iHeart Rate

Bryce Schmisseur

CST-451 Capstone Project Final Architecture & Design

Grand Canyon University

Instructor: Professor Mark Reha

Version 1.9

29 November 2020

Abstract

The project consists of two main applications, an Apple Watch app and a website. The combination of these two applications provide a user with a more informational and interactive interface to display the user's heart rate data. From the user's Apple Watch the application will send the heart rate information collected to the web application. The website will be able to display the collected information such as the current heart rate and past heart rate data entries. The website will also offer inputs to select certain period of time to display the information on a table and chart for visual aid. Overall, this project is to be able to get more use of the information collected on a user's smart watch.

History and Signoff Sheet

Change Record

Date	Author	Revision Notes
20-29 November 2020	Bryce Schmisseur	Initial draft v1.0
1 December 2020	Bryce Schmisseur	Revision v1.1
23 January 2021	Bryce Schmisseur	REST API Design
30 January 2021	Bryce Schmisseur	UML Diagrams
23 February 2021	Bryce Schmisseur	Key Technical Decisions
1 February 2021	Bryce Schmisseur	Site Map Diagram
1 March 2021	Bryce Schmisseur	Logical System Design
1 March 2021	Bryce Schmisseur	Physical Solution Design
22 March 2021	Bryce Schmisseur	Removed Kubernetes From technologies and design

Overall Instructor Feedback/Comments

Overall Instructor Feedback/Comments

Integrated Instructor Feedback into Project Documentation

☒ Yes ☐ No

Project Approval

☒ Professor Mark Reha

TABLE OF CONTENTS

DESIGN INTRODUCTION 4
DETAILED HIGH-LEVEL SOLUTION DESIGN 5
DETAILED TECHNICAL DESIGN 8
APPENDIX A – TECHNICAL ISSUE AND RISK LOG 20
APPENDIX B – REFERENCES 21
APPENDIX C – EXTERNAL RESOURCES 22

Design Introduction

Within the document there are many diagrams and explanations that all focus on the architectural design of the iHeart Rate web application and Apple Watch application. Listed in the document are proof of concepts that support the application itself and the technologies being used through the development of the project. All the technologies are listed below including both software and hardware technologies. The designs for both the physical and logical sides of the system are explained, with the physical outlining what is being implemented and the logical focusing on the code logic and where it will be in place. The last part of the documents outlines the technical design of both applications. This section goes into the overview of the application as well as decisions that have been made through the development of the project. This section also outlines the database design and class structure as well as have supporting information for the REST API, non-functional requirements, and maintenance of the application from the start of development to beyond deployment.

The table below outlines all the files, folders, and documents that are not found within the documents. An internal deliverable is a resource that is found within the submission folder under the name given. An external deliverable is a resource that can be found on the internet by the given like.

Deliverable Acceptance Log					
ID	Deliverable Description	Comments	Evaluator (internal or external as applicable)	Status	Date of Decision
1	MongoDB User collection sample	The file is to replicate the DDL script that can be outputted by a SQL database	Internal: Document found within the submission folder called iHeart_Rate_User_Table .json	Completed	Nov 28th
2	MongoDB Heart_Rate collection sample	The file is to replicate the DDL script that can be outputted by a SQL database	Internal: Document found within the submission folder called iHeart_Rate_Heart_Rate_Table .json	Completed	Nov 28th
3	MongoDB Dump	A folder containing all the files of the Mongo Database. This is used as a backup if something were to get corrupted.	Internal : Folder found within the submission folder called MongoDump	Completed	Nov 28th
4					
5					

Detailed High-Level Solution Design

For the iHeart Rate application all of the explanations and diagram down below fully support the design of the project. The proof of concepts supports the application itself and the technologies being used. Logical and physical system design outline the logical flow of code and the what the code is. After the design the document outlines all the database information, ER diagram, data dictionary, a mongo equivalent to a DDL which all follow the support of the mongo DB. The next diagrams show the sitemap as well as the UI design theses but show how the interaction will flow and as well as how it will look. The sequence diagrams display how to code goes through all different levels of the software and in which type. The next diagram outline how the application will be hosted to the cloud including all the different technologies being used. After that there are all class diagrams which help developed the application, explanation of nonfunctional requirements which are able to test the website and an explanation of monitor and logging.

Proof of Concepts

The table below describes various projects/applications that have proven not only the technologies being used can work together but also the application design works with other technologies:

Proof of Concepts		
Description	**Rationale**	**Results**
1 – Set up an enterprise java application which dealt with the front and back end of the project. Also set up a REST API using JAX-RS to be able to retrieve heart rate information from the apple watch.	This has proven that the concept of the application that is being created works with different technologies	Application was successful, the front end displayed the heart rate information from the apple watch to the user. The user was also able to select a time to view different time periods of heart rate information
2 – Setup up full stack application sending data from MongoDB through Express JS and on to the front end of React through a REST API.	This was created to show that the three technologies that are being used within the project can indeed get information two and from a level.	This application was a success. There were three major parts to get this completed, one being the database set up with a simple string of 'Hello World'. The second requirement was to have Express be able to retrieve that information to the database. At lastly the front end had to request that information by a REST API set up within Express.
3 – Setup up a full stack application that took in a React Component which went through the express REST API to be put into the database.	This was created in order to show that information can not only be retrieved from Mongo but also sent to Mongo from React.	The application was successful by being able to take in the first name of the user and even with the asynchronous style of express it made to the MongoDB

Hardware and Software Technologies

The table blow describes all the technologies, both hardware and software being used through the development of this project. The hardware technologies, left side, include the two devices that the application is being developed and tested on. On the right side with the software technologies, it outlines the languages, frameworks, libraires, IDEs, and other software elements that are being used to create and test the application as well.

Hardware Technologies	Software Technologies

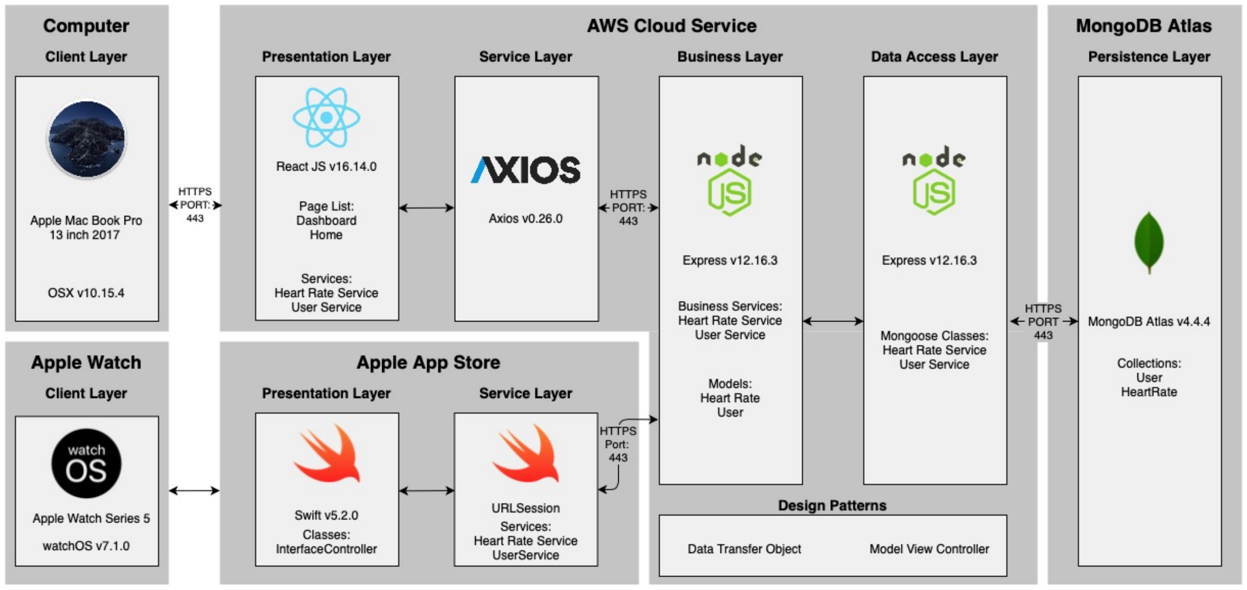
1 – Laptop Apple MacBook Pro (OS Catalina v10.15.4)	1 – React JS v16.14.0
2 – Smart Watch Apple Watch (OS watchOS v7.1)	2 – Express Node JS v12.16.3
	3 – MongoDB v4.4.0
	4 – Mongo Compass v1.23.0
	5 – Visual Studio Code v1.49.1
	6 – Postman v 7.36
	7 – xCode v12.1
	8 – Axios v.021.0
	9 - Material UI v4.11.0
	10 – Mongo Atlas v4.1.0
	11 – AWS
	12 – Swift v5.2
	13- Docker v19.03.13

