**ehutano+ Backend Development Plan & Architectural Blueprint**

**Project Objective:** To design, develop, and document the complete backend system for the "ehutano+" platform, ensuring functional equivalence with the features and user experiences simulated in ehutano v2/ehutano10.html.

**Source Document:** ehutano v2/ehutano10.html (content used for deriving requirements).

**1. Chosen Technology Stack**

* **Programming Language & Framework:** Python with FastAPI.
  + *Rationale:* FastAPI offers high performance, built-in data validation (via Pydantic), automatic OpenAPI documentation, and an asynchronous architecture suitable for I/O-bound operations common in web backends. Python's extensive libraries and ecosystem are also beneficial.
* **Database:** PostgreSQL.
  + *Rationale:* A robust, open-source relational database with strong support for complex queries, transactions, and data integrity, suitable for the structured data (users, inventory, orders, prescriptions) implied by the mock data.
* **Authentication:** JWT (JSON Web Tokens) for stateless authentication.
* **ORM (Object-Relational Mapper):** SQLAlchemy (specifically its async version with FastAPI).
  + *Rationale:* Provides a powerful and flexible way to interact with the PostgreSQL database, abstracting SQL complexities and integrating well with Pydantic models.
* **Deployment (Initial Consideration):** Docker for containerization, facilitating consistent environments and easier deployment.

**2. High-Level Architecture**

A **Modular Monolith** approach will be adopted initially. This means the application will be a single deployable unit, but internally structured into distinct modules corresponding to the major features/portals (User Management, Patient Portal, Pharmacy Portal, etc.). This provides good organization and allows for potential future separation into microservices if scalability demands it.

**Key Components:**

1. **API Layer (FastAPI):** Exposes RESTful API endpoints.
2. **Service Layer:** Contains business logic for each module.
3. **Data Access Layer (SQLAlchemy):** Interacts with the PostgreSQL database.
4. **Shared Utilities:** Logging, configuration, error handling, etc.

**3. Proposed Database Schema Design**

Derived from MOCK\_ data structures in ehutano10.html. Primary keys will typically be id (UUID or auto-incrementing integer). Timestamps (created\_at, updated\_at) will be included in most tables.

