# Software Design Document
for
# UCSB 360
Version 1.0

Prepared by

Group Name: Team Epsilon

| | | |
|---|---|---|
| Max Hinson | 4426771 | maxwellhinson@gmail.com |
| Jhon Faghih-Nassiri | 4111274 | jfaghihnassiri@gmail.com |
| Luke Buckland | 4060893 | bigduker20@yahoo.com |
| Andy Chou | 4061123 | andy168chipz@gmail.com |
| Jicheng Huang | 4088779 | jicheng0903@gmail.com |

| | |
|---|---|
| Instructor: | Chandra Krintz |
| Course: | CS 189A |
| Lab Section: | Friday 12 PM |
| Teaching Assistant: | Stratos Dimopoulos |
| Date: | 03/07/13 |

# Revision History

| Version | Primary Author(s) | Description of Version | Date Completed |
|---|---|---|---|
| 1.0 | Team Epsilon | Initial version for SDD | 03/07/13 |

# Table of Contents

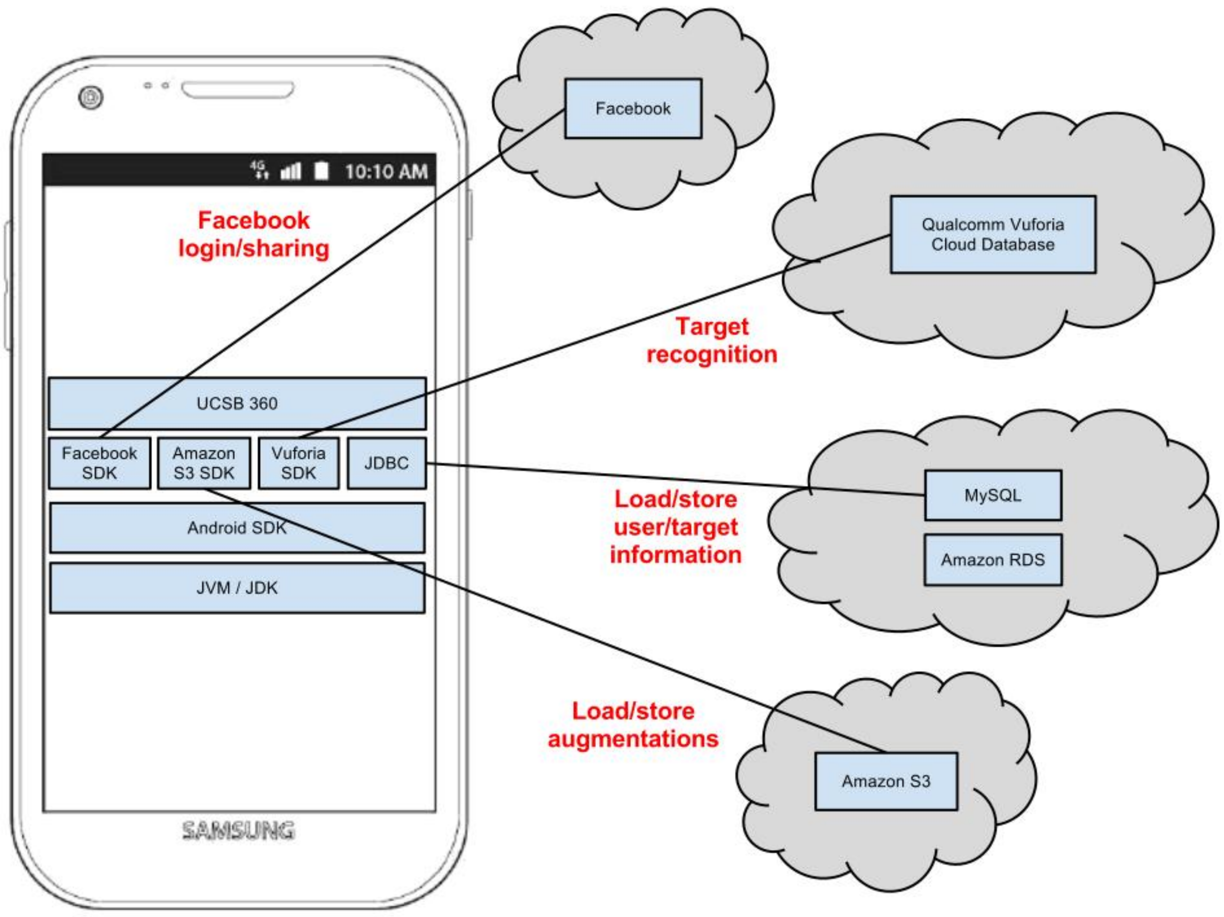
# 1 Introduction

## 1.1 Description

This document describes the system design for the UCSB 360 application. Topics that will be covered range from high-level system architecture and data flow to low-level class descriptions. This document will describe each module in detail, and will include UML diagrams, timing diagrams, and PRD tables.

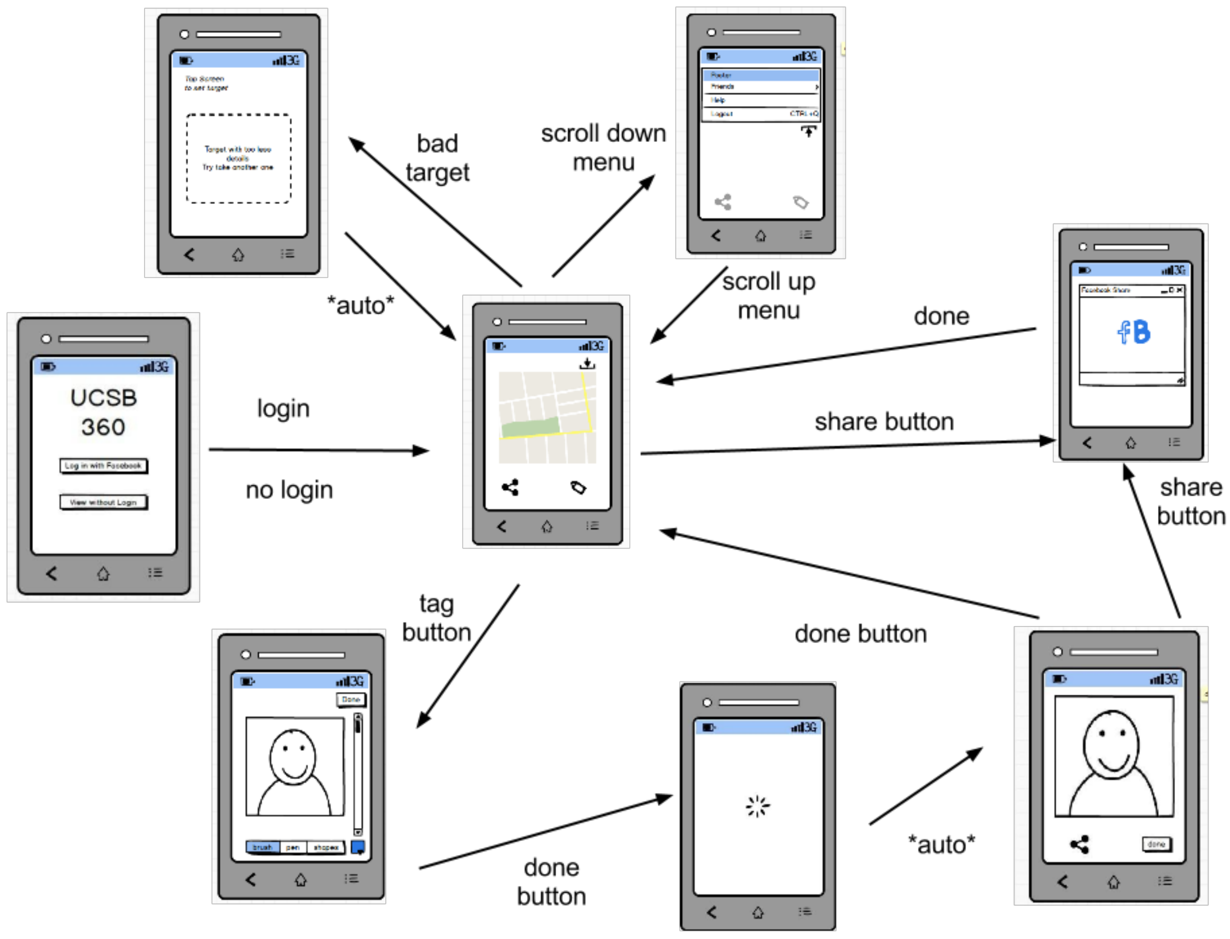
Additionally, preliminary test cases will be provided in a separate zip file. This file will contain two types of test cases: manual and JUnit. Detailed, written user interface tests will be performed manually by the team. JUnit test cases will be run on the mid- and low-level Java code. Both types of tests shall be run regularly to ensure that implementation adheres to the design.

## 1.2 System Overview Diagram



# 2 Graphical User Interface

## 2.1 Flow Chart



## 2.2 Login Page

user will have a choice between login to Facebook or use the application without login in. If user selects to login with facebook account, user will be directed to a login page provided by Facebook API. After authorization is successful, user will be directed to the camera view and start viewing. By contrast, if user clicked “view without login”, he will be directed to the camera view immediate but results in not allowed to share views user has seen with his/her friends.

