Mobile Chess Sprint II Deliverable

December 3, 2014

**Prepared For:**

Daniel Freeman

Todd Breedlove

**Prepared By:**

Stewart Taylor

Jacob Neal

Aaron Costner

Lewis Sanchez

Table of Contents

[Introduction 2](#_Toc405395598)

[Backlog Items for Sprint II 2](#_Toc405395599)

[Use Cases 3](#_Toc405395600)

[General Information 3](#_Toc405395601)

[Requirements/Feature Trace 3](#_Toc405395602)

[Revision History 3](#_Toc405395603)

[Insertion Points in other Use Cases 3](#_Toc405395604)

[Use Case Name 3](#_Toc405395605)

[N/A 3](#_Toc405395606)

[Actors 3](#_Toc405395607)

[Pre-Conditions 4](#_Toc405395608)

[Start Stimulus 4](#_Toc405395609)

[Use Case Main Course Steps 4](#_Toc405395610)

[Exception Conditions 4](#_Toc405395611)

[Post-Conditions 4](#_Toc405395612)

[Candidate Objects 5](#_Toc405395613)

[Assumptions 5](#_Toc405395614)

[Sequence Diagrams 6](#_Toc405395615)

[User selects either login or register button 6](#_Toc405395616)

[Login the user if the user info is correct 7](#_Toc405395617)

[Register the user if the user does not exist and the username, password and email pass validation 8](#_Toc405395618)

[Class Diagrams 9](#_Toc405395619)

[Code 10](#_Toc405395620)

[Main.cpp 10](#_Toc405395621)

[CG\_login.h 11](#_Toc405395622)

[CG\_login.cpp 13](#_Toc405395623)

[CG\_user.h 18](#_Toc405395624)

[CG\_user.cpp 19](#_Toc405395625)

[CG\_dbManager.h 21](#_Toc405395626)

[CG\_dbManager.cpp 22](#_Toc405395627)

[CG\_validator.h 25](#_Toc405395628)

[CG\_validator.cpp 27](#_Toc405395629)

[Screenshots 33](#_Toc405395630)

[Start state 33](#_Toc405395631)

[Successful Login 34](#_Toc405395632)

[Username Validation 35](#_Toc405395633)

[Password Validation 36](#_Toc405395634)

[Email Validation 37](#_Toc405395635)

[Encrypted Passwords 38](#_Toc405395636)

[Conclusion 39](#_Toc405395637)

# **Introduction**

This deliverable lists all design documents, code and screenshots associated with this sprint. The table of contents lists the order in which these documents were produced. For example -- our group went from the requirements, to the use cases, to the sequence diagrams, to the class diagrams and finally to the code itself.

# Backlog Items for Sprint II

There were only two backlog items for Sprint II: Connect user to login system and deploy a ‘Hello World!’ application to Android.

# Use Cases

Below is our only use case for Sprint II. The outline follows the same format as the use case outline on the sharepoint site. Some parts of sharepoint outline were omitted because they are not applicable to our project (i.e. Timing execution).

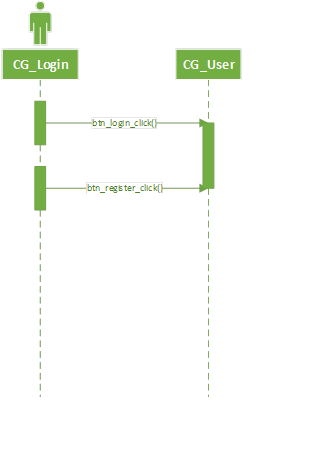
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| General Information | | | | | |
| Use Case ID Number : UC001  Subject Area : Chessgames Login  Description : Customer uses login to get into game account | | | | Responsible Analyst : Chessgames | |
| Requirements/Feature Trace | | | | | |
| REQ# | | Requirements Name and / or Short Description | | | |
| 001 | | Login allows customers to login to Chessgames account to play against other users. | | | |
|  | |  | | | |
|  | |  | | | |
|  | |  | | | |
| Revision History | | | | | |
| **Author** | | | **Date** | **Comments** | |
| Lewis Sanchez | | | 11/15/14 | First draft | |
| Jacob Neal | | | 11/18/14 | Updated | |
|  | | |  |  | |
|  | | |  |  | |
| Insertion Points in other Use Cases | | | | | |
| Use Case Name | | | **Use Case Number** | **Step Inserted After** | |
| N/A | | |  |  | |
|  | | |  |  | |
| Actors | | | | | |
| **Actor Name** | | | **P/S** | **Brief Description** | |
| Chessgames user | | | P | Chessgames user can login to their account. | |
|  | | |  |  | |
|  | | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pre-Conditions | | | | | | | | | | | | | | | |
| # | | | Description | | | | | | | | | | | | |
|  | | | Chessgames displays user account login page | | | | | | | | | | | | |
| Start Stimulus | | | | | | | | | | | | | | | |
| Chessgames user opens the application. | | | | | | | | | | | | | | | |
| Use Case Main Course Steps | | | | | | | | | | | | | | | |
| **Number** | | | | | **Description** | | | | | **Adds/Alt UC Name/Number** | | | **Bus Rule(s)#** | | |
| 01 | | | | | User inputs username and password | | | | |  | | |  | | |
| 02 | | | | | User selects either the “Login” or “Register” button. | | | | |  | | |  | | |
| 03 | | | | | Validate the username and password fields. | | | | |  | | |  | | |
|  | | | | |  | | | | |  | | |  | | |
|  | | | | |  | | | | |  | | |  | | |
|  | | | | |  | | | | |  | | |  | | |
|  | | | | |  | | | | |  | | |  | | |
| Exception Conditions | | | | | | | | | | | | | | | |
| **Exception Situations** | | | | | | | **Action(s) on Exception** | | | | | | | | **Adds/Alt Use Case #** |
| Invalid Username | | | | | | | Chessgames app displays message about the username not existing and returns to the login screen. | | | | | | | |  |
| Invalid password | | | | | | | Chessgames app displays message about the password not being correct and returns to the login screen. | | | | | | | |  |
| Invalid length | | | | | | | Chessgames app displays whether or not the username is too short or too long and returns to login screen. | | | | | | | |  |
| Invalid characters | | | | | | | Chessgames app displays whether or not there are invalid characters in the username and returns to login screen. | | | | | | | |  |
| Invalid periods | | | | | | | Chessgames app displays whether or not they have a period in the beginning of their username, or the end of the username and returns to login screen. | | | | | | | |  |
| Check for website | | | | | | | Chessgames app displays an error message for invalid username and returns to login screen. | | | | | | | |  |
| Post-Conditions | | | | | | | | | | | | | | | |
| **#** | | | **Description** | | | | | | | | | | | | |
| 1 | | | User is logged into account | | | | | | | | | | | | |
| 2 | | | Displays a welcome message saying, “Hello “ and the username. | | | | | | | | | | | | |
| 3 | | | Chessgames app returns to login screen | | | | | | | | | | | | |
| Candidate Objects | | | | | | | | | | | | | | | |
| **Class/Object Name** | | | | | | **Descriptions** | | | | | | **Possible**  **attributes** | | | |
| CG\_user | | | | | | Chessgames user | | | | | | Name, Elo, Country, etc | | | |
|  | | | | | |  | | | | | |  | | | |
| Assumptions | | | | | | | | | | | | | | | |
| **#** | | **Assumption** | | | | | | **Date**  **Raised** | **Raised**  **By** | | **Date**  **Verified** | | | **Verified By** | |
| 1 | | Customers only speak English. | | | | | | 11/18/14 | Lewis Sanchez | |  | | |  | |
|  | |  | | | | | |  |  | |  | | |  | |

