# **Project Mediconnect: The 2-Week Prototype Software Development Plan**

## **Part 1: The 2-Week Prototype Sprint Mandate**

### **1.1. Defining the Mission: The Production-Grade "Steel Thread"**

This document outlines the Software Development Plan (SDP) for the delivery of a production-grade working prototype of the Mediconnect platform. The mandate is to execute a focused, two-week (14-day) sprint to build the foundational, vertically-integrated slice of the application. This is not a disposable proof-of-concept; it is the "steel thread" upon which the full Mediconnect MVP will be constructed.

The objective is to de-risk the project by forcing the integration of all critical layers—frontend user interface, backend API services, the PostgreSQL database with its security model, authentication, and the CI/CD infrastructure—at the earliest possible stage. Every component delivered within this sprint must adhere to production standards for security, observability, and deployability, as stipulated in the project requirements.1 This approach validates the core architectural and business theses of the Mediconnect platform, ensuring a viable foundation for the subsequent three-month MVP development cycle detailed in the Product Requirements Document (PRD).1

Success for this two-week sprint is defined by the delivery of a web-hosted, demonstrably functional application that meets all criteria outlined in the "Working Prototype" definition. This includes two fully operational end-to-end business flows, a complete scaffolding of all other features behind functional "Coming Soon" gates, and the foundational infrastructure for continuous integration and deployment.1

### **1.2. Scope Finalization: Core Flows & "Coming Soon" Features**

To achieve the sprint objective within the two-week constraint, the scope is aggressively prioritized. The prototype will implement two patient-centric core business flows and create a comprehensive UI shell for all remaining features.

#### **Core Flows: ACTIVE**

The two flows selected for full implementation represent the primary value proposition of the GP-orchestrated Healthcare-as-a-Service (HaaS) model. They are the most critical paths for validating the platform's technical and operational viability.1

1. **GP Consultation Request:** This flow is the patient's primary entry point into the Mediconnect ecosystem. It encompasses the mandatory AI Intake process, the assignment of a General Practitioner, and the initiation of the hybrid in-app/WhatsApp consultation. Its implementation is critical to validating the "GP as the clinical front door" principle.1
2. **Digital Prescription Fulfillment:** This flow is a key market differentiator, addressing the critical problem of unsafe medication dispensing. It involves the secure generation of a digital prescription with a dual QR code/PDF system and the crucial security feature of QR code disablement upon PDF download. This flow validates both a core clinical function and a key partner interaction (pharmacy).1

#### **Gated Features: COMING\_SOON**

As per the prototype definition, all other features outlined in the PRD will be present in the user interface but gated behind a "Coming Soon" screen. Each gated feature will include a functional waitlist capture form (email/phone) with consent text and will trigger the requisite analytics events.1 This strategy provides full visibility into the platform's intended scope, gauges user interest in future functionality, and builds a waitlist for subsequent feature launches.

The gated features include, but are not limited to:

* Specialist Referral Process
* Diagnostic Test Ordering
* Health Records & Document Management
* Payments & Invoicing
* Dependents & Family Profiles
* Address Management
* Advanced Notification Preferences

#### **Table 1: Feature Flag & Prototype Scope Map**

The state of every feature within the prototype will be controlled by a global feature flag map. This configuration serves as the single source of truth for the UI, enabling dynamic control and forming the basis for future phased rollouts.

| Feature Name | Route/Screen ID | Prototype Status | Description |
| --- | --- | --- | --- |
| **Authentication** | PAT\_003 | ACTIVE | WhatsApp-only OTP login and session management. |
| **AI Health Assistant & GP Consult** | PAT\_005, PAT\_006 | ACTIVE | The primary chat interface, AI intake, and GP consultation request flow. |
| **Prescription View & Fulfillment** | PAT\_009, PAT\_010 | ACTIVE | Viewing prescriptions, QR code display, and finding partner pharmacies. |
| **Specialist Appointments** | SP\_HOME | COMING\_SOON | Gated screen for viewing and managing specialist referrals and appointments. |
| **Diagnostics & Lab Orders** | DX\_HOME | COMING\_SOON | Gated screen for tracking diagnostic orders and viewing results. |
| **Health Records & Documents** | PAT\_011 | COMING\_SOON | Gated screen for accessing historical consults, reports, and uploaded documents. |
| **Payments & Invoices** | PAY\_HOME | COMING\_SOON | Gated screen for managing payment methods and viewing billing history. |
| **Dependents & Family Profiles** | DEP\_HOME | COMING\_SOON | Gated screen for adding and managing care for family members. |
| **Address Management** | ADDR\_HOME | COMING\_SOON | Gated screen for managing delivery and consultation addresses. |
| **Health Education** | EDU\_HOME | COMING\_SOON | Gated screen for accessing curated health education content. |

