WEB ROUTING

SECURITY MANUAL

SLIPPERY ROCK UNIVERSITY

Joshua Gearhart jjg1018@sru.edu

Nicholas Bushée nab1017@sru.edu

Contents

[1. Synopsis 3](#_Toc120483317)

[2. Roles 3](#_Toc120483318)

[3.Access Limitations 3](#_Toc120483319)

[3.1 PErmissions based denials 3](#_Toc120483320)

[3.2 persistence of permissions denials in html 3](#_Toc120483321)

[3.3 ENSURING DENIALS VIA WEB SECURITY CONFIGURATION 6](#_Toc120483322)

[4. Password Security 7](#_Toc120483323)

[4.1 PASSWORD HASHING 7](#_Toc120483324)

[5. User entity requirements 7](#_Toc120483325)

[5.1 Username requirements 8](#_Toc120483326)

[5.2 PASSWORD REQUIREMENTS 8](#_Toc120483327)

[5.3 EMAIL REQUIREMENTS 8](#_Toc120483328)

[6. Handling Errors 8](#_Toc120483329)

[6.1 Login error Handling 8](#_Toc120483330)

[6.2 REGISTRATION ERROR HANDLING 9](#_Toc120483331)

[6.3 Data Type Error Handling 9](#_Toc120483332)

[6.4 Dependency ERROR Handling 10](#_Toc120483333)

[6.5 DATABASE ERROR HANDLING 10](#_Toc120483334)

[7. verifying no duplications 10](#_Toc120483335)

[8. Denying SQL INJECTION 10](#_Toc120483336)

[9. READING FILES 10](#_Toc120483337)

[10. LOGGING INCIDENTS 10](#_Toc120483338)

[11. AVOIDING BOTS 11](#_Toc120483339)

[12. FORGOTTEN PASSWORD 11](#_Toc120483340)

[13. EMAIL VALIDATION 11](#_Toc120483341)

[14. DATA LIMITATIONS 11](#_Toc120483342)

[15. FIGURES 12](#_Toc120483343)

# 1. Synopsis

This document is a complete overview of the Webrouting Application’s security measures and what is implemented to improve user experience and application security.

# 2. Roles

The Webrouting Application implements Role-Based Access Controls within the entirety of the application. This is to ensure that security is persisted throughout all aspects of the application. These mechanisms of security ensure that Users must be assigned roles to access different parts of the application and depending on the assigned roles defines what users can and cannot access.

# 3.Access Limitations

This section will highlight the implementation of security mechanisms to deny access to users based on their roles and how this is persisted in other aspects of the application as well as how this denies elevation of privilege.

## 3.1 PErmissions based denials

In the Webrouting Application there are four different roles implemented to fit different needs of the user, and the security of the application.

The first role is the ***Admin*** role it’s identifier in the program and in the database is “***ADMIN***” the admin can change any users account information. However, they can’t change the username. And when the admin is changing the user’s password, they cannot do this manually. They must click a check box to indicate they wish the user to change their password which will send the user a forgotten password email.

The second role is the ***Shipper*** role, it’s identifier in the program and in the database is “***SHIPPER***” the shipper role accounts have access to the shipments page. Shippers are allowed to create shipments for carriers to bid on, as well as accept those bids placed on shipments.

The third role implemented into the Webrouting application is the ***Carrier*** role. It’s identified as “***CARRIER***” Carriers functionality is high. They have access to contacts and creating them, locations pertinent to the carrier as well as being able to add locations. As well as locations carriers are also able to add vehicles and vehicle types. Drivers that are associated with vehicles. Technicians for doing work on the vehicles and making maintenance orders to deliver to a technician for work. As well as a carrier’s tab, shipments tab and routes tab.

And the fourth role of the program is the ***Master List*** role known as the “***MASTERLIST***” The master list role shares some functionality with the carrier role just modified. The master list can view all shipments, and all carriers. The master list is also able to view the routes of the shipments.

## 3.2 persistence of permissions denials in html

In the application Role Based permissions are softly implemented in the HTML pages. All HTML pages have a simple has Authority statement (Figure 1).

Diagram

Description automatically generated

Figure 1 – Activity Diagram illustrating HasAuthority method

As shown in Figure 1 we can see that if the user’s role is not permitted to the page, then the page is hidden from their view, inversely if the user’s role is privileged to access the page, then the page will be made available to them. This is important as it satisfies confidentiality in the program. If the user does not have a role with privilege to the page, then they do not need the option to access it. As well as this keeps malicious users from identifying possible targets to attack and manipulate on the system. In figures 2, 3, 4, 5, will illustrate the application examples of how this looks.



