Software Design Document

Card Czar Android App

CMSC 495

Group 2 Final Project

Kenneth Mikkalson

Alton Hinton

Shawn Henson

Sarah Holley

Tara Lawson

Richard Wysong

Table of Contents

[1 Introduction 4](#_Toc429913069)

[1.1 System Purpose 4](#_Toc429913070)

[1.2 System Scope 4](#_Toc429913071)

[1.3 Intended Audience 4](#_Toc429913072)

[1.4 Definitions 4](#_Toc429913073)

[1.5 References 5](#_Toc429913074)

[1.6 Document Overview 5](#_Toc429913075)

[2 System Description 5](#_Toc429913076)

[2.1 System Description 5](#_Toc429913077)

[3 System Architecture 6](#_Toc429913078)

[3.1 High Level Activities and Interfaces 6](#_Toc429913079)

[3.1.1 Activities 6](#_Toc429913080)

[3.1.1.1 Main Activity 6](#_Toc429913081)

[3.1.1.2 Host Start Activity 6](#_Toc429913082)

[3.1.1.3 Join Game Activity 7](#_Toc429913083)

[3.1.1.4 Waiting Room Activity 7](#_Toc429913084)

[3.1.1.5 Game Start Activity 8](#_Toc429913085)

[3.1.1.6 Game Play Activity 8](#_Toc429913086)

[3.1.1.7 Security Considerations for All Activities 8](#_Toc429913087)

[3.1.2 Server 9](#_Toc429913088)

[3.1.2.1 Security considerations for Server Side Interface 9](#_Toc429913089)

[3.2 Architectural Styles and Patterns 9](#_Toc429913090)

[3.2.1 MVC Architecture Style 9](#_Toc429913091)

[3.2.2 Client Server Architecture 9](#_Toc429913092)

[3.3 Architectural Styles and Patterns 10](#_Toc429913093)

[3.3.1 Database Design 10](#_Toc429913094)

[3.4 Java Class Diagram 11](#_Toc429913095)

[3.5 Sequence Diagram 12](#_Toc429913096)

[3.5.1 User Login Sequence Diagram 12](#_Toc429913097)

[3.5.2 Host Creates Game Room Sequence Diagram 13](#_Toc429913098)

[3.5.3 Join Game Sequence Diagram 14](#_Toc429913099)

[3.5.4 Game Play Sequence Diagram 15](#_Toc429913100)

[3.6 Algorithm Design 16](#_Toc429913101)

[3.6.1 Algorithm Design – Intent communications 16](#_Toc429913102)

[3.6.2 Algorithm Design – SQL operations 18](#_Toc429913103)

[3.6.3 Algorithm Design – HTTP communication 19](#_Toc429913104)

[3.6.4 Algorithm Design – URLencoding 20](#_Toc429913105)

[3.6.5 Algorithm Design – List randomization 21](#_Toc429913106)

[3.6.6 Algorithm Design – Server polling threads 21](#_Toc429913107)

[3.6.7 Algorithm Design – Handlers 23](#_Toc429913108)

[3.7 User and Database Interface Design 24](#_Toc429913109)

# 1 Introduction

## 1.1 System Purpose

The software design document (SDD) provides software design details for the Card Czar Android Application. The expected audience is the software development, analyst, and program management teams with a vested interest in the software design phase of development. It will serve as the expected guideline for all software development. The software design document is written in a technical fashion for software developers. The software design document lays out how the system will be developed and constructed. Refer to the Software Project Plan (SPP) to review the overall system goals of the Card Czar Application.

## 1.2 System Scope

The software design document contains a complete software design of the Card Czar Application. The internal architecture and user interface will be described in detail. The SDD is the guideline for the software system. The SDD will ensure that the software is designed in accordance with the requirements and the SPP document previously developed.

## 1.3 Intended Audience

This document is intended for the developers of the Card Czar Application.

