Unabridged Documentation of Company Products and Microservices Development

A PROJECT REPORT

Submitted by

Raghav Sarmukaddam

210410107020

In fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

Computer Engineering

Sardar Vallabhbhai Patel Institute of Technology, Vasad

Gujarat Technological University, Ahmedabad

May, 2025

Sardar Vallabhbhai Patel Institute of Technology

B/H S.T. Depot, Vasad, Dist. Anand, Gujarat

CERTIFICATE

This is to certify that the project report submitted along with the project entitled "**Unabridged Documentation of Company Products and Microservices Development"** has been carried out by **Raghav Sarmukaddam** under my guidance in fulfilment for the degree of Bachelor of Engineering in Computer Engineering, 8th Semester of Gujarat Technological University, Ahmadabad during the academic year 2024-25.

Prof. Keyur Upadhyay
Internal Guide

Prof. Jayna Shah
Head of Department

Completion PMMS

Completion Company

Sardar Vallabhbhai Patel Institute of Technology

B/H S.T. Depot, Vasad, Dist. Anand, Gujarat

DECLARATION

We hereby declare that the Internship / Project report submitted along with the Internship entitled Unabridged Documentation of Company Products and Microservices Development submitted in fulfilment for the degree of Bachelor of Engineering in Computer Engineering to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by me at Phonon Communication Pvt. Ltd. under the supervision of Dr. Ekata Shah and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

Name of the Student — Sign of the Student

Raghav Sarmukaddam

ACKNOWLEDGEMENT

The journey of knowledge acquisition is a continuous process, and this internship has been an invaluable chapter in my professional and personal development. I find myself fortunate and deeply indebted to several individuals who have contributed significantly to this experience.

I would like to express my heartfelt gratitude to Dr. Ekata Shah at Phonon Communications Pvt. Ltd., whose exceptional mentorship transcended typical supervision. Her consistent encouragement, positive demeanor, and genuine interest in my growth created an atmosphere where learning flourished naturally.

My experience with The Core Team has been nothing short of transformative. Their technical expertise was matched only by their willingness to collaborate and assist whenever obstacles arose. The bonds formed during discussions have left an indelible impression on my approach to work.

To my fellow interns who shared this journey—thank you for the moments of mutual support, the shared challenges, and the camaraderie that made even the most demanding days worthwhile.

I am particularly grateful to Prof. Keyur Upadhyay, my internal guide, and Prof. Jayna Shah, Head of Department, whose advice and unwavering support kept me encouraged, enabling me to maximize this learning opportunity.

ABSTRACT

The internship encompassed two primary projects: a QR Code Generator and Management Platform, and the Kairos Documentation Project. Both projects were undertaken to gain hands-on experience within an enterprise environment.

The QR Code Generator project aimed to develop a comprehensive web application for creating, customizing, and managing QR codes. Key objectives included implementing a microservices architecture for scalability, developing features for multiple QR code types, incorporating analytics capabilities for tracking scan data, and user authentication systems. The technical implementation utilized React with TypeScript for the frontend and Python (FastAPI) with MongoDB for the backend, containerized through Docker for seamless deployment.

The Kairos Documentation Project focused on establishing standardized technical documentation for a complex enterprise platform. The objectives included implementing Doxygen for automated code documentation, developing comprehensive documentation for production code components, facilitating knowledge transfer through clear architectural overviews, and centralizing information access for developers and stakeholders.

Through these projects, the internship provided valuable experience in full-stack development practices, microservices architecture, documentation standards, and collaborative software development within a professional context. Both projects contributed to a comprehensive understanding of software development lifecycles and enterprise application architecture, fulfilling the internship's objective of bridging theoretical knowledge with practical implementation in a professional environment.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1.0 COMPANY PROFILE

1.1 HISTORY

Phonon Communications Pvt. Ltd. was founded in 2006 as a proprietary firm by visionary entrepreneur Mr. Ujwal Makhija. From its inception, the company attracted prominent clients including Kingfisher, IndiGo, Sutherland, and HDFC, establishing itself as a trusted partner in enterprise communication. In 2014, the company took a significant step forward by securing the trademark for "Click to Call," a service that emphasized its commitment to seamless customer interaction technologies. The following year, in 2015, Phonon evolved into a private limited company, marking its transition into a more structured and scalable organization.

Building on this momentum, the company obtained an Audiotax licence in 2016 for the Vadodara service area, expanding its operational scope. In 2018, Phonon reinforced its position at the forefront of AI-driven communication solutions through the strategic acquisition of iDeliver, a company specializing in AI-based bots. This acquisition laid the groundwork for the launch of Phonon Central in 2019—a comprehensive self-service IVR and chat platform that streamlined customer engagement through automation.

Phonon's innovative trajectory continued in 2021 with the formation of a channel partnership with TrueCaller for Business, further enhancing the authenticity and reach of customer communications. Most recently, in 2024, the company achieved a major regulatory milestone by acquiring a Unified Audio Tax Licence for PAN India, allowing it to extend its services on a national scale.

Over the years, Phonon Communications has exemplified a unique blend of strategic vision, technological innovation, and customer focus—growing from a bootstrapped venture into a nationally recognized name in enterprise communication solutions.

1.2 PRODUCTS AND SCOPE OF WORK

Phonon Communications Pvt. Ltd. specializes in cloud telephony solutions, offering a suite of products and services designed to automate and enhance enterprise customer interactions. Their offerings cater to various industries, including banking, financial services, insurance (BFSI), travel, aviation, e-commerce, logistics, and automotive sectors.

1.2.1 Key Products and Services:

- Intelligent IVR (Interactive Voice Response): Phonon's IVR solutions provide prompt, intuitive, and personalized responses to customer queries. Features include speech-to-text capabilities, natural language processing, API integration, and comprehensive reporting and analytics.
- Contact Center Automation Suite: This AI-powered suite offers built-in call and chat distribution, supports multiple campaigns and dialing modes, and provides detailed reports to improve customer experience.
- Click-to-Call™ & Instant Connect: Enables instant voice communication between customers and enterprises directly from digital platforms, facilitating real-time engagement and support.
- Proactive Outbound Engagement: Automated outbound solutions that send personalized, omnichannel notifications over WhatsApp, SMS, email, and voice calls. This feature enhances customer experience by automating call blasts, reminders, and follow-ups, with live agent connect when needed.
- Visual IVR: A text-based IVR interface that allows customers to interact through a chat-like experience, reducing the load on telephony systems. Use cases include lead qualification, surveys, scheduling, upselling, customer onboarding, and service requests.

1.2.2 Industry Applications:

- Travel & Aviation: Solutions for rescheduling and cancellation notifications, intelligent customer support, reminders, and upsell/cross-sell bots to enhance passenger experience and increase revenue per customer.
- Banking & Financial Services: Services include digital and offline customer acquisition, relationship manager instant connect, eligibility bots, predictive customer interactions, and proactive fraud risk notifications.
- E-commerce & Logistics: Integrated solutions for retargeting, number masking, order status updates, FAQ bots, predictive customer support, and multi-factor authentication to serve consumers efficiently.

1.3 COMPANY PHILOSOPHY

Phonon specialises in providing omni-channel enterprise grade customer communication automation solutions.

At Phonon, they empower enterprises to deliver seamless, automated customer interactions across both traditional and new media channels. By leveraging Voice, SMS, Email, and modern platforms like Facebook, Twitter, chatbots, and push notifications, we help businesses achieve high-quality customer engagement with minimal costs.

Company's unified communications solutions offer real-time, scalable automation, transforming customer interactions. Phonon's intuitive applications make communication faster, more efficient, and engaging, delivering a "WOW" experience for users.

1.3.1 Vision and Mission

Vision: To provide the right information to end-users, at their convenience, with consistent quality across Voice, SMS, Email, Social Media, and Push Notifications.

