Delegation Application

Senior Project

Erik Matthew Phillips

Computer Science
California Polytechnic State University
San Luis Obispo, California

June 2018

Table of Contents

Abstract

Delegation is a cross-platform application to provide smart task distribution to users. In a team environment, the assignment of tasks can be tedious and difficult for management or for users needing to discover a starting place for where to begin with accomplishing tasks. Within a specific team, members possess individual skills within different areas of the team's responsibilities and specialties, and certain members will be better suited to tackle specific tasks. This project provides a solution, consisting of a smart cross-platform application that allows for teams and individuals to quickly coordinate and delegate tasks assigned to them.

Introduction

General Project Scope

This project covers the implementation of two different types of applications: desktop and mobile. The desktop application was developed for the macOS operating system and the mobile application developed for iOS. In addition to the front end development, the project covers the creation of a shared backend (model) and a shared database (Firebase) for the two front-end implementations. Lastly, the project has an intelligent algorithm to determine and recommend tasks for team members based on the success and completion of previous tasks assigned to the user.

Project Access

The project has been made available at: github.com/erikphillips/delegation-app

Background

The inspiration for this project came from first-hand experience with needing to delegate bug reports to team members based on their skills and areas of expertise. From this experience, I identified a need for a customizable automation system to perform the delegation of tasks with precision and provide teams with the ability to get real-time updates throughout their work-cycle.

The Delegation application suite provides teams with the ability to generate an automatic task management platform, from which assigned tasks can be processed. Similar management systems are already in existence, however these systems do not typically provide prediction results for users in a team setting, focusing on the specialties that a user possesses. The goal of the project is to provide such a platform for which teams, users, or companies can deploy their own custom machine learning models to perform prediction calculations on tasks curated for their purposes and delegation goals.

The application data model is built around three core principals: the user, the task, and the team. First, a user or member consists of a specific account assigned to an individual. This user account holds information such as a first and last name, email address, and password. The user is able to join a team, create tasks, and view other people's tasks. Second, a task consists of information such as the title, description, priority, and status of the task. Additionally, the task contains a current assignee and originator, which are both references to an existing user. Lastly, a team is made up of a group of users for which all members of that team have access to the other team members and their tasks. Tasks will be assigned to a specific team, and that team will be responsible to delegating the task to a team member to resolve and complete. This has been modeled after a typical software development company's organization structure.

The application has a set of custom icons and logos designed in part with Peter Phillips. The icons feature deep blues and whites, which can be seen throughout the application as a common color theme. The full list of logos and icons can be found at the following location:

github.com/erikphillips/delegation-app/tree/master/assets/logos

System Architecture and Design

Both the macOS and the iOS applications are built using Apple's latest SDKs and frameworks (iOS 11.3 and macOS 10.13.4), Xcode (Xcode 9.3), and the latest version of the Swift programming language (Swift 4.1). The macOS application has been designed for all modern Apple computers and the iOS application has been built for the iPhone 6/6s/7/8 lineup (not including plus sizes). With additional design resources, the application can be scaled for different phones and tablets, such as iPhone X and the iPad Pro lineup.

General Design

The application suite is built around the premise of three main object classes: the user, the task, and the team. These three classes are represented throughout the project, with their specific details outlined below. Each object class has a unique, persistent identifier that is tied to the object and is generated on the initial creation of the data by the user. The keys for these identifiers are: UUID (User Unique ID), TUID (Task Unique ID), and GUID (Group/Team Unique ID). By using the three main object classes, the design was kept simple and this allowed for creating less back-end infrastructure to interface with the Firebase APIs.

By using these three different object classes, the data is able to be separated and remain distinct. This design decision was based on previous knowledge of establishing applications, as well as basic necessity for the management of the functionality within the application. Additionally, in future version of this application, the three objects can easily be expanded to incorporate additional features, such as access privileges or additional data fields.

General Object Classes and Observables

The main object classes have all been implemented based on an observable model. This means that when data changes within the object (either from a database change or because another part of the application modified data attributes within the object), any front-end view that is currently observing the object will be notified about the update through a callback. This has been utilized throughout the application to ensure that the user always has access to the most recent data, as this is especially important due to the shared nature of the application between the desktop and mobile versions. Additionally, there is a setup callback executed when the object has completed initialization, which allows for the front-end to present the updated information. The typical structure of the observable model is illustrated below:

```
    public var observers = FBObservers<User>()
    private var setupComplete = false
    var setupCallback: (() -> Void)? {
        didSet {
            if setupComplete {
                setupCallback?()
            }
        }
    }
```

A list of `FBObserver` objects is created for each of the main object classes (user, task, and team). The `FBObservable` code is illustrated below:

```
class FBObserverInfo<T> {
    var callback: ((_ data: T) -> Void)?
    weak var canary: AnyObject?
}

class FBObservers<T> {
    private var observers = [FBObserverInfo<T>]()
    func observe(canary: AnyObject?, callback: @escaping (_ data: T) -> Void)
    func notify(_ data: T)
}
```

The observer model works well in conjunction with the database, as Firebase implements a real-time database which can notify observers when updates to the data occur. The object that is observing the data can then pass along an update itself and notify any views which are observing the object for which there has been an update to.

User Object Class

The user object consists of the following attribute fields within the class:

1. **UUID:** a string representing the unique ID for the user. The unique ID is a key that is used in the database and is assigned by the Firebase API through the create new user process discussed further below in a subsequent section.
2. **First name:** a string for the first name of the user.
3. **Last name:** a string for the last name of the user. Combining the first name, a space, and the last name will generate the user's full name.
4. **Email address:** a string representing the email address for the user. The email address will be the same as the email address registered with Firebase and acts as the username when the users logs into the application. Additionally, the email address can be a unique identifier for the user when such an identifier needs to be displayed via the front-end of the application.
5. **Phone number:** a string representing the unformatted phone number for the user.
6. **Teams:** a list of team objects for each team that the user is a member of.
7. **Tasks:** a list of task objects for each of the tasks that are currently assigned to the user.

Within the user object class, there are Firebase observable methods that allow for the management of the instantiated user object. When a user object is created, it is done via a persistent observe API call into Firebase, which provides callbacks for when the data changes within the database (more details on this in the Firebase discussion below). Examples of these methods are shown below:

```
func addNewTeam(guid: String)
func leaveTeam(guid: String)
func updateCurrentUser(firstname: String?, lastname: String?,
    email: String?, phone: String?, password: String?)
```

These methods allow for consistency within the object itself and also provide an easy method for the maintenance of the database. In addition, any objects that need to be represented within this user object class will utilize the corresponding methods within the other classes, thus providing consistency.

Task Object Class

The task object consists of the following attribute fields within the class:

1. **TUID:** a string representing the unique task identifier created automatically when generating the task for the first time and uploading it to the database.
2. **Priority:** an integer representing the relative priority for the given task. The priority value can range from 1 (representing the highest priority) to 5 (the lowest and default priority).
3. **Description:** a large string for the description of the task, typically spanning multiple lines within the rendered task details view on the front-end.
4. **Team GUID:** the group identifier for the team to which the task is currently assigned.
5. **Status:** a TaskStatus object representing the current state/status of the task. The possible values are none (error), open (default), assigned, in progress, and completed.
6. **Assignee UUID:** the current assignee's UUID.
7. **Originator UUID:** the UUID for the originator of the task.
8. **Team:** a team object representing the current team that the task is assigned to.

The task object class is also built upon the observable model, and the following methods are implemented to assist with the maintenance of the object and the database:

```
func advanceStatus()
func changeAssignee(to newUUID: String)
func changeTeam(to newGUID: String)
func updateTask(title: String?, priority: String?,
    description: String?, status: String?)
```

Team Object Class

The team object consists of the following attribute fields within the class:

1. **GUID:** a string representing the unique identifier of the team. In the current version, the GUID and the team name are the same value.
2. **Team name:** the current name of the team.
3. **Description:** a string representing the description of the team.
4. **Owner UUID:** the UUID for the current owner of the team and default task assignee.
5. **Members:** a list of strings representing the UUID for each of the members of the team.
6. **Tasks:** a list of task objects for all the tasks that belong to the team.

