

Report #2, Part #3: Home Security Automation

Group 2

14:332:452:01:00614 Software Engineering

Harmit Badyal

Nikunj Jhaveri

Abhishek Kondila

Kaavya Krishna-Kumar

Kaushal Parikh

Miraj Patel

Nirav Patel

Andrew Russomagno

Sagar Shah

Ashwin Suresh

March 17, 2019

Website: <https://sites.google.com/view/ruhome2019/home>

Individual Contributions Breakdown

Team Member Name										
	Sagar	Ashwin	Andrew	Abhishek	Nikunj	Miraj	Kaavya	Harmit	Kaushal	Nirav
Interaction Diagrams			50%				50%			
Class Diagram					100%					
Data Types and Operation Signatures					50%				50%	
Traceability Matrix						50%				50%
Architectural Styles						100%				
Identifying subsystems							100%			
Mapping subsystems to hardware								100%		
Persistent Data Storage		50%		50%						
Network Protocol						50%				50%
Global Control Flow	50%	50%								
Hardware Requirements	50%							50%		
Algorithms									100%	

Data Structures					50%	50%				
User Interface Design and Implementation					33%				33%	33%
Design of Tests		50%	50%							
Merging Contributions									100%	
Project Coordination and Progress Report	100%									
Plan of Work							100%			
Breakdown of Responsibilities				50%				50%		
References			100%							

Contents

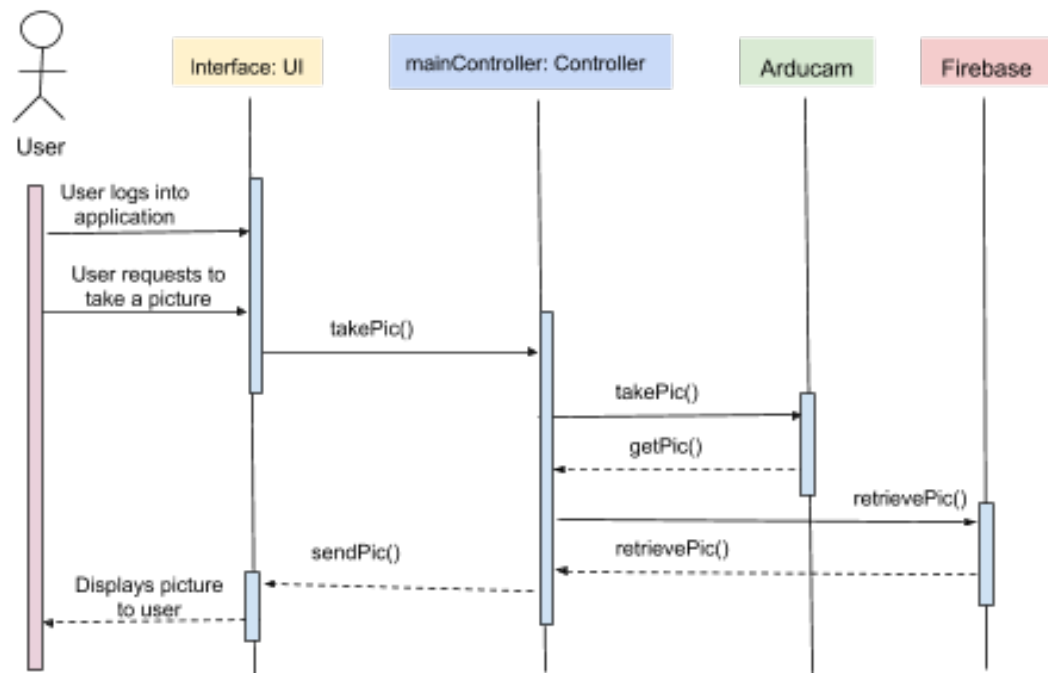
1	Interaction Diagrams	6
2	Class Diagram and Interface Specification	12
2.1	Class Diagram	12
2.2	Data types and Operation Signatures	13
2.2.1	LoginActivity	13
2.2.2	FirstTimeUserSetup	14
2.2.3	SystemSetup	15
2.2.4	MainControlActivity	16
2.2.5	AccessCamera	17
2.2.6	Pictures	18
2.2.7	SelectedPicture	19
2.2.8	GeneralUserSettings	20
2.2.9	NewUserSetup	22
2.2.10	AdminSettings	23
2.2.11	RegisteredFaces	24
2.2.12	SelectedRegisteredFaces	25
2.2.13	RegisterNewFace	26
2.2.14	Photo	27
2.2.15	User	27
2.3	Traceability Matrix	29
3	System Architecture and System Design	34
3.1	Architectural Styles	34
3.2	Identifying Subsystems	36
3.3	Mapping Subsystems to Hardware	37
3.4	Persistent Data Storage	37
3.5	Network Protocol	38
3.6	Global Control Flow	39
3.7	Hardware Requirements	39
4	Algorithms and Data Structures	40
4.1	Algorithms	40
4.2	Data Structures	41
5	User Interface Design and Implementation	42
6	Design of Tests	42

7	Project Management and Plan of Work	45
7.1	Merging the Contributions from Individual Team Members	45
7.2	Project Coordination and Progress Report	45
7.3	Plan of Work	46
7.4	Breakdown of Responsibilities	46
8	References	48

1 Interaction Diagrams

UC-1: Take Picture

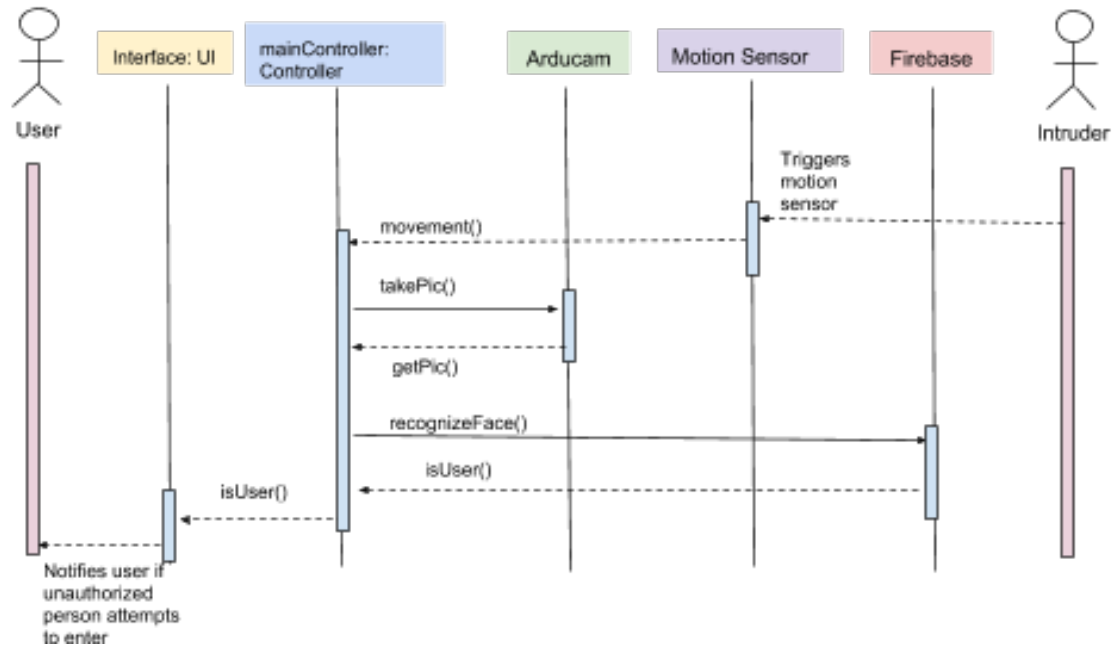
Use Case 1: Take Picture



This System Diagram will use the Single Responsibility Principle (SRP) as Firebase (our database connection) should only change if the user requests to take a picture. Further, we took into account the Liskov Substitution Principle (LSP) as the user interface for taking a picture (Arducam) should be able to execute taking and saving the picture. Lastly, we considered the expert doer principle as each part of our controller should know who it is communicating with in order to properly execute the request.

UC-2: Facial Recognition

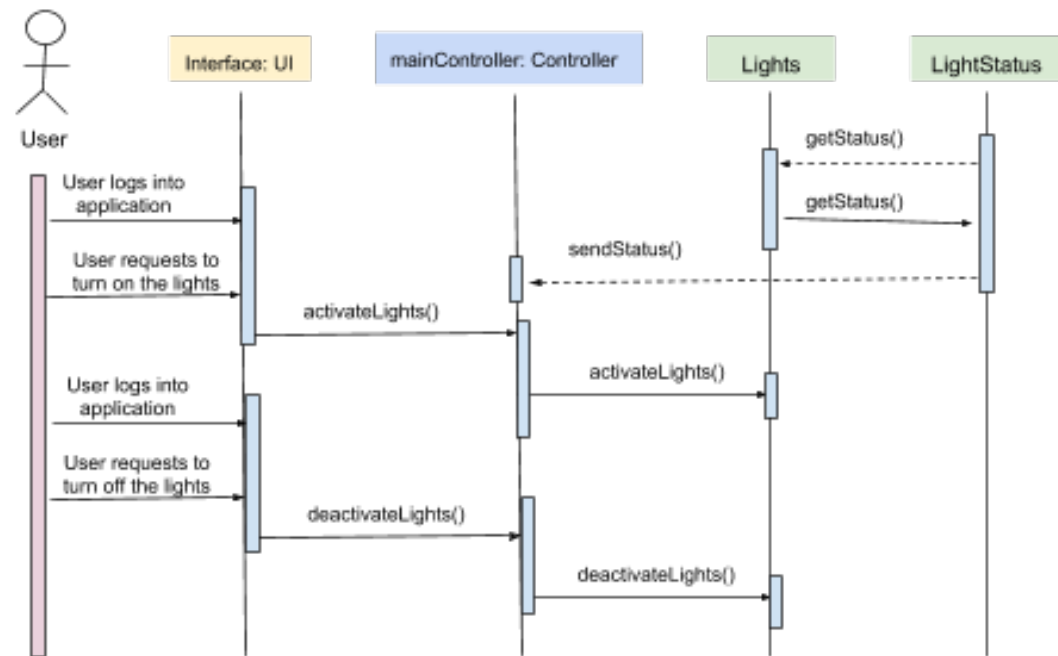
Use Case 2: Facial Recognition



For this system, we considered This System Diagram will use the Single Responsibility Principle (SRP) as Firebase (our database connection) should only change if the motion sensor is triggered to take a picture. We also considered the Low Coupling Principle as the controller should not take on too many responsibilities communicating as the sequence of actions between the Ardu Cam, Motion Sensor and Database are highly dependant on the signals from the controller. Further, we took into account the Liskov Substitution Principle (LSP) as the user interface for taking a picture (Arducam) should be able to execute taking and saving the picture. Lastly, we considered the Expert Doer principle as each part of our system should know who it is communicating with in order to properly execute the desired tasks.

