ActivePoints Clinician Dashboard Codebase Documentation

GDP Group 15

Joseph Padden - jp3g20

Rory Coulson - rc3g20

Neeraja Jayaraj Menon - njm1g20

Dillon Geary - dgfg1g20

December 2023

1 Introduction to the Codebase

This codebase contains all elements attributed to the Activepoints client dashboard.

As shown, the dashboard's front end is created using React. This interacts with a backend Flask API that interacts with both a Python AI module and the Activepoints PostgreSQL database using the respective APIs. Note the terms 'user' and 'patient' are used interchangeably throughout the documentation.

2 Retrieving Activepoints Data

The server.js file works as a serverside Node.js script using Express.js that handles data requests. Upon viewing the patient select screen, both the Garmin and Fitbit data is loaded via the 'fetchAPData()' function, this can however be moved to a refresh button functionality in the future if preferred. Calls from the React front end can then be handled and the correct data supplied.

'/api/ap_user_ids' returns the list of user_id's available.

'/api/daily_data' given a start date, end date and user_id will return the data (steps, intensity activity minutes, calories, sleep, minimum and maximum heart rate) from the given period of the given user.

'/api/targets' given a user_id will return the latest sleep, step, calorie and intensity targets for the given user. The step goal is currently provided by the World Health Organization recommendations, to link to the user data in the future and reconfigure with a database that provides this missing data.

3 React Front-end

The React front-end handles both the patient-select screen and the dashboard itself.

3.1 Patient Select

Found within src/scenes/PatientSelect, 'PatSelect.js' holds the React script for the patient-select screen. The following React function components are defined:

3.1.1 PatientCard

This is a React component which takes a single patient's data and the navigate function and creates the patient card buttons. When clicked, the dashboard is navigated to with the patient's user ID and name.

3.1.2 CardRow

This is a React component which takes a list of patients' data and a navigating function which takes the user to the dashboard. It returns a row of PatientCards in a scrollable view.

3.1.3 PatientSearchBar

This is a React component which takes a text input from the user to filter of patients in the table, the type of data to filter the table, the function that handles the change of data type to filter by, and a function that handles the change of text input.

3.1.4 FullPatientTable

This is a React component which takes a list of patients, the text to filter the table by, the type of data to filter by, and a navigating function which takes the user to the selected patients' dashboard. Within this component, the patient data is split to extract their name, clinician, and patient ID, which are then passed into the table rows to be rendered.

3.1.5 PatientTable

This takes a list of patients and a function which takes the user to the selected patients' dashboard. It renders the PatientSearchBar component with the FullPatientTable component under it.

3.1.6 PatSelect

This is then responsible for returning the whole patient-select page. It currently receives all user ID's and names from the Activepoints PostgreSQL and passes them into the PatientTable and CardRow for display.

Other files in the PatientSelect folder are all for styling:

- fonts.css contains styling details for the font on the page
- PatCard.css contains styling details for the patient cards
- PatTable.css contains styling details for the patient table
- PatSelect.css contains styling details for the patient select screen as a whole

3.1.7 Future User Management

At the moment, this screen assumes all users on the Activepoints PostgreSQL belong to the same clinician. For clinicians to only be able to see the data of their clients, a database table will have to be made by Activepoints that holds the relations between clinicians' accounts and the users on Activepoints back-end. Following this, the '/api/ap_user_ids should be altered so that only user_ids of the current clinicians' clients should be returned.

3.2 Dashboard

Found within frontend/src/scenes/Dashboard, 'Dashboard.js' holds the React script for the dashboard screen. Multiple React components are used from the 'src/components/' folder:

3.2.1 GoalStatCard

A component which displays how well a user has done in the last 7 days. This is done with a wheel which is filled as the user gets closer to their goal. The colour of the wheel signifies whether a user has reached their goal and is calculated using the percentage, with red to green indicating the degree of goal completeness and gold indicating the goal has been reached.