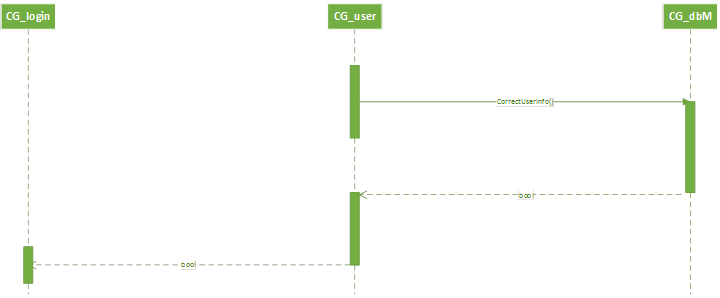
# Sequence Diagrams

Below are three sequence diagrams.

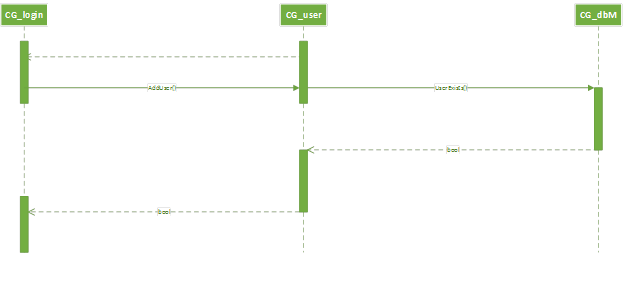
## User selects either login or register button



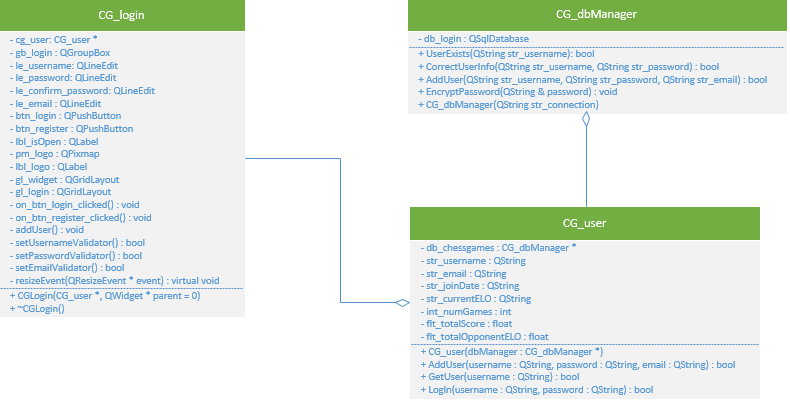
## Login the user if the user info is correct



## Register the user if the user does not exist and the username, password and email pass validation



# Class Diagrams



# Code

## Main.cpp

#include "CG\_login.h"

#include "CG\_user.h"

#include "CG\_dbManager.h"

#include <QApplication>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: ChessGames

\* Date Created: 11/07/2014

\* Date Last Modified: 12/01/2014

\* File Name: src/main.cpp

\*

\* Overview: Creates the foundation for our login application. Sets up the

\* database manager (Persistence Layer), chessgames user (Business Layer),

\* login widget (forms part of our Presentation Layer), and resizes our

\* window for our testing purposes.

