ID5: Update for Software Requirements

1. Introduction

1.1 Purpose

Commercial wearable devices have emerged as promising tools for measuring physical activity both in laboratory and real-world settings. However, researchers have faced challenges extracting these data in an efficient, scalable way. In this paper, we describe the Built Environment and Active Populations (BEAP) Engine, a web application/tool we developed to collect, process, and analyze physical activity data exported from Apple Watch and Fitbit devices. As a value-added feature, the tool also predicts past physical activity using machine learning methods and returns the file in a more user-friendly format.

1.2 Intended Audience

The app is designed for researchers who specialize in physical activity measurement and prediction. It serves academics, scientists, and professionals engaged in health-related research, wearable technology, and data analysis. Additionally, normal users can also utilize the app to upload their own data, enabling them to track and measure their physical activities for their personal use.

1.3 Scope

The software aims to address the following main goals and objectives:

1. **Deploying the Web Application:** The primary objective is to deploy the web application in a functional preliminary state, ensuring accessibility and usability for researchers. This involves setting up the necessary infrastructure, configuring servers, and deploying the application to a production environment.

2. **Documentation and Readability:** Extensively document all involved repositories to make the project more readable and open for any future work. This includes comprehensive documentation covering installation instructions, codebase structure, API endpoints, and data flow diagrams. Detailed

documentation promotes better understanding, collaboration, and maintainability.

3. **Feedback Mechanism:** Provide team feedback on difficulties and challenges during the development process. Establish a feedback mechanism, such as regular meetings or communication channels, to address issues, share insights, and foster a collaborative environment. Feedback helps identify areas for improvement, resolve conflicts, and ensure alignment with project objectives.
4. **Support for Garmin Data Processing:** Create coverage in Java and React components for potential additions of Garmin data processing, as requested by Dr. Fuller. This involves extending the application's functionality to support data from Garmin wearable devices, ensuring compatibility, and scalability for future enhancements.
5. **Performance Optimization:** Implement speed (performance) improvements to enhance user experience, particularly in handling large datasets, as identified by stakeholders. This includes optimizing data processing algorithms, improving database queries, and enhancing frontend responsiveness to reduce loading times and enhance overall system performance.

2. System Features and Requirements

2.1 Functional Requirements

1. **Upload Fitbit, Apple Watch, and Garmin Data:**
    - Users should be able to upload data from Fitbit, Apple Watch, and optionally Garmin devices for processing and analysis. The system should support various file formats such as JSON, XML, or CSV.
2. **Improved User Experience:**

    Enhance the overall user experience through a comprehensive redesign of the user interface (UI), focusing on accessibility, intuitiveness, and efficiency, to support researchers in their analysis and interpretation of physical activity data.

- **Intuitive Navigation:** Streamline navigation through a redesigned layout that logically organizes features and functions, enabling users to find and access the tools they need quickly.
- **Responsive Design:** Ensure the application's UI is fully responsive, providing an optimal viewing and interaction experience across a wide range of devices (from desktop monitors to mobile phones).
- **Loading Times:** Optimize UI elements to ensure rapid loading times, aiming for a maximum initial load time of 2 seconds, with subsequent page loads under 1 second.
- **Interaction Delays:** Minimize interaction delays (e.g., when switching between tools or uploading data), aiming for a response time of less than 100 milliseconds for 90% of interactions.
- **Clear Error Messaging:** Provide clear, understandable error messages that help users diagnose and resolve issues. For instance, if a data upload fails, the UI should display an error message explaining the reason (e.g., "File format not supported. Please upload a CSV, JSON, or XML file.").
- **Graceful Error Recovery:** Design the UI to enable easy recovery from errors, allowing users to correct issues without losing their progress. For example, if a session times out, offer the option to log in again and return to the same point.

3. **User Authentication:**

   Implement a secure, efficient, and user-friendly authentication system to verify the identity of users and control access to the application. Utilize bearer tokens to manage sessions and secure access to resources.

   - **Secure Login Process:** Establish a secure and straightforward login process that supports standard authentication mechanisms, including password-based login and optional two-factor authentication (2FA) for enhanced security.
   - **Bearer Token Implementation:** Upon successful authentication, issue a secure, time-limited bearer token that the user's client software must provide in subsequent requests to access protected resources.

- **Logout and Session Management:** Allow users to securely log out of the application, effectively invalidating their current session and bearer token. Provide mechanisms to manage active sessions across different devices.
- **Authentication Speed:** Aim for the authentication process, including the generation and validation of bearer tokens, to be completed within 2 seconds, ensuring a smooth and efficient user experience.
- **Scalability:** Ensure the authentication system can handle a large number of simultaneous authentication requests without significant performance degradation.
- **API Security:** Secure all API endpoints with appropriate authentication checks, requiring valid bearer tokens for access. Implement rate limiting and anomaly detection to protect against brute force attacks and abuse.
- **Invalid Credentials:** Provide immediate feedback on invalid login attempts due to incorrect email/password combinations, without specifying which part of the credentials is wrong to prevent information leakage.
- **Expired Tokens:** Notify users when their session has expired due to an outdated bearer token and prompt them to re-authenticate to continue their session.
- **Unauthorized Access Attempts:** Log and alert administrators on repeated unauthorized access attempts, indicating potential security threats.