### **1.3. Key Assumptions & Decisions Log (Sprint Baseline)**

To eliminate ambiguity and mitigate risks arising from conflicting information in the source documentation, the following decisions are locked in for the duration of the two-week sprint. These serve as the foundational contract for execution.1

1. **Authentication Will Be a Live Integration:** The user query permits a "safe stub" for authentication. However, the PRD's repeated emphasis on the WhatsApp-only OTP flow as a core differentiator and critical part of the user experience necessitates its validation from day one.1 Therefore, the prototype will implement a  
    **live, functional integration** with the WhatsApp Cloud API for OTP delivery. This is a high-risk, high-reward decision that directly aligns with the "production-grade" mandate.
2. **Core Flows Are Patient-Centric; Partner Portals are Deferred:** The two-week scope will focus exclusively on the **patient's journey** through the two core flows via the mobile/web application. The corresponding GP and Pharmacy Admin interactions (e.g., issuing a prescription, verifying a QR code) will be fully implemented in the backend API and verifiable via API calls and database state changes. The development of the web-based partner portals (GP\_Portal, PH\_Portal) is deferred to the next sprint. This is a necessary scope reduction to meet the aggressive timeline.
3. **Specialist & Diagnostics Flows are "Coming Soon":** While the PRD contains detailed user journeys for specialist referrals and diagnostics, their full implementation is a significant undertaking. To ensure the successful delivery of the two primary core flows, the Specialist Referral and Diagnostics flows are officially designated as COMING\_SOON features for this prototype sprint. Their UI entry points will be fully implemented as specified, with functional waitlist capture.1
4. **Prototype Frontend is Web-Hosted:** To maximize development velocity and simplify deployment for the two-week sprint, the prototype will be a mobile-first, responsive web application hosted on a platform like Vercel or Netlify, rather than a native Android application. This decision allows the team to focus on feature implementation without the overhead of native build tooling and app store deployment processes. The component structure will be designed for future migration to a React Native shell.
5. **Payment Integration is Out of Scope:** Resolving the conflict identified in the source analysis, this prototype will strictly adhere to the "no in-app payments" rule stated in the MVP scope.1 The "Payments" section of the UI will be a  
    COMING\_SOON gate.

## **Part 2: Prototype Architectural Blueprint**

### **2.1. System Architecture Slice**

The prototype's architecture is a focused vertical slice of the full system design outlined in the project documentation.1 It comprises the essential components required to deliver the core functionality and prove the viability of the technical strategy.

The architecture consists of:

* **Frontend Application:** A mobile-first web application built with a modern JavaScript framework (e.g., Next.js or React with Vite). It is responsible for all user-facing interactions, state management, and communication with the backend API.
* **Backend Service:** A single, monolithic service built with Node.js and TypeScript. It exposes a RESTful API, handles all business logic for authentication and the two core flows, enforces permissions, and interacts with the database and external services.
* **Database:** A managed PostgreSQL instance (e.g., via Supabase). It provides data persistence and, critically, enforces the security model through Row-Level Security (RLS) policies.
* **External Integrations:** A minimal set of crucial third-party services, primarily the **WhatsApp Cloud API** for authentication.
* **Infrastructure & CI/CD:** A cloud hosting platform (e.g., Vercel) and a CI/CD pipeline (e.g., GitHub Actions) to automate testing and deployment.

### **2.2. Data & Security Foundation**

Security is a non-negotiable, foundational requirement that will be implemented from the outset, not as a later addition. The prototype will establish the core of the platform's security posture.

