# AltoroJ security report

| | |
|---|---|
| Youssef Hussein Mahmoud | 20206111 |
| Ahmed El Sayed Moein | 20206120 |
| Ahmed Sami Darwish | 20206136 |
| Abdulrahman Emad Kamel | 20206148 |
| Youssef Ahmed Abdulghafar | 20206152 |

## Findings list

| ID | Name | Severity | After retest severity |
|---|---|---|---|
| **ID01** | Default Credentials in the **login page** to the admin panel | **High** | **Resolved** |
| **ID02** | SQL Injection in the **login page** | **High** | **Resolved** |
| **ID03** | Local File Inclusion (LFI) to WEB-INF init logs in the **home page** | **High** | |
| **ID04** | SQL Injection in **bank apply** to gold visa | **High** | **Resolved** |
| **ID05** | Insecure direct object reference (IDOR) in **show account history** | **High** | |
| **ID06** | Improper Access Control leading to admin pages | **High** | **Resolved** |
| **ID07** | Cross-Site Request Forgery (CSRF) in **transfer funds** | **High** | **Resolved** |
| **ID08** | Cross-Site Request Forgery (CSRF) in **adding a new user by admin** | **High** | **Resolved** |
| **ID09** | Cross-Site Request Forgery (CSRF) in **Adding Account to an existing user by admin** | **High** | **Resolved** |

| | | | |
|---|---|---|---|
| **ID10** | Cross-Site Request Forgery (CSRF) in **change user's password by admin** | **High** | **Resolved** |
| **ID11** | HTML Injection in the **home page** | **Medium** | **Resolved** |
| **ID12** | Cross-Site Scripting (XSS) in **home page search** | **Medium** | **Resolved** |
| **ID13** | Reflected Cross-Site Scripting (RXSS) in **bank search articles** | **Medium** | **Resolved** |
| **ID14** | Reflected Cross-Site Scripting (RXSS) in **customize language** | **Medium** | **Resolved** |
| **ID15** | Post-Based Cross-Site Scripting (PXSS) in **feedback** | **Medium** | **Resolved** |
| **ID16** | URL Redirection in **customize language** | **Medium** | **Resolved** |
| **ID17** | Clickjacking vulnerability | **Medium** | **Resolved** |
| **ID18** | HTML Injection in **home page search** | **Medium** | **Resolved** |
| **ID19** | Information disclosure (Sensitive data exposure) in **login.jsp code** | **Low** | **Resolved** |

# Finding details (test and retest)

**Name:** ID01: Default Credentials in the **Login Page** to the admin panel

**Test Severity: Critical**
**Test Score: 9.1**

**Description:**
Default credentials were discovered in the login page for accessing the admin panel. This vulnerability allows an attacker to access the admin panel using pre-configured or default usernames and passwords, potentially granting unauthorized access to sensitive administrative functions and data.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized access to sensitive information, manipulation of system configurations, or even complete compromise of the application and underlying infrastructure.

**Recommendations:**

- Immediately change the default credentials to strong, unique passwords that adhere to password complexity requirements.
- Implement multi-factor authentication (MFA) to add an extra layer of security to the login process and mitigate the risk of credential-based attacks.
- Regularly rotate passwords and ensure that they are not shared or reused across multiple accounts or systems.
- Conduct security assessments to identify and remediate default credential vulnerabilities, including thorough review of configuration files and settings.
- Educate administrators and users about the importance of using strong, unique passwords and avoiding default or common credentials.
- Monitor login attempts and enforce account lockout policies to prevent brute-force attacks against login credentials.

**Name:** ID02: SQL Injection in the **Login Page**

**Test Severity: Critical**
**Test Score: 9.3**

**Description:**
SQL Injection vulnerability was discovered in the login page. This vulnerability allows an attacker to manipulate the SQL queries executed by the application, potentially bypassing authentication mechanisms, accessing unauthorized data, or executing arbitrary SQL commands.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized access to sensitive information, such as user credentials or other confidential data stored in the database, as well as potential data manipulation or deletion.

**Recommendations:**

- Implement parameterized queries or prepared statements to prevent SQL injection attacks by properly escaping user input before executing SQL queries.
- Utilize stored procedures or ORM frameworks to abstract database interactions and minimize direct SQL query execution.
- Enforce principle of least privilege by restricting database access and permissions to only necessary operations and resources.
- Regularly update and patch the application and database management system (DBMS) to address known vulnerabilities and improve security posture.
- Conduct thorough security assessments, including penetration testing and code reviews, to identify and remediate SQL injection vulnerabilities.
- Implement web application firewalls (WAFs) or runtime application self-protection (RASP) solutions to provide additional layers of defense against SQL injection and other injection-based attacks.

---

**Name:** ID03: Local File Inclusion (LFI) to WEB-INF Init Logs in **Home Page**

**Test Severity:** **High**
**Test Score: 8.6**

**Description:**
Local File Inclusion (LFI) vulnerability was discovered in the home page. This vulnerability allows an attacker to include arbitrary files located within the WEB-INF directory, such as initialization logs or configuration files, which may contain sensitive information.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized access to sensitive information, including credentials, session tokens, or other confidential data stored in initialization logs or configuration files.

**Recommendations:**

- Review and restrict access permissions to sensitive files and directories, such as WEB-INF, to prevent unauthorized access.
- Implement input validation and output encoding to sanitize user-supplied input and prevent directory traversal attacks.
- Configure the web server or application container to disallow direct access to files within the WEB-INF directory.
- Monitor file system access and log activities to detect and respond to potential exploitation attempts.
- Regularly review and update file system permissions and configurations to minimize the attack surface and mitigate potential vulnerabilities.

---

**Name:** ID04: SQL Injection in **Bank Apply to Gold Visa**

**Test Severity:** Critical
**Test Score: 9.2**

**Description:**
SQL Injection vulnerability was discovered in the bank apply functionality for the Gold Visa card. This vulnerability allows an attacker to manipulate SQL

queries executed by the application when processing requests related to applying for the Gold Visa card, potentially gaining unauthorized access to sensitive data or performing malicious actions on the database.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized access to sensitive information stored in the database, including personally identifiable information (PII), financial records, or other confidential data, as well as potential data manipulation or deletion.

**Recommendations:**

- Implement parameterized queries or prepared statements to prevent SQL injection attacks by properly escaping user input before executing SQL queries.
- Utilize ORM frameworks or stored procedures to abstract database interactions and minimize direct SQL query execution.
- Enforce principle of least privilege by restricting database access and permissions to only necessary operations and resources.
- Regularly update and patch the application and database management system (DBMS) to address known vulnerabilities and improve security posture.
- Conduct thorough security assessments, including penetration testing and code reviews, to identify and remediate SQL injection vulnerabilities.

---

**Name:** ID05: Insecure Direct Object Reference (IDOR) in **Show Account History**

**Test Severity:** Critical
**Test Score: 9.3**

**Description:**
Insecure Direct Object Reference (IDOR) vulnerability was discovered in the "show account history" functionality. This vulnerability allows an attacker to manipulate input parameters or directly access resources, such as account records or transactions belonging to other users, which are not properly protected or authorized.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized access to sensitive information, including account balances, transaction details, or personally identifiable information (PII) of other users, as well as potential data manipulation or deletion.

**Recommendations:**

- Implement proper access controls and authorization mechanisms to restrict access to sensitive resources based on user privileges and ownership.
- Utilize indirect object references or opaque identifiers to prevent direct access to internal data structures or database records.
- Implement role-based access control (RBAC) or attribute-based access control (ABAC) to enforce fine-grained access policies and minimize the risk of IDOR vulnerabilities.
- Regularly review and update access control configurations and policies to ensure they align with security requirements and best practices.
- Conduct thorough security assessments, including code reviews and penetration testing, to identify and remediate IDOR vulnerabilities and other authorization issues.
- Implement logging and monitoring to detect and respond to unauthorized access attempts or suspicious activities related to sensitive resources.

---

**Name:** ID06: Improper Access Control Leading to **Admin Pages**

**Test Severity:** Critical
**Test Score: 9.3**

**Description:**
Improper Access Control vulnerability was discovered, allowing unauthorized users to access admin pages or functionalities that should be restricted to privileged users. This vulnerability may arise due to insufficient access controls, misconfigured permissions, or lack of proper authentication mechanisms.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized access to sensitive administrative functionalities, potentially allowing attackers to manipulate system configurations, access confidential information, or perform malicious actions that compromise the security and integrity of the application.

**Recommendations:**

- Review and update access control configurations to enforce proper authentication and authorization mechanisms, ensuring that only authenticated and authorized users can access admin pages or functionalities.
- Implement role-based access control (RBAC) or attribute-based access control (ABAC) to define and enforce fine-grained access policies based on user roles, privileges, and attributes.
- Conduct thorough security assessments, including access control reviews and penetration testing, to identify and remediate improper access control vulnerabilities and misconfigurations.
- Implement multi-factor authentication (MFA) and session management controls to enhance the security of authentication processes and prevent unauthorized access to sensitive functionalities.

---

**Name:** ID07: Cross-Site Request Forgery (CSRF) in **Transfer Funds**

**Test Severity:** High
**Test Score: 8.6**

**Description:**
Cross-Site Request Forgery (CSRF) vulnerability was discovered in the transfer funds functionality. This vulnerability allows an attacker to trick authenticated users into executing unauthorized transactions, such as transferring funds or making changes to account settings, by exploiting the trust established between the user's browser and the application.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized fund transfers, financial loss, or manipulation of account

balances, compromising the integrity and security of the user's financial transactions.

**Recommendations:**

- Implement CSRF protection mechanisms, such as anti-CSRF tokens or same-site cookie attributes, to prevent unauthorized requests from being executed by authenticated users.
- Validate and verify the integrity of incoming requests, including checking the origin and referer headers, to detect and block potential CSRF attacks.
- Utilize state-changing HTTP methods, such as POST, for sensitive operations like fund transfers, and enforce proper authorization checks to ensure that only authorized users can perform such actions.
- Conduct security reviews and penetration testing regularly to identify and address CSRF vulnerabilities and other potential security risks.
- Implement additional security measures, such as Content Security Policy (CSP) headers or security headers, to mitigate the risk of CSRF attacks and protect against other web-based threats.

---

**Name:** ID08: Cross-Site Request Forgery (CSRF) in **Adding a New User by Admin**

**Test Severity:** Medium
**Test Score: 6.9**

**Description:**
Cross-Site Request Forgery (CSRF) vulnerability was discovered in the functionality for adding a new user by an admin. This vulnerability allows an attacker to trick the admin user into unintentionally creating a new user account with arbitrary privileges or permissions by exploiting the admin's session and authority within the application.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized creation of user accounts with elevated privileges, such as administrative roles, compromising the security and integrity of the application's user management system.

**Recommendations:**

- Implement CSRF protection mechanisms, such as anti-CSRF tokens or same-site cookie attributes, to prevent unauthorized requests from being executed by authenticated users.
- Validate and verify the integrity of incoming requests, including checking the origin and referer headers, to detect and block potential CSRF attacks.
- Utilize state-changing HTTP methods, such as POST, for sensitive operations like adding new users, and enforce proper authorization checks to ensure that only authorized users can perform such actions.
- Implement additional confirmation mechanisms, such as CAPTCHA challenges or multi-step verification processes, for critical operations to mitigate the risk of CSRF attacks and prevent unauthorized actions.
- Implement additional security measures, such as Content Security Policy (CSP) headers or security headers, to mitigate the risk of CSRF attacks and protect against other web-based threats.

---

**Name:** ID09: Cross-Site Request Forgery (CSRF) in **Adding Account to an Existing User by Admin**

**Test Severity: Critical**
**Test Score: 9.3**

**Description:**
Cross-Site Request Forgery (CSRF) vulnerability was discovered in the functionality for adding an account to an existing user by an admin. This vulnerability allows an attacker to trick the admin user into unintentionally adding an account to an existing user with arbitrary permissions or privileges by exploiting the admin's session and authority within the application.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized addition of accounts to existing users with elevated privileges, such as administrative roles, compromising the security and integrity of the application's user management system.

**Recommendations:**

- Implement CSRF protection mechanisms, such as anti-CSRF tokens or same-site cookie attributes, to prevent unauthorized requests from being executed by authenticated users.
- Utilize state-changing HTTP methods, such as POST, for sensitive operations like adding accounts to existing users, and enforce proper authorization checks to ensure that only authorized users can perform such actions.
- Implement additional confirmation mechanisms, such as CAPTCHA challenges or multi-step verification processes, for critical operations to mitigate the risk of CSRF attacks and prevent unauthorized actions.
- Implement additional security measures, such as Content Security Policy (CSP) headers or security headers, to mitigate the risk of CSRF attacks and protect against other web-based threats.

---

**Name:** ID10: Cross-Site Request Forgery (CSRF) in **Change User's Password by Admin**

**Test Severity:** High
**Test Score: 8.2**

**Description:**
This vulnerability allows an attacker to forge a request to change a user's password by exploiting the admin privileges without the user's consent. CSRF attacks can trick an authenticated admin into executing unintended actions, such as changing a user's password, by disguising malicious requests as legitimate ones.

**Impact:**
The impact of this vulnerability is severe; successful exploitation allows an attacker to change a user's password without their knowledge or consent, compromising their account security and potentially leading to unauthorized access to sensitive information or actions.

**Recommendations:**

- Implement CSRF protection mechanisms such as unique tokens for each request.

- Require re-authentication for sensitive actions, particularly those with elevated privileges.
- Validate the origin of requests to ensure they originate from trusted sources.
- Enforce strict access controls to limit the actions that can be performed by admin users.
- Ensure that users can only modify their own passwords through secure, authenticated channels.

---

**Name:** ID11: HTML Injection in the **Home Page**

**Test Severity:** Medium
**Test Score: 4.8**

**Description:**
HTML Injection vulnerability was discovered on the home page of the application. This vulnerability allows an attacker to inject malicious HTML code into the home page, potentially leading to various attacks such as cross-site scripting (XSS), phishing, or defacement of the page.

**Impact:**
The impact of this vulnerability is moderate. Successful exploitation could lead to unauthorized code execution within users' browsers, compromising their session data, stealing credentials, or redirecting them to malicious websites.

**Recommendations:**

- Regularly sanitize and validate user input to prevent the injection of malicious HTML code.
- Implement Content Security Policy (CSP) headers to restrict the sources of content that can be loaded on the page.
- Encode user-supplied content to HTML entities to prevent it from being interpreted as HTML code.
- Conduct security reviews and penetration testing regularly to identify and address vulnerabilities proactively.

- Educate developers about secure coding practices, particularly regarding input validation and output encoding.

---

**Name:** ID12: Cross-Site Scripting (XSS) in **Home Page Search**

**Test Severity:** High
**Test Score: 8.3**

**Description:**
Cross-Site Scripting (XSS) vulnerability was discovered in the search functionality of the home page. This vulnerability allows an attacker to inject malicious scripts into the search input field, which can be executed within the context of other users' browsers when they perform a search.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to various attacks, including session hijacking, credential theft, defacement of the page, or the execution of arbitrary actions on behalf of the user.

**Recommendations:**

- Implement input validation and output encoding to sanitize user-supplied input and prevent the injection of malicious scripts.
- Use appropriate encoding mechanisms (such as HTML entity encoding or JavaScript escaping) to neutralize user input before rendering it in the browser.
- Apply Content Security Policy (CSP) headers to restrict the sources of scripts that can be executed on the page.
- Conduct security reviews and penetration testing regularly to identify and address XSS vulnerabilities.
- Educate developers about secure coding practices, particularly regarding input validation and output encoding, and provide training on the risks associated with XSS attacks.

---

**Name:** ID13: Reflected Cross-Site Scripting (RXSS) in **Bank Search Articles**

**Test Severity:** High
**Test Score: 8.8**

**Description:**
Reflected Cross-Site Scripting (RXSS) vulnerability was discovered in the bank search articles feature. This vulnerability allows an attacker to inject malicious scripts into the URL parameters used for searching articles, which get reflected back in the response, potentially leading to script execution within the context of other users' browsers.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to various attacks, including session hijacking, credential theft, defacement of the page, or the execution of arbitrary actions on behalf of the user.

**Recommendations:**

- Implement input validation and output encoding to sanitize user-supplied input and prevent the injection of malicious scripts.
- Use appropriate encoding mechanisms (such as URL encoding) to neutralize user input before using it in dynamic content.
- Apply Content Security Policy (CSP) headers to restrict the sources of scripts that can be executed on the page.
- Conduct security reviews and penetration testing regularly to identify and address RXSS vulnerabilities.
- Educate developers about secure coding practices, particularly regarding input validation and output encoding, and provide training on the risks associated with RXSS attacks.

---

**Name:** ID14: Reflected Cross-Site Scripting (RXSS) in **Customize Language**

**Test Severity: High**
**Test Score: 8.5**

**Description:**
A Reflected Cross-Site Scripting (RXSS) vulnerability was discovered in the customize language feature. This vulnerability allows an attacker to inject malicious scripts into the parameters used for customizing language settings, which get reflected back in the response, potentially leading to script execution within the context of other users' browsers.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to various attacks, including session hijacking, credential theft, defacement of the page, or the execution of arbitrary actions on behalf of the user.

**Recommendations:**

- Implement input validation and output encoding to sanitize user-supplied input and prevent the injection of malicious scripts.
- Use appropriate encoding mechanisms (such as HTML entity encoding) to neutralize user input before using it in dynamic content.
- Apply Content Security Policy (CSP) headers to restrict the sources of scripts that can be executed on the page.
- Conduct security reviews and penetration testing regularly to identify and address RXSS vulnerabilities.
- Educate developers about secure coding practices, particularly regarding input validation and output encoding, and provide training on the risks associated with RXSS attacks.

---

**Name:** ID15: Post-Based Cross-Site Scripting (PXSS) in **Feedback**

**Test Severity: High**
**Test Score: 8.5**

**Description:**
Post-Based Cross-Site Scripting (PXSS) vulnerability was discovered in the feedback feature. This vulnerability allows an attacker to inject malicious scripts into the feedback submission form, which get stored and reflected in the response, potentially leading to script execution within the context of other users' browsers.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to various attacks, including session hijacking, credential theft, defacement of the page, or executing arbitrary actions on behalf of the user.

**Recommendations:**

- Implement input validation and output encoding to sanitize user-supplied input and prevent the injection of malicious scripts.
- Use appropriate encoding mechanisms (such as HTML entity encoding) to neutralize user input before storing and rendering it in dynamic content.
- Apply strict content security policies to prevent the execution of untrusted scripts.
- Implement rate limiting and CAPTCHA mechanisms to mitigate automated attacks on feedback submission forms.
- Conduct regular security reviews and penetration testing to identify and address PXSS vulnerabilities.
- Educate developers about secure coding practices, particularly regarding input validation and output encoding, and provide training on the risks associated with PXSS attacks.

---

**Name:** ID16: URL Redirection in **Customize Language**

**Test Severity:** High
**Test Score: 8.3**

**Description:**
URL Redirection vulnerability was discovered in the customize language feature. This vulnerability allows an attacker to manipulate the URL parameters for customizing language settings to redirect users to malicious websites or phishing pages.

**Impact:**
The impact of this vulnerability is moderate. Successful exploitation could lead to unauthorized redirection of users to malicious websites, potentially resulting in phishing attacks, the theft of sensitive information, or the installation of malware.

**Recommendations:**

- Implement whitelist-based validation of redirection URLs to restrict redirection to trusted domains.
- Avoid passing user-controlled input directly into the redirection URL.
- Use a safe redirection method that does not rely on user-supplied input.
- Educate users about the risks associated with unexpected redirects and advise them to verify URLs before proceeding.
- Monitor and log redirection attempts to detect and mitigate suspicious activities.
- Conduct regular security reviews and penetration testing to identify and address URL redirection vulnerabilities.
- Stay informed about the latest security best practices and consider implementing security headers like Content Security Policy (CSP) to mitigate the risk of such vulnerabilities.

---

**Name:** ID17: Clickjacking Vulnerability

**Test Severity:** Medium
**Test Score: 5.7**

**Description:**
A clickjacking vulnerability was discovered in the application. This vulnerability allows an attacker to embed the application in a malicious website or iframe, tricking users into performing unintended actions by clicking on seemingly harmless elements that are layered on top of the application.

**Impact:**
The impact of this vulnerability is severe. Successful exploitation could lead to various attacks, including unauthorized actions performed by the user without their knowledge, such as clicking on disguised buttons or links that perform malicious actions within the application.

**Recommendations:**

- Implement X-Frame-Options HTTP header with the value "DENY" or "SAMEORIGIN" to prevent the application from being embedded in iframes on other domains.
- Utilize the Content-Security-Policy (CSP) frame-ancestors directive to restrict which domains can embed the application in iframes.
- Implement a frame-busting script to prevent the application from being framed by other websites.
- Educate users about the risks of clickjacking and advise them to be cautious when interacting with unfamiliar or suspicious elements on web pages.
- Regularly review and update security headers and policies to mitigate emerging threats and vulnerabilities.
- Conduct security reviews and penetration testing regularly to identify and address clickjacking vulnerabilities.
- Stay informed about the latest security best practices and consider implementing additional security measures, such as JavaScript-based protection mechanisms, to mitigate the risk of clickjacking attacks further.

---

**Name:** ID18: HTML Injection in **Home Page Search**

**Test Severity:** Low
**Test Score: 2.1**

**Description:**
HTML Injection vulnerability was discovered in the search functionality of the home page. This vulnerability allows an attacker to inject arbitrary HTML code into the search input field, which may be reflected in other users' browsers, potentially leading to various attacks such as Cross-Site Scripting (XSS), phishing, or defacement of the page.

**Impact:**
The impact of this vulnerability is moderate. Successful exploitation could lead to unauthorized execution of HTML or JavaScript code within users' browsers, potentially compromising their session data, stealing credentials, or redirecting them to malicious websites.

**Recommendations:**

- Implement strict input validation to sanitize user-supplied input and prevent the injection of HTML code.
- Encode user-generated content and dynamically generated HTML to prevent it from being interpreted as HTML code by the browser.
- Utilize appropriate output encoding mechanisms (such as HTML entity encoding) to neutralize user input before rendering it in the browser.
- Implement Content Security Policy (CSP) headers to restrict the sources of content that can be loaded on the page.
- Conduct regular security reviews and penetration testing to identify and address HTML injection vulnerabilities.
- Educate developers about secure coding practices, particularly regarding input validation and output encoding, and provide training on the risks associated with HTML injection attacks.

---

**Name:** ID19: Information Disclosure (Sensitive Data Exposure) in **login.jsp Code**

**Test Severity:** High
**Test Score: 8.5**

**Description:**
Information Disclosure (Sensitive Data Exposure) vulnerability was discovered in the code of the login.jsp file. This vulnerability may expose sensitive information, such as plaintext passwords, session tokens, or other confidential data, directly in the source code or through improper error handling.

**Impact:**

The impact of this vulnerability is severe. Successful exploitation could lead to unauthorized access to sensitive information, compromising user accounts, and privacy, and potentially leading to identity theft or unauthorized access to protected resources.

**Recommendations:**

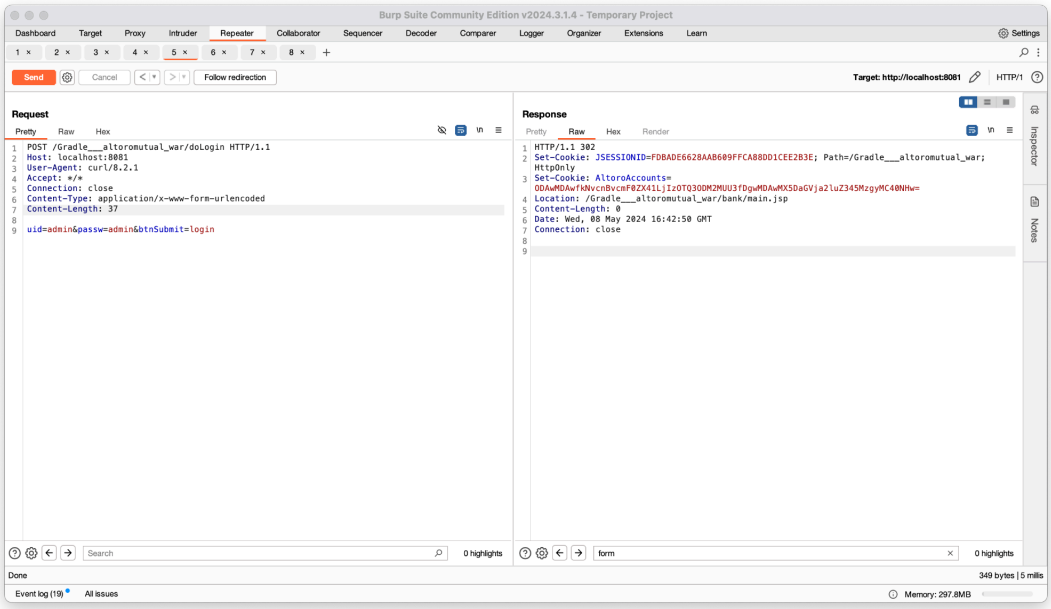
- Review the code of login.jsp and ensure that sensitive information is not directly exposed or stored in plaintext.
- Implement secure password storage mechanisms, such as hashing with salt, to protect user passwords from exposure during a breach.
- Utilize secure communication protocols (e.g., HTTPS) to encrypt sensitive data transmitted between the client and server.
- Implement proper error handling and response mechanisms to prevent the leaking of sensitive information in error messages or logs.
- Regularly review and update the codebase to address potential vulnerabilities and improve security practices.

---

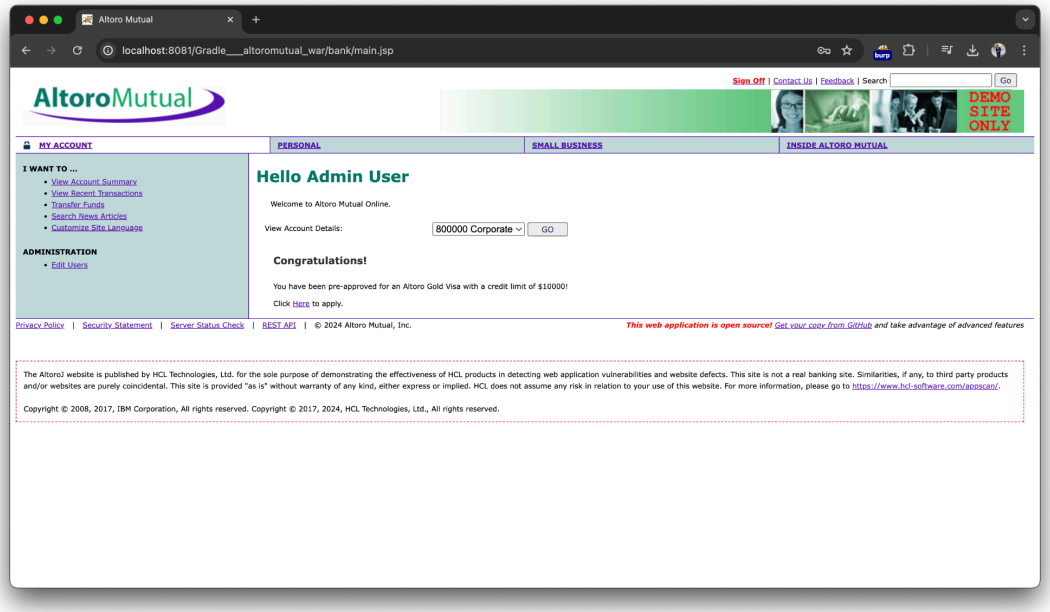
# Finding scenarios

## ID01

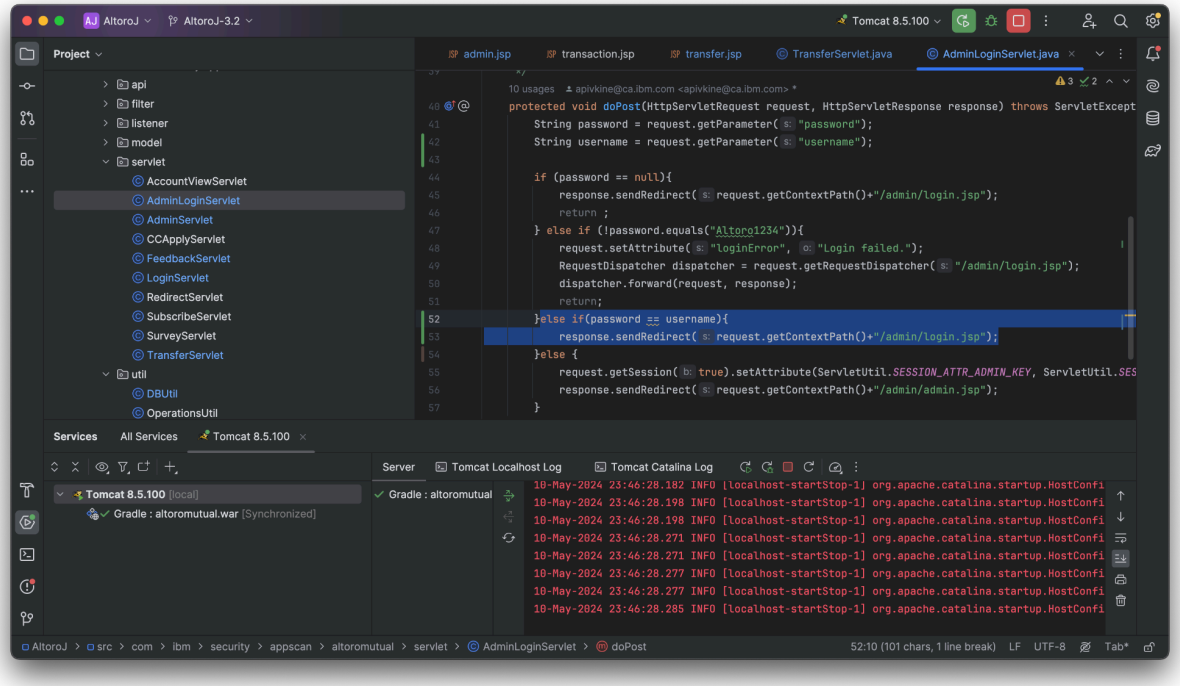
- Finding test steps
  - Go to the login page in doLogin function
  - Use Burp Suite to intercept the request with the following payload(default credentials): **uid=admin&passw=admin&btnSubmit=login**




  - Whoo! It logged in to the admin pages



- Fixing steps
  - Go to LoginServlet in the source code
  - if username equals to password it will be false

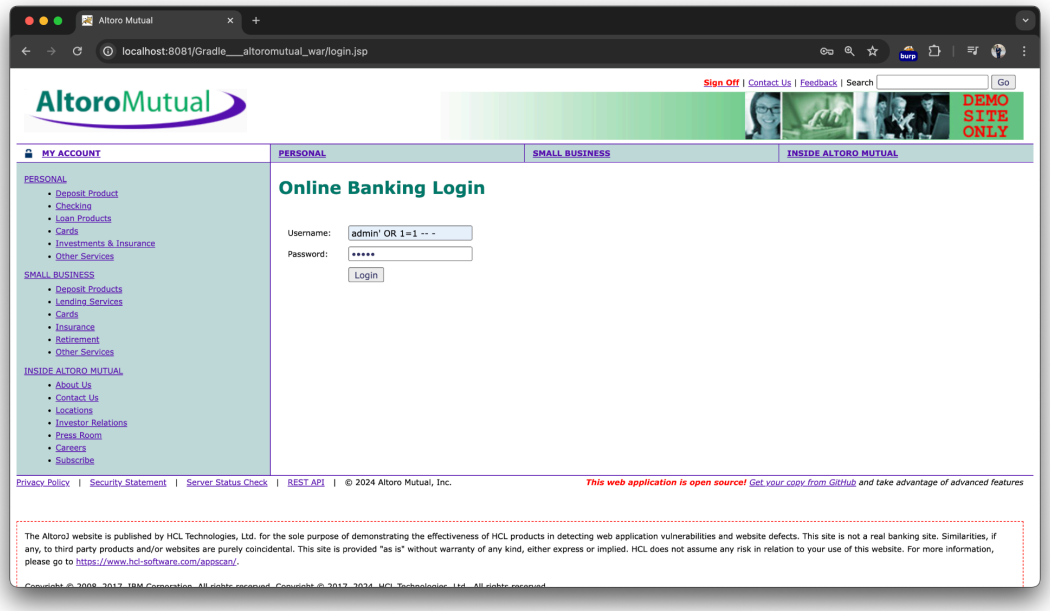



  - It is a simple solution but it is must prevent from the admins from adding such default credentials
- Finding RE-TEST steps
  - Log in with these credentials and will fail 😀

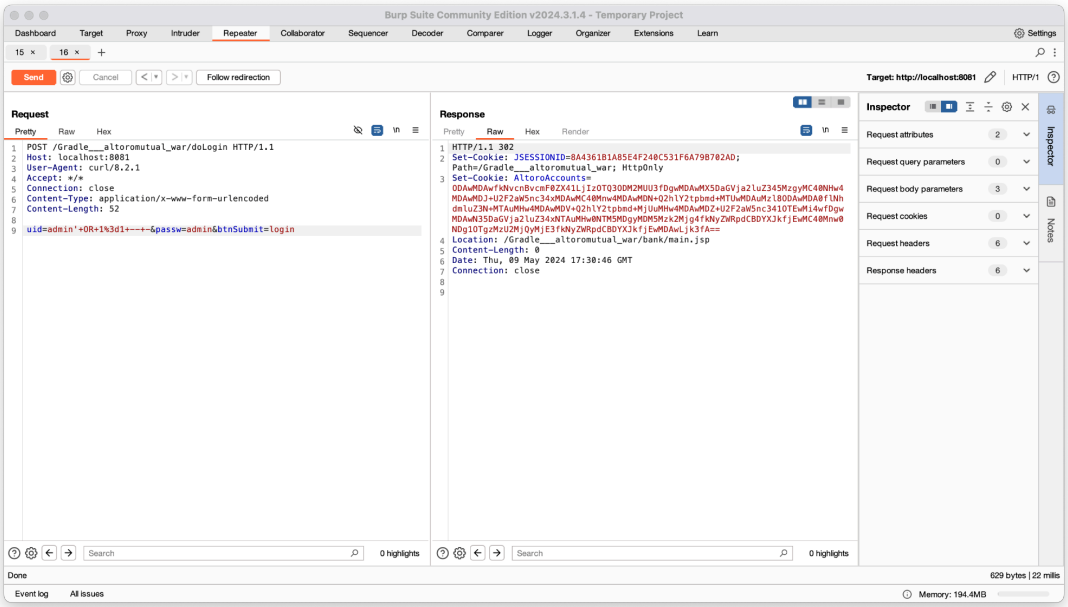
## ID02

- Finding test steps

- Go to the login page and try to make SQL injection on username input

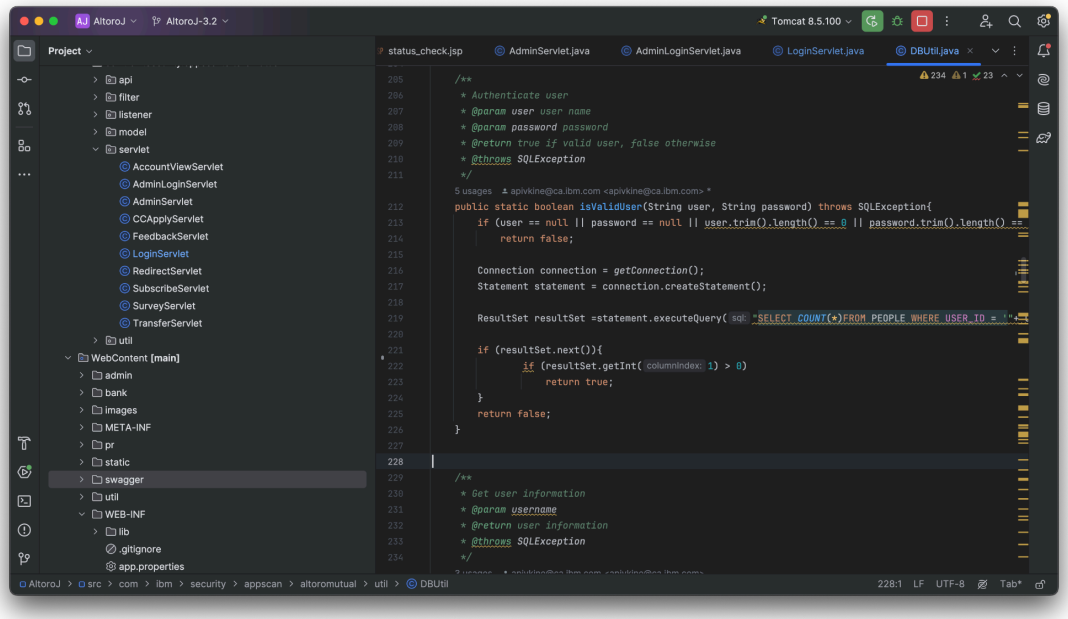



- Use BurbSuite to intercept the request with the payload: uid=admin'+0R+1%3d1+--+-&passw=admin&btnSubmit=login

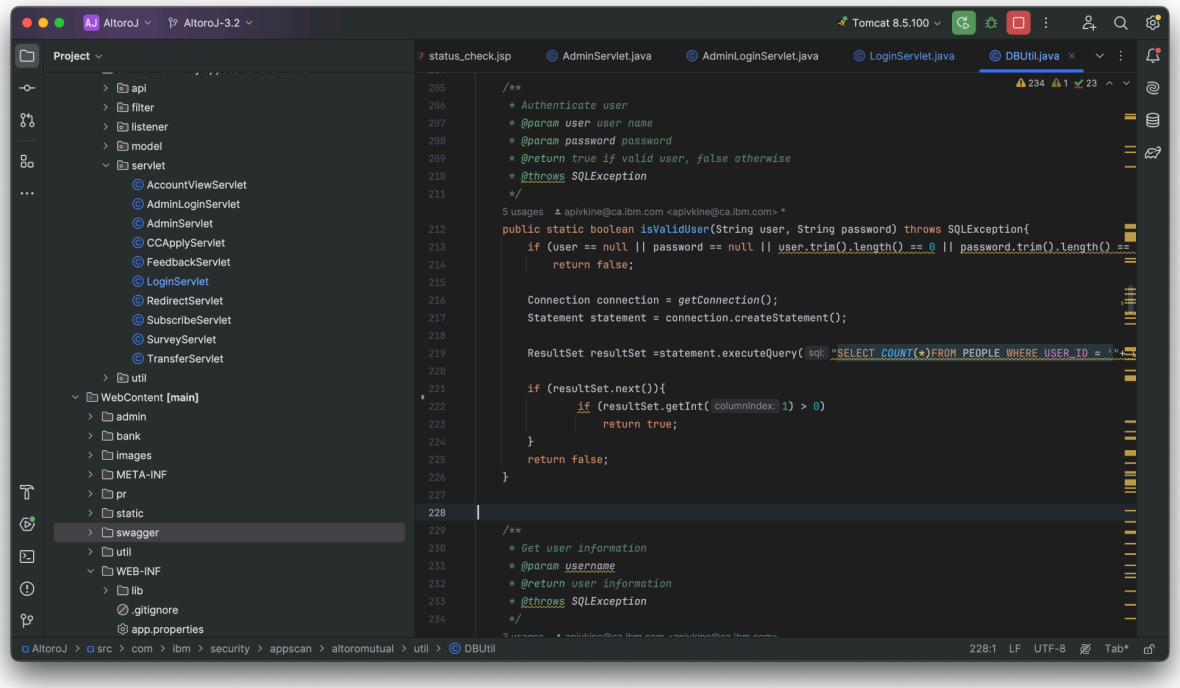



- Accessed to main pages and set cookies for user identity

- Fixing steps
  - Go to DBUtlis in isValidUser function
  - The input username and password are not sanitized !!

