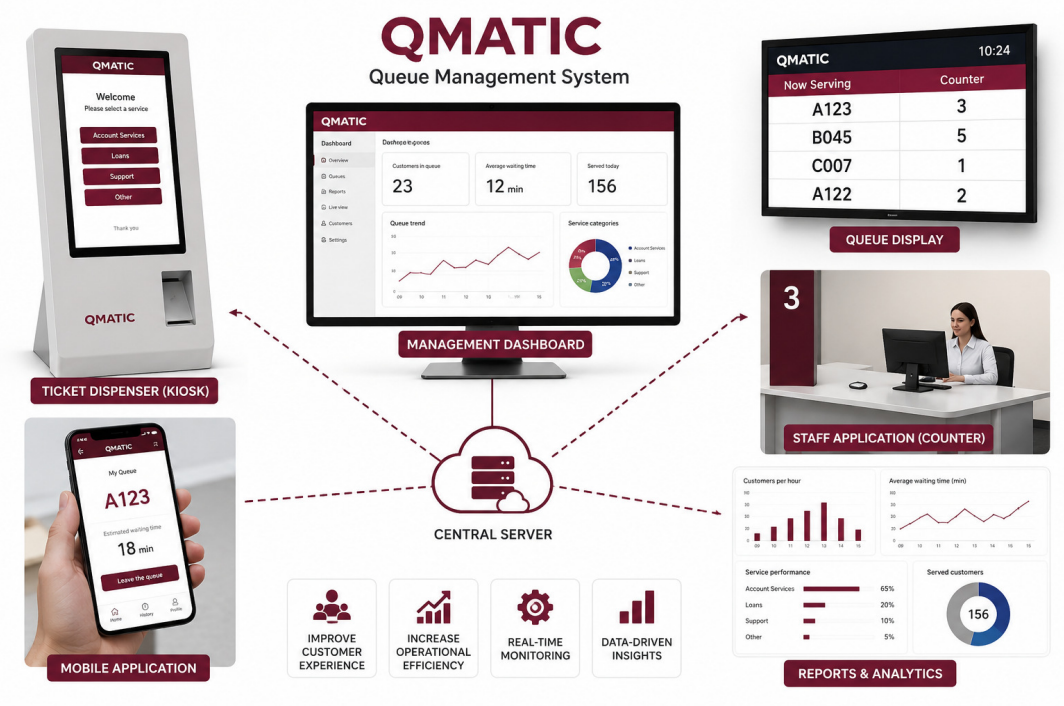


Figure 2.1: Traditional Queue Management System (Qmatic)

2.3.2 Skiplino

Skiplino presents a cloud-based queue-management and appointment platform designed to reduce waiting time and improve crowd management. Its official website highlights remote booking, branch visibility, SMS and push notifications, and fast setup. In some product lines, Skiplino also documents QR-code delivery to users [19, 20].

Strengths.

- strong support for remote booking and appointment scheduling.
- emphasis on rapid deployment and ease of adoption.
- good customer communication through notifications and branch-status visibility.

Limitations in relation to the present project.

- the public materials reviewed are less explicit about a backend-centered, cryptographically controlled self check-in model.
- the platform appears more software-centric than device-centric, which may reduce the visibility of integrated on-site verification workflows.



Figure 2.2: Cloud-Based Queue Management Solution (Skiplino)

2.3.3 QLess

QLess is a virtual queuing platform that emphasizes mobile-first waiting, appointment scheduling, callback queuing, and real-time communication. Its official product pages focus on freeing users from physical lines while preserving operational insight for staff and managers [21, 22].

Strengths.

- strong virtual waitlist and appointment capabilities.
- real-time notifications and communication with users.
- unified treatment of walk-ins, appointments, virtual visits, and callbacks.

Limitations in relation to the present project.

- the consulted public materials do not foreground a QR-based arrival-verification model.
- the solution is particularly strong in communication and virtual waiting, but less explicitly documented around secure physical self check-in.



Figure 2.3: Virtual Queue Management Platform (QLess)

2.3.4 Wavetec

Wavetec offers queue-management solutions with notable attention to physical-device ecosystems, including mobile queuing, signage, branch selection, and integrated operational flows. Its official pages describe a MobileQ solution through which users can enter a queue or book an appointment, receive updates, and monitor waiting information [23, 24].

Strengths.

- strong connection between mobile queueing and on-site operational infrastructure.
- explicit support for remote queue entry, appointment handling, and real-time waiting information.
- suitable for environments that require synchronized digital and physical queue management.

Limitations in relation to the present project.

- hardware-oriented deployments may increase integration complexity and cost.
- the public materials reviewed focus more on queue participation and customer flow than on signed, expiring, backend-validated QR tokens.



Figure 2.4: Enterprise Queue Management Solution (Wavetec)

2.3.5 Lineberty

Lineberty provides a digital-ticketing and queue-digitalization platform that supports waiting with and without appointments. Its official website and developer documentation describe digital tickets, real-time queue tracking, notifications, and APIs for booking, display, and validation workflows [25, 26].

Strengths.

- clear focus on replacing physical waiting with digital tickets.
- support for appointments, queue progression, and user notification.
- availability of APIs that can support customized integration scenarios.

Limitations in relation to the present project.

- the public positioning is centered on digital ticketing and customer-experience optimization rather than administrative workflow governance.
- compared with a dedicated administrative control platform, public materials place less emphasis on centralized staff and branch governance requirements.

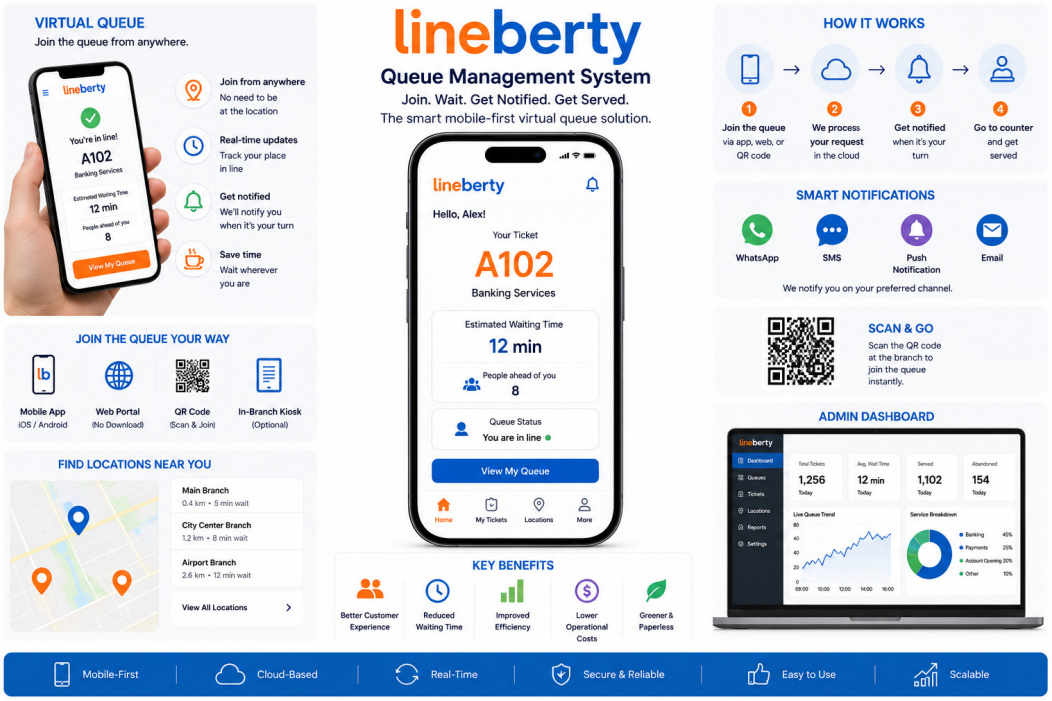


Figure 2.5: Mobile Virtual Queue Platform (Lineberty)

2.4 Comparative Summary Table

Table 2.1: Qualitative comparison of existing solutions and the SmartQdz proposal

Solution	**Remote Booking**	**Arrival Verification**	**Device Integration**	**Operational Visibility**	**Deployment Complexity**
Qmatic	Yes	Documented	Strong	Strong	High
Skiplino	Yes	Partially documented	Moderate	Strong	Low to moderate
QLess	Yes	Not explicit on QR-based verification	Moderate	Strong	Moderate
Wavetec	Yes	Documented	Strong	Strong	High
Lineberty	Yes	Ticket validation documented	Moderate	Moderate	Low to moderate
SmartQdz	Yes	Dynamic QR with backend validation	Strong	Strong	Moderate

The table above is a qualitative synthesis derived from official public product descriptions. When a feature is marked as *not explicit* or *partially documented*, this does not prove its absence in the corresponding product; it means that the consulted public materials did not describe it in a sufficiently detailed way for this dissertation.

2.5 Technical and Economic Comparison

The previous table summarizes functional characteristics. A deeper technical comparison is also necessary because two platforms may offer similar visible features while relying on very different deployment and integration models.

Table 2.2: Technical comparison of queue-management solutions

Solution	**Architecture Orientation**	**Real-Time Support**	**QR / Arrival Security**	**Scalability and Integration**
Qmatic	Enterprise journey-management platform with strong branch orchestration.	Strongly oriented toward live branch operations.	Arrival handling is documented, but token internals are not fully visible in public material.	Suitable for large deployments, but may require higher configuration and hardware investment.
Skiplino	Cloud-based queue and appointment platform.	Supports live visibility and user communication.	QR delivery appears in some product descriptions, but backend security details are limited publicly.	Easier adoption; integration depth depends on available product modules.
QLess	Mobile-first virtual waiting and appointment platform.	Strong communication and queue-update orientation.	QR-based physical check-in is not a central public emphasis.	Scales well for virtual waiting workflows; physical-device integration is less foregrounded.
Wavetec	Hardware-integrated queue ecosystem combining mobile, signage, and branch devices.	Strong branch-level operational synchronization.	Arrival and ticket workflows are documented at product level.	Powerful for large sites, but deployment and maintenance costs may be higher.
Lineberty	Digital-ticketing and API-enabled queue platform.	Supports ticket state and user notifications.	Ticket validation is documented through available API concepts.	Flexible integration potential through APIs; institutional governance depends on configuration.
SmartQdz	Prototype client-server architecture with mobile, dashboard, backend, database, and IoT devices.	WebSocket-based synchronization and push notifications are part of the proposed model.	Dynamic QR token with expiration, one-time use, and backend validation.	Designed for moderate-cost administrative deployment with controlled extensibility.

2.5.1 Deployment Complexity and Cost Considerations

Public pricing information is not always available for commercial queue-management vendors, and exact cost depends on licensing, number of branches, devices, support contracts, hosting choices, and integration services. For that reason, this dissertation uses a qualitative cost analysis instead of vendor-specific pricing.

Table 2.3: Qualitative cost factors affecting deployment

Cost Factor	**Impact on Queue-Management Deployment**
Software licensing	Commercial platforms may require subscription fees per branch, user, counter, or module. A custom prototype reduces licensing dependency but increases development responsibility.
Hardware devices	Kiosks, scanners, displays, printers, and local controllers can significantly increase initial deployment cost.
Integration effort	Connecting the queue system to identity services, SMS gateways, payment services, or institutional databases requires additional engineering work.
Maintenance and support	Commercial vendors may include support contracts, while custom solutions require local technical maintenance and monitoring.
Hosting and scalability	Cloud hosting simplifies access and scaling, but introduces recurring infrastructure costs and dependency on internet connectivity.

Within this cost perspective, SmartQdz aims for a moderate deployment model. It avoids a fully hardware-heavy architecture while still supporting essential connected devices such as QR scanners and branch displays. This balance is important for administrative institutions that need practical modernization without adopting an expensive enterprise ecosystem from the beginning.

2.5.2 Open-Source and Proprietary Considerations

Most mature queue-management platforms are proprietary products. Proprietary systems can provide stable support, polished interfaces, vendor maintenance, and ready-made deployment packages. However, they may also limit source-code access, restrict customization, impose licensing costs, and create vendor dependency.

An open-source or custom-built approach provides more control over data models, integration rules, localization, security logic, and institutional workflows. Its disadvantage is that the institution or development team becomes responsible for maintenance, updates, testing, deployment, and long-term support.

SmartQdz follows a custom prototype approach built with widely used technologies such as Flutter, React/Next.js, Laravel, and a relational database. This gives the project flexibility to adapt queue rules, QR validation, authorization, and local administrative constraints. At the same time, it requires disciplined engineering practices to reach the reliability level expected from production systems.

2.6 Discussion

The comparative review shows that contemporary queue-management solutions generally converge around a shared objective: reducing physical waiting and improving flow visibility through digital interaction. However, they diverge in architectural emphasis. Qmatic and Wavetec give substantial weight to enterprise orchestration and device-enabled branch operations. Skiplino emphasizes cloud deployment and ease of adoption. QLess strongly prioritizes flexible virtual waiting and multi-channel communication. Lineberty focuses on digital ticketing and queue digitization with and without appointments.

For the specific context of administrative services, the key design challenge is not merely to reproduce the largest possible feature set. Instead, the objective is to select a balanced architecture that preserves operational visibility while remaining affordable, understandable, and secure for local branches. In this respect, SmartQdz aims to combine several desirable characteristics:

- remote access and ease of use for citizens.
- clear arrival verification through dynamic QR tokens.
- moderate device integration without excessive hardware dependence.
- centralized control for staff and administrators.
- secure backend validation and traceable queue-state management.

This balance differentiates SmartQdz from purely virtual-waiting platforms on one side and from heavy enterprise hardware ecosystems on the other. The proposed solution is therefore positioned as a context-appropriate platform for administrative institutions that require both service modernization and operational realism.

2.7 Conclusion

This chapter reviewed a set of representative queue-management platforms and highlighted their main strengths, orientations, and limits with respect to the target context of this dissertation. The analysis confirms that existing solutions already address remote booking, queue visibility, and communication in mature ways, but they do not all foreground the same level of secure arrival verification and administrative workflow alignment.

The comparison therefore justifies the design direction of SmartQdz, which seeks to integrate citizen accessibility, moderate deployment complexity, real-time operational control, and secure QR-based self check-in in a single system. The next chapter develops this proposal through a detailed presentation of the platform architecture and core design artifacts.

Chapter 3

Design of the Proposed System

3.1 Introduction

This chapter presents the design of the proposed SmartQdz platform and translates the needs identified in the previous chapters into a structured technical solution. The design is guided by four major principles: modularity, security, operational traceability, and scalability. In practical terms, the platform must support remote booking, secure self check-in, real-time queue handling, and centralized administrative supervision without fragmenting the service workflow across disconnected tools.

The main design objectives can be summarized as follows:

- provide citizens with a clear digital channel for account management, booking, and queue tracking.
- ensure secure and fast arrival verification through dynamic QR tokens and backend validation.
- assist staff members through a real-time operational interface.
- offer administrators a centralized dashboard for organizational and service governance.
- maintain interoperability between applications, backend services, databases, and connected devices.

3.2 General Architecture of SmartQdz

At a high level, SmartQdz is designed as a coordinated digital ecosystem rather than a single isolated application. It combines a citizen-facing mobile client, a web-based administrative dashboard, a centralized backend API, a relational database, and connected branch devices. This organization ensures that booking, validation, queue execution, and supervision remain governed by one coherent operational model.

Although the detailed development environment is presented in Chapter 4, some technological choices already belong to the design stage because they shape integration boundaries, client capabilities, and deployment constraints. For that reason, the selected development architecture is summarized textually in the following sections.

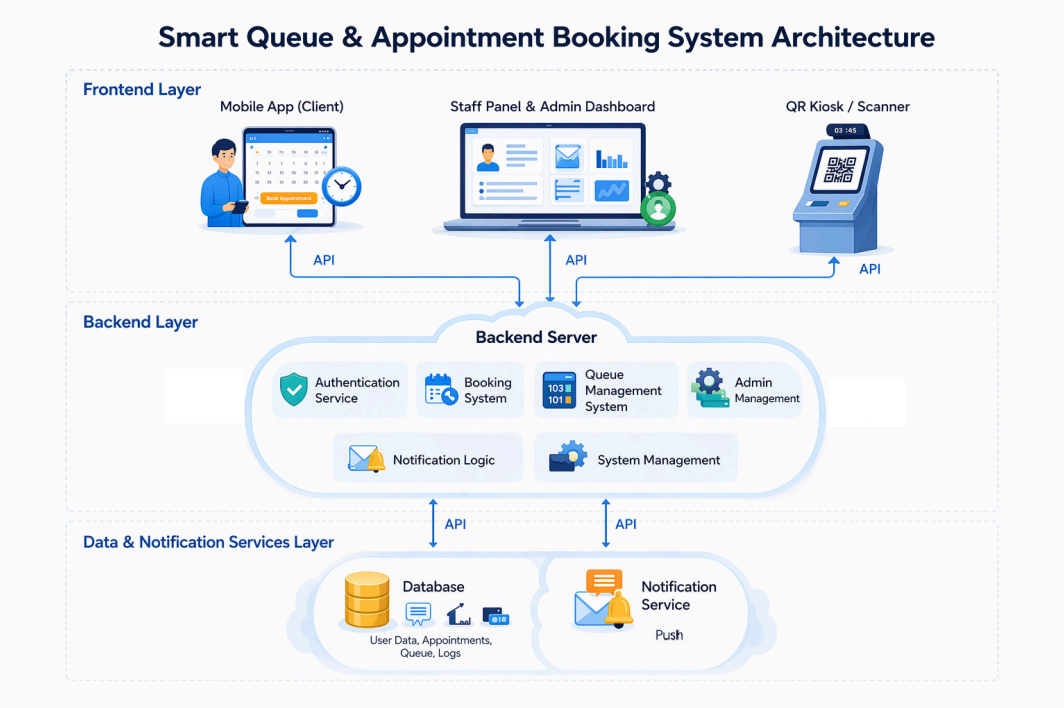


Figure 3.1: General architecture of the SmartQdz platform.

3.3 System Architecture

The design of this project utilizes a **Client–Server / N-Tier Layered Architecture**. In this architecture, the system components are separated into multiple layers, where each layer has a specific responsibility. The clients do not communicate directly with the database; instead, all requests pass through the central server, which processes requests, applies system rules, and manages data access.

3.3.1 Main Components of the System

The main components of the system are:

- **Mobile Application**
  Used by citizens to register, browse available companies and branches, select services, and create appointments.
- **Admin Dashboard**
  Used by administrators and employees to manage branches, services, counters, schedules, appointments, and queue operations.
- **Kiosk / Walk-in Interface**
  A touchscreen interface located inside branches that allows visitors to create walk-in tickets.
- **QR Scanner Device**
  A local device used to scan appointment QR codes and process customer check-ins.
- **Central Server**
  The core component responsible for processing requests, applying business rules, managing queues, and coordinating communication between all system components.
- **Database**
  Stores all system information including users, companies, branches, services, appointments, tickets, queue data, and operational records.

3.3.2 System Layers

Presentation Layer

This layer contains the client-side interfaces of the system, including:

- Mobile application
- Administrator dashboard
- Kiosk application
- QR scanner device

These interfaces allow users and staff to interact with the system by sending requests and displaying responses.

Application Layer

The Application Layer is responsible for receiving and processing requests coming from the different client applications. In this layer:

- User and device requests are authenticated.
- Input data is validated.
- Requests are routed to the appropriate services.
- Structured responses are returned to clients.

Business Logic Layer

The Business Logic Layer contains the core operational rules of the system, including:

- Appointment booking rules
- Queue management and prioritization
- Counter assignment
- Capacity management
- QR code validation
- Access control and permissions
- Branch and counter isolation

Data Access Layer

The Data Access Layer is responsible for managing communication with the database. It organizes data retrieval, storage, and updates in a controlled and structured manner.

Database Layer

The Database Layer consists of a relational database used to store all application data while ensuring data integrity and relationships between entities.

3.3.3 Role of the Central Server

The central server acts as the core authority of the system and is responsible for coordinating all operations. Its responsibilities include:

- Managing appointments
- Processing queues
- Assigning counters
- Managing service capacity
- Validating QR codes
- Enforcing permissions and access control
- Maintaining branch and counter organization

All system components communicate with the central server by sending requests and receiving responses.

3.3.4 Why This Architecture is Appropriate

This architecture is suitable because the system includes multiple clients that must operate according to the same centralized rules for appointment booking and queue management.

Centralizing the business logic in the backend provides several advantages:

- Improved security
- Prevention of duplicated logic across applications
- Easier maintenance
- Better scalability
- Simplified integration of future clients and devices

3.4 Real-Time Synchronization Architecture

Real-time synchronization is a core design requirement because queue information changes continuously during service execution. The proposed architecture separates two types of communication:

- **command requests:** authenticated HTTP API requests used to create bookings, validate QR codes, call clients, start service, complete service, or cancel entries.
- **state broadcasts:** WebSocket events emitted by the backend after the command is validated and the database transaction is completed.

This separation ensures that clients do not update critical queue state by themselves. They request an action, the backend applies authorization and business rules, and only then does the backend broadcast the resulting state to subscribed clients.

Table 3.1: Main real-time events in the proposed design

Event	**Triggered By**	**Subscribed Clients**
`QueueEntryCreated`	appointment confirmation or walk-in creation	mobile app, dashboard, branch display
`CheckInValidated`	accepted QR scan	dashboard, branch display, mobile app
`QueueEntryCalled`	staff call-next action	mobile app, display screen, staff dashboard
`ServiceStarted`	staff begins service	dashboard, administrator view
`ServiceCompleted`	staff closes service session	dashboard, reports module, mobile app
`QueueEntryCancelled`	user cancellation, admin action, or timeout	mobile app, dashboard, branch display

Channels are scoped by branch, counter, date, and slot. For example, a staff dashboard subscribed to a specific counter should not receive private updates from unrelated branches. This scope reduces unnecessary traffic and reinforces operational isolation.

3.5 Detailed Functional Architecture

The detailed functional architecture explains how the main modules cooperate from booking to service completion. Each module is responsible for a clearly delimited subset of functions while relying on backend validation and shared persistence to maintain consistency.

3.5.1 Client and Booking Module

The client module is the entry point of the system for citizens. It supports account creation, login, branch selection, service selection, appointment booking, and walk-in ticket requests. Once the booking request is submitted, the backend validates the request context, checks business rules, generates the corresponding queue token, and returns a confirmation through the mobile interface and the configured notification channel.

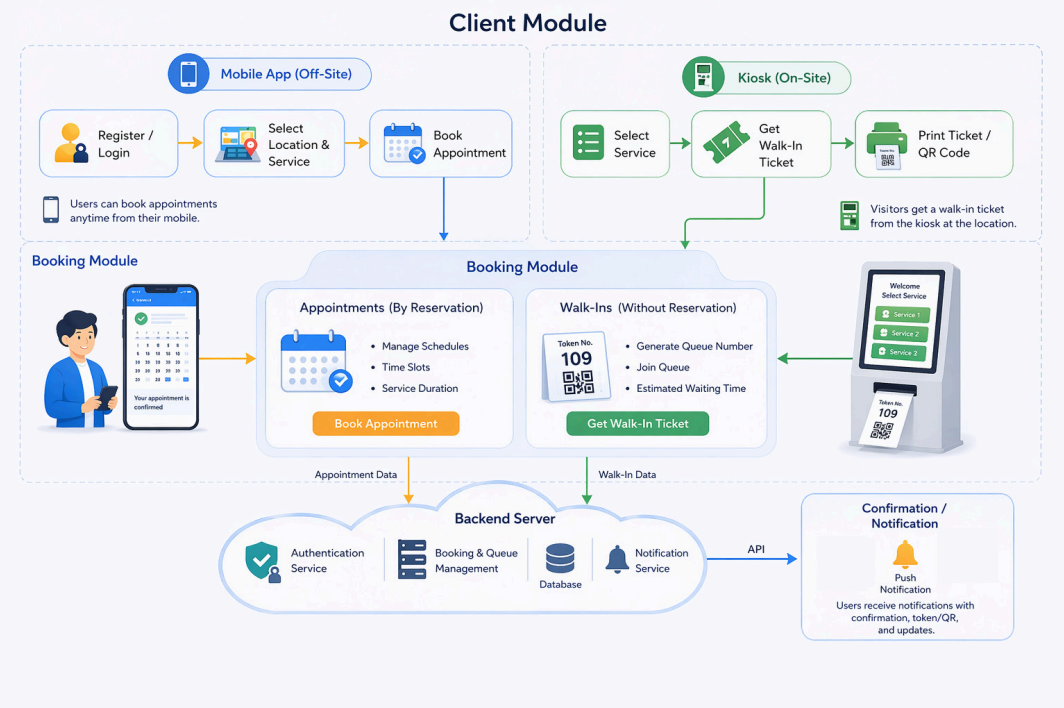


Figure 3.2: Client and booking module architecture and workflow.

3.5.2 QR Self Check-in Module

The QR self check-in module manages the transition from digital reservation to confirmed physical presence. When the user arrives at the branch, the QR token is scanned through a kiosk or scanner. The backend verifies the token value, checks expiration and usage status, updates the queue state, and logs the event. This mechanism reduces fraud, prevents duplicate use of a ticket, and aligns the queue state with real-world arrival events.

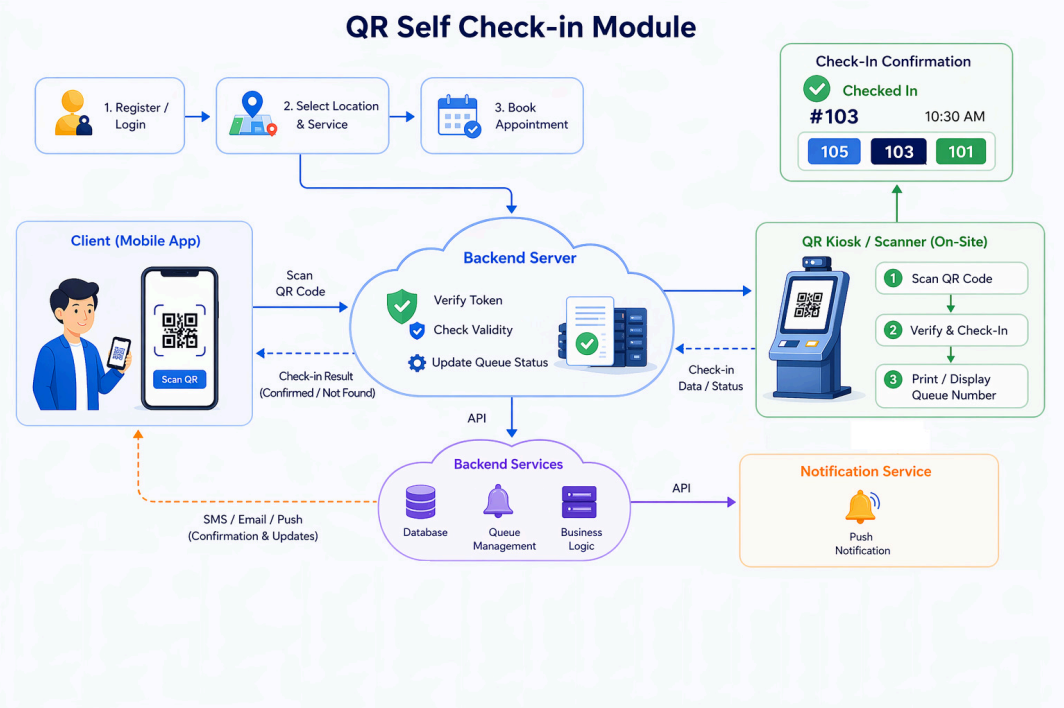


Figure 3.3: QR self check-in module architecture and workflow.

