operate

Model-View-Controller Design Document
with Application Flowchart

Erik Phillips
ephill07@calpoly.edu

iOS App Development
California Polytechnic State University
San Luis Obispo, California
February 9, 2017

Root Controller

Page Name:	N/A
Controller Name:	Root Controller
Model Name:	User Database, Results Database, Settings
View Name:	N/A

Description:
This controller will be loaded on the app's launch. Used to control the various different clusters and establish a connection with the database and verify credentials. The root controller will create the schedule when the admin has activated the scheduling by pulling data from both the User Database and the Settings, then store the completed schedule in the Results Database.

Function Signatures Used from User Database:

```
int establishConnectionToUserDatabase();
// Function to establish a secure connection with Firebase
      and pull down the needed information
```

Function Signatures Used from Results Database:

```
int establishConnectionToUserDatabase();
// Function to establish a secure connection with Firebase
      and pull down the needed information
```

Function Signatures Used from Settings:

```
int initSettings();
// Function to instanciate a settings objects for the admin if none is
      currently available. Stores this in local storage.
```

Welcome/Login Page

Controller Name:	Welcome Controller
Model Name:	User Database
View Name:	Welcome View

Description:
The welcome screen will be an inviting page for new and existing users. The page will be displayed on app launch and will handle user sign-in and sign-outs. The page will prompt for the username and password of the user and whether they are an admin. The Welcome Controller will access the User Database to determine if the user's credentials are valid.

Based on the results, the user will be taken to a different page:

- New Users: Registration Page
- Existing Users:
    - Schedule Not Created: Profile Update Page
    - Schedule Complete: Results Page
- Admin: Administrator Page

Function Signatures Used from User Database:

```
Boolean isValidLogin(String username, String password);
// Function to return true if the user has correct credentials
// This will set the currentUser to the newly logged in user

User getCurrentUser();
// Function to return the current logged in user
```

Registration Page

Controller Name: Registration Controller
Model Name: User Database
View Name: Registration View

Description:
The registration screen will gather information (see below) about new users to be stored in the database. When a new user is finished entering their information, the object is pushed to the database using the user's *username* as the dictionary key.

User Information to be collected:

- Name
- Email Address [username]
- Password (2 fields, first and verification)
- Staff or On-Call Community
- Staff Administrator Status
- Available On-Call Nights
- Priority Metrics (years as an advisor, grade standing, in/out of state)
    - Algorithm will determine additional priority ranking based on past ranking (if any) and a random factor given to each RA.

Function Signatures Used from User Database:

```
void addNewUser(User newUser);
// function to add a new user to the database
// this sets the current user to the newly added user

User getCurrentUser();
// Function to return the currently logged in user
```

Profile Update Page

Controller Name: Profile Controller
Model Name: User Database
View Name: Profile View

Description:
Users will be able to change information about themselves through the Profile Update Page. This should primarily be used to update on-call availability and priority metrics.

Users will be able to modify the following information about themselves if changes need to be made throughout the year:

- Password (2 fields, first and verification)
- Staff or On-Call Community
- Staff Administrator Status
- Available On-Call Nights
- Priority Metrics (years as an advisor, grade standing, in/out of state)
    - Algorithm will determine additional priority ranking based on past ranking (if any) and a random factor given to each RA.

Function Signatures Used from User Database:

```
User getCurrentUser();
// Function to return the current user

void replaceUser(User user, User newUser);
// Function to modify the information about the given user
// this function is used when the user needs to be replaced

// The following functions will assist with updating the user information:
void updateUserPassword(User user, String newPassword);
void updateUserStaff(User user, Staff newStaff);
void updateAdminStatus(User user, Boolean newIsAdmin);
void updateOnCallNights(User user, Nights[] newNights);
void updatePriority(User user, Priority newPriority);
```

Erik Phillips
MVC Document
CPE-436

Admin Settings Page

Controller Name: Settings Controller
Model Name: User Database, Results Database, Settings (Stored Locally)
View Name: Settings View

Description:
The administrator scene will display different settings (listed below) for the scheduling app. An option to edit user profiles, assign priorities, and initiate a schedule. Admins will be able to edit the resulting schedule before pushing it to the database.

