CW2: Software Development of MarkManager

Student number: 19131287

MSc in Computing Science for Cyber Security

Module: NETW7008
Word count: 2818

---

1. Introduction

The task was to build a student record management software, in this instance called MarkManager, which the lecturers can use to store the grades students get within the modules.

2. Setup

The software was supposed to be implemented in the form of a web app using CGI with C++. As a supporting component, a MySQL database has been used. To improve the UI, CSS and JS have been used but are not part of the logic.

2.1 Development Environment

The software has been tested on a Virtual Machine (VM) running Windows 10 Build 18363.1440. The IDE Visual Studio 2019 Version 16.9.2 has been used.

As Webserver XAMPP Version 3.2.4 with APACHE and MySQL has been used. The MySQL setup will be explained in chapter 4.2.1. The Apache server has been set up to use cgi-bin as the index folder and HTTPS. The following changes have been made to the standard installation to achieve this:

Within the file: `C:\xampp\apache\conf\extra\httpd-ssl.conf` the line 124 has been changed to `DocumentRoot "C:/xampp/cgi-bin"` and the server name to `localhost:443`. Then, Code 1 has been added to the file.

```
<Directory "C:/xampp/cgi-bin">
    AllowOverride All
    SSLOptions +StdEnvVars
    Options Indexes FollowSymLinks Includes ExecCGI
    Require all granted
</Directory>
```

Code 1: Part of the httpd.conf file

By executing the C:\xampp\apache\makecert.bat and choosing the PEM pass phrase: “1234”, a private key and server certificate have been created. The common name

localhost has been selected. The connection is still showing as insecure within the browser, but this derives from the fact that it is a self-issued certificate. This setup does not automatically redirect to HTTPS, but this is not important for a development environment.

3. Requirements

As part of this task, the following five functional and six non-functional requirements concerning the system have been given.

FR1	There are two kinds of users: lecturers and administrators. Both can register an account and set a password.
FR2	Lecturers can see a list of their modules and a list of the students on each module. They can also enter and change marks.
FR3	Administrators can assign lecturers to modules and students to modules. There is only one administrator account.
FR4	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 to a "mail spool" text file representing all the emails that have been sent.
FR5	The administrator account, in addition to the protections of FR4, must also be authenticated by a "hardware" token, which should be implemented as a piece of challenge-response software.

Table 1: Functional Requirements

NFR1	The system must be developed in C/C++. You may use CGI to interact with the web pages. You may use the C/C++ CGI libraries, which have been installed on SOTS, if you are using SOTS. Here is one of many tutorials on them: https://www.tutorialspoint.com/cplusplus/cpp_web_programming.htm
NFR2	The system must be robust and secure. Specifically, it should be capable of mitigating many kinds of attacks covered in the module, as detailed in the marking scheme. SSL must not be the sole means of preventing these attacks.
NFR3	The system must be designed with maintainability, security and reliability in mind and according to best practice in designing and implementing secure software. Defensive software practices should be used throughout

NFR4	Your code should be commented and have sensible and consistent naming
NFR5	The system should be responsive and easy to use
NFR6	You may use cryptographic libraries if you wish.

Table 2: Non-functional requirements

All requirements have been fulfilled, and the functional ones are demonstrated within video 1 of the appendix. In the following chapters, it will be explained in detail how the non-functional requirements have been fulfilled.

4. System Design

4.1 UI/UX

The UI has been designed in a minimalist neomorphic style with a green accent. This makes the UI easily understandable as elements pop in the eye. The design of the elements itself has been made using Figma. The elements are aligned vertically and adaptable for usage on a variety of screen sizes.

To achieve a good UX, the goal was to keep the number of elements on the screen at a bare minimum to ensure that the user understands the purpose of each page quickly. The page hierarchy has five levels and up to three levels after the login, as shown in Figure 1.