Mission**:** We will provide a scalable, secure, easy-to-use, and feature-rich single-stop platform that integrates with the best global AI and ML service frameworks and across channels (voice and text-based) to automate customer-business interactions.

1.4 IMPACT AND MARKET PRESENCE

Since its inception, Phonon Communications Pvt. Ltd. has played a pioneering role in redefining customer interaction in India's enterprise communication landscape. With clients across sectors such as BFSI, aviation, logistics, and e-commerce, the company has enabled high-efficiency, scalable communication channels for some of the country's leading enterprises.

Phonon's solutions are trusted by top-tier brands including IndiGo, HDFC Bank, ICICI Lombard, TATA AIG, Vistara, and PayU, reflecting its credibility and excellence in the domain. The company's innovations—such as Click-to-Call™, proactive outbound voice automation, and AI-driven bots—have contributed to:

- Reduction in customer support turnaround time
- Enhanced lead conversion through intelligent routing
- Operational cost savings via automation of repetitive queries

The acquisition of iDelivr and partnerships with global platforms like Truecaller for Business have further positioned Phonon as a leader in AI-powered communication solutions. Its influence is evident in how modern customer experiences are being shaped through real-time, intelligent, and context-aware engagement.

1.5 DEPARTMENT OVERVIEW AND WORKFLOW

Phonon Communications operates with a modular and service-oriented organizational structure that enables agile development and efficient client delivery. The company is structured into key functional departments, each playing a crucial role in delivering intelligent, omni-channel customer engagement solutions.

1.5.1 Departmental Functions and Responsibilities

Engineering Department

Responsible for designing and maintaining scalable, high-availability communication infrastructure.

- **Site Reliability Engineers (SREs):** Focus on infrastructure stability, uptime, observability, automation, and continuous integration.
- **Software Development Engineers (SDEs):** Build backend services (e.g., Spring Boot), frontend applications (e.g., AngularJS), and deployment workflows.
- **Application Support Engineers:** Oversee real-time issue resolution, log management , and performance optimization.

The team also manages databases (PostgreSQL, MongoDB, MySQL), message brokers (Kafka, ActiveMQ Artemis), servers (Tomcat, Apache), and tools like Log4j2, Logstash, Redis, Supervisor.d, Node.js, and PM2.

Product Department

Leads platform innovation and user-centric design. Product Managers and Designers work with tools like Jira, Figma, Confluence, Postman, Swagger, and Lucidchart to plan and deliver user journeys, features, and documentation. The department:

- Drives development of platforms like Phonon Central, ALOHA App, and Visual IVR.

- Converts client feedback into functional specifications and interfaces with Engineering and Sales teams.

Sales and Revenue Department

Dedicated to business growth and ensuring successful client onboarding. This team leverages tools such as Salesforce, Google Sheets, and BI dashboards to:

- Conduct client scoping and feature demonstrations.
- Gather requirements and ensure successful solution delivery.
- Monitor KPIs and identify upsell opportunities.

Customer Support Department

Focused on providing 24/7 support and maintaining SLAs. The team uses Freshdesk, internal dashboards, Grafana, and logging systems to:

- Track support tickets and manage escalations.
- Work closely with QA and Application Support for issue resolution.
- Gather feedback to inform product and engineering teams.

Commercial and Accounts Department

Handles licensing, procurement, vendor communications, and compliance management. The team monitors software versions and EoL tracking (as seen in inventory lists), using tools such as ERP dashboards, Java/MySQL inventory trackers, and Excel to:

- Oversee patch updates and vendor liaison.
- Maintain software and service contracts.

Human Resources (HR)

Supports hiring, onboarding, and workforce engagement. HR manages internal systems such as GreytHR, candidate tracking platforms, and Google Workspace tools to:

- Coordinate hiring with department heads.
- Maintain employee data and training roadmaps.
- Facilitate performance reviews and policy rollouts.

2.0 INTRODUCTION TO INTERNSHIP PROJECT(S)

2.1 SUMMARY

This section documents two significant technical initiatives undertaken during my internship tenure at [Company Name]:

Project 1: Development of a comprehensive Trackable Dynamic QR Code Generator Platform leveraging a microservices architecture. The solution implements a FastAPI backend coupled with a React and TypeScript frontend. This enterprise-grade platform facilitates the generation of QR codes with dynamic redirect capabilities, robust user authentication, detailed analytics tracking, and a highly scalable architecture designed for future expansion.

Project 2: Creation of extensive technical documentation for the Kairos Enterprise Communication Platform, a sophisticated product comprised of numerous interconnected microservices. This documentation effort involved thorough source code analysis, deep functional understanding of each service component, and the development of detailed documentation assets using both Confluence and Doxygen-generated technical specifications.

Both projects were strategically aligned with organizational objectives to enhance internal technical transparency, streamline developer onboarding processes, and improve the scalability and maintainability of customer-facing product offerings.

2.2 PURPOSE

For Project 1, the primary purpose was to engineer a lightweight yet robust QR code generation platform that would support dynamic URL redirection and comprehensive usage tracking. The implementation adhered to production-grade engineering principles including clear service separation, containerized deployment, and integrated observability features to ensure operational excellence.

For Project 2, the purpose centered on establishing structured and accessible technical documentation for the Kairos platform ecosystem. This initiative aimed to improve cross-functional visibility, accelerate developer onboarding, and enhance long-term maintainability across the diverse array of microservices that comprise the platform.

2.3 OBJECTIVE

- Design and implement a fully functional, modular Dynamic QR Code Generator platform featuring strict API separation through a gateway-centric architecture, comprehensive monitoring capabilities with detailed metrics collection, persistent data storage utilizing MongoDB's document model for flexibility and MinIO object storage for binary assets, all secured with JWT-based authentication and fine-grained authorization controls. The system should achieve sub-100ms response times for QR generation while maintaining scalability under varying load conditions.
- Produce thorough documentation for all key microservices within the Kairos platform, detailing component architectures, operational workflows, system dependencies, and library integrations. Additionally, establish standardized documentation practices through template creation and tool implementation to ensure consistency in future documentation efforts.

2.4 SCOPE AND BOUNDARIES

The QR Code Platform project encompassed the development of a complete microservices-based solution with well-defined capabilities and carefully considered scope boundaries to ensure successful delivery within the internship timeframe.

2.4.1 QR Code Platform Technical Scope

The platform architecture implemented six interconnected microservices as evidenced in the `microservices` directory structure, each serving a specific domain function:

The authentication framework provides comprehensive user management through the `user-service`, implementing JWT-based authentication with bcrypt password hashing and role-based access controls. The system supports multiple authentication flows including username/password and potential OAuth integration paths as configured in the environment variables. At the core of the platform, the `qr-service` delivers sophisticated QR code generation capabilities with configurable design parameters and dynamic redirect functionality. Generated QR codes are securely stored in MinIO object storage with appropriate metadata, enabling efficient management and retrieval. The service implements template-based generation for maintaining brand consistency across generated codes.

The analytics capabilities are handled by the dedicated `analytics-service`, which captures comprehensive visitor interaction data including IP addresses, geolocation information, access timestamps, device characteristics, and click-through patterns. This service implements an event-based architecture for high-throughput data collection with MongoDB aggregation pipelines for efficient data analysis. For specialized use cases, the `vcard-service` extends platform functionality with business card QR generation supporting international contact standards and custom field definitions. The `redirect-service` handles the critical user-facing functionality of processing QR code scans with ultra-low latency redirects while simultaneously triggering analytics events. The entire platform is orchestrated through containerization with Docker Compose as defined in the `docker-compose.yml` configuration, supporting consistent development and deployment environments. The `api-gateway` service handles cross-cutting concerns including request routing, authentication validation, and CORS policy enforcement, with Caddy providing reverse proxy capabilities and TLS termination.