## 1.4 Definitions

|  |  |
| --- | --- |
| Term | Definition |
| MVC | Model View Controller is a software design pattern that separates presentation of data from interaction with the data. |
| SDD | Software Design Document |
| SPP | Software Project Plan |
| LAMP | Linux, Apache, MySQL, PHP products |

## 1.5 References

* IEEE Software Engineering Standards Committee, “IEEE Std 1016 1998, IEEE Recommended Practice for Software Design Descriptions”, 23 September 1998.
* Project Plan, Card Czar Android Application
* Test Plan, Card Czar Android Application

## 1.6 Document Overview

The document is intended as a detailed description of the structure and design of the software comprising the Card Czar Application.

# 2 System Description

## 2.1 System Description

The Card Czar Android application is a spin-off of the popular board game Apples to Apples. The purpose of the software design document (SDD) is to identify key aspects of software verification and requirements validation.

The Card Czar is an android application multiplayer card game that can deliver hours of fun.  It allows a user to create a new game room and invite friends to play with them. For a game to begin there must be at least 3 players. Once a game is started each player is dealt at random a hand of response cards that will contain a noun (or response) on them. After all players are given the response cards a dealer will be randomly chosen to start the game. The dealer (Card Baron) will draw a bait card that consists of an adjective (or other text to be responded to). Each player then selects from their cards a noun that they think the dealer will choose as the best response to the question. Once all responses are received the dealer (Card Baron) then picks the answer they like the best and the player whose response it was gets a point and becomes the dealer (Card Baron) for the next round. This continues until a player has reached 5 points and becomes the winner (the Card Czar).

Maximum players: 6

Minimum players: 3

Bait cards: (Adjective/Question Options)

Response cards: (Noun/Answer Options)

# 3 System Architecture

## 3.1 High Level Activities and Interfaces

## 3.1.1 Activities

## 3.1.1.1 Main Activity

The Main Activity Component will provide the user the option to join an existing room or to create and host a new game.

|  |  |
| --- | --- |
| Requirement number | SRS Requirement |
| 3.1.1.1.1 | The Main Activity shall be the first page that users will see upon starting the application |
| 3.1.1.1.2 | The main activity page shall have a Join Existing Game button |
| 3.1.1.1.3 | The main activity page shall have a Host New Game button |
| 3.1.1.1.4 | Selecting the “Host New Game” button shall launch the Host Start Activity |
| 3.1.1.1.5 | Selecting the “Join Game” button shall launch the User Start Activity |

## 3.1.1.2 Host Start Activity

The host activity will allow users to define the room of the game to be created and the name of the hosting player. The name the player chooses will be used as their display name during game play.

|  |  |
| --- | --- |
| Requirement number | SRS Requirement |
| 3.1.1.2.1 | The Host Start Activity shall be the page the user sees when they choose to host a new Card Czar game. |
| 3.1.1.2.2 | The host start activity shall have a text field to enter the name of the room associated with the new game. |
| 3.1.1.2.3 | The host start activity shall have a text field to enter the host’s user name. |
| 3.1.1.2.4 | The host start activity shall have a button “Start New Room” button. |
| 3.1.1.2.5 | Selecting the “Start New Room” button shall create a new database instance for the room name entered. |
| 3.1.1.2.6 | The host start activity will display an error message to the user if a database with the requested room name already exists. |
| 3.1.1.2.7 | The host start activity will display an error message to the user if any error is encountered creating or setting up the database. |
| 3.1.1.2.8 | The host start activity will launch the Room Activity if the database was successfully created and set up. |

## 3.1.1.3 Join Game Activity

|  |  |
| --- | --- |
| Requirement number | SRS Requirement |
| 3.1.1.3.1 | The Join Game Activity shall be the page the user sees when they choose to join an existing Card Czar game. |
| 3.1.1.3.2 | The join game activity shall have a text field to enter the name of the room/game the user wants to join. |
| 3.1.1.3.3 | The join game activity shall have a text field to enter the user’s user name. |
| 3.1.1.3.4 | The join game activity shall have a button “Join Room” button. |
| 3.1.1.3.5 | Selecting the “Join Room” button shall attempt to add the user to the database for the room name entered. |
| 3.1.1.3.6 | The join game activity will display an error message to the user if the database for the room name entered does not exist. |
| 3.1.1.3.7 | The join game activity will display an error message to the user if the room has reached maximum capacity. |
| 3.1.1.3.8 | The join game activity will display an error message to the user if the room already has a user with the username entered. |
| 3.1.1.3.9 | The join game activity will launch the Waiting Room Activity if the user was successfully added to the database. |

