SENG 330 Assignment 4 Design Document

Dana Wiltsie & Keanu Enns

Nov. 30, 2018

Notes to Marker

The application can be built, tested, and run using the commands gradle build, gradle test, and gradle run respectively. Equivalently, the commands can be run through the gradle plugin in Eclipse. The final option for running the application is to right click the class IOTApp.java under package ca.uvic.seng330.assn3 and select “Run as” -> “Java application”.

Two automatic client accounts will be registered to the system upon start up. The user names are “basic” (the basic user account) and “admin” (the admin user account). The password for both accounts is “password”. In addition, the marker may feel free to create other users from the login screen or from the admin interface.

Camera objects are persisted in the application (information and state stored even after shutting down the app) despite this not being a requirement for the system; this is only true for cameras and not other devices. Also, along with the user accounts, 3 devices of each type will be created upon start up of the application to avoid tedious creation of devices when testing. This is the cause of the logged messages with user: “NO CURRENT USER.”

The acceptance tests to be marked are as follows:

- Scenario Z: 1,2
- Scenario A: 5,6 (Note for 6: The admin must select a user name and the notification level they want to assign that user. If the selection is “Custom”, the admin must enter a device name & type into the fields below before selecting “Set Notifications.”)
- Scenario B: 1,4,5,7 (Note for 1: ActivityLogs are persisted and the capacity of logs in the display is capped at 100. Note for 7: We took Client Control to mean the view that both basic users and admins alike see and have access to.)
- Scenario C: 1,2,6 (Note for 6: This, along with the similar acceptance test for the Thermostat, has buttons available in the UI for direct testing. The two buttons correspond to the lightbulb’s two tests as well. More specifically, these are tests C:6, D:6, E:3,4.)
- Scenario D: 2-6
- Scenario E: 3,4

Finally, it is worth noting, due to the terms used in the acceptance tests, that there is no Hub class; it is replaced by the many controllers that have different responsibilities.

The Design

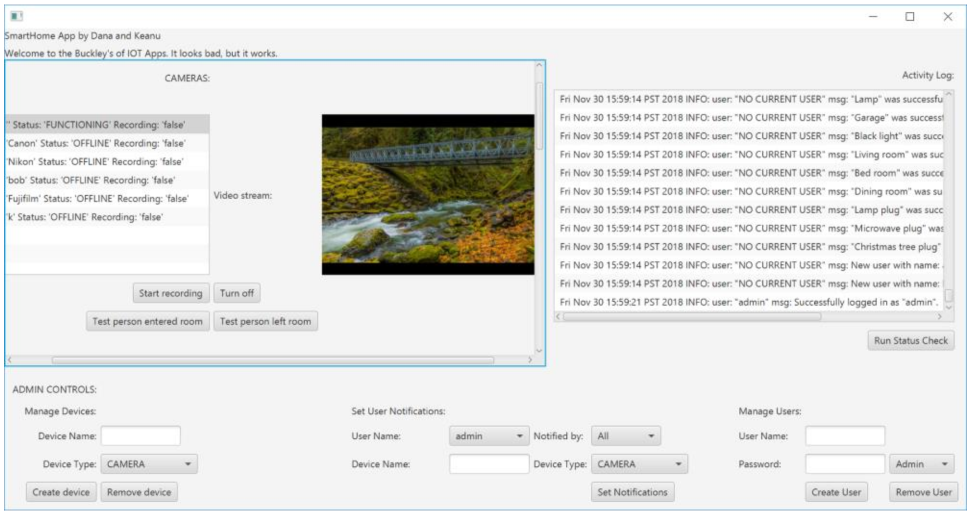
The overall design of the system is very similar to assignment 3 only with additional functionality in the form of new methods, class interactions, and the new ActivityLog model, view, and controller classes. The broad design is centered around the Model View Controller (MVC) pattern where the UI and input from the user is sent through the view classes to their corresponding controller classes. Once the controller classes are called upon by the views, they coordinate with other controllers to retrieve and store information within the model classes. These model classes serve as our real-world objects (e.g. devices and users) with basic attributes and occasionally perform their own activities but are primarily meant to be used by the controllers.