The frontend implementation, built with React 18 and TypeScript, provides a responsive user interface with Tailwind CSS for styling and Radix UI components for accessibility. The application features a comprehensive dashboard for QR code management, basic analytics visualization, and user administration functions.

2.4.2 Kairos Documentation Project Scope

The Kairos documentation initiative involved the comprehensive analysis of approximately 19 distinct microservices within the enterprise platform ecosystem. This work delivered detailed functional documentation explaining individual service purposes and capabilities, architectural documentation outlining system interactions and dependencies, Doxygen implementation for automated code documentation generation, and a structured Confluence knowledge base with standardized templates for ongoing documentation.

2.4.3 Intentional Scope Limitations

To ensure project success within the internship timeframe, several capabilities were intentionally deferred to future development phases. The frontend implementation focused on functional requirements rather than extensive UI/UX optimizations or design

refinements. While the system implements essential security best practices, comprehensive security audits and formal compliance certification were designated for future phases.

Similarly, while the analytics service captures and stores comprehensive interaction data, advanced visualization dashboards and business intelligence features were limited to basic reporting to maintain project focus. Performance testing was conducted at a development scale rather than production-level load testing, and the project scope specifically excluded refactoring of legacy codebases within the existing Kairos microservices ecosystem.

These scope boundaries were established through consultation with stakeholders to ensure that project deliverables aligned with available resources while still providing robust, production-quality functionality for the core use cases.

2.5 TECHNOLOGY AND LITERATURE REVIEW

Prior to implementation, I conducted extensive research on modern FastAPI design patterns, evaluated REST vs GraphQL approaches for API design, investigated microservice orchestration methodologies, and analyzed documentation strategies optimized for distributed system architectures. This research phase ensured alignment with industry best practices and established a solid foundation for both projects.

Table 2.1: Technology Domain and Tools selected

Technology Domain	**Selected Tools & Frameworks**
Backend API Development	Python (3.11.9), Pydantic & FastAPI
Frontend Development	React.js 18, TypeScript, Vite, Tailwind CSS, Radix UI
Technical Documentation	Confluence Enterprise, Doxygen, Markdown
Containerization & Deployment	Docker, Docker Compose, Docker Desktop
Data Storage & Caching	MongoDB, Minio Object Storage
Web Server & Proxy	Uvicorn (ASGI), Caddy Proxy
Version Control	Git, AWS CodeCommit

- The **qr_code_project (frontend/microservices)** uses FastAPI, Python, Pydantic, React, TypeScript, Vite, Tailwind CSS, Radix UI, MongoDB, Minio, Docker, Docker Compose, and Git.
- The **kairos** project uses Java, Spring Boot, Hibernate, MySQL, Redis, Maven, Git and includes several UI components (web-based).

2.6 PLANNING

2.6.1 Development Approach and Justification

Microservices Architecture was selected as the foundational architectural pattern for the QR Code Platform after careful evaluation of various architectural approaches. This decision was driven by the inherent benefits of service isolation, domain-specific scaling capabilities, and enhanced fault tolerance essential for a production-grade application. The architecture enables independent development cycles for each service, allowing for targeted optimizations and technology selection based on specific domain requirements rather than forcing a monolithic one-size-fits-all approach.

The system was decomposed into six distinct services, each with a clearly defined responsibility domain and carefully designed boundaries. This decomposition follows Domain-Driven Design principles, ensuring that each service encapsulates a specific business capability with minimal cross-service dependencies. The inter-service communication was implemented using both synchronous HTTP calls for immediate consistency requirements and asynchronous patterns for eventual consistency scenarios, striking an optimal balance between system responsiveness and data integrity.

API Gateway Service serves as the central entry point for all client interactions, implementing a unified interface that abstracts the underlying microservice complexity from frontend clients. This architectural component handles numerous cross-cutting concerns that would otherwise be duplicated across individual services. Authentication validation occurs at this layer, verifying JWT tokens before requests are forwarded to internal services, thus establishing a consistent security perimeter. Request routing logic directs traffic to appropriate backend services based on URL patterns and HTTP methods, while also implementing circuit breaker patterns to prevent cascading failures when downstream services experience issues.

Additionally, the gateway enforces CORS policies to secure cross-origin requests, particularly important for web clients, and implements sophisticated rate limiting to protect against denial-of-service attempts and abusive usage patterns. By adopting the Backend for Frontend (BFF) pattern, the gateway optimizes data transfer between frontend and backend systems, aggregating multiple service calls when necessary and transforming data structures to match frontend consumption patterns without requiring multiple round trips.

User Service forms the foundation of the platform's identity and access management infrastructure. It manages the complete user lifecycle from initial registration through authentication, profile management, and account deactivation. The service implements sophisticated password security using bcrypt hashing with appropriate work factors that balance security with performance, protecting user credentials even in the event of a database compromise. JWT token generation follows industry best practices with appropriate expiration policies, token refresh mechanisms, and payload minimization to reduce transmission overhead. The service implements comprehensive role-based access control (RBAC) that goes beyond simple role assignments to include fine-grained permission management, allowing for precise control over system capabilities. This granular approach to authorization ensures that users can only access functionality appropriate to their assigned roles, maintaining the principle of least privilege throughout the system. The service also provides administrative capabilities for user management, including account suspension, role assignment, and audit logging of security-relevant actions.

QR Service represents the core technical capability of the platform, providing sophisticated QR code generation functionality with numerous customization options. The implementation leverages optimized algorithms for QR code generation with adaptive error correction levels that balance code density with scan reliability based on content size and expected scanning conditions. Custom styling options include color selection, background transparency, embedded logos, and design templates that maintain scan compatibility while enhancing brand presence. Integration with MinIO for reliable asset storage ensures that generated QR codes are persistently stored with appropriate metadata, enabling retrieval, regeneration, and management over time. The service implements efficient caching strategies at multiple levels, including in-memory caches for frequently accessed codes and database-level caching for generation parameters, significantly reducing computational overhead for repeated code generation. Advanced features include expiration dates for

temporary codes, access tracking for individual codes, and batch generation capabilities for marketing campaigns or inventory management scenarios.

VCard Service extends the platform's capabilities with specialized functionality for business contact information, addressing the common use case of digital business cards. The service supports international contact standards including vCard 3.0 and 4.0 formats, ensuring compatibility with a wide range of contact management systems and mobile devices. Beyond basic contact fields, the implementation supports custom field definitions, localization options for multilingual business cards, and social media profile integration. A distinctive feature is support for profile image embedding, allowing business cards to include professional photographs or company logos while maintaining QR code scanability. The service implements comprehensive JSON schema validation against the vCard specification to ensure that all generated codes conform to established standards, preventing interoperability issues with contact management systems. Additional capabilities include analytics integration to track business card engagement, template management for corporate branding consistency, and automatic updates that propagate changes to the contact information across all generated QR codes.

Analytics Service provides comprehensive visibility into QR code usage patterns through a sophisticated event collection and processing pipeline. The service implements an event-driven architecture that captures scan events in real-time while maintaining system performance even under high traffic conditions. Data enrichment processes augment raw scan events with geospatial information, device characteristics, and temporal contexts, creating rich datasets for multidimensional analysis. Time-series data management techniques optimize storage and retrieval of historical scan data, enabling efficient trending analysis and pattern recognition across various timeframes. The service provides specialized aggregation endpoints that support dashboard visualizations with real-time updates through efficient database projection techniques and materialized views for common query patterns. Advanced analytics capabilities include funnel analysis for multi-step campaigns, cohort analysis for user segmentation, and anomaly detection for identifying unusual scanning patterns that might indicate fraudulent usage or technical issues with particular codes.

