

National College of Ireland

Project Submission Sheet – 2019/2020

Student Name: Ashok Kumar Kothandan

Student ID: X19138857

Programme: M. Sc., Cybersecurity **Year:** Sept 2019

Module: Secured Programming for Web

Lecturer: 17-12-2019

Submission Due Date:

Project Title: Project – Technical Report

Word Count: 5363

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Ashok Kumar Kothandan

Date: 17-12-2019

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only

Signature:

Date:

Penalty Applied (if applicable):

Secure Programming for Web

Project

National Bank of Ireland (Banking Application)

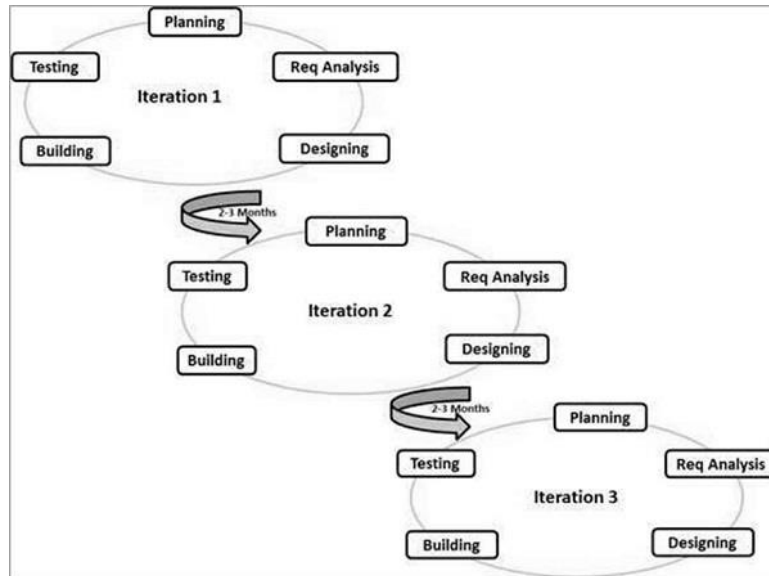
Technical Report

Ashokkumar Kothandan
National College of Ireland
MSc. Cybersecurity
2019-2020

EXECUTIVE SUMMARY

This is a working bank application which is designed in the manner with having the security aspects in mind. My application has multiple roles and permissions configured. I have used Hybrid Access Control technique where the access is based on roles, permissions, attributes, etc.

I have used agile model SDLC technique in designing this application. I started designing application and built and tested and again added new features it once done with the old one.



Basically, my application has three roles. They are ADMIN, EMPLOYEE, CUSTOMER. ADMIN role has multiple permission in creating, deleting new users.

BACKGROUND RESEARCH AND INVESTIGATIONS

NEEDS FOR APPLICATION

To design a banking application, I started to investigate on the needs for designing a web application for bank in the security aspect. As a bank application should me more secure in dealing with the money transactions. In this application I had an idea of customers creating their accounts online in the register column, whereas the admin will be creating the employee account and admin account.

In common all the users have access to create a new account, after login in they can view and edit their own profile. 'Customer' has the access can view his account details and can transfer money to other customer. Employee can deposit amount to the customers and can search for the details of the customer with the account number. Admin can create, delete, enable, disable all the user based on the permission. I have generated Account Number, NSC, BIC, IBAN for customers randomly.

SELECTION OF LANGUAGES AND TOOLS

To start with the application designing I need to choose the best programming language and tools to design a secure web application.

Java

With the 4 years of experience in working in JAVA I know the capability and the architecture of JAVA in means of performance and other rich API this service provides. Java is the best user-friendly programming language in the designing once the user knows to debug his errors. So, I decided to choose JAVA as my base programming language for the server side.

Even though I selected Java, I know implementation of securities is more complicated in this language comparing the other languages. So, I decided to search for frameworks that I can start designing a new web application having security in the aspect.

Spring Framework

Spring is an enterprise Java framework. Spring was designed to simplify Java EE development and make developers more productive. Spring makes use of Inversion of Control and Dependency Injection to promote good software coding practices and speed up development time [1]. I started spending more time in learning spring framework and It took one month to get the complete knowledge in learning the framework.

I use the Latest Spring Framework v5 to design the application. In spring framework, I used below specified technologies.

Spring MVC – MVC is Model-View-Controller design pattern where we will be getting all the three components together.

Spring Boot 2 – This featured helped in designing the application even better by simply annotating the components saving my lot of time from configuring the application.

Spring Security – Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements [2]. Spring security has given inbuilt function in creating a secured application.

Thymeleaf – Thymeleaf is a modern server-side Java template engine for both web and standalone environments [3]. This is used as a replacement for JSP and other server side templated in JAVA. The plugins this provide made easier in designing the front-end aspects.

MySQL (Database)

I used MySQL to store and retrieve the data, choose it because it is RDBMS and open source.

Hibernate

Hibernate is an Object-to-Relational-Mapping (ORM) framework. It simplifies database access for Java applications. By using the framework, you can easily store and retrieve Java objects by setting up some simple configuration mappings [1].

Bootstrap (CCS & JS)

Since I'm new in web designing to make the user interaction little responsive, I used Bootstrap toolkit of CCS and JS.

SOFTWARE DEVELOPMENT LIFE CYCLE



REQUIREMENT ANALYSIS

We are about to design a working bank application for the National Bank of Ireland, which can be used by customers, employee, admin of the bank. Now let us discuss on the requirement of each role.

Common login page and password reset page for all the users to reduce complexity without authentication. And after authentication users can view his profile and edit his profile in 'My Profile' tab. Users can change the password after authentication as well, but it should need user login details.

Customer – We need a register page for the customers for creating new account. We collect the required information from the customer and account is created by authenticating the email id. We need security in avoiding multiple account creation. After authentication customer homepage should show the account details of the customer. Allow money transfer for the customer when they have account details of the receiver. For authentication confirm the customer password before transferring.

Employee – Employee Account will be created by admin, so he can login in login page. Employer home page will be used for depositing money to the customer. Security should be high in depositing the money. Verify the employer before depositing the amount. Employer can search for the customer to get the details of them.

Admin – Admin account will also be created by admin with authorized permission. Admin should have the role in creating a new account to customer, employee, admin. Also, admin can enable or disable any user. Admin can also delete a user with authorized permission. All the admin is not allowed to do everything. When creating each admin should be given permission.

Some additional requirements are all the passwords need to be hashed not encrypted with best encryption tool available. All the important field need to be validated and sanitized properly, to defend any kind of injection.

In addition to this it is recommended for the developer to refer the OWASP top 10 2019 list of security risk and implement as much as security as possible.

ARCHITECTURE

Since we are following MVC design pattern we must create the models first. To Design the models, we will be using spring MVC with JPA hibernate that connected to the My SQL. By enabling the DDL auto update function data base is automatically created. So, let us discuss on the tables and fields we are going to create in store our values in the database.

MODEL

The following the ER (Entry Relationship) Diagram of each of the tables in the database we use. Database name of the Project is set as “National_Bank_Of_Ireland”.

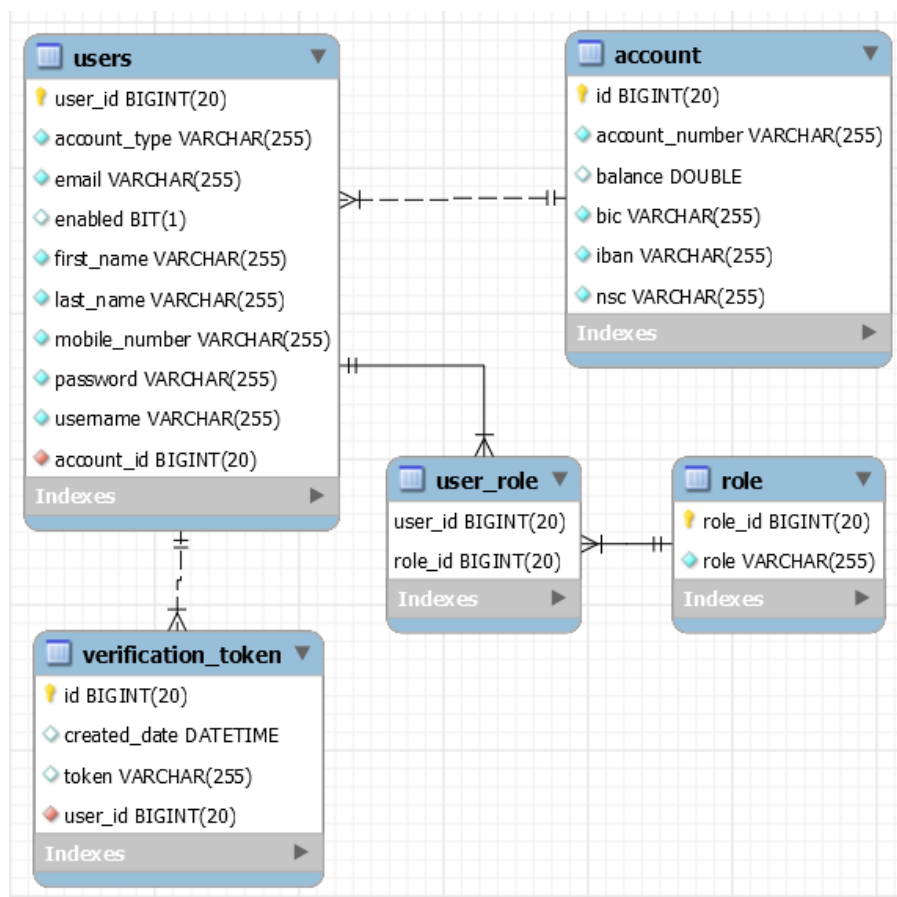


Table Users: to store the user details of all the roles.

User ID (Numeric) – Primary Key

Username (String) – Unique, Not Null

Email (String) – Unique, Not Null

Other details like First Name, Last Name, Mobile Number, Account Type – Not Null.

Account ID(Numeric) – Foreign Key of Account table.

Password (Hashed String) – Not Null.

Enabled (Boolean) – To enable or disable an account.

Table Account: to store the account details of the customers.

ID – Primary Key

Other details like Account Number, Balance, BIC, IBAN, NSC – Not Null

Table Role: to store the roles, and permissions in the database to retrieve it in future. This table is to be entered manually, once the database is created and never be alter in the future unless if there is a need in adding more roles and permissions.

Role ID (Numeric) – Primary Key

Role (String) – Not Null

Table User_Role: used to store the roles and permission of each user. This table is used because each user will be having many permissions to be handled. This will be created automatically when defining the role of the user in SET function as below.

```
@ManyToMany(fetch = FetchType.EAGER)
@JoinTable(name="user_role", joinColumns = @JoinColumn(name = "user_id"),
    inverseJoinColumns = @JoinColumn(name="role_id"))
private Set<Role> roles = new HashSet<>();
```

User ID – Foreign key of User table, Not Null.

Role ID – Foreign key of Role table, Not Null.

Verification Token Table: to store the verification token sent to the users to enable the user account. Also, the same table is used to store the OTP sent for resetting password. This table has an event that needs to be executed every day to delete the one-day expired tokens.

```
DELIMITER $$
CREATE EVENT `Token_Clean`
ON SCHEDULE EVERY 1 Hour STARTS '2019-09-01 00:00:00'
ON COMPLETION PRESERVE
DO BEGIN
    delete from national_bank_of_ireland.verification_token
    where created_date < DATE_SUB(NOW(), INTERVAL 1 DAY);
END;$$
```

ID (Numeric) – Primary Key

Created Date (Timestamp) – Not Null

Token (String) – Not Null

User ID (Numeric) – Foreign Key of User table, Not Null

VIEWS AND CONTROLLERS

After the model we will be discussing on the views and controllers of that are yet to be implemented. We will discuss the view and controllers based on the roles and the permission of the user.

Common User Control

First, we will discuss on the webpages that is common to all the users both before and after authentication.

Before Authentication

Login: Login page is common page to all the users. Before authentication, when a user tries to open any of authenticated page, he needs to be redirect to the login page automatically. This page required Username and Password as input. The input needs to be validated and session is created by verifying in the database if username and password of the user is correct. If not, the user will be thrown error. Implement the defense mechanism for controlling brute forcing.

Forgot password: Input of Username and Email ID is required to authenticate the user in this page. After validation of the given input the reset password OTP needs to be sent to the user to reset the password. Once the OTP is sent user is redirected to the reset password page, where the user will be allowed to reset the password confirming the OTP of the password.

