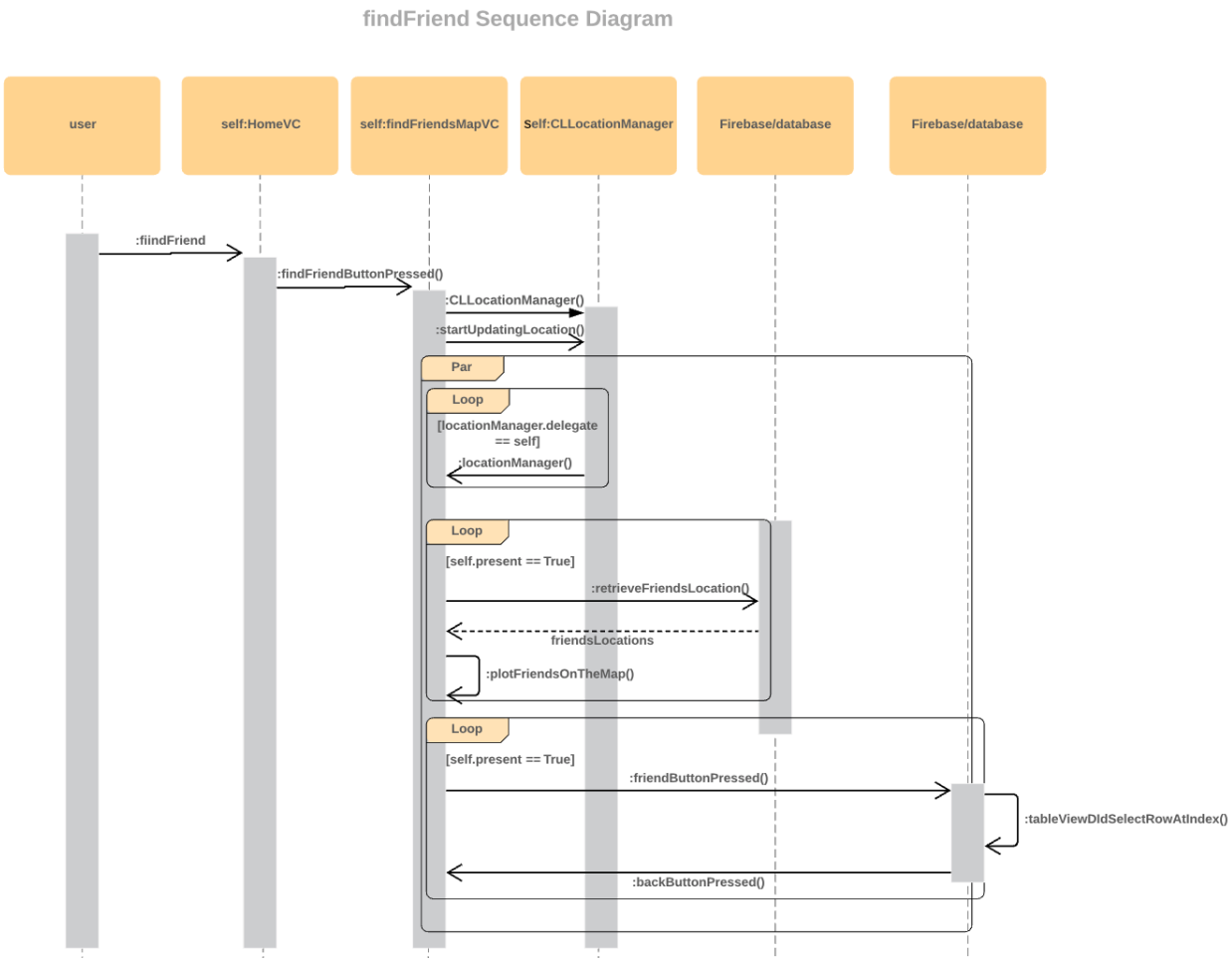
