Technical Report

WebWarden

Title: Web Warden

Student: Sean Brady

Number: x14715859

Email: x14715859@student.ncirl.ie

BSc (Hons) in Computing

Cyber Security

13 May 2018

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this Technical document is to describe and depict how the WebWarden software is expected to work. The main consumers the software will be aimed at are parents with children aged between 4-15 years, whose children will have access to multiple web browsing devices including the home network.

1.2 Project Scope

The scope of the project is to create a secure web filtering software for home network use that offers real-time web access control. An AWS database server will be used to store all the information. A Raspberry Pi 3 B will function as the proxy and contain the java code that will employ the web filtering techniques. To configure and control the proxy, a companion app developed on Android, will be created.

2 User Requirements Definition

To gather information on user requirements for this project, I released an anonymous survey through various social media sites. The survey contained 9 multiple choice questions which all centred around web content filters. I received a total of 28 responses. These highlighted the aspects of the project that people valued and disliked. As you can see from Figure 1, I noted the opinion of the survey takers regarding the real-time control of my web filtering software. The verdict, overall, was extremely positive with 96% saying it was a valid feature. Figure 2 shows that 85% value the idea of being able to monitor all web browsing devices. Finally, 93% thought the companion app was a useful feature. These results helped validate the usefulness of the niche features my web filtering software offered.

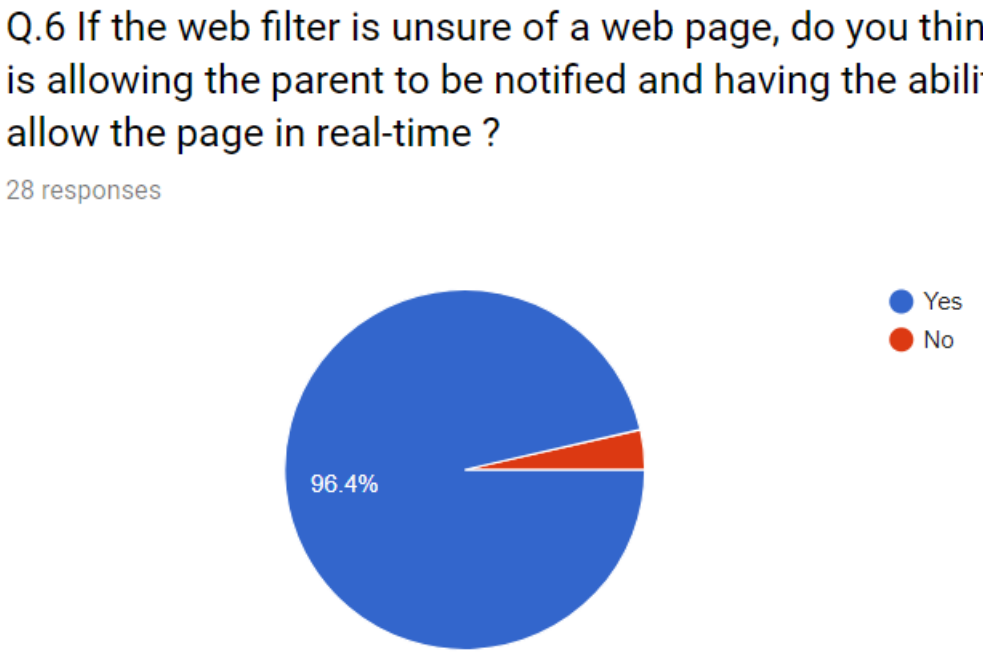

Figure 1

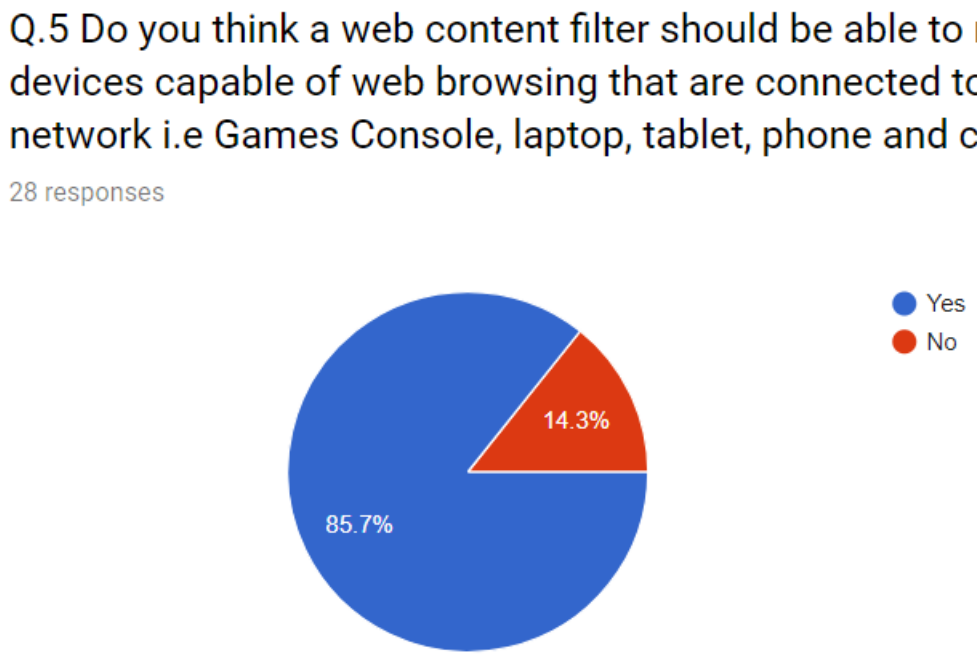

Figure 2

3 Functional Requirements

This part of the document specifies all the functional requirements in ranked order for the WebWarden software.

3.1 User Class – Home network admin

ID – FR1

Title: Pairs with Raspberry Pi proxy.

Description: Admin user can pair the WebWarden app with the proxy.

ID – FR2

Title: Web access control in real-time.

Description: Admin user using the WebWarden app can block or allow access to unknown websites in real-time.

ID – FR3

Title: Create account.

Description: The admin user can create an account for the WebWarden app.

ID – FR4

Title: Sign in.

Description: The admin user can log into their WebWarden account.

ID – FR5

Title: Sign-out

Description: The admin user can sign-out of their WebWarden account.

ID – FR6

Title: Generate access control password.

Description: Using the WebWarden app, the admin user can generate their unique access control password.

ID – FR7

Title: Database communication

Description: The WebWarden app can connect and communicate with the database correctly.

ID – FR8

Title: Proxy password.

Description: The admin can set a secure pairing password for the proxy.

ID – FR9

Title: Configure pre-banned websites.

Description: The admin can create, edit and delete the pre-banned site list using the WebWarden app.

ID – FR10

Title: Edit account.

Description: The admin user can edit their WebWarden account within the app.

3.2 User Class – Proxy

ID – FR11

Title: Block websites.

Description: The proxy will block banned sites from the pre-banned site list from being accessed and if the content score reaches the maximum limit.

3.3 User Class – Home network user

ID – FR12

Title: Connect to proxy

Description: The user can easily connect to the proxy.

4 Use cases

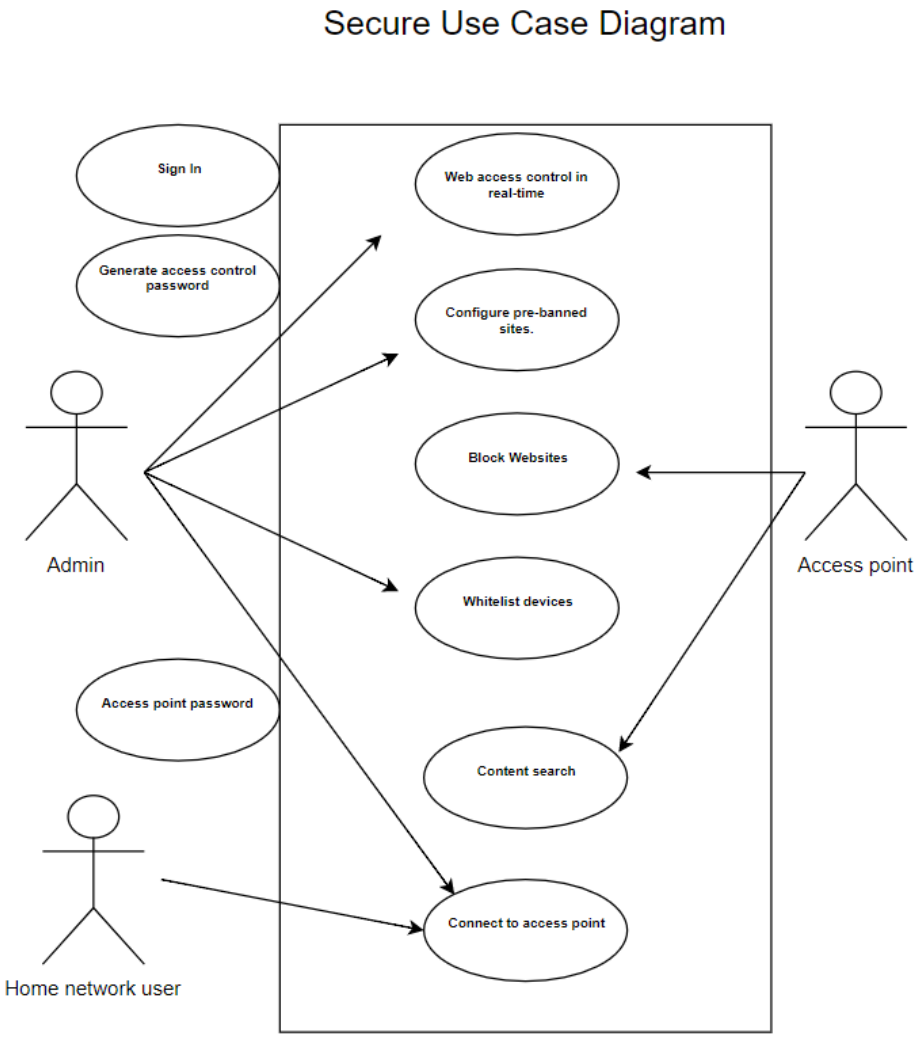

Figure 3

Use case: Web access control in real-time

Brief description: This use case describes how the admin of the home network can block or allow access to a website in real-time using the WebWarden app.

Actors: Home network administrator

Pre-conditions:

- The admin user is logged into their WebWarden account.
- The home network user is connected to the proxy.
- The admin user's WebWarden app is paired with the proxy.
- The proxy is connected to the home router.
- Both the user and admin have a working internet connection.

Basic Flow:

1. The use case begins when the home network user clicks on a URL link.
2. The proxy verifies if the URL is banned. [**E1**]
3. The proxy scans content of webpage.
4. The proxy scores content of page.
5. The proxy sends information and notification to WebWarden app.
6. The admin user clicks on the notification.
7. The system displays the site name.
8. The system prompts the admin user to block or grant access to the site.
9. The admin user selects the allow button. [**A1**]
10. The system prompts the user to enter in their access control password.
11. The admin user enters in their access control password.
12. The system validates the password. [**A2**], [**E4**]
13. The system returns the result back to the proxy.
14. The proxy allows the website to load.
15. The use case ends successfully.

Alternate Flow:

A1. *Selects Block*

If in step 9, the user selects the block button then:

1. The system returns the result back to the proxy.
2. The proxy blocks the website from loading.
3. The proxy displays “site blocked” message to home network user.
4. The use case ends successfully.

A2: *Incorrect access control password*

If at basic flow step 12, the admin user enters in the password incorrectly then:

1. The system displays the message “Incorrect password”.
2. The use case continues from basic flow step 11.

Exceptional Flow:

E1. *Site Banned*

If in step 2, the site is already banned then:

1. The proxy will block access to the site.
2. The use case ends unsuccessfully.

E2. *App Crash*

If at any stage during the use case the app crashes, then:

1. The use case ends with a failure condition.

E3. *App shutdown*

If at any stage during the use case the admin user shutdowns the app then:

1. The use case ends unsuccessfully.

E4. *Incorrect password limit*

If at basic flow step 12, the admin user enters the password incorrectly for 5 times then:

1. The system displays message “Password attempts exceeded, site will be blocked”.
2. The system returns the result back to the proxy.
3. The proxy blocks the website from loading.

4. The use case ends with a failure condition.

E5. *Proxy shutdown*

If at any stage before the use case is complete, the proxy loses power then:

1. The system displays the message “Connection to proxy lost”.
2. The use case ends with a failure condition.

Post Conditions

Successful Completion:

The admin user has successfully blocked or allowed access to a website.

Failure Condition:

The error logs are updated accordingly.

Use case: Sign In

Brief description: The admin user is signing into their WebWarden app account.

Actors: Home network administrator

Pre-conditions:

- The admin user has a working internet connection.
- The admin user’s device is connected to proxy.
- The admin user’s WebWarden app is paired with the proxy.

Basic Flow:

1. The use case begins when the admin user launches the WebWarden App.
2. The system prompts the admin user to sign up or sign in.
3. The admin user selects sign in.
4. The system prompts the admin user to enter in their username and password.
5. The admin user enters in their username.
6. The admin user enters in their password.
7. The admin user selects the sign in button.
8. The system connects to the database.
9. The system verifies username and password. [**A1**]
10. The system signs the user into their account.
11. The system presents the main app screen.
12. The use case ends successfully.

Alternate Flow:

A1. *Incorrect details.*

If the admin user entered in the incorrect username or password, then:

1. System displays “incorrect password or username” message.
2. The use case continues from basic flow step 4.

Exceptional Flow:

E1. *App Crash*

If at any stage during the use case the app crashes, then:

1. The use case ends with a failure condition.

E2. *App shutdown*

If at any stage during the use case the admin user shutdowns the app then:

1. The use case ends unsuccessfully.

Post Conditions

Successful Completion:

The admin user has successfully signed into their WebWarden account.

Failure Condition:

The error logs are updated accordingly.

Use case: Generate access control password

Brief description: The admin user generates their unique access control password.

Actors: Home network administrator

Pre-conditions:

- The admin user is signed into their account.
- The admin user has a working internet connection.
- The admin user's device is connected to the home router.
- The admin user's WebWarden app is paired with the proxy.

Basic Flow:

1. This use case begins when the admin user is at the main screen of the WebWarden app.
2. The admin user clicks the "access control" button.
3. The system navigates to the access control screen.
4. The admin user clicks "generate access control password". [**A1**]
5. The system generates a new access control password for the admin user.
6. The system connects to the database.
7. The system updates the database.
8. The use case ends successfully.