After Authentication

My Profile & Edit Profile: My profile page will display the personal information of the authenticated user. Information displayed are First Name, Last Name, Email, Phone Number, Account Type. User can change his personal information in the Edit profile page except the account type.

Change Password: User can change his password by providing his old password as input.

Logout: There is no specific logout page. Once the user clicks on the Logout, user session needs to be destroyed and authentication needs to be cleared, followed by redirected to the login page.

Customer Control

Before Authentication

Register: This allows the customer to create a new account. Allow the user to select a username and it should be unique, also email of the user needs to be unique to send the verification token and OTP. Password should be strong that protects from hacking easily. Each field should be sanitized properly.

After Authentication

Home: Once the customer is authenticated, he should be redirected to the home page. In index page customer can see their account details and account balance.

Money transfer: Customer can transfer his money to other customer internally. To transfer money user must know the account details of the receiver, which includes the BIC, NSC, IBAN. For security reason it is recommended to verify the user password.

Employee Control

After Authentication

Home: Once the employee is authenticated, he should be redirected to the home page. In index page employee can deposit on behalf of customer. Confirm the employee about the customer before depositing the money.

Search Customer: Employee can search for the customer detail with account number. The details include both account details and customer details.

Admin Control

After Authentication

Home: Admin index page will display the 12 operations he can do based on the permission he has. The following are the 12 permissions.

Create Customer	Create Employee	Create Admin
Enable Customer	Enable Employee	Enable Admin
Disable Customer	Disable Employee	Disable Admin
Delete Customer	Delete Employee	Delete Admin

Admin is not allowed to do other operations which he doesn't have any permission.

Create: Create page sends parameter with the user type that needs to be created. Based on the user type the creation page differs, as customer needs account type and admin need the permission list.

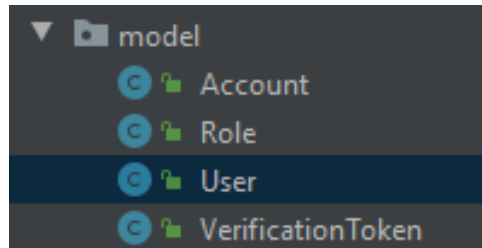
Enable/Disable/Delete: These pages send parameter with the user type that needs to be the action performed. Almost all the pages have same view and controller changes depends on the action need to be performed.

DESIGN AND IMPLEMENTATION

In the design and implementation phase we will be design the web application by creating the models and defining the configuration followed by the views and controllers of the page.

Model

As we already discussed on the model and its requirement in the architecture phase, we will be creating 4 java class for the table users, role, account, verification token.

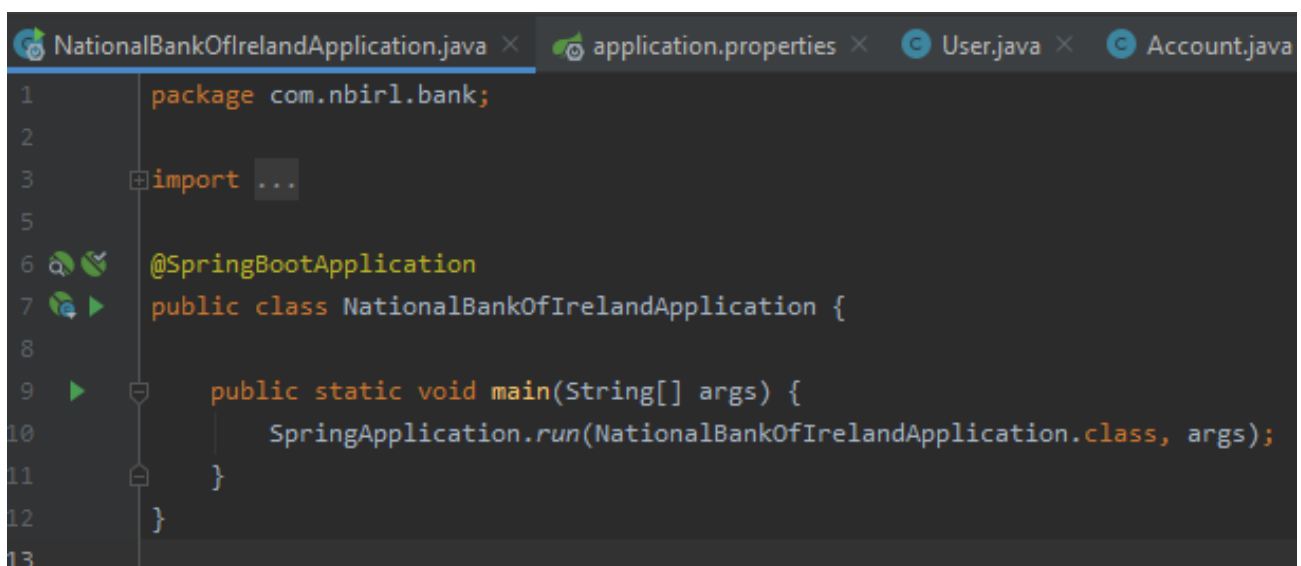


This java classes are only used for declaring the variables of the column name. You can refer the code for the variables created in each class under the 'model' section. Annotations are used to define the variables and tables. 'Lambok' is implemented for the auto getters and setter of the class.

Repositories and Services are created for all the Entities where the repositories are extending JPA Repository to retrieve the data from the database and services is extended to the repositories of the respective classes for extended data manipulations.

Configurations

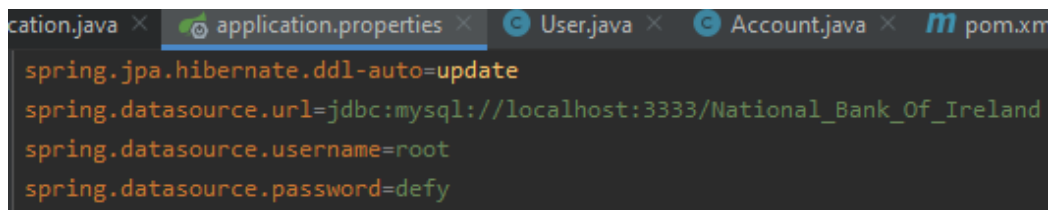
We use the default main initializer class. Implementation of @SpringBootApplication annotation defines the server that this is a spring boot application and the all the spring boot annotations are auto configured.



Pom.xml

We use Maven tool for the dependencies. Apache Maven is a build automation tool for the which is used to download the java dependencies automatically when the dependencies are mentioned in the POM.XML file. You can refer the code for POM.XML file for all the dependencies used in this project.

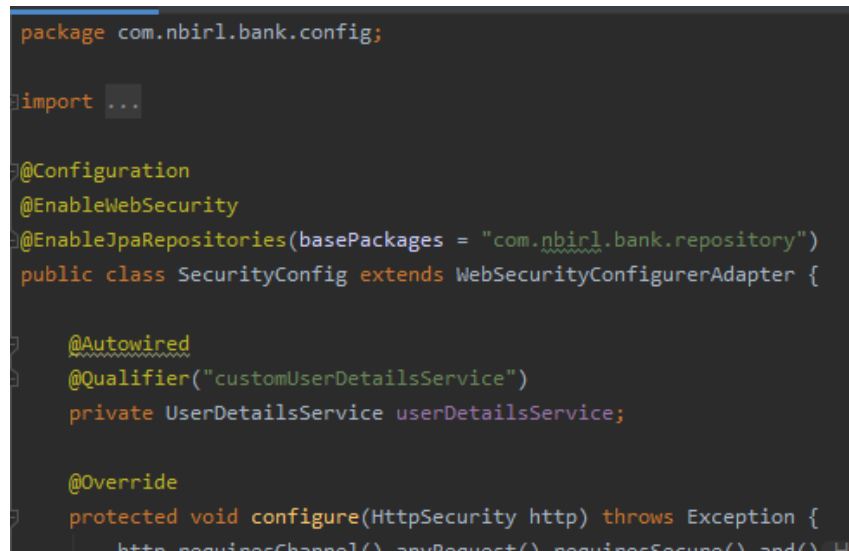
Database credentials are stored in the “application.properties” file where the spring boot automatically retrieve the data from it.

A screenshot of an IDE showing the 'application.properties' file. The tabs at the top are 'cation.java', 'application.properties', 'User.java', 'Account.java', and 'pom.xml'. The content of the file is:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3333/National_Bank_Of_Ireland
spring.datasource.username=root
spring.datasource.password=defy
```

SecurityConfig.java

This is one of the most important configurations where we will be configuring the spring security for enabling the security in our application. This class extends the “Web Security Configurer Adapter” class and we will override 3 methods to configure our security in it.

A screenshot of an IDE showing the 'SecurityConfig.java' file. The code is as follows:

```
package com.nbirl.bank.config;

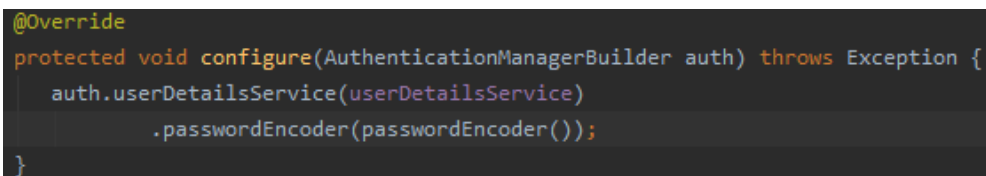
import ...

@Configuration
@EnableWebSecurity
@EnableJpaRepositories(basePackages = "com.nbirl.bank.repository")
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    @Qualifier("customUserDetailsService")
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.requiresChannel().anyRequest().requiresSecure().and().Ht
```

The configure “AuthenticationManagerBuilder” is used to retrieve the user login details for the verification of the login and the encoder we use for hashing password.

A screenshot of an IDE showing the 'SecurityConfig.java' file, specifically the 'configure' method. The code is as follows:

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService)
        .passwordEncoder(passwordEncoder());
}
```

The Configure “HttpSecurity” will be helpful in defining and enabling the inbuilt security configuration available in the spring security. We will be discussing elaborately on each of the securities we implemented in future.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.requiresChannel().anyRequest().requiresSecure().and().HttpSecurity
        .authorizeRequests().antMatchers( ...antPatterns: "/register", "/confirm-account", "/forgot-password", "/reset-password").permitAll() ExpressionUriAuth
        .antMatchers( ...antPatterns: "/money-transfer").hasRole("CUSTOMER")
        // .antMatchers("/confirm-deposit").hasIpAddress("11.11.11.11")
        .antMatchers( ...antPatterns: "/search-customer", "/deposit", "/confirm-deposit").hasRole("EMPLOYEE")
        .antMatchers( ...antPatterns: "/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated()
        .and().formLogin().loginPage("/login").defaultSuccessUrl("/").permitAll() FormLoginConfigurer<HttpSecurity>
        .and().logout().invalidateHttpSession(true).clearAuthentication(true).logoutRequestMatcher(new AntPathRequestMatcher( pattern: "/logout")).permitAll()
        .and().headers().cacheControl() HeadersConfigurer<H>.CacheControlConfig
        .and().xssProtection() HeadersConfigurer<H>.XXssConfig
        .and().frameOptions().sameOrigin() HeadersConfigurer<HttpSecurity>
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED).sessionFixation().migrateSession() SessionManagementConfigurers
        .and().csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse());
}
```

The configure “Websecurity” will ignore the most used files and folders in the web application like the Java script, User Define Errors pages, CCS, Fonts, Images folders.

