OXFORD BROOKES UNIVERSITY

MSc in Computer Science for Cyber Security

NETW7008: Secure Programming

Coursework 2 – Software Development

Anisa Elezi - 19177366

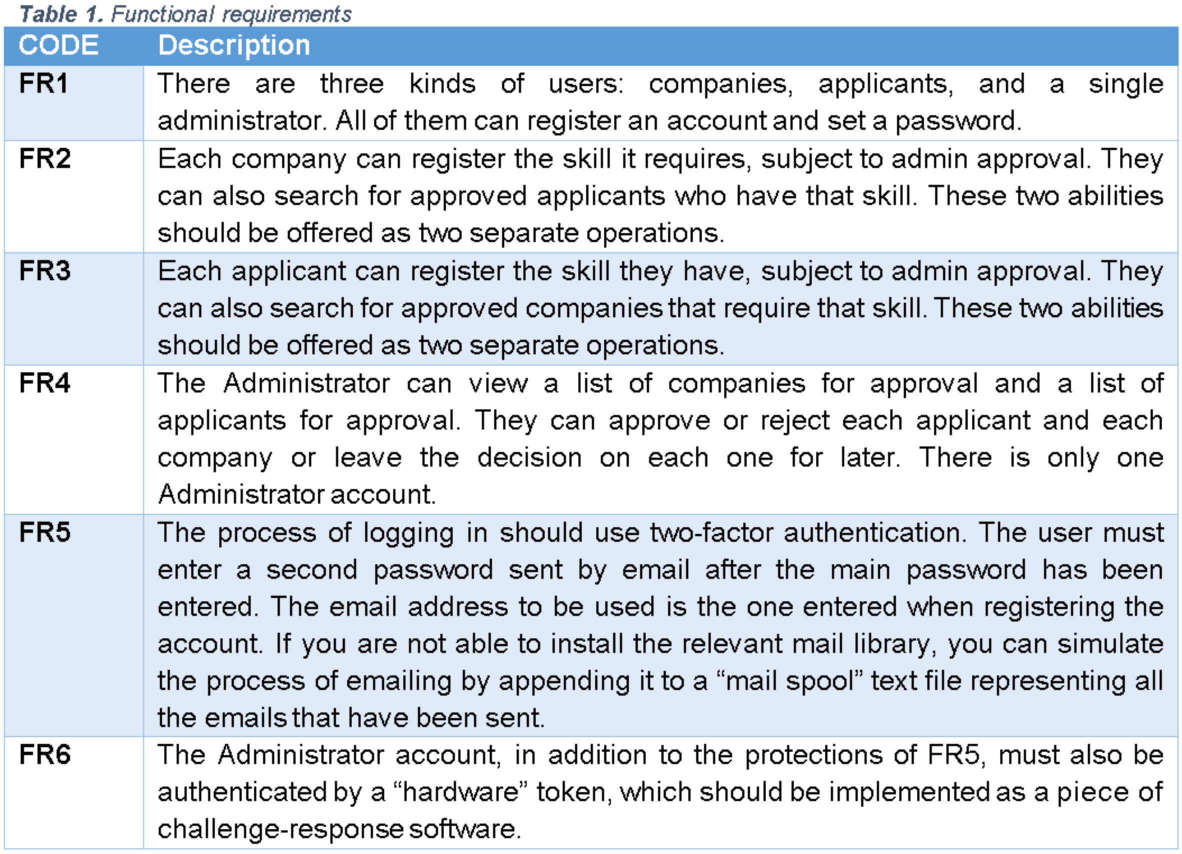
Introduction

The main goal of this project is to build a job recruitment website (**B**rookes **J**ob **Portal** – **BJP** **-** sots.brookes.ac.uk) where companies can publish the skills they are looking for and the applicants publish the skills they have.
This project is designed, implemented, and tested a **CGI** (**C**ommon **G**ateway **I**nterface) program, which is written in C++ language. CGI is the part of the Web server that can communicate with other programs running on the server [1].

1. Documented Implementation of Functional Requirements

This section has to do with the implementation of all functional requirements defined in **Error! Reference source not found.** through a secure software development stage.

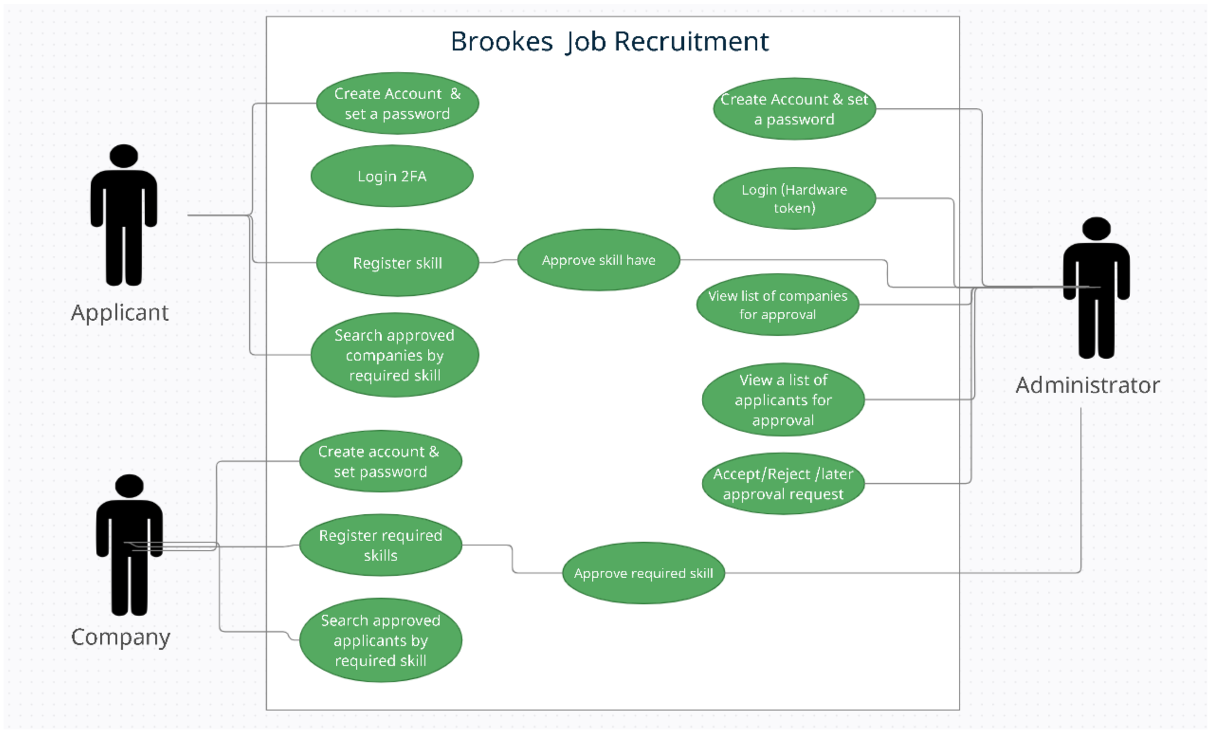
Table 1. Functional requirements



CODE	Description
FR1	There are three kinds of users: companies, applicants, and a single administrator. All of them can register an account and set a password.
FR2	Each company can register the skill it requires, subject to admin approval. They can also search for approved applicants who have that skill. These two abilities should be offered as two separate operations.
FR3	Each applicant can register the skill they have, subject to admin approval. They can also search for approved companies that require that skill. These two abilities should be offered as two separate operations.
FR4	The Administrator can view a list of companies for approval and a list of applicants for approval. They can approve or reject each applicant and each company or leave the decision on each one for later. There is only one Administrator account.
FR5	The process of logging in should use two-factor authentication. The user must enter a second password sent by email after the main password has been entered. The email address to be used is the one entered when registering the account. If you are not able to install the relevant mail library, you can simulate the process of emailing by appending it to a “mail spool” text file representing all the emails that have been sent.
FR6	The Administrator account, in addition to the protections of FR5, must also be authenticated by a “hardware” token, which should be implemented as a piece of challenge-response software.

As we can notice, in BJP software are three types of users and few actions they can do. To have a clear idea of this web app, **Figure 1** is designed as a Use Case diagram. Focusing on FR1, users of the BJP app are:

- admin
- companies
- applicants.



Brookes Job Recruitment
Create Account & set a password
Login 2FA
Register skill
Search approved companies by required skill
Create account & set password
Register required skills
Search approved applicants by required skill
Approve skill have
Approve required skill
Create Account & set a password
Login (Hardware token)
View list of companies for approval
View a list of applicants for approval
Accept/Reject /later approval request
Applicant
Company
Administrator

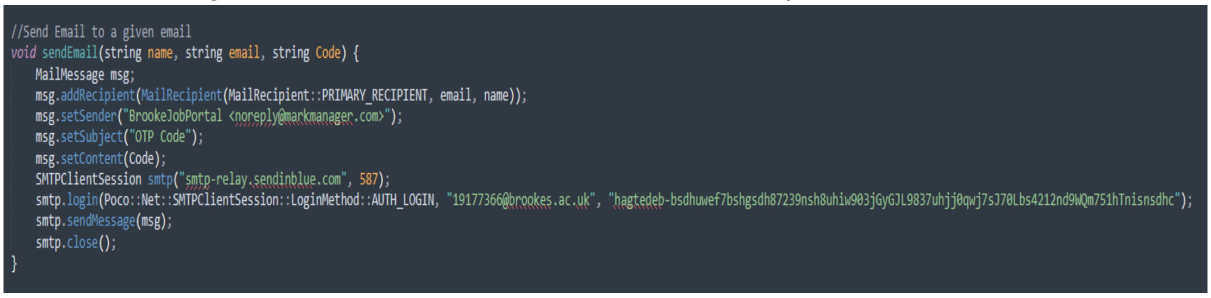
Figure 1. *Use Case diagram of a website*

1.1. Account registration (FR1)

All users of the BJP app should register their account giving their information such are full name, email, and password.