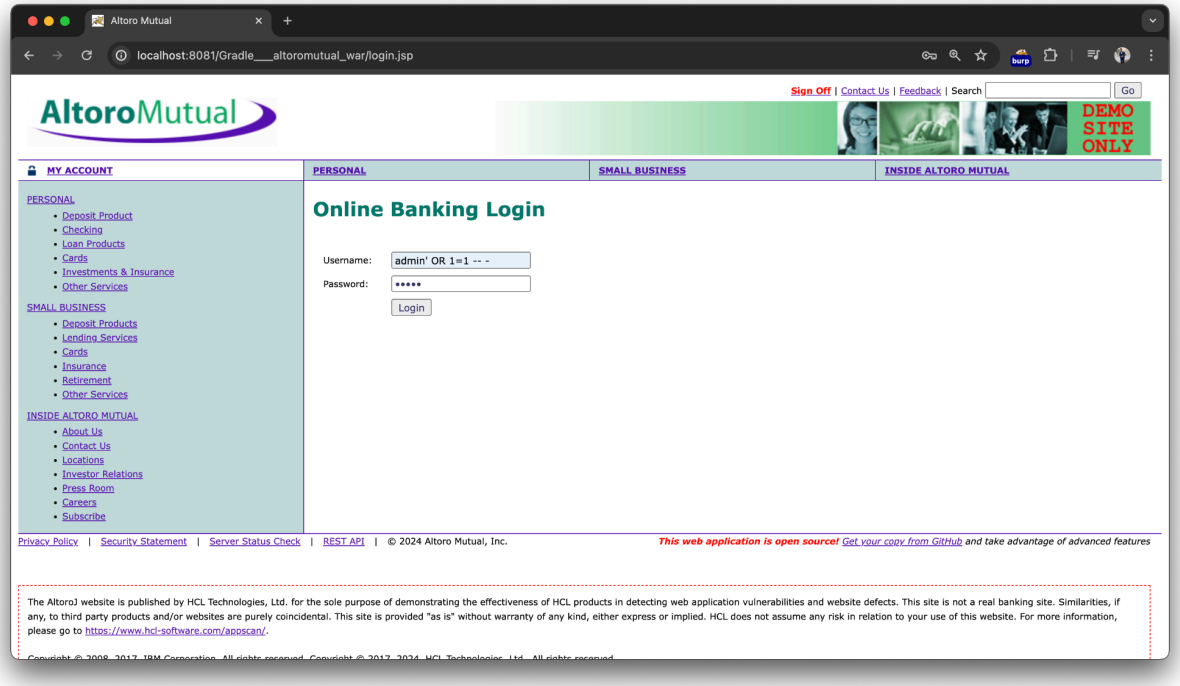



- Implement prepared  statement to prevent SQL injection

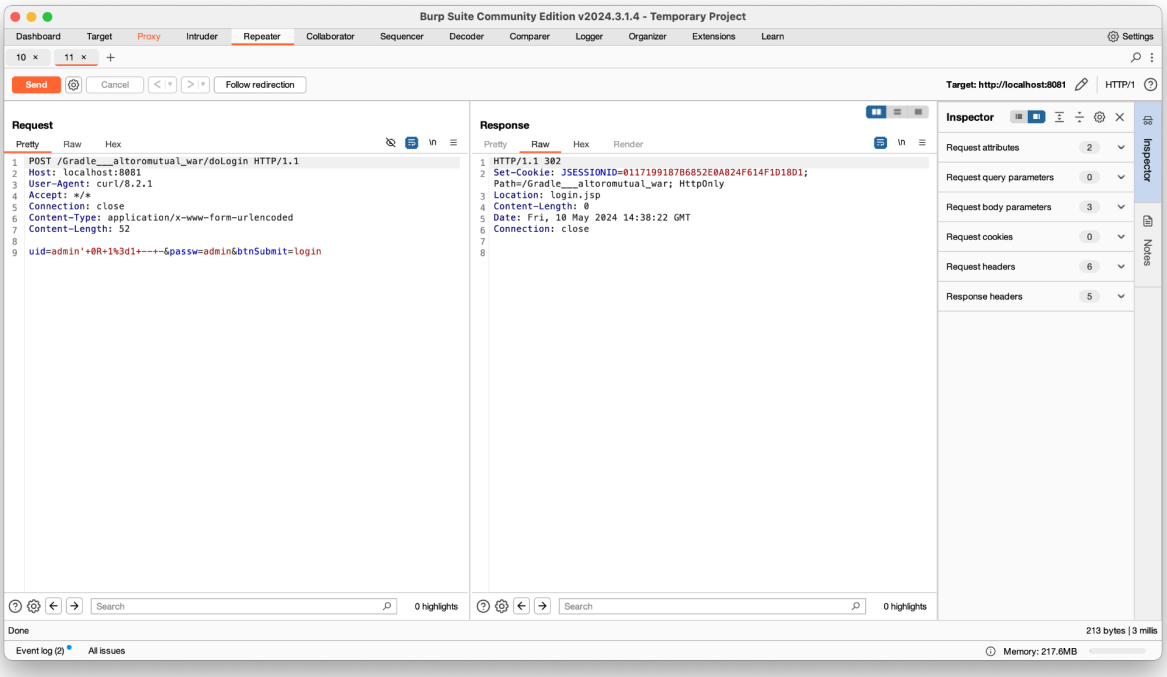



- Finding RE-TEST steps

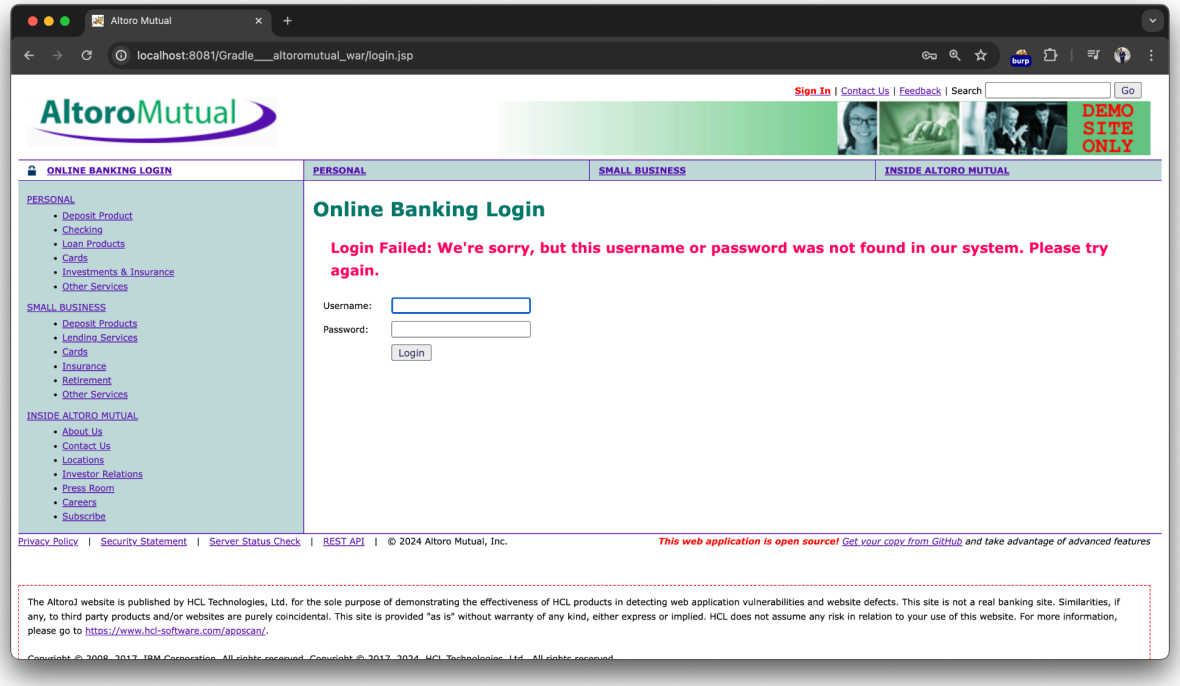
- Go to login page with same payload




- Use BurbSuite with the same payload on the same request



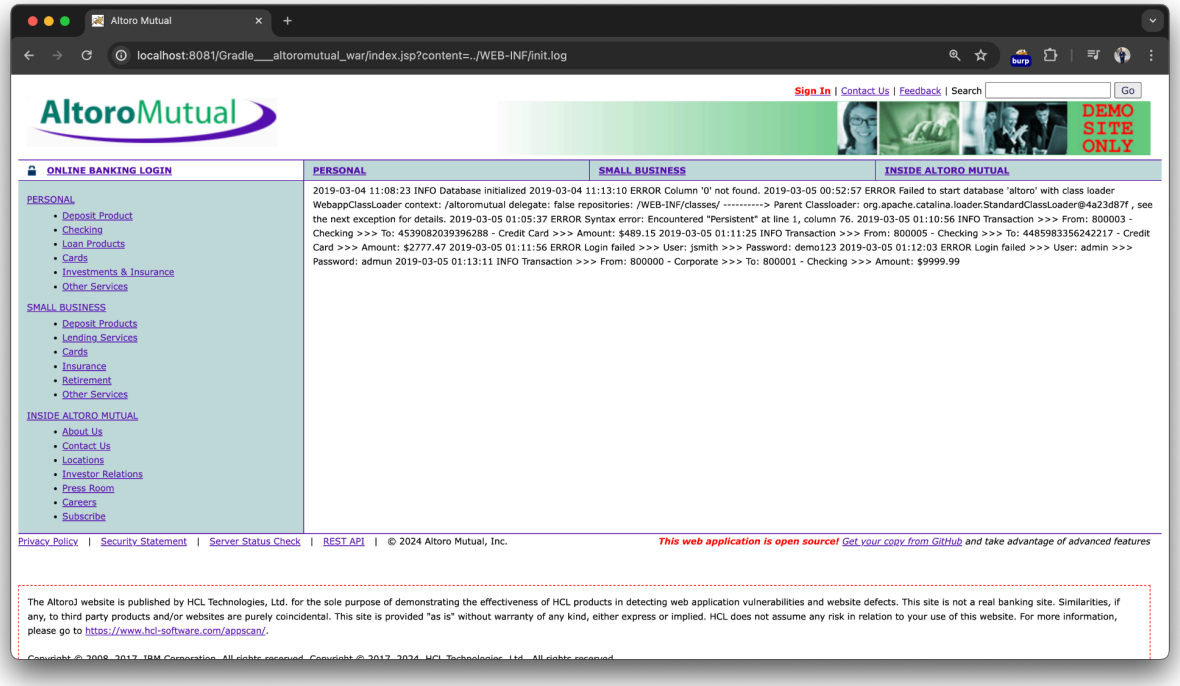
- Prevented the user from proceeding to protected routes!!



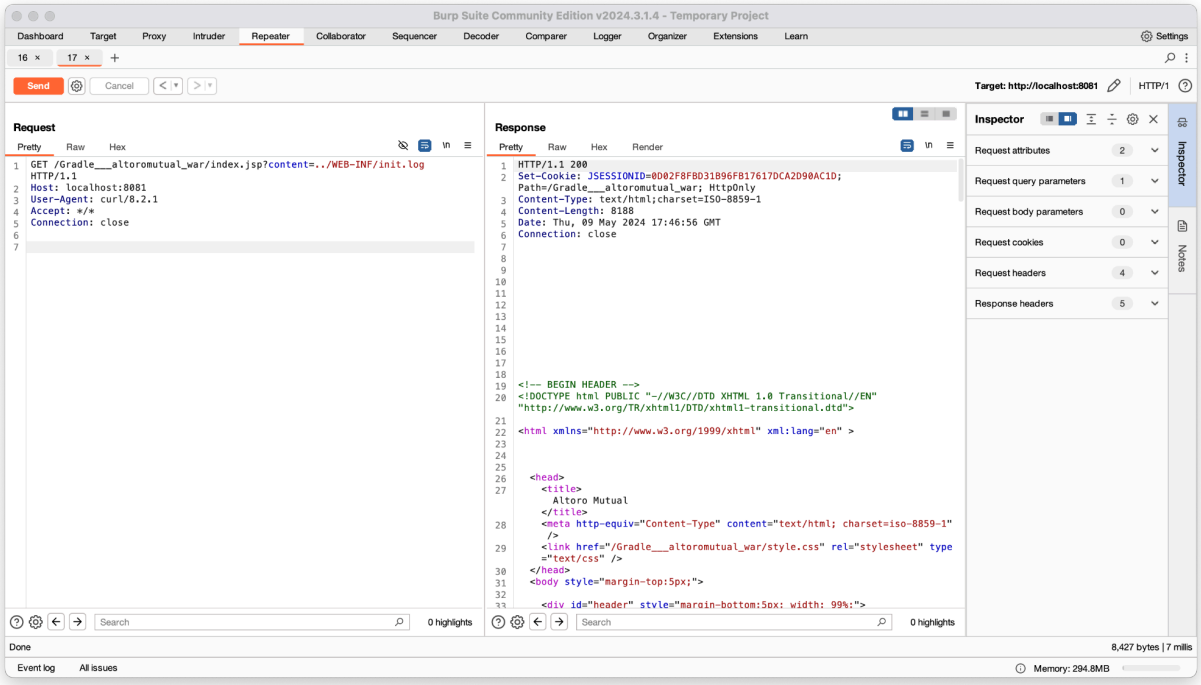

## ID03

- Finding test steps
  - Go to index and route to any subroute

- change the content parameter to “../WEB-INF/init.log”




- It gets the log file from the server ❌
- Use BurbSuite to intercept the request

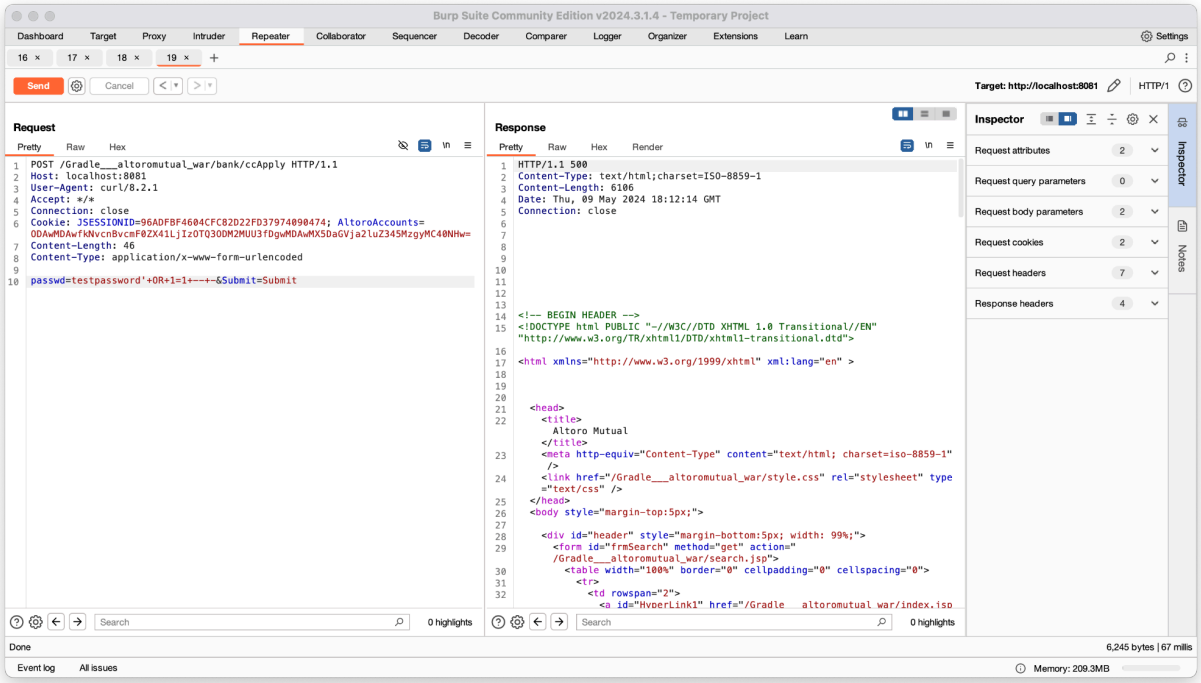



- Fixing steps
  -

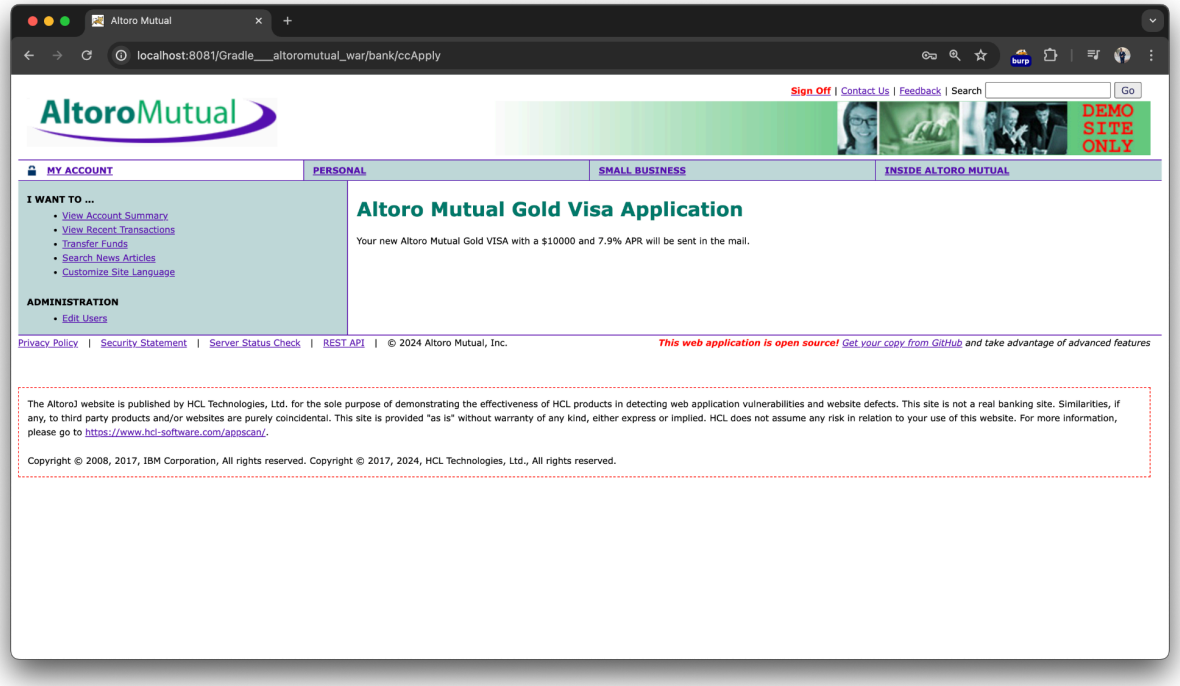
- Finding RE-TEST steps

## ID04

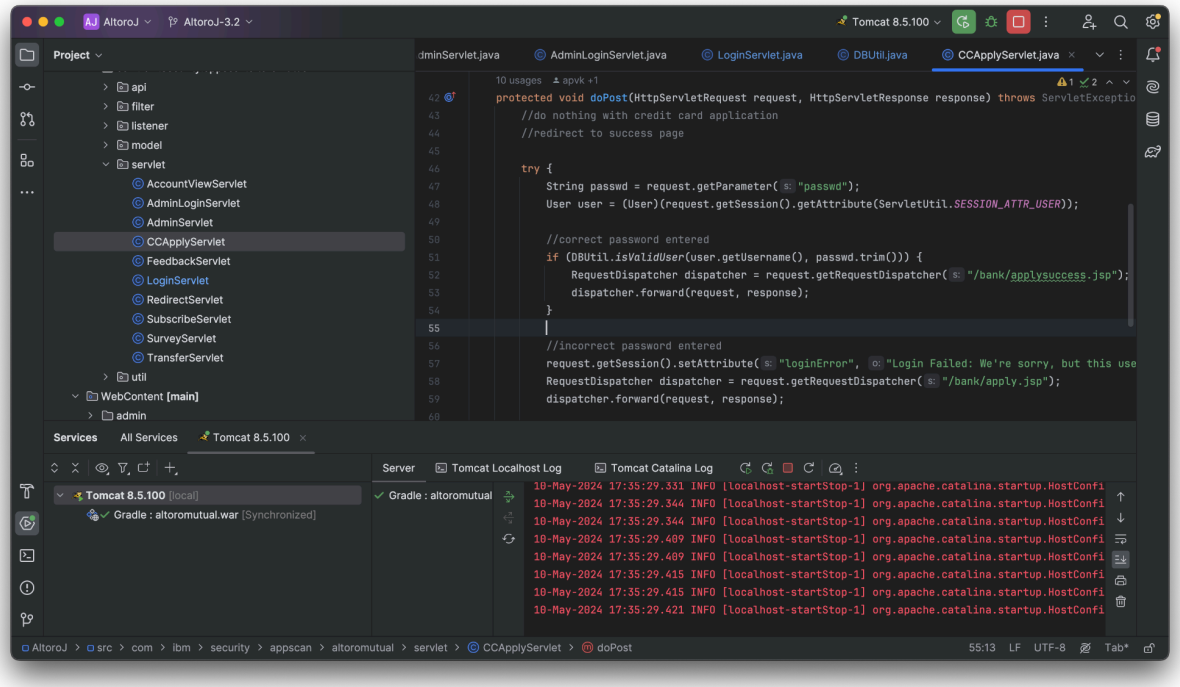
- Finding test steps
  - Go to credit card gold visa apply
  - Use BurbSuite to intercept the request with payload:passwd=testpassword'+OR+1=1+--+-&Submit=Submit



- SQL injection was detected and proceeded to the requested functionality

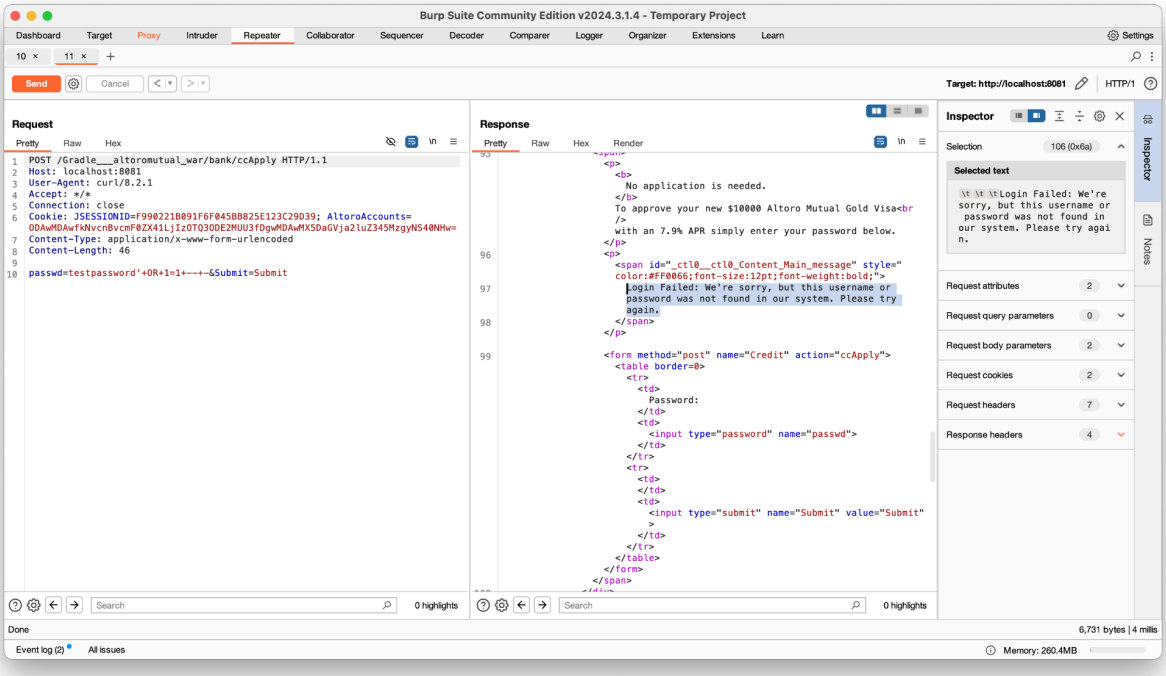



- Fixing steps
  - Go to CCApplyServlet

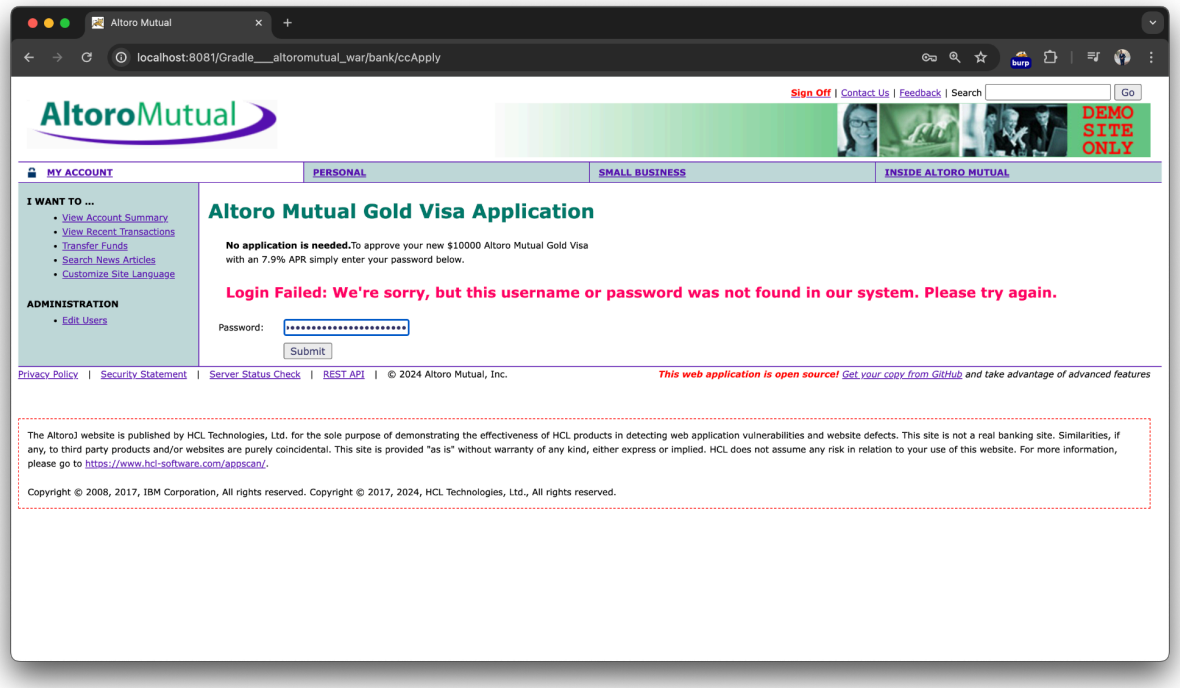


- This servlet uses the same isValidUser function in DBUtils that has resolved SQL injection vulnerability in ID02

- Finding RE-TEST steps
  - Use BurbSuite to intercept the request with the same payload

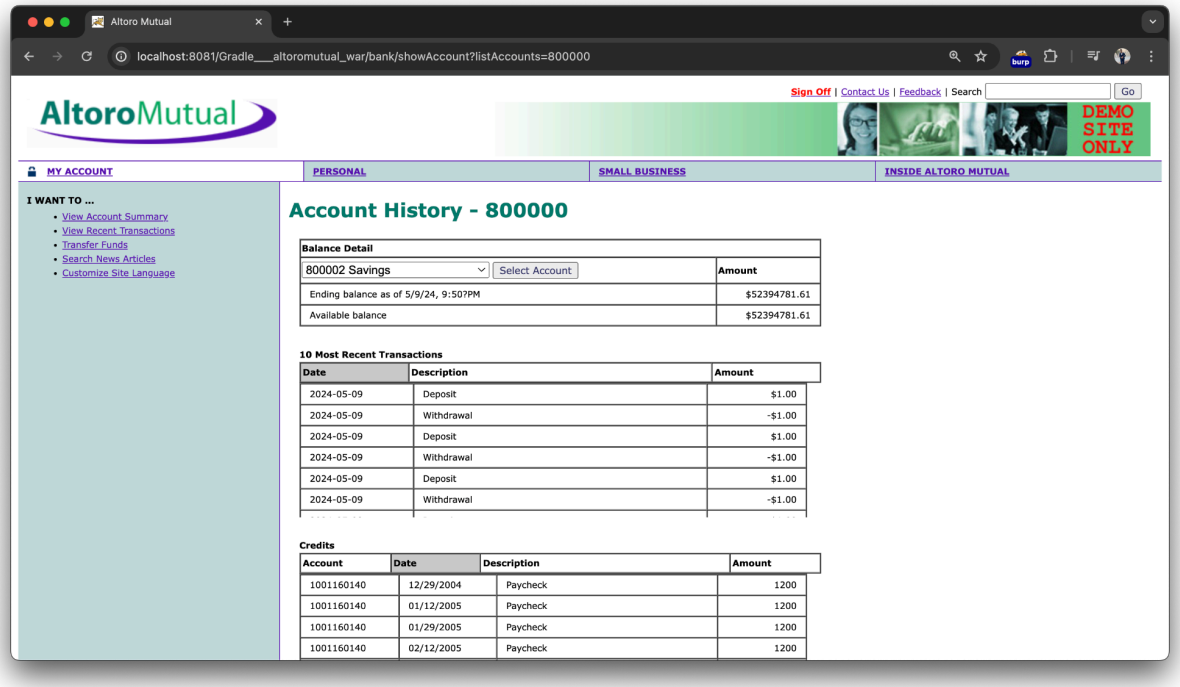



- Blocked from applying for Gold Visa credit card 😀



## ID05

- Finding test steps
  - Go sign in as a normal user
  - Go to account summary
  - Change the route to “bank/showAccount?listAccounts=800000”