Logical Solution Design:

The logical system design of the iHeart Rate project describes how the code will logically flow through the application. Starting from the client layer the user can interact with the project on two platforms: web browser and Apple Watch. An Apple MacBook Pro and Apple Watch Series 5 are listed as they are the two devices that the application is being developed and tested on.

The front end of the web application will be deployed using Vercel and the backend is deployed with Heroku. The user will interact with the front end of that application that is written with the React framework. React makes a use of components and services in order to display and retrieve information to and from the user. The components for this application include home, main and sign in to create all the views of the application. These components make use of the user and heart rate service in order to create REST API calls to the back end using the library of Axios. Axios is a JavaScript library that has the ability to make HTTPS request. These HTTPS requests are handled by the Express application for the back end of the application. The routes within the express application can pick out the data form the HTTPS request and create models from the data. Within Express there are two models that are implemented one for the user object and another for the heart rate object. These models make it easy to send the object data between both business and data services as the Express application uses the DAO design pattern. The business services either, user business service or heart rate business service are created in order to transfer data from the routes to the data services. There are only two data services as there are only two object models that the application deals with. The data services connect the backend of the application to the MongoDB Database. The database that is running on the cloud using Mongodb Atlas which contains two collections: user and heart rate in order to hold documents of the object models.

The application has a similar path of data and code logic however the beginnings of the process are different. As the Apple Watch app cannot be deployed on a cloud platform it must be hosted from the Apple App Store. The front end and service layer code for the apple watch app will be done in Swift. Swift is a responsible for setting up the front end of the application and making the REST API calls to the back end. With the addition of the URL Session library provided by Swift the creation of HTTP requests is just as easy as Axios. The HTTP request is then sent to the Express application in AWS just as Axios did with the request from React. After the request reaches the Express application the data follows the same flow as the request from React did to reach MongoDB.

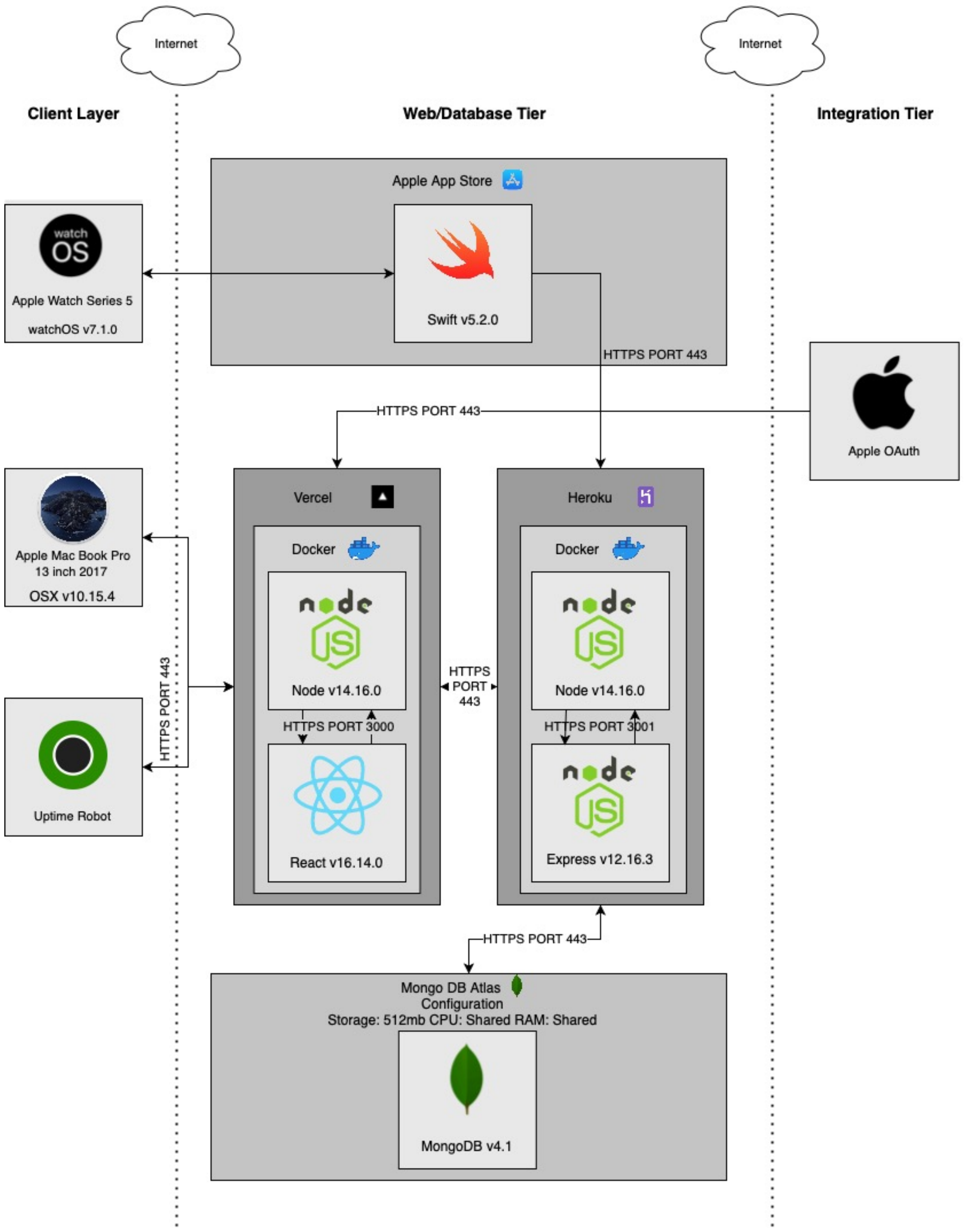


Physical Solution Design

The physical solution design below of the iHeart Rate project displays the technologies, services and platforms that are used in order to deploy the application. Starting with the web application there are two platforms in which interact with the entirety of the AWS container, the two being a web browser and Up Time Robot. Up Time Robot is a monitoring tool for cloud applications test the reliability of the application, this will be described more within the Operational Support Design section. The web application interacts with the front end of the application through HTTPS port 443 to make HTTPS requests. The first host the cloud host whether it be Vercel for the frontend or Heroku for the backend. The request will then meet docker container which will route it to the application. The connection between the frontend and the backend will follow the same path going from the docker container to the cloud platform to the internet. The same goes for the connection between the backend and the database, Mongo is using AWS to host a container in order to store the information. This connection is made through a mongoose object that has the srv to the Mongo Atlas database that will send request using HTTPS.