Once they have created their account and set the passwords, they can log in and authenticate through their credentials. But for the security issues:

- ➢ The company and applicant account should activate their account using the One Time Password (**OTP**) sent in their inbox. For this reason, is used the ***sendByEmail()*** method gets user's name, email and OTP code as a parameter.



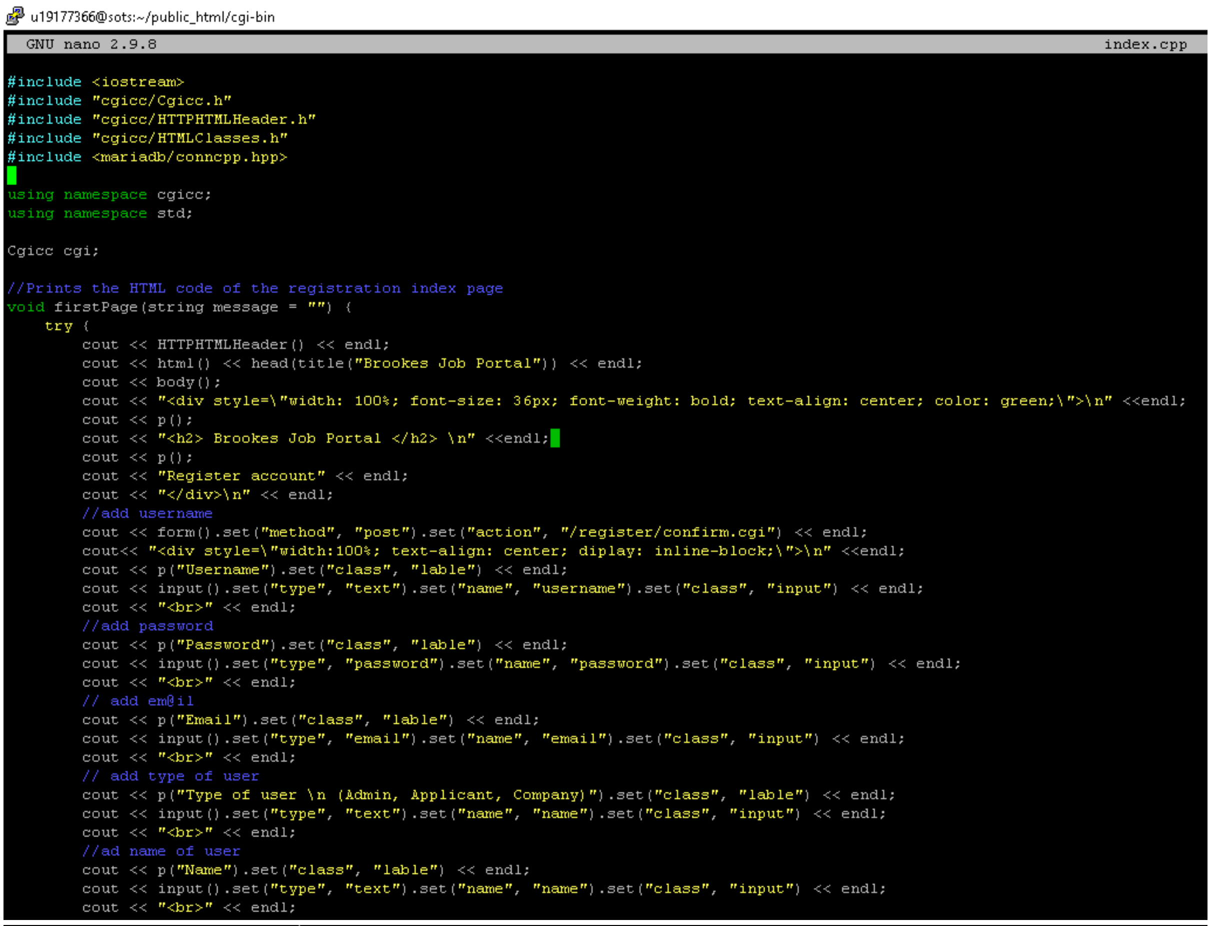
```
//Send Email to a given email
void sendEmail(string name, string email, string Code) {
    MailMessage msg;
    msg.addRecipient(MailRecipient(MailRecipient::PRIMARY_RECIPIENT, email, name));
    msg.setSender("BrookeJobPortal <noreply@markmanager.com>");
    msg.setSubject("OTP Code");
    msg.setContent(Code);
    SMTPClientSession smtp("smtp-relay.sendinblue.com", 587);
    smtp.login(Poco::Net::SMTPClientSession::LoginMethod::AUTH_LOGIN, "19177366@brookes.ac.uk", "hagtedeb-bsdhuwef7bshgsdh87239nsh8uhiw903jGyGJL9837uhjj0qwj7sJ70Lbs4212nd9WQm751hTnisnsdhc");
    smtp.sendMessage(msg);
    smtp.close();
}
```

To implement this method, there was needed to include several libraries like:

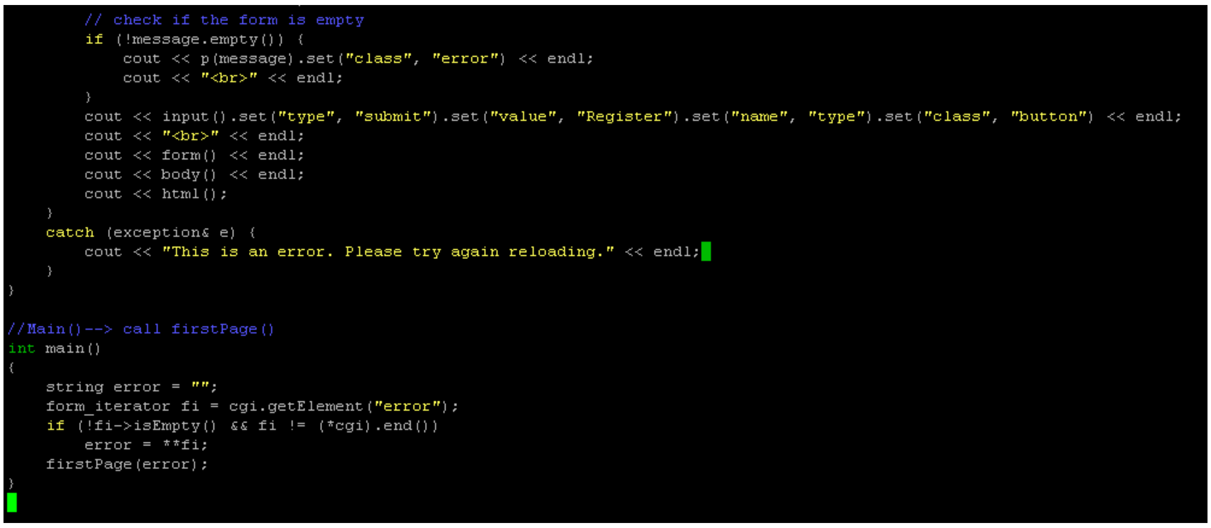
#include <Poco/Net/MailMessage.h>
#include <Poco/Net/MailRecipient.h>
#include <Poco/Net/NetException.h>
#include <Poco/Net/SMTPClientSession.h>