## 3.1.1.4 Waiting Room Activity

|  |  |
| --- | --- |
| Requirement number | SRS Requirement |
| 3.1.1.4.1 | The Waiting Room Activity shall be the page the user sees while waiting for a game they have joined to start. |
| 3.1.1.4.2 | The waiting room activity shall have a display a message to the user that the game has not yet started. |
| 3.1.1.4.3 | The waiting room activity shall launch a listener that waits for a message that indicates the game been started. |
| 3.1.1.4.4 | The waiting room activity shall launch the Game Play Activity when the listener receives a message the game has started. |

## 3.1.1.5 Game Start Activity

|  |  |
| --- | --- |
| Requirement number | SRS Requirement |
| 3.1.1.5.1 | The Game Start Activity shall be the page the host user sees while waiting to start the game. |
| 3.1.1.5.2 | The game start activity shall display a list of user that have joined the room and are waiting for the game to start. |
| 3.1.1.5.3 | The game start activity shall have a “User Refresh” button. |
| 3.1.1.5.4 | The game start activity shall have a “Start Game” button. |
| 3.1.1.5.5 | The game start activity update the list of users, those that have joined or left the room since the last time the list was refreshed, when the “User Refresh” button is selected. |
| 3.1.1.5.6 | The game start activity will make the “Start Game” button active when the minimum number of game players is reached. |
| 3.1.1.5.7 | The game start activity shall launch the Game Play Activity when the “Start Game” button is selected. |

## 3.1.1.6 Game Play Activity

|  |  |
| --- | --- |
| Requirement number | SRS Requirement |
| 3.1.1.6.1 | The Game Play Activity is what the user will see while playing the game. |
| 3.1.1.6.2 | The game play activity shall make the game host the dealer for the first round. |
| 3.1.1.6.3 | The game play activity shall allow the dealer to start a round by drawing and playing a bait card. |
| 3.1.1.6.4 | The game play activity shall allow the dealer to end a round by picking the winning response card. |
| 3.1.1.6.5 | The game play activity shall allow each player to draw 8 response cards in the first round. |
| 3.1.1.6.6 | The game play activity shall allow each player, except the dealer, to draw one response card for all subsequent rounds. |
| 3.1.1.6.7 | The game play activity shall allow each player, except the dealer, to play one response card during a round that they best believe fits the bait card. |
| 3.1.1.6.8 | The game player activity shall add a point to the player’s total whose response card was chosen as the winner by the dealer. |
| 3.1.1.6.9 | The game play activity shall make the player who won the round the dealer for the next round. |
| 3.1.1.6.10 | The game play activity shall end the game when a player wins 5 rounds. |
| 3.1.1.6.11 | The game play activity shall end the game when if no player has won 5 rounds but all the response cards have been played. |

## 3.1.1.7 Security Considerations for All Activities

|  |  |
| --- | --- |
| Requirement number | SRS Requirement |
| 3.1.1.7.1 | PII shall never be transmitted during any part of the game activities |
| 3.1.1.7.3 | All communication shall be done using SSL (TLS 1 or greater) |

## 3.1.2 Server

## 3.1.2.1 Security considerations for Server Side Interface

|  |  |
| --- | --- |
| Requirement number | SRS Requirement |
| 3.1.2.1.1 | All code shall be written with best known practices to prevent SQL injection vulnerabilities |

## 3.2 Architectural Styles and Patterns

## 3.2.1 MVC Architecture Style

MVC is a common software architecture used when developing Android apps. The Model component of the architecture will be implemented through a LAMP web service solution stack. Web services, written in PHP, will be run within an Apache HTTP container and interact with a MySQL database instance. Both the Apache HTTP Server and MySQL Server will be hosted on a Linux. The View component is what is seen on the user’s android phone or emulator and is defined within XML files. The Controller component is Java code that controls the exchange of data between the View component and Model components.

## 3.2.2 Client Server Architecture

The Card Czar application will also use a three-tier client server architecture for the interaction between the Android app and the data that tracks the app’s gameplay. This architecture is being used because the Card Czar game allows for multiple players, in different locations that need to be able to share data with each other throughout the game. So a shared database, on a remote server, is needed to accomplish this task.