**Core Models:**

* **User (**users**):**
  + id: PK
  + username: String, unique
  + email: String, unique
  + hashed\_password: String
  + full\_name: String
  + role: Enum (PATIENT, PHARMACY\_STAFF, DOCTOR, WHOLESALER\_STAFF)
  + is\_active: Boolean
  + phone\_number: String (optional)
  + profile\_picture\_url: String (optional)
  + *(Role-specific profile tables might be linked via one-to-one relationships if more distinct fields are needed, e.g., patient\_profiles, doctor\_profiles)*
* **PatientProfile (**patient\_profiles**):** (Linked to User)
  + user\_id: FK (references users.id)
  + medical\_aid\_provider: String (optional)
  + medical\_aid\_member\_id: String (optional)
  + medical\_aid\_verified: Boolean (default: false)
  + blood\_pressure: String (optional)
  + last\_checkup\_date: Date (optional)
* **Pharmacy (**pharmacies**):** (If multiple pharmacies are managed by the system, or if pharmacy staff are linked to a specific pharmacy)
  + id: PK
  + name: String, unique
  + address: String
  + phone\_number: String
  + license\_number: String
  + *(Staff users would have a FK to pharmacies.id)*
* **Medicine (**medicines**):** (Master list of all possible medicines)
  + id: PK
  + name: String, unique (e.g., "Paracetamol 500mg Tabs")
  + generic\_name: String (optional)
  + category: String (e.g., "Pain Relief", "Antibiotics")
  + description: Text (optional)
  + manufacturer: String (optional)
  + requires\_prescription: Boolean (default: false)
  + is\_antibiotic: Boolean (default: false)
  + default\_image\_url: String (optional)
* **InventoryItem (**inventory\_items**):** (Specific medicine stock at a specific pharmacy)
  + id: PK
  + pharmacy\_id: FK (references pharmacies.id)
  + medicine\_id: FK (references medicines.id)
  + batch\_number: String (optional)
  + expiry\_date: Date (optional)
  + stock\_quantity: Integer
  + price: Decimal (selling price at this pharmacy)
  + cost\_price: Decimal (optional, for pharmacy's internal use)
  + supplier: String (optional, could be FK to a suppliers table)
  + wholesale\_source: String (optional, as per mock data)
  + status: Enum (IN\_STOCK, LOW\_STOCK, OUT\_OF\_STOCK)
  + last\_updated: Timestamp
* **Prescription (**prescriptions**):**
  + id: PK
  + patient\_id: FK (references users.id where role is PATIENT)
  + doctor\_id: FK (references users.id where role is DOCTOR, optional if uploaded by patient)
  + doctor\_name\_manual: String (if doctor not on platform or Rx uploaded)
  + date\_issued: Date
  + status: Enum (PENDING\_REVIEW, QUOTE\_READY, ACTIVE, FILLED, CANCELLED, EXPIRED)
  + refills\_left: Integer (default: 0)
  + upload\_url: String (URL to the uploaded image/PDF, if applicable)
  + notes\_patient: Text (optional, from patient upload)
  + is\_quote\_ready: Boolean (default: false)
  + quote\_details\_json: JSONB (stores quote items and total, as per MOCK\_PATIENT\_DATA.prescriptions.quote)
* **PrescriptionItem (**prescription\_items**):** (Medicines listed on a prescription)
  + id: PK
  + prescription\_id: FK (references prescriptions.id)
  + medicine\_name\_manual: String (if medicine not in master list or for flexibility)
  + medicine\_id: FK (references medicines.id, optional)
  + dosage: String
  + quantity: Integer
  + instructions: Text
* **Order (**orders**):**
  + id: PK (e.g., "ORD789")
  + patient\_id: FK (references users.id where role is PATIENT)
  + pharmacy\_id: FK (references pharmacies.id, where the order is placed/fulfilled)
  + order\_date: Timestamp
  + status: Enum (PENDING\_PAYMENT, PROCESSING, READY\_FOR\_PICKUP, OUT\_FOR\_DELIVERY, DELIVERED, COMPLETED, CANCELLED)
  + total\_amount: Decimal
  + payment\_method: String (e.g., "Medical Aid", "EcoCash", "Cash on Delivery")
  + medical\_aid\_provider: String (optional)
  + medical\_aid\_member\_id: String (optional)
  + medical\_aid\_status: Enum (NOT\_APPLICABLE, PENDING\_PATIENT\_AUTH, CLAIM\_SUBMITTED, RECEIVED, PAID, REJECTED) (as per MOCK\_PHARMACY\_ORDERS)
  + amount\_covered\_by\_aid: Decimal (optional)
  + delivery\_address\_id: FK (references an addresses table, optional)
  + notes: Text (optional)
* **OrderItem (**order\_items**):**
  + id: PK
  + order\_id: FK (references orders.id)
  + inventory\_item\_id: FK (references inventory\_items.id, if sourced from specific stock)
  + medicine\_name: String (snapshot of medicine name at time of order)
  + quantity: Integer
  + price\_per\_unit: Decimal (snapshot of price at time of order)
  + subtotal: Decimal
* **Reminder (**reminders**):**
  + id: PK
  + patient\_id: FK (references users.id)
  + type: Enum (REFILL, APPOINTMENT, MEDICATION)
  + related\_medicine\_id: FK (references medicines.id, optional for REFILL/MEDICATION)
  + details: Text
  + due\_date: Timestamp
  + is\_dismissed: Boolean (default: false)
* **MedicineVerificationLog (**medicine\_verification\_logs**):**
  + id: PK
  + scanned\_data: String (e.g., QR code content)
  + user\_id: FK (references users.id, who performed scan, optional)
  + scan\_timestamp: Timestamp
  + verification\_status: Enum (VERIFIED, INVALID, EXPIRED, NOT\_FOUND)
  + medicine\_name\_match: String (optional)
  + batch\_number\_match: String (optional)
  + expiry\_date\_match: Date (optional)
  + pharmacy\_source\_match: String (optional)
  + wholesale\_source\_match: String (optional)
  + manufacturer\_match: String (optional)
  + is\_antibiotic\_match: Boolean (optional)
  + response\_message: Text
* **MarketWatchData (**market\_watch\_data**):** (This might be a more complex set of tables or aggregated views, or even sourced from an external analytics system. For simplicity, a single table for mock data representation):
  + id: PK
  + type: Enum (FAST\_MOVING, HIGH\_DEMAND, STOCK\_OUT, PRICE\_ALERT, NEWLY\_REGISTERED, RECENTLY\_IMPORTED, AI\_INSIGHT)
  + item\_name\_or\_insight: Text
  + details\_json: JSONB (to store varied fields like trend, reason, supplier, note, source, date, distributor, etc.)
* **Advert (**adverts**):**
  + id: PK
  + title: String
  + image\_url: String
  + link\_url: String
  + approved\_by\_ref: String (e.g., "MCAZ Ref: ...")
  + type: String (e.g., "Pharmacy Promotion")
  + is\_active: Boolean
* **WellnessActivity (**wellness\_activities**):**
  + id: PK
  + name: String
  + day\_of\_week: String
  + time: Time
  + location: String
  + total\_slots: Integer
  + icon\_emoji: String (optional)
  + description: Text (optional)
* **WellnessBooking (**wellness\_bookings**):**
  + id: PK
  + activity\_id: FK (references wellness\_activities.id)
  + user\_id: FK (references users.id)
  + booking\_timestamp: Timestamp
* **BlogPost (**blog\_posts**):**
  + id: PK
  + title: String
  + author\_name: String (or FK to users.id if authors are users)
  + publish\_date: Date
  + snippet: Text
  + full\_content: Text
  + image\_url: String (optional)
  + category: String
  + tags: Array of Strings (optional)
* **Wholesaler (**wholesalers**):** (If managing multiple wholesalers)
  + id: PK
  + name: String
  + *(Staff users with WHOLESALER\_STAFF role would have a FK to wholesalers.id)*
* **WholesalerCatalogItem (**wholesaler\_catalog\_items**):**
  + id: PK
  + wholesaler\_id: FK (references wholesalers.id)
  + medicine\_id: FK (references medicines.id)
  + wholesale\_price: Decimal
  + available\_stock: Integer
  + minimum\_order\_quantity: Integer (optional)
* **WholesalerOrder (**wholesaler\_orders**):** (Orders placed by Pharmacies to Wholesalers)
  + id: PK
  + pharmacy\_id: FK (references pharmacies.id)
  + wholesaler\_id: FK (references wholesalers.id)
  + order\_date: Timestamp
  + total\_amount: Decimal
  + status: Enum (PENDING\_CONFIRMATION, PROCESSING, SHIPPED, DELIVERED, CANCELLED)
* **WholesalerOrderItem (**wholesaler\_order\_items**):**
  + id: PK
  + wholesaler\_order\_id: FK (references wholesaler\_orders.id)
  + wholesaler\_catalog\_item\_id: FK (references wholesaler\_catalog\_items.id)
  + quantity: Integer
  + price\_per\_unit: Decimal