3.5.3 Staff Operations Module

The staff operations module supports front-office personnel during service execution. Through the operational interface, staff members can view the active queue, call the next client, start service, update the current state, end service, and record the final outcome. Each action is transmitted to the backend, which updates the corresponding queue entry and preserves a traceable service history.

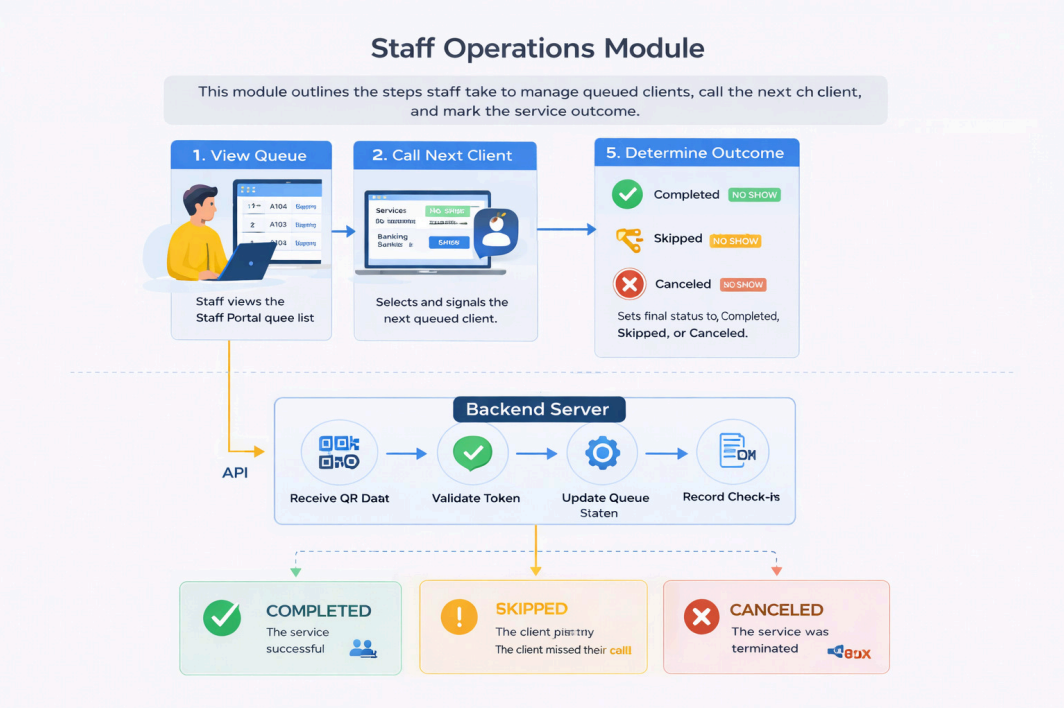


Figure 3.4: Staff operations module architecture and workflow.

3.5.4 Administration and Backend Module

The administration and backend module constitutes the control layer of the system. It centralizes organizational management, service configuration, queue rules, reporting, and user-role administration. It also hosts the backend services responsible for authentication, notifications, booking validation, queue coordination, and data persistence.

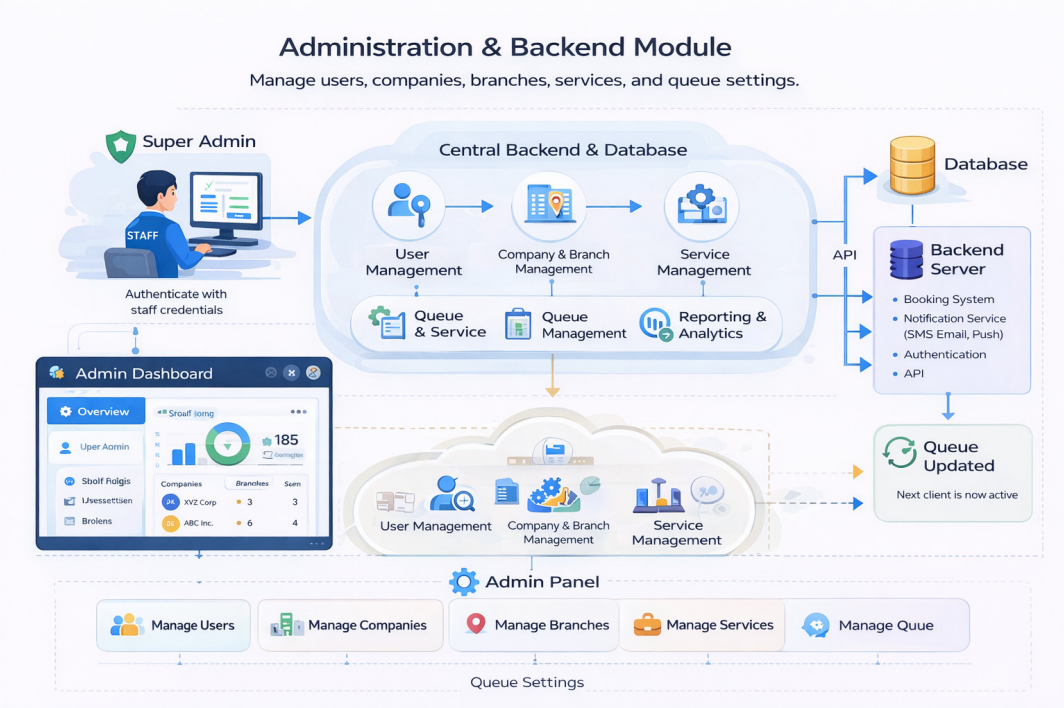


Figure 3.5: Administration and backend module architecture and workflow.

3.6 Queue Lifecycle

The queue lifecycle defines how a reservation or ticket moves through the system. Making this lifecycle explicit is important because each transition must be authorized, validated, logged, and synchronized in real time.

Table 3.2: Queue-entry lifecycle states

State	Meaning	Allowed Next States
`booked`	remote appointment created but not yet physically confirmed	`checked_in`, `cancelled`, `expired`
`checked_in`	QR validation accepted and arrival confirmed	`waiting`, `cancelled`
`waiting`	user is present and visible in the operational queue	`called`, `cancelled`, `no_show`
`called`	staff has called the user to a counter	`serving`, `no_show`, `waiting`
`serving`	service is currently being processed	`completed`, `cancelled`
`completed`	service session ended successfully	final state
`cancelled`	reservation or ticket was cancelled before completion	final state
`expired`	check-in window or slot validity ended	final state
`no_show`	user did not appear after being called	final state or administrative recovery

Walk-in tickets enter the lifecycle closer to the operational queue because the user is already physically present. Remote appointments must pass through QR check-in before they can become active waiting entries. This distinction prevents future reservations from consuming live queue positions before arrival.

3.7 Queue Prioritization Logic

The default ordering rule is FIFO within the same branch, counter, date, and time slot. However, the system also supports controlled priority cases. Priority must be explicit and auditable; it should not depend on manual reordering without trace.

1. entries are grouped by the queue-session scope: branch, counter, date, slot start, and slot end.
2. entries with special-needs priority are marked with a priority flag.
3. entries are ordered first by priority level, then by check-in or ticket creation time.
4. ties are resolved by queue-entry identifier or booking reference number.
5. every priority decision remains visible in the operational log.

This approach preserves fairness for normal users while allowing the institution to implement justified priority rules.

3.8 Main Actors and Functionalities

The complete service lifecycle of SmartQdz involves several actors, each with a distinct role and permission scope:

3.8.1 Citizens / Clients

Citizens are the primary beneficiaries of the platform. Their functionalities include account creation, authentication, branch and service selection, appointment reservation, walk-in ticket generation, QR-token reception, queue tracking, and self check-in.

3.8.2 Receptionists and Service Staff

Receptionists and service agents manage the operational phase of the process. They consult waiting lists, call the next client, start and update service sessions, and record outcomes such as completion, cancellation, or no-show.

3.8.3 Administrators / Super Administrators

Administrative users supervise the platform at the organizational level. They manage branches, services, schedules, users, permissions, and queue rules. Higher-level administrators can additionally oversee cross-branch or platform-wide settings.

3.8.4 Smart Devices and Connected Components

Connected components include QR kiosks, scanners, display screens, notification gateways, and supporting backend infrastructure. Their role is to bridge the physical service environment and the digital queue logic.

3.9 Device Communication

The Raspberry Pi scanner and other connected devices communicate with the backend as controlled clients. Each device is registered with a device identifier, branch association, and authentication credential. During operation, the device scans a QR code, extracts the token payload, and sends a validation request to the backend.

The backend verifies the device, checks that it belongs to the correct branch, validates the token, records the check-in result, and returns a minimal response suitable for the local interface. When the validation succeeds, the backend also broadcasts a queue update to dashboards and displays.

Table 3.3: Device communication responsibilities

Component	**Responsibility**
Raspberry Pi scanner	Captures QR code and sends token validation request.
Kiosk interface	Creates same-day walk-in tickets and displays local confirmation.
Backend API	Authenticates device, validates token, applies queue rules, and logs result.
Database	Stores device identity, token status, check-in records, and queue entries.
WebSocket broadcaster	Sends accepted queue-state changes to subscribed clients.
Display screen	Shows current ticket, next ticket, and counter information.

3.10 Functional Scope Summary

The functional scope clarifies how the main responsibilities are distributed between clients, staff, administrators, and connected devices.

Table 3.4: Functional scope summary

Aspect	**Interpretation**
Client-side functions	Cover registration, booking, ticket generation, follow-up, and cancellation.
Self check-in functions	Ensure that physical arrival is linked to secure digital validation.
Staff-side functions	Cover queue execution, service handling, and the recording of outcomes.
Administrative functions	Support organizational governance, configuration, supervision, and reporting.

3.11 Sequence Diagrams

Sequence diagrams describe the chronological exchange of messages between users, devices, and technical components. They are especially useful for clarifying the authentication flow, booking confirmation, QR-based check-in, and the service-execution workflow.

3.11.1 Booking Process Sequence

The booking process sequence illustrates how SmartQdz handles both remote appointments and on-site walk-in tickets. In the first scenario, the citizen uses the mobile application to select a branch, service, and time slot, while the backend validates the request, assigns the appropriate counter, and generates a secure QR token. In the second scenario, a visitor uses the kiosk to request a walk-in ticket, after which the backend creates the queue entry and applies priority rules when necessary. This workflow ensures synchronized queue management, secure validation, and real-time operational coordination between clients, staff, and backend services.

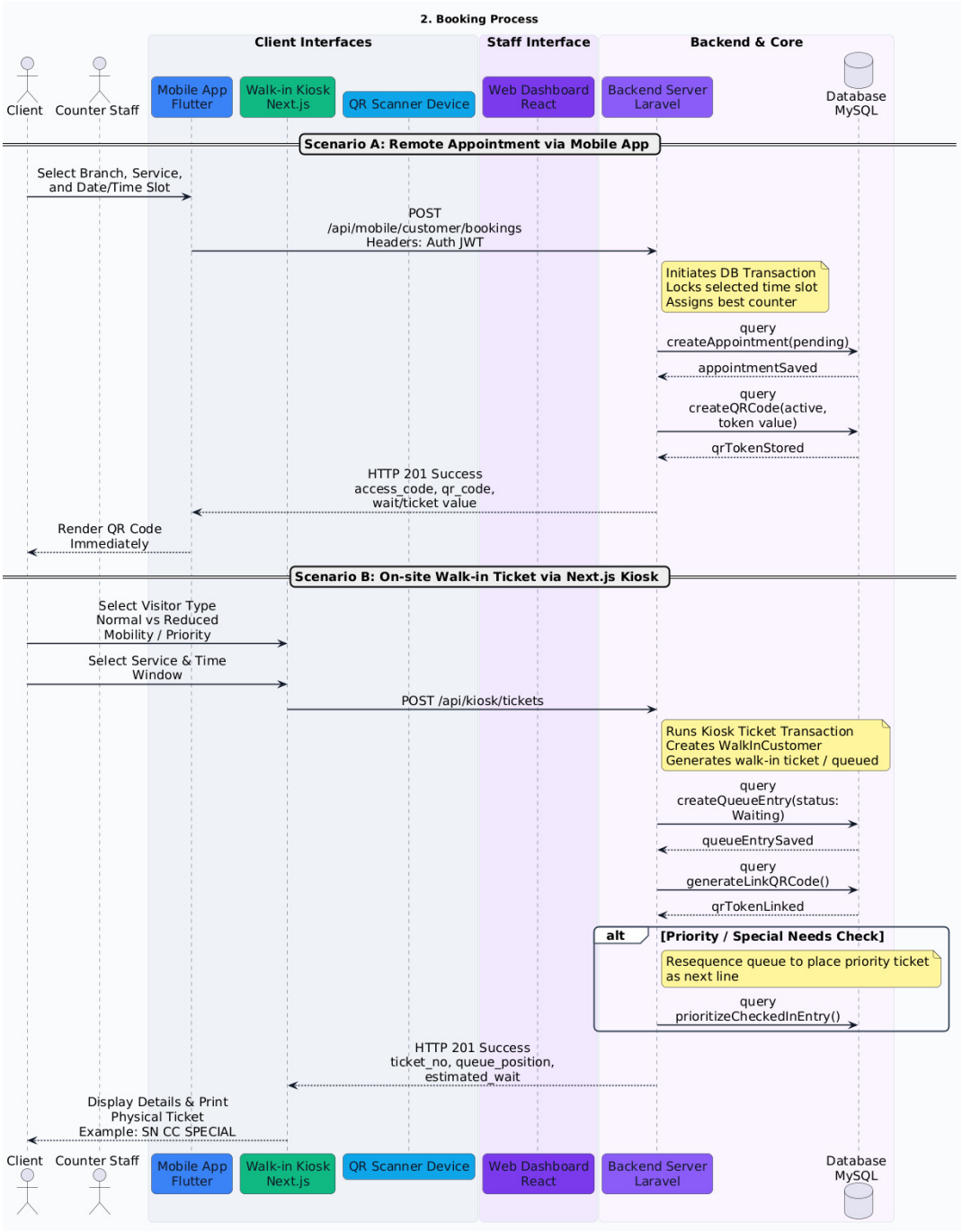


Figure 3.6: Booking process sequence diagram.

3.11.2 Authentication and Booking Sequence

The authentication and booking sequence begins when a user submits credentials through the mobile application or dashboard. The backend validates the credentials, issues a JWT access token, then verifies protected booking requests before applying branch, service, slot, and counter-assignment rules.

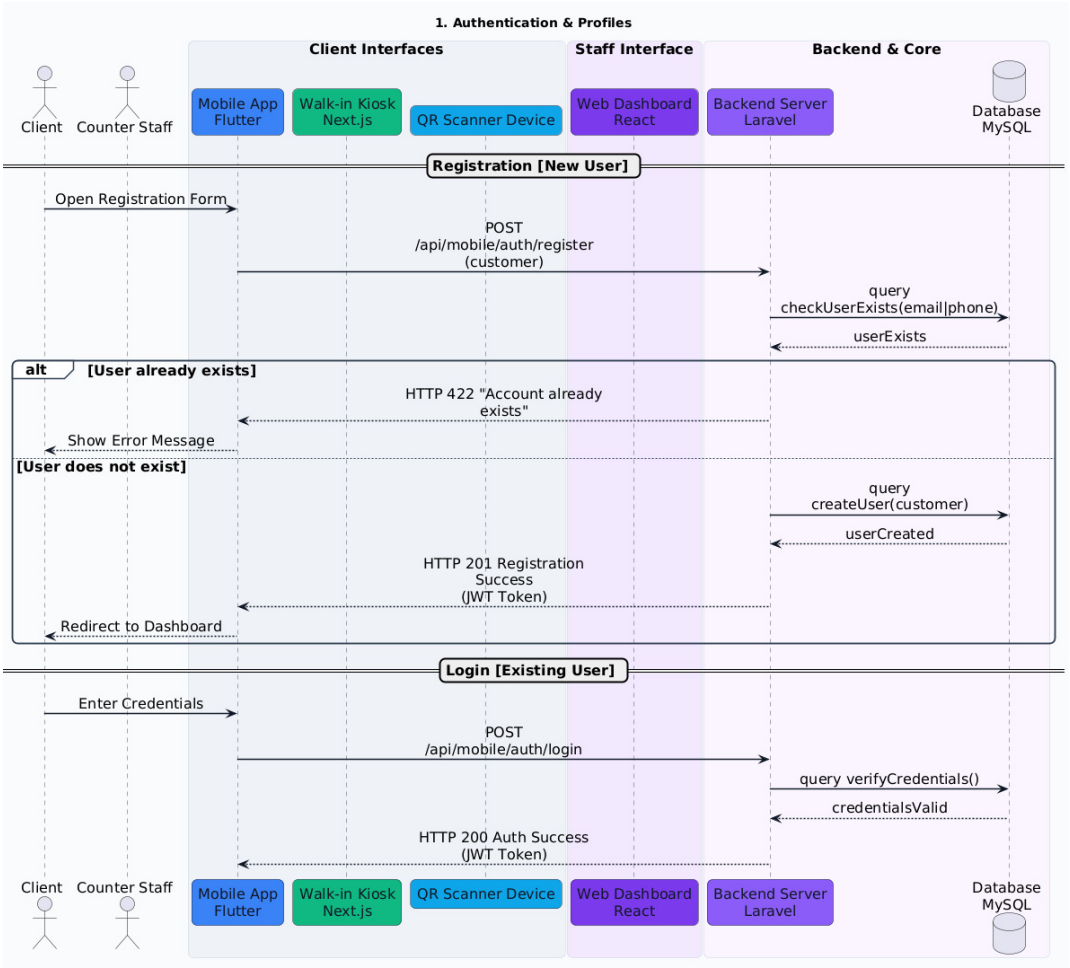


Figure 3.7: Authentication and booking sequence diagram.

3.11.3 QR Validation Sequence

The QR validation sequence is initiated by a scanner or kiosk. The device sends the scanned token to the backend, which authenticates the device, checks token status, validates the time window, verifies the appointment context, consumes the token, and creates or activates the queue entry. Invalid tokens return a rejection response and do not change the queue state.

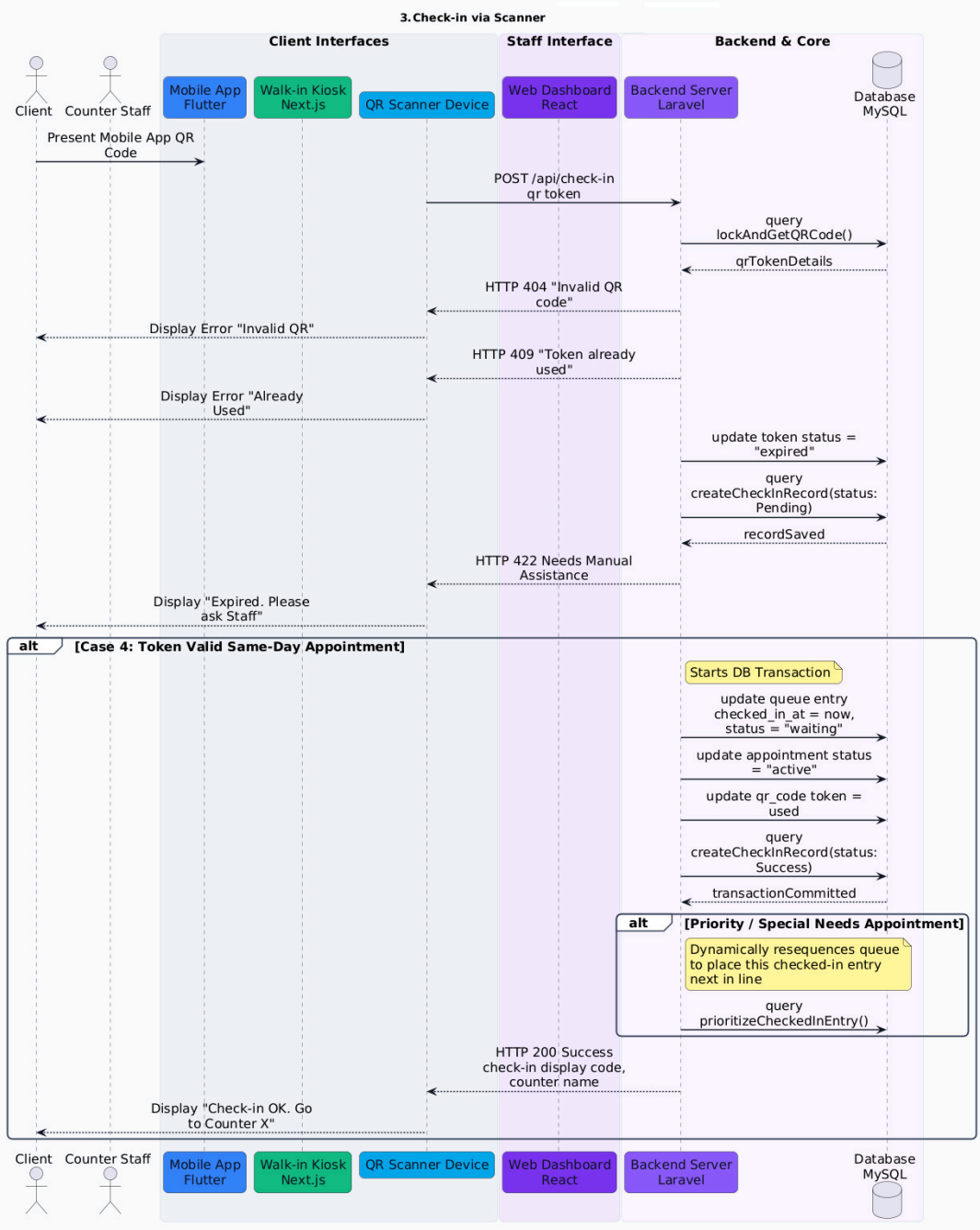


Figure 3.8: QR validation and self check-in sequence diagram.

3.11.4 Serving Process Sequence

The serving process sequence starts when a staff member calls the next eligible client. The backend updates the queue state, notifies subscribed clients, records the service start, and stores the final outcome when the service is completed.

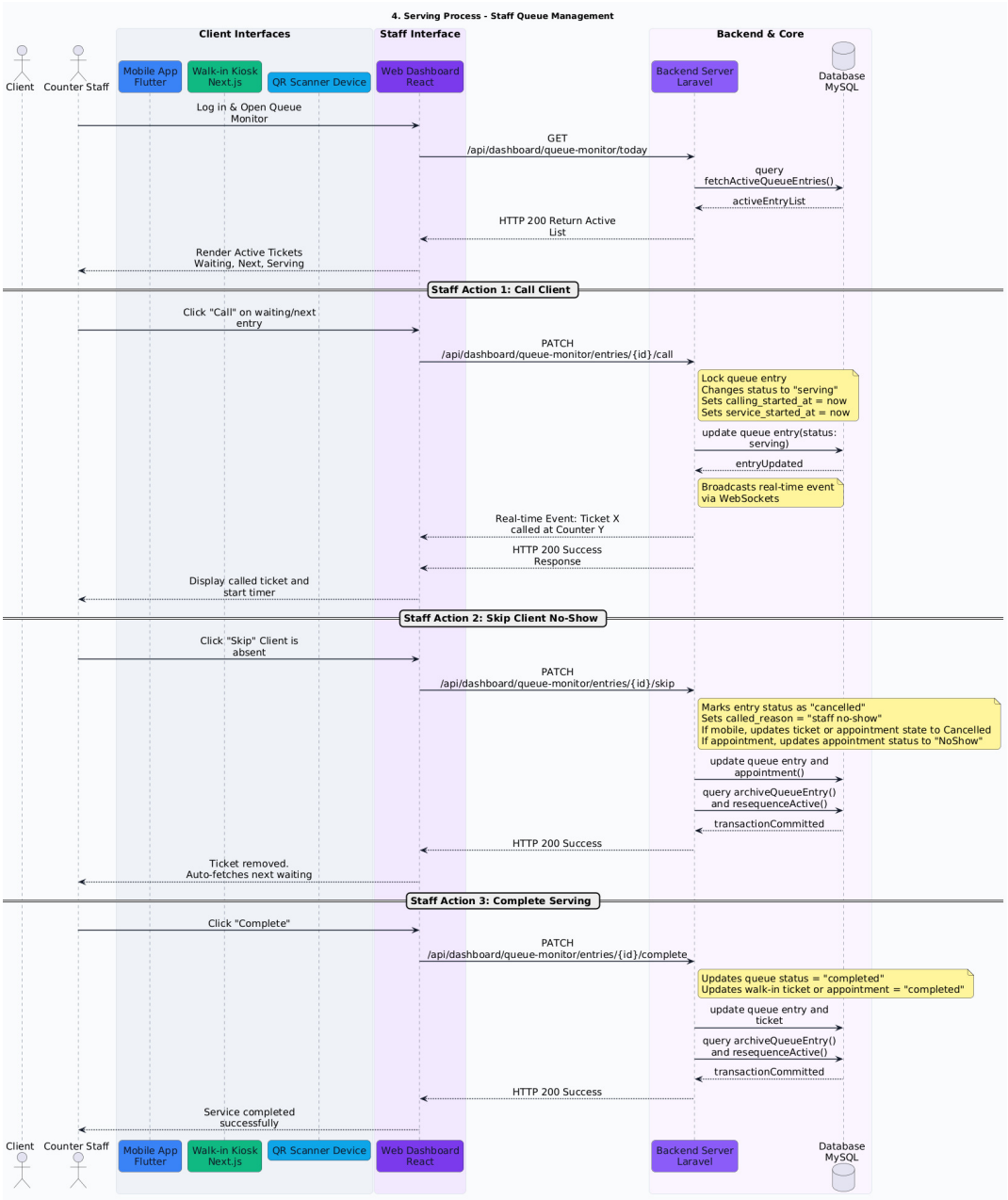


Figure 3.9: Serving process sequence diagram.

3.12 Data Model Summary

The data model describes the structural backbone of the platform. It formalizes persistent entities, key relationships, and the separation between authentication, booking, queue execution, organization management, and notification logic. A modular data model is essential for both maintainability and future scalability.

Table 3.5: Main entities of the proposed database design

Entity Group	**Role in the System**
User / UserRole	Store identity, credentials, and access rights.
Customer, Appointment, WalkInTicket	Represent the citizen-facing booking domain.
QrCodeToken, KioskDevice, CheckInRecord	Support secure QR validation and check-in traceability.
Company, Branch, Service, StaffMember	Represent the organizational and operational structure.
DailyQueueSession, QueueEntry	Support queue execution, status transitions, and service tracking.
Notification	Store or trigger outbound event communication.

3.13 Security Workflow

Security in SmartQdz is not limited to the presence of a QR code. It depends on the controlled combination of token generation, server-side validation, expiration logic, access control, and protected communication channels. The token must be unique, linked to a valid reservation, time-limited, and unusable after successful validation.

The security workflow begins when the backend generates a secure token after a booking or walk-in ticket is confirmed. That token is stored with metadata and delivered to the citizen through the application. Upon arrival, the token is scanned and sent to the backend. If the token is valid, unused, and not expired, the queue entry is updated and the event is logged. Otherwise, the request is rejected and the queue remains unchanged. This logic protects the platform against replay attempts, duplicate check-ins, and weak manual validation procedures.

3.14 Conclusion

This chapter translated the conceptual objectives of SmartQdz into a concrete system design. It described the general architecture, the main functional modules, the actors, the functional scope, the principal activity and sequence diagrams, the data model summary, and the security workflow.

Taken together, these design artifacts demonstrate that the proposed platform is both functionally coherent and technically structured. They provide the foundation for the next chapter, which presents the implementation environment and the practical realization of the system.