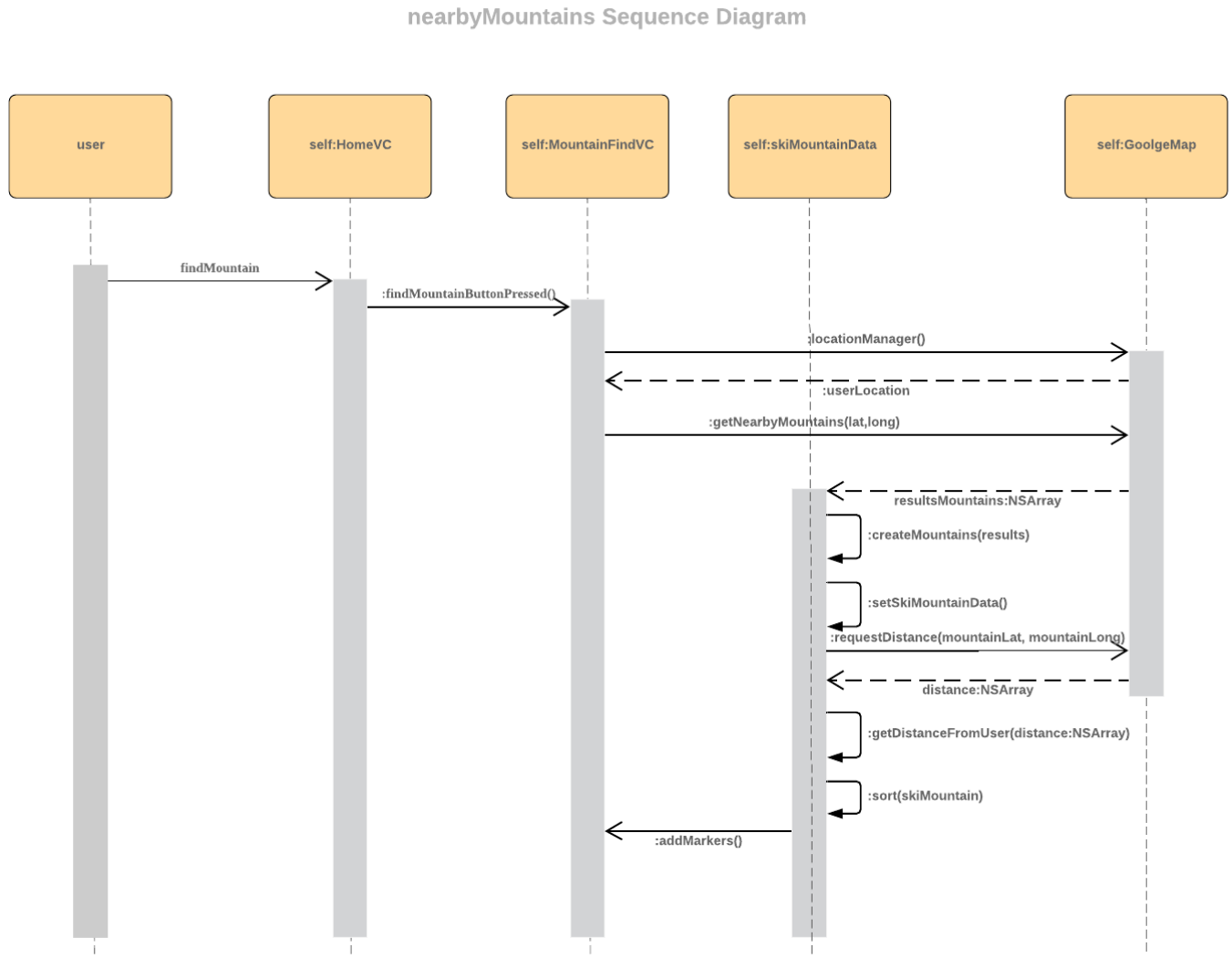
Team [0] Interim Release Deliverables