\*

\* Input: Requires a valid username, password, and email for registration.

\* If user is already registered then the user simply enters a username, and

\* password. All usernames must follow a certain guideline as outlined by

\* the validation the user will see when main is executing. Such validations

\* include no double spaces, etc.

\*

\* Output: Displays the window, with the size that is specified, along with

\* QLineEdits where the username, password, email for inputs, along with

\* two buttons, one for registering an account, and the other for loggin in.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

QWidget win;

win.setWindowTitle("Chessgames");

win.setStyleSheet("background-color: #448ed3");

//Set up the database manager (Persistence Layer)

CG\_dbManager db\_manager(QDir::currentPath() + "/chessgames.db");

//Set up the chessgames user (Business Layer)

CG\_user cg\_user(&db\_manager);

//Set up the login widget (Part of the Presentation Layer)

CG\_login login(&cg\_user, &win);

QGridLayout win\_layout;

win\_layout.addWidget(&login);

win.setLayout(&win\_layout);

//Default size (Just for testing purposes)

win.resize(400, 600);

win.show();

return a.exec();

}

## CG\_login.h

#ifndef LOGIN\_H

#define LOGIN\_H

#include <QWidget>

#include <QGridLayout>

#include <QGroupBox>

#include <QLineEdit>

#include <QPushButton>

#include <QLabel>

#include <QDir>

#include <QFrame>

#include "CG\_validator.h"

#include "CG\_user.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class: CG\_login

\*

\* Constructors:

\* CG\_login(CG\_user \* user, QWidget \* parent = 0)

\*

\* Slots:

\* void on\_btn\_login\_clicked()

\* Checks to see whether user entered valid username and password.

\* Displays appropriate message on whether user was able to login

\* or not.

\* void on\_btn\_register\_clicked()

\* Checks to see if user already exists in the database. If user

\* does not exist and they entered a valid username,

\* password and email, then addUser() is called.

\* void addUser()

\* Adds user to the database

\* void setUsernameValidator()

\* Uses regular expressions to see if user is valid or not.

\*

\* Methods:

\* virtual void resizeEvent(QResizeEvent \* event)

\* Resizes based on whether width is greater than height or vice

\* versa.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CG\_login : public QWidget

{

Q\_OBJECT

public:

CG\_login(CG\_user \* user, QWidget \* parent = 0);

private slots:

void on\_btn\_login\_clicked();

void on\_btn\_register\_clicked();

void addUser();

//This is for the text changed slot of the username

bool setUsernameValidator();

//This is for the text changed slot of the password

bool setPasswordValidator();

//This is for the text changed slot of the confirm password

bool setConfirmPasswordValidator();

//This is for the text changed slot of the email

bool setEmailValidator();

protected:

virtual void *resizeEvent*(QResizeEvent \* event);

private:

CG\_user \* cg\_usr;

CG\_validator cg\_validator;

QGroupBox gb\_login;

QLineEdit le\_username;

QLineEdit le\_password;

QLineEdit le\_confirm\_password;

QLineEdit le\_email;

QPushButton btn\_login;

QPushButton btn\_register;

QLabel lbl\_isOpen;

QPixmap pm\_logo;

QLabel lbl\_logo;

QGridLayout gl\_widget;

QGridLayout gl\_login;

};

#endif

## CG\_login.cpp

#include "CG\_login.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Constructor. Initializes widget, group box,

\* buttons and pixel map.

\*

\* Entry: User has opened the app.

\*

\* Exit: Data members are initialized and stylesheets are

\* set, widgets are grouped appropriately and signals

\* are connected to slots.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CG\_login::CG\_login(CG\_user \* user, QWidget \* parent) :

QWidget(parent), cg\_validator(lbl\_isOpen), gb\_login(""), btn\_login("Login"),

btn\_register("Register"), pm\_logo("cg\_logo\_hires\_app.png")

{

//Pass the reference to the CG\_user

cg\_usr = user;

//Hide the password entry

le\_password.setEchoMode(QLineEdit::Password);

le\_confirm\_password.setEchoMode(QLineEdit::Password);

gb\_login.setStyleSheet("background: #448ed3; border-style: outset; border-radius: 10px; border-

color: #448ed3; min-width: 10em; padding: 6px;");

btn\_login.setStyleSheet("background: #b6ee65;");

btn\_register.setStyleSheet("background: #b6ee65;");

le\_username.setStyleSheet("background: #FFFFFF;");

le\_password.setStyleSheet("background: #FFFFFF;");

le\_confirm\_password.setStyleSheet("background: #FFFFFF;");

le\_email.setStyleSheet("background: #FFFFFF;");

lbl\_logo.setPixmap(pm\_logo);

//Add all the controls to the login layout

gl\_login.addWidget(&lbl\_logo, 0, 0);

gl\_login.addWidget(&le\_username, 1, 0);

gl\_login.addWidget(&le\_password, 2, 0);

gl\_login.addWidget(&le\_confirm\_password, 3, 0);

gl\_login.addWidget(&le\_email, 4, 0);