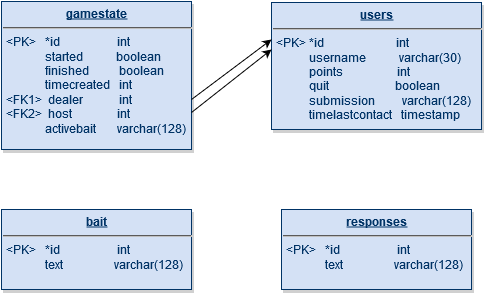
The data tier of the architecture will store the data associated with a game. This data, the name of the game room, game host, current bait dealer for the round, users playing the game, and the current point total for each player. The data is all stored within a MySQL relational database management system. The middle tier, also known as the web or application server, provides access to the data through REST services. These services allow for querying of the data, updating data or deletion of data. The client tier is the application’s user interface that will initiate calls to the middle tier based on user’s actions.

For example, a user can request to join a Card Czar game by providing the room name associated with the game they want to join and their user name. When they submit this information the application will call a middle tier service that will attempt to add the user to the game. The middle tier will first check with the data tier to ensure the room exists, the game for that room has not started and the room is not at maximum capacity. Assuming no error is encountered, the middle tier will then add the user to the data tier and to the room they requested.

## 3.3 Architectural Styles and Patterns

## 3.3.1 Database Design

Figure 3-1



## 3.4 Java Class Diagram

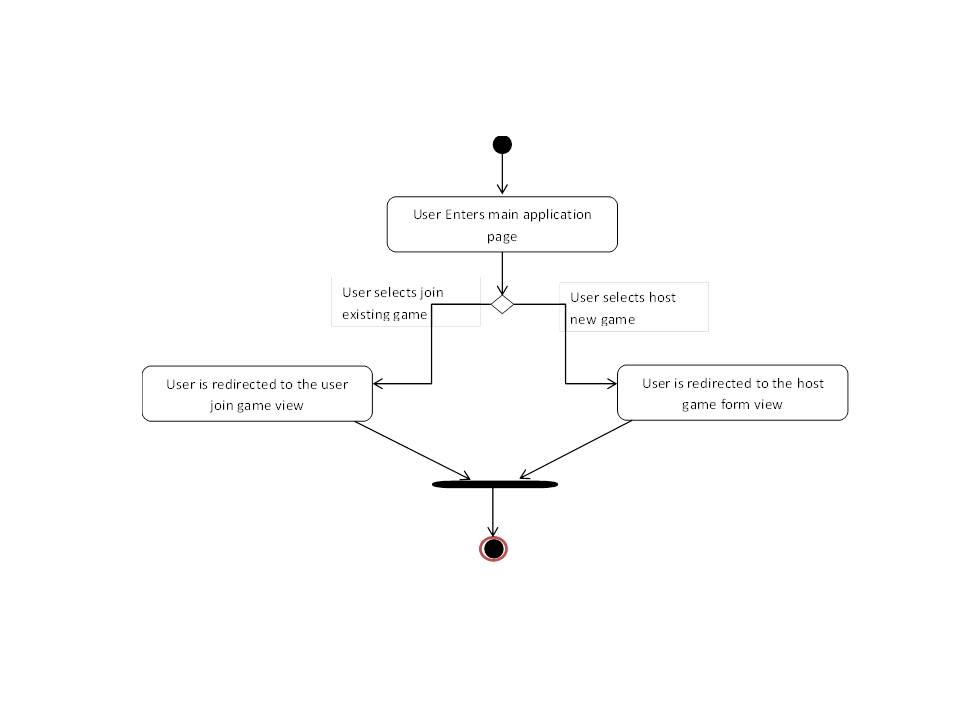
Figure 3-2



## 3.5 Sequence Diagram

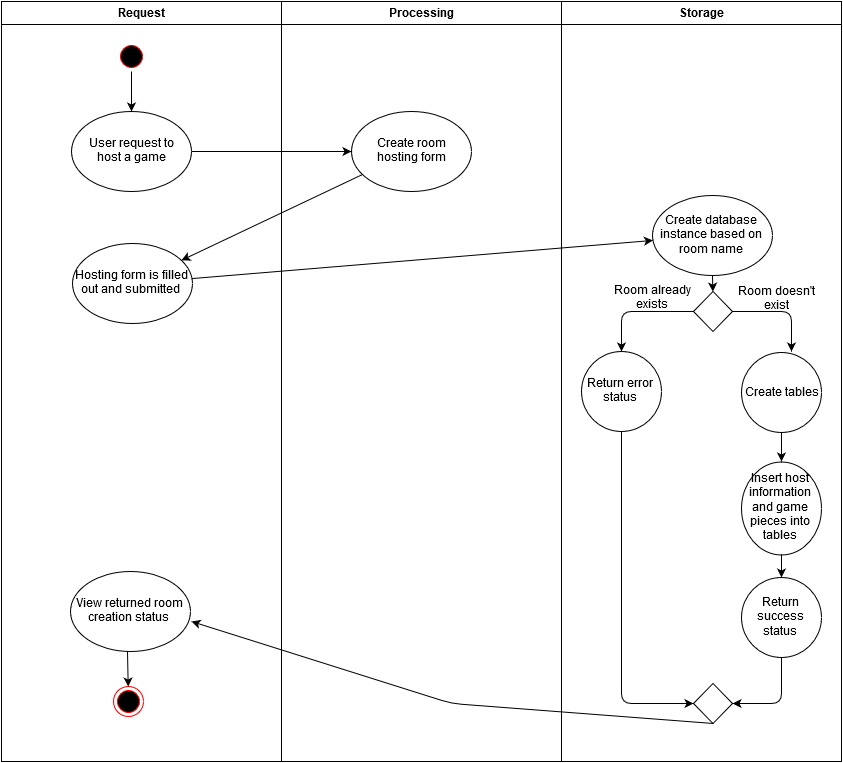
## 3.5.1 User Login Sequence Diagram

Figure 3-3



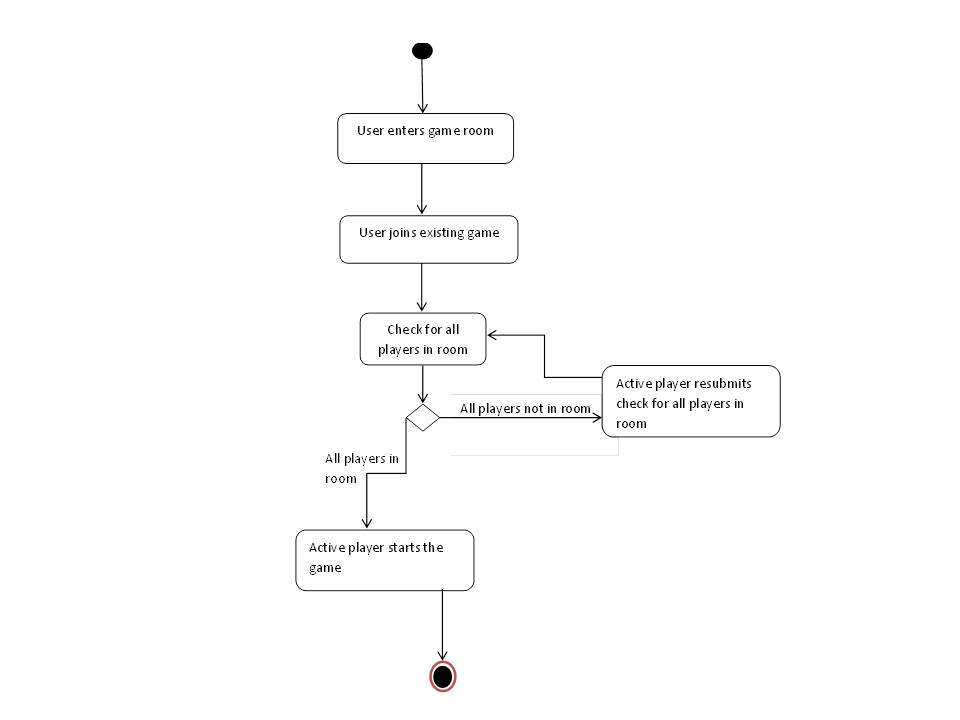
## 3.5.2 Host Creates Game Room Sequence Diagram