#### **Prototype Entity-Relationship Diagram (ERD)**

The prototype's data model includes only the tables essential for the scoped features. This lean schema allows for rapid implementation while establishing the correct relationships and constraints for future expansion.1

Code snippet

erDiagram

"user" {

UUID id PK

user\_role role

TEXT phone\_e164 UK

TEXT full\_name

}

"consult" {

UUID id PK

consult\_status status

UUID patient\_id FK

UUID gp\_id FK

TEXT chief\_complaint

TIMESTAMPTZ started\_at

}

"message" {

UUID id PK

UUID consult\_id FK

UUID sender\_id FK

message\_type type

TEXT body\_text

}

"prescription" {

UUID id PK

UUID consult\_id FK

UUID patient\_id FK

UUID prescriber\_id FK

prescription\_status status

JSONB items\_json

TEXT qr\_code

BOOLEAN qr\_enabled

TIMESTAMPTZ pdf\_downloaded\_at

}

"pharmacy\_claim" {

UUID id PK

UUID prescription\_id FK

UUID org\_id FK

claim\_status status

BOOLEAN qr\_verified

}

"audit\_event" {

UUID id PK

UUID actor\_user\_id FK

TEXT action

TEXT subject\_table

UUID subject\_id

}

"user" |

|--o{ "consult" : "is patient in"

"user" |

|--o{ "consult" : "is GP for"

"user" |

|--o{ "message" : "sends"

"user" |

|--o{ "prescription" : "is patient for"

"user" |

|--o{ "prescription" : "is prescriber for"

"user" |

|--o{ "audit\_event" : "is actor for"

"consult" |

|--o{ "message" : "contains"

"consult" |

|--o{ "prescription" : "results in"

"prescription" |

|--o{ "pharmacy\_claim" : "is claimed by"

#### **Row-Level Security (RLS) Implementation Plan**

The implementation of Row-Level Security is the first and most critical task for the backend workstream. It is a forcing function that ensures every database query is written with the correct security context from the beginning. Deferring RLS would introduce significant security risks and necessitate a costly refactor. The prototype will implement the RLS policies for all relevant roles on the scoped tables as defined in the PRD Addendum.1

**Execution Directive:**

1. Enable RLS on the user, consult, message, prescription, and pharmacy\_claim tables.
2. Implement a middleware in the backend service that sets the app.user\_id and app.role session variables for every authenticated database request.
3. Implement and test the following core policies 1:
   * **patient\_consults:** A patient can only access consult rows where their ID matches patient\_id.
   * **patient\_messages:** A patient can only access message rows linked to their consults.
   * **patient\_prescriptions:** A patient can only read prescription rows where their ID matches patient\_id.
   * **gp\_consults:** A GP can only access consult rows where their ID matches gp\_id.
   * **pharmacy\_verify\_rx:** A pharmacy admin can only read a prescription row if qr\_enabled is TRUE and a claim exists for their organization. This policy is critical for the PII-minimized QR verification flow.

### **2.3. API Contract**

The prototype will implement a subset of the full OpenAPI v3 specification. This API slice is the minimum required to power the authentication flow and the two core business journeys. All endpoints will correctly implement problem+json for error responses and support the x-idempotency-key header on all POST requests to ensure production-grade reliability.1

**Implemented Endpoints:**

* **Authentication**
  + POST /auth/whatsapp/otp: Initiates the OTP login process by sending a code to the user's WhatsApp number.
  + POST /auth/whatsapp/verify: Verifies the OTP and returns a JWT session token.
* **GP Consultation**
  + POST /consults: Creates a new consultation after a patient completes the AI intake.
  + GET /consults/{id}: Retrieves the details of a specific consultation, accessible only to the involved patient and GP.
  + GET /consults/{id}/messages: Retrieves the message history for a consultation.
  + POST /consults/{id}/messages: Sends a new message within a consultation.
* **Prescription Fulfillment**
  + POST /prescriptions: Issues a new digital prescription (action performed by a GP).
  + GET /prescriptions/{id}: Retrieves the details of a specific prescription, including the QR code data and PDF download status.
  + POST /prescriptions/{id}/download-pdf: A logical endpoint to handle the action of downloading the PDF, which triggers the QR disablement logic.
  + POST /pharmacy/claims/verify-qr: Allows a pharmacy admin to verify a QR code and receive a PII-minimized view of the prescription items.
  + POST /pharmacy/claims: Submits a fulfillment claim against a prescription after verification.

## **Part 3: The 14-Day Sprint Execution Plan**

### **3.1. Workstream Breakdown**

The project will be executed by three parallel, coordinated workstreams. This structure is designed for maximum velocity, allowing concurrent development on the backend, frontend, and infrastructure. Clear API contracts and daily synchronization are essential for this model's success.