## 2.3 Camera View

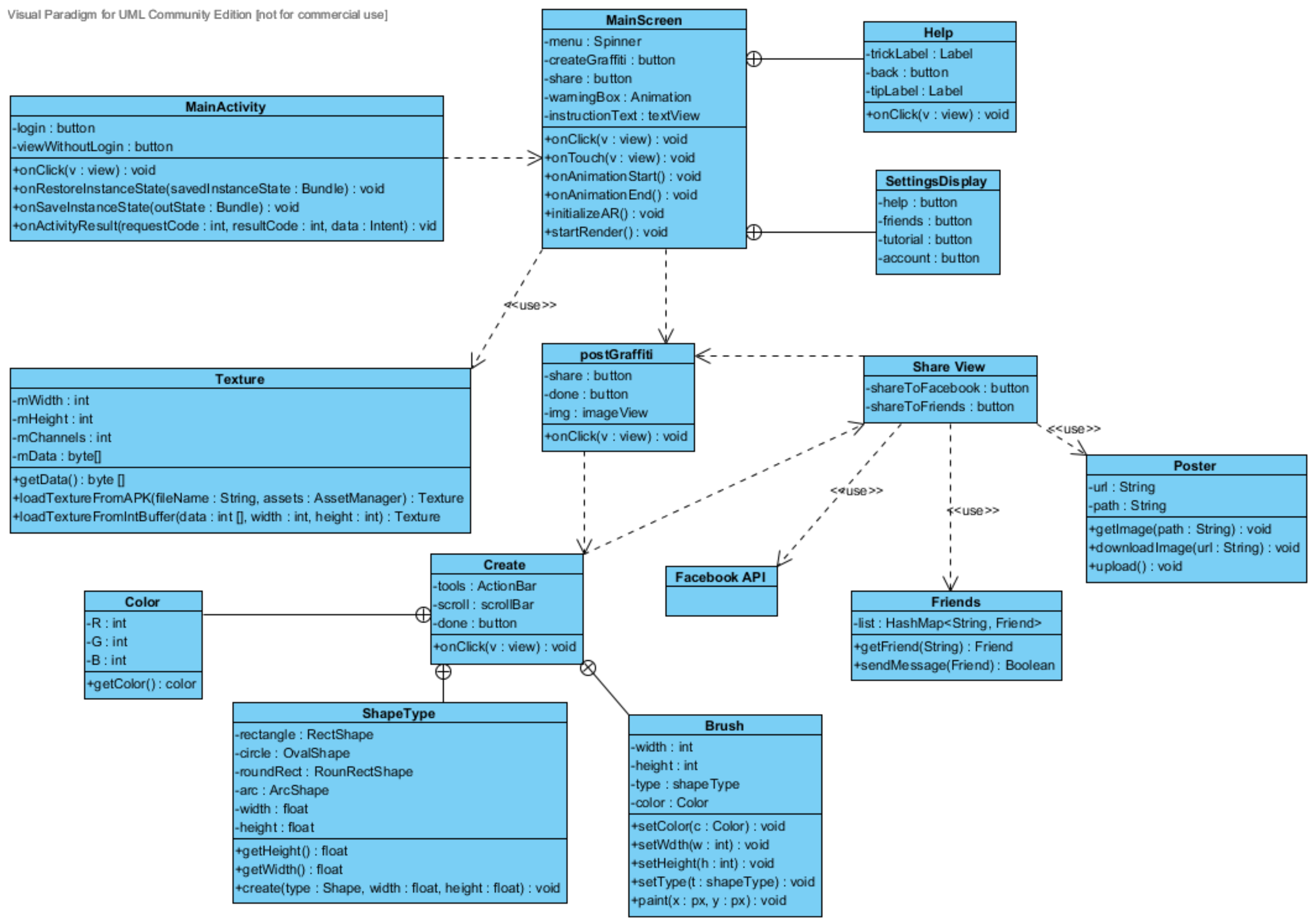
A drop down menu will be available to user on top of this screen, user can slide down the drop button in case he/she needs to do some setting about the application or read some help tips. Whenever user wants to start viewing augmentation, he can just single tap the screen. User will see a scanning bar running back and forth on the screen in order to tell the system is running, and everything a target is recognized from the camera there will be green dots show up on the target. Rendering will start as soon as the target is recognized, and the augmentation of the corresponding target will show up when finished loading from the cloud. By clicking the “share” button located at bottom left corner of the screen, the system will take a snapshot of the current view user is seeing and directs to the share view, and clicking the “tag” button on bottom right corner, system will take a snapshot of only the target from the camera view and direct user to the “create graffiti view”.

## 2.4 Share view

User will be able to see the snapshot system took and decided to post on his facebook wall by clicking the “share to facebook button”. This will direct user to the share page provided by Facebook API and post to wall process will be done in that page.
Or, user can share the view with his friends within this application by clicking the “share to friends” button.

## 2.5 Create Graffiti View

The captured target will be shown on the screen, and painting tools will be available at the bottom of the screen, user can click on each tool icon to use them for creating the augmentation. There is a color picker button on the right side of the tool icons, and a scroll bar for zoom in and zoom out will be available on the right side of the screen. Once user finished drawing his graffiti/augmentation, he can click the “done” button and directs to the share view to see how the final result will look like and decide if he wants to share it with his friends or not.


Visual Paradigm for UML Community Edition [not for commercial use]
MainScreen
-menu : Spinner
-createGraffiti : button
-share : button
-warningBox : Animation
-instructionText : textView
+onClick(v : view) : void
+onTouch(v : view) : void
+onAnimationStart() : void
+onAnimationEnd() : void
+initializeAR() : void
+startRender() : void
Help
-trickLabel : Label
-back : button
-tipLabel : Label
+onClick(v : view) : void
MainActivity
-login : button
-viewWithoutLogin : button
+onClick(v : view) : void
+onRestoreInstanceState(savedInstanceState : Bundle) : void
+onSaveInstanceState(outState : Bundle) : void
+onActivityResult(requestCode : int, resultCode : int, data : Intent) : vid
SettingsDisplay
-help : button
-friends : button
-tutorial : button
-account : button
<<use>>
Texture
-mWidth : int
-mHeight : int
-mChannels : int
-mData : byte[]
+getData() : byte []
+loadTextureFromAPK(fileName : String, assets : AssetManager) : Texture
+loadTextureFromIntBuffer(data : int [], width : int, height : int) : Texture
postGraffiti
-share : button
-done : button
-img : imageView
+onClick(v : view) : void
Share View
-shareToFacebook : button
-shareToFriends : button
<<use>>
Poster
-url : String
-path : String
+getImage(path : String) : void
+downloadImage(url : String) : void
+upload() : void
<<use>>
<<use>>
Create
-tools : ActionBar
-scroll : scrollBar
-done : button
+onClick(v : view) : void
Facebook API
Friends
-list : HashMap<String, Friend>
+getFriend(String) : Friend
+sendMessage(Friend) : Boolean
Color
-R : int
-G : int
-B : int
+getColor() : color
ShapeType
-rectangle : RectShape
-circle : OvalShape
-roundRect : RounRectShape
-arc : ArcShape
-width : float
-height : float
+getHeight() : float
+getWidth() : float
+create(type : Shape, width : float, height : float) : void
Brush
-width : int
-height : int
-type : shapeType
-color : Color
+setColor(c : Color) : void
+setWdth(w : int) : void
+setHeight(h : int) : void
+setType(t : shapeType) : void
+paint(x : px, y : px) : void

## 2.6 GUI Functions

-ActionBar()

A build in class of Android which is used to hold all the tools for creating a graffiti. Contains buttons like: Shape, Brush, Pen, Line, color, etc. Each button will contain a spinner which will list all possible shapes if user clicks on shape button, or size of different brushes when user clicks brush button.

-Animation()

A build in class of Android which is used for all error warnings. Dialog box will show up in middle of the screen whenever user made a mistake action that the program is not allowed to perform, and the dialog box will disappear/ fade away automatically after a set time period.

-Texture()

A class that  contains all information about the augmentation metadata. Whenever a target is recognized from the screen, its corresponding metadata will be stored in the texture class. Metadata will be loaded into a bitmap in order to draw it on the target.

-onClick()

This is a build in method of Android which will perform different operations depending on which button has being clicked on the current view. Different intent will be called in order to switch between different screen once the corresponding buttons are clicked.

# 3 Vuforia

## 3.1 Opening the camera and initializing Vuforia

### 3.1.1 Overview

By opening the viewer the user can:

- View targets through the camera
- Track targets and view the resulting augmentation

On initialization the viewer or CloudReco activity:

- Initializes the needed variables in the java code
- Initializes the interface between the C and java code
- Initializes the an opengl renderer and starts its operation
- Initializes the needed veriables in the C code
- Initializes the camera
- Initializes the QCAR library
- Initializes the tracking of targets
- Initializes the rendering of tracked targets over the camera view

### 3.1.2 OnCreate()

On creation the CloudReco activity begins the initialization of the camera and augmentation by calling the function to update the application status to the initialization mode.

### 3.1.3 initApplication()

The first step in initialization mode is to check what the screen orientation is and then save it as a variable. The updateActivityOrientation function is called which gets the current configuration of the screen and then sets flags to indicate the current orientation of the screen. The function storeScreenDimensions is also called which queries the screen width and height and then stores that in variables too.

### 3.1.4 InitQCARTask()

The QCAR library is what the code uses to access the vuforia functions. To begin initialization of the QCAR library a new InitQCARTask class is created and then executed. This class calls the QCAR.init() function and then loops, checking to see if it is finished, until it is. If the library fails to load then an error is returned and the application exits.