Sequence Diagrams

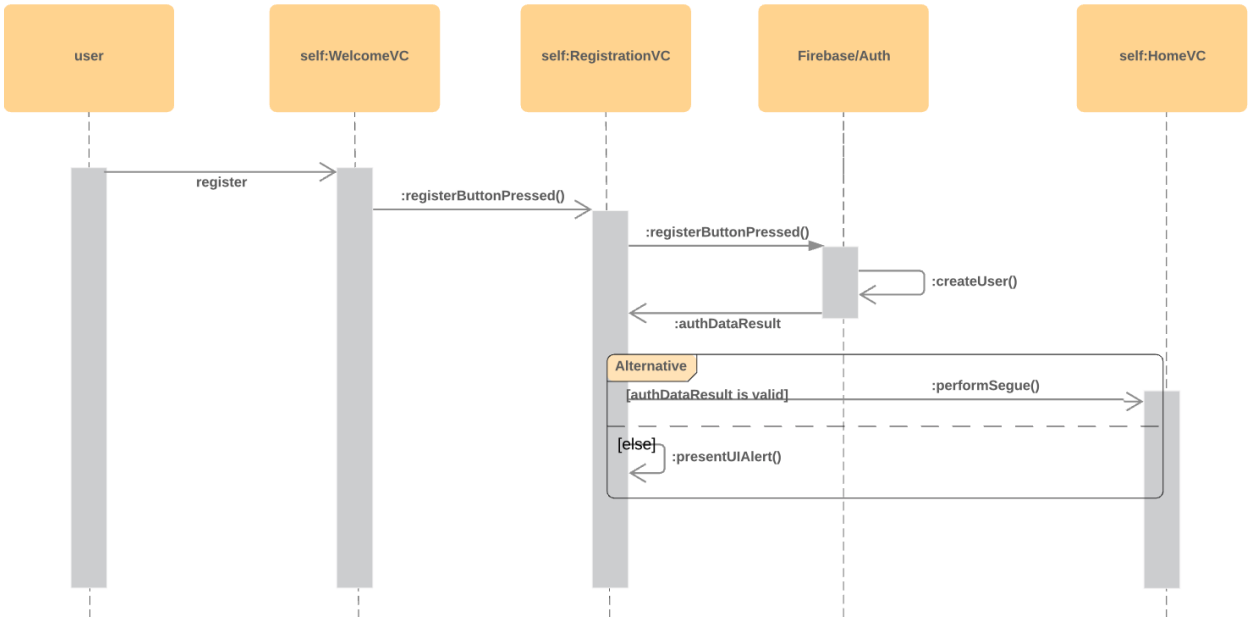


Corresponding to Use Case: PreciseLocation.