Alternate Flow:

A1: *Overwrite access control password*

If at step 4, the admin user already has an access control password then:

1. The system displays message "Overwrite access control password?"
2. The admin user selects yes button prompt.
3. The system prompts the admin user to enter in their old access control password and their sign in password.
4. The admin user enters in their old access control password.
5. The admin user enters in their sign in password.
6. The use case continues at basic flow step 5.

Exceptional Flow:

E1. *App Crash*

If at any stage during the use the app crashes, then:

1. The use case ends with a failure condition.

E2. *Database connection lose*

If at any stage during basic flow steps 6-7 the app loses connection to the database, then:

1. The system displays message “Error occurred connecting to database”.
2. The use case ends with a failure condition.

E3. *App shutdown*

If at any stage during the use case the admin user shutdowns the app then:

1. The use case ends unsuccessfully.

Post Conditions

Successful Completion:

The admin user has successfully created a new access control password.

Failure Condition:

The error logs are updated accordingly.

Use case: Proxy password

Brief description: The admin user using the WebWarden app sets the pairing password for the proxy.

Actors: Home Network Administrator

Pre-conditions:

- The admin user is signed into their account.
- The admin user has a working internet connection.
- The admin user’s device is connected to home network.
- The admin user’s WebWarden app is paired with the proxy.

Basic Flow:

1. This use case begins when the admin user is at the main screen of the WebWarden app.
2. The admin user selects the “access control” button.
3. The system navigates to the access control screen.
4. The admin user selects the “Proxy configuration” button.
5. The system navigates to the Proxy configuration screen.
6. The admin selects the “create proxy password” button.
7. The user enters in a new proxy password.
8. The system prompts the user to enter in their access control password.
9. The user enters in their access control password.
10. The system validates password.
11. The system connects to database.
12. The system edits database and saves new proxy password.
13. The use case ends successfully.

Exceptional Flow:

E1. *App Crash*

If at any stage during the use the app crashes, then:

1. The use case ends with a failure condition.

E2. *Database connection loss*

If at any stage during basic flow steps 11-12 the app loses connection to the database, then:

1. The system displays message “Error occurred connecting to database”
2. The use case ends with a failure condition.

E3. *App shutdown*

If at any stage during the use case the admin user shutdowns the app, then:

1. The use case ends unsuccessfully.

Post Conditions

Successful Completion:

The admin user successfully created a new pairing password for the proxy.

Failure Condition:

The error logs are updated accordingly.

Use case: Configure pre-banned websites

Brief description: This use case describes how the admin user, using the WebWarden app, can configure pre-blocked websites for the proxy.

Actors: Home Network Administrator

Pre-conditions:

- The admin user is signed into their account.
- The admin user has a working internet connection.
- The admin user’s device is connected to home network.
- The admin user’s WebWarden app is paired with the proxy.

Basic Flow:

1. This use case begins when the admin user is at the main screen of the WebWarden app.
2. The admin user selects the “Web control” button.
3. The system navigates to the Web control screen.
4. The admin user selects the “Block Website” button. [**A1**]
5. The system navigates to the Block Website screen.
6. The admin user enters in a website URL in the blank field provided.
7. The admin user selects the “add” button.
8. The system connects to database.
9. The system edits the database.
10. The use case ends successfully.

Alternate Flow:

A1: *Remove Banned Website*

If at basic flow step 4, the admin user selects “Remove banned site” then:

1. The system prompts the admin user for their access control password.
2. The admin user enters in their access control password.
3. The system verifies password.
4. The system displays the banned site list.
5. The admin user selects the site to remove.
6. The use case continues at basic flow step 8.

Exceptional Flow:

E1. *App Crash*

If at any stage during the use the app crashes, then:

1. The use case ends with a failure condition.

E2. *Database connection lose*

If at any stage during basic flow steps 8-9 the app loses connection to the database, then:

1. The system displays message “Error occurred connecting to database”.
2. The use case ends with a failure condition.

E3. *App shutdown*

If at any stage during the use case the admin user shutdowns the app then:

1. The use case ends unsuccessfully.

Post Conditions

Successful Completion:

The admin user successfully configured the pre-banned site list.

Failure Condition:

The error logs are updated accordingly

Use case: Block websites

Brief description: The proxy will block access to websites.

Actors: Proxy

Pre-conditions:

- Home network user connected to proxy.
- The proxy is connected to the home router.
- Home network user connected to the internet.

Basic Flow:

1. This use case begins when a home network user clicks on a URL link.
2. The proxy retains the website name and URL.
3. The proxy connects to the database.
4. The proxy searches banned website database for URL.
5. The proxy finds a match. [**A1**]
6. The proxy blocks the site from loading.
7. The proxy displays message “Website is blocked” to the home network user.
8. The use case ends successfully.

Alternate Flow:

A1: *No match*

If at basic flow step 5, the proxy does not find a match then:

1. The proxy will scan the content of the website.
2. The proxy will score inappropriate content on the site.
3. The proxy ends content search when maximum website score is reached. [**E1**]
4. The use case continues at basic flow step 6.

Exceptional Flow:

E1: *Score to low*

If at A1 step 3, the score is too low for the proxy to block automatically then:

1. The proxy will send notification to WebWarden app.
2. The use case ends unsuccessfully.

E2. *Database connection lose*

If at any stage during basic flow steps 3-4 the proxy loses connection to the database, then:

1. The use case ends with a failure condition.

Post Conditions

Successful Completion:

The app successfully blocks access to an inappropriate website.

Failure Condition:

The error logs are updated accordingly.

Use case: Create Account

Brief description: The user creates an account on the WebWarden app.

Actors: Home network administrator

Pre-conditions:

- The admin user has a working internet connection.
- The admin user's device is connected to proxy.
- The admin user's WebWarden app is paired with the proxy.

Basic Flow:

1. The use case begins when the admin user clicks into the WebWarden App.
2. The system prompts the admin user to sign up or sign in.
3. The admin user selects the sign-up button.
4. The system navigates to the sign-up screen.
5. The admin user enters in a unique username in the field provided.
6. The admin user enters in a unique sign-in password in the field provided.
7. The system validates password. [**A1**]
8. The system connects to the database.
9. The system validates the username.
10. The system adds the new admin account to the database.
11. The use case ends successfully.

Alternate Flow:

A1: *Invalid password*

If at basic flow step 7, the system verifies that the admin user has entered in an invalid password then:

1. The system displays the message "Invalid password, password must contain more than 8 characters and at least one numeric value."
2. The use case continues at basic flow 6.

A2: *Insecure password*

If at basic flow step 7, the system verifies that the admin use has entered their username as the password then:

1. The displays the message "Insecure password, username may not be used as valid password".
2. The use case continues at basic flow 6.

Exceptional Flow:

E1. *App Crash*

If at any stage during the use the app crashes, then:

1. The use case ends with a failure condition.

E2. *App shutdown*

If at any stage during the use case the admin user shutdowns the app, then:

1. The use case ends unsuccessfully.

E2. *Database connection lose*

If at any stage during basic flow steps 8-10 the app loses connection to the database, then:

1. The system displays message “Error occurred connecting to database”.
2. The use case ends with a failure condition.

Post Conditions

Successful Completion:

The admin user has successfully created a new account for the WebWarden app.

Failure Condition:

The error logs are updated accordingly.

Use case: Pairs with Raspberry Pi proxy.

Brief description: The admin user pairs their WebWarden app with the proxy.

Actors: Home network administrator

Pre-conditions:

- The admin user has a working internet connection.

Basic Flow:

1. The use case begins when the admin user launches the WebWarden app for the first time.
2. The system displays the Pair device screen.
3. The admin user selects the “Pair App” button.
4. The system communicates with the database.
5. The system prompts the admin user to enter in the proxy password.
6. The admin user enters in the unique proxy password. [**A1**]
7. The system pairs with the proxy.
8. The system connects to the database.
9. The system adds itself to the paired devices table.
10. The use case ends successfully.

Alternate Flow:

A1: *Default password*

If at basic flow step 6, the unique proxy password has not been configured then:

1. The admin user enters in the default pairing password set for the proxy.
2. The use case continues at basic flow step 7.

Exceptional Flow:

E1. *App Crash*

If at any stage during the use the app crashes, then:

1. The use case ends with a failure condition.

E2. *App shutdown*

If at any stage during the use case the admin user shutdowns the app, then:

1. The use case ends unsuccessfully.

E2. *Database connection lose*

If at any stage during basic flow steps 8-9 the app loses connection to the database, then:

1. The system displays message “Error occurred connecting to database”.
2. The use case ends with a failure condition.

E3. *Proxy shutdown*

If at any stage before the use case is complete, the proxy loses power then:

1. The system displays the message “Connection to proxy lost”
2. The use case ends with a failure condition.

Post Conditions

Successful Completion:

The admin user successfully paired their WebWarden app with the proxy.

Failure Condition:

The error logs are updated accordingly.

Use case: Sign-out

Brief description: The admin user signs out of their WebWarden account.

Actors: Home network administrator

Pre-conditions:

- The admin user has a working internet connection.
- The admin user’s WebWarden app is paired with the home network.
- The admin user is already logged in.

Basic Flow:

1. The use case begins when the admin user is at main menu screen.
2. The user selects the “Sign Out” button.
3. The system displays the message “Are you sure you want to sign-out?”
4. The admin user selects the “Yes” button prompt.
5. The system signs the admin user out of their account.
6. The use case ends successfully.

Alternate Flow:

Exceptional Flow:

E1. *App Crash*

If at any stage during the use the app crashes, then:

1. The use case ends with a failure condition.

E2. *App shutdown*

If at any stage during the use case the admin user shutdowns the app, then:

1. The use case ends unsuccessfully.

Post Conditions

Successful Completion:

The admin user has successfully signed out of their WebWarden account.

Failure Condition:

The error logs are updated accordingly.

Use case: Connect to database

Brief description: The WebWarden app and the proxy. can both create a connection with the database.

Actors: Home network administrator, proxy.

Pre-conditions:

- The admin user has a working internet connection.
- The proxy. is connected to the home router.

Basic Flow:

1. The use case begins when the admin user selects a function of the WebWarden app that requires a connection to the database. [**A1**]
2. The system gets the hostname of the database.
3. The system gets the port number of the database.
4. The system gets its username for the database.
5. The system gets its password for the database.
6. The system starts the connection to the database. [**E1**], [**E4**]
7. The system connects to the database.
8. The use case ends successfully.

Alternate Flow:

A1: *Proxy. connection*

If the device trying to connect to the database is the proxy, then:

1. The use case begins when the proxy is required to connect to the database.
2. The proxy gets the hostname of the database.
3. The proxy gets the port number of the database.
4. The proxy gets its username for the database.
5. The proxy gets its password for the database.
6. The proxy starts the connection to the database. [**E2**], [**E4**]
7. The proxy connects to the database.
8. The use case ends successfully

Exceptional Flow:

E1. *App Crash*

If at any stage during the basic flow the app crashes, then:

1. The use case ends with a failure condition.

E2. *App shutdown*

If at any stage during the basic flow the app is shutdown, then:

1. The use case ends unsuccessfully.

E2. *Incorrect username*

If at basic flow step 6 or A1 step 6, the username is incorrect:

1. The database will not allow the proxy or WebWarden app to connect.
2. The use case ends with a failure condition.

E.3 *Incorrect password*

If at basic flow step 6 or A1 step 6, the password is incorrect:

1. The database will not allow the proxy or WebWarden app to connect.
2. The use case ends with a failure condition.

Post Conditions

Successful Completion:

The WebWarden app or the proxy successfully connected to the database.

Failure Condition:

The error logs are updated accordingly.

The connection to the database was denied.

Use case: Edit profile

Brief description: The admin user using the WebWarden app edits their account.

Actors: Home network administrator

Pre-conditions:

- The admin user has a working internet connection.
- The admin user's WebWarden app is paired with the home network.
- The admin is logged into their WebWarden account.

Basic Flow:

1. The use case begins when the admin user clicks on any profile button.
2. The system navigates to the profile screen.
3. The admin user clicks the edit username button. [**A1**]
4. The system displays an empty text field.
5. The admin user enters in their new username into the text field.
6. The admin user selects the "Edit" button prompt.
7. The system prompts the admin user for their sign in password.
8. The admin user enters in their sign in password.
9. The system validates the password.
10. The system connects to the database.
11. The system validates the new username. [**A2**]
12. The system edits the username on the database.
13. The system logs the admin user out.
14. The use ends successfully.

Alternate Flow:

A1: *Edit password*

If at basic flow step 3, the admin user selects the edit password button then:

1. The system prompts the admin user to enter in their current login password and the new password.

2. The admin user enters in their current sign in password and their new password.
3. The system validates both passwords.
4. The system connects to the database.
5. The system edits the password on the database.
6. The system logs the admin user out.
7. The use case ends successfully.

A2: *Username exists*

If at basic flow step 11, the username already exists then:

1. The system displays message “Username already exists”.
2. The use case continues at basic flow step 5.

Exceptional Flow:

E1. *App Crash*

If at any stage during the use the app crashes, then:

1. The use case ends with a failure condition.

E2. *App shutdown*

If at any stage during the use case the admin user shutdowns the app, then:

1. The use case ends unsuccessfully.

E2. *Database connection lose*

If at any stage during basic flow steps 10-12 or the A1 steps 4-5 the app loses connection to the database, then:

1. The system displays message “Error occurred connecting to database”.
2. The use case ends with a failure condition.

Post Conditions

Successful Completion:

The admin user successfully edits their account credentials.

Failure Condition:

The error logs are updated accordingly.