The path of a request from the Apple Watch Application follows a similar path but just like the Logical System Design it does start out different. The Apple Watch application will be deployed on the App Store where any user can download it. Once the application starts making request it uses the same HTTPS port 443 in order to access the Express application through a secure connection. Once in the Express application the data will pass through the different layers all the way to be able to access the database just as the web request did. For all deployments certain processing powers and storage are allocated in order to get the information in-between the application as quickly and securely as possible

The database running on Mongo starts out with shared ram and CPU cores as well as 512mb to store information. Though this may seem small the database can scale up and down in performance based on the metrics is needed to perform action seamlessly. As for the frontend deployed in Vercel the application starts with 1024MB of RAM, shared CPU, and 512mb of storage with scalable storage and memory. Lastly the Heroku deployment which houses the backend of the application starts out with 512mb of RAM, shared CPU and saleable storage.



Detailed Technical Design

General Technical Approach:

For iHeart Rate there are two major components of the project: the web application and the Apple Watch application. The Apple Watch application is used to record heart rate information from the Apple Watch itself. This information is then sent to the web application in order to display the data to the user. The web application displays the information in a tabular and graphical format based on the given time from the user. The user will be able to enter a dates and times in which they would like to see heart rate information in between. The table and graph will then display the heart rate information both comparing the BPM of the heart rate and time. The user will also be able to see the average heartrate and the last recorded heart rate that was stored. To hold all the data of the specific user they will be asked to sign in with their Apple ID account information in order to make the sign in process easy and efficient for the user.

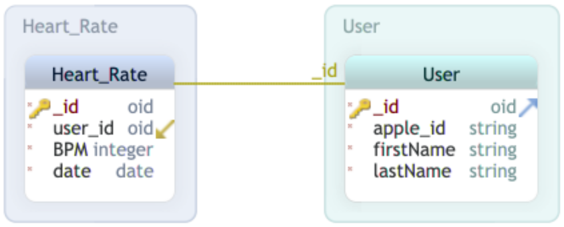
Key Technical Design Decisions:

Through the development of the project there have been various different design decision that have been changed though the technologies of the assignment have remained the same. For the web application the front end will be implemented using React JS and with using the Material UI library the application will be responsive. For the front end of the Apple Watch application, it will be coded using Swift. Both of the front end application: react and swift connect the same back end application which will be implemented in Express JS. Express will set up as a REST API with routes that will be connected to Mongodb Atlas through a library called Mongoose that can connect to the database easily. To be able to host this application on the could for the web application it will consist of multiple different cloud platforms and technologies. Docker will be used to set up two separate containers to house the React and Express applications. The frontend of the application will be hosted with Vercel due to it CI/CD capabilities as well as its compatibility with ReactJS. The backend of react will be hosted with docker on Heroku as it is a very easily platform to setup code pipelines that are easily implemented with docker. The Database is being hosted through MongoDB Atlas which sets up a container within AWS in order to store all the information.

The most major design decision that was changed within the application is the login process for the user. At first the application was questionable on how to connect the Apple Watch data to a user but by Apples oAuth API it fixed this problem and more. By having the user sign in with their own Apple ID allows the REST API call from both the React application and Apple Watch contain the same user identifiers. This also allows for the web application to not need a registration process as all user creation is done through Apple.

Database ER Diagram:

The diagram below displays the design documentation for the database. For this project the database will be hosted by MongoDB Atlas on the cloud. It will contain two collections: ‘User and ‘Heart_Rate’:



Database Data Dictionary

The tables below signify data dictionary for both the user and heart rate collection within MongoDB. These describe all the fields within each document in a collection.

User Collection

Field Name	Type	Description
_id	objectId	The unquie identifier that is created by Mongodb when a user is created
apple_id	objectId	The ID key given from Apple's oAuth REST API call when a user logs in
firstName	String	The first name of the user
lastName	String	The last name of the user

HeartRate Collection		
Field Name	**Type**	**Description**
_id	objectId	The unquie identifier that is created by Mongodb when a heart rate is inputed
user_id	objectId	The users object ID in which the heart rate belongs to
bpm	Int	The heart rate of the user signified by beats per minute
date	Date	The date in which the heart rate was recorded

Database DDL Scripts:

Due to the use of the non-relational Mongo database that has been implemented into the project there is no specific output of the full database. However, MongoDB allows to developer to be able to export the data within the database. In the submission folder the file labeled *iHeart_Rate_User_Table.json* and *iHeart_Reat_Heart_Rate_Table.json* are the two exported data files coming from the iHeart Rate database within mongo. Also included within the submission folder is a folder backup of the mongo database called *MongoDump*. This can be used to recreate the database if another user would like to setup the application or the if database is lost.

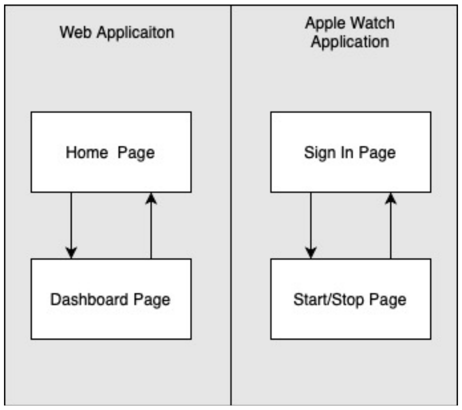
More information of these three files can be found within the Deliverable Acceptance Log above:

ID 1: *iHeart_Rate_User_Table.json*
ID 2: *iHeart_Reat_Heart_Rate_Table.json*
ID 3: *MongoDump*

Flow Chart/ Process Flows:
For this application there is no algorithms or processes to diagram for the completion of the project.