Admin Accessible Settings:

- Number of on-call nights
- Number of nights per RA per week
- Specific advisors on staff
- Priority Factors
  - Current grade level at Cal Poly
  - Number of years as an advisor
  - In-state or out-of-state
  - Random number generator
  - Past priority level

Erik Phillips
MVC Document
CPE-436

Function Signatures Used from Results Database:

```
void createScheduleForStaff(Staff staff);
// Function to create the schedule and push to the results database

Schedule[] getAllSchedulesForStaff(Staff staff);
// Function to create the schedule for the specified staff
```

Function Signatures Used from User Database:

```
Users[] getAllUsersByStaff(Staff staff);
// Function to return all the users for specified staff

void setAllUsersForStaff(Staff staff, User[] users);
// Function to set the users for the specified staff
```

Function Signatures Used from Settings:

```
Nights[] getOnCallNightsByStaff(Staff staff);
// Function to return the nights the currently scheduling for

void setOnCallNightsForStaff(Staff staff, Night[] nights);
// Function to set the on-call nights
```

Scheduling Results Page

Controller Name: Results Controller
Model Name: User Database, Results Database
View Name: Results View

Description:
The results scene will display individualized schedule results for the logged-in user. The results page will not be available until the schedule data placed in the Results Database (when the admin has completed the scheduling) is available and ready. The results page will also show on-call partners and the entire staff's on-call rotation.

The Results page will show user their on-call nights in a tabular form as well as allow users to download their schedule into Apple Calendar Format (.ics) to use with their preferred calendar application.

Function Signatures Used from Results Database:

```
Schedule[] getSchedulesForStaff(Staff staff);
// Function to return all the schedules created for the specified staff

Schedule[] getSchedulesForUser(User user);
// Function to return all the schedules created for the specified user
```

Function Signatures Used from User Database:

```
User getCurrentUser();
// Function to return the current user
```

Erik Phillips
MVC Document
CPE-436

User Database

The User Database will hold information needed to complete the schedule, as well as the login information for user authentication. The base class is *User* with various supporting class objects utilized across the system. The User Database will communicate with Firebase.

The classes are as follows:

```
public class User {
      private String   name              // user full name (first and last)
      private Integer  staffID           // user unique indentifier
      private String   emailAddress      // user email address
      private String   password          // user password
      private Staff    staff             // current staff or community
      private Priority priority          // priority level
      private Boolean  administrator     // admin access
      private Nights[] nights            // availible on-call nights
}

public class Staff {
      private String   staffName         // Name of the staff or community
      private Integer  staffID           // staff id number
      private Integer  staffSize         // number of staff members
      private Integer  maxStaffSize      // maximum number of staff members
      private User[]   staffMembers      // array of staff ids
}

private class Priority {
      private Integer  yearsAtCalPoly    // grade level at cal poly
      private Integer  yearsAsAdvisor    // number of years as an advisor
      private Boolean  inStateAdvisor    // true if advisor live in-state
      private Integer  randomNumber      // random number added to priority
      private Integer  pastPriority      // past priority ranking (if any)

      public  Integer  priority          // resulting priority
}

public class Nights {
      private Integer  dayOfWeek         // day of week (sunday = 0)
      private String   nameOfDay         // name of the week day
      private Integer  numberOfAdvisorsNeeded      // num advisors needed
      private Integer  currentNumberOfAdvisors     // current num advisor
      private Integer  advisorPreference           // advisor ranking
}
```

Settings Model (Local Storage)

All settings will be stored locally (i.e. not pushed up to the database) and will be actively managed by the administrator. Default values will be assigned if not manually entered on the Settings Page.

For the *Priority Factors* class, different values will be stored based on the admin's preferred settings. Default values are all set to false except for the random numbers variable, which is set to true.

The Settings Class is as follows:

```
public class Settings {
      private Staff    staff      // stores mutable staff object
      private User[]   advisors   // holds the staff members
      private PFactors priorities // priority factors class
      private Nights   nights     // specifies the nights on-call
}

private class PFactors {
      private Boolean  useGrade         // use the current grade level
      private Boolean  useAdvisorYears  // use advisor year number
      private Boolean  useHomeState     // use in-state/out-of-state
      private Boolean  usePastPriority  // use pre-exisiting priority
      private Boolean  useRandomNumbers // use random numbers
}
```