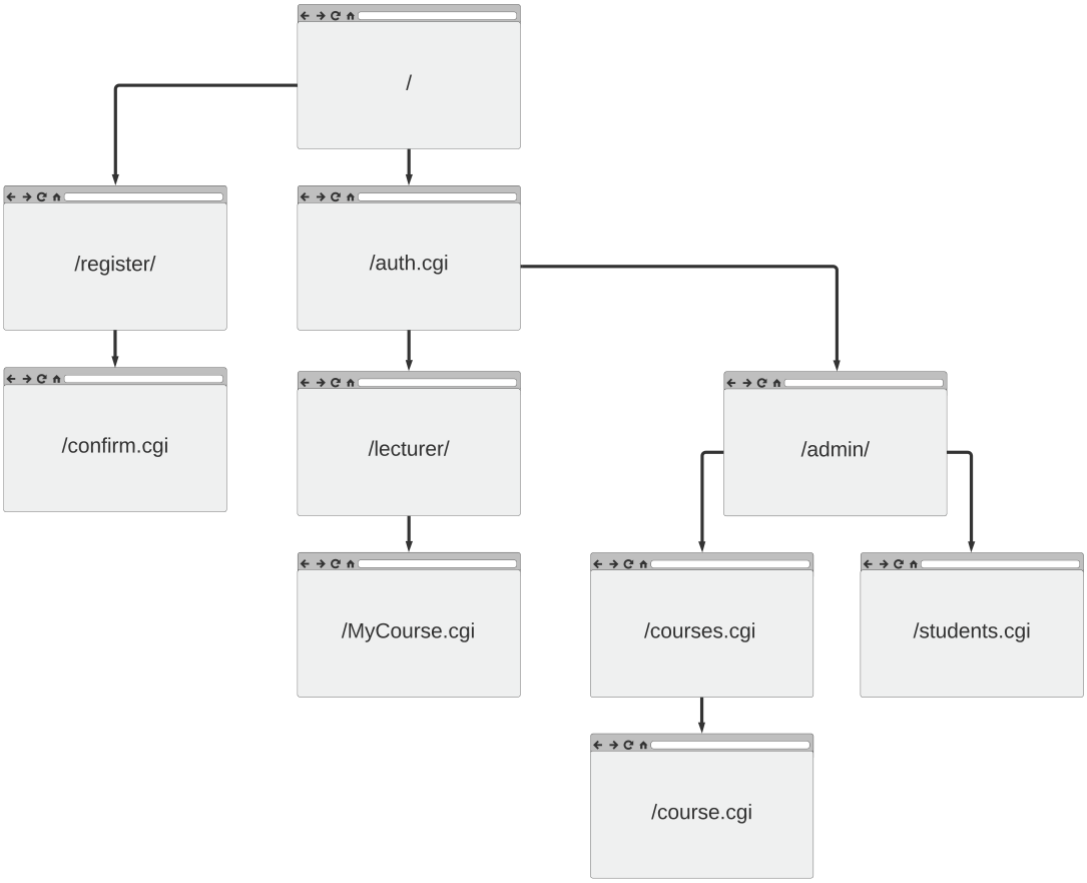

Figure 1: Page Hierarchy

On each, there is a back button on the top left, which lets the user go back in the hierarchy once logged in.

The choices within the UI/UX design have been made to fulfil NFR5.

4.2 System Components

The system consists out of 10 pages that are implemented as a CGI file in C++. All pages have access to a MySQL database, except index.cgi and /register/index.cgi. All the pages use CSS files within the /css folder, and the /register/confirm.cgi page also uses JS files within the /js folder to dynamically create a QR code on the client-side.

```
cgi-bin
├── admin
│   ├── courses
│   │   ├── course.cgi
│   │   └── index.cgi
│   ├── index.cgi
│   └── students.cgi
├── auth.cgi
├── css
│   ├── list.css
│   ├── main.css
│   └── qrcode.css
├── index.cgi
├── js
│   ├── qrcode.js
│   └── totp_qrcode.js
├── lecturer
│   ├── index.cgi
│   └── myCourse.cgi
└── register
    ├── confirm.cgi
    ├── img
    │   ├── error.svg
    │   └── freeotp.svg
    └── index.cgi
```

Figure 2: cgi-bin file structure without .dll files

The file structure is organised by component functionality and permission level. This contributes to the maintainability aspect of NFR3. Unfortunately, the DLL files could not be organised in this manner although placing all DLL files in a single folder would further improve the structure.