**4. Preliminary API Design (RESTful with FastAPI)**

Endpoints will be grouped by resource/module. All endpoints will require authentication except where specified (e.g., public medicine search, registration). RBAC will be applied at the endpoint or service level.

**Base URL:** /api/v1/

**Authentication (**/auth**)**

* POST /auth/register/patient: Patient registration.
* POST /auth/register/{role}: Generic registration for other roles (admin controlled).
* POST /auth/login: User login (returns JWT).
* POST /auth/logout: User logout (if using token blocklists).
* GET /auth/me: Get current authenticated user's details.

**Patient Portal (**/patient**)**

* GET /medicines: Search/list medicines (publicly accessible, but can show patient-specific views if logged in).
* GET /medicines/{medicine\_id}: Get medicine details.
* POST /patient/prescriptions/upload: Upload prescription.
* GET /patient/prescriptions: List patient's prescriptions.
* GET /patient/prescriptions/{rx\_id}: Get prescription details (including quote if ready).
* POST /patient/prescriptions/{rx\_id}/request-refill: Request a refill.
* GET /patient/orders: List patient's orders.
* GET /patient/orders/{order\_id}: Get order details.
* POST /patient/cart: Add item to cart.
* GET /patient/cart: View cart.
* POST /patient/checkout: Create order from cart or quote.
* POST /patient/verify-medicine: Submit QR scan data for verification.
* GET /patient/reminders: Get patient reminders.
* GET /patient/profile: Get/update patient profile.

**Pharmacy Portal (**/pharmacy**)** (Requires PHARMACY\_STAFF role)

* GET /pharmacy/orders: List orders for the pharmacy.
* GET /pharmacy/orders/{order\_id}: Get order details.
* PUT /pharmacy/orders/{order\_id}/status: Update order status.
* POST /pharmacy/orders/{order\_id}/submit-claim: Submit medical aid claim.
* GET /pharmacy/inventory: List inventory items.
* POST /pharmacy/inventory: Add new inventory item.
* PUT /pharmacy/inventory/{item\_id}: Update inventory item.
* POST /pharmacy/dispense: Dispense medicine (walk-in sale).
* GET /pharmacy/analytics/kpis: Get Key Performance Indicators.
* GET /pharmacy/analytics/sales-trends: Get sales trend data.
* POST /pharmacy/analytics/reports: Generate custom report.
* GET /pharmacy/market-watch: Get market watch data.
* GET /pharmacy/claims: List medical aid claims.

