



# SLIIT

*Discover Your Future*

## SECURE SOFTWARE SYSTEMS

### Security Vulnerabilities

Documentation – SE(WEEKEND)

IT Number	Name	Contribution
IT20132200	Imalsha R.C.	<ul style="list-style-type: none"><li>• Implement SSL Certificate.</li><li>• Setting up a Content Security Policy (CSP)</li></ul>
IT20011970	Bamunusinghe G.P.	<ul style="list-style-type: none"><li>• Implement CSRF Token.</li><li>• Null point safety</li><li>• Immutability and data encapsulation</li><li>• Information Disclosure Vulnerability</li></ul>
IT19973470	Gunawardana I.I.E.	<ul style="list-style-type: none"><li>• Broken Access Control.</li><li>• Sensitive Data Exposure</li></ul>
IT20139476	Kavinda W.W.H	<ul style="list-style-type: none"><li>• Session Time out.</li></ul>

## Contents

1. Introduction about application .....	3
2. Overview Before fixed.....	4
3. Vulnerability Tools .....	4
4. Fixed Vulnerabilities.....	7
5. Overview After fixed. ....	31

## 1. Introduction about application

This project focuses on implementing one of the most contemporary and well-organized Distributed Systems techniques. RESTful web services are what they're called. In order to meet the criteria, several kinds of technologies were used to implement the API. In this project we implemented Agri products purchasing system.

Agri product management system is an application which can purchase agree products online which are uploaded by the farmers. The buyers can purchase items which are needed in 24/7 hours. Users must first create an account, after which they may log in using their email address and password. Farmers can view the items that they added, and they can add new items to the system, as well as update and delete them.

Buyers can view items and search for items that they want to buy, and they can add those items to the cart. Buyers can buy multiple items at the same time. They can choose a delivery method after adding products to the cart.

The SOA style is used to implement this system. There are three levels to it. Integrated Applications, Data Repository, Reusable Business Service an ERP system is a reusable set of services that may be combined to form Integrated Applications.

There is primary data base in the data repository layer. It is Agriculture DB. These databases are linked to three different commercial services. Buyer service, Farmer service, Delivery and service are the three services.

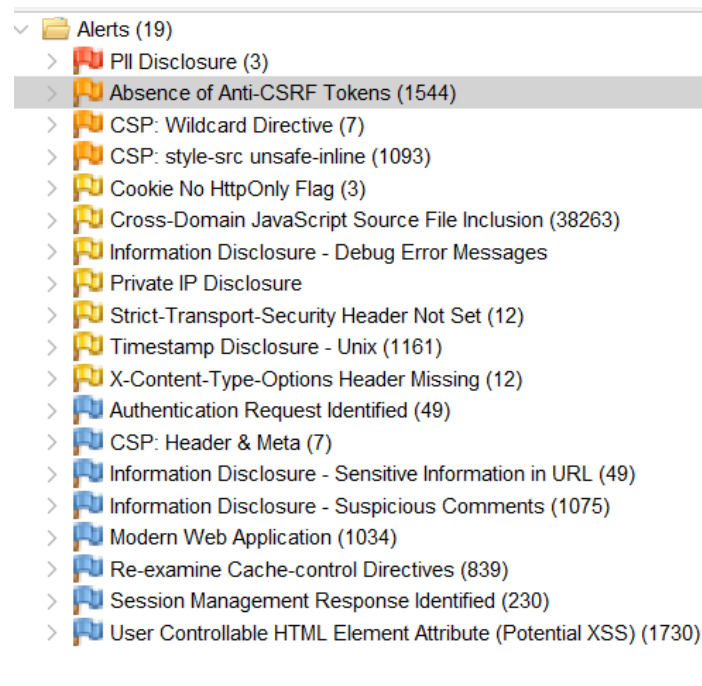
## 2. Overview Before fixed.

<https://github.com/DilkiNuwansara/Agri-WSO2-main>

## 3. Vulnerability Tools

### ZAP Tool

An open-source security testing tool called ZAP (Zed Attack Proxy) is used to identify weaknesses in online applications. ZAP, created by OWASP (Open online Application Security Project), offers automated scanners and a number of tools that permit security experts and developers to identify security flaws when creating and testing online applications. These are some important ZAP features:



**SPOT BUGS**

An open-source static code analysis tool called SpotBugs finds probable defects and problems in Java programs. It is a branch of Bill Pugh's popular FindBugs program, which was first developed. Without actually running the code, SpotBugs assists developers in finding flaws, weaknesses, and problematic coding patterns in Java codebases. The following are SpotBugs' main characteristics and aspects:

# SpotBugs Report

Produced using [SpotBugs4.4.2](#).

Project: Agri-WSO2-main[FarmerService]

## Metrics

507 lines of code analyzed, in 20 classes, in 7 packages.

Metric	Total	Density*
High Priority Warnings		NaN
Medium Priority Warnings	20	39.45
<b>Total Warnings</b>	<b>20</b>	<b>39.45</b>

*(\* Defects per Thousand lines of non-commenting source statements)*

## Summary

Warning Type	Number
<a href="#">Malicious code vulnerability Warnings</a>	20
<b>Total</b>	<b>20</b>

# OWSAP Tool



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by this tool. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the results of the analysis.

[How to read the report](#) | [Suppressing false positives](#) | Getting Help: [github issues](#)

[Sponsor](#)

## Project:

Scan Information ([show all](#)):

- *dependency-check version:* 8.4.0
- *Report Generated On:* Mon, 18 Sep 2023 18:22:00 +0530
- *Dependencies Scanned:* 0 (0 unique)
- *Vulnerable Dependencies:* 0
- *Vulnerabilities Found:* 0
- *Vulnerabilities Suppressed:* 0
- ...

## Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
------------	-------------------	---------	------------------	-----------	------------	----------------

## Dependencies (vulnerable)

This report contains data retrieved from the [National Vulnerability Database](#).  
This report may contain data retrieved from the [CISA Known Exploited Vulnerability Catalog](#).  
This report may contain data retrieved from the [Github Advisory Database \(via NPM Audit API\)](#).  
This report may contain data retrieved from [RetireJS](#).  
This report may contain data retrieved from the [Sonatype OSS Index](#).

## 4. Fixed Vulnerabilities

### Cross-Site Request Forgery (CSRF) Vulnerabilities.

The web security flaw known as cross-site request forgery enables attackers to trick users into taking actions they did not plan to take. It gives an attacker the ability to partially get around the same origin policy, which is meant to stop various websites from interfering with one another.

Additionally, once the attacker obtains the link and alters it, the process is relatively straightforward.

all they have to do is put the link in a source that the website administrator trusts and make sure it gets clicked.

if the site administrator is logged into their account. Typically, emails are used to send these URLs.

using fictitious justifications, such as reporting a broken link or updating information.

### Before

#### Backend

```
package lk.agri.service.impl;

import lk.agri.dto.UserAccountDTO;
import lk.agri.entity.UserAccount;
import lk.agri.repository.UserAccountRepository;
import lk.agri.service.UserAccountService;
import org.apache.tomcat.util.codec.binary.Base64;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

@Service
public class UserAccountServiceImpl implements UserAccountService {

    @Autowired
    private UserAccountRepository userAccountRepository;

    @Override
    public UserAccountDTO login(UserAccount userAccount) {
        byte[] encodedBytes = Base64.encodeBase64(userAccount.getPassword().getBytes());
        userAccount.setPassword(new String(encodedBytes));
        UserAccount userAccountObj = userAccountRepository.findByEmailAndPassword(userAccount.getEmail(), userAccount.getPassword());
        return new UserAccountDTO(userAccountObj);
    }

    @Override
    public UserAccountDTO signUp(UserAccount userAccount) {
        byte[] encodedBytes = Base64.encodeBase64(userAccount.getPassword().getBytes());
        userAccount.setPassword(new String(encodedBytes));
        UserAccountDTO userAccountDTO = new UserAccountDTO(userAccountRepository.save(userAccount));
        userAccountDTO.setPassword(null);
        return userAccountDTO;
    }
}
```