//Hide the email and confirm password until the user selects the register button

le\_confirm\_password.hide();

le\_email.hide();

le\_username.setPlaceholderText("Chessgames username");

le\_password.setPlaceholderText("Chessgames password");

gl\_login.addWidget(&btn\_login, 5, 0, 1, 2);

gl\_login.addWidget(&btn\_register, 6, 0, 1, 2);

gl\_login.addWidget(&lbl\_isOpen, 7, 0, 1, 2, Qt::AlignCenter);

//Set the group box to the login layout

gb\_login.setLayout(&gl\_login);

//Add the login group box to the layout of the widget

gl\_widget.addWidget(&lbl\_logo, 0, 0, 1, 1, Qt::AlignHCenter | Qt::AlignBottom);

gl\_widget.addWidget(&gb\_login, 1, 0);

//Set the widget to the layout

setLayout(&gl\_widget);

//Connect Login button to the appropriate slot

connect(&btn\_login, SIGNAL(released()), this, SLOT(on\_btn\_login\_clicked()));

//Connect Register button to the appropriate slot

connect(&btn\_register, SIGNAL(released()), this, SLOT(on\_btn\_register\_clicked()));

//Connect username validation color to username when text is changed

connect(&le\_username, SIGNAL(textChanged(QString)), this, SLOT(setUsernameValidator()));

//Connect password validation color to password when text is changed

connect(&le\_password, SIGNAL(textChanged(QString)), this, SLOT(setPasswordValidator()));

//Connect confirm password validation color to confirm password when text is changed

connect(&le\_confirm\_password, SIGNAL(textChanged(QString)), this,

SLOT(setConfirmPasswordValidator()));

//Connect email validation color to email when text is changed

connect(&le\_email, SIGNAL(textChanged(QString)), this, SLOT(setEmailValidator()));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check whether the user entered a valid

\* username and password.

\*

\* Entry: User has clicked the login button

\*

\* Exit: User is notified whether he has been sucessfully

\* logged in or not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CG\_login::on\_btn\_login\_clicked()

{

//If username and password is not correct, display

if (!cg\_usr->LogIn(le\_username.text(), le\_password.text()))

lbl\_isOpen.setText("Username or password is incorrect.");

else

//If username and password is correct, display user is logged in

lbl\_isOpen.setText("Successfully logged in. Hello, " + le\_username.text() + "!");

//Ensure email and password confirmation is hidden while logging in

le\_confirm\_password.hide();

le\_email.hide();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check if user already exists in database,

\* and call username validator.

\*

\* Entry: Username and password has been validated at login

\* screen.

\*

\* Exit: User is notified that user has been successfully

\* created.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CG\_login::on\_btn\_register\_clicked()

{

//If user already exists in the database

if (!cg\_usr->GetUser(le\_username.text()))

{

le\_confirm\_password.setPlaceholderText("Confirm password");

le\_confirm\_password.show();

le\_email.setPlaceholderText("Enter email");

le\_email.show();

connect(&btn\_register, SIGNAL(released()), this, SLOT(addUser()));

}

else

{

//User does not exist, change background to red and notify user

le\_username.setStyleSheet("background: #FF7777");

lbl\_isOpen.setText("User already exists.");

}

setUsernameValidator();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: User is added to database.

\*

\* Entry: Username and password has been validated at login

\* screen.

\*

\* Exit: User is notified that user has been successfully

\* created.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CG\_login::addUser()

{

if (setUsernameValidator() && setPasswordValidator() && setConfirmPasswordValidator() &&

setEmailValidator())

if (cg\_usr->AddUser(le\_username.text(), le\_password.text(), le\_email.text()))

lbl\_isOpen.setText("Successfully created user.");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Login screen is optimized to appropriate size.

\*

\* Entry: User opens login screen.

\*

\* Exit: Screen is resized depending on whether the width

\* width is greater than the height.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CG\_login::*resizeEvent*(QResizeEvent \* event)

{

//Resize depending on portrait orientation vs landscape orientation

if (width() > height())

gb\_login.setMaximumSize(width() / 2, height());

else

gb\_login.setMaximumSize(width(), height() / 2);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the username is valid when registering.

\*

\* Entry: User has clicked register.

\*

\* Exit: Displays whether the username is valid or not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_login::setUsernameValidator()

{

bool valid\_username = true;

//If email is visible, that must mean user clicked the register button

if (le\_email.isVisible())

{

//If user enters valid username, background = green. Else, background = red.

if (cg\_validator.CheckValidUsername(le\_username.text()))

le\_username.setStyleSheet("background: #77FF77");

else

{

valid\_username = false;

le\_username.setStyleSheet("background: #FF7777");

}

}

//Go back to login screen

update();

return valid\_username;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the password is valid when registering.

\*

\* Entry: User has clicked register.

\*

\* Exit: Displays whether the password is valid or not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_login::setPasswordValidator()

{

bool valid\_password = true;

//If email is visible, that must mean user clicked the register button

if (le\_email.isVisible())

{

//If user enters valid password, background = green. Else, background = red.

if (cg\_validator.CheckValidPassword(le\_password.text()))

le\_password.setStyleSheet("background: #77FF77");

else

{

valid\_password = false;

le\_password.setStyleSheet("background: #FF7777");

}

}

//Go back to login screen

update();

return valid\_password;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the password is the same when registering.

\*

\* Entry: User has clicked register.