5 Abuse cases

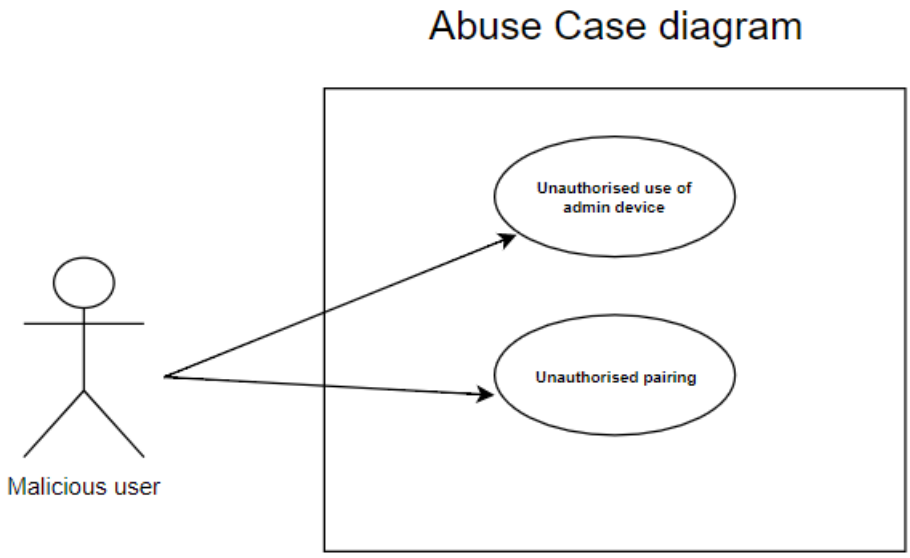

Figure 4

Abuse Case: Unauthorised use of admin device

Brief description: A malicious user takes the home network administrator's personal device and uses the WebWarden app to affect the system in a harmful way.

Actors: Malicious user, proxy.

Pre-conditions:

- Malicious user has obtained an admin device.
- Admin device connected to internet

Basic Flow:

1. This use case begins when the malicious user starts the WebWarden app.
2. The malicious user presses the "Web Control" button. [**A1**]
3. The system navigates to the Web Control screen.
4. The malicious user selects "Remove banned site" button.
5. The system displays the banned site list.
6. The malicious user selects the site to remove.
7. The system connects to database.
8. The system edits database.
9. The abuse case ends successfully.

Alternate Flow:

A1: *Website Access notification*

If at basic flow step 2, the malicious user gets a proxy. notification then:

1. The malicious user clicks the notification.
2. The system displays the site name and content score.
3. The system prompts the malicious user to block or grant access to the site.
4. The malicious user selects the grant button.
5. The system returns the result back to the proxy...
6. The proxy. allows the website to load.

7. The abuse case ends successfully.

Exceptional Flow:

E1. *App Crash*

If at any stage during the use the app crashes, then:

1. The use case ends with a failure condition.

E2. *Database connection lose*

If at any stage during basic flow the app loses connection to the database, then:

1. The system displays message “Error occurred connecting to database”.
2. The abuse case ends with a failure condition.

E3. *App shutdown*

If at any stage during the abuse case the malicious user shutdowns the app then:

1. The abuse case ends unsuccessfully.

Post Conditions:

Successful Completion:

The malicious user gained access to the system using the administrator’s personal device and harmfully misused the system in the following way:

Basic Flow:

- The malicious user successfully removed a banned site from pre-banned list. Home network users could potentially have access to the website.

A1:

- The malicious user successfully whitelisted a device. This device will now not be monitored.

A2:

- The malicious user successfully allowed access to a website using a real-time notification. A home network user has just gained access to a potentially inappropriate site.

Failure Condition:

The error logs are updated accordingly.

6 Non-Functional Requirements

6.1 Performance and Response Time:

A Raspberry Pi 3 B will be configured as web-filtering proxy. for the home network. Although the WIFI adapter on the Raspberry Pi is equipped with a 2.4 GHz WIFI 802.11n wireless adapter which can get download speeds of 150Mbps, the 10/100 ethernet port will be plugged into the home router thus, capping the maximum download speed a user can theoretically get to 100 Mbps. The

maximum download speed is also subject to the user's internet provider. With the addition of the web filtering software and the companion app I am expecting the proxy. to be slower at serving the webpages to the user. The reason being is that my software will be doing work in the background to see if the site needs to be blocked. If the proxy needs to contact the WebWarden app this will add further latency to website loading. The user's download speed will also be affected by the number of devices connected to the proxy at any given time. Although the software will ultimately hinder some areas of performance, this is a quantified payoff for the added security and protection my software will offer. These performance metrics may go up or down as I begin testing my software.

6.2 Availability:

The proxy will be the only gateway onto the home network for non-admin users. This means it must be constantly available even if the administrators of the home network are not present. The web filtering software on the proxy will constantly and automatically run whenever the proxy is powered on or left running. This ensures that non-admin users will always be monitored and protected while web browsing on their devices. The AWS database will be required to run continuously as the WebWarden app and the proxy depend on if for important functionality.

6.3 Operating System Requirements:

The current implementation for the WebWarden app is designed for the Android OS. Unfortunately, any users who do not own an Android device will not be able to avail of the WebWarden software. The Raspberry Pi will be running Raspbian OS which is built on the Linux kernel.

6.4 Security:

Because this project is deeply rooted in security it has become my number one priority. My project intends to transfer sensitive information over local home networks. To ensure this information is kept secure, all sensitive information passed across the local network will be encrypted. The current encryption I am considering is AES. The reason for this is that AES is known to have better performance and can encrypt larger file sizes than asymmetric encryption methods. I plan to implement access control measures for the AWS database using SSH protocol 2 authentication. To ensure that a malicious user won't just download a WebWarden app and change filtering settings for the proxy, the proxy will be configured with a pairing password set up by the home network admins. This password will have to be entered each time a new WebWarden app tries to pair with the proxy application. The admin user will have a unique access control password. It will be used for altering filtering settings of the proxy and allowing unknown sites to be accessed. The access control password is mainly there to protect from a malicious user taking an admin's personal device and changing the proxy filtering settings. The proxy will have WPA2 security with a 25-character long password and an extremely unique network name.

Interoperability:

The interoperability of my software was extremely important. I had to ensure each system component would be able to communicate. The java.sql interface can be used by both my Android and java application. It allows both applications to read and alter the database. The communication between both my Android device and the java application present on the proxy is achieved using the java.net interface, specifically the Socket class. The proxy using the java.net interface will be able to send notifications and then receive the response from the WebWarden app.

7 GUI

The design for WebWarden will follow the current design trends for Android apps being used today. I aim to make the user interface extremely seamless and aesthetically pleasing for the user. This will ensure that the app is not awkward or frustrating to navigate through.