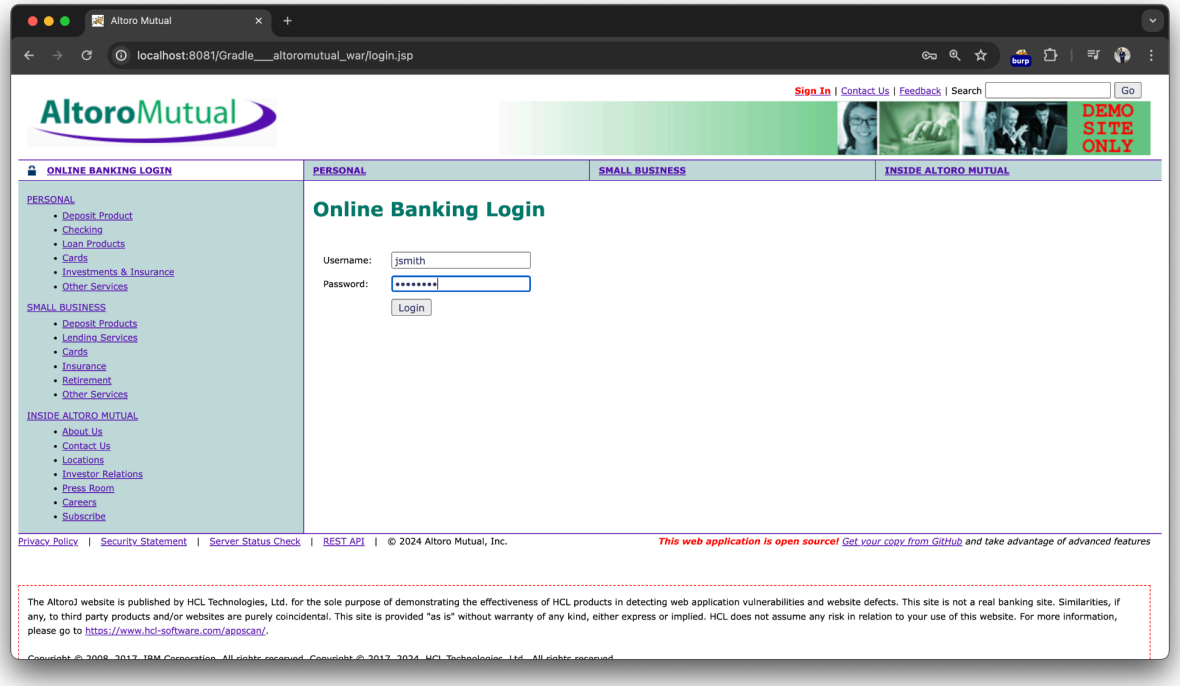



  - Accessed admin bank account 😣
- Fixing steps
  -
- Finding RE-TEST steps

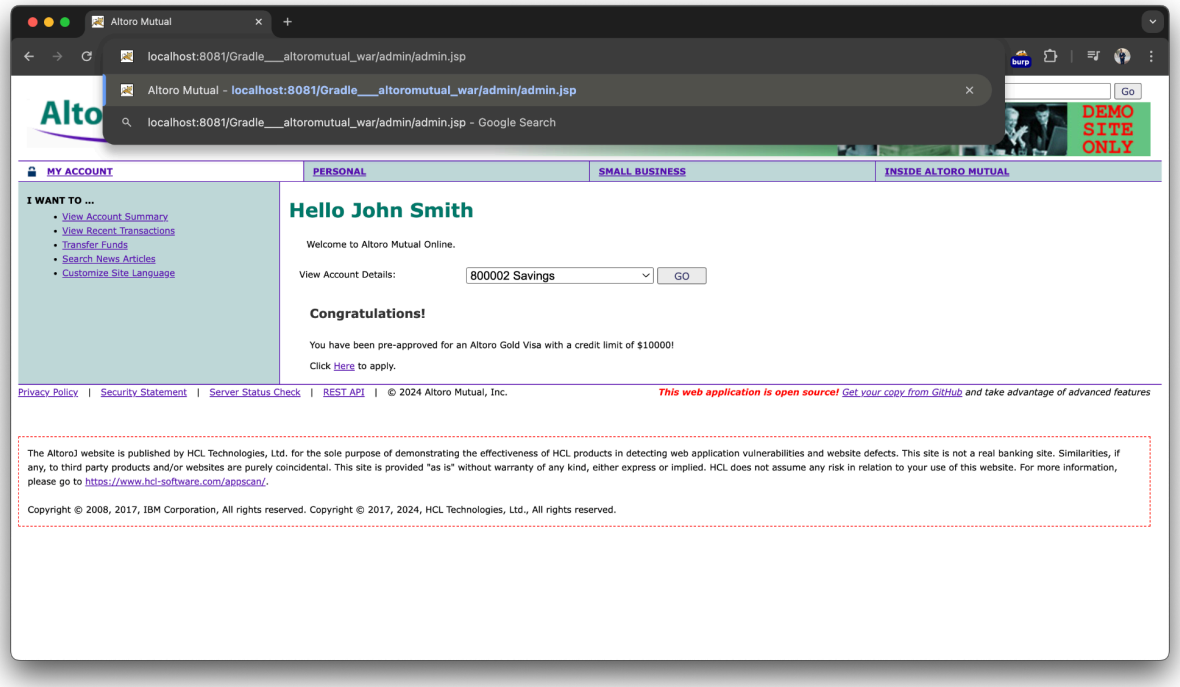
## ID06

- Finding test steps

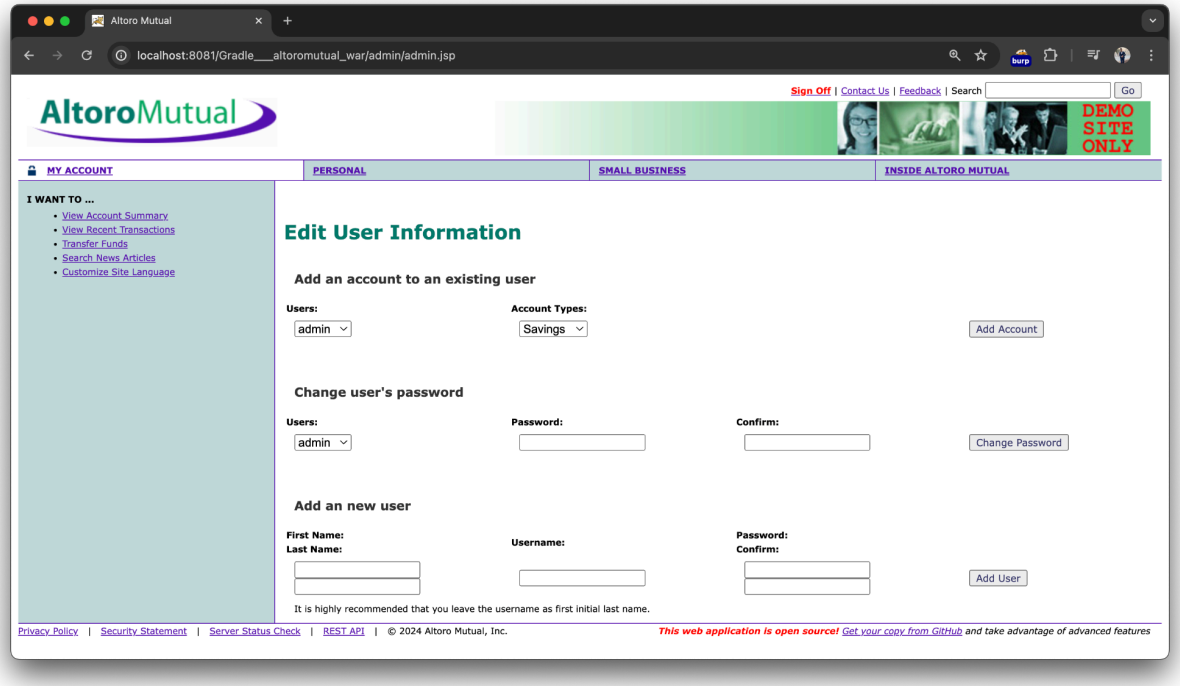
- Sign in as a normal user



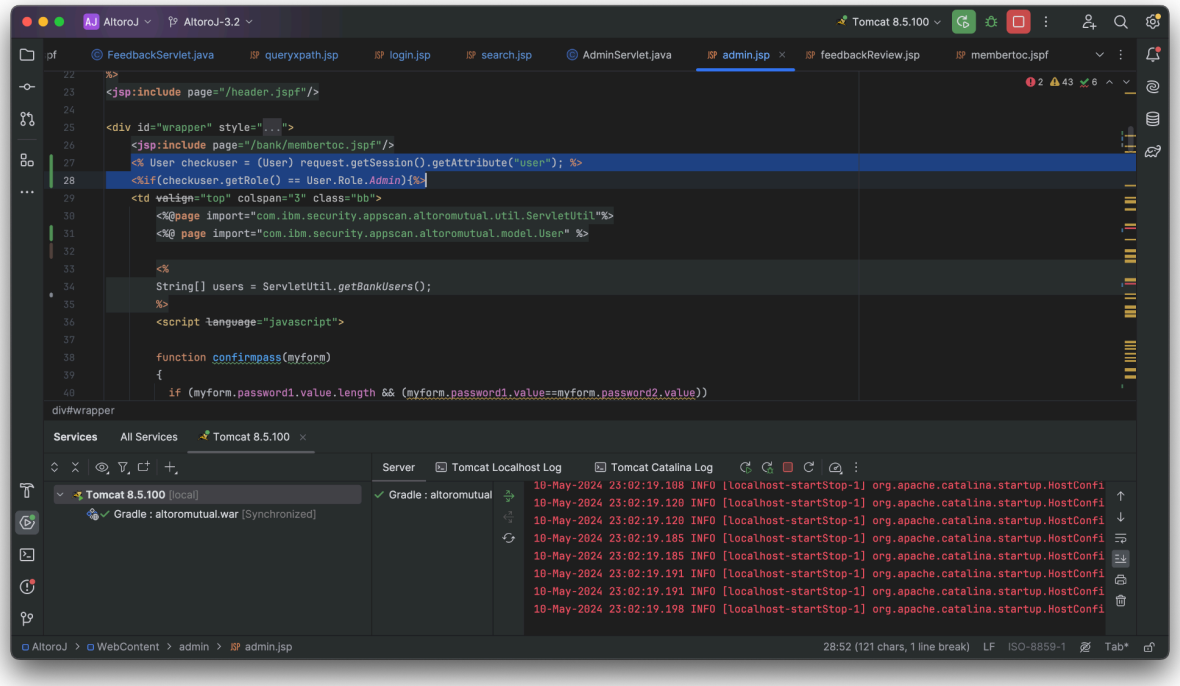

- route to admin page

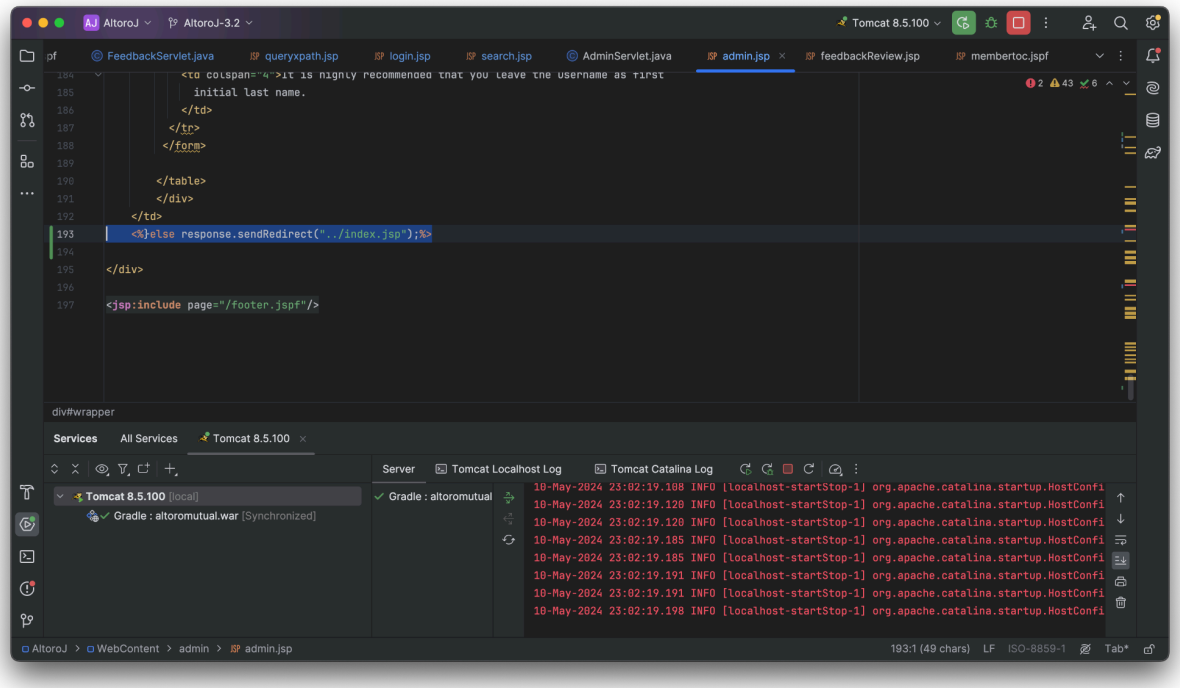


- improper access control accessed user to admin pages



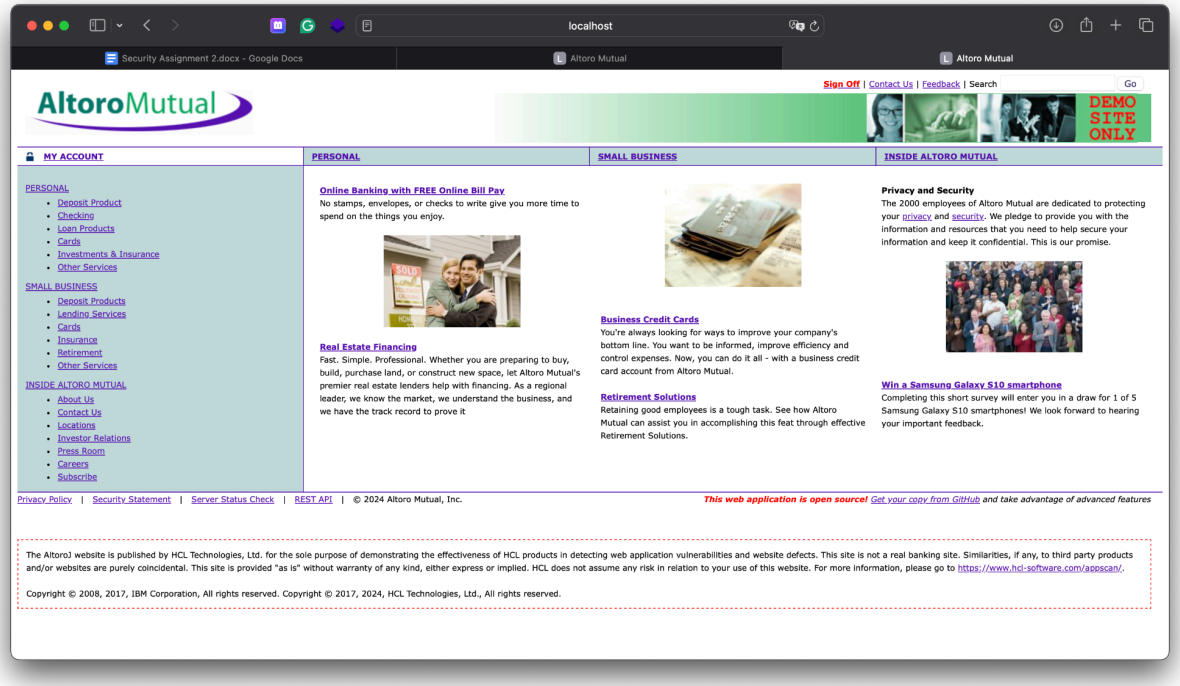

- Fixing steps
  - Go to admin.jsp in the admin subroute
  - Edit the access of the content to admin roles only if not, it redirects to index.jsp




- Finding RE-TEST steps
  - Sign in as a normal user

- go to admin page

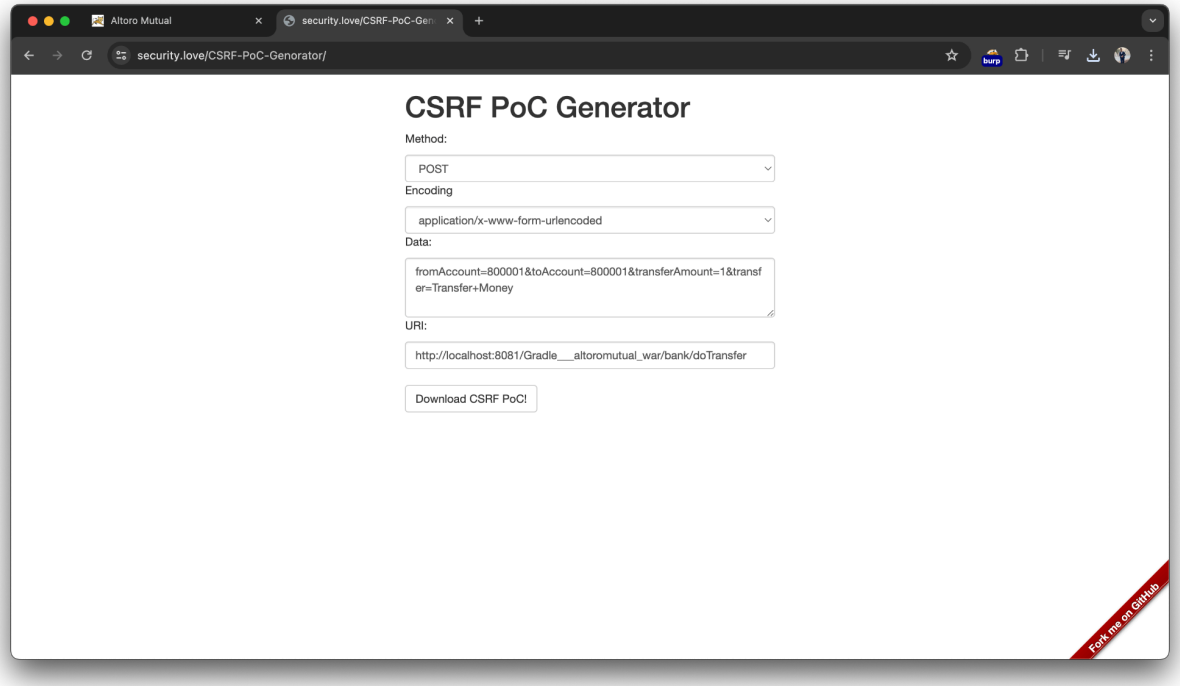


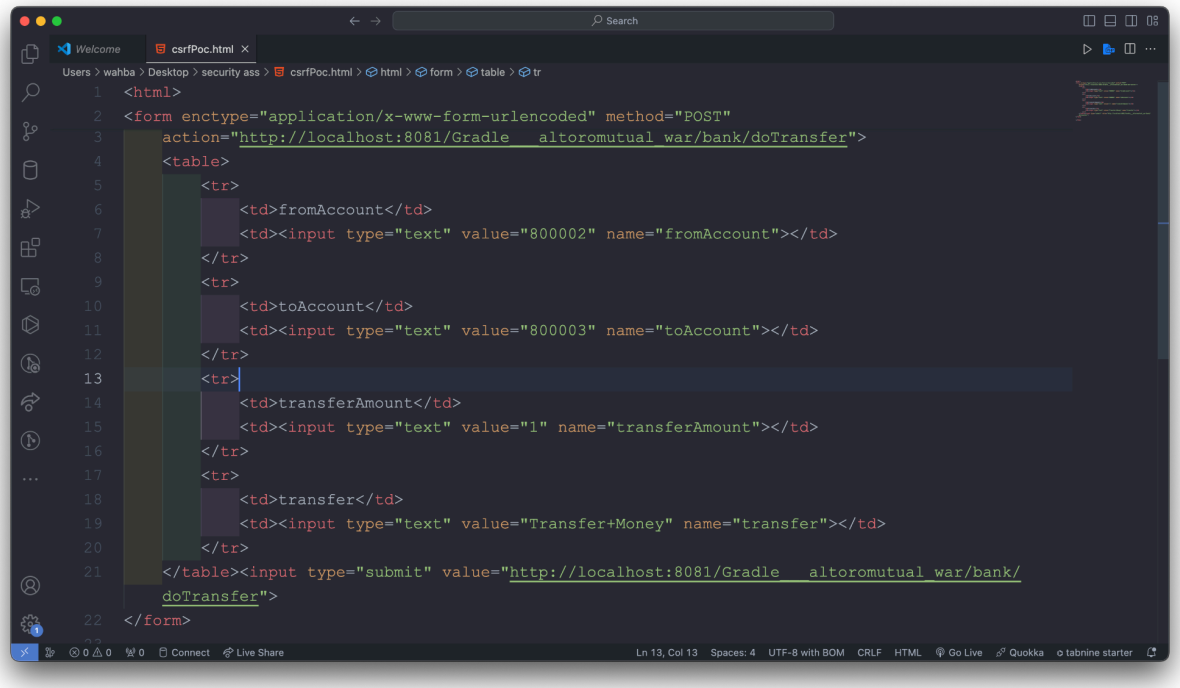

- Routed to index 😀

## ID07

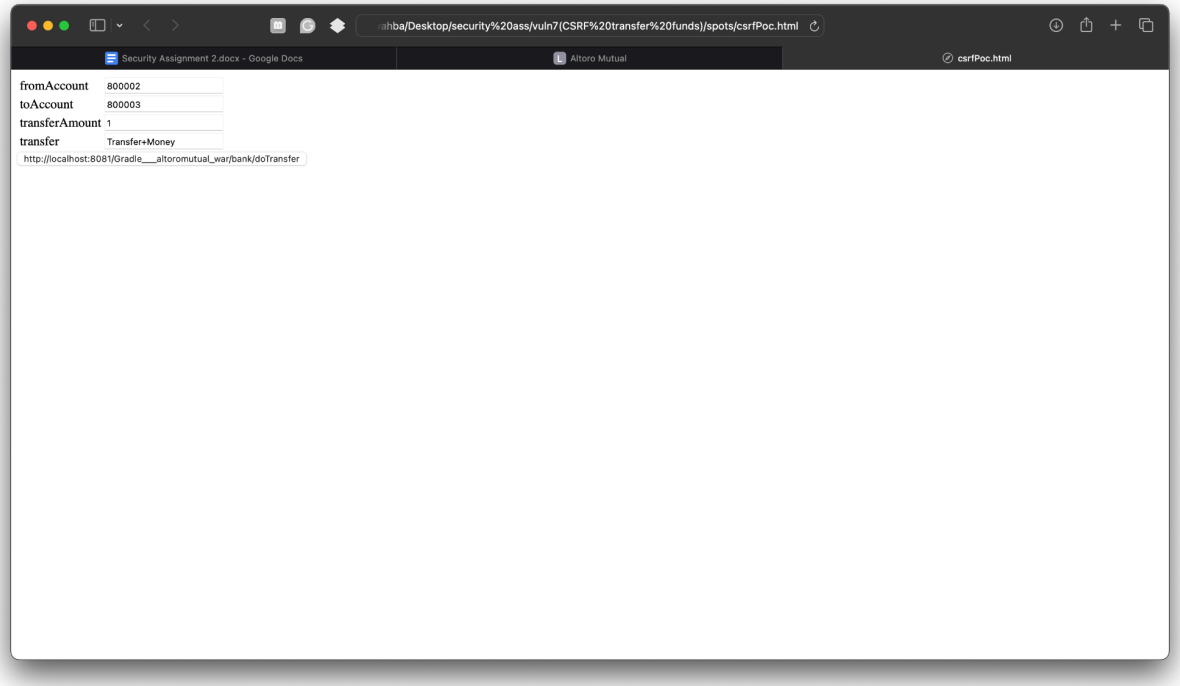
- Finding test steps

- Let’s prepare a CSRF PoC for transfer funds functionality😈

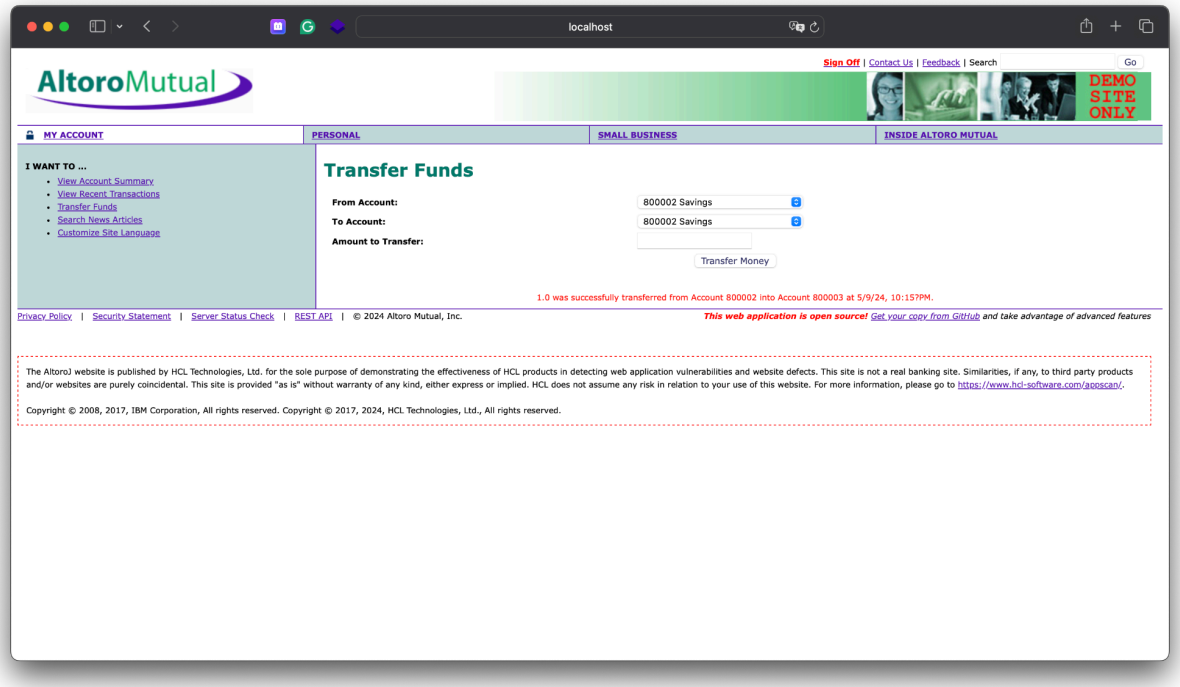


- Open the generated poc in the browser

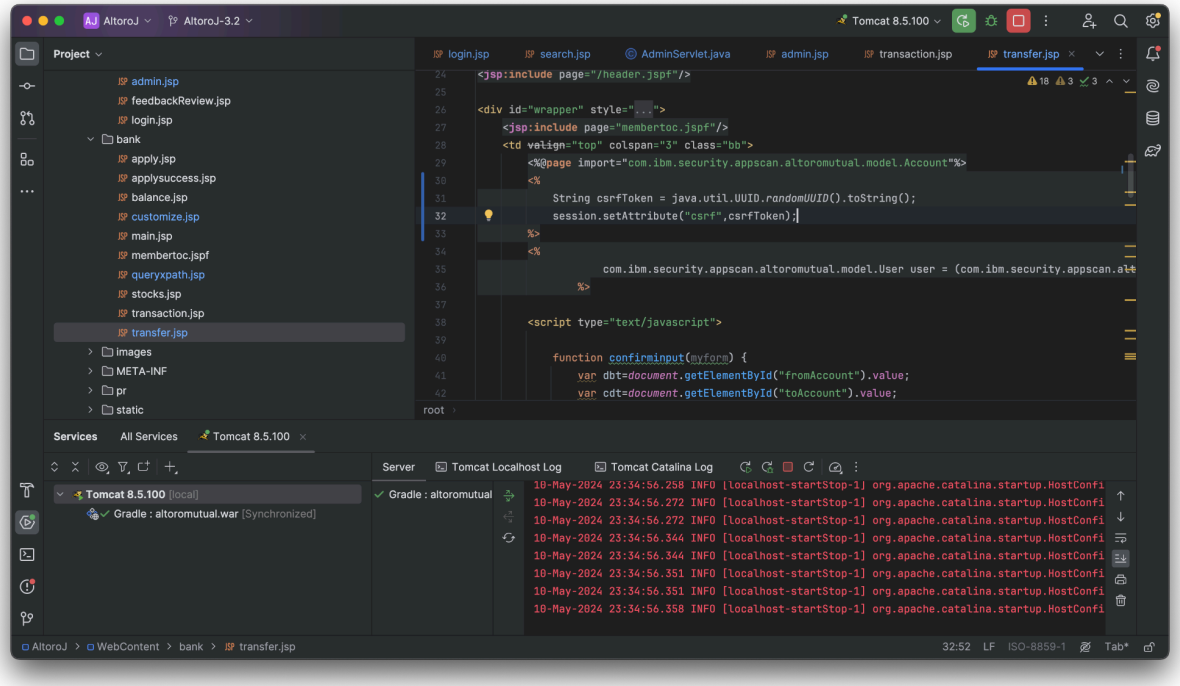


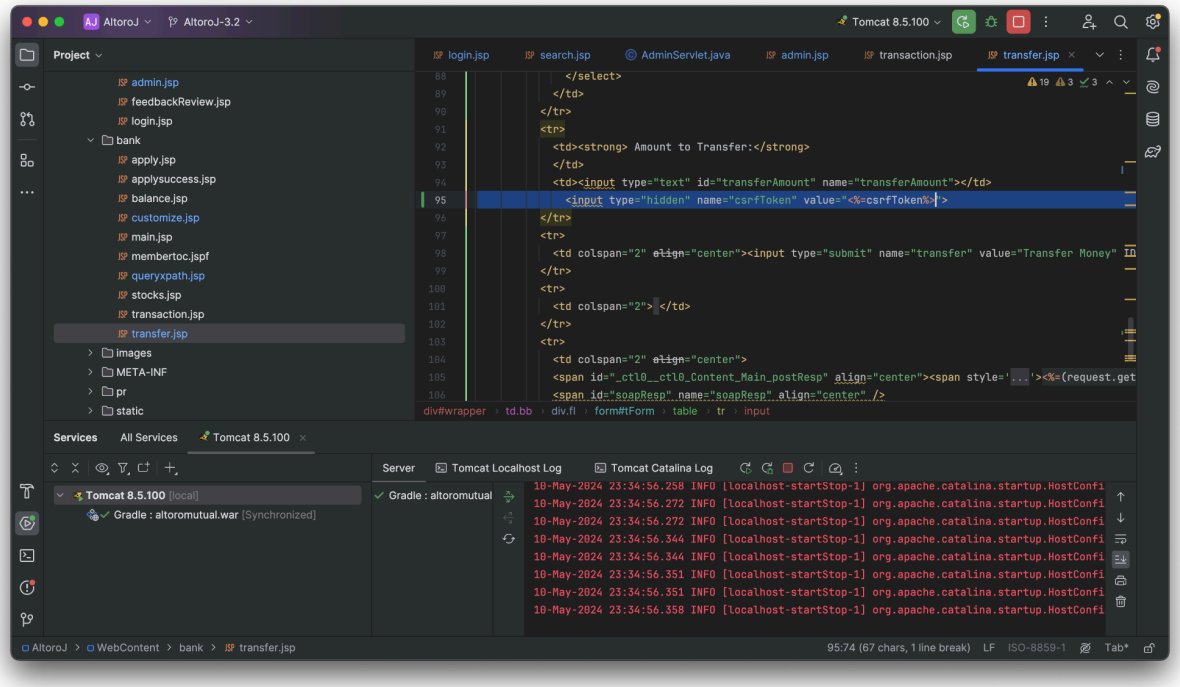

- Go and press the button
- Oooh! executed the transfer between the accounts specified 🥲




- Fixing steps

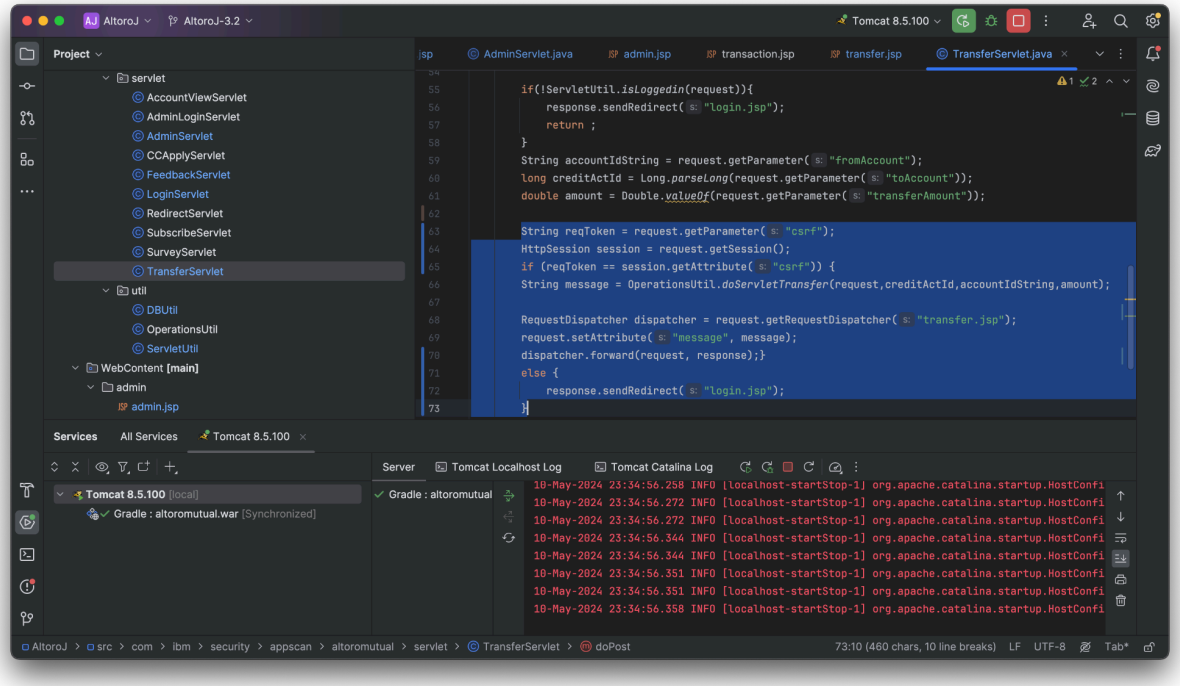
- add csrf token in the admin.jsp and in the submit form using hidden input