Redirect Service operates on the critical path for every QR code scan, requiring exceptional performance characteristics to ensure positive user experiences. The service achieves ultra-low latency redirects (typically sub-50ms) through aggressive caching, optimized database queries, and a streamlined request processing pipeline specifically designed for high-throughput scenarios. Real-time analytics event generation occurs asynchronously after the redirect response has been initiated, ensuring that tracking activities never impact the user-facing performance. Smart traffic management capabilities enable sophisticated use cases including A/B testing of destination URLs, time-based routing to different destinations depending on scan time, and geographical routing based on the user's location. The service implements custom redirect rules through a flexible rule engine that evaluates conditions against scan metadata to determine the appropriate destination. Additionally, security features protect against malicious redirects through URL validation, rate limiting per source IP, and configurable blacklists for known malicious destinations.

The selection of **FastAPI Framework** for the backend implementation followed a thorough comparative analysis against alternatives like Django REST Framework, Flask, and Express.js. FastAPI emerged as the optimal choice due to its exceptional performance characteristics, achieving request handling metrics comparable to Node.js while maintaining the expressive power and developer ergonomics of Python. The framework's intuitive developer experience significantly accelerated development velocity, particularly through its innovative dependency injection system and automatic request validation. The built-in OpenAPI documentation generation creates comprehensive, interactive API documentation that remains synchronized with the codebase, eliminating documentation drift and providing clear visibility into API capabilities for frontend developers. FastAPI's native support for asynchronous operations proved invaluable for I/O-bound operations like database queries and inter-service communication, allowing the system to maintain high throughput even under substantial load. The implementation leverages Pydantic extensively for data validation and serialization, ensuring type safety throughout the request/response lifecycle while providing clear error messages when validation fails. Dependency injection patterns facilitate clean separation of business logic from API endpoints, improving code organization and enabling comprehensive unit testing through straightforward mock injection.

For frontend development, **React with TypeScript** was selected to create a robust, maintainable user interface with strong typing guarantees. TypeScript's static type checking catches common errors during development rather than at runtime, substantially reducing debugging time and improving code quality. This approach was enhanced with Tailwind CSS for utility-first styling, providing a consistent design language while enabling rapid UI iteration without context switching between HTML and CSS files. The implementation uses Radix UI primitives as the foundation for accessible component design, ensuring that the application meets WCAG 2.1 accessibility standards while providing a polished user experience. Component composition patterns form the architectural backbone of the frontend, with a library of atomic, reusable components that combine to create complex interfaces while maintaining consistency and reducing duplication. State management leverages React Context API with custom hooks for specific domains (authentication, notifications, theme settings), striking an effective balance between Redux's centralized approach and component-local state. This strategy provides efficient state management without unnecessary complexity, allowing developers to reason about data flow more intuitively. The frontend implementation also features responsive designs that adapt gracefully across device sizes, from mobile phones to large desktop displays, ensuring a consistent user experience regardless of access method.

The data persistence layer employs **MongoDB & MinIO** as complementary technologies addressing different storage requirements. MongoDB was selected for its flexible schema design, which accommodates the varied data structures across different services while allowing for schema evolution as requirements change. The document model aligns naturally with JSON-based API payloads, reducing the object-relational impedance mismatch common in traditional SQL databases. The QR platform implements appropriate indexing strategies for query optimization, including compound indexes for frequently joined fields and text indexes for search functionality

MongoDB's aggregation pipelines power complex analytics operations, enabling sophisticated data transformations and calculations without requiring data transfer to application servers. For binary asset storage, MinIO provides an S3-compatible object storage system that handles QR code images, user uploads, and template files with high reliability and throughput. The implementation configures bucket policies to enforce access controls, uses presigned URLs for secure, time-limited asset access, and implements versioning for critical assets to protect against accidental data loss.

Together, these technologies create a flexible, scalable data layer that accommodates diverse storage requirements while maintaining performance at scale.

Containerization with Docker Compose forms the foundation of the deployment strategy, creating a consistent, reproducible environment across development, testing, and production stages. The Docker Compose configuration orchestrates the entire application stack, defining service dependencies, network configurations, and volume mappings in a declarative manner that serves as executable documentation of the system architecture. Each microservice is containerized with optimized multi-stage builds that separate dependency installation from application code, resulting in significantly reduced image sizes while ensuring that production containers contain only the necessary runtime components. The containerization strategy implements proper resource limits to prevent resource starvation between services, includes comprehensive health checks that enable orchestration systems to detect and recover from service failures, and configures appropriate logging drivers to centralize log collection. This approach not only simplifies development onboarding by providing a consistent environment but also enables straightforward horizontal scaling by deploying additional container instances for high-traffic services. The Docker-based architecture creates a clear migration path to orchestration systems like Kubernetes when more sophisticated scaling and management capabilities become necessary.

Caddy was selected as the **reverse proxy** after evaluating alternatives including Nginx and Traefik. Caddy's modern architecture offers automatic HTTPS certificate management through Let's Encrypt, eliminating the manual certificate renewal processes typical of traditional web servers. The implementation configures Caddy to handle TLS termination at the edge, removing encryption overhead from application servers while maintaining end-to-end security. Caddy's efficient traffic routing directs requests to appropriate backend services based on URL patterns while handling failure scenarios gracefully through configurable timeouts and retry policies. The proxy layer also implements security headers, CORS policies, and response compression to enhance both security and performance. Caddy's simple, human-readable configuration format significantly reduces operational complexity compared to traditional web servers, rapid adjustments to routing rules or security policies without extensive specialized knowledge.

The **Documentation Toolchain** implements a dual approach to address different documentation needs across the platform. Doxygen generates comprehensive API-level documentation directly from source code annotations, creating a detailed technical reference that remains synchronized with the codebase. This automated approach ensures that documentation coverage extends to internal components and helper functions that might otherwise be overlooked in manually maintained documentation. The Doxygen configuration generates both HTML and PDF outputs, catering to different consumption preferences while maintaining consistent content. For higher-level conceptual documentation and operational guides, Confluence provides a structured content management system optimized for human readability and team collaboration. The Confluence knowledge base implements standardized templates for different documentation types (architecture overviews, operational procedures, troubleshooting guides), ensuring consistent formatting and content coverage. Cross-linking between Confluence pages and Doxygen-generated API documentation creates a comprehensive documentation ecosystem that supports both high-level understanding and detailed technical reference, addressing the needs of different stakeholders from operations teams to developers.

2.6.2 Effort, Time & Cost Estimation

Table 1.2: Effort estimation and deliverables

Milestone	**Estimated Story Points**[1]	**Approximate Duration**	**Key Deliverables**
Architecture Design & Setup	5 SP	1 week	System architecture diagram, Docker network configuration, service boundaries definition, API contract specifications

[1] Story point estimation based on complexity where 1 SP ≈ 4-5 hours of focused development effort. The entire 16-week internship period was efficiently allocated across both the QR platform implementation (51 story points, approximately 255 development hours) and the Kairos documentation project (22 story points, approximately 110 hours), with some parallel work streams to optimize productivity.

