# **Email Writing Assistant: Full-Stack Application Development Master Prompt**

This document outlines the complete scope and requirements for building a robust Email Writing Assistant. The application will leverage AI for email generation, provide flexible LLM configuration, implement secure user authentication, and feature a dedicated settings page, all built with a React frontend, PHP backend, and MySQL/MariaDB database, optimized for HostGator deployment.

## **Project Overview**

The Email Writing Assistant will be a web application that helps users transform their raw thoughts into polished, context-aware emails using AI. It will support various tones and allow users to provide conversational context. Key features include:

* **AI-Powered Email Generation:** Convert user thoughts into well-structured emails.
* **Flexible LLM Integration:** Support for local (Ollama) and cloud-based LLMs (OpenAI, Claude, Gemini, Groq, Cohere).
* **User Authentication:** Secure user registration, login, and SSO integration.
* **Personalized Settings:** A dedicated page for managing LLM configurations and other user preferences.
* **Responsive User Interface:** An intuitive and adaptive frontend experience.
* **Robust Backend:** A PHP-based API for handling business logic and data persistence.
* **Relational Database:** MySQL/MariaDB for structured data storage.
* **Comprehensive Documentation:** Detailed setup and deployment instructions.

## **Core Components and Requirements**

### **1. Frontend (React)**

The frontend will be built using React, leveraging the provided email-writer-app.tsx as a starting point. It must be fully responsive and provide a seamless user experience.

**Key Requirements:**

* **Email Generation Interface:**
  + Utilize the existing email-writer-app.tsx structure for raw thoughts input, tone selection, and context email input.
  + Display generated emails clearly.
  + Implement copy-to-clipboard functionality.
  + Add a loading indicator during email generation.
* **Authentication Flow:**
  + **Login Page:** Allow users to log in with email/password or via SSO options (e.g., Google, Facebook – conceptual integration, actual implementation might be simplified for this prompt).
  + **Registration Page:** Allow new users to register with email and password.
  + **Dashboard/Home Page:** After successful login, redirect users to the main email writing interface.
* **Settings Page:**
  + A dedicated page accessible via a navigation link (e.g., in a header or sidebar).
  + **LLM Configuration:**
    - Radio buttons or a dropdown to select the preferred LLM provider (Ollama, OpenAI, Claude, Cohere, Gemini, Groq, etc.).
    - Input fields for API keys for each selected cloud-based LLM.
    - Input field for Ollama local endpoint URL (e.g., http://localhost:11434).
    - A "Save Settings" button to persist the user's choices.
    - Display current active LLM provider.
* **Navigation:** Clear navigation between the main email writer, login/registration, and settings pages.
* **Styling:** Use Tailwind CSS for a modern and responsive design. Ensure consistent look and feel across all pages.
* **API Interaction:** All frontend actions (login, registration, settings updates, email generation) must communicate with the PHP backend via RESTful API calls.

### **2. Backend (PHP / PHP Framework)**

The backend will be developed in PHP, preferably using a lightweight framework or plain PHP for simplicity and HostGator compatibility. It will serve as the API layer for the React frontend and handle all business logic, authentication, and LLM interactions.

**Key Requirements:**

* **API Endpoints:**
  + **Authentication:**
    - POST /api/register: User registration (email, password).
    - POST /api/login: User login (email, password).
    - POST /api/sso/google (or similar): Handle SSO callbacks/authentication.
    - POST /api/logout: Invalidate JWT.
  + **User Management:**
    - GET /api/user/me: Get current user details.
  + **LLM Configuration:**
    - GET /api/settings/llm: Retrieve current user LLM settings.
    - POST /api/settings/llm: Update user LLM settings (selected provider, API keys, Ollama URL).
  + **Email Generation:**
    - POST /api/generate-email: Accepts rawThoughts, tone, contextEmail. This endpoint will dynamically call the appropriate LLM based on the user's saved settings.
* **Authentication & Authorization:**
  + **JWT (JSON Web Tokens):** Implement JWT for stateless authentication.
    - Upon successful login/registration, generate a JWT and send it to the frontend.
    - Frontend will store the JWT (e.g., in localStorage or sessionStorage) and send it in the Authorization header for protected routes.
    - Backend will validate the JWT for all protected API calls.
  + **Middleware/Route Protection:** Implement middleware to protect API endpoints, ensuring only authenticated and authorized users can access them.
* **LLM Integration Logic:**
  + **Dynamic LLM Selection:** Based on the user's saved LLM configuration, the generate-email endpoint will:
    - If Ollama is selected: Make a curl request or use a PHP HTTP client to the specified Ollama endpoint.
    - If a cloud provider (OpenAI, Claude, Gemini, Groq, Cohere) is selected: Make a curl request or use a PHP HTTP client to their respective API endpoints, using the stored API key.
    - Implement robust error handling for API calls (network issues, invalid API keys, rate limits).
  + **Prompt Construction:** Construct the LLM prompt based on rawThoughts, tone, and contextEmail as defined in the frontend's generateEmail function.
* **Configuration Management:**
  + Use a .env file (or similar PHP-compatible environment variable loading) to store sensitive information like JWT secret, default API keys (if any), and database credentials.
  + **Crucially, user-specific API keys for LLMs must be stored securely in the database, encrypted if possible, and retrieved only when needed.**
* **Error Handling & Logging:** Implement comprehensive error handling for API calls, database operations, and LLM interactions. Log errors for debugging.

