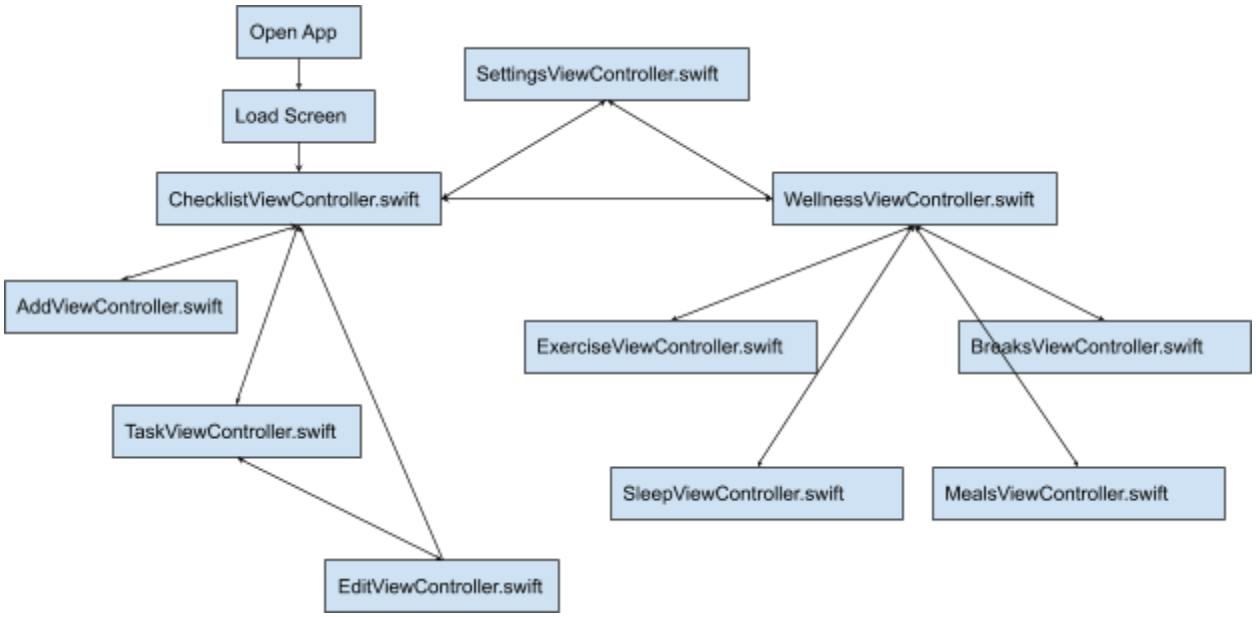
Iremia Design Document

The updated design document must also provide an overview of the organization of your code; depending on your implementation, this could be done using a class diagram (if it corresponds directly to your code), or a diagram showing your file organization, annotated with explanations; the format is up to you, but you must provide an explanation, using pictures and words, of your code structure that would allow someone looking at your code for the first time to understand its architecture

File Organization/Structure



Our Project Folder is Called Iremia, and contains two folders(Base.lproj and Assets.xcassets) and several swift files described below.

- Base.lproj:
    - Base.lproj contains the contents of our Storyboard, which is the Apple IDE built in UI tool that allows us to visually create our frontend pages for our Ios application. Not only can we see our designs come to life on the storyboard, but we can also connect variables created in the swift code to connect to our storyboard components, such as different text labels, tables, and buttons, and change them programmatically using the xcode frontend library UIKit.

- Assets.xcassets:
    - This folder contains all of our image and vector assets for our application, such our app logo, launchpage screen, and various icons.

By default, XCode projects includes the following swift files which we did not edit:

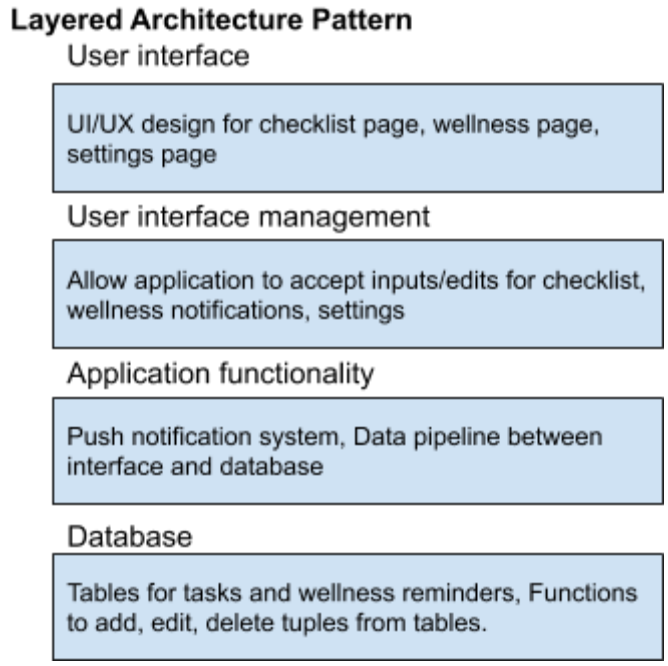
- SceneDelegate.swift
- AppDelegate.swift
- Info.plist

Our backend is mostly ViewControllers, which are a display screen Swift pushes in a stack style, allowing people to go to a new ViewController(push to stack) or to go back a screen(pop from stack).