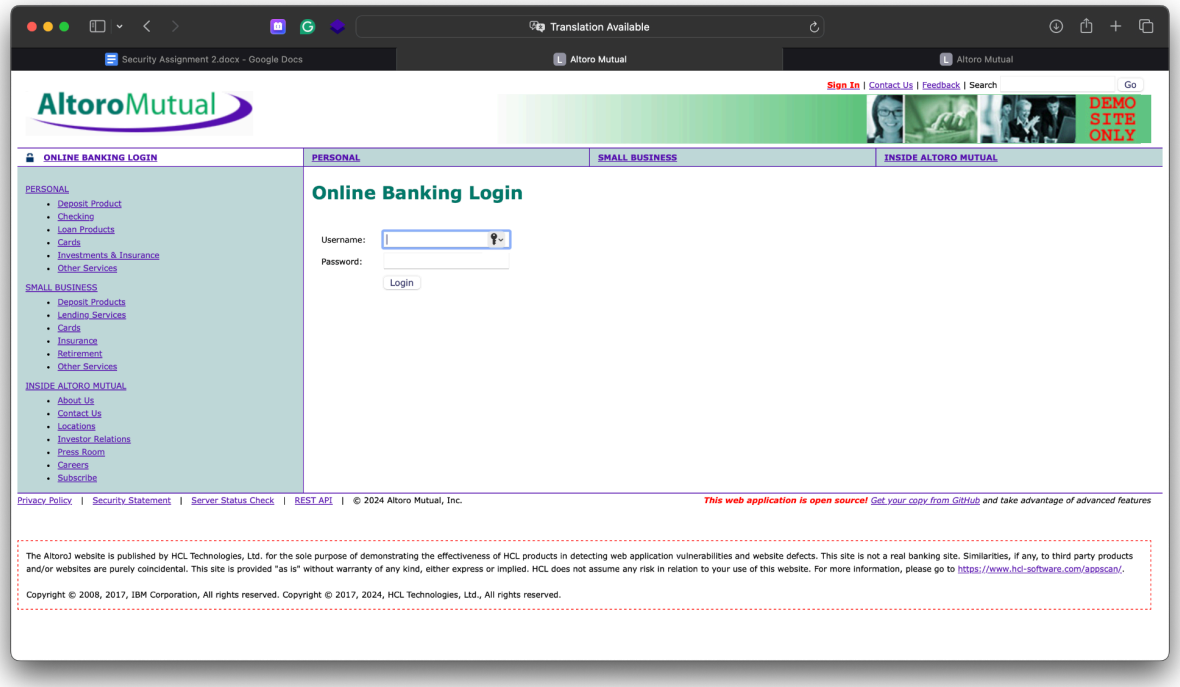




- check-in admin servlet if the csrf token in session is equal to the token that came in the request then execute else redirect to login and release

the session



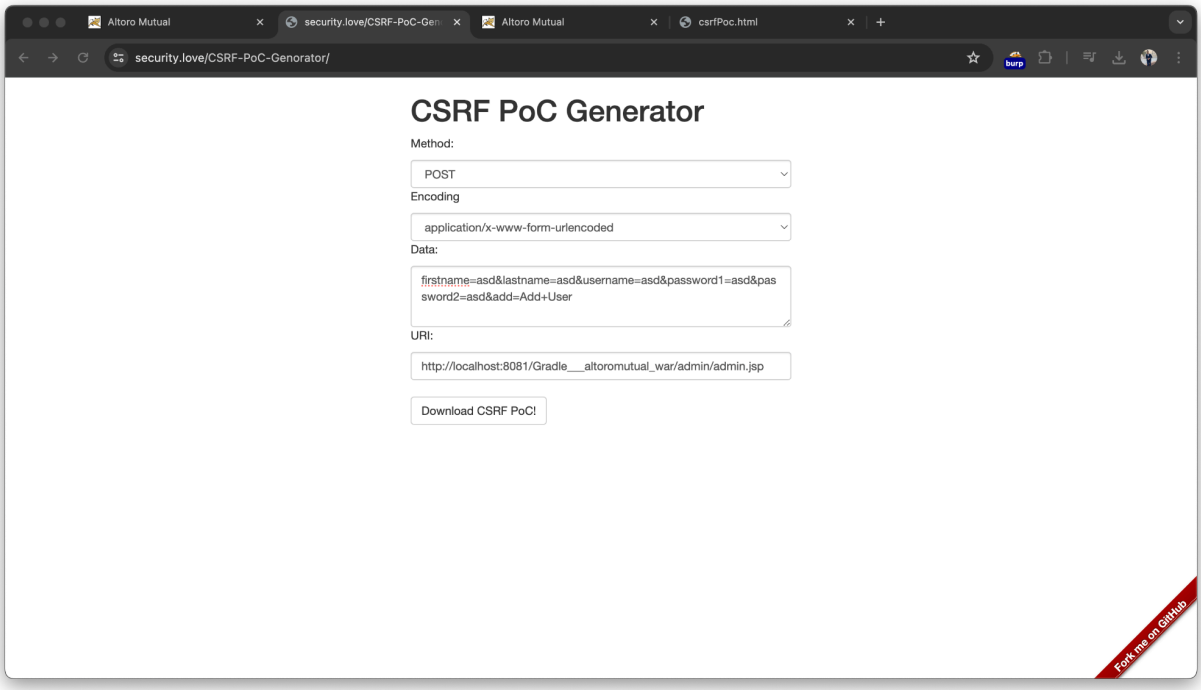

- Finding RE-TEST steps
  - Go to the HTML poc generated
  - the user will be logged out and redirected to login

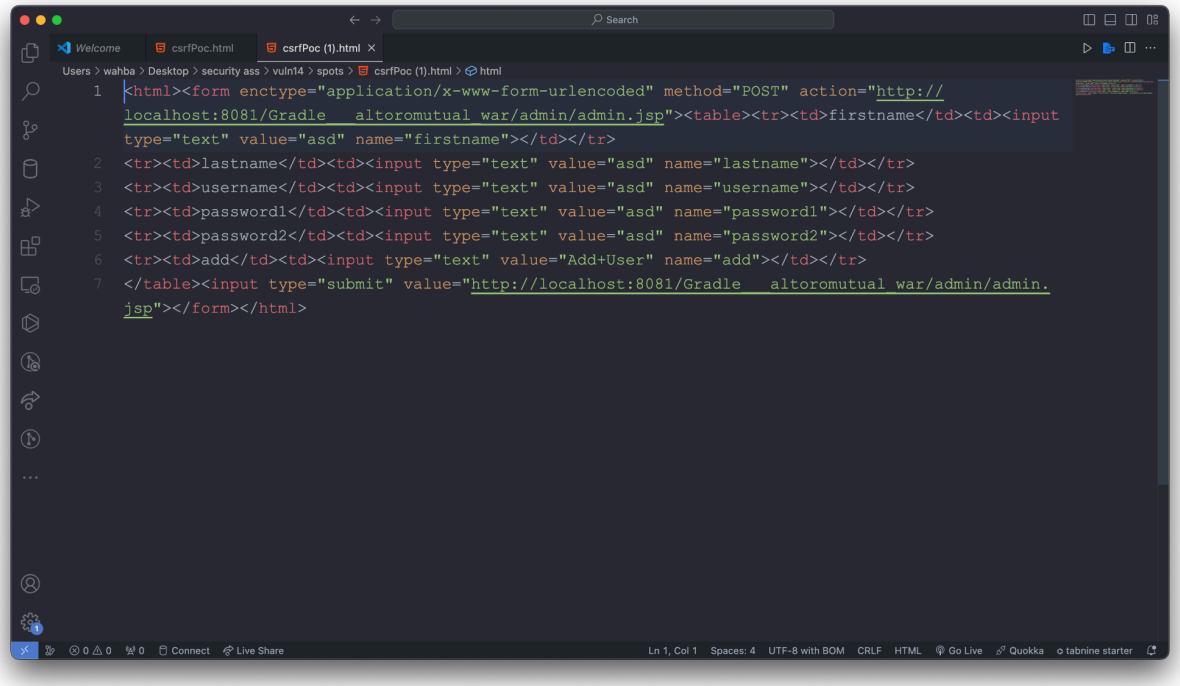


- Finding RE-TEST steps

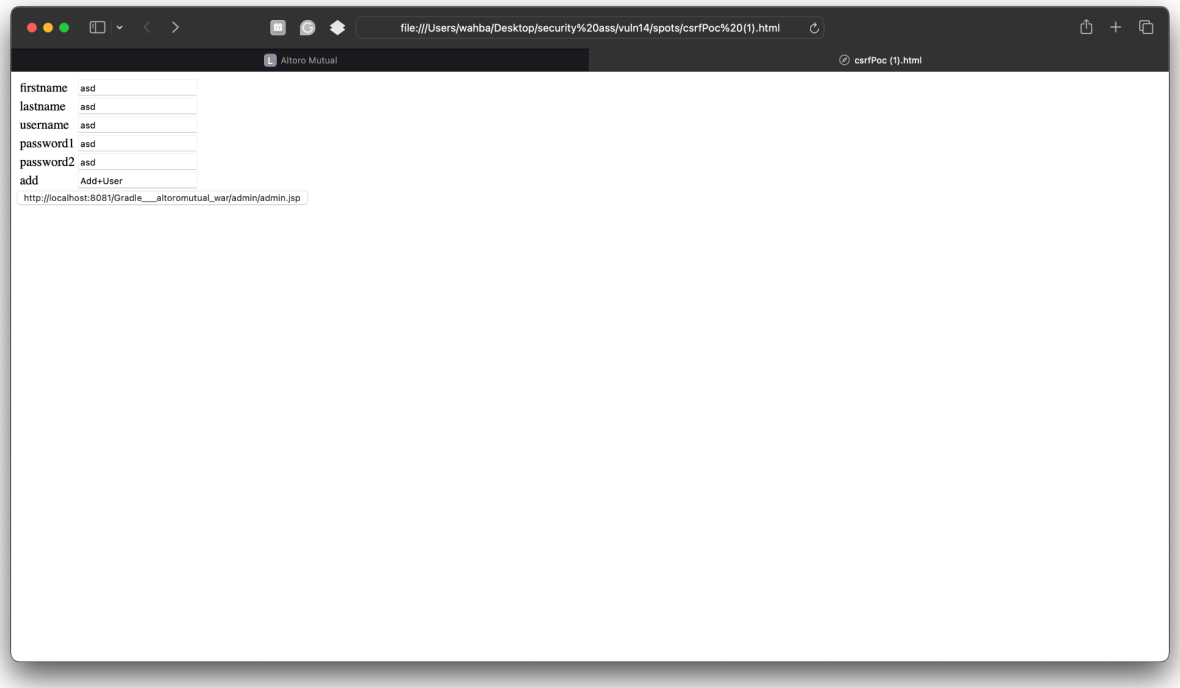
## ID08

- Finding test steps
  - Let’s prepare a CSRF PoC for add user functionality😈



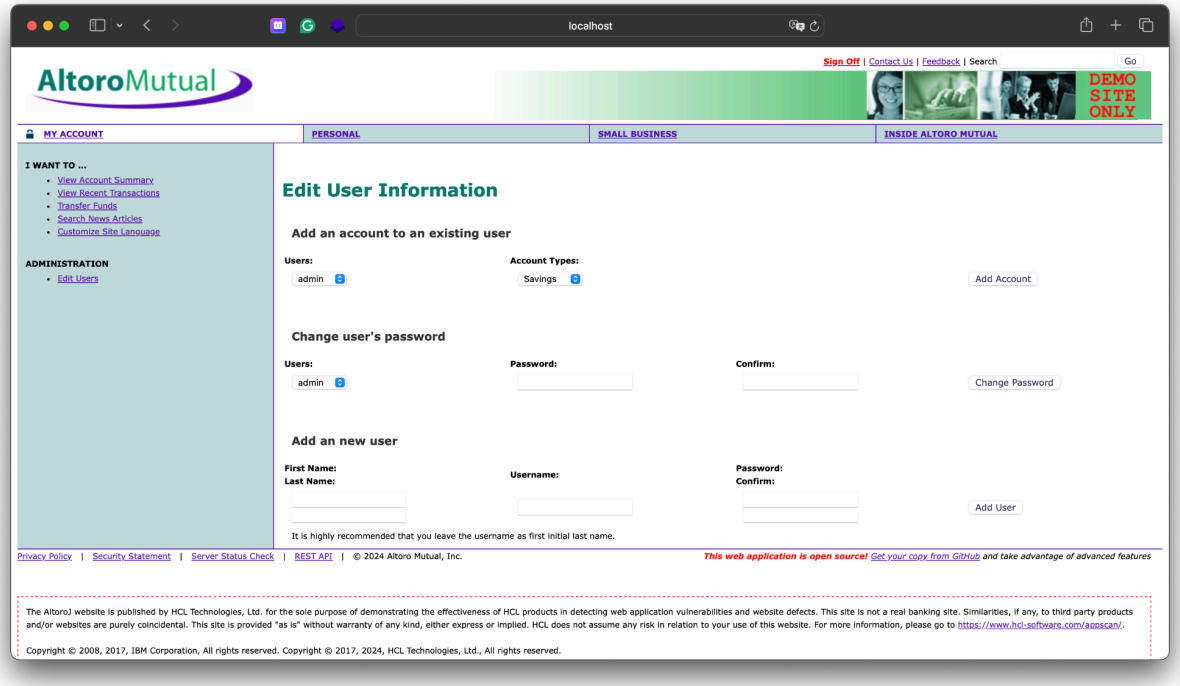



- Open the generated poc in the browser

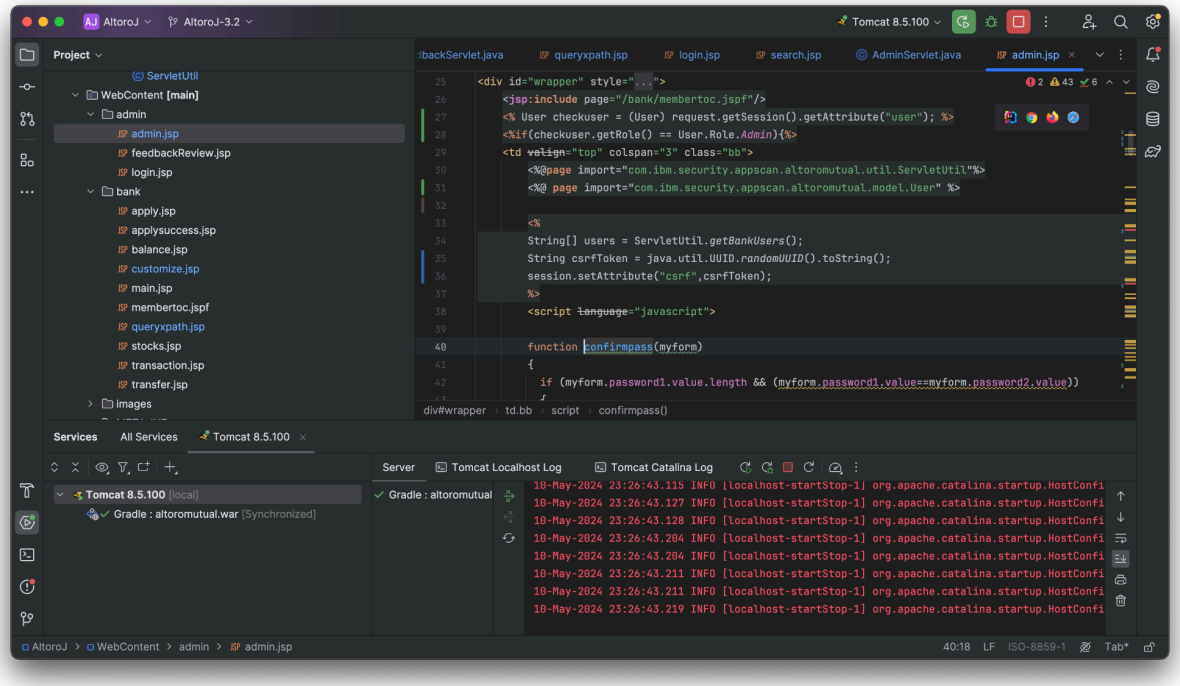


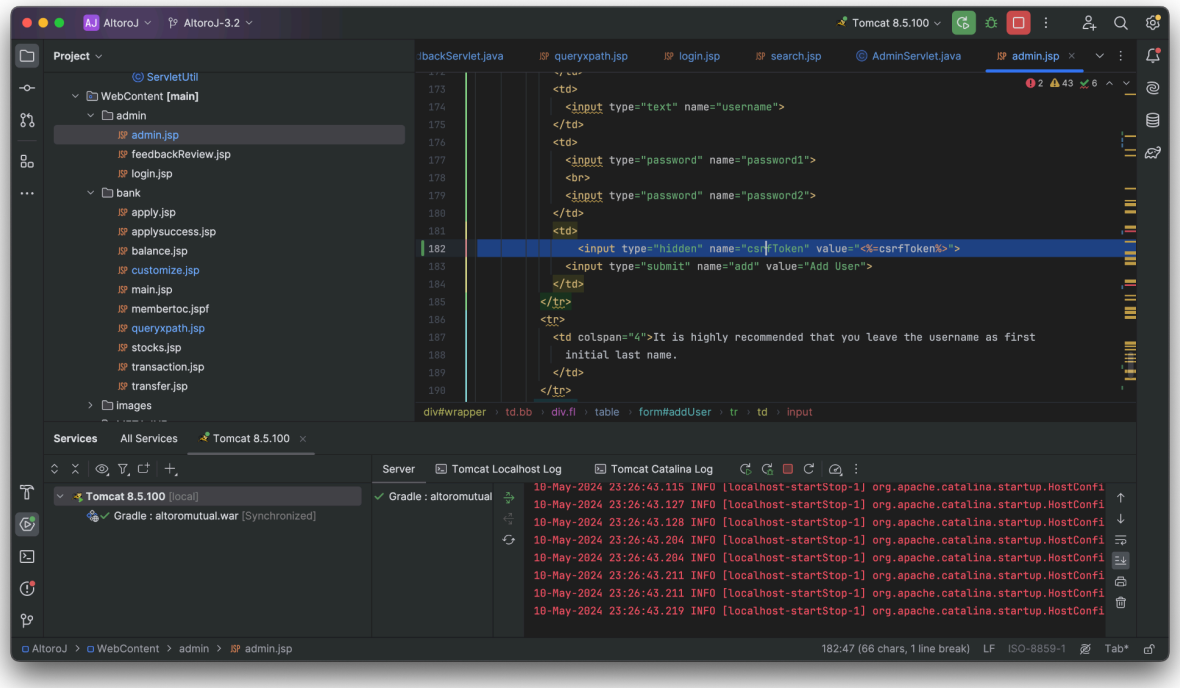

- Go and press the button

- Added new User!!



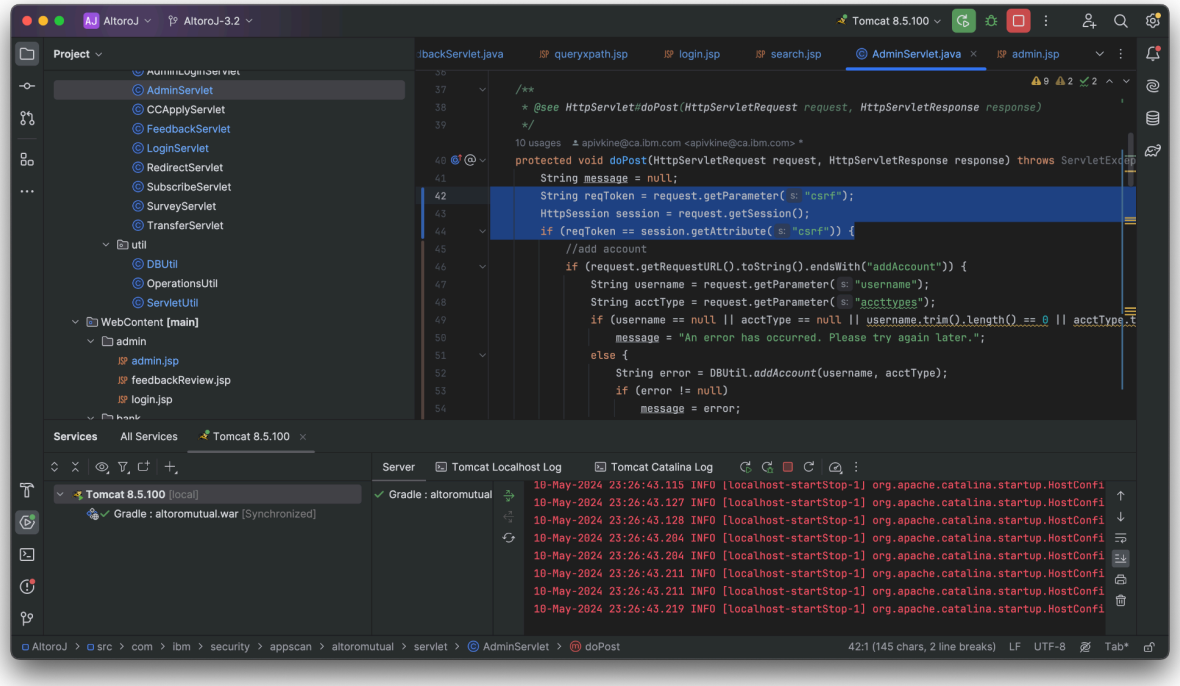

- Fixing steps
  - add csrf token in the admin.jsp and in the submit form using hidden input



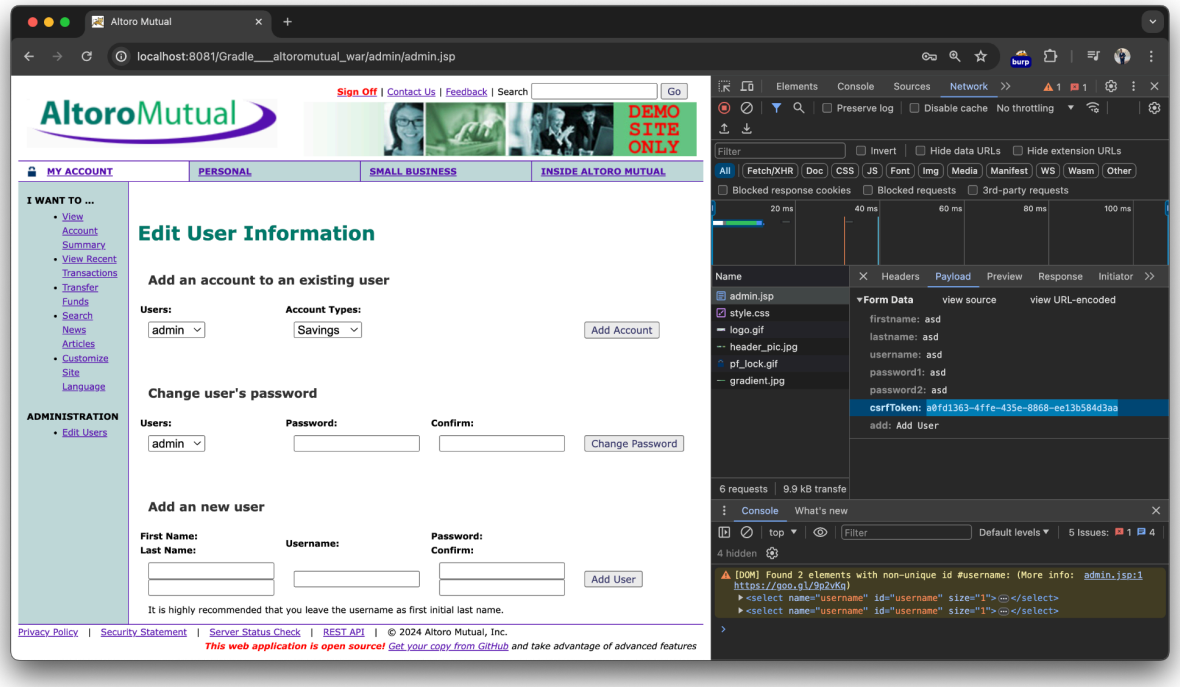




- check-in admin servlet if the csrf token in session is equal to the token that came in the request then execute else redirect to login and release

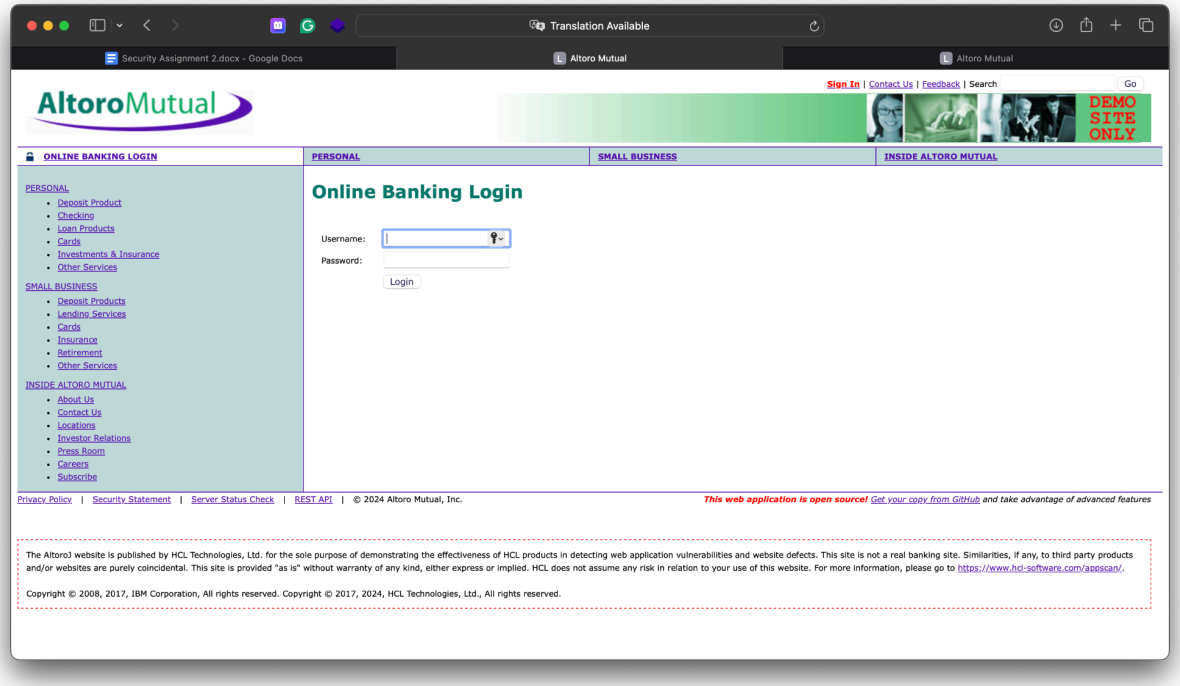
the session




- Finding RE-TEST steps
  - Go to the HTML poc generated



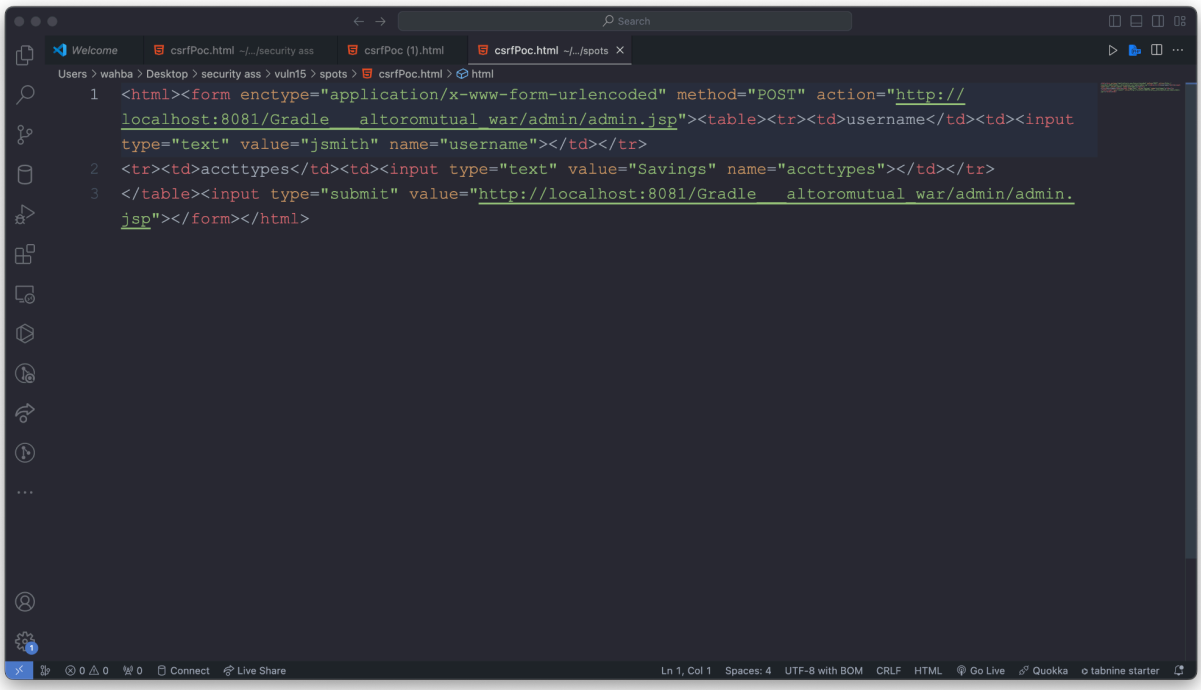
- the user will be logged out and redirected to login