* **Backend Workstream:** Responsible for the Node.js/TypeScript service. This team will implement the API endpoints, all business logic, database interactions (including the critical RLS policies), and the integration with the WhatsApp Cloud API. They are the custodians of the data model and security enforcement.
* **Frontend Workstream:** Responsible for the patient-facing web application. This team will build all UI components, manage application state, implement the user flows for the two core journeys, and, critically, build the comprehensive "Coming Soon" scaffolding for all other features, including the feature flag map and waitlist capture component.
* **Infrastructure Workstream (DevOps):** Responsible for the foundational platform. This team will provision the cloud environments on Supabase and Vercel, configure the CI/CD pipeline in GitHub Actions, manage secrets and environment variables, and ensure the database schema is deployed and maintained. They enable the other two teams to build and deploy code efficiently and securely.

### **3.2. Sprint Backlog**

This backlog represents the complete set of tasks for the two-week sprint. It is prioritized to tackle the highest-risk and highest-dependency items first. Story points are assigned using a modified Fibonacci scale to estimate effort and complexity.

| Epic | Story / Task | Priority | Workstream | Dependencies | Story Points | Acceptance Criteria |
| --- | --- | --- | --- | --- | --- | --- |
| **Foundation** | \*\*\*\* Provision Dev/Staging/Prod environments in Supabase & Vercel. | **Critical** | Infrastructure | - | 8 | Environments are accessible; domain names are configured. |
| **Foundation** | \*\*\*\* Set up CI/CD pipeline in GitHub Actions to build, test, and deploy to Staging. | **Critical** | Infrastructure | INFRA-1 | 8 | A push to the main branch triggers a successful deployment to the staging URL. |
| **Foundation** | \*\*\*\* Implement DB schema for prototype tables (user, consult, etc.) and RLS policies. | **Critical** | Backend | INFRA-1 | 13 | Schema is deployed; RLS policies are active and block unauthorized queries. |
| **Auth** | \*\*\*\* Implement POST /auth/whatsapp/otp and POST /auth/whatsapp/verify with live WhatsApp API. | **Critical** | Backend | BE-1 | 13 | A user can receive a real WhatsApp OTP and exchange it for a valid JWT. auth.otp.verified is logged. |
| **Auth** | **[FE-1]** Build Onboarding/Login UI screens (PAT\_003) and connect to auth endpoints. | **High** | Frontend | BE-2 | 8 | A user can enter their phone number, submit an OTP, and receive a session token, which is stored locally. |
| **GP Consult** | \*\*\*\* Implement POST /consults and GET /consults/{id} with RLS checks. | **High** | Backend | BE-1 | 8 | An authenticated patient can create a consult; only the patient or assigned GP can retrieve it. consult.created is logged. |
| **GP Consult** | \*\*\*\* Implement message endpoints (GET/POST /consults/{id}/messages). | **High** | Backend | BE-3 | 5 | Authenticated users can send and receive messages within their own consults. message.sent is logged. |
| **GP Consult** | **[FE-2]** Build Chat UI (PAT\_005, PAT\_006) for AI Intake and GP consultation. | **High** | Frontend | FE-1, BE-4 | 13 | User can complete the intake form, which creates a consult and transitions to a chat view where messages can be sent/received. |
| **Prescription** | \*\*\*\* Implement POST /prescriptions and GET /prescriptions/{id}. | **High** | Backend | BE-3 | 8 | A GP (via API) can issue a prescription linked to a consult. The patient can retrieve it. prescription.issued is logged. |
| **Prescription** | \*\*\*\* Implement QR disablement logic (pdf\_downloaded\_at, qr\_enabled). | **Critical** | Backend | BE-5 | 8 | Calling a logical "download PDF" endpoint sets qr\_enabled to false. prescription.qr.disabled is logged. |
| **Prescription** | \*\*\*\* Implement POST /pharmacy/claims/verify-qr with PII-minimized response. | **High** | Backend | BE-6 | 8 | A Pharmacy Admin (via API) can verify an active QR code and receive only the item list, not patient PII. |
| **Prescription** | **[FE-3]** Build Prescription View UI (PAT\_009) showing QR code and PDF download button. | **High** | Frontend | BE-5, BE-6 | 8 | UI correctly displays the prescription, and the download button triggers the QR disablement flow. |
| **Coming Soon** | **[FE-4]** Create global FeatureFlagMap configuration file. | **High** | Frontend | - | 3 | A single TypeScript/JSON file defines the ACTIVE or COMING\_SOON state for all features in Table 1. |
| **Coming Soon** | **[FE-5]** Build reusable ComingSoonGate component with waitlist capture and analytics. | **High** | Frontend | FE-4 | 8 | A single React component can be configured to display the gate, capture an email/phone, and fire the 3 required analytics events. |
| **Coming Soon** | **[FE-6]** Create all gated routes/screens using the ComingSoonGate component. | **Medium** | Frontend | FE-5 | 5 | All COMING\_SOON features from Table 1 have a route and display the gate correctly. |
| **Ops** | **[ALL-1]** Implement observability seed: structured logging and basic metrics. | **Medium** | All | INFRA-1 | 5 | API requests are logged with status codes and latency. Basic error rates are tracked. |
| **Ops** | **[ALL-2]** Write and validate end-to-end tests for the two core flows. | **High** | All | FE-2, FE-3 | 8 | Automated tests (e.g., using Cypress or Playwright) successfully run the happy path and one edge case for both core flows. |
| **Ops** | **[ALL-3]** Write the final demonstration script. | **Medium** | All | ALL-2 | 3 | A step-by-step script for the final demo is complete and validated against the staging environment. |