CheckListViewController.swift is the file that is always at the bottom of our ViewControllers stack. From there, you can navigate to the WellnessViewController.swift, SettingViewController.swift. These three mentioned ViewControllers are the main pages.

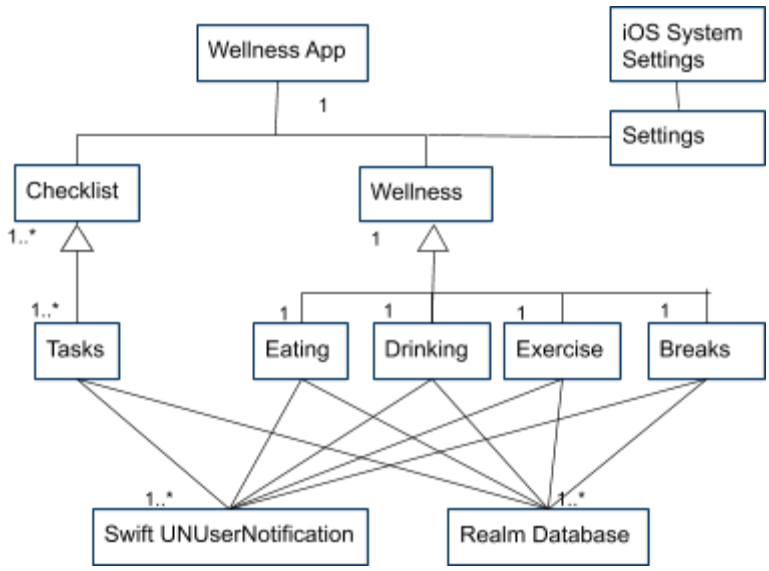
- ChecklistViewController.swift pertains to the following files/sub ViewControllers:
    - AddViewController.swift
    - TaskViewController.swift
    - EditViewController.swift
- WellnessViewController.swift pertains to the following files/sub ViewControllers:
    - ExerciseViewController.swift
    - BreaksViewController.swift
    - MealsViewController.swift
    - SleepViewController.swift
- SettingsViewController.swift is the sole file regarding Settings.

Architecture Pattern



Our User Interface was developed independently while User interface management and application functionality was developed by our backend team. After the base code and navigation was finished by the backend team, we were able to quickly integrate the User Interface to create a cohesive product. During this integration, we were also able to quickly add code to incorporate our Realm database, allowing the data to be saved in our product.

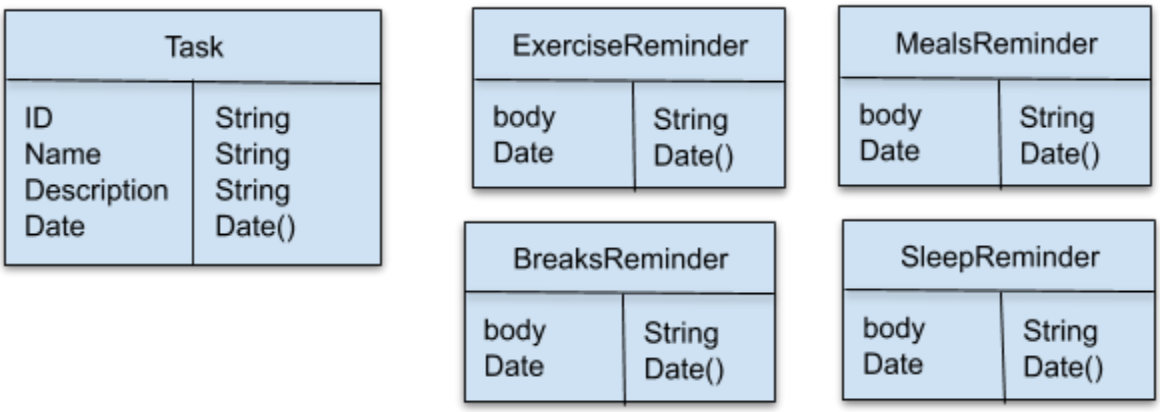
Class Diagram



The Wellness App is the product. It has three main sections, the Checklist, Wellness, and Settings Tab.