3.2.2 Graph

The graph takes the filtermode(steps, sleep, calories, heart rate or intensity), scale, theme, userID, goals and predictions as arguments. It then allows for a user to select a start and end date and displays the data for the given userID between the two dates. Data is requested from the ExpressJS backend using the userID and chosen dates. If heart rate has been selected, both the minimum and maximum heartrate of the selected days will be fetched and displayed as a stacked graph. If the end date that the clinician has selected is the same as the current date then the AI forecasting predictions using the users' historical data will be displayed with a faded colour and added in the graph legend.

3.2.3 Header

The header takes a user's name, a function to return to the patient select page, and the selected theme. This returns the header for the dashboard page showing the user's name, clinic name and the button for opening the sidebar. Clicking this button toggles the Sidebar.

3.2.4 OverviewStatCard

The overview stat card will take a data descriptor title (i.e. 'steps'), the user's value for that given field, the user's goal for that given field and a percentage change from the previous week in a string format.

3.2.5 RecStatCard

The recommendation stat card is used to provide a patient's recommendation for an attribute, (e.g. heart rate target zones or step recommendations), which includes displaying the next weeks' predictions and recommended daily increase values if necessary via bullet points. The component takes users' goal data, predictions, maximum heart rate average and a loading flag to build the UI. The card will display a loading visualisation while the app communicates with the Flask API to collect the predictions. The heart rate target zone recommendations however are instead calculated using the maximum daily heart rate values as there are no predictions for this attribute. By clicking on these cards the card will flip to display information detailing how the recommendations are generated for the clinician.

3.2.6 Sidebar

Toggling the sidebar presents Settings, Alerts, Export Data, Help, and Exit Dashboard buttons:

- Settings will present a dialog box for a user to select the accessibility setting they require (functionality contained in 'Settings.js'). This component requires a function handling the closing of the popup, a variable representing the state of the popup, the current value of the chosen theme, and a function handling the change of theme passed to it.

- The Help button will open a help dialog box with a user guide (functionality contained in 'Help.js'). The user guides are contained in the folder '/userGuides/'. This component requires a function handling the closing of the popup and a variable representing the state of the popup to be passed to it.

- The Alert button will open a dialog containing any alerts or updates regarding new data for syncing (given there is new data to be synced). As this functionality hasn't been fully implemented yet, it currently states there are no errors in the dialog box. Completing this functionality would involve the Alert component taking two parameters, one of the error message to display and one of a function that triggers when new data is available on the database. This function should also render a refresh button to sync the dashboard (current functionality contained in 'Alert.js'). This component requires a function handling the closing of the popup and a variable representing the state of the popup to be passed to it.

- Export Data allows the user to download all the patients' data in the form of either a CSV or JSON (functionality contained in 'ExportPopup.js' and 'Header.js'). This component requires a function handling the closing of the popup and a variable representing the state of the popup to be passed to it.

- Exit Dashboard takes the user back to the patient-select screen with the navigation function passed into 'Header.js'.

3.2.7 Patient Badge

The folder /PatientBadge/ contains files 'BadgeIcon.js' and 'BadgePopup.js'. Code for the appearance of the icon itself and how the group is chosen can be found in 'BadgeIcon.js', and code for the popup dialog functionality can be found in 'BadgePopup.js'. BadgeIcon requires the chosen theme, patient name, and patient ID. BadgePopup requires a function handling the closing of the popup, a variable representing the state of the popup, the patient group, and a grouping explanation.

3.2.8 Dashboard Theming

The application uses the library ThemeProvider to render the chosen themes. Colours for each type of text and component background for each theme can be found in 'Theme.styled.js'. Custom components created for these themes are found in 'DashboardComponents.styled.js' and 'PatientSelectComponents.styled.js' depending on whether the components are for the dashboard screen or patient select screen respectively. 'Global.js' contains theming for the whole application (anything that should stay the same for both the patient select and dashboard screen).