### **3.3. Day-by-Day Execution Gantt**

This schedule front-loads the highest-risk tasks, such as infrastructure setup and external API integrations, to provide the earliest possible feedback on project viability.

* **Day 0-1: Foundation & Authentication**
  + **Objective:** Establish a working, deployable application with live authentication.
  + **Infrastructure:** Environments provisioned. CI/CD pipeline is functional, deploying a "hello world" app to staging.
  + **Backend:** Implements and deploys the full WhatsApp OTP authentication flow. A test user can successfully log in on the staging environment.
  + **Frontend:** Builds the shell of the application with routing and the basic login screens.
  + **Goal:** Prove the core infrastructure and the highest-risk external dependency are viable.
* **Day 2-4: Core Flow Backend Implementation**
  + **Objective:** Complete all backend API endpoints for the two core flows.
  + **Backend:** Implements all endpoints for the GP Consult and Prescription Fulfillment flows, including all business logic, database transactions, RLS enforcement, and audit logging. The API is fully testable via a tool like Postman.
  + **Frontend:** Begins building reusable UI components (buttons, inputs, cards) based on the design system.
  + **Infrastructure:** Stabilizes the CI/CD pipeline and assists the backend team with database migrations.
* **Day 5-8: Core Flow Frontend Integration**
  + **Objective:** Build and connect the UI for the two core flows to the live backend API.
  + **Frontend:** Builds the complete user interface for the GP Consultation and Prescription Fulfillment journeys, integrating with the now-stable backend APIs. State management is fully implemented.
  + **Backend:** Supports the frontend team with API integration, fixes bugs, and refines API responses based on frontend needs.
  + **Goal:** Have two fully functional, end-to-end user journeys working in the staging environment.
* **Day 9-11: "Coming Soon" Scaffolding & Hardening**
  + **Objective:** Implement all COMING\_SOON gates and harden the existing features.
  + **Frontend:** Implements the feature flag map and the reusable ComingSoonGate component. Creates all the gated pages for the remaining features.
  + **All Teams:** Focus on writing and refining tests (unit, integration, E2E). Bugs identified in the core flows are prioritized and fixed. Observability is enhanced with more detailed logging and metrics.
  + **Goal:** The application is feature-complete according to the prototype scope and is stable.
* **Day 12-13: Demo Preparation & Final Polish**
  + **Objective:** Prepare for the final demonstration and apply final polish.
  + **All Teams:** Conduct full end-to-end testing on staging. The demonstration script is rehearsed. Final UI/UX tweaks are made based on team feedback. All documentation (e.g., READMEs, API docs) is finalized.
  + **Infrastructure:** The production environment is configured and locked down. A final deployment to staging is performed to ensure the process is smooth.
* **Day 14: Demonstration & Handoff**
  + **Objective:** Demonstrate the working prototype and hand off the assets.
  + **Morning:** A final smoke test is performed on the staging environment.
  + **Afternoon:** The live demonstration is conducted for stakeholders, following the finalized script.
  + **End of Day:** The project is handed off, including the source code repository, access to the deployed staging environment, and all supporting documentation.

## **Part 4: Core Flow Implementation Deep Dive**

### **4.1. Flow 1: GP Consultation Request (Orchestration Guide)**

This flow is the patient's entry point to care. Its implementation must be seamless and robust, combining the in-app experience with the WhatsApp integration.