UC-3: Light Control

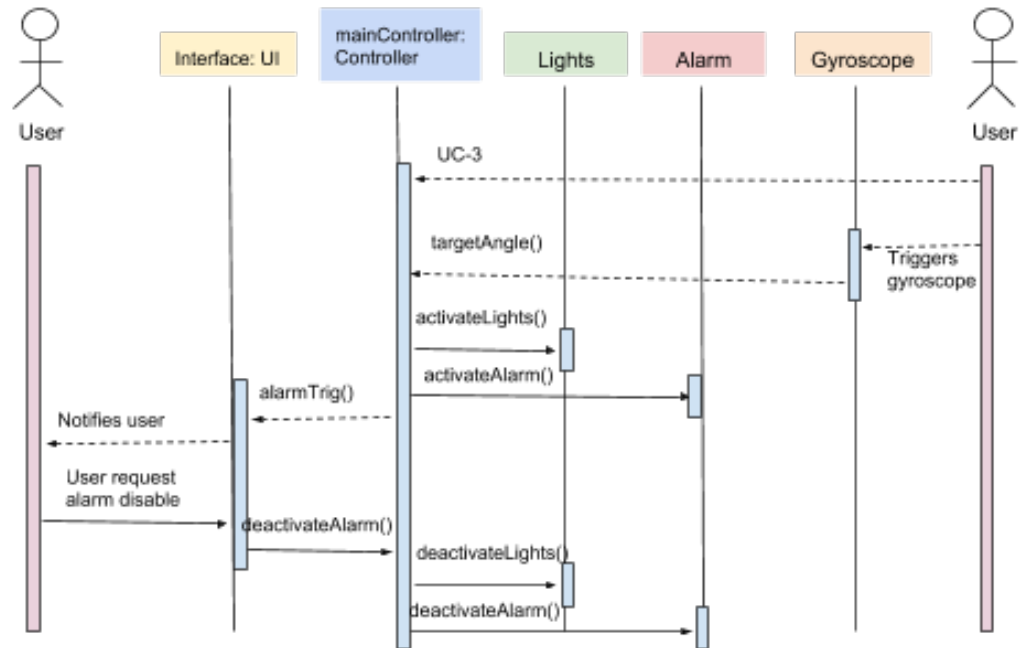
Use Case 3: Light Control



For this diagram, we considered the Low Coupling Principle as the controller should not take on too many responsibilities communicating since each of the following parts of the system is dependant on the controllers communication. Further we considered the LSP and Open Closed Principle (OCP) as each element of the system (ie. lights) be allowed to be enabled and disabled but the methods to activate should not be altered in the process. Lastly, we took into account the Interface Segregation Principle as the change of lights should only be dependant on the user request and controler interaction and nothing more.

UC-4: Gyroscope is Triggered

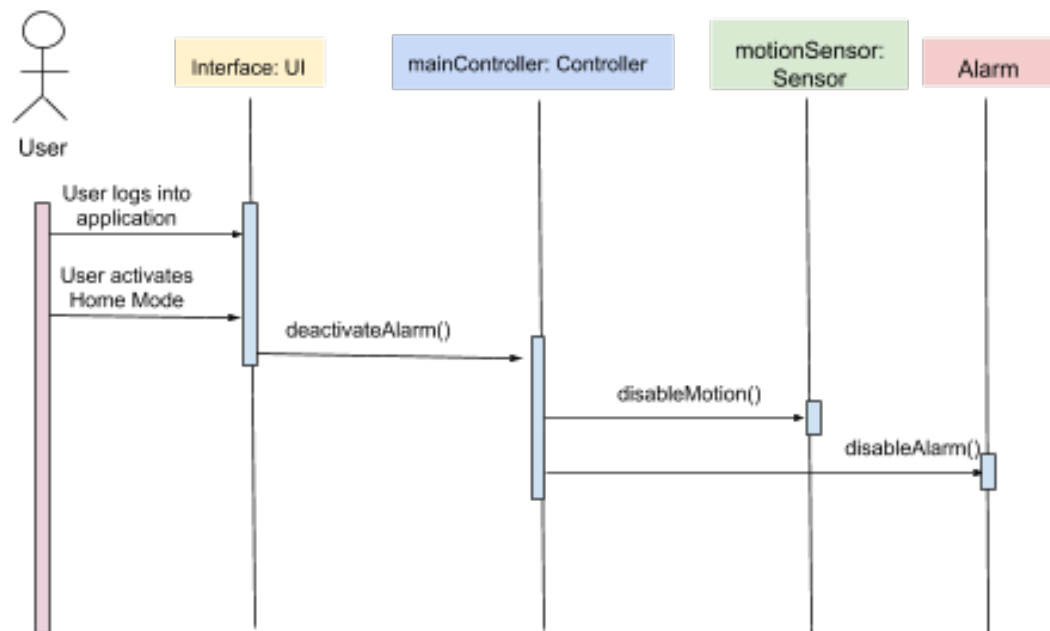
Use Case 4: Gyroscope is Triggered



For this diagram we took into account the Low Coupling Principle as the controller should not take on too many responsibilities communicating as the sequence of actions between the lights, alarm, and Motion Sensor are highly dependant on the signals from the controller. Further, we took into account the Interface Segregation Principle as the change of any part of the signal should only be dependant on the the activation from the gyroscope. Lastly, we took into account the Liskov Substitution Principle (LSP) as the user interface for disarming the alarm should be able to execute request properly.

UC-7: System Control Features

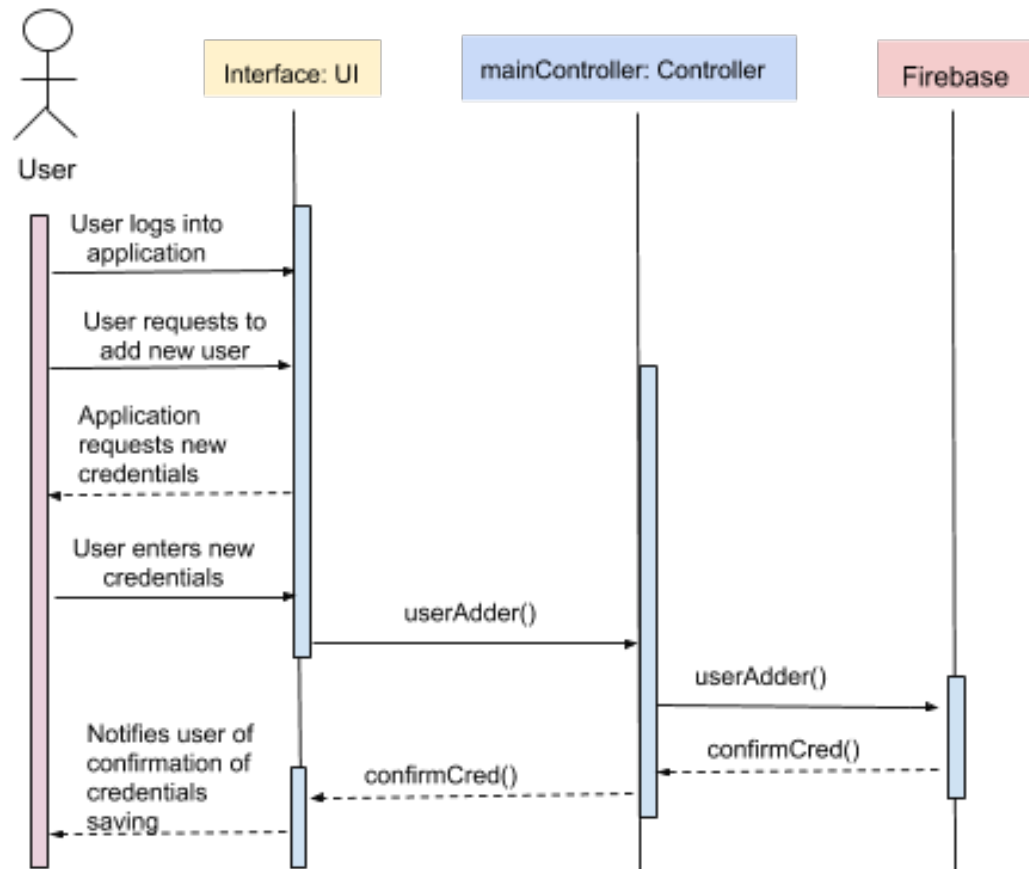
Use Case 7: System Control Features



For this diagram, we considered the Low Coupling Principle as the controller should not take on too many responsibilities communicating since each of the following parts of the system is dependant on the controllers communication. Further we considered the LSP and Open Closed Principle (OCP) as each element of the system (ie. motion sensor and alarm) should be allowed to be enabled and disabled but the methods to activate should not be altered in the process.