## Frontend

```
import {Injectable} from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {Observable} from "rxjs";
import {environment} from "../../environments/environment";

@Injectable({
  providedIn: 'root'
})
export class FarmerService {

  item;

  constructor(private http: HttpClient) {
  }

  addItem(item): Observable<any> {
    return this.http.post<any>('http://localhost:8082' + "/farmer/addItem", item);
  }

  updateItem(item, itemId): Observable<any> {
    return this.http.put<any>('http://localhost:8082' + "/farmer/updateItem/" + itemId, item);
  }

  removeItem(itemId): Observable<any> {
    return this.http.delete<any>(environment.backend_url3 + "/farmer/removeItem/" + itemId);
  }

  getItems(): Observable<any> {
    return this.http.get<any>(environment.backend_url3 + "/farmer/getItems/" + JSON.parse(localStorage.getItem('user')).email);
  }

  addChat(setItem: any): Observable<any>{
    console.log(setItem)
    return this.http.post(environment.backend_url3+ "/farmer/addChat", setItem);
  }

  getAllChats(): Observable<any>{
    return this.http.get<any>(environment.backend_url3 + "/farmer/getchat");
  }
}
```



## After

### Backend

```
package lk.agri.service.impl;

import lk.agri.dto.UserAccountDTO;
import lk.agri.entity.UserAccount;
import lk.agri.repository.UserAccountRepository;
import lk.agri.service.UserAccountService;
import org.apache.tomcat.util.codec.binary.Base64;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Optional;
import java.util.Random;

@Service
public class UserAccountServiceImpl implements UserAccountService {

    @Autowired
    private UserAccountRepository userAccountRepository;

    @Override
    public UserAccountDTO login(UserAccount userAccount) {
        byte[] encodedBytes = Base64.encodeBase64(userAccount.getPassword().getBytes());
        userAccount.setPassword(new String(encodedBytes));
        UserAccount userAccountObj = userAccountRepository.findByEmailAndPassword(userAccount.getEmail(), userAccount.getPassword());
        Optional<UserAccount> userUpdate= Optional.ofNullable(userAccountRepository.findByEmailAndPassword(userAccount.getEmail(), userAccount.getPassword()));
        if (userUpdate !=null) {
            UserAccount userObj = userUpdate.get();
            String token=(new Random().nextInt(10000)+1)+"-"+(new Random().nextInt(10000)+1)+"-"+(new Random().nextInt(10000)+1)+"-"+(new Random().nextInt(10000)+1);
            userObj.setToken(token);
            userAccountRepository.save(userObj);
        }
        return new UserAccountDTO(userAccountObj);
    }

    @Override
    public UserAccountDTO signUp(UserAccount userAccount) {
        byte[] encodedBytes = Base64.encodeBase64(userAccount.getPassword().getBytes());
        userAccount.setPassword(new String(encodedBytes));
        UserAccountDTO userAccountDTO = new UserAccountDTO(userAccountRepository.save(userAccount));
        userAccountDTO.setPassword(null);
        return userAccountDTO;
    }
}
```

```

package lk.agri.Security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;
import org.springframework.security.web.csrf.CsrfTokenRepository;
import org.springframework.security.web.csrf.HttpSessionCsrfTokenRepository;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private CSRFFilter csrfFilter;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests().antMatchers("/**").permitAll();//test
        http.addFilterBefore(csrfFilter, BasicAuthenticationFilter.class);
    }

    @Bean
    public CsrfTokenRepository csrfTokenRepository() {
        HttpSessionCsrfTokenRepository repository = new HttpSessionCsrfTokenRepository();
        repository.setHeaderName("X-CSRF-TOKEN");
        return repository;
    }
}

```

```

package lk.agri.Security;
import com.fasterxml.jackson.databind.ObjectMapper;
import lk.agri.entity.UserAccount;
import lk.agri.repository.UserAccountRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;
import org.springframework.web.filter.OncePerRequestFilter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

@Service
@Component
public class CSRFFilter extends OncePerRequestFilter {

    @Autowired
    private UserAccountRepository userAccountRepository;

    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, FilterChain filterChain) throws ServletException, IOException, RuntimeException {

        String authorizationHeader = httpServletRequest.getHeader("X-CSRF-TOKEN");
        String authorizeUser = httpServletRequest.getHeader("USER");

        if (authorizeUser != null) {
            UserAccount userAccountObj = userAccountRepository.findByEmail(authorizeUser);
            if (userAccountObj == null) {
                // Handle the case where the user account is not found
                logger.warn("User account not found for email: " + authorizeUser);

                // You can choose how to handle this situation; for example, you might want to return a 404 Not Found response.
                // Here, we'll return a 404 response as an example.
                Map<String, String> errorResponse = new HashMap<>();
                errorResponse.put("path", "/");

                // Convert the JSON object to a string
                ObjectMapper objectMapper = new ObjectMapper();
                String jsonResponse = objectMapper.writeValueAsString(errorResponse);

                // Set the HTTP response status to 403 Forbidden (or any other appropriate status code)
                httpServletResponse.setStatus(HttpServletResponse.SC_FORBIDDEN);

                // Set the response content type to JSON
                httpServletResponse.setContentType(MediaType.APPLICATION_JSON_VALUE);

                // Write the JSON error response to the response body
                httpServletResponse.getWriter().write(jsonResponse);
                return;
            }
        } else {
            if (!httpServletRequest.getRequestURI().startsWith("/farmer/csrf-token")) {
                if (authorizationHeader == null || !authorizationHeader.equals(userAccountObj.getToken())) {
                    logger.warn("JWT Token does not match expected value or is missing.");

                    // Create a JSON object for the error message
                    Map<String, String> errorResponse = new HashMap<>();
                    errorResponse.put("path", "/");

                    // Convert the JSON object to a string
                    ObjectMapper objectMapper = new ObjectMapper();
                    String jsonResponse = objectMapper.writeValueAsString(errorResponse);

                    // Set the HTTP response status to 403 Forbidden (or any other appropriate status code)

```

```

        httpServletResponse.setStatus(HttpServletResponse.SC_FORBIDDEN);

        // Set the response content type to JSON
        httpServletResponse.setContentType(MediaType.APPLICATION_JSON_VALUE);

        // Write the JSON error response to the response body
        httpServletResponse.getWriter().write(jsonResponse);

        // Stop the filter chain
        return;
    }
}
}
}

// Continue the filter chain
filterChain.doFilter(httpServletRequest, httpServletResponse);
}
}