**Step 1: Frontend - AI Intake (PAT\_006)**

* **Components:** Build a dynamic form component, AIIntakeForm, that renders questions based on a JSON configuration. Use components like QuestionCard and NavigationButtons.1
* **State Management:** As the user answers questions, store the responses in a local state object (e.g., using Zustand or React's useState).
* **Submission:** On final submission, the "Request GP Consultation" button triggers an API call to POST /consults. The request body will contain the patient's ID (from their auth state) and the collected intake responses.

**Step 2: Backend - Consultation Creation (POST /consults)**

* **Controller Logic:** The endpoint receives the intake data. It must be protected, requiring a valid JWT.
* **Transaction:** The service layer initiates a database transaction to ensure atomicity:
  1. Create a new row in the consult table with status: 'active', linking the patient\_id and an assigned gp\_id (initially, this can be a hardcoded test GP). The chief\_complaint is populated from the intake data.
  2. Create an initial message in the message table, linked to the new consult.id. The sender\_id can be a system user, and the body\_text can be a welcome message like, "Thank you for providing your information. A GP will be with you shortly."
  3. Insert a consult.created event into the audit\_event table, capturing the actor\_user\_id (the patient) and the subject\_id (the new consult.id).1
* **Response:** The API returns a 201 Created status with the newly created Consult object.

**Step 3: Frontend - Transition to Chat (PAT\_005)**

* **Routing:** Upon a successful response from POST /consults, the frontend application navigates the user to the ChatThread screen, passing the new consult.id.
* **Data Fetching:** The ChatThread component uses the consult.id to fetch the initial messages by calling GET /consults/{id}/messages.
* **Real-time Connection:** The component establishes a real-time subscription (e.g., using Supabase Realtime) to the message table, filtered by the current consult.id, to listen for new messages.

**Step 4: Backend & Frontend - Messaging**

* **Frontend:** The MessageInput component, on send, calls POST /consults/{id}/messages with the message text. The message is optimistically added to the local UI state for a responsive feel.
* **Backend:** The POST endpoint validates that the sender is a participant in the consult (via RLS), inserts the new message into the message table, and logs a message.sent audit event. The real-time service automatically broadcasts the new message to subscribed clients.

### **4.2. Flow 2: Digital Prescription Fulfillment (Orchestration Guide)**

This flow demonstrates a critical clinical function with a unique security model. The backend logic is paramount.

**Step 1: Backend - Prescription Issuance (POST /prescriptions)**

* **Action:** This endpoint is called by a GP (for the prototype, this will be simulated via an API client like Postman, acting as the GP).
* **Logic:**
  1. The service validates that the actor has the gp role and is associated with the consult\_id in the request.
  2. It generates a unique, secure token for the QR code.
  3. A new row is inserted into the prescription table with status: 'issued', qr\_enabled: true, and pdf\_downloaded\_at: null. The items\_json is populated from the request body.1
  4. A prescription.issued event is logged to the audit\_event table.
* **Response:** Returns the full Prescription object, including the qr\_code data.

**Step 2: Frontend - Viewing the Prescription (PAT\_009)**

* **Data Fetching:** The PrescriptionView screen calls GET /prescriptions/{id} to fetch the prescription details.
* **Conditional UI:** The UI must conditionally render based on the API response:
  + If qr\_enabled is true, display the QRCodeDisplay component with the qr\_code data. The "Download PDF" button is active.
  + If qr\_enabled is false, hide the QR code and display a message: "QR code has been disabled because the PDF was downloaded".1 The "Download PDF" button may be styled as "Re-download PDF."
* **User Action:** When the user clicks "Download PDF," the frontend calls the logical endpoint POST /prescriptions/{id}/download-pdf.

**Step 3: Backend - PDF Download & QR Disablement**

* **Controller Logic:** This endpoint is the crux of the security model. It performs a critical, atomic transaction.
* **Transaction:**
  1. It finds the prescription record by its ID.
  2. It checks if qr\_enabled is already false. If so, it can simply serve the PDF.
  3. If qr\_enabled is true, it updates the row, setting qr\_enabled = false and pdf\_downloaded\_at = now().
  4. It logs two critical audit events: prescription.pdf.downloaded and prescription.qr.disabled.1
* **Response:** After the transaction succeeds, it generates (or retrieves) the PDF and streams it back to the client.