UC-10: User Profile Creation and Settings

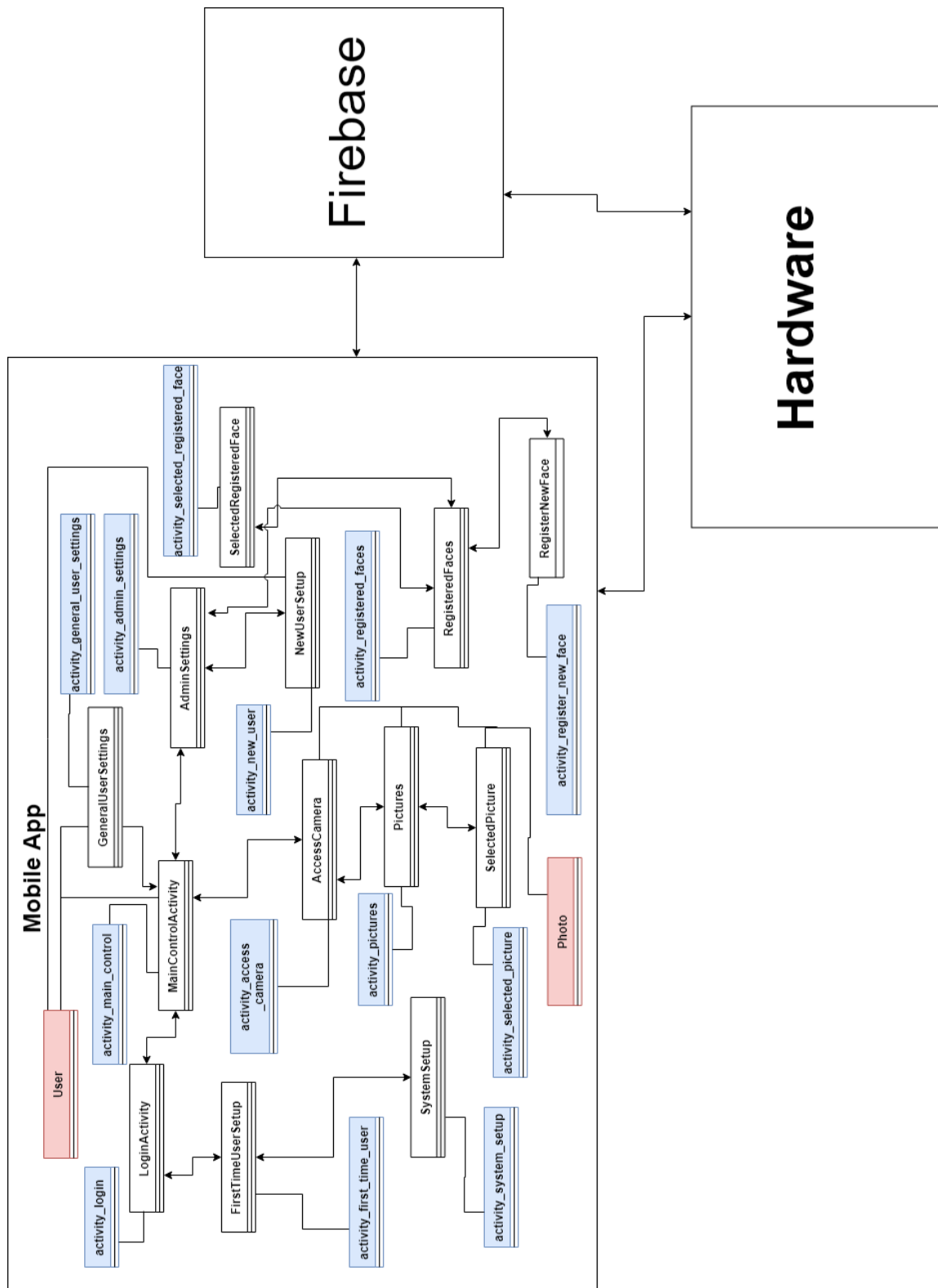
Use Case 10: User Profile Creation and Settings



For this system we took into account we considered the Low Coupling Principle as the controller should not take on too many responsibilities communicating since its main goal is to add the new user in the database. Moreover, we took into account the Interface Segregation Principle as the change to create the new user should not change any other element of the controller, base, or user app interface.

2 Class Diagram and Interface Specification

2.1 Class Diagram



Our class diagram has 3 main components, the mobile app, the hardware which incorporates the arduino, lights, camera, alarm and gyroscope, and Firebase. All three main components are connected to each other and are in constant communication. The main set of classes is contained in the mobile app. The mobile app is built using Android Studio and the code is written in Java. Our design requires that each new page in the app have its own class to control any events or actions on the page. Every double arrow line in the diagram describes a possible transition from one page to another. Each white background box represents a controller class in the UML diagram. These are the classes that each control one page in the app. The blue shaded boxes represent the UI (User Interface) code in the app. The UI code is done in xml. Each UI related file is shown connected to its controller class. The red shaded boxes represents objects we will create. The Photo object will allow us to easily utilize and display the photos taken from the Firebase and the User object will enable us to store the specific user's privileges to allow us to check whether or not a specific user action is allowed. These objects will also help us pass data back and forth between the classes without having to pull data from the firebase everytime.

2.2 Data types and Operation Signatures

2.2.1 LoginActivity

LoginActivity
<ul style="list-style-type: none"> - LoginButton - SetupButton - editTextEmail - editTextPassword - TextViewSignin - firebaseAuth
<ul style="list-style-type: none"> - registerUser() - LoginUser() - onClick()

```

LoginButton:Button
SetupButton:Button
editTextEmail:EditText
editTextPassword:EditText
TextViewSignin:TextView
firebaseAuth:FirebaseAuth

```

```

registerUser():void
LoginUser():void

```

`onClick():void`

In this page the user can log in to the system with their email and password or for a first time user they can use the setup button to redirect them to a new page and begin the initial setup of the Home Security Automation system. The `LoginButton` and `SetupButton` are two buttons that allow the user to navigate to the main control activity page or the setup of pages to set up the initial system. The `editTextEmail` and `editTextPassword` objects are two input textboxes for the user to enter their login credentials into. The `TextViewSignin` is a text output message on the screen. The `firebaseAuth` is part of the Firebase API that will allow the user to authenticate themselves to log in to the app. The `registerUser()` method will redirect the user to a new screen to register themselves and set up the system. The `LoginUser()` will check the user's credentials and log them into the app. The `onClick()` method will take in the user's button input and redirect to the proper method.

2.2.2 FirstTimeUserSetup

FirstTimeUserSetup
<ul style="list-style-type: none">- <code>CreateAccountButton</code>- <code>Cancel</code>- <code>editTextEmail</code>- <code>editTextPassword</code>- <code>editTextConfirm</code>- <code>TextViewMessage</code>- <code>firebaseAuth</code>
<ul style="list-style-type: none">- <code>registerUser()</code>- <code>cancel()</code>- <code>onClick()</code>

`CreateAccountButton:Button`

`Cancel:Button`

`editTextEmail:EditText`

`editTextPassword:EditText`

`editTextConfirm:EditText`

`TextViewMessage:TextView`

`firebaseAuth:FirebaseAuth`

`registerUser():void`

`cancel():void`

`onClick():void`

This page lets a user create the first account on their system by entering a email and a password which the Firebase system will confirm and add to its authentication database. After successfully registering a user, the user will be redirected to a page to connect and setup up the Arduino system. The `CreateAccountButton` is a button what will take the user's credentials and create a new user account on the Firebase and for the app. The `Cancel` button is used to take the user back to the previous screen. The `editTextEmail` and `editTextPassword` objects are two input textboxes for the user to enter their login credentials into. The `firebaseAuth` is part of the Firebase API that will allow the user to authenticate themselves and register for the app. The `TextViewMessage` is a text output message on the screen. The `registerUser()` method is create the user's account on the Firebase and then redirect them to the next page to setup the Arduino system. The `cancel()` method takes the cancel button press and takes the user back to the previous screen. The `onClick()` method will take in the user's button input and redirect to the proper method.

2.2.3 SystemSetup

SystemSetup
<ul style="list-style-type: none">- HomeImage- WifiImage- PhoneImage- CancelButton- ConfirmButton- TextViewMessage
<ul style="list-style-type: none">- cancel()- confirm()- onClick()

```
HomeImage:ImageView
WifiImage:ImageView
PhoneImage:ImageView
CancelButton:Button
ConfirmButton:Button
TextViewMessage:TextView
```

```
cancel():void
confirm():void
onClick():void
```

This class is used to help the user set up their phone with the Arduino system. It checks through the wifi and the Arduino system and will connect the app to the user's home security system. The `HomeImage`, `WifiImage`, and `PhoneImage` are image objects to show the user the required components for the system to connect and be set up. The `CancelButton` and `ConfirmButton` buttons are for the user to either cancel the action, or confirm the setup once its completed. The `TextViewMessage` is an output text on the screen to communicate messages to the user. The `cancel()` and `confirm()` methods respond the the `CancelButton` and `ConfirmButton` and perform the designated actions. The `onClick()` method will take in the user's button input and redirect to the proper method.

2.2.4 MainControlActivity

MainControlActivity
<ul style="list-style-type: none"> - LightsOn - AlarmOn - AccessCameraButton - DialEmergencyServicesButton - PresetMode - SettingsButton - ExitButton
<ul style="list-style-type: none"> - setHomeMode() - setAwayMode() - setOffMode() - turnOnOffLights() - disarmArmSecurity() - onClick() - logOut() - onActivityResult()

LightsOn:ToggleSwitch

AlarmOn:ToggleSwitch

AccessCameraButton:Button

DialEmergencyServicesButton:Button

PresetMode:Integer [-1 = Away Preset, 0 = Off Preset, 1 = Home Preset]

SettingsButton:Button

ExitButton:Button

setHomeMode():void

setAwayMode():void

```

setOffMode():void
turnOnOffLights():void
disarmArmSecurity():void
onClick():void
logOut():void
onActivityResult():void

```

This class serves as the backend for the Security System Control Panel window on the mobile application. The user will be able to customize their home automation system settings or use preset combinations Home and Away modes. This window will also serve as a portal to other windows, such as the settings page, the login page (logging out), and the camera page. The **LightsOn** ToggleSwitch is a .xml component that will indicate whether the lights are currently on or not, and will allow the user to switch its state via the **turnOnOffLights()** method. The **AlarmOn** ToggleSwitch is a .xml component that will indicate whether the alarm system is currently armed or disarmed, and will allow the user to switch its state via the **disarmArmSecurity()** method. The **AccessCameraButton** is a button that will allow the user to transition to the Access-Camera page (class #5). The **DialEmergencyServicesButton** is a button for the user's convenience to quickly call 9-1-1. The **PresetMode** integer will indicate if a preset mode is currently selected, and will connect to an .xml slider switch component that will indicate the current mode appropriately through the front-end. The user will be able to transition between the preset modes with the **setHomeMode()**, **setAwayMode()**, **setOffMode()** methods. The **SettingsButton** will allow the user to transition to the settings page (which can either be the admin settings page or normal user settings page). The **ExitButton** will allow the user to log out of his/her account by calling the **logOut()** method. The **onClick()** method will take in the user's button input and redirect to the proper method. The **onActivityResult()** method will reload up this screen when the user returns to it from another screen.

2.2.5 AccessCamera

AccessCamera
<ul style="list-style-type: none"> - TakePicture - CallButton - BackButton - PhotosButton - firebaseDB - StorageRef - image

<ul style="list-style-type: none"> - <code>onClick()</code> - <code>takePicture()</code> - <code>callEmergency()</code> - <code>Back()</code> - <code>onActivityResult()</code>

```

TakePicture:Button
CallButton:Button
BackButton:Button
PhotosButton:Button
image:ImageView
firebaseDB:FirebaseDatabase
StorageRef:StorageReference

```

```

onClick():void
takePicture():void
callEmergency():void
Back():void
onActivityResult():void

```

This class gives the user access to the camera and allows the user to take a picture of the current camera view and display the image on the app. The user can also navigate to a page to see older photos taken by the Arducam. The **TakePicture** attribute represents a button used to take a picture on the Arduino camera. The **CallButton** is used by the user to quickly call 9-1-1. The **BackButton** is a button object that will return the user to the previous screen. The **PhotosButton** will take to user to another screen to view all the photos currently stored on the Firebase. The **FirebaseDB** and **StorageRef** attributes are a part of the Firebase API to access the Firebase storage and database systems. The **image** attribute is an **ImageView** object that is used to display the photo. The **onClick()** method will take in the user's button input and redirect to the proper method. The **takePicture()** method responds to the **TakePicture** button and will take a picture on the Arducam. The **callEmergency()** method will respond to the **CallButton** and will call 9-1-1. The **Back()** method will respond to the **BackButton** and return the user to the previous screen. The **onActivityResultMethod** will reload up this screen when the user returns to it from another screen.