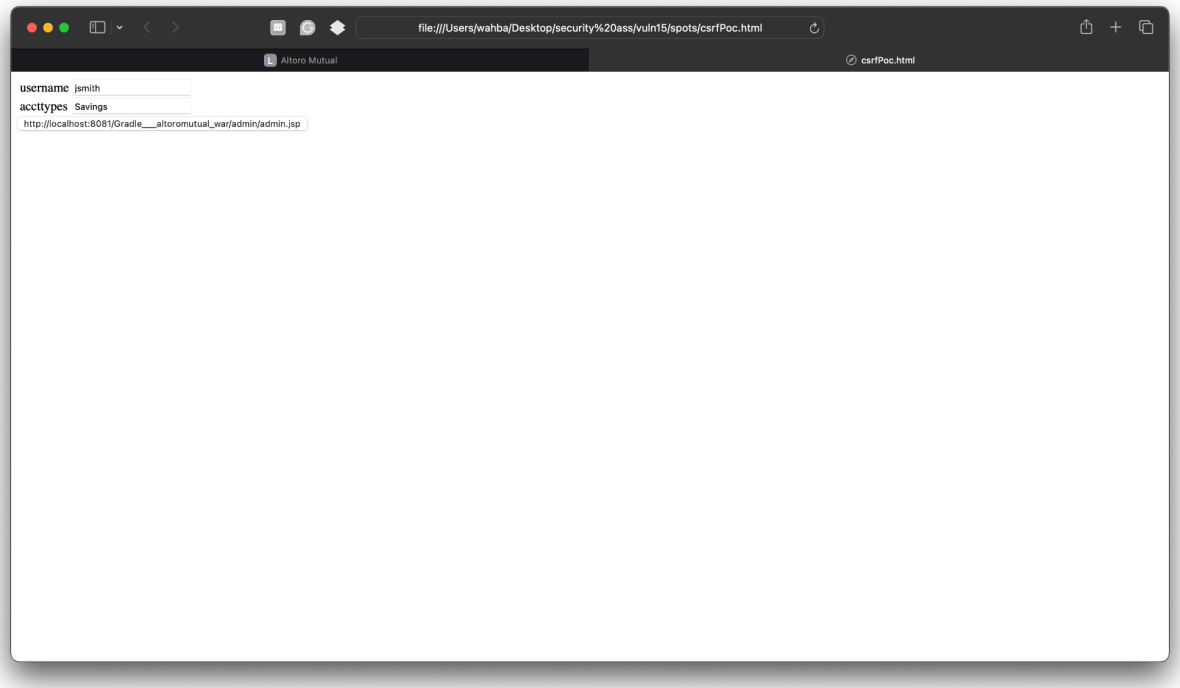

## ID09

- Finding test steps

- Let’s prepare a CSRF PoC for add account to userfunctionality😈

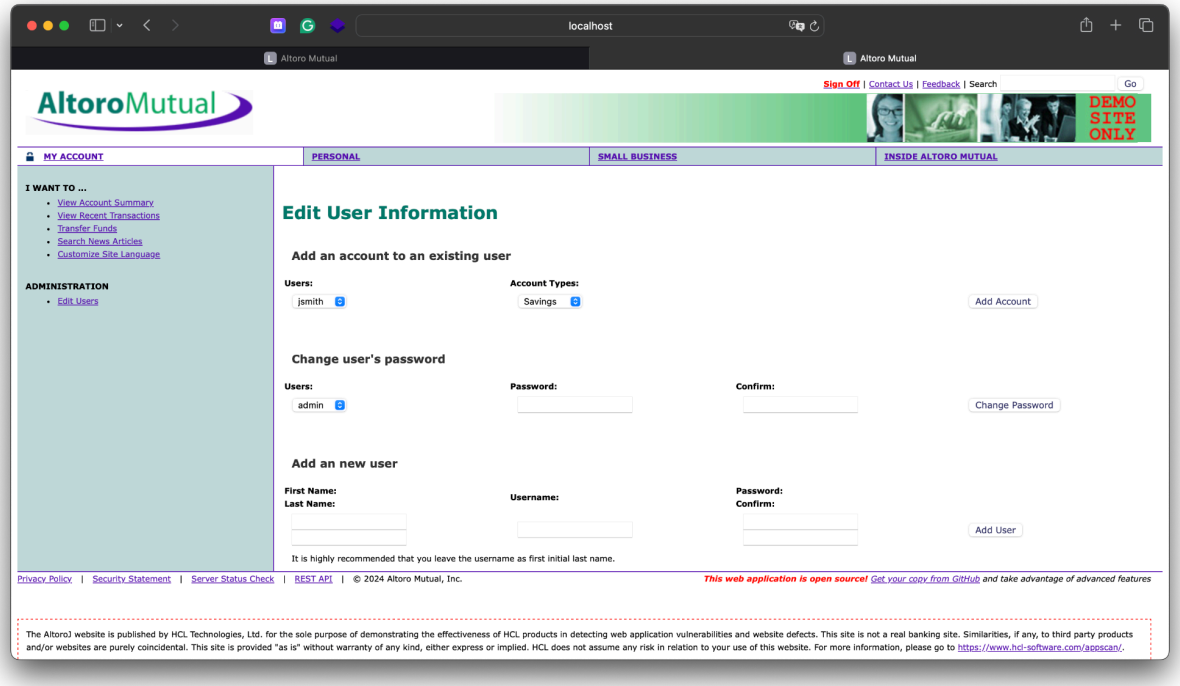



- Open the generated poc in the browser

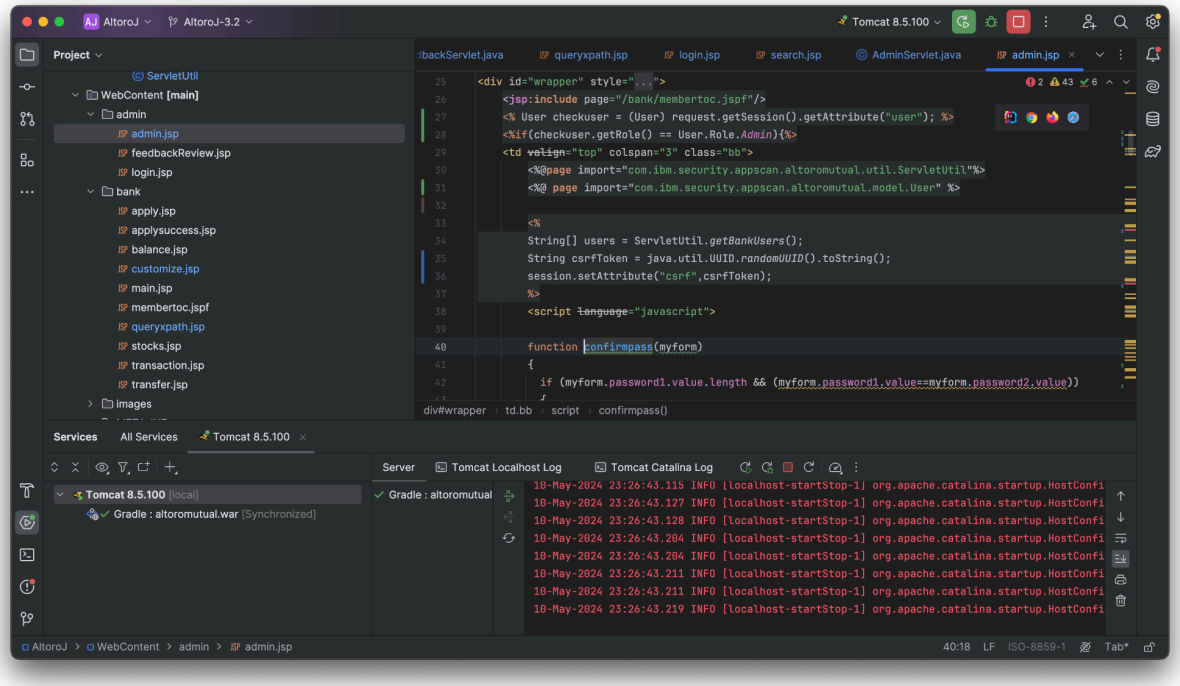


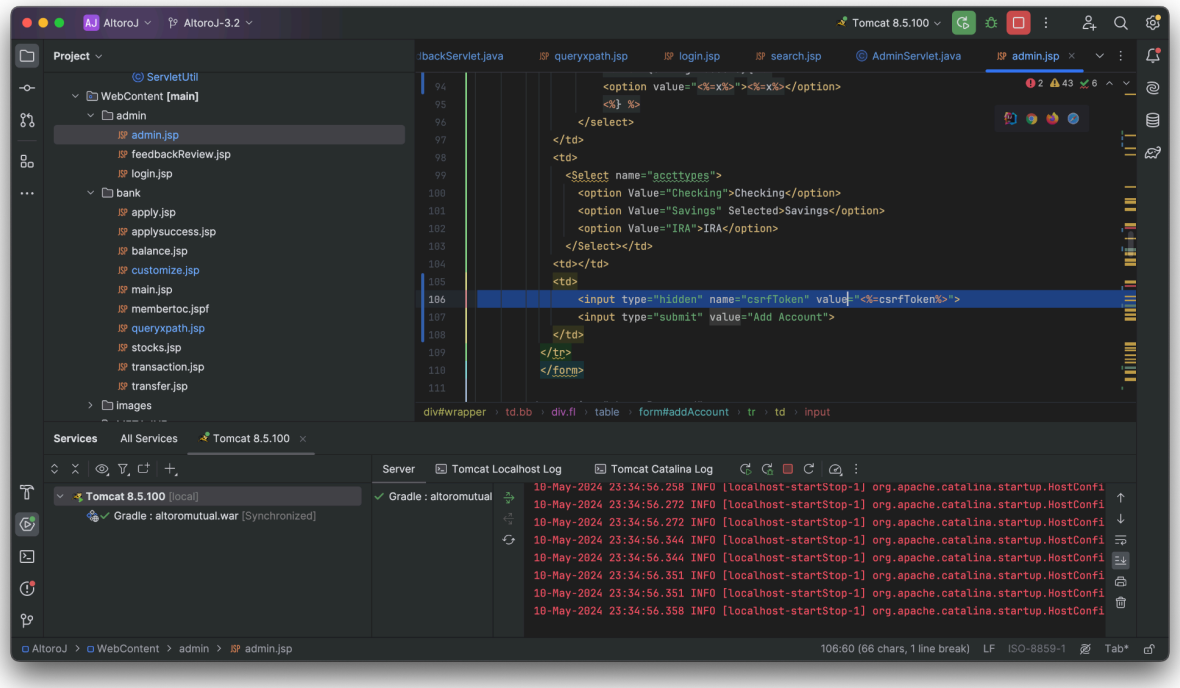

- Go and press the button

- Added an account to user



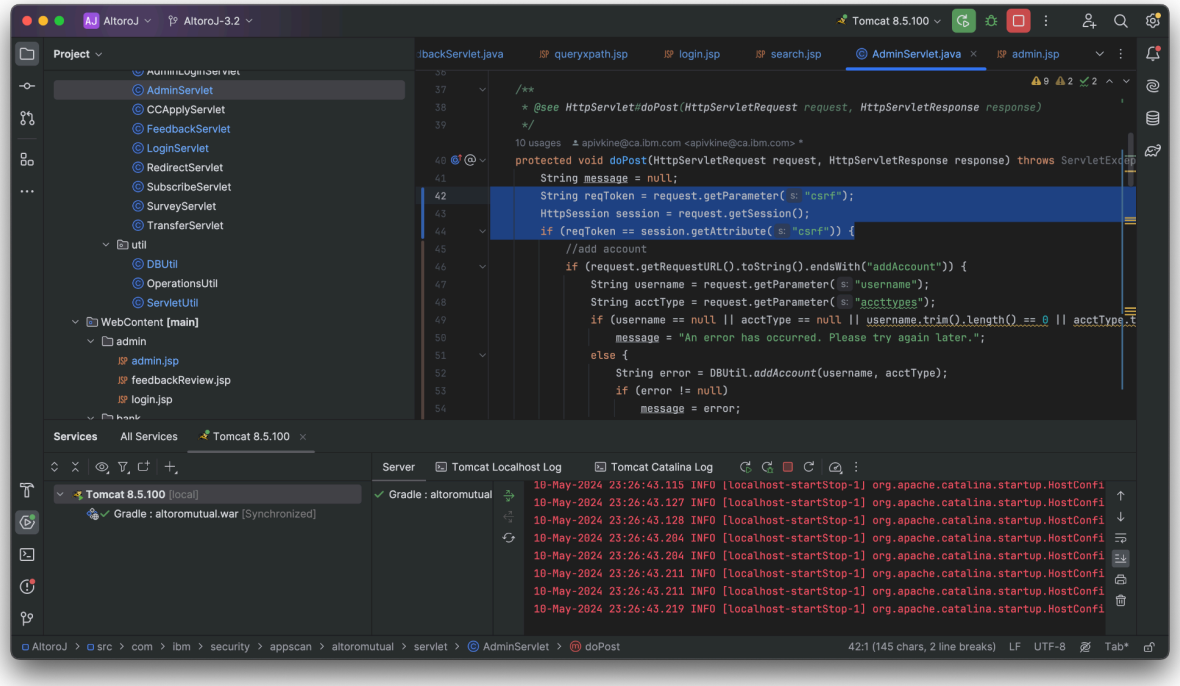

- Fixing steps
  - add csrf token in the admin.jsp and in the submit form using hidden input



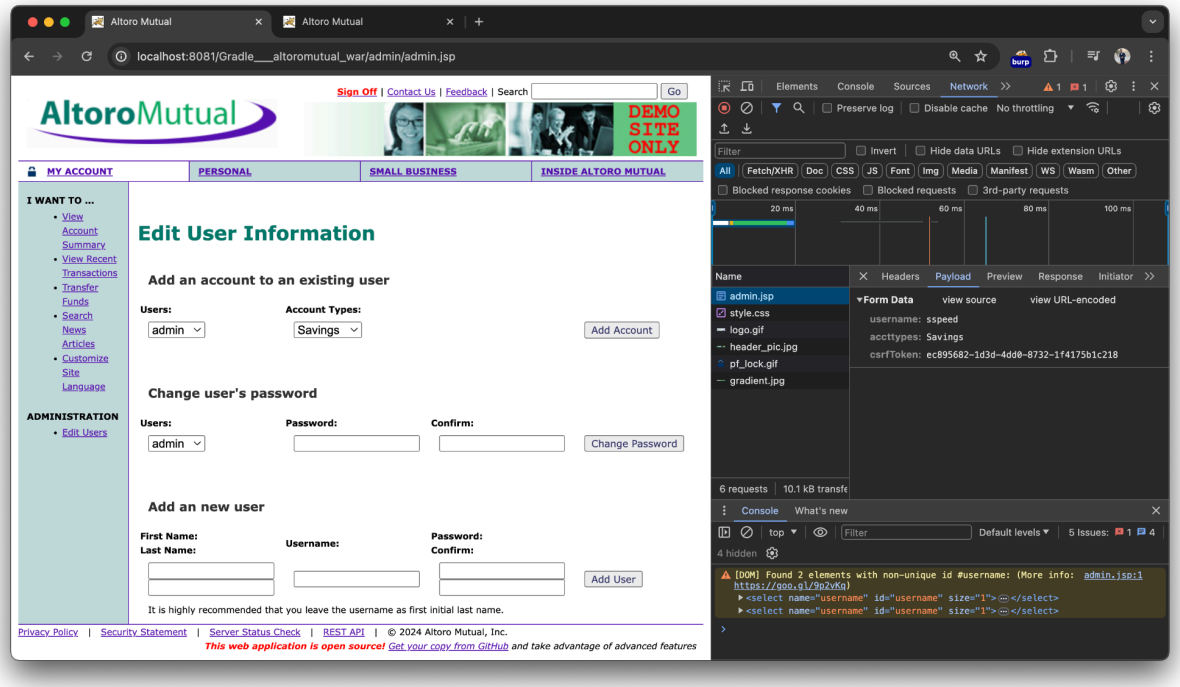




- check-in admin servlet if the csrf token in session is equal to the token that came in the request then execute else redirect to login and release

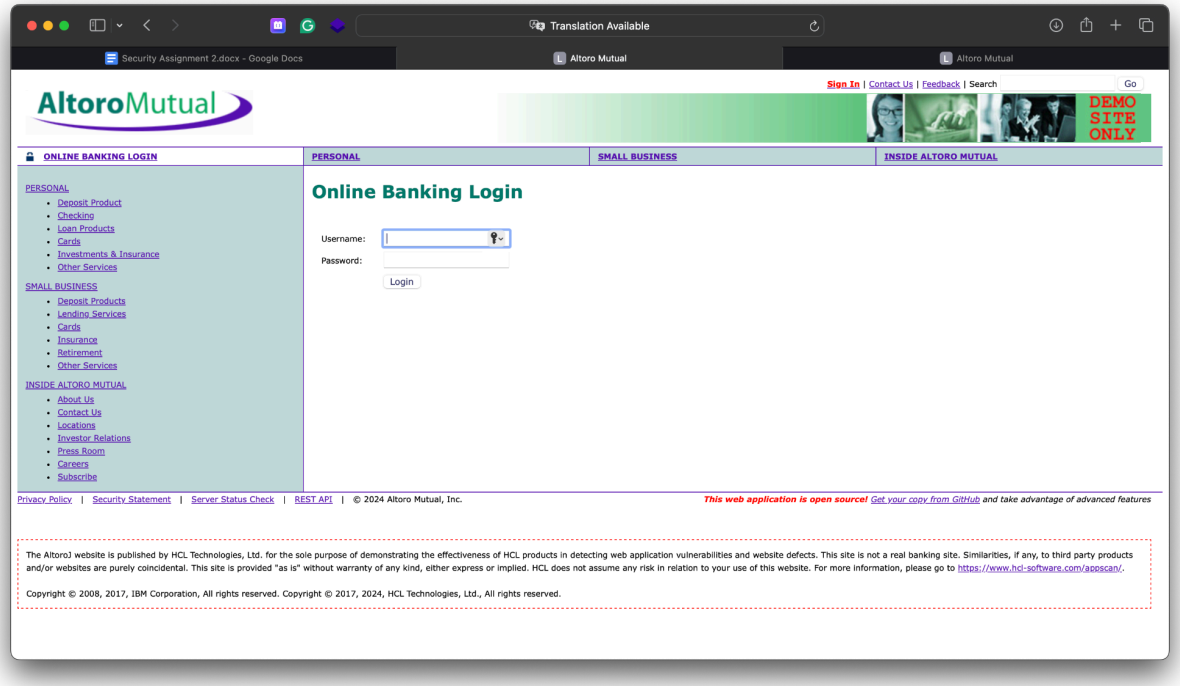
the session




- Finding RE-TEST steps
  - Go to the HTML poc generated



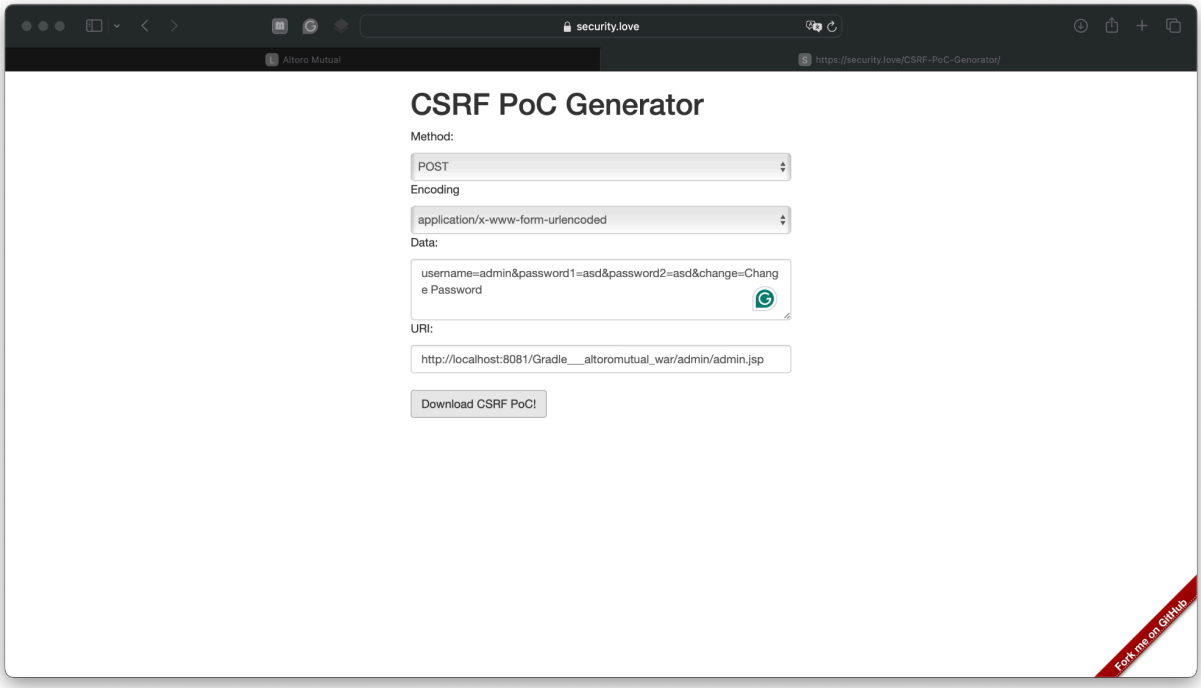
- the user will be logged out and redirected to login

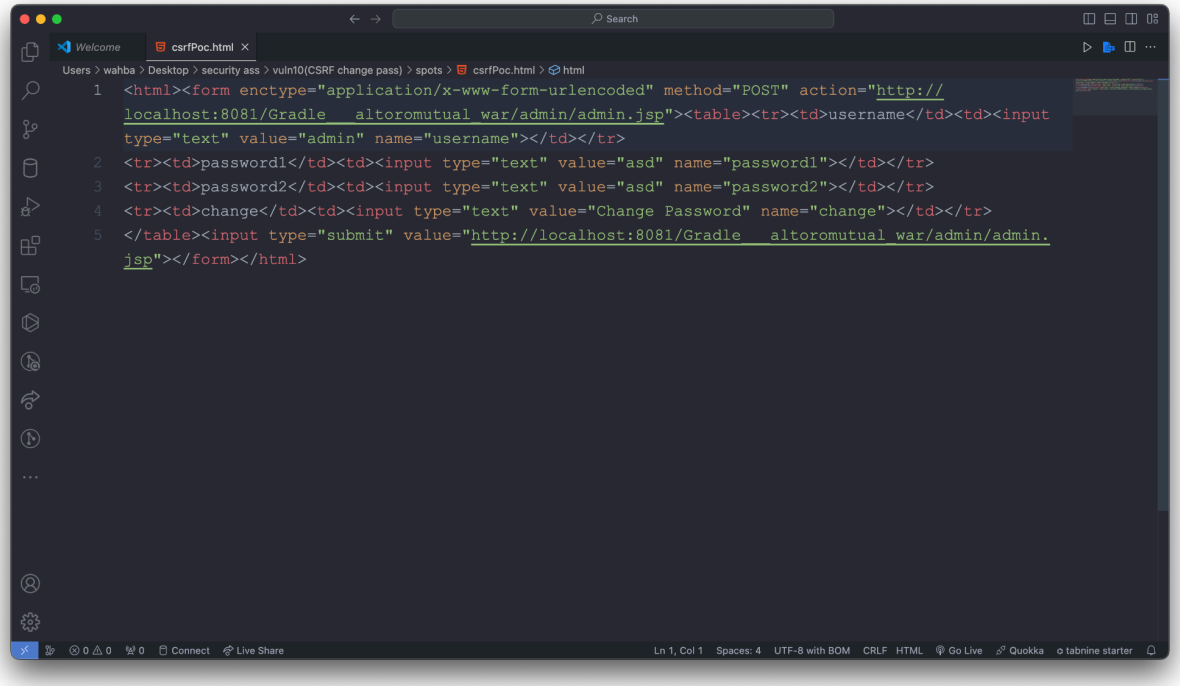



## ID10

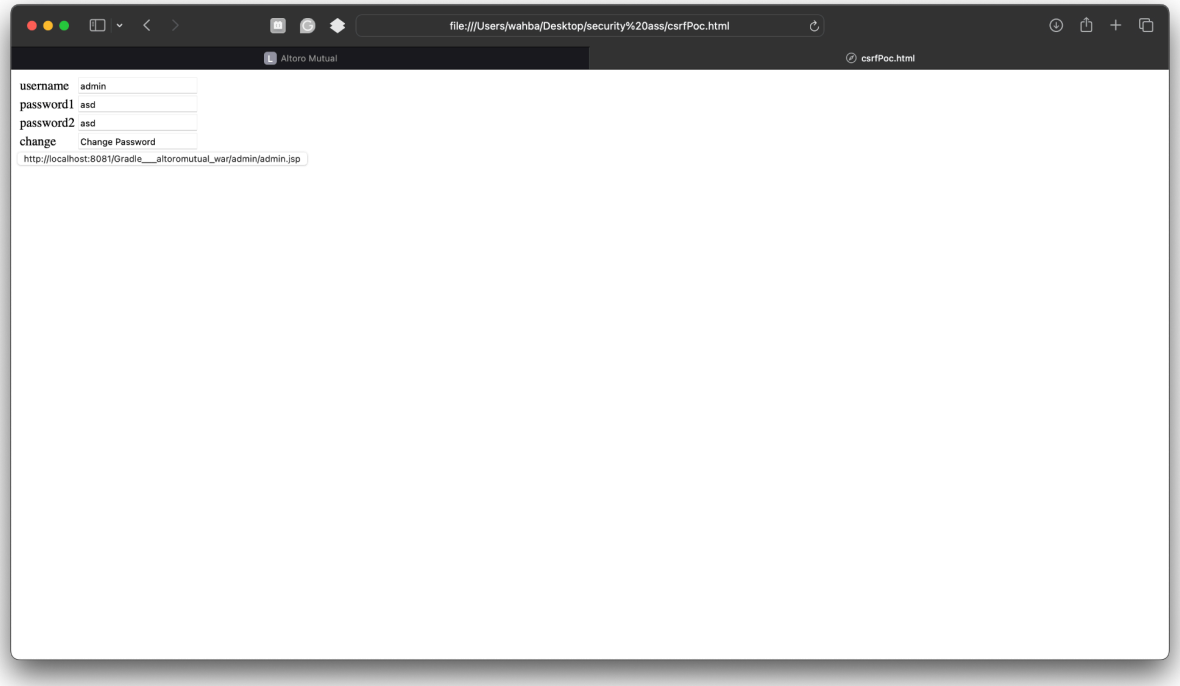
- Finding test steps

- Let’s prepare a CSRF PoC for change password fo user functionality😈

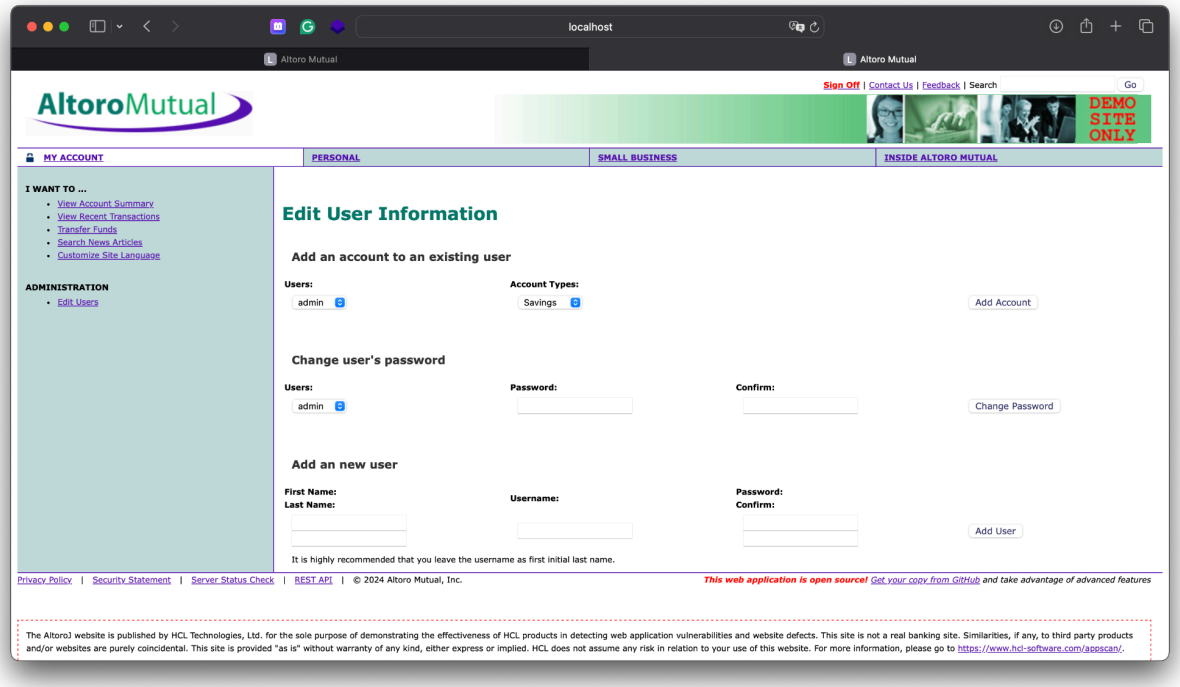


- Open the generated poc in the browser

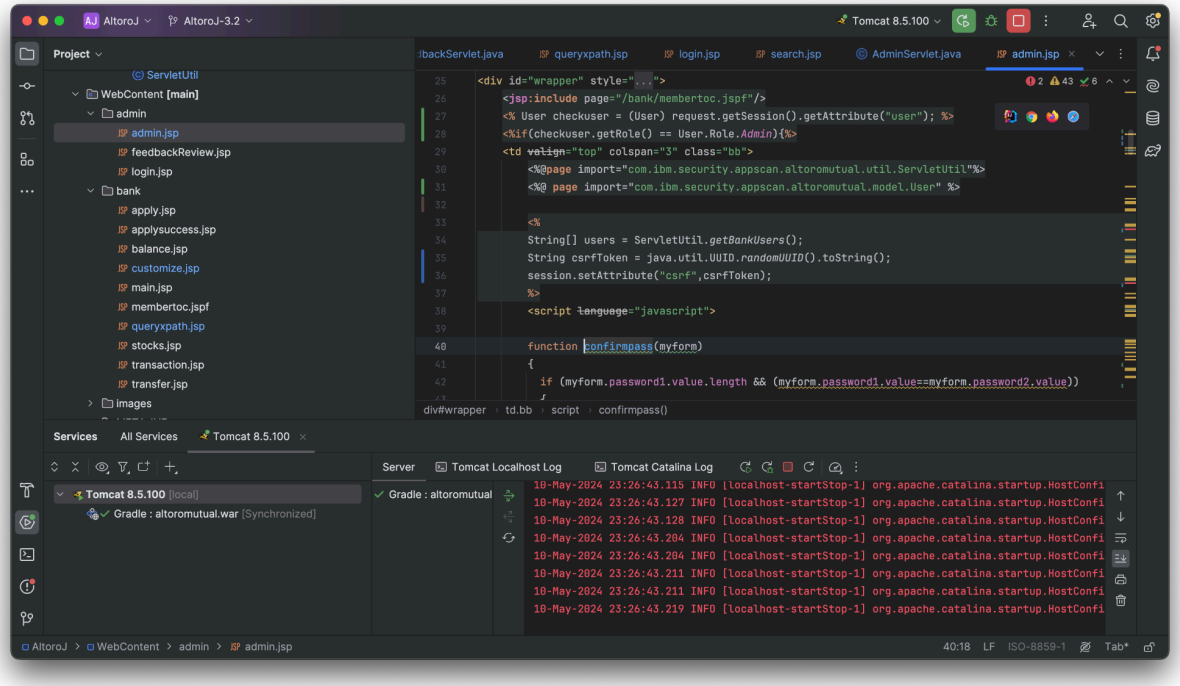


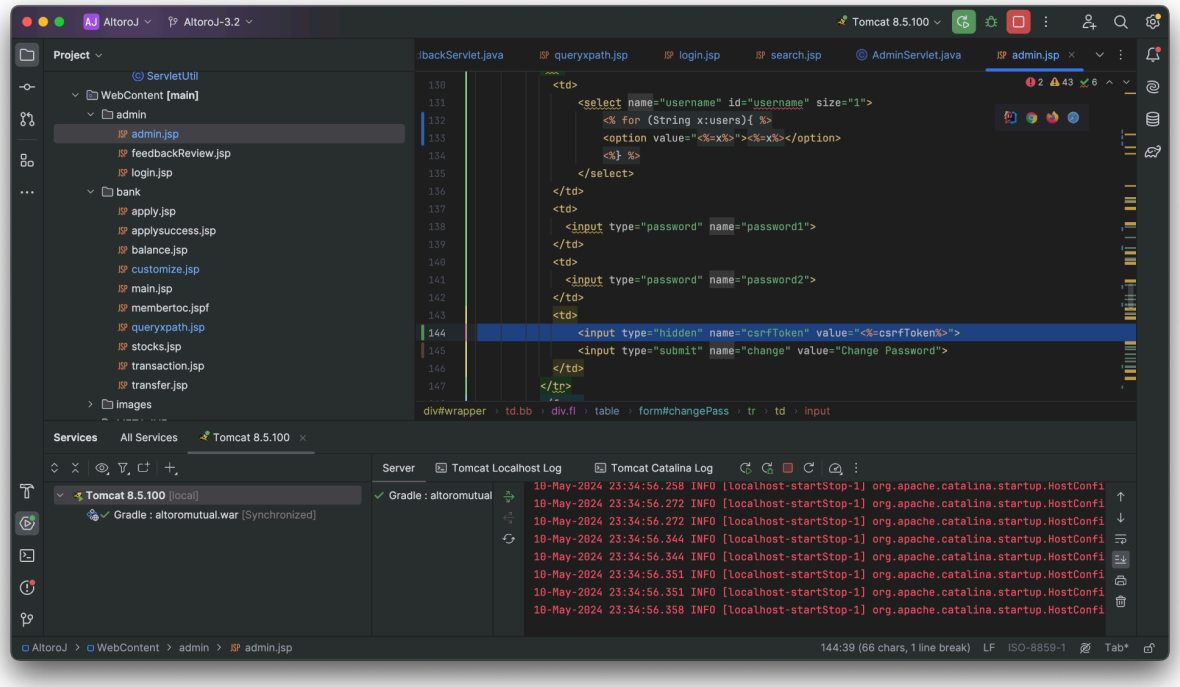

- Go and press the button
- Password for user changed




- Fixing steps

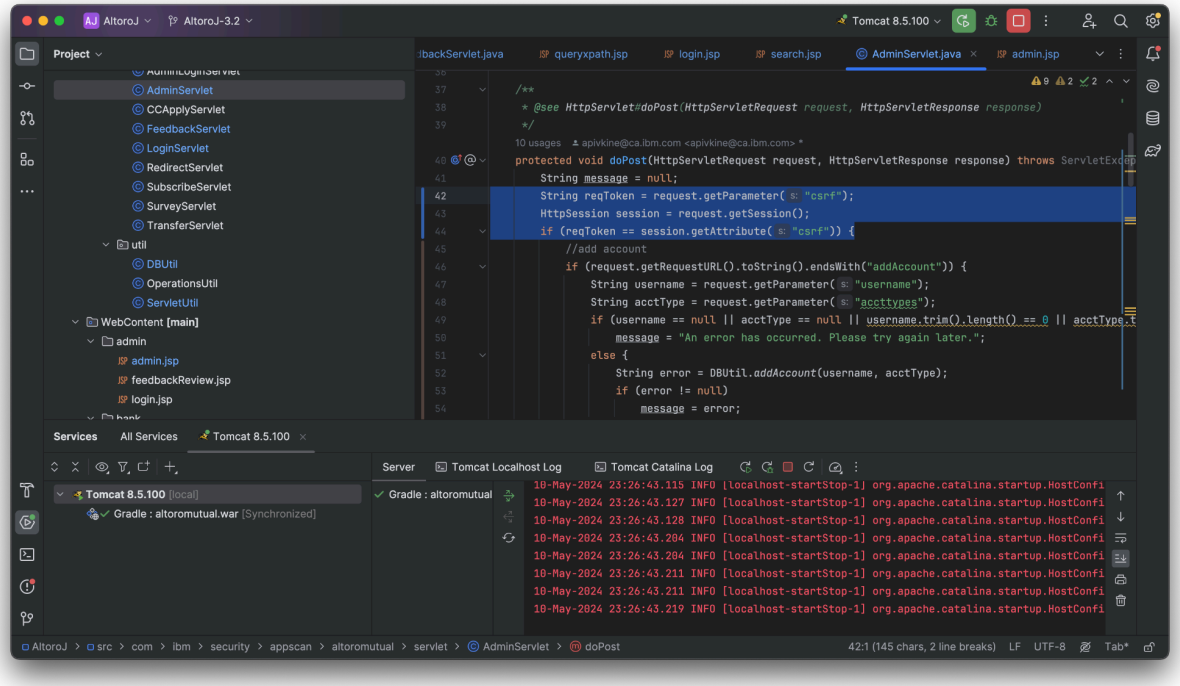
- add csrf token in the admin.jsp and in the submit form using hidden input



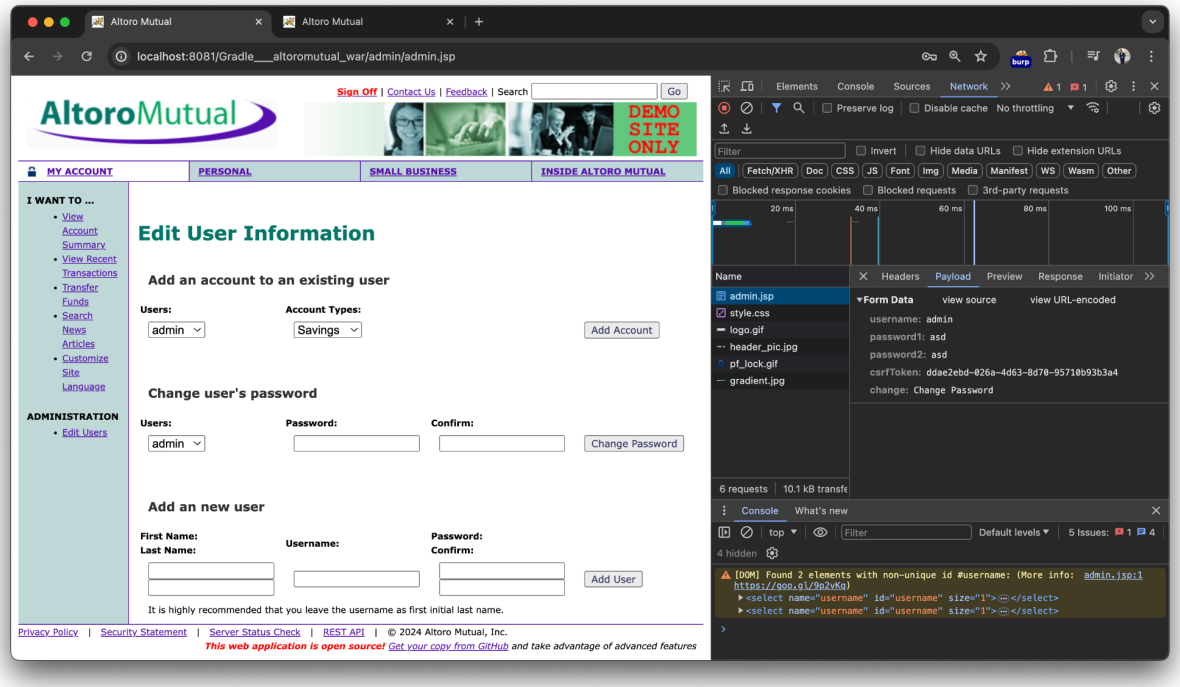




- check-in admin servlet if the csrf token in session is equal to the token that came in the request then execute else redirect to login and release