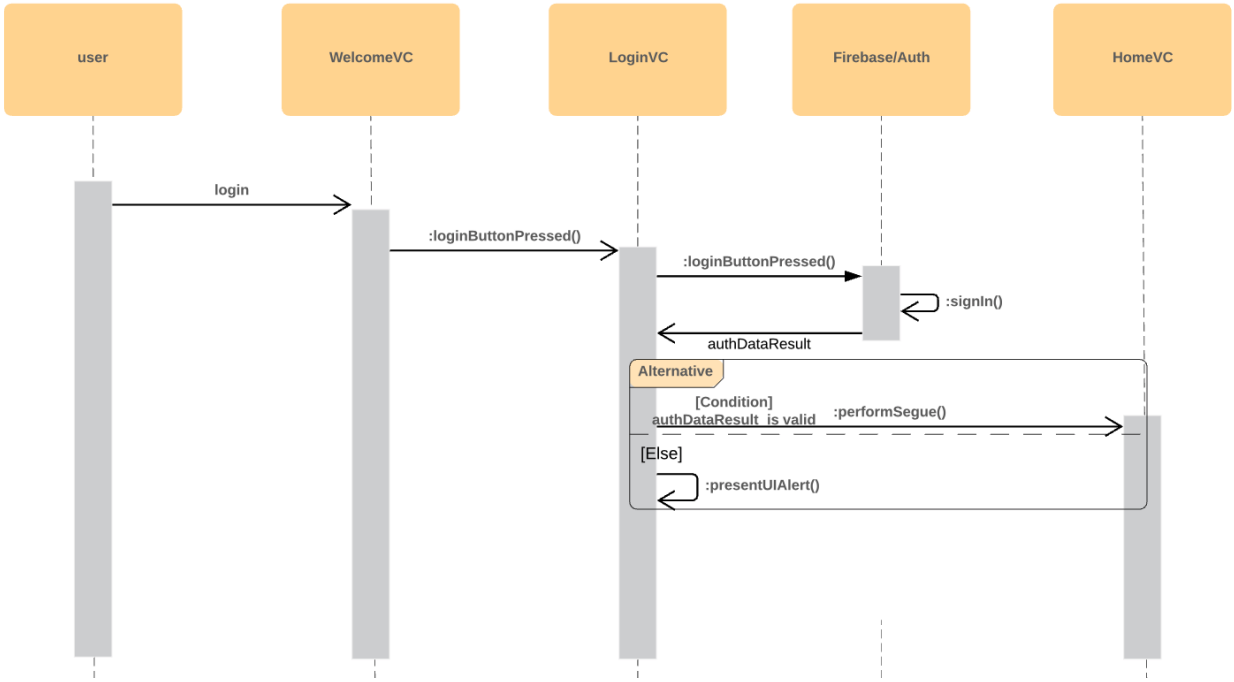


Corresponding to Use Case: NearbyMountain.