```

## frontend

```

constructor(private http: HttpClient) {
}

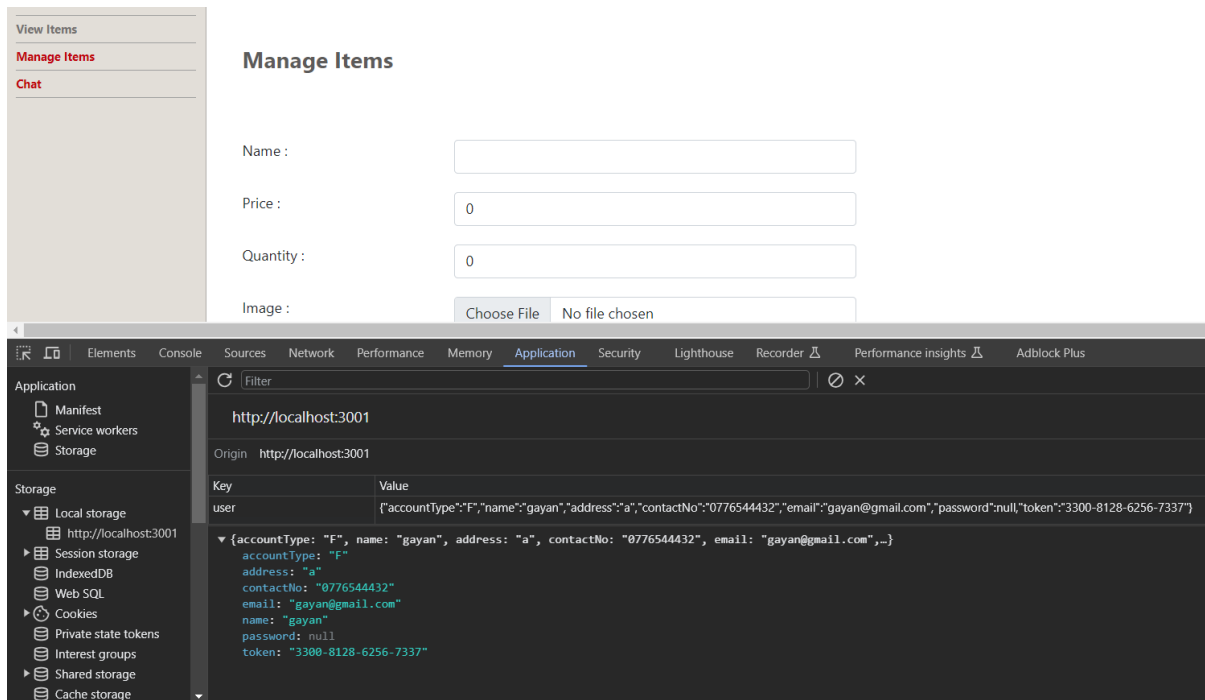
addItem(item): Observable<any> {
    const headersToken = new HttpHeaders()
        .set('X-CSRF-TOKEN', JSON.parse(localStorage.getItem('user')).token) // Replace with your header name and value
        .set('USER', JSON.parse(localStorage.getItem('user')).email);
    return this.http.post<any>('http://localhost:8082' + '/farmer/addItem', item, { headers: headersToken });
}

updateItem(item, itemId): Observable<any> {
    const headersToken = new HttpHeaders()
        .set('X-CSRF-TOKEN', JSON.parse(localStorage.getItem('user')).token) // Replace with your header name and value
        .set('USER', JSON.parse(localStorage.getItem('user')).email);
    return this.http.put<any>('http://localhost:8082' + '/farmer/updateItem/' + itemId, item, { headers: headersToken });
}

removeItem(itemId): Observable<any> {
    const headersToken = new HttpHeaders()
        .set('X-CSRF-TOKEN', JSON.parse(localStorage.getItem('user')).token) // Replace with your header name and value
        .set('USER', JSON.parse(localStorage.getItem('user')).email);
    return this.http.delete<any>(environment.backend_url3 + '/farmer/removeItem/' + itemId, { headers: headersToken });
}

getItems(): Observable<any> {
    // Make the GET request with the custom headers
    const headersToken = new HttpHeaders()
        .set('X-CSRF-TOKEN', JSON.parse(localStorage.getItem('user')).token) // Replace with your header name and value
        .set('USER', JSON.parse(localStorage.getItem('user')).email);
    return this.http.get<any>({
        environment.backend_url3 + '/farmer/getItems/' + JSON.parse(localStorage.getItem('user')).email,
        { headers: headersToken }
    });
    // return this.http.get<any>(environment.backend_url3 + '/farmer/getItems/' + JSON.parse(localStorage.getItem('user')).email);
}

```



## Immutability and data encapsulation

Immutability is the idea that an object's state cannot be changed after it is formed in computer science and software engineering. In terms of programming, it indicates that once saved values in an object are set, they cannot be modified. Immutable data structures create new objects with the updated values rather than altering already-existing objects. Immutable objects make the code simpler, more predictable, understandable, and logically sound.

One of the guiding concepts of object-oriented programming (OOP) is data encapsulation. It refers to grouping together into a single entity known as a class, data (attributes or properties) and methods (functions or procedures) that operate on the data. Encapsulation assists in preventing unauthorized access to an object's internal state and limiting access to internal data to those who have passed through clearly defined interfaces (public methods).

## Before

```
public void setCartDetails(List<CartDetailDTO> cartDetails) { this.cartDetails = cartDetails; }
```

## After

```
public List<CartDetailDTO> getCartDetails() {  
    return new ArrayList<>(cartDetails);  
}  
  
public void setCartDetails(List<CartDetailDTO> cartDetails) {  
    this.cartDetails = new ArrayList<>(cartDetails);  
}
```

## Information Disclosure Vulnerability

Information Disclosure Vulnerability refers to a security flaw where an application, system, or service inadvertently reveals sensitive information to unauthorized users or entities.

## Before

```
public List<CartDetailDTO> getCartDetails() {  
    return cartDetails;  
}
```

## After

```
public List<CartDetailDTO> getCartDetails() {  
    return new ArrayList<>(cartDetails);  
}
```

## **Null pointer safety**

Null pointer exceptions (NPEs), commonly referred to as null safety, are a programming concept that aims to avoid them in software programs. When a program tries to access or call a method on an object that is null, a null pointer exception is thrown. Null pointers can cause sudden software crashes and abnormal behaviour.

### **Before**

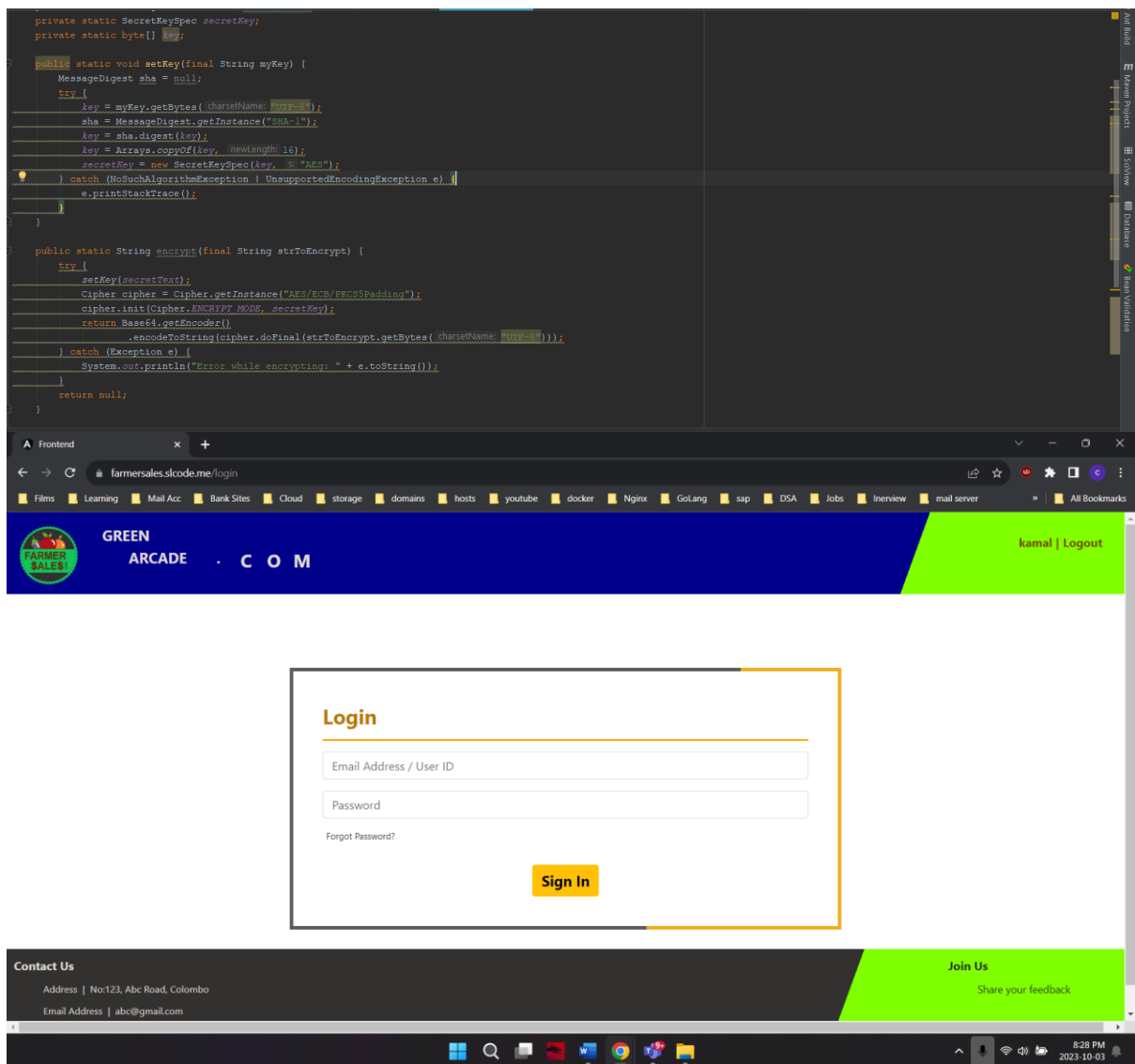
```
public void setCartDetails(Set<CartDetail> cartDetails) {  
    this.cartDetails = cartDetails;  
}
```

