ID3: Mini-milestone list

Ticket issues/ mini-milestone list:

Open Ticket issues (Missing activities):

- **Add run time assertions to backend code #137**
- **Research the backend with focus on possible tests #134**
- **Implement Home page from Design #133**
- **Design Home Page #132**
- **Complete the implementation of the File Upload Page #130**
- **Create a document with a plan to test the backend #127**
- **Implement Navbar Component in React with Typescript #120**
- **Implement Predicted Data Page #119**
- **Design the Navbar #117**
- **Design the Predicted Data Page #116**
- **filedropzoneControls should have a default radio button selected so they dont upload a file before a radio is selected #107**
- **Adjust CI pipline to only run playwright tests on merge requests not evey commit #103**
- **Login/SignUp Page functionality #91**

Closed Ticket issues(Completed activities):

- **Implement a reusable Help component #144**
- **Research on how to perform Mocking in our system #138**
- **Create a spike protoype using a profiler #136**
- **Create a document with a plan on how to use a Profiling tool #135**

- **Look into Profilers - Profiling tools #131**
- **Update Requirements Test Matrix #129**
- **Update General Test Matrix #128**
- **Research on how much time uploading a file of 500 MB would take #126**
- **Modify playwright pipeline to run in docker #125**
- **Create production version of our docker files and docker compose #124**
- **Create Github Secrets for sensitive credentials #123**
- **Research and Implement Automated Deployment #122**
- **Implement AuthContext for User Authentication in React with TypeScript #121**
- **Implement the UI for the Processed Data Page #118**
- **Design the Processed Data Page #115**
- **Create config file for local/prod environments #66**

Updated PROJECT ARCHITECTURE

Our system is crafted for efficiency and security. It employs a modular and layered design, consisting of three main components:

UI:

Drives the user interactions. Implemented using JavaScript, React Library, Material UI Library(MUI)

- **JavaScript, React, and TypeScript:** The frontend of the application is built using JavaScript, the React library for building user interfaces, and TypeScript for static typing to enhance code reliability.
- **Material UI Library (MUI):** Material UI is used to provide a consistent and visually appealing design across the application. It includes pre-designed components that can be easily integrated into the React application.
- **Jest Tests:** Jest tests are implemented to ensure that the frontend components render correctly and that there are no rendering errors. These tests help maintain the quality of the user interface.

Core Modules:

Serves as an intermediary between the UI and the backend modules. This is where most of the logic resides. We employ the usage of Redux. Communication between the UI and BEAP Engine's API is handled by HTTPS requests. When the app is loaded, Redux requests all the required data and stores it. This makes the web application faster after initial loading since it does not need to request the same data every time a page is destroyed. The entry points are the representational state transfer (REST) application program interface's (API), which enable other applications to interact with the BEAP Engine such as web, and mobile applications.

Backend Modules:

Split into two....

1. **R Repository:**

Using the R programming language to process raw Apple Watch and Fitbit data. Separating this module from the core library, we can modify the processor and predictor approaches as well as update the R version without the need to compile the entire BEAP Engine project.

The R repository consists of two sub-modules: data processor and feature extractor. Neither of these sub-modules store user data permanently. However, upon finishing their tasks, the raw data is cleared from memory and the processed data is returned to the core module to be persisted encrypted to the data repository. The Data Processor, part of the R Repository in the BEAP Engine, transforms raw Apple Watch and Fitbit data into a readable format, presenting key metrics in an Xlsx file. Meanwhile, the Feature Extractor, also in the R Repository, employs machine learning algorithms like Random Forest and SVM to categorize user activities such as lying, sitting, walking, and running. Together, these modules enhance the system's ability to process and derive meaningful insights from user-generated data efficiently.

1. **Data Repository:**

Utilizing PostgreSQL, we securely store user data, with the added layer of encryption using the **pgcrypto** module. User data, including raw, processed and predicted data is kept encrypted using symmetric key algorithms in the data repository and the key is kept confidential in the BEAP Engine to ensure security

properties. All the queries for Create, Read, Update, Delete (CRUD) operations are handled inside the data repository module, and we use the PostgreSQL functions. In this way, we have separated our data logic from the other parts of the system. By doing so, we have the freedom to modify the user data whenever needed, for example encrypting data before storing it. Additionally, running expensive and time-consuming queries in this layer leads to better overall performance. Modifying the queries and creating new views from the existing data without affecting the other modules of the system is another reason for our separation.

Data Flow:

1. **Export Data:**
    - Users export their data from Fitbit or Apple Watch in XML or JSON format.
2. **Frontend-Backend Interaction:**
    - The frontend sends exported data to the backend through RESTful APIs.
3. **R Repository Processing:**
    - The R Repository processes raw data, converting it into a readable format.
4. **Data Storage:**
    - Processed data is securely stored in the PostgreSQL database within the Data Repository.

Modularity:

1. **Loose Connections:**
    - Components like the Core Module, Data Repository, and R Repository are loosely connected, allowing for low-cost modifications in each block without affecting others. This modularity enhances flexibility.

Navigation and Pages:

1. **Navigation Bar:**
    - The Navigation Bar provides links to different pages, including File Upload, Processed Files, Home, Logout, and Predicted Files.
2. **Individual Pages:**

- Each page, such as the File Upload Page and Processed Files Page, has been implemented with styles and Jest tests, ready for further functionality.

Security Measures:

1. **Transport Layer Security (TLS):**
    - TLS protocol secures connections between different components, ensuring data privacy during transmission. It is used for securing connections between the client and BEAP Engine, UI and backend, and BEAP Engine and R/Data Repositories.
2. **Symmetric Key Encryption:**
    - User data, including raw, processed, and predicted data, is stored encrypted using symmetric key algorithms. The encryption key is securely stored on the server.