### 3.1.5 initTracker()

InitTracker is the first native C function to be called from the java code. The initTracker function in CloudReco.cpp calls the QCAR::TrackerManager::getInstance() function to get a reference to the current trackerManager. From there the initTracker function from the trackerManager class is called which initializes an imageTracker as part of the trackerManager and then returns the address of the imageTracker. This is stored in a pointer which is then checked to make sure it contains an address, indicating the imageTracker was properly created. The function returns 1 on success and 0 on failure.

### 3.1.6 initAppicationAR()

If initTracker returns success the activity moves on to initializing the augmented reality.  To do this it calls the initApplicationAR function which in turn calls the native initApplicationNative function while passing it the screen width and height.

#### 3.1.6.1 initApplicationNative()

This native function starts off by calling QCAR::setHint to set the number of simultaneous image targets. It then uses the env->GetObjectClass() function to return a handle to the CloudReco.java class which is stored to allowed access to the java functions and classes from the C. The QCAR::registerCallback() function is then used to tell the QCAR library that whenever a tracking cycle is finished and a new AR state is available the updateCallback function should be called. The function then stores the screen width and height in a QCAR::Vec2f vector variable. It initializes the individual handles to the showErrorMessage, createProductTexture and getProductTexture functions in the java code using the env->GetMethodID() function. Finally it initializes the flags that indicate the state of the program by calling the initStateVeriables() function.

#### 3.1.6.2 return from initApplicationNative()

On returning from initApplicationNative the initApplicationAR function initializes a QCARSampleGLView class. This contains the rendered augumentation and initializes parts of the openGl glview. The corresponding openGl renderer is then created and added to the class. A layout is then inflated to overlay the camera and a loading bar is added to the visible part of the layout.

### 3.1.7 InitCloudRecoTask()

The next step in the initialization is to begin the searching for targets. To do this an InitCloudRecoTask is executed. This class on execution runs the Native initCloudReco() function in the background.

#### 3.1.7.1 initCloudReco()

This native function starts off by getting a reference to the tracker manager with QCAR::TrackerManager::getInstance(). It then uses this reference and the trackerManager.getTracker() function to get the address of the relevant imageTracker. It stores this in a pointer and then calls getTargetFinder on this pointer. Once the target finder is stored in another pointer it is used to call the startInit() function with the relevant access keys of the cloud database. Once finished this is checked to make sure it properly initializes the target finder with the clouds target information. The result code is returned.

#### 3.1.7.2 return from initCloudReco()

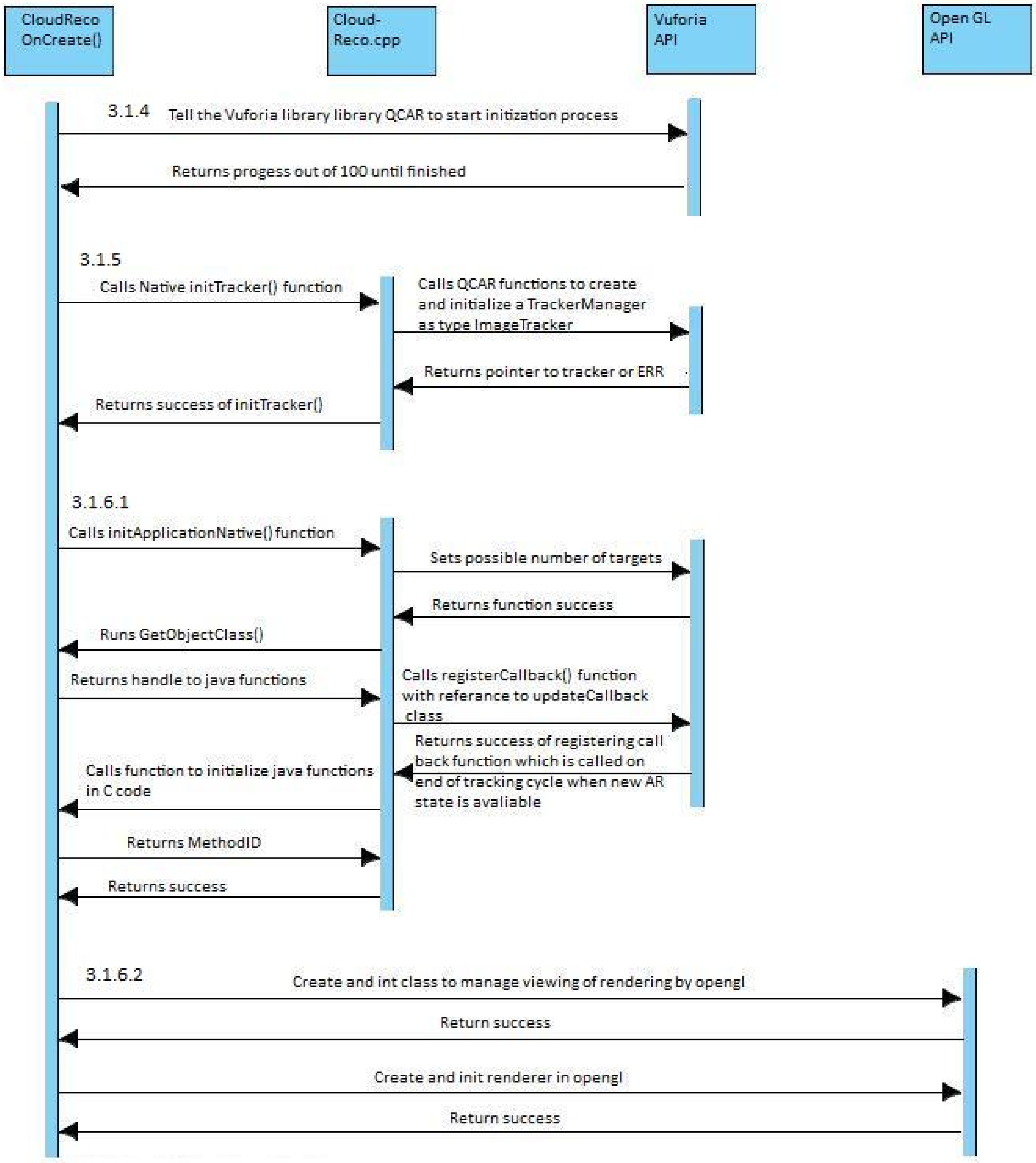
On return from initCloudReco the InitCloudRecoTask checks to make sure it returned correctly and sends out an error while the application exits if not.

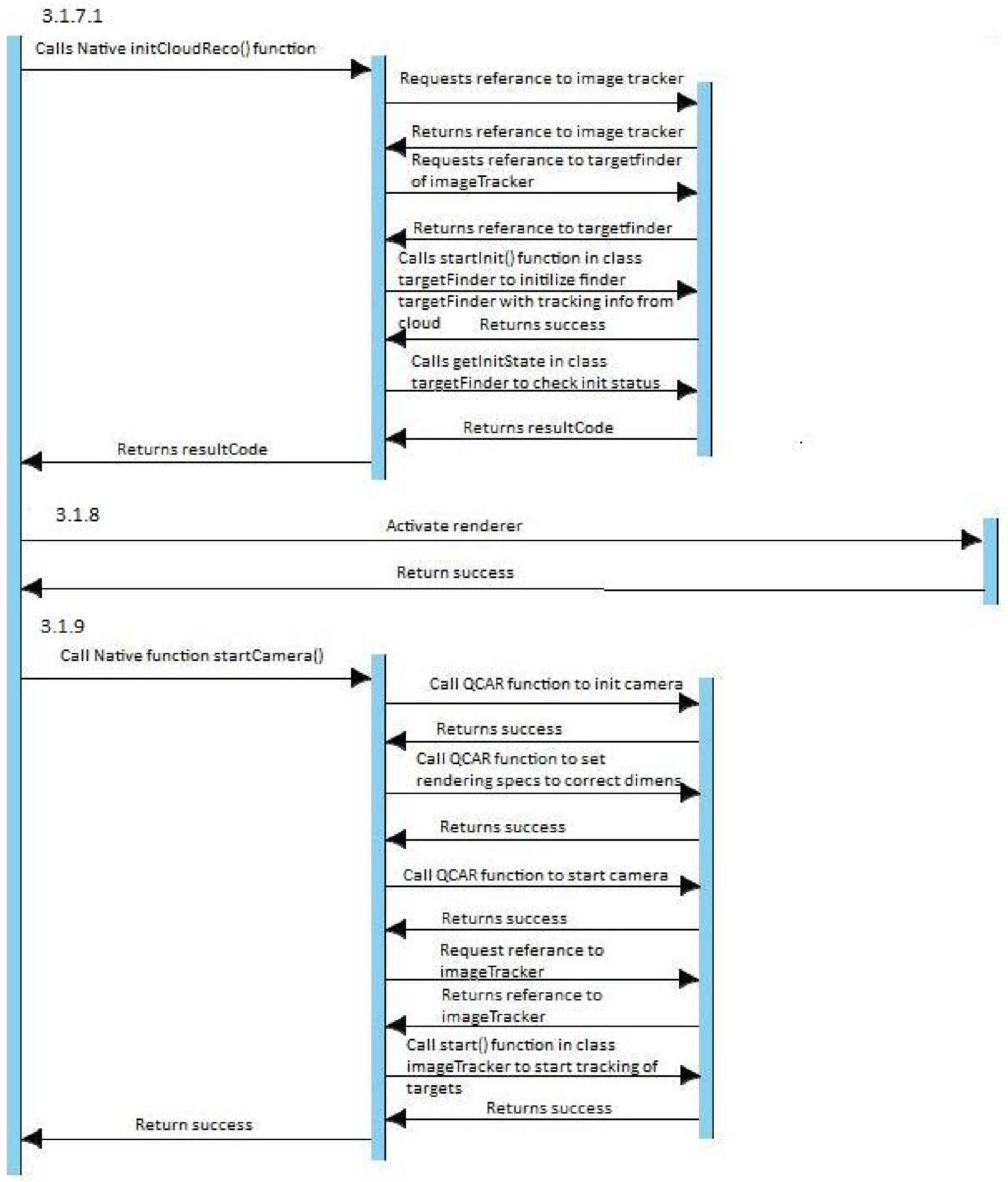
### 3.1.8 Activating renderer and adding content view

On returning from the InitCloudRecoTask the activity activates the renderer in the aforementioned QCARSampleGLView class. It then adds the openGL surface view to the view screen with the addContentView function.