**Doctor Portal (**/doctor**)** (Requires DOCTOR role)

* POST /doctor/erx: Create electronic prescription.
* GET /doctor/erx: List E-Rx created by the doctor.
* GET /doctor/erx/{erx\_id}: Get E-Rx details.
* GET /doctor/patients: Search/list patients (access control critical).
* GET /doctor/patients/{patient\_id}: View patient profile/summary.
* GET /doctor/appointments/today: Get today's appointments.
* GET /doctor/notifications: Get doctor-specific notifications.

**Wholesaler Portal (**/wholesaler**)** (Requires WHOLESALER\_STAFF role)

* GET /wholesaler/catalog: List wholesaler's product catalog.
* POST /wholesaler/catalog: Add item to catalog.
* PUT /wholesaler/catalog/{item\_id}: Update catalog item.
* GET /wholesaler/orders: List orders received from pharmacies.
* PUT /wholesaler/orders/{order\_id}/status: Update wholesaler order status.

**Community Wellness Hub (**/wellness**)** (Mostly public, some actions require auth)

* GET /wellness/activities: List wellness activities.
* POST /wellness/activities/{activity\_id}/book: Book an activity (requires auth).
* GET /wellness/blog: List blog posts.
* GET /wellness/blog/{post\_id}: Get blog post details.
* GET /wellness/resources/{resource\_name}: Get specific health resource.
* POST /wellness/tools/bmi: Calculate BMI.
* GET /wellness/antibiotic-stewardship: Get antibiotic info.

**5. Addressing Non-Functional Requirements**

* **API Design:** FastAPI with Pydantic models will ensure well-structured and automatically documented (OpenAPI) RESTful APIs. Versioning will be handled via URL prefix (/api/v1/).
* **Database Design:** PostgreSQL with SQLAlchemy ORM. Schemas are designed for normalization where appropriate, with clear relationships.
* **Security:**
  + **Authentication:** JWT.
  + **Authorization:** FastAPI dependencies for RBAC.
  + **OWASP Top 10:** FastAPI provides some inherent protection (e.g., against XSS via Jinja2 templating if used, but primarily API). Input validation via Pydantic. SQL Alchemy helps prevent SQL injection. HTTPS will be mandatory in production.
  + **Input Validation:** Pydantic models for request bodies.
  + **Sensitive Data:** Careful handling, encryption at rest for highly sensitive fields if necessary (though not explicitly detailed for this phase).
* **Performance & Scalability:** FastAPI's async nature and Python's ecosystem allow for good performance. Stateless API design. PostgreSQL is scalable. Caching strategies (e.g., Redis) can be added later.
* **Error Handling & Logging:** FastAPI's exception handling. Standard Python logging module. Consistent error responses.
* **Code Quality:** Adherence to Python (PEP 8) and FastAPI best practices. Thorough commenting.

**6. Deliverables (Target Output)**

* **Complete Backend Source Code:** Python/FastAPI project, organized by modules.
* **Database Schema Definitions:** SQLAlchemy models.
* **API Documentation:** Auto-generated OpenAPI (Swagger UI) by FastAPI.
* **README.md:** Setup, environment configuration, and basic deployment steps (Dockerized).
* **Tests:** Unit tests (Pytest) for service logic and critical utilities. Integration tests for API endpoints.

**7. Development Approach**

An iterative approach will be taken, likely focusing on:

1. Core User Management & Authentication.
2. Patient Portal (Medicine Search, Prescription Upload & Basic Ordering).
3. Pharmacy Portal (Order Fulfillment, Basic Inventory).
4. Connecting Patient & Pharmacy flows for prescription quoting.
5. Doctor Portal (E-Rx creation).
6. Wholesaler Portal.
7. Community Wellness Hub.
8. Advanced features for each portal (Analytics, Market Watch, etc.).
9. Refinement, comprehensive testing, and documentation.

Ambiguities in ehutano10.html will be noted, and reasonable assumptions made, with a preference for simpler interpretations initially. For instance, "AI Insights" will start as mock data served by an API.

**8. Final Check Principle**

Throughout development, the guiding principle will be to ensure that every piece of mock data structure and every simulated user interaction in ehutano10.html can be mapped to a functional backend component (database model, business logic, API endpoint). The goal is a backend that could seamlessly power the frontend simulation.

This plan provides the initial blueprint. Development will proceed based on these guidelines, with further detailed design for each module and iterative refinement.