**EE6052 Web Application Design**

**Programming Project2**

**Group 14**

Xiaolong Liang, UL student ID 15021793, M.Eng. Computer and Communications Systems

Benjamin Keil, UL student ID 13116371, B.Sc. Computer Systems

Abdul Halim, UL student ID 13029096, B.Sc. Computer Systems

Enda O’shea, UL student ID 13062344, B.Sc. Computer Systems

David Hammill, UL student ID 13069667, B.Sc. Computer Systems

University of Limerick

Limerick, Ireland

**Project Description**

The project is an online shop application using EJB, entity classes and servlets/JSF/HTML. Features implemented using servlets, JSF pages, and EJBs only. Use a stateful session bean for the shopping cart. Use a message-driven bean for the logging facility. This project is developed in NetBeans.

**Project Requirements**

* Access to the shop is limited via an authentication scheme.
  + Access rights are role-based, i.e. customer and administrator.
* Provide at least two accounts
  + Customer joe with password 1D10T?
  + Administrator toor with password 4uldo0!
* Functionality
  + Customer
    - Browse through all items
    - Search products by ID number and browse through the search results
    - Search products by name and browse through the search results
    - Add displayed items to shopping cart
    - Remove items from their shopping cart
    - Edit their profile – must contain at least name, Customer ID and a message to other users. Name and ID are taken from Customer table, message can be any text – allow at least for 500 characters
    - View profiles from other users – provide search by name and search by ID
    - Check out or cancel current order
  + Administrator
    - Add new products to the sale database
    - Remove products from the sale database
    - Increase/decrease available amount of any product
  + When customers check out, the quantity for your items in the database is adjusted correspondingly. Make sure the quantity of any product in the database cannot drop below 0.
  + When customers cancel their order, the database should remain unchanged.
* Logging facility
  + Every time a customer confirms an order or cancels an order, a corresponding entry is added to the log (use either log-file or table in database).
  + Every time an administrator adds or removes a product, a corresponding entry is added to the log.
* Security against specific OWASP Top 10 vulnerabilities
  + A1: Injection
  + A2: Broken Authentication & Session Management
  + A3: Cross-Site Scripting (XSS)
  + A4: Insecure Direct Object References
  + A7: Missing Function Level Access Control
  + A8: Cross-Site Request Forgery (CSRF)
  + A10: Invalidated Redirects and Forwards (your application must contain at least one user input-dependent redirect or forward)
  + Discuss what techniques were used to ensure the application is not vulnerable to the listed OWASP vulnerabilities.
  + Discuss how the application was tested to ensure the chosen defence is working correctly.

**Fulfilment of Project Requirements: Access via Authentication Scheme**

We are using form-based authentication using JDBC realm. Required Users, Groups are created by app on depoyment but GlasshFish server needs to be configured to allow this authentication mechanism properly. Please refer to “README\_GlassFish\_Config.txt” for Server configureation details.

* Access to the shop is limited via an authentication scheme.
  + Access rights are role-based, i.e. customer and administrator.
* Provide at least two accounts
  + Customer joe with password 1D10T?
  + Administrator toor with password 4uldo0!

Customer log in:



Administrator log in:

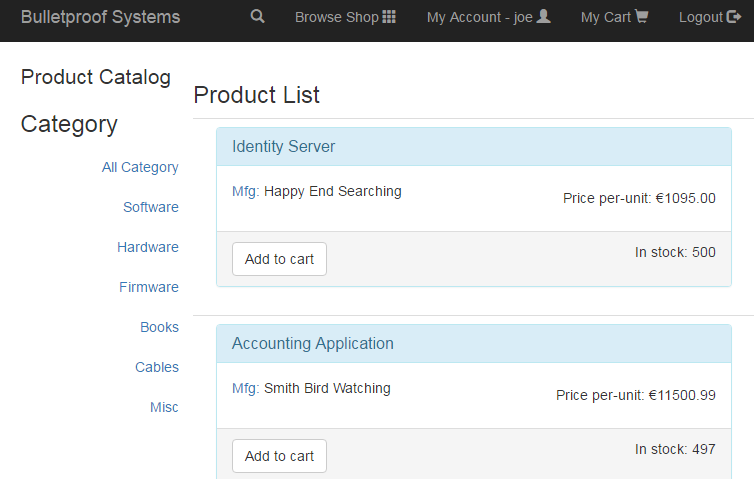


* + - *The customer and administer log in the system using the same login webpage.*
    - *Only valid user name can log in the system.*

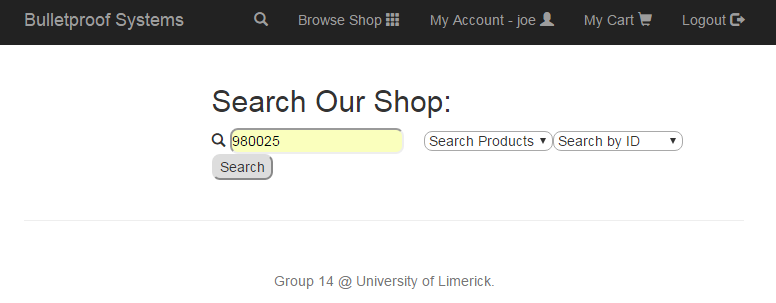


**Fulfilment of Project Requirements: Customer Functions**

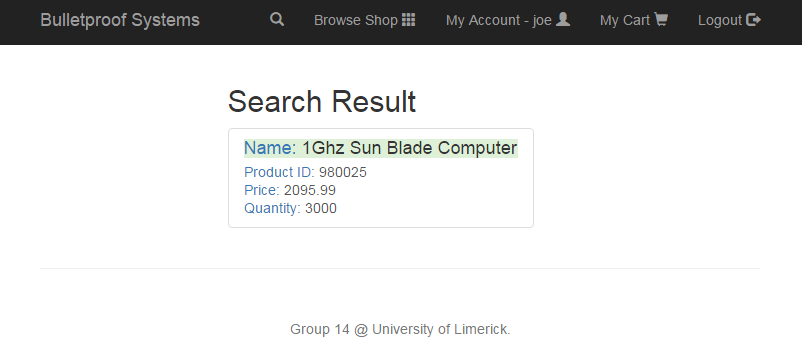
* + - Browse through all items when Logged in as customer:



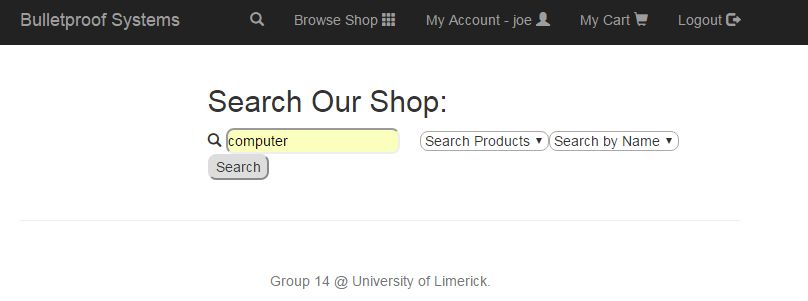
* + - Search products by ID number and browse through the search results