3.3 Future Work

The availability of data available in the PostgreSQL database could be updated to integrate further functionality into the dashboard. Regarding current issues with data that impact the dashboard, the Fitbit datasets currently lack a step goal field. As this data is available, its integration would lead to step goals being better aligned for all users of the system. This will likely require the database table that stores the step goal data to be included in the PostgreSQL database and then be linked together within the ExpressJS App.

Within the Graph React component, there is potential to bring in functionality where when one clicks on a bar representing a day, the clinician will see an hourly breakdown of that day. To have this functionality, however, hourly data must be added to the PostgreSQL backend by adjusting the data access to the Fitbit and Garmin APIs. To activate this, remove the if statement from line 526 of graph.js, for now, dummy data is in the place where data should instead be gathered from the API.

The heart rate target zone recommendations are calculated using the percentage of maximum heart rate values, following the calculations provided here: `https://trainingtilt.com/how-to-calculate-heart-rate-zones`. Future work could investigate recommending based on the resting heart rate values instead that could provide a greater level of certainty to the recommendations, which would require gathering more user data by adjusting the data access for the Fitbit and Garmin APIs.

4 AI Module

The AI module is composed of multiple Python files and Juypter Notebook scripts that contain machine learning pipelines that run on the client and public collected Fitbit and Garmin datasets.

4.1 Data Cleaning and Feature Engineering

The datasets must be loaded into the repository within a '/data/' folder with further detail specified in the README.md instruction files within the codebase. There are a number of cleaning and feature engineering scripts within '/data_clean_and_exploration/' folder that aim to restructure and further extract important features useful to the models while removing excess noise. It's important to run all these notebook scripts before modelling as the following models will rely on the output of these.

4.2 Forecasting

4.2.1 Ensemble Models and Configuration Files

Found in the 'backend/ai-module/AI/Forecasting/models/' folder, two ensemble models, XGBoost and RandomForest, have been built for time series regression forecasting on a patient's historic data. These were found particularly well-suited due to being able to learn non-linear patterns and work well with relatively small datasets. Run the forecasting with 'run.py' within the /Forecasting/ folder and update the 'config.py' file to adjust how the model is run. Within the configurations, via the 'FORECAST_AND_VALIDATE' variable, one can run only some of the training data and leave the rest as validation and testing data so one can extract an RMSE score and get a preview of how well the model will do in production. Then in production set the configurations, via the 'FORECAST_FUTURE' variable, to run on all the data and produce future predictions that aren't known.

4.2.2 Running Loaded Models and Future Forecasting

The forecasting models will be saved and can be loaded, via the 'LOAD_SAVED' variable, and get future predictions on an older model. This allows one to retrain the models however much is considered necessary and allows for saving backup models if one model outputs poor predictions. The future predictions can be extended to any timedelta but this is set as default to the next week, via the 'FUTURE_PREDICTION_DURATION' configuration variable.

4.2.3 Feature Engineering and Plotting

When running 'run.py', the data is pulled and goes through further feature engineering to extract datetime features that were found to be particularly useful for improving the prediction accuracy. Note the best results came from running on two months of hourly public data. The machine learning pipeline will output plots of the model results and can save the predictions for use in the dashboard.

4.3 Grouping

4.3.1 K-Means Model and Configuration Files

Found in the 'backend/ai-module/AI/Grouping/' folder a K-Means model has been built to cluster patients on different features within the datasets. Once configured the 'config.py' file similar to the forecasting, one can run 'group.py' to cluster and label each datapoint based on the selected features. There's also an option to run new patient datapoints on a saved model to get labels on values without retraining. The number of clusters can be adjusted for the model by updating the 'NUM_GROUPS' configuration variable, which should be set to the number of groups/badge values you want to include.