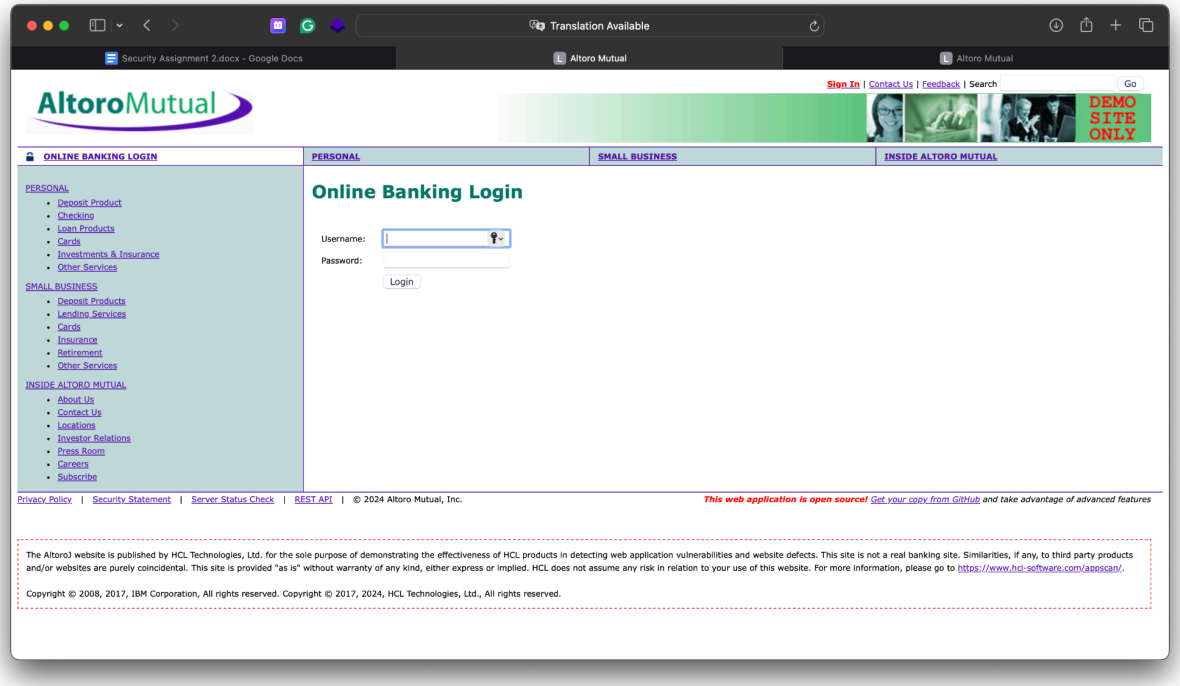
the session




- Finding RE-TEST steps
  - Go to the HTML poc generated



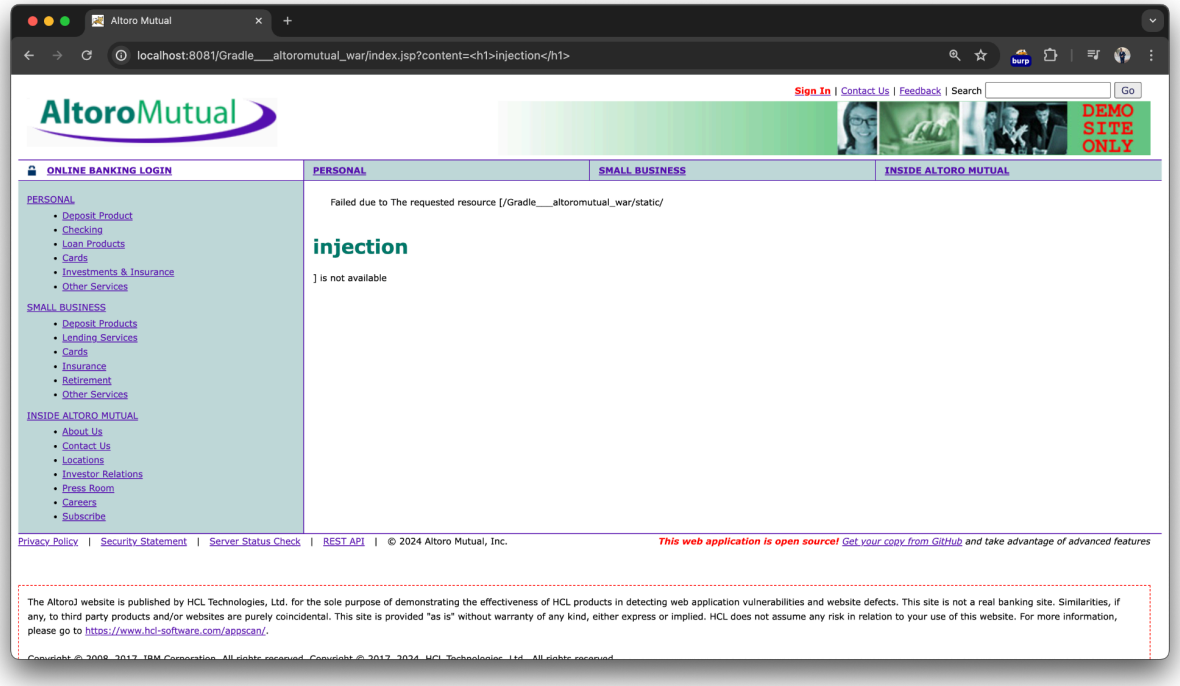
- the user will be logged out and redirected to login




## ID11

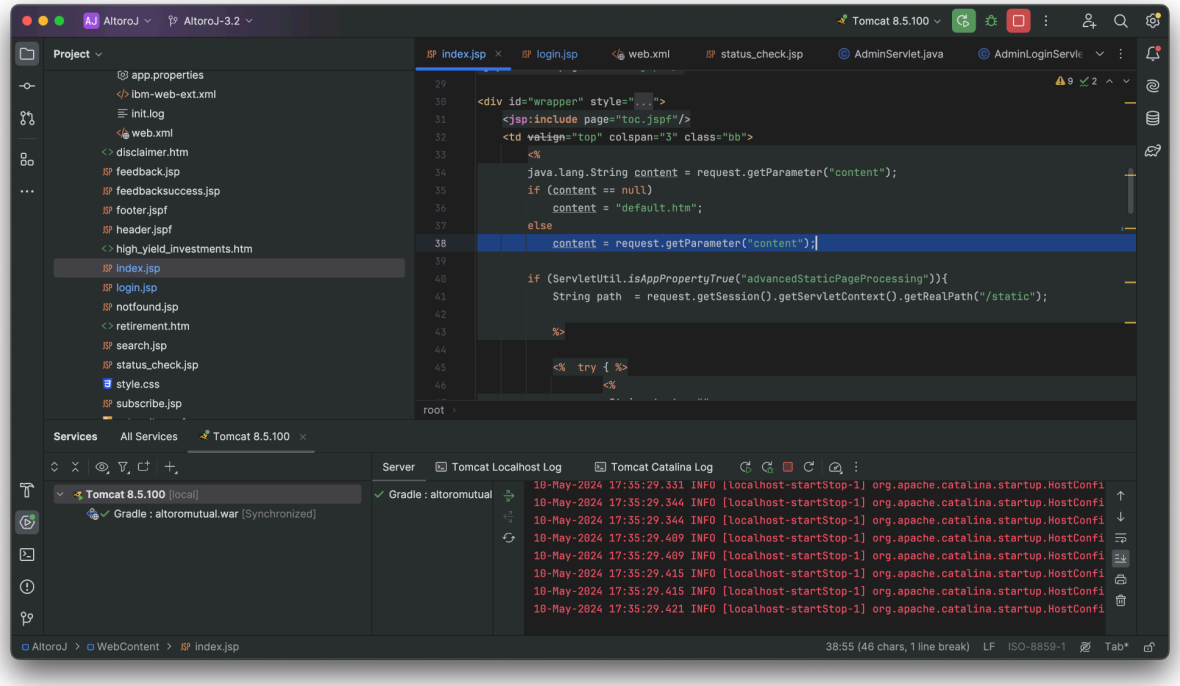
- Finding test steps
  - In the index screens(not protected), While routing to different pages content parameter is passed.

- let’s inject any HTML code to it

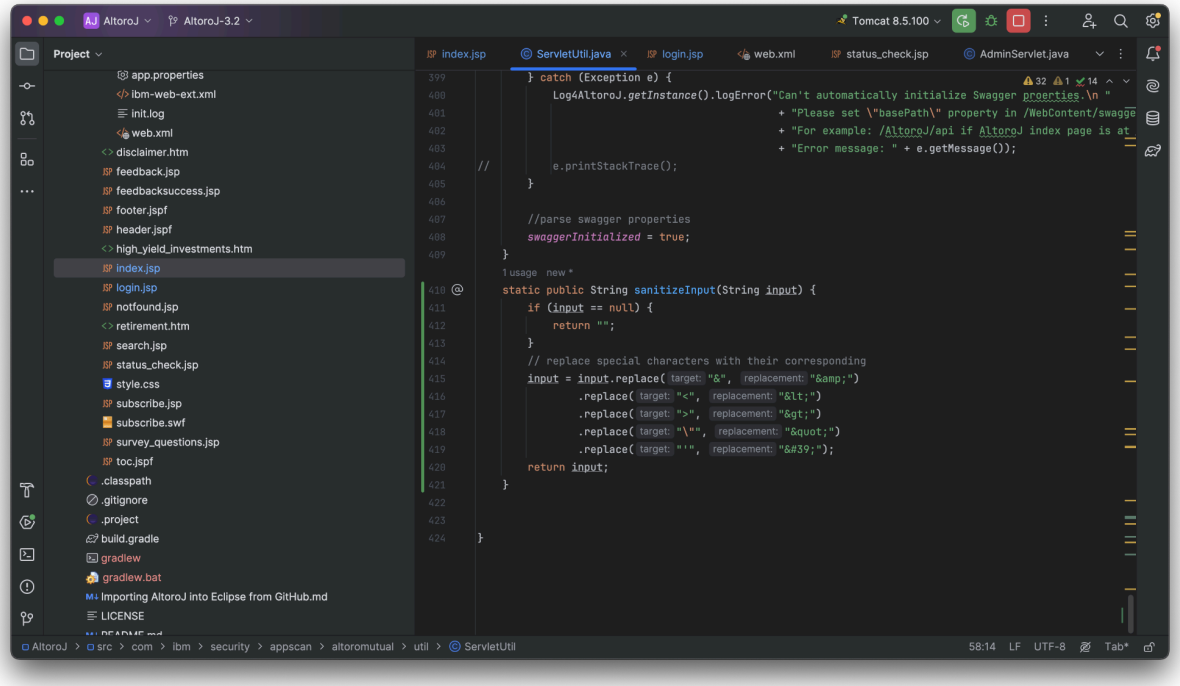



- HTML injection detected 😕

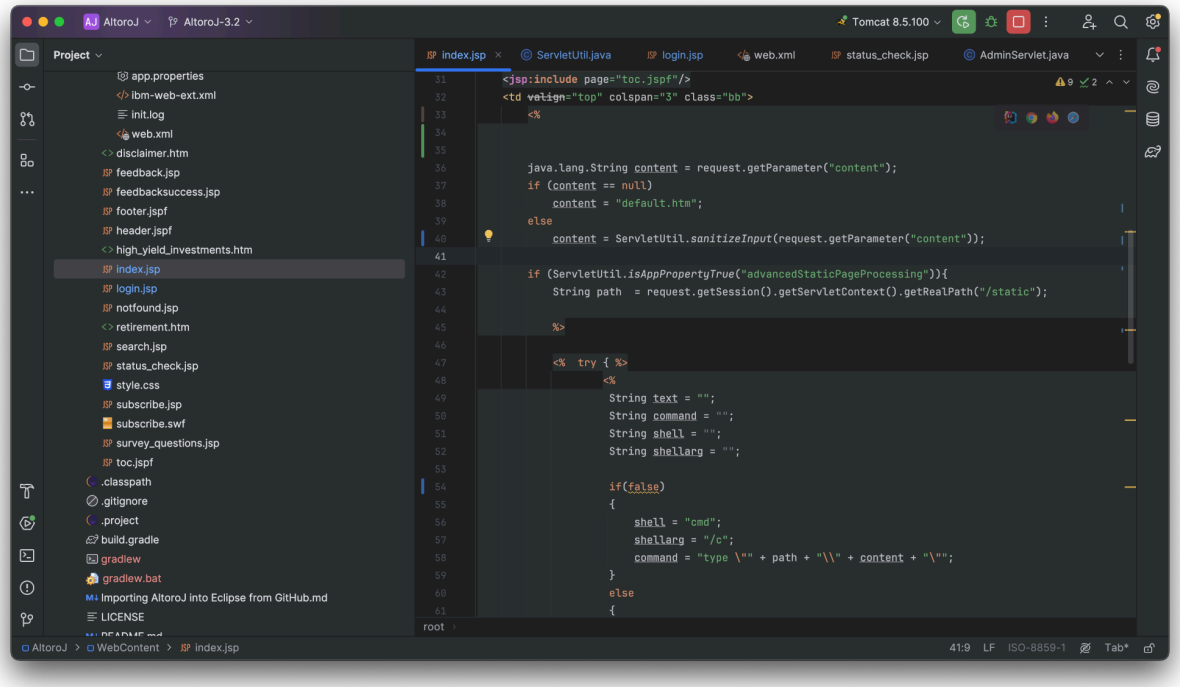
- Fixing steps
  - Go to index.jsp file in the source code
  - content parameter not sanitized



- Go to ServletUtil to implement the sanitize function for inputs

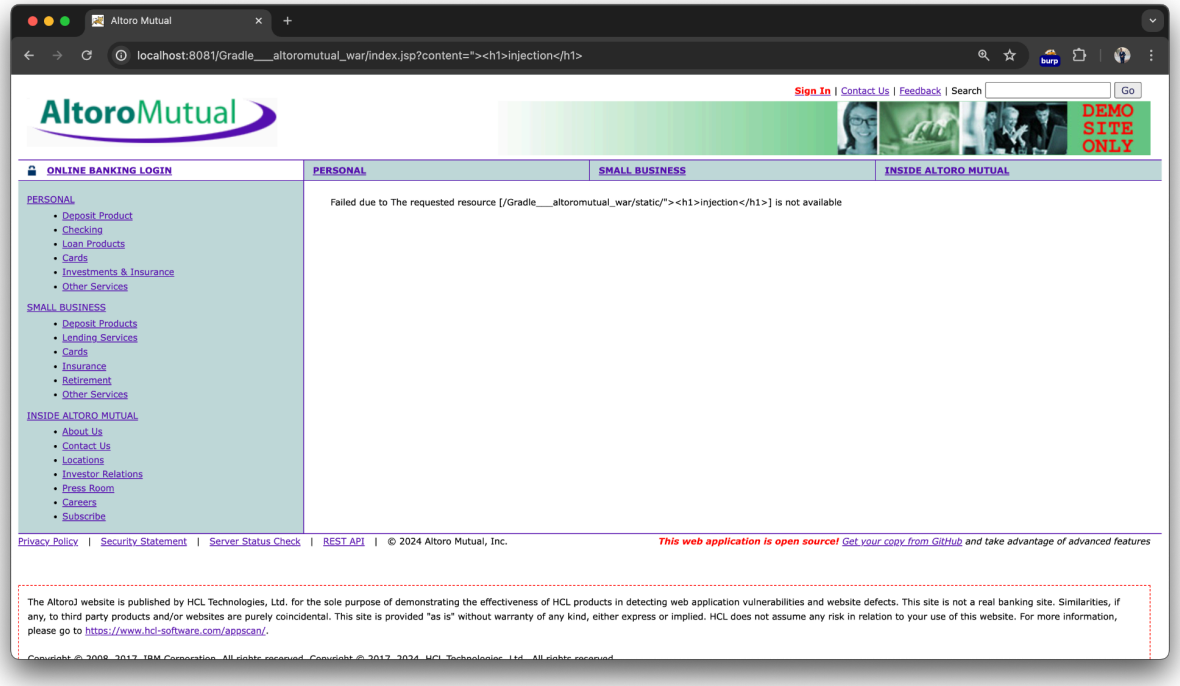



- Use the sanitizeInput function implemented to sanitize content parameter against HTML injection




- Finding RE-TEST steps

- Let’s try the same injection on the same parameter in the index.jsp page

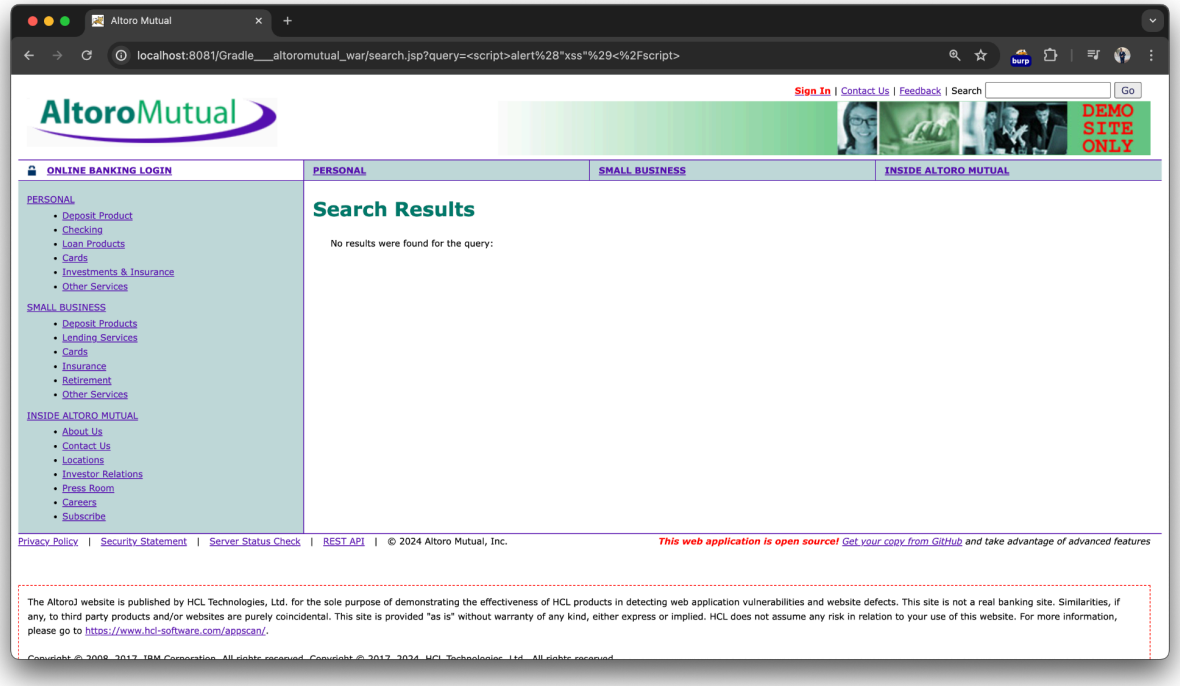



- Resolved 😀

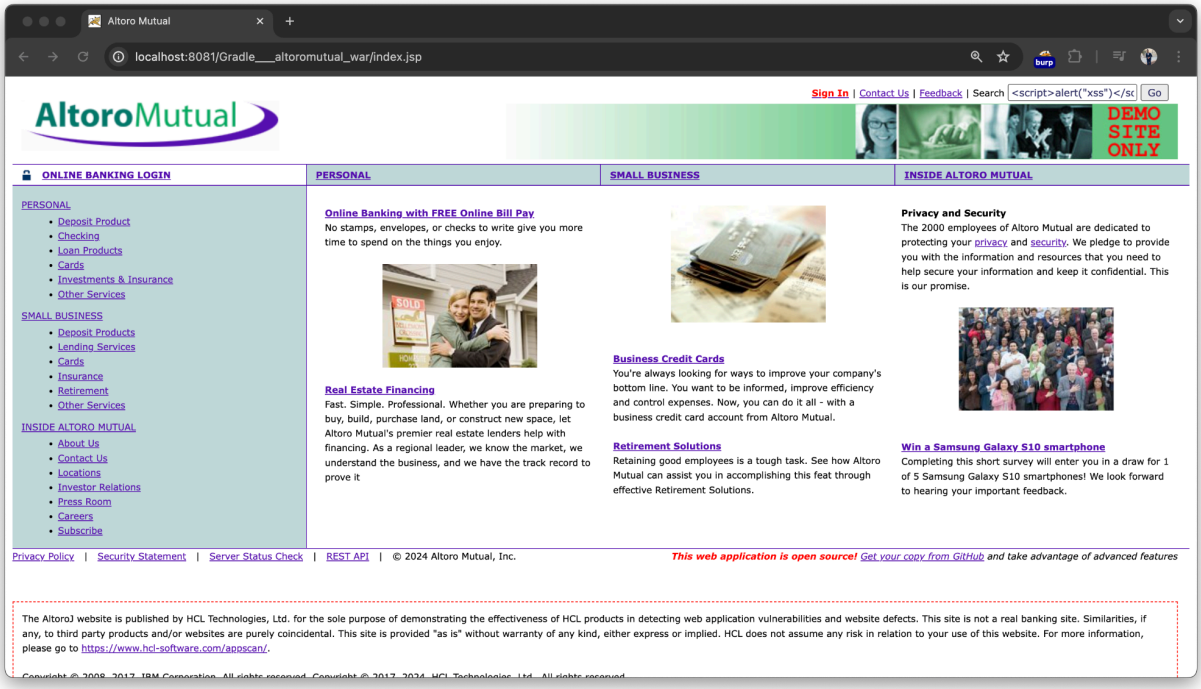
## ID12

- Finding test steps

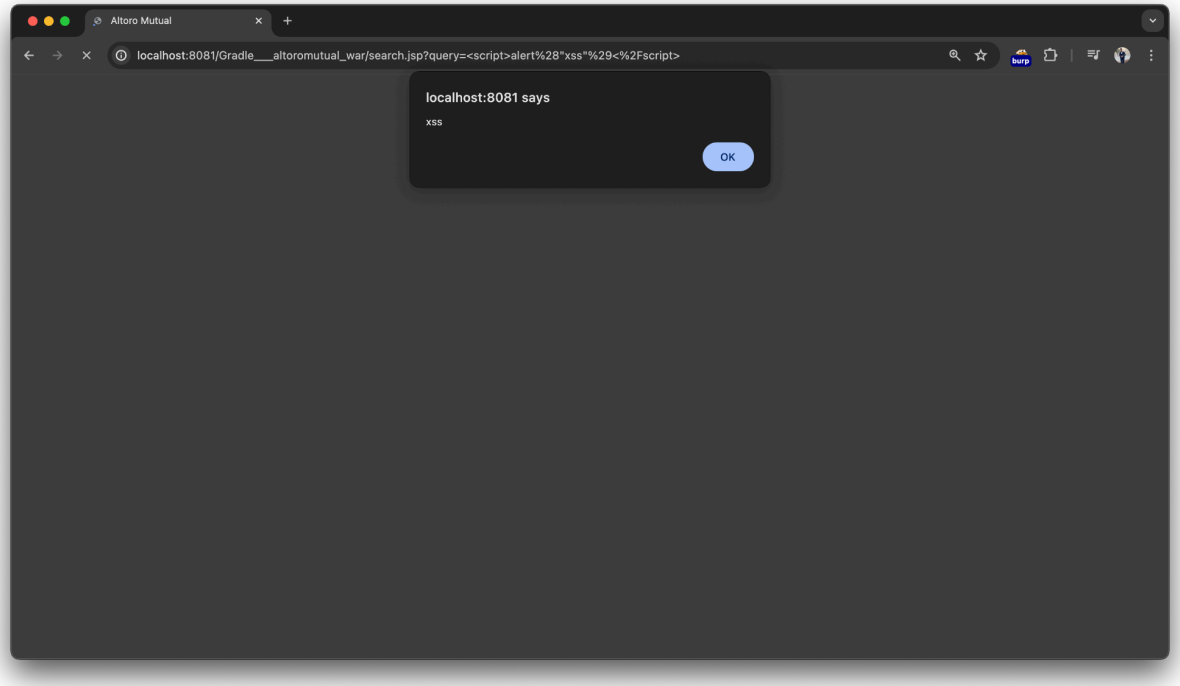
- In the index page go to the search in top right of the page



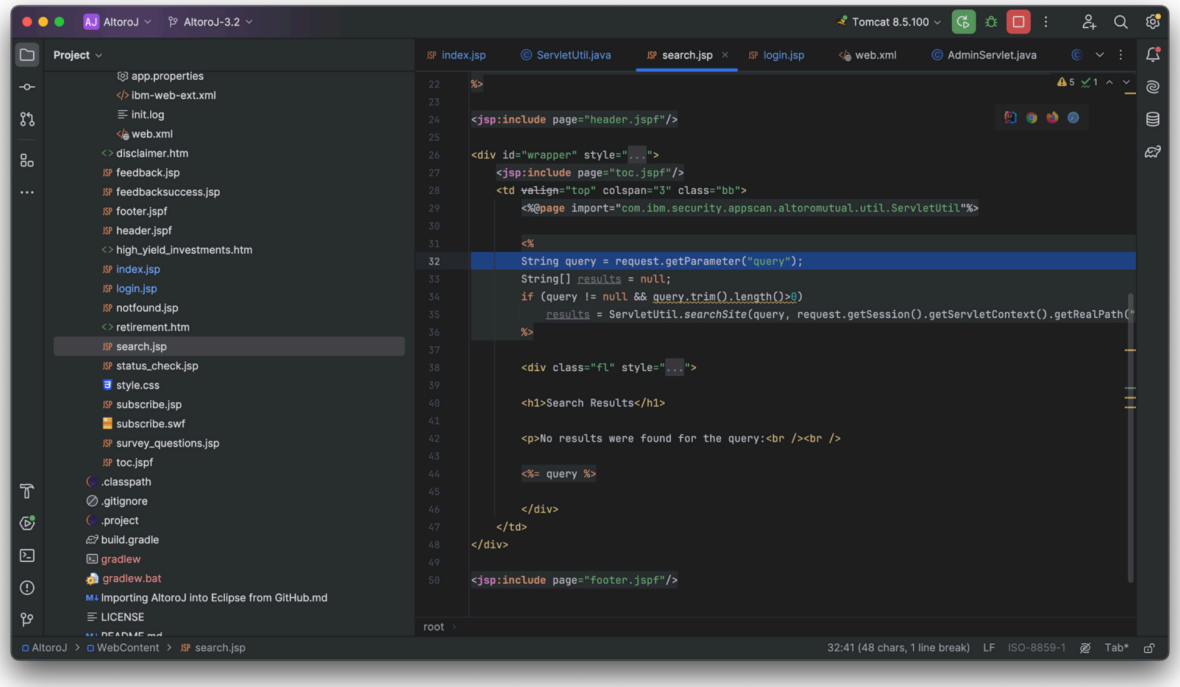

- Try to write a JS script with: <script>alert("xss")</script>



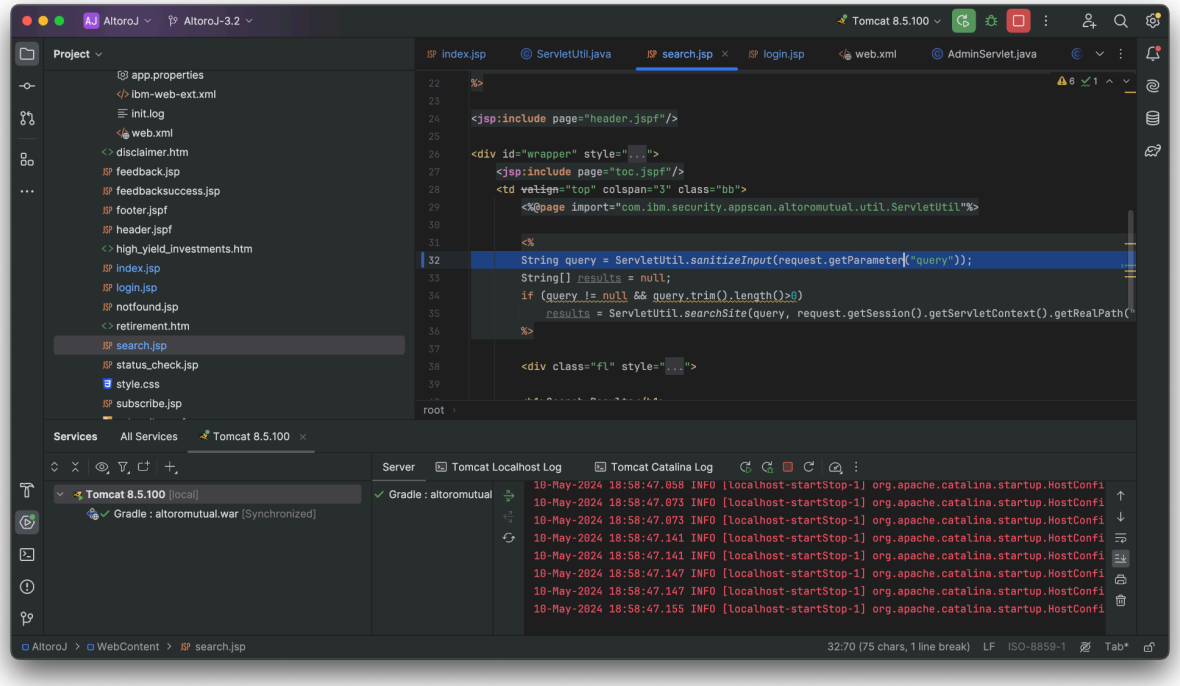
- XSS detected and executed the alert function



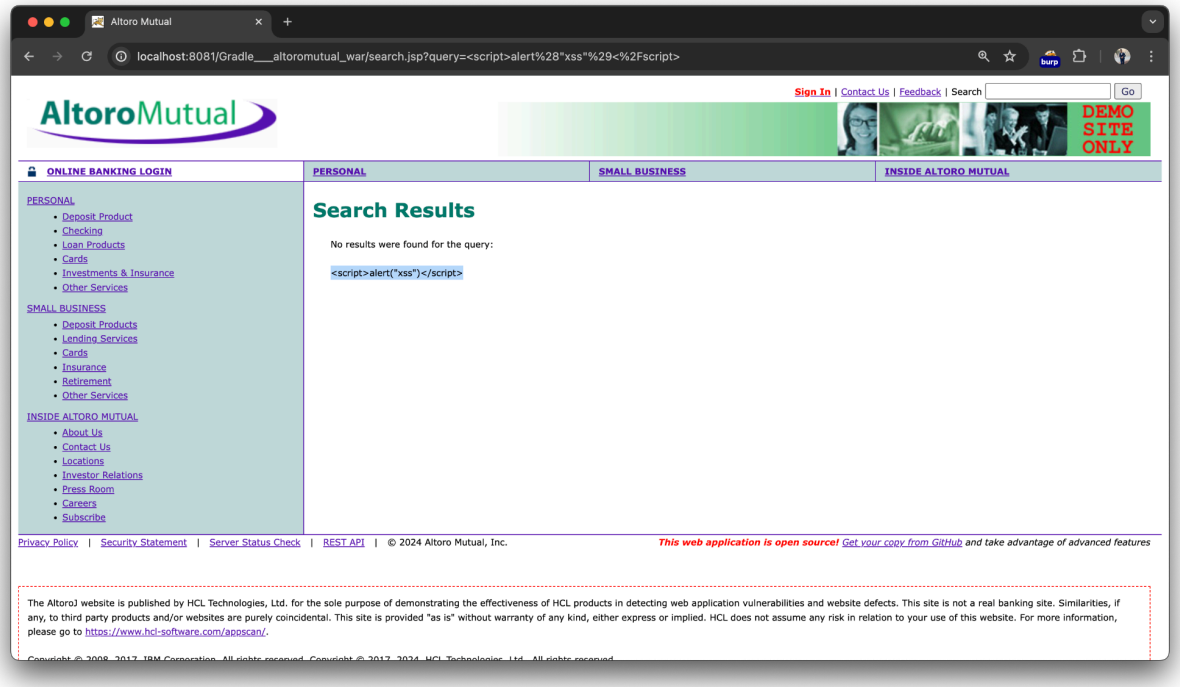

- Fixing steps
  - Go to search.jsp function in the source code
  - the query parameter is not sanitized or encoded