Figure 3-4



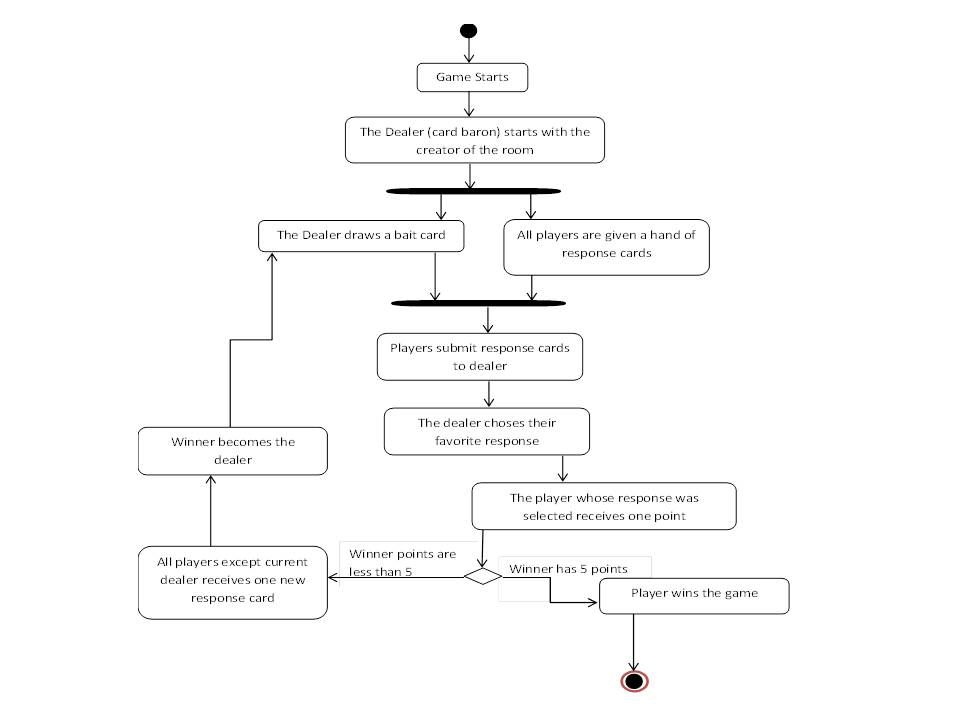
## 3.5.3 Join Game Sequence Diagram

Figure 3-5



## 3.5.4 Game Play Sequence Diagram

Figure 3-6



## 3.6 Algorithm Design

Card Czar uses 7 key algorithms (aside from simple programming techniques):

1. Intent communication
2. SQL operations
3. HTTP communication
4. URL encoding
5. List randomization
6. Server polling threads
7. Handlers

## 3.6.1 Algorithm Design – Intent communications

In addition to intending the next Activity, Card Czar makes use of bundled intent extras to facilitate communication between the Activities. Following is an example of RoomActivity intending GameplayActivity.

Starting with RoomActivity, this Activity gets extras (roomname and username) that were bundled together from the previous activity and sent to RoomActivity. Eleswhere (in RoomActivity), the contents are re-bundled and sent to GameplayActivity. In this case, another variable, a Boolean by the name of host, is also bundled and sent.

**public class** RoomActivity **extends** Activity {

String **roomname**; *// room name (database name)*

String **username**;

@Override

**protected void** onCreate(Bundle savedInstanceState) {

Extract intent.extras bundle.

**super**.onCreate(savedInstanceState);

setContentView(R.layout.***activity\_room***);

*// Get vars from previous activity*

Bundle extras = getIntent().getExtras();

**roomname** = extras.getString(**"roomname"**);

**username** = extras.getString(**"username"**);

}

…

Intent roomIntent = **new** Intent(**this**, GameplayActivity.**class**);

Bundle extras = **new** Bundle();

extras.putString(**"roomname"**, **roomname**);

Package bundle and send.

extras.putString(**"username"**, **username**);

extras.putBoolean(**"host"**, **true**);

roomIntent.putExtras(extras);

startActivity(roomIntent);

In the receiving Activity, variables are similarly extracted from the bundle (as they were in RoomActivity):

**public class** GameplayActivity **extends** Activity {

String **roomname**; *// room name (database name)*

String **username**;

Boolean **host**; *// if role="host", then user can boot other users*

@Override

**protected void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.***activity\_gameplay***);

*// Get vars from previous activity*

Bundle extras = getIntent().getExtras();

Extract intent.extras bundle.