operate

Results Database

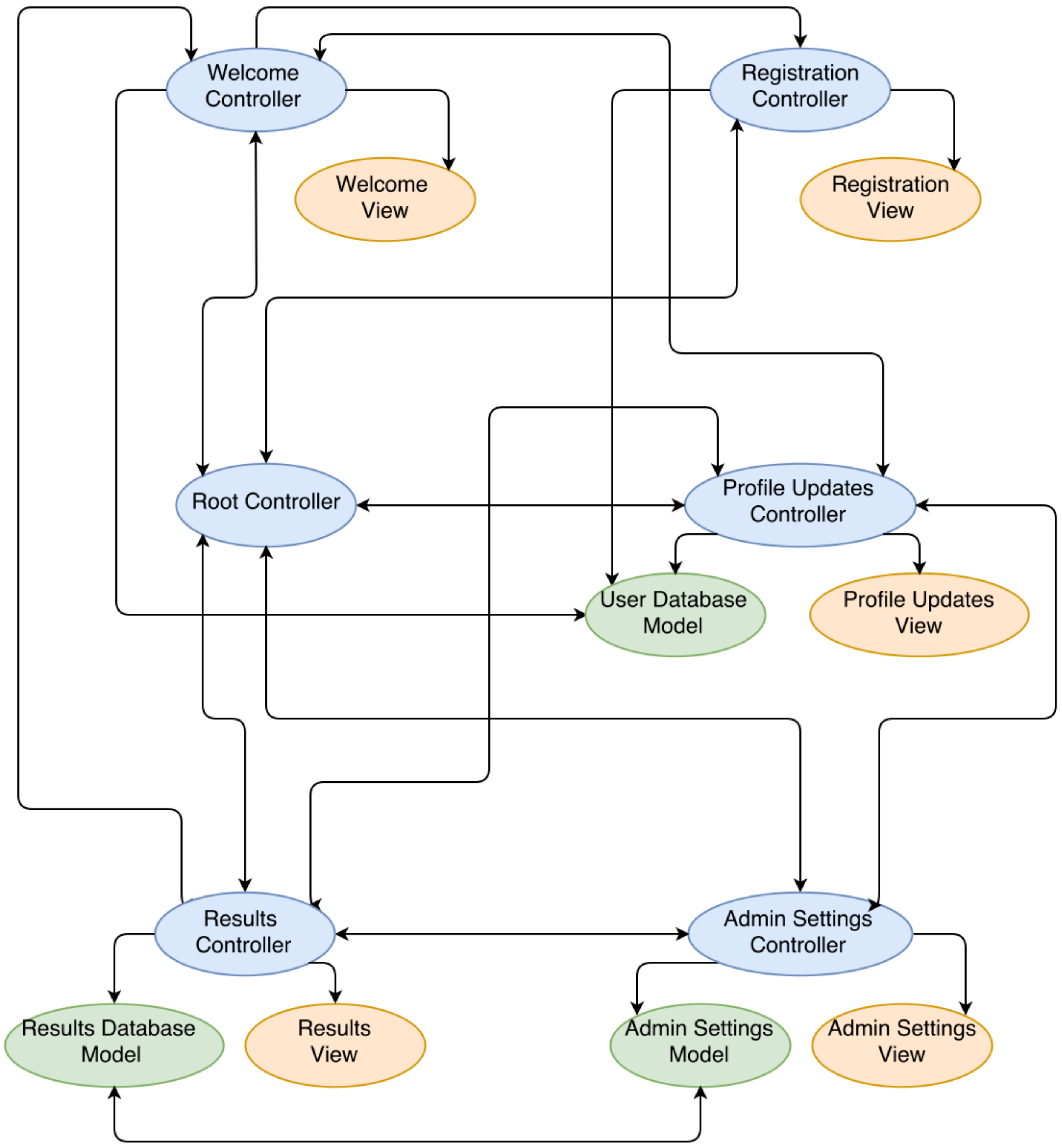
The results will be stored within a Schedule class object consisting of users and their nights on-call. Multiple schedules will be able to be stored per staff for the administrator to view and select the final. The Results Database will communicate with Firebase.

Schedule class as follows:

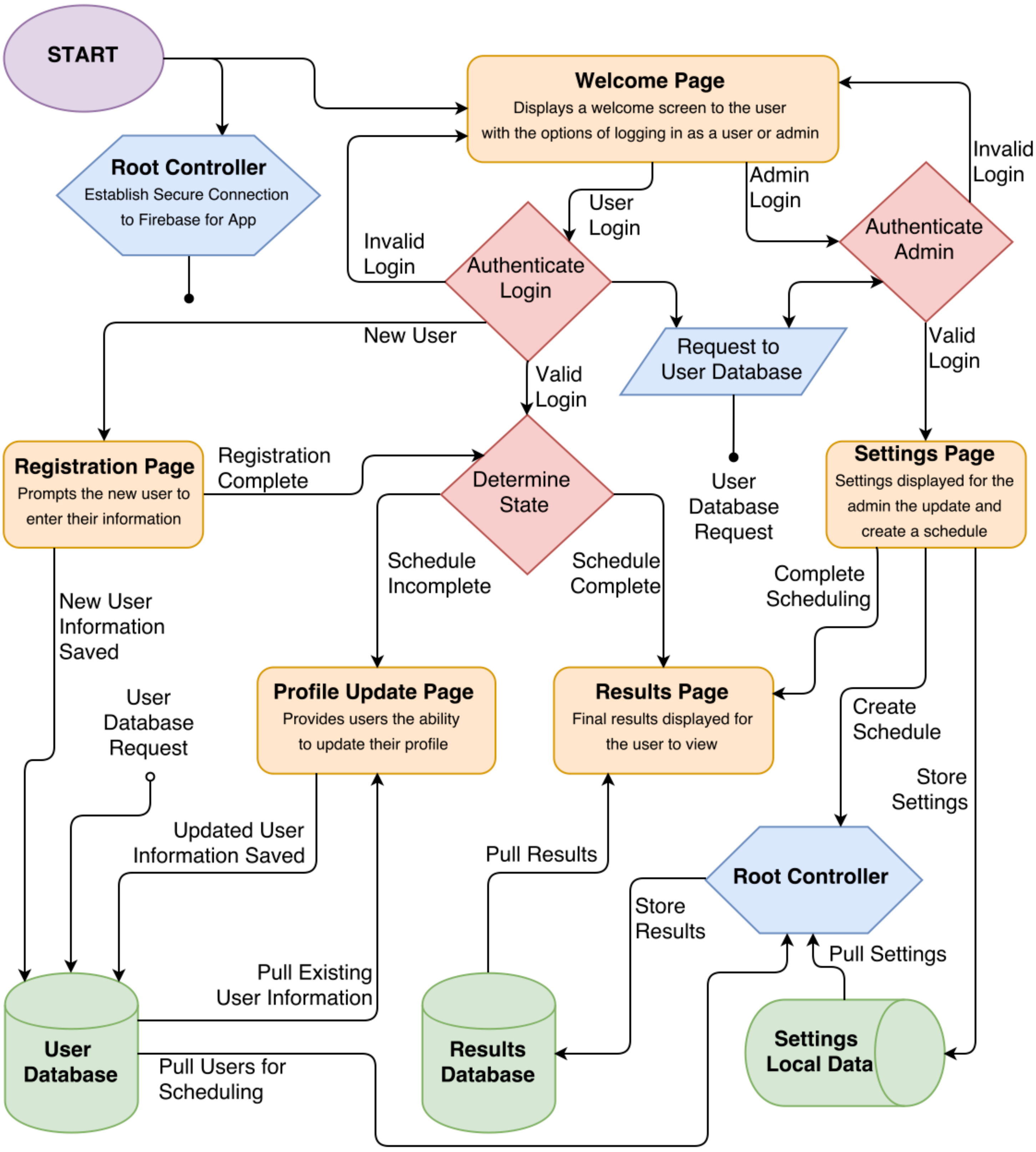
```
public class Schedule {
      private Staff     staff           // staff class object
      private Nights[]  onCallNights    // array of the nights represented
      private Users[][] advisorsOnCall  // double array of users maped to
                                           the onCallNights listed above
}

public class Staff {
      private String   staffName        // Name of the staff or community
      private Integer  staffID          // staff id number
      private Integer  staffSize        // number of staff members
      private Integer  maxStaffSize     // maximum number of staff members
      private User[]   staffMembers     // array of staff ids
}

public class Nights {
      private Integer  dayOfWeek        // day of week (sunday = 0)
      private String   nameOfDay        // name of the week day
      private Integer  numberOfAdvisorsNeeded      // num advisors needed
      private Integer  currentNumberOfAdvisors     // current num advisor
      private Integer  advisorPreference           // advisor ranking
}
```

Erik Phillips
Model-View-Controller Document
February 7th, 2017

operate



Erik Phillips
Application Flowchart
February 7th, 2017


START
Root Controller
Establish Secure Connection to Firebase for App
Welcome Page
Displays a welcome screen to the user with the options of logging in as a user or admin
User Login
Admin Login
Invalid Login
Authenticate Login
Authenticate Admin
Invalid Login
New User
Valid Login
Request to User Database
Valid Login
Registration Page
Prompts the new user to enter their information
Registration Complete
Determine State
User Database Request
Settings Page
Settings displayed for the admin the update and create a schedule
Schedule Incomplete
Schedule Complete
Complete Scheduling
New User Information Saved
User Database Request
Profile Update Page
Provides users the ability to update their profile
Results Page
Final results displayed for the user to view
Create Schedule
Store Settings
Updated User Information Saved
Pull Results
Root Controller
Store Results
Pull Settings
Pull Existing User Information
User Database
Pull Users for Scheduling
Results Database
Settings Local Data