2.2.6 Pictures

Pictures

- photoList
- Exit
- StorageRef
- gridview
- onClick()
- onActivityResult()
- displayImage()
- exit()

```
photoList:List<Photo>
Exit:Button
StorageRef:StorageReference
gridview:GridView
```

```
onClick():void
onActivityResult():void
displayImage():void
exit():void
```

This class displays the to user the set of pictures the Arducam has taken over time. The pictures are loaded from the Firebase storage reference and are stored into a List<Photo> object. A user can select a specific image and have the image displayed in a full screen mode. This will take the user to another screen in the app. The **photoList** object contains a list of photos that are obtained from the Firebase. The **Exit** object is a button for the user to return to the previous screen. The **StorageRef** object is used to obtain a Firebase storage instance to retrieve data. The **gridview** object is used to display the images in a grid. The **onClick()** method will read in user input such as a button or photo press and call the appropriate method. The **onActivityResult()** method reloads this screen when the user returns to it after viewing a specific photo. The **displayImage()** method sends the user to a new screen to view a photo and the exit method returns the user to the previous screen in the app.

2.2.7 SelectedPicture

SelectedPicture

- selectedPicture
- imageView
- deleteButton
- exitButton
- StorageRef
- imageInfo
- deletePicture()
- onClick()
- exit()

```

selectedPicture:Photo Object
imageView:ImageView
deleteButton:Button
exitButton:Button
StorageRef:StorageReference
imageInfo:TextView

```

```

deletePicture():void
onClick():void
exit():void

```

This class magnifies a picture that the user selected from the previous Picture Library Window (#6). It displays the time the picture was taken (dd/mm/yy: hh: mm: ss), as the title. The **selectedPicture** variable is a photo object that will hold the image and a vector that holds the date captured of the photo. The **StorageReference** variable, called **StorageRef**, will help in acquiring the image from the Firebase DB by giving a reference point to the database for accessing it. The **imageView** is a .xml component that serves as a container for displaying the image. The **deleteButton** is a button that will call the **deletePicture()** method upon being pressed. The **deletePicture()** will delete the image from the Firebase database of pictures. The **exitButton** will call the **exit()** method once pressed, which will transition the user to the previous page. There will be a textview .xml component that is simply a textbox. It will display the photo's date of capture, for the user's convenience and understanding of what photo is being examined. The **onClick()** method will read in user input such as a button press and call the appropriate method.

2.2.8 GeneralUserSettings

GeneralUserSettings

<ul style="list-style-type: none"> - tableView - enableFingerPrint - backButton - manager - cipher - keyStore - keyGenerator - image
<ul style="list-style-type: none"> - onClick() - addFingerPrint() - generateKey() - initCipher() - Back()

```
tableView:TableView
enableFingerPrint:Button
backButton:Button
manager:FingerprintManager
Cipher:Cipher
keyStore:KeyStore
keyGenerator:KeyGenerator
image:ImageView
```

```
onClick():void
AddFingerPrint():void
generateKey():void
initCipher():void
Back():void
```

This page is used for the general user settings. Any non-admin user can view the privileges for themselves in a table and they can enable fingerprint authentication on their app. The `tableView` object is used to display information about the user's privileges. The `enableFingerPrint` button is used by the user to enable fingerprint authentication on their app. The `backButton` button and `Back()` method are used together to read user input and return to the previous screen. The `onClick()` method is used to listen for user input and respond with the appropriate method. The `manager`, `cipher`, `keyStore`, `keyGenerator` attributes and the `AddFingerPrint()`, `generateKey()`, and `initCipher()` methods are all used together to enable fingerprint authentication for the user. The `image` object is used to display images on the screen to the user.

2.2.9 NewUserSetup

NewUserSetup
<ul style="list-style-type: none">- usernameTextField- passwordTextField- confirmPasswordTextField- createaccountButton- username- password- ExitButton- firebaseAuth- StorageRef
<ul style="list-style-type: none">- onClick()- Back()- CreateAccount()

```
usernameTextField:TextField
passwordTextField:TextField
confirmpasswordTextField:TextField
createaccountButton:Button
username:String
password:String
ExitButton:Button
firebaseDB:FirebaseDatabase
StorageRef:StorageReference

onClick():void
Back():void
CreateAccount():void
```

This class serves as the backend for the first time user setup page. The `usernameTextField` is the .xml component where the user enters their desired username. The `passwordTextField` is the .xml component where the user enters their desired password. The `confirmpasswordTextField` is the .xml component where the user attempts to match the password entered in the above text field (`passwordTextField`). The `createaccountButton` will check the two password textfields to see if they match. If they do, the username and password will subsequently be sent to the Firebase database. If not, an error message will be printed. The username and password variables will thus store the values passed, and this will be sent to the Firebase database. The `firebaseDB` and `StorageRef` attributes are a part of the Firebase API to access the Firebase storage and database systems. The `onClick()` method will take in the

user's button input and redirect to the proper method. The `Back()` method will transition the user back to the previous page. The `CreateAccount()` method will send the username and password variables to the `firebaseDB` and `StorageRef` methods to store into the database. It will be called once the `createaccountButton()` is pressed, and the passwords of the two text fields are deemed to match.

2.2.10 AdminSettings

AdminSettings
<ul style="list-style-type: none"> - tableView - backButton - lightsEnabled - alarmssystemEnabled - cameraEnabled - facialrecognitionsettingsButton - createnewuserButton - lightsandalarmenabled
<ul style="list-style-type: none"> - onClick() - Back() - CreateNewUser()

```
tableView:TableView
backButton:Button
lightsEnabled:ToggleSwitch
alarmssystemEnabled:ToggleSwitch
cameraEnabled:ToggleSwitch
facialrecognitionsettingsButton:Button
createnewuserButton:Button
Lightsandalarmenabled:Boolean

onClick():void
Back():void
CreateNewUser():void
```

This class is the backend for the Admin Settings page of the Mobile Application, which will enable the homeowner (administrator) of the security system to add users to the system and choose which privileges they have, i.e which aspects of the security system they can change. The `tableView` is a `TableView.xml` component, which serves as a container to indicate which users have which privileges. The `backButton` is a button that will call the `Back()` method, and transition the user to the previous page. The `lightsEnabled`

ToggleSwitch is a .xml component that will indicate whether the user that is being created will have the privilege to change the lights. The `alarmSystemEnabled` ToggleSwitch is a .xml component that will indicate whether the user that is being created will have the privilege to disarm/arm the system. The `cameraEnabled` ToggleSwitch is a .xml component that will indicate whether the user that is being created will have the privilege to take pictures and access the Firebase DB pictures. The `facialrecognitionsettings` button is a button that will transition the user to the Facial Recognition Settings UI page of the mobile app. The `createNewuserButton` is a button that will transition the user to the First Time User Setup Page of the Mobile Application, and will call the `CreateNewUser()` method, which will take the current state of the ToggleSwitches and set the privileges of the new user accordingly. The `lightsandalarmlenabled` variable is a boolean set to true if the new user is able to arm/disarm the security system and turn on/off the lights, and will enable the user to thus have the privilege of switching modes from “Home” to “Away” to “OFF”. The `onClick()` method will read in user input such as a button press and call the appropriate method.

2.2.11 RegisteredFaces

RegisteredFaces
<ul style="list-style-type: none"> - photoList - Exit - StorageRef - gridView - AddFace
<ul style="list-style-type: none"> - onClick() - onActivityResult() - displayImage() - exit() - addNewFace()

```
photoList:List<Photo>
Exit:Button
StorageRef:StorageReference
gridview:GridView
AddFace:Button

onClick():void
onActivityResult():void
displayImage():void
exit():void
```

This class displays to user the set of registered faces on the system. The pictures are loaded from the Firebase storage reference and are stored into a `List<photo>` object. A user can select a specific image and have the image displayed in a full screen mode. This will take the user to another screen in the app. The `photoList` object contains a list of photos that are obtained from the Firebase. The `Exit` object is a button for the user to return to the previous screen through the use of the `exit()` method. The `StorageRef` object is used to obtain a Firebase storage instance to retrieve data. The `gridview` object is used to display the images in a grid. The `onClick()` method will read in user input such as a button or photo press and call the appropriate method. The `onActivityResult()` method reloads this screen when the user returns to it after viewing a specific photo. The `displayImage()` method sends the user to a new screen to view a photo and the `exit()` method returns the user to the previous screen in the app. The `AddFace` button and `addNewFace()` method will redirect the user to a new screen to register a new face for the security system.

2.2.12 SelectedRegisteredFaces

SelectedRegisteredFaces
<ul style="list-style-type: none"> - selectedRegisteredFace - exitButton - deleteRegisteredFaceButton - StorageRef - imageView
<ul style="list-style-type: none"> - onClick() - deleteRegisteredFace() - exit()

```

selectedRegisteredFace:Photo Object
exitButton:Button
deleteRegisteredFaceButton:Button
StorageRef:StorageReference
imageView:ImageView

onClick():void
deleteRegisteredFace():void
exit():void

```

This class magnifies a picture of a `registeredFace` that the user selected from the previous Registered Faces Library Window (#12). The `selectedRegisteredFace` variable is a photo object that will hold the image. The `StorageReference` variable, called `StorageRef`,

will help in acquiring the image from the Firebase DB by giving a reference point to the database for accessing it. The `imageView` is a .xml component that serves as a container for displaying the image. The `deleteRegisteredFaceButton` is a button that will call the `deleteRegisteredFace()` method upon being pressed. The `deleteRegisteredFace()` will delete the image from the Firebase database of pictures. The `exitButton` will call the `exit()` method once pressed, which will transition the user to the previous page. The `onClick()` method will read in user input such as a button press and call the appropriate method.

2.2.13 RegisterNewFace

RegisterNewFace
<ul style="list-style-type: none"> - TakePicture - BackButton - RetakeButton - firebaseDB - StorageRef - image - AddFaceButton
<ul style="list-style-type: none"> - onClick() - takePicture() - Back() - onActivityResult() - displayImage() - addNewFace()

TakePicture:Button

BackButton:Button

RetakeButton:Button

image:ImageView

firebaseDB:FirebaseDatabase

StorageRef:StorageReference

onClick():void

takePicture():void

Back():void

onActivityResult():void

displayImage():void

addNewFace():void

This class gives the user access to the camera and allows the user to take a picture of the current camera view and display the image on the app. This is similar to the `AccessCamera` class (#5), however, this is for the sole purpose of adding a registered face to the `firebaseDB` so that the security system will recognize the new entry upon their next arrival. The `TakePicture` attribute represents a button used to take a picture on the Arduino cam, via the `takePicture()` method. The `BackButton` is a button object that will return the user to the previous screen, by calling the `back()` method.. The `RetakeButton` will allow the user to retake a photo, and will call the `onActivityResult()` method, which will reload this screen, removing the previously taken photo from the `imageView`. The `onActivityResult()` method will reload the screen when the user returns to it from another screen. The `firebaseDB` and `StorageRef` attributes are a part of the Firebase API to access the Firebase storage and database systems. The `image` attribute is an `ImageView` object that is used to display the photo. The `onClick()` method will take in the user's button input and redirect to the proper method.