```
@Override
public void configure(WebSecurity web) {
    web.ignoring().antMatchers( ...antPatterns: "/webjars/**", "/js/**", "/error/**"
        , "/css/**", "/fonts/**", "/libs/**", "/img/**");
}
```

Views and Controllers

Login page: Below is the front end of the login page, you can find the source code in templates folder



← → ↻ ⚠ Not secure | localhost/login ☆ 🧑

National Bank Of Ireland

LOG IN REGISTER

Please Sign In

Username

Password

Log In

You don't have an account? [Register](#)

Forgot Password? [click here](#)

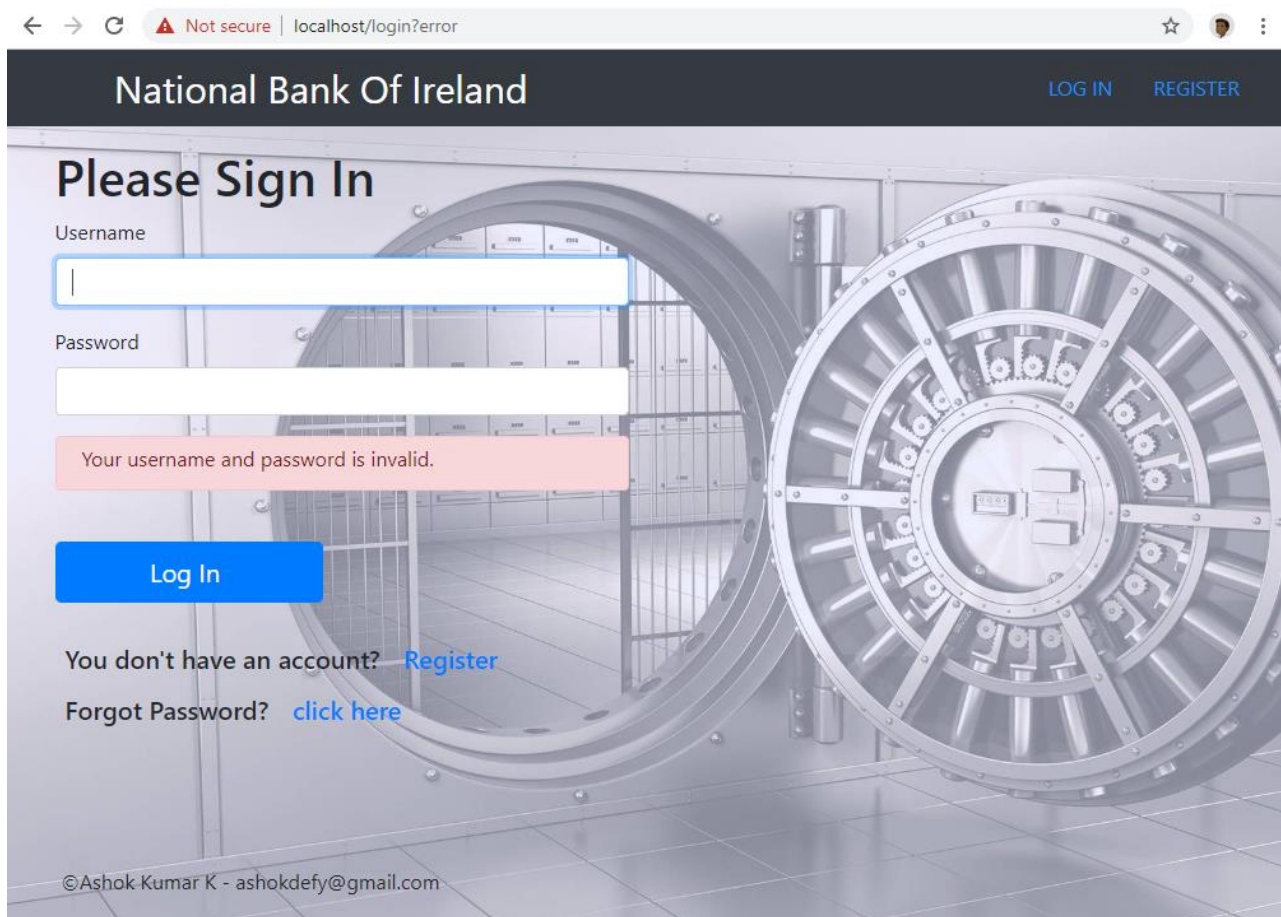
©Ashok Kumar K - ashokdefy@gmail.com

The controller of the login page is simple as this is taken care by the spring security by itself. If the user id and password verification fail, then the login will be sending the parameter error as true that we are adding the error message to the model attribute and redirect it to the login page. If the login credentials are correct, then the user will be redirected to the login page.

```
@GetMapping("/login")
public String login(HttpSession httpSession, Model model, String error){
    httpSession.invalidate();
    if (error != null)
        model.addAttribute( attributeName: "error", attributeValue: "Your username and password is invalid.");
    return "login";
}

@PostMapping("/login")
public String processLogin() { return "redirect:/index"; }
```

Login page with error parameter when giving wrong username and password as input will be as shown below.



The screenshot shows a web browser window with the address bar displaying "localhost/login?error". The page header for the "National Bank Of Ireland" includes "LOG IN" and "REGISTER" links. The main heading is "Please Sign In". Below this, there are input fields for "Username" and "Password". A red error message box states "Your username and password is invalid." Below the error message is a blue "Log In" button. At the bottom, there are links for "You don't have an account? Register" and "Forgot Password? click here". The footer contains the text "©Ashok Kumar K - ashokdefy@gmail.com".

Register page: Customers who would like to create a new bank account can register on the below available option in the login page. Registration page will allow us to get the inputs from the user and create a new account for them. We have Google captcha here to control the brute for of account creation. Below provided screenshot shows the registration page.

← → ↻ Not secure localhost/register

National Bank Of Ireland

LOG IN REGISTER

Registration

Username

First Name

Last Name

Select a Account Type
 Savings

E-mail

Mobile Number
 Select the Country code

Password

Confirm Password

☐ I'm not a robot

Register

©Ashok Kumar K - ashokdefy@gmail.com

All the user input is validated after the submission of the form. Following figure shows the controller code of the registration page.

```
@PostMapping("/register")
public String registration(@ModelAttribute("userForm") User userForm, BindingResult bindingResult,
    @RequestParam(name="g-recaptcha-response") String recaptchaResponse, HttpServletRequest request) {
    userValidator.validate(userForm, bindingResult);

    String ip = request.getRemoteAddr();
    String captchaVerifyMessage =
        recaptchaService.verifyRecaptcha(ip, recaptchaResponse);
    if(!captchaVerifyMessage.equals(""))
        bindingResult.rejectValue( field: "reCaptcha", errorCode: "user.reCaptcha.error", defaultMessage: "Recaptcha requires verification. ");

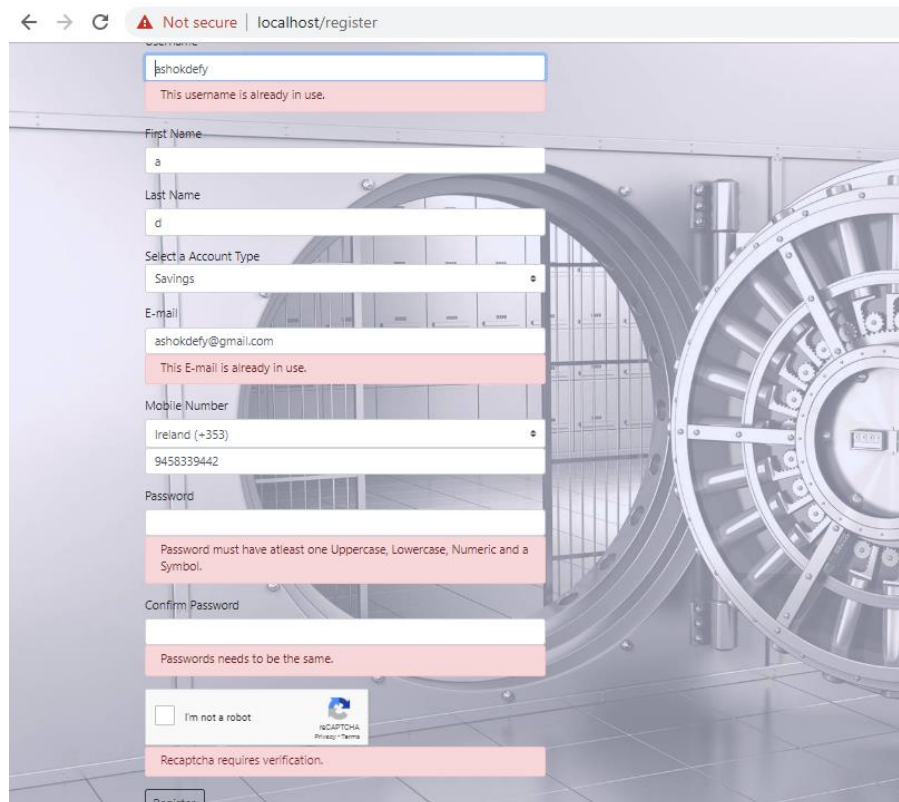
    if (bindingResult.hasErrors())
        return "register";
    userService.save(userForm);
    VerificationToken verificationToken = new VerificationToken(userForm);

    verificationTokenRepository.save(verificationToken);

    SimpleMailMessage mailMessage = new SimpleMailMessage();
    mailMessage.setTo(userForm.getEmail());
    mailMessage.setSubject("Complete Registration!");
    mailMessage.setFrom("noreply@nbirl.com");
    mailMessage.setText("To confirm your account, please click here : "
        + "http://localhost:8080/confirm-account?token="+verificationToken.getToken());
    javaMailSender.send(mailMessage);

    return "/verification/redirect-mail";
}
```

userValidator.validate method belongs to UserValidator class which is used to validate the user input if there is any error and add the errors if any found. You can check the code for the UserValidator class in the validator folder. Let us see the errors that comes when giving wrong information and existing username.



The screenshot shows a web browser window with the address bar displaying 'localhost/register'. The page has a background image of a vault door. The registration form contains the following fields and validation messages:

- Username:** 'ashokdefy'. Error: 'This username is already in use.'
- First Name:** 'a'
- Last Name:** 'd'
- Select an Account Type:** 'Savings'
- E-mail:** 'ashokdefy@gmail.com'. Error: 'This E-mail is already in use.'
- Mobile Number:** 'Ireland (+353)' and '9458339442'
- Password:** Error: 'Password must have atleast one Uppercase, Lowercase, Numeric and a Symbol.'
- Confirm Password:** Error: 'Passwords needs to be the same.'
- Recaptcha:** 'I'm not a robot' checkbox is unchecked. Error: 'Recaptcha requires verification.'

A 'Register' button is at the bottom of the form.

Consider the value that we gave are validated and no errors found then the UserService class save method is called with the user details sent to it.

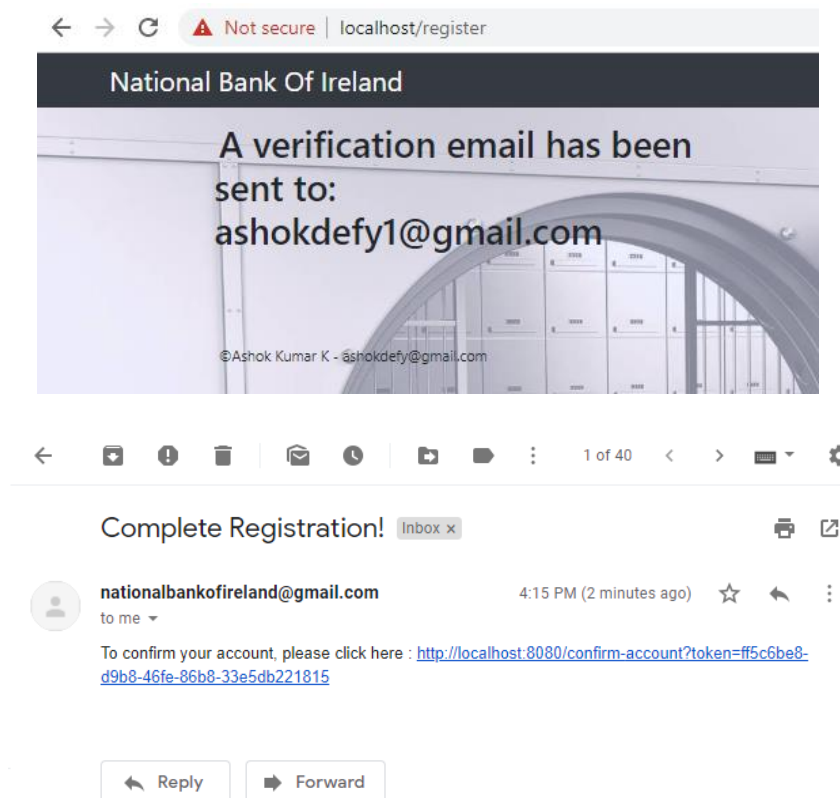
```
@Override
public void save(User user) {
    user.setUsername(user.getUsername().toLowerCase());
    user.setFirstName(user.getFirstName().substring(0, 1).toUpperCase() + user.getFirstName().substring(1).toLowerCase());
    user.setLastName(user.getLastName().substring(0, 1).toUpperCase() + user.getLastName().substring(1).toLowerCase());
    user.setPassword(passwordEncoder.encode(user.getPassword()));
    user.setEmail(user.getEmail().toLowerCase());
    user.setEnabled(false);