Chapter 4

Implementation and Evaluation

4.1 Introduction

This chapter presents the practical implementation of the proposed SmartQdz platform and explains how the design described in the previous chapter is translated into an operational prototype. The chapter focuses on the development environment, the software and hardware components used, the database implementation, the mobile and administrative interfaces, the integration of QR-enabled devices, and the evaluation strategy adopted to validate the feasibility of the system.

4.2 Development Architecture and Technology Stack

The architecture combines mobile, web, backend, database, and device technologies selected for scalability, maintainability, cross-platform support, and operational efficiency.

Table 4.1: Selected technologies supporting the system architecture

Architectural Layer	**Selected Technology**	**Design Role**
Citizen mobile interface	Flutter / Dart	Cross-platform mobile application for registration, appointment booking, QR ticket display, and queue tracking [27, 28].
Administrative interface	React / Next.js	Web-based administrative dashboard for branch management, monitoring, and operational control [29, 30].
Backend services	Laravel REST API	Centralized processing of business logic, authentication, queue orchestration, and request handling [31].
Authentication layer	JWT authentication	Secure transmission of authentication and authorization information between clients and backend services [36].
Data persistence	Relational SQL database	Structured storage of users, appointments, branches, services, counters, queue entries, and logs [32].
QR and device layer	QR generator and Raspberry Pi scanner	Generation and validation of QR-based tickets and branch-level check-in operations [9, 33, 34].
Deployment layer	Cloud hosting infrastructure	Centralized deployment and scalable access to backend and web services [35].

Flutter provides the citizen mobile interface, React/Next.js supports the administrative dashboard, Laravel centralizes backend rules, and the SQL database stores operational data. QR devices extend the platform into the branch environment.

4.3 Development Environment and Tools

The implementation environment combines development, testing, deployment, and collaboration tools used throughout the prototype lifecycle.

Table 4.2: Development environment and implementation tools

Tool / Technology	**Category**	**Purpose**
Flutter / Dart	Mobile development framework	Development of the cross-platform citizen mobile application for booking and queue management [27, 28].
React / Next.js	Web development framework	Development of the administrative dashboard and browser-based operational interfaces [29, 30].
Laravel REST API	Backend framework	Implementation of business logic, authentication, queue processing, and service orchestration [31].
MySQL	Relational database system	Persistent storage of users, appointments, branches, services, queues, and operational records [32].
JWT Authentication	Security mechanism	Secure authentication and authorization between clients and backend services [36].
Git / GitHub	Version control and collaboration	Source-code management, version tracking, and collaborative software development [37, 38].
Docker	Containerization platform	Packaging and deployment of backend services in isolated and reproducible environments [39].
Visual Studio Code (VS Code)	Development environment	Primary code editor used for frontend and backend implementation tasks [40].
Google Stitch	Design and prototyping tool	Interface design, visual prototyping, and UI/UX collaboration support [42].
Postman	API testing tool	Testing and validation of REST API endpoints and request workflows [41].
Raspberry Pi 4 and QR scanner	Connected devices	Branch-level QR scanning, customer check-in, and queue validation operations [34].
Cloud hosting services	Deployment infrastructure	Centralized hosting and remote access to backend and web services [35].

These tools cover source-code management, API testing, interface prototyping, deployment packaging, and branch-level QR scanning. They support implementation without changing the architecture defined in Chapter 3.

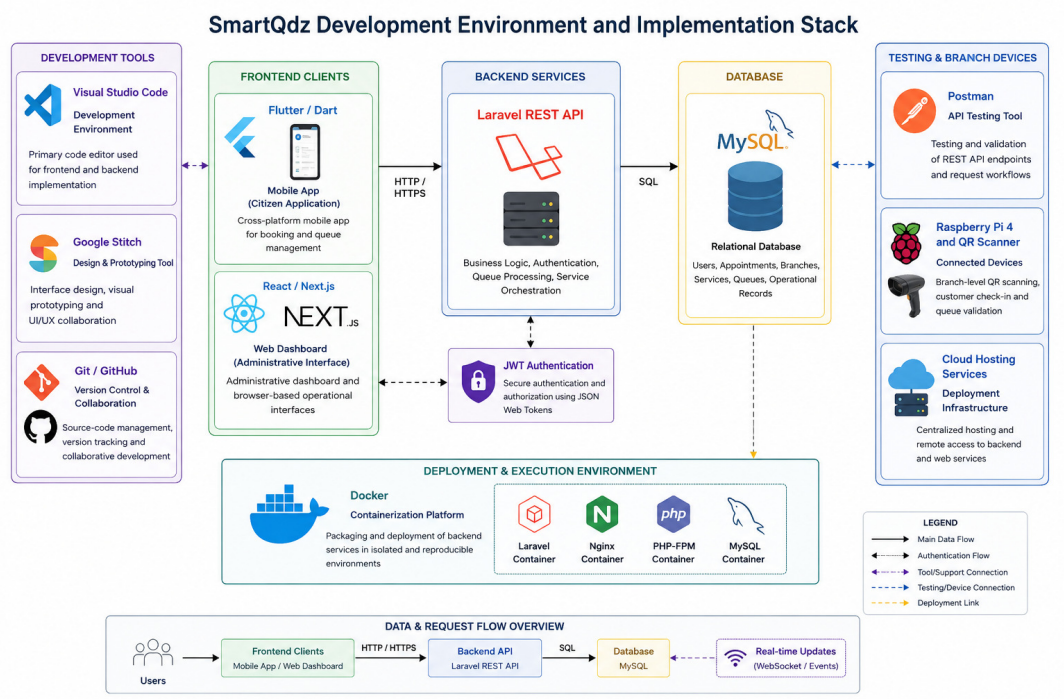


Figure 4.1: Development environment and implementation stack.

4.4 API Design and Backend Implementation

The backend exposes protected REST endpoints that define the integration contract between the mobile app, dashboard, kiosk, scanner, database, and notification services.

4.4.1 Request and Response Examples

Appointment creation requires an authenticated request. The token is sent in the HTTP authorization header:

```
Authorization: Bearer <access_token>
Content-Type: application/json
```

A simplified appointment-creation request can be represented as follows:

```
POST /api/mobile/appointments
{
```

```
  "branch_id": 5,
  "service_id": 12,
  "slot_date": "2026-05-21",
  "slot_start": "09:00",
  "slot_end": "09:30"
}
```

When the request is accepted, the backend returns the created appointment, display code, assigned counter, queue scope, and QR token reference:

```
{
  "status": "success",
  "appointment_id": 284,
  "display_code": "C02-31",
  "counter_id": 2,
  "queue_position": 31,
  "qr_token_id": 918,
  "check_in_window": {
    "starts_at": "2026-05-21T08:45:00",
    "ends_at": "2026-05-21T09:30:00"
  }
}
```

For QR check-in, the scanner sends the token payload and its device identifier:

```
POST /api/devices/qr/check-in
{
  "device_id": "BR05-SCAN-01",
  "token": "<qr_token_payload>"
}
```

If validation fails, the backend returns a structured error without changing the queue:

```
{
  "status": "error",
  "code": "QR_TOKEN_EXPIRED",
  "message": "The check-in token is no longer valid."
}
```

4.4.2 Backend Service Responsibilities

The Laravel backend is structured around services that isolate business decisions from controllers. Controllers receive requests and return responses, while service classes perform the operational logic. Important services include:

- `OperationsScheduleTimeSlotService` for slot generation and remaining-capacity calculation.
- `CounterQueueAssignmentService` for selecting the most appropriate counter.
- `QrTokenService` for token generation, expiration, validation, and consumption.
- `QueueLifecycleService` for state transitions from booking to completion.
- `NotificationDispatchService` for queue-related messages.
- `AuditLogService` for preserving administrative and security traces.

4.5 Database Implementation

The database uses a relational model to preserve consistency, support fast queue retrieval, and manage reservations, QR tokens, queue entries, check-ins, notifications, and audit logs. Each entity has a defined operational role: bookings describe service demand, QR tokens verify arrival, queue entries represent live state, and logs preserve traceability.

4.5.1 Database Integrity Rules

The implementation relies on database-level integrity as well as application-level validation. The most important rules are:

- appointments must reference valid users, branches, services, counters, and slots.
- a QR token must belong to one appointment and must be unique.
- a consumed QR token cannot be consumed again.
- queue sessions are unique by branch, counter, date, and slot.
- active queue positions must not be duplicated within the same queue session.
- cancelled, expired, completed, and rejected records are excluded from remaining-capacity calculations.

- audit and check-in records are append-only from the application perspective.

Indexes are added on queue-scope fields because live queue screens repeatedly filter by branch, counter, date, slot, and status. This improves dashboard response time and reduces database load during peak service periods.

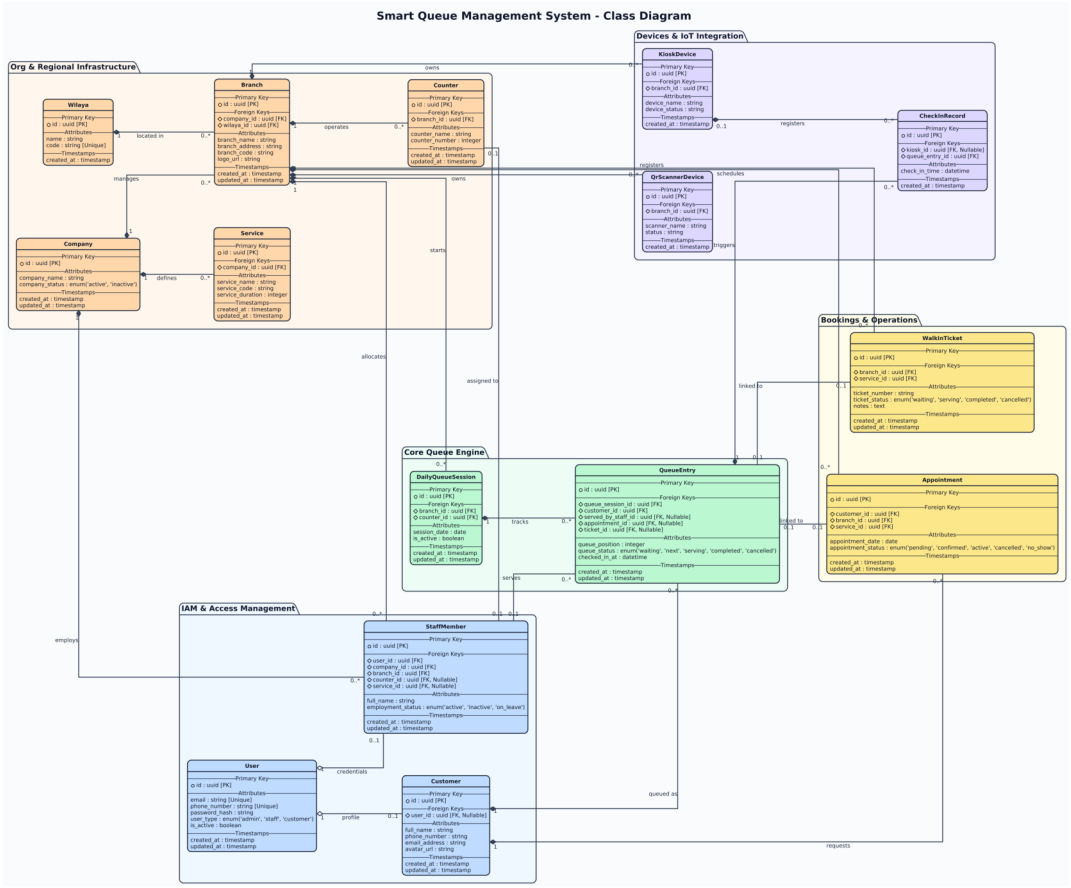


Figure 4.2: Database implementation of the SmartQdz platform.

4.6 Booking and Queue Logic

The booking and queue logic is backend-centered. Mobile, dashboard, kiosk, and scanner clients send requests to the Laravel API, which applies validation, authorization, capacity, assignment, and state-transition rules before updating the database.

4.6.1 Main Booking Actors

The operational actors are the mobile application for remote appointments, the dashboard for staff and administration, the kiosk for same-day walk-ins, the Raspberry Pi scanner for QR check-in, the Laravel API for business rules, and the database for persistent state.

4.6.2 Branch and Service Selection

The workflow begins with branch and service selection. Services do not own independent physical queues; they define which counters may process the request. After counter assignment, the booking belongs to that counter, date, and slot. Mobile branch visibility may also be filtered by wilaya.

4.6.3 Slot Availability

Booking slots are generated from branch schedules by `OperationsScheduleTimeSlotService`. The backend returns start/end time, period, capacity mode, total and remaining capacity, and whether an eligible active counter is still available.

A slot is considered available only when at least one eligible active counter has remaining capacity.

4.6.4 Remote Appointments and Walk-ins

The system supports two booking modes:

- **Remote Appointments:** created through the mobile application and validated later using QR check-in.
- **Walk-in Tickets:** created directly inside the branch through the kiosk interface or the administrative dashboard.

Remote appointments may be future or same-day reservations, while walk-ins are limited to same-day requests. After counter and slot assignment, both modes share the same counter-based queue model.

4.6.5 Capacity and Remaining Logic

Capacity is calculated at:

```
branch + counter + date + slot_start + slot_end
```

The service only limits eligible counters. It does not own the queue. Two capacity modes are supported:

- **separate_capacity:** appointments and walk-ins consume independent capacities.
- **shared_capacity:** both channels consume one shared capacity.

Cancelled, expired, completed, rejected, or inactive bookings are excluded from active capacity calculations.

4.6.6 Counter Assignment Logic

When a booking request is submitted, `CounterQueueAssignmentService` selects the counter using:

- branch consistency.
- service-to-counter association.
- counter active status.
- slot availability.
- current counter load.
- capacity rules.
- lowest-load priority.
- `display_order` as tie breaker.

If a counter becomes full, the backend attempts to assign another eligible counter. If no counter remains available, the booking request is rejected.

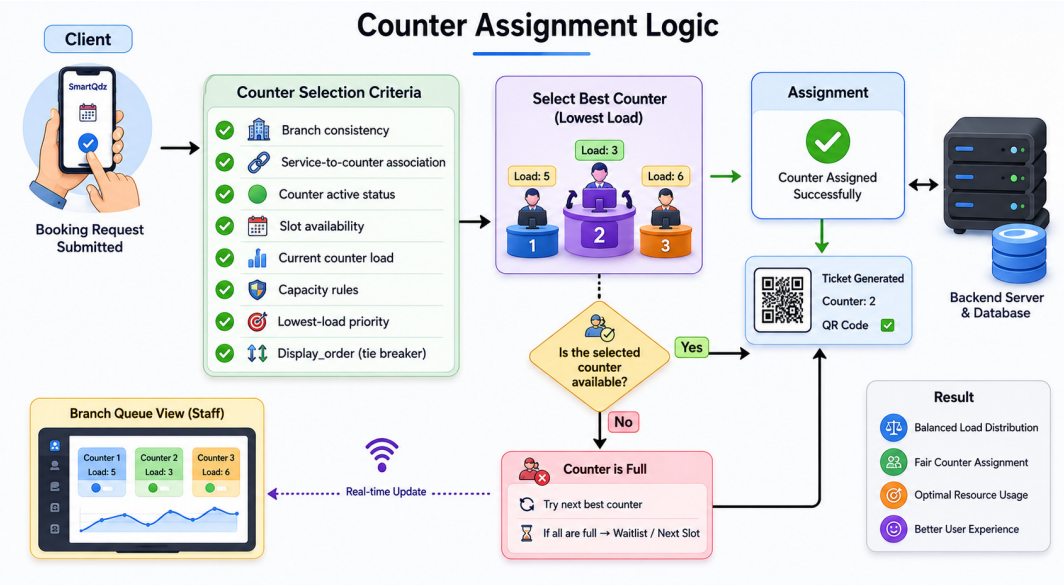


Figure 4.3: Counter assignment workflow.

4.6.7 Appointment Creation

Remote appointment creation is handled through the mobile booking API. The backend:

1. validates the authenticated user and selected branch, service, and slot.
2. checks wilaya restrictions when applicable.
3. prevents duplicate active bookings for the same service and date.
4. locks the slot namespace to avoid race conditions.
5. verifies slot availability.
6. selects the most appropriate counter.
7. locks the booking reference namespace.
8. calculates the booking reference number.
9. creates the appointment.
10. generates a one-time QR token for check-in.

4.6.8 Walk-in Creation

Walk-in tickets are created from the kiosk or dashboard for same-day requests. The backend validates the device or operator, checks availability, selects a counter, creates the queue session if needed, and inserts the entry directly into the operational queue. Walk-ins do not require QR check-in.

4.6.9 Queue Session and Queue Position Logic

Operational queues are managed using `daily_queue_sessions`. Each session is identified by:

```
branch_id + counter_id + session_date + slot_start + slot_end
```

Queue positions are calculated inside the selected session. Services handled by the same counter during the same slot share one sequence, and cancelled or completed entries are excluded from active displays.

4.6.10 Booking Reference Number and Display Code

The `booking_reference_number` is calculated per branch, counter, date, and slot range rather than per service. `BookingCodeFormatter` then generates visible codes such as:

```
CCC-31
```

For special-needs bookings, the code becomes:

```
SN-CCC-SPECIAL
```

4.6.11 QR Check-in Validation

QR check-in applies only to remote appointments. Each appointment receives a secure token during booking, and the scanner or kiosk submits it to the backend at arrival.

The backend verifies:

- token existence.
- token validity.
- token expiration status.
- appointment status.
- valid check-in time window.

After successful validation, the token is consumed and the appointment enters the operational queue.

4.6.12 Cancellation and Rebooking

When an appointment is cancelled, the backend removes it from active capacity calculations, cancels associated queue entries, and invalidates QR tokens. The freed capacity can then be reused if no active duplicate booking exists.

4.6.13 Duplicate and Capacity Protection

The backend prevents invalid or conflicting reservations through:

- validation rules.
- duplicate active-booking checks.
- slot locking.
- booking reference locking.
- database constraints.
- retry handling for reference conflicts.

4.6.14 Live Queue and Queue Monitor

The Queue Monitor and Live Queue interfaces retrieve data from appointments, walk-in tickets, sessions, and queue entries. Displayed data must preserve:

- branch isolation.
- counter isolation.
- slot isolation.
- active queue filtering.
- correct booking display codes.

Service filtering only changes visible rows and must not recalculate queue positions.

4.6.15 Special Needs Logic

Special-needs users receive priority handling. Their visible booking code follows:

```
SN-CCC-SPECIAL
```

The normal visible position is replaced with a priority indicator, while internal ordering remains preserved.

4.6.16 Branch, Counter, and Slot Isolation

Operational consistency requires:

- branch data remains isolated.
- counters maintain independent queues.
- different slots do not share queue positions.
- multiple services handled by the same counter share one operational queue.
- cancelled reservations do not consume active capacity.

The operational queue scope is:

```
branch + counter + date + slot_start + slot_end
```

4.7 Real-Time and Notification Implementation

4.7.1 WebSocket Implementation

The real-time layer is event-driven. After a queue transaction is committed, the backend dispatches an event to the relevant branch or counter channel, and subscribed clients refresh the affected queue view.

Table 4.3: Implemented WebSocket channel model

Channel Scope	**Example**	**Purpose**
Branch channel	`branch.5`	updates branch-wide dashboard widgets and display screens.
Counter channel	`branch.5.counter.2`	updates staff interface for one counter queue.
User channel	`user.84`	sends private appointment or queue-position updates to a citizen.
Admin channel	`admin.company.1`	sends administrative monitoring events to authorized supervisors.

Events are emitted only after successful database transactions. If the WebSocket connection is interrupted, the client refreshes through the API and resubscribes.

4.7.2 Push Notification Implementation

Push notifications cover events that should reach the user while the mobile app is in the background. The backend sends the payload through the configured provider, such as FCM [45].

Table 4.4: Notification events and payload purpose

Event	**Recipient**	**Purpose**
Booking confirmed	citizen	confirm appointment details and QR availability.
Reminder before slot	citizen	reduce no-show risk before the planned time.
QR check-in accepted	citizen, staff dashboard	confirm physical arrival and queue activation.
Turn approaching	citizen	prepare user to move to the service area.
Client called	citizen, display screen	announce counter assignment and service call.
Cancellation or rejection	citizen, admin log	inform user and preserve operational trace.

The notification table stores event type, target user, payload summary, delivery status, and timestamp, separating queue success from delivery failure.

4.8 Security Implementation

Security is implemented through layered controls: JWT authentication, role-based authorization, server-side validation, device authentication, token expiration, HTTPS/TLS, and audit logging.

Table 4.5: Security controls in the implementation

Control	**Implementation Role**
JWT access token	authenticates users and protects API requests.
Token expiration	limits the lifetime of authenticated sessions.
Authorization middleware	restricts actions according to user role and branch scope.
Password hashing	stores credentials using one-way secure hashing.
QR token hashing	avoids storing raw QR secrets as plain reusable values.
Device credential	authenticates scanners and kiosks before accepting check-in requests.
Transactional validation	prevents partial updates during booking, check-in, and service actions.
Audit logs	records sensitive operations for later inspection.
HTTPS/TLS	protects API communication during deployment.

4.8.1 JWT and Middleware Flow

After login, the backend returns a signed JWT access token. Middleware verifies signature, expiration, user status, role, and organizational scope before protected requests continue. Branch administrators are limited to assigned branches, while super administrators can access global configuration.

4.8.2 QR Validation Internals

QR validation follows a server-authoritative sequence:

1. the scanner authenticates itself with the backend.
2. the backend hashes or normalizes the submitted token payload.
3. the token record is searched in the database.
4. the backend verifies token status, expiration, appointment status, branch, counter, date, and check-in window.
5. the token row is locked during consumption to prevent replay under concurrent scans.
6. the appointment or queue entry is moved to the next valid state.

7. the check-in attempt is stored in `check_in_records`.

8. a WebSocket event and optional notification are dispatched.

Rejected validation attempts do not change the queue state. They are stored with a failure code such as `TOKEN_NOT_FOUND`, `TOKEN_EXPIRED`, `TOKEN_ALREADY_USED`, `WRONG_BRANCH`, or `CHECKIN_WINDOW_CLOSED`.

4.9 Deployment Architecture

The prototype uses a centralized backend deployment. Mobile, dashboard, kiosk, and scanner clients communicate with the Laravel API over secured channels; the backend connects to the SQL database, WebSocket service, and notification provider.

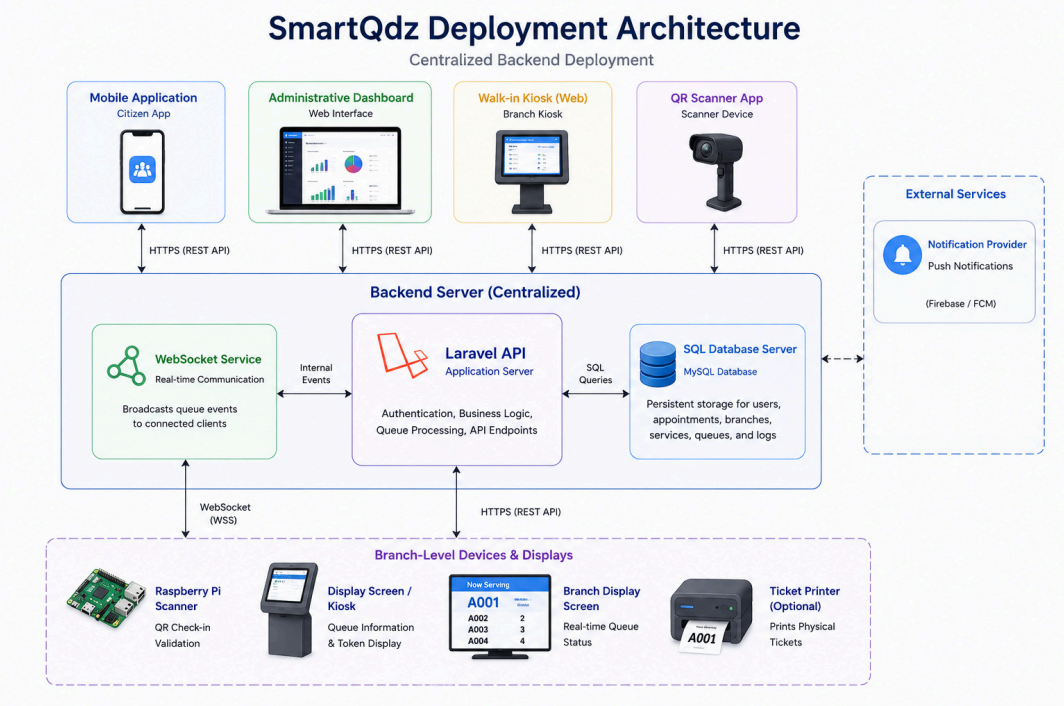


Figure 4.4: Deployment architecture of the SmartQdz prototype.

Table 4.6: Deployment components

Component	**Deployment Role**
Mobile application	installed on citizen smartphones and communicates with backend APIs.
Administrative dashboard	hosted web interface for staff and administrators.
Laravel API server	central authority for authentication, business logic, and queue state.
SQL database server	persistent storage for operational and administrative data.
WebSocket service	broadcasts real-time queue events to authorized clients.
Notification provider	delivers push notifications to mobile devices.
Raspberry Pi scanner	branch-level device for QR check-in validation.
Display screen or kiosk	local interface for branch queue visibility and walk-in entry.

4.10 Mobile Application Interfaces

The mobile application is the citizen-facing interface. Users authenticate, choose a branch and service, reserve an appointment or request a ticket, display the QR token, and track queue status. The workflow remains sequential so that booking, ticket display, and self check-in are easy to follow.

4.10.1 Mapping Integration Using OpenStreetMap

To improve branch accessibility and user orientation, SmartQdz integrates a lightweight mapping module based on OpenStreetMap [43]. This solution avoids dependency on proprietary services such as Google Maps while providing an open and cost-effective cartographic integration.

In the administrative dashboard, the map is used during branch creation and configuration. The administrator can search for an address using OpenStreetMap services, select a valid location within the chosen wilaya, and automatically store the corresponding geographic coordinates, `latitude` and `longitude`. The position can also be adjusted manually on

the map.

In the mobile application, the map allows users to visualize available branches and locate the agency associated with their appointment. The appointment-details page can additionally request GPS permission to display the user position, the branch location, and an approximate distance estimation between both points.

The module also handles localization errors such as denied permissions, disabled GPS services, or unreliable positions by displaying informative messages instead of inaccurate geographic data.

This mapping functionality remains independent from the core queue-management logic. It does not affect booking workflows, queue prioritization, QR validation, or ticket generation. The backend only provides branch coordinates, while the mobile application and dashboard handle cartographic visualization.

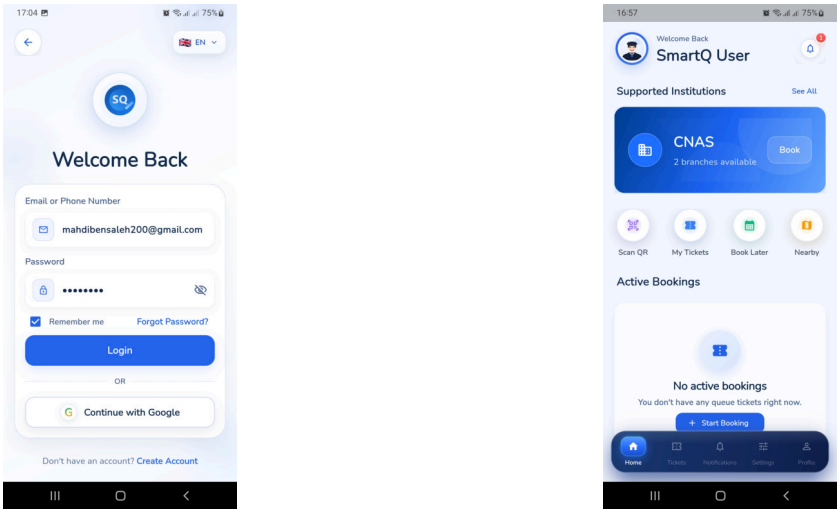


Figure 4.5: Mobile login and home screens of the citizen application.