### 3.1.9 startCamera()

The final step in initializing the camera and augmentation view is starting the camera. This is done with the startCamera native function. This function begins by using QCAR::CameraDevice::getInstance().init() to initialize the device. It then calls the configureVideoBackground() function. This begins by getting a reference to the camera device with QCAR::CameraDevice::getInstance(). It uses this to store the video mode of the camera. It then initializes a bunch of camera variables including the height and width of the screen based on the found orientation of the device. On return from the configureVideoBackground function the default video mode of the camera device is set. The camera is started with QCAR::CameraDevice::getInstance().start(). If this fails the function returns. Once this is done a reference to the trackerManager and then imageTracker is obtained. The imageTracker reference is used with the start() function in the class to start the image trackers tracking. A reference to the target finder is then obtained and the target recognition started with the startRecognition() function in the targetFinder class. This concludes the initialization of the camera and augmentation.


CloudReco OnCreate()
Cloud-Reco.cpp
Vuforia API
Open GL API
3.1.4
Tell the Vuforia library library QCAR to start initization process
Returns progess out of 100 until finished
3.1.5
Calls Native initTracker() function
Calls QCAR functions to create and initialize a TrackerManager as type ImageTracker
Returns pointer to tracker or ERR
Returns success of initTracker()
3.1.6.1
Calls initApplicationNative() function
Sets possible number of targets
Returns function success
Runs GetObjectClass()
Returns handle to java functions
Calls registerCallback() function with referance to updateCallback class
Returns success of registering call back function which is called on end of tracking cycle when new AR state is avaliable
Calls function to initialize java functions in C code
Returns MethodID
Returns success
3.1.6.2
Create and int class to manage viewing of rendering by opengl
Return success
Create and init renderer in opengl
Return success


3.1.7.1
Calls Native initCloudReco() function
Requests referance to image tracker
Returns referance to image tracker
Requests referance to targetfinder of imageTracker
Returns referance to targetfinder
Calls startInit() function in class targetFinder to initilize finder targetFinder with tracking info from cloud
Returns success
Calls getInitState in class targetFinder to check init status
Returns resultCode
Returns resultCode
3.1.8
Activate renderer
Return success
3.1.9
Call Native function startCamera()
Call QCAR function to init camera
Returns success
Call QCAR function to set rendering specs to correct dimens
Returns success
Call QCAR function to start camera
Returns success
Request referance to imageTracker
Returns referance to imageTracker
Call start() function in class imageTracker to start tracking of targets
Returns success
Return success

## 3.2 Native C++ code rendering augmentation in new frame

### 3.2.1 Overview

The native C++ code for each frame:

- Recognizes targets that need to be tracked
- Starts tracking them
- Refreshes the camera background and background information variables
- Generates textures for each target
- Renders the textures over the targets in the appropriate orientation

### 3.2.2 updateCallback::QCAR_onUpdate()

This function is called every time a tracking cycle is finished and there is a new AR state available. First the address of the trackerManager is requested with QCAR::TrackerManager::getInstance() and stored. From there the address of the image tracker is requested with trackerManager.getTracker (QCAR::Tracker::IMAGE_TRACKER)) and stored in a pointer. From here the targetFinder is requested from the imageTracker and then stored in a pointer as well. A functions of the targetFinder class are called to updatethe search results for trackables, get the count of these trackables and then access the address of each trackable reference independently. The address of each trackable is stored in a pointer and then used to create new trackers to follow each trackable, and also access metadata about the trackable. For each trackable the java function createProductTexture() called and passed this metadata.

### 3.2.2.1 createProductTexture java function

The metadata contains the URL of a database which connects tracking targets with the actual augumentations to be rendered overtop of them. The image is downloaded and then converted into a bitmap with the BitmapFactory library. This bitmap is then converted into a buffer of pixels and which is then loaded into a texture format. Finally productTextureIsCreated() is called to set a flag in the C code to indicate the texture is created.

### 3.2.3 updateRenderView()

After the trackers have been started and the textures created the C code runs the updateRenderView() function in CloudReco.java. This function starts off by storing the current screen rotation. From there it stores the screen dimensions as well. It then calls the native function updateRendering() and passes it the screen height and width.

### 3.2.3.1 updateRendering()

The updateRendering function stores the screen dimensions in the C code and calls configureVideoBackground(). This begins by getting a reference to the camera device with QCAR::CameraDevice::getInstance(). It uses this to store the video mode of the camera. It then initializes a bunch of camera variables including the height and width of the screen based on the found orientation of the device.

### 3.2.3.2 Return from updateRendering()

On return from updateRendering the updateRenderView function calls the native setProjectionMatrix() function. This function gets a reference to the current camera calibration with QCAR::CameraDevice::getInstance().getCameraCalibration(). With this it gets a reference to the projection matrix and stores it like so: projectionMatrix = QCAR::Tool::getProjectionGL(cameraCalibration, 2.0f, 2500.0f). Projection matrix is simply stored in a global veriable for future use. Upon return from setProjectionMatrix updateRendering exits.

### 3.2.4 renderFrame()

The next step in the C code is to run the function renderFrame(). This function starts off by storing the current state of the application and storing it as the begging of the rendering state. The video background is then drawn with QCAR::Renderer::getInstance().drawVideoBackground(). From there the function generateProductTextureInOpenGL() is called.

### 3.2.4.1 generateProductTextureInOpenGL()

This function starts off by getting a reference to the texture object created in java. It then uses Texture::create to convert it to an openGL texture. The texture is then generated in openGL using several commands.

### 3.2.4.2 return to renderFrame()

Upon returning to renderFrame it runs state.getNumTrackableResults(). The function then loops doing the following for every trackable result. It first gets a reference to each trackable result with state.getTrackableResult(). It then gets the pose of each result and converts it to a modelViewMatrix with QCAR::Tool::convertPose2GLMatrix() which it then stores for later use.

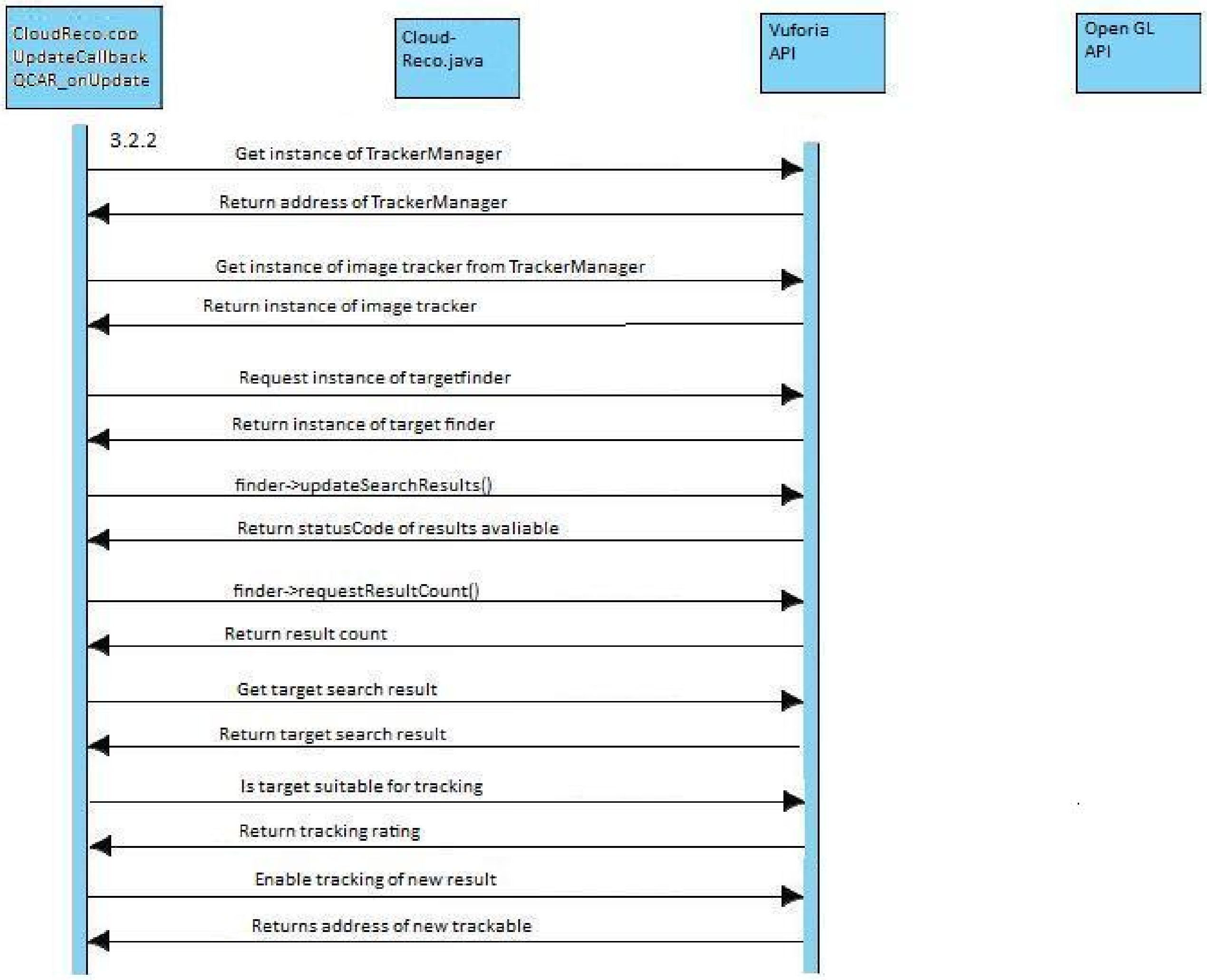
Next the address of the trackableResult it passed to renderAugmentation().

