ACMETRACK System Design

1. Purpose and Motivation

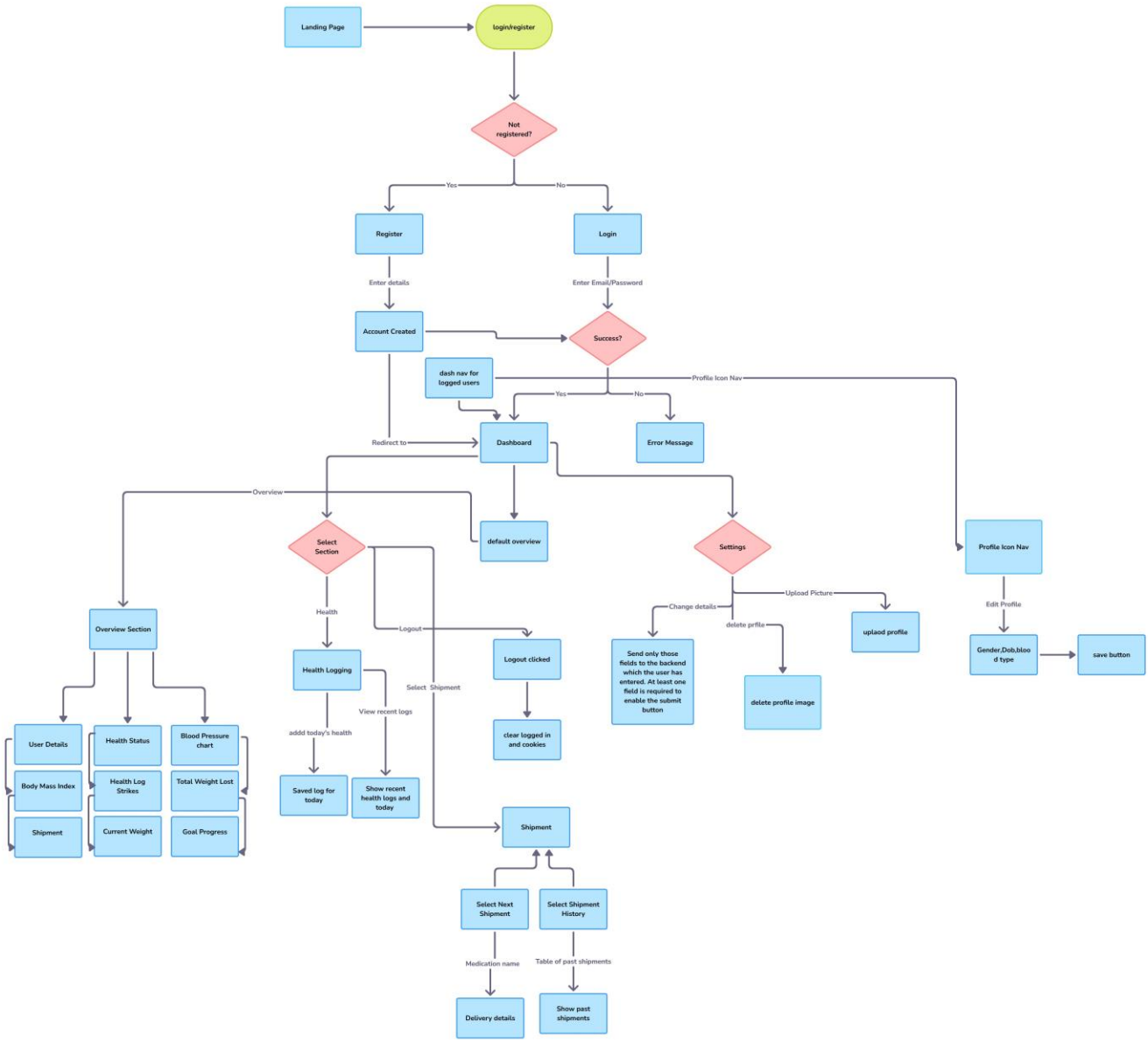
Why Implement ACMETRACK?

ACMETRACK is a prototype health and medication management platform designed to enable users to track health metrics and monitor medication shipments. Its key objectives are:

- **User Empowerment**: Facilitate tracking of health data (weight, heart rate, blood pressure) and medication schedules to support self-managed care.
- **Medication Logistics**: Provide visibility into shipment status for timely medication refills.
- **Health Insights**: Offer metrics like BMI and weight trends to encourage engagement.
- **Compliance Tracking**: Implement a strike system to promote regular health logging.
- **Prototype Validation**: Test core features and user experience for future iterations.