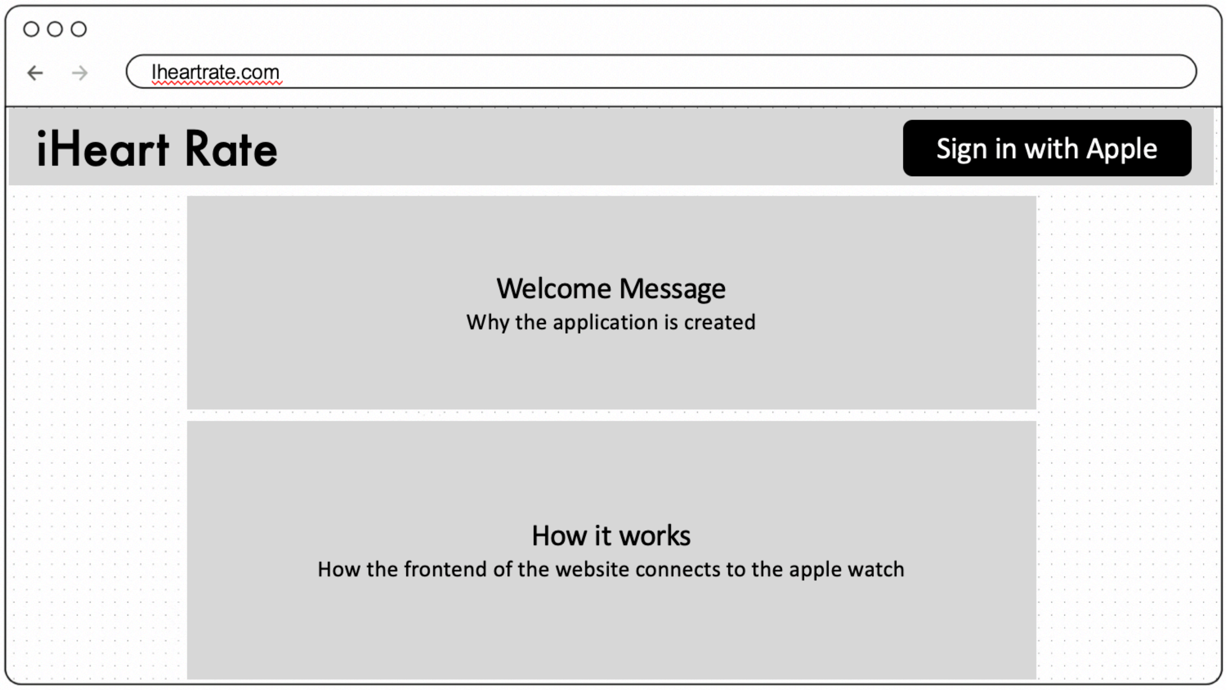
Sitemap Diagram:

The figure below shows the site map diagram for both the web and apple watch application. On the left, the web application, the website will begin the user at the home page which will have the Sign In With Apple button in order to take them to the graphical viewing page. On the right hand side the Apple Watch Application is displayed, this is a very simple application with two pages: the sign in page, and the start/stop button page.

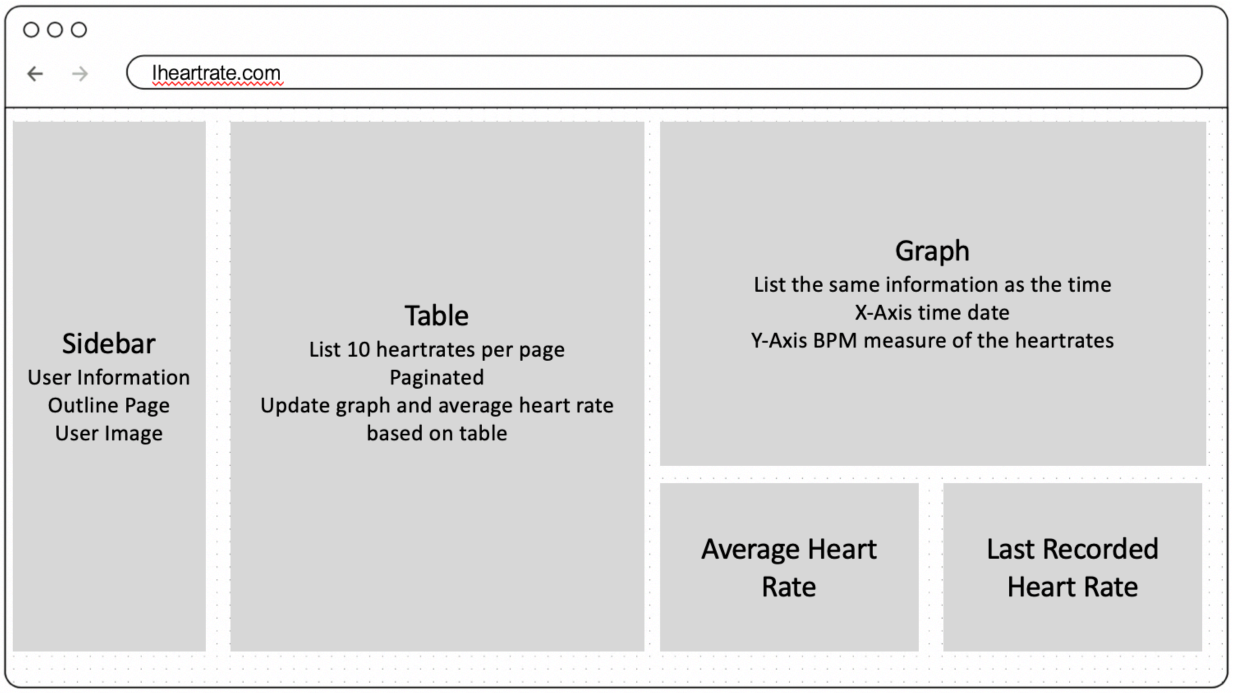


User Interface Diagrams:
Web Application

The first page that will be displayed to the user will be the home page where the iHeartRate Applications are described. There will be a little section on why the app was created as well as how the app works with the apple watch. This page will also contain the 'Sign in with Apple' button that will lead them to the user's dashboard.

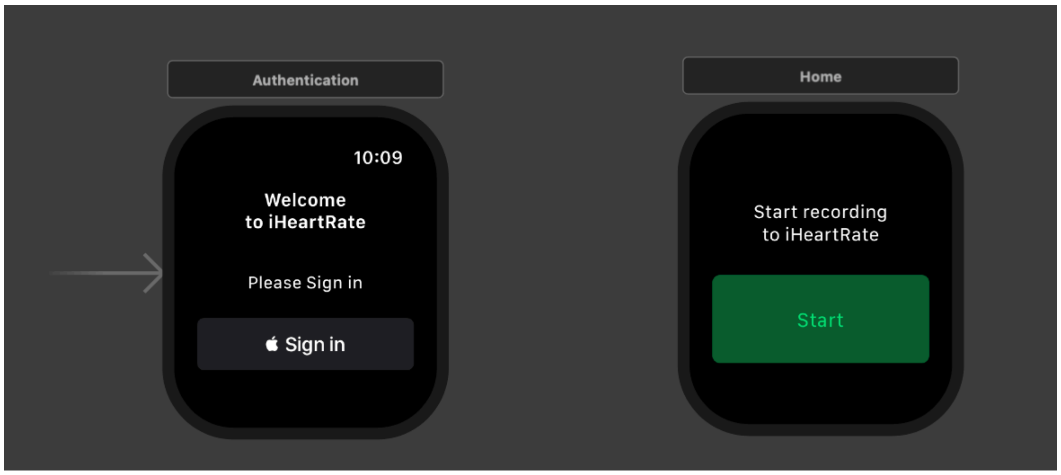

Iheartrate.com
iHeart Rate
Sign in with Apple
Welcome Message
Why the application is created
How it works
How the frontend of the website connects to the apple watch

The dashboard page that the user can access within the web application is the main page displaying all the users heart rate information. This is where the user will spend most their time as it is the page where the information is actually provided to the user. The page is set up with a side menu that displays a welcome message, their own image, name, and the contents of the page. The right panel of the page will contain a graph, a paginated table, the average heart rate, and the last recorded heartrate. Lastly in the upper left and corner there will be a button in order to select dates. Once pressed it will give a popup to the user that will allow them to enter certain dates and time in order to grab heart rate information with the selected time span.