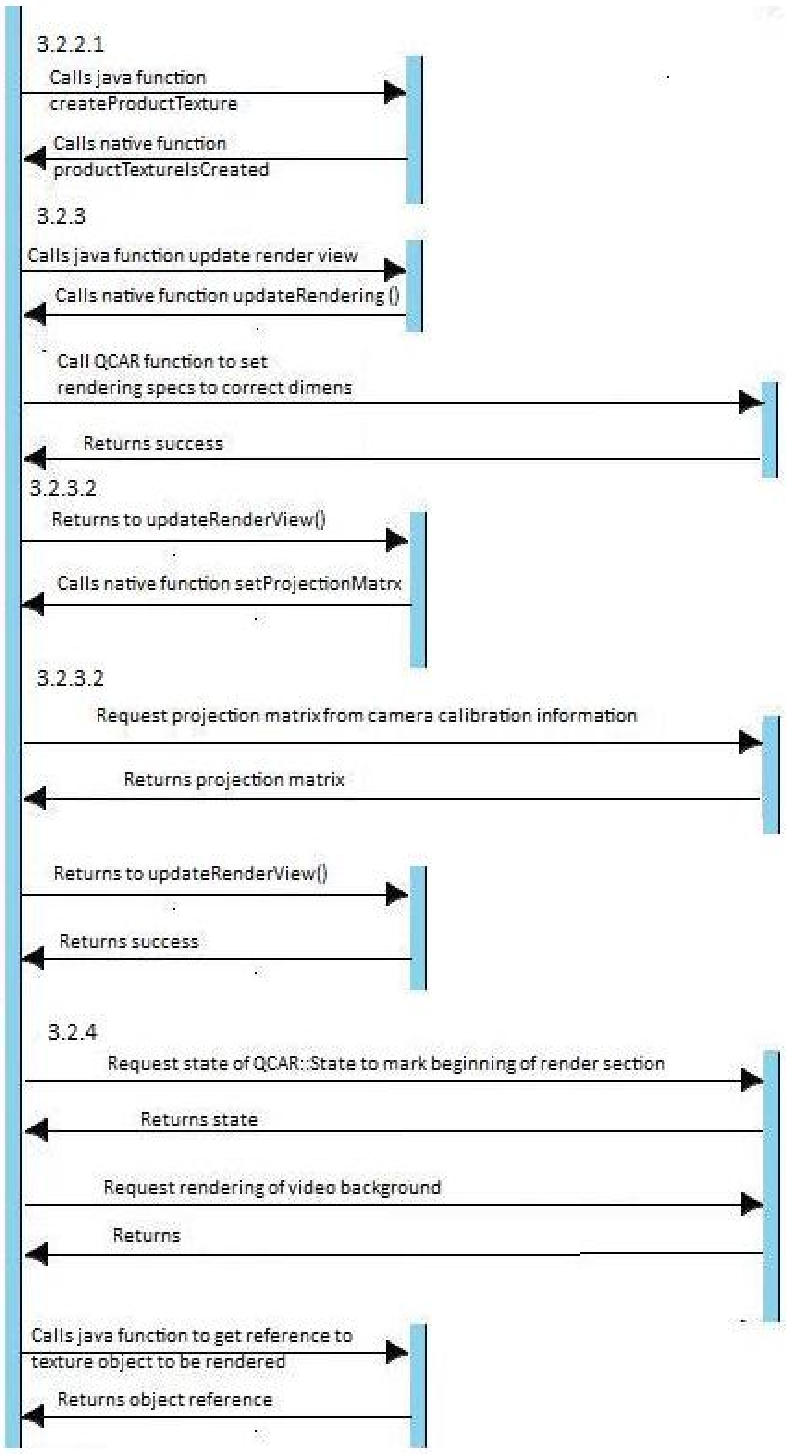
#### 3.2.4.3 renderAugumentation()

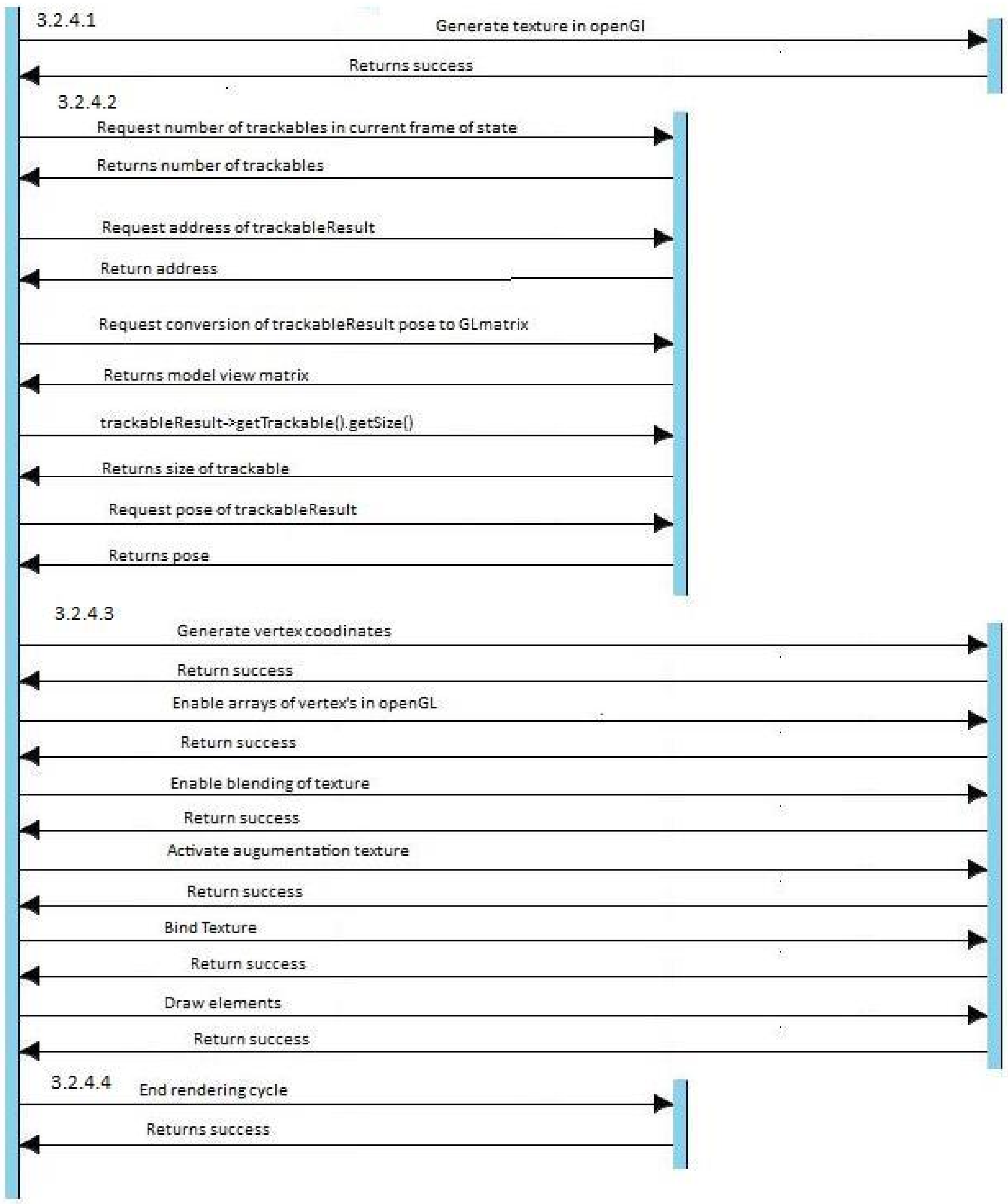
This function starts off by scaling the pose of the plane relative to the target. It then applies the necessary 3d transformations to the plane. From there it generates vertex's for the necessary points, enables these in openGL and then enables blending in openGL. It then activates the relevant texture, binds it and draws the elements. Finally it cleans up all the openGL references.

