Description:

Design an application that encourages us to be good to ourselves by providing stress-relieving tips and/or exercises and that can monitor stress triggers.

General Requirements:

1. Interface the application with sensor data from an Apple Watch.

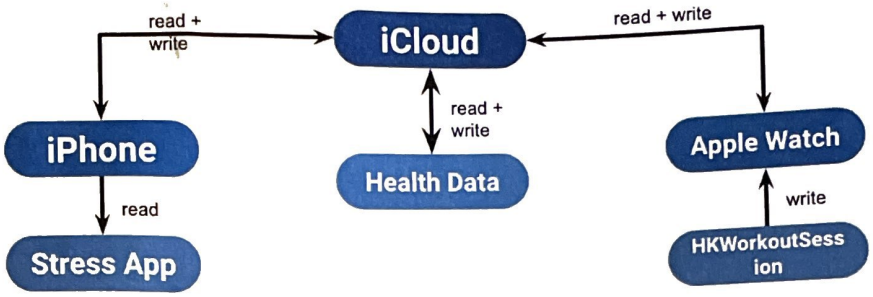
2. Investigate the reasons for stress and anxiety on college campuses.

3. Identify triggers or indicators that might be indicators of stress.

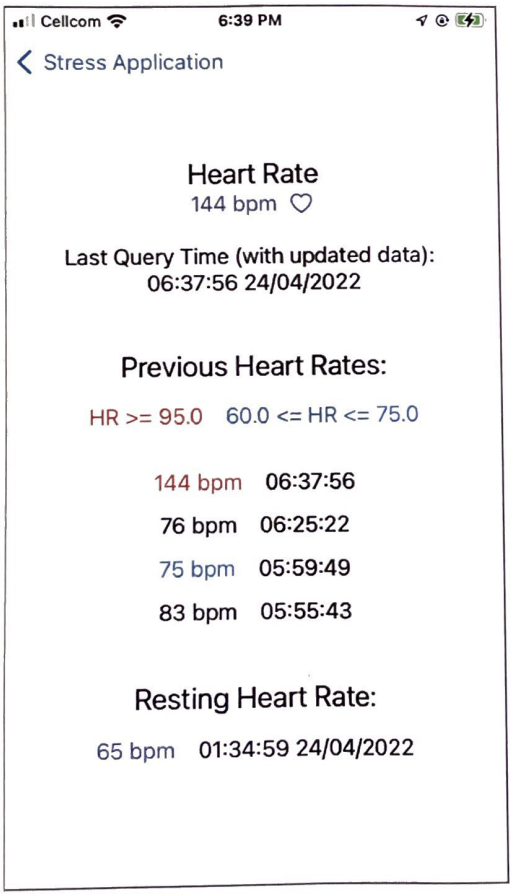
4. Gather health data during workouts and times of rest.

5. Provide historical data to help identify patterns and/or significant changes.

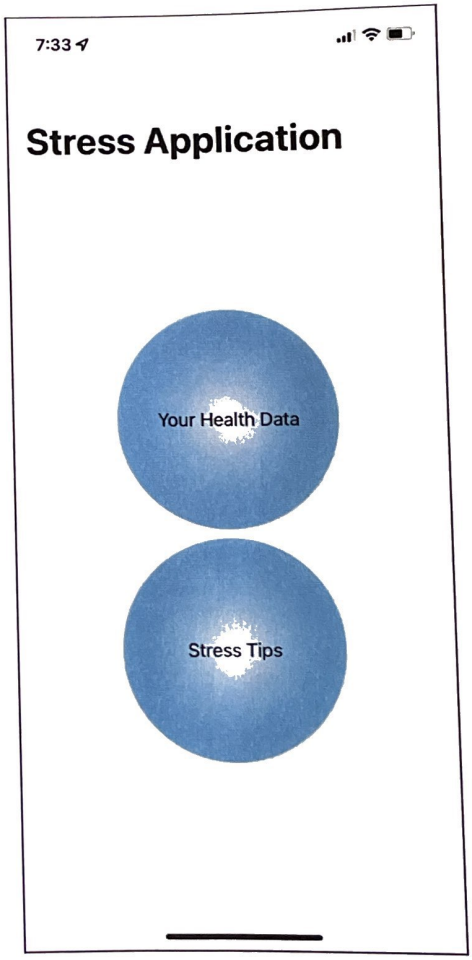
6. Inform user of times of high stress or significant change in health data.