1.2. Company / Applicant functionality (FR2)

1. Companies can:
   - register their account and password
   - store all the skills they are looking for, which should be approved by the admin
   - Search for an applicant that meets the required skill
2. Applicant can:
   - Register their account and password
   - Store a skill they have, which should be approved by the admin
   - Search for a company that is looking for a specific skill



```
u19177366@sots:~/public_html/cgi-bin
  GNU nano 2.9.8                                                                          index.cpp

#include <iostream>
#include "cgicc/Cgicc.h"
#include "cgicc/HTTPHTMLHeader.h"
#include "cgicc/HTMLClasses.h"
#include <mariadb/conncpp.hpp>

using namespace cgicc;
using namespace std;

Cgicc cgi;

//Prints the HTML code of the registration index page
void firstPage(string message = "") {
    try {
        cout << HTTPHTMLHeader() << endl;
        cout << html() << head(title("Brookes Job Portal")) << endl;
        cout << body();
        cout << "<div style=\"width: 100%; font-size: 36px; font-weight: bold; text-align: center; color: green;\">\n" <<endl;
        cout << p();
        cout << "<h2> Brookes Job Portal </h2> \n" <<endl;
        cout << p();
        cout << "Register account" << endl;
        cout << "</div>\n" << endl;
        //add username
        cout << form().set("method", "post").set("action", "/register/confirm.cgi") << endl;
        cout<< "<div style=\"width:100%; text-align: center; diplay: inline-block;\">\n" <<endl;
        cout << p("Username").set("class", "lable") << endl;
        cout << input().set("type", "text").set("name", "username").set("class", "input") << endl;
        cout << "<br>" << endl;
        //add password
        cout << p("Password").set("class", "lable") << endl;
        cout << input().set("type", "password").set("name", "password").set("class", "input") << endl;
        cout << "<br>" << endl;
        // add em@il
        cout << p("Email").set("class", "lable") << endl;
        cout << input().set("type", "email").set("name", "email").set("class", "input") << endl;
        cout << "<br>" << endl;
        // add type of user
        cout << p("Type of user \n (Admin, Applicant, Company)").set("class", "lable") << endl;
        cout << input().set("type", "text").set("name", "name").set("class", "input") << endl;
        cout << "<br>" << endl;
        //ad name of user
        cout << p("Name").set("class", "lable") << endl;
        cout << input().set("type", "text").set("name", "name").set("class", "input") << endl;
        cout << "<br>" << endl;
```



```
        // check if the form is empty
        if (!message.empty()) {
            cout << p(message).set("class", "error") << endl;
            cout << "<br>" << endl;
        }
        cout << input().set("type", "submit").set("value", "Register").set("name", "type").set("class", "button") << endl;
        cout << "<br>" << endl;
        cout << form() << endl;
        cout << body() << endl;
        cout << html();
    }
    catch (exception& e) {
        cout << "This is an error. Please try again reloading." << endl;
    }
}

//Main()--> call firstPage()
int main()
{
    string error = "";
    form_iterator fi = cgi.getElement("error");
    if (!fi->isEmpty() && fi != (*cgi).end())
        error = **fi;
    firstPage(error);
}
```

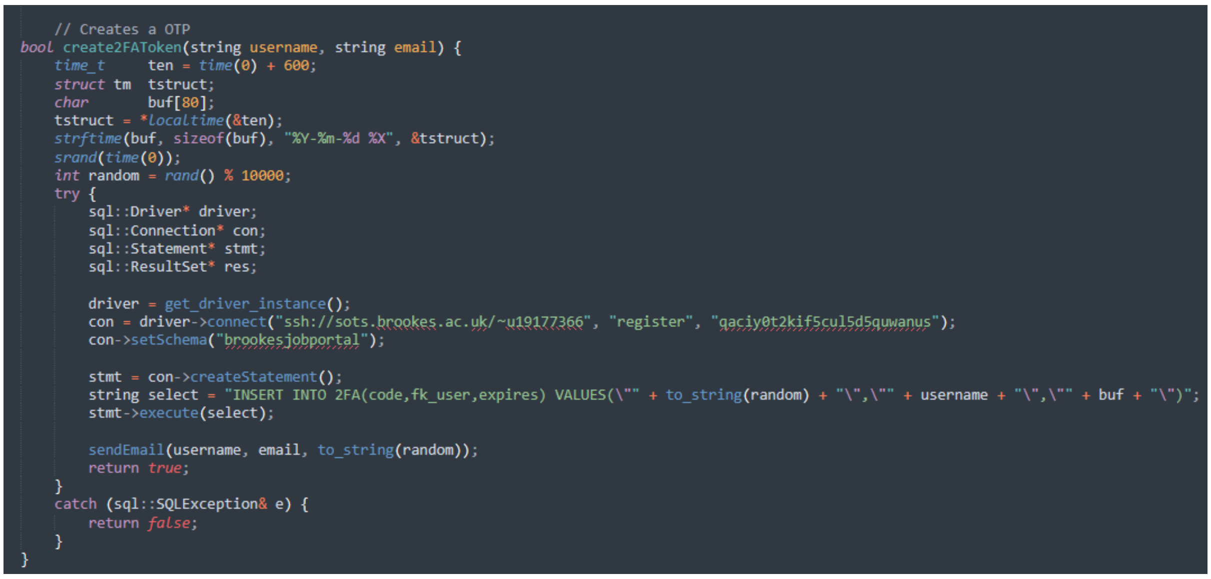
1.3. Administrator functionality (FR3)

Admin can:

- Add once itself and only one can be an administrator
- View a list of companies that are asking for skills approvement
- View a list of applicants that are asking for skills approvement
- Approve, reject, or leave the later decision on these actions

1.4. Two-factor authentication (FR5)

Two basic user type need to be activated and this was plan to be done after entering their emailed One Time Password. Whereas, the single admin need not only OTP as a normal user but also a hardware token that I planned to realised through QR code. It should be scanned with a Google Authenticator.

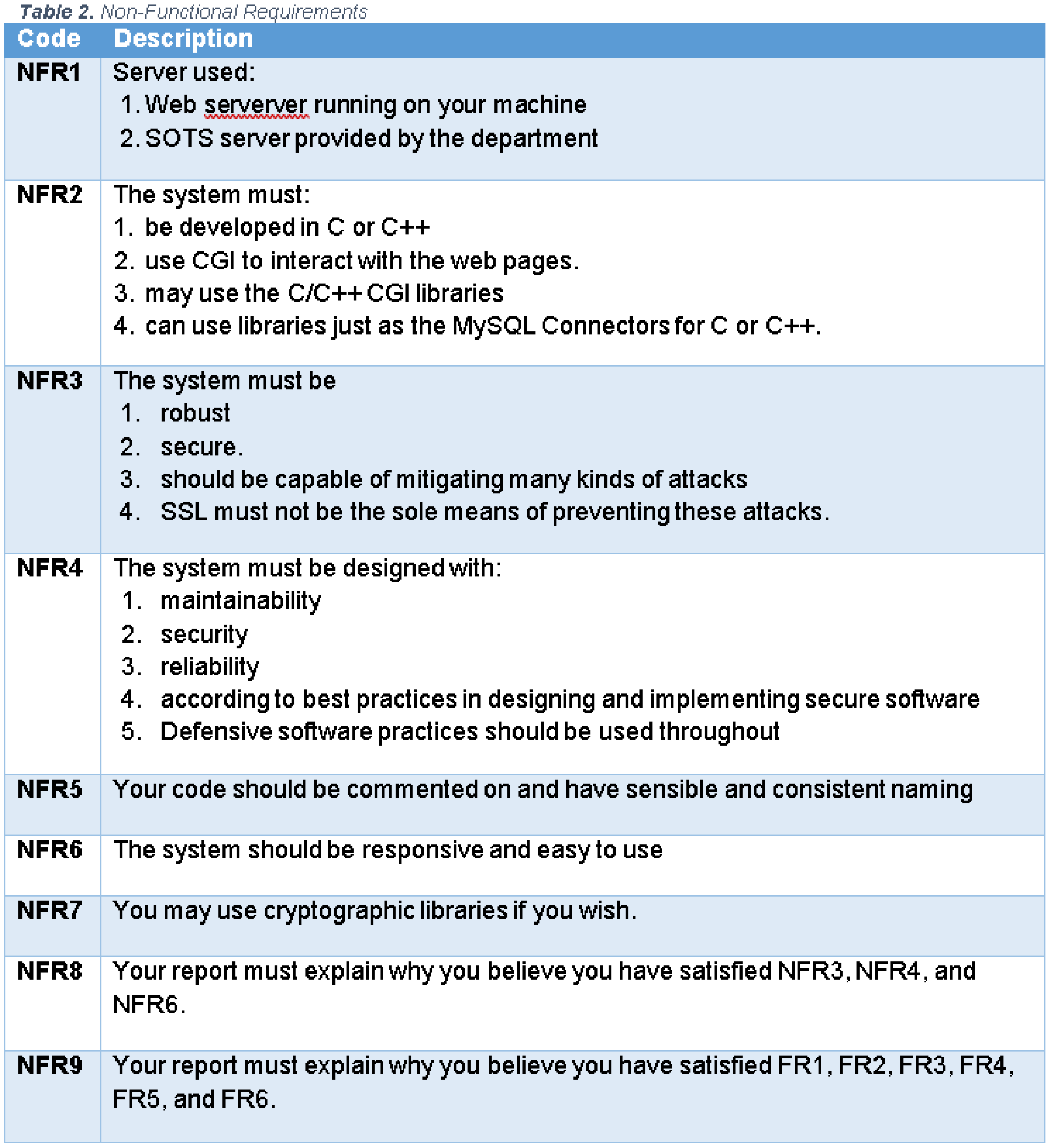


```
    // Creates a OTP
bool create2FAToken(string username, string email) {
    time_t     ten = time(0) + 600;
    struct tm  tstruct;
    char       buf[80];
    tstruct = *localtime(&ten);
    strftime(buf, sizeof(buf), "%Y-%m-%d %X", &tstruct);
    srand(time(0));
    int random = rand() % 10000;
    try {
        sql::Driver* driver;
        sql::Connection* con;
        sql::Statement* stmt;
        sql::ResultSet* res;

        driver = get_driver_instance();
        con = driver->connect("ssh://sots.brookes.ac.uk/~u19177366", "register", "qaciy0t2kif5cul5d5quwanus");
        con->setSchema("brookesjobportal");

        stmt = con->createStatement();
        string select = "INSERT INTO 2FA(code,fk_user,expires) VALUES(\"" + to_string(random) + "\",\"" + username + "\",\"" + buf + "\")";
        stmt->execute(select);

        sendEmail(username, email, to_string(random));
        return true;
    }
    catch (sql::SQLException& e) {
        return false;
    }
}
```

2. Documented Implementation of Non-functional Requirements

The non-functional requirements of the BJP app are described in **Error! Reference source not found.**,
which we can use to develop secure software.