### **After**

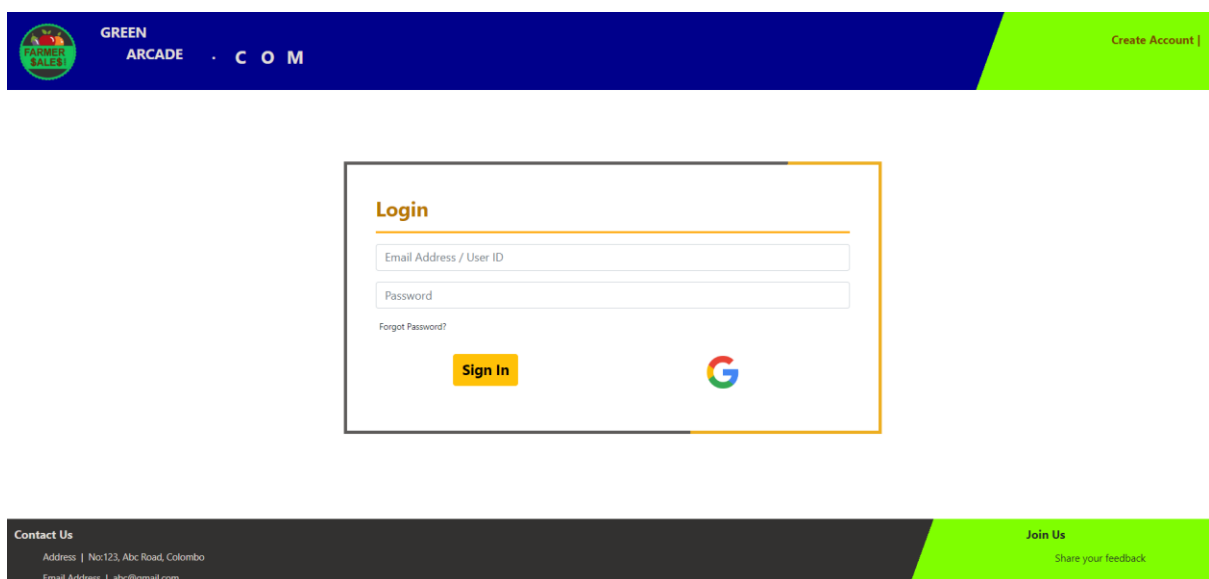
```
public void setCartDetails(Set<CartDetail> cartDetails) {  
    if (cartDetails != null) {  
        this.cartDetails.clear(); // Clear the current contents  
        this.cartDetails.addAll(cartDetails); // Add all elements from the provided set  
    }  
}
```

## **OAuth (Open Authorization): Securing API Access**


OAuth (Open Authorization) is an open standard for access delegation and authorization. It allows users to grant third-party applications limited access to their resources without sharing their credentials (like passwords). OAuth is commonly used in the context of web and mobile applications that need to access user data from other services or APIs.



After





 Sign in with Google

## Choose an account

to continue to [SSD](#)



**Gayan Poornima**  
gayanpoornima@gmail.com



**gayan poornima**  
gayanpoornimago@gmail.com



**Use another account**

To continue, Google will share your name, email address, language preference, and profile picture with SSD.

English (United States) ▼

[Help](#)

[Privacy](#)

[Terms](#)

## Broken Access control

Broken Access Control refers to a situation where an application does not properly restrict users from accessing resources or performing actions they should not be allowed to do. It occurs when access control checks are not consistently enforced or can be easily bypassed, allowing unauthorized users to perform actions or access data that should be restricted to them.

If one user logged in their own account, they should not be allowed to access other account holders' resources.

## Before

```
package lk.agri.service.impl;

import ...

@Service
public class UserAccountServiceImpl implements UserAccountService {

    @Autowired
    private UserAccountRepository userAccountRepository;

    @Override
    public UserAccountDTO login(UserAccount userAccount) {
        UserAccount userAccountObj = userAccountRepository.findByEmailAndPassword(userAccount.getEmail(), userAccount.getPassword());
        return new UserAccountDTO(userAccountObj);
    }

    @Override
    public UserAccountDTO signUp(UserAccount userAccount) {
        UserAccountDTO userAccountDTO = new UserAccountDTO(userAccountRepository.save(userAccount));
        userAccountDTO.setPassword(null);
        return userAccountDTO;
    }
}
```

## After

```
@SuppressWarnings("all")
private JwtUtil jwtUtil;

@Override
public UserAccountDTO login(UserAccount userAccount) {
    // byte[] encodedBytes = Base64.encodeBase64(userAccount.getPassword().getBytes());
    // userAccount.setEmail(Encryption.encrypt(userAccount.getEmail()));
    // userAccount.setPassword(Encryption.encrypt(userAccount.getPassword()));
    UserAccount userAccountObj = userAccountRepository.findByEmailAndPassword(userAccount.getEmail(), userAccount.getPassword());
    UserAccountDTO userAccountDTO = new UserAccountDTO(userAccountObj);
    userAccountDTO.setUserToken(jwtUtil.generate(userAccountDTO));
    userAccountDTO.setAccountType(jwtUtil.decode(userAccountObj.getAccountType()));
    return userAccountDTO;
}

@Override
public UserAccountDTO signUp(UserAccount userAccount) {
    // byte[] encodedBytes = Base64.encodeBase64(userAccount.getPassword().getBytes());
    // userAccount.setEmail(Encryption.encrypt(userAccount.getEmail()));
    // userAccount.setPassword(Encryption.encrypt(userAccount.getPassword()));
    // userAccount.setAccountType(jwtUtil.generate(new UserAccountDTO(userAccount)));
    UserAccountDTO userAccountDTO = new UserAccountDTO(userAccountRepository.save(userAccount));
    userAccountDTO.setPassword(null);
    return userAccountDTO;
}

@Override
public UserAccountDTO loginGoogle(UserAccount userAccount) {
    userAccount.setEmail(Encryption.encrypt(userAccount.getEmail()));
    UserAccount userAccountObj = userAccountRepository.findByEmail(userAccount.getEmail());
    UserAccountDTO userAccountDTO = new UserAccountDTO(userAccountObj);
    userAccountDTO.setUserToken(jwtUtil.generate(userAccountDTO));
    userAccountDTO.setAccountType(jwtUtil.decode(userAccountObj.getAccountType()));
    return userAccountDTO;
}
```

## Before

```
getItems() {
    this.farmerS.getItems().subscribe(items => {
        console.log(items);
        this.items = items;
    })
}
```

## After

```
getItems() {
  this.farmersS.getItems().subscribe(items => {
    console.log(items)
    this.items = items;
  })
  .(err)=>{
    // console.log(err)
    this.router.navigate( commands: ['login'])
    localStorage.clear()
  }
}
```

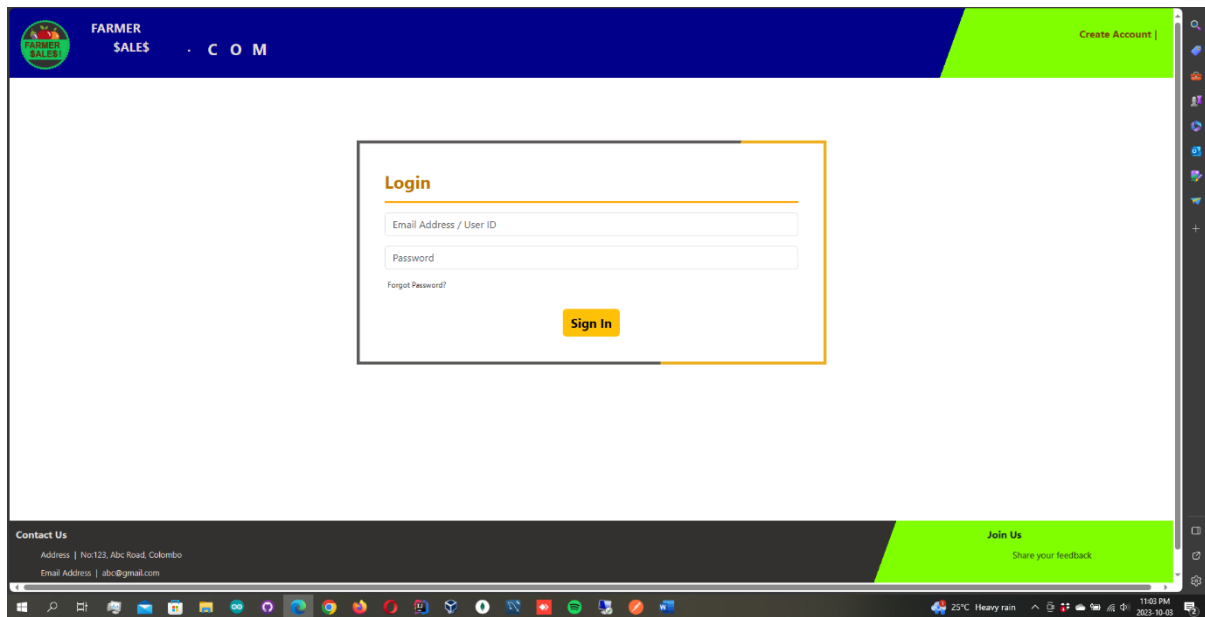
## Before

```
getItems() {
  if (this.txt === '') {
    this.txt = undefined;
  }
  this.buyerS.getItems(this.txt).subscribe(items => {
    console.log(items)
    this.items = items;
  })
}
```