API Gateway & Authentication	8 SP	2 weeks	JWT implementation, role-based authorization model, service routing logic, CORS policy enforcement, rate limiting rule
Core Services Implementation	13 SP	3 weeks	User service with bcrypt password hashing, QR code generation with MinIO integration, VCard service with schema validation
Analytics & Redirect Services	8 SP	2 weeks	Event-based tracking system, geolocation enrichment, aggregation pipelines, high-performance redirect handler with A/B testing capabilities
Frontend Development	12 SP	3 weeks	React/TypeScript dashboard, Tailwind CSS responsive layouts, Radix UI component system, QR management interface, analytics visualization with Recharts
Testing & Deployment Pipeline	5 SP	1 week	Unit/integration test suites, Docker Compose production configuration, Caddy reverse proxy setup, CI/CD workflows
Kairos Documentation Research	10 SP	2 weeks	Service inventory, architectural diagrams, dependency mapping
Kairos Documentation Creation	12 SP	3 weeks	Confluence spaces, API references, developer guides

2.6.3 Dependencies

Technical Stack Dependencies:

- **Backend Framework and Tools:**
  - FastAPI ecosystem (v0.104.1) for API development with asynchronous support
  - Pydantic (v2.5.2) for data validation, serialization, and schema definition
  - Uvicorn (v0.24.0) as the ASGI server for production deployment
  - Python-jose (v3.3.0) for JWT token generation and validation
  - Python-multipart for handling form data and file uploads
  - Httpx for asynchronous HTTP client capabilities in inter-service communication
- **Database and Storage:**
  - MongoDB as the primary database with PyMongo (v4.5.0) and Motor (v3.3.1) for asynchronous operations
  - MinIO (v7.2.0) for S3-compatible object storage of QR code images and assets
  - Python-magic (v0.4.27) for file type detection and validation
- **Containerization and Deployment:**
  - Docker for service containerization with multi-stage builds
  - Docker Compose for local development and staging environment orchestration
  - Caddy (latest) as the reverse proxy with automatic HTTPS support
- **Frontend Technologies:**
  - React (v18.2.0) as the UI framework with functional components and hooks
  - TypeScript (v5.3.2) for type-safe development
  - Vite (v6.2.3) for fast builds and development server
  - Tailwind CSS (v3.3.5) for utility-first styling
  - Radix UI component primitives for accessible UI elements
  - Recharts for data visualization of analytics metrics
- **Security and Authentication:**
  - Passlib with bcrypt (v1.7.4) for secure password hashing
  - JWT-based authentication with token rotation and expiration
  - Environment-based configuration management with python-dotenv

External API Dependencies:

- No external third-party APIs were integrated into the QR platform implementation, ensuring:
  - Complete system autonomy with no external rate limiting concerns
  - Elimination of external service availability dependencies

Development Workflow Dependencies:

- Git-based version control with feature branch workflow
- Code quality checks including linting and type checking
- Documentation updates as part of the development cycle

This comprehensive dependency mapping ensured that all required components were identified early in the development process, preventing unexpected blockers and enabling effective resource planning throughout the project lifecycle.

3.0 QR PLATFORM ANALYSIS

This section delves into the technical aspects of the QR Code Platform developed. It covers the system's architecture, key features, implementation details, challenges encountered, and potential future enhancements.

3.1 SYSTEM ARCHITECTURE

The platform utilizes a modern microservices architecture for the backend and a reactive frontend, ensuring scalability, maintainability, and separation of concerns.

3.1.1 System Architecture Flow Analysis

The architectural diagram presented illustrates the structural organization and data flow pathways of the QR code management platform. This architecture employs a multi-layered approach with clear separation of concerns, conforming to established distributed systems design principles.

Client Interaction Tier

The uppermost tier represents the client-facing components through which end users interface with the system. The **User Device/Browser** establishes the initial connection point, communicating with the **User Interface** component which facilitates user-system interaction.

Intermediate Processing Tier

The middle section comprises two critical components:

- The **Presentation Layer** implements a React Single Page Application (SPA) architecture, employing the Model-View-Controller pattern to separate presentation logic from business logic. This layer utilizes modern state management techniques to maintain application coherence.
- The **API Gateway** serves as a centralized request broker, implementing authentication verification, request routing, and cross-origin resource sharing (CORS) security protocols. This component follows the API Gateway pattern, providing a unified entry point for heterogeneous backend services.

Service Orchestration Tier

The foundational tier encompasses the core business logic and data management components:

1. **Microservice Ecosystem**: A collection of domain-specific FastAPI services, each with discrete responsibilities:
    - **User Service**: Authentication and user identity management.
    - **VCard Service**: Digital business card data orchestration.
    - **QR Service**: QR code generation and configuration management.
    - **Analytics Service**: Telemetry collection and statistical processing.
    - **Redirect Service**: URL resolution and redirection handling.
2. **Persistent Storage Systems**:
    - **Redis Cache**: In-memory data structure store implementing the cache-aside pattern.
    - **MinIO Object Storage**: S3-compatible binary large object (BLOB) repository.
    - **MongoDB**: Document-oriented database for structured data persistence.

Intercomponent Communication

The directional vectors between components denote request/response pathways and data transmission protocols. These communication patterns follow established microservice communication principles, including:

- Synchronous HTTP/REST communications for user-initiated requests.
- Service-to-service communications for backend operations.
- Persistence operations between services and their respective data stores.

This architecture adheres to contemporary software engineering principles, including high cohesion within services, loose coupling between services, and the implementation of established design patterns. The resultant system demonstrates characteristics conducive to horizontal scalability, fault isolation, and independent component evolution.

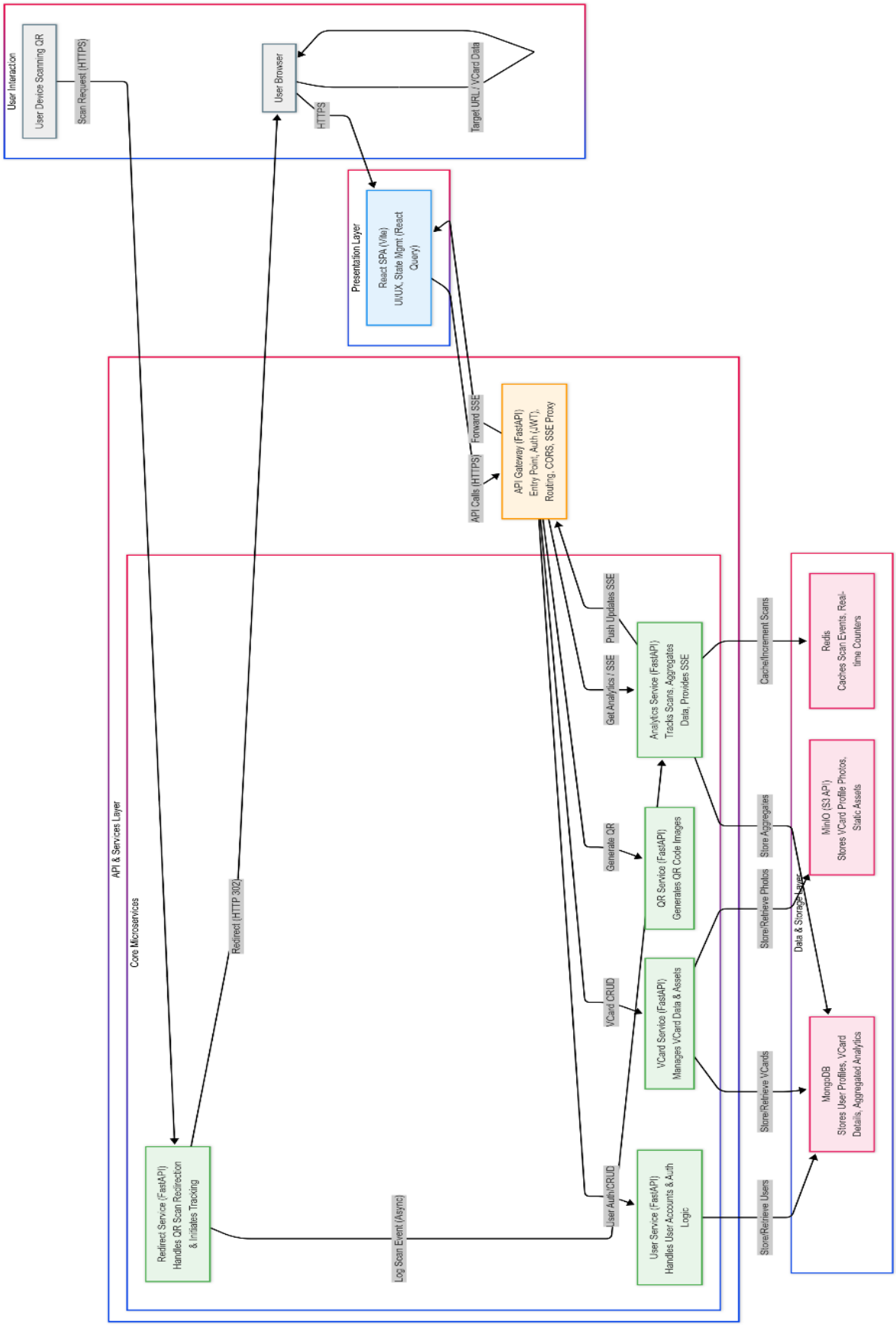

Figure 3.1: System Architecture

3.1.2 Frontend Architecture

The frontend is developed using React with TypeScript, built with Vite for an optimized development experience. It follows a component-based architecture, leveraging standard React patterns for state management, routing, and data fetching.

