Julie Huang

Assignment 4 - OSS Analysis and Design

4/6/2018

Glucosio: Workout Mode

GitHub repo:

https://github.com/Glucosio/glucosio-android

Use cases:

- Set to workout mode
- Check glucose level
- Exit workout mode
- Check ketones
- View workout glucose level history

Use Case: User opens “Workout Mode”

Iteration: 1

Primary Actor: Diabetic user

Goal in Context: To set the mode to “Workout Mode” to assist with workout

Preconditions: User has logged on to own account (if needed); personal data is synced with account

Trigger: User would like to work out and use Glucosio to help track blood sugar

Scenario:

1) User opens app and is logged on (or logs on)
2) User finds “Workout Mode” in the menu
3) User opens “Workout Mode”
4) System prompts user to enter pre-exercise glucose information

Exceptions:

- User selects incorrectly in menu

- User accidentally selects “Workout Mode” without meaning to

Priority: High priority, to be implemented after basic functions

When Available: 1st increment

Frequency of Use: Frequent

Channel to Actor: Interface, mobile app

Secondary Actors: none

Channel to Secondary Actors: N/A

Open Issues:

- Should pre-exercise glucose info input be separate from the other inputs, or will they be the same functionality?

Use Case: Check glucose level

Iteration: 1

Primary Actor: Diabetic user

Goal in Context: The user would like to check if glucose level is at a safe range to begin/continue exercise as well as keep track of change from beginning to end of workout

Preconditions: Correct glucose ranges have been configured and entered into systems; user has manually checked blood sugar

Trigger: User has selected “Workout Mode” from menu

Scenario:

1) Users are logged on and in “Workout Mode”
2) User is prompted to enter glucose information
3) User manually checks glucose level
4) User enters information into app
5) Information is logged into system
6) System checks range and informs user if it’s safe to exercise with suggestions

Exceptions:

- User incorrectly inputs glucose information
- User doesn’t input any information

Priority: High

When Available: 1st increment

Frequency of Use: Frequent

Channel to Actor: Mobile app

Secondary Actors: N/A

Channel to Secondary Actors: N/A

Open Issues:

- Will user be able to go back and change their input if they entered incorrectly?
- Take into account measurement units (mmol/L vs mg/dL)

Use Case: Exit “Workout Mode”

Iteration: 1

Primary Actor: Diabetic user

Goal in Context: User would like to return to main functionality

Preconditions: User is currently in “Workout Mode”

Trigger: User wants to exit “Workout Mode”

Scenario:

1) Users are logged on and in “Workout Mode”
2) User would like to exit and return to main functionality
3) User selects “Exit”
4) System returns to main functionality

Exceptions:

- User tries to navigate elsewhere without exiting mode

Priority: High

Frequency of Use: Frequent

When Available: 1st increment

Channel to Actor: Mobile app

Secondary Actors: N/A

Channel to Secondary Actors: N/A

Open Issues:

- How to handle if user forgets to exit “Workout Mode”?

Use Case: Check ketone level

Iteration: 1

Primary Actor: Diabetic user

Goal in Context: Make sure ketone level is safe before beginning exercise

Preconditions: User is currently in “Workout Mode” and is logged in to account

Trigger: User has entered “Workout Mode” and entered pre-exercise information

Scenario:

1) Users are logged on and in “Workout Mode”
2) User has entered initial pre-exercise information
3) If user’s glucose level is above 250 mg/dL is a type 1 diabetic, system prompts user to check ketone level and warns against exercising

Exceptions:

- User does not want to check/enter ketone information

Priority: High

Frequency of Use: Low to moderate

When Available: 2nd increment

Channel to Actor: Mobile app

Secondary Actors: N/A

Channel to Secondary Actors: N/A

Open Issues:

- Checking if user is type 1 or type 2 diabetic
- Incorporate way for user to bypass ketones warning

Use Case: View workout glucose level history

Iteration: 1

Primary Actor: Diabetic user

Goal in Context: User wants to view glucose levels inputted throughout workout

Preconditions: User is currently in “Workout Mode” and has entered information

Trigger: User clicks on “View Workout History”

