Elimu Smart: Software Engineering Document 🚀

1. Introduction ✨

This document serves as the comprehensive software engineering guide for Elimu Smart, a career guidance platform designed for Kenyan secondary school students. It outlines the system's vision, functional and non-functional requirements, architectural design, development processes, deployment strategies, and a detailed roadmap. The platform emphasizes a **security-first, mobile-optimized approach** with robust payment integration, and utilizes **logic-based Artificial Intelligence (AI)** without incorporating advanced Machine Learning (ML) for predictive modeling or blockchain technology.

2. System Overview & Vision 💡

Elimu Smart aims to be the leading career guidance platform in Kenya, providing intelligent assessment, subject-to-career mapping, extensive resources, and real-time counselor support. Its core vision is to empower students with the knowledge and tools needed to make informed career decisions, facilitating a seamless transition from secondary school to higher education and beyond.

2.1 Goals

- Provide personalized, contextually relevant career guidance.
- Ensure secure and seamless M-Pesa payment integration.
- Offer a dynamic, **role-based and theme-driven user experience** for students, counselors, and administrators.
- Maintain high performance and system availability on a **Virtual Private Server (VPS)**.
- Establish a scalable architecture for future growth and potential cloud migration.

3. Functional Requirements (FRs) ✅

The following sections detail the essential capabilities of the Elimu Smart platform.

3.1 User Management & Authentication

- **FR-UM-001**: Support multi-option user registration (Email, Phone, Google, Facebook) for Student, Counselor, and Administrator roles.
- **FR-UM-002**: Implement SMS OTP verification for phone number-based registrations and updates.
- **FR-UM-003**: Provide secure password reset functionality.
- **FR-UM-004**: Enable basic user profile creation and editing, capturing personal, academic, career interests, and family information.

- **FR-UM-005**: Display **role-based user interfaces, navigation, and distinct visual themes** dynamically upon login.

3.2 Academic & Career Guidance

- **FR-ACG-001**: Offer an interactive KCSE Grade Calculator with real-time point calculation.
- **FR-ACG-002**: Implement a University Predictor that calculates admission probability and suggests alternative courses based on KCSE grades and historical KUCCPS data.
- **FR-ACG-003**: Provide a Career Assessment Engine for personality, interest, skills, and work values evaluations, generating rule-based outcomes.
- **FR-ACG-004**: Display Subject-to-Career Mapping based on the Kenyan curriculum, suggesting career pathways using defined logic.
- **FR-ACG-005**: Host a comprehensive Career Guidance Hub with curated resources (articles, guides) relevant to various careers.
- **FR-ACG-006**: Integrate and display real-time and historical KUCCPS placement data within a dashboard.
- **FR-ACG-007**: Allow students to track their application status to universities.
- **FR-ACG-008**: Implement a Progress Tracking Dashboard for students to set goals, monitor milestones, and visualize achievements.

3.3 Communication & Support

- **FR-CS-001**: Provide a real-time Live Chat system for students to communicate with counselors.
- **FR-CS-002**: Implement a **rule-based chatbot** for basic career guidance, answering common questions.
- **FR-CS-003**: Enable counselors to manage chat sessions, student assignments, session notes, and schedules via a dedicated dashboard.
- **FR-CS-004**: Integrate with ZeptoMail for sending transactional emails (welcome, verification, invoices, notifications).

3.4 Subscription & Payment

- **FR-SP-001**: Display tiered subscription plans (Free, Student Premium, Transition Complete) with feature breakdowns.
- **FR-SP-002**: Allow users to subscribe, upgrade, or downgrade their plans seamlessly.
- **FR-SP-003**: Integrate M-Pesa Daraja API for STK Push payments, verification, and confirmation.
- **FR-SP-004**: Generate and deliver automated payment receipts via SMS and email.
- **FR-SP-005**: Provide a comprehensive Invoice Management System for generation, tracking, and download of PDF invoices.
- **FR-SP-006**: Maintain a separate, PCI-compliant database for all payment-related data.

3.5 Administrative & Reporting

- **FR-AR-001**: Provide role-specific Analytics Dashboards for administrators and counselors to view system usage and key metrics (excluding predictive modeling).
- **FR-AR-002**: Enable administrators to manage user accounts, roles, and permissions.
- **FR-AR-003**: Allow content administrators to manage and update career resources and platform content.

3.6 Logic-Based AI Features

- **FR-AI-001**: Implement rule-based scoring and categorization for career assessments to provide deterministic career suggestions.
- **FR-AI-002**: Develop a keyword-matching and pre-defined association system for intelligent content suggestions within the Career Guidance Hub.
- **FR-AI-003**: Utilize predefined logic to automate notifications or trigger actions based on user profile completion, activity milestones, or specific events.
- **FR-AI-004**: Implement a rule-based matching algorithm for the Mentorship Matching System, based on predefined criteria (expertise, interests, goals).

4. Non-Functional Requirements (NFRs) 🔒

These requirements define the quality attributes and constraints of the system.

4.1 Performance

- **NFR-PER-001**: API response times should average less than 500ms.
- **NFR-PER-002**: Database query times should average less than 100ms.
- **NFR-PER-003**: Initial page load time should be under 2 seconds.
- **NFR-PER-004**: Mobile performance (Lighthouse score) should be consistently above 90.
- **NFR-PER-005**: Frontend caching (Service Worker) and backend API response caching for static data should be implemented.
- **NFR-PER-006**: Image assets should be optimized (e.g., WebP with fallbacks).