\*

\* Exit: Displays whether the confirm password is the same.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_login::setConfirmPasswordValidator()

{

bool valid\_confirm\_password = true;

//If email is visible, that must mean user clicked the register button

if (le\_confirm\_password.isVisible())

{

//If user enters the same password, background=green. Else, background=red.

if (setPasswordValidator() && le\_password.text()==le\_confirm\_password.text())

le\_confirm\_password.setStyleSheet("background: #77FF77");

else

{

valid\_confirm\_password = false;

le\_confirm\_password.setStyleSheet("background: #FF7777");

}

}

//Go back to login screen

update();

return valid\_confirm\_password;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Ensures the email is valid when registering.

\*

\* Entry: User has clicked register.

\*

\* Exit: Displays whether the email is valid or not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_login::setEmailValidator()

{

bool valid\_email = true;

//If email is visible, that must mean user clicked the register button

if (le\_email.isVisible())

{

//If user enters valid email, background = green. Else, background = red.

if (cg\_validator.CheckValidEmailAddress(le\_email.text()))

le\_email.setStyleSheet("background: #77FF77");

else

{

valid\_email = false;

le\_email.setStyleSheet("background: #FF7777");

}

}

//Go back to login screen

update();

return valid\_email;

}

## CG\_user.h

#ifndef CG\_USER\_H

#define CG\_USER\_H

#include <QString>

#include "CG\_dbManager.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class: CG\_user

\*

\* Constructors:

\* CG\_user(CG\_dbManager \* dbManager)

\* Sets up the user to reference the main db manager.

\*

\* Methods:

\* bool AddUser(QString username, QString password, QString email)

\* Returns whether or not the user was successfully added to the

\* database.

\* bool GetUser(QString username)

\* Returns whether or not a user with the given username exists.

\* bool LogIn(QString username, QString password)

\* Returns whether or not the user info given is correct.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CG\_user

{

public:

CG\_user(CG\_dbManager \* dbManager);

bool AddUser(QString username, QString password, QString email);

bool GetUser(QString username);

bool LogIn(QString username, QString password);

private:

CG\_dbManager \* db\_chessgames;

QString str\_username;

QString str\_email;

QString str\_joinDate;

QString str\_currentELO;

int int\_numGames;

float flt\_totalScore;

float flt\_totalOpponentELO;

};

#endif

## CG\_user.cpp

#include "CG\_user.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Constructor. Passes the reference of the

\* main db manager to the class.

\*

\* Entry: The db manager has been set up.

\*

\* Exit: A reference to the main db manager has been passed

\* to the class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CG\_user::CG\_user(CG\_dbManager \* dbManager)

{

db\_chessgames = dbManager;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check whether the user entered a valid

\* username and password.

\*

\* Entry: User has clicked the login button

\*

\* Exit: Returns whether or not the username and password

\* are correct.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_user::LogIn(QString username, QString password)

{

bool logged\_in = false;

if (db\_chessgames->CorrectUserInfo(username, password))

logged\_in = true;

return logged\_in;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check if user exists in database.

\*

\* Entry: User has clicked the register button

\*

\* Exit: Returns whether or not the user exists in the

\* database.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_user::GetUser(QString username)

{

bool user\_exists = false;

if(db\_chessgames->UserExists(username))

user\_exists = true;

return user\_exists;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To add a user to the database.

\*

\* Entry: Username and password has been validated at login

\* screen.

\*

\* Exit: Returns whether or not the user has been

\* successfully added to the database.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_user::AddUser(QString username, QString password, QString email)

{

bool added\_user = false;

if (db\_chessgames->AddUser(username, password, email))

added\_user = true;

return added\_user;

}

## CG\_dbManager.h

#ifndef CG\_DBMANAGER\_H

#define CG\_DBMANAGER\_H

#include <QString>

#include <QVariant>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlQuery>

#include <QCryptographicHash> // Needed for encrypting in SHA256

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class: CG\_dbManager

\*

\* Constructors:

\* CG\_dbManager(QString str\_connection)

\* Opens the SQLite database for connection based upon the db path passed.

\*

\* Methods:

\* bool UserExists(QString str\_username)

\* Returns whether or not a user with the passed username exists in

\* the database.

\* bool CorrectUserInfo(QString str\_username, QString str\_password)

\* Returns whether or not the username and password info are correct

\* and in the database.

\* bool AddUser(QString str\_username, QString str\_password, QString str\_email)

\* Returns true if the user is successfully added into the database.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CG\_dbManager

{

public:

CG\_dbManager(QString str\_connection);

bool UserExists(QString str\_username);

bool CorrectUserInfo(QString str\_username, QString str\_password);

bool AddUser(QString str\_username, QString str\_password, QString str\_email);

void EncryptPassword(QString & password);

private:

QSqlDatabase db\_login;

};

#endif

## CG\_dbManager.cpp

#include "CG\_dbManager.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Constructor. Opens the database based upon the

\* db path specified.

\*

\* Entry: User has opened the app.

\*

\* Exit: The database is set to SQLite and the path to the

\* db is specified.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CG\_dbManager::CG\_dbManager(QString str\_connection)

{

//Connect to database

db\_login = QSqlDatabase::addDatabase("QSQLITE");

db\_login.setDatabaseName(str\_connection);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check whether or not the user entered the

\* correct username and password and it exists in

\* the database.