2.2.14 Photo

Photo
- url
- date
+ getURL()
+ getDate()

```
url:String
date:String
```

```
getURL():String
getDate:String
```

This class contains the `Photo` object. The `url` attribute is a string data type that contain information to display the object and the `date` attribute holds the date of the photo. The two methods are "getter" methods that are used to retrieve the attribute data for a `Photo` object.

2.2.15 User

User

<ul style="list-style-type: none"> - username - password - lightsPriv - alarmPriv - callPriv - cameraPriv - modePriv
<ul style="list-style-type: none"> + getUsername() + getPassword() + getLightsPriv() + setLightsPriv() + getAlarmPriv() + setAlarmPriv() + getCallPriv() + setCallPriv() + getCameraPriv() + setCameraPriv() + getModePriv() + setModePriv()

```

username:String
password:String
lightsPriv:Boolean
alarmPriv:Boolean
callPriv:Boolean
cameraPriv:Boolean
modePriv:Boolean

```

```

getUsername():String
getPassword():String
getLightsPriv():Boolean
setLightsPriv():void
getAlarmPriv():Boolean
setAlarmPriv():void
getCallPriv():Boolean
setCallPriv():void
getCameraPriv():Boolean
setCameraPriv():void
getModePriv():Boolean

```

`setModePriv():void`

This is the **User** object class. It stores the user's **username** and **password** along with their privileges. There are several attributes, **lightsPriv**, **alarmPriv**, **callPriv**, **cameraPriv**, **modePriv**, each control a different user privilege. Each attribute has a "getter" and "setter" method associated with it to get the value of the attribute and to set the value of the attribute.

2.3 Traceability Matrix

Domain Concepts (Columns) vs. Classes (Rows)

	Controller (ABU)	Mobile App Pic Interface	Mobile App Security Interface	Server Connection	Arducam	Database Connection	Alarm	Lights
LoginActivity			X	X		X		
FirstTimeUserSetup	X	X	X	X		X		
SystemSetup	X			X		X		
MainControlActivity	X			X	X		X	X
AccessCamera	X	X		X	X			
Pictures		X		X		X		
SelectedPicture		X				X		
GeneralUserSettings	X			X		X	X	X
AdminSettings	X			X	X	X	X	X
NewUserSetup		X	X	X		X		
RegisteredFaces		X		X		X		
RegisterNewFaces		X	X					
SelectedRegisteredFace		X				X		
User			X	X		X		
Photo		X				X		

We took the liberty of combining certain domain concepts:

Lights = [Lights + LightsChecker + LightStatus + UserLightsPref + Lights UI]

Controller (ABU) = [Controller (ABU) + Gyroscope + TargetAngle]

Alarm = [Alarm + RemoteAlarmDisable]

Domain Conceptions

Controller (ABU):

FirstTimeUserSetup: This class provides the process by which the user begins setting up their ABU by creating a username and password and proceeding to the systemsetup page.

SystemSetup: This class administer the necessary networking connection setup to connect the user's device to the same network as the ABU and allow for all of the other functionality to be possible.

MainControlActivity: This class is the main menu layout of the mobile app which lets the user toggle different components of the ABU, such as the lights and the alarm.

AccessCamera: This class allows the user to utilize the arducam on the ABU and be able to take pictures with it.

GeneralUserSettings: This class provides a means to view the user's permissions and privileges for the components of the ABU.

AdminSettings: This class allows the admin to view the permissions of everyone connected to their ABU and allows the admin to enable and disable certain permissions for other users.

Mobile App Pic Interface:

FirstTimeUserSetup: This class provides the setup process when the user is setting their ABU for the first time. It requires the user to take a photo and register a face which utilizes the mobile app pic interface.

AccessCamera: This class provides a means to access the Arducam and utilizes the mobile app pic interface when viewing the camera view and taking a picture.

Pictures: This class allows the user to view all of the pictures taken by the Arducam on the ABU, and it utilizes the mobile app pic interface to view them.

SelectedPicture: This class gives the user a way to view a specific picture in the pictures class, but in a larger scale along with the time of capture and is a part of the mobile app pic interface.

NewUserSetup: This class allows the admin to create a new user and when this is done, it must utilize the mobile app pic interface when taking a picture to register the new user's face.

RegisteredFaces: This class provides a way for the user to view the pictures of all the registered faces, which utilizes the mobile app pic interface to view them.

RegisterNewFaces: This class gives the user a way to register a new face for the ABU and when the user does this it utilizes the mobile app pic interface.

SelectedRegisteredFace: This class provides the user to view a specific photo from the registeredfaces class and utilizes the mobile app pic interface to do this.

Photo: The photo class is the primary content that is located within the mobile app pic interface.

Mobile App Pic Interface:

LoginActivity: This class provides the user with the ability to log into the ABU and involves the mobile app security interface for credentials.

FirstTimeUserSetup: This class gives the user the ability to create user credentials when setting up the ABU for the first time and involves the mobile app security interface for credentials.

NewUserSetup: This class allows the admin to create a new user profile and utilizes the mobile app security interface for the credentials.

RegisterNewFaces: This class provides a way for the user to register new faces for the ABU, however before this occurs the user must provide authentication for security purposes and this involves the mobile app security interface.

User: The mobile app security interface deals with the user class primarily and its credentials.

Server Connection:

LoginActivity: This class gives the user a way to log into the ABU and when the user enters their credentials a server connection is involved to authenticate the credentials and connect to the ABU.

FirstTimeUserSetup: This class provides the user with the initial setup process and requires a server connection to send user credentials to the server to store them.

SystemSetup: This class allows the user's device to connect to the ABU through the wifi network and establishes the connection from the phone to the server of the ABU.

MainControlActivity: This class acts as a main menu so the user can toggle different components of the ABU such as the lights and the alarm and in order to send information to the ABU to perform these actions it requires a server connection to send data to the server and then to the firebase so the ABU can detect the changes and perform the action.

AccessCamera: This class provides the user with a way to access the arducam to take pictures and for this to occur a server connection must be made in order for the arducam to become active on the ABU.

Pictures: This class allows the user to view all of the pictures taken by the Arducam and to have this access to the pictures a server connection must be made to the Firebase database to get the data.

GeneralUserSettings: This class provides the user with a way to view their permissions, but in order to get the permission data a server connection must be made to the Firebase database.

AdminSettings: This class gives the admin a way to view all of the other user's permissions and have the ability to change them and to get the permission data a server connection must be made to the Firebase database.

NewUserSetup: This class allows the admin to create new user profiles and to add all the credentials to the server a server connection must be made.

RegisteredFaces: This class allows the user to see all of the registered faces on the ABU and to get the data of the faces a server connection must be made to the Firebase database.

User: The data utilized using the user class such as the credentials must be stored within the database and in order to do this a server connection must be made to store the data.

Arducam:

MainControlActivity: This class provides the basis for the main menu of the mobile app, you would be able to navigate to the arducam page of the mobile (the way you can take a picture).

AccessCamera: This class gives the user direct access to the arducam (The page where you can click a button to take a picture).

AdminSettings: Since this class is what provides the backend for the Admin Settings page, it is important when the user wants to change privileges on who can/cannot take pictures through the Arducam.

Database Connection:

LoginActivity: When a user wants to login to the account their input credentials (username and password) must be cross-referenced to the actual credential on the database.

FirstTimeUserSetup: All the new user information must be loaded onto the database for future purposes

SystemSetup: When the user first wants to set up the system, this class will be used to connect the app to the arduino system. In order to do this, the app will need to connect to the server and then to the database to establish a connection with the ABU

Pictures: This allows the user to view pictures that the Arducam has taken. All the pictures are saved on the database so we would need to have a connection to it in order to provide the user with the pictures

SelectedPicture: This class allows the zooming of a taken picture, and with the picture shows up some information about it such as when the picture was taken. This info will also need to be pulled from the database.

GeneralUserSettings: This class provides the backend for the general settings page which is where the user can see their privileges. This information has to be pulled from the database.

AdminSettings: This is similar to the general user settings but is meant for the admin. Since the admin can view and update the user privileges, we once again need to view and update information on the database.

NewUserSetup: If you want set up a new user for the first time, their information such as their picture, credentials, and privileges need to be saved on the database.

RegisteredFaces: This class displays to user the set of registered faces on the system. These pictures need to be loaded from the database.

SelectedRegisteredFace: Like the SelectedPicture this class also allows the zooming of a picture only this time the picture is of a registered person. With the picture comes the information about when the picture was taken so once again we need a database connection.

User: This class is what provides the backend to store a user's username, password and privileges. This information is stored on the database, thus we need a database connection

Photo: This class is what brings to the user all the data of a picture and the picture itself. This info is also stored on the database.

Alarm:

MainControlActivity: Since this class provides what is the backend of the main menu, you can access the alarm controls from here as well. From there, you can change the state of the alarm.

GeneralUserSettings: This is the class that provides the settings page for the general user. This page includes the privileges of the user and alarm may be one of them, therefore, Alarm is a related concept.

AdminSettings: This is the class that provides the settings page for the admin. This page includes the privileges of the admin and alarm control is one of them, therefore, Alarm is a related concept.

Lights:

MainControlActivity: Since this class provides what is the backend of the main menu, you can access the light controls from here as well. From there, you can change the state of the alarm.

GeneralUserSettings: This is the class that provides the settings page for the general user. This page includes the privileges of the user and lights may be one of them, therefore, Lights is a related concept.

AdminSettings: This is the class that provides the settings page for the admin. This page includes the privileges of the admin and lights control is one of them, therefore, Lights is a related concept.

3 System Architecture and System Design

3.1 Architectural Styles

The architecture styles of this project are the typical styles you would find for a project revolving around the Internet of Things (IoT). The project can be encompassed by four conceptual models: database, server-client, event-driven architecture, and layered architecture.

Database Architecture: Since our whole product is based on taking pictures of people entering the house or simply of people who are at the door, we would need to make use of a database. For our product, we are using the Google Firebase as the database in which we can store these images. Furthermore, the database will not only store the pictures taken, but it will also contain information about the users of the app such as their privileges. If the user ever requests pictures taken by the ArduCam, the database is where they will get it from. Also any updates to the system, e.g. updating user privileges, needs to be stored somewhere to provide consistent behavior.