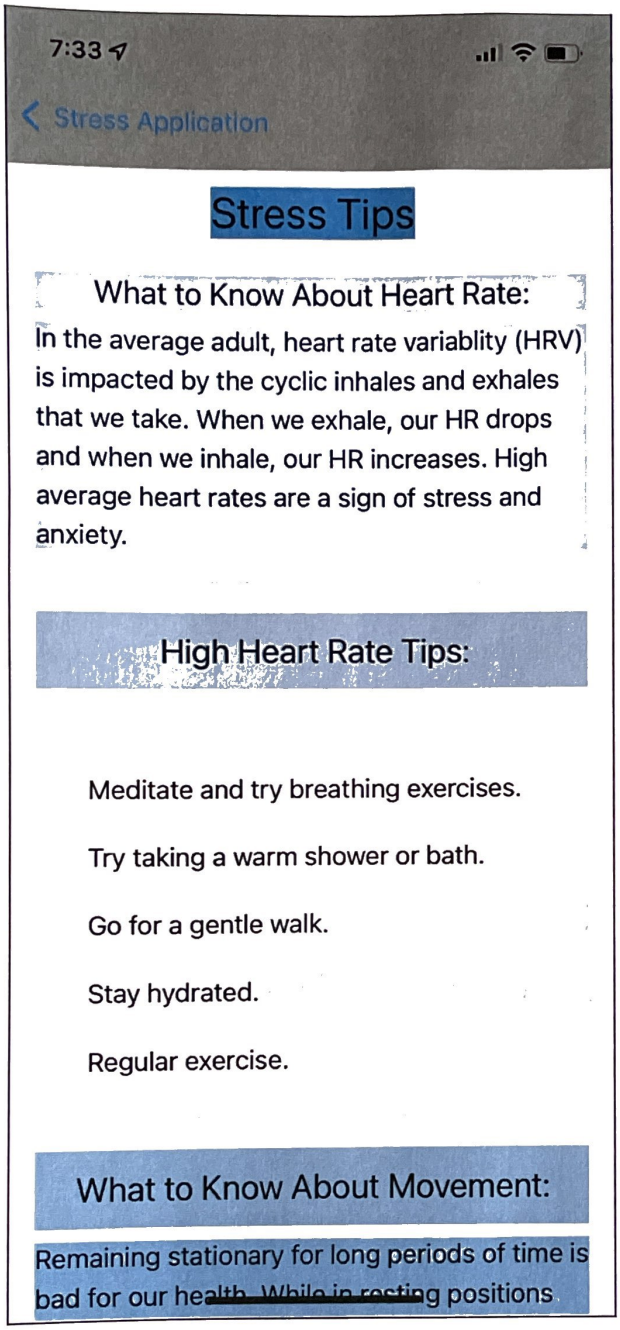
Data flow

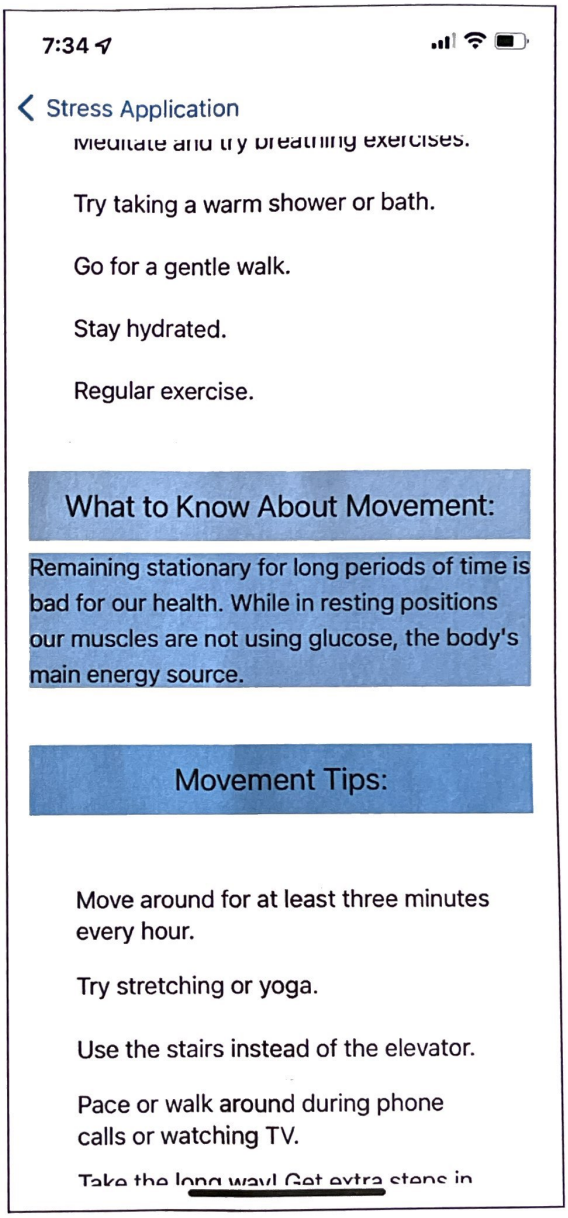


Stats View



Landing View





Stress Tips View

Extensions:

- Usage of different queries
  - HKAnchoredObjectQuery → returns most recent data and monitors for changes
  - HKActivitySummaryQuery → runs in background and monitors for changes

    ***Could possibly eliminate the need of using timers if a query can monitor for changes in data and return it back to the user**

- heartRateVariabilitySDNN quantity sample
  - Measures standard deviation of heart rate intervals

    ***May be a better way to interpret/monitor for stress when considering heart rate as a main indicator of stress**

- Implement two-way communication
  - Apple watch application with iOS companion for data communication

    ***Could possibly eliminate the need for queries and timers if able to successfully able to send HKLiveWorkoutSession statistics directly to iOS companion application to be interpreted**

Apple Developer sample code available for this

- Push notifications on either device

  ***Good way to show that data is being interpreted live**

- Graphs based on data

  ***Apple Developer tutorial "Getting Started with HealthKit" (wwdc20-10664) uses graphs and is a good example of how to turn aggregate data into graphs**

- Breathing animation exercises

  ***Lots of YouTube tutorials available to do this as means of handling stress**

How To

- Download project files off of blog
- Run in XCode
  - → plug iPhone into computer to connect and run on your device
    - → prefferred to do this over simulator as my applications are looking for user's datas... simulators cannot provide this so its harder to test functions and see accurate SwiftUI views

- Will need Apple Developer's license at some point to build on your device & access health data

→ developer.apple.com/account → add certificate / team to project
- → used to see your developer certificate(s) (that you attatch to XCode projects to run them)

→ devices that you run your projects on

→ When you connect your device you will need to select it as your build location... go to XCode menu bar:
Product > destination > your device
- → select your device. The first time you do this you may be asked to register your device
