1. Project Overview

The Web Platform is a comprehensive, multi-functional application built on the CodeIgniter 4 framework. It serves as a portal for registered users to access a suite of powerful digital services. The platform is designed with a modular architecture, featuring a robust user authentication system, an account management dashboard with an integrated balance and payment system, and an administrative panel for user oversight.

The core services offered include:

- **Gemini AI Studio:** An advanced interface allowing users to interact with Google's Gemini AI, featuring text and multimedia prompts, conversational memory, and context-aware responses.
- **Cryptocurrency Data Service:** A tool for querying real-time balance and transaction data for Bitcoin (BTC) and Litecoin (LTC) addresses.

The application integrates with several third-party APIs, including Paystack for secure payments, Google reCAPTCHA for spam prevention, and the respective blockchain and AI service endpoints.

2. Core Technologies

- **Backend:** PHP 8.1+, CodeIgniter 4
- **Frontend:** Bootstrap 5, JavaScript, HTML5, CSS3
- **Database:** MySQL (via MySQLi driver)
- **Key Libraries:** Parsedown (for Markdown rendering), TinyMCE (for rich text editing), Kint (for debugging)
- **Development Tooling:** Composer, PHPUnit

3. Installation and Setup

1. **Prerequisites:** Ensure you have PHP 8.1+, Composer, and a MySQL database server installed.
2. **Clone Repository:** Clone the project to your local machine.
3. **Install Dependencies:** Run `composer install` to download the required PHP packages.
4. **Configure Environment:**
    - Copy the env file to a new file named `.env`.
    - Open `.env` and configure the following variables:
        - `CI_ENVIRONMENT`: Set to `development`.
        - `app.baseURL`: The base URL of your application (e.g., `http://localhost:8080/`).
        - **Database:** Configure `database.default.hostname`, `database.default.database`, `database.default.username`, and `database.default.password`.
        - **API Keys:** Provide your secret keys for `PAYSTACK_SECRET_KEY`, `GEMINI_API_KEY`, `recaptcha.siteKey`, and `recaptcha.secretKey`.

    - **Email:** Set up your SMTP server details under the `email.*` variables for sending verification and password reset emails.
5. **Install Frontend Assets:** Download the TinyMCE community edition and place its contents in the `public/assets/tinymce/` directory to enable the rich text editor.
6. **Database Migration:** Run the migrations to create the necessary database tables:
   ```
   php spark migrate
   ```
7. **Run the Application:** Start the development server:
   ```
   php spark serve
   ```

4. Application Architecture

The project strictly follows the **Model-View-Controller (MVC)** architectural pattern, extended with a **Service layer** to adhere to modern software design principles and the project's internal coding standards (`clinerules.md`).

- **`app/Controllers`:** Controllers are the entry point for user requests. They are kept lean and are responsible for orchestrating the application flow, validating input, and calling upon models and services to perform business logic. They do not contain database queries or complex logic. The `BaseController` is used to handle logic common to all pages, such as checking for cookie consent status.
- **`app/Models`:** Models are responsible for all database interactions. They use CodeIgniter's Query Builder and Entities to ensure data is accessed securely and consistently.
- **`app/Entities`:** Entities are object-oriented representations of database table rows, allowing for clean, typed interaction with data.
- **`app/Views`:** Views handle all presentation logic. They are organized into subdirectories named after their corresponding controller (e.g., `app/Views/auth/`). All dynamic data is escaped using `esc()` to prevent XSS attacks. The UI is built on the **Bootstrap 5** framework and utilizes a main layout (`app/Views/layouts/default.php`) for consistency. Reusable components, such as a custom Bootstrap 5 pagination view, flash messages, and the site-wide cookie consent banner, are managed as partial views to ensure a uniform user experience.
- **`app/Libraries` (Services):** This directory contains the core business logic. Classes like `PaystackService`, `GeminiService`, `CryptoService`, and the sophisticated `MemoryService` encapsulate interactions with external APIs and complex application logic, making them reusable and testable.
- **`app/Helpers`:** This directory contains simple, stateless procedural functions that are globally available. For example, `cookie_consent_helper.php` provides a function to check a user's cookie consent status from anywhere in the application.
- **`app/Config`:** This directory centralizes all application configuration, including a custom `AGI.php` file that fine-tunes the behavior of the AI's memory system.
- **`app/Filters`:** These classes act as middleware to protect routes. `AuthFilter` ensures a user is logged in, while `BalanceFilter` checks for sufficient funds before allowing access to paid services.