The diagram illustrates the core components of the React frontend. User interactions trigger component rendering and state updates managed by libraries like React Context API. React Router handles navigation between different views (pages). An API client (like Axios or Fetch) communicates with the backend API Gateway to fetch and send data. TypeScript ensures type safety throughout the codebase. Vite provides fast development builds and optimized production bundles. Tailwind CSS is used for utility-first styling.

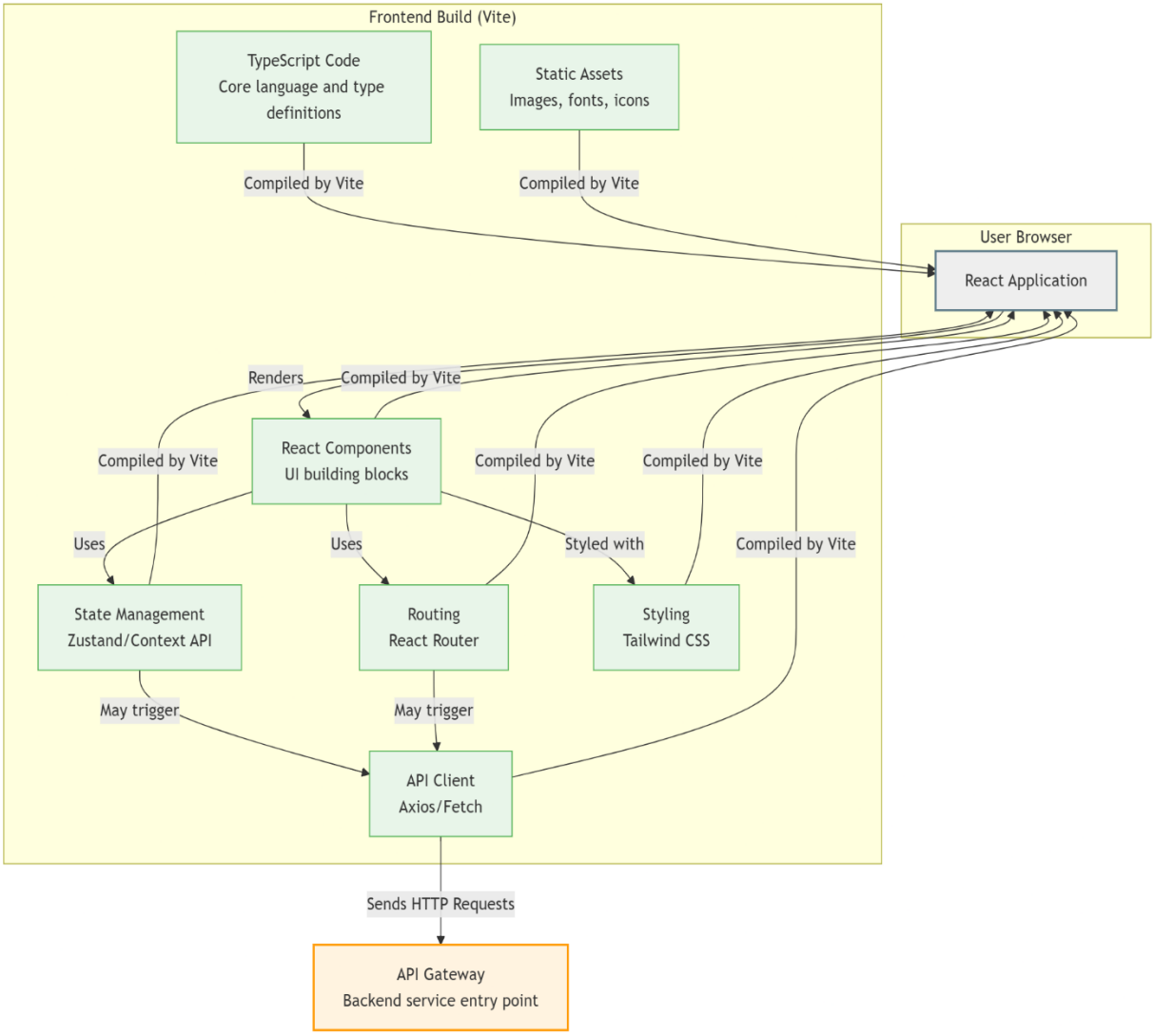

Figure 3.2: Frontend Architecture

3.1.3 Backend Architecture

The backend is implemented as a set of containerized microservices using FastAPI (Python), communicating via REST APIs through an API Gateway. This design promotes modularity and independent scaling of services. The following diagram shows the microservices architecture. A Reverse Proxy (Caddy) handles incoming HTTPS traffic and forwards it to the API Gateway. The API Gateway is the single entry point for all client requests, responsible for routing, authentication, and potentially rate limiting. It directs traffic to the appropriate backend microservice (User, QR Code, VCard, Redirect, Analytics, Auth). Each service manages its specific domain logic and interacts with shared infrastructure components like MongoDB (database), Redis (caching, potentially real-time events), and MinIO (object storage for QR codes, logos, etc.). Asynchronous communication between services (e.g., for triggering analytics on redirects) could optionally be handled via a message queue. All services are containerized (likely using Docker) and orchestrated (e.g., via docker-compose).