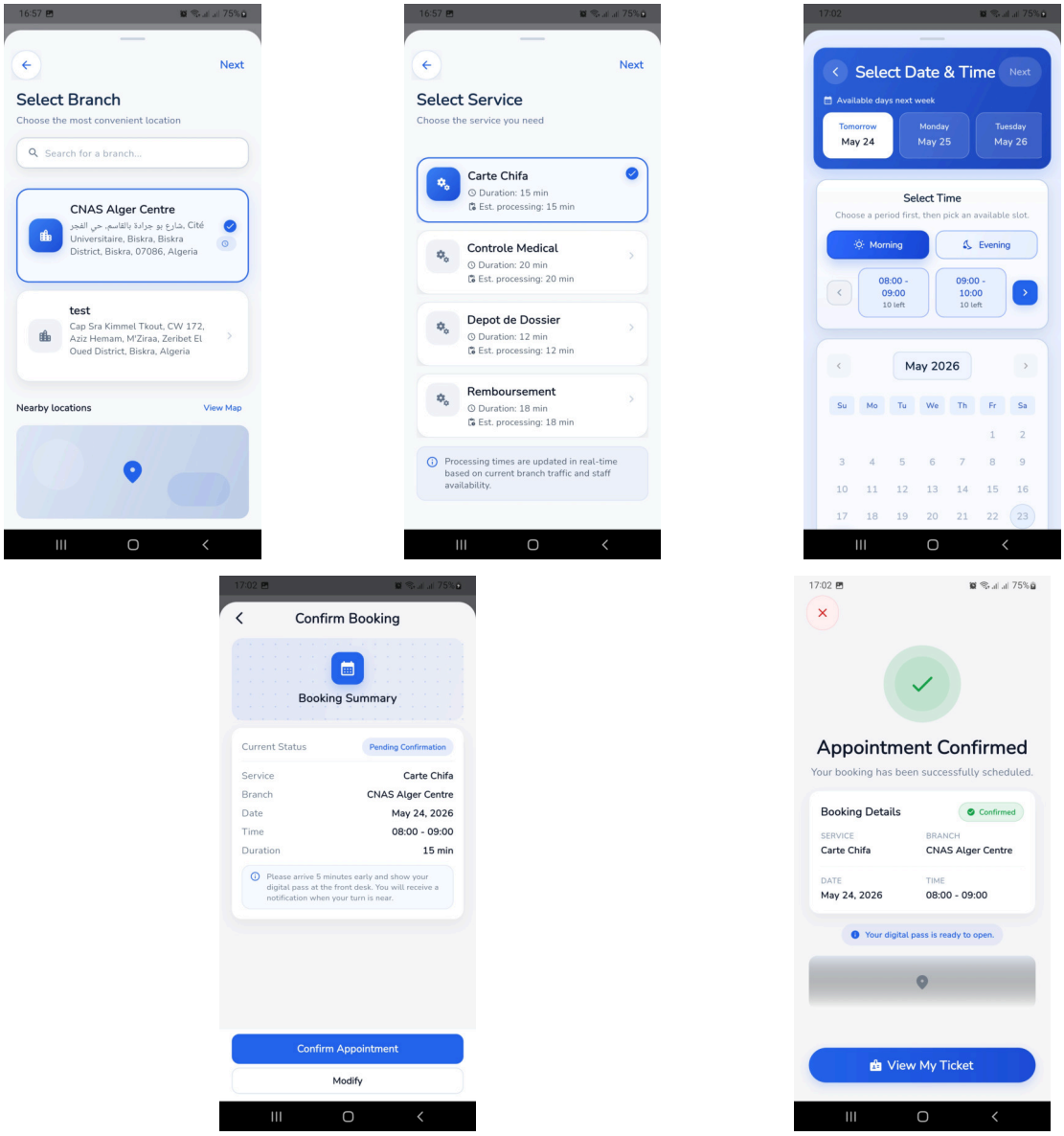


Figure 4.6: Booking screens of the mobile application: branch and service selection, appointment scheduling, and walk-in options.

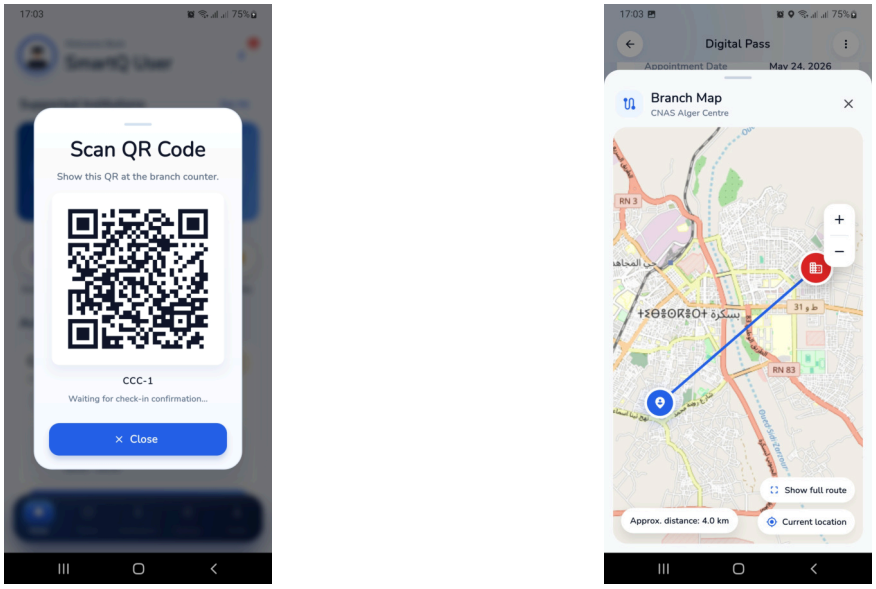


Figure 4.7: QR ticket display and queue-tracking screens of the mobile application.

4.11 Administrative Dashboard Interfaces

The administrative dashboard is the operational control center. It allows staff and administrators to monitor queues, call clients, update service sessions, configure services, manage branches and users, and consult reports. The overview summarizes queue activity, while specialized screens handle live operations, configuration, user management, and analytics.

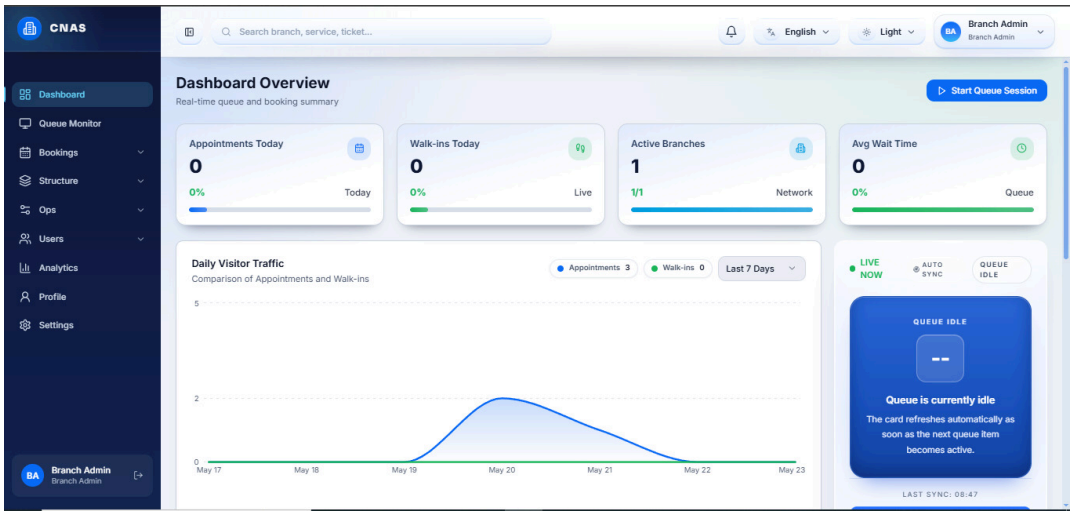


Figure 4.8: Administrative dashboard overview.

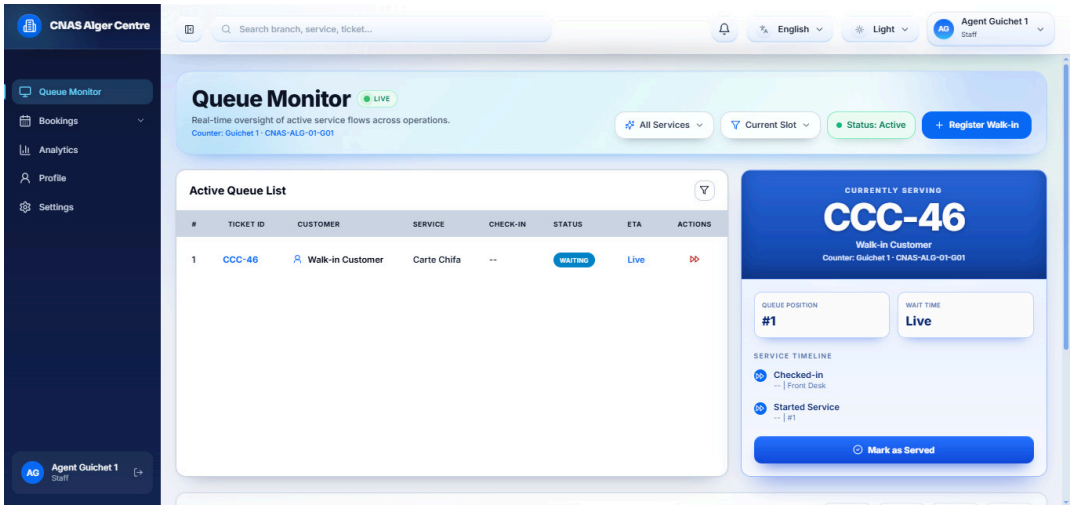


Figure 4.9: Queue-management screen.

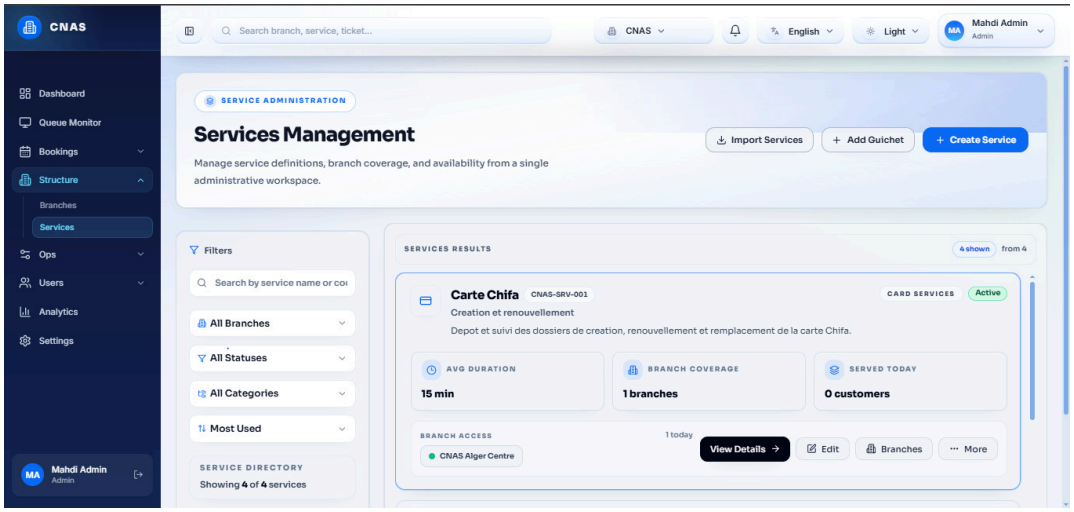


Figure 4.10: Service-configuration screen.

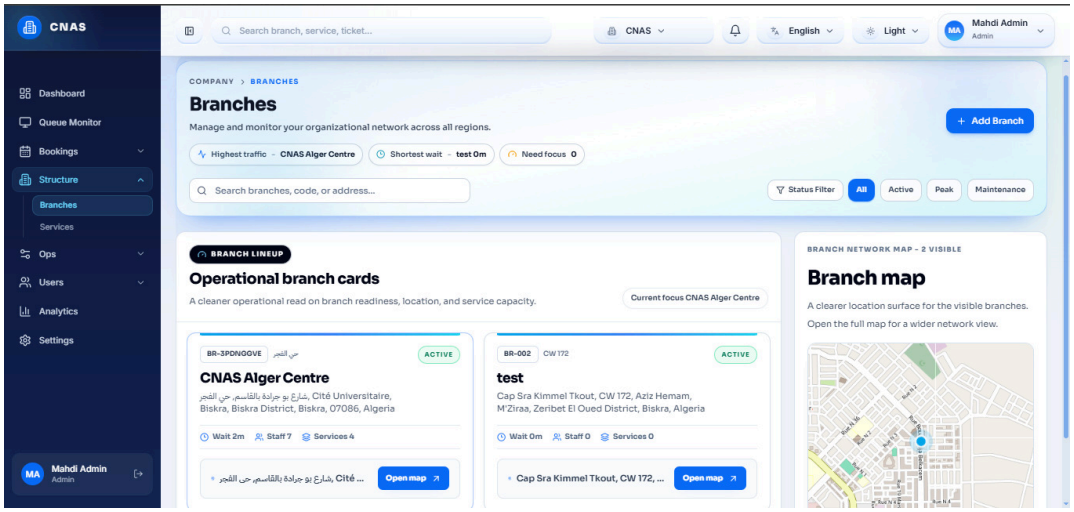
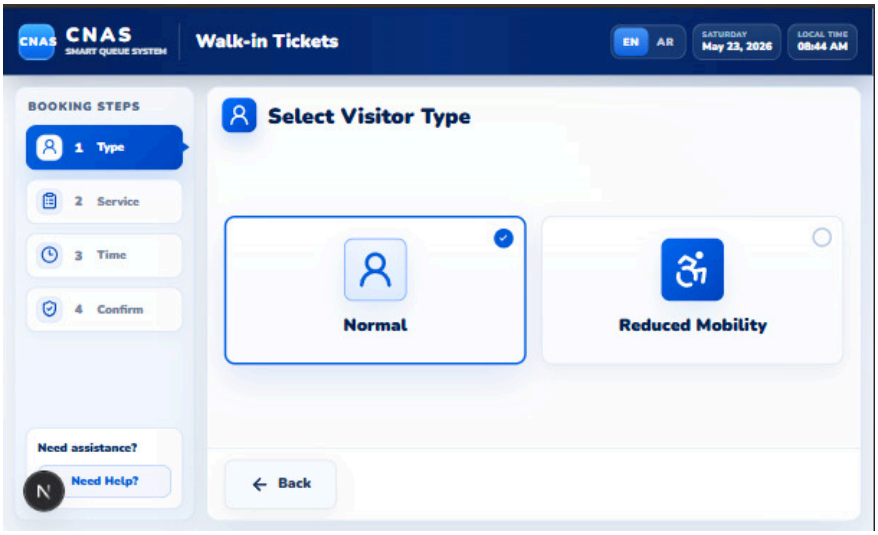


Figure 4.11: User and branch management screen.

4.12 QR and IoT Device Integration

SmartQdz implements a real-time integration mechanism between QR verification services and connected branch devices. When a citizen arrives at a branch, the QR code is scanned using a kiosk or scanner device, which securely transmits the token to the backend server for validation. After successful verification, the queue state is updated automatically and synchronized with connected display devices through real-time communication technologies. This implementation ensures efficient coordination between the backend system, scanning devices, and queue-display interfaces while reducing manual intervention and improving service responsiveness.





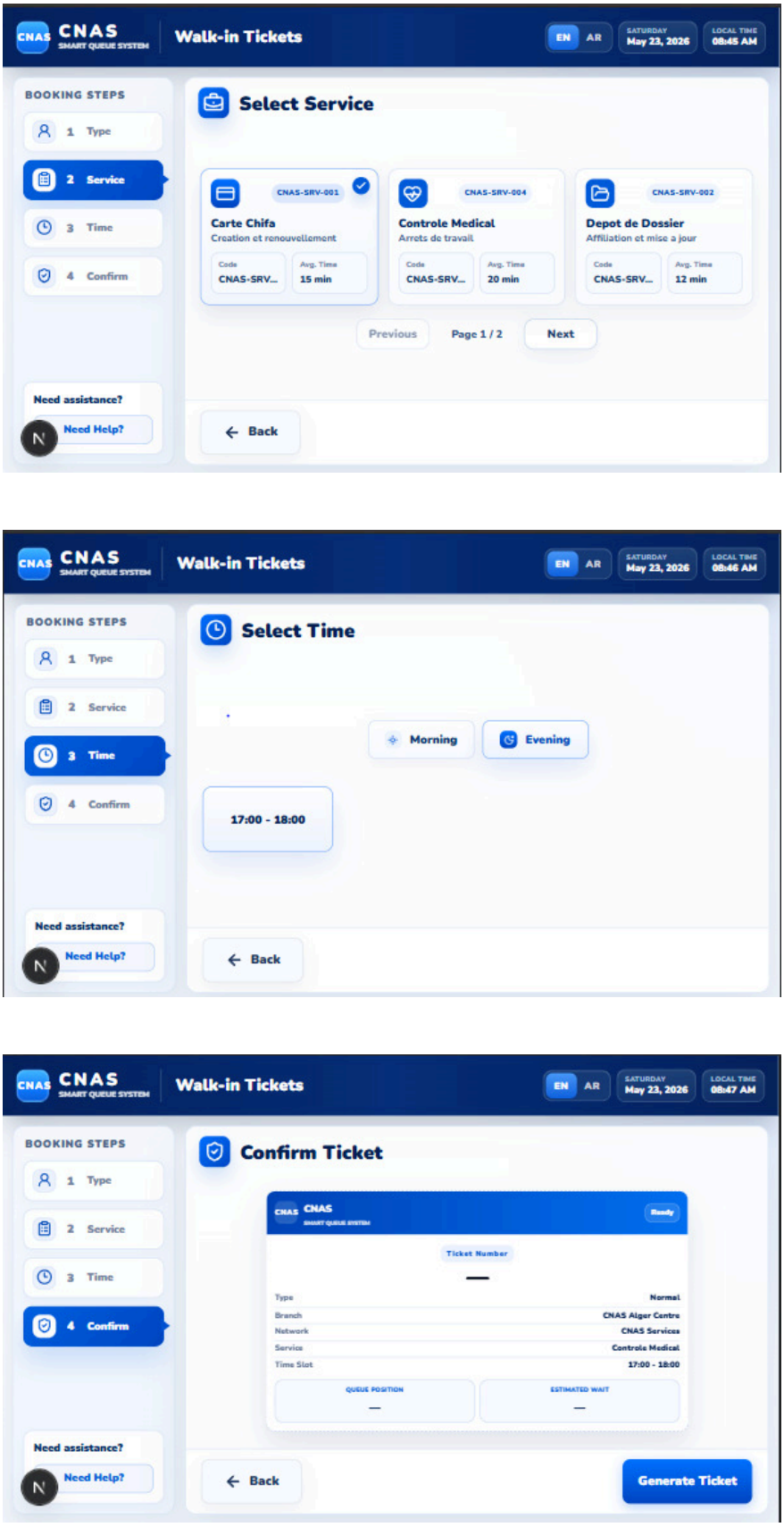


Figure 4.12: Public Kiosk Interfaces of the SmartQdz Platform



Figure 4.13: SmartQdz Kiosk Hardware Prototype Based on Raspberry Pi

4.13 Testing and Results

The evaluation is a prototype-level engineering validation. It verifies workflows, security checks, synchronization, and database constraints under controlled scenarios, without claiming full institutional benchmarking.

4.13.1 Evaluation Method

The test plan covers five categories:

1. **functional validation:** verifies that each main user workflow produces the expected state transition.
2. **API validation:** checks request format, authentication headers, response structure, and error codes.
3. **security validation:** verifies token expiration, role restrictions, device authentication, and QR replay protection.
4. **real-time validation:** measures whether queue updates are broadcast to subscribed clients after backend commits.
5. **performance validation:** observes response time, check-in latency, and behavior under concurrent booking attempts.

4.13.2 Functional Test Scenarios

Table 4.7: Functional test scenarios

Scenario	**Input / Action**	**Expected Result**	**Status Evidence**
User registration and login	valid user credentials	JWT token returned and profile loaded	API response and mobile screen.
Appointment booking	branch, service, date, slot	appointment created, counter assigned, QR token generated	database row and QR screen.
Walk-in ticket creation	kiosk request for same-day service	queue entry created directly in active session	kiosk confirmation and dashboard row.
QR check-in	valid token scanned inside window	token consumed and entry moved to waiting state	check-in log and WebSocket event.
Staff call next	staff clicks call-next	highest-priority eligible entry becomes called	dashboard update and display event.
Service completion	staff closes active service	queue entry becomes completed and report counters update	database status and report widget.
Cancellation	user or admin cancels active booking	appointment removed from active capacity and QR invalidated	API response and audit log.

4.13.3 Security and Error Tests

Table 4.8: Security validation tests

Test	**Expected Backend Behavior**	**Validation Evidence**
Expired JWT request	request rejected before controller execution	HTTP error response and auth log.
Staff action outside branch scope	action rejected by authorization middleware	forbidden response and audit entry.
Expired QR token	check-in rejected, token remains unusable	failure code in check-in records.
Reused QR token	second scan rejected after first successful consumption	token consumed timestamp unchanged.
Unknown device	request rejected before token validation	device authentication failure log.
Duplicate booking	new active booking rejected for same service/date rule	validation error and unchanged capacity.

4.13.4 Performance and Real-Time Indicators

The following indicators should be recorded during the final prototype test run from API logs, backend timestamps, WebSocket timestamps, and database records.

Table 4.9: Quantitative indicators for prototype evaluation

Indicator	**Measurement Method**	**Prototype Target**	**Interpretation**
Login API response time	timestamp before request and after response	under 1 second in local prototype	verifies authentication responsiveness.
Slot availability response time	API timing for available-slot endpoint	under 1.5 seconds in local prototype	verifies schedule and capacity calculation cost.
QR validation latency	scanner request time to backend response time	under 1 second in local prototype	verifies check-in usability at branch entrance.
WebSocket propagation delay	database commit time to client event receipt	under 2 seconds in local prototype	verifies live synchronization.
Concurrent booking conflict rate	simultaneous requests for last available slot	zero duplicated active capacity	verifies locking and constraints.
Notification dispatch delay	backend event time to provider acceptance time	under 5 seconds in normal conditions	verifies communication readiness.
Error handling coverage	number of tested invalid cases over planned cases	all critical cases tested	verifies robustness against invalid operations.

These indicators demonstrate feasibility at prototype level. Production deployment would still require load testing with realistic branches, counters, users, and simultaneous events.

4.13.5 Load and Concurrency Testing

The key concurrency scenario is multiple requests for the last available slot. Only one request should consume capacity; the others must fail or retry safely. The same principle applies to booking-reference generation, QR token consumption, and call-next actions.

Table 4.10: Concurrency tests

Scenario	**Risk**	**Expected Result**
Two users book last slot at same time	overbooking	one booking succeeds and the other is rejected or retried safely.
Two scans submit same QR token	replay or duplicate check-in	first scan consumes token and second scan is rejected.
Two staff users call next simultaneously	duplicate call assignment	only one queue entry is moved to called state.
WebSocket reconnect during queue update	missed event	client refreshes latest queue state after reconnection.

4.14 Conclusion

This chapter presented the practical realization of SmartQdz and described the tools, interfaces, database structures, connected devices, and evaluation approach used during implementation. The implementation phase confirms that the proposed architecture can be translated into an operational platform supporting both citizens and administrative staff.

Although the current evaluation remains limited to prototype-level validation, the results demonstrate that the proposed solution is coherent, feasible, and suitable for further institutional refinement. The final chapter now concludes the dissertation by summarizing the main contributions, limitations, and future perspectives of the work.

General Conclusion

This dissertation presented SmartQdz as a smart queue-management platform designed to improve administrative service delivery through digital booking, dynamic QR-based self check-in, real-time queue monitoring, and centralized operational control. The work began by analyzing the limits of conventional queue systems and by situating the project within the broader movement toward digital public services. It then examined representative market solutions, proposed a context-appropriate architecture, and described the implementation logic of the platform.

From a technical perspective, the proposed solution combines a citizen-facing mobile application, a staff operations interface, an administrative dashboard, backend APIs, a relational database, and connected devices such as QR scanners and smart displays. This combination allows the platform to link digital reservation logic with the physical service environment while preserving traceability and administrative visibility.

Main Results

The work produced a coherent prototype architecture covering the complete service lifecycle: user authentication, appointment creation, walk-in ticketing, QR token generation, self check-in, queue-state synchronization, staff service execution, and administrative supervision. The design also formalized the queue state machine, role-permission matrix, API contract, database schema, WebSocket event model, QR validation internals, and error-handling strategy.

At prototype level, the expected evaluation indicators are functional workflow success, rejection of invalid or reused QR tokens, prevention of duplicated active bookings, successful real-time propagation of queue updates, and consistent queue-state transitions. These indicators show that the proposed approach is technically feasible. However, final quantitative values such as API latency, WebSocket propagation delay, QR validation time, and concurrent-booking behavior must be recorded from execution logs during the final test run.

Limitations

Despite the relevance of the proposed model, several limitations remain. First, the platform depends on stable network connectivity for real-time synchronization and secure token validation. Second, the deployment of scanners, kiosks, and display devices may require additional logistical effort and institutional resources. Third, the current validation remains at prototype scale and does not yet constitute a full multi-branch production deployment. Fourth, the performance evaluation must be extended with larger user volumes, realistic service schedules, and long-running monitoring before institutional adoption.

Future Work

Several extensions may strengthen the platform in future iterations. Advanced analytics and predictive models could improve waiting-time estimation and demand balancing. Smart scheduling could recommend time slots according to historical demand, staff availability, and service duration. OCR-assisted document preparation could help users verify required files before arrival. Integration with national identity or institutional identity services could reinforce trust in authentication and reduce duplicate accounts. Finally, broader cloud-based deployment, cross-branch monitoring, and stronger analytics dashboards could prepare the platform for larger-scale administrative adoption.

Overall, SmartQdz demonstrates that a balanced combination of digital booking, secure QR verification, connected devices, and centralized service governance can provide a credible foundation for the modernization of administrative queue management.

References

[1] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA, USA: Pearson, 2015.

[2] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2014.

[3] A. Charland and B. Leroux, "Mobile application development: web vs. native," *Communications of the ACM*, vol. 54, no. 5, pp. 49–53, 2011.

[4] M. A. El-Kassem, *Cross-Platform Mobile Application Development*. London, UK: Springer, 2022.

[5] Google, "Flutter for Desktop," Official Flutter Documentation. Available: https://flutter.dev/desktop. Accessed: May 9, 2026.

[6] K. Bjørn-Hansen, C. Rieger, T. A. Majchrzak, and G. Gröner, "Comparing performance of cross-platform desktop applications," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2020, pp. 28–37.

[7] K. Ashton, "That 'Internet of Things' Thing," *RFID Journal*, 2009.

[8] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[9] ISO/IEC 18004, *QR Code Bar Code Symbology Specification*. Geneva, Switzerland: International Organization for Standardization, 2015.