Iheartrate.com
Sidebar
User Information
Outline Page
User Image
Table
List 10 heartrates per page
Paginated
Update graph and average heart rate based on table
Graph
List the same information as the time
X-Axis time date
Y-Axis BPM measure of the heartrates
Average Heart Rate
Last Recorded Heart Rate

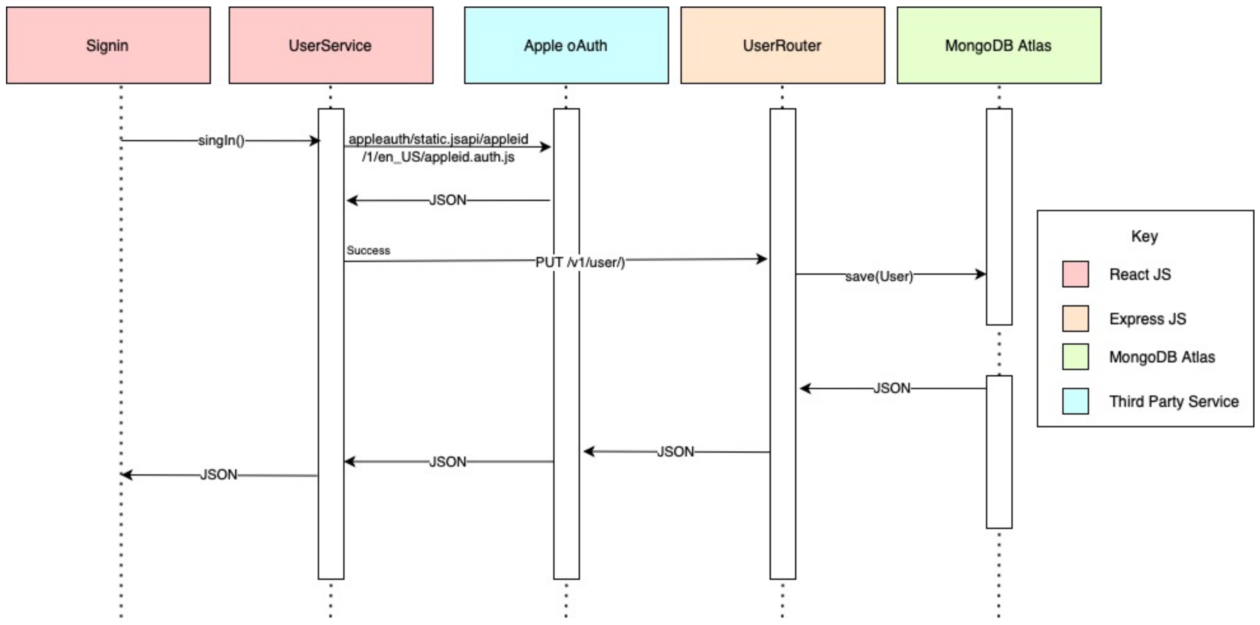
Apple Watch Application

Just as the website did the Apple Watch application will only contain to pages. The first page being a welcome page will display a welcome message and the Sign in Apple Button. The second page will contain a singular button that will change to start if the application is off but stop if the application is recording.



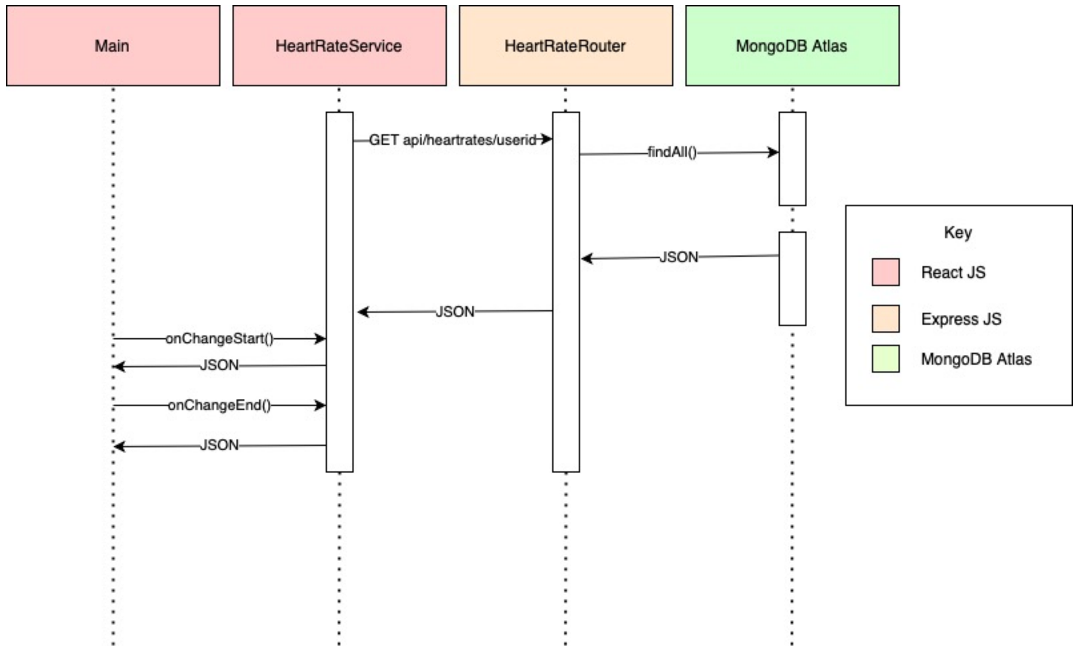
UML Sequence Diagram
Login Sequence

The diagram below displays the flow of data that happens when a user request to login with ‘Sign in with Apple’ from the React application:



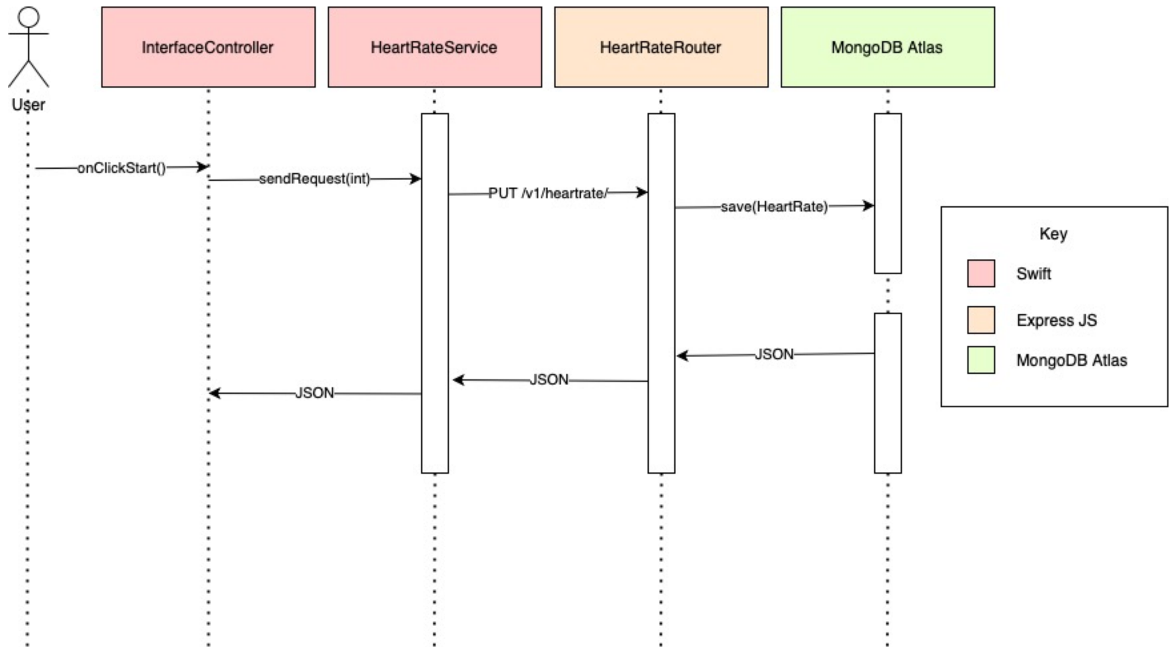
Date Sequence

The diagram shows the flow of data when the User is with Main component of the react application. The component will first grab all the heart rate information from by the user's ID. Once the user changes the date it will trigger the onChange to get the updated data list.