**roomname** = extras.getString(**"roomname"**);

**username** = extras.getString(**"username"**);

**host** = extras.getBoolean(**"host"**);

## 3.6.2 Algorithm Design – SQL operations

Card Czar relies on storing values in a SQL database, using PHP files to achieve this. Following is a simple example of one of the PHP files which receives POST data (the room name, which is also the database name) from the Apache server and returns a list of users to be processed by the requesting Java class:

<?php

*// LINK TO SQL*

$link = **mysql\_connect**(**'localhost'**, **'root'**, **'password'**);

Connect to SQL.

if (!$link) { **die**(**'Could not connect: '** . **mysql\_error**()); }

*// PULL POST DATA*

Get POST data from Apache for editing the appropriate database.

$db\_name = $\_POST[**"roomname"**];

*// SELECT THAT DB*

Select database (based on room name).

**mysql\_select\_db**($db\_name , $link) or

**die**(**"user list Select DB Error: "**.**mysql\_error**());

*// GET USERS (do while because for mysql\_feth\_array gets first user)*

Get everything from users table.

$tablecontents = **mysql\_query**(**"SELECT \* FROM users"**);

if ($myrow = **mysql\_fetch\_array**($tablecontents))

Iterate through results and print them. Note that the if-statement iterates the row 1 time, so a do-while loop is used.

{ do { **echo** $myrow[**"username"**].**"|"**; }

while ($myrow = **mysql\_fetch\_array**($tablecontents));

}

else { **echo "User list SELECT \* FROM users Error"**; }

*// CLOSE DATABASE*

**mysql\_close**($link);

?>

An example of the return data might be: Moe|Larry|Curly

## 3.6.3 Algorithm Design – HTTP communication

Card Czar relies solely on a LAMP server for inter-smartphone communication. It is therefore imperative that each smartphone (or emulator) can communicate effectively with the LAMP server to conduct the necessary SQL operations from the previous example (3.5.2). This is done entirely using HTTP through use of the Apache HttpComponents™ libraries. The http session will be encapsulated in an SSL session (https).

*// Get list of users from LAMP*

**try** {

String url = **"http://ec2-52-3-241-249.compute-1.amazonaws.com/ccz\_user\_list.php"**;

HttpClient httpclient = **new** DefaultHttpClient();

Open http connection.

HttpPost post = **new** HttpPost(url);

List<NameValuePair> urlParameters = **new** ArrayList<>();

Set up one or more parameters for POST.

urlParameters.add(**new** BasicNameValuePair(**"room"**,**roomname**));

post.setEntity(**new** UrlEncodedFormEntity(urlParameters));

Stream content from network socket and put in String result.

HttpResponse response = httpclient.execute(post);

**result** = EntityUtils.*toString*(response.getEntity());

Close connection.

response.getEntity().consumeContent();

Log.*d*(**"GP user\_list result:"**, **result**);

} **catch** (IOException e) { e.printStackTrace(); }

Paired with the previous example, the above example would pass the room name (database name) and the String result would be set to “Moe|Larry|Curly”

## 3.6.4 Algorithm Design – URLencoding

Card Czar makes liberal use of URLencoding. Escape characters such as single or double quotes are replaced by URL codes. Not only does this preserve escape characters, but it serves to foil SQL injection attacks. The following is a snippet of PHP that makes use of URLencoding:

*// GET POST DATA*

$db\_name = **urlencode**($\_POST[**"room"**]);

URLencode values.