- **`app/Database`:** Contains all database migrations, which define the application's schema in a version-controlled manner.

5. Database Schema

The database schema is defined by the following migration files and tables:

- **`users`**: Stores user credentials, profile information, account balance, administrative status, and tokens for email verification and password resets.
- **`payments`**: A log of all payment transactions made through Paystack, including the amount, status, reference, and the full API response for auditing.
- **`prompts`**: Enables users to save and reuse frequently used AI prompts, linking them to their user ID.
- **`interactions`**: The core of the AI's long-term memory. Each row represents a single conversational turn (user input and AI output), storing the raw text, relevance score, vector embedding, and keywords.
- **`entities`**: Acts as a knowledge graph for the AI's memory. It tracks unique concepts (entities/keywords), their relationships, and links them back to the interactions where they were mentioned.
- **`user_settings`**: Stores user-specific preferences, such as the enabled/disabled state of the AI's Assistant Mode, ensuring a consistent user experience across sessions.
- **`campaigns`**: Stores reusable email campaign templates, including their subject and body, for administrative use.

6. Key Features and Modules

6.1. Authentication (`AuthController`)

- **Registration:** A secure, multi-step process that includes form validation, reCAPTCHA verification, creation of a user record with a default starting balance, and dispatching an email with a unique verification token.
- **Login & Logout:** Standard email/password authentication that verifies credentials against the hashed password in the database. It also checks for email verification status before creating a session.
- **Password Reset:** A secure, token-based flow for users to reset forgotten passwords.
- **Security:** Implements `password_hash()` for secure password storage and is protected by the `AuthFilter` on all sensitive routes.

6.2. User Dashboard & Account Management (`HomeController`, `AccountController`)

- **Dashboard (`/home`):** Provides a personalized welcome, displays the current account balance, and offers quick links to the primary application services.
- **Account Page (`/account`):** Shows detailed profile information and a paginated history of all payment transactions, including status and reference numbers.

6.3. Admin Panel (`AdminController`, `CampaignController`)

- **User Management:** Provides a paginated and searchable list of all registered users.
- **Financial Oversight:** Displays the total aggregated balance of all users.

- **Balance Adjustment:** Allows administrators to manually deposit or withdraw funds from a user's account. All calculations use PHP's `bcmath` extension for arbitrary-precision arithmetic to prevent floating-point inaccuracies.
- **User Deletion:** Provides functionality to delete users, with a safeguard to prevent an admin from deleting their own account.
- **Email Campaigns (`CampaignController`):** An admin-only feature to compose and send mass emails to all registered users. The system uses a dynamic email template to personalize greetings (e.g., using the user's name). Administrators can also save, load, and delete campaign messages as templates for future use, streamlining the outreach process.

6.4. Payment System (`PaymentsController`, `PaystackService`)

- **Initiation:** The user submits an amount to deposit. The system creates a `pending` payment record and redirects the user to the secure Paystack payment gateway.
- **Verification:** Upon the user's return from Paystack, the application verifies the transaction status with the Paystack API using a unique reference.
- **Balance Update:** If the payment is successful, the user's account balance is credited with the deposited amount using a precise `bcadd` operation.