4.2 Security

- **NFR-SEC-001**: Adhere strictly to PCI DSS for all payment processing and data storage.
- **NFR-SEC-002**: Implement input validation, XSS protection, and CSRF protection on all user-facing forms and API endpoints.
- **NFR-SEC-003**: Use parameterized queries to prevent SQL injection.
- **NFR-SEC-004**: Sensitive payment data must be encrypted at rest (e.g., using PostgreSQL's pgcrypto).
- **NFR-SEC-005**: Enforce robust Role-Based Access Control (RBAC) at the API level.
- **NFR-SEC-006**: Implement API rate limiting to prevent abuse.
- **NFR-SEC-007**: Maintain comprehensive audit trails for all critical user and financial transactions.
- **NFR-SEC-008**: All communication will be encrypted using HTTPS/SSL.

4.3 Scalability & Reliability

- **NFR-SCL-001**: The architecture should support scaling of database capacity and compute resources on the VPS to accommodate increasing user load.
- **NFR-SCL-002**: Implement a microservices approach to allow independent scaling of components.
- **NFR-SCL-003**: System uptime should be greater than 99.9%.
- **NFR-SCL-004**: Data backups and disaster recovery procedures should be in place for the VPS.

4.4 Maintainability & Extensibility

- **NFR-MNT-001**: Codebase should be clean, modular, and well-documented with inline comments.
- **NFR-MNT-002**: Follow established coding standards (ESLint, Prettier for JS/TS; PHPStan for PHP).
- **NFR-MNT-003**: The system should be designed to easily integrate new features and third-party services.

4.5 Usability & User Experience (UX)

- **NFR-UX-001**: The platform must have a responsive design, optimized for mobile (PWA) and desktop devices.
- **NFR-UX-002**: Provide clear, intuitive navigation and user flows for all roles.
- **NFR-UX-003**: Ensure accessibility standards are met (e.g., sufficient contrast, keyboard navigation).
- **NFR-UX-004**: Provide clear feedback for user actions (e.g., loading states, success messages, error handling).

4.6 Deployment & Operations

- **NFR-DEP-001**: Deployment will be primarily on a **Virtual Private Server (VPS)**.
- **NFR-DEP-002**: Automated deployment scripts (deploy-production-complete.php) must ensure consistent and reliable deployments.
- **NFR-DEP-003**: Comprehensive logging and monitoring of system health, performance, and errors on the VPS environment.
- **NFR-DEP-004**: Environment variables (.env) must be used for configuration management.

5. Architecture Design & Plan 🏛️

Elimu Smart employs a modern, layered architecture designed for modularity, security, and performance.

5.1 High-Level Architecture

The system is composed of a React/TypeScript frontend (PWA), PHP microservices backend,

and a dual PostgreSQL database system, all deployed on a VPS. External integrations include M-Pesa and ZeptoMail.

5.1.1 Architectural Principles Applied

- **Component-Based Design**: Modular components (React, PHP services) for reusability and maintainability.
- **Security-First**: Integrated security at every layer, from frontend input validation to backend access control and database encryption.
- **Mobile-First**: Design and development prioritizing mobile responsiveness and PWA capabilities.
- **Microservices Approach**: Specialized API endpoints and services for clear separation of concerns and independent scalability.
- **Scalable Foundation**: Designed to accommodate increasing user load on a VPS and for future migration to cloud-managed services.

5.1.2 Website Use Flow Diagram & Categories

This diagram illustrates the primary user journeys and categorization of functionalities within the Elimu Smart platform. It emphasizes how theme-based access impacts the user's visual and functional experience post-authentication.

Categories of Use Flow:

- **Authentication & Profile Management**: Registration, Login, Password Reset, Profile Creation/Editing. **(Leads to theme application)**
- **Academic Guidance**: KCSE Calculator, University Predictor, KUCCPS Data Dashboard.
- **Career Exploration**: Career Assessments, Subject-to-Career Mapping, Career Guidance Hub.
- **Support & Communication**: Live Chat with Counselors, Chatbot Interaction.
- **Subscription & Payments**: Plan Selection, M-Pesa Payments, Invoice Management, Payment History.
- **Administrative & Reporting**: User Management, Content Management, Analytics Dashboards (for Admin/Counselor).
- **Community & Mentorship**: Forums, Mentorship Matching (future phases).

5.1.3 Theme-Based Access & User Experience

The platform implements a robust theme-based access system to provide a tailored user experience for each role (Student, Counselor, Administrator). Upon successful authentication, the system determines the user's role and dynamically applies the corresponding theme, affecting the visual layout, color scheme, available navigation elements, and default dashboard view.

- **Implementation**: This is achieved by dynamically adding CSS classes to the <body> element (e.g., theme-student, theme-counselor, theme-admin) which trigger distinct CSS

custom properties and component styles via Tailwind CSS. The PageRouter or a similar top-level component handles the conditional rendering of role-specific UI elements.