Table 2. *Non-Functional Requirements*

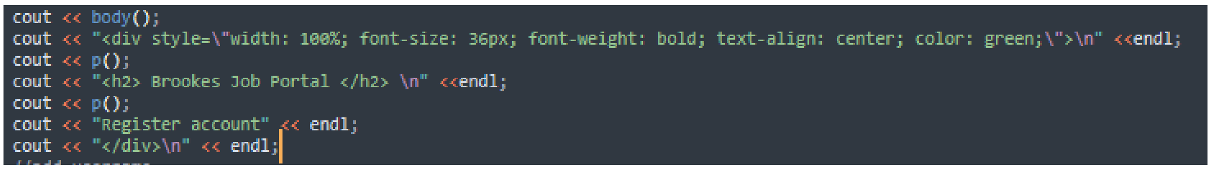


Code	Description
NFR1	Server used: 1. Web serververer running on your machine 2. SOTS server provided by the department
NFR2	The system must: 1. be developed in C or C++ 2. use CGI to interact with the web pages. 3. may use the C/C++ CGI libraries 4. can use libraries just as the MySQL Connectors for C or C++.
NFR3	The system must be 1. robust 2. secure. 3. should be capable of mitigating many kinds of attacks 4. SSL must not be the sole means of preventing these attacks.
NFR4	The system must be designed with: 1. maintainability 2. security 3. reliability 4. according to best practices in designing and implementing secure software 5. Defensive software practices should be used throughout
NFR5	Your code should be commented on and have sensible and consistent naming
NFR6	The system should be responsive and easy to use
NFR7	You may use cryptographic libraries if you wish.
NFR8	Your report must explain why you believe you have satisfied NFR3, NFR4, and NFR6.
NFR9	Your report must explain why you believe you have satisfied FR1, FR2, FR3, FR4, FR5, and FR6.

As a non-functional requirement ***(NRF1 and NRF2)*** of this web application are the language and environment used to implement it.

1. The web server used in this project is **SOTS** (www.sots.brookes.ac.uk) server

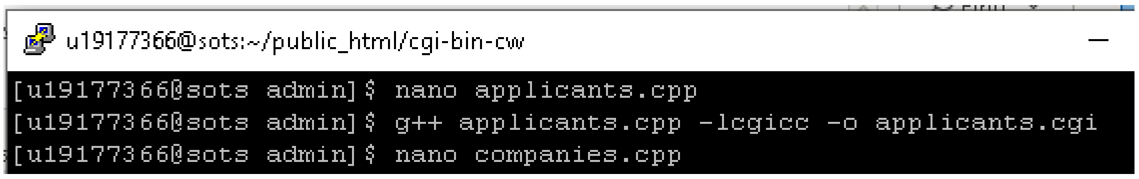
2. Hypertext Markup Language **HTML** and Cascade Style Sheet **CSS** are used for designing the web app. In this case, I choose to implement (CSS) internally as below



```
cout << body();
cout << "<div style=\"width: 100%; font-size: 36px; font-weight: bold; text-align: center; color: green;\">\n" <<endl;
cout << p();
cout << "<h2> Brookes Job Portal </h2> \n" <<endl;
cout << p();
cout << "Register account" << endl;
cout << "</div>\n" << endl;
```

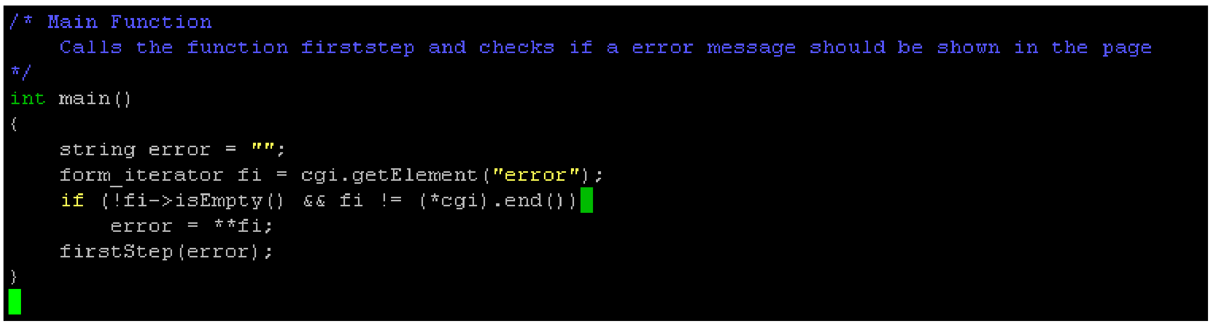
3. The used language for web app development is **C++,** and there are lots of benefits to using c++ in web applications listed below:

4. Common Gateway Interface **CGI** is used to ensure communication between the system and web pages. CGI is often used to generate **dynamic** Web content (*generated programmatically when the page is requested*) using client input, databases and other information services. I have compiled and run the program with the CGICC library, using this command
   g++ file.cpp -lcgicc -o file.cgi



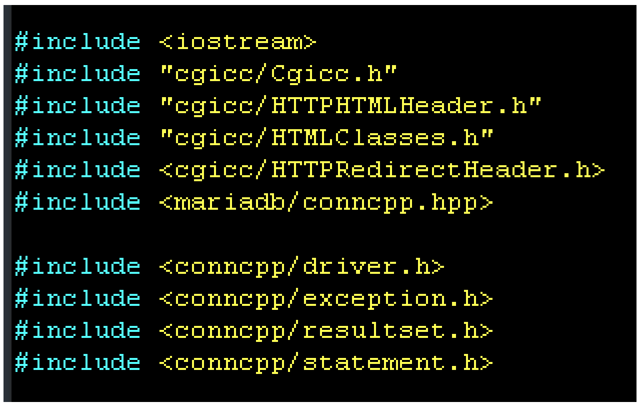
```
u19177366@sots:~/public_html/cgi-bin-cw
[u19177366@sots admin]$ nano applicants.cpp
[u19177366@sots admin]$ g++ applicants.cpp -lcgicc -o applicants.cgi
[u19177366@sots admin]$ nano companies.cpp
```

It is used CGI to iterate through all HTML elements



```
/* Main Function
    Calls the function firststep and checks if a error message should be shown in the page
*/
int main()
{
    string error = "";
    form_iterator fi = cgi.getElement("error");
    if (!fi->isEmpty() && fi != (*cgi).end())
        error = **fi;
    firstStep(error);
}
```

5. **MySQL** is the language used for database implementation in this project. For this reason, there are different libraries used such as those of C++ language and ***Mariadb*** connector in C++ codes.

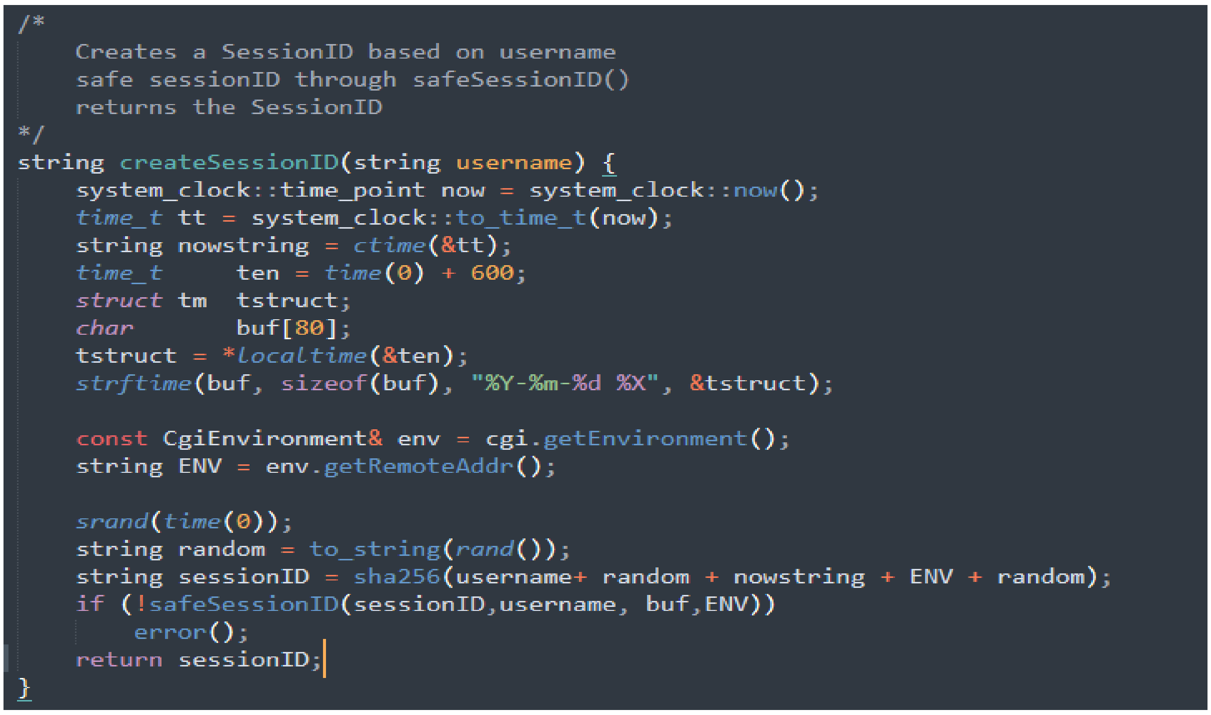


```
#include <iostream>
#include "cgicc/Cgicc.h"
#include "cgicc/HTTPHTMLHeader.h"
#include "cgicc/HTMLClasses.h"
#include <cgicc/HTTPRedirectHeader.h>
#include <mariadb/conncpp.hpp>

#include <conncpp/driver.h>
#include <conncpp/exception.h>
#include <conncpp/resultset.h>
#include <conncpp/statement.h>
```