The team object is initialized with an observable model as well, but as most of the complex computations happen within the task and user objects, the team class has only one method implemented for its management:

```
func updateTeam(description: String?, owner: String?)
```

Shared Model

To make development easier between the two applications within the Delegation suite, a shared model was created to implement custom objects and Application Programming Interface (API) calls to the database. This allows for easy development and consistent results across the two different front-end applications. A sampling of the Utility API calls that are available to both applications are as follows:

```
validateEmail(_ email: String) -> Status
validatePhoneNumber(_ phoneNumber: String) -> Status
validatePasswords(pswd: String, cnfrm: String) -> Status
validateUser(_ user: User?) -> Status
validateTask(_ task: Task?) -> Status
validateTeam(_ team: Team?) -> Status
unformat(phoneNumber sourcePhoneNumber: String) -> String
format(phoneNumber sourcePhoneNumber: String) -> String?
```

The above functions are utilized throughout the create account process within the application and when the user is modifying or updating their personal information for their account. Validation of the user provided data is especially important as the content of the user data is unknown. Therefore, any user provided data which is stored in the database must go through the corresponding validation functions. The entire application has been built using Swift, which is a type safe programming language that does not allow for unchecked access into memory, which is the typical culprit of security issues, therefore there is not a large security threat when using and displaying user input.

In order to provide a consistent structure within the database, the following API calls were implemented. These methods were designed specifically to ensure that the same data is created and generated when creating new users or logging in current user accounts, etc. The methods implemented are as follows:

```
createNewUser(email: String, password: String,
    callback: ((_ uuid: String, _ status: Status) -> Void))
logoutCurrentUser() -> Status {
loginUser(username: String, password: String,
    callback: ((_ uuid: String?, _ error: Error?) -> Void)) {
emailAddressInUse(email: String, callback: ((_ status: Status) -> Void))
teamNameInUse(teamname: String, callback: ((_ status: Status) -> Void))
fetchAllTeams(callback: ((_ teams: [Team]) -> Void))
fetchAllTeamGUIDs(callback: ((_ teams: [String]) -> Void))
fetchTeamOwnerUUID(guid: String,
    callback: ((_ status: Status, _ uuid: String) -> Void))
performWelcomeProcedure(username: String, password: String,
    callback: ((_ user: User?, _ tasks: [Task]?, _ status: Status) -> Void))
```

Methods that utilize a callback perform asynchronous network tasks, thus the callback is executed when the asynchronous tasks have completed.

The `createNewUser` method generates a new Firebase user instance with the provided email address and password. This allows for users to login with their email address and password, with the application database not needing to store the passwords or manage encryption itself. Rather, this responsibility is given to Firebase and the Google Cloud Platform infrastructure. The only methods that are used to create or deactivate an active user session is the `loginUser` and `logoutCurrentUser`, which are wrapping methods around the Firebase API calls. Additional methods of login include Facebook, Github, or Google, however these have not been implemented in this version.

The `InUse` methods were designed and implemented to manage creating new users or teams and to ensure that the email address or team name is not currently in use. The application was designed so that no user can have the same registered email address as another user. Additionally, there is a requirement that team names are different from one another. The initial database in the first version was created to be used by one organization within a company, and if additional companies or organizations were to utilize the application, a new database would be created for their specific company. This ensures that billing can remain separate, if applicable, but it does create a scalability issue which has not been addressed within the scope of this project.

Lastly, the `performWelcomeProcedure` methods and the `fetch` methods wrap multiple repetitive calls into one API for the front end to handle. This allows for more seamless code integration into the two different front end applications without having to repeat API call sequences multiple times in the two different front-end applications.

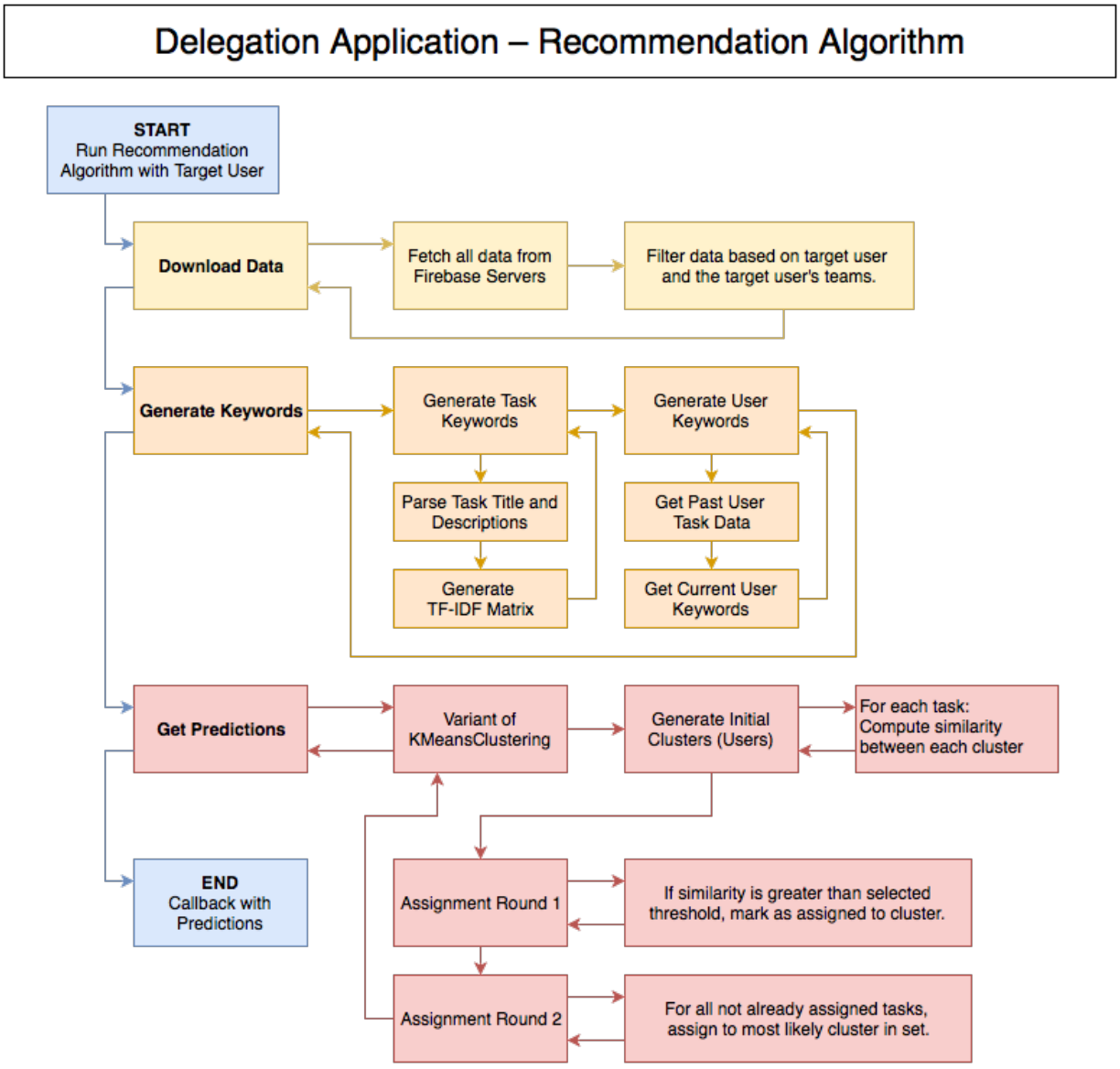
Recommendation Algorithm

Within the shared model is a recommendation algorithm that will generate predictions for users on currently unassigned tasks that may be able to be completed by the targeted user. The algorithm works based on a variant of the K-Means Clustering algorithm in which each user within a team is considered a cluster. From there, the task data is parsed and the title and description are used to generate a sparse Term-Frequency to Inverse-Document-Frequency (TF-IDF) matrix. The user's data is downloaded and parsed to generate keywords for the user, based on their past task assignments. The data in the TF-IDF matrix is then used to compute similarity between any given task and the user (the cluster).

Tasks currently undergo two rounds of assignments. The first round looks at all tasks and assigns them to a cluster if the similarity is greater than a specified threshold. Within the first round of assignment, a task may be assigned to multiple clusters if it has a high enough similarity to other tasks correlated to that cluster. The design decision and rational behind this is that an unassigned task may be able to be completed by more than one user, and thus the task should not be limited to only one assignee. The second round of assignment attempts to make up for tasks that did not meet the initial threshold in the first round of assignments. This second round will look at all the tasks that have not already been assigned and assign them to the cluster in which they most likely belong. After these two rounds of assignments, all the tasks should be assigned to a user cluster.

The overall goal with the recommendation algorithm is to provide recommendations for assigning tasks that are open to users within the team. Once the classification takes place, the tasks are presented to the user in the front-end application views, for the user to optionally take ownership of the tasks. With future versions of the application introducing membership roles within the application, users with higher privileges would be able to see recommended tasks for the people they manage and assign the tasks to them from there. Additionally, future versions may include automatic delegation of tasks, however this would require more data and an improved algorithm to determine precisely which user should be assigned the task.