#### 3.2.4.4 return to renderFrame()

Upon return to renderFrame the function disables the various vertexArrays used in the rendering and then signals the end of the rendering with QCAR::Renderer::getInstance().end().


CloudReco.cpp
UpdateCallback
QCAR_onUpdate
Cloud-Reco.java
Vuforia API
Open GL API
3.2.2
Get instance of TrackerManager
Return address of TrackerManager
Get instance of image tracker from TrackerManager
Return instance of image tracker
Request instance of targetfinder
Return instance of target finder
finder->updateSearchResults()
Return statusCode of results avaliable
finder->requestResultCount()
Return result count
Get target search result
Return target search result
Is target suitable for tracking
Return tracking rating
Enable tracking of new result
Returns address of new trackable


3.2.2.1
Calls java function createProductTexture
Calls native function productTextureIsCreated
3.2.3
Calls java function update render view
Calls native function updateRendering ()
Call QCAR function to set rendering specs to correct dimens
Returns success
3.2.3.2
Returns to updateRenderView()
Calls native function setProjectionMatrx
3.2.3.2
Request projection matrix from camera calibration information
Returns projection matrix
Returns to updateRenderView()
Returns success
3.2.4
Request state of QCAR::State to mark beginning of render section
Returns state
Request rendering of video background
Returns
Calls java function to get reference to texture object to be rendered
Returns object reference


3.2.4.1
Generate texture in openGl
Returns success
3.2.4.2
Request number of trackables in current frame of state
Returns number of trackables
Request address of trackableResult
Return address
Request conversion of trackableResult pose to GLmatrix
Returns model view matrix
trackableResult->getTrackable().getSize()
Returns size of trackable
Request pose of trackableResult
Returns pose
3.2.4.3
Generate vertex coodinates
Return success
Enable arrays of vertex's in openGL
Return success
Enable blending of texture
Return success
Activate augumentation texture
Return success
Bind Texture
Return success
Draw elements
Return success
3.2.4.4
End rendering cycle
Returns success

# 4 Facebook

## 4.1 Login Introduction

One of the main features of UCSB 360 is the ability to log in with Facebook. By integrating with the Facebook Android SDK 3.0, UCSB 360 allows users to perform basic Facebook related tasks.

### 4.1.1 Login Overview

By clicking on the Facebook login button, the user can:

- Type in user email and  password to login to facebook
- proceed to view targets through camera as shown in part 3

On initialization integrating with Facebook SDK:

- Store user information(Name, Birthday, Location, Friend List)
- Initialize User info
- Initialize Friend List

### 4.1.2
### onCreate()

This method will initialize the button used on the login page.

### 4.1.3
### createSession()

Creates a facebook session. If no session is present, the method will create a new one and pass return a session object.

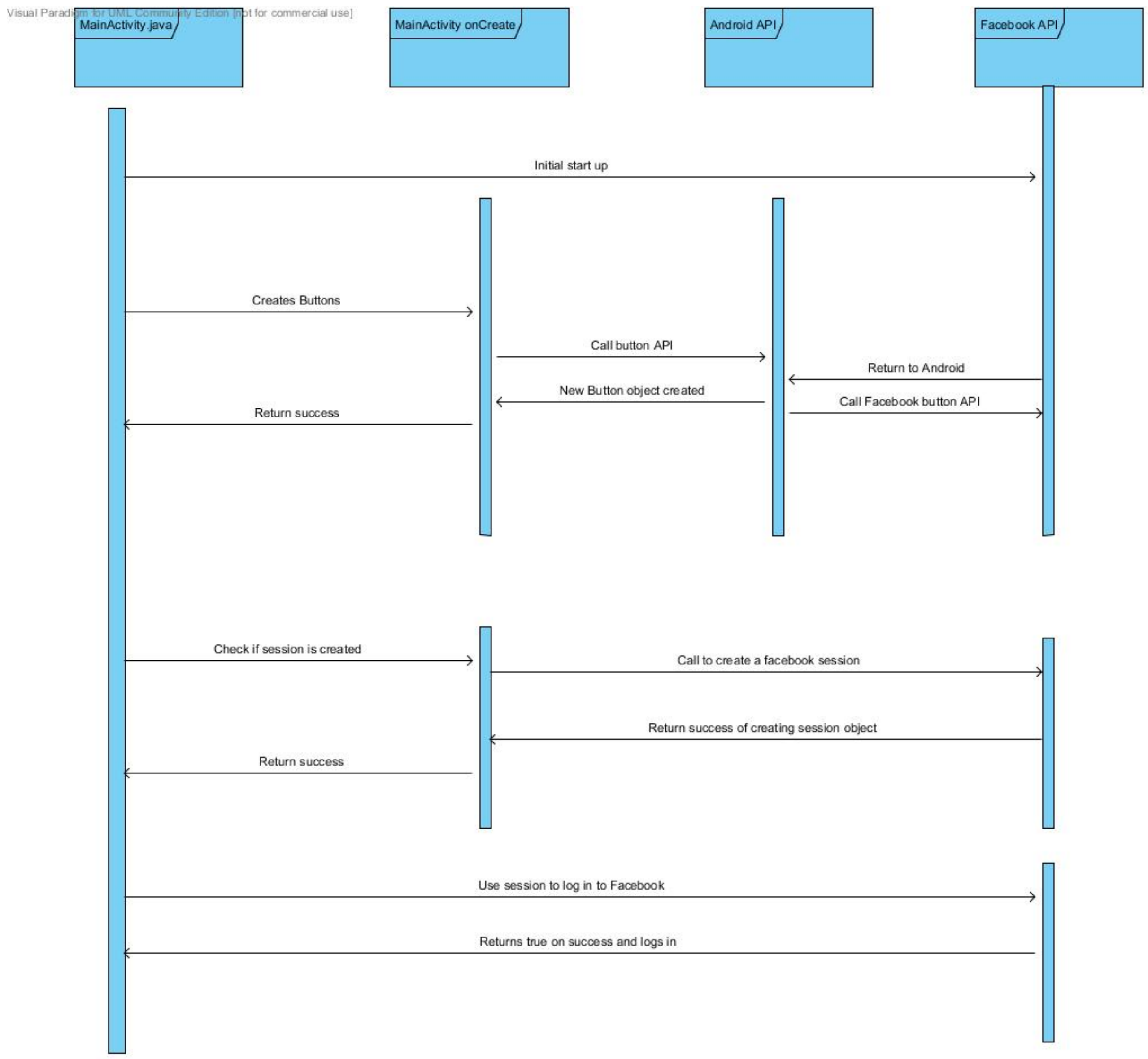
### 4.1.4
### onActivityResult()

This method is called after the user logs in facebook, it will direct the user to the targetview. If the user is not logged in, nothing will happen.
By calling the facebook onActivityResult to override the Android onActivityResult, the method is able to get the current state of the facebook token.

### 4.1.5
### onRestoreInstanceState
This method is called after onStart(), when the activity is being re-initialized. This method is a default implementation by the Android SDK.

### 4.1.6
### onSaveInstanceState
This method is called before an activity is being killed, so that the state can be restored in the next onCreate() method call. This method is a default implementation by the Android SDK.

### 4.1.7
### onClick()
This is a built in method by Android. This method is used to implement the functionality of the two buttons on the main menu. Each of the buttons will direct the user to a different UI page. Specifically, login in with facebook will attempt to connect to facebook, it will create an error message if the user stops the login process anytime through the login process.


Visual Paradigm for UML Community Edition [not for commercial use]
MainActivity.java
MainActivity onCreate
Android API
Facebook API
Initial start up
Creates Buttons
Call button API
Return to Android
New Button object created
Call Facebook button API
Return success
Check if session is created
Call to create a facebook session
Return success of creating session object
Return success
Use session to log in to Facebook
Returns true on success and logs in