6.5. Services

A. Crypto Data Service (`CryptoController`, `CryptoService`)

- **Access Control:** Protected by the `BalanceFilter`, ensuring users have a sufficient balance before making a query.
- **Functionality:** Users can query for the balance or transaction history of any public Bitcoin or Litecoin address.
- **API Integration:** The `CryptoService` abstracts the calls to third-party blockchain explorers (`blockchain.info` and `api.blockchair.com`).
- **Billing:** A small, fixed fee is deducted from the user's balance for each successful query.

B. Gemini AI Studio (`GeminiController`, `MemoryService`, `EmbeddingService`)

- **Rich Text Prompting:** The prompt input area is a rich text editor powered by a self-hosted instance of **TinyMCE**. This allows users to format their prompts with headings, bold text, lists, and links, sending structured HTML directly to the Gemini AI. The AI understands this structure, allowing for more nuanced and context-aware requests.
- **Multimedia Prompts:** Users can submit text prompts along with various media files (images, audio, video, PDFs) to the Gemini API.
- **Assistant Mode (Conversational Memory):** A highly advanced feature powered by `MemoryService`.
  - **Context Retrieval:** Before sending a prompt, the service performs a **hybrid search** of past interactions. It combines a **vector search** (for semantic

similarity) with a **keyword search** (for lexical relevance) to retrieve the most relevant memories.
    - **Prompt Augmentation:** This retrieved context is prepended to the user's current prompt, providing the AI with a rich conversational history.
    - **Memory Update:** After receiving a response, the system "learns" by rewarding the relevance of the context that was used, decaying the score of unused memories, and saving the new interaction with its own vector embedding for future recall.
    - **Persistent Setting:** The user's choice to enable or disable Assistant Mode is saved to their profile, providing a consistent experience across sessions.
- **Token-Based Billing:** The system calculates the exact cost of each query based on the number of input and output tokens, converts the price from USD to KES, and deducts the precise amount from the user's balance.
- **Prompt Management:** Provides a UI for users to save, load, and delete their favorite prompts.
- **Rich Output:** AI-generated Markdown is parsed into clean HTML for an enhanced user experience.

6.6. Public-Facing Pages & SEO

Beyond the core authenticated services, the platform includes several public-facing pages to engage users and improve search engine visibility.

- **Static Pages (`HomeController`):** Manages the main landing page, terms of service, and privacy policy.
- **Marketing Pages (`GeminiController`, `CryptoController`):** Both core services feature a dedicated public landing page (`/ai-studio` and `/crypto-query`) that explains the tool's benefits and encourages user registration.
- **Contact & Portfolio (`ContactController`, `PortfolioController`):** Includes a professional portfolio and a secure contact form with reCAPTCHA integration for inquiries.
- **Dynamic Sitemap (`SitemapController`):** Automatically generates a `sitemap.xml` file based on the application's named routes. This ensures that all public pages are properly indexed by search engines, enhancing SEO.

7. Documentation and Best Practices (`clinerules.md`)

The project is governed by a strict set of internal rules documented in `.clinerules/clinerules.md`, which ensures a high standard of code quality, security, and maintainability.

- **Code Standards:** All code is PSR-12 compliant and uses modern PHP features like strict typing.
- **Security:** Security is paramount, with key features including:
    - Global CSRF protection.
    - Strict output escaping (`esc()`) in all views to prevent XSS.

    - Exclusive use of the Query Builder or Entities to prevent SQL injection.
    - Route protection via Filters.
- **Routing:** The application uses named routes exclusively (`url_to()`), making URLs easy to manage and update.
- **PHPDoc:** Comprehensive PHPDoc blocks are mandatory for all classes, methods, and properties, ensuring the codebase is self-documenting and IDE-friendly.
- **Post/Redirect/Get (PRG) Pattern:** All form submissions are handled using the PRG pattern. Controller actions process data, set flash messages, and then redirect, preventing duplicate form submissions.
- **Centralized Configuration:** All environment-specific variables are stored in the `.env` file, keeping sensitive credentials out of version control.