The flowchart for the recommendation algorithm is illustrated below:



Firebase Realtime Database

The Firebase Realtime Database is open source and powered by Google which allows for access across iOS devices to create a model which can easily be accessed by the developer. The use of Firebase removes the burden on developers to implement and maintain a server and backend for the application to successfully run. This goal is accomplish very successfully with the design and availability of Firebase for iOS applications. Firebase however was not designed for use with macOS and use with macOS is not officially supported by the Google Firebase team. Therefore, in order to get the instance of Firebase operating within the macOS application, it required downloading the source code and setting special targets within the project. This allows for use of the Firebase API through the macOS application, but the initial connection to the Realtime Database can be rather unreliable.

In order to get an active instance of Firebase within the macOS application, the Firebase application must be configured within the AppDelegate class for the Xcode application. The following is the required code, but as the macOS application does not have the same structure as the iOS counterparts, there can be errors when attempting to access Firebase when the application is still waiting on the configuration to complete.

```
func applicationDidFinishLaunching(_ aNotification: Notification) {
    FirebaseApp.configure()
}
```

Once the Firebase instance has been configured, the realtime database is able to be accessed throughout the macOS application. The following code is specifically for the iOS applications, which will run when the application has been launched by the user, and before the welcome view is presented:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    FirebaseApp.configure()
    return true
}
```

The database has a structure that will allow for rapid access and processing of the data, which is stored in a JSON format. Due to the JSON storage format of data, it can be rather difficult to store objects the same way they are utilized in Swift, therefore some design decisions were made for the database in order to provide the most efficient access to the data, which is outlined below. Additionally, the schema and live database can be found at:

https://delegation-app.firebaseio.com/