- Use the same sanitize function implemented before in ServletUtil




- Finding RE-TEST steps
  - Let's try the same script in the input search

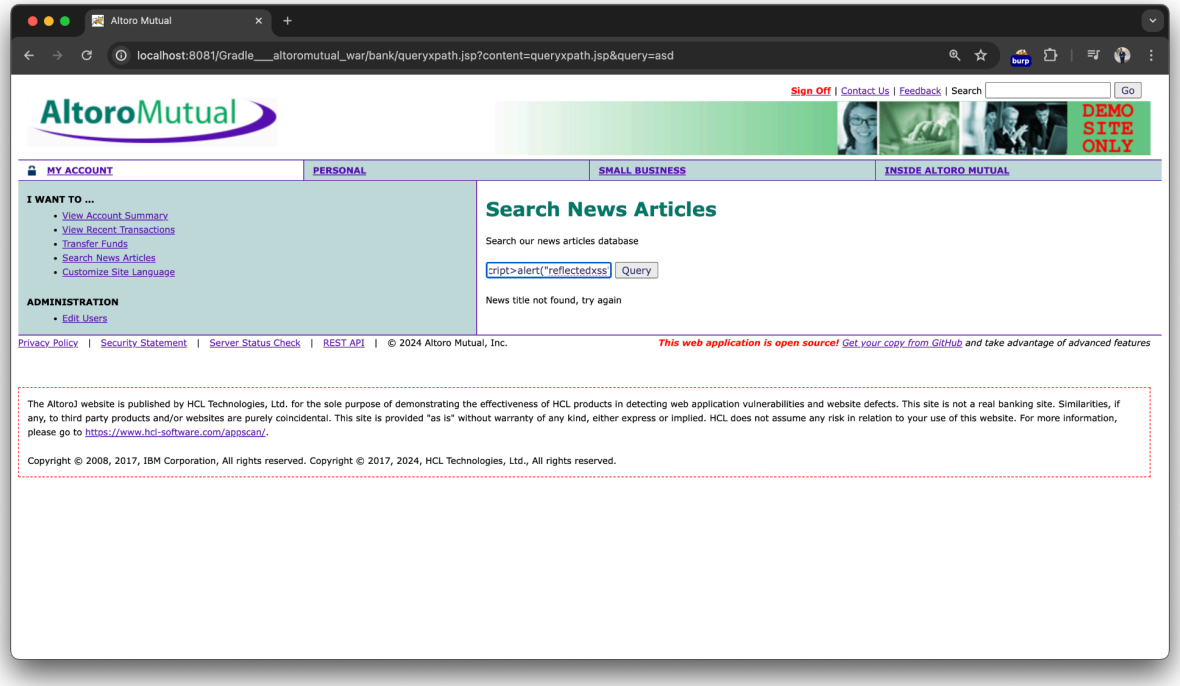



  - Encoded input and resolved XSS 😀

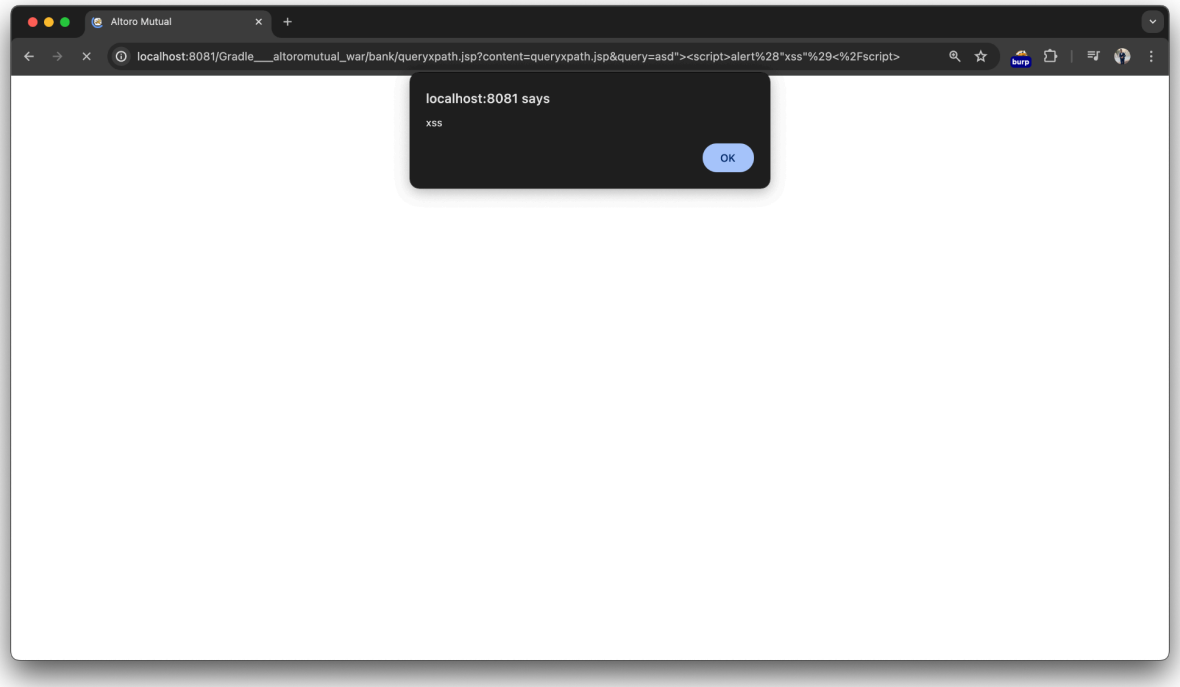
# ID13

- Finding test steps
  - Sign in in the site
  - Go to search articles route

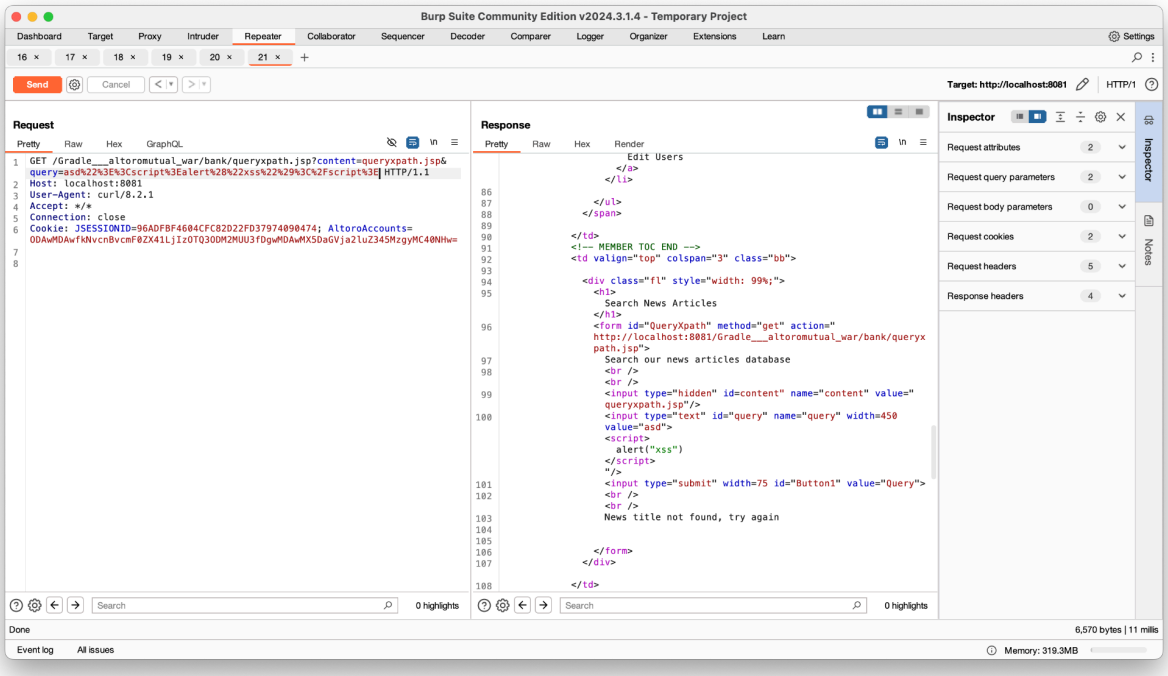
- Let’s inject a script in the search input



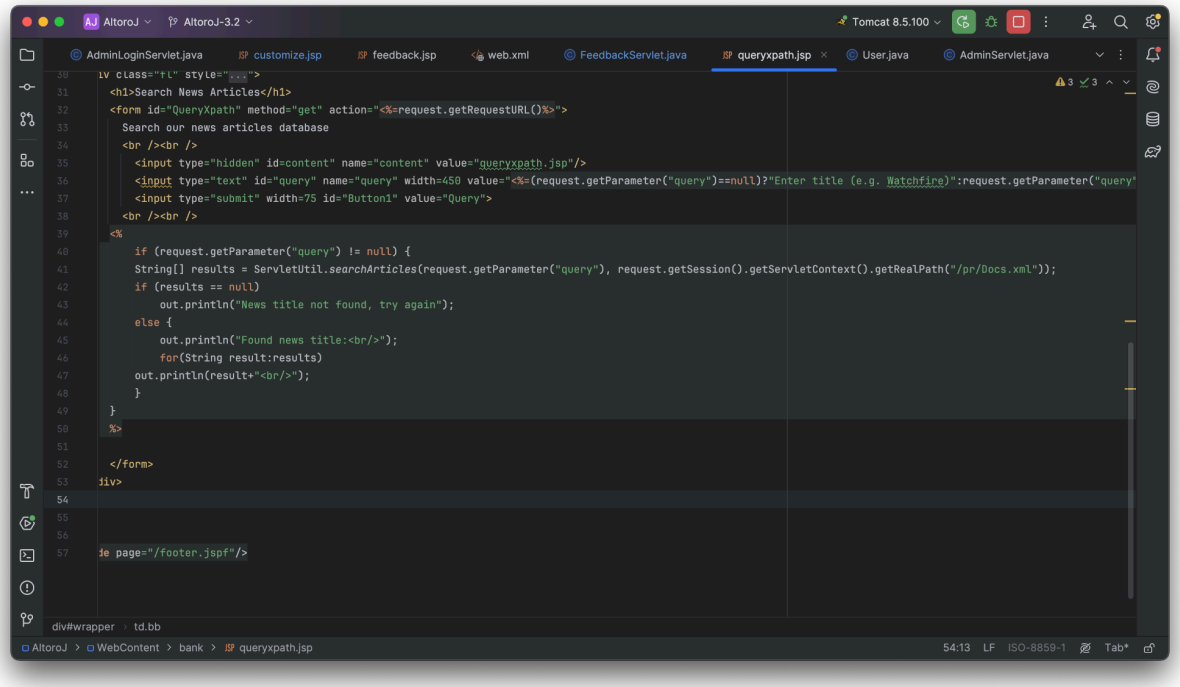

- The script is executed RXXS 😵



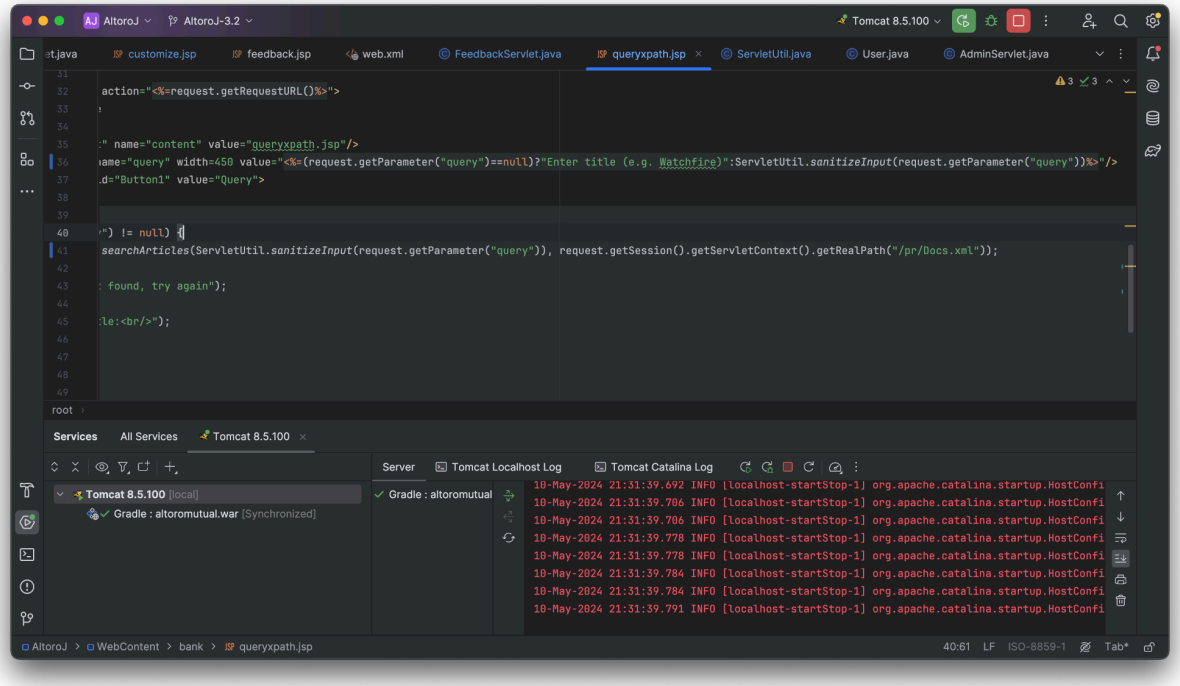

- Use BurbSuite to intercept the request



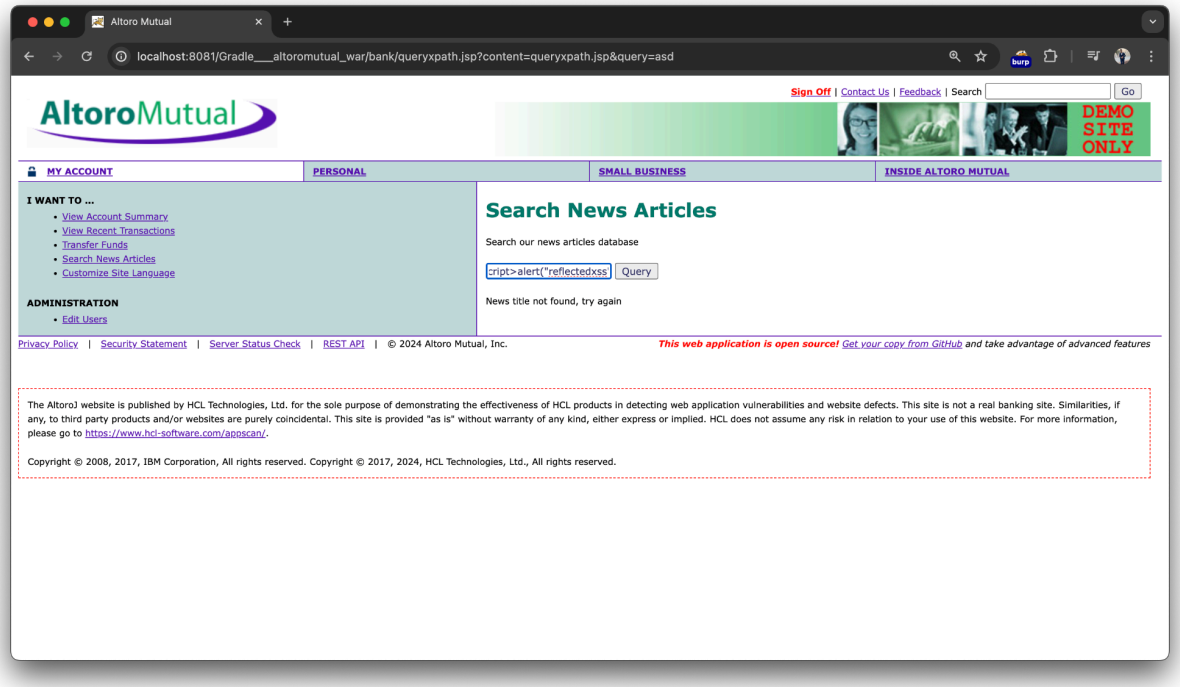

- Fixing steps
  - Go to querxpath.jsp in bank route
  - The query input is not sanitized



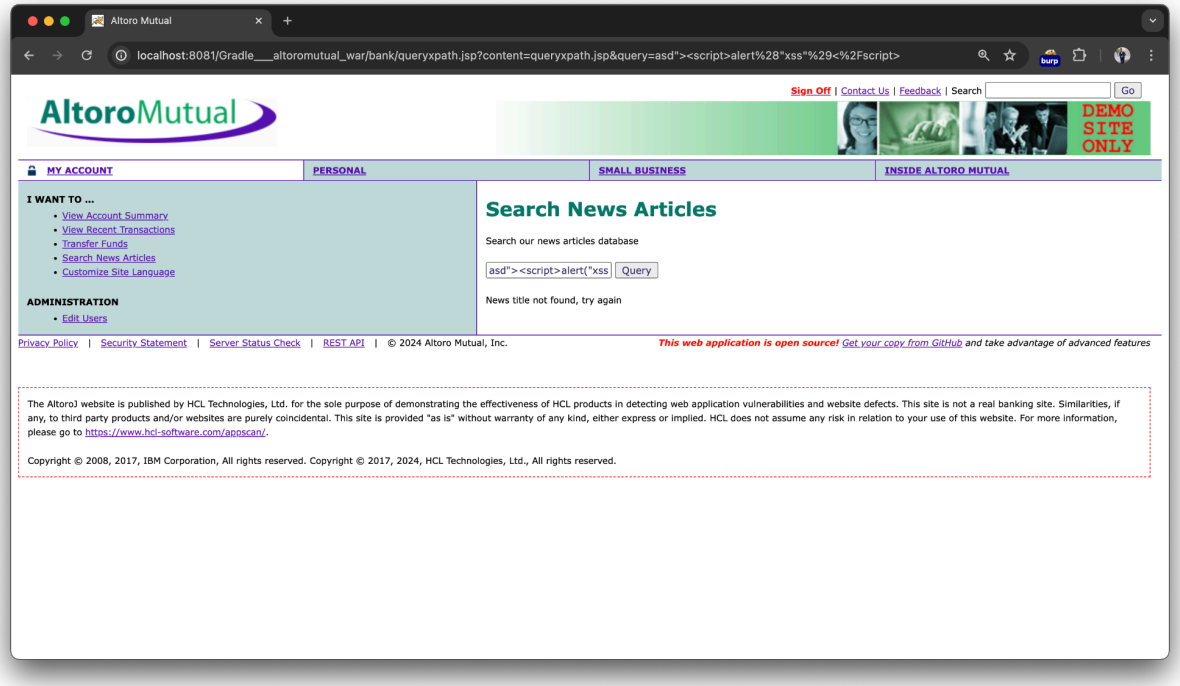
- Use the same sanitize function implemented before in ServletUtil




- Finding RE-TEST steps
  - Let's do the same steps and test the script in the HTML



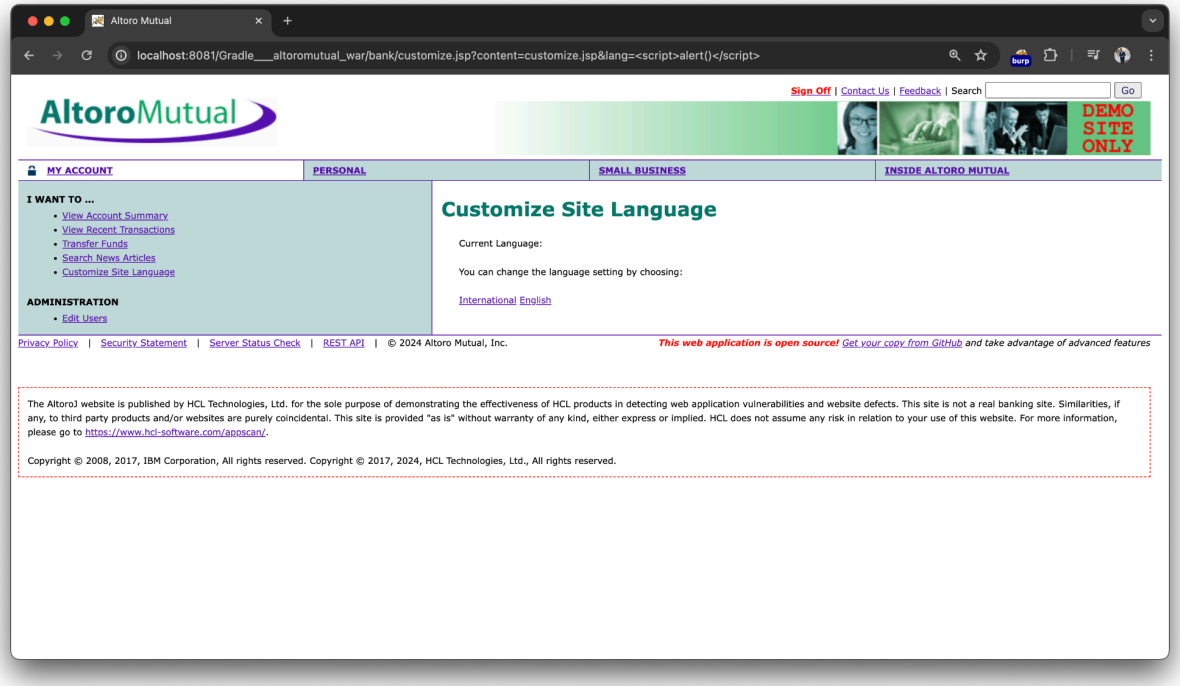
- Resolved RXSS 😀



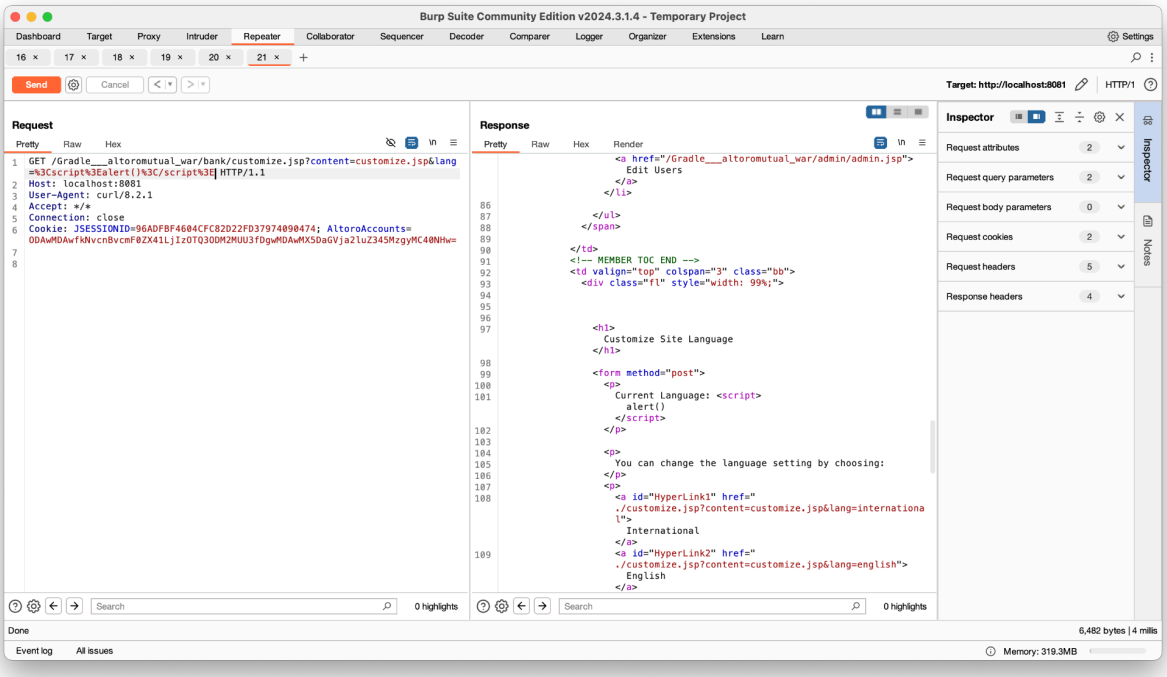

## ID14

- Finding test steps

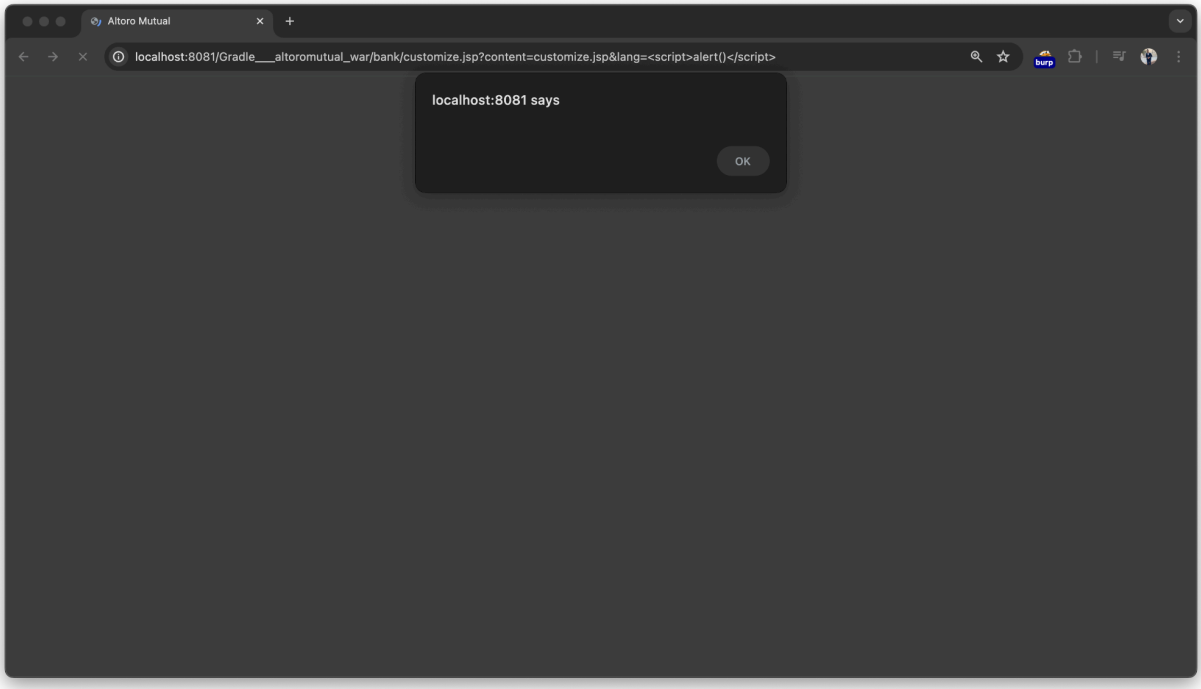
- Go login in to the site and go to Customize Language page



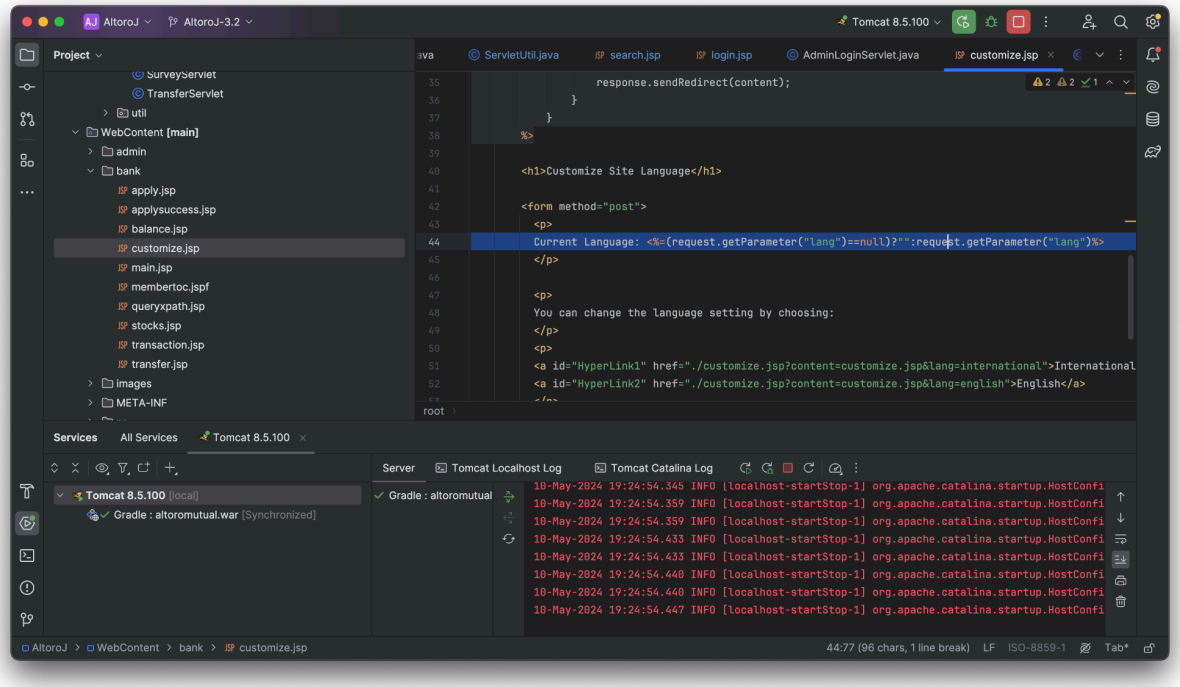

- Use BurbSuite to intercept the request with injected script in lang parameter



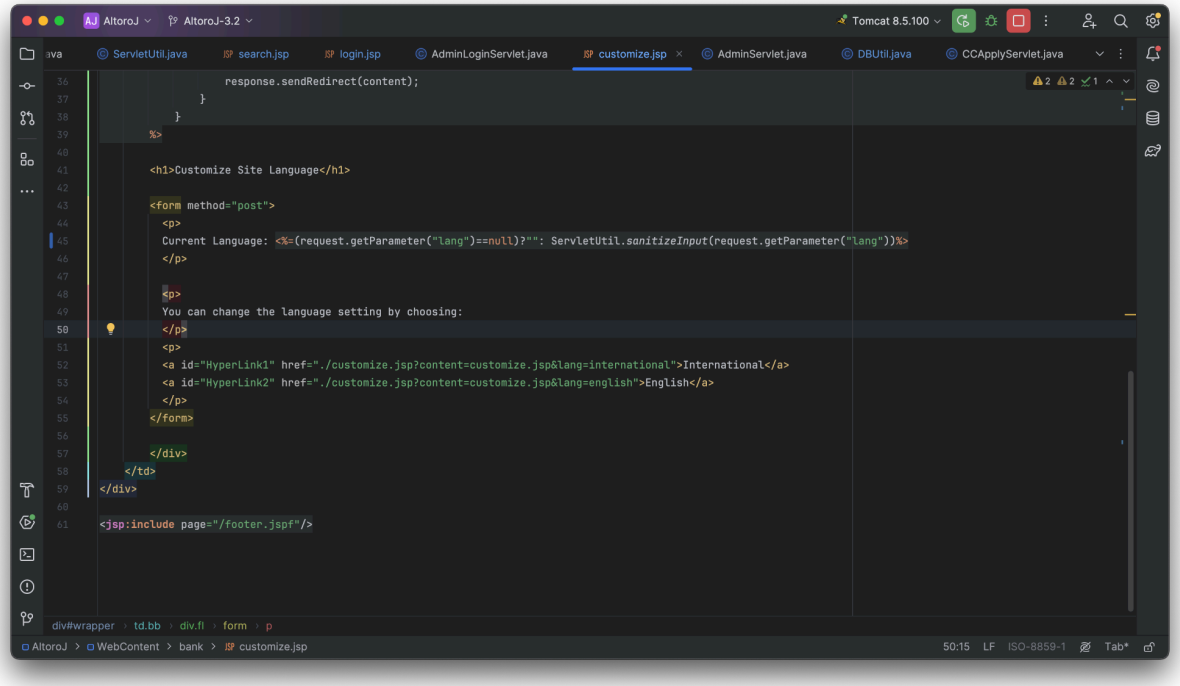
- It injected the script in the web page



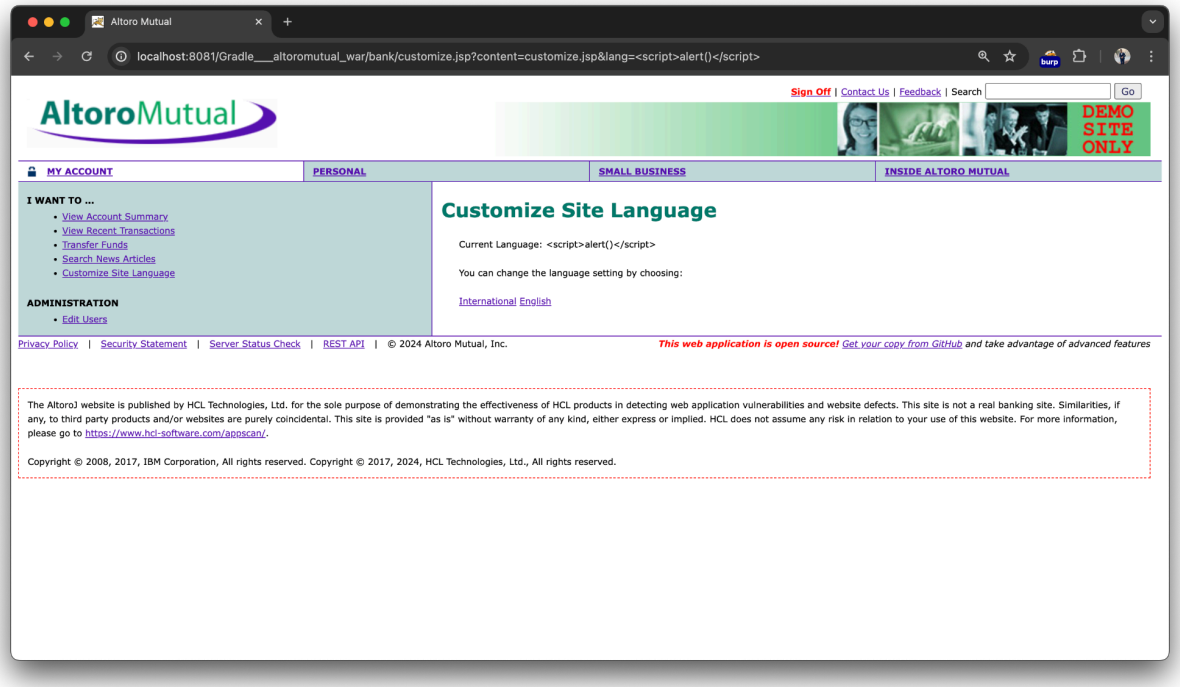

- Fixing steps
  - Go to customize.jsp file inside the bank route
  - We found non non-sanitized lang parameter passed



