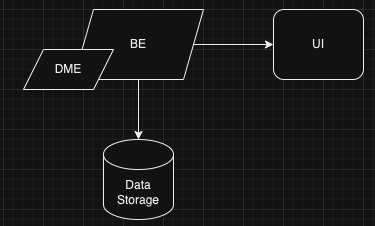
### **High-Level Design (HLD)**

#### **Project Title: Domain Monitoring System**

#### **1. Overview**

* **Objective**: Build a system that allows users to register, log in, and monitor multiple domains for liveness and SSL certificate details. The platform will provide real-time monitoring updates and support concurrent scanning of domains.
* **Scope**: This project focuses on the primary functionalities of user registration, login, domain management, and multithreaded monitoring, with scalability to transition to a database system in later stages.

#### **2. System Architecture**



* **Architecture Diagram**: The architecture is organized into several components:
  + **Frontend (UI)**: Handles user interaction for registration, login, domain input, and result display.
  + **Backend (Flask)**: Manages user sessions, API requests, and domain monitoring functions.
  + **Domain Monitoring Engine (DME)**: A multithreaded service that checks each domain for liveness and SSL status.
  + **Data Storage**: Temporary JSON-based storage for user and domain data, to transition to a database in future versions.
* **Component Descriptions**:
  + **UI Component**: Built with HTML, CSS, and JavaScript, the UI allows users to register, log in, add domains, and view monitoring results.
  + **Backend (Flask)**: Acts as the server, handling user authentication, domain submission, and delivering monitoring results.
  + **Domain Monitoring Engine**: Uses concurrent.futures.ThreadPoolExecutor to scan multiple domains at once, verifying HTTP response and SSL data.
  + **Data Storage (JSON Files)**: Stores user data in users.json and each user’s domain data in a separate JSON file <username>\_domains.json.

#### **3. User Flows**

* **User Registration Flow**:
  + User enters a unique username and password on the registration page.
  + Backend validates input and stores user details in users.json.
  + User receives confirmation of successful registration.
* **Login Flow**:
  + User provides credentials to log in.
  + Flask’s session object is used to authenticate and maintain user sessions.
  + Upon successful login, the user is redirected to their personalized dashboard.
* **Domain Management Flow**:
  + **Single Domain Entry**: User submits a domain via an input field; the backend validates and stores it.
  + **Bulk Domain Upload**: User uploads a .txt file with multiple domains; backend processes and validates each entry.
  + User’s domains are stored in <username>\_domains.json for future checks.
* **Domain Monitoring and Display Flow**:
  + The monitoring engine performs liveness and SSL checks on each domain using ThreadPoolExecutor.
  + Results are stored in <username>\_domains.json.
  + The dashboard displays domain data in a table format, with columns for domain, liveness status, SSL expiration date, and SSL issuer.

#### **4. Key Features and Modules**

* **User Management Module**:
  + **Registration**: Users can register with a unique username and password, stored in users.json.
  + **Login**: Handles user authentication and session management.
* **Domain Management Module**:
  + **Add Domain**: Supports both single entry and bulk upload of domains for monitoring.
  + **Monitoring System**:
    - **Liveness Check**: Uses the requests library to confirm if the domain is live.
    - **SSL Check**: Uses Python’s ssl library to retrieve SSL expiration date and issuer.
    - **Multithreaded Scanning Module**:
      * Uses concurrent.futures.ThreadPoolExecutor to perform concurrent domain checks, allowing multiple domains to be processed simultaneously.
      * Handles 100 domains per user with efficiency and minimal delay.
* **Data Storage**:
  + **users.json**: Stores basic registration details for all users.
  + **<username>\_domains.json**: Individual files for each user containing domain data, liveness status, SSL expiration date, and SSL issuer.
  + JSON structure is organized for easy migration to a database.

#### **5. Technology Stack**

* **Backend**: Python with Flask framework for managing user sessions, handling requests, and routing.
* **Frontend**: HTML, CSS, and JavaScript for forms, dashboard display, and table formatting.
* **Data Handling**: JSON for user and domain data storage.
* **Concurrency**: Python’s concurrent.futures.ThreadPoolExecutor to enable multithreaded domain checks.