# 5 Database Interface and the Mid-Level Module

## 5.1 Overview

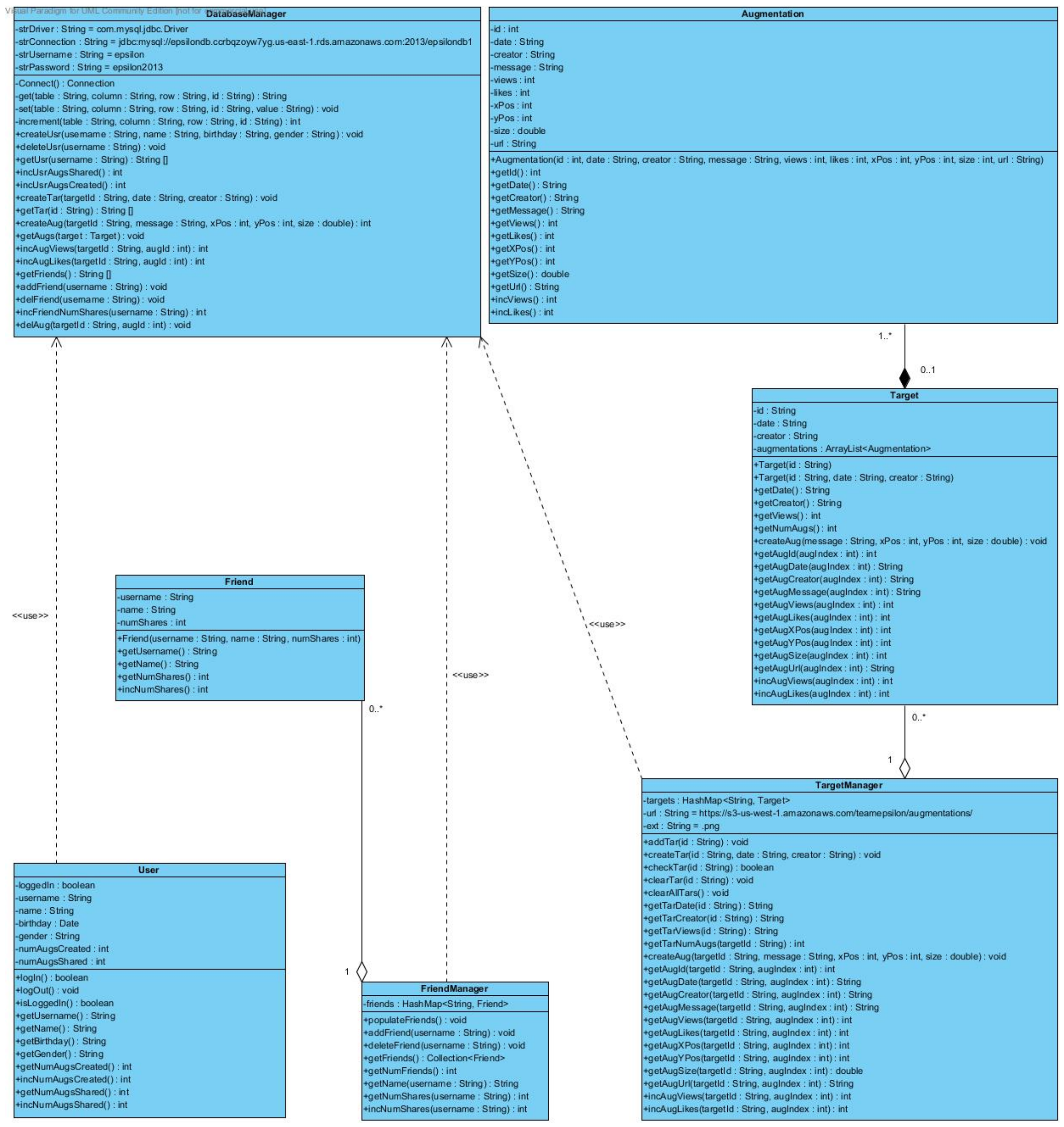
Another major component in the UCSB 360 system is the database. The database stores all information for users, friends, targets, and augmentations. It is imperative that the application moves data back and forth between the application and the database as efficiently as possible. A group of mid-level interface classes has been created to achieve this goal.

The Friend, Target, and Augmentation classes are nothing more than simple storage containers for data that is loaded from the database.

Each Manager class consists entirely of static members and methods, which mitigates the need for instantiation. These manager classes typically contain collections of instantiated classes that hold other data. Most importantly, they serve to mediate the transfer of data between the database and the rest of the system.

The User, FriendManager, and TargetManager classes act as an interface between the loaded data and the rest of the system. This means that access to the DatabaseManager, Friend, Target, and Augmentation classes should be entirely contained within this module.

JavaDoc documentation for these modules are included in a separate zip file.


Visual Paradigm for UML Community Edition [not for commercial use]
DatabaseManager
-strDriver : String = com.mysql.jdbc.Driver
-strConnection : String = jdbc:mysql://epsilondb.ccrbqzoyw7yg.us-east-1.rds.amazonaws.com:2013/epsilondb1
-strUsername : String = epsilon
-strPassword : String = epsilon2013
-Connect() : Connection
-get(table : String, column : String, row : String, id : String) : String
-set(table : String, column : String, row : String, id : String, value : String) : void
-increment(table : String, column : String, row : String, id : String) : int
+createUsr(username : String, name : String, birthday : String, gender : String) : void
+deleteUsr(username : String) : void
+getUsr(username : String) : String []
+incUsrAugsShared() : int
+incUsrAugsCreated() : int
+createTar(targetId : String, date : String, creator : String) : void
+getTar(id : String) : String []
+createAug(targetId : String, message : String, xPos : int, yPos : int, size : double) : int
+getAugs(target : Target) : void
+incAugViews(targetId : String, augId : int) : int
+incAugLikes(targetId : String, augId : int) : int
+getFriends() : String []
+addFriend(username : String) : void
+delFriend(username : String) : void
+incFriendNumShares(username : String) : int
+delAug(targetId : String, augId : int) : void
Augmentation
-id : int
-date : String
-creator : String
-message : String
-views : int
-likes : int
-xPos : int
-yPos : int
-size : double
-url : String
+Augmentation(id : int, date : String, creator : String, message : String, views : int, likes : int, xPos : int, yPos : int, size : int, url : String)
+getId() : int
+getDate() : String
+getCreator() : String
+getMessage() : String
+getViews() : int
+getLikes() : int
+getXPos() : int
+getYPos() : int
+getSize() : double
+getUrl() : String
+incViews() : int
+incLikes() : int
1..*
0..1
Target
-id : String
-date : String
-creator : String
-augmentations : ArrayList<Augmentation>
+Target(id : String)
+Target(id : String, date : String, creator : String)
+getDate() : String
+getCreator() : String
+getViews() : int
+getNumAugs() : int
+createAug(message : String, xPos : int, yPos : int, size : double) : void
+getAugId(augIndex : int) : int
+getAugDate(augIndex : int) : String
+getAugCreator(augIndex : int) : String
+getAugMessage(augIndex : int) : String
+getAugViews(augIndex : int) : int
+getAugLikes(augIndex : int) : int
+getAugXPos(augIndex : int) : int
+getAugYPos(augIndex : int) : int
+getAugSize(augIndex : int) : int
+getAugUrl(augIndex : int) : String
+incAugViews(augIndex : int) : int
+incAugLikes(augIndex : int) : int
<<use>>
Friend
-username : String
-name : String
-numShares : int
+Friend(username : String, name : String, numShares : int)
+getUsername() : String
+getName() : String
+getNumShares() : int
+incNumShares() : int
<<use>>
<<use>>
0..*
0..*
1
TargetManager
-targets : HashMap<String, Target>
-url : String = https://s3-us-west-1.amazonaws.com/teamepsilon/augmentations/
-ext : String = .png
+addTar(id : String) : void
+createTar(id : String, date : String, creator : String) : void
+checkTar(id : String) : boolean
+clearTar(id : String) : void
+clearAllTars() : void
+getTarDate(id : String) : String
+getTarCreator(id : String) : String
+getTarViews(id : String) : String
+getTarNumAugs(targetId : String) : int
+createAug(targetId : String, message : String, xPos : int, yPos : int, size : double) : void
+getAugId(targetId : String, augIndex : int) : int
+getAugDate(targetId : String, augIndex : int) : String
+getAugCreator(targetId : String, augIndex : int) : String
+getAugMessage(targetId : String, augIndex : int) : String
+getAugViews(targetId : String, augIndex : int) : int
+getAugLikes(targetId : String, augIndex : int) : int
+getAugXPos(targetId : String, augIndex : int) : int
+getAugYPos(targetId : String, augIndex : int) : int
+getAugSize(targetId : String, augIndex : int) : double
+getAugUrl(targetId : String, augIndex : int) : String
+incAugViews(targetId : String, augIndex : int) : int
+incAugLikes(targetId : String, augIndex : int) : int
User
-loggedIn : boolean
-username : String
-name : String
-birthday : Date
-gender : String
-numAugsCreated : int
-numAugsShared : int
+logIn() : boolean
+logOut() : void
+isLoggedIn() : boolean
+getUsername() : String
+getName() : String
+getBirthday() : String
+getGender() : String
+getNumAugsCreated() : int
+incNumAugsCreated() : int
+getNumAugsShared() : int
+incNumAugsShared() : int
1
FriendManager
-friends : HashMap<String, Friend>
+populateFriends() : void
+addFriend(username : String) : void
+deleteFriend(username : String) : void
+getFriends() : Collection<Friend>
+getNumFriends() : int
+getName(username : String) : String
+getNumShares(username : String) : int
+incNumShares(username : String) : int

## 5.2 Database Interface

### 5.2.1 DatabaseManager

The DatabaseManager class mediates all traffic to and from the database. It contains the driver and connection information needed to connect to the MySQL database. It implements many helper functions that allow the other Manager classes to load information from the database.

## 5.3 User Information Management

### 5.3.1 User

The User class is a static class that holds information about the user that is currently logged in. When the user logs in, the User object makes function calls to the DatabaseManager to load user information. It then activates the FriendManager.