**Step 4: Backend - Pharmacy QR Verification (POST /pharmacy/claims/verify-qr)**

* **Action:** This endpoint is called by a Pharmacy Admin (simulated via an API client). The request contains the qr\_payload.
* **Logic:**
  1. The service decodes the QR payload to get the prescription.id.
  2. It queries the prescription table for that ID. **Crucially, the RLS policy pharmacy\_verify\_rx will automatically filter this query.** The database will only return a row if qr\_enabled is TRUE and a claim for the pharmacy's organization already exists (or is being created).
  3. If a row is returned, the API responds with a PII-minimized object containing only the prescription\_id and items\_json.
  4. If no row is returned (because the QR is disabled or invalid), the API returns a 409 Conflict or 404 Not Found error.

## **Part 5: "Coming Soon" Scaffolding & Analytics Implementation**

### **5.1. The Feature Flag Map**

The foundation of the "Coming Soon" architecture is a single, centralized configuration file. This approach avoids hardcoding UI states and allows for easy updates as features are developed.

**File: src/config/featureFlags.ts**

TypeScript

export enum FeatureStatus {

ACTIVE = 'ACTIVE',

COMING\_SOON = 'COMING\_SOON',

DISABLED = 'DISABLED',

}

export interface FeatureFlag {

id: string;

name: string;

status: FeatureStatus;

teaser: string;

}

export const featureFlagMap: FeatureFlag =;

### **5.2. Building the Gated UI**

The key to implementing the numerous "Coming Soon" screens efficiently is to build a single, reusable component that is driven by the configuration map. This avoids duplicating code for each gated feature.

Component: src/components/ComingSoonGate.tsx

This component will accept a FeatureFlag object as a prop.

* It will display a prominent "Coming Soon" title.
* It will render the teaser text from the feature flag object to give users a preview of the functionality.
* It will include the WaitlistCaptureForm component.
* It will fire the viewed\_gate analytics event in a useEffect hook when it mounts, passing the feature.name as a property.

Routing:

The application's router will be configured with paths for all features. Routes for ACTIVE features will render the functional components. Routes for COMING\_SOON features will render the ComingSoonGate component, passing the corresponding feature flag object as a prop.

### **5.3. Waitlist Capture & Analytics**

The waitlist form is a critical piece of functionality for gauging user interest and building a launch audience for new features.

**Component: src/components/WaitlistCaptureForm.tsx**

* **UI:** The form will contain an input field for an email address or phone number, a checkbox for consent to be contacted, and a submit button. The consent text must be clear and unambiguous.
* **State:** It will manage its own state for the input field and submission status.
* **Analytics:**
  1. When the user focuses on the input field or clicks the submit button for the first time, it fires the clicked\_notify event.
  2. Upon successful form submission, it fires the submitted\_waitlist event.
* **API Call:** On submission, it will call a new, simple backend endpoint (e.g., POST /waitlist-entries) that saves the user's contact information and the feature they are interested in to a new waitlist\_entry database table. This provides a persistent record of user interest.

This component-based, configuration-driven approach transforms a potentially large and repetitive task into a manageable and scalable solution, which is the only realistic way to meet this extensive requirement within the two-week sprint.

## **Part 6: Prototype Handoff & Demonstration Protocol**

### **6.1. Definition of "Done" Checklist**

This checklist serves as the final acceptance criteria for the two-week sprint. The prototype is considered "Done" only when all items are verified as complete on the staging environment.

* **[ ] Core Functionality:**
  + [ ] User can register and log in using a real phone number via WhatsApp OTP.
  + [ ] User can complete the AI Intake and successfully request a GP consultation.
  + [ ] A consult record is created in the database with the correct patient and GP associations.
  + [ ] User can send and receive messages in the consultation chat thread.
  + [ ] A prescription can be issued (via API) and appears correctly in the user's UI.
  + [ ] The prescription UI correctly displays the QR code and PDF download option.
  + [ ] Downloading the PDF successfully and permanently disables the QR code, reflected in both the UI and the database (qr\_enabled: false).
  + [ ] A pharmacy admin (via API) can successfully verify an active QR code.
  + [ ] A pharmacy admin (via API) receives a 409 Conflict error when attempting to verify a disabled QR code.
* **[ ] "Coming Soon" Scaffolding:**
  + [ ] All features marked COMING\_SOON in Table 1 have a navigable route in the UI.
  + [ ] Each gated route displays the ComingSoonGate component with the correct feature name and teaser text.
  + [ ] The waitlist capture form is functional for every gated feature.
  + [ ] Submitting the waitlist form successfully saves the entry to the database.
  + [ ] The viewed\_gate, clicked\_notify, and submitted\_waitlist analytics events are correctly fired for each gated feature.
* **[ ] Production Readiness:**
  + [ ] The project has a CI/CD pipeline that automatically tests and deploys the application to a staging URL.
  + [ ] All implemented API endpoints are documented with OpenAPI stubs.
  + [ ] All API error responses conform to the problem+json standard.
  + [ ] All POST endpoints correctly handle the x-idempotency-key header.
  + [ ] Structured logs are generated for all API requests and critical events.
  + [ ] End-to-end smoke tests for the two core flows pass successfully on staging.
  + [ ] The demonstration script runs successfully from start to finish on the staging environment.