4. **User Logout:**
    - Provide users with the ability to securely log out from the application. Clear session data and invalidate authentication tokens to ensure user privacy and security.
5. **Data Processing and Prediction:**
    - Allow users to process and predict physical activity data using machine learning algorithms such as Support Vector Machine (SVM), Random Forest, and Decision Tree. Provide options for customizing prediction parameters and model selection.

Nice-to-Have Features:

1. **Upload Garmin Data:**

- Optionally allow users to upload data from Garmin devices for additional analysis and comparison. Ensure seamless integration with existing data processing pipelines and algorithms.

2. **Process/Predict Garmin Data:**
   - Extend processing and prediction capabilities to include data from Garmin devices for comprehensive research insights. Develop specialized algorithms and models tailored to Garmin data formats and metrics.

2.2 External Interface Requirements

1. **User Authentication Interface:**
   - Implement a user-friendly login interface with options for email/password authentication.
   - Provide password recovery mechanisms such as email verification or security questions for account recovery.
2. **User Logout Interface:**
   - Design a simple and intuitive interface for users to log out from the application. Include options for session termination and account logout.
3. **Prediction Functionality:**
   - Enable users to initiate the prediction process based on uploaded device data. Provide clear instructions and visual feedback on prediction progress and results.
4. **Data Selection Interface:**
   - Implement a user-friendly interface, such as radio buttons or checkboxes, for users to select specific data for processing or prediction. Allow users to customize data selection based on timestamps, activity types, or device sources.
5. **Data Processing Interface:**
   - Provide users with a streamlined process for data processing, ensuring ease of use and efficiency. Include options for data preprocessing, feature selection, and model training/validation.
6. **File Upload Interface:**

- Design a user-friendly drop zone interface for seamless file uploads. Support drag-and-drop functionality and provide feedback on file upload progress and success.

7. **Navigation Bar:**
    - Implement a responsive navigation bar for easy access to various application functionalities. Include menu items for home, profile, settings, data upload, and data analysis.
8. **Progress Bar Functionality:**
    - Display a progress bar to indicate the status of file uploads or processing tasks. Provide real-time updates on progress percentage and estimated completion time.
9. **User Registration Interface:**
    - Provide a user-friendly registration interface for new users to create accounts. Collect essential information such as name, email, password, and optional profile details.
10. **Data Deletion Functionality:**
    - Allow users to delete uploaded data or processed results as needed for data management purposes. Implement secure data deletion mechanisms to ensure compliance with data privacy regulations.
11. **Device Selection Interface:**
    - Implement a user-friendly interface for selecting the device type (e.g., Fitbit, Apple Watch, Garmin) for data manipulation. Provide clear instructions and visual cues for device selection.
12. **Download Processed Data:**
    - Enable users to download processed or predicted data in their preferred format for further analysis or sharing. Support common file formats such as CSV, Excel, JSON, or PDF.

2.3 Nonfunctional Requirements

1. **Scalability and Performance:**

- Ensuring the application can handle large amounts of data files efficiently and performantly. Implement scalable architecture patterns such as microservices, containerization, and horizontal scaling.
- Setting maximum file upload limits and implement efficient data processing algorithms to handle large datasets. Monitor system performance metrics such as CPU utilization, memory usage, and response times to identify bottlenecks and optimize resource allocation.
- Conducting load testing and performance profiling to validate scalability and performance under different usage scenarios. Optimize database queries, caching strategies, and network bandwidth utilization for maximum throughput and responsiveness.

- **Supported Wearable Devices:** Defining the models and versions of Apple Watch, Fitbit, and Garmin devices supported, including specific data formats each device exports (e.g., JSON for Apple Watch, CSV for Fitbit).
- **Third-party Services:** Listing any third-party services the system integrates with including the versions and communication protocols used (e.g., HTTPS).
- **Browser Compatibility:** Specifying the web browsers and versions supported by the application (e.g., Chrome, Firefox, Safari), ensuring accessibility across different user platforms.

2. **Security:**
    - Employ robust security measures to protect user data and ensure privacy. Implement encryption for data transmission (TLS/HTTPS) and storage (AES/PGP) to prevent unauthorized access.
    - Use industry-standard authentication and authorization mechanisms to enforce access control and user

Permissions. Implement role-based access control (RBAC) or attribute-based access control (ABAC) to restrict access to sensitive data and functionalities.

- Conduct regular security audits and vulnerability assessments to identify and mitigate potential security risks. Keep software dependencies and libraries up to date to patch known vulnerabilities and security vulnerabilities.