Scenario:

1) Users are logged on and in “Workout Mode”
2) User has entered glucose information
3) User selects “View Workout History”
4) System displays history

Exceptions:

- There is no history to display--user has not logged anything

Priority: High

Frequency of Use: Low to moderate

When Available: 3rd increment

Channel to Actor: Mobile app

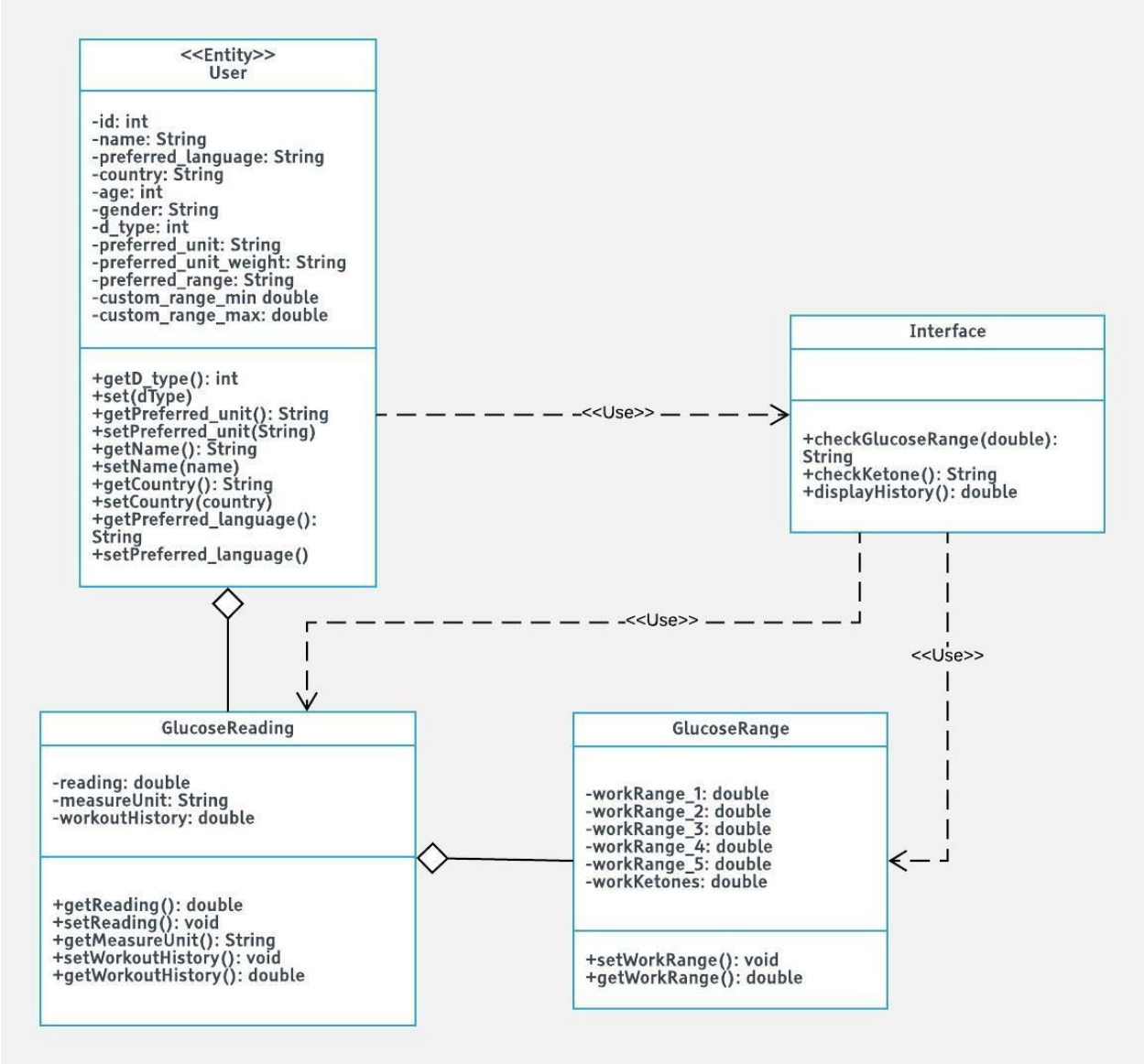
Secondary Actors: N/A

Channel to Secondary Actors: N/A

Open Issues:

- How will users view history? As a graph? Or just by text?