    Role role = roleService.findByName("ROLE_CUSTOMER");
    Set<Role> roles = new HashSet<>();
    roles.add(role);
    user.setRoles(roles);
    user.setMobileNumber("+" + user.getCountrycode() + user.getMobileNumber());
    Account account = new Account();
    account.setAccountNumber("NBI" + randomNumGenerator() + randomNumGenerator());
    user.setAccountType(user.getAccountType());
    account.setBalance(0.0);
    account.setBic("BIC" + randomNumGenerator());
    account.setIban("IE" + randomNumGenerator() + "NBI" + randomNumGenerator() + randomNumGenerator());
    account.setNsc(randomNumGenerator() + "");
    accountRepository.save(account);
    user.setAccount(account);
    userRepository.save(user);
}

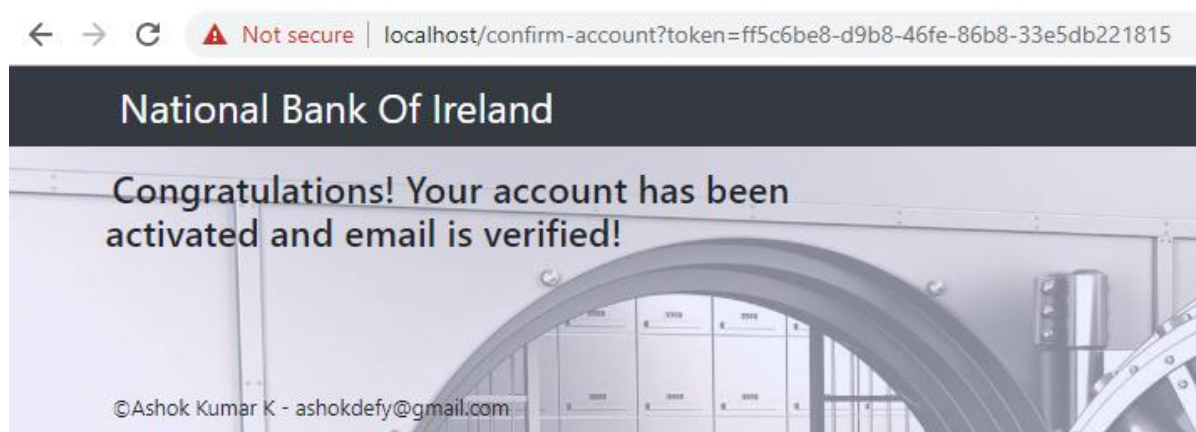
private int randomNumGenerator() {
    Random random = new Random();
    return random.nextInt( bound: 9999);
}
```


In this method all the values are formatted. Username is converted to lowercase to avoid case sensitive account creation. Password is encoded with the BCryptPasswordEncoder which is inbuilt in the spring boot. Role Customer is added to the role. Some random account number and other details are generated. Account is disabled when the account is created. Finally, the user and account data are saved in the database.

Followed by a verification token is created and user is saved in it. Verification mail is sent to the user Email and the page is forward to 'redirect-mail.html' as shown below.



On clicking the link, the user account will be activated with the verification of the token value in the controller.



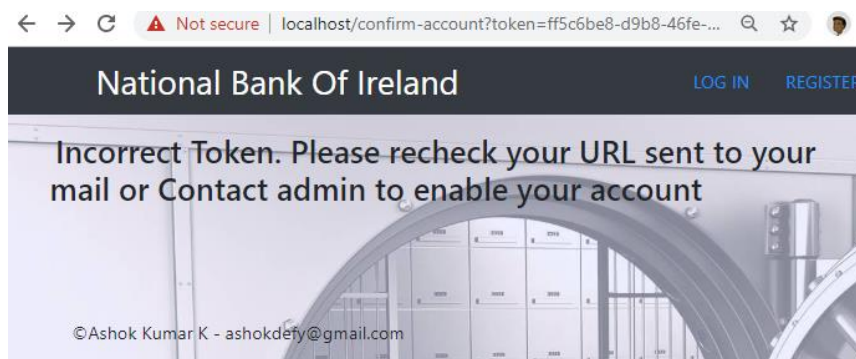
Let's check on the control of the 'confirm-account' page, how it works. The token is sent in the parameter of the header.

```
@RequestMapping(value="/confirm-account")
public String confirmUserAccount(@RequestParam( value = "token" , required = false)String token
{
    VerificationToken verificationToken = verificationTokenRepository.findByToken(token);

    if(verificationToken == null)
        return "error/verification-error";

    User user = userRepository.findByEmail(verificationToken.getUser().getEmail());
    user.setEnabled(true);
    verificationTokenRepository.delete(verificationToken);
    userRepository.save(user);
    return "/verification/account-verified";
}
```

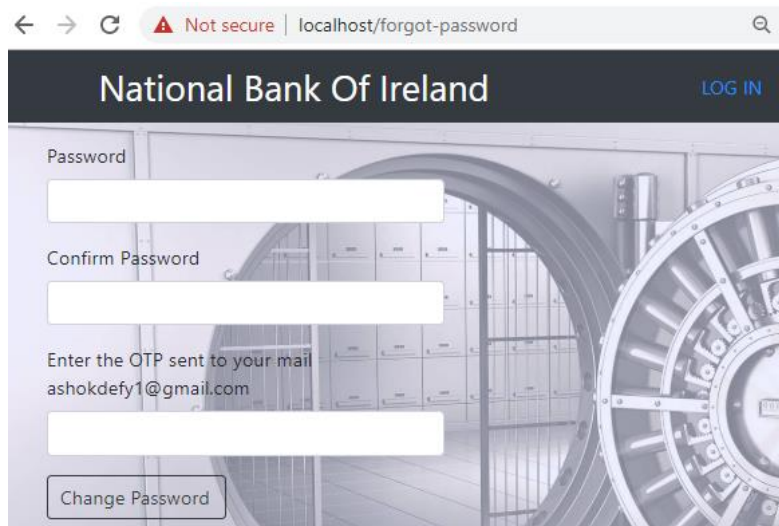
If the token sent is null or there is no token available in the database, then the verification-error page will be redirected as shown below.



Forgot Password Page: If the user forgot the password and would like to reset the password then the user can proceed with this page. To change the password user must give username and email address as input. For security reasons the username and email are verified, and the One-time password will be sent to the email of the user.



Once the username and email match OTP will be sent to the user email and redirected to the below page. OTP is confirmed to change the password.



The screenshot shows a web browser window with the address bar displaying 'localhost/forgot-password'. The page title is 'National Bank Of Ireland' with a 'LOG IN' link. The form contains three input fields: 'Password', 'Confirm Password', and 'Enter the OTP sent to your mail ashokdefy1@gmail.com'. A 'Change Password' button is at the bottom.

Reset Password!



nationalbankofireland@gmail.com
to me ▾

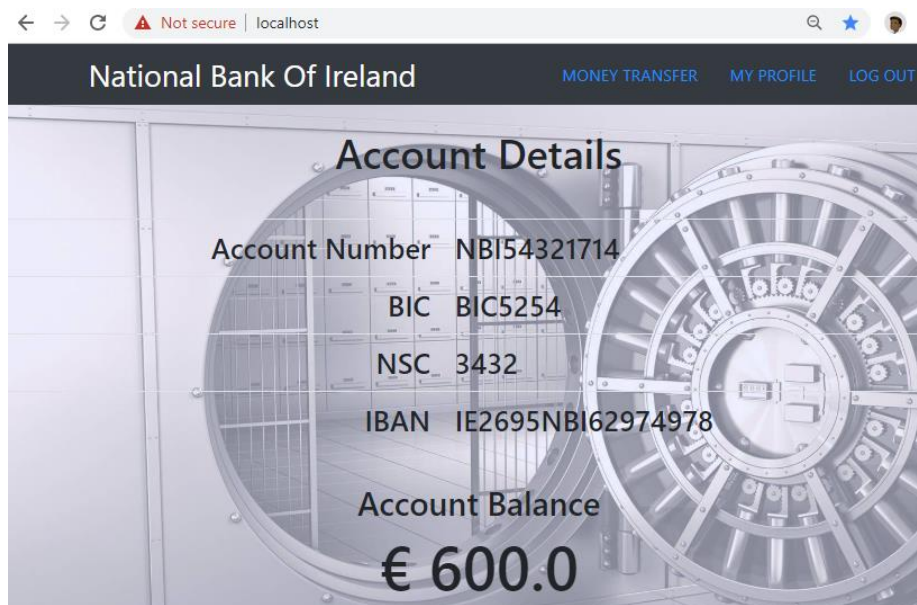
OTP to reset your National Bank of Ireland Account is : 57096390

↩ Reply

➡ Forward

Index page: Once the user is logged in, he will be re-directed to the index or the homepage. Index page of the user depends on the user role.

Customer Role: Customer will be displayed their account details with their account balance.



The screenshot shows a web browser window with the address bar displaying 'localhost'. The page title is 'National Bank Of Ireland' with links for 'MONEY TRANSFER', 'MY PROFILE', and 'LOG OUT'. The main content is titled 'Account Details' and displays the following information:

Account Number	NBI54321714
BIC	BIC5254
NSC	3432
IBAN	IE2695NBI62974978
Account Balance	€ 600.0

Employee Role: Employee will be displayed with the deposit fields on behalf of customer accounts.

← → ↻ ⚠ Not secure | localhost

National Bank Of Ireland

[SEARCH CUSTOMER](#) [MY PROFILE](#) [LOG OUT](#)

DEPOSIT AMOUNT

Account Number

Confirm Account Number

Amount to Deposit

Though this is an unsafe action we are logging all the deposits made by the employee to the customer. When the employer gives the correct account and the deposit amount, the employee will be asked for the confirmation before with the username for verification. Once he confirms the amount will be deposited to the use

← → ↻ ⚠ Not secure | localhost/deposit

National Bank Of Ireland

[SEARCH CUSTOMER](#) [MY PROFILE](#)

Do you want to deposit
€100.0 to the Account
Number NBI37741968 for
the user Dinesh Kumar

```
2019-12-15 19:41:56.153 INFO 15292 --- [-nio-443-exec-6] c.n.bank.controller.EmployeeController : employee is trying to deposited $100.0 in the accountNBI37741968
```

← → ↻ ⚠ Not secure | localhost/confirm-deposit

National Bank Of Ireland

[SEARCH CUSTOMER](#) [MY PROFILE](#) [LOG OUT](#)

Account Holder Details

Name	Dinesh Kumar
Email	dinesh@gmail.com
Phone Number	+353899419953
Account Type	SAVINGS

Account Details

Account Number	NBI37741968
BIC	BIC586
NSC	9596
IBAN	IF6705NBI5674615

```
2019-12-15 19:41:56.153 INFO 15292 --- [-nio-443-exec-6] c.n.bank.controller.EmployeeController : employee is trying to deposited $100.0 in the accountNBI37741968
2019-12-15 19:43:41.852 INFO 15292 --- [-nio-443-exec-1] c.n.bank.controller.EmployeeController : employee has deposited the amount to the user
```

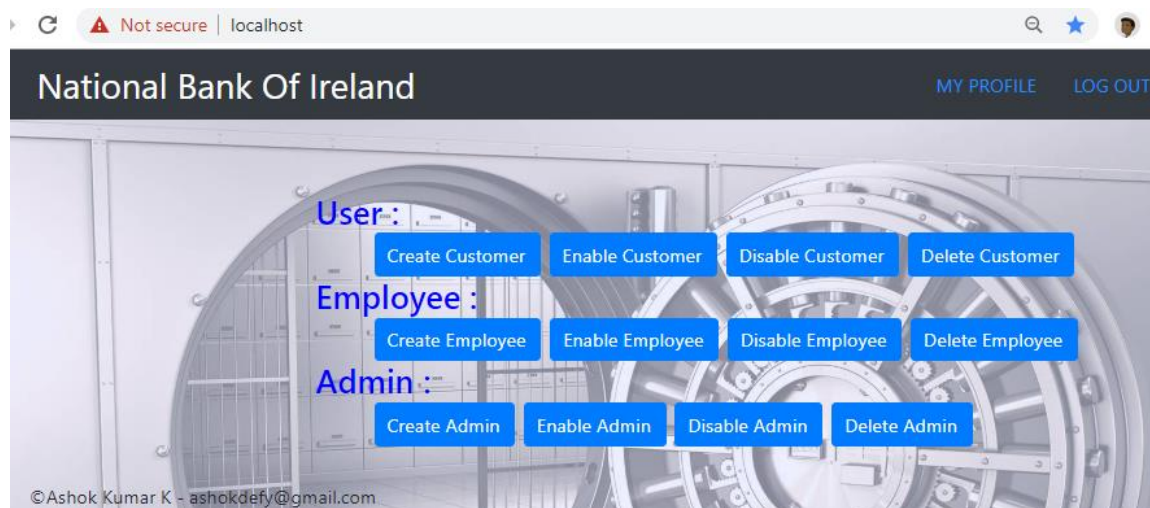

Once the customer confirms the deposit their will be a log on the deposit. As in the above screenshot the name of the employee is “employee”, so it shows employee has deposited the amount to the user. The below represent the logging code of the deposit.