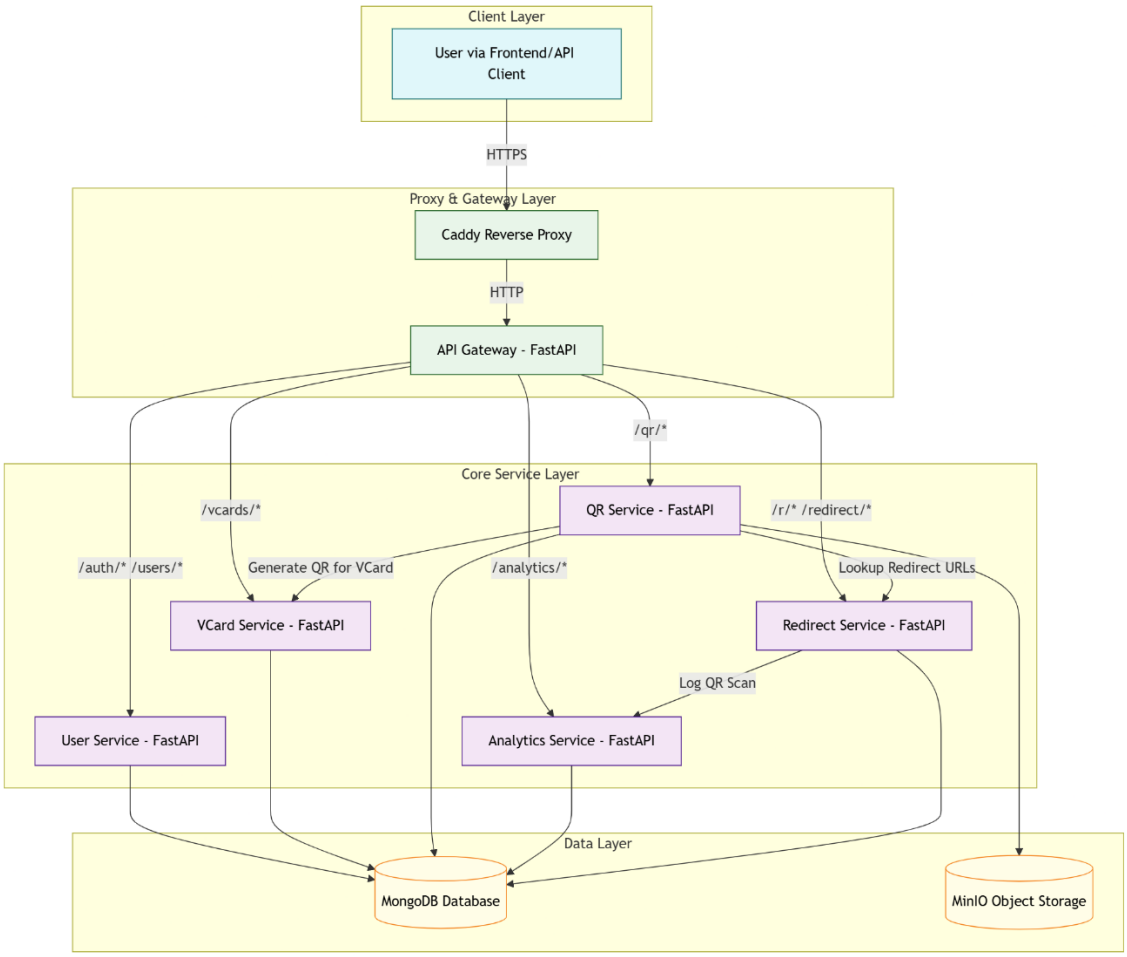

Figure 3.3: Backend Architecture

3.1.4 Database Design

MongoDB, a NoSQL document database, was chosen for its flexibility and scalability, aligning well with the varied data structures required by the platform. Key collections include:

users: Stores user account information, including credentials (hashed passwords), profile details, API keys, and references to their created QR codes or VCards. Example Fields: *_id*, *username*, *email*, *hashed_password*, *api_key*, *created_at*, *updated_at*.

qrcodes: Contains details for each generated QR code. This includes the type (URL, vCard, text, etc.), the target data, configuration parameters (color, logo image reference stored in MinIO), associated user ID, a unique short identifier for redirection, and tracking metadata. Example Fields: *_id*, *user_id*, *short_code*, *type*, *data* (can be a string URL, ObjectId reference to vcards, etc.), *config* (object with color, logo\url), *scan_count*, *is_active*, *created_at*, *updated_at*.

vcards: Stores structured contact information used for vCard-type QR codes, linked from the qrcodes collection. Example Fields: *_id*, *user_id*, *first_name*, *last_name*, *organization*, *title*, *phone*, *email*, *address*, *created_at*.

analytics_events: Records scan events, capturing details like the associated QR code ID, timestamp, user agent, IP address (potentially anonymized), and geographical location (if available/consented). This collection can grow large and might benefit from specific indexing strategies or time-series optimizations if MongoDB version supports it. Example Fields: *_id*, *qr_code_id*, *timestamp*, *ip_address*, *user_agent*, *referer*, *location* (object).

Relationships are primarily managed through embedded references (*ObjectIds*) linking documents across collections (e.g., *user_id* in qrcodes referencing users). Appropriate indexing (e.g., on *short_code*, *user_id*, *timestamp*) is crucial for query performance.

3.2 KEY FEATURES AND FUNCTIONALITY

The QR Code project provides a comprehensive platform for the creation, management, and analysis of QR codes, primarily focusing on digital business cards (vCards) but architected for extensibility. The system exposes its capabilities through a modern web-based user interface and a robust backend API.

The core functionalities include:

1. User Authentication and Management: The system incorporates a dedicated user-service responsible for secure user registration, login, and account management. Authentication is managed via JSON Web Tokens (JWT), ensuring secure access to user-specific resources and functionalities across the platform.
2. vCard Creation and Management: Users can design and manage digital business cards through the vcard-service. This includes inputting personal and professional details, potentially selecting from predefined templates, and updating information as needed. The system likely stores associated assets (e.g., profile pictures, logos) using Minio object storage.
3. QR Code Generation: The qr-service dynamically generates unique QR codes associated with the created vCards or other linkable resources. This service likely leverages standard QR code generation libraries (e.g., qrcode, Pillow for potential customization) and ensures each code is uniquely identifiable for tracking purposes.
4. Dynamic Redirection: QR codes generated by the system are not static. The redirect-service handles the resolution of QR code scans, directing the end-user to the appropriate vCard or linked resource. This dynamic nature allows for updating the target destination without altering the printed QR code itself.
5. Scan Analytics and Tracking: A key feature is the ability to track user engagement with the QR codes. The analytics-service captures scan events forwarded by the redirect-service. It processes and stores data such as scan time, potentially location (if available and consented), and device type, providing users with valuable insights into the performance and reach of their QR codes. The frontend likely utilizes charting libraries like Recharts to visualize this data.
6. Centralized API Access: An api-gateway serves as the single entry point for the frontend application, routing requests to the appropriate backend microservice. This simplifies client-side logic, centralizes concerns like authentication checks (potentially), and provides a unified interface to the system's capabilities.
7. Web-Based User Interface: A responsive frontend application, built with React, TypeScript, and Vite, provides users with an intuitive interface to access all features. It employs a structured design system (leveraging Tailwind CSS and Radix UI components) for visual consistency and usability, enabling users to manage their vCards, view QR codes, and analyze scan statistics effectively.

3.4 CHALLENGES AND SOLUTIONS

Developing a distributed system like the QR Code project presents several inherent technical challenges. Key challenges encountered and their respective solutions include:

1. Challenge: Microservice Complexity: Managing multiple independent services introduces complexities in deployment, inter-service communication, monitoring, and debugging. Ensuring consistency and handling failures across services requires careful design.
    - Solution: Containerization with Docker and orchestration via Docker Compose significantly simplifies deployment and environment consistency. An API Gateway centralizes external communication, reducing client complexity. Employing asynchronous communication patterns (likely via Kafka) decouples services, improving fault tolerance; if one service (e.g., analytics) is temporarily down, others (e.g., redirect) can continue functioning. Standardized logging and health checks (e.g., /metrics endpoints mentioned in README.md) aid monitoring.
2. Challenge: Data Consistency: Maintaining data consistency across different microservices (e.g., ensuring a QR code always points to a valid vCard, linking analytics to the correct user) can be difficult without distributed transactions.
    - Solution: An event-driven approach using Kafka allows services to react to events from other services (e.g., vCard created, QR scan occurred) without tight coupling. This eventually consistent model is often suitable for this type of application. Careful API design ensures necessary data is passed between services during synchronous requests where required. MongoDB's flexible schema can also accommodate embedding related data where appropriate, reducing the need for complex joins.
3. Challenge: Scalability and Performance: Services like the redirect-service and analytics-service might face high load. Inefficient database queries or synchronous processing could lead to bottlenecks.
    - Solution: The microservices architecture inherently allows for independent scaling of services based on load. The redirect-service can be scaled out

independently to handle numerous scan requests. Asynchronous processing in the analytics-service (consuming from Kafka) prevents scan ingestion from blocking the user-facing redirect operation. Using asynchronous frameworks (FastAPI) and drivers (Motor) maximizes I/O efficiency. Potential use of Redis for caching frequently accessed data (like redirect targets) could further enhance performance.

4. Challenge: Frontend Development Complexity: Building a feature-rich, interactive, and consistent user interface requires robust tooling and methodology. Managing state, handling asynchronous data fetching, and ensuring responsive design can be complex.
    - Solution: Utilizing a modern framework like React with TypeScript provides a strong foundation. Vite enhances the development experience. Adopting a design system with predefined components (based on Radix UI and Tailwind CSS) ensures visual consistency and speeds up development. Libraries like axios and potentially server-state tools simplify API interaction and data caching/synchronization.