2. System Architecture

ACMETRACK employs a **monolithic, cloud-native architecture** with a client-server model, chosen for its simplicity and suitability for a prototype. The system is divided into three primary layers:



2.1 Frontend (Client Layer)

- **Purpose**: Provides an interactive interface for health tracking, shipment monitoring, and profile management.
- **Technology**: React.js, served on port 5173.
- **Features**:
    - Forms for health log entry and profile updates.
    - Dashboards for health metrics and shipment status.
    - Responsive UI for web and mobile browsers (to be fully enhanced in future).

- **Deployment**: Containerized with Docker

2.2 Backend (API Layer)

- **Purpose**: Handles all business logic, API requests, and integrations within a single monolithic application.
- **Technology**: Node.js with Express.js, served on port 3002.
- **Components**:
    - **Controllers**: Manage health, shipment, and user operations.
    - **Middleware**: Authentication, file uploads, and error handling.
    - **Utilities**: Standardized responses and external service integrations.
- **API Design**: RESTful with versioning (`/api/v1/`).
- **Monolithic Rationale**: A single codebase simplifies development, testing, and deployment for the prototype phase, reducing complexity and accelerating iteration.
- **Deployment**: Dockerized, with potential scaling via Kubernetes in future iterations.

2.3 Database Layer

- **Purpose**: Stores user profiles, health logs, and shipment data.
- **Technology**: MongoDB (NoSQL) for flexible data modeling.
- **Collections**:
    - `Users`: User details and authentication data.
    - `HealthLogs`: Daily health metrics and medications.
    - `Shipments`: Medication delivery details.
- **Deployment**: MongoDB Atlas for managed hosting with replica sets.

2.4 External Services

- **Image Storage**: Cloudinary for profile picture management.
- **Future Services**: Email notifications for account or shipment updates.

2.5 Infrastructure

- **Containerization**: Docker for environment consistency.
- **Orchestration**: Docker Compose for development; Kubernetes-ready for future scaling.
- **Cloud**: AWS (EC2 or ECS) for deployment.

Why Monolithic Architecture for Prototype?

- **Rapid Development**: A monolithic architecture consolidates all functionality into a single codebase, enabling faster implementation and iteration during the prototype phase.
- **Simplified Deployment**: Single application deployment reduces infrastructure complexity, ideal for testing and validation.
- **Lower Overhead**: Avoids the complexity of microservices (e.g., service discovery, inter-service communication) when the system scope is still being defined.
- **Cost-Effective**: Minimizes resource requirements for a proof-of-concept, focusing on feature validation over scalability.
- **Future Transition**: The modular design (separate controllers, routes) allows refactoring into microservices if the prototype succeeds and requires scaling.

3. Key Components

3.1 Authentication and Authorization

- **Mechanism**: JWT-based with access and refresh tokens.
- **Security**: Tokens in HTTP-only cookies; passwords hashed.
- **Compliance**: Strike system tracks logging adherence.

3.2 Health Log Management

- **Features**:
    - Create validated health logs (weight, heart rate, etc.).
    - Retrieve logs, summaries, and metrics (BMI, weight trends).
    - Analyze trends (e.g., monthly blood pressure).
- **Validation**: Ensures data integrity with clear error messages.

3.3 Shipment Tracking

- **Features**:
    - Fetch recent and historical shipments.
    - Calculate progress based on status (e.g., 50% for "in transit").
    - Format dates for readability.
- **Data**: Includes medication, carrier, and tracking details.

3.4 Profile Management

- **Features**:
    - Update user details (name, gender, blood type).
    - Manage profile pictures via external storage.
    - Display dashboard with metrics (e.g., age).
- **Image Handling**: Secure uploads with format restrictions.

3.5 Error and Response Handling

- **Errors**: Custom class with status codes and messages.
- **Responses**: Standardized format for consistency.
- **Async Handling**: Centralized error propagation.

4. Why This Implementation?

4.1 Technology Rationale

- **MongoDB**: Flexible schemas for evolving health data; supports aggregations.
- **Node.js/Express**: Lightweight, fast for API development.
- **React**: Component-based UI for dynamic experiences.
- **Cloudinary**: Off Plague image management.
- **Docker**: Ensures portability for prototype testing.
- **JWT**: Stateless authentication for simplicity.