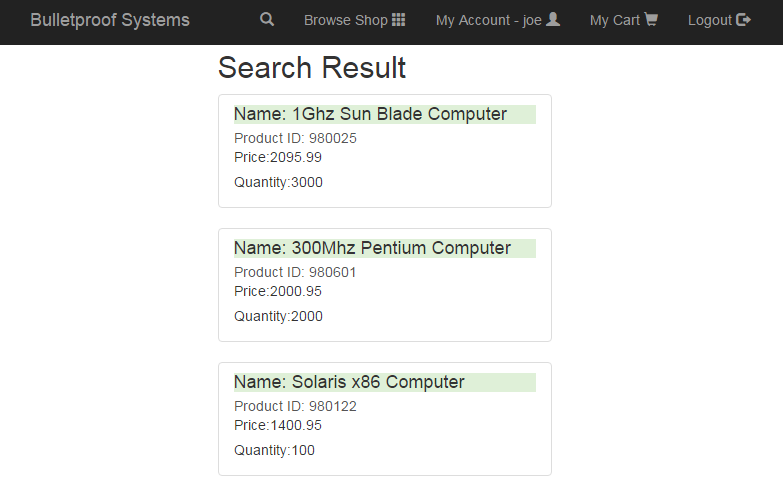
Search result



* + - Search products by name and browse through the search results

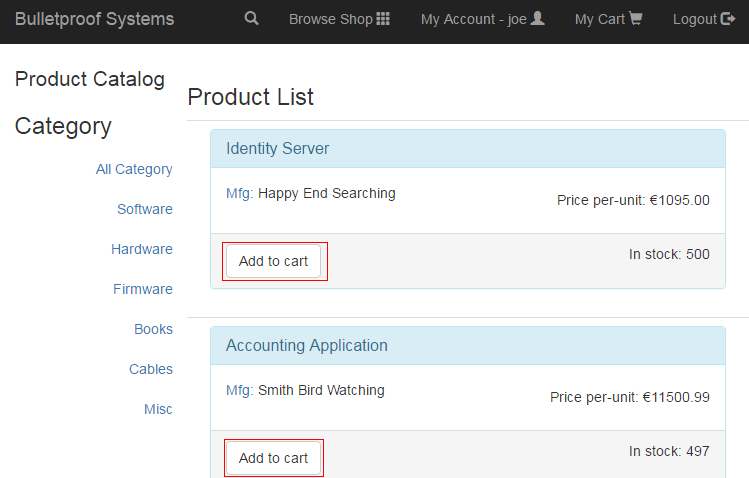


Search result

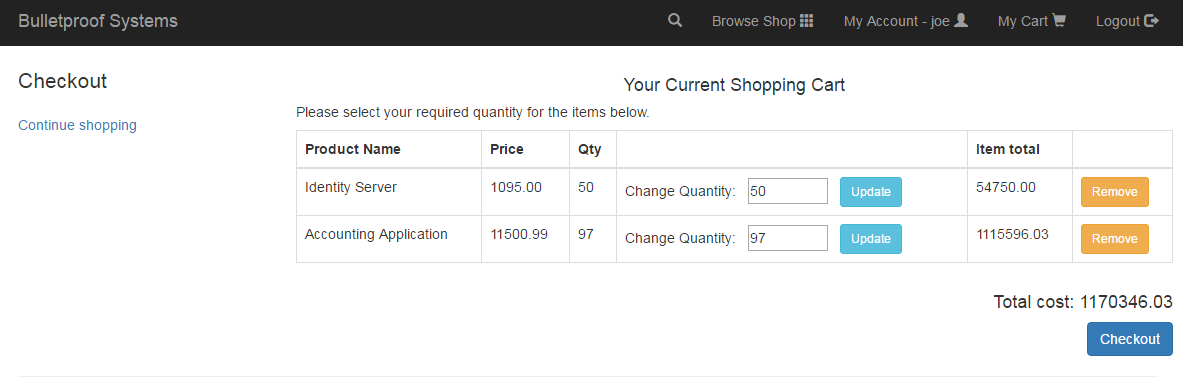


* + - Add displayed items to shopping cart

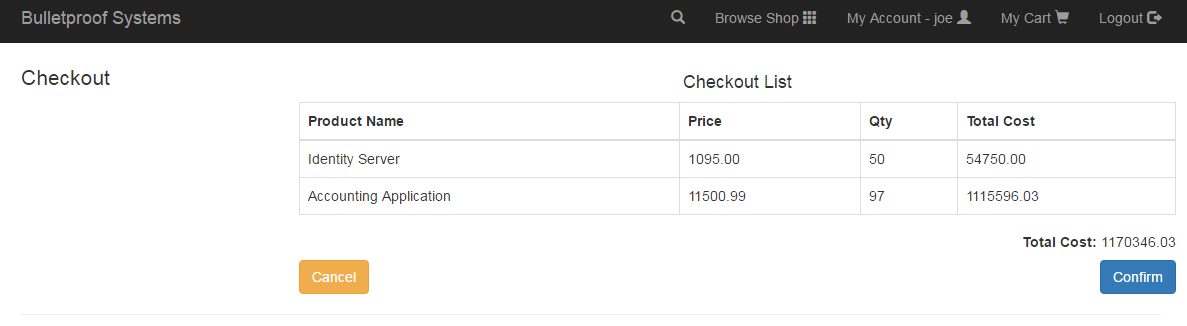
Add two products “Identity Server” and “Accounting Application” to cart:



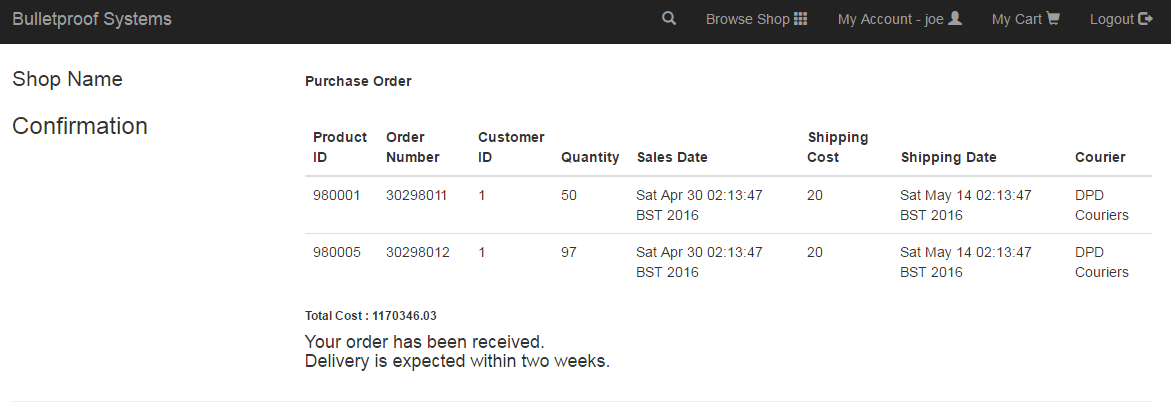
* + - Browse shopping cart (Update the quantity to 50 and 97 respectively)