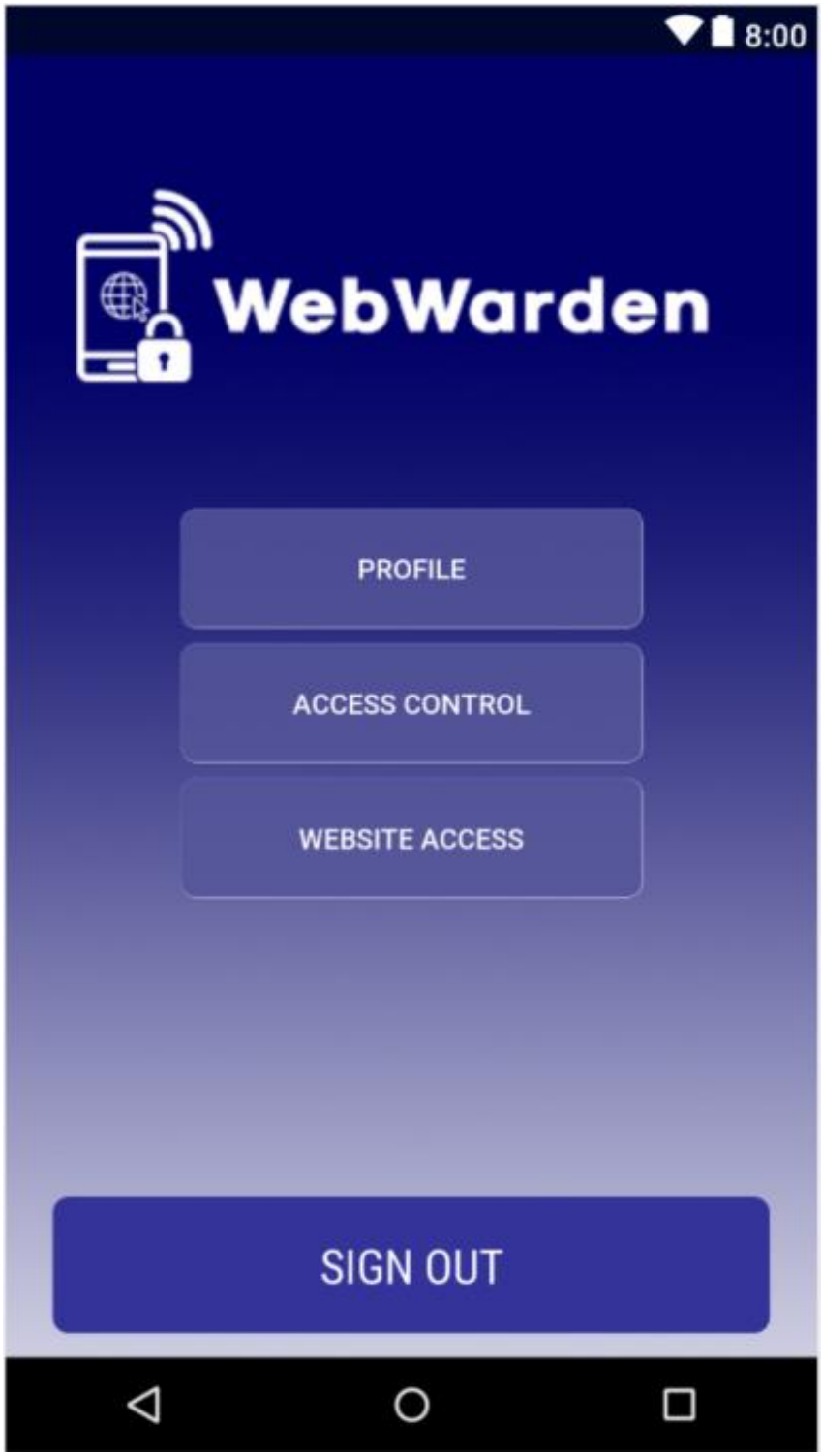

Figure 5. Main Menu

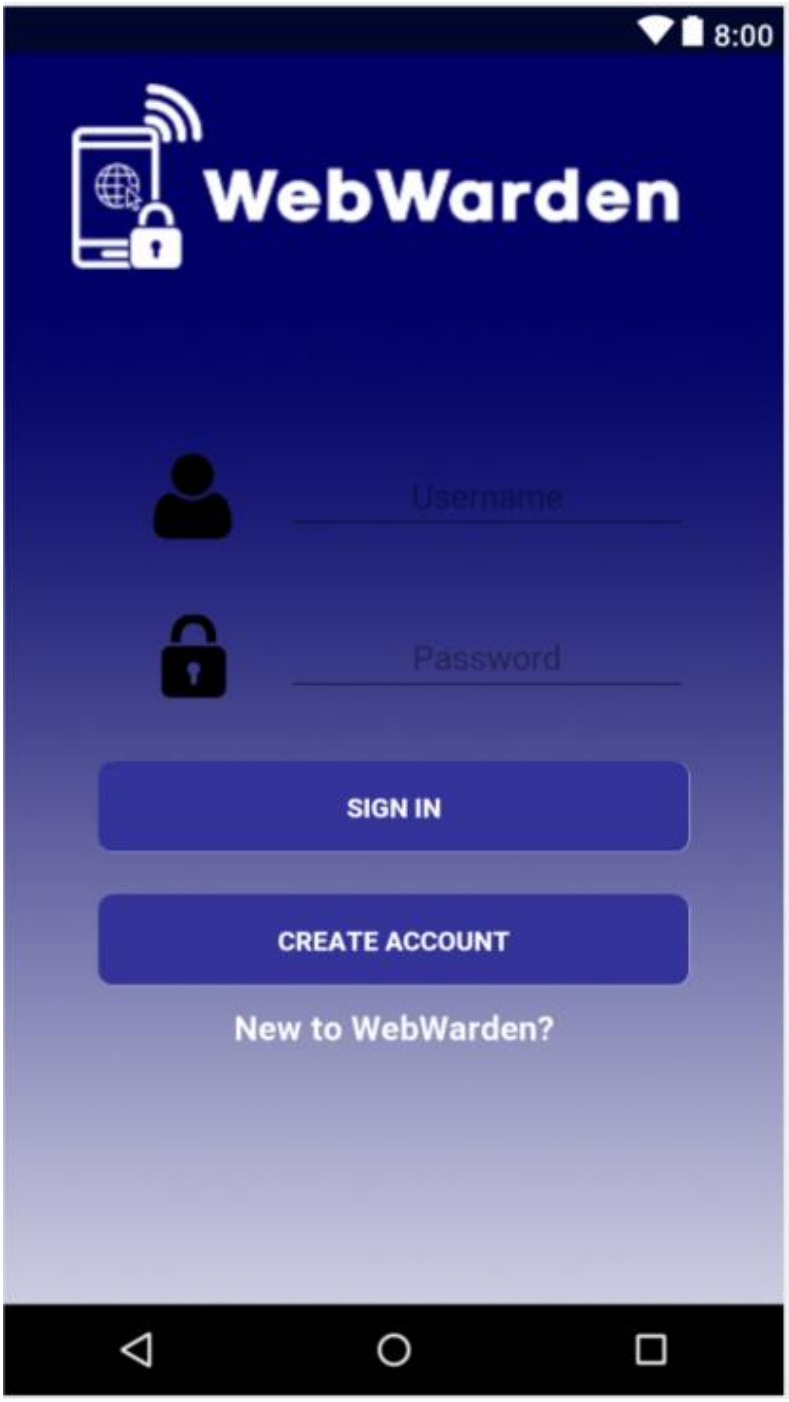

Figure 6.Sign In

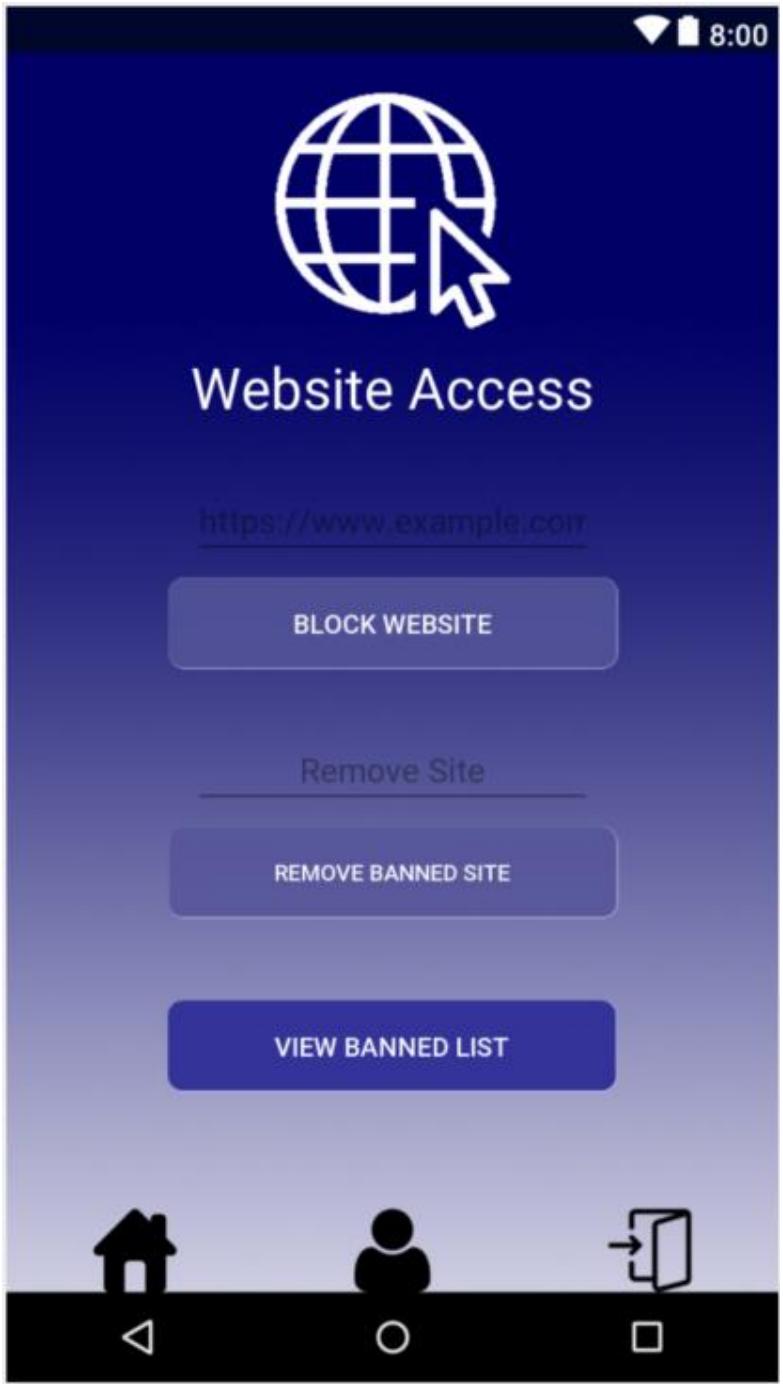

Figure 7.Website Access

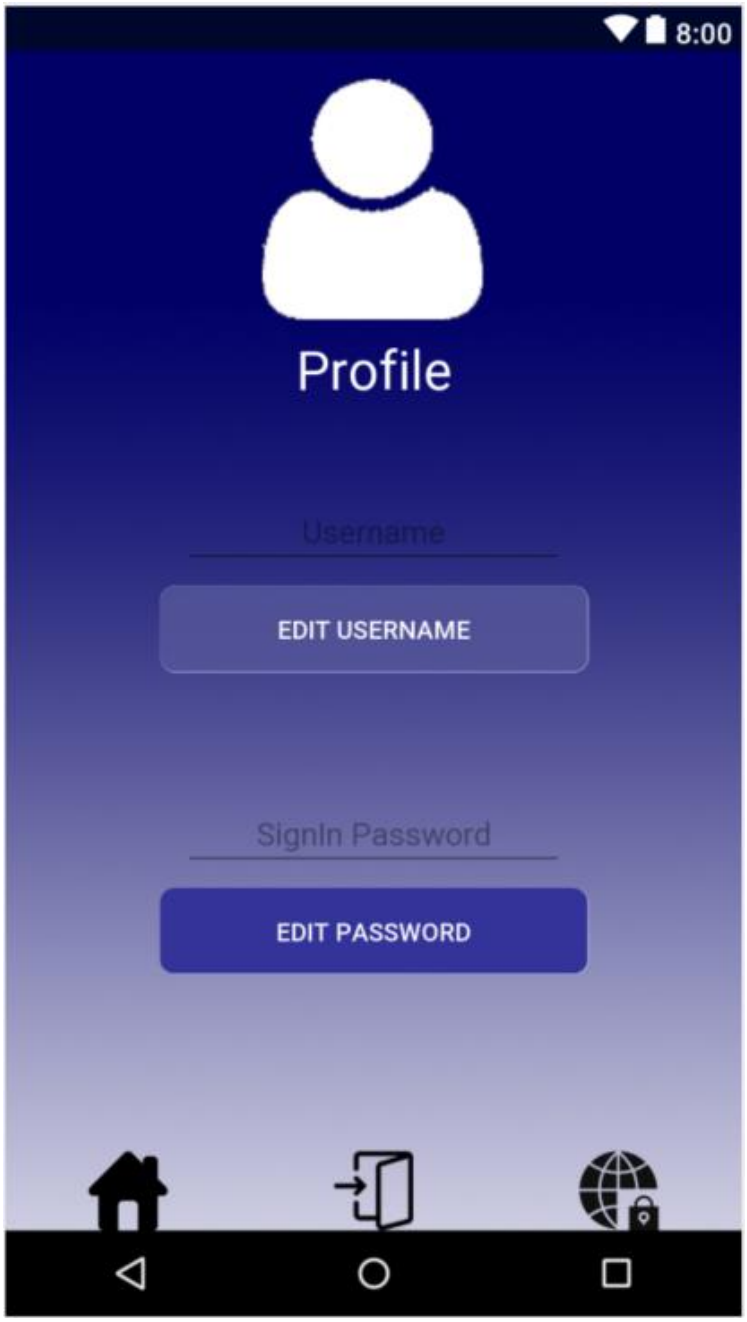

Figure 8.Profile Screen

Figure 9.WebWarden Alert

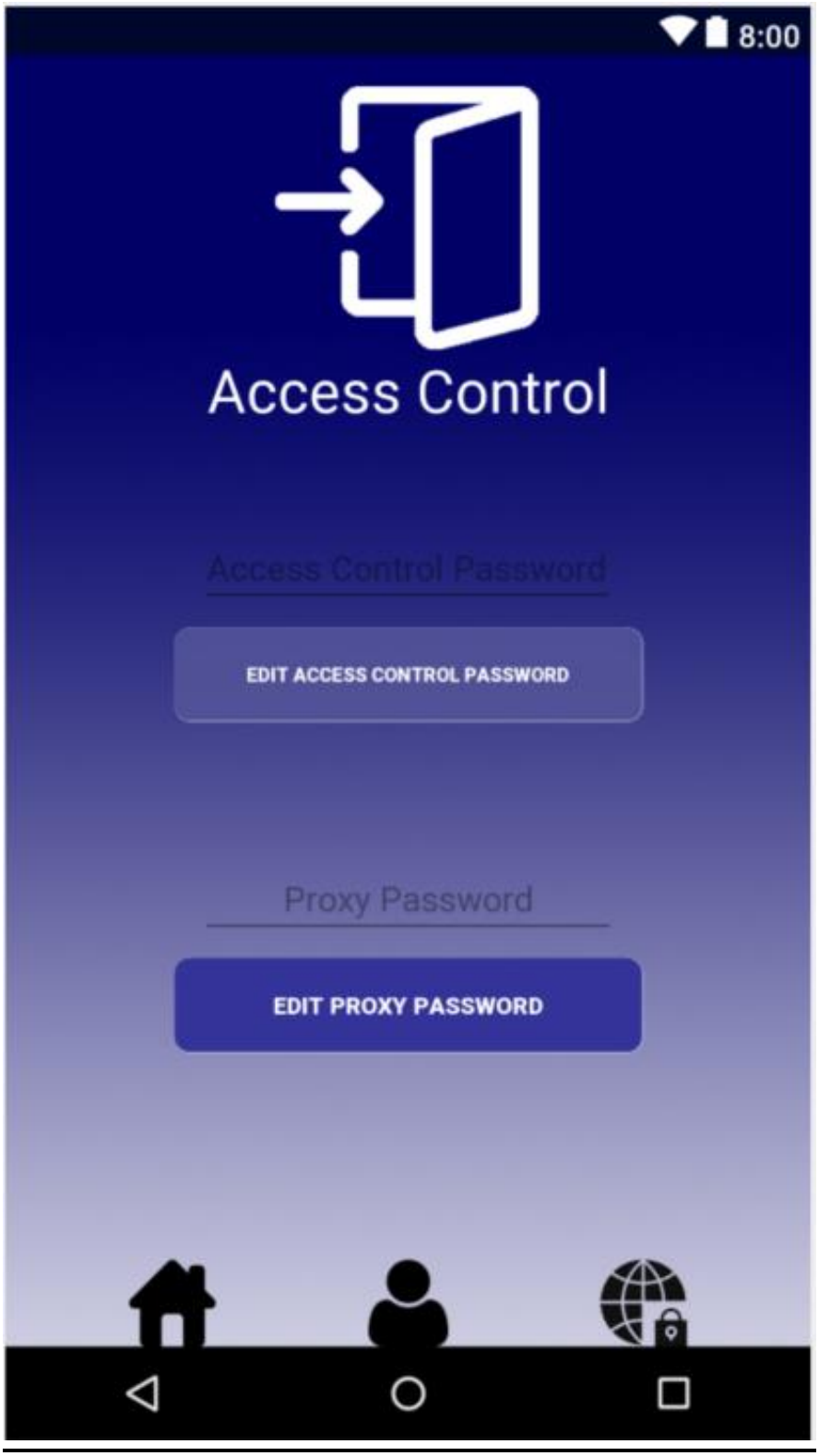

Figure 10. Access Control

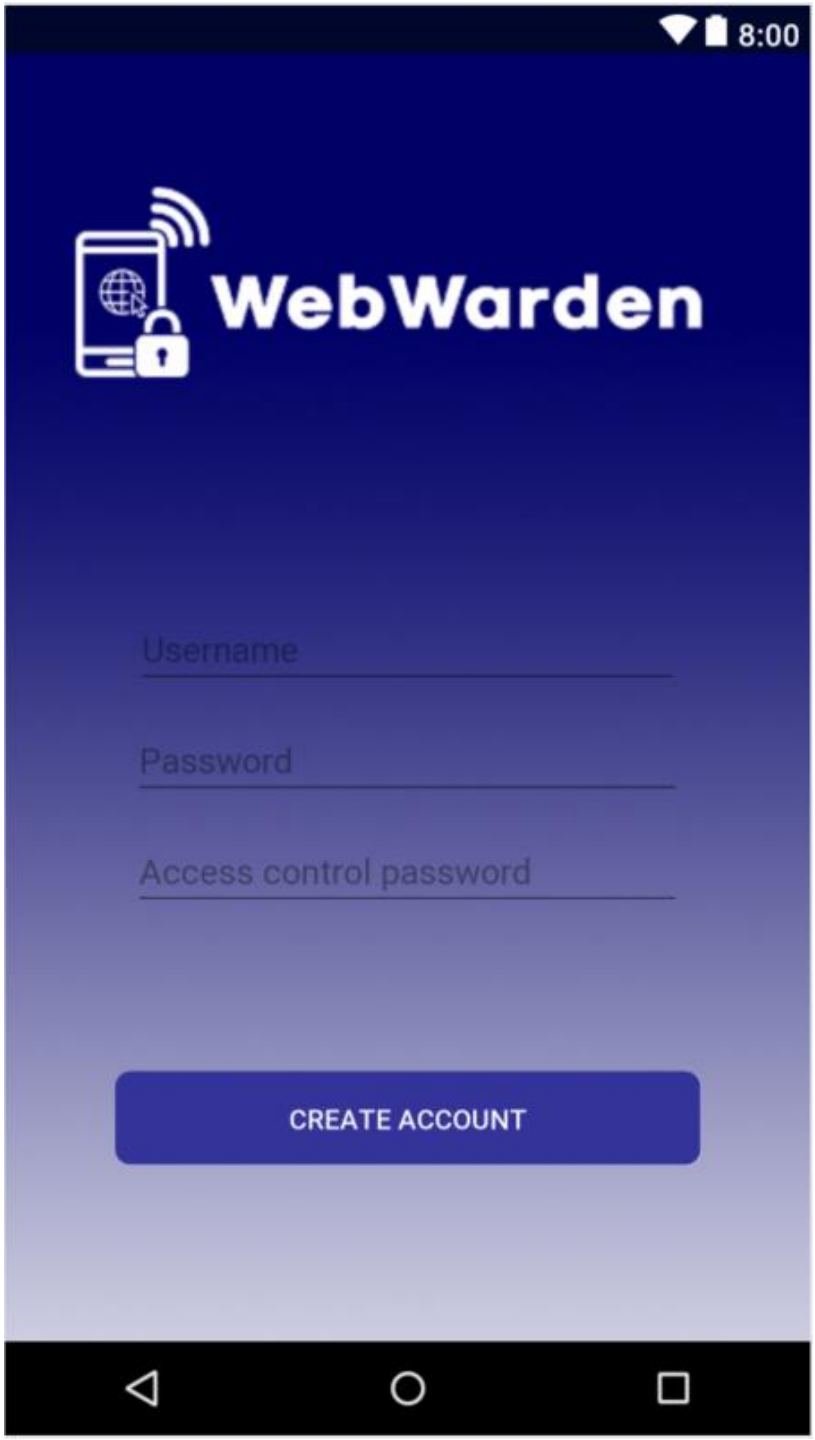

Figure 11.Create Account

8 System Architecture

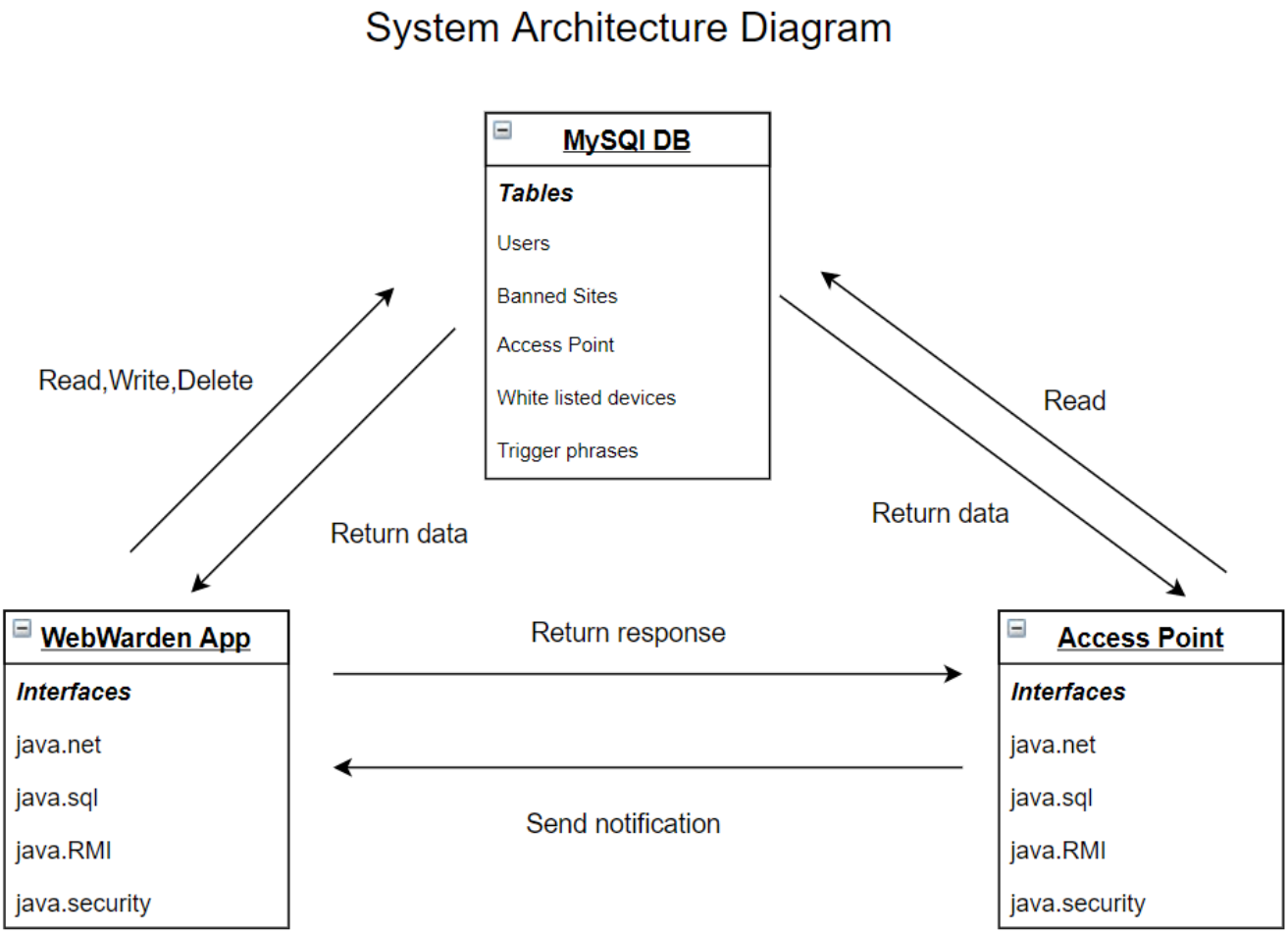

Figure 12

The three main components of the system architecture are the MySQL database, the WebWarden app and the proxy. Both the proxy and WebWarden app can communicate with the database. Only the WebWarden app has the privilege to read, write and delete to the database. The proxy will only have the privilege to read the database. I designed the architecture this way to help enforce the principle of least privilege. The database can send information back to both applications. The proxy java application and the WebWarden app will both employ the java RMI interface to communicate. This will allow the proxy to send the real-time notifications to the WebWarden app and for the WebWarden app to send back its response.

9 Interface Requirements

To allow both the Android app and the Java application on the proxy to communicate with the MySQL database I will be implementing the JDBC API used in Java. This will allow for the reading and writing to the MySQL databases. Both the Android app and the Java application on the proxy will need to communicate. To facilitate this communication, I will be employing the Java RMI. The Java Socket API allows communication between a server and clients.

10 Implementation

10.1 Proxy

To code the proxy in Java I had to make use of the Java Sockets API. The main method of the ProxyMain application sets up the ServerSocket on port 8080. I then created a ProxyInfo class that contained static variables. These variables would hold the values for the word list, blocked sites list and the warden IP address. Before I create the ServerSocket I connect to the database and retrieve the values for the variables in the ProxyInfo class. Details on the code used to connect to the database will be provided later in the technical report. Once the static variables are obtained, the ProxyMain class will kick off the UpdateThread. The UpdateThread is used to ensure that the proxy is getting the most up to date blocked site list. If a WebWarden app user updates the blocked site list, the UpdateThread will be notified and added or removed to the new site to the list. The UpdateThread will then alter the static variable that contains the blocked site. The updated variable will then be passed to any incoming web requests. After the ProxyMain has completed this initial setup it will create a while loop and whenever a client connects it will start a new ProxyThread. I used a “while true” loop to ensure that the proxy would run indefinitely and would stop incoming requests from being blocked.

```
        }

        new Thread(new UpdateThread()).start();
        try {
            final ServerSocket server = new ServerSocket(8080);

            while (true) {
                new ProxyThread(server.accept(),ProxyInfo.wardenIp,ProxyInfo.wordist,ProxyInfo.blockedList).start();
            }

        } catch (IOException e) {
            System.out.println(e);
        }
    }
```

Figure 13. ProxyMain class creating new request thread.

The core web filtering code is stored in the ProxyThread class. The class is implemented to filter both HTTPS and HTTP requests. Each request must be handled differently. For HTTPS requests, my application will forward data between the server and client. To do this I created a pattern variable. This pattern will match a “CONNECT” request. The reason for matching a CONNECT request is because any browser set up with a proxy, when they try access a HTTPS website, will begin the request with a CONNECT. After this I parse out the website URL from the CONNECT request and open a new Socket to that website. I then run two threads which use the forwardData () method. By creating two threads I can have the client passing data to the server and the server passing data back to the client all asynchronously.

If the request is a standard HTTP request I first parse the website out. I then use the Java HttpURLConnection connect to the website and forward all the data back to the original client socket.

```
private static void forwardData(Socket inputSocket, Socket outputSocket) {
    try {
        InputStream inputStream = inputSocket.getInputStream();
        try {
            OutputStream outputStream = outputSocket.getOutputStream();
            try {
                byte[] buffer = new byte[4096];
                int read;
                do {
                    read = inputStream.read(buffer);
                    if (read > 0) {
                        outputStream.write(buffer, 0, read);
                        if (inputStream.available() < 1) {
                            outputStream.flush();
                        }
                    }
                } while (read >= 0);
            } finally {
                if (!outputSocket.isOutputShutdown()) {
                    outputSocket.shutdownOutput();
                }
            }
        } finally {
            if (!inputSocket.isInputShutdown()) {
                inputSocket.shutdownInput();
            }
        }
    } catch (IOException e) {
        e.printStackTrace();  // TODO: implement catch
    }
}
```

Figure 14. ForwardData method used for HTTPS requests

```
conn = (HttpURLConnection) url.openConnection();
conn.setDoInput(true);
conn.setRequestMethod("GET");

// Get server streams.
final InputStream streamFromServer = conn.getInputStream();


// Read the server's responses
// and pass them back to the client.
byte by[] = new byte[BUFFER_SIZE];

int index = streamFromServer.read(by, 0, BUFFER_SIZE);
while (index != -1) {
    output.write(by, 0, index);
    output.flush();
    index = streamFromServer.read(by, 0, BUFFER_SIZE);
}
//output.flush();

// The server closed its connection to us, so we close our
// connection to our client.
output.close();
clientInput.close();
conn.disconnect();
```

Figure 15. Http code for the proxy