2.1. Session handling using cookies (NFR3, NFR4)

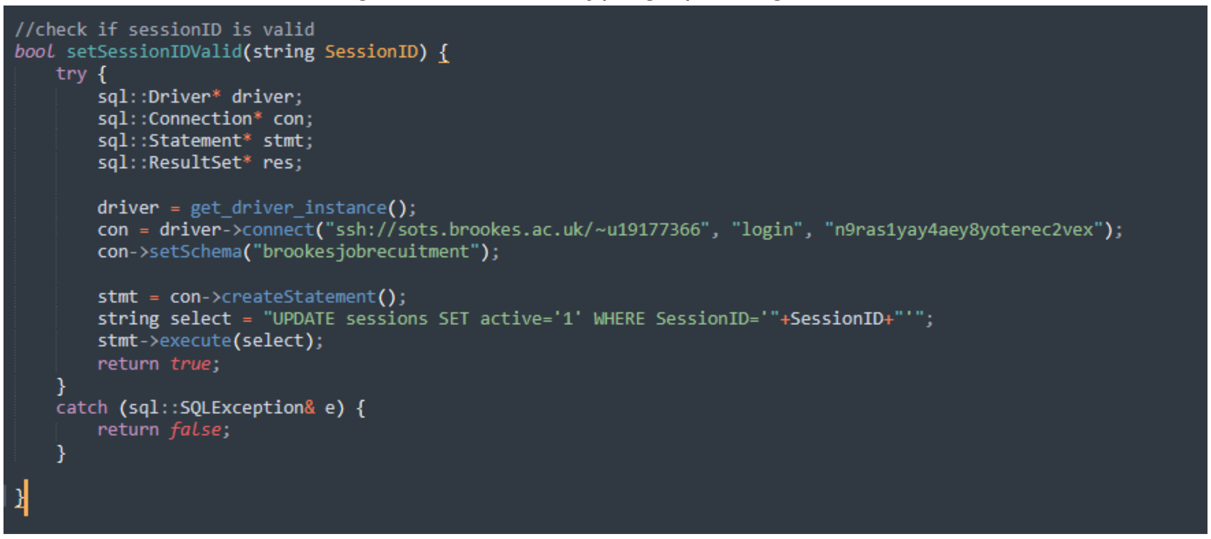
The app is designed to be secure, and to support this are created a function called ***createSessionID()*** and ***setCookie()*** as follows:

- Session



```
/*
    Creates a SessionID based on username
    safe sessionID through safeSessionID()
    returns the SessionID
*/
string createSessionID(string username) {
    system_clock::time_point now = system_clock::now();
    time_t tt = system_clock::to_time_t(now);
    string nowstring = ctime(&tt);
    time_t      ten = time(0) + 600;
    struct tm   tstruct;
    char        buf[80];
    tstruct = *localtime(&ten);
    strftime(buf, sizeof(buf), "%Y-%m-%d %X", &tstruct);

    const CgiEnvironment& env = cgi.getEnvironment();
    string ENV = env.getRemoteAddr();

    srand(time(0));
    string random = to_string(rand());
    string sessionID = sha256(username+ random + nowstring + ENV + random);
    if (!safeSessionID(sessionID,username, buf,ENV))
        error();
    return sessionID;
}
```

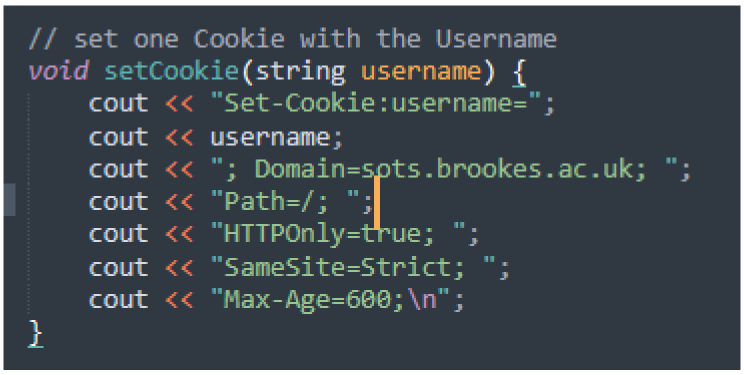
- A SessionID is created through
  - Username
  - DateTime
  - Random value
  - IP of the client.
- SessionID is linked to Client IP and username, so verification of the session is done through the checking of validation of SessionID, IP, and username checking.
- DateTime is set a lifetime at 10 min after which the user needs to be logged in again because the SessionID has expired.
- SessionID is secured through the SHA256 cryptographic algorithm.



```
//check if sessionID is valid
bool setSessionIDValid(string SessionID) {
    try {
        sql::Driver* driver;
        sql::Connection* con;
        sql::Statement* stmt;
        sql::ResultSet* res;

        driver = get_driver_instance();
        con = driver->connect("ssh://sots.brookes.ac.uk/~u19177366", "login", "n9ras1yay4aey8yoterec2vex");
        con->setSchema("brookesjobrecuitment");

        stmt = con->createStatement();
        string select = "UPDATE sessions SET active='1' WHERE SessionID='"+SessionID+"'";
        stmt->execute(select);
        return true;
    }
    catch (sql::SQLException& e) {
        return false;
    }

}
```

➢ Cookie

- Cookie should have the same lifetime (10 min) as SessionID
- Using the HttpOnly flag when generating a cookie helps mitigate the risk of the client-side script (XSS attack) accessing the protected cookie [2]

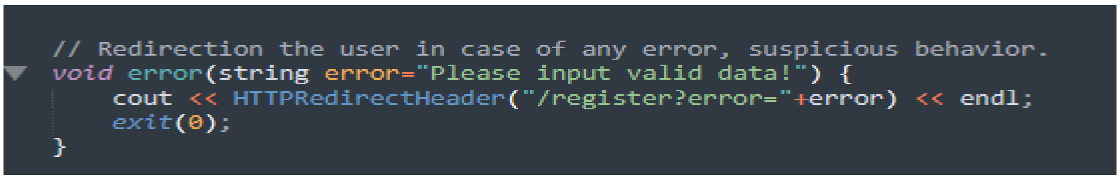


```
// set one Cookie with the Username
void setCookie(string username) {
    cout << "Set-Cookie:username=";
    cout << username;
    cout << "; Domain=sots.brookes.ac.uk; ";
    cout << "Path=/; ";
    cout << "HTTPOnly=true; ";
    cout << "SameSite=Strict; ";
    cout << "Max-Age=600;\n";
}
```

2.2. Redirection of users not logged in (NFR3, NFR4)

There are several cases when the client should be redirected to the main page:

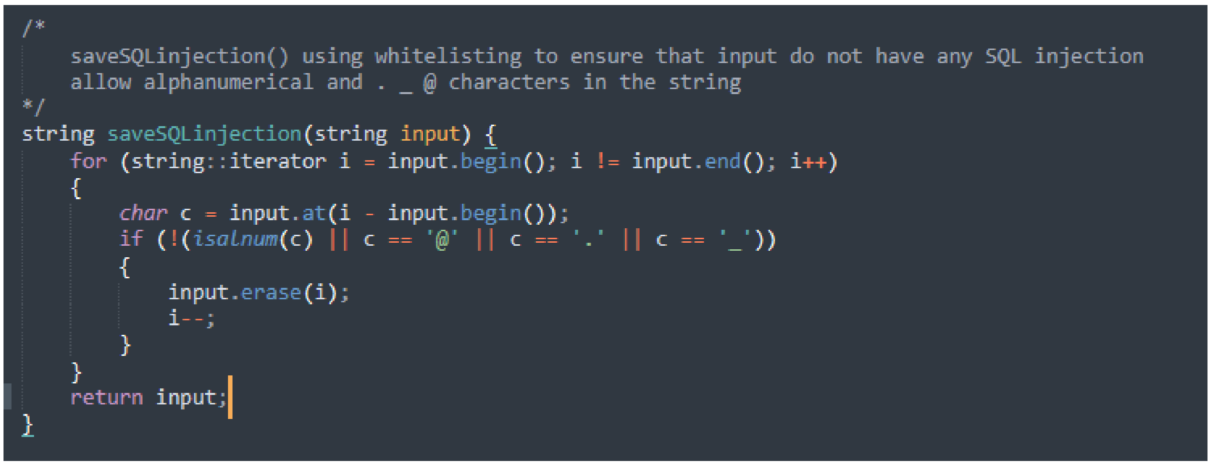
- If SessionID has expired (expired lifetime)
- If SessionID does not match with the given username or incorrect client IP address
- If the user is accessing any page without logging
- If a normal user tries to access higher privilege pages such are those of admin



```
// Redirection the user in case of any error, suspicious behavior.
void error(string error="Please input valid data!") {
    cout << HTTPRedirectHeader("/register?error="+error) << endl;
    exit(0);
}
```