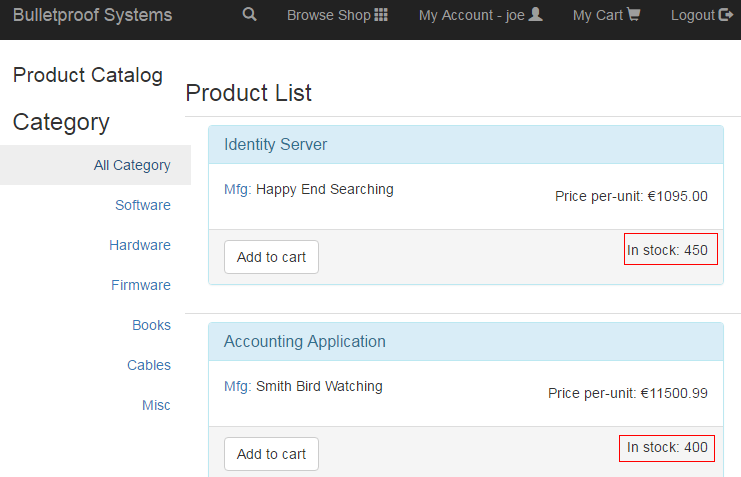
* + - Checkout shopping cart

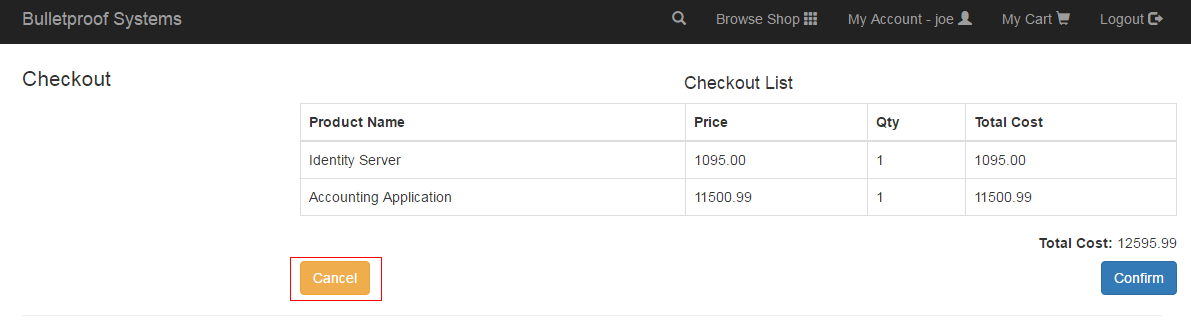


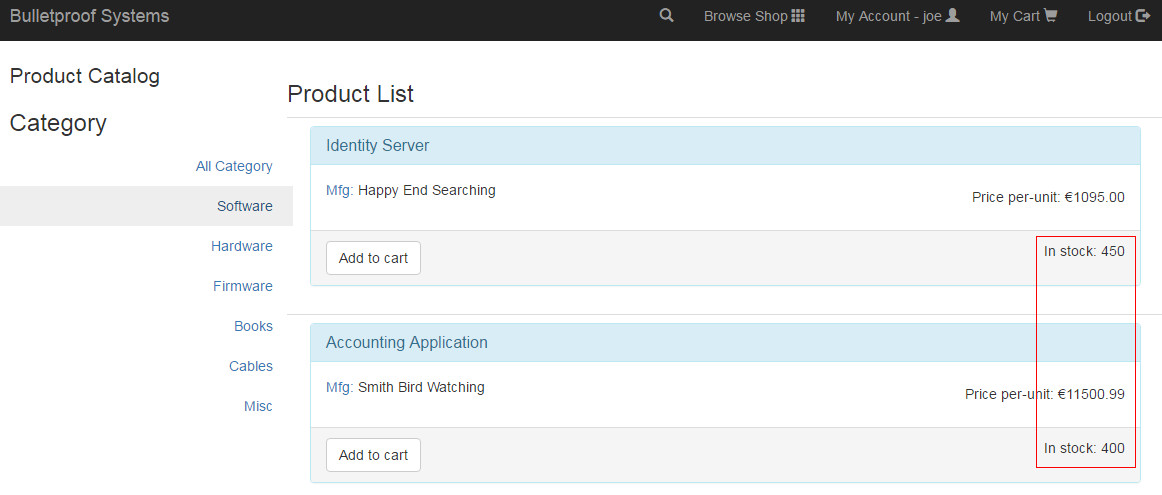
Confirm Checkout List



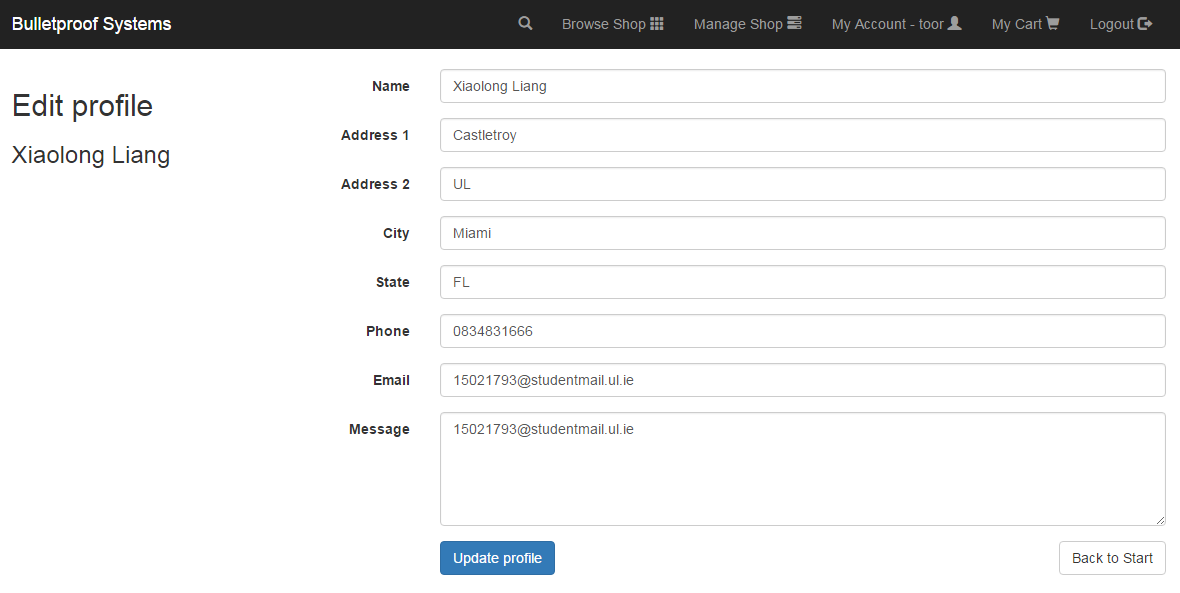
When customers check out, the quantity for your items in the database is adjusted correspondingly.



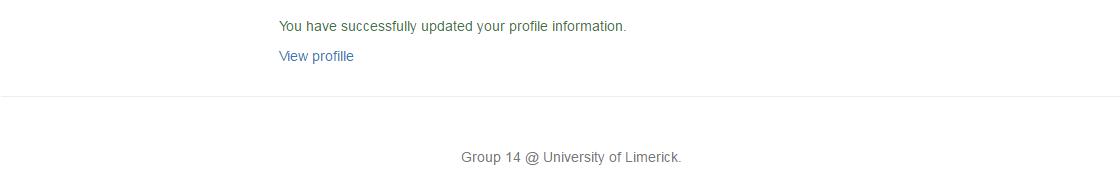
When customers cancel their order, the database should remain unchanged

Browser Shop again and the quantity didn’t change.

* + - Edit user profile

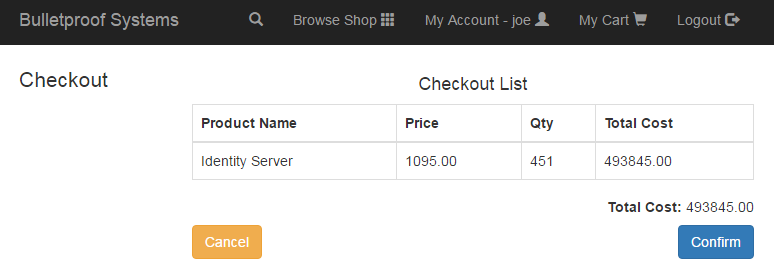


Update profile.

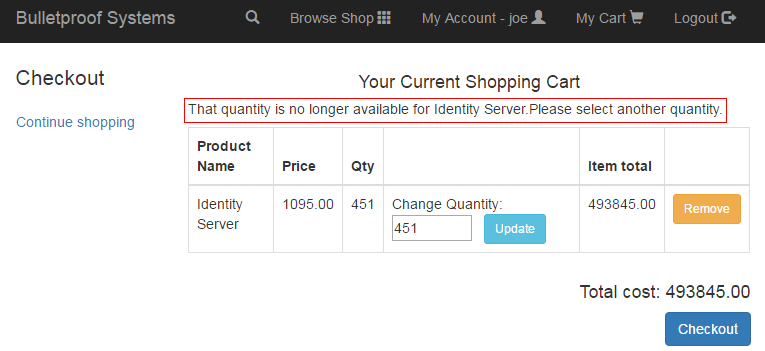


* + - The quantity of a product in the database cannon drop below 0.

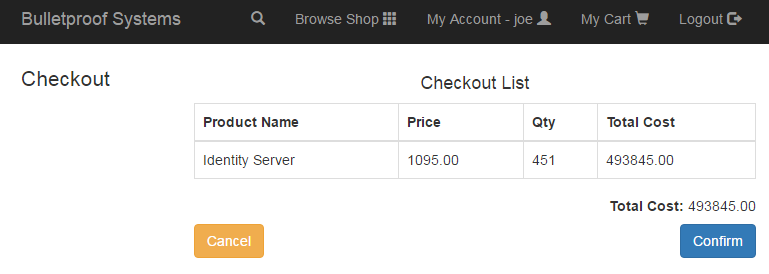
If the custom buy 451 (database is 450) Identity Servers as show below,



Confirm the Checkout List will lead to a message displaying on user’s screen.

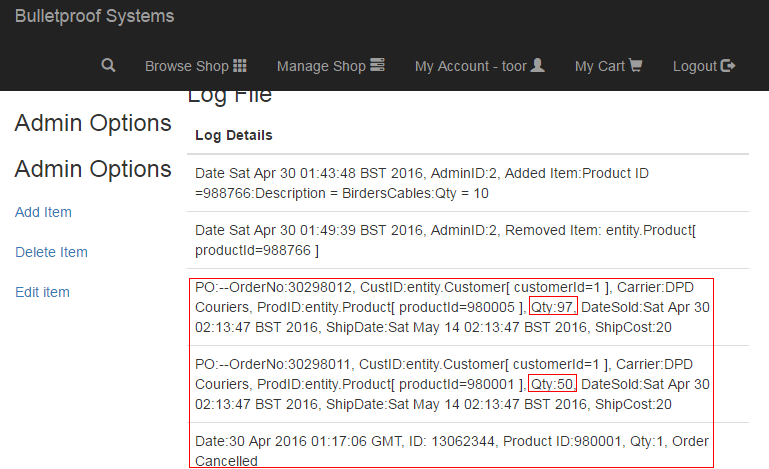


The customer cannot checkout.



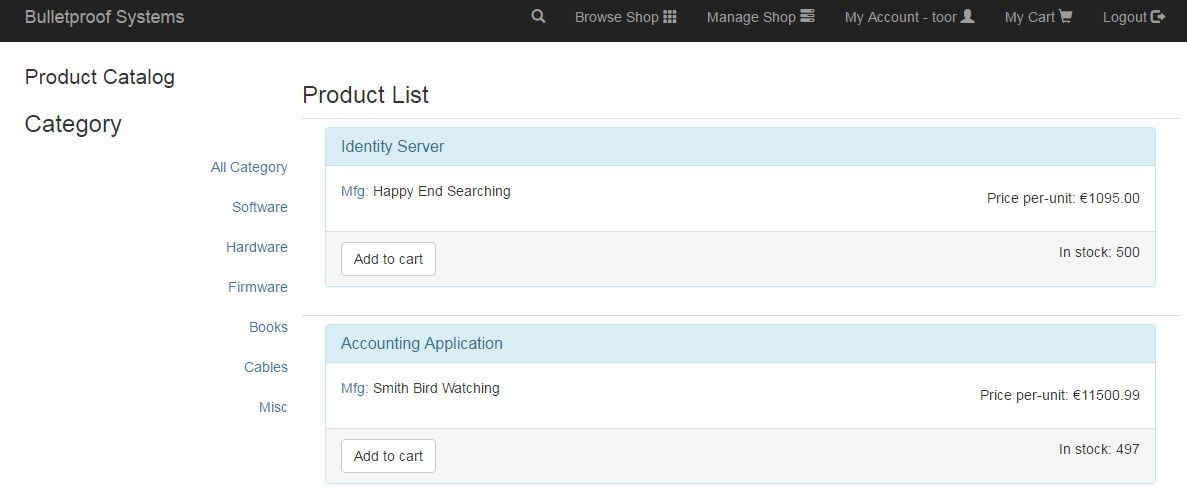
Now we logged in as toor

The confirmation and cancels of two orders we did before are logged as below.