As mentioned in the design document for assignment 3, the design of the system focuses on end-to-end functionality for each element of the system. A model, controller, and view were created together for these elements (e.g. Activity Logging) and designed to interact with the models, controllers, and views of other elements rather than creating the views and models for every element and then later figuring out how to bridge the gap with controllers.

To give an idea of how the system communicates, we walk through a scenario in which an admin wishes to create a device from the admin controls. When the admin selects the “Device Name” field under “Manage Devices” on the left side of the admin controls, enters a name and device type for a new device, and clicks “Create device,” an event is triggered in the AdminView class. The view then calls the method createNewDevice() in the AdminController class and passes in the given name and device type. The controller then uses the device type to decide what device controller to call on. If, for example, the device type is “CAMERA” then the admin controller calls createCamera() on the camera controller which then checks to see if that camera name is already registered. If the camera controller has a camera by that name, then it throws a RegistrationException and the admin controller catches it, sets a warning text in its view, and logs the error. If it doesn’t, then it creates a new camera object with the given name, puts it into its maps, updates the cameras in the camera view, saves the information to a file, and logs the successful camera creation.

Logging

We decided to remove the logger classes of SLF4J in order to more effectively control logging in the application and display messages from the admin interface as seen in the Figure 1.


SmartHome App by Dana and Keanu
Welcome to the Buckley's of IOT Apps. It looks bad, but it works.

CAMERAS:

```
" Status: 'FUNCTIONING' Recording: 'false'
'Canon' Status: 'OFFLINE' Recording: 'false'
'Nikon' Status: 'OFFLINE' Recording: 'false'
'bob' Status: 'OFFLINE' Recording: 'false'
'Fujifilm' Status: 'OFFLINE' Recording: 'false'
'k' Status: 'OFFLINE' Recording: 'false'
```

Video stream:

Activity Log:

```
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Lamp" was successfu
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Garage" was successf
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Black light" was succ
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Living room" was suc
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Bed room" was succe
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Dining room" was su
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Lamp plug" was succ
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Microwave plug" was
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: "Christmas tree plug"
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: New user with name:
Fri Nov 30 15:59:14 PST 2018 INFO: user: "NO CURRENT USER" msg: New user with name:
Fri Nov 30 15:59:21 PST 2018 INFO: user: "admin" msg: Successfully logged in as "admin".
```

ADMIN CONTROLS:

Manage Devices:
Device Name:
Device Type: CAMERA

Set User Notifications:
User Name: admin
Notified by: All
Device Name:
Device Type: CAMERA

Manage Users:
User Name:
Password:
Admin

Figure 1. The admin interface of the application. The Activity Log is displayed in the upper right, the basic user view in the upper left, and the admin controls along the bottom.

Many calls to the SLF4J logger class were made in assignment 3, thus to prevent the need to completely replace these calls, we gave the ActivityLogController class info(), warn(), and error() methods and simply replaced the object type of each class's logger attribute from Logger to ActivityLogController. The ActivityLogController writes to a file named ActivityLogs.json and reads from them upon construction of the controller class. Finally, the ActivityLogController holds a reference to the device controllers (which implement the Runnable interface) and runs a thread for each device controller within its statusCheck() method to perform a status check for each device in the system.

New Functionality

Figure 2 illustrates most, but not all, of the changes discussed here. To meet the requirements of the system, the admin controller was given methods for creating and removing users, as well as setting notifications for basic users (admins always see all devices). Within the Device controllers, each device name maps to a list of names of users that want to be notified by that device's activities. Setting notifications for a user consists of adding or removing the user's name from the lists that the devices hold (via the unlink/linkUserToDevice() methods). When a device controller updates the devices displayed in its corresponding device view, it looks for devices that have the current user in their user lists (hence the reference to the CurrentUser class held by the CameraController class in Figure 2).