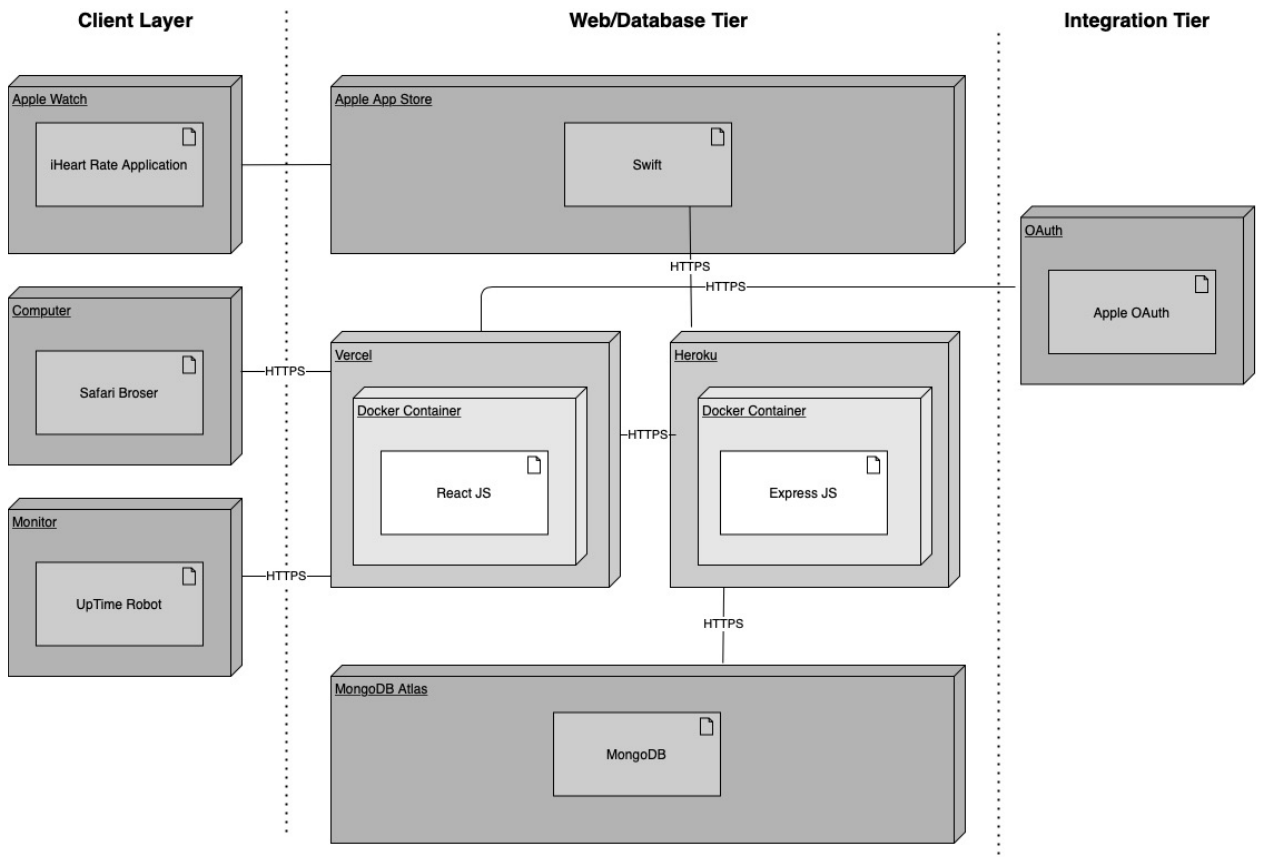
Heart Rate Sequence

The diagram below displays the sequence the data from the apple watch takes when a new heart rate needs to be saved within the database.



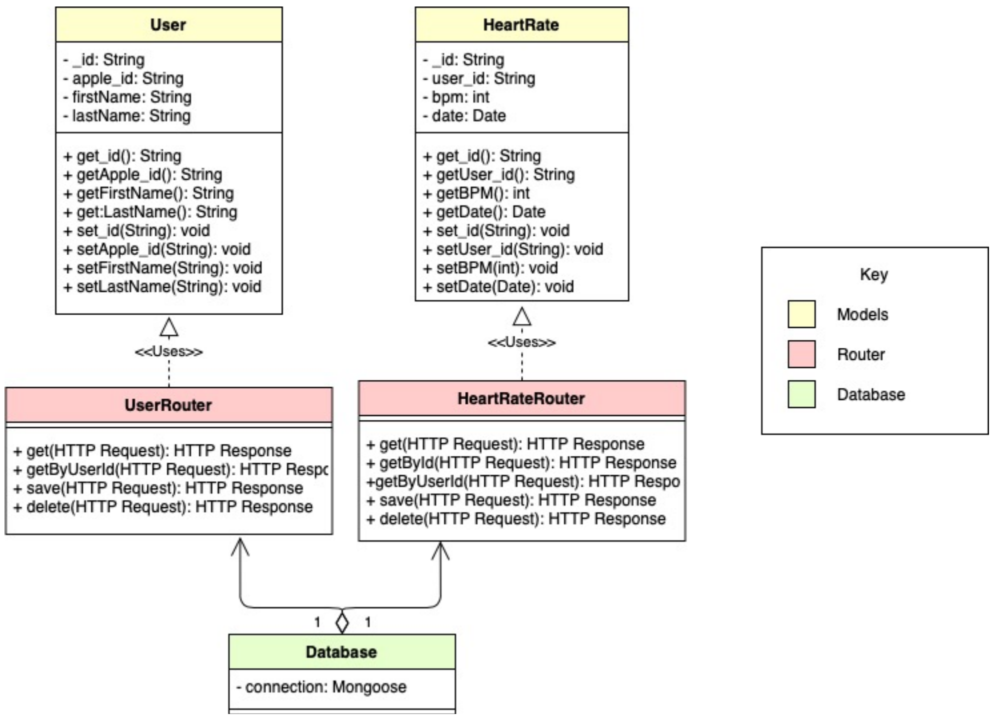
UML Deployment Diagram

In the figure below outlines the deployment structure of both the web and Apple Watch application. The deployment will contain the Apple Watch Application being hosted on the Apple App Store, React will be deployed using Vercel in a docker container, Express will also be inside a docker container though hosted on Heroku, and the database will be deployed with MongoDB Atlas.