  - → should kind of do it for you but may need to go to developer.apple.com/account under devices and do it manually

- Press the play button ("⌘ Start the active scheme")

- Product > Clean build folder when switching between build locations or weird/long compiling times

Watch Application - File Connections

CapstoneProjectApp
- → StartView // SwiftUI screen we see first
- → SummaryView // appears when workoutManager's showingSummaryView variable == true

StartView → SessionPagingView // once a user selects a session, display SessionPagingView

SessionPagingView
- → ControlsView
- → MetricsView
- → NowPlayingView // UIView provided by WatchKit

Use TabView selection to switch between

MetricsView → ElapsedTimeView

SummaryView → ActivityRingsView

Notes:

StepCountView is not used in the project. It was an attempt to track a user's stepCount before I figured out how to get the stepCount query to work.

WorkoutManager is not a SwiftUIView, it is a class. The "observable object" property of the definition is important as it allows instances of the class to be used inside of views.

↳ In CapstoneProjectApp `@StateObject private var workoutManager = WorkoutManager()` is created so views have access to the WorkoutManager and when views update the current state of the workoutManager is not destroyed.

iOS Application - File Connections

testiOSApp → LandingView //swiftUI screen we see first

LandingView
- → StatsView //see user data
- → StressTipsView // see tips
- → LandingView

] Use Navigation Links to switch between them

Notes

@StateObject private var workoutManager = WorkoutManager
makes functions of WorkoutManager accessible throughout each View ... in particular testiOSApp calls
workoutManager.getLatestHR() and
workoutManager.getRestHR() in the timer function on .onAppear

StatsView accesses the @Published vars in WorkoutManager and displays them.

iOS Application Info.Plist

Watch Application - Data through ~~HeartRateManager~~

In order to read health data from a user's iCloud account / HealthKit you must update your project's info.plist file. This allows you to write a description to the user showing/explaining how and why their data will be used.