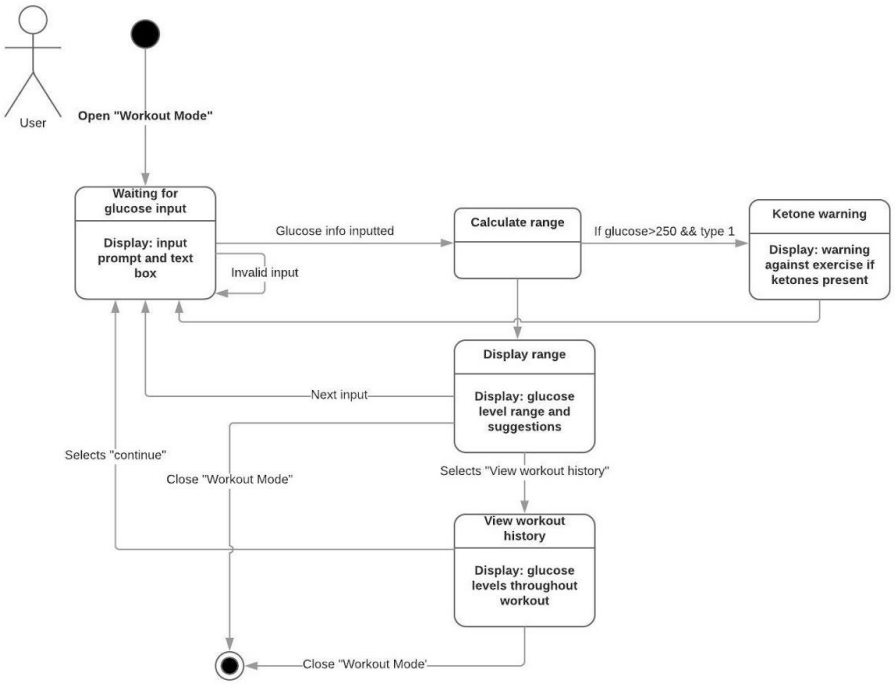
Class Diagram



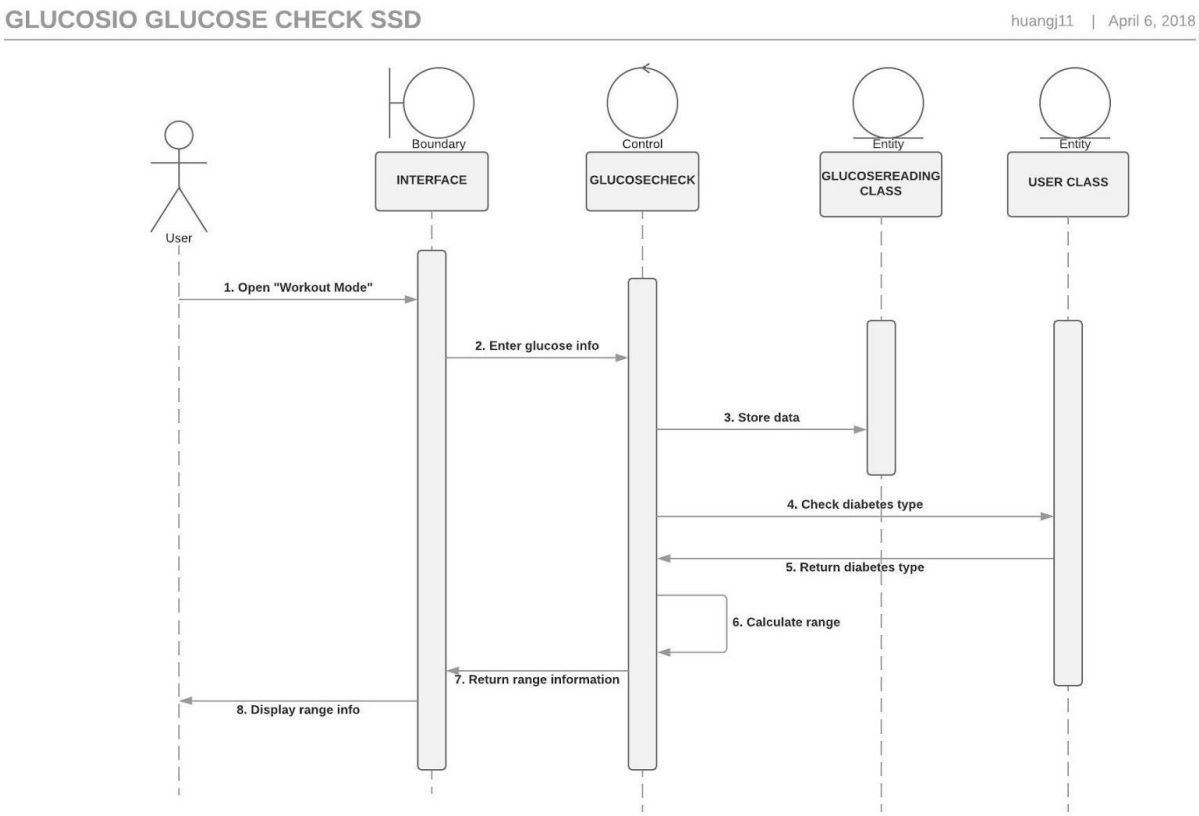
State Diagram

GLUCOSIO STATE DIAGRAM

huangj11 | April 5, 2018



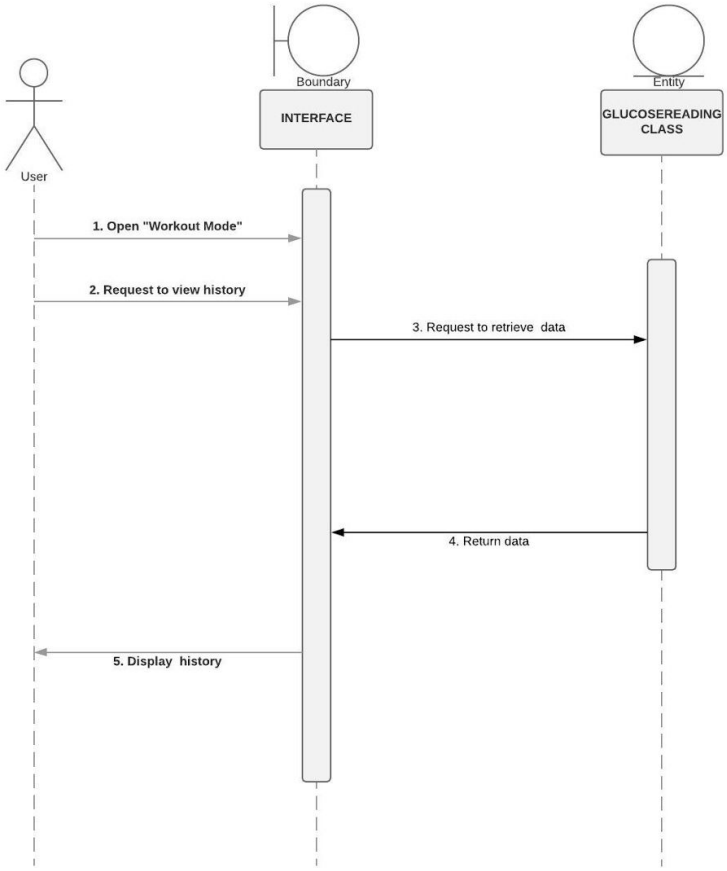
Check Glucose Ranges SSD