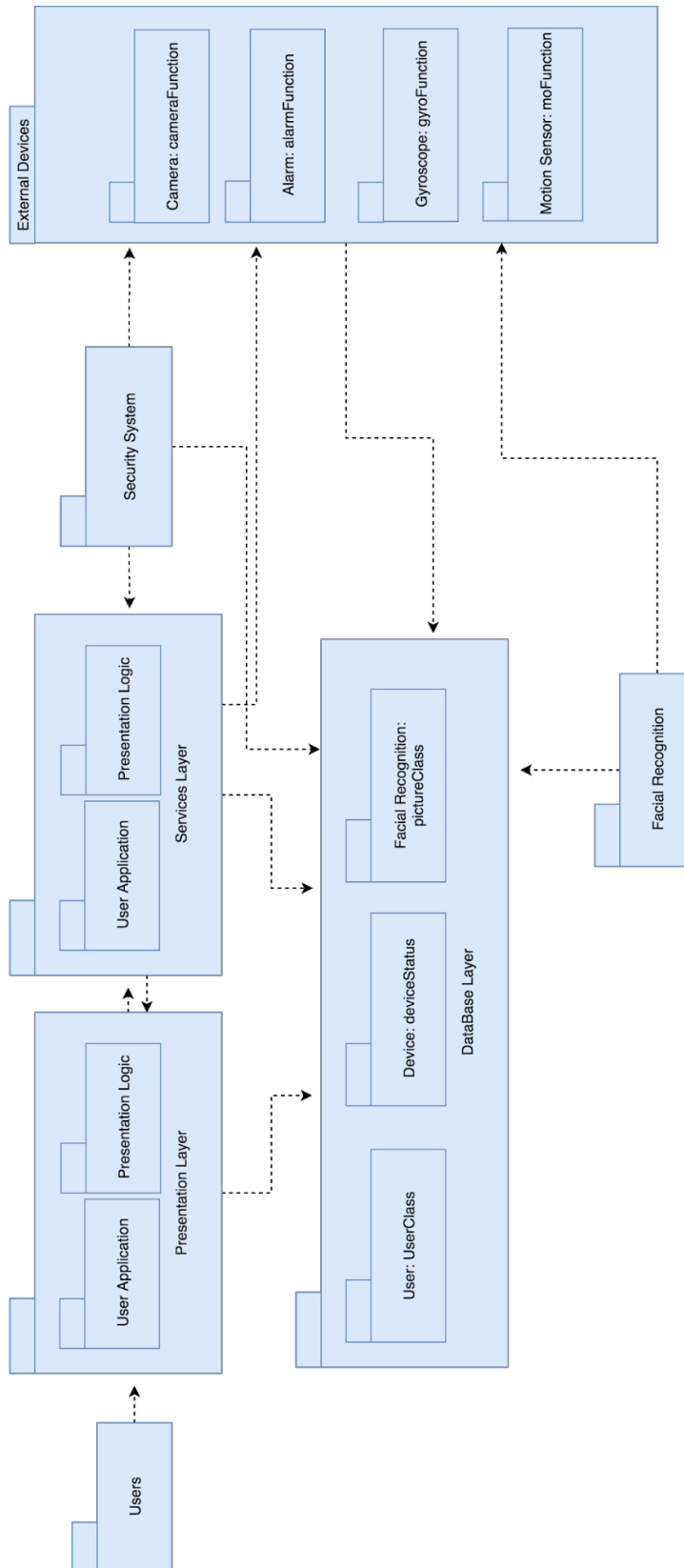
Server-Client Architecture: If we want multiple users to be accessing the product through different devices, some sort of controlled communication is important. In the modern world, the server-client architecture is the most widely used approach for such a problem. If multiple devices (clients) require access to the home automation system,

we need a central system (server) that can use multithreading to provide each client the resources/services they require.

Event-driven Architecture: Home-Automation is all about providing the user with convenient access to modify the states of the home (i.e lights, alarm, camera). In terms of software architecture, events are defined as “state changes” and so in order to drive the system on events we require major communication between the user and the system. For example, if the user uses the mobile app to toggle the switches or to take a picture, this message needs to be relayed to the actual hardware for the event to occur. The same applies when the hardware (Arduino Base Unit - ABU) wants to send notifications to the user.

Layered Architecture: The reason behind why we will be using a layered architecture is because we will need to make abstractions on how our product interacts with the user. In a layered architecture, each layer is separated in a way to create a hierarchy. Each layer provides the necessary services to the layer above it. For example, The most basic layer would be how our product actually does it’s work (technical layer). This part would include the all the mechanisms of the ABU (motion sensors, gyroscope, etc), the actual storing of data and the client and server. Then we would have the domain layer in which the server would be retrieving information from the technical layer for the user. This layer would also include general functions such as user authentication which allows a user to get to this layer. Lastly, we would have the user-interface layer. This is where the app would be found. This is the layer in which the user mostly directly connect to the system. You can see how the bottom most layer (technical) does not need to know anything above any of the above layer, and this pattern follows on as you keep moving upward through the layers towards the user. In this way, we can subdivide the entire system.

3.2 Identifying Subsystems



We were able to divide our system into several different subsystems to better represent the relationships of the layers in our project. There were four main layers that we divided our system into. First we identified the presentation layer which is the view that the user will see. This included the user application and the basic presentation logic as this is the only part of the system that our user will have an influence over. From there we defined the services layer. In this layer we defined the part of the user application that will be influenced by the external devices. The execution of requests on the external devices will influence the logic on the application and vice versa.

All of these layers need access to the database package of the system. The presentation layer and the service layer use the database package as a way to update their application and the external devices use the update the database package with every new request completed. Next, we have the facial recognition package which is standalone, as the logic for the facial recognition is different than the logic used for the external devices. Lastly, we have the security package which authenticates the user before being able to enter the user application.

3.3 Mapping Subsystems to Hardware

Our system requires multiple computers for its use. The system has a client and a server subsystem. The clients are our mobile devices and they will be running on their own machines. Each client has to connect to the server which will be running on the arduino wifi board. The arduino wifi board allows us to set up a server that can connect to the firebase. It will be able to send and receive data from the firebase. This server will also allow mobile devices (clients) to send command signals from one end, and the server will send these signals to the arduino base unit on the other end.

3.4 Persistent Data Storage

With our system, we need to be able to save data that can be accessed by both the user and the arduino. We plan to use Firebase to satisfy our database storage needs. In order to utilize our facial recognition functionality we need to store all the faces that should be approved by the facial recognition software to automatically disable the alarms when opening the door. The Arduino Base Unit should be able to access this and cross reference these pictures with pictures it takes to determine if the images match. Also, the ABU should be able to store images of perceived intruders in a separate collection in the Firebase, so that the user will be able to access these images remotely. In addition to pictures for the saved faces, and intruder images, we have to be able to store info about users that have access to the Android app and adjust privileges if necessary, and be able to authenticate them with a password. Because we are implementing Firebase, we don't necessarily use relational tables, as it is an unstructured database that follows more closely to a json format. Json stands for JavaScript object notation, which is a form of storing and displaying information in a collection of key:value pairs.

Firebase

- store saved faces for facial recognition
- store pics of intruders/unrecognized visitors
- user info for login authentication
 - username, password, privileges
 - sheets (collections) vs. tables
- There are many commands that we can use when we connect firebase with our android app, such as:
 - `FirebaseDataReference = FirebaseDatabase.getInstance().getReference();`
 - This establishes a connection to your firebase with the proper authentication. This lets you read and write through this reference.
 - `DatabaseReference messageRef = FirebaseDataReference.child();`
 - The child of the current reference is a way to navigate when a key value has multiple values.
 - `StorageReference storageReference = FirebaseStorage.getInstance().getReferenceFromUrl(imageUrl);`
 - Similar to the database reference, this is the reference to the storage area that will hold images.
 - `FirebaseDatabaseReference.child(MESSAGES_CHILD).push().setValue(friendlyMessage);`
 - Instead of reading from the database, this is how to write into the database
 - `StorageReference storageReference = FirebaseStorage.getInstance().getReference(mFirebaseUser.getId()).child(key).child(uri.getLastPathSegment());`
 - `putImageInStorage(storageReference, uri, key);`
 - This is a way to put images into the storage of the database.

3.5 Network Protocol

In order to connect the entire system so that multiple devices can interact with it, a server client network system will be implemented. The server client network system will utilize simple java sockets. When a user logs into the mobile application, they, as a client, ping the server and attempt to make a connection to the server. By utilizing multithreading, multiple users can connect to the aforementioned server at the same time using their mobile application and all send requests which will be handled on a queue basis. Once a device is connected to the server, the user can send requests to the server, which in turn send requests to the firebase, and then finally tell the Arduino Base Unit to switch its components on and off. When the user toggles the lights, for example, on the mobile

application, data is sent as a client to the server. The server will then send data to edit on/off values in the Firebase database. The Arduino Base Unit is connected to Firebase and with its configuration it can detect a change in value within Firebase and will toggle its components in correspondence. In this case if the user tapped the button for turning the lights on, the value of the Lights_On variable in Firebase will change to one, and in response, the Arduino Base Unit's corresponding pin out for the lights will produce a high value and thus turn the light on.

3.6 Global Control Flow

Our program is event-driven, since certain events will occur depending on the user triggering it. For example, the alarm will only go off if the door is opened and away mode is still enabled. Furthermore, the order that the events occur are variable, since they are all dependent on what action the user initiates. While we do have loops and waits, they can generate different actions based on the order that they are executed.