```
delegation-app/
      tasks/
            <TUID>/
                  assignee: String of type <UUID>
                  description: String
                  originator: String of type <UUID>
                  priority: Integer
                  status: String of type Status
                  team: String of type <GUID>
                  title: String
      teams/
            <GUID>/
                  current_tasks/
                        <TUID>: String of type <TUID>
                  description: String
                  members/
                        <UUID>: String of type <UUID>
                  owner: String of type <UUID>
                  teamname: String
      users/
            current_tasks/
                  <TUID>: String of type <TUID>
            information/
                  email: String
                  firstname: String
                  lastname: String
                  phone: String
            teams/
                  <GUID>: String of type <GUID>
```

The structure of the Firebase database is designed with great intention and planning. The design requires an understanding of not only the data, but the way in which the data will be accessed. Additionally, the matter in which the data is saved also plays a large part in the structure of the database. Because of the JSON format of the database, conversions often need to be made to ensure that data is preserved when translating from a Swift object model to the JSON model. This process can be rather slow and cumbersome to deal with, but the speed of access and the observability of nodes within Firebase offers many worth-while features.

Due to the design of the Firebase database, a specific path is able to be loaded without having to load the entire database into memory. For instance, to gather specific information about a user (assuming that the UUID is known), the path within the database would be: `/users/UUID/information/`. This would then only download and gather the data rooted with the `information/` node, thus saving network bandwidth for the app. In addition to being able to inspect data rooted below a specific node, the Firebase database allows for observable data, which performs a callback when any data is updated below the root which is being observed. This allows for the application the receive real-time updates when changes are made without the need to constantly refresh data or inspect the contents of the database through costly IO operations.

For this application suite, the database went through several different iterations and many design changes were made throughout development. In order to get the most efficient user experience, some compromises were made, including requiring unique team names within a single Firebase database instance. The unique team names allow for writing data to the database much more efficiently, as a call does not need to be made to gather information about a team. Having unique team names has some major drawbacks however, including the need for a unique database instance for each company or organization where team names may overlap.

For managing users, one iteration of the database was considered to require unique usernames for each member within the site. This would be in addition to the unique email address required for registration and login. When requiring unique user names, accessing the database would have been made much simpler and efficient, but the username would not really serve any purpose beyond the developer's role. The username would not mean anything to the users within the app, and companies are not built around usernames, rather they utilize email addresses and full names. Therefore, the database uses a unique user ID (UUID), which thus requires a load from the database to display any information about the user (such as their name or email).

For interfacing with the Firebase database, a custom wrapper around the Firebase API calls was created to ensure that data remained consistent across the different instances of the two front-end applications. This was previously discussed in the shared model section.

iOS

The iOS application starts with a login screen and allows the user to either create a new account or login to their existing account. When logging in, the user is presented with a TabView, consisting of four tabs: Home, Tasks, Teams, and Settings. These tabs build the foundation of the user's navigation experience. Having four tabs for the most used features of the application allows the user to quickly change the screen they are viewing. Additionally, since each tab is a distinct view, the user's current navigation session is preserved across tab view selection changes. This allows the user to jump between different items that they may be viewing within the application.

The iOS operating system is designed around the user experiencing one screen/window at a time, thus there is no option to view other items, windows, or instances of the application. Keeping this is mind, the iOS application was designed to maximize the screen space that is given and through the use of TableViews, the user is easily able to scroll through content presented within a window.

Welcome View

On the login window, the user is able to enter their credentials in the text fields provided. If they have not yet created an account, they are presented with the ability to do so. One affordance implemented into the account creation process is that if a user enters an email address and password, then clicks the create account button, that information will be pre-populated in the account creation view that follows. This will save the user time if they believe that they have a registered account, but do not, and makes the application appear to be more professional.