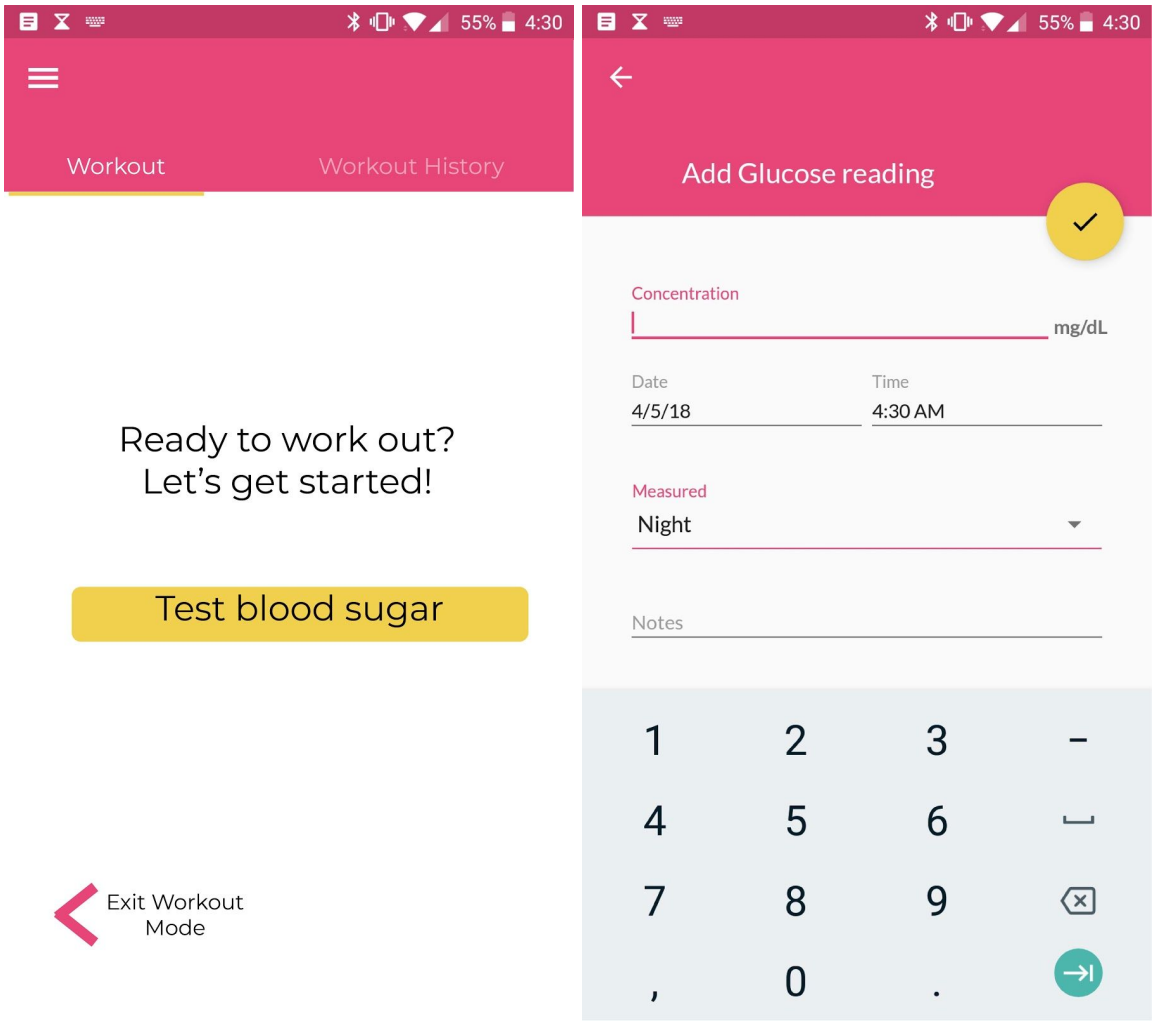
View Workout History SSD

GLUCOSIO VIEW HISTORY SSD

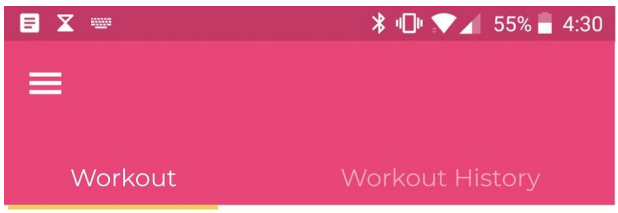
huangj11 | April 6, 2018

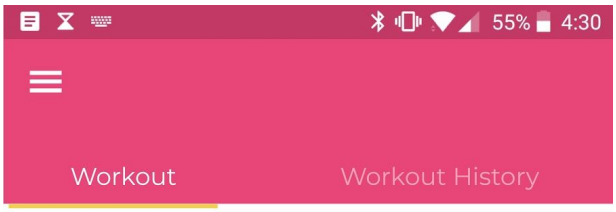


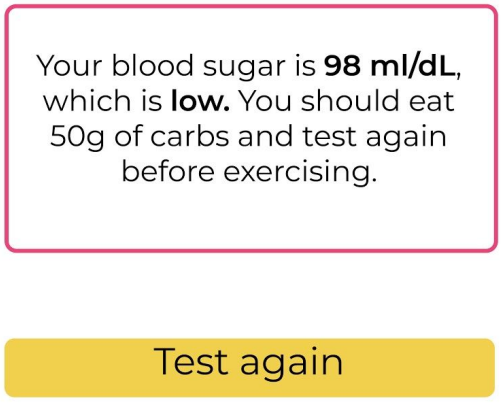