```
logger.info( principal.getName() + " is trying to deposited $" + account.getTransfer() + " in the account" + account.getAccountNumber());
```

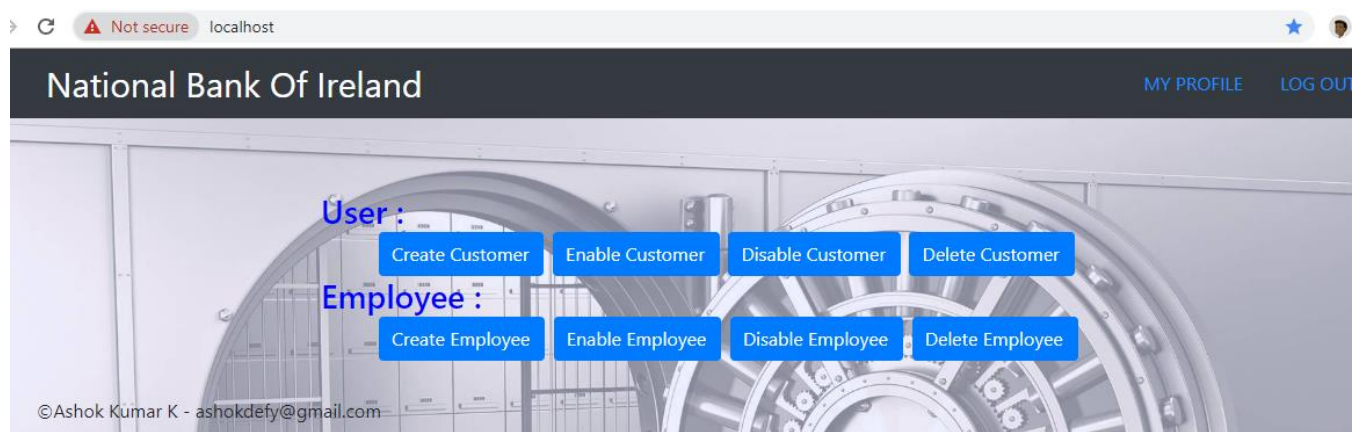
Also, in spring security we can specify the IP address of the employee where they can use this deposit page. In the below screenshot I have commented the line for the testing purpose this can be enabled when needed.

```
.antMatchers( ...antPatterns: "/money-transfer").hasRole("CUSTOMER")  
//.antMatchers("/confirm-deposit").hasIpAddress("11.11.11.11")  
.antMatchers( ...antPatterns: "/search-customer", "/deposit", "/confirm-d  
.antMatchers( ...antPatterns: "/admin/**").hasRole("ADMIN")
```

Admin Role: Admin home page will display the permission he has to perform an action on all the user account, as shown below.



Not all the admin can see or use these actions. Below is the screenshot of the admin1, who don't have permission in creating or deleting ADMIN accounts.



This is achieved with the help of spring security method. The below screenshot, a part of code from Admin Controller, In the line 58 I will be checking the authorities of the user (here admin) where he has any Permission to CREATE_(\$Type\$--> type of the user he is going to create which will be sent in parameter). If he is not permitted, then he will be redirected to the error 403 page (Access Denied page). If the type name doesn't match with the user type, then the page will be redirected to the error 404 page (Page not found Status).

```
51 @GetMapping("/create")
52 public String Create(@RequestParam("type") String type, Model model, Authentication authentication) {
53     model.addAttribute( attributeName: "user", new User());
54     model.addAttribute( attributeName: "type", type.toUpperCase());
55     if(!(type.equals("admin")||type.equals("employee")||type.equals("customer")))
56         return "error/404";
57
58     if(!authentication.getAuthorities().contains(roleRepository.findByRoleName("CREATE_"+type.toUpperCase())));
59         return "error/403";
60
61     if(type.equals("admin"))
62         model.addAttribute( attributeName: "permission",authority);
63     return "admin/create";
64 }
```

I have implied this concept in all the 4 different operations based for both the GET and POST methods, refer code ADMIN CONTROLLER for it.

My Profile Page: This page will display the user details like Name, Mobile Number, Email ID, Account Type. This is a common page to all the users and accessed by everyone.



Update Profile page: From the My profile user can update the user details by clicking on “Click here to update”. Once the values are updated, they can save the changes only if the given inputs are validated properly.

← → ↻ ⚠ Not secure | localhost/profile/update 🔍 ☆ 👤

National Bank Of Ireland

[MONEY TRANSFER](#) [MY PROFILE](#) [LOG OUT](#)

Edit Profile

First Name

Last Name

Email

Phone Number

[Back to My Profile](#) [Update](#)

©Ashok Kumar K - ashokdefy@gmail.com

Change Password Page: This is a common page which allows all the users to update new password. Here the user will be giving the old password to change the new one. The old password of the user will be verified and if true then allows to change the new password.

→ ↻ ⚠ Not secure | localhost/profile/password 🔍 ☆

National Bank Of Ireland

[MONEY TRANSFER](#) [MY PROFILE](#) [LOG OUT](#)

Change Password

Enter Your Old Password

Enter Your New Password

Enter Your New Password Confirm

[Back to My Profile](#) [Confirm](#)

Money Transfer Page: This is a customer page where other roles are not allowed to visit. Here the customer must give the correct details of the receiver i.e. the Account number, BIC, NSC and IBAN. If all the above matches, then the customer can transfer money from his account to his account. For security reasons the user must confirm his password before transferring the money to other customer.

National Bank Of Ireland MONEY TRANSFER MY PROFILE LOG OUT

Money Transfer

Account Number	<input type="text"/>
BIC	<input type="text"/>
NSC	<input type="text"/>
IBAN	<input type="text"/>
Amount	<input type="text" value="0.0"/>
Remaining Balance	600.0
Confirm your password	<input type="password"/>

Search Customer: This is an employee page, where an employee can search for any customer with the account number.

National Bank Of Ireland SEARCH CUSTOMER MY PROFILE LOG OUT

Click to go back, hold to see history

Account Holder Details

Name	Shreyas Kp
Email	shreyasprince17@gmail.com
Phone Number	+3530894960035
Account Type	SAVINGS

Account Details

Account Number	NBI42355316
BIC	BIC2414
NSC	5156
IBAN	IE7162NBI99436604
Account Balance	5000.0

SECURITY IMPLEMENTATION

Now let us discuss on the securities that we have implemented in the whole project. We will be considering OWASP top 10 cyber threats or risks released in year 2019. Let us start with the SecurityConfig.java class. This is the implementation of spring security configuration. As we already discussed this class will override three methods and we will be discussing them now in detail.

configure(AuthenticationManagerBuilder auth)

This class is used to initialize the authentication manager that is the login details that spring needs to check the authentication of user before accessing home or any authenticated pages.

```
@Autowired
@Qualifier("customUserDetailsService")
private UserDetailsService userDetailsService;

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService)
        .passwordEncoder(passwordEncoder());
}
```

We will be defining the 'userDetailsService' with the default 'UserDetailsService' but we will override one method of the 'UserDetailsService' class using 'CustomUserDetailsService' class. Below is a piece of code from the 'CustomUserDetailsService' where we will check for the user details before authenticating. We will add the username and roles to the class. Before this we will be checking if the user is blocked by IP or not. We will discuss on this further

```
@Override
@Transactional(readOnly = true)
public UserDetails loadUserByUsername(String username) {
    String ip = request.getRemoteAddr();
    if (loginAttemptService.isBlocked(ip)) {
        logger.error("User "+username+" has been blocked for brute forcing");
        throw new UsernameNotFoundException(String.format("User with username=%s has been blocked ", username));
    }
    User user = userRepository.findByUsername(username);
    if (user != null) {
        Set<GrantedAuthority> authorities = new HashSet<>();
        Iterator iterator = user.getRoles().iterator();
        while (iterator.hasNext())
            authorities.add((GrantedAuthority) iterator.next());
        return new org.springframework.security.core.userdetails.User(user.getUsername(),
            user.getPassword(), user.isEnabled(), accountNonExpired: true, credentialsNonExpired: true, accountNonLocked: true, authorities);
    }
    else
        throw new UsernameNotFoundException(String.format("User with username=%s was not found", username));
}
```

Brute Force Defense

In our application we have implemented defense for controlling the brute force attack. We will use 'LoginAttemptService' class to define the model attributes for controlling the brute force. If the user login attempts cross 10 times, then the IP of the user will be blocked for a day without any notice to the user.

```

@Service
public class LoginAttemptService {

    private final int MAX_ATTEMPT = 10;
    private LoadingCache<String, Integer> attemptsCache;

    public LoginAttemptService() {
        super();
        attemptsCache = CacheBuilder.newBuilder().
            expireAfterWrite( duration: 1, TimeUnit.DAYS).build((CacheLoader) (key) -> { return 0; });
    }

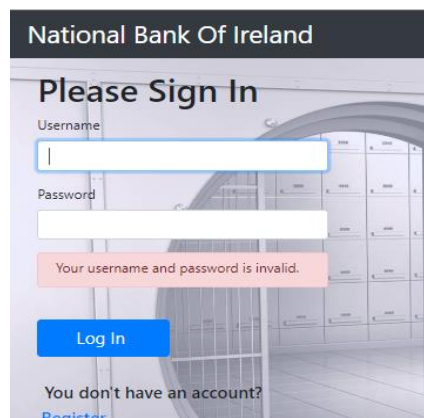
    public void loginSucceeded(String key) { attemptsCache.invalidate(key); }

    public void loginFailed(String key) {
        int attempts = 0;
        try {
            attempts = attemptsCache.get(key);
        } catch (ExecutionException e) {
            attempts = 0;
        }
        attempts++;
        attemptsCache.put(key, attempts);
    }

    public boolean isBlocked(String key) {
        try {
            return attemptsCache.get(key) >= MAX_ATTEMPT;
        } catch (ExecutionException e) {
            return false;
        }
    }
}

```

We are not intimating the user about their IP address is blocked for 1 day. So, if the user tries to login for more than 10 times though giving the correct password they won't be allowed and will be returned to login page with 'incorrect login details message'.



Every attempt they try to login after their IP blocked will be logged for brute forcing. This is achieved using 'AuthenticationListeners' for every attempt. This can be found in the authorizer folder.

Classes: 'AuthenticationFailureListener' to add failure event count. 'AuthenticationSuccessListener' to destroy the event count.

```

2019-12-16 10:56:44.052 ERROR 14952 --- [-nio-443-exec-9] c.n.b.s.auth.CustomUserDetailsService : User admin has been blocked for brute forcing
2019-12-16 10:56:45.639 ERROR 14952 --- [-nio-443-exec-1] c.n.b.s.auth.CustomUserDetailsService : User admin has been blocked for brute forcing

```

Password Encoder

We use BCryptPasswordEncoder for hashing all the password used. This hashes the password with different salts each time. And we can easily compare the password with the 'matches' method.

```
@Bean
public BCryptPasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Below is the screenshot from TransferController where we will be comparing the password of the user with the database and input we got from the user.

```
if(!bCryptPasswordEncoder.matches(password, user.getPassword()))
```

configure(HttpSecurity http)

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.requiresChannel().anyRequest().requiresSecure().and() HttpSecurity
        .authorizeRequests().antMatchers("/register", "/confirm-account", "/forgot-password", "/reset-password").permitAll() ExpressionUrlAuthorization
        .antMatchers("/money-transfer").hasRole("CUSTOMER")
        // .antMatchers("/confirm-deposit").hasIpAddress("11.11.11.11")
        .antMatchers("/search-customer", "/deposit", "/confirm-deposit").hasRole("EMPLOYEE")
        .antMatchers("/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated()
        .and().formLogin().loginPage("/login").defaultSuccessUrl("/").permitAll() FormLoginConfigurer<HttpSecurity>
        .and().logout().invalidateHttpSession(true).clearAuthentication(true).logoutRequestMatcher(new AntPathRequestMatcher(pattern: "/logout")).permitAll()
        .and().headers().cacheControl() HeadersConfigurer<H>.CacheControlConfig
        .and().xssProtection() HeadersConfigurer<H>.XXssConfig
        .and().frameOptions().sameOrigin() HeadersConfigurer<HttpSecurity>
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED).sessionFixation().migrateSession() SessionManagementConfigurer<HttpSecurity>
        .and().csrf().csrfTokenRepository(cookieCsrfTokenRepository);
}
```

In the HTTP Security we secure all the pages based on the authentication and we will imply the inbuilt securities available in spring boot.

`requiresChannel().anyRequest().requiresSecure()` – This allows all the request to go through https and secures the connection. We are using SSL to have secure connection in HTTPS. We are running the server in port 443 and using the JKS - SSL Certificate and added the certificate files.