4.2.1 Database Structure

The database has been designed normalised and relational. The database has been developed with MySQL Workbench, and the visual structure can be seen in Figure 3.

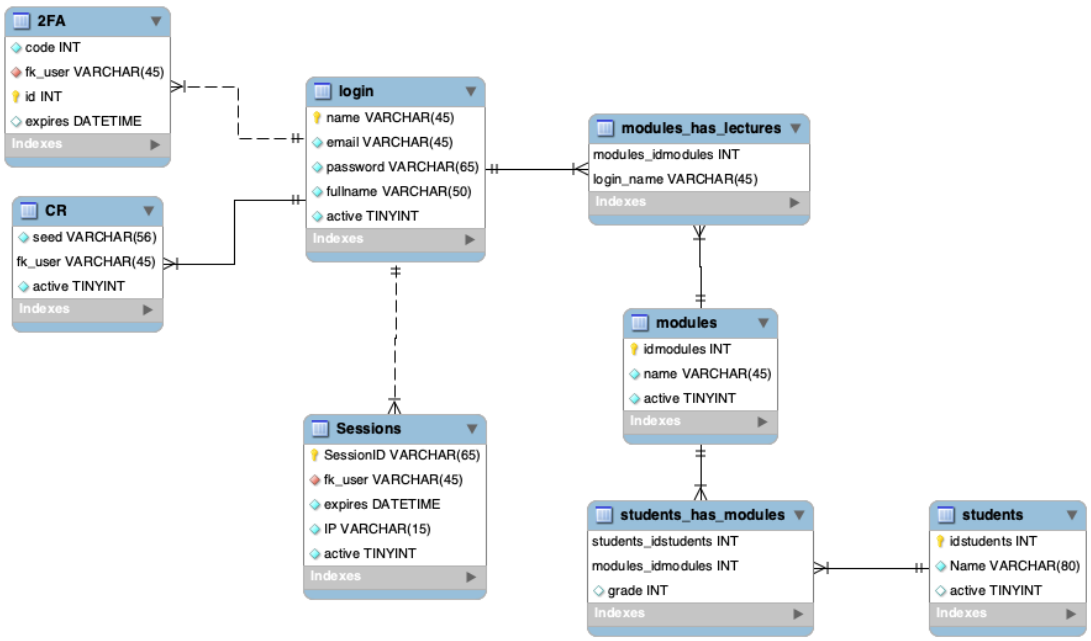

Figure 3: MySQL Workbench export of the database

The normalised and non-redundant design chosen for this database to ensure a maintainable and reliable database contributes to NFR3. One improvement that should be implemented in a productive system, that would increase the reliability and responsiveness of the system, is a clean-up daemon. This daemon would delete expired

sessions and expired 2FA codes from the database and could be implemented in the form of a python script.

5. Security Justification

Many steps have been taken to ensure the system's safety, which can be categorised into six categories.

5.1 Database Security

5.1.1 Principal of least privilege

The following five database user accounts have been created with the least UPDATE, INSERT and DELETE privilege necessary: user, admin, login, stateful and register. This strongly limits the possibilities of SQL injections.