When a user is entering their credentials, the return button on the device has been modified in this window instance, and throughout the entire application, to move the focus from the current text input field to the next field and finally submit the form using the default button or action when the user has entered all their information and completed the form. This feature was added throughout the application to provide a better user experience and to reduce the friction that a user may feel when using the application. In future versions of the application, there will be the ability to skip the login process, and save the user credentials to allow for faster access and speed up the sign-on process.

Account Creation

Starting with account creation, the user is guided through several different views to collect their information and initialize their account. Once the user has provided their verified information, they are given the opportunity to create a new team, join an existing team, or skip the rest of the account creation process. This choice was provided to the users so that they are able to easily get started using the application without needing to spend time on account setup. When choosing to join a team, the user is given the opportunity to view all teams within the database instance that they have access to and join whichever teams they would like. Once a user has created an account, the user is brought back to the login page that they were at before. In future versions of the application, the teams will have special access permissions so that new users cannot simply join any random team. Additionally, there will be user roles and when a new user is creating a new team, they will become the manager and owner of that team which they created.

Main View and Navigation

The main view within the application consists of a TabBar with four tabs. Each of the tabs is equipped with a Navigation Controller so that the user is able to easily move between windows that are presented to them. Since the TabBar Controller maintains the view state when switching between tabs, it is the ideal method for presenting this type of content to the user. Additional presentation methods which were explored include the implementation of a sidebar, however upon initial exploration of the feature, the difficulty of implementation, along with the poor layout of its contents and not enough actual content to place within the sidebar, it was not chosen for this version.

Tasks

The tasks view is implemented based on a TableView, which allows each row within the table to contain information about each task that the user is currently viewing. From here, the user can select a task to view more detailed information and also create a new task.

Viewing the details of a specific task uses a predefined TableView to display the specific contents and details about the task. From this view, the user is able to edit information about the task by pressing a button to indicate that they would like to edit the information. By requiring the user to press an additional button in order to edit the information, there is a slight barrier to this process, but it prevents a user from accidentally modifying the task. From this view, the user is also able to edit the assignee and the team that the task is associated with. Both of these methods present an additional window to the user for this selection due to the fact that the user will need to select the specific entity for which they would like to change the task's association with. This task detail view is shared across multiple different parent views, which provides a better user experience because the presented view remains the same as in other contexts.

Predictions

The predictions tab is the first tab displayed, as the goal of this is to show a user any new unassigned tasks the moment they login to the application. From here, the user is able to run the recommendation algorithm to generate results. Currently, the initiation of the recommendation algorithm requires manual execution as the algorithm uses a lot of device resources (both network bandwidth and device CPU resources). In future versions, the application recommendations will be generated on a server, therefore reducing the amount of resources that the user's device will need.

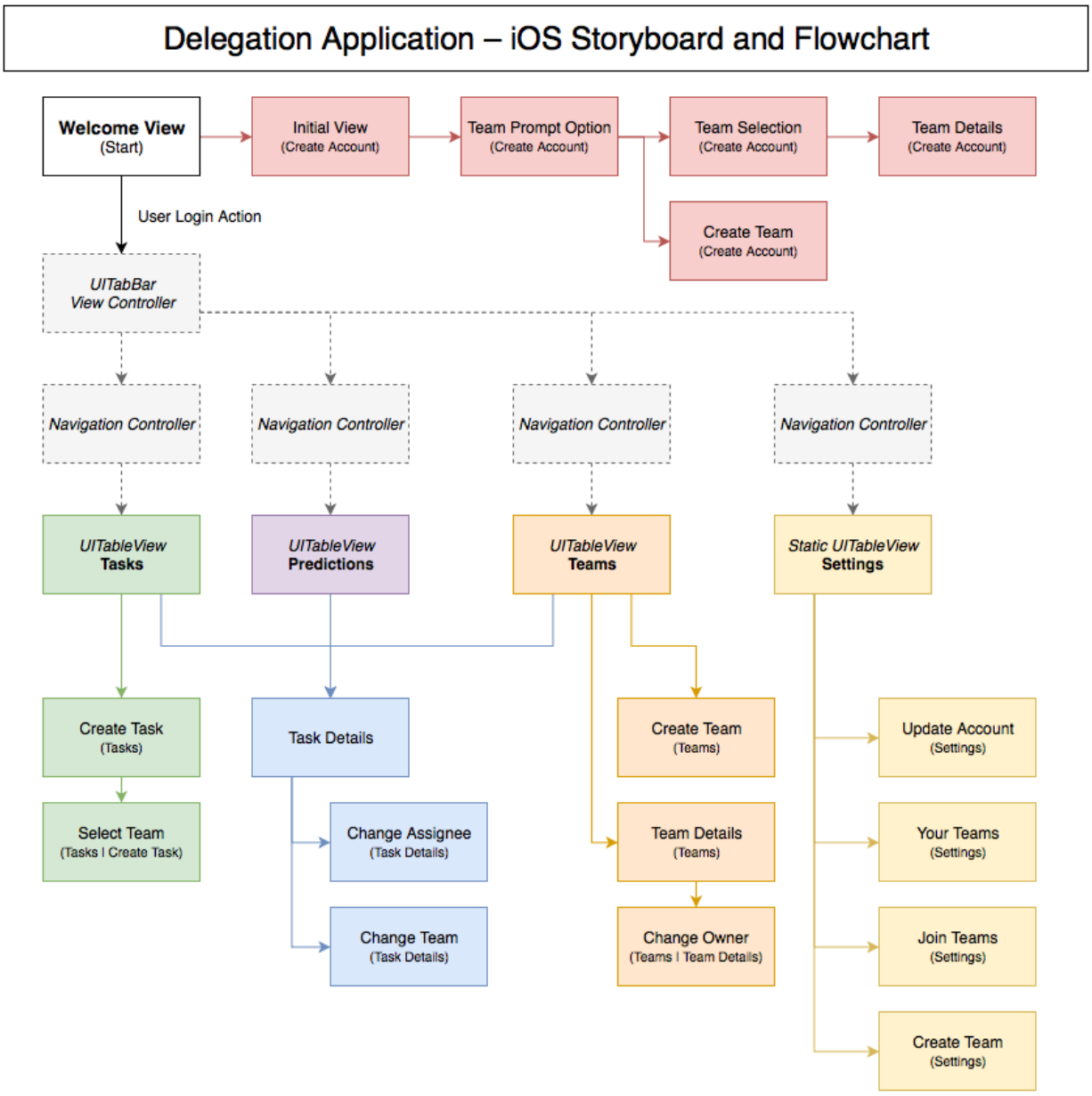
Teams

The teams tab is where the user is able to view all the teams for which they are associated with and are displayed in a TableView. From here, users can select a team to view all the tasks associated with the team and also select the information icon to view detailed information about the team. When viewing detailed information about the team, the user is able to edit information about the team itself and save those changes. In future versions of the application, only the owner will be able to modify the team information and its members as there will be specific user access permissions provided to each user.

Settings

The settings tab consists of a static TableView containing information about the user and their profile, cell links to view the current teams and to join existing teams, the ability to logout and delete an account (currently unimplemented), and a link to contact the developer of the application. Additionally, there is information about the application version number and the specific build number of the application which can be utilized when attempting to debug issues.

The following chart is the storyboard for the iOS application:



macOS

The macOS application was built around one main window which contains the listings of tasks for the current user. The main window is built upon a vertical split view controller, which allows for the display of two sections of data within the window. There is an additional toolbar at the top of the window which allows for access to the other features of the applications, such as account settings, creating a new team or task, running predictions, and refreshing the current view's data.