$user\_name = **urlencode**($\_POST[**"user"**]);

A function handles URLencoding arrays:

function urlencode\_array($array) {

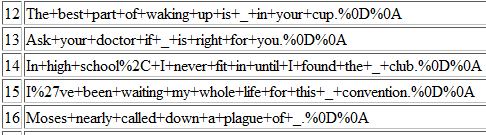
$out\_array = **array**();

URLencode each value.

foreach($array as $key => $value) {

$out\_array[**urlencode**($key)] = **urlencode**($value); }

**return** $out\_array; }

An example of URLencoding the list that is inserted into the database:  
  
  
  
Notice that spaces have been replaced by ‘+’, commas by %2C, and the apostrophe has been replaced by %27.

## 3.6.5 Algorithm Design – List randomization

Card Czar pulls the first entries of bait and responses, deleting as it does so. This provides for a very easy mechanism for “drawing cards”. This means that the list is randomized when the list is populated, rather than having a random manipulation of the database when the “card draw”-like operation is performed.

$bait=**file**(**'/var/www/html/b.txt'**);

Simple way to shuffle list.

**shuffle**($bait);

$bait = urlencode\_array($bait);

$bait = **"('"** .**implode**(**"'),('"**,$bait).**"')"**;

$query = **"INSERT INTO bait (text) VALUES $bait;"**;

**mysql\_query**($query, $link) or **die**(**"Error adding bait: "**.**mysql\_error**());

## 3.6.6 Algorithm Design – Server polling threads

Since CarCzar is a multiplayer game, players often end up waiting for other players. These threads may be waiting for a change to the SQL database or an input to their smartphone/emulator. This is done using threads that sleep or wait for a response. The following is an example that waits for a SQL change.

**class** CheckStartThread **implements** Runnable {

**private volatile boolean running** = **true**;

@Override

**public void** run() {

**while** (**running**) {

**try** {

Wait for 1.21 seconds.

Thread.*sleep*(1210);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

*// Get gamestate:started value from LAMP*

**try** {

String url = **"…/ccz\_check\_start.php"**;

HttpClient httpclient = **new** DefaultHttpClient();

HttpPost post = **new** HttpPost(url);

Check the server.

List<NameValuePair> urlParameters = **new** ArrayList<>();

urlParameters.add(**new** BasicNameValuePair(**"room"**, **roomname**));

post.setEntity(**new** UrlEncodedFormEntity(urlParameters));

HttpResponse response = httpclient.execute(post);

**result** = EntityUtils.*toString*(response.getEntity());

Log.*d*(**"Result of check start"**, **result**);

} **catch** (IOException e) {

e.printStackTrace();

}

*// If gamestate:started is true, send message to handler and stop thread*

Message msg = Message.*obtain*();

**if** (Objects.*equals*(**result**, **"OK"**)) {

Perform action and stop waiting when condition is met.

**handler**.removeCallbacks(**this**);

**handler**.sendMessage(msg);

**this**.**running**=**false**;  
 }

} *// end while* } *// end run()* } *// end userPollThread*

## 3.6.7 Algorithm Design – Handlers

Card Czar uses pure threads which cannot update the GUI. This must be done by Handler. Following is an example of message passing with handler.

First, the message bundling and passing:

*// When desired result, move past while and to step 2*

Message msg = Message.*obtain*();

**if** (!**result**.equals(**"Submissions are neither empty nor full"**)) {

**handler**.removeCallbacks(**this**); *// Clear message queue*

Bundle bundle = **new** Bundle();

bundle.putString(**"operation"**,**"displayResponses"**);

Bundle data.

bundle.putString(**"data"**, **result**);

msg.setData(bundle);

Pass to handler.

**handler**.sendMessage(msg);

waitForAllSubmissions=**false**;

}

} *// end while (waitForAllSubmissions)*

And the corresponding handler:

**handler**=**new** Handler(){

@Override

**public void** handleMessage(Message msg) {

Bundle bundle = msg.getData();

Unbundle data.

String operation = bundle.getString(**"operation"**);

String data = bundle.getString(**"data"**);

**if** (operation.equals(**"displayResponses"**)) {

data = **"empty|"**+data;

String [] responses = data.split(**"\\|"**);

**int** numresponses = responses.**length**-1;

Perform GUI operations.

**if** (numresponses>0) { button1.setText(responses[1]); }

**if** (numresponses>1) { button2.setText(responses[2]); }

...

} } }; *// end handler*

## 3.7 User and Database Interface Design

Figure 3-7