\*

\* Entry: User has clicked the login button

\*

\* Exit: Returns whether or not the login info is correct.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_dbManager::CorrectUserInfo(QString str\_username, QString str\_password)

{

int count = 0;

if(db\_login.open())

{

//Calls encryption password

EncryptPassword(str\_password);

QSqlQuery qry( db\_login );

qry.prepare( "SELECT \* FROM CG\_user WHERE Username= ? AND Passwd= ? COLLATE

NOCASE");

qry.addBindValue(str\_username);

qry.addBindValue(str\_password);

if(qry.exec())

for (; qry.next(); count++);

db\_login.close();

}

//Returns if the database finds a username and password match

return count > 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To check whether or not the user exists in the

\* database.

\*

\* Entry: User has clicked the register button

\*

\* Exit: Returns whether or not the user exists.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_dbManager::UserExists(QString str\_username)

{

int count = 0;

if(db\_login.open())

{

QSqlQuery qry( db\_login );

qry.prepare("SELECT \* FROM CG\_user WHERE Username= ? COLLATE NOCASE");

qry.addBindValue(str\_username);

if(qry.exec())

for (; qry.next(); count++);

db\_login.close();

}

return count > 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To add a user into the database with the passed

\* parameters: username, password and email.

\*

\* Entry: User has clicked the register button

\*

\* Exit: Returns whether or not the user was successfully

\* added into the database.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_dbManager::AddUser(QString str\_username, QString str\_password, QString str\_email)

{

bool added\_user = false;

if(db\_login.open())

{

//Calls encryption password

EncryptPassword(str\_password);

QSqlQuery qry( db\_login );

qry.prepare("INSERT INTO CG\_user (Username, Passwd, Email) VALUES(?, ?, ?)");

qry.addBindValue(str\_username);

qry.addBindValue(str\_password);

qry.addBindValue(str\_email);

if(qry.exec())

added\_user = true;

db\_login.close();

}

return added\_user;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: To encrypt a password using the SHA256 hashing

\* function.

\*

\* Entry: User is attempting to login or register.

\*

\* Exit: Alters the password string based upon the SHA256

\* encryption.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CG\_dbManager::EncryptPassword(QString & password)

{

// Takes the text in the le\_password and converts it to Utf8, so it can be placed

// in a type of 'const char \*' next

QByteArray passwordInBytes = password.toUtf8();

// casting the data in passwordInBytes (currently in the form of 'Utf8' to a type

// of 'const char \*'

const char \* convertedPasswordToVerify = passwordInBytes.constData();

// Instantiating an object that will create the hashing key for the password.

// Takes an argument specifying the encryption type you would like be executed on

// the string

QCryptographicHash sha256PasswordEncryptionGenerator(QCryptographicHash::Sha256);

// Adding the data to password encryption generator

sha256PasswordEncryptionGenerator.addData(convertedPasswordToVerify);

// Converting the password to the hash key using the result method and placing it

// in password.

password = (QString)sha256PasswordEncryptionGenerator.result();

}

## CG\_validator.h

#ifndef VALIDATOR\_H

#define VALIDATOR\_H

#include <QString>

#include <QRegExp>

#include <QLabel>

#include <QDebug>

#include <QDir>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlQuery>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class: CG\_validator

\*

\* Constructors:

\* CG\_validator(QLabel & feedback)

\*

\* Methods:

\* bool CheckUsernameLength(QString username)

\* Returns a boolean to show if username matches the correct length.

\* bool CheckUsernameNotNull(QString username)

\* Returns a boolean to show if username is not null.

\* bool CheckUsernameValidCharacters(QString username)

\* Returns a boolean showing if username contains only valid characters.

\* bool CheckUsernameInvalidPeriods(QString username)

\* Returns a boolean showing if username uses periods correctly.

\* bool CheckUsernameForWebsite(QString username)

\* Returns a boolean showing if username looks like a website.

\* bool CheckUsernameForInvalidSpaces(QString username)

\* Returns a boolean showing if username uses spaces incorrectly.

\* bool CheckValidUsername(QString username)

\* Returns a boolean that is the result of running the username

\* through the gauntlet of username validations.

\* bool CheckPasswordLength(QString password)

\* Returns a boolean showing if a password is of a correct length.

\* bool CheckRequiredPasswordCharacters(QString password)

\* Returns a boolean showing if a password contains required characters.

\* bool CheckValidPassword(QString password)

\* Returns a boolean that is the result of running the password

\* through the gauntlet of password validations.

\* bool CheckValidEmailAddress(QString email)

\* Returns a boolean showing if an email address is valid.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CG\_validator

{

public:

CG\_validator(QLabel & feedback);

bool CheckUsernameLength(QString username);

bool CheckUsernameNotNull(QString username);

bool CheckUsernameValidCharacters(QString username);

bool CheckUsernameInvalidPeriods(QString username);

bool CheckUsernameForWebsite(QString username);

bool CheckUsernameForInvalidSpaces(QString username);

bool CheckValidUsername(QString username);

bool CheckPasswordLength(QString password);

bool CheckRequiredPasswordCharacters(QString password);

bool CheckValidPassword(QString password);

bool CheckValidEmailAddress(QString email);

private:

QLabel \* lbl\_feedback;

};

#endif

## CG\_validator.cpp

#include "CG\_validator.h"

CG\_validator::CG\_validator(QLabel & feedback) :

lbl\_feedback(&feedback)

{ }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the username passed in is of a length

\* between 2 and 20 characters. Returns a value describing

\* whether the QString passed or failed this check.