[10] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Harlow, UK: Pearson, 2017.

[11] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.

[12] Y. Zhang and X. Chen, "Secure authentication in IoT systems," *IEEE Access*, vol. 8, pp. 72004–72020, 2020.

[13] A. Boulmakoul, L. Karim, and A. Lbath, "Smart queue management system using IoT," *International Journal of Computer Applications*, vol. 162, no. 10, 2017.

[14] United Nations Department of Economic and Social Affairs, *UN E-Government Survey 2024*. Available: https://desapublications.un.org/publications/un-e-government-survey-2024. Publication date: September 2024. Accessed: May 9, 2026.

[15] National Institute of Standards and Technology, *SP 800-63-4, Digital Identity Guidelines*. Available: https://csrc.nist.gov/pubs/sp/800/63/4/final. Published: July 31, 2025. Accessed: May 9, 2026.

[16] Qmatic, "Queue Management Solutions," Available: https://www.qmatic.com/solutions/queue-management/. Accessed: May 9, 2026.

[17] Qmatic, "Queue Management Software and Tools," Available: https://www.qmatic.com/products. Accessed: May 9, 2026.

[18] Qmatic, "ISO 27001:2022 Certificate," Available: https://www.qmatic.com/hubfs/pages/gl/about/Files/Qmatic%20ISO%2027001-2022.pdf. Accessed: May 9, 2026.

[19] Skiplino, "Queue Management System," Available: https://skiplino.com/. Accessed: May 9, 2026.

[20] Skiplino, "Retail," Available: https://skiplino.com/. Accessed: May 9, 2026.

[21] QLess, "Queue Management Systems by QLess," Available: https://qless.com/. Accessed: May 9, 2026.

[22] QLess, "Products," Available: https://www.qless.com/products/. Accessed: May 9, 2026.

[23] Wavetec, "Mobile-Q – Virtual Mobile Queuing App," Available: https://www.wavetec.com/solutions/queue-management/mobile-queuing/. Accessed: May 9, 2026.

[24] Wavetec, "Mobile App – How it Works," Available: https://www.wavetec.com/solutions/queue-management-system/mobile-queuing/how-it-works/. Accessed: May 9, 2026.

[25] Lineberty, "Virtual queue management solution," Available: https://en.lineberty.com/. Accessed: May 9, 2026.

[26] Lineberty, "Lineberty Booking API," Available: https://developer.lineberty.net/api1/index.html. Accessed: May 9, 2026.

[27] Google, "Dart programming language," Official Documentation. Available: https://dart.dev/. Accessed: May 17, 2026.

[28] Google, "Building user interfaces with Flutter," Official Flutter Documentation. Available: https://docs.flutter.dev/ui. Accessed: May 17, 2026.

[29] Meta Platforms, "React," Official Documentation. Available: https://react.dev/. Accessed: May 17, 2026.

[30] Vercel, "Next.js Docs," Official Documentation. Available: https://nextjs.org/docs. Accessed: May 17, 2026.

[31] Laravel, "Laravel Documentation," Official Documentation. Available: https://laravel.com/docs. Accessed: May 17, 2026.

[32] Oracle, "MySQL Reference Manual," Available: https://dev.mysql.com/doc/refman/en/. Accessed: May 17, 2026.

[33] Project Nayuki, "QR Code generator library," GitHub repository. Available: https://github.com/nayuki/QR-Code-generator. Accessed: May 17, 2026.

[34] Raspberry Pi Ltd, "Raspberry Pi 4 Tech Specs," Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. Accessed: May 17, 2026.

[35] Google Cloud, "Hosting Applications on Google Cloud," Available: https://cloud.google.com/hosting-options. Accessed: May 17, 2026.

[36] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force, May 2015. Available: https://datatracker.ietf.org/doc/html/rfc7519. Accessed: May 17, 2026.

[37] Git SCM, "Git Documentation," Available: https://git-scm.com/doc. Accessed: May 21, 2026.

[38] GitHub, "GitHub Docs," Available: https://docs.github.com/. Accessed: May 21, 2026.

[39] Docker Inc., "Docker Documentation," Available: https://docs.docker.com/. Accessed: May 21, 2026.

[40] Microsoft, "Visual Studio Code Documentation," Available: https://code.visualstudio.com/docs. Accessed: May 21, 2026.

[41] Postman, "Postman Learning Center," Available: https://learning.postman.com/docs/. Accessed: May 21, 2026.

[42] Google, "Stitch," Available: https://stitch.withgoogle.com/. Accessed: May 21, 2026.

[43] OpenStreetMap Foundation, "OpenStreetMap," Available: https://www.openstreetmap.org/. Accessed: May 21, 2026.

[44] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, Internet Engineering Task Force, December 2011. Available: https://datatracker.ietf.org/doc/html/rfc6455. Accessed: May 21, 2026.

[45] Google, "Firebase Cloud Messaging Documentation," Available: https://firebase.google.com/docs/cloud-messaging. Accessed: May 21, 2026.

[46] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queueing Theory*, 4th ed. Hoboken, NJ, USA: Wiley, 2008.

Appendix A

Additional Technical Documentation

A.1 Introduction

This appendix serves as a comprehensive technical companion to the architectural design and system implementation chapters of the SmartQdz platform. While the core chapters of this dissertation focus on high-level design choices, comparative studies, and structural workflows, this section provides supplementary engineering documentation, complete Unified Modeling Language (UML) specifications, low-level implementation details of critical software modules, real-time message structures, and physical deployment configurations.

The material presented herein is intended to provide software architects, systems engineers, and deployment operators with the technical blueprints required to reproduce, scale, and maintain the SmartQdz platform in a production public service environment. By documenting the modular internals, real-time state machines, security validation pipelines, and hardware integration boundaries, this appendix ensures the engineering transparency and reproducibility of the entire enterprise system.

A.2 Complete UML Documentation

The system architecture and operational workflows of SmartQdz are modeled using the Unified Modeling Language (UML) to represent structural and behavioral characteristics. This section details the complete use case model, detailing all associated actors and system use cases, followed by behavioral activity models that govern the booking lifecycle, the IoT-enabled check-in process, and the core queue service state transitions.

A.2.1 Complete Use Case Diagram

The use case model provides a functional boundary of the SmartQdz platform, mapping out the relationships between external actors and internal system functionalities. The model covers administrative, staff-level, customer-facing, and device-level interactions, establishing an integrated, multi-tenant public service ecosystem.

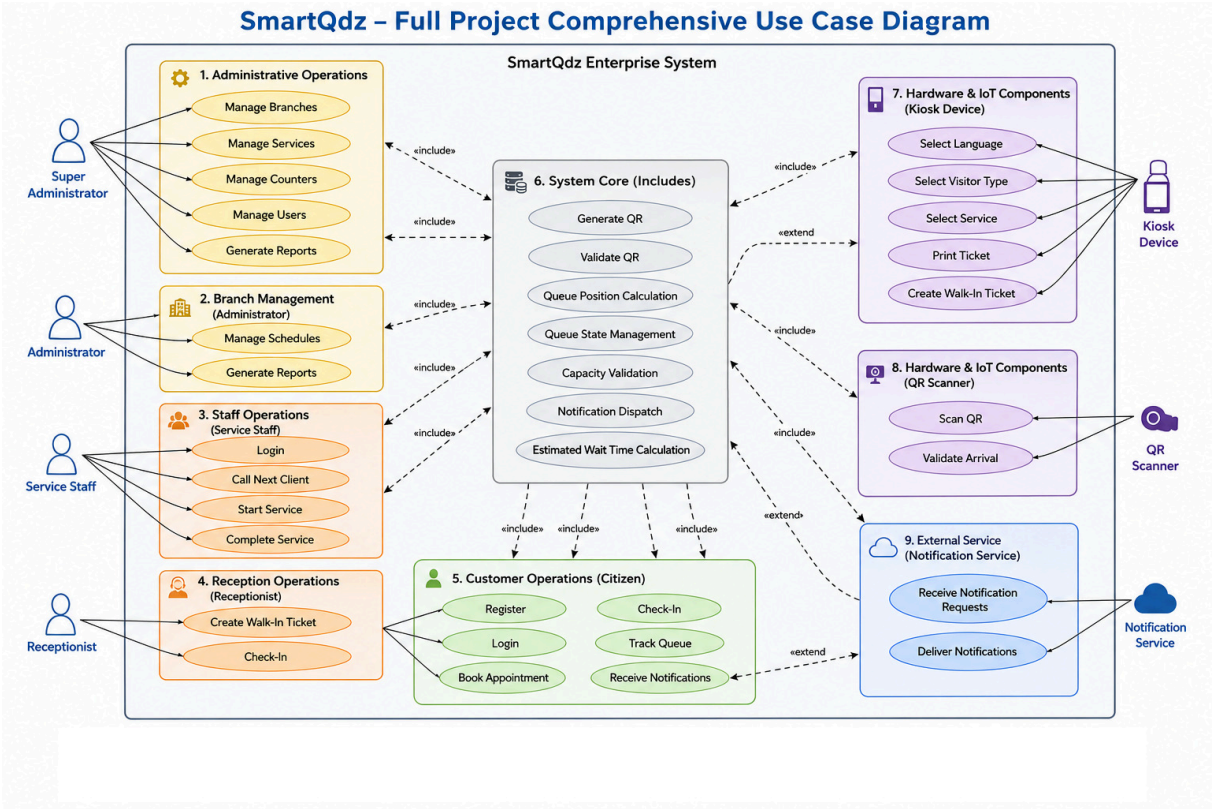


Figure A.1: Exhaustive use case boundary diagram showcasing multi-actor interactions, administrative overrides, and automated device validation scopes.

Description of System Actors

The SmartQdz platform recognizes eight distinct actors that interact with its logical boundaries, each defined by unique roles, security permissions, and operational objectives:

- **Citizen:** The primary consumer of public services. The citizen interacts with the system via the cross-platform mobile application to search for branches, consult real-time queue states, register secure profiles, schedule remote appointments, generate check-in tokens, and receive real-time push notifications regarding their queue progress.
- **Receptionist:** A branch-level administrative staff member responsible for welcoming visitors who arrive without previous mobile bookings. The receptionist operates the

administrative dashboard to register walk-in tickets, assign priority flags for special-needs citizens, and manually resolve arrival discrepancies.

- **Service Staff:** The counter agents assigned to process transactions within designated service windows. The service staff interacts with the system through a lightweight web interface to call the next citizen in sequence, initiate service transactions, declare delays, redirect citizens to other counters, and mark service sessions as completed.
- **Administrator:** A branch-level manager who oversees the daily operations of a specific agency or office. The administrator has permissions to adjust daily schedules, activate or deactivate specific service counters, allocate staff resources, and view localized operational reports and historical logs.
- **Super Administrator:** The highest administrative authority within the system. The super administrator operates at the tenant or national level, possessing complete read and write access across all branches. Their responsibilities include branch registration, service catalog management, counter schema definition, user account provisioning, role assignment, and global analytics generation.
- **Kiosk Device:** An autonomous, branch-level physical terminal deployed in the lobby. The kiosk runs a specialized web interface that allows walk-in citizens to select their desired services, select their language, declare priority status, and print a physical paper ticket containing their display code.
- **QR Scanner:** An IoT hardware node (typically a Raspberry Pi equipped with a serial barcode camera or physical image sensor) positioned at the branch entrance. The device operates as a secure client that scans mobile check-in tokens, decodes the validation payload, and securely transmits REST validation requests to the backend.
- **Notification Service:** An external cloud-based actor (such as Google Firebase Cloud Messaging) that receives API push payloads from the backend and routes them asynchronously to citizens' mobile applications.

Description of Major Use Cases

The functional capabilities of the platform are organized into seventeen critical use cases:

- **Register:** Allows citizens to create a verified account by providing their name, phone number, email address, password, and optional demographic metadata, stored securely in the database.
- **Login:** Authenticates citizens, staff, and administrators using secure credentials, returning a signed JSON Web Token (JWT) that defines their authorized scope.

- **Book Appointment:** Enables mobile citizens to select a branch, service category, date, and specific time slot, verifying slot capacity before writing a secure reservation.
- **Create Walk-In Ticket:** Allows lobby visitors (via the kiosk) or receptionists to instantly generate a same-day queue entry, bypass remote validation rules, and register directly into the counter sequence.
- **Generate QR:** Automatically runs on the backend after a successful mobile booking, generating a unique, cryptographically signed, one-time-use QR token containing appointment metadata and timestamps.
- **Check-In:** Performs physical arrival verification. When scanned at the branch entrance, the system decodes the QR token, validates its integrity, checks the arrival window, and marks the appointment as active.
- **Track Queue:** Provides citizens with real-time updates of their position, estimated wait times, and active serving states through active mobile and dashboard connections.
- **Receive Notifications:** Receives asynchronous alerts (push, SMS, or in-app) triggered by backend events, such as appointment confirmations, check-in success, and service calls.
- **Call Next Client:** Allows counter staff to pull the highest-priority waiting customer from the active counter queue session using priority queue sorting rules.
- **Start Service:** Records the exact timestamp when a staff agent begins processing a customer's transaction, shifting the queue entry's state to "Serving" and starting the transaction timer.
- **Complete Service:** Marks a transaction as finalized, logging service codes, processing durations, and staff notes, then closing the queue entry and freeing the counter for the next customer.
- **Manage Branches:** Allows super administrators to create, update, or suspend physical branch records, including geographic coordinates, addresses, and multi-tenant keys.
- **Manage Services:** Enables global administrators to define the organizational catalog of services, setting standard durations, maximum capacities, priority weights, and department linkages.
- **Manage Counters:** Governs physical counter instances, mapping counters to specific physical branches and assigning which service categories each counter is authorized to process.

- **Manage Schedules:** Controls branch-level time structures, allowing administrators to define operating hours, slot intervals, weekly exceptions, and public holidays.
- **Manage Users:** Oversees system accounts, controlling account provisioning, role assignments (RBAC), counter links, and password resets.
- **Generate Reports:** Aggregates operational performance indicators (such as wait times, processing times, staff utilization, and hourly volume) into branch and system-wide charts and tables.

A.2.2 UML Activity Diagram – End-to-End Operational Lifecycle

The behavioral model of the SmartQdz platform is represented by a single, comprehensive UML Activity Diagram that maps the entire transactional and physical lifecycle of a citizen's service request. This unified model integrates the remote mobile booking phase, the security validation gates at the branch lobby, and the real-time queue service transactions managed by counter agents, including all normal paths and alternative error flows.

The comprehensive lifecycle progresses through the following sequential logical blocks:

1. **JWT Authentication Validation:** The citizen opens the mobile application to start a booking. The mobile client transmits the user's JWT in the Authorization header. The backend validates the signature and expiration. If the token is invalid, the request is rejected with a `401 Unauthorized` response, terminating the session.
2. **Selection & Capacity Validation:** The citizen selects their target branch, service category, date, and time slot. The backend checks slot capacity by calculating the remaining reservation capacity. If the capacity is full, the request is redirected back to the selection screen.
3. **Duplicate Booking Validation:** The system queries active appointments to verify the user does not have a duplicate booking for the same service and date. If a duplicate exists, the request is blocked, terminating the booking attempt.
4. **Booking Creation & Counter Assignment:** If validation passes, the appointment is written to the database with a `pending` status, and the scheduling service assigns the target counter using load balancing rules.
5. **QR Token Generation & Booking Confirmation:** The backend generates a cryptographically signed, one-time-use QR token containing the appointment metadata. The appointment status is confirmed, and the QR token is rendered on the user's mobile screen.

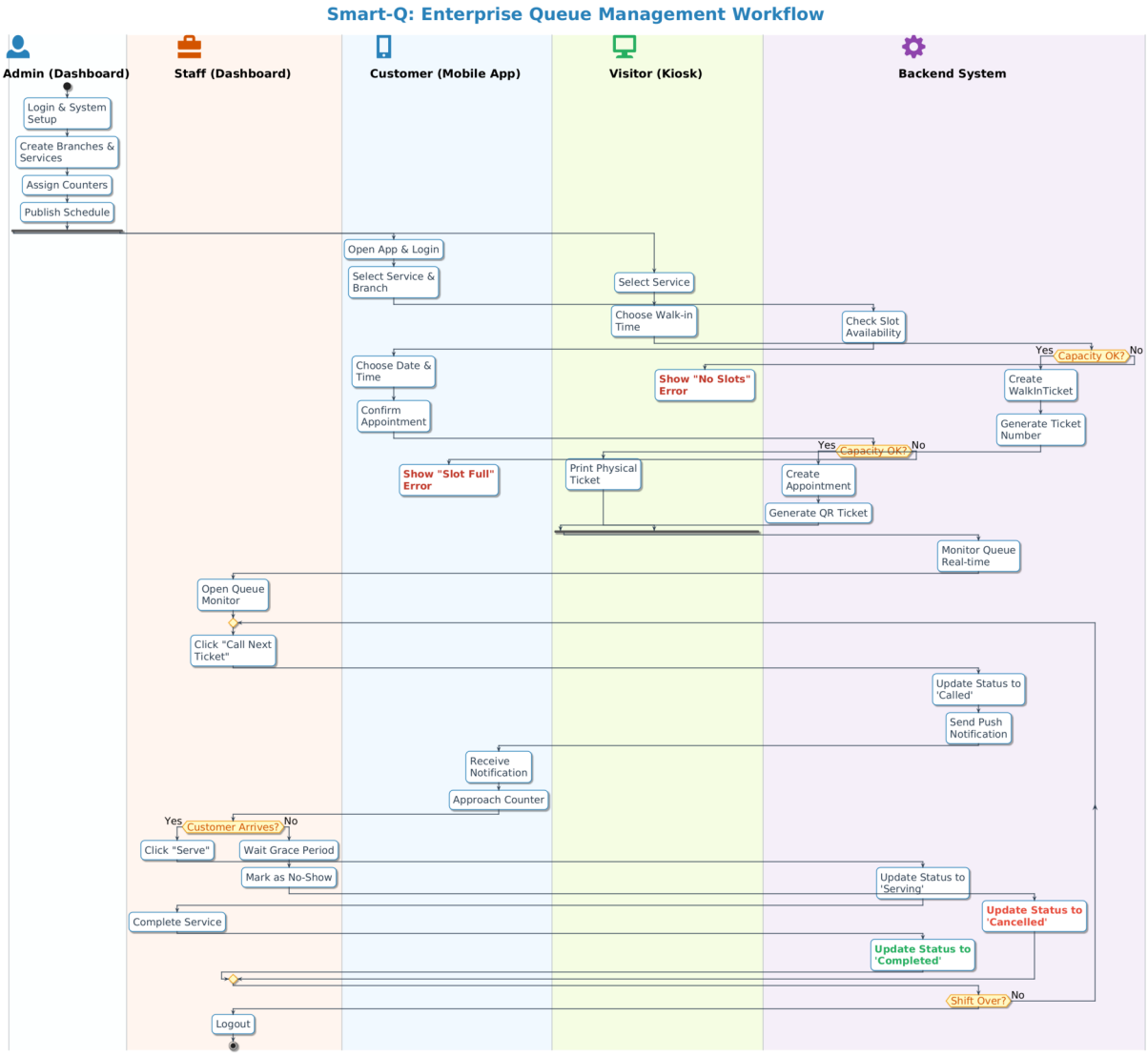


Figure A.2: End-to-End SmartQdz Appointment, Check-In, and Queue Service Lifecycle Activity Diagram

6. **Pre-Arrival Alternative Paths:** Before arrival, the citizen can cancel the booking remotely (status shifts to `cancelled`). If the citizen fails to appear before the slot window closes, the booking is flagged as `expired`. Both actions free slot capacity.

7. **Lobby Arrival & QR Scan:** Upon arrival at the branch lobby within the arrival window, the citizen scans their QR code at the IoT check-in scanner powered by a Raspberry Pi.

8. **Token Decoding & Cryptographic Signature Verification:** The scanner decodes the token and submits the check-in request to the backend. The backend validates the cryptographic signature. If signature verification fails, the scanner displays an `Invalid Token` alert and rejects check-in.

9. **Anti-Replay & Branch Validation:** The backend locks the token row to prevent concurrent scans and checks if the token has been consumed or has expired. It also validates that the scanner belongs to the branch where the slot was booked. If validation fails, check-in is rejected.

10. **Check-In Approval & Queue Registration:** Upon check-in approval, the token is marked as `consumed`. The citizen's appointment status changes to `waiting`, and the system registers the user in the daily queue session, calculating the display code.

11. **Waiting & Called States:** The citizen enters the waiting queue. When a counter agent clicks "Call Next", the queue entry transitions to the `called` state. The counter display screen announces the ticket code in real-time via WebSockets.

12. **No-Show (Expired) Check:** If the called citizen fails to appear at the counter within the grace window, the staff agent flags them as absent, moving the queue entry to the `expired` state.

13. **Start Service & Serving State:** If the client appears, the agent clicks "Start Service", moving the entry to the `serving` state and starting the transaction timer.

14. **Complete Service & Completed State:** When the transaction is completed, the agent enters service notes and submits completion, shifting the entry to the `completed` state and closing the transaction record.

A.3 Detailed Implementation

This section delivers exhaustive engineering documentation regarding the internal modules, algorithms, security schemas, real-time message structures, and device integrations supporting the SmartQdz platform.

A.3.1 Authentication Module

The authentication framework uses stateless JSON Web Tokens (JWT) to secure communication between clients and backend services. This module is divided into four main layers:

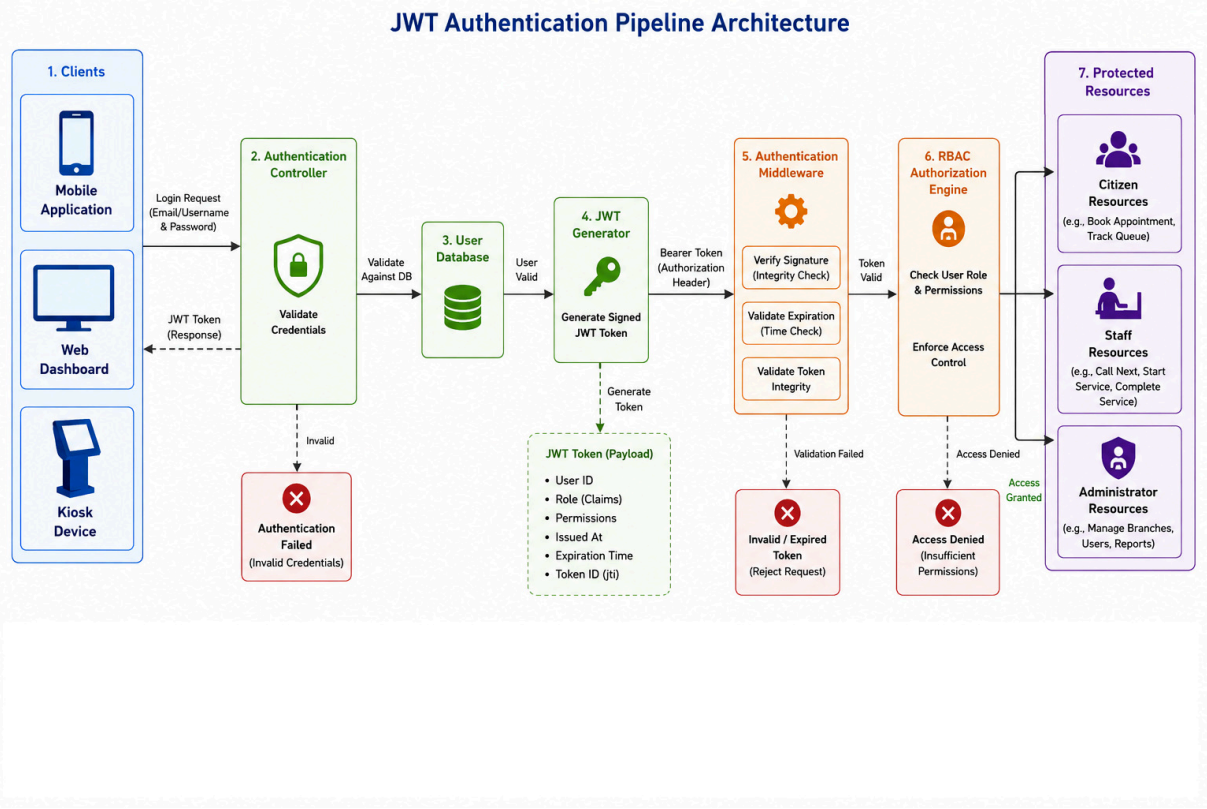


Figure A.3: Architectural block diagram of the authentication pipeline, illustrating token generation, middleware verification, and RBAC enforcement.

JWT Generation

Authentication requests (such as citizen login or administrative dashboard access) are processed by Laravel's authentication services. Upon successful credential verification, the system generates a JWT payload using HS256 (HMAC with SHA-256) signature algorithms. The JWT structure consists of:

- **Header:** Specifies the signature algorithm and token type.
- **Payload:** Contains the user identifier (`sub`), issuer (`iss`), generation time (`iat`), expiration time (`exp`), and user-specific claims such as assigned role and branch scope constraints.
- **Signature:** Created by hashing the encoded header, payload, and a secure server-side environmental secret key.

Login Flow

The login workflow ensures credentials are exchanged securely. The client transmits the credentials over HTTPS using a POST request. The controller forwards the request to the authentication service, which hashes the incoming password using bcrypt and compares it to the database record. If authentic, the JWT token is issued along with a secure refresh token to sustain sessions without requiring repeated credential entries.

Middleware Verification

Protected API routes are guarded by a specialized middleware class: `JwtRouteProtectionMiddleware`. When a request is received, the middleware extracts the token from the HTTP Authorization header: `Bearer <token>`. The token is decrypted, and its signature is verified using the server secret key. If the token has been altered, contains an invalid issuer, or has passed its expiration timestamp (`exp`), the middleware rejects the request immediately, returning an `HTTP 401 Unauthorized` JSON payload.

Role-Based Access Control (RBAC)

Once verified, the request is subjected to authorization checks managed by role middleware. The platform enforces four distinct permission tiers:

- **Citizen Role:** Limited to self-service endpoints, including profile management, mobile booking, personal ticket tracking, and notifications.
- **Staff Role:** Restrained to counter-level queue operations. Counter agents can only call clients, start, and complete services for the counters and branches to which their accounts are explicitly assigned.
- **Administrator Role:** Granted full access to branch-level configurations, reports, active schedules, and staff accounts within their designated agency.
- **Super Administrator Role:** Unrestricted global access, enabling the manipulation of all system properties, tenant variables, global branch registrations, and system configurations.

A.3.2 Queue Management Module

The queue management module contains the core scheduling, routing, and synchronization logic of the platform, designed to operate with transactional safety under concurrent load.