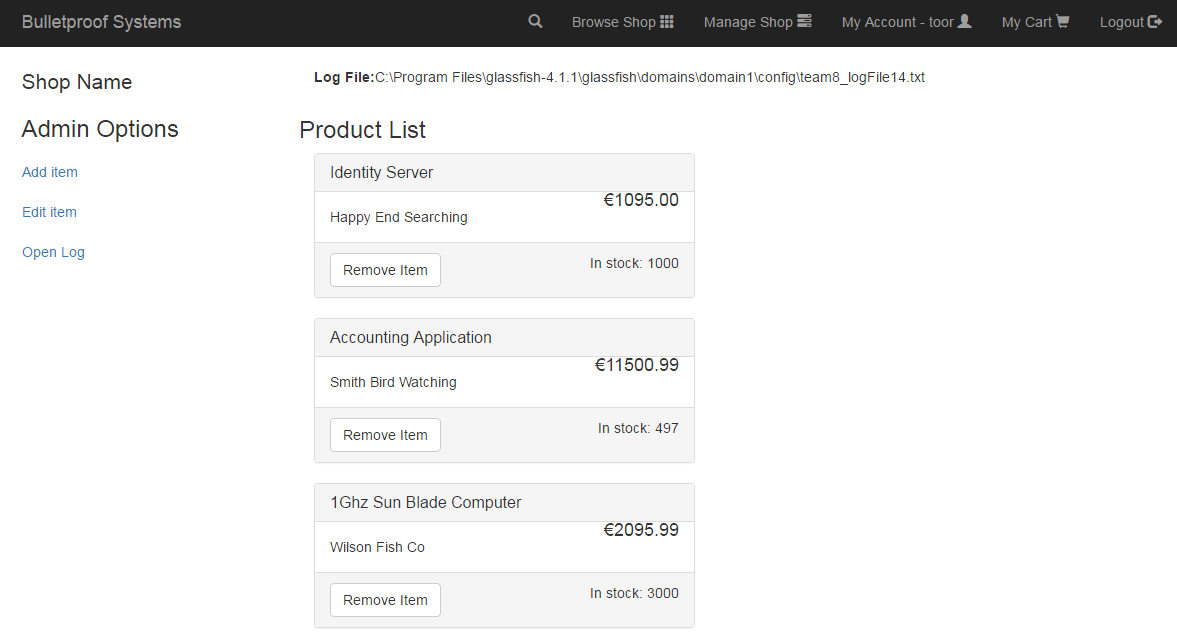


**Fulfilment of Project Requirements: Administrator Functions**

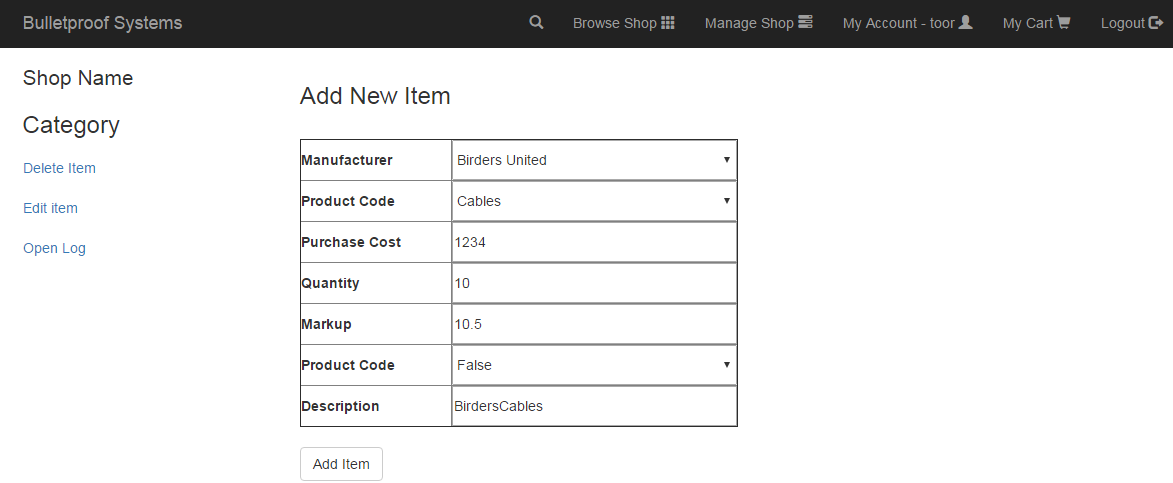
Logged in as toor:



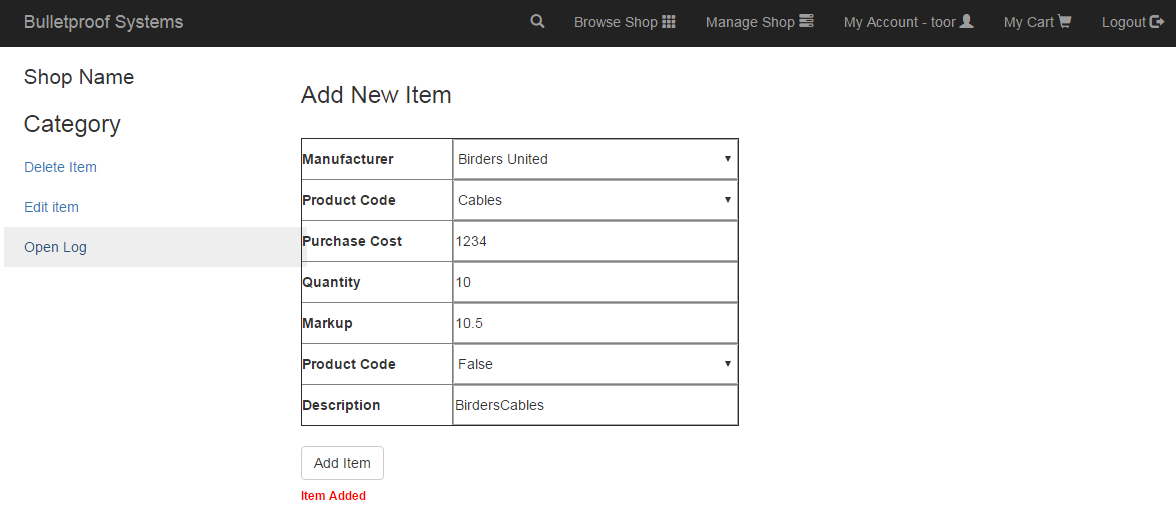
Manage shop (Only logged in as toor can access the function “Manage Shop”)



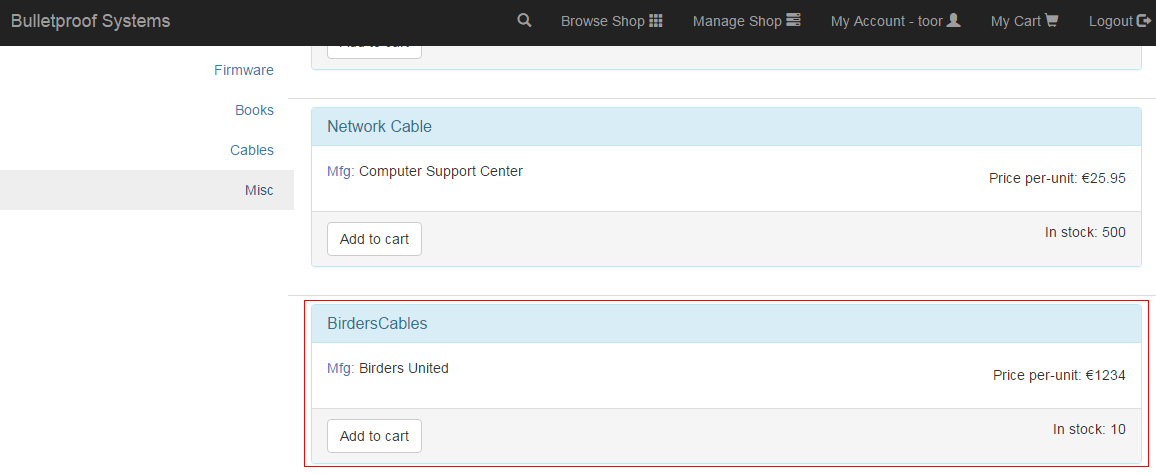
* + - Add new products to the sale database



Item Added:

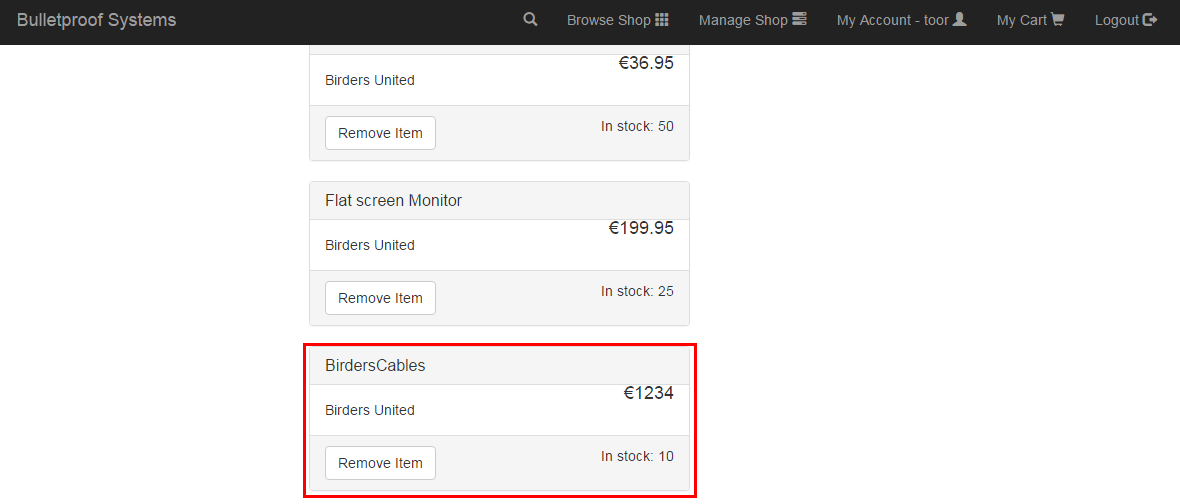


Item added successfully

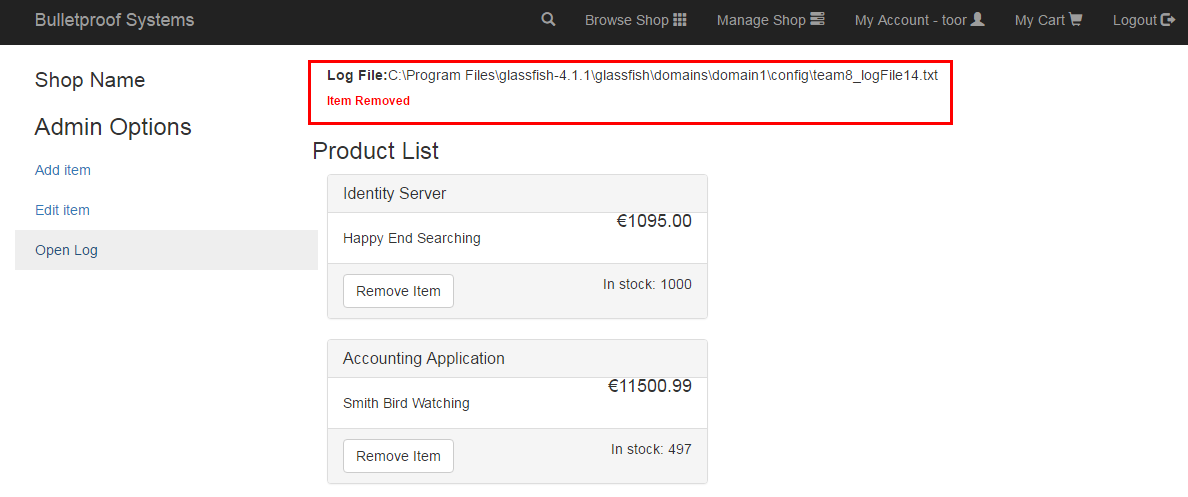


* + - Remove products from the sale database

Remove item BirdersCables we just added.

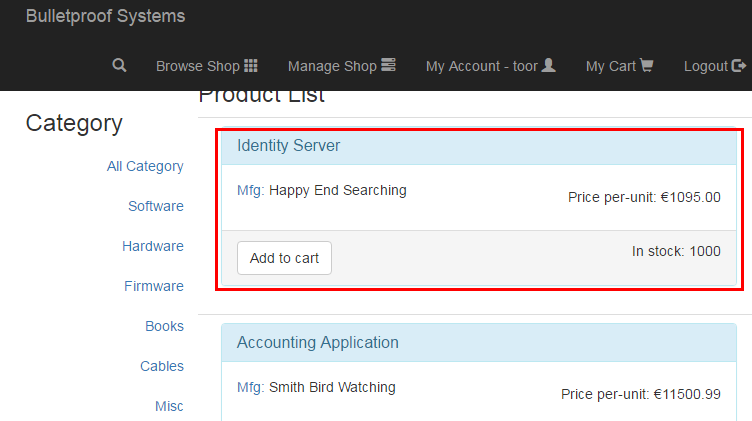


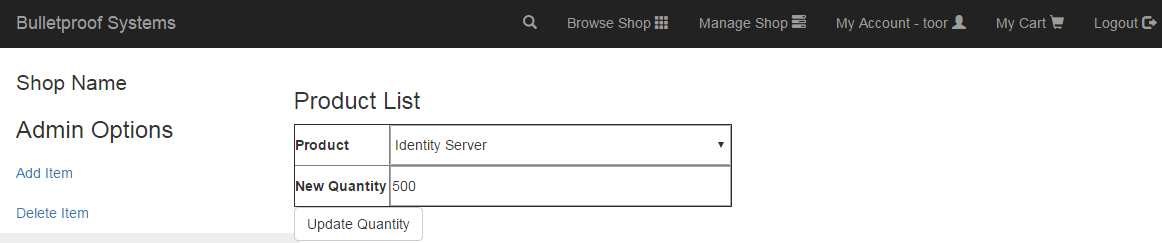
Item removed successfully

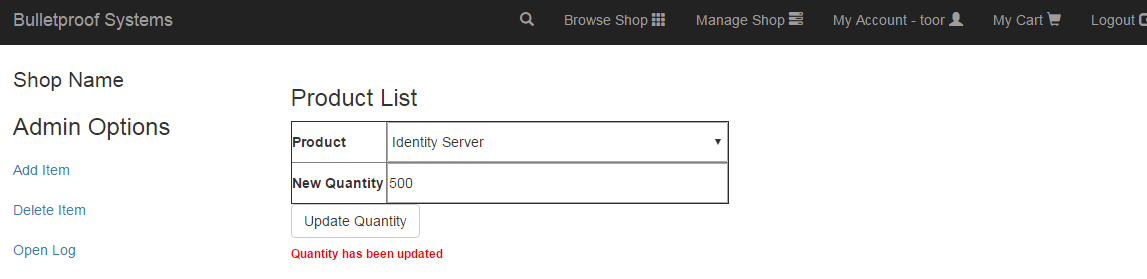


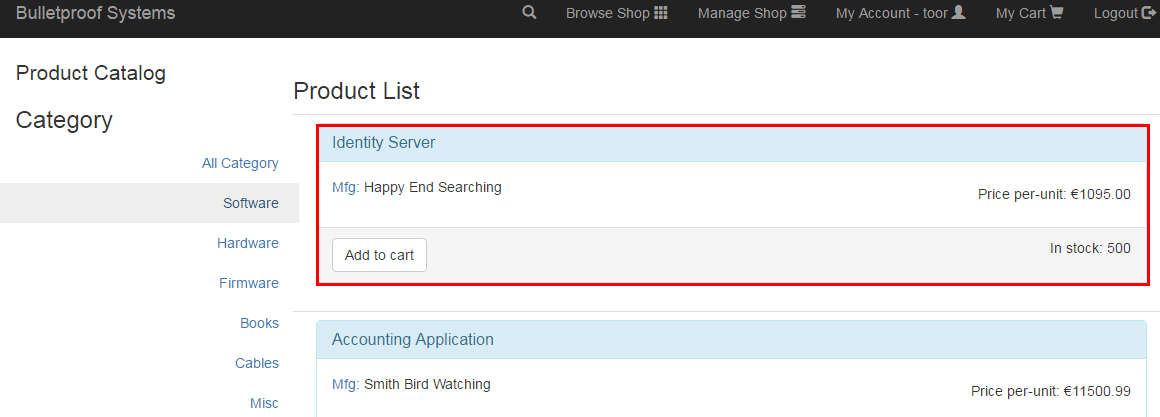
* + - Increase/decrease available amount of any product

The quantity of Identity Server updated before is 1000

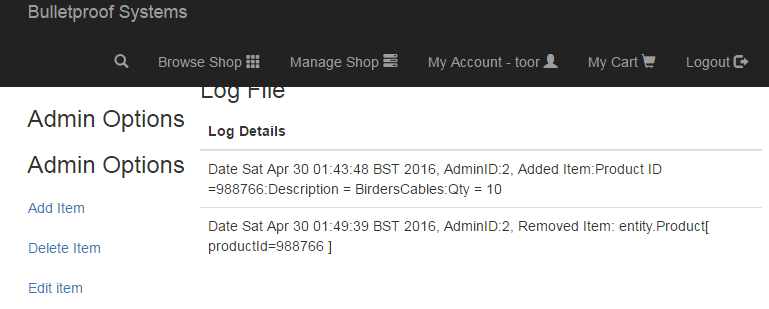


Update the item Identity Server to New Quantity of 500

Quantity has been updated

Quantity updated successfully

* + - Every time an administrator adds/removes a product an entry is added to the log

Two entries of adding and removing an item above were added to the log.

**Fulfilment of Project Requirements: Detail how the techniques used that can ensure the application is not vulnerable**

**A1 – Injection:**

These flaws occur when an application sends untrusted data to an interpreter. These types of flaws are found in SQL queries, LDAP queries, XPath queries, OS commands and other program arguments etc. These are easy to discover when examining code, but more hard via testing. In few cases, scanners and fuzzers can help attackers find these flaws.

Example in Shopping cart context:

In the context of shopping cart, we are using some commands in entity class like this:

"SELECT p FROM Product p WHERE p.productName=:pname OR plproductCode=:pid”

To prevent SQL injection.