Before any sites are loaded to the user I execute my web filtering code. To verify that the site is not on the banned list I iterate over the blocked site arraylist and do a check to see if it is equal to any of the banned sites. If it is, I return a “403 Forbidden” response and close the thread. This will stop the site from loading on the client side. Once the request clears the blocked site verification, I then start my content search. To help with the website scraping I implemented the Jsoup library. By using Jsoup I get all the text from the website declared in the client request. I then parse all this text into a string variable. Once this is completed, I use an advanced FOR loop to iterate over the bad words arraylist. I created a pattern and matcher variable and then tried to find any matches. When a word is found it increments an int value which I created earlier. After the word search is done, I do a check to see if that incremented value is over 20, if it is, the site will be automatically banned. Another alternative that can happen is that if the site contains more than 4 bad words but less than 20, a notification will be sent to the current warden of companion app. The notification is sent using a Socket and specifies the site that its trying to access. The Socket will then wait for a response from the ServerSocket. The Socket will then either block or allow the website to load based on the response it receives.

```
ArrayList<String> badsite = new ArrayList();

for(String site:blockedList) {
    if (request.contains(site)) {
        blocksite = true;
    }
}

if (matcher.matches()) {
    OutputStreamWriter outputStreamWriter = new OutputStreamWriter(clientSocket.getOutputStream(),
            "ISO-8859-1");
    if (blocksite) {

        outputStreamWriter.write("HTTP/" + matcher.group(3) + " 403 Forbidden\r\n");
        outputStreamWriter.write("Proxy-agent: Simple/0.1\r\n");
        outputStreamWriter.write("\r\n");
        outputStreamWriter.flush();
        return;
    }
```

Figure 16. Code used to block sites that match a listed bad site.

```
int i = 0;
final String userAgent = "<Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36>";
System.out.println("Website:" + matcher.group(1));
Document doc = Jsoup.connect("https://" + matcher.group(1)).userAgent(userAgent).get();
String text = doc.body().text();

for (String badword : wordist) {
    int word = 0;
    Pattern p = Pattern.compile(" "+badword+" ");
    Matcher m = p.matcher(text);
    while (m.find()) {
        i++;
        word++;
    }
}

/*
    This code sends the message to the android app
 */
if (i > 19) {
    response = "block";
} else if (i > 4) {
    try {
        String website = "https://" + matcher.group(1);
        Socket socket = new Socket(wardenIP, 5000);
        System.out.println(website);
        DataOutputStream os = new DataOutputStream(socket.getOutputStream());
        BufferedWriter bufferWriter = new BufferedWriter(new OutputStreamWriter(os));
        bufferWriter.write(website + "\n");
        bufferWriter.flush();

        DataInputStream is = new DataInputStream(socket.getInputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(is));

        response = br.readLine();

        os.close();
        is.close();
        socket.close();
        System.out.println("Result:" + response);
    } catch (IOException ex) {
        response = "block";
    }
```

Figure 17. Code used to search a website for bad or offensive words

10.2 Main Menu

When the main menu first starts in the android application it will launch three methods deviceSync, checkWardenStatus and getAccessControl each running a new thread. Each thread must connect to the database and to ensure that only one thread is connecting to the database at one time, I incorporated a synchronised statement. This synchronised statement is locked by a static object called *lock.2.* Once each method has completed, they will call the notify () method to release the lock and allow the other threads to continue. The reason for running these threads is that android does not allow a network operation to run on its main thread (UI thread) so I must run network functions on background threads.

The first method deviceSync () will check if the user is synced with the proxy. The thread will run and connect to the database and get value of the users sync status. If the user is not synced with the proxy, an alert dialog will be displayed. If the user enters in the correct password the thread will update its status in the database and exit. On the other hand, if an incorrect password is entered the main menu will become locked and a re-sync button will appear. All threads will be told not to run and will not execute their code. This stops the service from being started and any user preferences being saved. To unlock the main menu the user will have to click on the re-sync button and enter in the correct password.

```
private void deviceSync(){

    databasePassword();

    (Thread) run() → {
            synchronized (ThreadLock.lock2){
                System.out.println("Thread1 started");
                try {

                    Class.forName("com.mysql.jdbc.Driver");
                    connection = DriverManager.getConnection(DB_URL, databaseUN, new String(PW));
                    PreparedStatement pstmt = connection.prepareStatement(query1);
                    PreparedStatement pstmt2 = connection.prepareStatement(query2);
                    PreparedStatement pstmt3 = connection.prepareStatement(query3);
                    pstmt.setString( i: 1, username);

                    connection.setAutoCommit(false);

                    resultSet1 = pstmt.executeQuery();
                    while (resultSet1.next()) {
                        syncedNameResult = resultSet1.getString( s: "username");
                    }

                    resultSet2 = pstmt2.executeQuery();
                    while(resultSet2.next()){
                        encodedProxyPW = resultSet2.getString( s: "ap_password");
                    }

                    resultSet3 = pstmt3.executeQuery();
                    while(resultSet3.next()){
                        encodedProxySalt = resultSet3.getString( s: "salt");
                    }

                    connection.commit();
                    //connection.close();
                } catch (ClassNotFoundException ex) {
                    System.out.println(ex);
                    //Logger.getLogger(ProxyMain.class.getName()).log(Level.SEVERE, null, ex);
                } catch (SQLException ex) {
                    System.out.println(ex);
                    //Logger.getLogger(ProxyMain.class.getName()).log(Level.SEVERE, null, ex);
                }
```

Figure 18. Start of deviceSync method.