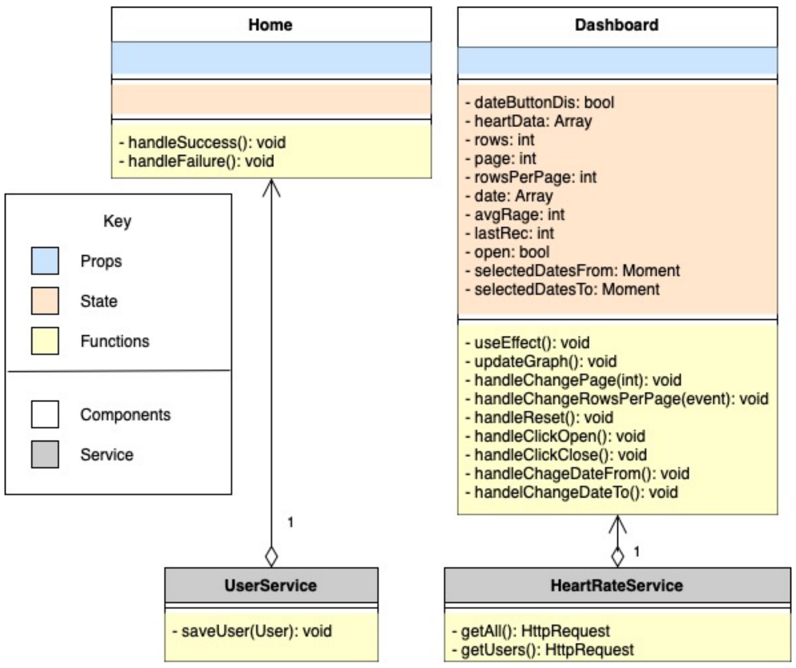
UML Diagrams:
Express Class Diagram

The Diagram below display the class design of the back end of the application which is implemented by Express JS. Starting from the top of the diagram there are two models: user and heart rate. Both the user and heart rate properties are described above within the *Database ER Diagram*. The methods of both are listed below their corresponding properties and consist of the getters and setters of the properties. Moving down from both the models each has its own router functions which sets up the REST API service. These functions do not contain any properties, but each individual method is a separate REST API call. These calls used a library within Node called mongoose that makes it easy to connect with MongoDB. The object model is setup using mongoose in order to input the data in to the correct connection. Setting up the mongoose object with the connection string to the database, within the router will then persist the data into the database.



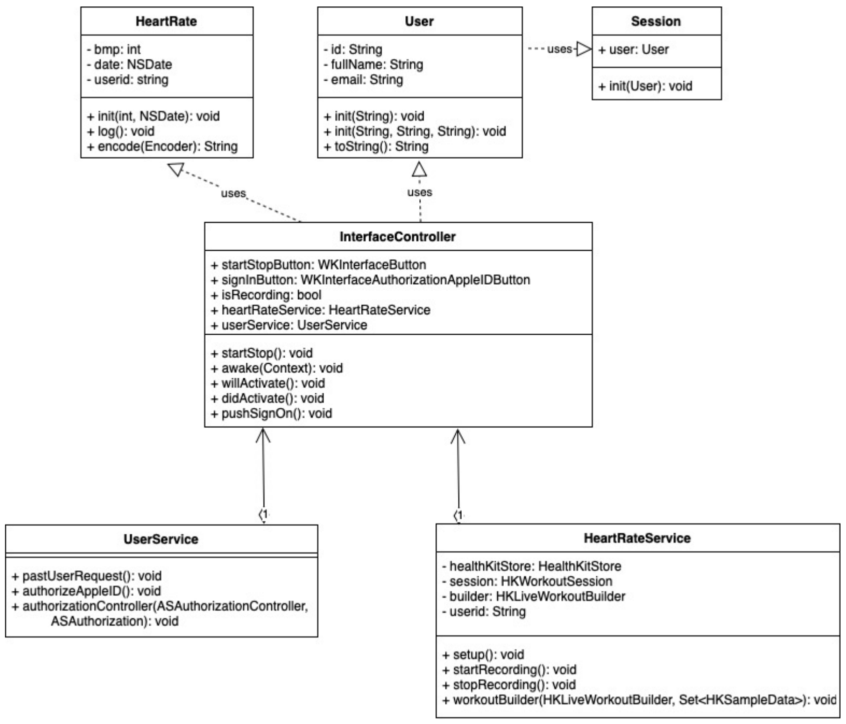
React Class Diagram

The diagram below outlines the UML Class Diagram structure of the front end of the application implemented with react. React works with components and within each of them there are props, state and methods. Props in react are variables that are sent between different components such as a user that is signed into the application. State are variables that are only specific to each component and can be used to display certain information on the page based on user input for an example. Lastly methods are just the actions that the component such as an onChange method in order to handle an input from the user. The home page does not have very many moving components on the screen as can see by the class diagram. The only logic that is occurring for this page is the 'Sign in with Apple feature' most the logic is completed by Apple oAuth functionality itself. On the other hand, the dashboard page has a lot of state variables and methods. This page contains properties to set the table, graph, average heart rate, last recorded heart rate and all the properties that help with that information. The methods within this page are to handle and set all the components for the user output and render the page when it loads in. There are no props used through this project because the user information for the user will be stored in an HTTP cookie so that the information can be accessed anywhere.



Swift Class Diagram

The diagram below outlines the class structure within swift for the Apple Watch application. The interface controller set up the view and the workout that is needed to record the heart rate of the user. The BMP of the user is then sent to the Heart Rate Service to be able to make an HTTPS request using URL Session library.


HeartRate
- bmp: int
- date: NSDate
- userid: string
+ init(int, NSDate): void
+ log(): void
+ encode(Encoder): String
User
- id: String
- fullName: String
- email: String
+ init(String): void
+ init(String, String, String): void
+ toString(): String
uses
Session
+ user: User
+ init(User): void
uses
uses
InterfaceController
+ startStopButton: WKInterfaceButton
+ signInButton: WKInterfaceAuthorizationAppleIDButton
+ isRecording: bool
+ heartRateService: HeartRateService
+ userService: UserService
+ startStop(): void
+ awake(Context): void
+ willActivate(): void
+ didActivate(): void
+ pushSignOn(): void
1
1
UserService
+ pastUserRequest(): void
+ authorizeAppleID(): void
+ authorizationController(ASAuthorizationController, ASAuthorization): void
HeartRateService
- healthKitStore: HealthKitStore
- session: HKWorkoutSession
- builder: HKLiveWorkoutBuilder
- userid: String
+ setup(): void
+ startRecording(): void
+ stopRecording(): void
+ workoutBuilder(HKLiveWorkoutBuilder, Set<HKSampleData>): void

Service API Design:

Express REST API

The link below provides the documentation of the REST API that has been created with in Express JS. This is what is used within the backend to send information to the front end from the database and retrieve information to send to the back end:

https://documenter.getpostman.com/view/9123994/TVmJge2W#8766fd99-93af-4ebf-94f8-2f2e0437af93

Apple oAuth Rest API

As the application is using Apple's oAuth REST API all the documentation and implementation can be found at in Appendix B – References under 'Sign in with Apple'

NFR's (Security Design, etc.):
Secure REST API calls

For the application there are many different REST API calls made to sign in the user, save information, and retrieve information from the database. To secure all the API calls some are already secured by the third party service such as apple, though saving and retrieving information from the database needs its own security. First looking at signing in with Apple there are already secure measures in place in order to secure the information from both the iHeart Rate application but also their Apple ID account. Apple uses oAuth2 in order to secure all request and responses to sign in the user. To secure the request and responses to the database the application will be using the GitHub Developers service using oAuth as a pass through for every REST API. With both of these implementations the delivery and acceptance of all information across all layers will be secured.