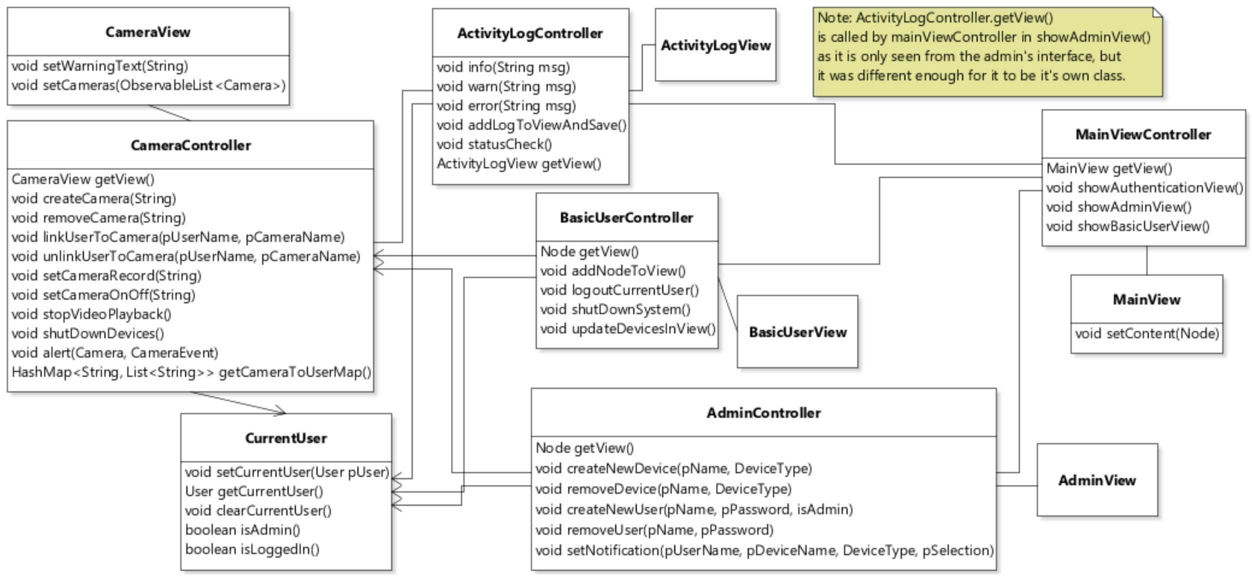

CameraView
void setWarningText(String)
void setCameras(ObservableList<Camera>)
CameraController
CameraView getView()
void createCamera(String)
void removeCamera(String)
void linkUserToCamera(pUserName, pCameraName)
void unlinkUserToCamera(pUserName, pCameraName)
void setCameraRecord(String)
void setCameraOnOff(String)
void stopVideoPlayback()
void shutDownDevices()
void alert(Camera, CameraEvent)
HashMap<String, List<String>> getCameraToUserMap()
CurrentUser
void setCurrentUser(User pUser)
User getCurrentUser()
void clearCurrentUser()
boolean isAdmin()
boolean isLoggedIn()
ActivityLogController
void info(String msg)
void warn(String msg)
void error(String msg)
void addLogToViewAndSave()
void statusCheck()
ActivityLogView getView()
ActivityLogView
Note: ActivityLogController.getView()
is called by mainViewController in showAdminView()
as it is only seen from the admin's interface, but
it was different enough for it to be it's own class.
MainViewController
MainView getView()
void showAuthenticationView()
void showAdminView()
void showBasicUserView()
MainView
void setContent(Node)
BasicUserController
Node getView()
void addNodeToView()
void logoutCurrentUser()
void shutDownSystem()
void updateDevicesInView()
BasicUserView
AdminController
Node getView()
void createNewDevice(pName, DeviceType)
void removeDevice(pName, DeviceType)
void createNewUser(pName, pPassword, isAdmin)
void removeUser(pName, pPassword)
void setNotification(pUserName, pDeviceName, DeviceType, pSelection)
AdminView

Figure 2. An illustration of new methods and the ActivityLog classes (the model for which is not shown due to its simplicity). Just as in the diagrams for assignment 3, it can be imagined that all the references to and from CameraController are similar in the other device controllers.

Changes have also been made to the BasicUserController class from assignment 3. It now contains a reference to each device controller because it is responsible for facilitating the shutDownSystem() method which safely turns off all of the devices by calling shutDownDevices() on each device controller. It also uses its references to the device controllers to update the devices in its view (which consists of a list of device views) by calling update[Devices]InView() on each controller.

Many more aspects and small changes could be discussed, but we leave the remaining design changes and details up to the marker to discover and evaluate.