4.3.2 Sorting the Clusters

The labels are currently sorted for when grouping two features, for example, if using steps and intensity as features the function 'sort_cluster_labels' in 'group.py' will sort each cluster by the distance from the origin, which in this example will correlate to the fitness of the patient. This is converted for the dashboard as a badge letter, with group A being the highest fitness group.

4.3.3 Data Transformations and Scaling

During the machine learning pipeline, the data will be transformed to unskew the features selected via a boxcox transformation where the lambda values are set in the input data configuration file for different features, with the aim to reach zero skewed features which will help improve the clustering model's performance. Afterwards, the features are standardized, setting the mean to 0 and a standard deviation of 1. Without scaling most clustering models will be biased towards ignoring the features will a smaller scale while being biased towards the features with larger ones. This will be plotted by the 'group.py' to allow for analyzing the feature distributions before the data is fed into the model.

4.3.4 Extracting the Group Badge

After running 'group.py', run 'get_user_grouping.py', making sure to update the 'user_id' you want, to get the labels for a specific patient of the dataset you trained on within a specified time period. The patient's labels are then averaged over a time period specified and a badge letter is outputted to be used in the dashboard.

4.4 Recommendations

The recommendations script can be found in 'AI/Recommendations/recommendations _from_forecasting_and_goals.py' file. This file uses the saved predictions from forecasting and the patient's goals over the same time period to create a recommendation on how much to change from what they're currently doing to reach their goals, with a daily increase recommendation if needed that is then shown in the dashboard to the clinician. This file displays the logic for generating these recommendations but has now been moved to front-end 'RecStatCard' to reduce the communication overhead.

4.5 Explainable AI

The grouping explanations are output using a library called SHAP. This is a tool that is model agnostic and therefore works with any type of machine learning model if needed. The file 'grouping_with_shap.py' contains two functions local_exp() and global_exp(). The former outputs an explanation of the patients individual grouping given the transformed data, predictive model, and patient ID. The latter function outputs a general explanation of the grouping as a whole given the transformed data and predictive model. While both functions work, the current model only uses the local explanation due to the limited features of the dataset, making global explanations trivial. Given there is data for the grouping with an increased number of features, global_exp() would also be suitable to include. Both functions output to the terminal the textual interpretation of the technical output from the library. The function is called in 'src/api/apiInterface.py' within the 'getGroupShap()' function to communicate the textual output to the dashboard for visualisation in the badge popup.

4.6 Flask AI API Interface

A Flask API can be found in 'backend/ai-module/src/api/apiInterface.py' that serves the React app function calls from 'frontend/src/utility/apiCommunicator.js.' The API interface uses the AI module files with certain configuration setups for use in the dashboard. It has also been connected to the PostgreSQL database for running the XGBoost forecasting model on the real users' data while the grouping is currently based on an example public dataset userID. One can adjust the configurations in this file to change how the app makes its predictions and groupings.

4.7 Future Use and Development

Although heart rate has been grouped on, using just the limited daily minimum and maximum values against the intensity lacks significant meaning for a clinician. Further work has already looked at using heart rate minutes data from public datasets. Experimental notebooks can be found in 'AI/notebooks/Recommendations/HeartRate' that explore calculating more useful cardiovascular fitness indices including resting heart rate and heart rate recovery. However, only initial work has explored this and future work could include working with a specialised clinician to find out the most useful applications for these grouping results. Grouping could also be developed by focusing on different features, for example, grouping patients by their likelihood of achieving their goals could provide value for a clinician.

Forecasting could be developed further in the future when a greater quantity of data is gathered, by experimenting with deep learning models including RNNs and LSTMs that require large datasets but could exceed the current ensemble models performances. The forecasting predictions could also be used to estimate when patients could expect to reach their goals if they keep up their current progress, providing value as a motivation tool.

Lastly, the AI grouping is not currently connected to the PostgreSQL database and instead can be run on client and public dataset CSV files with a set userID. This can be updated by querying all the user datapoints in the database that will be grouped on and feeding this into the grouping model pipelines instead of the CSV data.