DB User	Task
user	Handles the operations done by lecturers
admin	Handles the operations done by the admin
login	Handles the login and session creation
stateful	Checks the session validity
register	Handles the registration of new users

Table 1: DB user separation

```
CREATE USER 'user'@'localhost' IDENTIFIED BY 'd9pifetoyesad2cekipoyolis';
GRANT SELECT ON MarkManager.modules_has_lectures TO 'user'@'localhost';
GRANT SELECT ON MarkManager.students_has_modules TO 'user'@'localhost';
GRANT UPDATE(grade) ON MarkManager.students_has_modules TO 'user'@'localhost';
GRANT SELECT ON MarkManager.students TO 'user'@'localhost';
GRANT SELECT ON MarkManager.modules TO 'user'@'localhost';

CREATE USER 'admin'@'localhost' IDENTIFIED BY 'nocikocofot9com4cif1boq6t';
GRANT SELECT ON MarkManager.modules_has_lectures TO 'admin'@'localhost';
GRANT SELECT ON MarkManager.students_has_modules TO 'admin'@'localhost';
GRANT INSERT ON MarkManager.modules_has_lectures TO 'admin'@'localhost';
GRANT INSERT ON MarkManager.students_has_modules TO 'admin'@'localhost';
GRANT DELETE ON MarkManager.modules_has_lectures TO 'admin'@'localhost';
GRANT DELETE ON MarkManager.students_has_modules TO 'admin'@'localhost';
GRANT SELECT ON MarkManager.students TO 'admin'@'localhost';
GRANT SELECT ON MarkManager.modules TO 'admin'@'localhost';
GRANT SELECT(name,fullname,active) ON MarkManager.login TO 'admin'@'localhost';
GRANT INSERT ON MarkManager.students TO 'admin'@'localhost';
GRANT UPDATE(active) ON MarkManager.students TO 'admin'@'localhost';
GRANT INSERT ON MarkManager.modules TO 'admin'@'localhost';
GRANT UPDATE(active) ON MarkManager.modules TO 'admin'@'localhost';

CREATE USER 'login'@'localhost' IDENTIFIED BY 'n9ras1yay4aey8yoterec2vex';
GRANT SELECT ON MarkManager.TOTP TO 'login'@'localhost';
GRANT SELECT ON MarkManager.2FA TO 'login'@'localhost';
GRANT INSERT ON MarkManager.2FA TO 'login'@'localhost';
GRANT SELECT ON MarkManager.login TO 'login'@'localhost';
GRANT SELECT ON MarkManager.Sessions TO 'login'@'localhost';
GRANT INSERT ON MarkManager.Sessions TO 'login'@'localhost';
GRANT UPDATE(active) ON MarkManager.Sessions TO 'login'@'localhost';

CREATE USER 'stateful'@'localhost' IDENTIFIED BY 'dajuyihobumasopin7qefiroa';
GRANT SELECT ON MarkManager.Sessions TO 'stateful'@'localhost';

CREATE USER 'register'@'localhost' IDENTIFIED BY 'qaciy0t2kif5cul5d5quwanus';
GRANT SELECT ON MarkManager.login TO 'register'@'localhost';
GRANT INSERT ON MarkManager.login TO 'register'@'localhost';
GRANT UPDATE ON MarkManager.login TO 'register'@'localhost';
GRANT UPDATE(active) ON MarkManager.login TO 'register'@'localhost';
GRANT SELECT ON MarkManager.TOTP TO 'register'@'localhost';
GRANT INSERT ON MarkManager.TOTP TO 'register'@'localhost';
GRANT UPDATE(active,seed) ON MarkManager.TOTP TO 'register'@'localhost';
GRANT SELECT ON MarkManager.2FA TO 'register'@'localhost';
GRANT INSERT ON MarkManager.2FA TO 'register'@'localhost';
```

Figure 4: Database user permissions

5.1.2 Storage of encrypted passwords

Only the register and login database users have access to the login passwords. Still, a leak could potentially happen. Therefore, the passwords are only saved as a SHA256 hash within the database. This makes a password leak almost useless. To further enhance the password encryption, the password could be salted before encrypting. The best-case implementation would consist out of two salts. One global salt is stored within the program and not in the database, and one salt, unique for each user/password, is stored in the DB. Thereby, every password hash would include the global salt, and each password would have one unique salt. As a consequence, it would be necessary to create a rainbow table for each password hash, and having access not only to the password hashes, but also to the salt table within the DB and the global salt within the program.