The next method checkWardenStatus will verify if the user is the current WebWarden. The method will start off by querying the database to check the user's current warden status. If the user is not the warden an alert dialog will be displayed asking if they would like to become the WebWarden. Becoming the WebWarden means that all website access control notifications will be delivered to their phone only. If the user accepts the request, the method will update their value in the database.

```
private void checkWardenStatus(){
    if(PW.equals("")){
        databasePassword();
    }
    (Thread) run() → {
            synchronized (ThreadLock.lock2){
                userChoose = null;
                if(runThread){
                    try {

                        Class.forName("com.mysql.jdbc.Driver");
                        connection = DriverManager.getConnection(DB_URL, databaseUN, new String(PW));
                        PreparedStatement pstmt = connection.prepareStatement(query6);

                        pstmt.setString( i: 1, username);

                        connection.setAutoCommit(false);

                        resultSet4 = pstmt.executeQuery();
                        while (resultSet4.next()) {
                            wardenValue = resultSet4.getString( s: "warden");
                        }


                        connection.commit();
                    } catch (ClassNotFoundException ex) {
                        System.out.println(ex);
                        //Logger.getLogger(ProxyMain.class.getName()).log(Level.SEVERE, null, ex);
                    } catch (SQLException ex) {
                        System.out.println(ex);
                        //Logger.getLogger(ProxyMain.class.getName()).log(Level.SEVERE, null, ex);
                    }
```

Figure 19.Start of checkWardenStatus method

The final method getAccessControl will retrieve the user's encoded access control password and the encoded salt used for the password. The reason I store these values in the user preference is to help with optimization. The access control password is used a lot throughout this application and by storing the hashed password and its salt decreases the number of database connection needed. The password is never stored as plain text. Like the other methods, this one connects to the database and retrieves the access control password and salt for the current user. At the end of the method there is a check to see if the notification service is running. If the service is on it will be stopped and restarted. Once this method has finished that's the complete execution of the main menu.

```
private void getAccessControl(){
    if(PW.equals("")){
        databasePassword();
    }
    (Thread) run() → {
            synchronized (ThreadLock.lock2){
                System.out.println("Thread3 started");
                if(runThread){
                    try {

                        Class.forName("com.mysql.jdbc.Driver");
                        connection = DriverManager.getConnection(DB_URL, databaseUN, new String(PW));
                        PreparedStatement pstmt = connection.prepareStatement(query8);

                        pstmt.setString( i: 1, username);

                        connection.setAutoCommit(false);

                        resultSet5 = pstmt.executeQuery();
                        while (resultSet5.next()) {
                            encodedAcPW = resultSet5.getString( s: "ac_password");
                            encodedSalt = resultSet5.getString( s: "ac_salt");
                        }


                        connection.commit();
                    } catch (ClassNotFoundException ex) {
                        System.out.println(ex);
                        //Logger.getLogger(ProxyMain.class.getName()).log(Level.SEVERE, null, ex);
                    } catch (SQLException ex) {
                        System.out.println(ex);
                        //Logger.getLogger(ProxyMain.class.getName()).log(Level.SEVERE, null, ex);
                    }
                    myEditor.putString( s: "acPW",encodedAcPW);
                    myEditor.putString( s: "acSalt",encodedSalt);
                    myEditor.apply();
                    if(isMyServiceRunning(MyService.class)){
                        stopService(new Intent( packageContext: MainMenu.this, MyService.class));
                    }
                    startService(new Intent( packageContext: MainMenu.this, MyService.class));
```

Figure 20. getAccessControl method

10.3 MyService & SiteControl

The MyService class provides the functionality for the warden user to receive real time notifications. The service contains a ServerSocket listening on port 5000. When it receives a connection from the proxy server it reads the input stream to obtain the name of the website trying to be accessed. After getting the name of the website it starts a new SiteControl activity. This activity is the page that will appear to the application user when an unknown website is trying to be accessed. Once the SiteControl activity has started the service will only wait 25 seconds before replying to the proxy socket with automatic block response. This feature is built in to ensure that if a warden misses the notification, a deadlock won't occur and there will be no authorised access of potentially harmful websites. To receive the warden's choice from the SiteControl activity, I created broadcast receiver on the SiteControl buttons. When one of these buttons is selected in the timeframe the service will be able to get this selection. This selection will be written to the output stream and sent to the proxy. When a user terminates the app, the service will get restarted, but with a different service ID. When

the service ID is equal to one this means that the user is still in the app and the SiteControl activity can be displayed directly to the user. If the service ID is not equal to one, there is a check that will instead phone notifications to the user which will appear in their toolbar. The time that the notification is sent is passed to the SiteControl and every time the application sends one, the notification ID is incremented by one. By incrementing the ID, it means that every new notification sent will get its own alert space in the user's toolbar.

```
if (id < 3) {
    userChoose = "";
    System.out.println("App Running!!");
    String alert;
    server = new ServerSocket( port: 5000);
    server.setSoTimeout(10000);
    try {
        socket = server.accept();
        os = new DataOutputStream(socket.getOutputStream());
        bw = new BufferedWriter(new OutputStreamWriter(os));

        is = new DataInputStream(socket.getInputStream());
        br = new BufferedReader(new InputStreamReader(is));
        website = br.readLine();

        Intent intent = new Intent( packageContext: MyService.this, SiteControl.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        intent.putExtra( name: "website", website);
        intent.putExtra( name: "startMainMenu", value: false);
        long def = 0;
        intent.putExtra( name: "timeStart",def);
        startActivity(intent);

        try {
            sleep( millis: 20000);
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }

        System.out.println("User chose:"+userChoose);
        if(userChoose.equals("allow")){
            result = "allow";
        }else{
            result = "block";
        }

        bw.write(result);
        bw.flush();

        is.close();
        os.close();
        socket.close();
        server.close();
```

Figure 21 MyService code to display notification

The SiteControl activity will only be triggered by the service class. When the activity is opened, it checks its current time versus the time that the notification was sent to the user. If more than thirty seconds has elapsed between the two, then the activity will close and a "Notification Expired" message will be displayed to the user. After this check the SiteControl will get the website that the service sent to it. The website will then be displayed in the WebView that is located on the activity. To display any website in a WebView you must create a new WebView Client and override the URL loading. There is an allow and block button on the activity. When a user selects the allow button they will be prompted to enter in their access control password. The user has one attempt to get the password right. If its incorrect, a blocked response will be sent back to the service automatically.

```
    if(startTime != 0){
        start = new Date(startTime);
        open = new Date(openTime);

        timePassed = open.getTime() - start.getTime();

        if(timePassed > 30000){
            Toast.makeText( context: this,  text: "Notification Expired", Toast.LENGTH_LONG).show();
            finish();
        }
    }

    TextView websitetv = (TextView) findViewById(R.id.websiteTV);
```

Figure 22 Code used to verify if the notification is still valid

```
private void getAccessPassword(){

            AlertDialog alertDialog = new AlertDialog.Builder( context: SiteControl.this).create();
            alertDialog.setCanceledOnTouchOutside(false);
            alertDialog.setTitle("SignIn Password");
            final EditText input = new EditText( context: SiteControl.this);
            input.setInputType(InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD);
            alertDialog.setView(input);
            alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL,  text: "Enter",
                    (OnClickListener) (dialog, which) → {
                            editable = input.getText();
                            pwLength = editable.length();
                            editable.getChars( i: 0, i1: pwLength-1,userInputPW, i2: 0);
                            salt = decodeSalt(acSalt);

                            userEncodedPw = hashPassword(userInputPW,salt);

                            if(!userEncodedPw.equals(acPW)){
                                correctPassword = false;
                            }
                            if(correctPassword){
                                Intent broadcast1 = new Intent( action: "getting_data");
                                broadcast1.putExtra( name: "value",  value: "allow");
                                sendBroadcast(broadcast1);
                                finish();
                            }else{
                                Intent broadcast1 = new Intent( action: "getting_data");
                                broadcast1.putExtra( name: "value",  value: "block");
                                sendBroadcast(broadcast1);
                                finish();
                            }
                            dialog.dismiss();
                    });
            alertDialog.show();
}
```

Figure 23. This code launches an alert dialog asking for the user's access control password.

10.4 WebsiteAccess

On the WebsiteAccess activity the user can add a website to the blocked list. To ensure that a user adds a valid website, I see if the application can make a connection with that address. If the connection is unsuccessful a message will be displayed to the user. When a valid address is entered the application will take the address and remove either the https:// or http:// heading and then connect to the database and add the website. After this a new Socket will be created and a message will be sent to the proxy with the website value.

```
private void addSite(){

    (Thread) run() → {
            website = add.getText().toString();
            if(website.contains("https://")){
                StringBuffer change = new StringBuffer(website);
                newAddress = change.substring(8,website.length());
                try {
                    url = new URL(website);
                    urlConnection = (HttpURLConnection) url.openConnection();
                    urlConnection.disconnect();
                } catch (MalformedURLException e) {
                    incorrectAddress = true;
                    e.printStackTrace();
                } catch (IOException e) {
                    incorrectAddress = true;
                    e.printStackTrace();
                }

            }else if(website.contains("http://")){
                StringBuffer change = new StringBuffer(website);
                newAddress = change.substring(7,website.length());
                try {
                    url = new URL(website);
                    urlConnection = (HttpURLConnection) url.openConnection();
                    urlConnection.disconnect();
                } catch (MalformedURLException e) {
                    incorrectAddress = true;
                    e.printStackTrace();
                } catch (IOException e) {
                    incorrectAddress = true;
                    e.printStackTrace();
                }
            }else{
                runOnUiThread(() → { showMessage1(); });
                return;
            }
```

Figure 24. Code to verify valid website.

```
...
try {
    host = new Socket( host: "192.168.0.39", port: 3000);
    os = new DataOutputStream(host.getOutputStream());
    bw = new BufferedWriter(new OutputStreamWriter(os));

    bw.write( str: "add site\n");
    bw.write( str: newAddress+"\n");
    bw.flush();

    os.close();
    host.close();

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

Figure 25. Socket sending new website.

10.5 Security

10.5.1 AES encryption and decryption

Both the proxy server and the android application must connect to a database at some point during their execution. To protect the data base credentials, I stored them in a config.properties file. Prior to saving the database password in the config file, I created an AES encryption program and passed the plain text password through this application. The program implemented a 128-bit AES algorithm. After the password was encrypted I placed it into the config file. To connect to the database both applications would get the values from the config file and a decrypt method I created would be used to decipher the encrypted database password. The code for this is depicted in **Figure 23.**

```
private void databasePassword(){
    try {
        try {
            databaseUN = Util.getProperty( key: "username",getApplicationContext());
            databasePW = Util.getProperty( key: "password",getApplicationContext());
        } catch (IOException e) {
            e.printStackTrace();
        }

        MessageDigest sha = MessageDigest.getInstance("SHA-1");
        byte[] key = {'#', '9', 'F', '2', 'd', 'Y', 'V', 'H', '5', 'e', ']', '=', 'x', 't',
        key = sha.digest(key);
        key = Arrays.copyOf(key, newLength: 16);
        SecretKeySpec sKey = new SecretKeySpec(key, algorithm: "AES");
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, sKey);
        PW = new String(cipher.doFinal(Base64.decodeBase64(databasePW)));
        key = null;

    } catch (NoSuchAlgorithmException ex) {
        System.out.println(ex);
        //LoginLogger.error("NoSuchAlgorithmException: "+ex);
    } catch (NoSuchPaddingException ex) {
        System.out.println(ex);
        //LoginLogger.error("NoSuchPaddingException: "+ex);
    } catch (InvalidKeyException ex) {
        System.out.println(ex);
        //LoginLogger.error("InvalidKeyException: "+ex);
    } catch (IllegalBlockSizeException ex) {
        System.out.println(ex);
        //LoginLogger.error("IllegalBlockSizeException: "+ex);
    } catch (BadPaddingException ex) {
        System.out.println(ex);
        //LoginLogger.error("BadPaddingException: "+ex);
    }

}
```

Figure 26. This method is used to obtain the encrypted database password

10.5.2 Hashes and Salts

After a user has created an account or altered a password within the application these passwords must be stored in the database. All passwords stored in the database are hashed beforehand and then both their encoded hash and encoded salt will be stored in the database. I used the Java PBKDF2WithHmacSHA1 hash along with a key length of 256 and a 32-byte salt. All these features are outlined and recommended by OWASP in their tutorial on how to hash in Java (Owasp.org, nd). I could not use the Java PBKDF2WithHmacSHA512 hash as android does not have access to it. The password salts were also used with the SecureRandom generator to make the salt much harder to predict and crack.

When a user enters a password into any field, I would always ensure to keep that password stored in a char array. When a password is stored as a String in java it is extremely insecure. Strings are

stored as plaintext and are also stored in the JVM string pool for easy reuse. Until the garage collector comes along and destroys the String it will be visible and if a malicious user attacked the system and got access to random parts of the applications memory all the passwords stored in Strings could become compromised. After I am finished with the password I can reset the char array to null and the password will be wiped from the memory. You cannot edit a String because they are immutable variables. **Figure 24** shows the code I used to create the salt and hash user inputted data.

```
private byte[] createSalt(){
    r = new SecureRandom();
    byte[] salty = new byte[32];
    r.nextBytes(salty);
    return salty;
}



private String hashPassword(char[] pw,byte[] salter){
    SecretKeyFactory skf = null;
    String hashString = "";

    try {
        skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        PBEKeySpec spec = new PBEKeySpec(pw, salter, iterationCount: 3, keyLength: 256);
        SecretKey key = skf.generateSecret(spec);
        hashString= Base64.encodeBase64String(key.getEncoded());
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (InvalidKeySpecException e) {
        e.printStackTrace();
    }
    pw = null;
    salter = null;
    return hashString;
}
```

Figure 27 Hash and salt code

10.5.3 Access Control

I created separate database users for both the proxy server and android application. Each user was granted different access rights to enforce the rule of least privilege. The proxy user for the database was granted very limited access to the database and some privileges granted at the field level to ensure maximum access control. The other database user had higher privileges, but were all justified as they were functional for the application. No database user had the ability to drop any tables or delete the database. I also never give out the root level access to the database at point in the either applications. **Figure 25** displays the MySQL queries I used to grant the user their privileges.

```
grant SELECT ON proxy.blocked_sites TO 'proxy'@'%';
grant SELECT ON proxy.wordlist TO 'proxy'@'%';
grant SELECT(warden,user_ip) ON proxy.users TO 'proxy'@'%';
grant INSERT ON proxy.users TO 'user'@'%';
grant Select,Update ON proxy.users TO 'user'@'%';
grant Select,Insert ON proxy.synced_users TO 'user'@'%';
grant Select,Update ON proxy.proxy TO 'user'@'%';
grant Select,Insert,Delete ON proxy.blocked_sites TO 'user'@'%';
```

Figure 28. Database user privileges

10.5.4 Prepared Statements

There are numerous sections throughout this application where the system retrieves user inputted data and then queries the database based on the data. This opens a huge risk for the application as you should never trust user inputted data. To counter the threat of an SQL injection attack, I used java prepared statements. OWASP surmises that prepared statements are one of the best defences against injection attacks (Wichers et al., 2018). **Figure 26** illustrates an example of some of the prepared statements used throughout the app.

```
try {
    Class.forName("com.mysql.jdbc.Driver");
    connection = DriverManager.getConnection(DB_URL, databaseUN, new String(PW));
    PreparedStatement pstmt = connection.prepareStatement(query1);
    pstmt.setString( i: 1, username);
    pstmt.setString( i: 2, hashedUserPass);
    pstmt.setString( i: 3, hashedAcPass);
    pstmt.setString( i: 4, userIP);
    pstmt.setString( i: 5, encodedSalt);
    pstmt.setString( i: 6, encodedAcSalt);
    pstmt.setString( i: 7,  s: "false");

    connection.setAutoCommit(false);
    pstmt.executeUpdate();
    connection.commit();
    connection.close();
} catch (ClassNotFoundException ex) {
    System.out.println(ex);
    //Logger.getLogger(ProxyMain.class.getName()).log(Level.SEVERE, null, ex);
} catch (SQLException ex) {
    System.out.println(ex);
    //Logger.getLogger(ProxyMain.class.getName()).log(Level.SEVERE, null, ex);
}
```

Figure 29.Prepared Statement

10.5.5 Weak Passwords

OWASP states in their Authentication Cheat Sheet that when a user creates a password that certain complexity rules should be enforced. These rules are that there should be at least one digit, one upper case and one lower case letter within the password (Keary et al., 2017). To enforce these rules, I created a method that would take a password created by a user and verify that everyone one of these conditions was met. Eugene describes this as a valid method for making sure weak passwords are not created (H. Spafford, 1991, p. 4). My method will also look at the length of the user sign in password to verify that it is over nine characters long. **Figure 27** shows how I coded the password check method.

