1-Mini-milestone list ID1

Ticket issues/ mini-milestone list:

Open Ticket issues (Missing activities):

1-Mini-milestone list ID1:

Creating mini milestone list for ID1

2-List of risks:

Top 10 Risks

3-Risk report

4-System Requirements

7-Report of testing results:

Defect report for defects founds, planned tests, current and future testability investments, hooks if used, logging mechanisms, and description of automated tests

8-Statement of requirements to incorporate in next deliverable

9-Plan for ID2:

Design and architecture , documentation of anticipated steps (mini-milestones) in carrying out the project and a critical path through this network

10-Activity report:

Summarizing work undertaken by each group member to date, and the activity log (giving time estimates).

12-Smoke Tests:

not necessary for ID1 if ID1 is a spike prototype:

checks that the basic functionality is in place. make sure that software is in a state that is useful enough to share and test

17-Figure out how the TLS of the website works:

How does it set up the TLS?

25-Predicted Data Page:

- Implement the basic UI
- Refer to old repository
- Make sure you write unit tests and make sure you get approval before merging
- Estimate: 2 Days

33-Testing Matrix:

- articulate features and test coverage
- figure out where to host
- is it possible to create hyperlinks to specific tests or test suites in the test document

40-Test list:

A document where each test and test suite is described so that it can easily be referenced in a test matrix.

41-Rollback Log

42-Multilevel Logging:

Install Rollbar to project.

Set up logging artifacts. Where should logs be saved? Need something that we can use to generate an artifact to submit with each deliverable.

43-Crashlytics

Install and configure crashlytics

44-Testing Strategy Document

This needs to be a living document that we can add to as we learn more.

Includes
-what to test
-what to test for
-best practices

Juan said Mostly done (missing logging strategy)

45-Java & React Spike Prototype

make small hello world application that uses React for frontend and Java spring in backend.

Must see if it is possible:
-must support testing and test report methods
-must be able to support logging tools

46-Java Backend & R module

must demonstrate by creating prototype that it is feasible to interact with R modules provided by Dr.Fuller using java spring

- need to figure out testing and logging for this

Add pull request templates

Add templates for pull request so we know what to put

Closed Ticket issues(Completed activities):

test issue:

Created an issue to learn more how ticket issues on github works

19 - CI Pipeline Setup:

Expected Workflow

1. On Pull Request ready, Build and Test the code
2. On Build and test
    a. Install Server Dependencies
    b. Install Client Dependencies
    c. Run Unit Tests
    d. Run E2E tests

For this instance since we will only work with react for now we will only have it setup to Install Client Dependencies -> Run Unit Tests solely on react

5- Documentation of team personnel roles:

Assigned team member to different roles

6- Documentation of reviews and meetings with stakeholders:

Notes from the meeting with stakeholders and Peer reviews/ code inspection and reviewing documentation

11 - CI Set Up Documentation:

Ralph performed a tutorial, full recorded video explanation
here: https://drive.google.com/file/d/1UAYqCBXRLoIy7WQUzr1Kg1oIkBrUJZgL/view

Doc is explained here: https://github.com/UniversityOfSaskatchewanCMPT371/term-project-2024-team-3/wiki/ID-0

13-Figure out where the node.js server is (High Priority)

How does the backend work? Do we need to do any real work for this? Where is it? Report findings on the ticket/issue

Eric Found the backend: https://github.com/walkabillylab/BAEPLabEngine

14 - Figure out how the authentication works and files go

How does the authentication work right now? Report findings on the ticket/issue

Ralph said Related to Issue 18 and gave below explanation:

The authentication is currently manually handled in java springMVC backend that is provided with the project.