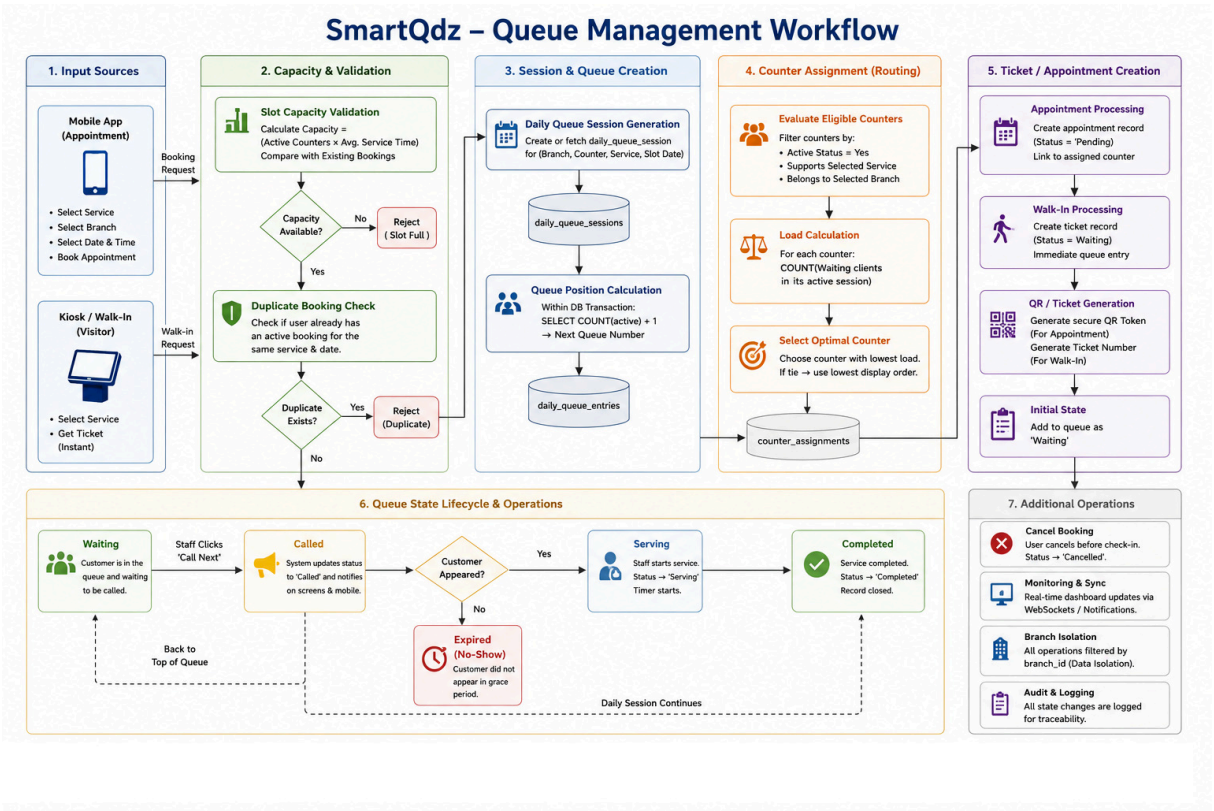


Figure A.4: System workflow illustrating daily session generation, capacity validation, load calculation, and counter routing operations.

Queue Session Creation

Daily queues are partitioned into `daily_queue_sessions`. These records are automatically initialized by the backend scheduler when the first check-in or walk-in transaction occurs for a specific branch, counter, and service slot on a given date. The session isolates operational queues, ensuring that queue sequence arrays do not leak across different dates, physical counters, or slot periods.

Queue Position Calculation

The system calculates queue positions inside the active `daily_queue_session` using transactional increments. The database runs a `SELECT COUNT` of active entries within the target session plus one inside a database transaction to calculate the next sequence index, preventing duplicate number allocations when multiple citizens check-in or request tickets simultaneously.

Queue State Management

Queue state transitions are strictly governed by state machine classes. Transitions to "Called", "Serving", and "Completed" require validation of the preceding state. If a staff agent attempts to call a client whose state is "Completed" or "Expired", the request is rejected. Every transition logs changes to the database to preserve operational history.

Counter Assignment Logic

The routing engine selects the optimal counter using `CounterQueueAssignmentService`. When a citizen books an appointment or requests a kiosk ticket, the service evaluates all counters assigned to that branch, filtering by:

1. Counter operational state (must be set to active).
2. Service linkage (the counter must be configured to process the specific service category).
3. Active workload (the counter with the lowest number of waiting clients in its daily session receives priority).
4. Tie-breaker configuration (if multiple counters have identical queue sizes, the system selects the one with the lowest display order).

Branch Isolation Logic

To support multi-branch networks, the database schema enforces branch isolation. Every table containing operational queue states (appointments, daily sessions, scanner logs, kiosk transactions) contains a foreign key linking to a master `branches` table. All backend queries are partitioned by `branch_id`, preventing data leaks and ensuring that branch dashboards only view their respective queue states.

Slot Capacity Validation

When a booking request is received, the capacity checker queries `OperationsSchedule TimeSlotService`. The system calculates the standard slot capacity by multiplying the average service duration by the number of active counters assigned to that service. If the active bookings for that slot equal or exceed this capacity, the booking transaction is aborted.

Duplicate Booking Prevention

To prevent citizens from occupying multiple slots simultaneously, the system enforces a strict duplicate check:

```
SELECT COUNT(*) FROM appointments
WHERE user_id = :user_id
  AND service_id = :service_id
  AND slot_date = :slot_date
  AND status IN ('pending', 'checked_in');
```

If this count is greater than zero, the booking controller rejects the request, returning a descriptive validation message.

Appointment and Walk-In Processing

The module maintains two input streams: * **Appointments:** Created remotely via the mobile API, resulting in a pending appointment record and a secure QR token. These entries are excluded from active sorted queues until physical check-in occurs. * **Walk-ins:** Generated at the physical kiosk or dashboard. They are directly registered into the active sorted queue with a "Waiting" status, bypass the QR verification phase, and receive an instant display code.

A.3.3 QR Module

The QR module governs the generation, storage, security, and verification of arrival tokens, securing the transition from digital reservation to physical queue activation.

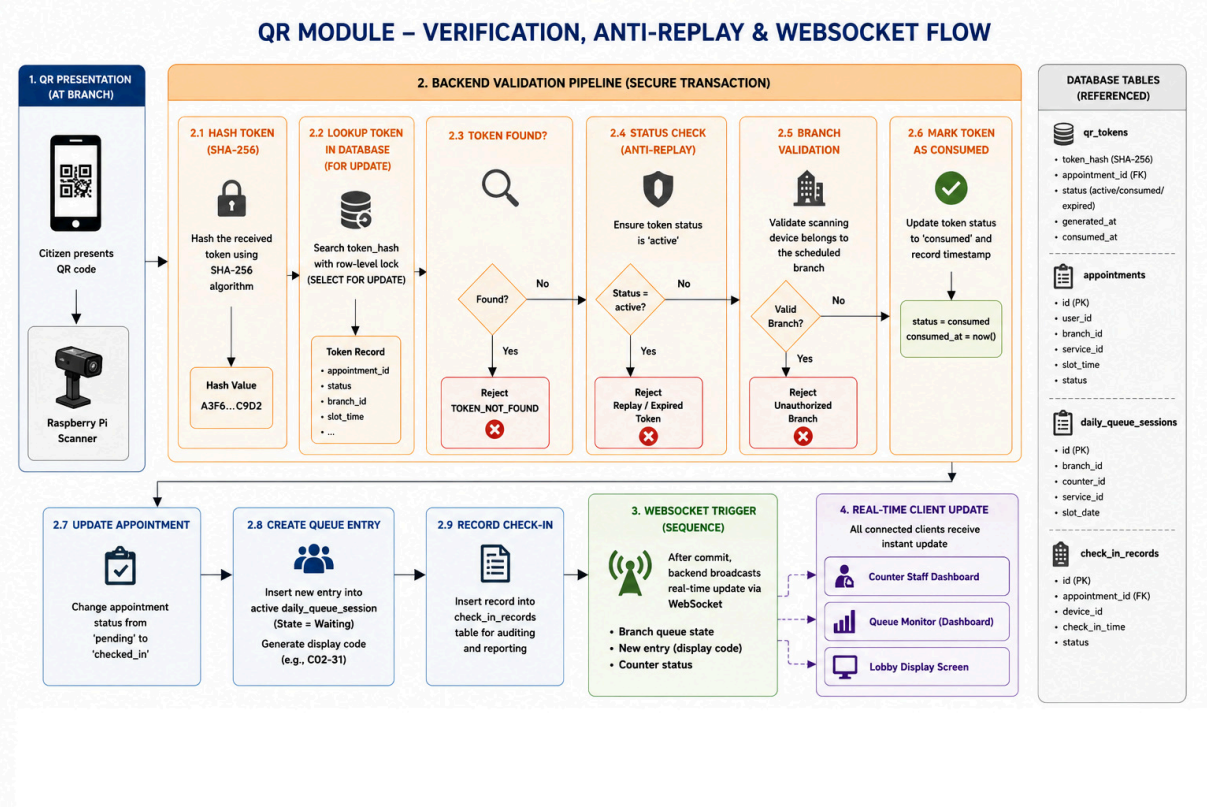


Figure A.5: Block diagram detailing the cryptographic verification flow, anti-replay validation, and WebSocket trigger sequence of the QR module.

QR Token Generation

When an appointment transaction is confirmed, `QrTokenService` generates a unique cryptographically secure token. The token string is constructed using a high-entropy cryptographically secure pseudo-random number generator (CSPRNG), concatenated with the appointment identifier and the scheduled slot timestamp. This plaintext string is then hashed using SHA-256 before storage in the database, ensuring that even if the database is compromised, the tokens cannot be reproduced.

QR Storage Strategy

The plain random string is transmitted to the mobile client over HTTPS and is never stored in its raw form on the server. The server database only preserves: * The SHA-256 hash of the token (`token_hash`). * The appointment foreign key link (`appointment_id`). * The consumption status (`status` $\in$ `active`, `consumed`, `expired`). * Generation and consumption timestamps.

QR Expiration Mechanism

Tokens are configured with strict temporal validation boundaries. The token record is evaluated against the arrival window:

$$t_{\text{check-in}} \in [T_{\text{slot_start}} - \Delta_{\text{buffer}}, T_{\text{slot_end}}]$$

where Δ_{buffer} represents the administrative arrival grace window (typically set to 15 minutes). If the scan timestamp $t_{\text{check-in}}$ falls outside this range, the validation engine marks the token status as `expired` and rejects the arrival.

QR Validation Process

When scanned, the scanner forwards the token value to the secure endpoint `/api/devices/qr/check-in`. The validation pipeline executes within an isolated database transaction: 1. The plaintext token is hashed using SHA-256. 2. The database is queried using the hashed value with a row-level lock (`SELECT FOR UPDATE`). 3. If no matching row is found, the system logs a `TOKEN_NOT_FOUND` security warning and returns a validation error. 4. The system validates that the token status is active. If already marked as consumed, the check-in is rejected as a replay attempt. 5. The scheduled branch of the appointment is validated against the scanning device's authorized branch ID.

One-Time Usage Logic (Anti-Replay)

To prevent a citizen from reusing their QR code or sharing it with others, the database transaction instantly updates the token status to `consumed` and records the exact timestamp of consumption. The database schema enforces a unique constraint on the appointment ID in the tokens table, guaranteeing that an appointment can never be linked to more than one valid token.

Check-In Processing

Upon successful token validation, the appointment status transitions from `pending` to `checked_in`. The backend generates the operational display code (e.g., `C02-31`), writes a new entry into the active daily queue session, and records the check-in event in `check_in_records` for auditing and performance reporting.

Queue Synchronization After Validation

Immediately after committing the check-in transaction, the backend dispatches a real-time sync job. This job generates a WebSocket payload containing the updated branch queue state, the newly activated display code, and counter status, broadcasting it instantly to counter staff and lobby display screens.

A.3.4 Real-Time Communication Module

The real-time layer uses WebSockets to distribute queue transactions instantly, ensuring that all branch screens, staff dashboards, and mobile clients maintain a synchronized view of operational states.

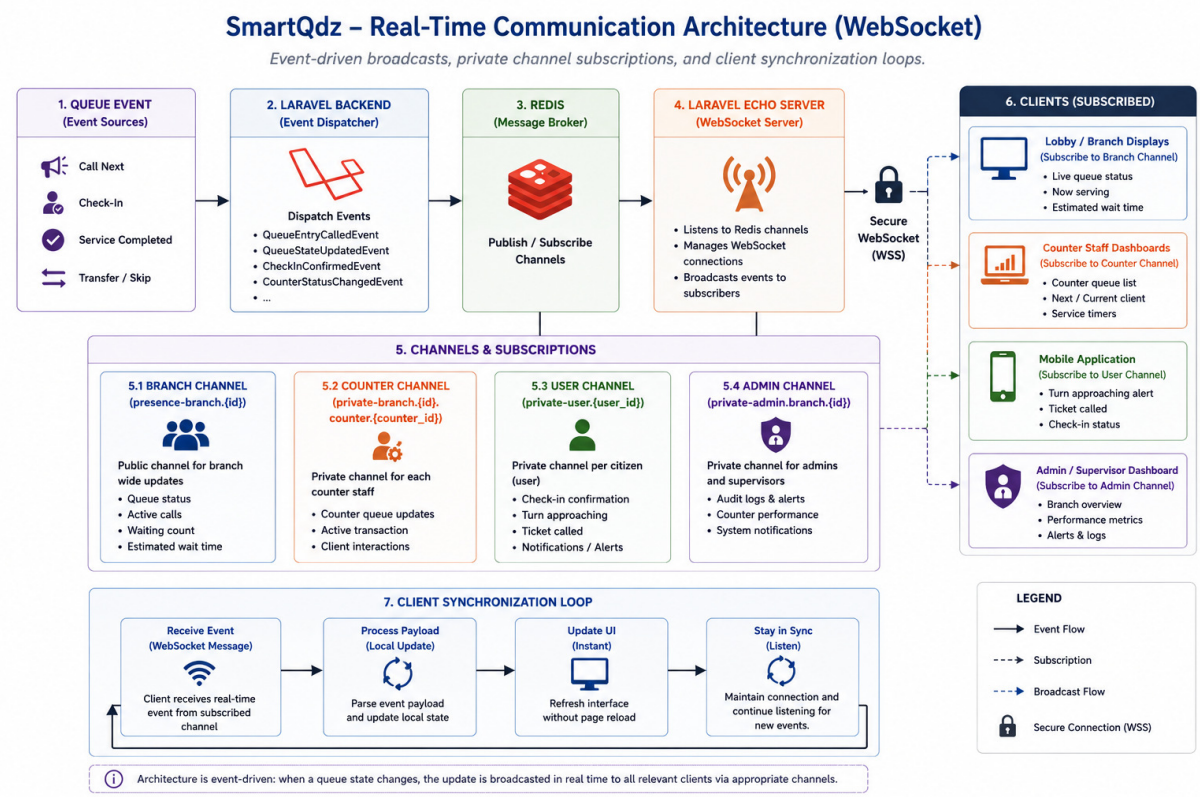


Figure A.6: Real-time network architecture diagram demonstrating event-driven broadcasts, private channel subscriptions, and client synchronization loops.

WebSocket Architecture

The real-time infrastructure uses an event-driven design. The Laravel backend dispatches events to a local Redis server acting as an in-memory database and message broker. A Node.js socket server (or Laravel Echo Server) listens to the Redis channels and pushes

the updates to connected WebSocket clients over secure TCP channels, maintaining low latency and minimizing server overhead.

Event Broadcasting

Operational controllers dispatch events using standard broadcast structures. An event class (e.g., `QueueEntryCalledEvent`) is initialized with the affected queue model and database identifiers. When serialized, the class exposes a `broadcastOn` method defining the targets, and a `broadcastWith` method containing the JSON message payload.

Channel Organization

To isolate messages and protect user privacy, the WebSocket layer organizes traffic into four distinct channel scopes:

- **Branch Channels (`presence-branch.{id}`):** Public display screens and lobby monitors subscribe to this channel. It broadcasts branch-wide queue status updates, display code calls, and estimated wait times.
- **Counter Channels (`private-branch.{id}.counter.{counter_id}`):** Dedicated staff interfaces subscribe to this channel, receiving targeted updates regarding their specific counter queue sequence, active transactions, and staff interactions.
- **User Channels (`private-user.{id}`):** Citizens' mobile applications subscribe to their respective private channels, receiving direct notifications when their check-in is confirmed, their turn is approaching, or their ticket is called.
- **Admin Channels (`private-admin.branch.{id}`):** General supervisors and branch administrators subscribe to this channel, receiving administrative audit alerts, counter latency updates, and performance overruns.

Queue Update Events

When an entry is updated (e.g., checked-in, called, or completed), the system broadcasts the update payload:

```
{
  "event": "QueueStateUpdated",
  "channel": "presence-branch.5",
  "data": {
```

```
    "session_date": "2026-05-31",
    "active_calls": [
      { "counter": 2, "display_code": "C02-31", "state": "called" }
    ],
    "waiting_count": 14,
    "timestamp": 1780222471
  }
}
```

Dashboard, Mobile, and Kiosk Synchronization

Upon receipt of a queue update event, subscribed clients process the payload locally: * **Dashboards:** Counter staff interface instantly redraws waiting counters and update active timers without full-page reloads. * **Mobile Clients:** The mobile interface updates the citizen's estimated wait time and triggers local haptic alerts when the ticket position approaches the front of the queue. * **Kiosk Devices:** Kiosks update real-time service indicators, disabling services that are temporarily suspended by administrators.

Error Recovery Strategy

If the WebSocket connection is interrupted due to network fluctuations, the client automatically initiates an exponential backoff reconnection loop. While disconnected, the client falls back to polling the REST API at extended intervals. Upon successful reconnection, the client queries the REST API for the latest state to synchronize any missed events and resubscribes to the secure channels.

A.3.5 Notification Module

The notification engine handles asynchronous user alerts, combining Firebase Cloud Messaging (FCM) and local queued dispatchers to ensure reliable delivery.

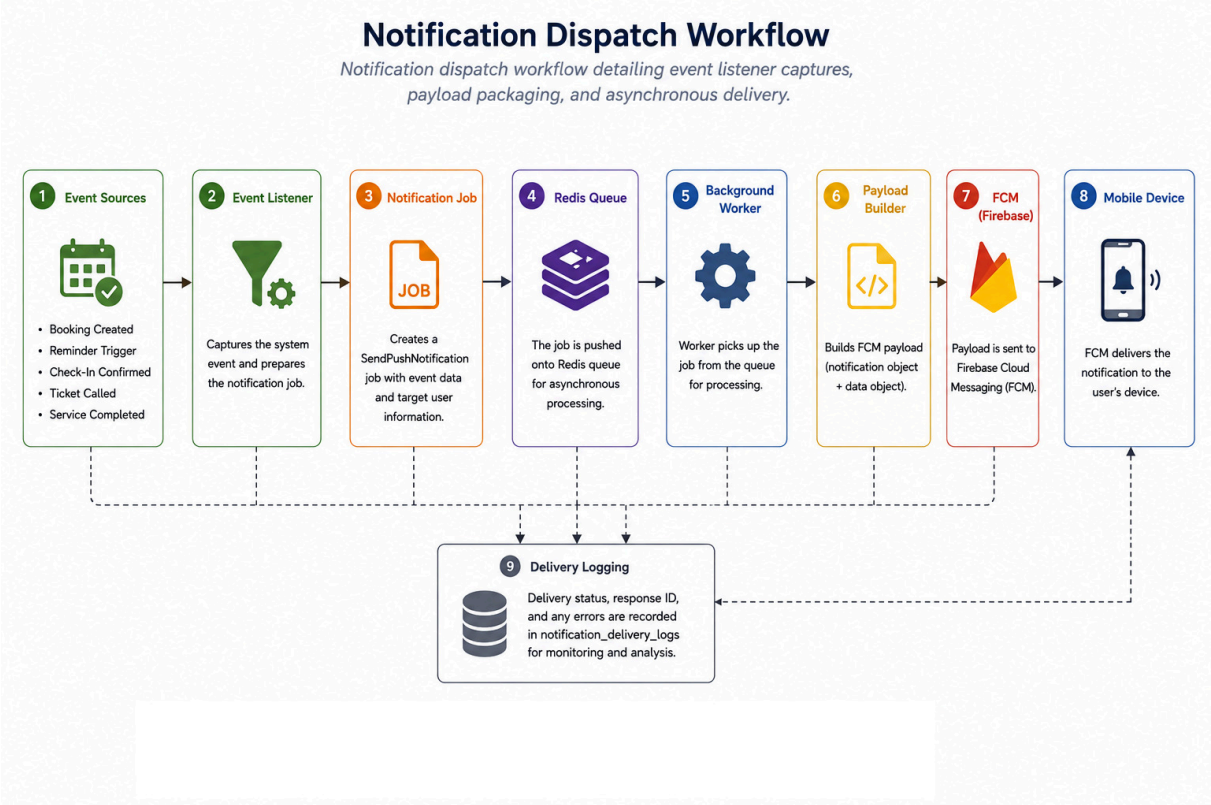


Figure A.7: Notification dispatch workflow detailing event listener captures, payload packaging, and asynchronous delivery.

Push Notifications

The mobile application registers the citizen's unique FCM device token upon login, storing it in `user_device_tokens`. When a system event triggers a notification, the backend builds an FCM payload containing: * A **Notification object** (title, body, sound) that triggers system-level alerts on iOS and Android. * A **Data object** (appointment ID, target route, status) that is parsed by the mobile app in the background to handle navigation.

Booking Confirmation Notifications

Immediately following a successful booking transaction, the system dispatches a confirmation payload containing the branch details, service category, date, slot time, and the unique reservation code, providing the citizen with an immediate record of their scheduled visit.

Reminder Notifications

To minimize no-shows, the backend runs a scheduled cron job every five minutes. The job scans for pending appointments scheduled to start within a configured threshold (e.g., 30 minutes). If matching appointments are found, a reminder is dispatched, advising the citizen to arrive at the branch for check-in.

Queue Progress Notifications

As counter staff call waiting customers, the system monitors the active positions of checked-in citizens. When the number of waiting citizens ahead of a citizen falls below a configured threshold (e.g., 3 people), the system dispatches a progress notification: "Your turn is approaching. Please prepare to proceed to Counter X".

Service Completion Notifications

Upon completion of a transaction, a completion payload is dispatched, summarizing the service duration and providing a link to a feedback form, enabling continuous evaluation of branch performance.

Notification Dispatch Flow

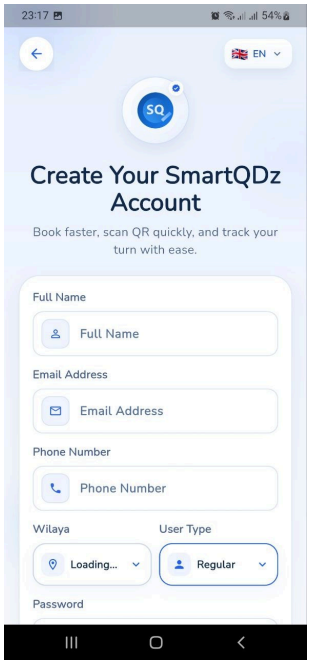
To preserve API response times, the notification module executes asynchronously: 1. An event listener catches a queue transaction and initializes a `SendPushNotification` job. 2. The job is pushed onto a Redis-backed queue system. 3. A background worker picks up the job, resolves the target device tokens, packages the FCM payload, and submits it to Firebase APIs. 4. The delivery status, response identifiers, and any transmission errors are logged in `notification_delivery_logs` for analysis.

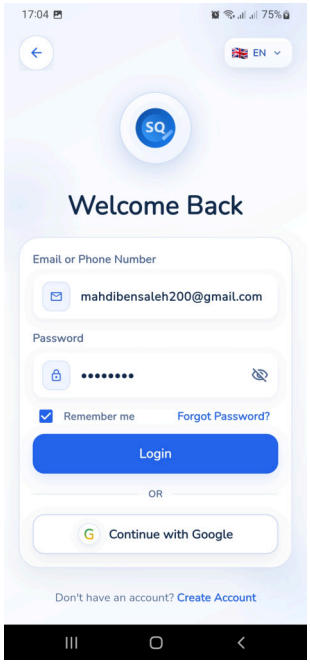
A.4 Additional Interface Documentation

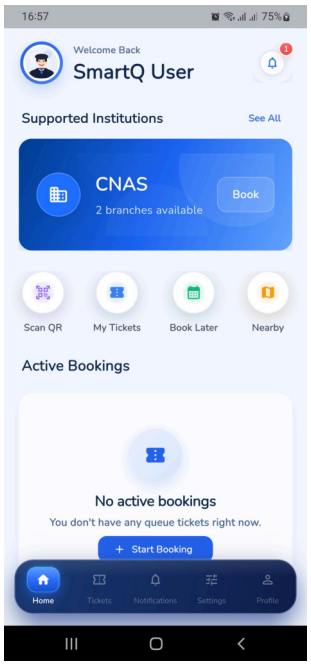
This section provides structural documentation for the visual interfaces of the platform, detailing the layouts and interactive states of the mobile application, administrative dashboard, kiosk terminal, and physical hardware prototype.

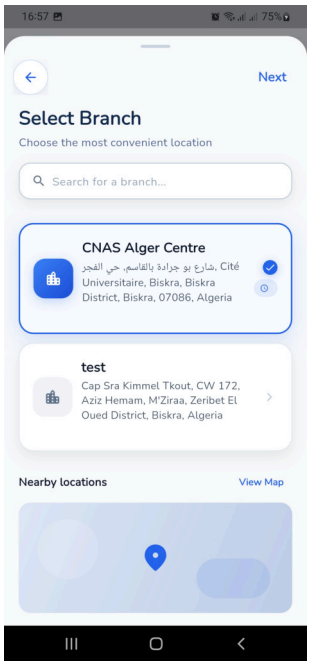
A.4.1 Mobile Application Screens

The mobile application is designed with an intuitive, step-by-step user interface.


Create Your SmartQDz Account
Book faster, scan QR quickly, and track your turn with ease.
Full Name
Email Address
Phone Number
Wilaya
User Type
Loading...
Regular
Password


Welcome Back
Email or Phone Number
mahdibensaleh200@gmail.com
Password
Remember me
Forgot Password?
Login
OR
Continue with Google
Don't have an account? Create Account


Welcome Back
SmartQ User
Supported Institutions
See All
CNAS
2 branches available
Book
Scan QR
My Tickets
Book Later
Nearby
Active Bookings
No active bookings
You don't have any queue tickets right now.
Start Booking
Home
Tickets
Notifications
Settings
Profile


Next
Select Branch
Choose the most convenient location
Search for a branch...
CNAS Alger Centre
Cité Universitaire, Biskra, Biskra District, Biskra, 07086, Algeria
test
Cap Sra Kimmel Tkout, CW 172, Aziz Hemam, M'Ziraa, Zeribet El Oued District, Biskra, Algeria
Nearby locations
View Map

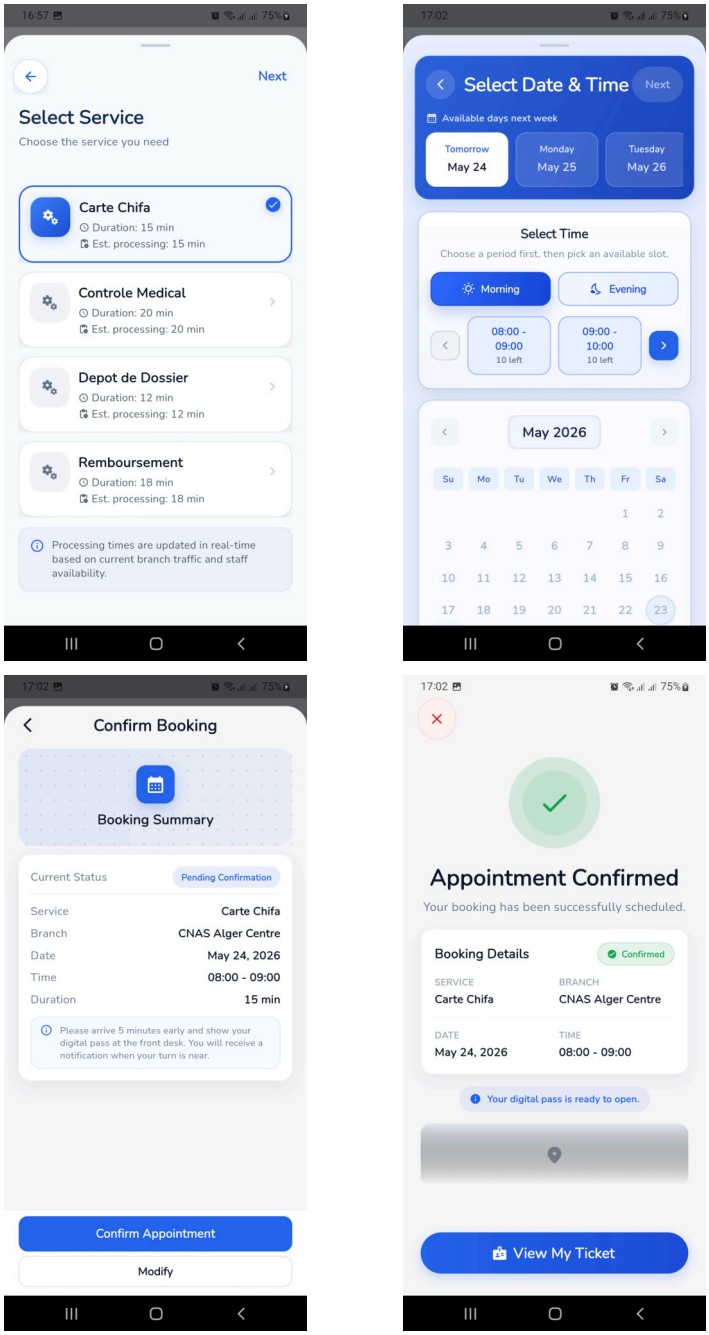


Figure A.8: Layout representations of the mobile application, showcasing registration, branch selection, scheduling, and active ticket views.