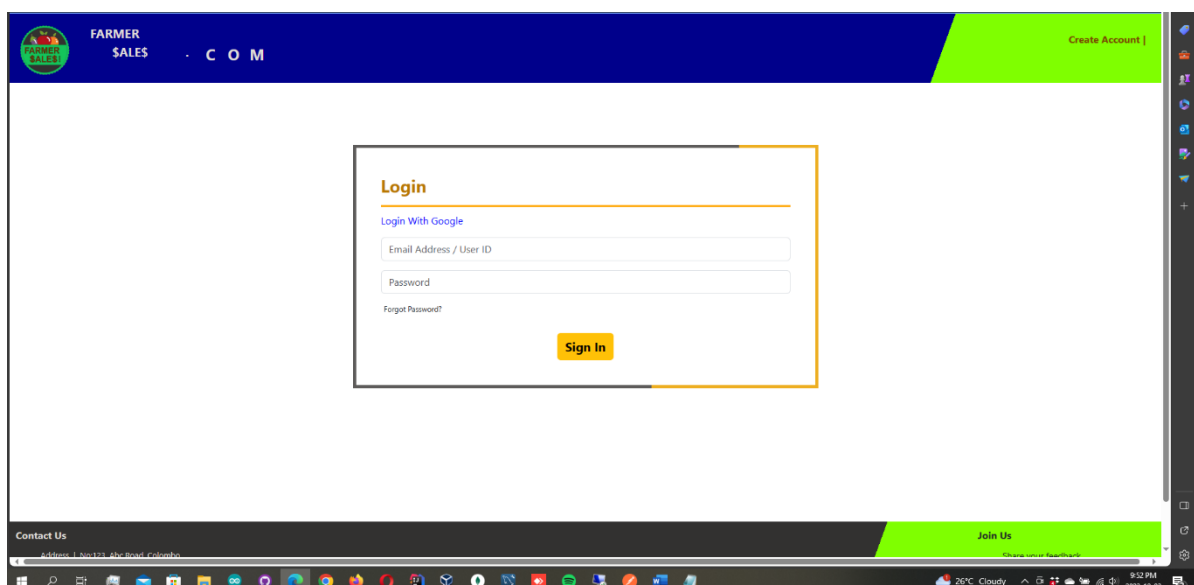
## After

```
getItems() {
  if (this.txt === '') {
    this.txt = undefined;
  }
  this.buyerS.getItems(this.txt).subscribe(items => {
    console.log(items)
    this.items = items;
  })
  .(err)=>{
    // console.log(err)
    this.router.navigate( commands: ['login'])
    localStorage.clear()
  }
}
```

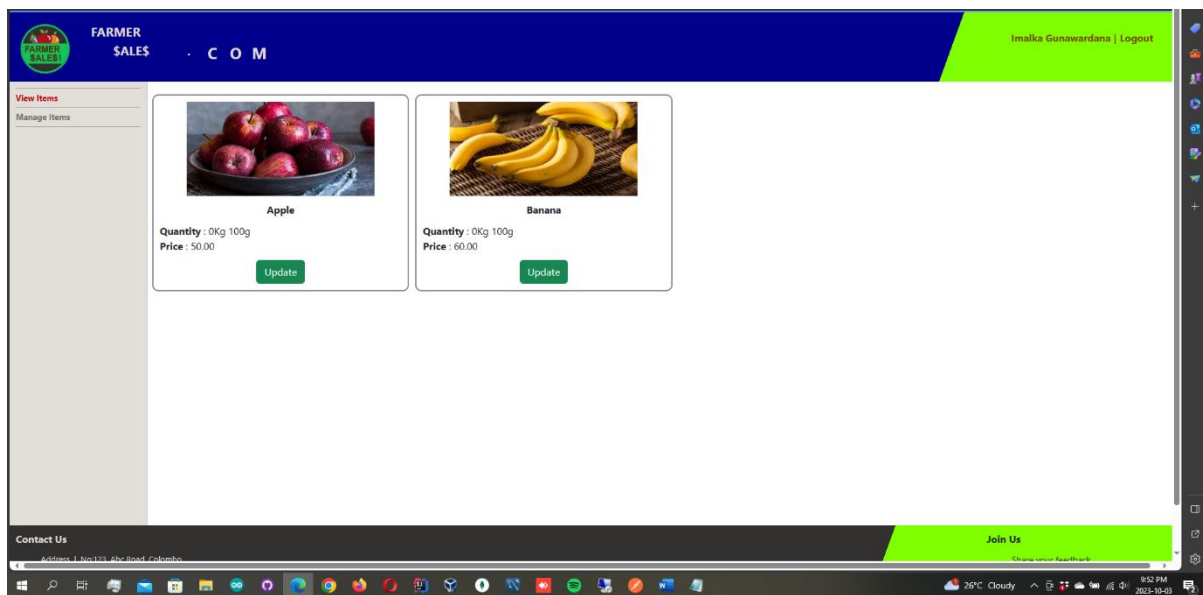
## Before



## After



## After



```
@Component
public class JwtFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, FilterChain filterChain) throws ServletException, IOException, RuntimeException {

        String authorizationHeader = httpServletRequest.getHeader("Authorization");

        if (authorizationHeader != null) {
            String decrypt = Encryption.decrypt(authorizationHeader);
            JSONParser parser = new JSONParser();
            try {
                JSONObject json = (JSONObject) parser.parse(decrypt);
                if (json.get("accountType").toString().equals("farmer")) {

                } else {
                    throw new RuntimeException("Invalid Token");
                }
            } catch (ParseException e) {
                e.printStackTrace();
            }
        } else {
            throw new RuntimeException("No Token");
        }

        if (user != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        }
    }
}
```

```
email: "",
password: ""
};
logged = true;

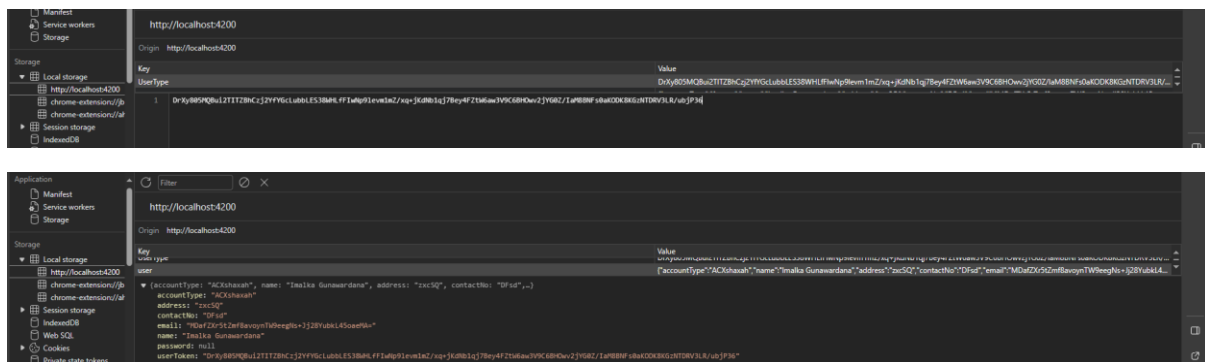
constructor(private loginService: LoginService, private navBarService: NavBarService, private router: Router, private authService: AuthService) {
}

ngOnInit(): void {
}

onSubmit() {
    this.loginService.eccLogin(this.user).subscribe((user) => {
        console.log(user);
        localStorage.setItem(key: 'user', JSON.stringify(user));
        localStorage.setItem(key: 'userToken', user['userToken']);
        if (user != null && user['accountType'] === 'farmer') {
            this.router.navigate(commands: ['/main/farmer/view_items'])
        } else if (user['accountType'] === 'buyer') {
            this.router.navigate(commands: ['/main/buyer/view_items']);
        } else {
            this.logged = false;
        }
    }, (err) => {
        this.logged = false;
    })
}

loginGoogle() {
}
```

## User Tokens



## Sensitive data exposure

An application or system fails to adequately protect sensitive data, making it accessible to unauthorized individuals or entities. This vulnerability is a common target for attackers seeking to steal sensitive information such as credit card numbers, personal identification data, healthcare records, and other confidential information. If an attacker get access to the database. They can see sensitive data. The application should make those sensitive data unreadable.