What I included for the Watch Application Info.plist:

Privacy - Health Share Usage Description
value: Your Session related data will be used to display your saved workout.

Privacy - Health Update Usage Description
value: Sessions tracked on the Apple Watch will be saved to HealthKit.

What I included for the iOS Application Info.plist

Privacy - Health Share Usage Description
value: Access data to provide summaries and notifications.

Privacy - Health Update Usage Description
value: Sessions tracked on the Apple Watch will be saved to HealthKit.

I would suggest following an Apple Developer tutorial for setting up this portion of your project.

If you cannot find your .Plist find a tutorial that explains how to set one up.
→ this happened to me and I followed a tutorial posted by "Medium" under better programming (linked in my blog)

iOS Application
Watch Application - Data through Workout Manager
The iOS Application's WorkoutManager is based on the Watch Application's WorkoutManager.

When I did this project I started with WorkoutManager by basing it off of the Apple Developer tutorial "Build a workout app for Apple Watch" - linked on my blog.

In both the iOS and Watch application there will be a requestAuthorization function
→ create Set typesToShare // what you want to write to healthstore
typesToRead // what you want to read from HealthStore
insert the HKQuantityTypes you want to see

After you create your sets to read/write, use your healthStore object to request Authorization from a user
healthStore.requestAuthorization(toShare: typesToShare, read: typesToRead)

Make sure you call this requestAuthorization function somewhere in the project.

→ the iOS application uses its WorkoutManager object in the testiOSApp file... basically requesting authorization from a user first thing in .onAppear()
WorkoutManager.requestAuthorization()

→ the watch application uses its WorkoutManager object in the StartView file... basically requesting authorization from a user first thing in .onAppear()
WorkoutManager.requestAuthorization()

⚠ once a user has approved authorization your device remembers
→ if authorization is not given your app will not run

~~iOS Application~~

Watch Application - Data through WorkoutManager

The Watch's WorkoutManager utilizes HKWorkoutSession and HKLiveWorkoutBuilder

→ HKWorkoutSession object called Session tracks a user's workout on Apple Watch

→ HKLiveWorkoutBuilder object called builder constructs a workout incrementally based on live data from an active workout session.

→ func startWorkout(workoutType: HKWorkoutActivityType)
initaizes / sets up Session and builder
→ assigns delegate to self
→ extensions to HKWorkoutSessionDelegate and HKLiveWorkoutBuilderDelegate necessary for implementing / calling the DispatchQueue and updateForStatistics(statistics) respectively

The function updateForStatistics unpacks the workout data generated by the HKLiveWorkoutBuilder. The DispatchQueue runs on a background thread and is continiously monitoring for stat changes and returning them in (statistics).

↳ the switch case looks at each HKQuantityType we are looking to keep track of / update on our MetricsView
→ outside declare @Published var varName: varType = default value to hold unwraped statistics corresponding value / variable
→ want to use @Published var so other files can access the data

Apple documentation / tutorials / searches will be most helpful in seeing how things work / what they're supposed to do.

iOS Application – Data Through WorkoutManager

The iOS WorkoutManager uses queries
→ HKSampleQuery ... returns all matching samples currently saved in the HealthKit store

→ only returns a value once (does not monitor for changes in data)
→ use a timer to call the query continiously

→ query does not force new data to be generated, just asks for the most recent

→ unpack query results inside of query definition (assuming query is successful)
→ store results for the sampleType requested in a @Published var so that variable/data is accessible elsewhere (whatever UI view that displays user metrics)

Notes:
WorkoutManager contains a handful of functions from Workout Manager that do not work as a HKLiveWorkoutBuilder was unable to be implemented on an iOS device.
→ just served as grounds for how to handle/request HealthKit data.

WorkoutManager contains 2 functions at the end of the file that attempt to query data in a different way (unsuccessful).