A.4.2 Administrative Dashboard Screens

The administrative dashboard provides branch managers and staff with comprehensive monitoring and control tools.

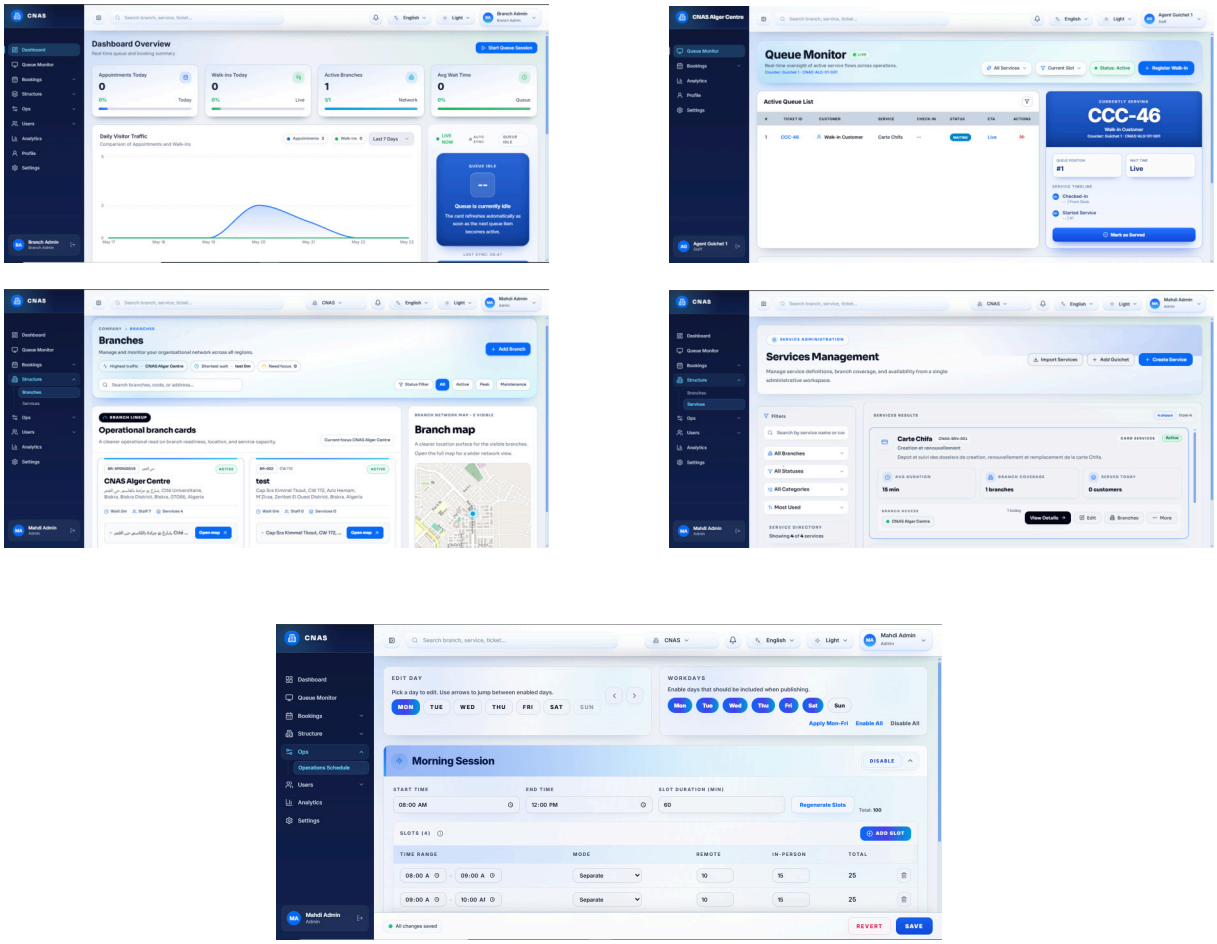


Figure A.9: Visual layout of the administrative control center, showcasing real-time queue states, counter controls, and analytics widgets.

A.4.3 Public Kiosk Screens

The kiosk terminal features high-contrast touch controls designed for self-service branch registration.

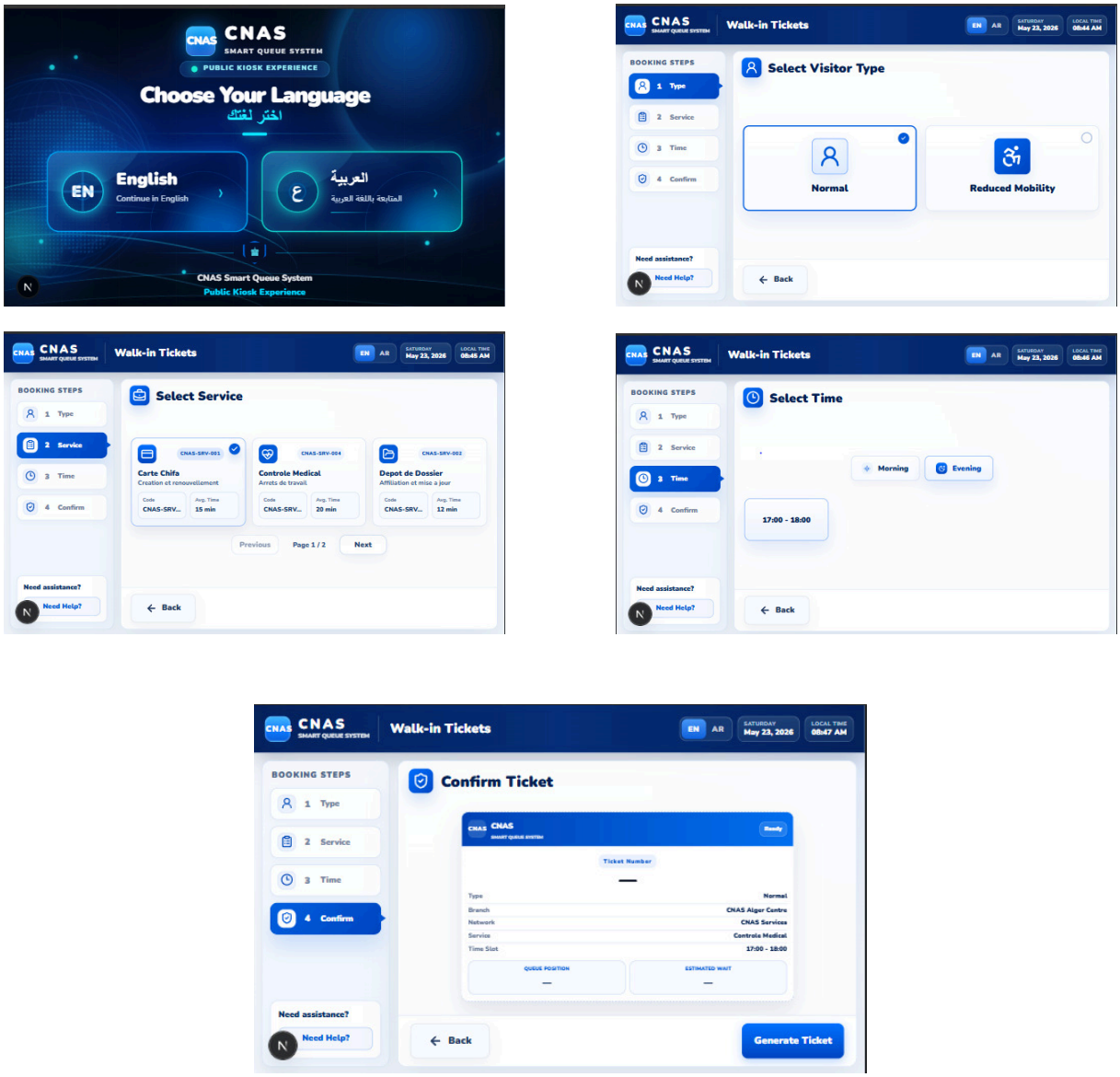


Figure A.10: Layout representation of the lobby kiosk interface, showcasing language selection, service categories, and confirmation views.

A.4.4 Hardware Prototype Images

This section provides visual references for the physical components of the lobby kiosk terminal and scanner assembly.

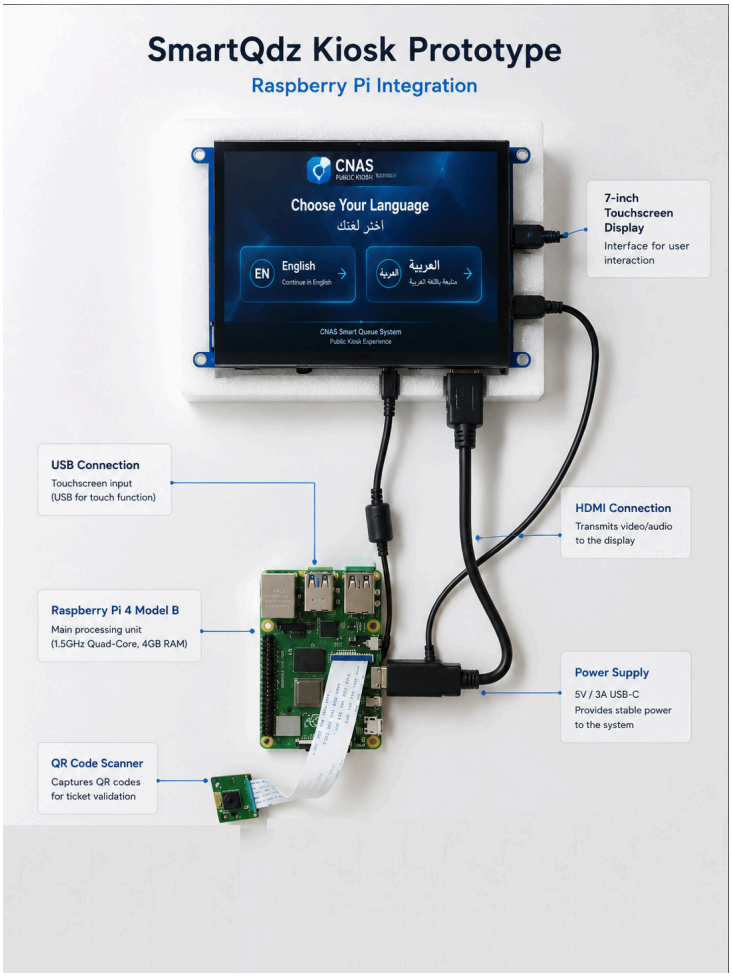


Figure A.11: Visual references of the Raspberry Pi 4 hardware prototype, showing enclosure details and touchscreen integration.



Figure A.12: Photographic views of the completed kiosk housing, thermal printer integrations, and scanner mounts.

A.5 Additional Technical Notes

This section provides supplementary technical notes regarding system scalability, security, hardware expansion, and deployment configurations.

A.5.1 System Scalability Considerations

The architecture of the platform is designed to scale horizontally to support high-volume public service networks. The stateless design of the Laravel API allows the application layer to be distributed across multiple redundant servers behind a load balancer. Read-heavy database operations can be optimized by implementing read/write replicas, routing queue status queries to read replicas while reserving the master database for write transactions.

A.5.2 Security Considerations

Security controls are enforced at every layer of the architecture: * **Data in Transit:** All communications between clients, devices, and servers are encrypted using TLS 1.3. * **Input Validation:** Every API endpoint implements strict schema validation, sanitizing incoming payloads to prevent SQL injection and cross-site scripting (XSS) attacks. * **Storage Encryption:** Sensitive database records, including passwords, audit logs, and JWT signatures, are encrypted or hashed using industry-standard cryptographic algorithms.

A.5.3 Future Hardware Expansion

The IoT integration layer can be expanded to support additional branch devices, such as digital counter displays, auditory call announcers, and passenger counting sensors, by defining standardized API contracts and secure device registration protocols.

A.5.4 Multi-Branch Deployment Strategy

Large-scale deployments can use a multi-tenant cloud database architecture. Centralized servers host the core application services, while each physical branch is assigned an isolated logical namespace, allowing national agencies to manage their entire branch network through a unified platform.

A.5.5 High Availability Considerations

To ensure service continuity during network interruptions, the platform uses redundant cloud database replicas, automated failover configurations, local offline caching on scanning devices, and secondary notification channels, guaranteeing that branch operations can continue even if connectivity is temporarily lost.

A.6 Conclusion

This appendix provides supplementary technical documentation, UML models, low-level implementation details, interface layouts, and hardware integration specifications for the SmartQdz platform. By documenting these engineering internals, this section ensures that the architecture, workflows, and integrations described in the main chapters are fully transparent and reproducible.

The blueprints, message payloads, security controls, and hardware integrations detailed herein provide systems engineers and deployment operators with the technical baseline required to deploy, maintain, and scale the SmartQdz platform, supporting efficient public service delivery.

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
اللجنة الوطنية التنسيقية لمتابعة الابتكار وحاضنات الأعمال الجامعية

دليل إعداد مشروع

SmartQdz

نظام ذكي لإدارة طوابير الانتظار باستخدام إنترنت الأشياء والمصادقة عبر رموز QR

2026

الغرض من الوثيقة	الحصول على شهادة مؤسسة ناشئة
الإطار التنظيمي	القرار الوزاري 1275
اسم المشروع	SmartQdz

بطاقة معلومات

حول فريق الإشراف وفريق العمل

1. فريق الإشراف

الصفة	المعلومات
المشرف الرئيسي	صهيب عياد

2. فريق العمل

الطالب	التخصص	الكلية/الجامعة
شنة عبد الرؤوف	قسم الإعلام الآلي	كلية العلوم الدقيقة وعلوم الطبيعة والحياة، جامعة محمد خيضر - بسكرة
بن صالح مهدي	قسم الإعلام الآلي	كلية العلوم الدقيقة وعلوم الطبيعة والحياة، جامعة محمد خيضر - بسكرة
لعجال مبروك عبد المؤمن	قسم الإعلام الآلي	كلية العلوم الدقيقة وعلوم الطبيعة والحياة، جامعة محمد خيضر - بسكرة

المحتويات

مقدمة **5**

المحور الأول: تقديم المشروع **6**

فكرة المشروع (الحل المقترح) 6

القيم المقترحة 7

فريق العمل 7

أهداف المشروع 7

مؤشرات الأداء المستهدفة 8

جدول زمني لتحقيق المشروع 8

المحور الثاني: الجوانب الابتكارية **9**

طبيعة الابتكارات 9

مجالات الابتكارات 9

الرؤية المستقبلية وقابلية التوسع 10

المحور الثالث: التحليل الاستراتيجي للسوق **11**

عرض القطاع السوقي 11

قياس شدة المنافسة 12

الاستراتيجية التسويقية 12

خطة دخول السوق 13

المحور الرابع: خطة الإنتاج والتنظيم **14**

عملية تقديم الخدمة 14

التقنيات والموارد المطلوبة 15

اليد العاملة 15

الشراكات الرئيسية 16

المحور الخامس: الخطة المالية **17**

نموذج الإيرادات 17

التكاليف والأعباء 18

رقم الأعمال والأرباح المتوقعة 18

خطة الخزينة 19

المحور السادس: النموذج الأولي التجريبي **20**

معمارية النظام 24

آلية عمل رمز QR 24

الأمن السيبراني وحماية البيانات 24

خاتمة **26**

الملاحق **27**

الملحق 01: الميزانية التقديرية للمؤسسة الناشئة 27

الملحق 02: جدول حسابات النتائج المتوقع 27

الملحق 03: خطة الخزينة 28

الملحق 04: مخطط نموذج الأعمال 28

مقدمة

تندرج هذه المذكرة ضمن دليل إعداد مشروع مؤسسة ناشئة في إطار القرار الوزاري 1275، وتهدف إلى عرض مشروع **SmartQdz** بوصفه نظاماً ذكياً لإدارة طوابير الانتظار داخل المؤسسات العامة والخدمية.

يعالج المشروع مشكلة الاكتظاظ وطول أوقات الانتظار وضعف التحقق من الحضور في الطوابير التقليدية، من خلال منصة رقمية تجمع بين تطبيق موجه للمواطنين، بوابة تشغيل للموظفين، لوحة تحكم إدارية، وأجهزة متصلة لإنجاز المصادقة الميدانية عبر رموز QR ديناميكية.

هوية المشروع

اسم المشروع: SmartQdz.

طبيعة المشروع: منصة رقمية مدعمة بإنترنت الأشياء لتنظيم الطوابير، حجز المواعيد، استخراج التذاكر، وتأكيد الحضور الذاتي داخل المؤسسة.

الفئة المستهدفة: المؤسسات الإدارية والخدمية التي تستقبل أعداداً كبيرة من المواطنين وتحتاج إلى تنظيم تدفقهم بشكل آمن وشفاف.

المحور الأول: تقديم المشروع

فكرة المشروع (الحل المقترح)

تعريف مختصر بالمشروع

SmartQdz هو نظام ذكي لإدارة طوابير الانتظار باستخدام إنترنت الأشياء والمصادقة عبر رموز QR. يهدف المشروع إلى عصرنة حجز المواعيد، استخراج التذاكر الفورية، وتأكيد الحضور الذاتي داخل المؤسسات.

مجال النشاط: تكنولوجيا المعلومات، تطوير المنصات الرقمية، تطبيقات الهاتف، وإنترنت الأشياء.

نشأة الفكرة وطبيعة الحل: تعاني العديد من المؤسسات العامة والخدمية من الاكتظاظ، طول أوقات الانتظار، ضعف تنظيم تدفق المواطنين، وصعوبة التحقق من الوصول الحقيقي في الطوابير التقليدية. يقترح مشروع SmartQdz منصة رقمية متكاملة تمكّن المواطن من حجز الخدمة مسبقاً، والحصول على تذكرة رقمية، ثم تأكيد حضوره عند الوصول عبر رمز QR ديناميكي ومحدود الصلاحية.

وصف مختصر للمشروع: يعتمد النظام على بنية متعددة الطبقات تشمل تطبيق هاتف للمواطنين، بوابة عمليات للموظفين، لوحة تحكم إدارية، وأجهزة متصلة مثل أكشاك QR والماسحات الضوئية. يقوم الخادم المركزي بتوليد رمز استجابة سريعة فريد يستعمل لتأكيد الحضور الآمن ومنع إعادة استخدام التذكرة أو تمريرها بطريقة غير شرعية.



شكل 1: شعار أو صورة توضيحية للمشروع

القيم المقترحة

القيمة الموجهة للمواطن

يتيح النظام للمواطن التسجيل، اختيار الفرع والخدمة، حجز موعد عن بعد، الحصول على تذكرة رقمية، وتجنب فترات الانتظار العشوائية داخل المؤسسة. الهدف العملي هو تقليل زمن الانتظار بنسبة قد تصل إلى 60% في المؤسسات ذات الضغط المرتفع.

القيمة الموجهة للمؤسسة

يوفّر SmartQdz توثيقاً دقيقاً وآمناً للحضور، يحد من الاستخدام المزدوج للتذاكر، يمنح الموظفين واجهة تشغيل لحظية، ويوفر للإدارة لوحة تحكم لمراقبة نشاط الطابور وتسجيل نتائج الخدمة. كما يساعد على تقليل الضغط داخل قاعات الانتظار، تحسين إنتاجية الموظفين، وتوفير بيانات دقيقة حول أوقات الذروة.

فريق العمل

جدول 1: توزيع أدوار فريق العمل

العضو	التخصص	الدور في المشروع
شنة عبد الرؤوف	قسم الإعلام الآلي	تطوير التطبيق والواجهات وتجربة المستخدم
بن صالح مهدي	قسم الإعلام الآلي	تطوير الخادم المركزي وقواعد البيانات وواجهات API
لعجال مبروك عبد المؤمن	قسم الإعلام الآلي	التكامل مع أجهزة إنترنت الأشياء واختبار النموذج الأولي

أهداف المشروع

1. تزويد المواطنين بقناة رقمية للحجز وتتبع الطوابير.
2. تعزيز التحقق من الوصول عبر مصادقة QR ديناميكية.
3. توفير واجهة تشغيلية للموظفين للتعامل مع الطوابير في الوقت الفعلي.
4. منح الإداريين تحكماً مركزياً في الفروع والخدمات والمستخدمين.
5. تحسين جودة الخدمة وتقليل الضغط داخل مقرات المؤسسات.

مؤشرات الأداء المستهدفة

جدول 2: مؤشرات الأداء الرئيسية *KPI*

المؤشر	القيمة المستهدفة	طريقة القياس
تقليل مدة الانتظار	من 40% إلى 60%	مقارنة متوسط الانتظار قبل وبعد اعتماد النظام
نسبة الحضور المؤكد	أكثر من 90%	عدد رموز QR الممسوحة مقارنة بالحجوزات المؤكدة
المؤسسات المستهدفة في السنة الأولى	5 مؤسسات محلية	عقود تجريبية أو تجارية
عدد المستخدمين المتوقع	10,000 مستخدم خلال السنة الأولى	حسابات المواطنين المسجلة في التطبيق
معدل الأخطاء في التذاكر	أقل من 2%	الحالات الملغاة أو المرفوضة أثناء التحقق

جدول زمني لتحقيق المشروع

جدول 3: الجدول الزمني المقترح لإنجاز المشروع

المرحلة	الأعمال الأساسية	المدة التقديرية
التحليل والتصميم	تحديد المتطلبات، تصميم قاعدة البيانات، إعداد مخطط الواجهات	شهر واحد
التطوير البرمجي	بناء تطبيق الهاتف، API، لوحة التحكم، وواجهة الموظف	شهران
تكامل الأجهزة	ربط ماسحات QR وأجهزة Pi Raspberry بالخادم	شهر واحد
الاختبار والتحسين	تجربة النموذج الأولي، قياس الأداء، تصحيح الأخطاء	شهر واحد

المحور الثاني: الجوانب الابتكارية

طبيعة الابتكارات

يقوم الابتكار في مشروع SmartQdz على ابتكار في العمليات ودمج التكنولوجيا المادية والرقمية. فبدل الاعتماد على طابور تقليدي أو تذاكر ثابتة قابلة للتبادل، يعتمد النظام على مسار رقمي يبدأ من حجز الخدمة عن بعد وينتهي بتأكيد الحضور داخل المؤسسة عبر رمز QR ديناميكي.

ابتكار في العمليات

يعيد المشروع تنظيم طريقة استقبال المواطنين، ترتيب الأولويات، استدعاء المستفيدين، وتسجيل نتائج الخدمة. وبذلك لا يقتصر الحل على رقمنة التذكرة، بل يغيّر طريقة إدارة الطابور كاملة من لحظة الحجز إلى لحظة تقديم الخدمة.

دمج التكنولوجيا المادية والرقمية

يربط النظام تطبيق الهاتف والخادم المركزي بلوحات التحكم وأجهزة المسح داخل المؤسسة، مما يخلق تزامناً فورياً بين حضور المواطن في المكان الفعلي وحالته الرقمية داخل النظام.

مجالات الابتكارات

- **المصادقة الآمنة والديناميكية:** يستخدم SmartQdz رموز QR مشفرة ومحدودة الصلاحية يتم التحقق منها عبر خادم مركزي لمنع إعادة الاستخدام أو التلاعب.
- **التكامل مع إنترنت الأشياء:** تعتمد المنصة على أجهزة مسح مثل Pi Raspberry والأكشاك والشاشات الذكية لتحديث حالة الطابور فورياً.
- **تجربة مستخدم أكثر وضوحاً:** يستطيع المواطن حجز الخدمة ومتابعة وضعه دون البقاء لساعات داخل المؤسسة.
- **تحكم إداري مركزي:** تمنح لوحة الإدارة صلاحيات لإدارة الفروع والخدمات والمستخدمين ومراقبة الضغط على كل فرع.

- **تقليل المخاطر التشغيلية:** يساعد النظام على الحد من الازدحام، الأخطاء اليدوية، وتكرار التذاكر.



شكل 2: مخطط أو رسم يوضح الابتكار في *SmartQdz*

الرؤية المستقبلية وقابلية التوسع

لا يقتصر SmartQdz على إدارة طابور داخل فرع واحد، بل يمكن تطويره تدريجياً ليصبح منصة وطنية لإدارة تدفق المواطنين في المؤسسات الخدمية. تعتمد قابلية التوسع على بنية سحابية متعددة الفروع، تسمح بإضافة مؤسسات جديدة وخدمات جديدة دون إعادة بناء النظام.

- **التنبؤ بالازدحام بالذكاء الاصطناعي:** تحليل البيانات التاريخية للتنبؤ بأوقات الذروة وتوجيه الموارد البشرية.
- **منصة وطنية متعددة الفروع:** تمكين المؤسسات من إدارة عدة فروع وخدمات من لوحة تحكم واحدة.
- **إشعارات ذكية:** إرسال تنبيهات للمواطن حول موعده، تغيّر الطابور، أو اقتراب دوره.
- **إحصائيات متقدمة:** استخراج تقارير حول مدة الانتظار، عدد الخدمات المنجزة، ونسب الغياب.
- **تكاملات مستقبلية:** ربط النظام بأنظمة الدفع الإلكتروني أو الهوية الرقمية عند توفر الأطر التنظيمية المناسبة.

المحور الثالث: التحليل الاستراتيجي للسوق

عرض القطاع السوقي

السوق المستهدف: يستهدف مشروع SmartQdz المؤسسات الإدارية المحلية مثل البلديات والدوائر، مراكز البريد، المستشفيات، العيادات، وشركات تقديم الخدمات العامة التي تستقبل أعداداً كبيرة من المواطنين وتحتاج إلى تنظيم تدفقهم بطريقة فعالة.

حاجة السوق: تعاني هذه المؤسسات من الاكتظاظ، الانتظار غير المنظم، صعوبة قياس جودة الخدمة، وغياب بيانات دقيقة عن أوقات الذروة ونسب الحضور. لذلك تمثل حلول إدارة الطوابير الذكية فرصة مهمة لتحسين تجربة المواطن ورفع كفاءة المؤسسة. كما أن المؤسسات الصغيرة والمتوسطة تحتاج إلى حل أقل تكلفة من الأنظمة الدولية الثقيلة، وقابل للتكييف مع الواقع المحلي.