\*

\* Entry: Takes in a QString which represents a username of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the username fits the

\* length of a valid username. A false value is returned if

\* the username's length is less than 2 characters long or more

\* than 20 characters long.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckUsernameLength(QString username)

{

bool valid\_length = true;

if (username.length() < 2) //If a username is less than 2 characters long.

{

valid\_length = false;

lbl\_feedback->setText("Username is too short.");

}

else if (username.length() > 20) //If a username is more than 20 characters long.

{

valid\_length = false;

lbl\_feedback->setText("Username is too long.");

}

return valid\_length;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the username passed in is null. If a

\* username is null, the feedback label will be updated to

\* let the user know they must have a username.

\*

\* Entry: Takes in a QString which represents a username of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the username is null.

\* If a username is null, a feedback label will let the user

\* know the error and return a false value. Otherwise, if the

\* username is not null, this will return a true value.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckUsernameNotNull(QString username)

{

bool not\_null = true;

if (username.isNull()) //If the username is null.

{

not\_null = false;

lbl\_feedback->setText("Please enter username.");

}

return not\_null;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the username passed in only contains

\* valid characters within it. Valid characters for a

\* username include all lowercase and uppercase letters,

\* numbers, dashes (-), spaces, and periods (.). If a

\* username happens to contain a character that isn't

\* valid, it will return a value describing that.

\*

\* Entry: Takes in a QString which represents a username of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the username only

\* contains valid characters. If an invalid character exists

\* in the username, it will return a false value and update a

\* label to inform the user of the issue.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckUsernameValidCharacters(QString username)

{

bool valid\_characters = true;

QRegExp regex("^([a-z]\*[A-Z]\*[0-9]\*\\-\*\\.\*\\s\*)\*$"); //Any string with only the

//characters

//a-z, A-Z, 0-9, -, (space),

//and .

if (!username.contains(regex)) //If the username does not contain only valid characters.

{

valid\_characters = false;

lbl\_feedback->setText("Username contains invalid characters.");

}

return valid\_characters;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the username passed in uses periods (.)

\* correctly within the string of characters. A username

\* cannot start with a period, nor can it end with a period.

\* If a username matches either of these cases, it will

\* inform the user of the error.

\*

\* Entry: Takes in a QString which represents a username of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the username has a

\* period (.) at the start of end of the username. This will

\* return a false value and set a label to inform the username

\* of the error.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckUsernameInvalidPeriods(QString username)

{

bool valid\_period\_placement = true;

QRegExp FrontPeriodRegex("^\\.+.\*$"); //Any string that starts with a period.

QRegExp BackPeriodRegex("^.\*\\.+$"); //Any string that ends with a period.

if (FrontPeriodRegex.indexIn(username) != -1) //If the username starts with a

//period.

{

valid\_period\_placement = false;

lbl\_feedback->setText("Username can't have a period at the front.");

}

if (BackPeriodRegex.indexIn(username) != -1) //If the username ends with a period.

{

valid\_period\_placement = false;

lbl\_feedback->setText("Username can't have a period at the back.");

}

return valid\_period\_placement;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the username passed in looks like a

\* website or not. If it does, it will inform the user

\* that a username cannot look like a website and return

\* a false value.

\*

\* Entry: Takes in a QString which represents a username of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the username looks

\* like a website. If it does, it will return false and set a

\* label to inform the user of the error in their username.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckUsernameForWebsite(QString username)

{

bool notAWebsite = true;

QRegExp WebsiteRegex("^.\*\\.+\\b(com|co|uk|org|net|edu)\\b$");

//A language describing a variety of email endings.

if (WebsiteRegex.indexIn(username) != -1) //If the username matches the website

//regular expression.

{

notAWebsite = false;

lbl\_feedback->setText("Username can't be a website.");

}

return notAWebsite;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the username passed in uses spaces

\* correctly within the string. A space is used incorrectly

\* if the username starts or ends with a space, or if there

\* are at least 2 spaces in a row. If spaces are used

\* incorrectly in the username, it will set a label to

\* inform the user of the error and return false.

\*

\* Entry: Takes in a QString which represents a username of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the username uses

\* spaces correctly within the string. It will set a label to

\* describe the error if it returns a false value.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckUsernameForInvalidSpaces(QString username)

{

bool validSpaces = true;

QRegExp startWithSpace("^\\s.\*$");

QRegExp endWithSpace("^.\*\\s$");

QRegExp doubleSpaces("^.\*\\s\\s.\*$");

if (startWithSpace.indexIn(username) != -1) //If the username starts with a space.

{

validSpaces = false;

lbl\_feedback->setText("Username can't start with a space.");

}

if (endWithSpace.indexIn(username) != -1) //If the username ends with a space.

{

validSpaces = false;

lbl\_feedback->setText("Username can't end with a space.");

}

if (doubleSpaces.indexIn(username) != -1) //If the username has 2+ spaces in a

//row.

{

validSpaces = false;

lbl\_feedback->setText("Username can't contain 2 consecutive spaces.");

}

return validSpaces;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the username passed in fits all the

\* requirements for a valid username. This involves

\* checking to see if a username is not null, of a specific

\* length, has only valid characters, doesn't use periods

\* incorrectly, doesn't look like a website, and that

\* there are no incorrectly used spaces.