**A2 – Broken Authentication and Session Management:**

These types of flaws are found in the context of building the account logging, secret questions, password management and few issues.

Example in Shopping cart context:

In this context, we are not passing the session ID’s in the URL which prevents the attackers to manipulate the ID’s and broke the authentication.

**A3 – Cross-Site Scripting (XSS):**

JSF is designed to have built-in XSS prevention. We can safely redisplay all user-controlled input including request headers, request parameters and request bodies by using any JSF component.

In the case of this project:

<h:outputText value=”#{userProfileBean.customerDetails.name}” />

<h:outputText value=”#{userProfileBean.customerMessage}” escape=”true”/>

<h:inputText id=”movie” value=”#{item.product\_quantity}”></h:inputText>

Etc…

Checking input in edit profile

<h:inputText id="name"

styleClass="form-control"

value="#{editProfileBean.name}"

p:placeholder="Name"

required="true"

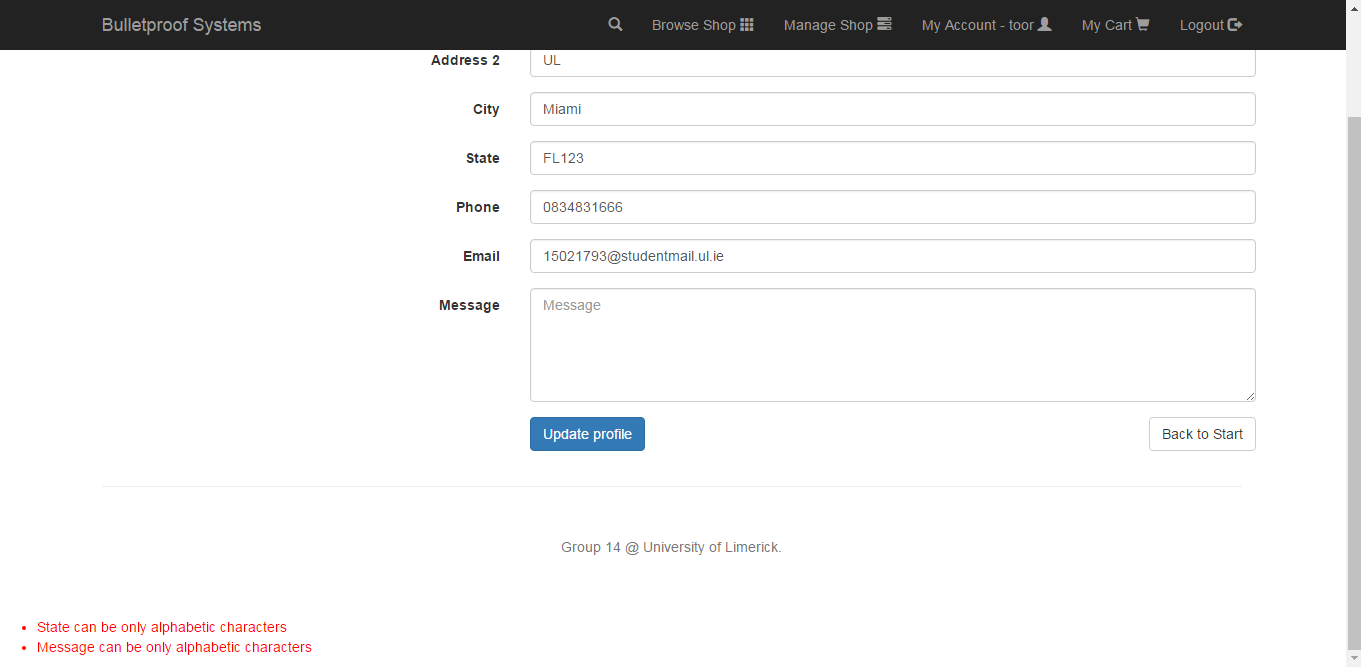
requiredMessage="Name is required"

validatorMessage="Name contains invalid characters">

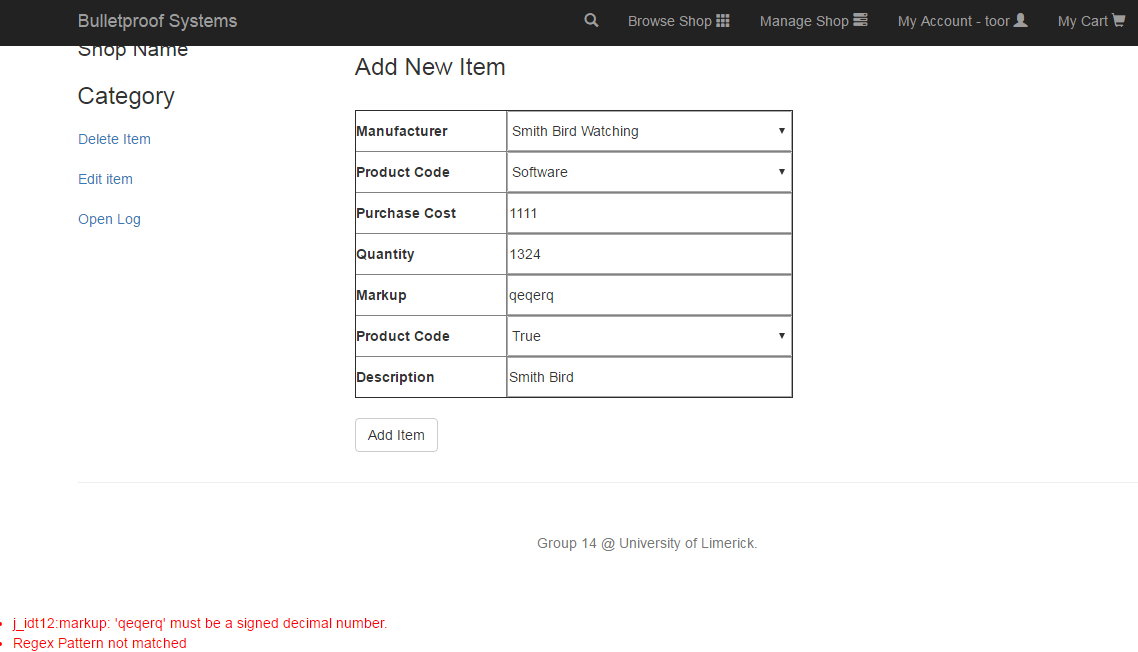
<f:validateRegex pattern="^[^\?\%\+\$\(\)\{\}\;\'\&amp;\&lt;\&gt;]+$" />

</h:inputText>

Only valid input can be updated



Adding invalid new item



**A4 – Insecure Direct Object References:**

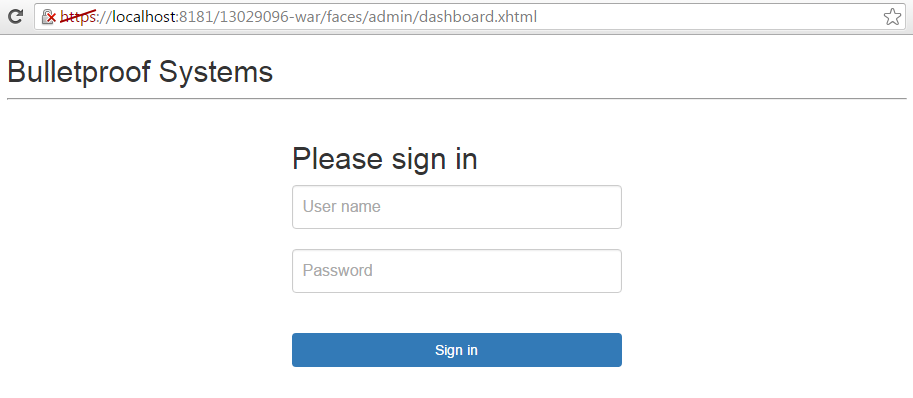
In many applications, the access is done by the actual name or key of the object. In such circumstances, these applications will not authorize the entry always, which results in insecure direct object access. This flaw can be easily noticed by manipulating some fields in the applications.

In the context of this project, we use per user or session indirect object references. Invalid users cannot directly target unauthorized resources. Alternatively, we are not passing the user’s ID or username to the URL.

**A7 – Missing Function Level Access Control:**

Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

In this project, only authorized user can login to the system and cannot access any webpage when not logging in.



This function is implemented in the web.xml file:

<security-constraint>

<display-name>AuthorizedAccess</display-name>

<web-resource-collection>

<web-resource-name>shopping\_catalog</web-resource-name>

<description/>

<url-pattern>/faces/shop/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<description/>

<role-name>user</role-name>

</auth-constraint>

</security-constraint>

Only logged in user role can access the url patterned /face/shop/\*, user without login cannot access any url with pattern /face/shop/\*

**A8 – Cross Site Request Forgery (CSRF):**

JSF 2.x has already built-in CSRF prevention, featured with javax.faces.viewstate hidden filed in the form when using server side state saving.

We’ll have a CSRF attack hole only when using views as in <f:view transient=”true”>, or there’s a XSS attack somewhere.