3.5 FUTURE ENHANCEMENTS

While the QR Code project delivers a robust set of core features, several avenues exist for future enhancement and expansion:

1. Expanded QR Code Types: Introduce support for additional QR code types beyond vCards, such as Wi-Fi configurations, calendar events, URLs, plain text, SMS, or app store links. This would broaden the application's utility.
2. Advanced QR Code Customization: Allow users greater control over the visual appearance of their QR codes, including adding logos, changing colors (foreground/background), and potentially modifying shapes or adding custom frames, while ensuring scannability.
3. Enhanced Analytics and Reporting: Provide more sophisticated analytics, including geographical heatmaps of scans, device/browser breakdowns, scan trends over time, and the ability to generate downloadable reports. Implement filtering and segmentation options for deeper insights.

4. Team/Organization Features: Introduce multi-user accounts, allowing teams or organizations to collaborate on managing QR codes and viewing collective analytics. Implement role-based access control (RBAC) to manage permissions.
5. Template Library: Develop a richer library of pre-designed vCard templates for users to choose from, simplifying the creation process.
6. Third-Party Integrations: Integrate with CRM systems, marketing automation platforms, or contact management tools to allow seamless data flow (e.g., exporting scanned contact details).
7. A/B Testing for QR Codes: Allow users to create multiple destination URLs for a single dynamic QR code and track which performs better, enabling optimization of marketing campaigns.
8. Improved Scalability and Resilience: Implement more advanced infrastructure patterns, potentially exploring Kubernetes for orchestration, refining Kafka topic partitioning and consumer group strategies, and implementing more sophisticated caching layers with Redis. Enhance monitoring and alerting using tools like Prometheus and Grafana as suggested in the README.md.

4.0 KAIROS DOCUMENTATION ANALYSIS

4.1 DOCUMENTATION SCOPE AND STRUCTURE

The Kairos Documentation Project was initiated to systematically capture, organize, and present technical knowledge about the various services within the *Kairos* platform—an internal enterprise-grade software system built using a microservices architecture. This platform comprises numerous interconnected services that together deliver critical communication and processing workflows. Over time, as the platform evolved, the absence of consistent and accessible documentation became a bottleneck, especially for new developers and cross-functional teams. The purpose of this documentation effort was not just to describe the existing code but to establish a maintainable, developer-friendly structure for future updates.

The scope of the documentation includes creating dedicated documents for each major service, organized in a way that separates conceptual understanding from technical deep-dives. To accomplish this, every document was structured under three standardized headers:

1. **Functionalities** – covering the business logic, service responsibilities, and interaction patterns with other components.
2. **Components** – outlining internal architecture, key classes or modules, and data flow models.
3. **Libraries** – detailing third-party and internal dependencies, utility functions, and compatibility notes.

This structure was chosen to ensure both readability and completeness, especially for audiences ranging from developers and testers to system architects. Even though only a portion of the full documentation plan was completed during the internship period—primarily covering the *Product Manager* and *File Processor* services—the groundwork has been laid for continued documentation work across the rest of the platform.

4.2 KEY COMPONENTS DOCUMENTED

The two services selected for documentation were carefully chosen to reflect different operational layers of the Kairos system: one with a high degree of configuration logic and the other dealing with backend data handling and automation. These are:

- Kairos Product Manager: This service is responsible for the management of products, which are modular configurations that can be enabled, disabled, or customized depending on client requirements. The documentation here covers the life cycle of a product—from creation to versioning—and includes a breakdown of how metadata is stored, validated, and retrieved. It also explains the data structures used internally and how various endpoints interact to deliver the intended outcomes.
- Kairos File Processor: A backend service that handles the ingestion, validation, transformation, and routing of input files. It performs various rule-based checks and workflows, often acting as a bridge between manual inputs and automated system operations. Documenting this service involved tracing the complete flow of a file through its different processing states, understanding the configuration-driven logic, and capturing how the system handles anomalies such as file corruption or format mismatches.

Both services were documented not just at a code level, but with an emphasis on their functional roles within the system, how they integrate with other modules, and what dependencies they rely on. This comprehensive approach ensures that even someone unfamiliar with the system can understand what the service does and how it fits into the broader architecture.

4.3 DOCUMENTATION METHODOLOGY

The first step involved codebase exploration, where each service was analyzed to understand its purpose, dependencies, and flow. This was followed by reverse engineering, especially for legacy services that lacked prior documentation or design references.

For each module, a lightweight “Code Digest” was created—a document that captures an overview of the module in a concise manner. This served as both a quick reference and a stepping stone toward the full documentation. Each digest was then expanded into a detailed page following the Functionalities–Components–Libraries format.

The information was gathered through various means including:

- Code-level reading and tracing execution paths.
- Collaborating with developers for clarification and validation.

Wherever possible, visual aids such as service diagrams, sequence flows, and data schemas were introduced to complement the textual information. These visuals helped in conveying complex logic or integration patterns in a more digestible form.

Contributions were made to internal platforms that support collaborative editing, version control, and review workflows, ensuring that the documentation process itself was integrated into the software development lifecycle.

4.4 TOOLS AND TECHNOLOGIES USED

The tools used in the Kairos Documentation Project were selected for their ease of integration into existing workflows and their support for structured content creation. Most documents were written in Markdown, providing a balance between simplicity and formatting capabilities. These were then converted to PDFs using Markdown-to-PDF tools to create offline-friendly reference material.

Initial exploration into automated documentation tools like Doxygen was undertaken, especially for generating function-level or class-level descriptions from source code. However, it was quickly realized that while these tools are powerful for code-level introspection, they fall short when it comes to documenting service-level behavior, business logic, and integration flows—areas where manual insights and contextual explanations are critical.

Documents were maintained using platforms like StackEdit and integrated into shared internal wikis to promote transparency and accessibility. This allowed team members to contribute, comment, and review the documentation as it evolved, fostering a culture of collaboration around technical knowledge.

4.5 CHALLENGES AND SOLUTIONS

Several challenges were encountered during the course of this documentation initiative. One major challenge was the lack of prior documentation, particularly for legacy or highly integrated services. Understanding such services required deep dives into the code,

studying logs, and in some cases, running the services locally to observe their behavior. To address this, a systematic code-walkthrough strategy was adopted—breaking down services into logical flows and mapping them out one by one.

Another challenge was maintaining consistency and clarity across different service documents. Without a predefined structure, it's easy for documentation to become inconsistent in tone, depth, or style. To overcome this, a universal template was developed early in the process and strictly followed for every service documented.

Further, due to the sheer number of services in the platform, prioritization became important. Services were chosen based on complexity, usage frequency, and criticality. The aim was to build a repeatable process on a small scale that could then be expanded to cover the entire system in future phases.

In cases where service behavior was tied to external configurations or runtime environments, local simulation or developer assistance was sought. This collaboration ensured that documentation did not remain theoretical but reflected actual behavior as observed in production or staging environments.

4.6 FUTURE DOCUMENTATION NEEDS

While this phase of the project successfully delivered high-quality documentation for two important services, it also exposed the larger scope and long-term importance of structured documentation within the Kairos ecosystem. Future documentation efforts should focus on:

- Expanding coverage to include the remaining services, especially those that interface directly with external systems or contain complex business rules.
- Creating onboarding guides tailored to developers, testers, and DevOps personnel to speed up integration into active projects.
- Incorporating automated documentation pipelines wherever possible.
- Enriching existing documents with visual diagrams, flowcharts, and configuration snapshots to make them more intuitive and engaging.
- Establishing a review and update cycle to ensure documents remain relevant as code evolves.

By continuing the momentum set during this internship, the Kairos platform can move towards becoming not just a powerful product but also a well-documented, developer-friendly system that supports faster onboarding, smoother development, and more efficient maintenance.