- The Settings tab is the simpler of the three, providing navigation to the iPhone settings to enable notifications from the Wellness App.
- The Checklist is a dynamic page of Tasks where each task has a notification object and Realm Database information
- The Wellness is a static page of four repeat reminders(Eating, Drinking, Exercise, Breaks) where each of these repeat reminders has a notification object and Realm Database information

Database

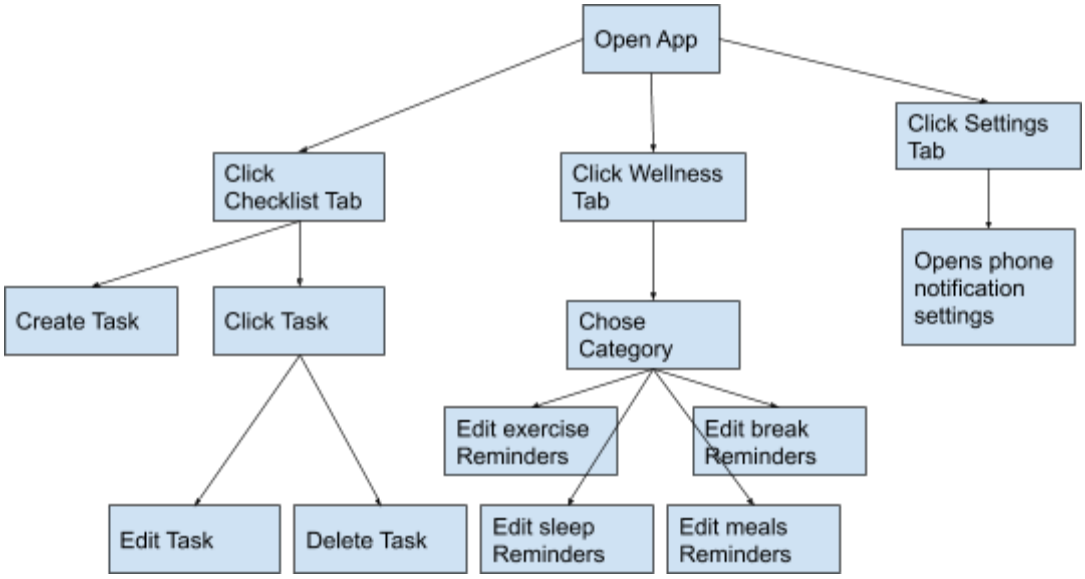


Each box in the above diagrams represents one of the five tables in our database. The top of each box is the name of the table. The bottom left column of each table is the name of the variable. The bottom right column is the data type of the corresponding variable.

<u>How to interact with database:</u>

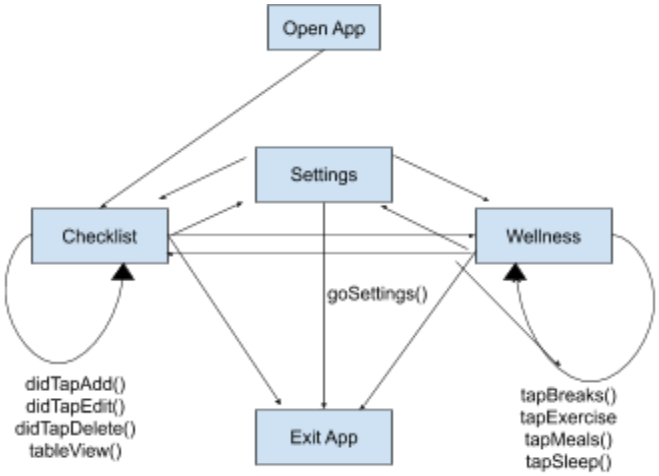
- Initialize Realm database
    - private let realm = try! Realm()
- Reading:
    - We can populate a local array of a certain class object for each table
        - *arrayName* =realm.objects(*ObjectName*.self).map({ $0 })
    - Read each class object variable from the local array
        - *outputVariable* = *arrayName*[*index*].*variableName*
- Writing:
    - Open database
        - realm.beginWrite()
    - Create object and input variables
        - let *newObjectName* = *ClassObjectName*()
        - *newObjectName*.*variableName* = *inputVariable*
    - Add object to database
        - realm.add(*newObjectName*)
    - Write changes and close database
        - try! realm.commitWrite()
- Querying task with taskid
    - let myItem = realm.objects(Task.self).filter("id == %@", taskID)

Activity Diagram:



At any point in the diagram, someone could click off into the other tab or cancel/go back but that would clutter up the diagram. We focused on the sequence of events a user would probably follow to do an action in the app.

State Diagram:



MVP Requirements (Successfully met)

1. Planning
    1.1 The user should be able to make a checklist of tasks to complete
        1.1.1 Ability to add, edit, and delete tasks
            1.1.1.1 By default, the tasks should be sorted in chronological order, based on when the task was added, with tasks added first appearing at the top
        1.2.2 Ability to store and display event information
            1.2.2.1 Users should be able to add information including a description of the event/ task and a due date.
2. Wellness
    2.2 Ability to notify user to eat/drink based off of customized user preferences
        2.2.1 Allow users to set when they want to receive meal notifications
            2.2.1.1 Users can enter what time of day they want to receive meal notifications
    2.3 Reminders to maintain exercise habit
        2.3.1 Allow users to set a time when they want to receive exercise notification
            2.3.1.1 Users can enter what time of day they want to receive exercise notification
3. Reminders/Notifications
    3.1 Ability to set reminders for events
        3.1.1 Ability to set dates and times
    3.3 Ability to receive reminders

Testing

We completed thorough testing of our application and from our testing we found no substantial bugs.