- **Theme Details**:
  - **Student Theme (Default User Experience)**:
    - **Primary Focus**: Career exploration, academic tools, self-guided assessments, learning resources, and direct support.
    - **Visuals**: Bright, engaging, and encouraging color palette. Primary Color: **Orange (#f97316)**. Secondary Colors: **Warm yellows and oranges**. Design Philosophy: Energetic and encouraging. User-friendly layouts with prominent calls to action for career discovery and academic tools.
    - **Dashboard View**: Central dashboard displays progress tracking, quick links to KCSE calculator, career assessments, and recent activity.
    - **Navigation**: Primary navigation focuses on "Home," "Careers," "Assessments," "KUCCPS," "Chat," "Profile," "Subscription."
    - **Accessibility**: Designed for clarity and ease of use for students.
  - **Counselor Theme (Professional & Collaborative)**:
    - **Primary Focus**: Managing student interactions, monitoring student progress, providing guidance, and accessing analytics related to their students.
    - **Visuals**: Professional, clean, and organized aesthetic, often with a slightly muted or distinct color palette compared to the student theme, promoting efficiency. Primary Color: **Yellow (#eab308)**. Secondary Colors: **Warm yellows and golds**. Design Philosophy: Professional yet approachable.
    - **Dashboard View**: Dedicated dashboard showing assigned students, active chat sessions, upcoming appointments, and aggregated statistics for their student cohort.
    - **Navigation**: Includes "My Students," "Chat," "Appointments," "Analytics," "Resources," "Profile."
    - **Tools**: Provides quick access to tools for making notes on student profiles, scheduling, and sending bulk messages to their assigned students.
  - **Administrator Theme (Comprehensive Control)**:
    - **Primary Focus**: System-level management, user administration, content oversight, platform-wide analytics, and security settings.
    - **Visuals**: Authoritative and information-dense layout, potentially with a darker or more distinct "admin" color scheme to clearly differentiate it from other roles. Primary Color: **Purple (#a855f7)**. Secondary Colors: **Lilac and purple tones**. Design Philosophy: Authoritative and sophisticated.
    - **Dashboard View**: Overview of all users, active subscriptions, system performance metrics, content management quick links, and security alerts.
    - **Navigation**: Contains "Users," "Content," "Subscriptions," "Payments," "Analytics (Global)," "Settings," "Security."
    - **Tools**: Provides comprehensive tools for user CRUD operations, content publishing, monitoring system health, and managing integrations.

5.2 Technology Stack

- **Frontend**: React 18 (TypeScript), Tailwind CSS v4, ShadCN/UI.
- **Backend**: Parse Platform (BaaS), PHP Microservices.
- **Databases**: PostgreSQL (Main: platform_users; Payment: elimu_smart_payments).
- **Real-time**: Parse Live Query (for chat).
- **Payment Gateway**: M-Pesa Daraja API.
- **Email Service**: ZeptoMail by Zoho.
- **Hosting Environment**: **Virtual Private Server (VPS)**.

5.3 Component Architecture

- **Frontend**:
  - App.tsx (Root Component, handles global state, theming, routing).
  - useAuth Hook (Centralized authentication state management, isAuthenticated, userData, authMode).
  - **Authentication Flow Components**: (LoginPage.tsx, SignupPage.tsx, RoleSelectionPage.tsx, CounselorSignup.tsx). These manage the unauthenticated user journey via AuthPageRouter.
  - **Navigation Components**: (Header.tsx for top navigation, BottomNavigation.tsx for mobile-optimized tabs, PageRouter.tsx for main content routing).
  - **Dashboard & Overview Components**: (Homepage.tsx, DashboardStats.tsx, StatsOverview.tsx, WelcomeSection.tsx).
  - **Career Guidance Components**: (SubjectToCareerMapper.tsx, CareerQuizResults.tsx for assessments, CareerGuidanceHub.tsx, CareerSpotlight.tsx).
  - **University & Placement Components**: (KuccpsPlacementTracker.tsx, UniPlacementTracker.tsx, CoursesPage.tsx, ScholarshipsPage.tsx).
  - **Communication Components**: (AskCareerCounselor.tsx, LiveChat.tsx, CounselorDashboard.tsx).
  - **Subscription & Payment Components**: (SubscriptionManager.tsx, SubscriptionPage.tsx, InvoiceManager.tsx).
  - **UI Component Library**: (/components/ui/ - leverages ShadCN/UI for reusable, accessible UI elements like buttons, forms, cards).
  - **Role-Based Theming System**: Dynamic CSS custom properties applied via document.body.classList.add('theme-admin') etc., based on user role.
- **Backend**:
  - **API Layer (/api/)**: Serves as the entry point for all frontend requests. Examples include:
    - auth.php: Handles login, signup, logout, profile management.
    - career.php: Manages career data queries, assessment submissions.
    - mpesa-callback.php: Receives M-Pesa payment confirmations.
    - subscription.php: Manages subscription plans and billing.
    - Each API endpoint uses input validation and calls relevant service layer methods.
  - **Service Layer (/services/)**: Contains the core business logic and interactions with