Welcome View

When the application is launched, the user is presented with a window in which they can enter their account credentials. Wherever a user is within the application, there is a keyboard shortcut (⌘+O) which allows this window to be opened and to launch a new session of the Delegation application when new credentials are entered. From this window, a user is also able to create a new account.

Account Creation

When creating a new account, the macOS application simply asks for the basic profile information for the user. If the user would like to create or join a team, they are able to do so from the settings window once they have created their account and signed in. This method was used in the macOS application as a method for trying to eliminate the amount of windows presented to the user. In future versions of the application, the account registration process can be streamlined to match the iOS application.

Toolbar

The toolbar within the main application window is able to be customized by the user and provides the means to access certain portions of the application. From here, the user can select the icon to create new tasks and teams, run the prediction algorithm, and view their account settings. Additionally, there is a segmented control within the toolbar that allows basic filtering of the tasks displayed within the main table view of the window. A user can optionally refresh the task data within the main view by using a refresh button within the toolbar. In future versions of the application, the search feature will be implemented and will be accessible from this toolbar to locate specific users or tasks.

Segmented Control and Filtering

Within the segmented control buttons used for filtering, there are three options: Personal, Team, and Recommendations. The first (personal) displays all tasks that are personally assigned to the current user from across all teams that they are a part of. The second (team) displays all the tasks across all users for the teams that the current user is currently a member of. The last option (recommendation) displays all the recommended tasks generated by the prediction algorithm for the current user. These are the three main options that were brought over from the iOS application, and the segmented control provided the most similar interface to the TabView used in iOS.

Sidebar Content

The sidebar content with the main window provides an additional layer of filtering. The user is presented with two different options: all teams or specific teams. Here, the user is able to select to view tasks for all teams, or select a specific team from the dropdown menu. When the user selects the different options, the displayed tasks will change to meet the filtering criteria. This sidebar filtering works in conjunction with the segmented control filtering to fine tune the tasks which are displayed to the user.

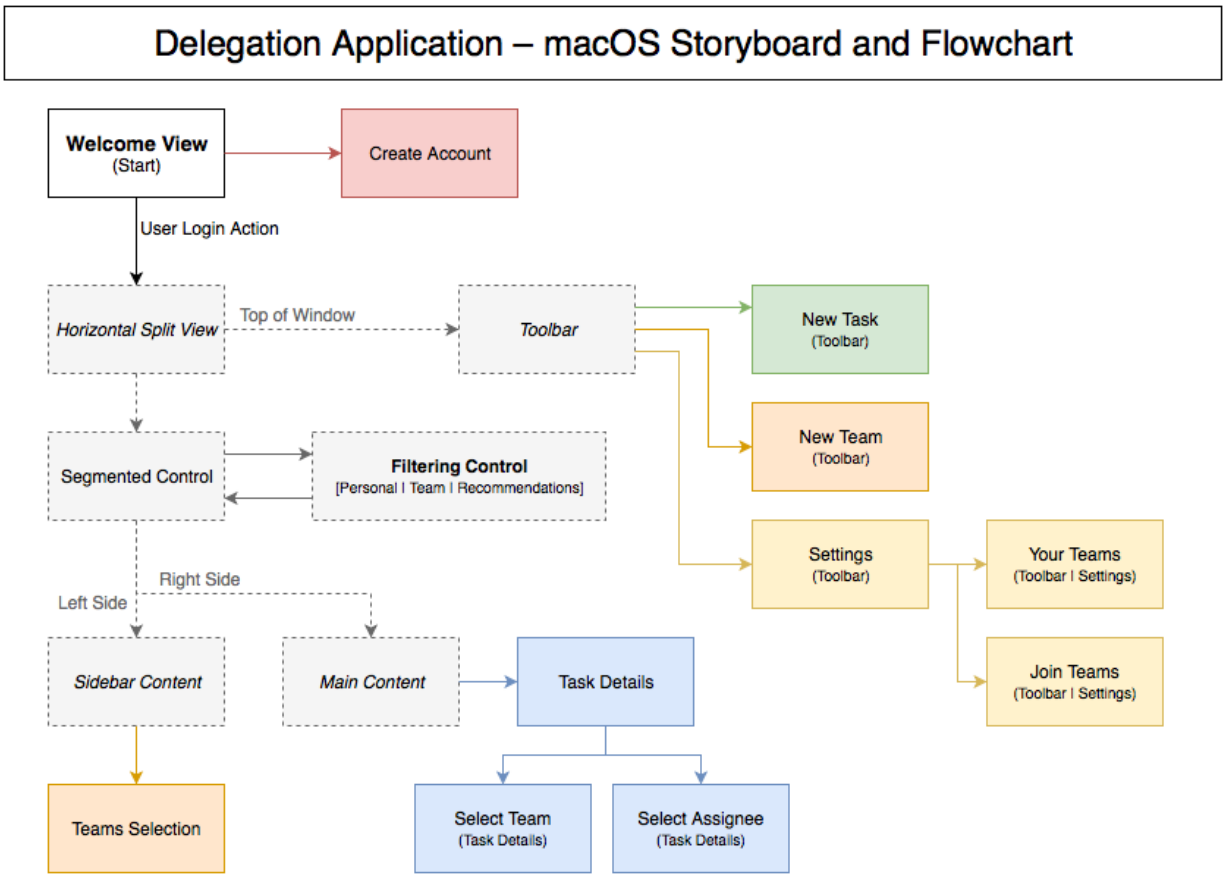
Main Content

The main content within the window are the tasks. Each task is given a specific row and the task information is displayed within the columns of the table. Currently, the columns are able to be arranged within the view, but there is no sorting or specific task filtering (such as deleting the listed tasks). In future versions of the application, the user will be able to sort the results based on different columns and change the tasks that are selected. Additionally, there will be the ability to select multiple tasks and to modify any selected number of tasks at the same time.

Task Details

By double-clicking on a selected row within the main content, the details for the selected task are displayed. The details are brought up in a separate window and multiple windows can be displayed at the same time. Within the task details window, the user is able to update the task information and change the associated team or the assignee, both of which are displayed via a new popover window.

The following chart is the storyboard for the macOS application:



Data Experiments for Task Prediction

One large aspect of this project is the recommendation algorithm and the ability to provide users with recommended tasks for them to complete. In order to provide that to the user, the application will need to utilize a prediction algorithm to generate a list of tasks for each user. The algorithm that is currently implemented is rather rudimentary, but with the increased power of machine learning (and on-device processing with CoreML), these predictions have the ability to make a larger impact.

In the course *CPE 369: Introduction to Distributed Computing*, the final project was open to student choice for whatever project the students would like to cover. The project I decided to work on was related to my senior project in that my team attempted to process a large dataset of software issues to find similarities between bug reports. The goal of the project was to provide labels and a correlation between bug reports within the dataset that was used, which contained over 200,000 reports from both the Mozilla and Eclipse projects.

The dataset is called *Mozilla and Eclipse Defect Tracking Dataset* and can be found here:
https://github.com/ansymo/msr2013-bug_dataset

After doing some substantial research when starting this senior project, this dataset was the largest and most useful one I could locate at the time. Additionally, with the direct connection to the goals of the project, the Mozilla and Eclipse bug dataset seemed to be the perfect fit for this project.

Data Project Description