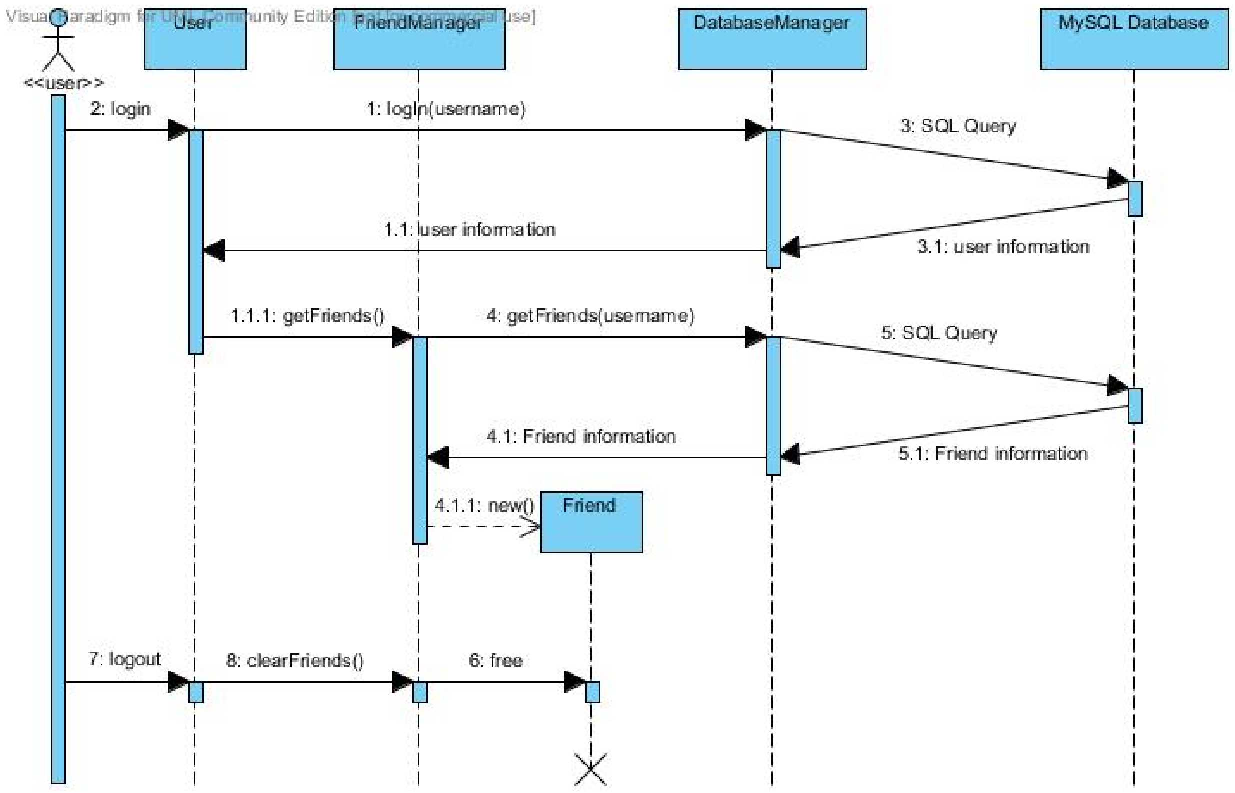
### 5.3.2 FriendManager

The FriendManager class is a static class that manages all friends that the user has added. It contains a list of Friend objects, which it populates via function calls to the DatabaseManager and by adding friends at runtime.

### 5.3.3 Friend

The Friend class is a simple container that stores all information related to a user's friend. It is instantiated when loading a user's friends list on startup and when a user adds a new friend during runtime.

### 5.3.4 Loading user and friend information



## 5.4 Target Management

### 5.4.1 TargetManager

The TargetManager class is a static class that acts as an interface between loaded targets and the rest of the system. This class interacts with the DatabaseManager to load target information from the database. Additionally, it provides functions that allow other modules to retrieve information for loaded targets.

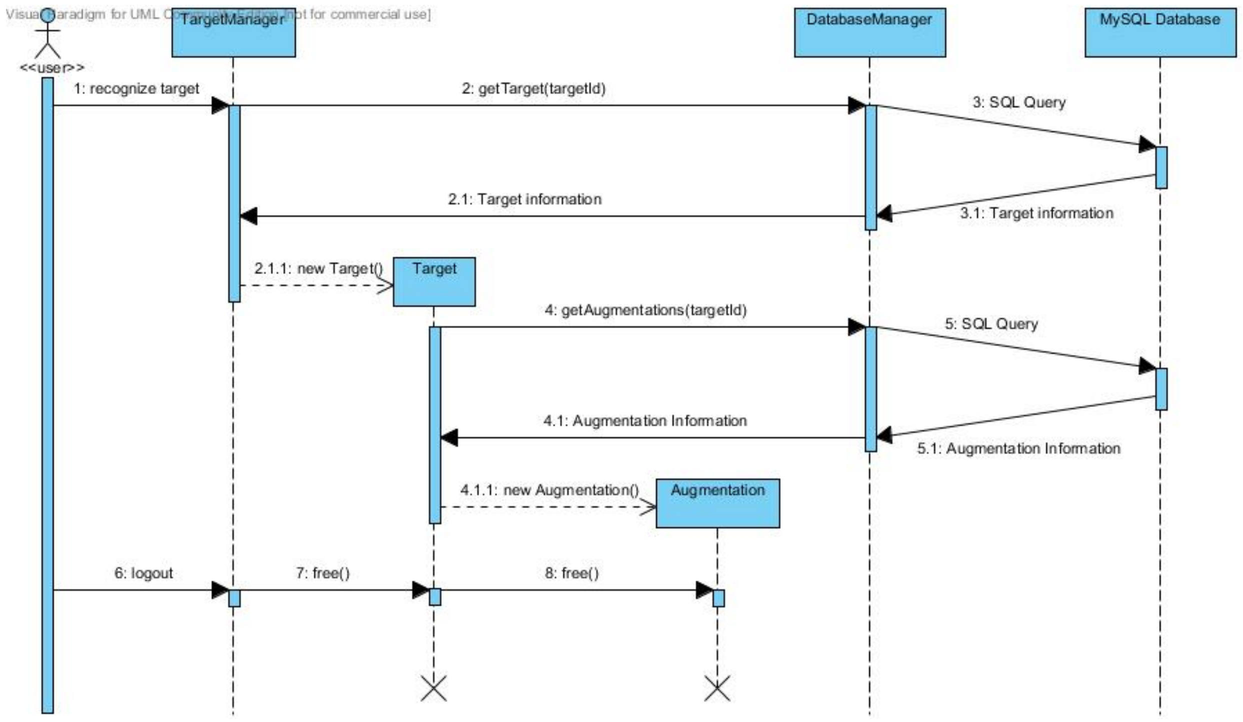
### 5.4.2 Target

The Target class holds information pertaining to a target that has been recognized by the Vuforia module. This includes a list of the augmentations associated with the target.

### 5.4.3 Augmentation

The Augmentation class holds information pertaining to a single augmentation. Most importantly, this class holds the URL for the augmentation image that is stored in the Amazon Web Services S3 file system. The Vuforia module will download this file and render it on the image target.

### 5.4.4 Loading target and augmentation information

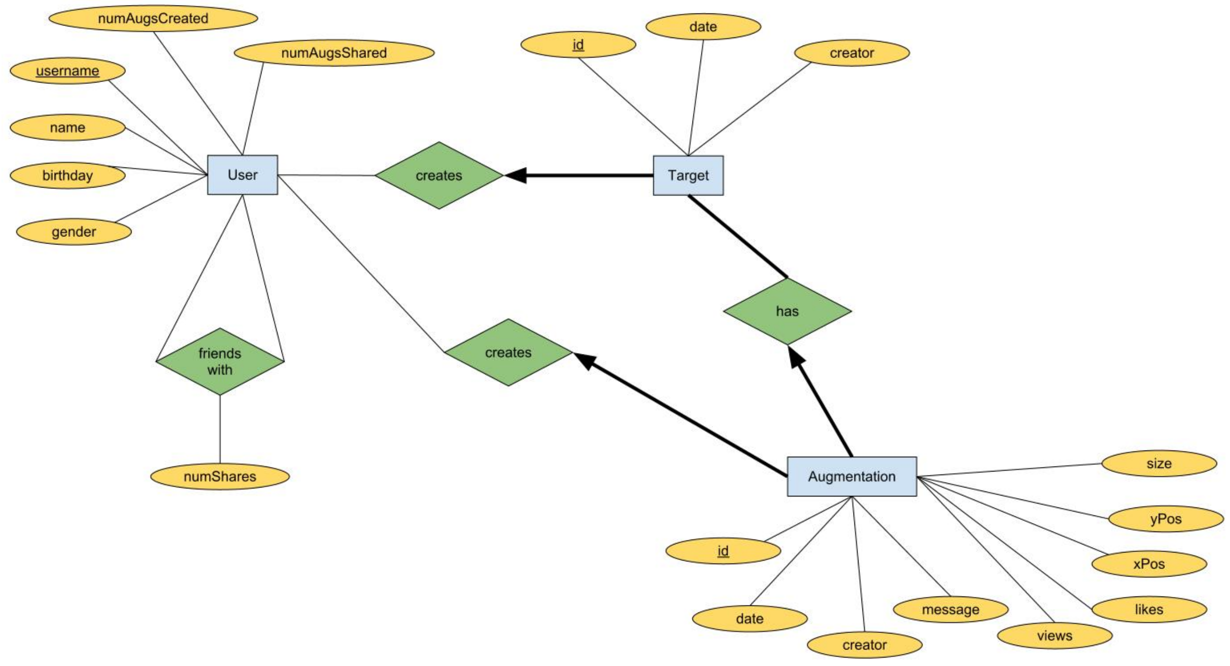


# 6 Database

## 6.1 Overview

The UCSB 360 application shall interact with a MySQL database hosted on the Amazon Web Services Relational Database System. This database will store information about users, targets, and augmentations. Specific details about the structure of the database are shown in Section 6.2.

## 6.2 ER Diagram


numAugsCreated
username
numAugsShared
name
birthday
gender
User
creates
Target
id
date
creator
has
friends with
numShares
creates
Augmentation
size
yPos
xPos
likes
views
message
creator
date
id