external services/databases. Examples:
    - MpesaService.php: Encapsulates M-Pesa API calls (STK Push, transaction status).
    - SubscriptionService.php: Manages subscription lifecycle, payment processing, invoice generation.
    - ZeptoMailService.php: Handles all email sending.
    - KuccpsPlacementService.php: Logic for fetching and calculating KUCCPS data.
    - PaymentDatabaseService.php: Dedicated service for secure payment database operations.
- **Database Architecture**:
    - **Main Database (platform_users)**: Contains core application data.
        - **Schema for platform_users:**

```
-- Table for Users (Students, Counselors, Admins)
CREATE TABLE users (
    user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20) UNIQUE,
    user_type VARCHAR(20) NOT NULL CHECK (user_type IN ('student', 'counselor',
'admin')),
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    gender VARCHAR(10),
    county VARCHAR(50),
    school_name VARCHAR(100),
    school_type VARCHAR(50),
    current_class VARCHAR(20),
    expected_kcse_year INT,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Career Assessments (e.g., Personality, Interest, Skills)
CREATE TABLE career_assessments (
    assessment_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    assessment_type VARCHAR(50) NOT NULL, -- e.g., 'personality', 'interest',
'skills'
    assessment_date TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    raw_score JSONB, -- Store detailed scores or responses as JSON
    calculated_outcome TEXT, -- Text description of the outcome
```

```
    suggested_careers TEXT[], -- Array of suggested career IDs/names based on
rules
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Career Information
CREATE TABLE careers (
    career_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    career_name VARCHAR(100) UNIQUE NOT NULL,
    description TEXT,
    required_subjects TEXT[],
    typical_salary_range VARCHAR(50),
    growth_outlook TEXT,
    related_universities UUID[], -- Array of university IDs
    related_courses UUID[], -- Array of course IDs
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for University Information
CREATE TABLE universities (
    university_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    university_name VARCHAR(150) UNIQUE NOT NULL,
    location VARCHAR(100),
    type VARCHAR(50), -- e.g., 'public', 'private'
    admission_requirements TEXT,
    website_url VARCHAR(255),
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Courses offered by Universities
CREATE TABLE courses (
    course_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    course_name VARCHAR(150) NOT NULL,
    university_id UUID NOT NULL REFERENCES universities(university_id) ON
DELETE CASCADE,
    duration_years INT,
    cluster_points_2023 NUMERIC(5,2), -- Example: 45.67
    cluster_points_2022 NUMERIC(5,2),
    required_subjects TEXT[],
    description TEXT,
```

```
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    UNIQUE (course_name, university_id)
);

-- Table for KUCCPS Placement Data (historical and real-time updates)
CREATE TABLE kuccps_placements (
    placement_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(user_id) ON DELETE SET NULL, -- Null if not
specific user placement
    student_kcse_year INT,
    university_id UUID REFERENCES universities(university_id) ON DELETE SET
NULL,
    course_id UUID REFERENCES courses(course_id) ON DELETE SET NULL,
    placement_year INT NOT NULL,
    cluster_points NUMERIC(5,2),
    status VARCHAR(50), -- e.g., 'placed', 'not_placed', 'pending'
    placement_date DATE,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Live Chat Sessions
CREATE TABLE chat_sessions (
    session_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES users(user_id) ON DELETE
CASCADE,
    counselor_id UUID REFERENCES users(user_id) ON DELETE SET NULL,
    start_time TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    end_time TIMESTAMPTZ,
    status VARCHAR(20) NOT NULL DEFAULT 'open', -- 'open', 'closed', 'pending'
    session_notes TEXT,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Chat Messages
CREATE TABLE chat_messages (
    message_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES chat_sessions(session_id) ON
DELETE CASCADE,
    sender_id UUID NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    message_text TEXT NOT NULL,
```

```
    message_time TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Progress Tracking (e.g., assessment completion, goals)
CREATE TABLE user_progress (
    progress_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    metric_type VARCHAR(50) NOT NULL, -- e.g., 'assessment_completion',
'goal_achievement'
    metric_name VARCHAR(100) NOT NULL, -- e.g., 'Personality Assessment',
'Apply to 3 Unis'
    progress_value NUMERIC(5,2), -- e.g., 0.75 for 75% completion
    status VARCHAR(20), -- e.g., 'in_progress', 'completed'
    last_updated TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for System Logs (general application events)
CREATE TABLE system_logs (
    log_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    log_level VARCHAR(10) NOT NULL, -- 'INFO', 'WARN', 'ERROR'
    message TEXT NOT NULL,
    context JSONB, -- Additional details as JSON
    timestamp TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Mentor-Mentee Relationships
CREATE TABLE mentorship_relationships (
    relationship_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    mentee_id UUID NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    mentor_id UUID NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    status VARCHAR(20) NOT NULL DEFAULT 'pending', -- 'pending', 'active',
'ended'
    start_date DATE DEFAULT CURRENT_DATE,
    end_date DATE,
    notes TEXT,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    UNIQUE (mentee_id, mentor_id)
);

-- Table for Community Forums - Topics
CREATE TABLE forum_topics (
```

```
    topic_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    content TEXT,
    category VARCHAR(50),
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Community Forums - Posts/Replies
CREATE TABLE forum_posts (
    post_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    topic_id UUID NOT NULL REFERENCES forum_topics(topic_id) ON DELETE
CASCADE,
    user_id UUID NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    parent_post_id UUID REFERENCES forum_posts(post_id) ON DELETE
CASCADE, -- For replies
    content TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
```