Responsiveness:

With the incorporation of Material UI in the project the responsive of the web application will mostly be taken care of using that library. Material UI is a library within Node JS in which gives the developer HTML like tags that are used to create the aspects of the components. By using the tags provided by the library Material UI then give each component open and expand or collapse based on the current screen size. Even though Material UI already has all this information out of the box the developer can customize each individual element as the please within a configuration file. Using this library throughout the project will ensure that most screen sizes will be supposed and make the application easy to use and navigate.

Operational Support Design:
Logging:

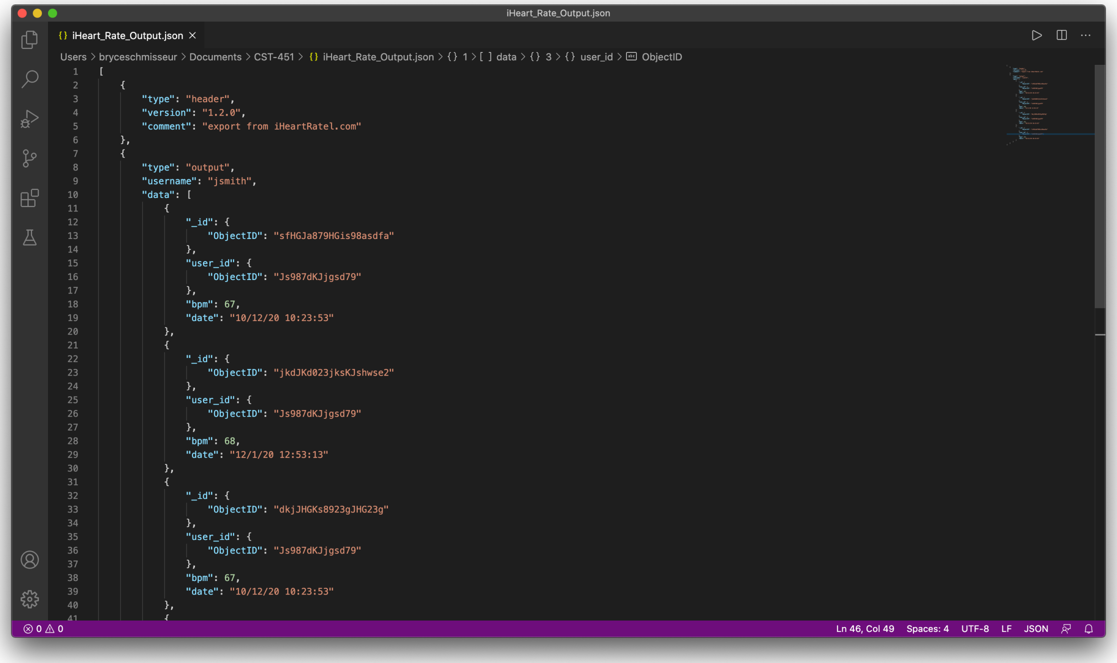
Throughout the various different languages of this project there are also various different forms of logging. Logging in this application and all others help when developing the application to understand where the application is within the code. Logging within the apple watch project is done within Xcode using swift and is made really easy with the library of NSLog. NSLog is a logging library included within swift in order to create formatted log statements to print out to the console. For React JS and Express JS logging for an application is not as easy. For these two frameworks currently, logging is done through browser so any one can see the log statements within the web browser console. Though for the development of the application these logging statements do provide a way to see was is happening with in the code. However, these statements will need to be taken out of the time of deployment as to not have to program relevel sensitive information of the project or the users.

Monitoring:

Monitoring of an application is necessary when in order to keep the services available to the user. Within cloud environments there are many popular and reliable third party services that can track and notify the user of the state of the application and any errors that it finds. One of these third party services is called Uptime Robot. This web application allows the developer to input the host URL and make a request the users desired amount per day. Uptime will then give the web site a metric for the reliability of the website based on the number of nines principle. It can also notify the developer through a desired contact method if the request failed in any way. This monitoring service will be implemented in order to track the reliability of the web application that is hosted on AWS.

Other Documentation:
User Export:

The document below will be a file that the user will be able to export from the web application of iHeart Rate. This file will contain all the heart rate information that has been collected from their apple watch as seen in seen in the example. This file will be formatted in JSON in order to make it easier to read for the user. This will also give the user the ability to upload this information to any other application that will accept it.


iHeart_Rate_Output.json
iHeart_Rate_Output.json
Users > bryceschmisseur > Documents > CST-451 > iHeart_Rate_Output.json > {} 1 > [ ] data > {} 3 > {} user_id > ObjectID
Ln 46, Col 49 Spaces: 4 UTF-8 LF JSON

```json
[
    {
        "type": "header",
        "version": "1.2.0",
        "comment": "export from iHeartRatel.com"
    },
    {
        "type": "output",
        "username": "jsmith",
        "data": [
            {
                "_id": {
                    "ObjectID": "sfHGJa879HGis98asdfa"
                },
                "user_id": {
                    "ObjectID": "Js987dKJjgsd79"
                },
                "bpm": 67,
                "date": "10/12/20 10:23:53"
            },
            {
                "_id": {
                    "ObjectID": "jkdJKd023jksKJshwse2"
                },
                "user_id": {
                    "ObjectID": "Js987dKJjgsd79"
                },
                "bpm": 68,
                "date": "12/1/20 12:53:13"
            },
            {
                "_id": {
                    "ObjectID": "dkjJHGKs8923gJHG23g"
                },
                "user_id": {
                    "ObjectID": "Js987dKJjgsd79"
                },
                "bpm": 67,
                "date": "10/12/20 10:23:53"
            },
            {
```

Appendix A – Technical Issue and Risk Log

The table below shows all the issues and risk that were encountered along the way of devolving this application.

Issues and Risk Log							
Issue or Risk	**Description**	**Project Impact**	**Action Plan/Resolution**	**Importance**	**Date Entered**	**Date to Review**	**Date Resolved**
I/R	What is the issue or risk?	How will this impact scope, schedule, and cost?	How do you intend to deal with this issue?				
R	Little to no documentation on how to integrate Apple OAuth for apple watch	This will affect my ability to implement Sign in with apple for my apple watch	Study more and find examples of Sign In With Apple working within an Apple Watch Application	*High*	*Jan 28th*	*Feb 4th*	*Feb 7th*
R	Little to no documentation on how to integrate Apple OAuth for ReactJS	This will affect my ability to implement Sign in with apple for my apple watch	Study more and find examples of Sign In With Apple working within an React Application	*High*	*Feb 14th*	*Feb 28th*	*Feb 26th*

Appendix B – References

“MongoDB - Data Modelling.” *Tutorialspoint*, www.tutorialspoint.com/mongodb/mongodb_data_modeling.htm.

“Sign in with Apple REST API.” *Apple Developer Documentation*, developer.apple.com/documentation/sign_in_with_apple/sign_in_with_apple_rest_api.

Perrin, Chad. “The Three Elements of Access Control.” *TechRepublic*, TechRepublic, 15 Aug. 2007, www.techrepublic.com/blog/it-security/the-three-elements-of-access-control/.

Appendix C – External Resources

GIT URL:	*https://github.com/bschmisseu/iHeartRate-Front* *https://github.com/bschmisseu/iHeartRate-Backend* *https://github.com/bschmisseu/IHeartRate-AppleWatch*
Hosting URL:	https://iheartrate-bschmisseu-gcuedu.vercel.app
App Store:	Currently not deployed on the Apple App Store