- Implement secure coding practices such as input validation, output encoding, and parameterized queries to prevent common security vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

1. **Performance Optimization:**
    - Improve data processing efficiency to minimize processing time and server load. Optimize algorithms for data preprocessing, feature extraction, and model inference to reduce computational complexity and resource utilization.
    - Implement caching mechanisms and data indexing strategies to accelerate data retrieval and query execution. Utilize in-memory databases (e.g., Redis) for caching frequently accessed data and results.
    - Leverage asynchronous processing and parallel computing techniques to distribute workload across multiple CPU cores or nodes. Implement task queues and job schedulers for background processing of long-running tasks and batch jobs.
    - Monitor system performance metrics in real-time and use performance profiling tools to identify performance bottlenecks and optimize critical code paths. Continuously benchmark system performance against predefined service level objectives (SLOs) and iterate on performance improvements based on user feedback and usage patterns.
    - **Data Processing Time:** The system should process uploaded data files within a timeframe that scales linearly with the file size. For example, processing a file of up to 50 MB should take no longer than 2 minutes under normal server load conditions.
    - **System Availability:** Aim for a system availability of 99.9% uptime outside of scheduled maintenance windows, ensuring researchers have reliable access.
    - **Response Time:** Ensure the web application's response time does not exceed 2 seconds for data retrieval actions and 5 seconds for data processing and prediction tasks under normal usage conditions.

    - **Login Failures:** Document the system's response to failed login attempts, such as displaying an error message after three unsuccessful attempts and locking the account for a specified duration or until manual verification.
    - **Data Upload Issues:** For issues related to file uploads (e.g., format not supported, file too large), the system should provide a clear error message detailing the problem and suggesting solutions.
    - **Processing Delays:** In case of processing delays due to high server load, the system should inform the user of the expected wait time and offer the option to receive an email notification upon completion.
    - **Unauthorized Access Attempts:** Document how the system detects and handles unauthorized access attempts to restricted areas, including logging the attempts, notifying administrators, and blocking the offending IP addresses if necessary.
- All of these requirements were generalized in the following requirements that we are using in our Requirement test Matrix
    1. Pages Load and show all main elements
    2. User can log In and Sign up to the page/ User can log out of the page
    3. User can upload their data
    4. User can download their data transformed into csv
    5. User can manage the uploaded Data
    6. User can use the ML models
    7. User can process a great amount of data
    8. Security : Allow access to certain pages only when user is logged in

3. Use Cases

Use Case 1: Data Upload and Processing

Actors: Researcher (Primary user)

Preconditions: The researcher is authenticated and logged in.

Main Flow:

1. The researcher selects the option to upload data from the navigation bar.
2. The system presents a file upload interface, including options for Fitbit, Apple Watch, and Garmin devices.
3. The researcher selects the device type and drags and drops data files into the drop zone.
4. The system validates the file format and begins the upload process, displaying a progress bar.
5. Upon successful upload, the researcher is prompted to initiate data processing.
6. The researcher customizes data processing options, including selecting specific data points and processing parameters.
7. The system processes the data, leveraging machine learning algorithms for prediction and analysis.
8. Upon completion, the system notifies the researcher and presents a summary of the processed data.

Postconditions: Processed data is available for review and further analysis.

Exception Paths:

- If the file format is not supported, the system notifies the researcher and aborts the upload process.
- If processing fails due to a system error, the system logs the error and notifies the researcher, offering options to retry or contact support.

Use Case 2: User Authentication and Session Management

Actors: New user, System

Preconditions: None

Main Flow:

1. The new user selects the option to register on the BEAP Engine homepage.

2. The system presents a registration form requesting essential information (name, email, password).
3. The user completes the form and submits it.
4. The system validates the provided information and creates a new user account.
5. The user receives a confirmation email and activates the account.
6. The new user logs in with the registered credentials.
7. The system authenticates the user and grants access to the application.

Postconditions: The user is registered, authenticated, and logged into the system.

Exception Paths:

- If registration fails due to incomplete or invalid information, the system notifies the user and requests corrections.
- If login fails (e.g., incorrect password), the system provides feedback and the option to reset the password.

Use Case 3: Error Handling and Feedback Mechanism

Actors: Researcher, System Administrator

Preconditions: The researcher is using the system.

Main Flow:

1. The researcher encounters an error or issue while using the system (e.g., data upload failure).
2. The system captures the error, logs detailed information for troubleshooting, and provides a user-friendly error message to the researcher.
3. The researcher uses the feedback mechanism to report the issue, providing additional context and details.
4. The system administrator reviews the reported issue, investigates the cause, and initiates corrective actions.
5. The researcher is notified about the resolution status and any steps required on their part.

Postconditions: The issue is resolved, and the researcher can continue using the system.

Exception Paths:

- If the system fails to capture or log the error, manual reporting mechanisms (e.g., email support) are available for the researcher to report the issue.