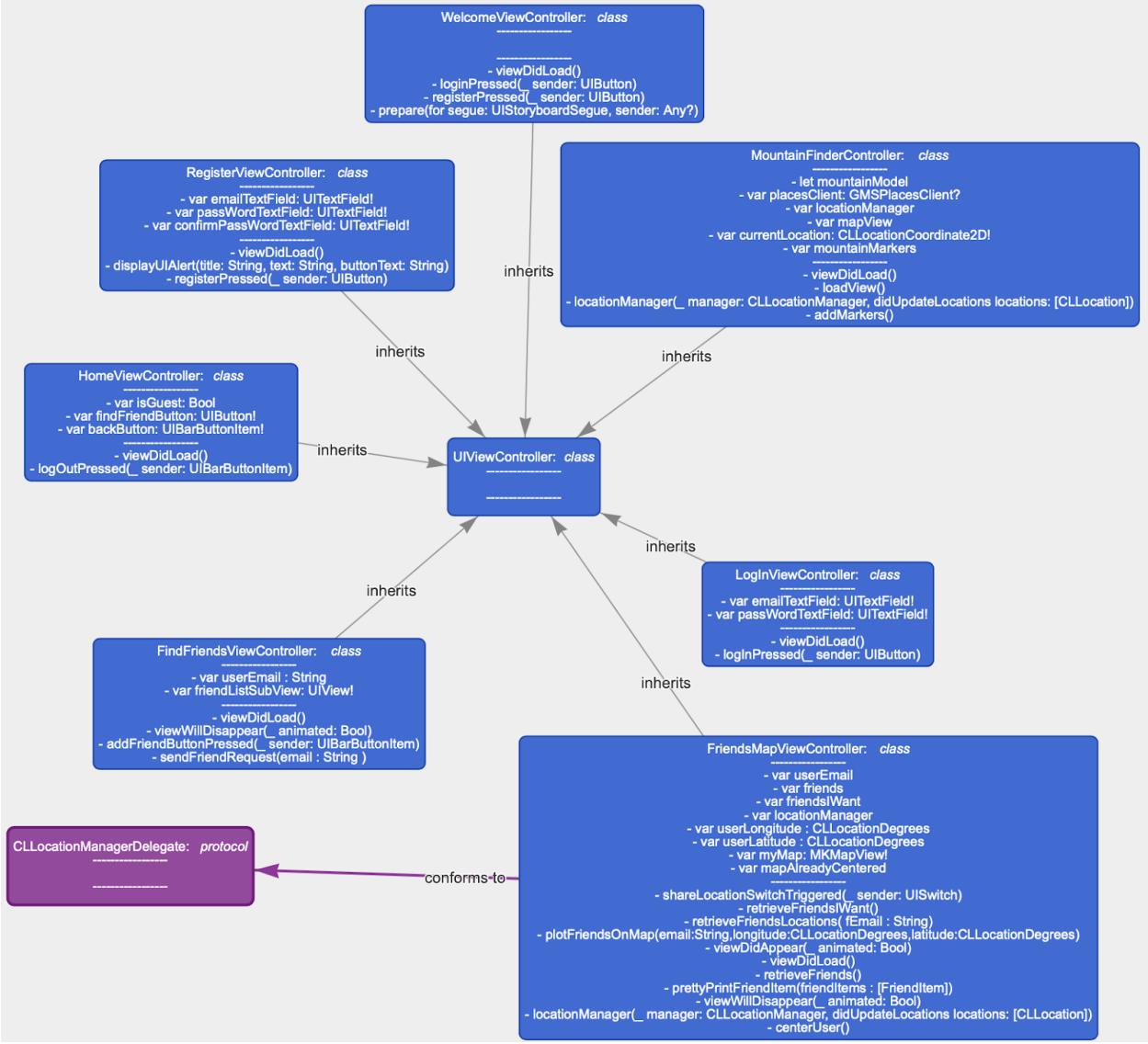
Registration Sequence Diagram

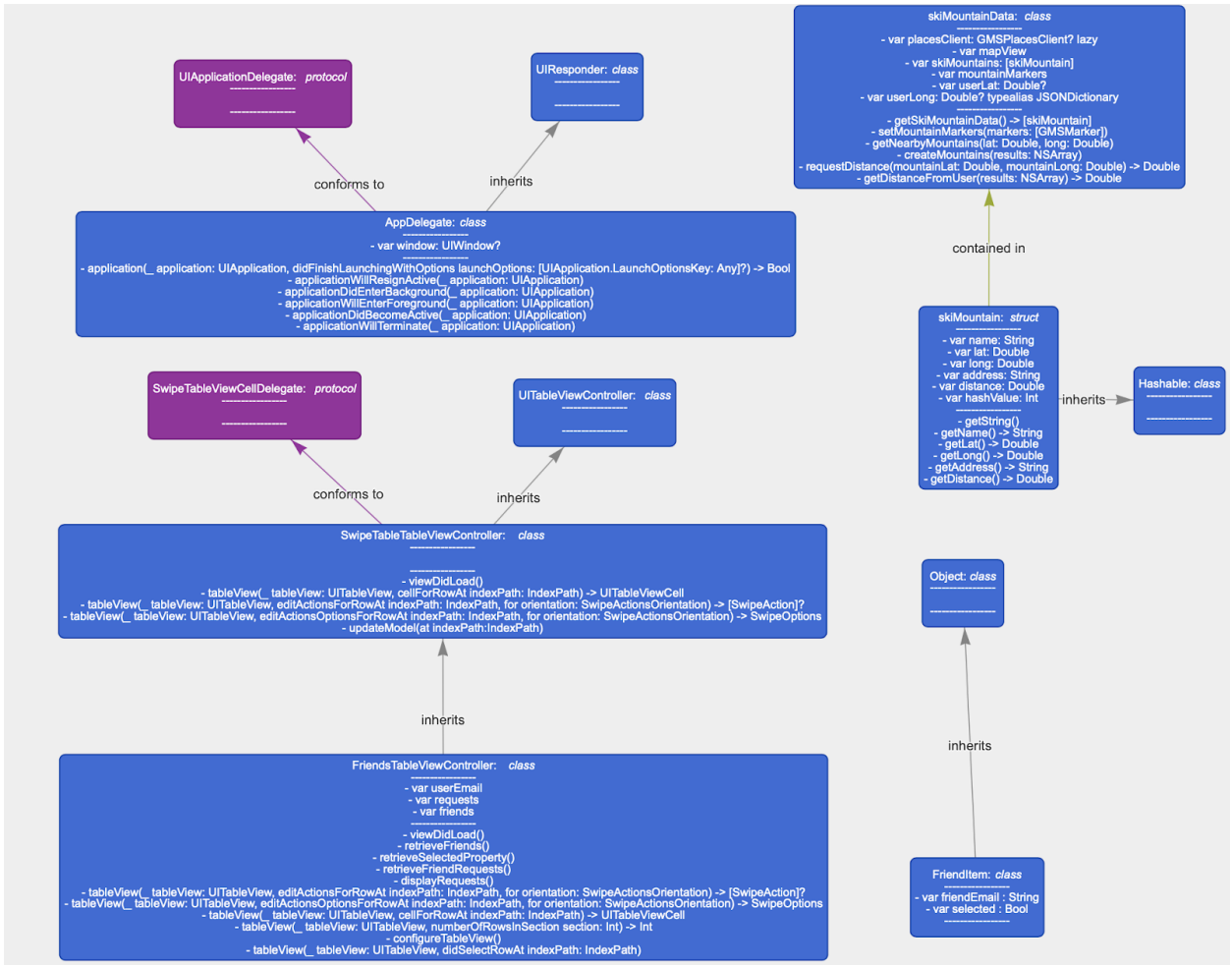


Login Sequence Diagram



Static Class Diagrams




skiMountainData: class
- var placesClient: GMSPlacesClient? lazy
- var mapView
- var skiMountains: [skiMountain]
- var mountainMarkers
- var userLat: Double?
- var userLong: Double? typealias JSONDictionary
- getSkiMountainData() -> [skiMountain]
- setMountainMarkers(markers: [GMSMarker])
- getNearbyMountains(lat: Double, long: Double)
- createMountains(results: NSArray)
- requestDistance(mountainLat: Double, mountainLong: Double) -> Double
- getDistanceFromUser(results: NSArray) -> Double
UIApplicationDelegate: protocol
UIResponder: class
conforms to
inherits
AppDelegate: class
- var window: UIWindow?
- application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool
- applicationWillResignActive(_ application: UIApplication)
- applicationDidEnterBackground(_ application: UIApplication)
- applicationWillEnterForeground(_ application: UIApplication)
- applicationDidBecomeActive(_ application: UIApplication)
- applicationWillTerminate(_ application: UIApplication)
contained in
skiMountain: struct
- var name: String
- var lat: Double
- var long: Double
- var address: String
- var distance: Double
- var hashValue: Int
- getString()
- getName() -> String
- getLat() -> Double
- getLong() -> Double
- getAddress() -> String
- getDistance() -> Double
inherits
Hashable: class
SwipeTableViewCellDelegate: protocol
UITableViewController: class
conforms to
inherits
SwipeTableViewController: class
- viewDidLoad()
- tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
- tableView(_ tableView: UITableView, editActionsForRowAt indexPath: IndexPath, for orientation: SwipeActionsOrientation) -> [SwipeAction]?