```
server.port=443
server.ssl.enabled=true
server.ssl.key-alias=tomcat
server.ssl.key-store-password=password
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-provider=SUN
server.ssl.key-store-type=JKS
```

`.authorizeRequests().antMatchers(" ").permitAll()` – The requests given inside 'AntMatchers' is allowed by everyone without authentication. So, we are allowing the following requested permitted to all the users.

register, confirm-account, forgot-password, reset-password

`.anyRequest().authenticated()` – This allows the other request from the user only if the user is authenticated.

`.antMatchers("").hasRole("CUSTOMER")` – This helps in verifying the role of the user before approving the request of the page. For example, to money-transfer page the user must have the CUSTOMER role to perform the action. In other words, only Customer can visit these pages. Admin or Employee will be moved to ERROR 403 page that is “Access Denied”.

`.antMatchers("").hasIpAddress("192.168.1.1")` – This helps in verifying the IP address of the user before approving the request of the page. We have not implemented here. In real world scenario we can mention the IP address of the computer which can perform this action. We will be using this when the employee needs to deposit amount to the user account. From this we can allow the deposit of money from other places will be prevented.

`.and().formLogin().loginPage("/login").defaultSuccessUrl("/").permitAll()` – to intimate the login page to the spring security we will use loginPage method. DefaultSuccessURL will be used to redirect once the user is success fully authenticated.

`.and().logout().invalidateHttpSession(true).clearAuthentication(true).logoutRequestMatcher(new AntPathRequestMatcher("/logout")).permitAll()` – On log out all the session and authentication are cleared thus the user will be redirected to the login page as he is not authorized for authentication pages.

`.and().headers().cacheControl()` – This allows to control the cache. On implementing this the request sent by browser will have max-age to '0' for the request and the response received will have cache control: private. This indicates that the resource is user specific and it can still be cached only on a client browser.

▼ Request Headers [view source](#)

```
Accept: text/html,application/xhtml+xml,application/javascript;
font-awesome/5.0.10/web-fonts-with-css/css/fontawesome
exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,fr;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
Cookie: Idea-3b730081=4569eb32-7d9d-475a-a6b0-5
```

▼ Response Headers [view source](#)

```
Cache-Control: private
Content-Language: en-US
Content-Type: text/html; charset=UTF-8
Date: Mon, 16 Dec 2019 16:00:37 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
```

`.and().xssProtection()` – This allows the user from the reflected XSS attacks by the attackers. By default, the XSS protection in the current version of the browser is enabled. In case if the user has disabled the XSS protection in his browser, this security will enable for each response received.

▼ Response Headers [view parsed](#)

```
HTTP/1.1 200
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=3153
```

`.and().frameOptions().sameOrigin()` – This allows in the protection of ClickJacking attacks by setting the XFrame option to 'Same Origin'. Thus, the IFRAME is not allowed by the other site and only allowed in the sites of same origin.

```
Domains
X-Frame-Options: SAMEORIGIN
Content-Type: text/html;charset=UTF-8
```

`.and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED).sessionFixation().migrateSession()` – New session is created if it is required. Session fixation is managed by the spring security with the sessionFixation method being enabled.

We also secure the session with the session properties applied in application.properties file. Session timeout is set to 5m to make the application more secure as this is a banking application and it needs more security. HTTP-ONLY and SECURE are enabled for security reasons. HTTPONLY: browser script won't be able to access the cookie when enabled. SECURE : Cookie will be sent only over HTTPS connection when enabled.

```
server.servlet.session.timeout=5m
server.servlet.session.cookie.http-only=true
server.servlet.session.cookie.secure=true
```

ConnectorConfig.Java

This class is used to redirect all the request from the HTTP 80 port to HTTPS 443 port and sends the data only in the secured connection.

```
@Configuration
public class ConnectorConfig {

    @Bean
    public ServletWebServerFactory servletContainer() {
        TomcatServletWebServerFactory tomcat = postProcessContext(context) -> {
            SecurityConstraint securityConstraint = new SecurityConstraint();
            securityConstraint.setUserConstraint("CONFIDENTIAL");
            SecurityCollection collection = new SecurityCollection();
            collection.addPattern("/");
            securityConstraint.addCollection(collection);
            context.addConstraint(securityConstraint);
        };
        tomcat.addAdditionalTomcatConnectors(redirectConnector());
        return tomcat;
    }

    private Connector redirectConnector() {
        Connector connector = new Connector("org.apache.coyote.http11.Http11NioProtocol");
        connector.setScheme("http");
        connector.setPort(80);
        connector.setSecure(false);
        connector.setRedirectPort(443);
        return connector;
    }
}
```

`csrf().csrfTokenRepository(cookieCsrfTokenRepository)` – This security will automatically send the CSRF token to control the cross-site request forgery by adding a token called CSRF in the form field if available.

```
method="POST">
<input type="hidden" name="_csrf" value="db056dd8-92d4-44ec-b6f5-
3e411a149579"> == $0
<table class="table table-hover">
```

In case some pages may not have form field so, it also sends the token via Cookies.

```
Connection: keep-alive
Cookie: Idea-3b730081=4569eb32-7d9d-475a-a6b0-565059ad438
2; XSRF-TOKEN=db056dd8-92d4-44ec-b6f5-3e411a149579; JSESSIONID=84B46BA176BC1E6C3C1933E9A3E05DA6
Host: localhost
```

OWASP TOP 10 SECURITY RISKS-2019

1. SQL Injection

SQL injection flaws occurs when untrusted data sent to the server as a part of field value. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization [4].

In spring boot, we are using JPA(Java Persistence API) which is an ORM framework instead of JDBC prepared statements. And for the safer side we are not using native queries to pull values from the data. Thus, we are safer from the injection attacks.

2. Broken Authentication and Session Management

Broken authentication vulnerability allows an attacker to gain control over any account he doesn't have authorization. This commonly done by session ID/Cookie high jacking methods. To control this, we have few securities in our application.

Session Fixation Protection.

```
http.sessionManagement().sessionFixation().migrateSession()
```

Cookies of the Session ID and CSRF are sent in secure and HTTP-only state. This encrypts the cookies and doesn't allow the browser to access it.

To control the brute force attacks, we have implemented Google captcha to the REGISTER page and IP address blockage for the LOGIN page.

All the password is validated to have at least one Capital Letter, one Small Letter, one Numeric, & one Symbol to be added. All the passwords are hashed with BCrypt password encoder technology before saving to the database.

3. Cross-Site Scripting (XSS)

Spring framework by default sanitizes the input from the user and we are enabling the XSS protection in the Security Configuration. This allows to allow XSS protection even though the user disabled it in the browser.

4. Broken Access Control

All the pages that need authentication disallow the user without authentication and automatically redirects the user to the Login page. Pages based on role and permissions will be allowed only for the specific roles and the users who doesn't have role will be redirected to the "Access Denied Page".

5. Security Misconfiguration

We use the recent updated version of Java, TOMCAT Web Server and other Spring framework. And the securities required are been configured correctly. All the unnecessary services are removed. The whole concept of spring security is well understood before using it.

6. Sensitive Data Exposure

No unwanted data is exposed to the user, other than what is required. Every secured action like money transfer and other stuff requires the user password. Also, the passwords are hashed and stored in database so. Other than user no one knows it. All data is transmitted in with the TSL Layer with my own SSL certificate. Customer role can only view his own information and not others information. Employee role can view all the customer information but no other roles information. Admin can create or delete all the roles and it differs based on the permission.

TESTING

Manual Testing

As most of the security is tested in the design phase itself, now let us do some more core testing in this phase.

Brute force is not allowed in Login page. As we know that we have IP blocked after 10 inputs.



```

2019-12-17 17:03:29.922 ERROR 18528 --- [-nio-443-exec-2] c.n.b.s.auth.CustomUserDetailsService : User admin has been blocked for brute forcing
2019-12-17 17:03:43.773 ERROR 18528 --- [-nio-443-exec-8] c.n.b.s.auth.CustomUserDetailsService : User employee has been blocked for brute forcing
2019-12-17 17:03:52.290 ERROR 18528 --- [-nio-443-exec-10] c.n.b.s.auth.CustomUserDetailsService : User admin1 has been blocked for brute forcing

```

Registration page: All the important data are verified before registering an account.

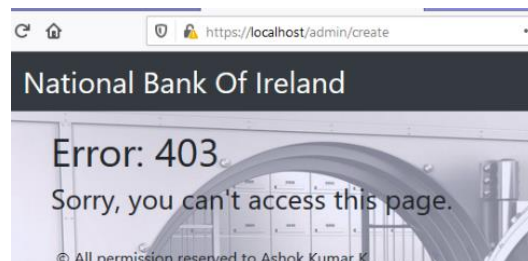
The registration page form includes fields for Username, Password, and Confirm Password. The Username field has a validation error: "Username must have at least 6 characters. Username should be alphanumeric." The Password field has a validation error: "Password must have atleast one Uppercase, Lowercase, Numeric and a Symbol." The Confirm Password field has a validation error: "Passwords needs to be the same." There is also a reCAPTCHA checkbox with the text "I'm not a robot" and a message: "Recaptcha requires verification."

Forgot password page: Reset OTP will be sent only if the username and email is matching to the user in the database

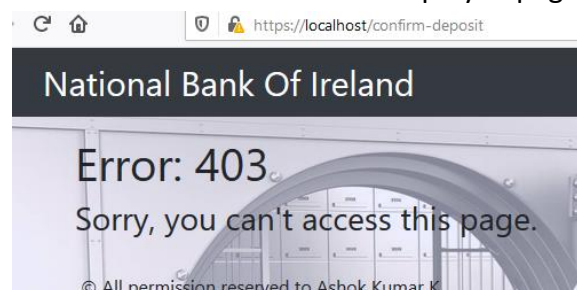
The forgot password page form includes fields for Username and E-mail. The Username field contains "admin" and the E-mail field contains "admin@gmail.com". A validation error message is displayed: "User Email doesn't match with username". There is a "Confirm" button and a link to "Register" with the text "You don't have an account?".

Accessing page without authorized role:

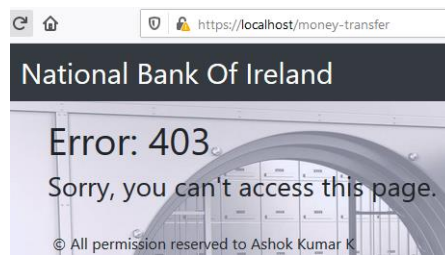
When Customer or Employee tries to access admin pages:



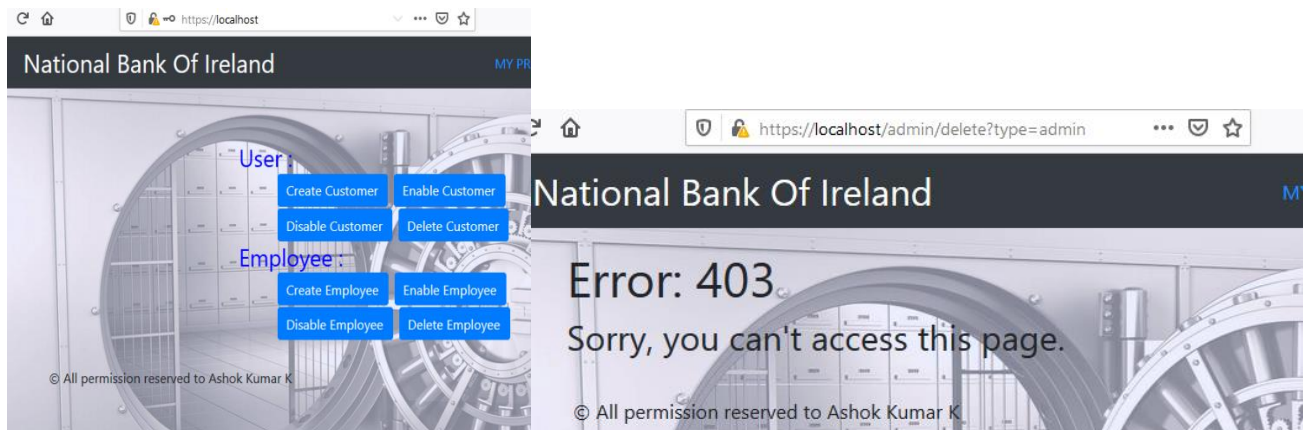
When Admin or Customer tries to access employee pages:



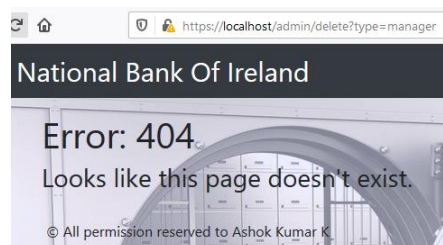
When Employee or Admin tries accessing Customer pages:



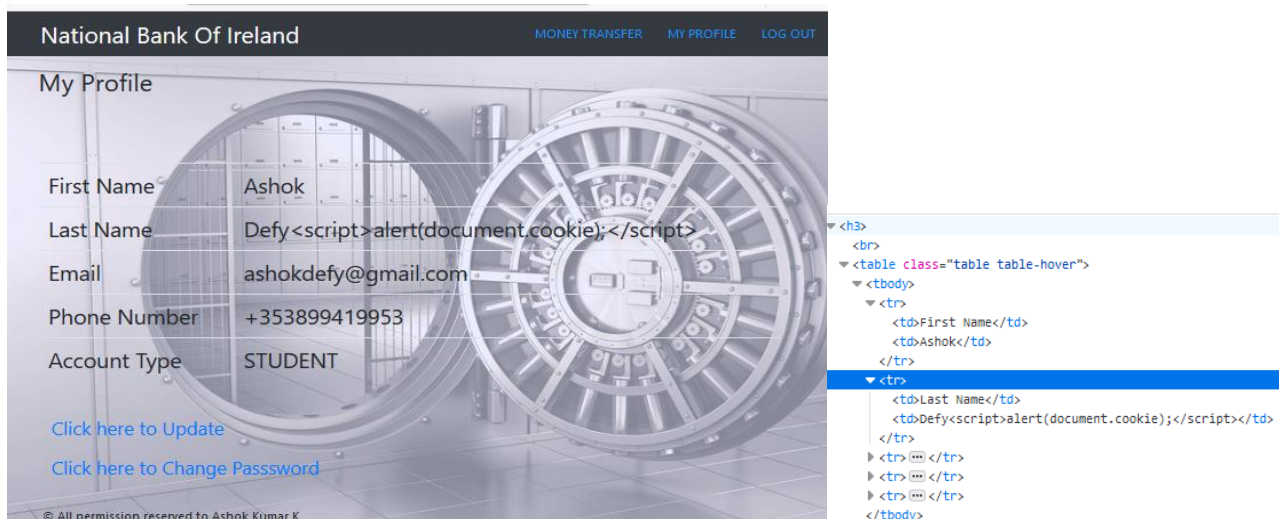
When admin who doesn't have permission to delete new admin tries to access the page.



Try injecting some other type in the admin URL when creating new account, type other than admin, customer, employee.



Trying Stored XSS attack by changing the first name to a script user input is sanitized to string and not executed. Screenshot of source code is shown below.



Though giving the correct account number customer must give the correct other account details and password to perform the transaction.

National Bank Of Ireland

MONEY TRANSFER MY PROFILE LOG OUT

Money Transfer

Account Number NBI28492914

BIC ds
BIC doesn't match with account number.

NSC d
NSC doesn't match with account number

IBAN fg
IBAN doesn't match with account number.

Amount -500
Enter a valid amount to transfer.

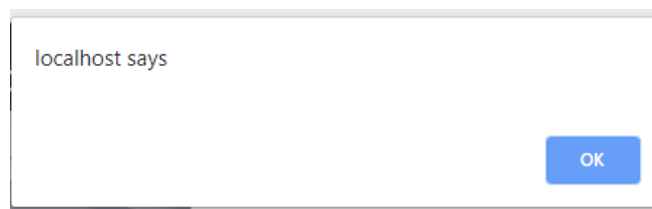
Remaining Balance 600.0

Confirm your password
Enter the correct password

Transfer

Though we have cookies browser can't able to access the cookies used.