There are two main phases of the project which include parsing the data and running analytics for the data, both of which are described below. The first step is parsing the data.

Data Parsing

The parsing phase takes the raw data distributed across multiple files and joins reports together to create a record based on the ID of the bug report and includes all the fields that the data analytics will be performed on. Specifically, the fields that the project worked with are as follows: report ID, short description, component, assignee, status, and priority. The attributes of the Delegation application tasks follow a similar design, with additional fields for easier task management.

Data Analytics

The second phase of the project deals with data analytics, which is broken into two different parts: Data Extraction (TF-IDF matrix) and K-Means Clustering. To generate the TF-IDF matrix, the following algorithm was used:

$$TF(d,t) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

$$IDF(t) = log_2\left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}}\right)$$

$$TF\text{-}IDF(d,t) = TF(d,f) \cdot IDF(t)$$

The term frequency calculation provides normalization as the number of terms is divided by the total number of terms within the document. This more complex equation is not required, but adds additional benefit so as not to skew any data calculations by means of normalization. The resulting TF-IDF matrix contains weighted values corresponding to each word that appears in the raw data files. As an additional optional step, stop words are removed from the calculation to provide better accuracy.

In order to provide a data analytics, the project focused on a clustering method called K-Means Clustering, which starts by choosing K cluster centers, then assigning each record to a cluster. Once all records are assigned a cluster, the center of the cluster is recalculated and then the assignment phase starts over again. This repeats until there are no changes to the clusters (or until the sum squared error stops changing). This process should be repeated multiple times due to the random element of selecting the original centers as the choice of the center can greatly skew the final results.

Data Project Results and Conclusions

The results from the project were rather inconclusive and did not consistently classify reports into the same clusters that they belonged to. Because of this and the inability to acquire better data to begin with, the results from this project were not included in the database for the Delegation application. With additional data, and more specifically, historical data for each task, better analysis can be performed on the records to determine the recommendations for task assignment. In future versions of the application, historical data can be captured and maintained within the database for each task, which will help labeling data and providing specific accuracy measurements for the success of the algorithms.

Future Work

The Delegation application suite is far from completion and there are many improvements that I would like to make in addition to features that the application can make use of. Due to the massive amount of work that comes along with designing from the ground up two applications for separate platforms, I was not able to accomplish all the goals that I set out to. Some notable features that I would have liked to include are as follows:

1. Separation and support for multiple companies/organizations.
2. User roles within specific teams (including positions such as team owner, manager, user/member, administrator, auditor, etc.).
3. Advanced automation/recommendation abilities, including:
    1. Enhanced token parsing (lemmatization, token stemming, special character/phrase identification, n-grams tokenization, named entity recognition, etc.).
    2. Clustering of tasks on a per-user basis.
    3. Server support for machine learning algorithm processing.
    4. Utilization of on-device machine learning capabilities (i.e. CoreML).
4. On-device storage and offline application use.
5. Custom sorting and filtering in addition to specific data fields for displayed tables.
6. Application documentation (including in-app help pages).
7. Design and distribution for multiple mobile device classes and sizes.

A full list of the open issues and feature requests can be found online via GitHub at:
github.com/erikphillips/delegation-app/issues

I plan on continuing development of the project and use it as a proof-of-concept for the potential future implementation of a similar system.

Reflections

Throughout this process I have enjoyed the ability to work on and design my own application, taking into account all the software design principles that I have learned throughout my time at Cal Poly. Combining knowledge from Mobile Application Development, Artificial Intelligence, and Software Engineering classes, to incorporating principles from Interaction Design and Operating Systems, this project has allowed me to merge and implement the concepts and theories discussed in those classes to produce a real-world product.

For a long time I have wanted to learn about designing an application for macOS, and found that it is rather straightforward and simple. The hardest part I found throughout this process was not having a structured/guided environment like I have had in the past in a classroom setting. Because of this, I found that ensuring that I had a complete understanding of the design concepts was difficult, but proved to be a valuable learning experience through all the iterations of the product and correcting mistakes I made.

I greatly enjoy learning about and using the Swift programming language and I have learned a lot about its abilities through this project. Continuing into the future, I plan on utilizing the Swift programming language more and learning about additional advanced features that it contains.