The associated tables are

`tbl_login_user`

`tbl_access_group`

`tbl_role`

`tbl_user`

`tbl_user_tbl_access_group`

`tbl_user_tbl_raw_data`

`tbl_user_tbl_role`

`tbl_incorrect_logins`

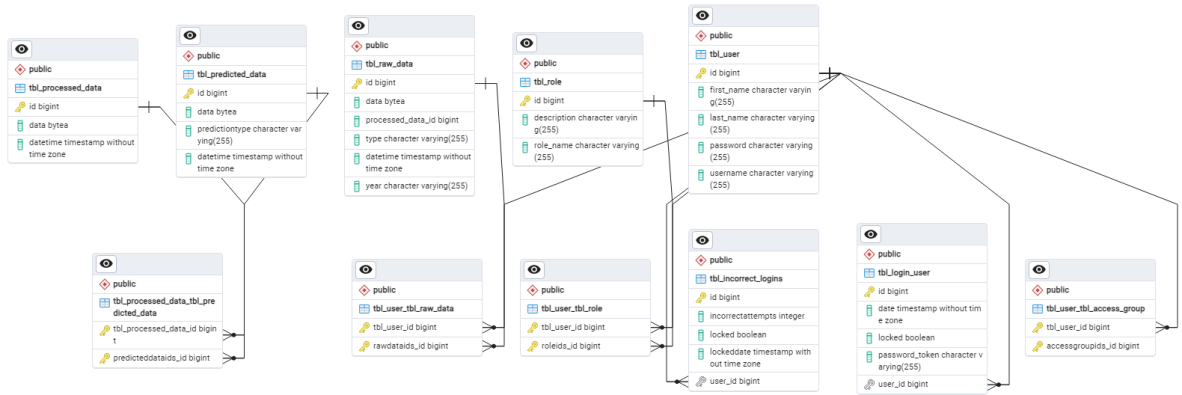
The files are parsed in the backend and associated data is stored in its respective tables for processing/predicting.

- Predicted Data is stored in `tbl_processed_data`
- Processed Data is stored in `tbl_predicted_data`
- Raw Data is stored in `tbl_raw_data`

15 - Figure out the tables

What are the tables used, its dependencies. Report findings on the ticket/issue

Ralph Provided table below:



16-Figure out how R Library is connected with the app

How does it process the files and predict. Where does it call it? Where? When? How? Do we need to do anything? Report findings on the ticket/issue

Ralph:

For our provided backend, we have to supply the path in the environment variables in postgresql.properties and r_repo.properties files.

Make tomcat as the RCode path directory owner (after tomcat user being created and tomcat is installed)

This would be r_repo.path={path-to-R-repo}

Also have to make sure the directories exists on the server

r_repo.path/zips
r_repo.path/tmp
r_repo.path/fitbit/data/output
r_repo.path/fitbit/data/raw
r_repo.path/applewatch/data/output
r_repo.path/applewatch/data/raw

18 - Initialize Repository w/react.ts

We need to setup our repository. This should include the following:

- Make sure our repository has React + Typescript setup
- Setup our app structure
  - This includes having a directory for only frontend, backend, r, and docs
- Required dependencies are installed
  - Refer to [this] (https://github.com/walkabillylab/BeapEngineUI/blob/b01d0d82860a96cd76db9d218c7ab50b8ead1f0c/package.json#L5-L30) for a list of dependencies used
  - There should be more dependencies but we will talk about which ones we will need to install as well

20 - Install Linter and Prettier and implement with pre-commit hooks

Pre-commit hooks are pretty much run when you try to commit. If the linter/prettier fails then the commit will not go through and make you fix dem errors.

- Install Linter and Prettier packages
- Run Linter and Prettier as a pre-commit hook
  - Linter Handles Static Analysis
  - Prettier ensures code is consistently formatte

Ralph: Install linter and prettier and implement with pre commit hooks #29

21- Implement React Router

- Setup Router for File Upload Page, Predicted Data Page, Process Component Page, Home Page
- Would be good to create empty page components to connect them to the router
- Make sure you write unit tests and make sure you get approval before merging
- Estimate: 1 Day

22-Setup Home Page

- Copy the home page of what Dr. Fuller had
- Don’t implement any login stuff
- Make sure you write unit tests and make sure you get approval before merging
- Estimate: 2 Days

23- Setup File Upload Page

- Upload button, submit button, and dropzone for the files.
- Don’t have to implement it to actually upload
- Add a UI feedback (e.g toast) on upload files to the page
- Make sure you write unit tests and make sure you get approval before merging
- Estimate: 2 Days

24-Process Data Page

- Implement the basic UI
- Refer to old repository
- Make sure you write unit tests and make sure you get approval before merging
- Estimate: 2 Days

26-Implement Nav Bar (is soft blocked by react router issue)

- Setup nav links to lead to Process Page, Predicted Data Page, and File Upload Page
- Appears after login
- Make sure you write unit tests and make sure you get approval before merging
- Estimate: 2

34-Setup Git Branch Protection Rules

Setup rules for protecting branches from unsafe merges

Ralph mentioned Rules to be implemented:

- Enforcing 1 approval before you can merge in development/feature branches
- Enforcing 2 approvals for merges to master and release
- Enforcing CI checks to pass before you can merge
- Require branches to be up to date before merging
- Locking out direct committing to masters

Ardalan and Ralph check and said: Everything is set up and done!

39-Install Playwright

Juan Installed the Playwright and everything is set up and done!

Create Initial Deliverable Checklist:

This is checklist to be reviewed for every deliverable to ensure that all components required are worked on/submitted on time

Kamal created the default checklist for deliverables

Create a preliminary Code Inspection Checklist

This document will be a starting checklist for code inspections and reviews.

Kamal created the checklist for code inspection and was reviewed by Ralph and Ardalan

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.

2. **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.