## Before

	email	account_type	address	contact_no	name	password
	imalkagunawardana1@gmail.com	farmer	zxc	123	Imalka Gunawardana	123
	imalkagunawardana4@gmail.com	buyer	qwe	234	Gayan	234
	NULL	NULL	NULL	NULL	NULL	NULL

## After

	email	account_type	address	contact_no	name	password
	MDafZx5tZmf8avoyntW9eegNs+j28YubkL45...	DrXy805MQBui2TITZBhCzjZyfyGdLubLES38WH...	zxcSQ	DFsd	Imalka Gunawardana	hAz5U4ZfmGc9IZ+2Mqng3w ==
	MDafZx5tZmf8avoyntW9UhgZTaksMcb1JUGf...	mgurtd+hrKl4kvl4mkc9flbwprlUwxc92E9l5YoS...	xcdwQv	hxuwSX	Gayan	5RwIG/655X++ScnTjI/x4w ==
	NULL	NULL	NULL	NULL	NULL	NULL

## String Encryption with AES

```

private static SecretKeySpec secretKey;
private static byte[] key;

public static void setKey(final String myKey) {
    MessageDigest sha = null;
    try {
        key = myKey.getBytes(CharsName.UTF_8);
        sha = MessageDigest.getInstance("SHA-1");
        key = sha.digest(key);
        key = Arrays.copyOf(key, newLength: 16);
        secretKey = new SecretKeySpec(key, "AES");
    } catch (NoSuchAlgorithmException | UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}

public static String encrypt(final String strToEncrypt) {
    try {
        setKey(secretText);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        return Base64.getEncoder().
            .encodeToString(cipher.doFinal(strToEncrypt.getBytes(CharsName.UTF_8)));
    } catch (Exception e) {
        System.out.println("Error while encrypting: " + e.toString());
    }
    return null;
}

```

## SSL Certificate

An SSL certificate, which stands for Secure Sockets Layer certificate, is a digital certificate that provides a secure, encrypted connection between a web browser and a web server. SSL certificates are used to ensure that data transmitted between a user's browser and a website's server remains confidential and cannot be intercepted or tampered with by malicious actors.

Below are the special features of the SSL certificate.

- **Encryption** - SSL certificates enable encryption of data exchanged between a user's browser and a web server. This encryption ensures that even if someone intercepts the data, they cannot read it without the proper encryption key.
- **Authentication** - SSL certificates also serve to authenticate the identity of the website or server to which the user is connecting. This helps users trust that they are indeed connecting to the legitimate website and not a malicious one.
- **Trust** - SSL certificates are issued by trusted Certificate Authorities (CAs). When a website presents a valid SSL certificate issued by a recognized CA, web browsers display a padlock icon or other indicators to signal that the connection is secure and trustworthy.
- **HTTPS** - Websites that use SSL certificates typically use the HTTPS (Hypertext Transfer Protocol Secure) protocol instead of HTTP. The "S" in HTTPS stands for "secure," indicating that the connection is encrypted and secure.
- **Data Integrity** - SSL certificates also ensure the integrity of the data being transmitted. This means that the data cannot be altered or tampered with during transit between the user and the server.

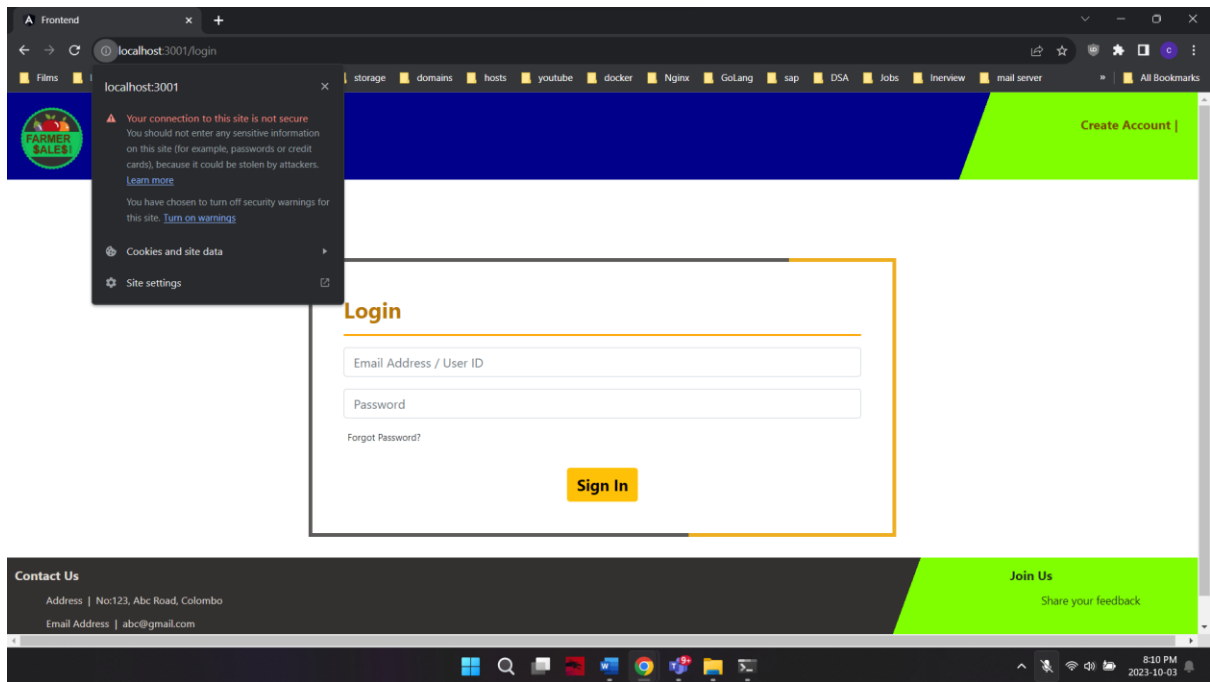
SSL certificates are essential for securing sensitive information such as login credentials, credit card numbers, and personal data that users input on websites. They are commonly used on e-commerce websites, online banking platforms, email services, and any other websites where privacy and security are critical.

Below are the Vulnerabilities that avoid using an SSL certificate.

- **Man-in-the-Middle (MitM) Attacks** - SSL certificates prevent MitM attacks by encrypting the data exchanged between the client and server. Even if an attacker intercepts the data, they cannot read or modify it without the encryption key.
- **Eavesdropping** - Without HTTPS, data transmitted over the internet is susceptible to eavesdropping. SSL encryption ensures that only the intended recipient can decrypt and read the data.
- **Data Tampering** - HTTPS ensures that data remains unchanged during transit. If an attacker attempts to modify data in transit, the recipient can detect the tampering and reject the data.
- **Spoofing and Phishing** - SSL certificates provide authentication, making it difficult for attackers to impersonate a legitimate website. Users can trust that they are connecting to the real site and not a fake one.
- **Data Interception** - Intercepting unencrypted data is relatively easy for attackers. SSL certificates prevent this interception, making it much more challenging for malicious actors to steal sensitive information.
- **Session Hijacking** - SSL certificates protect against session hijacking by encrypting session cookies and preventing attackers from stealing users' session information.
- **Insecure Public Wi-Fi** - When users connect to public Wi-Fi networks, their data can be vulnerable. HTTPS secures their connections, reducing the risk of data theft on unsecured networks.

**Before implement SSL Certificate**





## After implement SSL Certificate

The image below shows the details about the SSL certificate used for this application.

---

Server Hostname

Check SSL

✓

farmersales.slcode.me resolves to 104.21.59.134

✓

Server Type: cloudflare

✓

The certificate should be trusted by all major web browsers (all the correct intermediate certificates are installed).

✓

The certificate will expire in 45 days.