\*

\* Entry: Takes in a QString which represents a username of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the username fits all

\* the specifications of a valid username. This is done by

\* running it through a gauntlet of validations.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckValidUsername(QString username)

{

bool valid\_username = true;

valid\_username = CheckUsernameNotNull(username) &&

CheckUsernameLength(username) &&

CheckUsernameValidCharacters(username) &&

CheckUsernameInvalidPeriods(username) &&

CheckUsernameForWebsite(username) &&

CheckUsernameForInvalidSpaces(username);

if (valid\_username)

lbl\_feedback->clear();

return valid\_username;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the password passed in is at least 8

\* characters long and less than or equal to 64 characters

\* long. Returns a boolean that describes if the password

\* fulfills this requirement, and will set a label if it

\* does not.

\*

\* Entry: Takes in a QString which represents a password of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the password is of a

\* correct length. Will change the error label if password

\* does not meet the specifications.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckPasswordLength(QString password)

{

bool valid\_length = true;

if (password.length() < 8) //If a password is less than 8 characters long.

{

valid\_length = false;

lbl\_feedback->setText("Password is too short.");

}

else if (password.length() > 64) //If a password is more than 64 characters long.

{

valid\_length = false;

lbl\_feedback->setText("Password is too long.");

}

return valid\_length;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the password passed in has the following

\* characters included in it: at least 1 lowercase letter,

\* at least 1 uppercase letter, and at least 1 number.

\* Will update an error label if the password does not meet

\* these requirements.

\*

\* Entry: Takes in a QString which represents a password of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the password contains

\* the required characters.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckRequiredPasswordCharacters(QString password)

{

QRegExp requiredUpperCase("^.\*[A-Z].\*$"); //Contains an uppercase letter.

QRegExp requiredLowerCase("^.\*[a-z].\*$"); //Contains a lowercase letter.

QRegExp requiredNumber("^.\*[0-9].\*$"); //Contains a number.

bool contains\_requiredCharacters = true;

if (!password.contains(requiredUpperCase) || !password.contains(requiredLowerCase)

|| !password.contains(requiredNumber)) //If the password does not contain

//required valid characters.

{

contains\_requiredCharacters = false;

lbl\_feedback->setText("Password must contain atleast 1 lowercase letter, 1

uppercase letter, and 1 number.");

}

return contains\_requiredCharacters;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the email passed in takes on the form of

\* an email address. An email address has at least 1

\* character followed by an @ symbol, then at least 1

\* character followed by a period, and then 1 of a variety

\* of email endings, like edu and com.

\*

\* Entry: Takes in a QString which represents an email of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the email takes on

\* the correct form of an email address.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckValidEmailAddress(QString email)

{

QRegExp valid\_email("^.+@.+\\.\\b(com|co|uk|org|net|edu)\\b$");

bool is\_email = true;

if (valid\_email.indexIn(email) == -1) //If the email doesn't match the email

//regular expression.

{

is\_email = false;

lbl\_feedback->setText("You must have a valid email address.");

}

if (is\_email)

lbl\_feedback->clear();

return is\_email;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Purpose: Check to see if the password is valid.

\*

\* Entry: Takes in a QString which represents a password of a user

\* that is attempting to login or register.

\*

\* Exit: Returns a boolean value describing if the password is valid.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool CG\_validator::CheckValidPassword(QString password)

{

bool valid\_password = true;

valid\_password = CheckPasswordLength(password) &&

CheckRequiredPasswordCharacters(password);

if (valid\_password)

lbl\_feedback->clear();

return valid\_password;

}

# Screenshots

Below are screenshots of the mobile chess application running. Each screenshot represents a different state of the application.

## Start state



## Successful Login



## Username Validation



## Password Validation

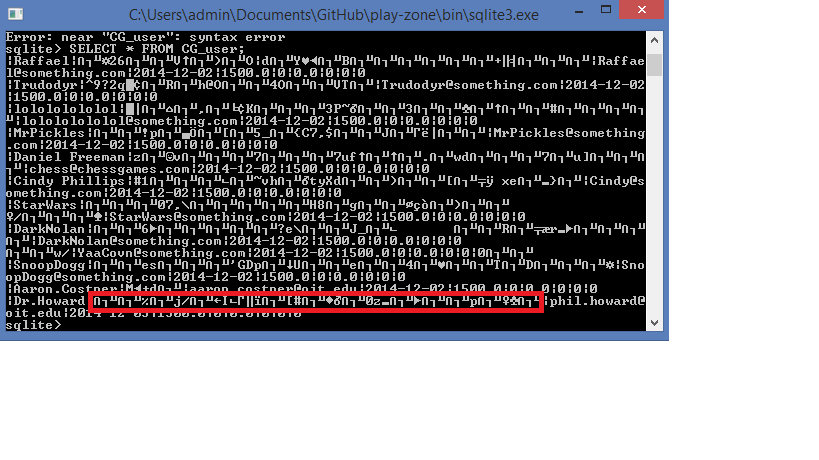


## Email Validation



## Encrypted Passwords

Below is a screenshot of our SQLite database, chessgames.db. Near the bottom of the console window, the red rectangle denotes a user’s password. This password characters are encrypted using the encryption scheme SHA-256 so no one will never see the password in plaintext.



# Conclusion

From the design documents to our PowerPoint presentation, this sprint can be considered a success. We finished the sprint ahead of schedule, went above/beyond the requirements and even received positive feedback from our product owner and stakeholder. Hopefully we will have the same success for Sprint III.