```
private Boolean passwordCheck(char[] pw,boolean userPass) {
    Boolean test1 = false;
    Boolean test2 = false;
    Boolean test3 = false;
    Boolean test4 = false;
    Boolean finalResult = false;

    if(userPass){
        if (pw.length <10 ) {
            return finalResult;
        }

        for (char a : pw) {
            if (Character.isDigit(a)) {
                test1 = true;

            }

            if (Character.isLowerCase(a)) {
                test2 = true;

            }

            if (Character.isUpperCase(a)) {
                test3 = true;

            }
        }
        if (test1 && test2 && test3) {
            finalResult = true;
        }
    }else{
```

Figure 30.passwordCheck method

11 Testing

11.1 Stress Test

Using a server tool called Webserver Stress Tool 8, I wanted to get details on how my proxy server would work under high user traffic. My test had the same conditions apart from the number of users I specified. Each test would run for 2 minutes and a user would click on a link every second. The scope of the test was limited to only HTTP websites. By stress testing the proxy, it gives me a rough indication of how many users at a time it can support. I did 5 tests - the number of users would increase for each test. The highest number of users I tested was 100. The proxy did not crash for any of the tests, but CPU usage started to spike for the 50 and 100 user limit tests. While each stress test was running I continued to surf the web while connected to the proxy and didn't notice any delays in page response time. For the 100 users test, the page response time did become slightly slower, but it was almost negligible. The WebWarden companion app continued to receive requests for the 5 and 10 user tests, but for the other higher limit test the WebWarden started missing some notifications. This is since WebWarden only uses one listening thread to accept a notification. The results of the test are displayed in the graph below.

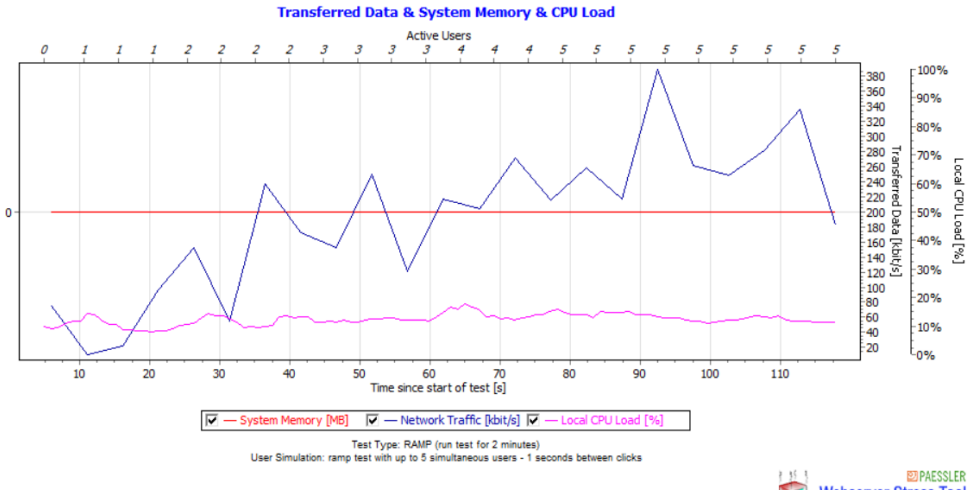

Figure 31.Stress test 5 users

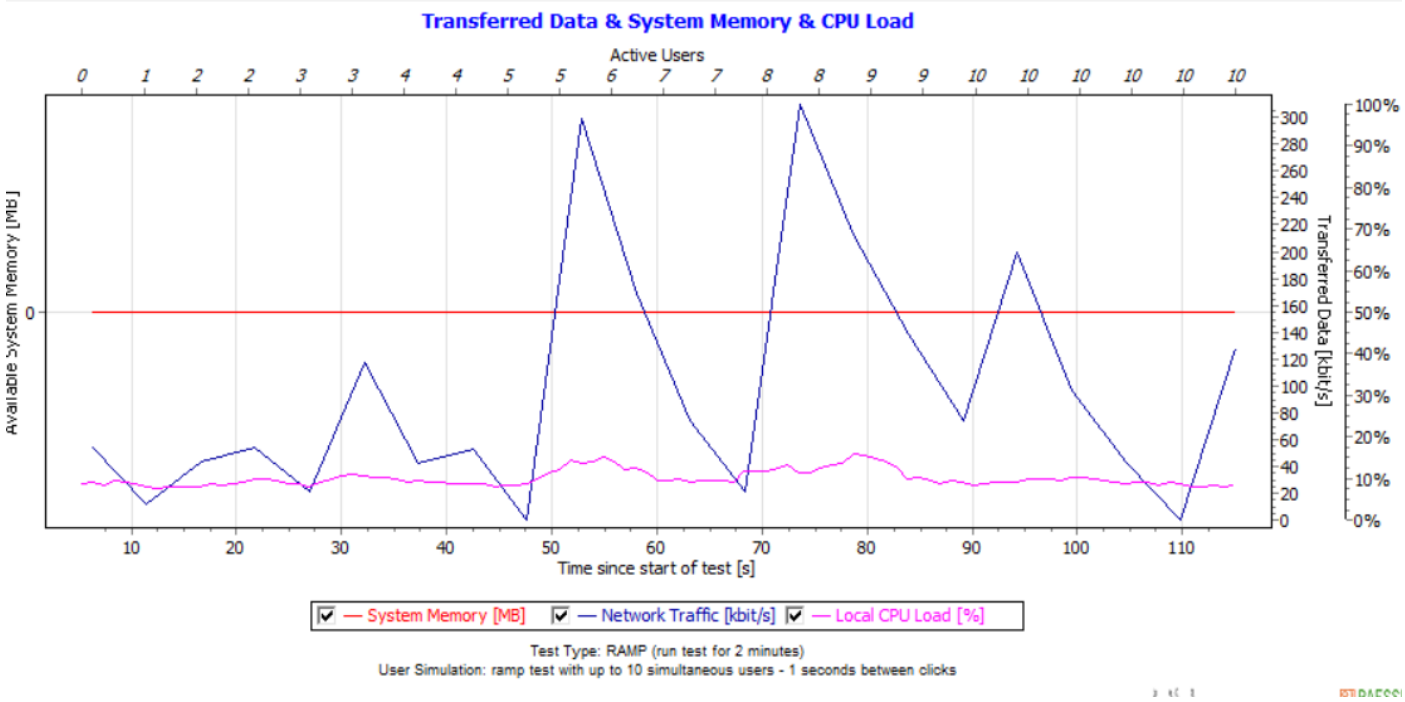

Figure 32.Stress test 10 users.

Figure 33 Stress test 10 users

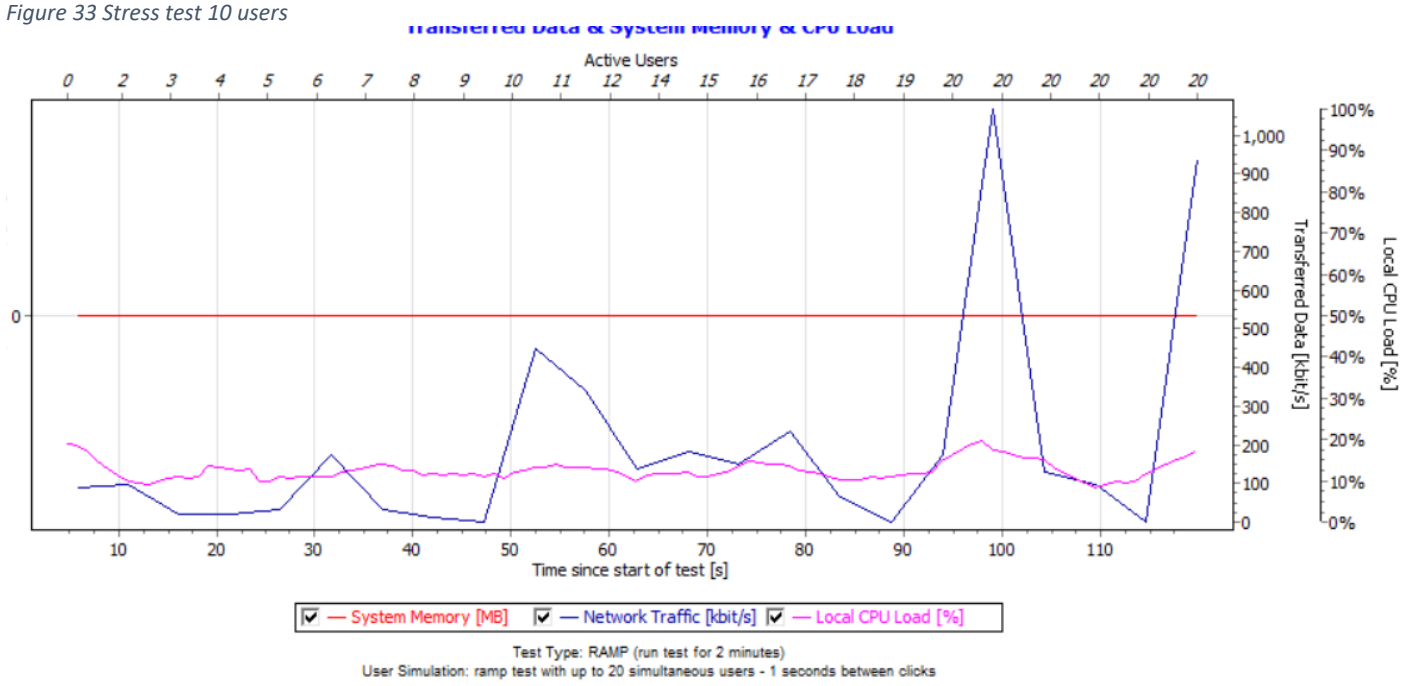

Figure 34.Stress test 20 users.

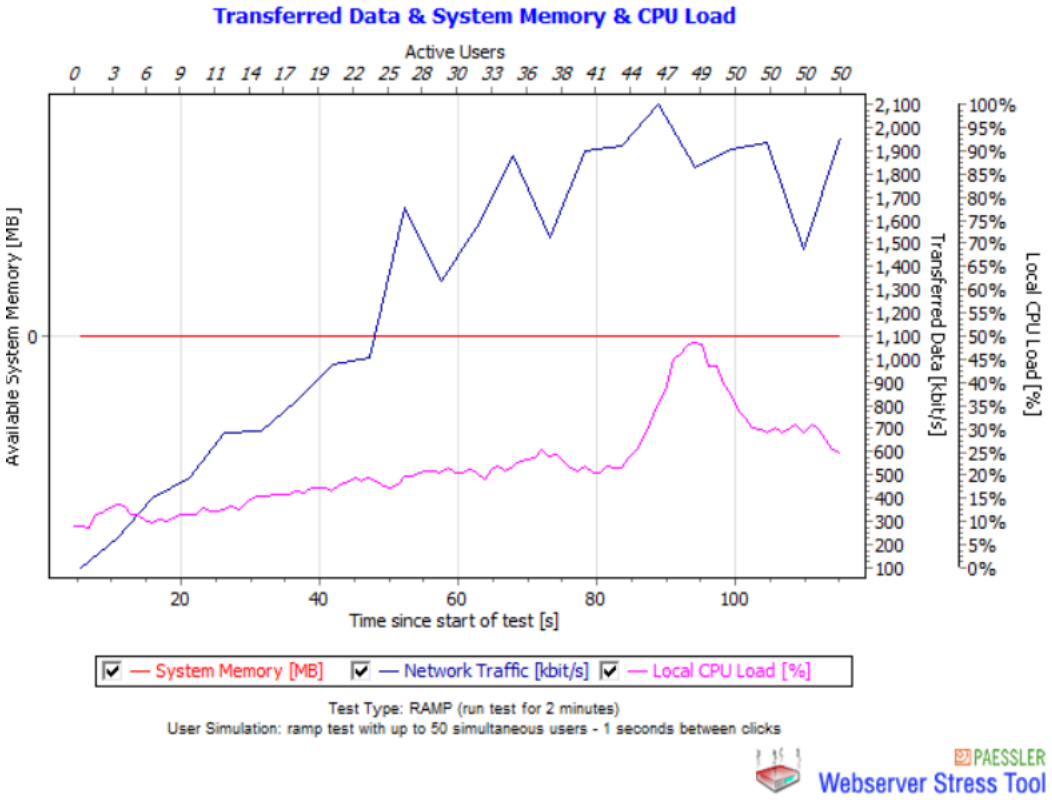

Figure 35. Stress test 50 users.

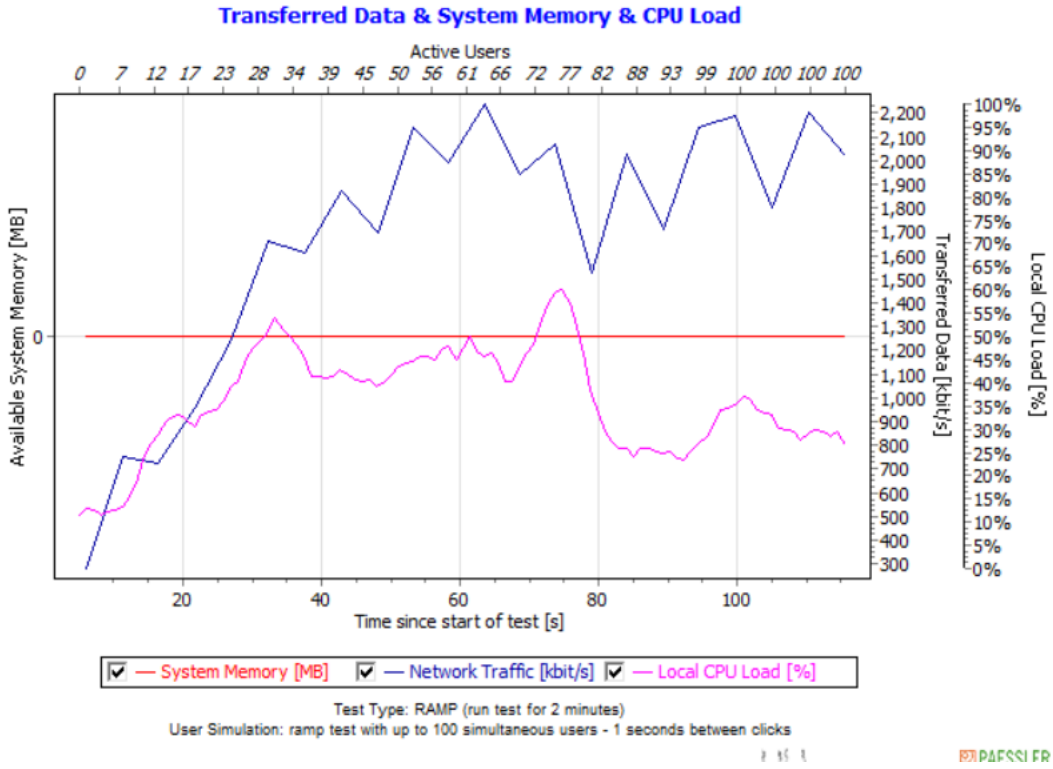

Figure 36. Stress test 100 users.

11.2 Static Analysis

To catch any code vulnerabilities which I might have missed, I used a static analysis tool called VisualCodeGrepper. This tool will scan through both my android and proxy server applications and display any errors. The tool produced a few false positives, but did manage to catch two java classes that I had not set final in my android project. It also notified me that I was implementing the java.Util.Random library, but upon verification I actually was transforming the Random variable into a SecureRandom object later on in the class. The results of the scans are displayed below.

Figure 37. Util class not declared final.

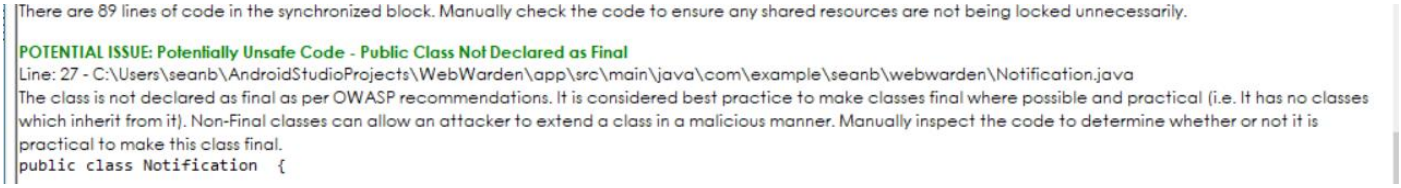

Figure 38. Notification class nor declared final

11. 3 System Testing

Throughout the production lifecycle of the app I would constantly test all the functionality on different android systems. By doing this, it ensured that the app would work on most android APIs with less bugs and errors occurring. This testing helped me solve a very important functionality. Android owns Base64 library and all its methods are only available to android phones that are running on API version 26 or higher. I use the Base64 libraries to decode and encode all user hashes. By testing the hashing functions on a phone that was running API version 23, I noticed that an error occurred. To solve this issue, I imported the Apache commons-codec jar that contains its own Base64 methods that can be implemented. When this jar was added I could then use the hashing function with lower level android APIs.

12 Conclusion

My primary goal for this project was to allow a user to control their home network access in real time. After all my planning and implementation, I can conclude that this goal was achieved. A person using this app in conjunction with my proxy server will have an elevated control over what sites people are browsing. The fluid and easy controls of the android application validate my decision of choosing to make a phone app rather than a computer program for the real time filtering. By also implementing the blocked site check and the word filtering, it ensures that a user is not getting constantly bombarded with website requests. Developing on Android app was an extremely new experience. It took me a while to grasp the fundamental way threads are handled in android compared to a normal java development environment. By the end of the development I feel I am thoroughly educated in android multi-threading.

Another reason I undertook this project was to improve my limited network skills. The core of this project was deeply rooted in network programming and it forced me to research and understand the basic concepts behind it. After conducting intensive research I was able to piece together working code and slowly started to get a grasp on how socket programming worked. This new-found confidence in socket programming allowed me to implement features I thought were out of scope or beyond my skill level.

One of the biggest aspects I take away from this project is the secure coding fundamentals and principles I learned. I noticed myself becoming almost obsessed with trying to implement everything securely. These hardened habits will be carried on to future projects. I am extremely happy with the result of the project. For me this project represents an accumulation of everything I have learned throughout my four years of college and I look forward, with much enthusiasm, to creating similar projects soon.

13 References

Owasp.org. (n.d.). Hashing Java - OWASP. [online] Available at: https://www.owasp.org/index.php/Hashing_Java [Accessed 6 Apr. 2018].

H. Spafford, E. (1991). Preventing Weak Password Choices. [online] West Lafayette: Purdue University, pp.1-4. Available at: https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1874&context=cstech [Accessed 25 Mar. 2018].

Keary, E., Timo, J., Goosen, M., Krawczyk, P., Neuhaus, S. and Aude Morales, M. (2017). Authentication Cheat Sheet - OWASP. [online] Owasp.org. Available at: https://www.owasp.org/index.php/Authentication_Cheat_Sheet [Accessed 3 Apr. 2018].

Wichers, D., Manico, J., Seil, M. and Mishra, D. (2018). SQL Injection Prevention Cheat Sheet - OWASP. [online] Owasp.org. Available at: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet [Accessed 3 Apr. 2018].

Ahmad, T. (2018). *Replacing spinner with drop down alert in android application*. [online] Androidtrainningcenter.blogspot.ie. Available at: http://androidtrainningcenter.blogspot.ie/2012/07/how-to-create-custom-and-default-alert.html [Accessed 6 May 2018].

Android Developers. (2018). *Communicate with the UI thread | Android Developers*. [online] Available at: https://developer.android.com/training/multiple-threads/communicate-ui [Accessed 11 Apr. 2018].

Android?, H. (2018). *How do we use runOnUiThread in Android?*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/11140285/how-do-we-use-runonuithread-in-android [Accessed 13 May 2018].

Javarevisited.blogspot.ie. (2018). *How to create HTTP Server in Java - ServerSocket Example*. [online] Available at: https://javarevisited.blogspot.ie/2015/06/how-to-create-http-server-in-java-serversocket-example.html [Accessed 8 Feb. 2018].

McMahon, K. (2018). *Android Check if Service is Running*. [online] Gist. Available at: https://gist.github.com/kevinmcmahon/2988931 [Accessed 1 May 2018].

Melton, J. (2018). *A Simple Multi-Threaded Java HTTP Proxy Server – John Melton's Weblog*. [online] Jtmelton.com. Available at: https://www.jtmelton.com/2007/11/27/a-simple-multi-threaded-java-http-proxy-server/ [Accessed 3 Jan. 2018].

Stack Overflow. (2018). *creating a Java Proxy Server that accepts HTTPS*. [online] Available at: https://stackoverflow.com/questions/9357585/creating-a-java-proxy-server-that-accepts-https [Accessed 8 Apr. 2018].

White, J. (2018). *Android Non-UI to UI Thread Communications (Part 1 of 5) - Intertech Blog*. [online] Intertech Blog. Available at: https://www.intertech.com/Blog/android-non-ui-to-ui-thread-communications-part-1-of-5/ [Accessed 13 May 2018].]]

14 Appendix

14.1 Monthly Journals

Reflective Journal

Student name: Sean Brady x14715859

Programme: BSc in Computing

Month: September

My Achievements