- **Payment Database (elimu_smart_payments)**: Isolated database for PCI-compliant payment data.
  - **Schema for elimu_smart_payments:**

```
-- Table for Payment Intents/Initiations (M-Pesa STK Push requests)
CREATE TABLE payment_intents (
    intent_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL, -- User ID from main DB, but no FK to maintain
separation
    transaction_type VARCHAR(50) NOT NULL, -- e.g., 'buy_subscription',
'counseling_session'
    amount NUMERIC(10, 2) NOT NULL,
    currency VARCHAR(10) DEFAULT 'KES',
    mpesa_checkout_request_id VARCHAR(255) UNIQUE, -- M-Pesa's unique
request ID
    mpesa_phone_number VARCHAR(20),
    status VARCHAR(50) NOT NULL DEFAULT 'pending', -- 'pending', 'completed',
'failed', 'cancelled'
    payload JSONB, -- Raw request/response data for debugging
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
```

```
);

-- Table for Completed Payment Records
CREATE TABLE payments (
    payment_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    intent_id UUID REFERENCES payment_intents(intent_id) ON DELETE SET NULL,
    user_id UUID NOT NULL, -- User ID from main DB, no FK
    amount NUMERIC(10, 2) NOT NULL,
    currency VARCHAR(10) DEFAULT 'KES',
    mpesa_receipt_number VARCHAR(50) UNIQUE, -- M-Pesa transaction ID
    transaction_date TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    payment_status VARCHAR(50) NOT NULL, -- 'success', 'failed', 'reversed'
    provider_response JSONB, -- Raw response from M-Pesa callback
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Subscription Plans (e.g., Free, Premium, Pro)
CREATE TABLE subscription_plans (
    plan_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    plan_name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT,
    monthly_price NUMERIC(10, 2) NOT NULL,
    annual_price NUMERIC(10, 2),
    features TEXT[], -- List of features included
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for User Subscriptions
CREATE TABLE subscriptions (
    subscription_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL, -- User ID from main DB, no FK
    plan_id UUID NOT NULL REFERENCES subscription_plans(plan_id) ON DELETE
RESTRICT,
    status VARCHAR(20) NOT NULL, -- 'active', 'cancelled', 'expired'
    start_date TIMESTAMPTZ NOT NULL,
    end_date TIMESTAMPTZ, -- For fixed-term or expired subscriptions
    next_billing_date TIMESTAMPTZ,
    auto_renew BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
```

```
-- Table for Invoice Generation and Management
CREATE TABLE invoices (
    invoice_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL, -- User ID from main DB, no FK
    subscription_id UUID REFERENCES subscriptions(subscription_id) ON DELETE
SET NULL,
    invoice_number VARCHAR(50) UNIQUE NOT NULL,
    invoice_date TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    due_date TIMESTAMPTZ,
    amount_due NUMERIC(10, 2) NOT NULL,
    currency VARCHAR(10) DEFAULT 'KES',
    status VARCHAR(20) NOT NULL, -- 'draft', 'issued', 'paid', 'overdue', 'cancelled'
    payment_id UUID REFERENCES payments(payment_id) ON DELETE SET NULL,
-- Link to payment record
    pdf_url VARCHAR(255), -- Path or URL to stored PDF
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Detailed Invoice Line Items
CREATE TABLE invoice_line_items (
    line_item_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    invoice_id UUID NOT NULL REFERENCES invoices(invoice_id) ON DELETE
CASCADE,
    description TEXT NOT NULL,
    quantity INT NOT NULL,
    unit_price NUMERIC(10, 2) NOT NULL,
    total_price NUMERIC(10, 2) NOT NULL,
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Table for Audit Trail for Payment Data Access (for PCI Compliance)
CREATE TABLE pci_access_log (
    log_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID, -- User ID from main DB, no FK
    action VARCHAR(50) NOT NULL, -- 'read', 'write', 'delete'
    table_name VARCHAR(50) NOT NULL,
    record_id UUID, -- ID of the record accessed/modified
    ip_address INET,
    timestamp TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
```

```
-- Table for Storing Raw Webhook Events from Payment Providers
CREATE TABLE webhook_events (
    event_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    provider VARCHAR(50) NOT NULL, -- e.g., 'M-Pesa'
    event_type VARCHAR(100) NOT NULL,
    payload JSONB NOT NULL, -- Raw webhook payload
    processed BOOLEAN DEFAULT FALSE,
    processing_error TEXT,
    received_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
```

5.4 Data Flow

- **User Authentication Flow**: User sends login request to /api/auth.php -> Backend validates credentials against platform_users -> Generates JWT token -> Frontend stores token -> Subsequent API requests include token for validation -> Backend applies Role-Based Access Control (RBAC) and signals user role to frontend -> Frontend applies the **appropriate theme** and renders the role-specific dashboard.
- **API Request Flow**: Frontend component makes API call (e.g., to /api/career.php) -> apiWrapper.ts intercepts request -> In demo mode, mock data is returned; in production, request forwarded to PHP API -> PHP endpoint calls relevant service method (e.g., CareerService) -> Service interacts with platform_users or elimu_smart_payments database -> Response returned to frontend -> Frontend updates state and UI.
- **Real-time Chat Flow**: Frontend establishes WebSocket connection to Parse Live Query Server -> Messages sent from one client are broadcast through the server to relevant subscribed clients in real-time.
- **Payment Flow**: User initiates payment on frontend (SubscriptionManager.tsx) -> Calls POST /api/subscription.php -> Backend SubscriptionService calls MpesaService.initiateSTKPush() -> M-Pesa sends STK Push to user phone -> User confirms payment -> M-Pesa sends callback to /api/mpesa-callback.php -> Backend updates payment_transactions in elimu_smart_payments -> Triggers invoice generation (SubscriptionService.generateInvoice()) and email notification (EmailService.sendInvoice()) -> Frontend updates subscription status.

6. Development Process ⚙️

The development will follow a phased, iterative approach, adhering to agile principles for flexibility and continuous improvement.

6.1 Methodology

- **Phased Development**: Structured into distinct phases (MVP, Core Features, Expansion, Scale & Optimize).
- **Iterative Cycles**: Within each phase, development will occur in shorter sprints, allowing

for continuous feedback and refinement.
- **Version Control**: Git with a feature branching workflow (main for production, develop for integration, feature branches for new development).
- **Code Review**: All code changes require peer review before merging.