5.2 Input Validation

5.2.1 Prevention of SQL injection

Every value that the client provides will be stripped of non-alphabetical, non-numerical and non-space characters. Only alphanumerical character and the `@ _ .` characters are allowed within the string before inserted into a SQL query. This way, a SQL code injection will be prevented. At the same time, a Server-side XSS attack will be prevented. This strict restriction is chosen because many characters can be misused, as shown by (PortSwigger, no date). An exception to this is the password, as the input is hashed before use.

5.2.2 Semantic permission validation

Although the permission of the database users is very limited, not every permission given to the database-user 'user' is automatically given to the logged-in lecturer. Therefore, whenever an UPDATE is called by the logged-in lecturer, the input will be semantically checked to ensure the lecturer has the permission to make this data transformation.

5.3 Session Management

5.3.1 Session ID creation

The Session ID is created by concatenating the username, a random value, the DateTime, the client IP address, and the random value. The concatenation is then hashed with SHA256.

5.3.2 Cookie management and security

Every session has a lifetime of 10min. After 10min the SessionID will be expired, and the user will have to login again. The Cookies have the same lifetime. Furthermore, the Cookies are set HTTPOnly to prevent XSS attacks.

5.3.3 IP and Username check

Each SessionID is linked to a username within the database (Figure 3). Furthermore, the SessionID is linked to the IP address the session was set up with. To verify a Session, the program does not only check if the SessionID is valid but also checks the client ID address and the username provided within the cookie. Only if all three checks are passed, the request will be granted.

5.4 Login Process

5.4.1 Normal User

A regular lecturer will need to use 2FA in order to login into the system. The first authentication is the password set by the user. The second authentication consists of an OTP code that will be sent to the email address associated with the account. The code will only be valid for 10min and then expires.

5.4.2 Admin User

The admin will go through a 3FA in order to access the system. The first two authentications are equal to the regular user authentication. The additional

authentication is a TOTP. The admin will need to enter the code generated by the token generator (for example, Google Authenticator) to access the system.

5.5 Registration Process

Both admin and regular users will be asked to enter their full name, email, password and username during the registration.

The regular user's account will be activated after the user confirmed its email address by entering an OTP, which will be sent to the email inputted.

The admin account will only be activated if the user enters the OTP sent by email and a TOTP generated by a token app. There will be a QR code on the web page that can be scanned with a token generator app as Google Authenticator or Authy. The app will receive a seed by scanning the QR code, which it will use to generate the TOTP in the future without connecting to the server.

5.6 Exception Handling

5.6.1 Expired Session

In case the sessionID is expired, or the sessionID does not fit the username provided, or the IP address is not correct, the user will be redirected to the login page with a message telling him the session is expired.

5.6.2 Without privilege

In case the user tries to access a page that he has no privileges to access, he will be redirected to the main page of his account.

5.6.3 Without login

Users who try to access any page but the login page without providing a session id or username will be redirected to the main page with a message asking them to log in.

5.6.4 Wrong input

Should the user try to enter a wrongly typed input, he would only be redirected to the index page with a generic message, saying that the request was not allowed or valid. Should the input be correctly typed and syntactically valid, but a wrong value entered, such as wrong login information, the user would get a more direct message on the page without being redirected while not giving away too much information about the system.

6. Possible improvements

The outcome of this project fulfils the security requirements from the original task. Additionally, an HTTP to HTTPS automated redirect should be implemented in a productive environment.

Furthermore, the system does not differentiate between data provided over a GET or POST. This could be exploited by creating a URI with the correct parameters to change the system and then tricking a login user into clicking onto this link. A possible scenario would be a student sending an email to a lecturer containing a link to improve the

students' grades. This imposes a threat to data integrity. Therefore, the system should be improved by adding the ability to differentiate between POST and GET requests.