Remind me

✓

The hostname (farmersales.slcode.me) is correctly listed in the certificate.

Server

Common name: slcode.me

SANs: slcode.me, \*.slcode.me

Valid from August 19, 2023 to November 17, 2023

Serial Number: 8eafa4f4c3e8fe5d11cdf7a6f70575e7

Signature Algorithm: sha256WithRSAEncryption

Issuer: GTS CA 1P5

↓

Chain

Common name: GTS CA 1P5

Organization: Google Trust Services LLC

Location: US

Valid from August 12, 2020 to September 29, 2027

Serial Number: 0203bc50a32753f0918022edf1

Signature Algorithm: sha256WithRSAEncryption

Issuer: GTS Root R1

↓

Chain

Common name: GTS Root R1

Organization: Google Trust Services LLC

Location: US

Valid from June 18, 2020 to January 27, 2028

Serial Number: 77bd0d6cdb36f91aea210fc4f058d30d

Signature Algorithm: sha256WithRSAEncryption

Issuer: GlobalSign Root CA

---



farmersales.slcode.me resolves to 104.21.59.134



Server Type: cloudflare



The certificate should be trusted by all major web browsers (all the correct intermediate certificates are installed).



The certificate will expire in 45 days.

Remind me



The hostname (farmersales.slcode.me) is correctly listed in the certificate.



Common name: slcode.me  
SANs: slcode.me, \*.slcode.me  
Valid from August 19, 2023 to November 17, 2023  
Serial Number: 8eafa4f4c3e8fe5d11cdf7a6f70575e7  
Signature Algorithm: sha256WithRSAEncryption  
Issuer: GTS CA 1P5



Common name: GTS CA 1P5  
Organization: Google Trust Services LLC  
Location: US  
Valid from August 12, 2020 to September 29, 2027  
Serial Number: 0203bc50a32753f0918022edf1  
Signature Algorithm: sha256WithRSAEncryption  
Issuer: GTS Root R1



Common name: GTS Root R1  
Organization: Google Trust Services LLC  
Location: US  
Valid from June 18, 2020 to January 27, 2028  
Serial Number: 77bd0d6cdb36f91aea210fc4f058d30d  
Signature Algorithm: sha256WithRSAEncryption  
Issuer: GlobalSign Root CA

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > farmersales.slcode.me

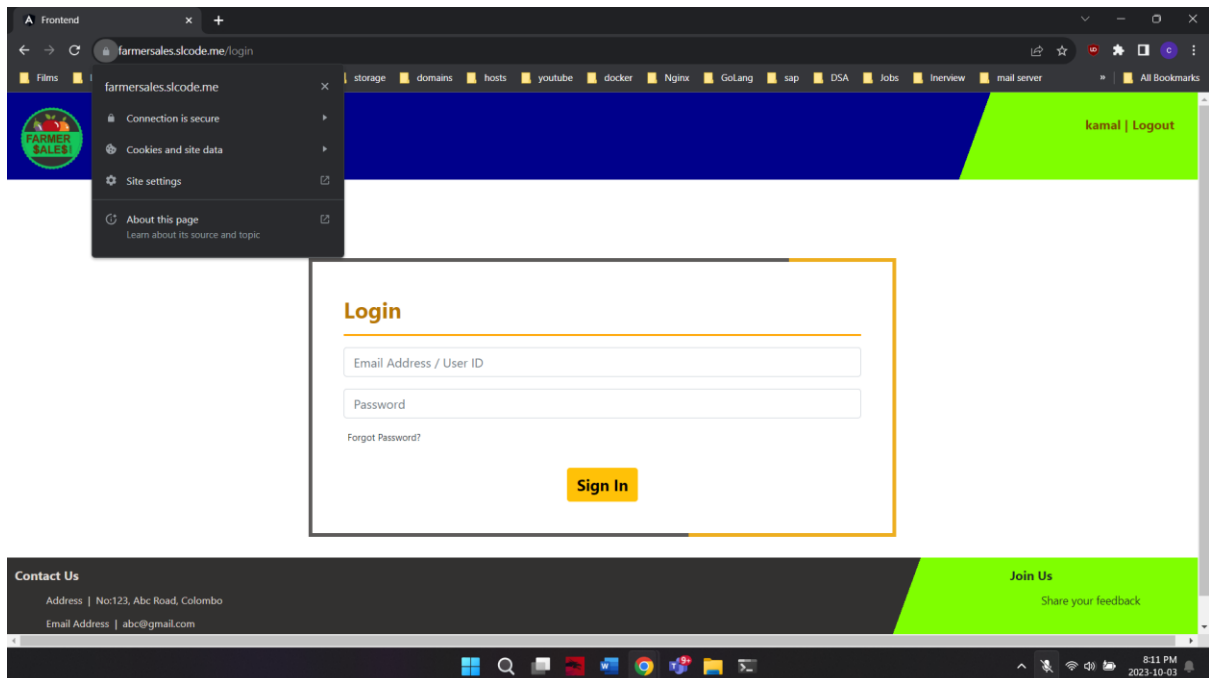
## SSL Report: farmersales.slcode.me

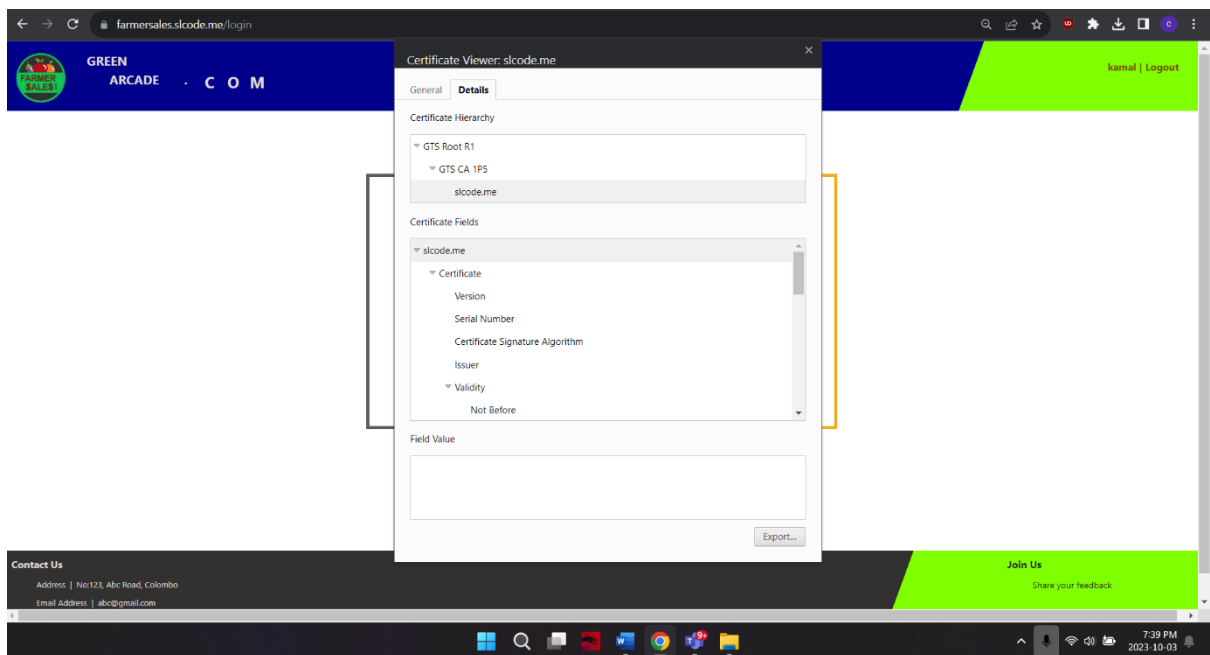
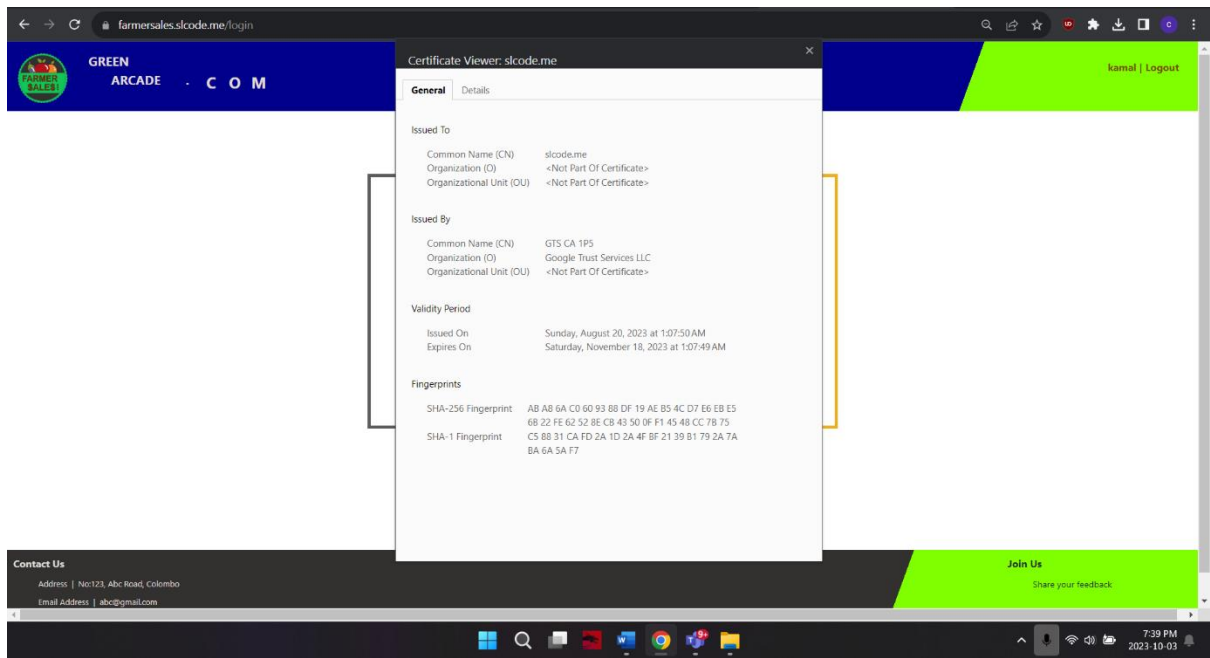
Assessed on: Tue, 03 Oct 2023 14:36:01 UTC | [Hide](#) | [Clear cache](#)