6.2 Testing Strategy

- **Unit Testing**: Automated tests for individual components and service methods (React components, PHP service classes).
- **Integration Testing**: Verify interactions between frontend and backend APIs, and between backend services and databases.
- **End-to-End Testing**: Simulate user journeys through the entire application flow.
- **Performance Testing**: Regular load testing to ensure the system performs under expected traffic, especially on the VPS.
- **Security Testing**: Regular vulnerability scanning and penetration testing.
- **Manual Testing**: Comprehensive UAT (User Acceptance Testing) by stakeholders.

6.3 Continuous Integration/Continuous Delivery (CI/CD)

- Automated build processes for the React frontend.
- Automated execution of unit and integration tests upon code commit.
- Automated deployment scripts (php deploy-production-complete.php) for reliable releases to the **VPS**.
- Monitoring of deployment success/failure.

6.4 Documentation Standards

- **Inline Comments**: Extensive comments explaining complex logic, algorithms, and function headers.
- **API Documentation**: Clear specifications for all backend API endpoints.
- **System Architecture Documentation**: This living document and related architectural diagrams.
- **ReadMe Files**: For setup, deployment, and contribution guidelines.

6.5 Agile Sprint Plan 🗓️

This plan outlines a series of 2-week sprints to implement the Elimu Smart platform, following an agile methodology. Each sprint focuses on delivering a set of functional, tested features that contribute to the overall product vision.

Sprint Duration & Structure

- **Sprint Length**: 2 weeks (10 working days).
- **Sprint Team**: Assumes a cross-functional team (Frontend, Backend, Full-Stack, QA).
- **Sprint Ceremonies**:
    - **Sprint Planning (Day 1, 4 hours)**: Team selects backlog items for the sprint and plans how to accomplish them.

    - **Daily Stand-ups (Daily, 15 minutes)**: Quick sync on progress, plans for the day, and impediments.
    - **Sprint Review (Day 10, 2 hours)**: Demo completed work to stakeholders, gather feedback.
    - **Sprint Retrospective (Day 10, 1 hour)**: Team reflects on the sprint and identifies improvements.

Phase 1: MVP Launch (Approx. 3 Sprints)

This phase focuses on establishing the core platform infrastructure and essential user flows.

- **Sprint 1: Core User Authentication & Profile Foundation**
    - **Objective**: Establish secure user registration, login, and basic profile creation.
    - **Key Deliverables**:
        - **Backend**: FR-UM-001, FR-UM-003 (initial APIs for email/phone registration, password reset). Parse User schema setup.
        - **Frontend**: FR-UM-001 (Registration/Login UI for all roles), FR-UM-005 (Basic role-based UI display, **initial theme application logic**), basic profile creation wizard (FR-UM-004).
        - **DevOps**: Initial **VPS** setup for development environment.
    - **Testing Focus**: User registration, login, password reset.
- **Sprint 2: KCSE Calculator & Basic Profile Persistence**
    - **Objective**: Implement core academic tools and persist basic user data.
    - **Key Deliverables**:
        - **Backend**: FR-UM-004 (API for saving/updating basic profile info), initial KUCCPS data storage structure.
        - **Frontend**: FR-ACG-001 (Interactive KCSE Grade Calculator UI with real-time calculations), FR-UM-004 (Profile editing forms).
        - **DevOps**: Configure PostgreSQL databases (platform_users) on **VPS**.
    - **Testing Focus**: KCSE calculator accuracy, profile data saving.
- **Sprint 3: M-Pesa Integration & Subscription Foundation**
    - **Objective**: Enable payment processing and initial subscription management.
    - **Key Deliverables**:
        - **Backend**: FR-SP-003 (M-Pesa STK Push initiation API), FR-SP-006 (Separate payment database setup on **VPS**).
        - **Frontend**: FR-SP-002 (UI for initiating subscription), basic subscription plan display (FR-SP-001).
        - **Integrations**: Initial ZeptoMail setup for basic transactional emails (FR-CS-004).
    - **Testing Focus**: M-Pesa payment flow, subscription status updates.

Phase 2: Core Features Enhancement (Approx. 4 Sprints)

This phase enhances the guidance and communication aspects, laying the groundwork for more intelligent features.

- **Sprint 4: Career Assessment Engine & Rule-Based AI** 🧠

    - **Objective**: Deploy the career assessment and its rule-based AI for suggestions.
    - **Key Deliverables**:
        - **Backend**: FR-ACG-003 (Scoring logic for assessments), FR-AI-001 (Rule-based career suggestion logic).
        - **Frontend**: FR-ACG-003 (Career Assessment UI), FR-AI-001 (Display of rule-based career suggestions).
    - **Testing Focus**: Assessment logic, accuracy of career suggestions.
- **Sprint 5: KUCCPS Data & Subject Mapping**
    - **Objective**: Integrate comprehensive KUCCPS data and enable subject-to-career mapping.
    - **Key Deliverables**:
        - **Backend**: FR-ACG-006 (API for full KUCCPS data retrieval), FR-ACG-002 (University Predictor logic with KUCCPS data).
        - **Frontend**: FR-ACG-006 (KUCCPS Dashboard UI), FR-ACG-004 (Subject-to-Career Mapping display).
    - **Testing Focus**: KUCCPS data accuracy, mapping logic.
- **Sprint 6: Live Chat Real-time & Progress Tracking**
    - **Objective**: Establish real-time live chat and empower students with progress tracking.
    - **Key Deliverables**:
        - **Backend**: FR-CS-001 (WebSocket integration for live chat), FR-ACG-008 (Progress tracking data storage).
        - **Frontend**: FR-CS-001 (Live Chat UI for student-counselor), FR-ACG-008 (Progress Tracking Dashboard UI).
        - **DevOps**: Configure Parse Live Query server on **VPS**.
    - **Testing Focus**: Real-time message delivery, progress tracking.
- **Sprint 7: Content Suggestion AI & Automated Triggers**
    - **Objective**: Deliver intelligent content and automate notifications.
    - **Key Deliverables**:
        - **Backend**: FR-AI-002 (Keyword-matching logic for content suggestions), FR-AI-003 (Predefined logic for automated notifications/triggers).
        - **Frontend**: FR-ACG-005 (Career Guidance Hub with FR-AI-002 intelligent content suggestions).
    - **Testing Focus**: Content relevance, notification triggers.