4.2 Design Principles

- **Modularity**: Separated concerns within the monolith for maintainability.
- **Simplicity**: Monolithic structure accelerates prototype development.
- **Security**: Robust authentication and validation.
- **Reliability**: Error handling and connection retries.
- **User Focus**: Intuitive APIs and formatted outputs.

4.3 Prototype Focus

- **Speed to Market**: Monolithic architecture enables rapid feature development and user feedback collection.
- **Feature Validation**: Prioritizes core functionality (health tracking, shipment monitoring) over premature optimization.

- **Iterative Design**: Modular code allows easy refactoring for future microservices if needed.

5. Scalability and Performance

5.1 Scaling (Prototype Constraints)

- **Backend**: Single instance sufficient for prototype; future horizontal scaling with load balancers.
- **Database**: MongoDB replica sets for read scalability.
- **Frontend**: Static assets via CDN for performance.

5.2 Caching

- **Redis (Future)**: Cache frequent queries for performance.
- **API Caching**: Short TTL for GET responses in production.

5.3 Database Optimization

- **Indexes**: On `userId`, `createdAt` for query speed.
- **Aggregations**: Optimized for analytics.
- **Pagination**: Limits data retrieval.

5.4 Performance

- **Latency**: <200ms for API responses.
- **Throughput**: Support 1,000 concurrent users for prototype testing.
- **Queries**: <50ms with indexes.

6. Security

6.1 Authentication

- **JWT**: Signed tokens with secure storage.
- **Expiry**: Short-lived access tokens; longer refresh tokens.
- **Future**: Role-based access for admins.

6.2 Data Protection

- **Encryption**: Data at rest (MongoDB Atlas) and in transit (HTTPS).

- **Validation**: Schema and API checks prevent injection.
- **Uploads**: Restricted file types and external storage.

6.3 Future Compliance

- **Future Healthcare**: Secure storage for HIPAA alignment.
- **Future Privacy**: Data deletion/export for GDPR.
- **Future Rate Limiting**: Mitigate brute-force attacks.

7. Future Enhancements

- **Admin Role-Based Access (RBA)**:
    - Introduce an admin role to approve user access to the dashboard, replacing direct signup.
    - Users register but require admin approval to access features, enhancing security and compliance.
    - Admins can manage user accounts, review health logs, and monitor compliance (e.g., strikes).
- **Hospital Account Creation**:
    - Enable hospitals or healthcare providers to create accounts for patients, streamlining onboarding.
    - Hospital admins can bulk-create user accounts with pre-filled details (e.g., name, blood type).
    - Integrate with hospital systems via secure APIs to sync patient data (with consent).
- **Redis Integration**:
    - Implement Redis for caching frequently accessed data (e.g., recent health logs, user profiles).
    - Reduce database load and improve API response times (target: <100ms for cached queries).
    - Use Redis for session management to enhance authentication scalability.
- **Notification System**:
    - Add real-time notifications for shipment updates, health log reminders, and admin approvals.
    - Support multiple channels: email, SMS, and in-app notifications.
    - Use a message queue (e.g., AWS SQS or RabbitMQ) to handle notification delivery reliably.
- **AI-Based Chatbot**:
    - Integrate an AI-powered chatbot (e.g., powered by xAI's Grok) to assist users with health queries.

    - Provide guidance on logging health data, interpreting metrics, and managing medications.
    - Enable natural language interactions for a user-friendly experience.
- **Health Warning System**:
    - Implement AI-driven analysis of health logs to detect anomalies (e.g., high blood pressure, irregular heart rate).
    - Send automated warnings to users and healthcare providers (if authorized) for critical health events.
    - Use predefined thresholds and machine learning models to improve warning accuracy.
- **Fully Responsive Website**:
    - Enhance the frontend to be fully responsive across all devices (desktops, tablets, smartphones).
    - Use CSS frameworks (e.g., Tailwind CSS) and media queries to ensure seamless UX on varying screen sizes.
    - Optimize layouts for touch interactions and smaller screens, improving accessibility and user engagement.
- **Microservices Transition**:
    - Refactor the monolith into microservices (e.g., user, health, shipment services) for scalability.
    - Use an API gateway for routing and load balancing.
- **Real-Time Updates**:
    - Introduce WebSockets for live shipment and notification updates.
- **Advanced Analytics**:
    - Develop predictive health insights using machine learning.
- **Mobile Apps**:
    - Build native iOS and Android apps for broader accessibility.