User Interface

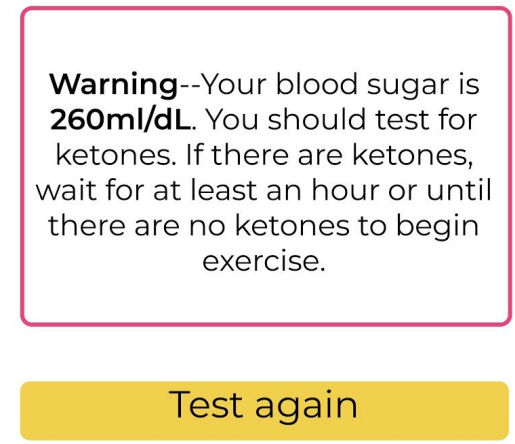


1. Prompt to enter blood sugar upon beginning "Workout Mode"
2. Screen to enter glucose information

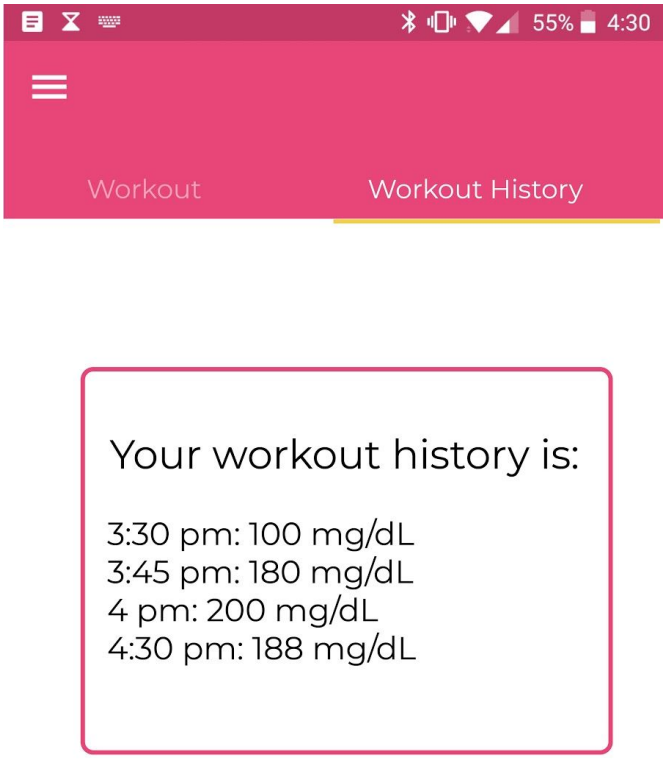








3. Return range information
4. Display ketone warning



5. Display workout history

<u>8 Golden Rules</u>

Consistency

- The added pages are consistent with each other as well as the rest of the app in terms of:
    - Color scheme
    - Layout