[Scan Another >>](#)

	Server	Test time	Grade
1	<a href="#">104.21.59.134</a> Ready	Tue, 03 Oct 2023 14:31:40 UTC Duration: 65.735 sec	<b>B</b>
2	<a href="#">2606:4700:3037:0:0:0:ac43:b1ea</a> Ready	Tue, 03 Oct 2023 14:32:45 UTC Duration: 65.300 sec	<b>B</b>
3	<a href="#">172.67.177.234</a> Ready	Tue, 03 Oct 2023 14:33:51 UTC Duration: 65.122 sec	<b>B</b>
4	<a href="#">2606:4700:3030:0:0:0:6815:3b86</a> Ready	Tue, 03 Oct 2023 14:34:56 UTC Duration: 65.279 sec	<b>B</b>

SSL Report v2.2.0



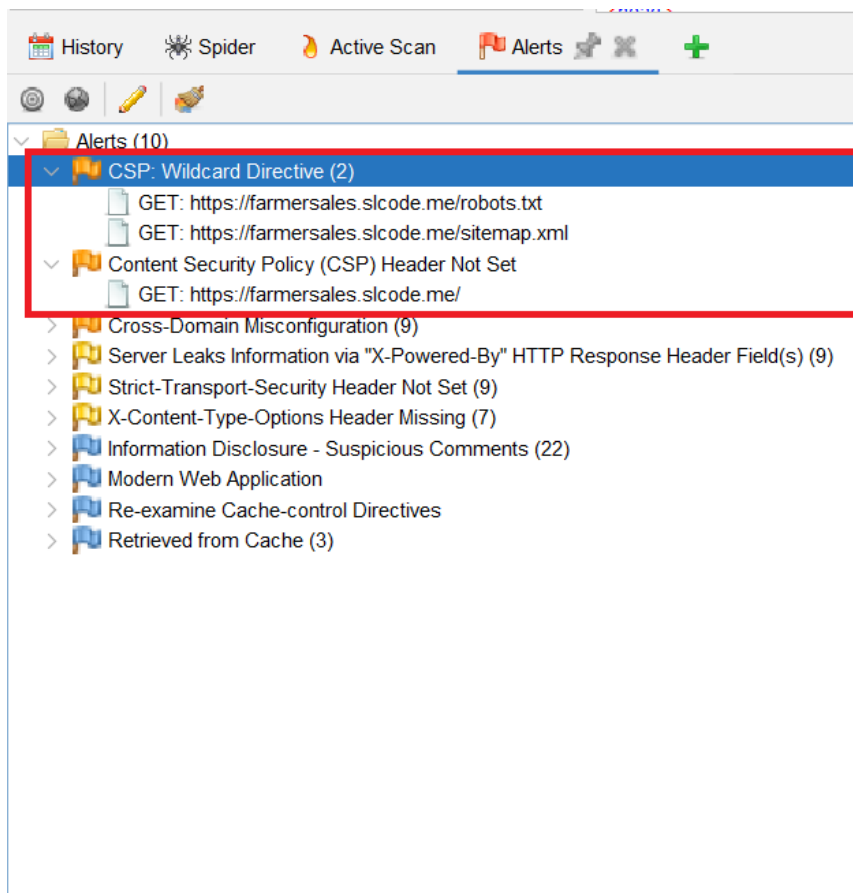


## Setting up a Content Security Policy (CSP)

Setting up a Content Security Policy (CSP) header is an important security measure to protect your web application from various types of attacks, such as **cross-site scripting (XSS)** and **data injection**. To configure the Content-Security-Policy header in an Angular app served by Nginx on an Ubuntu server below steps are followed.

- 1) Add the follow code in to the server block in the NGINX Configuration.

```
add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval' https://farmersales.slcode.me; style-src 'self' 'unsafe-inline' https://farmersales.slcode.me; img-src 'self' data: https://farmersales.slcode.me; font-src 'self' https://farmersales.slcode.me;";
```



```
server {
    server_name farmersales.slcode.me;

    location / {
        proxy_pass http://localhost:4200;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval' https://farmersales.slcode.me; style-src 's";
    }
}
```

Name	×	Headers	Payload	Preview	Response	Initiator	Timing
✗ css2?family=Quicksand:w...		Access-Control-Allow-Credentials:	true				
✗ xhr_streaming?t=169634...		Access-Control-Allow-Origin:	https://farmersales.slcode.me				
✗ eventsource		Cache-Control:	no-store, no-cache, no-transform, must-revalidate, max-age=0				
📄 iframe.html		Cf-Cache-Status:	DYNAMIC				
📄 sockjs.bundle.js		Cf-Ray:	81066700398eb2f9-CMB				
✗ htmlfile?c=_jp.afmzoog		Content-Security-Policy:	default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval' https://farmersales.slcode.me; style-src 'self' 'unsafe-inline' https://farmersales.slcode.me; img-src 'self' data: https://farmersales.slcode.me; font-src 'self' https://farmersales.slcode.me;				
📄 iframe.html		Content-Type:	application/javascript; charset=UTF-8				
📄 sockjs.bundle.js		Date:	Tue, 03 Oct 2023 16:01:11 GMT				
☐ xhr?t=1696348873247		Nel:	{"success_fraction":0,"report_to":"cf-nel","max_age":604800}				
☐ xhr?t=1696348873376		Report-To:	{"endpoints":[{"url":"https://a.nel.cloudflare.com/v/report/v3?s=QkCMq1i0auCt1e3YkEKIA12RjIOVallvtpa9Z74zLZCUAjpaznlFMKwXWVxj%2F%2Ft53bd9xtNChh%2FQ%2BL%2BmPAxz2uZrLn8umtZkmVNbaZJWNz7qt1eEizF5CmZS7cTe0zQj%2BaRLAsi%2FMk4JHVqi7bACeKL6DM%3D"}],"group":"cf-nel","max_age":604800}				
☐ xhr?t=1696348873505		Server:	cloudflare				
☐ xhr?t=1696348898639		Vary:	Origin				
☐ xhr?t=1696348923844							
☐ xhr?t=1696348949024							
☐ xhr?t=1696348974186							
15 requests   2.5 kB transferr							

## 5. Overview After fixed.

<https://github.com/IT20011970/SSD-Assignment/>

### **OneDrive Video Link: -**

<https://drive.google.com/drive/folders/1SHgGOGovnxqiHfBlaZHS9DE0pCWUGal?usp=sharing>