### **3. Database (MySQL / MariaDB)**

A relational database schema suitable for MySQL or MariaDB to store user data and LLM configurations.

**Key Tables:**

* **users table:**
  + id (PRIMARY KEY, INT, AUTO\_INCREMENT)
  + email (VARCHAR, UNIQUE, NOT NULL)
  + password\_hash (VARCHAR, NOT NULL - store hashed passwords)
  + sso\_provider (VARCHAR, NULL - e.g., 'google', 'facebook')
  + sso\_id (VARCHAR, NULL - unique ID from SSO provider)
  + created\_at (DATETIME)
  + updated\_at (DATETIME)
* **user\_settings table:**
  + id (PRIMARY KEY, INT, AUTO\_INCREMENT)
  + user\_id (INT, FOREIGN KEY REFERENCES users.id, UNIQUE - one-to-one relationship with users)
  + preferred\_llm (VARCHAR, NOT NULL - e.g., 'ollama', 'openai', 'gemini')
  + ollama\_endpoint (VARCHAR, NULL)
  + openai\_api\_key (VARCHAR, NULL - **encrypted**)
  + claude\_api\_key (VARCHAR, NULL - **encrypted**)
  + gemini\_api\_key (VARCHAR, NULL - **encrypted**)
  + groq\_api\_key (VARCHAR, NULL - **encrypted**)
  + cohere\_api\_key (VARCHAR, NULL - **encrypted**)
  + updated\_at (DATETIME)

**Database Considerations:**

* **Password Hashing:** Use strong hashing algorithms (e.g., password\_hash() in PHP) for storing user passwords.
* **API Key Encryption:** Implement encryption for sensitive API keys stored in the database. PHP's openssl\_encrypt() and openssl\_decrypt() can be used.
* **Indexes:** Add appropriate indexes for frequently queried columns (e.g., email in users, user\_id in user\_settings).

### **4. Deployment & Documentation**

Crucial for HostGator deployment and project maintainability.

**Key Requirements:**

* **README.md (GitHub Ready):**
  + **Project Title & Description:** Clear and concise.
  + **Features:** List all key functionalities.
  + **Technologies Used:** List React, PHP, MySQL/MariaDB, Tailwind CSS, etc.
  + **Installation & Setup Steps:**
    1. **Prerequisites:** PHP version, Composer, Node.js, npm/yarn, MySQL/MariaDB.
    2. **Clone Repository:** git clone <repo-url>
    3. **Backend Setup:**
       - Navigate to backend directory.
       - composer install (if using Composer).
       - **Environment Variables:** Instructions to create .env file from .env.example (template provided).
         * DB\_HOST, DB\_NAME, DB\_USER, DB\_PASS
         * JWT\_SECRET
         * (Optional) Default API keys for LLMs if not user-specific.
       - **Database Setup:** SQL script for creating tables (users, user\_settings). Instructions on how to import it.
       - **Start PHP Server:** Instructions for local development server (e.g., php -S localhost:8000 -t public).
    4. **Frontend Setup:**
       - Navigate to frontend directory.
       - npm install or yarn install.
       - npm run dev or yarn dev (for local development).
       - Instructions on how to configure the frontend to point to the PHP backend API URL.
    5. **LLM Configuration:**
       - Detailed instructions on how to set up Ollama locally.
       - Instructions on obtaining API keys for cloud LLMs.
       - Explanation of how to configure LLMs via the in-app settings page.
  + **Usage:** How to use the application after setup.
  + **Contributing:** Guidelines for contributions.
  + **License.**
* **Code Comments:** Extensive and clear comments throughout both frontend (React JSX/JS) and backend (PHP) code, explaining logic, functions, and complex sections.
* **Error Messages:** User-friendly error messages on the frontend and detailed error logging on the backend.

## **Development Workflow Considerations**

* **Modular Design:** Structure the PHP backend with clear separation of concerns (e.g., controllers, models, services, authentication logic).
* **Security:** Prioritize security in all aspects:
  + Input validation and sanitization on both frontend and backend.
  + Protection against SQL injection, XSS, CSRF.
  + Secure handling of API keys and sensitive user data.
* **Scalability (for future):** While targeting HostGator, consider a design that could be extended or scaled in the future.

This master prompt provides a detailed blueprint for your Email Writing Assistant. You can now proceed with implementing each component, starting with the database schema, then the PHP backend (authentication, LLM integration, settings API), and finally integrating it with the React frontend. Remember to continuously update your README.md as you build out the features.