Figure 2 – Admin Role example of the HasAuthority function

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 3 – Shipper Role example of the HasAuthority function

A picture containing text

Description automatically generated

Figure 4 – Carrier Role example of the HasAuthority function

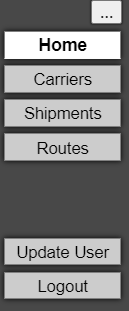


Figure 5 – Master List Role example of the HasAuthority function

## 3.3 ENSURING DENIALS VIA WEB SECURITY CONFIGURATION

The previous sub-section implements Role Based permissions softly. This section will talk about a hard implementation of role-based permissions. In the Webrouting application Spring boot is used as a framework for the application. In this framework the availability of configuring security is high. The webrouting application uses this availability to implement security, by implementing the ***WebSecurityConfigurerAdapter*** the application is allowed to implicitly approve and deny access to users based on their roles. The previous section talks about satisfying confidentiality. If a malicious user were to gain information on other pages not accessible by their account without the ***WebSecurityConfigurerAdapter*** they would be able to possibly elevate their privilege or gain access to data, they are not permitted to. By implementing this into the application we persist confidentiality, availability, and integrity, as well as denying elevation of privileges. Figure 6 will show an example of this.

Shape

Description automatically generated

Blank Page

Figure 6 – Shipper Account Trying to Access an Admin Page

# 4. Password Security

This section will highlight the use of Password security implemented into the Webrouting Application and how this satisfies the security requirements of confidentiality, integrity, and availability of the Applications passwords.

## 4.1 PASSWORD HASHING

In the Webrouting Application passwords are stored in the database. The user enters their password not knowing what happens to it. In the backend the program utilizes a package called the “***BCryptPasswordEncoder”*** which is a password hashing (Figure 7).

|  |  |
| --- | --- |
| Password | Password |
| Hashed Password | '$2a$10$EVBsfb2HGqaMlI9z443kR.zM.Tn66fT.7nbvsVhDeqAw.fc4HyXOG' |

Figure 7 – Example of BCryptPasswordEncoder

This security mechanism is what helps keep the user’s password confidential, as well as keeps the passwords integrities as the actual password is not changed for the user. Furthermore, the only person who can access this hashed password would be the Data base administrator. No role in the application has direct access to a user’s password except for the user, the user may update their password, which will then be hashed and changed in the database, the admin role may reset a user’s password, but this will send a reset password link to the user’s email.

# 5. User entity requirements

This section will discuss User entity requirements, or more plainly user defined inputs and requirements implemented on those inputs this is achieved through a custom ***User Validator*** and will be discussed in the sub-sections of this section.

## 5.1 Username requirements

In the Webrouting Application a user may make an account with their account a username is associated with them.

The requirement of the username is as follows:

* Username **must** be unique
* Username **must** be six characters long minimally and cannot exceed thirty-two characters

The user is also confined to this username, and they cannot change it.

Note: Section

## 5.2 PASSWORD REQUIREMENTS

The Webrouting Application also implements password requirements through the ***User Validator***

The requirement of the password is as follows:

* Password **must** be between eight and cannot exceed thirty-two characters

The user may change their password by logging in and updating it, or by following the steps of forgetting their password.

## 5.3 EMAIL REQUIREMENTS

In this application Email requirements are also implemented upon the user. Using the ***User Validator*** class, the requirements for emails are as follows:

* A user’s email must be between 3 and 64 characters long
* A user’s email must contain an @ character
* A user’s email cannot be the same as another user’s email
* A user’s email must be a real email account (See Section 13)

# 6. Handling Errors

Errors should be always handled during Application run times and prevented. This section of the security manual will highlight areas where errors are handled at.

## 6.1 Login error Handling

During the login process if a user enters an incorrect username or password a message is displayed to the user indicating that a field, they filled out is incorrect (Figure 8).

Graphical user interface, application, website

Description automatically generated

Figure 8 – Login Errors on Page

It is important to note that the error message does not explicitly tell you what is wrong, this is important since malicious actors do not know which one is wrong with this message and ensures that confidentiality in security is maintained.

## 6.2 REGISTRATION ERROR HANDLING

Error handling is also handled very similarly to the login in the registration, this error handling is a bit more implicit as it needs to be so the user knows what field to change if the error occurs (See Figure 9).

Graphical user interface, application

Description automatically generated

Figure 9 – Registration Errors on Page

## 6.3 Data Type Error Handling

Data type errors are handled throughout the Webrouting Application errors will be thrown for users that try to input a datatype that the input does not take.

## 6.4 Dependency ERROR Handling

Dependencies exist throughout the Webrouting application. Entities are checked throughout the application. If an entity is required due to a dependency than the entity cannot be removed, and an error message is thrown to the user.

## 6.5 DATABASE ERROR HANDLING

Pertinent to System Administration, when the application is run, default data values are attempted to be added. If the default data is already in the application, then the errors are caught, and an explanation is provided (Figure 10).