This month I barely got an idea ready for the project proposal. I honestly was so stressed! The weekend before I was just sitting in my room trying to come up with an idea that was somewhat decent. The hardest part is coming up with a project that's complex enough for fourth year, but not too complex that you won't get it done in time. I came up with the idea while lying on the floor outside my room. It was my "Eureka!!" moment. My idea was to create a web filtering device that will send notifications to an app when a suspected dodgy website was being viewed. The person in control of the app could then either block or allow the site to be viewed in real-time. I then relayed this idea to a friend in my course to get their opinion on it. I was really relieved when they said they liked it. I now just had to get it past the judges.

The Sunday before the presentation I quickly looked over the technologies I hoped to implement. After much researching, I decided I would go for the raspberry 3 B. While waiting outside the judge's door, it dawned on me that perhaps I hadn't prepared enough and had a little moment of panic. When I began my speech I could see the confusion on the judge's faces. This threw me off a little as I'm generally a confident public speaker. The confidence began to fade and I started to spew word vomit everywhere. Luckily, they could somehow discern my project from the mumble jumble mess that was my proposal. They passed me, but they had concerns. I left feeling pretty deflated and almost afraid because now I had to try and attempt this project.

My Reflection

Looking back, I should have practiced my proposal speech on someone. It doesn't matter if it was my Mom, the dogs or even the kitchen sink I should have practiced. I felt I didn't really get my idea across as well as I should have. This is something that I will change in the future for sure! Having just a little practice for a big speech makes the world of difference and helps you formulate what you're going to say.

Intended Changes

My time management has to improve. This is the first year I have ever had to implement a time management program cause I'm the type of guy that leaves everything for the last minute. You the know the old saying "Diamonds are formed under pressure " I sort of applied that saying to most of my projects and reports throughout my college life. That technique will not serve me well this year. I'm aiming for 1:1 and I need to be finishing projects at least a week before they are due, just so I can keep on top of everything this semester.

Supervisor Meetings

Date of Meeting: N/A

Items discussed: N/A

Action Items: N/A

Reflective Journal

Student name: Sean Brady

Programme: BSc in Computing

Month: October

My Achievements

This month I created my project proposal. This really helped paint a complete picture of how I was going to create my project. I had a few issues with it at the beginning especially with the background part of the project proposal. I was a little too informal, so it didn't read well. With a few tips from my supervisor I completely changed that part and just focused on the research I put in and how this helped me formulate my idea. After two more drafts I was finally ready to upload. Overall, I was quite pleased with my project proposal and how it turned out.

My Reflection

I feel my writing needs to improve. When I would read over my drafts I was quite surprised at how bad some parts read. This is just mainly down to practice, but it is something that I will look to improve upon.

I feel my time management has improved since September. I have delivered all my assignments at least a week early. It is still early days though, so I will see how well it transitions into November because that is the most time-consuming month.

Intended Changes

In November I'm going to try and manage my projects a bit better. At the current moment I just do one project at a time. It is working but I feel in November I'm going need to work on multiple projects simultaneously. This is very important because the requirement spec is due on the 10th week but I have four other projects due. Trying to chip away at each project simultaneously is going to be extremely important.

Supervisor Meetings

Date of Meeting: 20th and 17th of October.

Items discussed: Project Proposal.

Action Items: I was appointed Irina as my supervisor. Over the course of October Irina would help me with my Project Proposal. I sent her two drafts and she would point out parts that needed improving or changing.

Reflective Journal

Student name: Sean Brady

Programme: BSc in Computing

Month: November

My Achievements

I started creating my requirement this month. I manged to get a large portion of the AI down this month as well, so I was quite pleased. I finally understand how API work as well and don't feel like I'm letting my team down anymore.

My Reflection

A lot of anxiety has occurred this month because I feel I might not have the skills to carry out this project. Just researching everything about HTTPS and HTTP I almost regret choosing this project. I wish I had chosen something a bit closer to what I know how to create.

Intended Changes

I need to start studying more when I get home. I also need to stop letting projects block me from working on other assignments. I have an awful habit of just focusing on one project and forgetting the rest.

Supervisor Meetings

Date of Meeting: 24th November.

Items discussed: Requirement Spec.

Action Items: Irina just gave me some helpful tips on how I could improve my requirement spec.

Reflective Journal

Student name: Sean Brady

Programme: BSc in Computing

Month: December

My Achievements

I finally have all my projects done and feel very happy with all the work I uploaded. I managed to achieve quite a high grade in my Midterm presentation which came as a huge surprise!

My Reflection

I was really burned out after this semester. I wish I could have studied a bit more for my exams. I feel I messed up on the AI exam, but I honestly was not motivated to study it. I hope this does not happen in the summer time.

Intended Changes

Get motivated again for the coming semester. Just need the final push for the last few months and then I'm done.

Supervisor Meetings

Date of Meeting: 5th December.

Items discussed: Just discussed the midterm presentation.

Action Items: Irina gave me feedback on all the comments that Paul Stynes mentions during my midterm presentation.

Reflective Journal

Student name: Sean Brady

Programme: BSc in Computing

Month: January

My Achievements

I was very pleased with my module results! I finally managed to get the HTTP proxy working somewhat for my software project. I just need to tackle the HTTPS now.

My Reflection

I was looking forward to my next lectures, but both pen testing and forensic analysis have extremely bad lab rooms that don't allow us to do work so this is extremely frustrating.

Intended Changes

I will need to get on track with my studying.

Supervisor Meetings

Date of Meeting: 20th January.

Items discussed: Just talked about some key parts of my software project.

Action Items: I will now implement a proxy instead of an access point.

Reflective Journal

Student name: Sean Brady

Programme: BSc in Computing

Month: February

My Achievements

Finally got the HTTPS working for my software project.

My Reflection

I need to apply more time to my software project.

Intended Changes

I hope to stay after college and start studying more. Hopefully this will give me the motivation to start some of my assignments.

Supervisor Meetings

Date of Meeting: NA.

Items discussed: NA.

Action Items: NA

Reflective Journal

Student name: Sean Brady

Programme: BSc in Computing

Month: March

My Achievements

Got my first penetration testing report done. I really enjoyed doing it and it was very interesting and I learned loads.

My Reflection

Still not studying when I get home as I'm feel a little demotivated, so I feel I'm getting into bad habits.

Intended Changes

I'm going to try to make it to more forensic classes, but I feel I'm falling a bit behind in that class.

Supervisor Meetings

Date of Meeting: 23rd March

Items discussed: How the software project was going.

Action Items: I must implement proper referencing in my software project.

Reflective Journal

Student name: Sean Brady

Programme: BSc in Computing

Month: April

My Achievements

Didn't achieve much this month!

My Reflection

I completely messed up. I didn't do enough work over the two weeks break and I ended up falling behind with all my assignments. I most likely will end up doing my forensic assignment during the week that I am meant to be studying.

Intended Changes

Try becoming more motivated because the final push is just around the corner.

Supervisor Meetings

Date of Meeting: 2nd April

Items discussed: Discussed how much I have completed of the software project.

Action Items: Irina gave me a little tip about using the Secure Random library in java to implement salts for each user.