```
Cache-Control: max-age=0
Connection: keep-alive
Cookie: Idea-3b730081=4569eb32-7d9d-475a-a6b0-565059ad4382; ui_session=2986ad8c0a5b3df4d7028d5f3c06e936c5e0377be26dd1ab91af249959a9fa6d38b313b646dc161a3e3a9bcb7eaadc7fd9ccac0e3f259955f27e5cc92072e78; XSRF-TOKEN=f3d00e11-26d4-46e5-a050-7967f209fe95; JSESSIONID=510479F3573A7586CCD29AE4638AD23B
Host: localhost
```

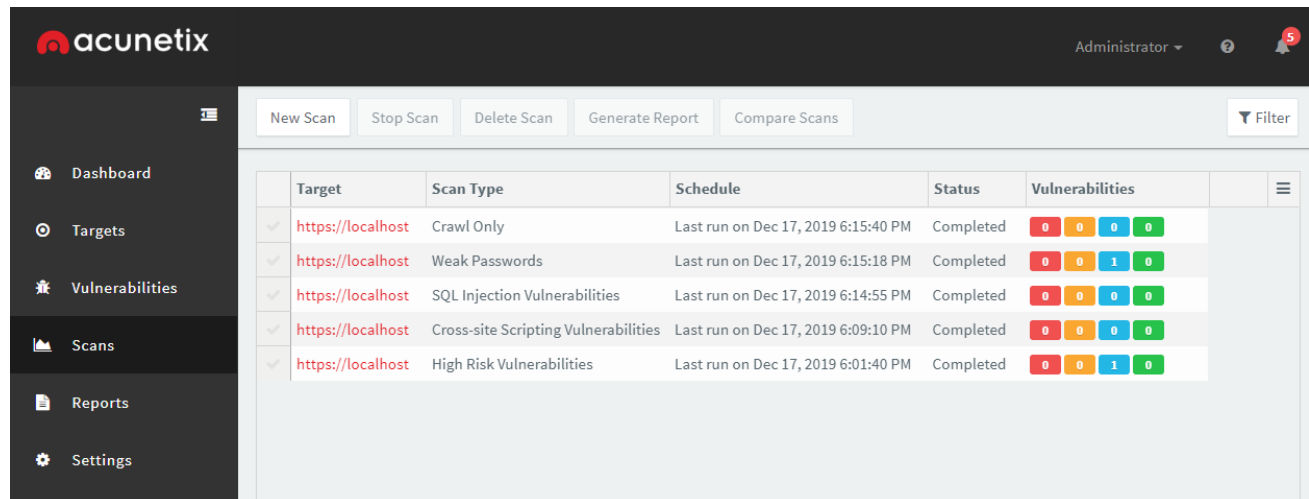