### **6.2. The Demonstration Script**

This script provides a narrative for the final stakeholder demonstration, following the journey of a new user named Sarah.

**Narrator:** "Welcome to the Mediconnect prototype demonstration. Today, we'll follow a new user, Sarah, as she experiences our core healthcare journey for the first time."

1. **Onboarding & Authentication:**
   * "Sarah opens the Mediconnect web app and is prompted to sign in with her phone number." (Show login screen).
   * "She enters her Kenyan phone number. The system sends a verification code directly to her WhatsApp." (Enter number, show a screenshot of the WhatsApp message).
   * "She enters the code and is securely logged in, establishing a session." (Enter OTP, navigate to the home screen).
2. **GP Consultation Request:**
   * "Sarah is feeling unwell and decides to consult a doctor. She starts the AI Health Assistant." (Navigate to the chat screen, click "Start Health Check").
   * "She answers a few simple, adaptive questions about her symptoms. This structured intake will save the GP valuable time." (Complete the intake form).
   * "Upon completion, her request is sent, and she is placed in a chat with her assigned GP. The system has already sent a welcome message." (Show the chat thread).
   * "Sarah can now communicate with her GP via the in-app chat." (Send a message like "Hello Doctor").
3. **Digital Prescription Fulfillment:**
   * "After the consultation, Sarah's GP has issued her a prescription. It appears instantly in her app." (Navigate to the Prescriptions screen, showing the new prescription).
   * "She can view the details, including the secure QR code for verification at a partner pharmacy." (Show the prescription details with the active QR code).
   * "Sarah knows she'll be near a pharmacy that isn't a partner, so she chooses to download the PDF. The app clearly warns her that this will disable the QR code for security." (Click the "Download PDF" button).
   * "The PDF is downloaded. Now, when she returns to the prescription, the QR code is visibly disabled, preventing any fraudulent use." (Show the prescription view again with the "QR Disabled" message).
4. **"Coming Soon" Scaffolding & Future Vision:**
   * "Mediconnect is a comprehensive platform. Let's explore the future. Sarah needs to see a specialist for a follow-up." (Navigate to the "Specialist Care" section).
   * "The feature is marked 'Coming Soon.' The teaser explains the value, and Sarah can join the waitlist to be notified when it's available." (Show the ComingSoonGate screen).
   * "She enters her email, gives consent, and submits. This action is recorded, helping us prioritize our roadmap based on real user demand." (Submit the waitlist form).
   * "This same experience exists for all our upcoming features, like Diagnostics, Health Records, and Payments, giving users full visibility into our vision." (Quickly click through 2-3 other gated screens).

**Narrator:** "This concludes our demonstration. In just two weeks, we have built a secure, functional, and deployable foundation for the Mediconnect platform, validating our core architecture and delivering the two most critical user journeys."

### **6.3. Next Steps & Recommendations**

This prototype successfully establishes a production-grade foundation for the Mediconnect platform. The following next steps are recommended to build upon this momentum:

1. **Prioritize Partner Portals:** Immediately begin development of the web-based portals for General Practitioners and Pharmacy Admins. This will complete the user loop for the two implemented core flows and enable end-to-end testing with real healthcare partners.
2. **Implement the Next Core Flow:** Commence development of the **Specialist Referral** flow. This is the next most critical journey in the HaaS model and will build directly upon the existing consultation and user management features.
3. **Conduct a Foundational Security Audit:** Before adding significant new functionality, conduct a focused security audit and penetration test of the components built in this sprint, especially the authentication flow and the Row-Level Security policies.
4. **Initiate Pilot User Acceptance Testing (UAT):** Onboard the initial cohort of 20-50 pilot users in Nairobi to begin using the prototype for the two active flows. Gathering real-world feedback on the core experience at this early stage is invaluable.