- tableView(_ tableView: UITableView, editActionsOptionsForRowAt indexPath: IndexPath, for orientation: SwipeActionsOrientation) -> SwipeOptions
- updateModel(at indexPath:IndexPath)
Object: class
inherits
inherits
FriendsTableViewController: class
- var userEmail
- var requests
- var friends
- viewDidLoad()
- retrieveFriends()
- retrieveSelectedProperty()
- retrieveFriendRequests()
- displayRequests()
- tableView(_ tableView: UITableView, editActionsForRowAt indexPath: IndexPath, for orientation: SwipeActionsOrientation) -> [SwipeAction]?
- tableView(_ tableView: UITableView, editActionsOptionsForRowAt indexPath: IndexPath, for orientation: SwipeActionsOrientation) -> SwipeOptions
- tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
- tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
- configureTableView()
- tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)
FriendItem: class
- var friendEmail : String
- var selected : Bool
UITableViewCell: class
inherits
mountainCell: class
- var mountainLabel: UILabel!
- var mountainAddress: UILabel!
- var distanceAway: UILabel!
- configureCell(mountain: skiMountain)

CRC Cards

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
MountainFinderController Gets users current location and displays it on the map along with markers for each nearby ski mountain Communicational Cohesion - 5	skiMountainData - Data Coupling - 5

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
skiMountainData Given the users current location, it gets 20 nearby mountains and calculates the distance from the user to each mountain Communicational Cohesion - 5	MountainFinderController - Data Coupling -5 skiMountain - Data Coupling - 5