Phase 3: Platform Expansion & Security (Approx. 4 Sprints)

This phase expands the platform's reach and strengthens its security and administrative capabilities.

- **Sprint 8: Coaching Platform (Booking & Profiles)**
    - **Objective**: Launch the coaching platform's core features.
    - **Key Deliverables**:
        - **Backend**: APIs for coach profiles, appointment booking logic.

        - **Frontend**: Coach directory, appointment booking UI.
    - **Testing Focus**: Booking flow, profile management.
- **Sprint 9: Enhanced Analytics & Mentorship Matching AI**
    - **Objective**: Provide deeper insights and introduce mentorship.
    - **Key Deliverables**:
        - **Backend**: FR-AR-001 (Expanded data collection for analytics), FR-AI-004 (Rule-based matching algorithm for mentorship).
        - **Frontend**: FR-AR-001 (Advanced Analytics Dashboards), FR-AI-004 (Mentorship Matching UI).
    - **Testing Focus**: Analytics accuracy, mentorship matching logic.
- **Sprint 10: Community Forums & Parent Portal**
    - **Objective**: Foster community and provide parent visibility.
    - **Key Deliverables**:
        - **Backend**: Database schema and APIs for forums, topics, posts. Parent/guardian access control.
        - **Frontend**: Community Forums UI, Parent/Guardian Portal UI (FR-CS-003).
    - **Testing Focus**: Forum functionality, parent portal data display.
- **Sprint 11: API Gateway & Advanced Security**
    - **Objective**: Implement an API Gateway and bolster security.
    - **Key Deliverables**:
        - **DevOps**: API Gateway setup (FR-DEP-001, FR-DEP-002) on **VPS** (e.g., Nginx as reverse proxy with routing).
        - **Security**: NFR-SEC-005, NFR-SEC-006 (Centralized rate limiting, enhanced RBAC through gateway), NFR-SEC-008 (Review HTTPS/SSL setup).
        - **Backend**: FR-UM-001 (Two-Factor Authentication (2FA) for critical roles).
    - **Testing Focus**: API routing, rate limiting, 2FA.

Phase 4: Scale & Optimize (Approx. 4 Sprints + Ongoing)

This phase focuses on long-term scalability, global reach, and native mobile presence.

- **Sprint 12: Multi-Language & Initial Partnerships**
    - **Objective**: Prepare for wider adoption and external collaborations.
    - **Key Deliverables**:
        - **Frontend**: Internationalization (i18n) framework.
        - **Backend**: Multi-language string management.
        - **Content**: Initial Swahili localization.
        - **Backend/Frontend**: Initial Institutional Partnership Integrations.
    - **Testing Focus**: Language switching, partner data integration.
- **Sprint 13: Native Mobile App (Core Features) 📱 & Basic Career Guidance Chatbot 💬**
    - **Objective**: Develop the foundational native mobile application and implement the rule-based chatbot.
    - **Key Deliverables**:

        - **Mobile App**: React Native project setup, porting of core components (Authentication, Dashboard, KCSE Calculator, Subscription), offline capability, push notifications.
        - **Mobile App**: M-Pesa integration within native app.
        - **Backend**: FR-CS-002 (Rule-based chatbot logic and backend).
        - **Frontend**: FR-CS-002 (Chatbot UI for web platform).
    - **Testing Focus**: Native app functionality, offline mode, notifications, chatbot responses.
- **Sprint 14: Cloud Migration Planning & CDN**
    - **Objective**: Plan for future cloud migration and implement CDN for performance.
    - **Key Deliverables**:
        - **DevOps**: Comprehensive Cloud Infrastructure Migration plan (from **VPS**).
        - **DevOps**: Global CDN implementation for static assets.
        - **Architecture**: Research and decide on containerization strategy (e.g., Docker for PHP microservices).
    - **Testing Focus**: CDN asset loading, initial cloud readiness assessment.
- **Sprint 15+: Scaling, Expansion, & Compliance**
    - **Objective**: Implement advanced scaling, expand internationally, and ensure robust compliance.
    - **Key Deliverables**:
        - **DevOps**: Database replication, load balancing, auto-scaling setup (if migrated to cloud).
        - **DevOps**: Disaster Recovery Plan development and testing.
        - **Backend**: Payment Gateway Diversification for new regions.
        - **Legal**: Compliance features (e.g., GDPR/CCPA readiness).
        - **Operations**: Regular security audits and penetration testing.
    - **Testing Focus**: High availability, international functionality, compliance.

7. Deployment Strategy 💻

The platform is designed for efficient deployment and operation on a **Virtual Private Server (VPS)**.