الشريحة الأولى المستهدفة: المؤسسات العمومية والمحلية ذات الضغط اليومي المرتفع، لأنها الأكثر حاجة إلى حل يجمع بين الحجز عن بعد، التحقق الميداني، ولوحة متابعة لحظية.

قياس شدة المنافسة

جدول 4: تحليل المنافسين

المنافس	نقاط القوة	نقاط الضعف مقارنة بمشروعنا	الحصة السوقية
الأنظمة التقليدية	بسيطة ولا تتطلب تطبيقاً أو خادماً	لا تدعم الحجز المسبق، لا توفر إحصائيات، ولا تمنع التلاعب بالتذاكر	واسعة محلياً
Qmatic	تحليل تشغيلي قوي وتوجيه للمؤسسات الكبرى	تكلفة نشر عالية واعتماد مفرط على الأجهزة باهظة الثمن	عالية دولياً
Skiplino	حجز عن بعد وإشعارات جيدة وسرعة نشر	تركيز أقل على التحقق المادي المشفر للحضور عبر QR Check-in	متوسطة
Wavetec	أجهزة قوية وتزامن جيد للطوابير	تعقيد في الدمج وتركيز أقل على الرموز الديناميكية المنتهية الصلاحية	عالية دولياً
حلول محلية بسيطة	تكلفة منخفضة وسهولة تركيب	غالباً لا تستعمل إنترنت الأشياء ولا توفر إدارة مركزية متقدمة	محدودة

موقع SmartQdz في السوق

يتميز المشروع بتركيزه على المؤسسات المحلية التي تحتاج إلى حل أقل تكلفة وأكثر قابلية للتكييف، مع مصادقة ميدانية عبر QR ديناميكي وتكامل مع أجهزة بسيطة مثل Raspberry Pi. يجمع SmartQdz بين الحجز عن بعد، إنترنت الأشياء، الإدارة المركزية، والتكلفة المنخفضة.

الاستراتيجية التسويقية

- **المنتج:** منصة رقمية متكاملة لإدارة الطوابير والحجوزات والتحقق من الحضور.
- **السعر:** اعتماد نموذج مرن يجمع بين بيع الأجهزة والاشتراك الشهري أو السنوي.
- **التوزيع:** البيع المباشر للمؤسسات، مع إمكانية نشر تطبيق المواطن عبر متاجر التطبيقات.
- **الترويج:** عروض تجريبية داخل مؤسسات محلية، المشاركة في المعارض الجامعية والتكنولوجية، وتقديم دراسات حالة تثبت تقليل وقت الانتظار.
- **القنوات الرقمية:** نشر محتوى مهني عبر LinkedIn وصفحات المؤسسات، مع عروض تقديمية موجهة لصناع القرار.

- **الشراكات:** التعاون مع البلديات، حاضنات الأعمال، وموردي الأجهزة الإلكترونية لتسهيل التجارب الأولى.

خطة دخول السوق

جدول 5: مراحل الوصول إلى السوق

المرحلة	الإجراء	الهدف
التجربة الأولى	عرض تجريبي مجاني أو منخفض التكلفة داخل بلدية أو مصلحة جامعية	إثبات تقليل الانتظار وجمع بيانات واقعية
البيع المباشر	تقديم عرض تجاري للمؤسسات ذات الطوابير اليومية	الحصول على أول عقود مدفوعة
التوسع المحلي	استهداف البلديات، مراكز البريد، والعيادات في نفس الولاية	تخفيض تكلفة التركيب والدعم
التوسع الوطني	اعتماد نموذج سحابي متعدد الفروع وشركاء تركيب محليين	الوصول إلى ولايات ومؤسسات جديدة

المحور الرابع: خطة الإنتاج والتنظيم

عملية تقديم الخدمة

يعتمد إنتاج الخدمة في SmartQdz على مسار رقمي وميداني مترابط، يبدأ من طلب الخدمة عن بعد وينتهي بتقديم الخدمة داخل المؤسسة بعد التحقق من الحضور.

جدول 6: مراحل تقديم الخدمة

المرحلة	الوصف	الأداة المعتمدة
1	يختار المستخدم الفرع والخدمة ويقوم بالحجز عبر تطبيق الهاتف	تطبيق المواطن
2	يصدر الخادم المركزي تذكرة رقمية ورمز QR فريد	API Laravel وقاعدة بيانات MySQL
3	عند الوصول، يمسح المواطن الرمز عبر الكشك أو الماسح الضوئي	Scanner QR
4	يتحقق الخادم من صلاحية الرمز ويحدّث حالة الطابور	الخادم المركزي ولوحة التشغيل
5	يستدعي الموظف المواطن ويُسجل نتيجة الخدمة	بوابة الموظف ولوحة الإدارة



شكل 3: مخطط مراحل تقديم الخدمة في *SmartQdz*

التقنيات والموارد المطلوبة

- **تطبيق الهاتف:** تطوير تطبيق موجه للمواطنين باستخدام Flutter وDart لضمان العمل على أنظمة الهواتف الذكية المختلفة.
- **الخادم المركزي:** بناء واجهات API RESTful باستخدام Laravel لإدارة الحجوزات، المستخدمين، الخدمات، والتذاكر.
- **قاعدة البيانات:** استعمال MySQL لتخزين بيانات الفروع والخدمات والحجوزات وحالات الطابور.
- **الأجهزة المتصلة:** استخدام Pi Raspberry أو ماسحات QR للتحقق من حضور المواطنين داخل المؤسسة.
- **لوحات التشغيل:** توفير واجهة للموظفين ولوحة تحكم إدارية لمتابعة النشاط في الوقت الفعلي.

اليد العاملة

جدول 7: اليد العاملة والمهارات المطلوبة

المورد البشري	المهام
مطورو تطبيقات الهاتف	تطوير تطبيق المواطن وتحسين تجربة الاستخدام
مطورو الواجهة الخلفية	بناء API، قواعد الحجز، وإدارة قاعدة البيانات
مهندسو إنترنت الأشياء	تركيب وصيانة ماسحات QR وأجهزة Pi Raspberry
فريق الدعم والصيانة	متابعة الأعطال، التحديثات، ودعم المؤسسات المتعاقدة

الشراكات الرئيسية

تحتاج المؤسسة الناشئة إلى شراكات مع الهيئات الحكومية والمؤسسات العمومية لتجربة الحل، ومع موردي الأجهزة الإلكترونية لتوفير ماسحات QR والأكشاك، إضافة إلى مزودي الخدمات السحابية لضمان استضافة مستقرة وآمنة للمنصة.

المحور الخامس: الخطة المالية

الأرقام التالية تقديرية وتهدف إلى بناء تصور أولي قابل للنقاش أمام لجنة التقييم. يمكن تعديلها لاحقاً بعد الحصول على عروض أسعار حقيقية للأجهزة والاستضافة.

نموذج الإيرادات

يعتمد SmartQdz على نموذج عمل مختلط يجمع بين البرمجيات كخدمة، بيع الأجهزة، والدعم التقني.

جدول 8: نموذج الإيرادات المقترح

مصدر الإيراد	الوصف	التسعير التقديري
تركيب الحل المقترح	تجميل البرامج و تكوين العمال بالمؤسسة و تركيب الشاشات و الاكشاك	250,000 دج لكل مؤسسة
اشتراك المنصة SaaS	استعمال لوحة الإدارة، واجهة الموظف، وتطبيق المواطن	25,000 دج شهرياً لكل مؤسسة
كراء الاكشاك	شاشات و كشك تاكيد الحضور و استخراج التذاكر	15,000" دج شهريا لكل نقطة تحقق
الصيانة والدعم	تحديثات، مراقبة، دعم فني، وتكوين الموظفين	10,000 دج شهرياً لكل مؤسسة
لوحة تحكم متقدمة	إحصائيات تفصيلية، تقارير، وصلاحيات موسعة	15,000 دج شهرياً كخيار Premium
التخصيص والتكامل	ربط النظام بأنظمة داخلية أو متطلبات خاصة	حسب حجم المشروع

التكاليف والأعباء

جدول 9: تكلفة الإطلاق الأولي

العنصر	التكلفة التقديرية	ملاحظة
تطوير التطبيق والمنصة	50,000 دج	تكلفة داخلية أولية للنموذج القابل للعرض
الكشك و قارئ QR	75,000 دج	لكل نقطة تحقق أساسية
شاشة وملحقات تركيب	30,000 دج	حسب تجهيزات المؤسسة
خادم واستضافة أولية	20,000 دج	فترة تجريبية أو شهر أول من التشغيل
اسم نطاق وشهادة أمان	10,000 دج	حماية الاتصال والهوية الرقمية للمنصة
التسويق والعروض التجريبية	40,000 دج	وثائق، تنقلات، وعروض للمؤسسات
التركيب والدعم الأولي	30,000 دج	تجربة ميدانية وتكوين موظفين
المجموع التقديري	**255,000 دج**	قابل للتغير حسب عدد الأجهزة

رقم الأعمال والأرباح المتوقعة

جدول 10: سيناريوهات رقم الأعمال في السنة الأولى

السيناريو	الفرضية	رقم الأعمال المتوقع	النتيجة التقريبية
محافظ	5 مؤسسات، متوسط 6 أشهر اشتراك، ونقطة تحقق واحدة لكل مؤسسة	1,400,000 دج	ربح أولي بعد تغطية تكاليف الإطلاق والتشغيل
متوسط	12 مؤسسة، متوسط 8 أشهر اشتراك، ونقطة تحقق واحدة لكل مؤسسة	4,200,000 دج	قدرة على توسيع الفريق والدعم الفني
متفائل	20 مؤسسة، اشتراك سنوي، مع خدمات Premium لبعض المؤسسات	أكثر من 8,000,000 دج	توسع تجاري على مستوى عدة ولايات

طريقة حساب السيناريو المحافظ

تم احتساب السيناريو المحافظ على أساس 5 مؤسسات، كل مؤسسة تدفع 25,000 دج شهرياً لمدة 6 أشهر، مع بيع نقطة تحقق واحدة بقيمة 70,000 دج، ودعم شهري بقيمة 10,000 دج. هذا يعطي تصوراً أولياً قابلاً للتعديل حسب العقود الفعلية.

خطة الخزينة

جدول 11: خطة خزينة مختصرة للسنة الأولى

الفترة	المداخيل	المصاريف
الأشهر 1-3	تمويل ذاتي، دعم حاضنة، أو أول عرض تجريبي	تطوير المنتج، شراء أول جهاز، وتجهيز العرض التجاري
الأشهر 4-6	عقود تجريبية مدفوعة جزئياً أو بيع أول تجهيزات	التركيب، الدعم، الاستضافة، والتنقل
الأشهر 7-9	اشتراكات شهرية من أول المؤسسات المتعاقدة	دعم فني، تحسين المنتج، وتسويق مباشر
الأشهر 10-12	عقود اشتراك سنوية وصيانة دورية	توسيع الفريق، شراء أجهزة إضافية، وتحسين الأمن

المحور السادس: النموذج الأولي التجريبي

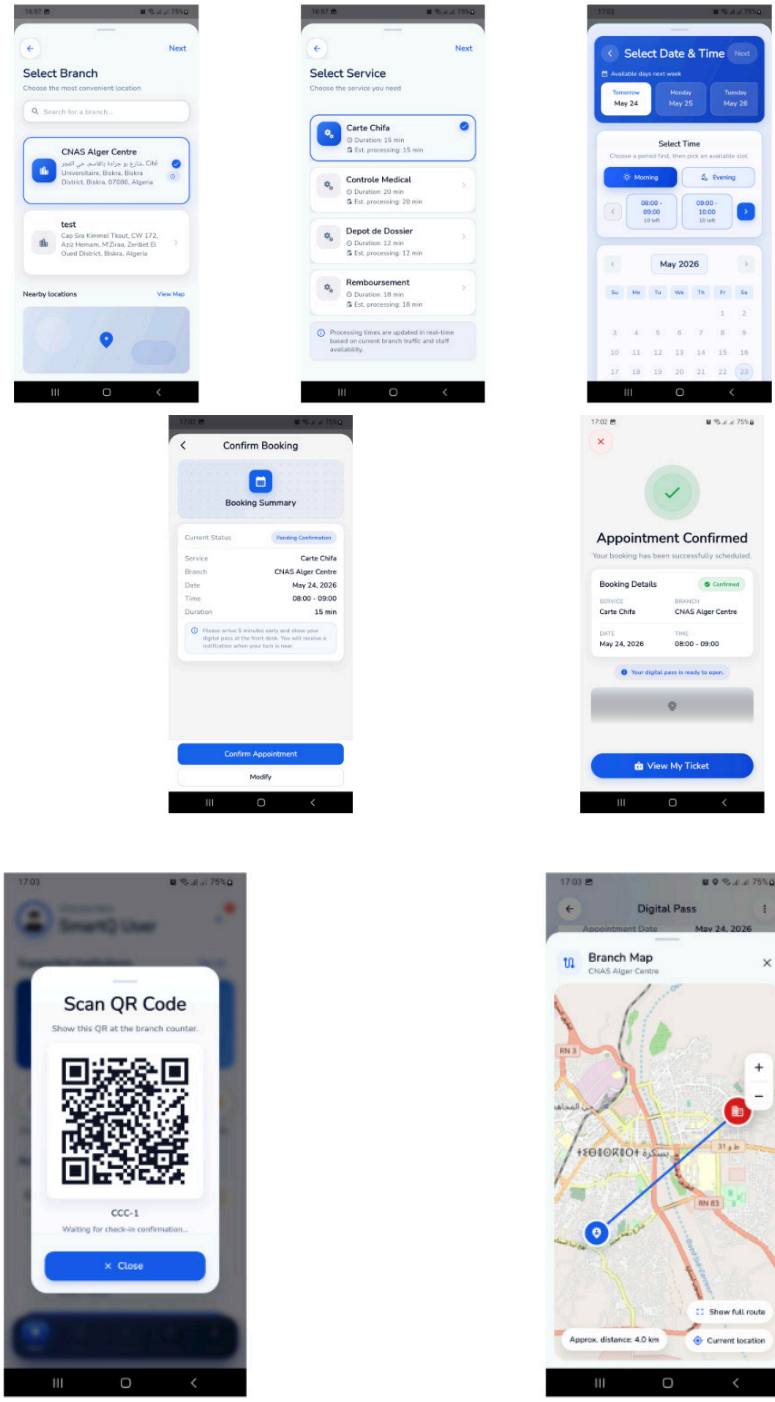
تم بناء نموذج أولي متكامل يثبت الجدوى التقنية لمشروع SmartQdz من خلال ربط تطبيق الهاتف بالخادم المركزي وأجهزة التحقق الميدانية.

مكونات النموذج الأولي

يبرهن النموذج الأولي على إمكانية حجز الخدمة، توليد تذكرة رقمية، إنشاء رمز QR ديناميكي، التحقق من الرمز داخل المؤسسة، وتحديث حالة الطابور فورياً في واجهة الموظف ولوحة الإدارة.

جدول 12: طبقات النموذج الأولي

الطبقة	الوصف التقني
تطبيق الهاتف	مبرمج باستخدام Flutter وDart لضمان العمل على مختلف أنظمة الهواتف الذكية
الخادم المركزي	مبني باستخدام API Laravel لتسيير الحجوزات، التذاكر، قواعد البيانات، وقواعد التحقق
قاعدة البيانات	تعتمد على MySQL لتخزين بيانات المستخدمين والفروع والخدمات وحالات الطابور
الأجهزة المتصلة	استخدام Pi Raspberry أو ماسح QR لإرسال طلبات تأكيد الحضور فورياً إلى الخادم
لوحة الإدارة	واجهة لمتابعة الفروع والخدمات والمستخدمين ومؤشرات نشاط الطابور



شكل 4: واجهة تطبيق المواطن

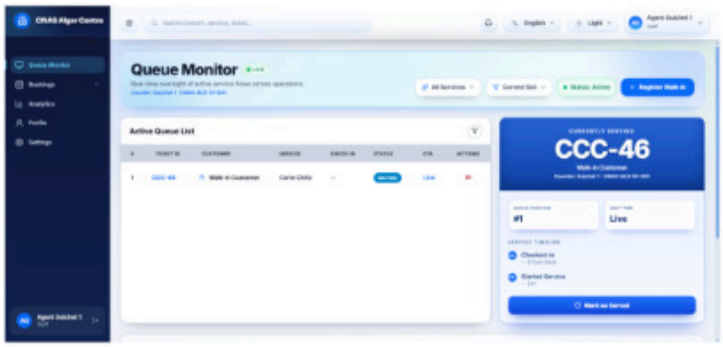


Figure 4.9: Queue-management screen.

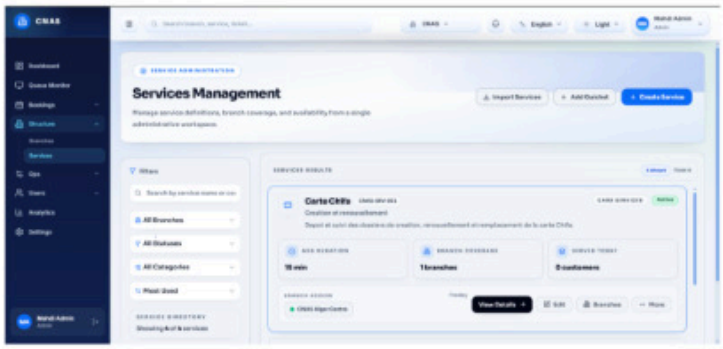


Figure 4.10: Service-configuration screen.

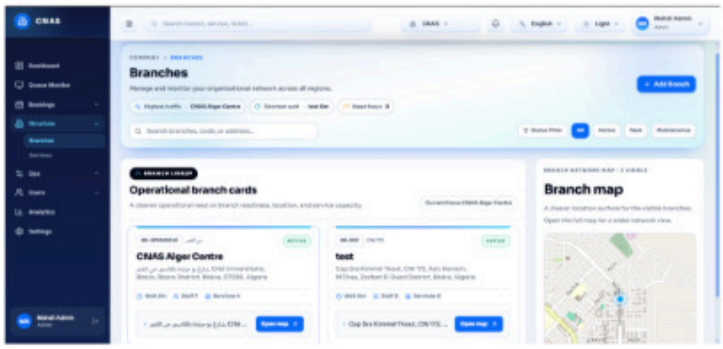
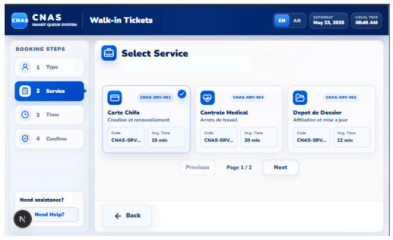


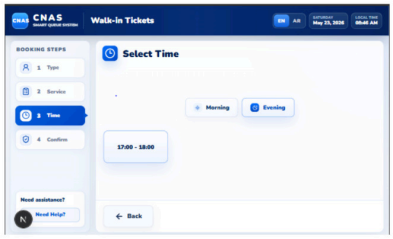
Figure 4.11: User and branch management screen.

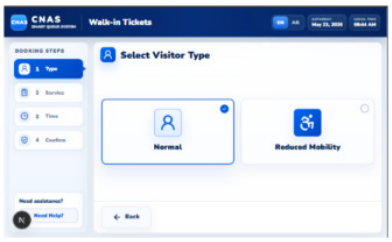
شكل 5: واجهة الموظف أو لوحة الإدارة

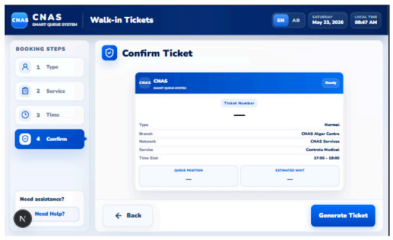












شكل 6: جهاز مسح رمز *QR* داخل المؤسسة

معمارية النظام

جدول 13: المسار التقني للنظام

المكون	الدور داخل المعمارية
تطبيق المواطن	إرسال طلب الحجز، عرض التذكرة، واستقبال الإشعارات
API Laravel	معالجة الطلبات، توليد QR، التحقق من الصلاحية، وتحديث حالة الطابور
Database MySQL	تخزين المستخدمين، الفروع، الخدمات، الحجوزات، وسجلات الحضور
Scanner QR أو Pi Raspberry	قراءة الرمز داخل المؤسسة وإرسال طلب تحقق إلى الخادم
واجهة الموظف	استدعاء المواطنين، تسجيل الخدمة، ومتابعة الطابور في الوقت الفعلي
لوحة الإدارة	إدارة الفروع والخدمات والمستخدمين ومراقبة المؤشرات

تدفق البيانات

المواطن يحجز الخدمة من التطبيق ← الخادم يولد تذكرة ورمز QR ← المواطن يمسح الرمز عند الوصول ← جهاز التحقق يرسل الطلب إلى الخادم ← الخادم يحدّث حالة الطابور ← الموظف يستدعي المواطن من واجهة التشغيل.

آلية عمل رمز QR

1. عند إنشاء الحجز، يولّد الخادم رمزاً فريداً مرتبطاً بالمستخدم، الفرع، الخدمة، ووقت الحجز.
2. يتم تحويل الرمز إلى QR يظهر داخل تطبيق المواطن.
3. عند الوصول إلى المؤسسة، يمسح المواطن الرمز عبر جهاز التحقق.
4. يرسل الجهاز الرمز إلى الخادم الذي يتحقق من الصلاحية والمدة وعدم الاستعمال السابق.
5. إذا كان الرمز صحيحاً، يتم تأكيد الحضور وتحديث ترتيب الطابور.

الأمن السيبراني وحماية البيانات

- **رموز ديناميكية محدودة الصلاحية:** منع استعمال نفس التذكرة أكثر من مرة أو بعد انتهاء مدتها.

- **مصادقة المستخدمين:** تسجيل دخول آمن للمواطنين والموظفين والإداريين حسب الصلاحيات.
- **تشفير الاتصال:** استعمال HTTPS وشهادات أمان لحماية البيانات أثناء النقل.
- **صلاحيات متعددة المستويات:** فصل صلاحيات المواطن، الموظف، مدير الفرع، ومدير النظام.
- **سجلات تدقيق:** تسجيل عمليات الحجز، التحقق، التعديل، والاستدعاء لتسهيل المراقبة.
- **حماية البيانات الشخصية:** تخزين الحد الأدنى من بيانات المواطن واستعمالها فقط لأغراض الحجز والخدمة.

خاتمة

يهدف مشروع SmartQdz إلى إحداث تحول رقمي حقيقي في إدارة الطوابير داخل المؤسسات الجزائرية، عبر حل ذكي منخفض التكلفة يجمع بين تطبيق الهاتف، الخادم المركزي، أجهزة إنترنت الأشياء، والمصادقة الآمنة عبر رموز QR ديناميكية.

لا يقتصر المشروع على الجانب التقني فقط، بل يقدم نموذج عمل قابل للتحول إلى مؤسسة ناشئة من خلال الاشتراكات الشهرية، بيع أجهزة التحقق، عقود الصيانة، وخدمات التخصيص للمؤسسات. كما يستهدف سوقاً واضحاً يشمل البلديات، الدوائر، مراكز البريد، المستشفيات، والهيئات الخدمية التي تواجه ضغطاً يومياً في استقبال المواطنين.

تتمثل قوة SmartQdz في قابليته للتوسع وطنياً، إذ يمكن تطويره لاحقاً بإضافة التنبؤ بالازدحام عبر الذكاء الاصطناعي، إشعارات ذكية للمواطنين، إحصائيات تشغيلية متقدمة، وربط متعدد الفروع عبر منصة سحابية موحدة.

الخلاصة

يمثل SmartQdz مشروعاً قابلاً للتطبيق التجاري والتوسع، لأنه يعالج مشكلة واقعية، يقدم قيمة واضحة للمواطن والمؤسسة، يعتمد على تقنيات متاحة ومنخفضة التكلفة، ويملك نموذج إيرادات قابل للنمو.

الملاحق

الملحق 01: الميزانية التقديرية للمؤسسة الناشئة

جدول 14: الميزانية التقديرية

البيان	التكلفة التقديرية
تطوير التطبيق والمنصة	50,000 دج
الكشك وقارئ QR	75,000 دج لكل نقطة تحقق
شاشة وملحقات تركيب	30,000 دج
الاستضافة والأمان	30,000 دج كبداية تشغيلية
التسويق والتجارب	40,000 دج
التركيب والدعم الأولي	30,000 دج
المجموع التقديري	**255,000 دج**

الملحق 02: جدول حسابات النتائج المتوقع

جدول 15: حسابات النتائج المتوقعة

العنصر	مصدره في مشروع SmartQdz
رقم الأعمال	من 1,400,000 دج إلى أكثر من 8,000,000 دج حسب السيناريو
أعباء الاستغلال	الاستضافة، الدعم الفني، الصيانة، والاتصال
الأعباء الإدارية	التسيير، المحاسبة، الوثائق، وتنظيم العلاقات مع المؤسسات
النتيجة الخام	تتحسن بعد تغطية كلفة الإطلاق والأجهزة الأولى
النتيجة الصافية	ترتبط بعدد العقود السنوية وقيمة الاشتراكات والصيانة

الملحق 03: خطة الخزينة

جدول 16: خطة خزينة مختصرة

المرحلة	المداخيل المتوقعة	المصاريف المتوقعة
بداية المشروع	تمويل ذاتي أو دعم أولي	تطوير النظام وشراء معدات الاختبار
التجربة الميدانية	عقود تجريبية محدودة أو دعم مؤسساتي	تركيب الأجهزة، التنقل، والدعم الفني
الإطلاق التجاري	اشتراكات المؤسسات وبيع الأجهزة	الاستضافة، الصيانة، والتسويق
التوسع	عقود سنوية وصيانة دورية	توسيع الفريق، تحسين المنصة، وتأمين الخدمة

الملحق 04: مخطط نموذج الأعمال

جدول 17: *Canvas Model Business* لمشروع *SmartQdz*

الشركاء الرئيسيون	الأنشطة الرئيسية	القيمة المقترحة
الهيئات الحكومية والمؤسسات العمومية، موردو الأجهزة الإلكترونية، مزودو الخدمات السحابية	تطوير المنصة والتطبيق، تركيب الأجهزة بالمؤسسات، تأمين البيانات والمصادقة	إلغاء الانتظار العشوائي، حجز عن بعد مؤمن، منع التلاعب بالتذاكر عبر QR Dynamic
العلاقة مع العملاء	**شرائح العملاء**	**الموارد الرئيسية**
عقود طويلة مع المؤسسات، دعم فني متواصل، خدمة ذاتية للمواطن	البلديات والدوائر CASNOS , CNAS , TELECOM ALGERIE , POSTE ALGERIE , وقطاع الخدمات العامة	فريق التطوير، الخوادم وقواعد البيانات، أجهزة الحواسيب و الأكشاك و اجهزة العرض
قنوات الوصول	**هيكل التكاليف**	**مصادر الإيرادات**
المبيعات المباشرة للمؤسسات، متاجر التطبيقات، المعارض التكنولوجية والجامعية	الاستضافة السحابية، شراء قطع الغيار والأكشاك، رواتب فريق التطوير والدعم	كراء أجهزة المسح الذكية، اشتراكات إدارة المنصة، رسوم الصيانة الدورية، لوحة Premium

خلاصة الملحق

يوضح نموذج الأعمال أن SmartQdz مشروع B2B/B2G يستهدف المؤسسات التي تحتاج إلى تنظيم تدفق المواطنين، ويحقق إيراداته من الجمع بين الأجهزة، البرمجيات كخدمة، والدعم المستمر.