    - Method of navigation (the bar at the top to swipe between “Workout” and “Workout History”)

Shortcuts for frequent users

- Easy access to the main functionality through the “Exit Workout Mode” button, rather than manually opening up the navigation bar again

Easy error handling

- If the user misses a required field, the system will prompt the user to enter a valid input

Informative feedback

- If there is an invalid input (ex. , the system will notify the user and prompt them to try again
- When the user goes between pages within “Workout Mode”, the yellow bar highlights the page they are on
- After the user inputs their glucose information, the system will take them to the page that tells them their glucose information, range, and suggestions

Maintain inner locus of control

- Each button/link is has self-explanatory names that suggests its functionality so users will not be surprised by unexpected actions
- Although the app offers suggestions to the user, it’s not intrusive to the user’s desire to work out. Users can still log more glucose levels and the

Easy to reverse action

- There’s a back button on the input glucose information screen that allows users to go back in the case that they selected the button by accident
- Similarly, users can return back to the main page via the “Exit workout mode” button if they selected the page by accident

Reduce short-term memory load

- After entering glucose information, the system displays the level again along with the range information so the use does not have to remember exactly what they inputted
- The history displays all the entered information throughout the session

Dialogue to yield closure

- After entering glucose information, the system displays the information to indicate that the information has been successfully entered

Design

Modularity and encapsulation:

The system is modular and displays encapsulation in that it uses classes to store related information, such as glucose readings, range information, and user information. It also has an interface class that calls and executes functions using data from the classes. Each function will perform one action, promoting modularity. Each class will work on their own set of data, and there will be limited interaction between modules (low coupling). Data such as user information and glucose readings will be private, and only accessible through getter and setter methods.

Elegance and efficiency of algorithms:

Much of the algorithm and database has already been implemented by the OSS, and I will be utilizing them to increase efficiency. The algorithm I intend to implement will be a simple comparison. The glucose ranges ("Lower than 100 mg/dL", "100 – 150 mg/dL", "150 – 200 mg/dL", "250 – 300 mg/dL", and "300 mg/dL or higher") are stored in a data structure, and the system will take an inputted glucose reading and compare with the ranges. As there are a limited number of ranges of comparison, this will not be computationally intensive.

Appropriateness of data structures:

I will be utilizing the database already established in the OSS for glucose reading information. For the glucose ranges, I will use a hash table for easy lookup.

Testing

Unit:

I will test each class and make sure the data structures are working and that they are handling the information correctly. Tests will include boundary cases as well as invalid inputs to see if the system can handle those cases in the appropriate manner.

Integration:

Each module will be tested individually at first to make sure they work, and then be integrated with other components to make sure the interfaces of the modules are matching. For example, the function to input data will be tested first. If that works, and the information is stored properly, the function to compare ranges will be tested next as it utilizes the inputted data.

Essentially, I will be using the bottom up approach to testing and work up to the rest of the system.

System:

To test the system, I will have several outside users test the system, at least one of which will be a diabetic athlete. I will also utilize regression testing to make sure my added feature(s) won't affect the rest of the system.

Debugging tools:

I will use Android Studio's debugging tools that allows me to set breakpoints and test parts of code and preview it amongst other functions--similar to Eclipse.

Test Cases

Functionality Tested	**Inputs**	**Expected Output**	**Actual Output**
Enter "Workout Mode"		User is in "Workout Mode"; system displays prompt for glucose info	
Exit "Workout Mode"		System returns to main functionality	
Check glucose level range	Glucose level	Glucose range and suggestions; Error message if user did not input valid number (ex. Negative number)	

Check to see if user glucose level is above 250 mg/dl	Glucose level	Returns true if above 250 mg/dl; else return false	
Check to see if user is type 1 or type 2	User diabetes type (already stored in system, retrieve)	Returns true if type 1; else return false	
Display ketone warning	Ketone level	System displays ketone level warning if glucose level is above 250 mg/dl and use is type 1	
Display workout glucose level history	Glucose level information	System displays all glucose info from current workout session	