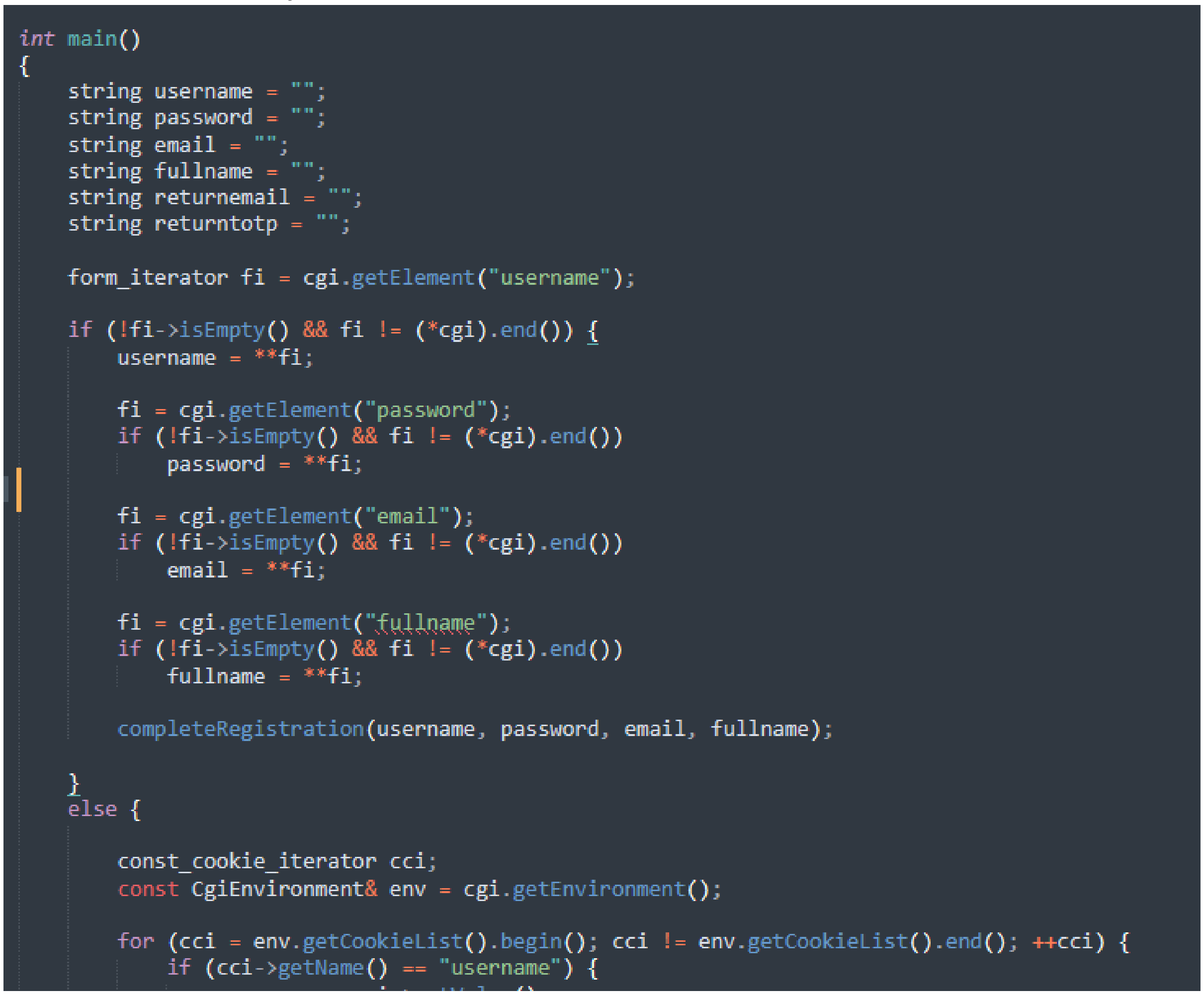
2.3. Input sanitization (NFR3, NFR4)

In this section is create a **saveSQLinjection()** which care about special characters that might be used to inject something in database. This is done using **whitelisting** allowing only ***alphanumeric*** and special chars such are “_”, “**.**”, “@”.



```
/*
    saveSQLinjection() using whitelisting to ensure that input do not have any SQL injection
    allow alphanumerical and . _ @ characters in the string
*/
string saveSQLinjection(string input) {
    for (string::iterator i = input.begin(); i != input.end(); i++)
    {
        char c = input.at(i - input.begin());
        if (!(isalnum(c) || c == '@' || c == '.' || c == '_'))
        {
            input.erase(i);
            i--;
        }
    }
    return input;
}
```

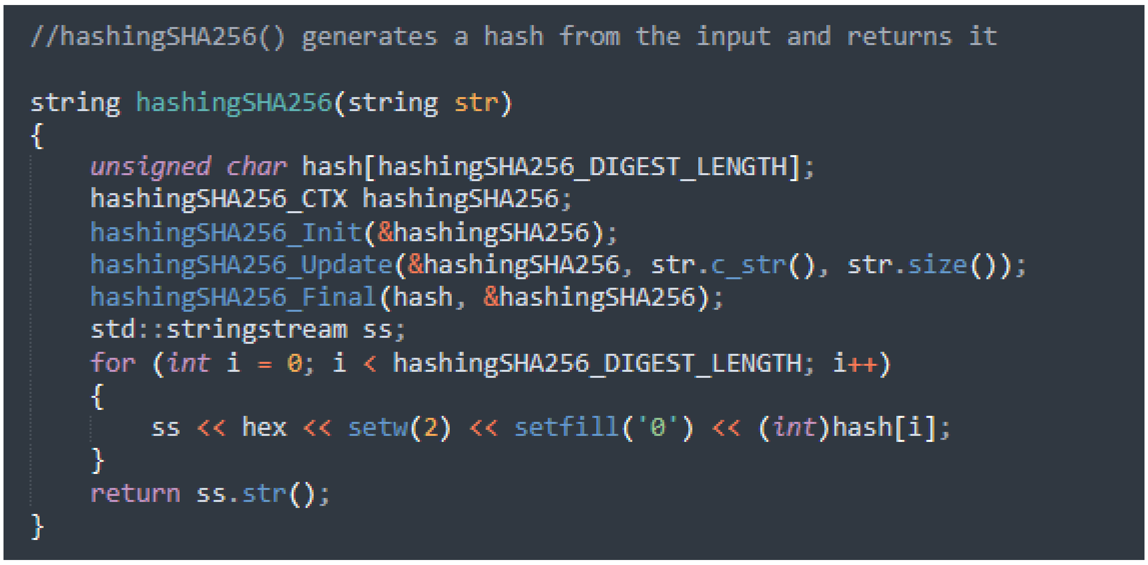
And also some validity in form



```
int main()
{
    string username = "";
    string password = "";
    string email = "";
    string fullname = "";
    string returnemail = "";
    string returntotp = "";

    form_iterator fi = cgi.getElement("username");

    if (!fi->isEmpty() && fi != (*cgi).end()) {
        username = **fi;

        fi = cgi.getElement("password");
        if (!fi->isEmpty() && fi != (*cgi).end())
            password = **fi;

        fi = cgi.getElement("email");
        if (!fi->isEmpty() && fi != (*cgi).end())
            email = **fi;

        fi = cgi.getElement("fullname");
        if (!fi->isEmpty() && fi != (*cgi).end())
            fullname = **fi;

        completeRegistration(username, password, email, fullname);

    }
    else {

        const_cookie_iterator cci;
        const CgiEnvironment& env = cgi.getEnvironment();

        for (cci = env.getCookieList().begin(); cci != env.getCookieList().end(); ++cci) {
            if (cci->getName() == "username") {
```

2.4. Encryption of stored information (NFR3, NFR4)

To ensure hashed value, is created a hashingSHA256() which is able to return a hashed value of a given input.