As per time dependency, there are a few timers in our system. Mainly, once the alarm is triggered, the user will have 30 seconds to disable it through the mobile application. Despite that, our system is mostly an event response system. Some examples of this include:

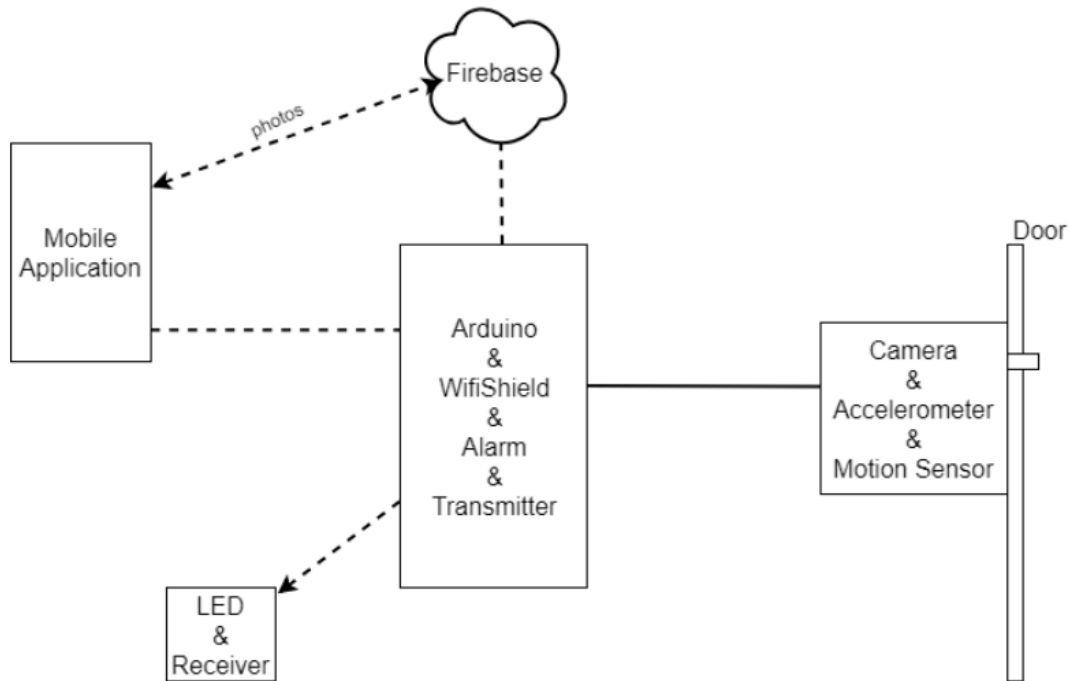
- The motion sensor immediately triggers the Arducam to take a picture
- The picture will be automatically cross-checked with the list of saved faces stored in firebase
- Anyone with the mobile app should also immediately be able to view the pictures that the arducam captured
- If desired, the user can also take an immediate picture of what is outside the door with the Arducam
- Once the accelerometer detects motion of the door opening, the alarm and lights will be enabled immediately

For concurrency, we plan on using multiple threads when we initiate different users as they connect to the base unit. This will prevent multiple users from simultaneously changing the system and causing issues. We can prevent these issues by using mutex locks so variables controlling certain devices cannot get changed at the same time. We can also create threads when we push images up into the firebase server, as there might be some wifi latency that we can shorten by having multiple images uploading at the same time with threads.

3.7 Hardware Requirements

Below is the proposed structure for our hardware will be set up, and connected. In the base unit, the arduino will control the actions of the external devices. To communicate

with the mobile application, the wifi shield is needed to let the arduino effectively become an Internet of Things device, allowing us to control and trigger events through the mobile application over the internet, and to connect the arduino to Firebase to pull data and photos from that online database. We can connect the alarm directly to the arduino in the base unit. However, to communicate with the light, since that will often be an external light somewhere else in the house, we will be using a transmitter and a receiver to communicate signals wirelessly. While we only have one LED, this system easily offers expandability towards adding multiple LEDs. Finally, the Camera, motion sensor, and accelerometer will be in an enclosed box attached to the door, with the camera poking through the existing peephole of the door. They will be connected through a wire to the base unit.

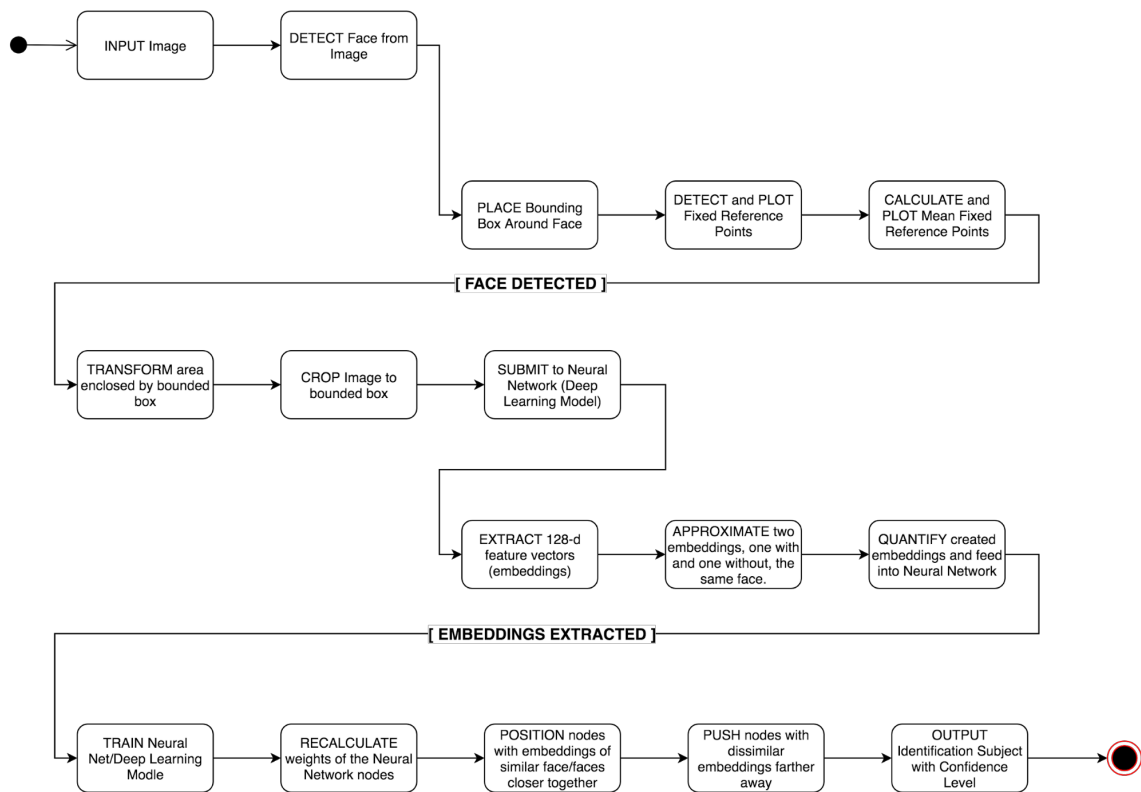


4 Algorithms and Data Structures

4.1 Algorithms

The facial recognition API sourced in the OpenCV library utilizes face detection and deep learning in order to accurately recognize people in images and video streams. The overall process begins by receiving an input image. The presence and location of faces are detected and found from the input using pre-trained models from dlib and OpenCV, both of which are libraries with facial detection functions [21]. The detection algorithm places a bounding box around the profile of the face, and calculates average fixed reference points based on all the fixed reference points that can be detected [22]. Using the bounding box and mean fiducial points, the face is then transformed for the neural network, which is a construct in machine learning that utilizes different algorithms to process complex data

inputs [23]. Using the real-time pose estimation function from the dlib library and the affine transformation function from the OpenCV library, the input image is transformed to make the eyes and bottom lip appear in the same location on each image [22]. The deep neural network then quantifies the face (cropped by the bounding box) by embedding the face on a 128-dimensional unit hypersphere [22]. The embedding generalizes the inputted face, and is able to compare with other embeddings based on quantitative measures, making clustering, similarity detection, and classification more efficient and successful [21]. Creating the embedding involves training the neural net by positioning nodes (which have equal or near equal embeddings) closer to each other, and pushing the embeddings with different values (which don't have near similar faces) farther away [21]. The below activity diagram summarizes the process pipeline.



4.2 Data Structures

Our application makes minimal use of data structures as most of our data will be stored into the Firebase. Our app will utilize arraylists that contain all the images the user can view. The arraylist will contain photo objects that the app can easily manipulate and select from different pictures to display to the user. Storing the pictures in an arraylist also reduces the amount of times the app must interface with the Firebase to retrieve data in one session. Every time the app is loaded and the user wants to view the pictures stored on the Firebase the app will retrieve the data from the firebase and load it into an arraylist for the current session. We utilize arraylists because they can easily change size and allow for

random memory access. Arraylists are also better for displaying data in different types of Android UI components as many UI functions are set to already interface with arraylists. Furthermore, we will also be using arraylists for the list of users that will appear on the mobile app. The users will of course be user objects which will contain the user privileges.

5 User Interface Design and Implementation

Utilizing Android Studio, we have created very similar User Interface Design for the project. While the process is still in progress, it seems as though there have not been any significant changes to our designs and we will be following the mock up designs. The mock up designs were structured in a way to keep the design intuitive and possible to replicate on an android development platform such as Android Studio. A potential edit would be to implement pop up boxes upon waiting for fingerprint authentication or the status of an action (i.e. succeeded or failed). There may also be the need to implement a photo inspection page so the user can choose whether they want to use the picture they just took to add to the facial recognition database. The feature of facial recognition will also require more than one picture so the model can be trained properly, instead it will require five pictures of the person being registered into the facial recognition database, so this user interface will indicate the number of pictures taken, the ability to view them and the ability to take other pictures.

6 Design of Tests

Testing Manual Operation of Arduino Base Unit

- User manually takes picture on the arducam
 - We want to be able to have a user make the system take a picture of whatever the camera currently sees to show that the camera does have functionality beyond the automatic photo taken when the system is activated and can be controlled manually.
 - In our demo, we are planning to connect to the arduino and send the signal for it to activate the camera to take a picture and retrieve the photo taken.
 - For our demos beyond the first, we plan to have the mobile app/arduino system have a more directly connected relationship to each other so the function in the application talks to the arduino to take the photo without needed so much of a middle man laptop.
- User manually turns on/off motion sensor
 - We want to be able to turn on/off the motion sensor to show how when the system is disengaged, the system's motion sensor will not be constantly seeking

out movement, which would then consequently trigger the arducam to take a picture of any visitor that approaches the door.

- In our demo, we plan on showing our progress by directly connecting a laptop to the Arduino Base Unit and using commands from the laptop to simulate how the user would be able to manually take a picture of the exterior of the door.
 - For our final product and the next demo, we plan on having the Android app have the capability to directly connect to the ABU and directly toggle this setting, without the need for a direct connection established between the laptop and the ABU.
- User can tell if door is open (accelerometer)
 - We want to be able to know if the door is opened or in the process of being opened by having the accelerometer report data of motion and angle and send notification of the doors status.
 - In our demo, we plan to have a display with a miniaturized model door with a hinge and begin to open it until the accelerometer identifies the motion and angle and notifies on the device used of the door's status changing from open to close.
 - For our future demos, we plan to have the mobile app connect more directly to the arduino so the notification is presented to the user in the application without needing any device in between.

