ID2: Mini-milestone list

Ticket issues/ mini-milestone list:

Open Ticket issues (Missing activities):

- Everything is closed and done

Closed Ticket issues(Completed activities):

- **Create Activity Diagram for File Upload Page (UI)**
- **Bug: Processed Data Page has Button that sends users to the wrong page**
- **Create Basic Playwright tests for ID2 pages**
- **Create Mocking Pages that intentionally fail our UI unit tests**
- **Create/Update Test Matrix - T. Cases - Requirements**
- **QA App.tsx Navbar.tsx HomePage.spect.ts Navbar.spec.ts Navbar.test.tsx**
- **QA PredictedDataPage.tsx, ProcessDataPage.tsx, PredictedDataPage.test.tsx, ProcessDataPage.test.tsx**
- **QA FileUploadPage.tsx, HomePage.tsx, FileUploadPage.test.tsx, HomePage.test.tsx**
- **QA FileDropZone, FileDropControls, *.test.tsx, *.test.tsx**
- **Design the Login/SignUp Page**
- **Playwright NavBar tests**
- **Jest Snapshot Tutorial/ Documentation**
- **Rollbar documentation/Tutorial**
- **Create Playwright Navbar Tests**
- **rollbar spike prototype**
- **Design the File upload page**

- **File Upload Page Functionality**
- **Create a branch naming convention**
- **Create a detailed documentation on running local environments using docker**
- **Setup new CI pipeline for playwright**
- **Connect the backend with the R repository**
- **Move R and Java backend to our repo and setup Docker Containers #62**
- **50 - Results of Code reviews done with the Code Checklist**
- **47 - Update Test list**
- **48 - Install React Rollbar #54**
- **49 - Update Test Matrix for ID2**
- **Add pull request templates**
- **25-Predicted Data Page #27**

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.