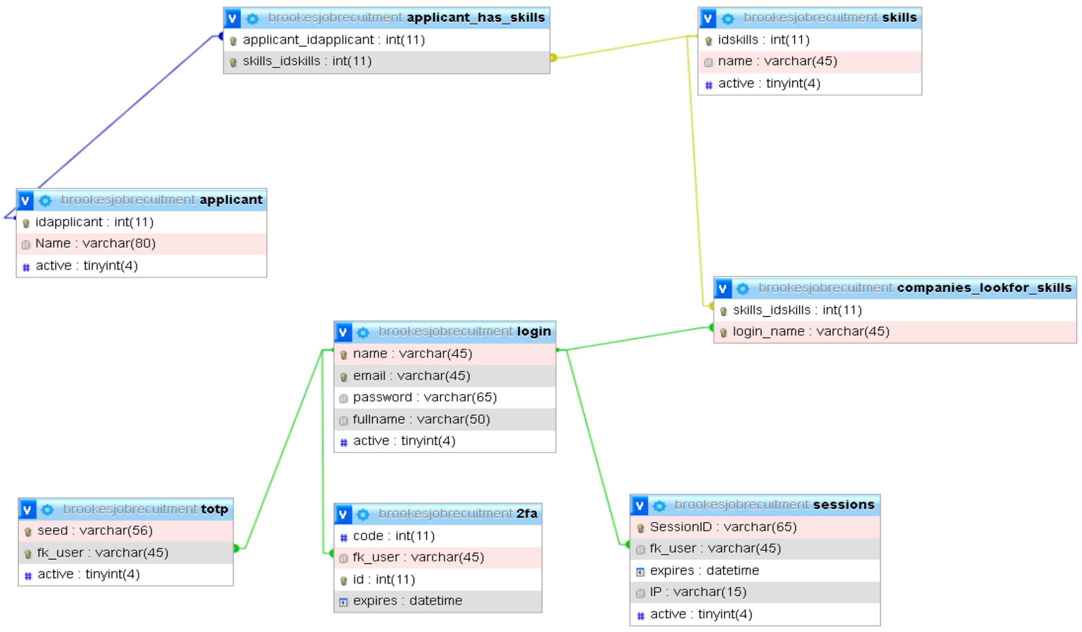


```
//hashingSHA256() generates a hash from the input and returns it

string hashingSHA256(string str)
{
    unsigned char hash[hashingSHA256_DIGEST_LENGTH];
    hashingSHA256_CTX hashingSHA256;
    hashingSHA256_Init(&hashingSHA256);
    hashingSHA256_Update(&hashingSHA256, str.c_str(), str.size());
    hashingSHA256_Final(hash, &hashingSHA256);
    std::stringstream ss;
    for (int i = 0; i < hashingSHA256_DIGEST_LENGTH; i++)
    {
        ss << hex << setw(2) << setfill('0') << (int)hash[i];
    }
    return ss.str();
}
```

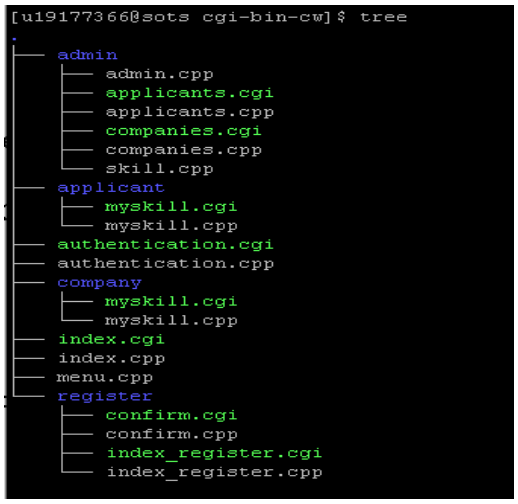
3. Other documentation

3.1. Documentation of system design including demonstration

This section reports the system design of the BJP web app. This app is using a database, so the following is a screenshot from an Entity Relationship diagram of our database. The design ensures non-redundancy, and it is normalised, so this design is maintainable and reliable, in this way we satisfy the ***NRF4 requirement***.



Also, the code in C++ is annotated and a careful name was chosen, which helps in interpreting and understanding logic flow easily, that satisfy also ***NRF5 requirement***. The first planning schema for this application was to be like the picture below:



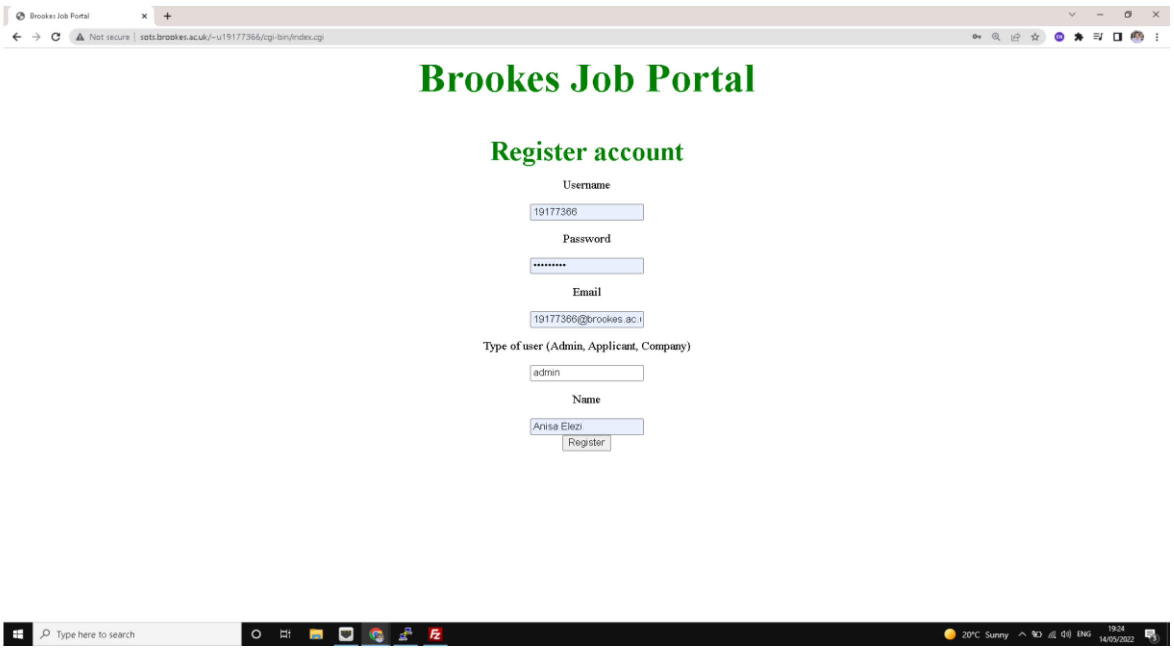
```
[u19177366@sots cgi-bin-cw]$ tree
.
├── admin
│   ├── admin.cpp
│   ├── applicants.cgi
│   ├── applicants.cpp
│   ├── companies.cgi
│   ├── companies.cpp
│   └── skill.cpp
├── applicant
│   ├── myskill.cgi
│   └── myskill.cpp
├── authentication.cgi
├── authentication.cpp
├── company
│   ├── myskill.cgi
│   └── myskill.cpp
├── index.cgi
├── index.cpp
├── menu.cpp
└── register
    ├── confirm.cgi
    ├── confirm.cpp
    ├── index_register.cgi
    └── index_register.cpp
```

But I was not able to implement all of them.

3.2. Justification of system design

As we can notice, BJP is a minimalist design style, satisfying the NFR6 requirement:

1. To satisfy the User Experience (**UX**) goal, there are only a few displayed elements offering easily understandable usage of the BJP app.
2. Satisfying User Interface (**UI**), the elements are designed to display vertically, so it offers adjustable in different sizes of screen. Also, the colours and accents are chosen carefully to give the user sense of concentration and focus on the right element/page.
3. The following images demonstrate an adjustable web page:



Brookes Job Portal

Register account

Username

19177366

Password

••••••••••

Email

19177366@brookes.ac.u

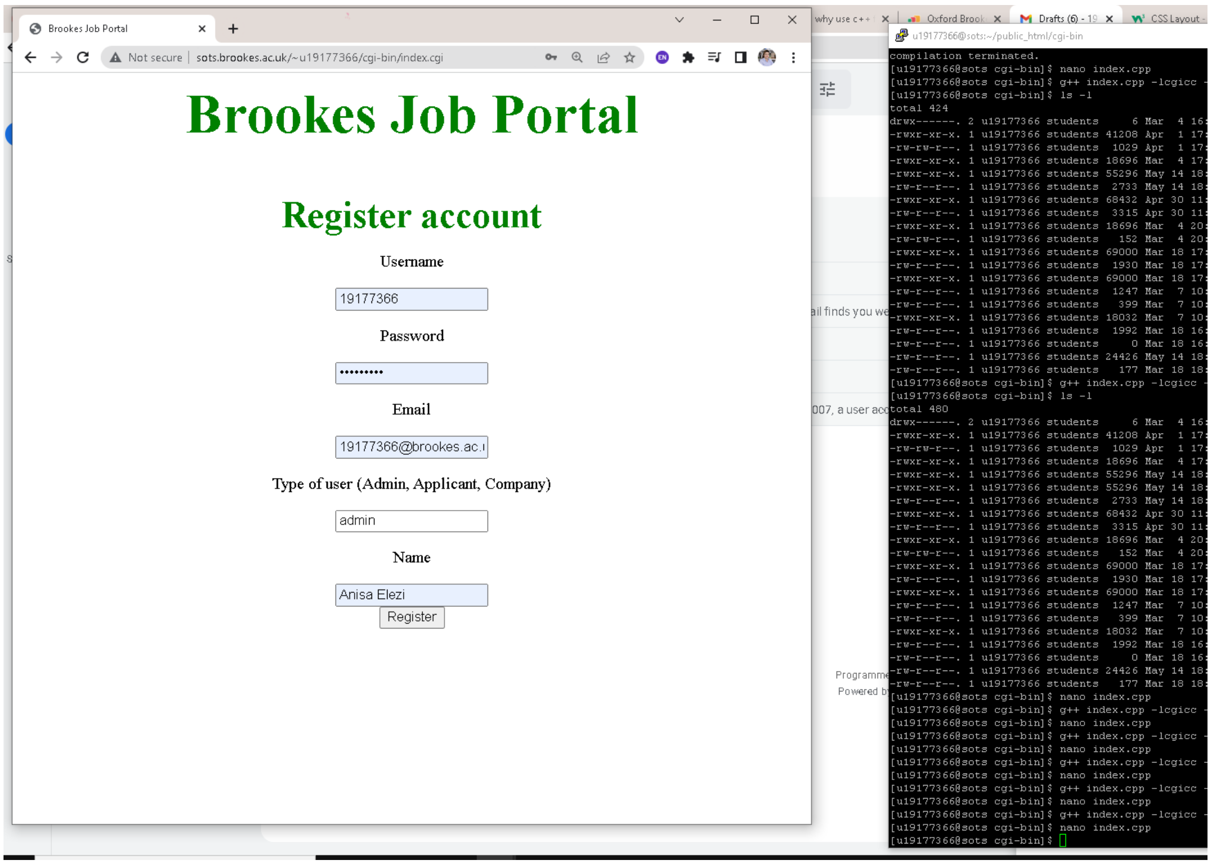
Type of user (Admin, Applicant, Company)

admin

Name

Anisa Elezi

Register


Brookes Job Portal
Not secure | sots.brookes.ac.uk/~u19177366/cgi-bin/index.cgi
Oxford Brook
Drafts (6) - 19
CSS Layout -
u19177366@sots:~/public_html/cgi-bin

Brookes Job Portal

Register account

Username: 19177366

Password: •••••••••

Email: 19177366@brookes.ac.u

Type of user (Admin, Applicant, Company): admin

Name: Anisa Elezi

Register

```
compilation terminated.
[u19177366@sots cgi-bin]$ nano index.cpp
[u19177366@sots cgi-bin]$ g++ index.cpp -lcgicc -
[u19177366@sots cgi-bin]$ ls -l
total 424
drwx------. 2 u19177366 students     6 Mar  4 16:
-rwxr-xr-x. 1 u19177366 students 41208 Apr  1 17:
-rw-rw-r--. 1 u19177366 students  1029 Apr  1 17:
-rwxr-xr-x. 1 u19177366 students 18696 Mar  4 17:
-rwxr-xr-x. 1 u19177366 students 55296 May 14 18:
-rw-r--r--. 1 u19177366 students  2733 May 14 18:
-rwxr-xr-x. 1 u19177366 students 68432 Apr 30 11:
-rw-r--r--. 1 u19177366 students  3315 Apr 30 11:
-rwxr-xr-x. 1 u19177366 students 18696 Mar  4 20:
-rw-rw-r--. 1 u19177366 students   152 Mar  4 20:
-rwxr-xr-x. 1 u19177366 students 69000 Mar 18 17:
-rw-r--r--. 1 u19177366 students  1930 Mar 18 17:
-rwxr-xr-x. 1 u19177366 students 69000 Mar 18 17:
-rw-r--r--. 1 u19177366 students  1247 Mar  7 10:
-rw-r--r--. 1 u19177366 students   399 Mar  7 10:
-rwxr-xr-x. 1 u19177366 students 18032 Mar  7 10:
-rw-r--r--. 1 u19177366 students  1992 Mar 18 16:
-rw-r--r--. 1 u19177366 students     0 Mar 18 16:
-rw-r--r--. 1 u19177366 students 24426 May 14 18:
-rw-r--r--. 1 u19177366 students   177 Mar 18 18:
[u19177366@sots cgi-bin]$ g++ index.cpp -lcgicc -
[u19177366@sots cgi-bin]$ ls -l
total 480
drwx------. 2 u19177366 students     6 Mar  4 16:
-rwxr-xr-x. 1 u19177366 students 41208 Apr  1 17:
-rw-rw-r--. 1 u19177366 students  1029 Apr  1 17:
-rwxr-xr-x. 1 u19177366 students 18696 Mar  4 17:
-rwxr-xr-x. 1 u19177366 students 55296 May 14 18:
-rwxr-xr-x. 1 u19177366 students 55296 May 14 18:
-rw-r--r--. 1 u19177366 students  2733 May 14 18:
-rwxr-xr-x. 1 u19177366 students 68432 Apr 30 11:
-rw-r--r--. 1 u19177366 students  3315 Apr 30 11:
-rwxr-xr-x. 1 u19177366 students 18696 Mar  4 20:
-rw-rw-r--. 1 u19177366 students   152 Mar  4 20:
-rwxr-xr-x. 1 u19177366 students 69000 Mar 18 17:
-rw-r--r--. 1 u19177366 students  1930 Mar 18 17:
-rwxr-xr-x. 1 u19177366 students 69000 Mar 18 17:
-rw-r--r--. 1 u19177366 students  1247 Mar  7 10:
-rw-r--r--. 1 u19177366 students   399 Mar  7 10:
-rwxr-xr-x. 1 u19177366 students 18032 Mar  7 10:
-rw-r--r--. 1 u19177366 students  1992 Mar 18 16:
-rw-r--r--. 1 u19177366 students     0 Mar 18 16:
-rw-r--r--. 1 u19177366 students 24426 May 14 18:
-rw-r--r--. 1 u19177366 students   177 Mar 18 18:
[u19177366@sots cgi-bin]$ nano index.cpp
[u19177366@sots cgi-bin]$ g++ index.cpp -lcgicc -
[u19177366@sots cgi-bin]$ nano index.cpp
[u19177366@sots cgi-bin]$ g++ index.cpp -lcgicc -
[u19177366@sots cgi-bin]$ nano index.cpp
[u19177366@sots cgi-bin]$ g++ index.cpp -lcgicc -
[u19177366@sots cgi-bin]$ nano index.cpp
[u19177366@sots cgi-bin]$ g++ index.cpp -lcgicc -
[u19177366@sots cgi-bin]$ nano index.cpp
[u19177366@sots cgi-bin]$ g++ index.cpp -lcgicc -
[u19177366@sots cgi-bin]$ nano index.cpp
[u19177366@sots cgi-bin]$ 
```

3.3. Testing and Security Audit

To test this web app, following is e testing plan with several **test cases** which include:

- *Test case description* (based on functional and non-functional requirements)
- *Test steps*
- *Test data*
- *Expected result*
- *Actual result*
- *Pass/Fail*

To audit this web application, is used ISO 27001 standard. Forms prepared on HTML and CGI scripts are used to ensure a non-comprisable web application. This is done through:

- Being secure runnable script, only with owner permission. In this case only u19177366 can run it on SOTS server.
- Another security feature is integrated SSL in SOTS, using secure shell connection.
- Ensure not such a complex script as it bring several problems.
- Check the communication with other programs on this system such as sending em@il. For this should be checked if this interaction is secure.
- Check if it is executed with Set User ID (SUID) privileges, and this should not be allowed in any case. To ensure it, it was changing the privileges with the following command:

  chmod u+s myfile
- Check if there is any input validation in form. This is done through several manner like: if-else condition, try-catch and also a special method that escape special chars used for SQL injection and XSS attack.

- Ensure if there is any special process used to check if the dynamic generated pages have not any malicious tag (in HTML code).
- Ensure to fulfil General Data Protection Regulation in privacy of user data. This is done using hashes SHA256.
- Ensure that incoming data from user are checked, in order to prevent any unauthorised access in other pages or even in database. This is done checking URL or even cookies and sessionID. Moreover, there is a special check if the input field contain any statement to cause buffer overflow or even any SQL injection.
- Ensure to use SSL to encrypt data during the transmission.

Conclusion

The aim of this coursework was to design, implement and test a secure web application realised on SOTS server using C++ and CGI web programming and MariaDB as a database. There was needed some knowledge on HTML and CSS as well.
It was such a challenge for me as far as I was not used to do everything from scratch. However, I went through all the steps from functional and non-functional requirements to documenting of this Brookes Job Portal web application. There is created a function to hash the input values with SHA256 and another method that implement whitelist to allow all the alphanumeric and special characters.

References

[1] CGI Programming on the World Wide Web

[2] https://owasp.org/www-community/HttpOnly

[3] C++ How to Program, Fourth Edition (Harvey M. Deitel, Paul J. Deitel)