```
> alert(document.cookie);
< undefined
> document.cookie;
< ""
> document.URL
< "https://localhost/login"
> |
```

Automation Testing:

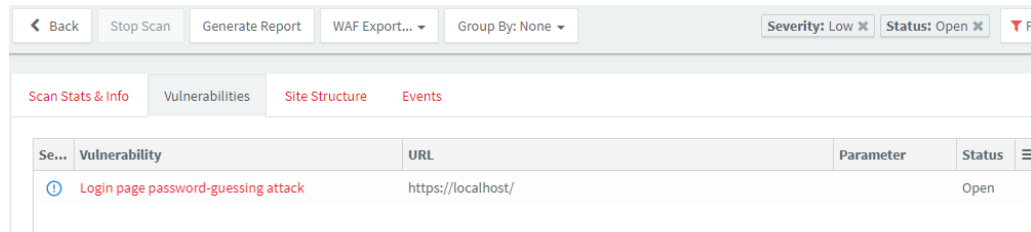
We are using Acunetix, automation vulnerabilities scanner for scanning the vulnerabilities in our web application. We tried 5 different types of scanning types as follows.

Vulnerabilities : RED: Severe, YELLOW: Medium, Blue: Low, Green: Information



	Target	Scan Type	Schedule	Status	Vulnerabilities
✓	https://localhost	Crawl Only	Last run on Dec 17, 2019 6:15:40 PM	Completed	0 0 0 0
✓	https://localhost	Weak Passwords	Last run on Dec 17, 2019 6:15:18 PM	Completed	0 0 1 0
✓	https://localhost	SQL Injection Vulnerabilities	Last run on Dec 17, 2019 6:14:55 PM	Completed	0 0 0 0
✓	https://localhost	Cross-site Scripting Vulnerabilities	Last run on Dec 17, 2019 6:09:10 PM	Completed	0 0 0 0
✓	https://localhost	High Risk Vulnerabilities	Last run on Dec 17, 2019 6:01:40 PM	Completed	0 0 1 0

We found only 2 LOW vulnerabilities. One in Week Passwords, another one High Risk Vulnerabilities. On checking both the vulnerabilities were same.



Se...	Vulnerability	URL	Parameter	Status
①	Login page password-guessing attack	https://localhost/		Open

Login page password-guessing attack

Low Open

Vulnerability description

A common threat web developers face is a password-guessing attack known as a brute force attack. A brute-force attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols until you discover the one correct combination that works.

This login page doesn't have any protection against password-guessing attacks (brute force attacks). It's recommended to implement some type of account lockout after a defined number of incorrect password attempts. Consult Web references for more information about fixing this problem.

The vulnerability affects https://localhost/

Discovered by Login page password-guessing attack

Attack details

Not available in the free trial

HTTP request

The impact of this vulnerability

An attacker may attempt to discover a weak password by systematically trying every possible combination of letters, numbers, and symbols until it discovers the one correct combination that works.

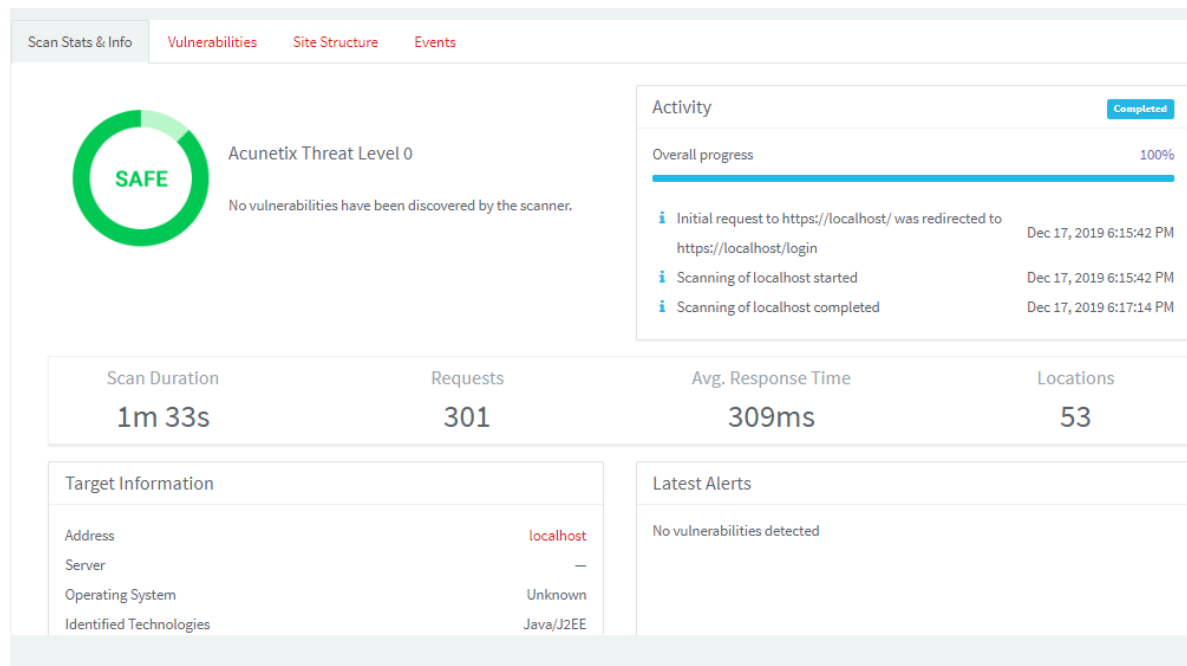
How to fix this vulnerability

It's recommended to implement some type of account lockout after a defined number of incorrect password attempts.

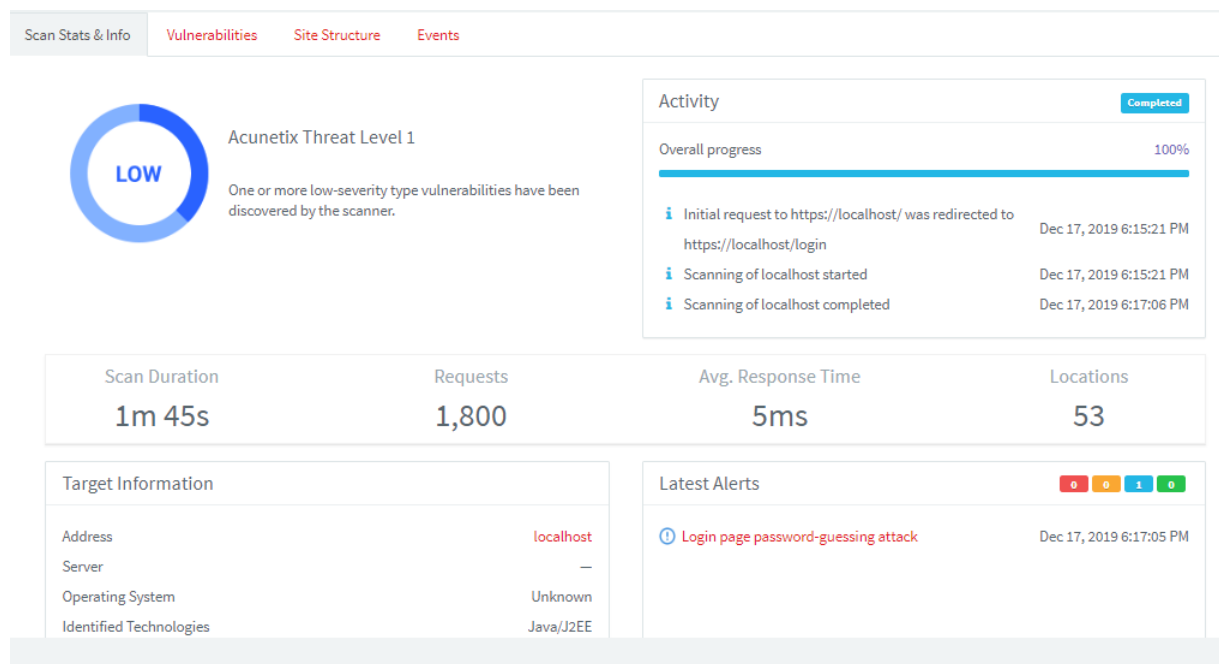
Classification

It reported no specific protection against brute force was implemented. This is because we are not allowing the user to know that the user IP was blocked to access the page. So I hope this is not a big issue to deal with as we already implemented the brute force protection by blocking the IP address.

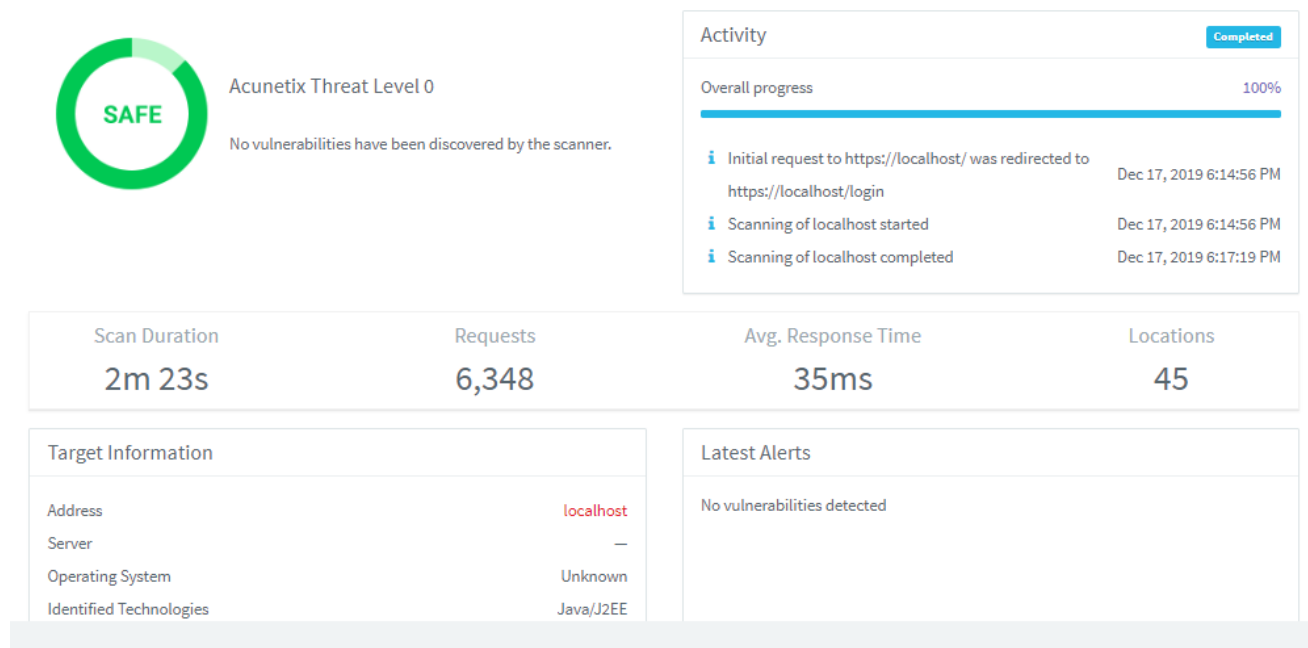
Scan Report of Crawl Only:



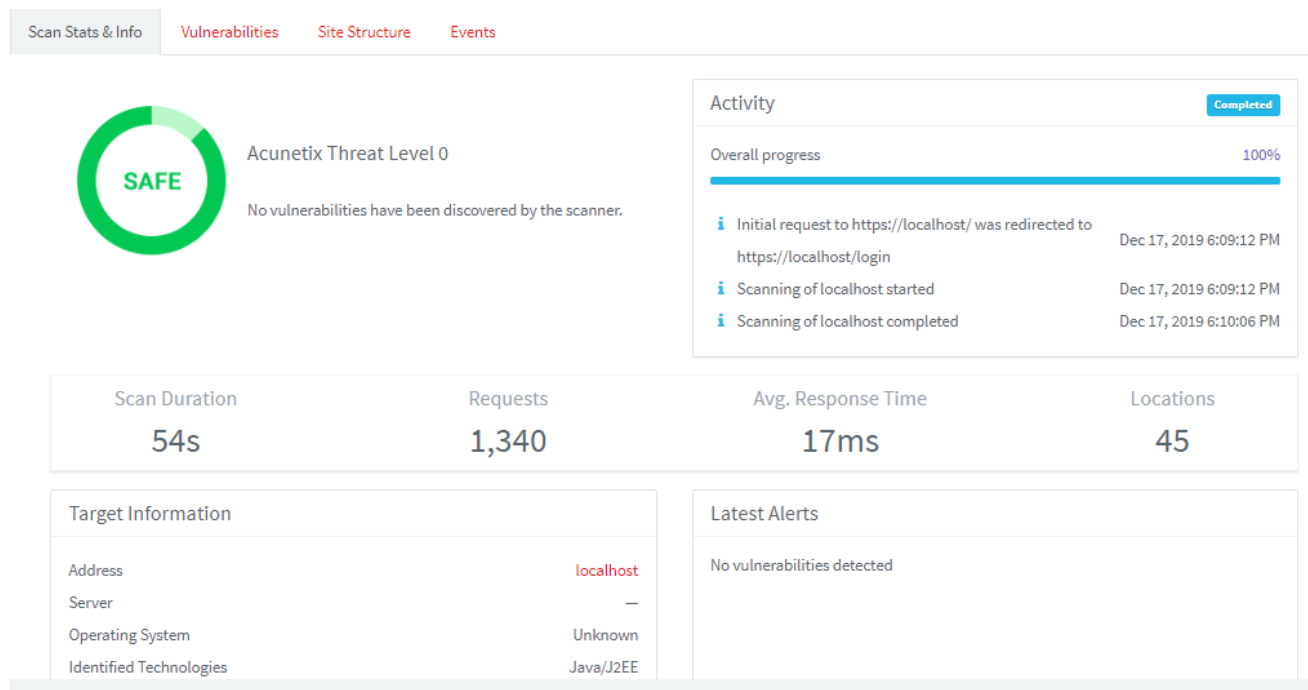
Scan Report of Weak Password:



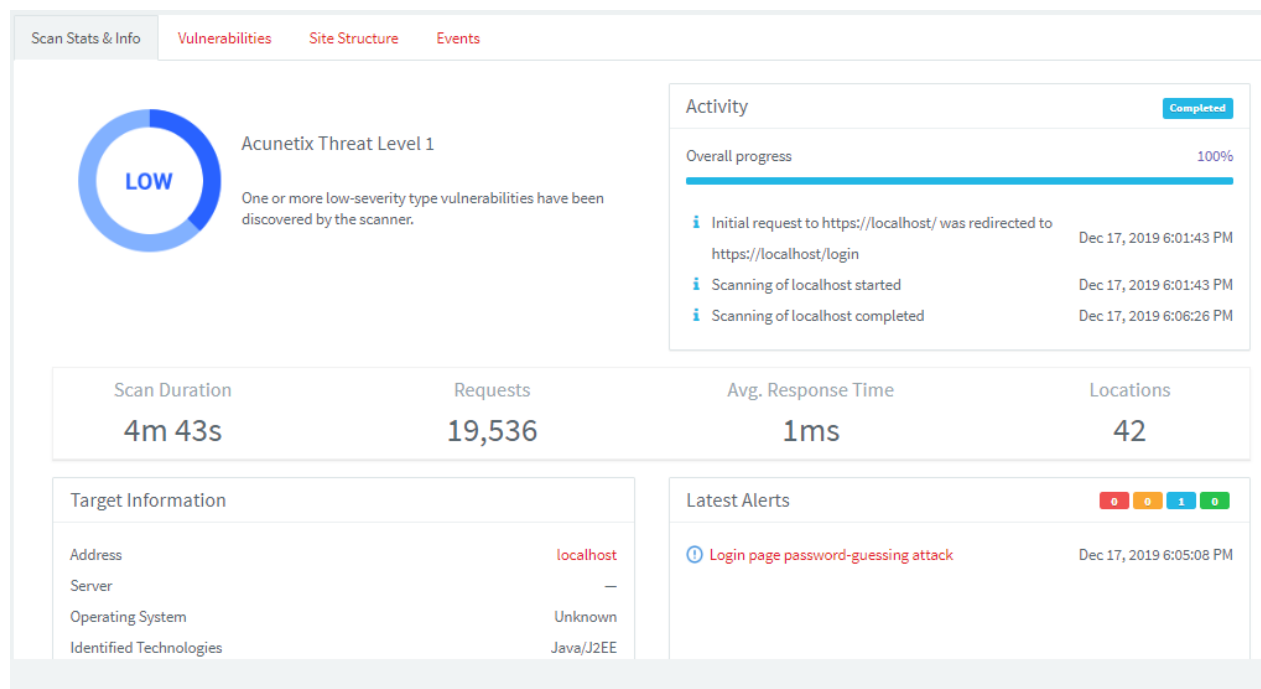
Scan Report of SQL Injection Vulnerabilities:



Scan Report of Cross Site Scripting(XSS):



Scan Report of High-Risk Vulnerabilities:



Code Review:

Code Reviewer Name : Shreyas Sudhir Barde			
File Name	Method Name(Line Number)	Security Issue	Fix Description
ProfileController.java	editProcess()	The Email ID got from the user is not verified whether correct or not.	Email ID needs to be validated before approving the changes

References:

- [1] Medium. (2019). 14 Top Spring Tutorials to Learn Spring Java Framework Online -(Updated 2020). [online] Available at: <https://medium.com/quick-code/top-tutorials-to-learn-spring-framework-for-the-java-application-12db01d9c288>.
- [2] Spring.io. (2019). Spring Projects. [online] Available at: <https://spring.io/projects/spring-security>.
- [3] Thymeleaf.org. (2019). Thymeleaf. [online] Available at: <https://www.thymeleaf.org/>.
- [4] Codementor.io. (2019). Backend Spring Boot Security | Codementor. [online] Available at: <https://www.codementor.io/@mrifni/backend-spring-boot-security-uj4x5yddh>.
- [5] Blog.sucuri.net. (2019). [online] Available at: <https://blog.sucuri.net/2018/10/owasp-top-10-security-risks-part-i.html>.