I would like to thank Professor Lubomir Stanchev, Ryan Nett, Brain Schwartz, and Lea Chandler for assistance with The Debugger's project in the Distributed Computing class for learning about and working with the software bug dataset. I would also like to thank Peter Phillips for his assistance in designing the logo and icons for the Delegation application suite.

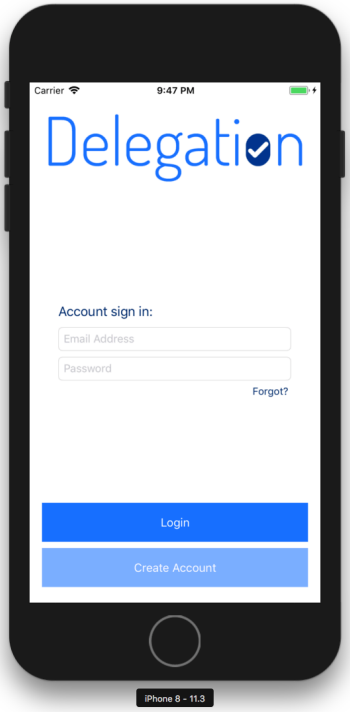
I would like to extend my sincere gratitude to Professor John Bellardo for advising me throughout this project and this past year.

Conclusion

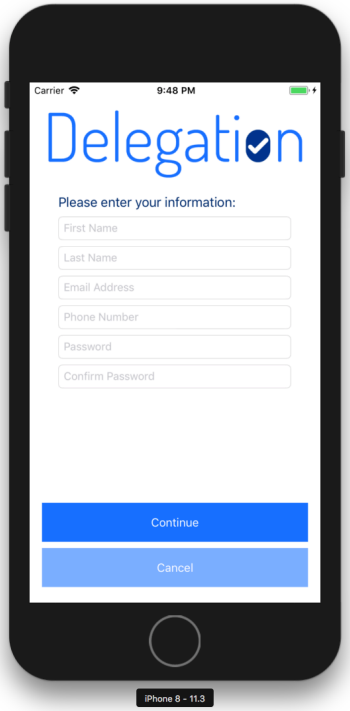
The Delegation application provides smart task distribution to users in a team environment, which can be tedious and difficult for management to assign manually. Within specific teams, members possess individual skills which should be utilized fully by assigning them tasks that they are able to accomplish. This project provides a well designed solution, consisting of a smart cross-platform application which is accessible to users on both the desktop and mobile devices, providing live data and updates for tasks within their account. Though the use of this application, users and teams will be empowered to accomplish tasks and complete their assignments.

Appendix A: iOS Application Images

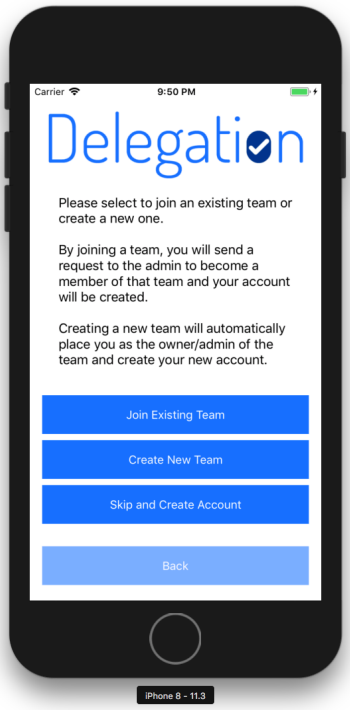
The following section outlines and describes specific views within the iOS application, along with captions for the views. The following does not include all possible views, but attempts to provide a sampling of typical scenes that might be presented to the user.



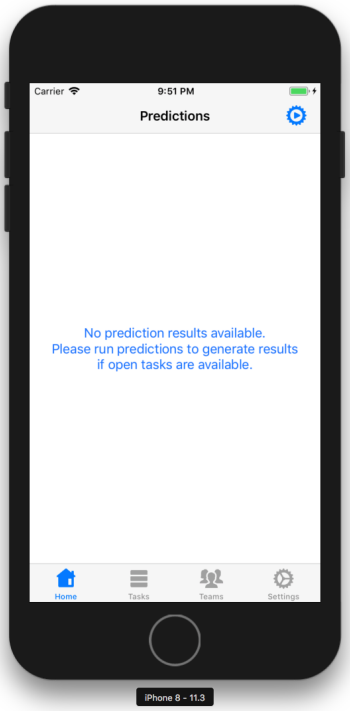
The welcome screen, ready for the user to enter their account credentials.



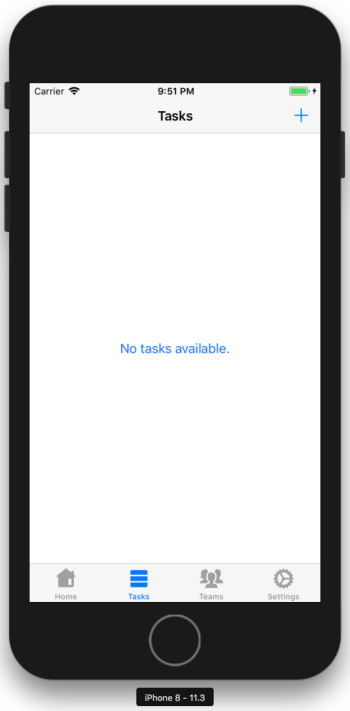
The create account screen where the user will enter their information to create a new account.



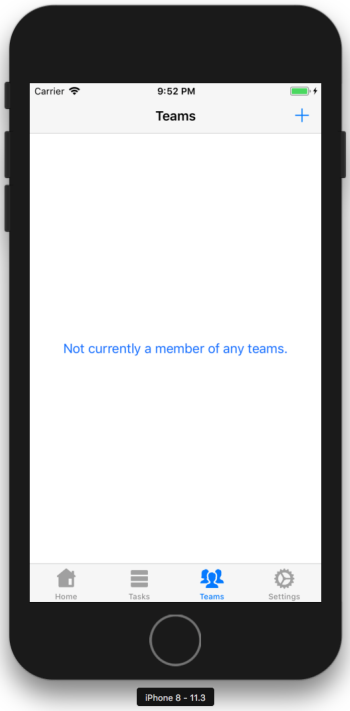
The team prompt screen which allows the user to join an existing team, create a new team, or simply create their account.



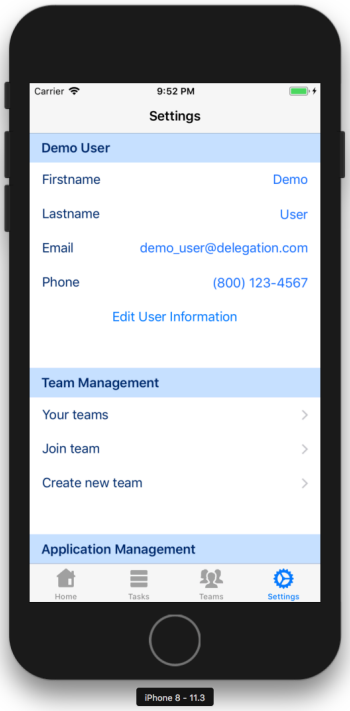
The predictions tab (home) displays predicted tasks for the user to complete when the algorithm is run.



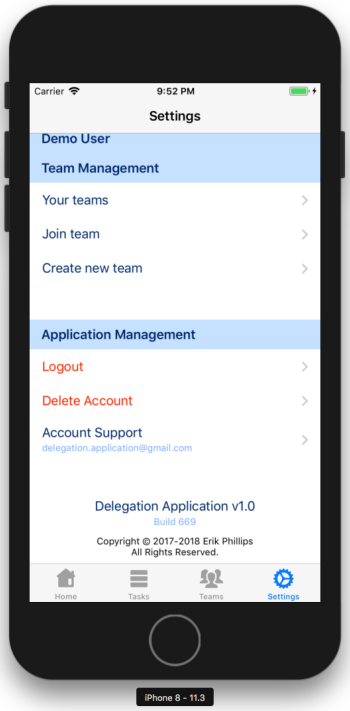
The tasks tab displays all the user's current tasks assigned to them.



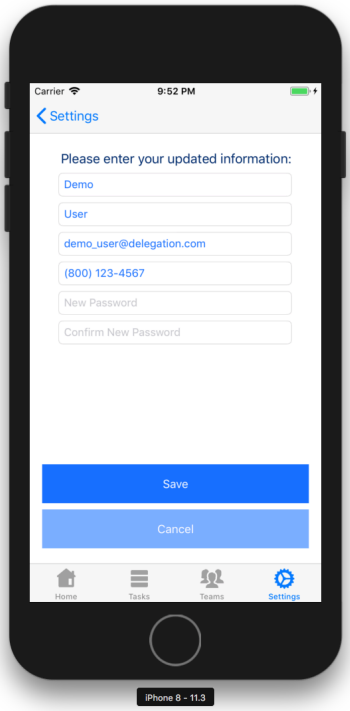
The teams tab displays all the user's current teams which they are a member of.



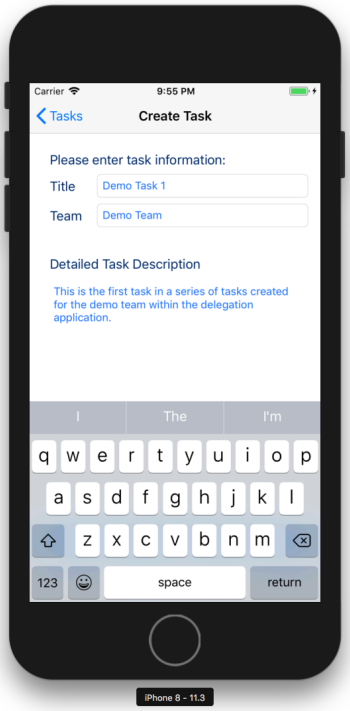
The settings tab displays all the user's information about their account as well as provide the ability to manage team membership.



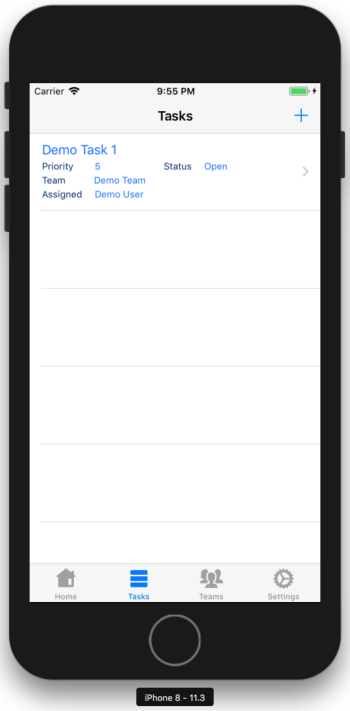
Under application management, a user can logout, delete their account, or contact support. There is also the application information here as well.



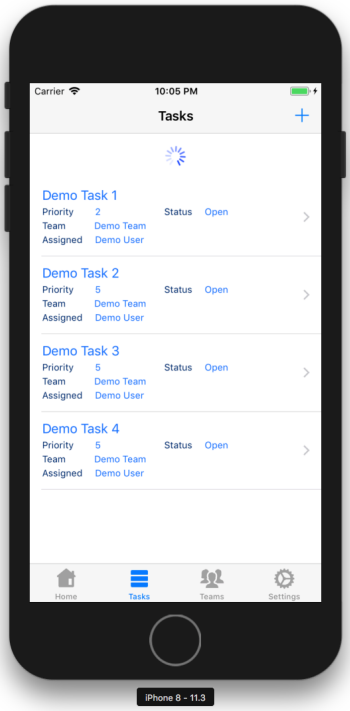
Users have the ability to update their information from within the settings tab.



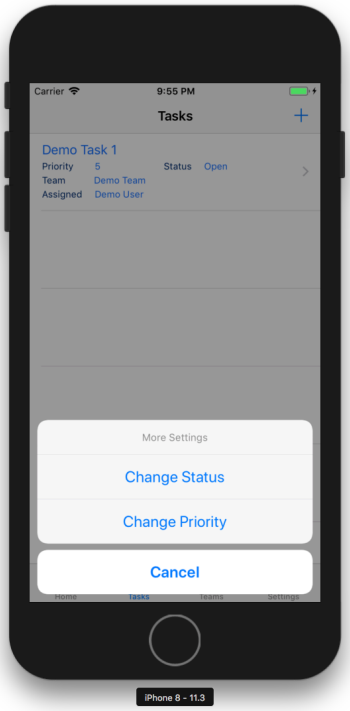
Users have the ability to create new tasks, which are then generated and displayed within the tasks tab.



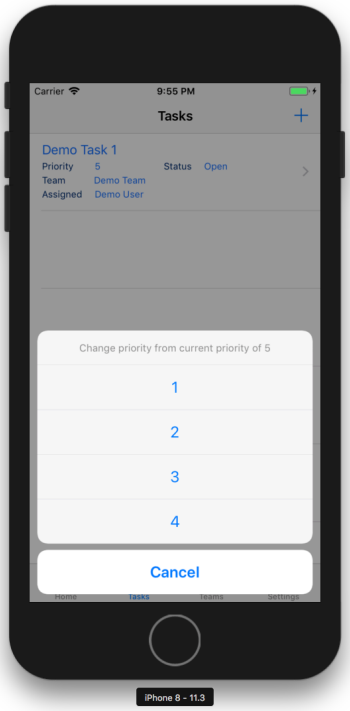
Tasks are assigned to the account owner by default and displayed within the tasks tab.



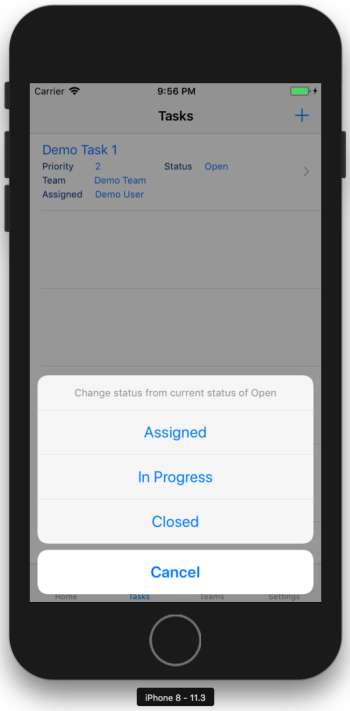
The tasks do not automatically refresh within the view, so there is a pull-down refresh system.



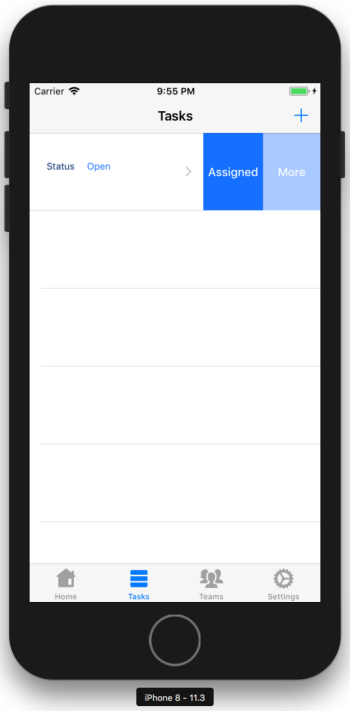
Swiping left on a tasks provides the user with a quick ability to change the status or priority of a task.



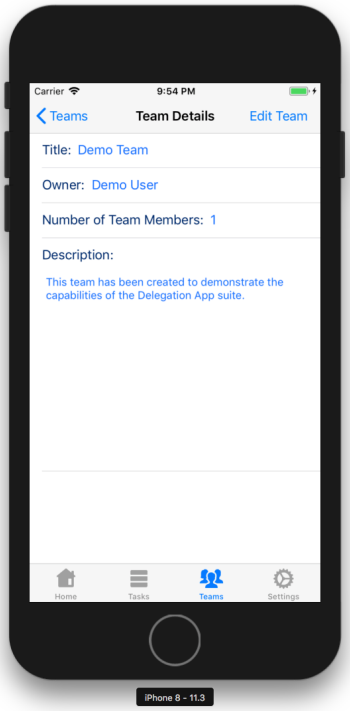
Popup menu to change the task's priority.



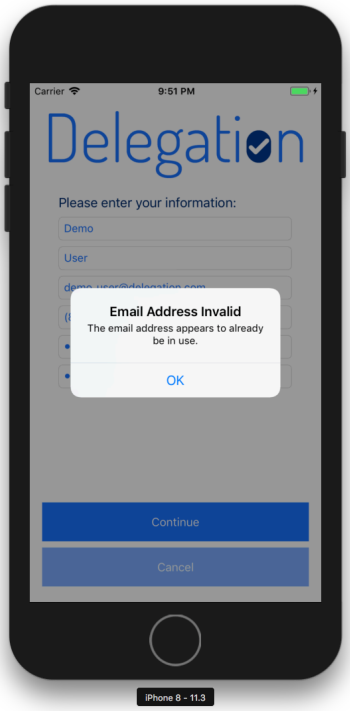
Popup menu to change the task's status.



The swipe-left feature implemented for each task.



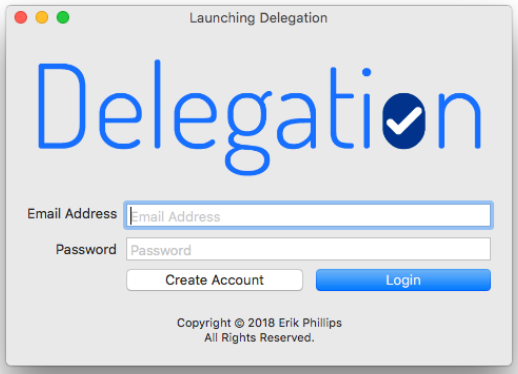
The user has the ability to view specific details about each team.



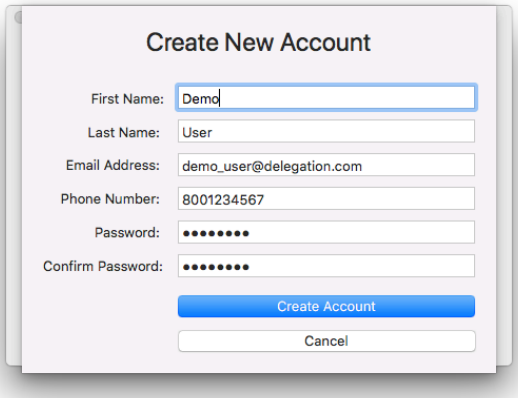
An example of the error message format used throughout the application.

Appendix B: macOS Application Images

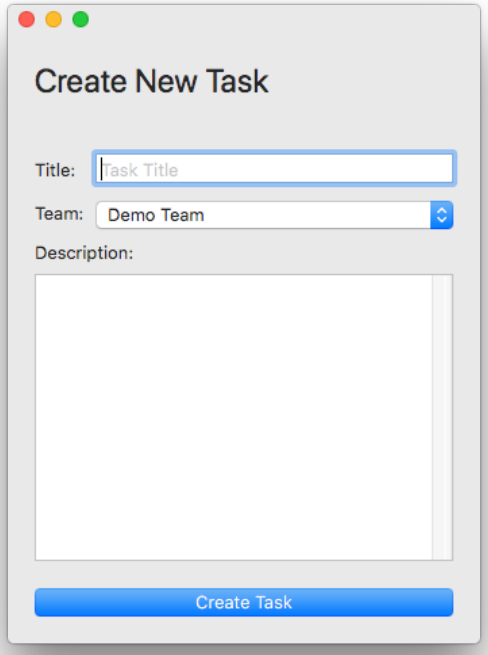
The following section outline and describes specific views within the macOS application, along with captions for the views. The following does not include all possible views, but attempts to provide a sampling of typical scenes that might be presented to the user.



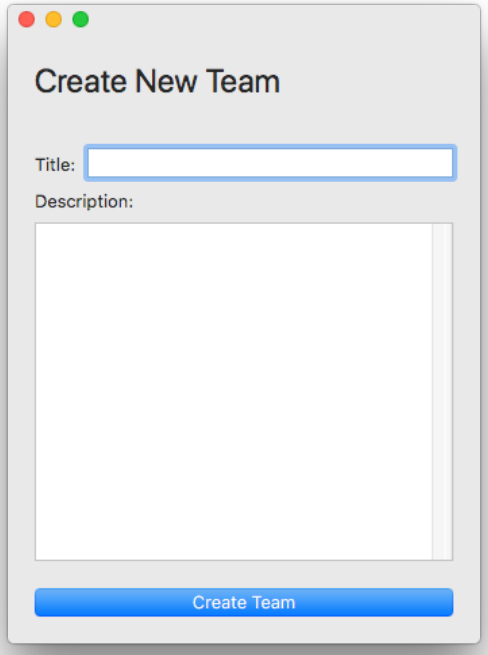
The initial window presented to the user to login to their account.



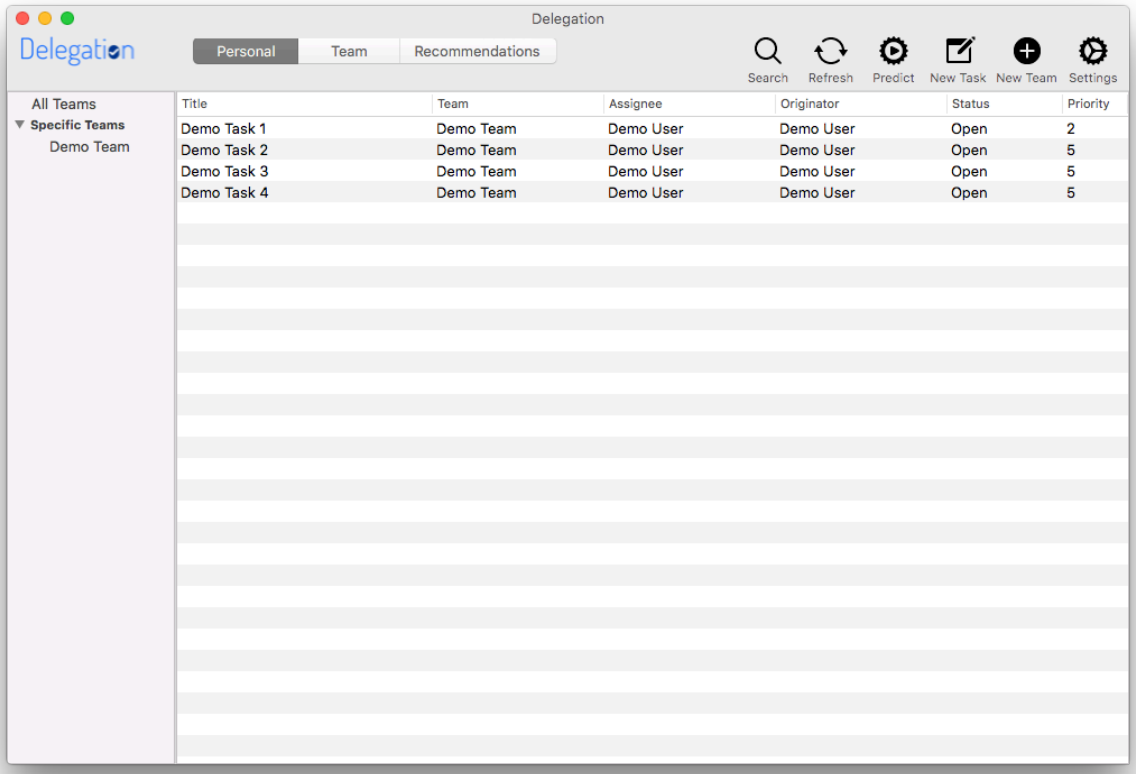
Users are able to create a new account within the initial window.



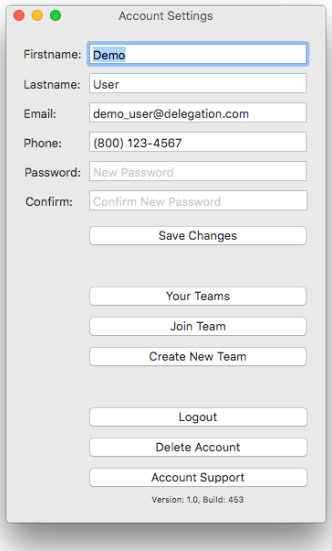
Users can create new tasks after logging in to their account.



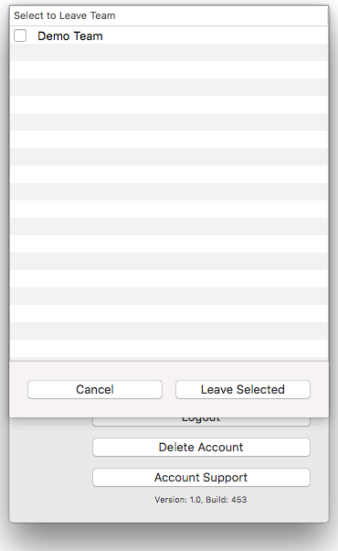
Users are able to create new teams.



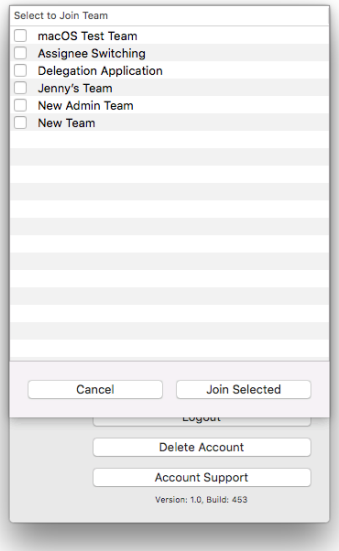
The main window provides the main functionality for the application. The tab bar at the top allows for persistent action buttons which are customizable by the user. The left sidebar provides specific filtering to view content for one specific team or all teams. The segmented control within the tab bar allows for advanced filtering of displayed tasks.



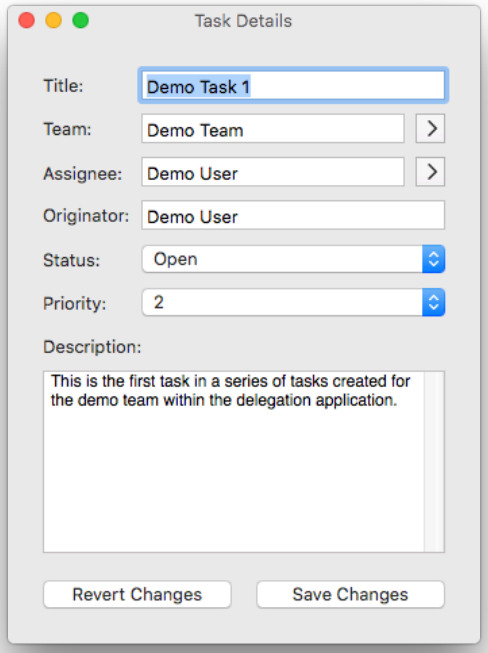
The user is able to view and edit their account information within the account settings window.



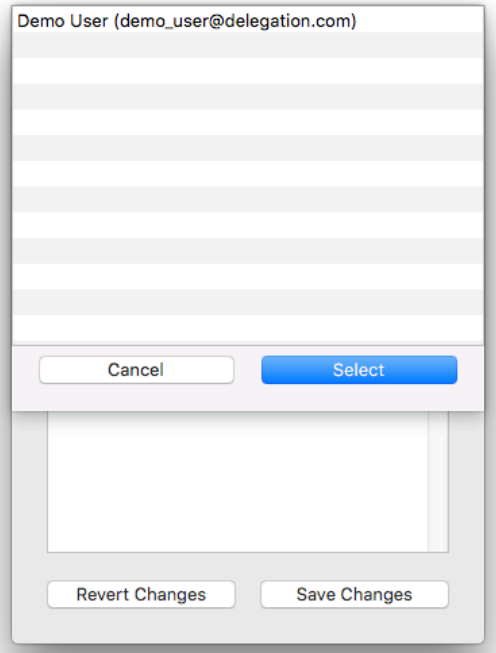
Users are provided the ability to leave teams that they are currently a member of.



Users are able to join new teams which they are not currently a member of.



When a task is double-clicked from the main window, the task's details window is displayed with all the information about the task, which is able to be edited by the user.



From the task details, users are able to change the assignee for a specific task. This is the same interface for assigning a task to a new team.