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
ViewMountainListController Displays the list of nearby mountains with information about each mountain. Ie. name, address, distance etc Communicational Cohesion - 5	skiMountainData - Data Coupling - 5

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
mountainCell Assigns each mountain attributes to a specific label in the UI Communicational Cohesion - 5	skiMountain - Data Coupling - 5

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
HomeViewController Offer three segues for the user: login, register, guest login. Functional cohesion - 6	WelcomeViewController - Control Coupling-3

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
WelcomeViewController Give the user two main functionality of the application: find mountains and find friends. Functional cohesion - 6	FindMountainsViewController - Control Coupling-3 FindFriendsViewController - Control Coupling - 3 HomeViewController - Control Coupling - 3

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
FindFriendsViewController Get the friends' location from the database and notify the MapView. Informational cohesion - 7	FriendsTableViewController - Control Coupling - 3 FriendsMapViewController - Data coupling - 5

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
FriendsTableViewController Display the user's friends list and waiting for user to select his/her friends to get the location data from database Informational cohesion - 7	friendItem - Data coupling - 5

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
friendItem Formulate each friend’s information and property as an object Informational cohesion - 7	FriendsTableViewController - Data coupling - 5

Class Name, Responsibility, Cohesiveness Degree	Collaborator(s) and Coupling Degree
FriendsMapViewController Display the user and the user’s friends on the map in real time. Functional cohesion - 6	FindFriendsViewController - Data coupling -5

Design Approach

In the iOS application, one of the design patterns we are using is the model, view and controller design pattern also known as the MVC pattern. Since we have numerous different views that the user can see but a single set of data for ski mountains we needed it to be easily accessible. The model consists of a list of ski mountains objects that store information about each mountain such as the name of the mountain, latitude and longitude, address and distance from the user. The views are each screen the user can see from the welcome screen to the find nearby mountains screen to the find my friends screen. The controller deals with communication between the model and view. It can access the list of ski mountain objects and make a marker on the map for each mountain. It can also adjust the map camera and move it so it is centered as the users current location as well as adjusting how zoomed into the map you are and what type of map you are looking at. We also incorporated the singleton design pattern into our class managing the list of ski mountains. We did this as it is not necessary to create multiple instances of the same group of data. This way after initially creating the list of ski mountains nearby and you want to display a list of them to the user, you don't need to re create the list and calculate all the distances again.

In addition, we have used the observer pattern in multiple places in our iOS application. For example, in the GUI aspect, whenever we put a UIButton in the user interface and link that UIButton to the method usually called "UIButtonPressed", this means we have created an observer. As long as the button is present on the current view controller, the button is observing whether it gets pressed by someone. If somebody does this, the method linked to it will be triggered to perform corresponding events, ie. go to the next view controller, present UIAlert. The other case worth using the observer pattern is in the find friends feature since we need to update friends' locations on the map in real time. This was done by using the powerful API in firebase, and providing the observer methods with the appropriate database path, we're able to retrieve the other users' longitude and latitude data from firebase in real time.

Contribution Summary and Status Report

Stakeholders' Name	Contributions
Eric Partridge	1. Implemented the function to show mountains on the location. 2. Implemented the function to show a list of mountains based on distance. 3. Wrote CRC cards, design approach.
Shuze Liu	1. Drawed sequence diagrams. 2. Formalized the document. Connected each member,
Haoran Hu	1. Implemented the function to show the precise location of friends. 2. Drawed sequence diagrams, CRC cards, static class diagram. 3. Wrote design approach.
Damin Xu	1. Implemented scripts to get information of ski mountains and aggregated information into firebase.
Huiming Chen	1. Kept working on the website.

For the interim release, we have implemented the function to show the precise location of friends which is the creative feature in our app. Also, we implemented the function to get the user's location and use this location to locate mountains and show them on the map. Along with this function, we implemented a list to show mountains based on distance and we can add weather information to each mountain's interface. Moreover, we have built a database and wrote scripts to get information from the mountains' websites and aggregate them in the firebase.