A second vulnerability left open within the system is DOS and brute force attacks. However, a DOS attack should not be prevented by the system but by an IDS between the system and the internet.

A brute force attack imposes a significant threat to the system. As all three authentication methods, password, OTP and TOTP, are vulnerable to this. The system does not give a limit on attempts to enter a password, OTP or TOPT. Therefore, an attacker could brute force the password. This could take a bit of time, but as the password does not change, this does not present a problem. After gaining the password, the attacker would then need to brute force the OTP and TOPT. The OTP has a time limit of 10min and 9999 possibilities. If an attacker sends 16 requests per second, he could test out all options. This can be easily achieved with a python script. Code 1 could Brute force the OTP in about 10min on the test VM.

```python
import requests
import threading

authurl = 'http://localhost/auth.cgi'
username = str(input("Enter Username\n"))
session = str(input("Enter Session ID\n"))
found = True

cookies = {'username': username,'SessionID': session}

def find_code(n):
    i = n*200
    global found
    for j in range(i,i+200):
        data = {'2FA': j }
        requests.post(authurl, data = data, cookies=cookies)
        print(j)
    print("Thread "+str(n)+" finished")

for i in range(50):
    threading.Thread(target=find_code, args=(i,)).start()
```

Code 2: Brute Force OTP Python Script

The 2-factor authentication for the lecturer accounts can therefore not be considered safe, which proves that 2FA could be insecure if not correctly implemented.

The same principle can be made with the authentication for the admin account. Here the TOTP changes every 30seconds, but the OTP is still valid for 10minutes. Nevertheless, the user does not get any feedback if the OTP is correct unless the TOTP is also correct. Therefore 10^{10} combinations would need to be tested within a 30second time windows. This, although theoretically not impossible, can be considered practically impossible.

Assuming that this immense number of requests are not registered and blocked by the IDS, the system should limit the amounts of login attempts a computer can make, possibly 5. Further, the system could allow one account only to be accessed by two computers at the same time, within a 10min time window. Thereby, taking away the possibility of a distributed brute force attack.

The last adjustment that should be made is imposing password requirements. This requirement should be monitored by JavaScript on the client-side in real-time to ensure usability and then checked again on the server-side before accepting and storing the password during the registration process.

7. Conclusion

The system can be protected against SQL injections and XSS. Furthermore, strict session management has been implemented. A login mechanism has been implemented with 2FA and 3FA, which lays the strong foundation for a secure system. Issues such as brute force detection have not yet been addressed within the solution, but the security risk opposed by them has been assessed. Overall, all functional and non-functional requirements have been fulfilled.

References & Bibliography

Munteanu Marian (2013) *how to send email with c++*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/19767431/how-to-send-email-with-c (Accessed: 24 April 2021).

PortSwigger (no date) *SQL Injection: Bypassing Common Filters*, *PortSwigger*. Available at: https://portswigger.net/support/sql-injection-bypassing-common-filters (Accessed: 23 April 2021).

Stefan Trost (2018) *XAMPP: SSL/HTTPS für lokale Projekte einrichten*, *askingbox.de*. Available at: https://www.askingbox.de/tutorial/xampp-ssl-https-fuer-lokale-projekte-einrichten (Accessed: 23 April 2021).

Appendix 1: Presentation of the Program Functions (Video)

A 5min presentation of the program functions has been made and can be found here:

https://drive.google.com/file/d/1e6fVRs75cVxFo05NGJ2vlWAuB5PvG7Ej/view?usp=sharing

Appendix 2: Presentation of the Program Code (Video)

A 5min presentation of the program code has been made and can be found here:

https://drive.google.com/file/d/1AcJEhSXdixj78bc79QY9E_1An1YXgUJ3/view?usp=sharing