### **Low-Level Design (LLD)**

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

#### **1. Detailed Component Design**

* **User Management Module**
  + **Registration**:
    - **Endpoint**: /register
    - **Method**: POST
    - **Data Flow**:
      * User submits username and password.
      * Flask handles validation to ensure the username is unique.
      * User details are stored in users.json with the following structure:  
          
        {

"username": "example\_user",

"password": "plain\_text\_password"

}

* + - **Security Considerations**: Password stored in plain text initially; future updates will include encryption.
  + **Login**:
    - **Endpoint**: /login
    - **Method**: POST
    - **Data Flow**:
      * User submits credentials.
      * Flask validates the credentials against users.json.
      * Upon success, a session is created using Flask’s session object, allowing user access to the dashboard.
    - **Session Management**: Flask session object is used to maintain user state across requests.
* **Domain Management Module**
  + **Add Domain**:
    - **Single Domain Input**:
      * **Endpoint**: /add\_domain
      * **Method**: POST
      * **Input**: Single domain URL (validated for URL format).
      * **Data Flow**:
        + Upon submission, the domain is stored in <username>\_domains.json.
        + Structure of each domain entry:

{

"domain": "example.com",

"status": "Pending",

"ssl\_expiration": "N/A",

"ssl\_issuer": "N/A"

}

* + - **Bulk Domain Upload**:
      * **Endpoint**: /bulk\_upload
      * **Method**: POST
      * **Input**: .txt file containing one domain per line.
      * **Data Flow**:
        + Flask reads the file, validating each domain entry.
        + Valid domains are added to <username>\_domains.json.
  + **Monitoring System**:
    - **Liveness Check**:
      * **Library**: requests
      * **Process**:
        + HTTP GET request is sent to the domain.
        + If the response status code is 200, status is set to “Live”; otherwise, it is set to “Down”.
    - **SSL Check**:
      * **Library**: ssl
      * **Process**:
        + Retrieves SSL certificate information for each domain.
        + Updates ssl\_expiration with the expiration date and ssl\_issuer with the certificate authority.

#### **2. DataStore/File Structure Design**

* **users.json**: Stores all registered users with basic details.  
    
  [

{

"username": "example\_user1",

"password": "plain\_text\_password1"

},

{

"username": "example\_user2",

"password": "plain\_text\_password2"

}

]

* **<username>\_domains.json**: Stores each user’s monitored domains, including the monitoring status and SSL details.  
    
  [

{

"domain": "example.com",

"status": "Live",

"ssl\_expiration": "2025-12-31",

"ssl\_issuer": "Example Certificate Authority"

}

]

#### **3. API Contracts**

* **Registration Endpoint**
  + **Endpoint**: /register
  + **Method**: POST
  + **Request Body**:  
      
    {

"username": "example\_user",

"password": "example\_password"

}

* + **Response**:
    - Success: {“status”: “Success”, "message": "Registration successful"}
    - Error (e.g., username already exists): {“status” “Error, "message": "<error message>"}
* **Login Endpoint**
  + **Endpoint**: /login
  + **Method**: POST
  + **Request Body**:  
      
    {

"username": "example\_user",

"password": "example\_password"

}

* + **Response**:
    - Success: {“status”: “Success”, "message": "Login successful"}
    - Error (e.g., invalid credentials): {“status” “Error, "message": "<error message>"}
* **Add Domain Endpoint**
  + **Endpoint**: /add\_domain
  + **Method**: POST
  + **Request Body**:  
      
    {

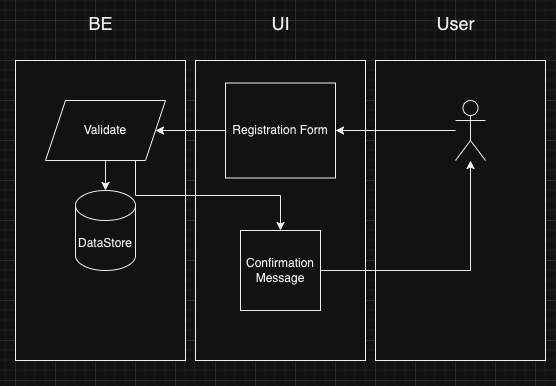
"domain": "example.com"

}

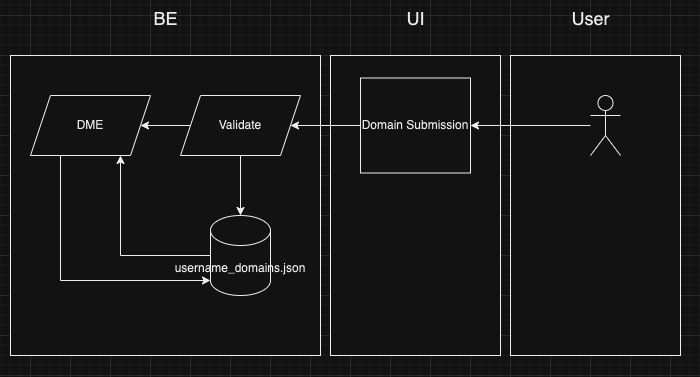
* + **Response**:
    - Success: {“status”: “Success”, "message": "Domain added successfully"}
    - Error (e.g., invalid domain format): {“status” : "Error", “message”: "Invalid domain format"}
* **Bulk Upload Endpoint**
  + **Endpoint**: /bulk\_upload
  + **Method**: POST
  + **Request**: .txt file containing domains
  + **Response**:
    - Success: {“status”: “Success”, "message": "Bulk upload successful"}
    - Error: {“status: "Error", “message”: "Invalid file format or content"}

#### **4. Data Flow Diagrams**

* **User Registration Flow**:



* + User enters details in the form.
  + Backend validates and stores the details in users.json.
  + Confirmation is sent back to the user.
* **Domain Monitoring Flow**:



* + User submits a domain (single or bulk).
  + Domain names are validated.
  + Domains are stored in <username>\_domains.json with default values (status as “Pending”).
  + Monitoring engine checks the domain’s liveness and SSL status, updating the JSON file.

#### **5. Concurrency and Multithreading**

* **Multithreading Design**:
  + **Library**: concurrent.futures.ThreadPoolExecutor
  + **Process**:
    - The backend creates a thread pool with a limited number of threads (based on server capabilities).
    - Each thread independently performs domain checks (liveness and SSL).
    - Results are stored back in the user’s JSON file.

#### **6. Error Handling and Validation**

* **User Input Validation**:
  + **Registration**: Ensure unique username, non-empty password.
  + **Domain Validation**: Check if domain format is valid (e.g., example.com).
  + **File Validation**: For bulk uploads, ensure each line contains a valid domain.
* **Error Messages**:
  + Registration errors (e.g., duplicate username).
  + Login errors (e.g., incorrect password).
  + Domain errors (e.g., invalid domain or file format).
* **Logging**:
  + Record errors encountered during domain checks and API calls.
  + Maintain logs for debugging concurrency issues and tracking I/O operations.

#### **7. Frontend Structure**

* **User Forms**:
  + **Registration Form**: Fields for username and password.
  + **Login Form**: Fields for username and password.
  + **Domain Submission**:
    - Single domain text field and file upload for bulk domains.
* **Dashboard Layout**:
  + Table for displaying domains, with columns for:
    - Domain name
    - Liveness status
    - SSL expiration date
    - SSL issuer