Testing App Operation

- App should be able to log user in and out
 - We want the app to be accessible by multiple users and we want to make sure that the mobile app itself is not accessible by just anyone. That is why we wanted to implement a login page to restrict access to users with privileges and also restrict users with restricted access.
 - For this demo, we are going to demonstrate how our mobile app has the ability to restrict access to only those users that have access with their usernames and passwords. We will demonstrate this by pulling up a version of the Android app on an Android device, and show the user authentication system for the app. We will also be able to show how the main page allows for the creation of new accounts in addition to logging in.
 - In the final product and the next demo, we plan to show how we can create additional accounts with restricted access, and show how user settings/privileges can be added or modified. We also plan on implementing fingerprint authentication if the Android device allows for it.
- App should allow management of user settings

- We want the app to not only be able to have more than one user but also possess the ability to show said user's settings so different features or priorities can be adjusted depending on the user.
 - In our demo, we plan to show that after a authorized user is able to login then they can go to a page that displays their user settings and to show that they can't be changed and toggled.
 - In future demos, we plan to have the direct connectivity between the app and the whole system be more viable and have different users login and adjust their settings to then demonstrate what works and what does not based on what that user's settings allow.
- App should connect to firebase and read and write data
 - We want our app to be have a connection to our firebase in order for it to read and write information necessary for the system whether it is in the form of login data and face data for facial recognition for example.
 - In our demo, we plan to have some communication between the app and the firebase in order to show other functions such as logging in as that is a function that needs that connection and also show off the firebase and layout.
 - In future demos, we plan to take this connectivity to the next level so it will all be connected as a system allowing each function to work properly without a intermediary device acting as a trigger or anything and show interconnected actions that lead to the manipulation of the alarm system.

Facial Recognition

- Person Detection
 - We want to be able to have the facial recognition software determine if the user is one of the registered users so that the system can confidently disarm the system knowing that a registered user is outside.
 - For this demo we are going to demonstrate our progress made in the facial recognition software by showing how a picture will demonstrate the closeness of matches to an individual.
 - For the final product and the next demo, we hope to have this facial recognition software run automatically upon image capture, and determine if it wants to disarm the system. The results and matching should be much more accurate than they currently are.

7 Project Management and Plan of Work

7.1 Merging the Contributions from Individual Team Members

No issues were encountered while compiling the final report. Each member followed the directions of the respective section they were assigned very closely. I served as an anchor in checking over their work and making sure that everyone followed the guidelines laid out on the class website. The closest thing to an issue was the time as to when everyone finished their parts. Combining everyone's part into latex adds another challenge as the automatic formatting requires strenuous programming behind the scenes to ensure that the layout is to standard, and the final version of the report is presentable. Receiving individual contributions close to the deadline left very little time for the latex formatting, however, this has never resulted in a tardy submission, nor an achievement in less than the full potential of the report.

7.2 Project Coordination and Progress Report

Over the past few weeks, we have been working on the design aspect of the project and have been actively working on the actual hardware and software implementation of our home security system. With our demo coming up, we have to make sure we are ready to present several tests that show the progress we have made on our project.

In regards to hardware, we have started working on connecting various devices (alarms, camera, LED lights) to the arduino board and have successfully connected the arduino board to the wifi shield allowing for the board to potentially be connected to the mobile app, so that it can be controlled wirelessly. As of right now, we have manual wired connection between the laptop and the arduino board. We have tested the alarm, and have set the alarm to trigger when a signal is sent. We are also able to read data from the motion sensor as well as the accelerometer when directly connected. In the future, the motion sensor and the accelerometer should be triggered automatically and should cause the alarms and lights to turn on. Currently the arducam saves pictures onto the connected computer, but in the future it should be able to connect directly to a Raspberry Pi computer and connect to the Firebase and save images there.

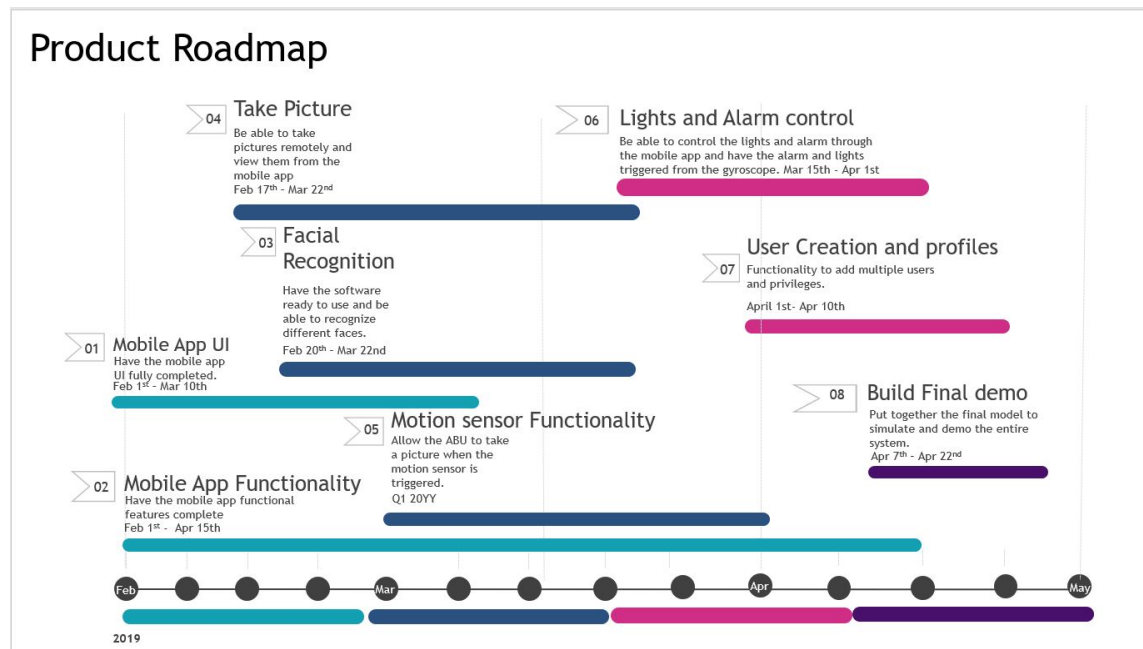
Our mobile app currently has the ability to log in and log out, with password protected accounts. The user interface is relatively set up, but there is still much room for improvement. In the future, the app should be able to connect to the arduino via its wifi shield, and have the functionality to turn on/off certain features of the arduino board. It also will eventually have the ability to push and pull images to/from the Firebase.

For our facial recognition development, we currently have it set up so that given a collection of images taken from before can be compared to the image taken through the arducam and using OpenCV, we can more confidently determine who the person is. In the

future, the software should be able to automatically determine who the person is and if they match anyone in the Firebase. We also will be able to have family members upload selfies of themselves through the app and make the facial recognition more accurate and work more seamlessly.

In terms of use cases we have tackled, we have gotten to at least attempting to solve use cases UC-1: Take Picture, UC-2: Facial Recognition, UC-3: Light Control, UC-6: Fingerprint & Passcode Verification, UC-9: System Connectivity, UC-11: Create New User Account, UC-12: User Logout. As described, we plan on strengthening the functionalities of these use cases and also implementing the remaining use cases.

7.3 Plan of Work



Above is the estimated plan of work for our project. We have shifted our dates of completing specific parts of our project based on the rate of what we have already accomplished and having a realistic view of what we can accomplish in the future.

7.4 Breakdown of Responsibilities

The project is well divided into hardware section and software section and we plan on combining the two parts in the end.

The Software section of the project is with Nikunj, Nirav, Abhishek, and Ashwin. We are making the application in Android Studio using Java. The responsibility of UI of the application is with Nirav and Kaushal. The responsibility of the backend development with

Firebase is Abhishek and Nikunj. This includes the Data Structure of the database we're using and how to optimize storing data and photos. Miraj is dealing with the Algorithms that correlate with the Application. This includes optimizing class structure and design of the overall app. Ashwin deals with Debugging aspect of the application.

The Hardware section is mainly Sagar and Harmit. They are doing a great job of combining their knowledge and building a system that efficiently integrating the hardware component to the app and the real world application.

Design Testing for each side of the project is divided between Andrew and Ashwin. Andrew checks with the hardware part of the system while, Ashwin does system testing with the android application.

Kaavya is running through documenting the project, along with managing everyone's responsibility and helping out where she sees fit.

8 References

[1]ARDUINO/BLUETOOTH APP CONNECTION.

<https://www.youtube.com/watch?v=evVRCL9-TWs>

[2]ARDUINO WIFI CONNECTION.

<https://www.instructables.com/id/How-to-connect-your-Arduino-WiFi-shield-to-a-custo/>

[3]ARDUINO CAMERA.

<https://www.instructables.com/id/ArduCAM-Mini-ESP8266-Web-Camera/>

[4]ARDUINO/ANDROID APP REAL-TIME COMMUNICATION.

<https://medium.com/coinmonks/arduino-to-android-real-time-communication-for-iot-with-firebase-60df579f962>

[5]ARDUINO WEB SERVER.

<https://startingelectronics.org/tutorials/arduino/ethernet-shield-web-server-tutorial/basic-web-server/>

[6]RASPBERRY PI/ARDUINO COMMUNICATION.

<https://maker.pro/raspberry-pi/tutorial/how-to-connect-and-interface-raspberry-pi-with-arduino>

[7]ARDUINO/ANDROID APP CONNECTIVITY.

<https://www.makeuseof.com/tag/6-easy-ways-connect-arduino-android/>

[8]GANTT CHARTS.

<https://www.officetimeline.com/make-gantt-chart/google-docs>

[9]ARDUINO FACE RECOGNITION.

<https://www.hackster.io/team-enzi/alexa-controlled-face-recognizing-arduino-door-bell-465a58>

[10]FIREBASE USER REGISTRATION.

<https://www.youtube.com/watch?v=0NFwF7L-YA8>

[11]FIREBASE USER LOGIN AND USER SESSION.

<https://www.youtube.com/watch?v=KFULmVXpO-A>

[12]USER STORIES.

https://en.wikipedia.org/wiki/User_story

[13]FURPS.

<https://en.wikipedia.org/wiki/FURPS>

[14]PROJECT ROADMAP.

<https://blog.asana.com/2018/08/product-roadmap-tips-templates/>

[15]ACCEPTANCE TESTING.

https://en.wikipedia.org/wiki/Acceptance_testing

[16]INTERACTIVE DIAGRAMS.

https://en.wikipedia.org/wiki/Unified_Modeling_Language#Interaction_diagrams

[17]SEQUENCE DIAGRAMS.

https://en.wikipedia.org/wiki/Sequence_diagram

[18]SOFTWARE ARCHITECTURE.

https://en.wikipedia.org/wiki/Software_architecture#Examples_of_Architectural_Styles_.2F_Patterns

[19]ARDUINO-FIREBASE LIBRARY

<https://github.com/FirebaseExtended/firebase-arduino>

[20]LAYERED-ARCHITECTURE

<https://herbertograca.com/2017/08/03/layered-architecture/>

[21]OPENCV FACIAL RECOGNITION

<https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/>

[22]FACIAL RECOGNITION ALGORITHM

<https://cmusatyalab.github.io/openface/#overview>

[23]UDACITY TENSORFLOW COURSE

<https://classroom.udacity.com/courses/ud187>