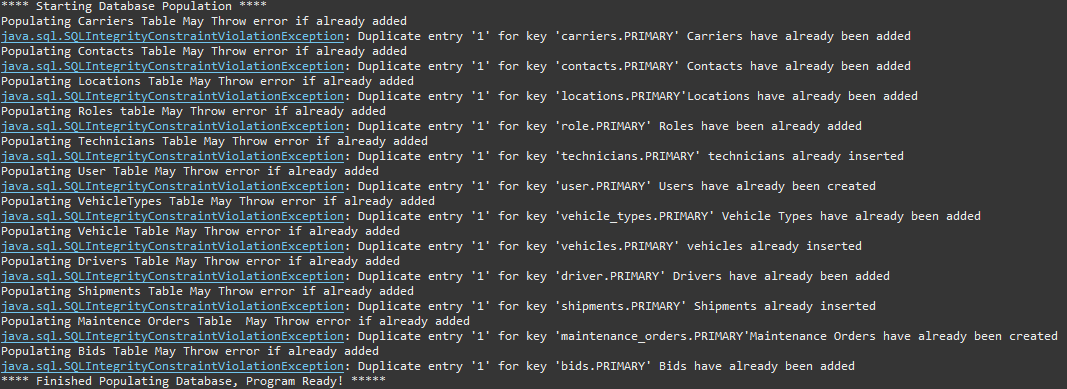


Figure 10 – Database Error Handling

# 7. verifying no duplications

In the Webrouting Application duplication checking is implemented to verify that information is not repeated. This can be seen in many aspects of the program varying from duplicate driver’s license numbers, usernames, and email addresses.

# 8. Denying SQL INJECTION

SQL Injection is a major issue to web applications and can seriously impact the services offered by them. The Webrouting Application handles SQL injection attacks via implementation of the CRUD repository and Hibernate Library. This compiles into an anti-SQL injection framework that satisfies confidentiality, integrity, and availability to users.

# 9. READING FILES

Reading files is apart of the Webrouting Application solely in the Shipments Controller class. The files are being iterated through and added to the database. If errors occurred than errors are logged for admin use.

# 10. LOGGING INCIDENTS

A logging class is implemented throughout the application where logging is pertinent to gathering information. Information needs to be logged for admin use when errors occur. This is so admins can understand errors encountered by users and attempt to mitigate these problems when encountered.

# 11. AVOIDING BOTS

Bots can be a detrimental issue to online service applications. They can pollute applications and provide false information to users. The Google ReCAPTCHA is implemented during sign-up to detect and determine bots, if a bot is detected through the Google API, then their request to prove they are human is denied, and they will not be able to create an account.

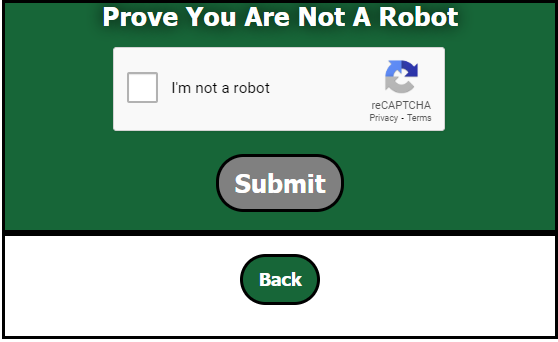


Figure 11 – Google ReCAPTCHA

# 12. FORGOTTEN PASSWORD

A forgot password is implemented into the Webrouting Application, this allows users to reset their password by following the button and entering their email address. A malicious actor could use this to try and reset another user’s password, However, An email is sent to the user in-case of this. As well as this does not deny account access. The user may change their account password or may not.

# 13. EMAIL VALIDATION

Upon account creation a user is denied access to their account until they follow through with the email verification. Once their email is verified, they are enabled in the database and gain access to the application. This enforces users to have a real email address as well as prove that it is real.

# 14. DATA LIMITATIONS

Data limitations are integrated into the application. This enforces that certain data stays in the standards of real data. These limitations are persisted into user inputs allowing for users not to by-pass data constraints in the database.

# 15. FIGURES

Figure 1 – Activity Diagram illustrating HasAuthority method 4

Figure 2 – Admin Role example of the HasAuthority function 4

Figure 3 – Shipper Role example of the HasAuthority function 5

Figure 4 – Carrier Role example of the HasAuthority function 5

Figure 5 – Master List Role example of the HasAuthority function 6

Figure 6 – Shipper Account Trying to Access an Admin Page 7

Figure 7 – Example of BCryptPasswordEncoder 7

Figure 8 – Login Error Page 9

Figure 9 – Registration Errors on Page 9

Figure 10 – Database Error Handling 10

Figure 11 – Google ReCAPTCHA 11