7.1 Initial Deployment (VPS)

- **Server Setup**: Provision a Linux-based VPS (e.g., Ubuntu, CentOS).
- **Software Installation**: Install Apache/Nginx (web server), PHP 7.4+ (or later compatible version), PostgreSQL, and Parse Platform server components.
- **Database Setup**: Execute setup-production-databases.php to create and configure both platform_users and elimu_smart_payments databases.
- **Code Deployment**: Use deploy-production-complete.php script to deploy the React build (static files) and PHP backend files to the web server directory.
- **Configuration**: Set up environment variables (.env) for database credentials, M-Pesa API keys, ZeptoMail credentials, and other configurations.

- **SSL/TLS**: Install and configure an SSL certificate (e.g., Let's Encrypt) for HTTPS.
- **DNS Configuration**: Point domain DNS records to the VPS IP address.

7.2 Monitoring & Maintenance (VPS)

- **System Monitoring**: Implement tools (e.g., Nagios, Zabbix, or basic shell scripts) to monitor CPU, memory, disk usage, and network traffic on the VPS.
- **Application Monitoring**: Monitor API response times, database query performance, and error rates using custom logging and analysis.
- **Logs Management**: Centralize application, web server, and database logs for easier debugging and auditing.
- **Regular Backups**: Schedule automated backups of both PostgreSQL databases and application code.
- **Security Updates**: Regularly apply OS, PHP, PostgreSQL, and Parse Platform security patches.

7.3 Future Scaling & Cloud Migration (Post-VPS)

While initially on a VPS, the architecture is designed to support a future migration to a more horizontally scalable cloud infrastructure (e.g., AWS, Azure, GCP). This will involve:

- **Containerization**: Dockerizing the PHP microservices and React application to facilitate easier deployment across multiple instances.
- **Load Balancing**: Implementing a load balancer to distribute traffic across multiple VPS instances (if scaling vertically on VPS becomes insufficient) or eventually cloud instances.
- **Managed Database Services**: Transitioning to cloud-managed PostgreSQL services for easier replication, failover, and scaling.
- **Cloud CDN**: Implementing a Content Delivery Network for global content delivery optimization.

8. Key Considerations & Constraints 🛑

- **Logic-Based AI**: All AI capabilities (career matching, chatbot, content suggestions, mentorship matching) will be **rule-driven** and deterministic. There will be no implementation of complex machine learning algorithms for predictive modeling, statistical inference, or pattern recognition that typically require large datasets and training.
- **No Blockchain**: Blockchain technology is explicitly out of scope for all aspects, including credentials and payments.
- **PCI Compliance**: Non-negotiable adherence to PCI DSS for payment data handling, necessitating the dual database architecture and strict security protocols.
- **Mobile-First Design (PWA)**: The primary mobile experience will be delivered via a Progressive Web App. Native mobile app development is a lower priority and will be considered in later phases.
- **Shared Hosting Compatibility**: While deployed on a VPS, the backend leverages PHP and Parse Platform which inherently supports setups compatible with traditional shared

hosting environments, ensuring flexibility.

9. Development Roadmap 🗺️

This roadmap outlines the phased development plan, ensuring a structured progression of features.

Phase 1: MVP Launch (Review & Refine - Months 0-3)

- **Objective**: Verify and refine core user authentication, KCSE calculator, basic profiles, and M-Pesa/subscription foundations.
- **Key Deliverables**: Functional multi-option registration, SMS OTP, password reset, interactive KCSE calculator, basic student profile CRUD, M-Pesa STK push integration, basic subscription management UI, initial PWA setup.
- **Technical Focus**: Solidify core PHP microservices, React components, dual PostgreSQL setup on **VPS**.

Phase 2: Core Features Enhancement (Months 3-6)

- **Objective**: Implement core career guidance and communication functionalities.
- **Key Deliverables**: Complete Career Assessment Engine with scoring logic (rule-based AI), enhanced KUCCPS Data Integration and Dashboards, a functional Progress Tracking Dashboard, and Real-time Live Chat via WebSockets.
- **Technical Focus**: Expand backend services for assessments and KUCCPS, integrate WebSocket server, refine rule sets for logic-based AI.

Phase 3: Platform Expansion & Security (Months 6-9)

- **Objective**: Introduce advanced communication, community, and security features.
- **Key Deliverables**: Coaching Platform & Booking System (with video consultation), Advanced Analytics & Reporting dashboards (non-ML), **Mentorship Matching System (logic-based AI)**, Community Forums, Parent/Guardian Portal, API Gateway implementation, Two-Factor Authentication (2FA), Multi-language support (Swahili), initial Institutional Partnership Integrations.
- **Technical Focus**: Implement complex business logic, integrate third-party APIs (WebRTC), strengthen security measures, begin i18n implementation.

Phase 4: Scale & Optimize (Months 9-12+)

- **Objective**: Focus on large-scale growth strategies, including native mobile apps and potential infrastructure migration.
- **Key Deliverables**: **Native Mobile App Development (React Native)** for core features with offline capabilities and push notifications, **Basic Career Guidance Chatbot (Rule-Based AI)**, planning and execution for **Cloud Infrastructure Migration** from VPS (if necessary for further horizontal scaling), Global CDN implementation, International Market Expansion strategy, Enterprise-grade High Availability setup, and Advanced

Compliance Features (e.g., GDPR/CCPA readiness).

- **Technical Focus**: React Native development, cloud architecture design, DevOps automation for scalable infrastructure, advanced security and compliance audits.