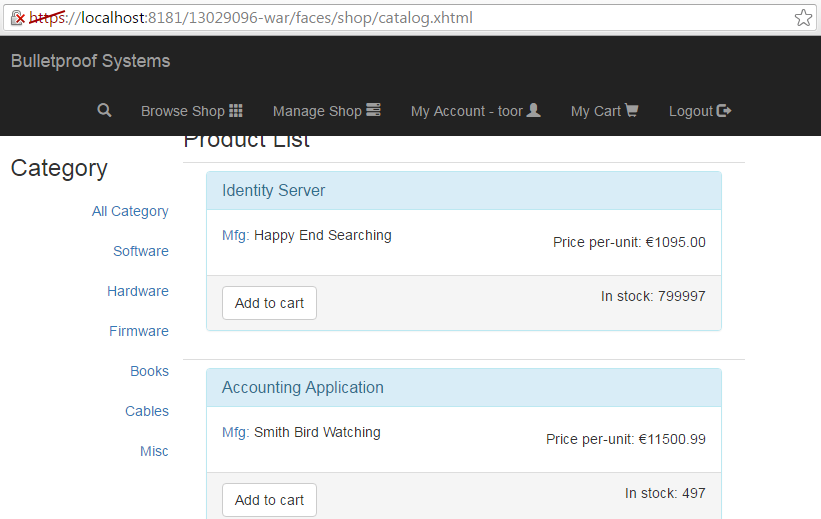
In the context of this project, all transient is false.

**A9 – Failure to Restrict URL Access:**

In many cases, some applications are not protecting the page requests properly. Sometimes, URL protection is managed via configuration and system is misconfigured. Hardest part is to detect the vulnerable pages.

Example in Shopping cart context:

Logging in the system using administrator username “toor”



After logging out the system and input the URL <http://localhost:8080/13029096-war/faces/shop/catalog.xhtml>,

The catalog page cannot be opened unless log in again.



The implantation of avoiding this vulnerability:

* User that logged in the system as customer can access all the webpages.

<security-constraint>

<display-name>AuthorizedAccess</display-name>

<web-resource-collection>

<web-resource-name>shopping\_catalog</web-resource-name>

<description/>

<url-pattern>/faces/shop/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<description/>

<role-name>user</role-name>

</auth-constraint>

</security-constraint>

* User that logged in the system as administrator can access all the webpages.

<security-constraint>

<display-name>AdminAccess</display-name>

<web-resource-collection>

<web-resource-name>AllAdminOperations</web-resource-name>

<description/>

<url-pattern>/faces/admin/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<description>Admin Only Access</description>

<role-name>admin</role-name>

</auth-constraint>

<user-data-constraint>

<description>Secured Login</description>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

* Login authentication make sure only the valid user can log in the system

<login-config>

<auth-method>FORM</auth-method>

<realm-name>file</realm-name>

<form-login-config>

<form-login-page>/login.xhtml</form-login-page>

<form-error-page>/error.xhtml</form-error-page>

</form-login-config>

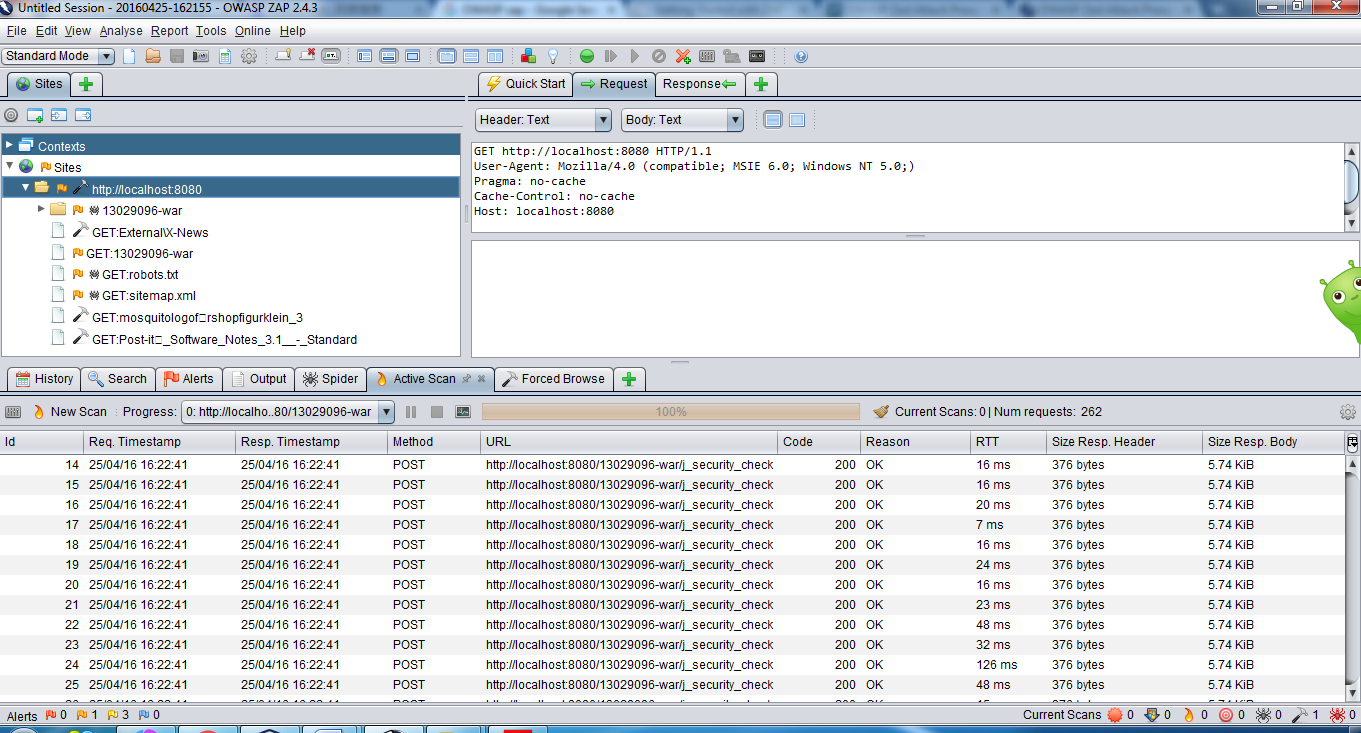
</login-config>

**OWASP ZAP Description**

* Security against specific OWASP Top 10 vulnerabilities
* OWASP ZAP Tool was run to test the application for security flaws.
* OWASP ZAP did not detect any security flaws.

OWASP Zed Attack Proxy (ZAP) is a common tool that used to find security vulnerabilities automatically in the web application while developing and testing the applications. This open-source tool was developed at the Open Web Application Security Project (OWASP). It is the Swiss army knife of web assessment tools. Its active scanner is integrated into many of the other functions of the application. Having a proxy and other tools built in is a huge plus. One of the unique features of ZAP is that its sensitivity and scan aggressiveness can be manually configured. There are three sensitivity settings---high, medium and low. ZAP allows a user to save sessions and persist sessions allowing you to take a break from your testing and come back to it. This is also a helpful feature when you need to confirm fixes and remediation.

Furthermore, ZAP is strong in consistency. Any difference between tests in ZAP is most likely caused by user. Before initiating the active scanner, it would be wise to spider the site multiple times. It has the ability to build on previous spider results until there are no new pages found.



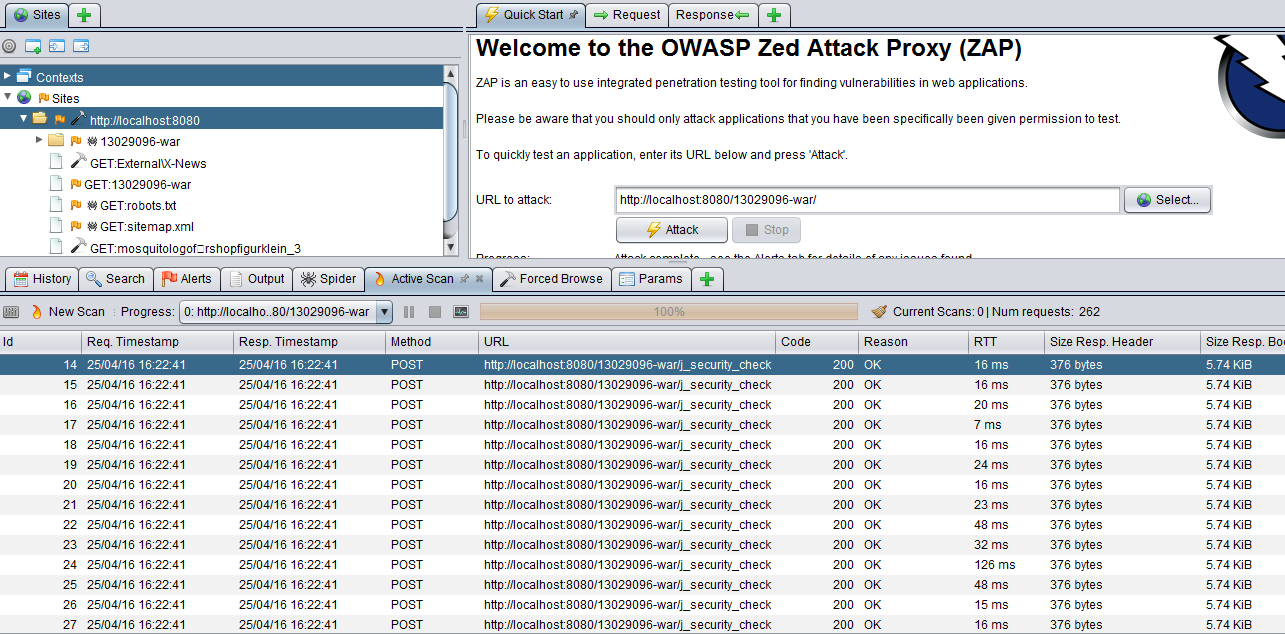
**Fulfilments of Project Requirements: Discuss how the application was tested to ensure the chosen defence is working correctly.**

* OWASP ZAP was used to testing the following vulnerabilities:
* A1 (injection)
* A3 (cross-site scripting) (XSS)
* A5 (security misconfiguration) (some instances)
* A6 (sensitive data exposure) (some instances)
* A8 (Cross site request forgery) (some instances)
* A10 (invalidated redirects and forwards)
* The other vulnerabilities (as well as certain cases of some of those listed above) are hard or impossible to test for using automation and require manual testing. This usually involves surfing around with a browser proxied through OWASP ZAP, setting breakpoints, and then manually modifying requests before they are sent to the server.

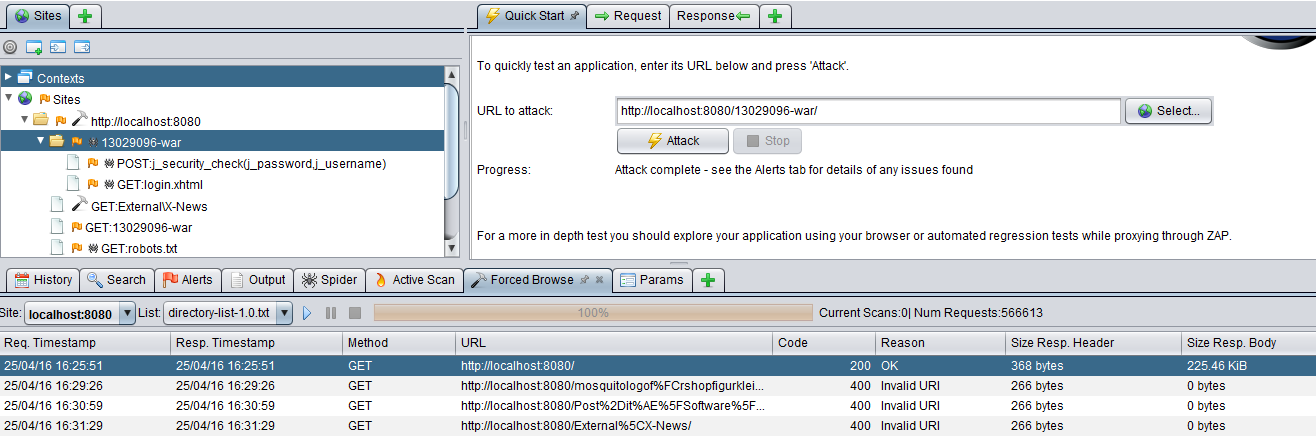
**Quick Start** is used to quickly test a web application with the active scanner.

The following release quality active scan rules are included in quick start.

* Buffer Overflow
* Code Injection
* Command Injection
* Client Browser Cache
* Cross Site Scripting (reflected)
* CRLF Injection
* Directory Browsing
* External Redirect
* Format String Error
* Parameter Tampering
* Path Traversal
* Remote File Include
* Server Side Include
* SQL Injection

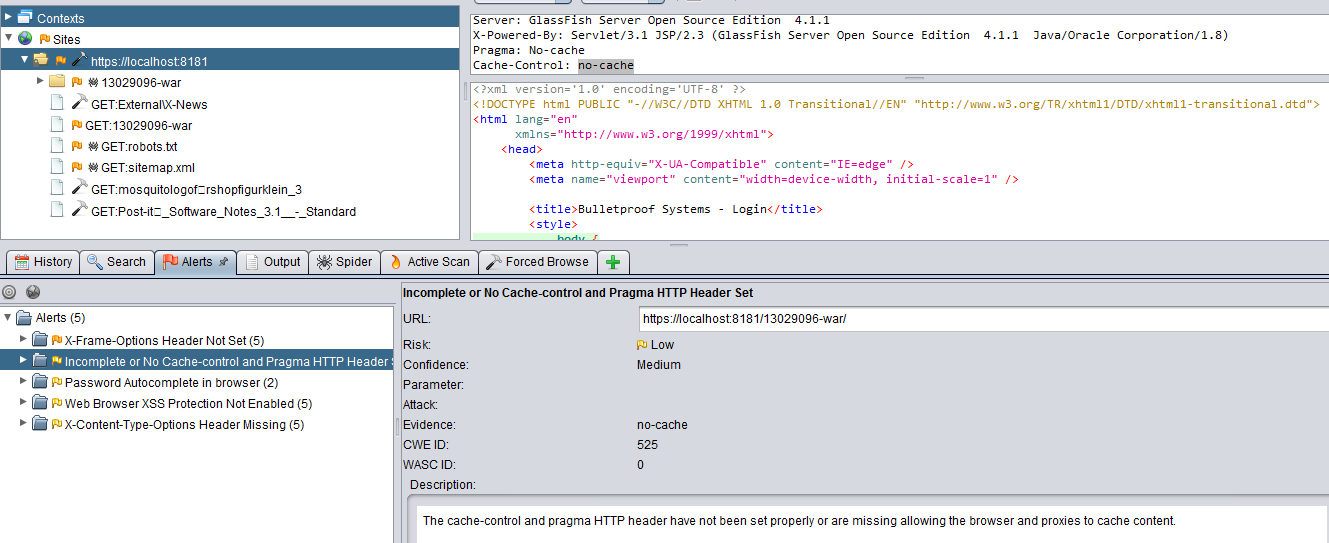


**Forced Browse** is used to try to discover directories and files using forced browsing. A set of files are provided which contain a large number of file and directory names. ZAP attempts to directly access all of the files and directories listed in the selected file directly rather than relying on finding links to them. It was used to test OWASP A7 Missing Function Level Access Control.

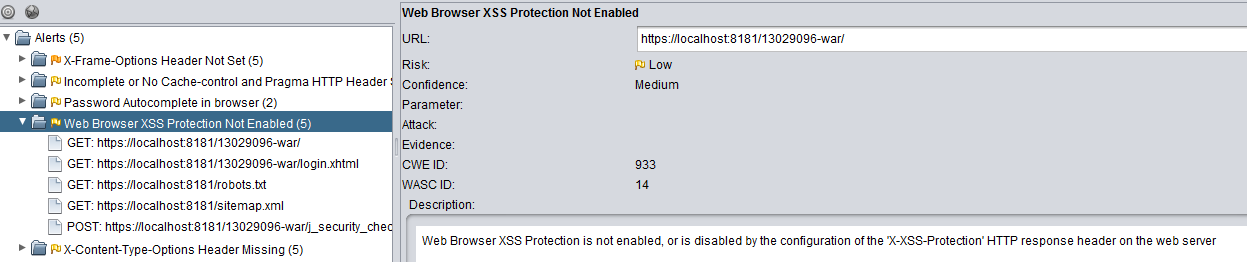


From the result above, we can see that only the URL of <http://localhost:8080/> can be opened, the rest of them are invalid URLs and cannot be opened without logging in.

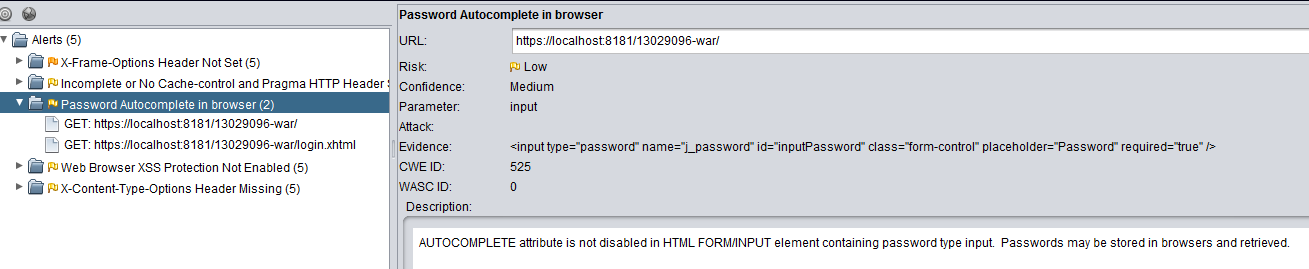
**Alerts tab** is used to show the results after active scanning of the website.



After the active scanning, we can see that there is a low risk of XSS --- web browser XSS Protection is not enabled, or is disabled by the configuration of the “X-XSS-Protection” HTTP response header on the web server, which is caused by the configuration of the Browser.



Alert “Password Autocomplete in browser” can be avoided if cancel the auto-fill function of the Browser.



Other than that, there are no vulnerabilities such as Injection, CSRF, Missing Function Level Access Control, Broken Authentication & Session Management, Failure to Restrict URL Access and Insecure Direct Object Reference.

**Summary**

After testing our application with OWASP ZAP tool, there are just a few medium standard alerts. Our application is not vulnerable to the required OWASP Top 10 vulnerabilities.