- Use the sanitizeInput function in ServletUtil implemented before



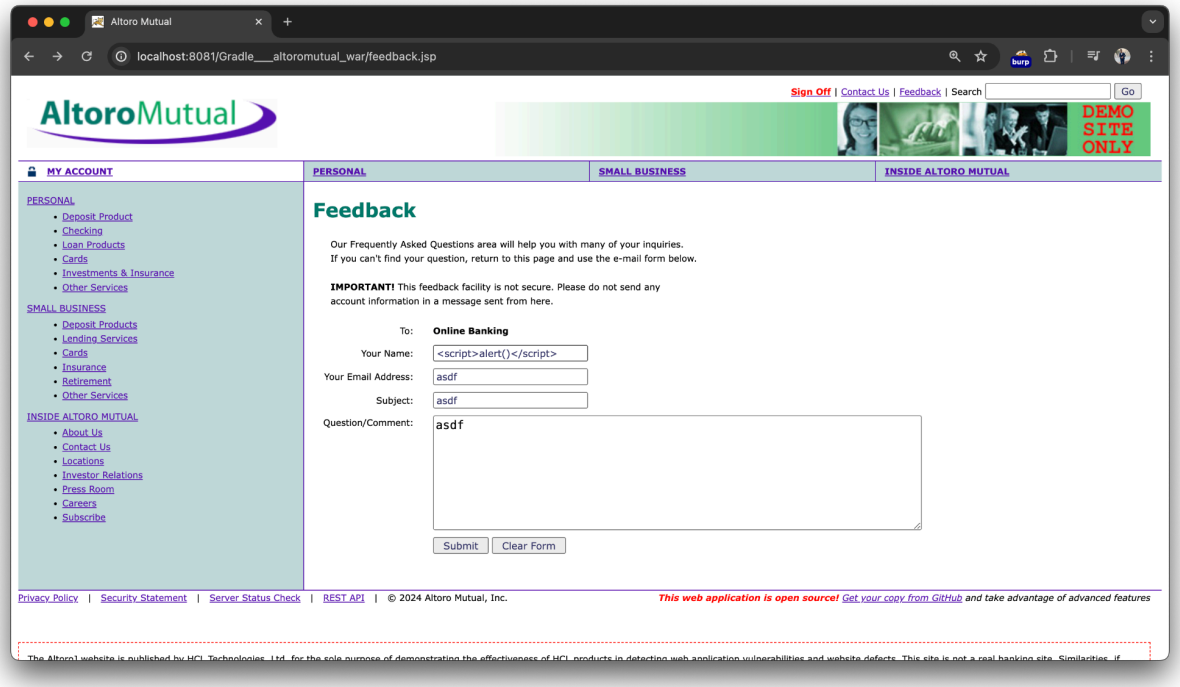

- Finding RE-TEST steps
  - Test the web page on the site manually first



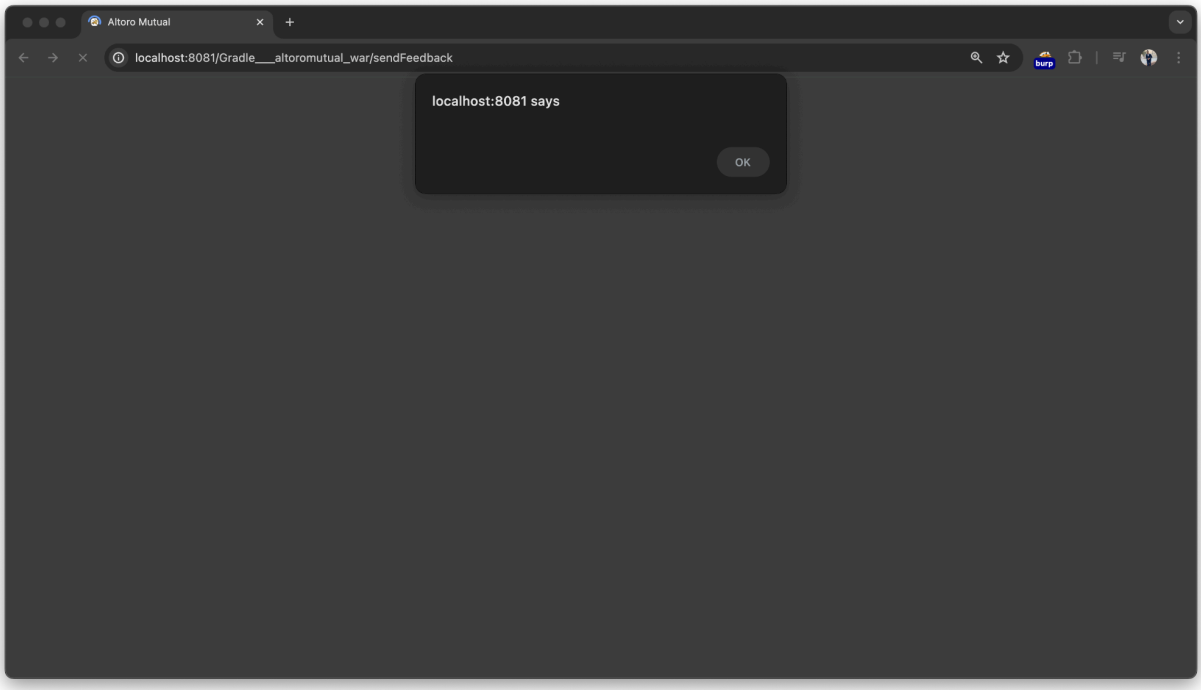

  - Resolved RXSS 😀

# ID15

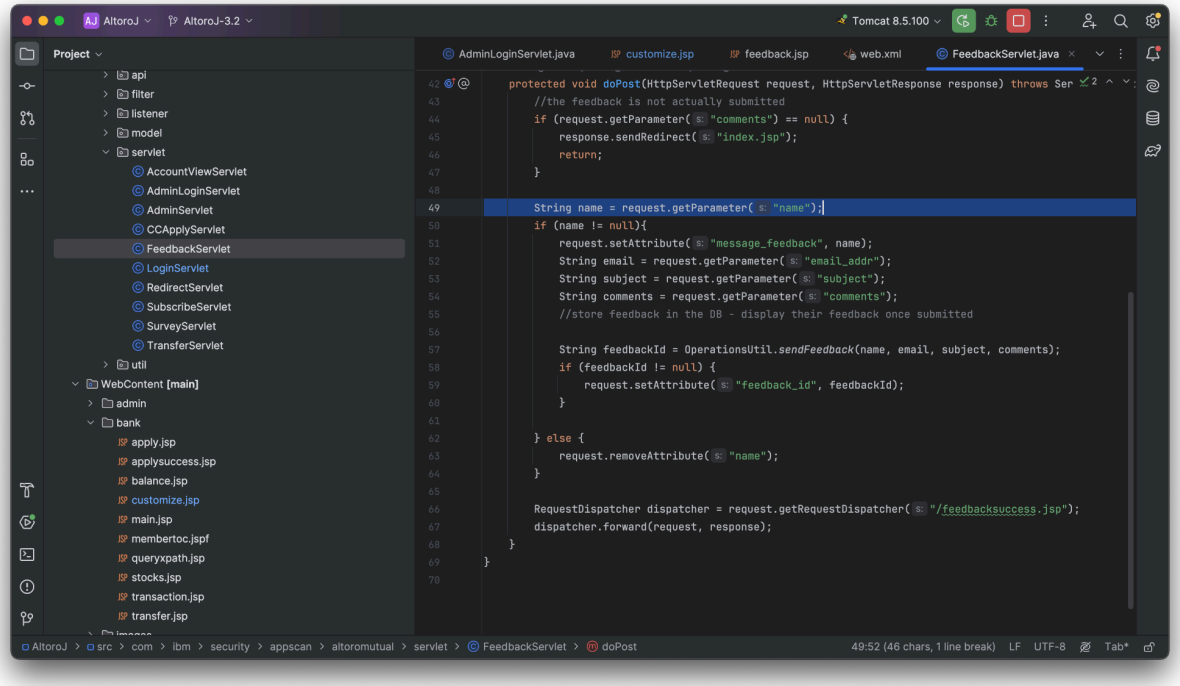
- Finding test steps
  - Go to feedback page and try inject a script in any input



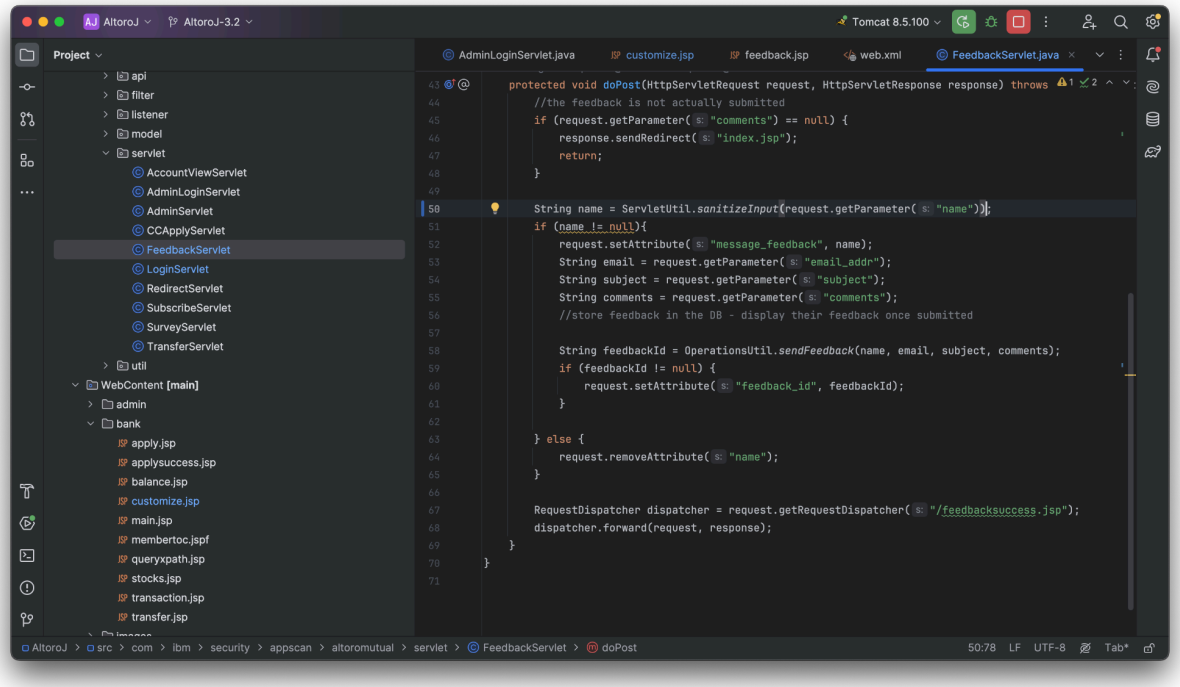

  - The script is injected and executed after submit



- Fixing steps
  - Go to FeedbackServlet file in source code
  - The name input name is not sanitized or encoded

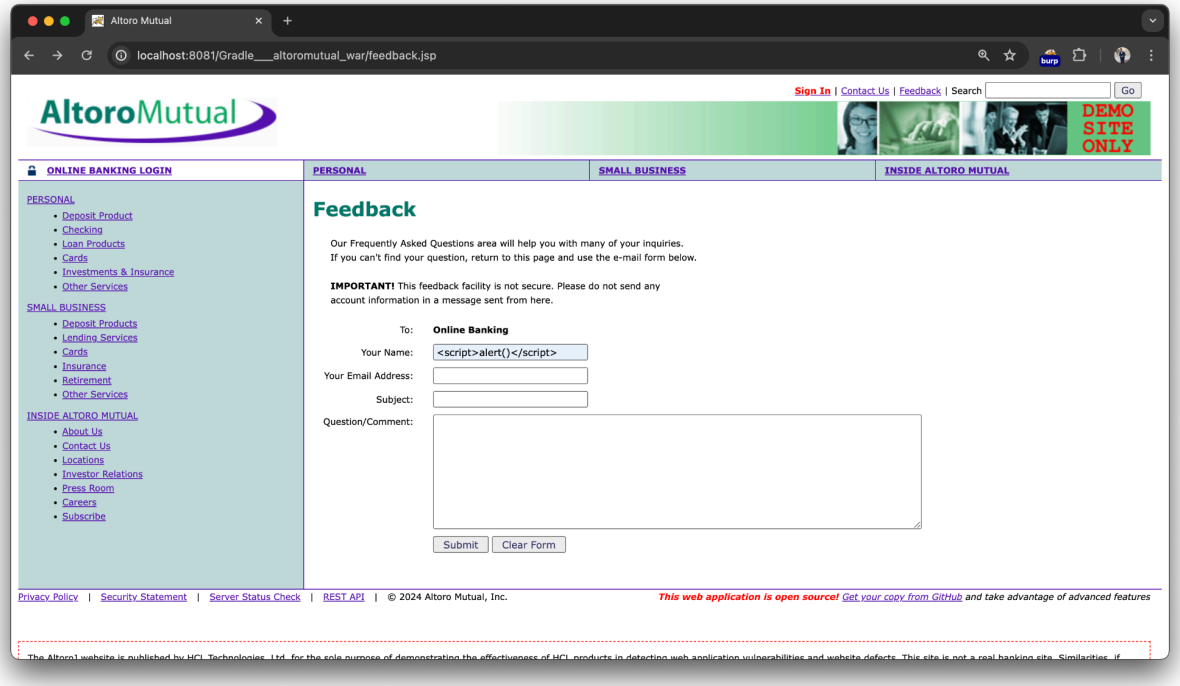



  - Use the sanitizeInput function implemented before in the ServletUtil

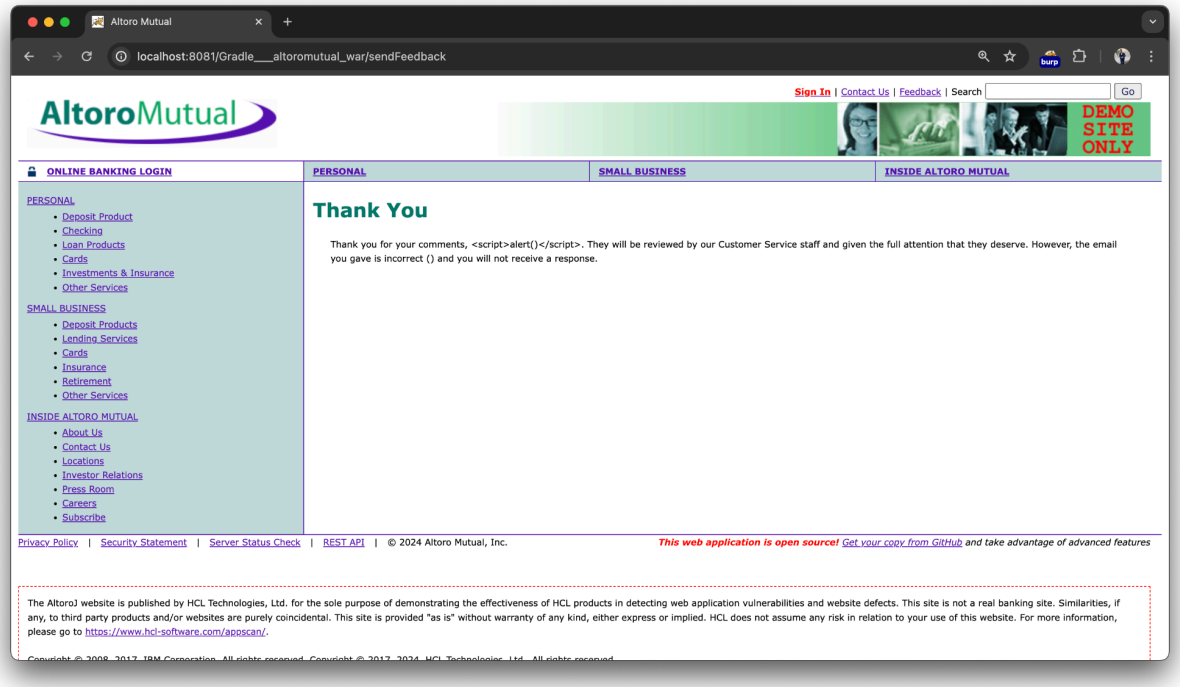



- Finding RE-TEST steps

- Let’s retest the same script on the webpage

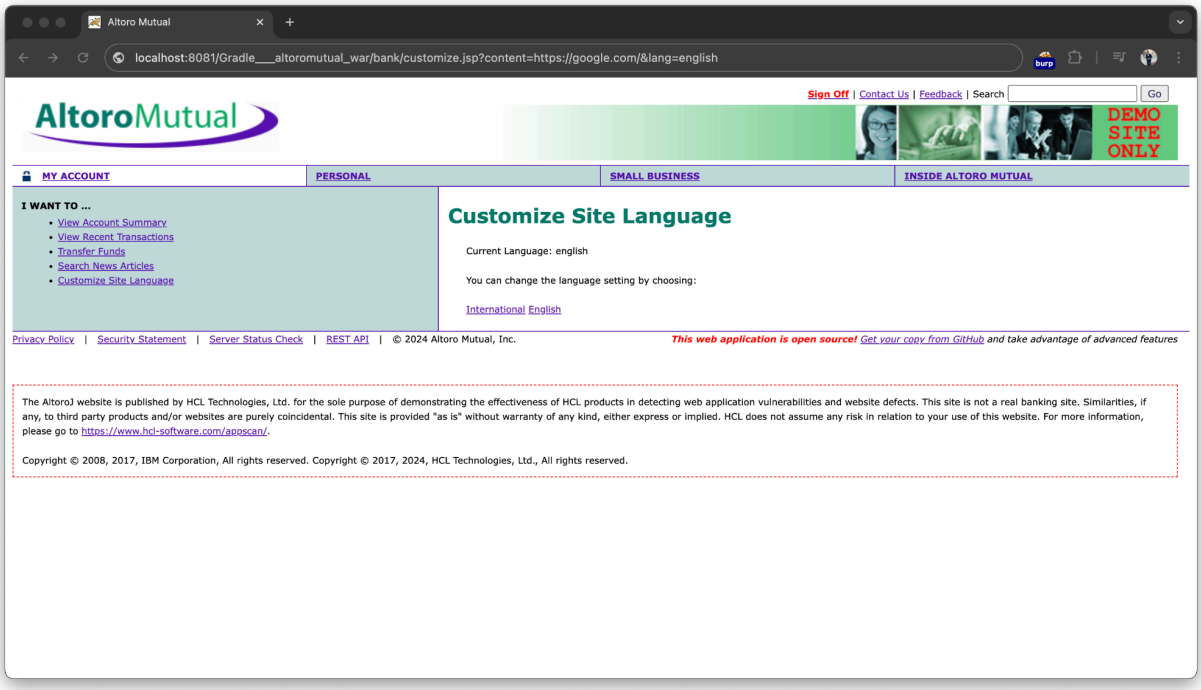



- It’s gone perfectly

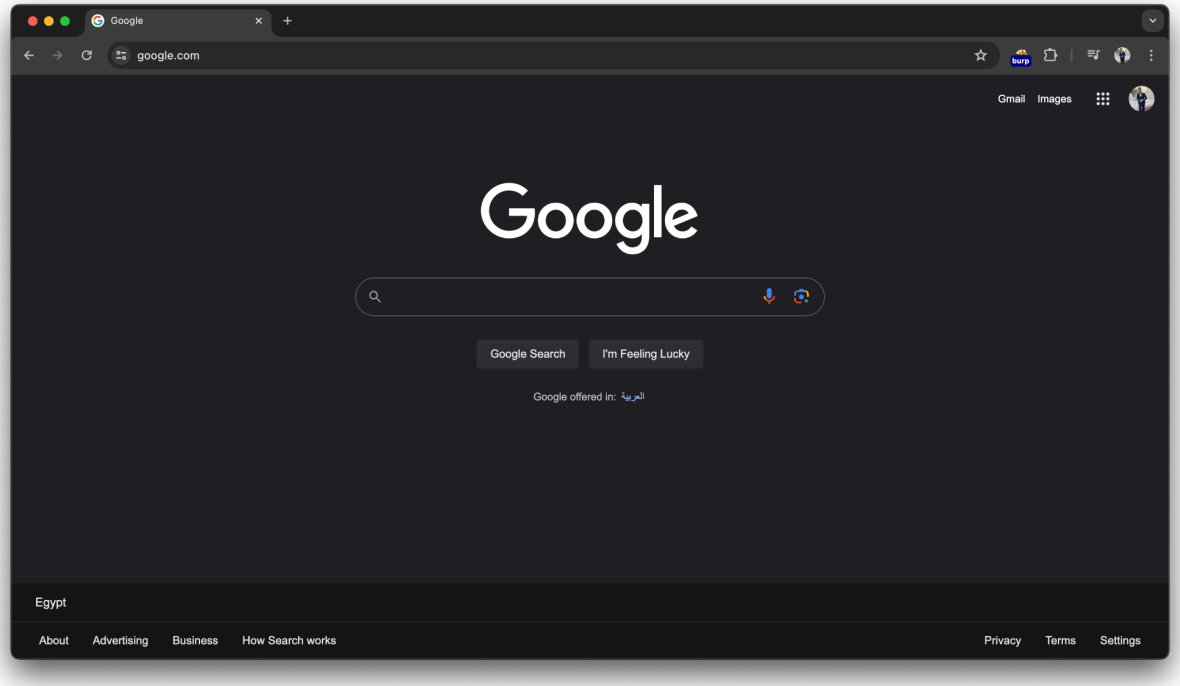



## ID16

- Finding test steps
  - Go and sign in
  - Go to customize language route
  - Try to change the lang parameter with any link in our case (https://google.com)

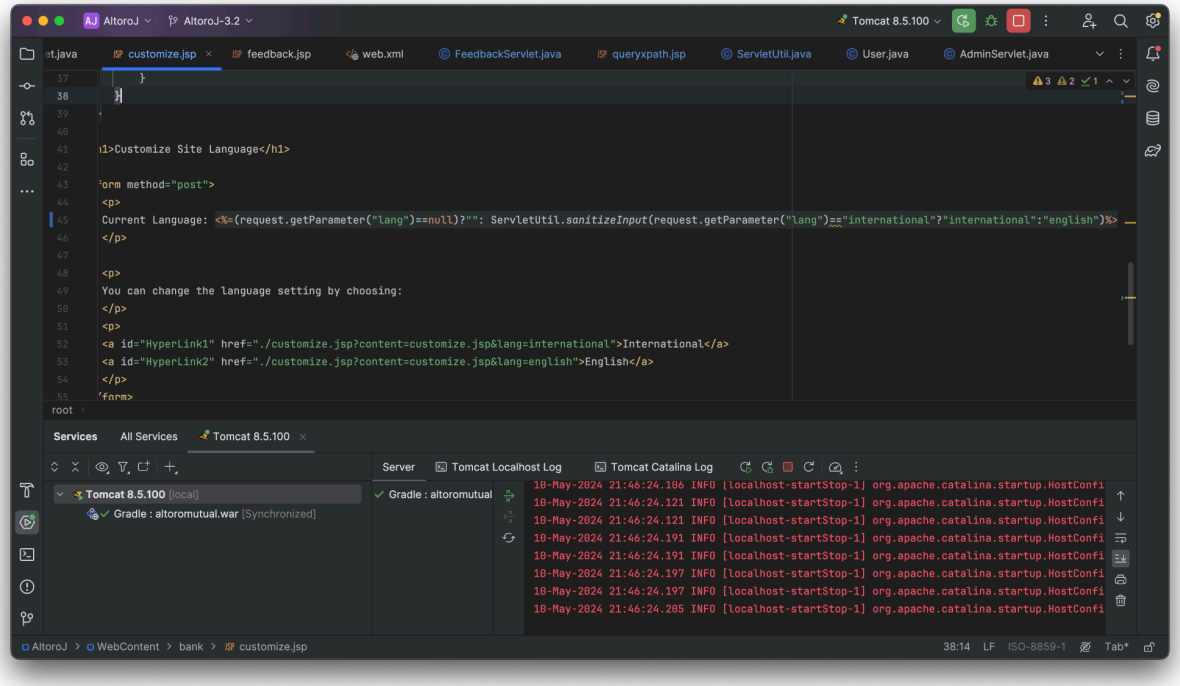


- The site is redirected to the specified link 😡

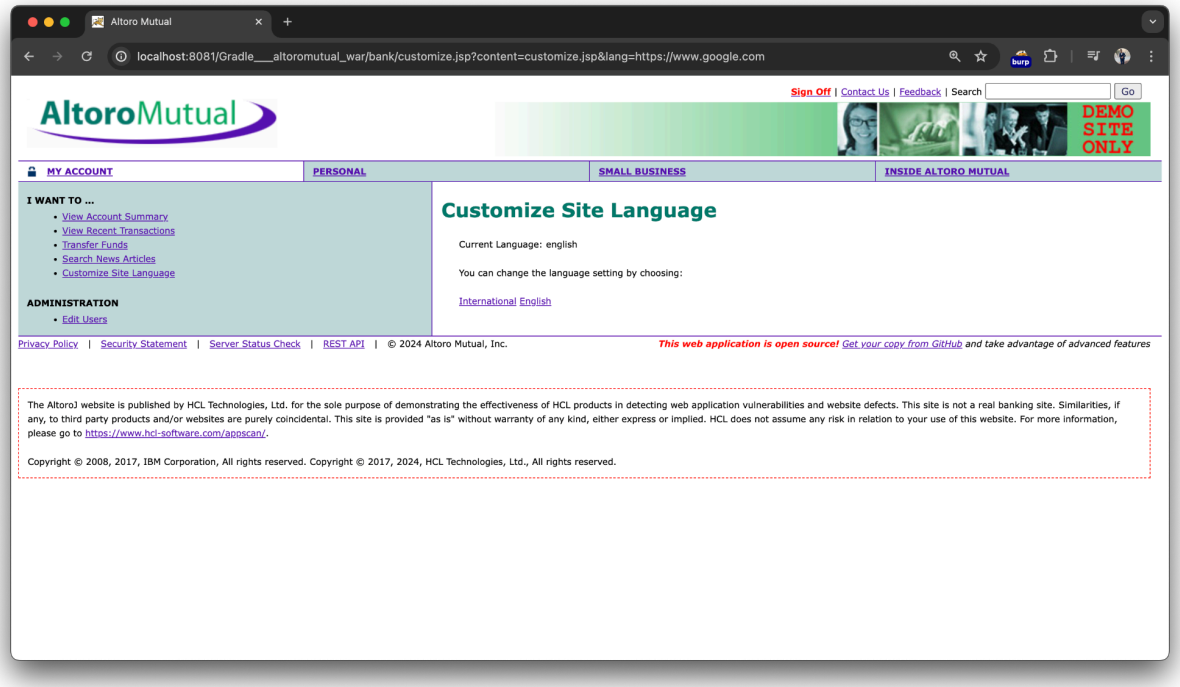



- Fixing steps
  - Go to customize.jsp file in bank route
  - We want to prevent redirection so escape or sanitize input, additionally we want to restrice the languages to be either “english” or

“international” as written in the code



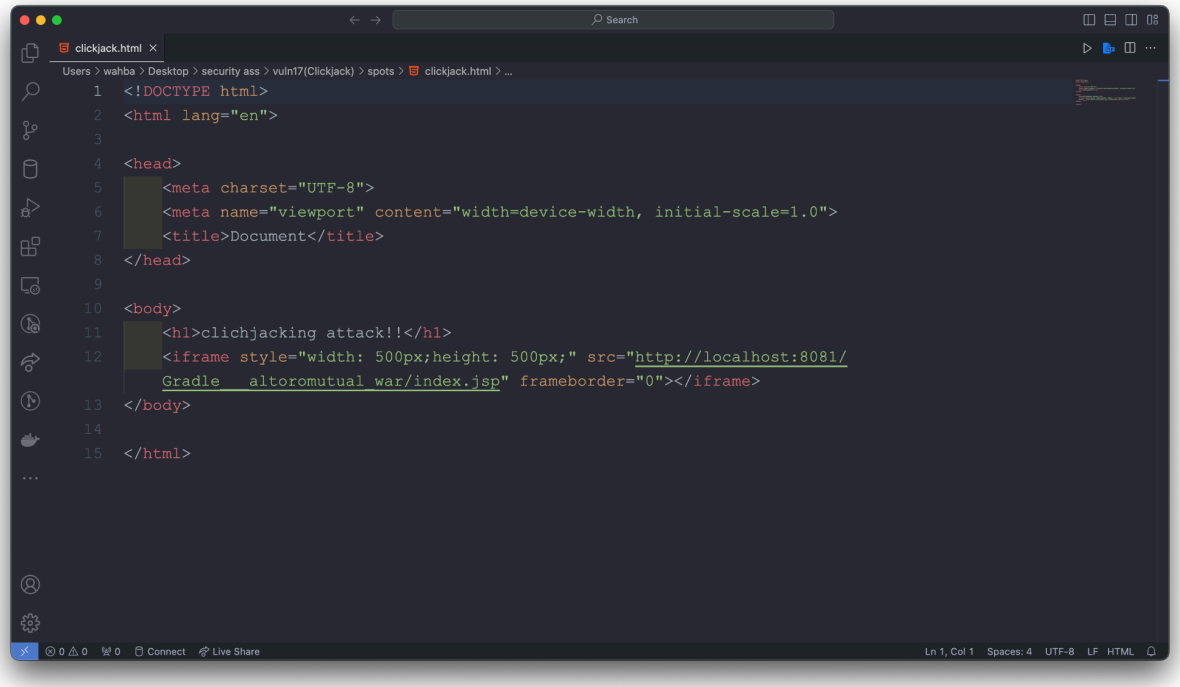

- Finding RE-TEST steps
  - Go and make the same request with the same payload of the link



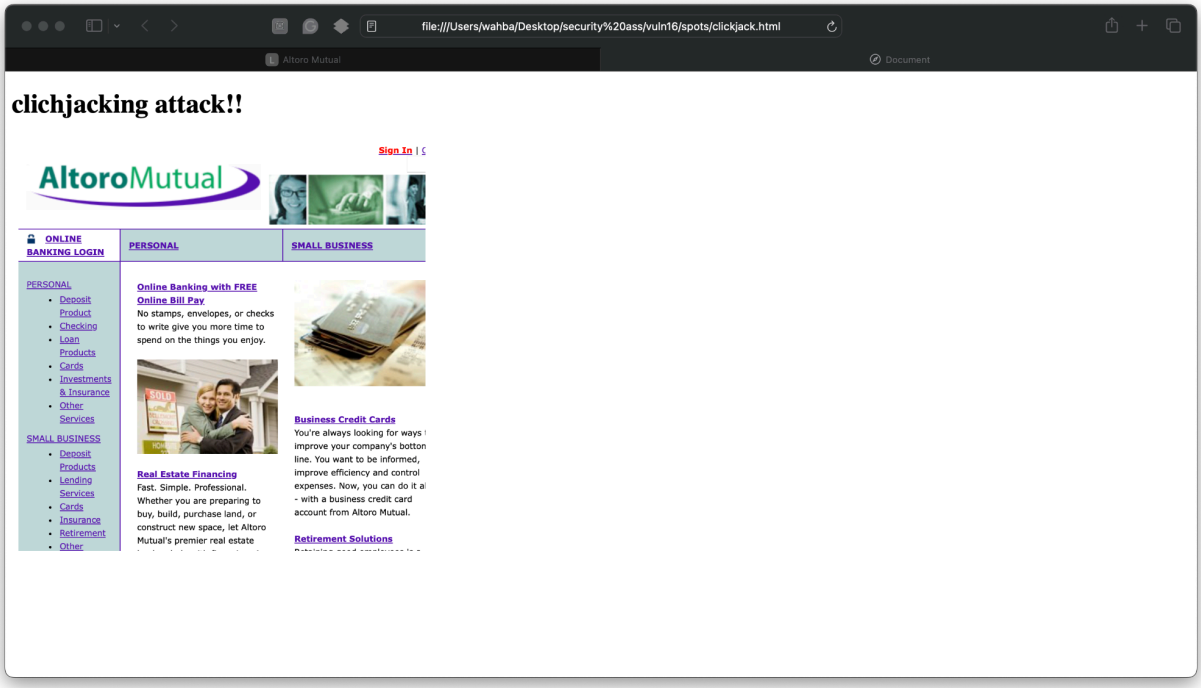

  - Redirection is resolved and default value is english as shown 👍

## ID17

- Finding test steps
  - Tried to get the base URL of the web page and inject it in an iframe in an HTML file to see if it causes the issue or not

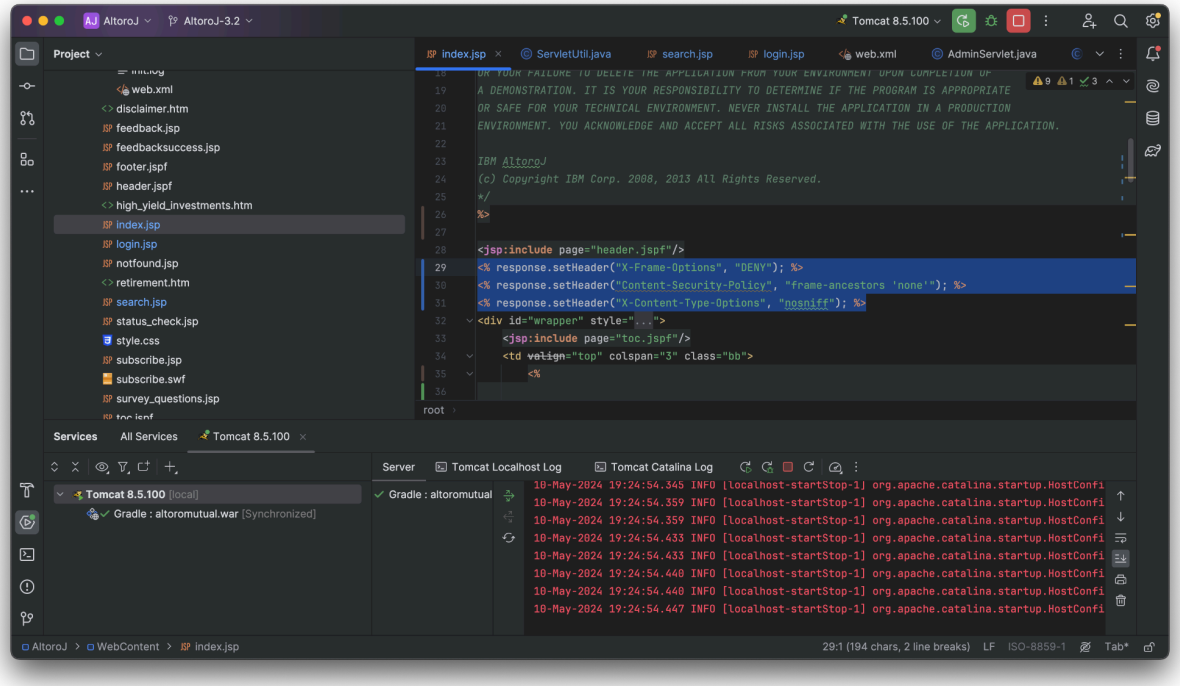


- Opened this HTML file in the browser and found it accessible



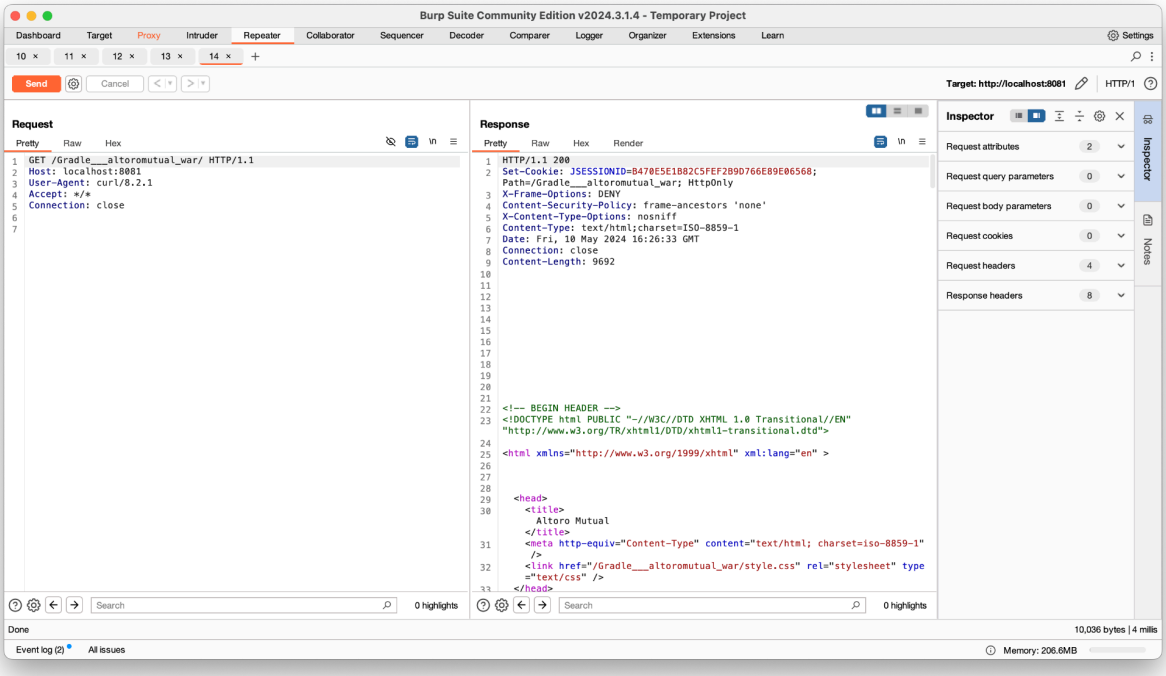

- Fixing steps
  - To mitigate clickjacking vulnerabilities we should set some response headers like `X-Frame-Options` `Content-Security-Policy` `X-Content-Type-Options`

- To avoid sniffing and clickjacking

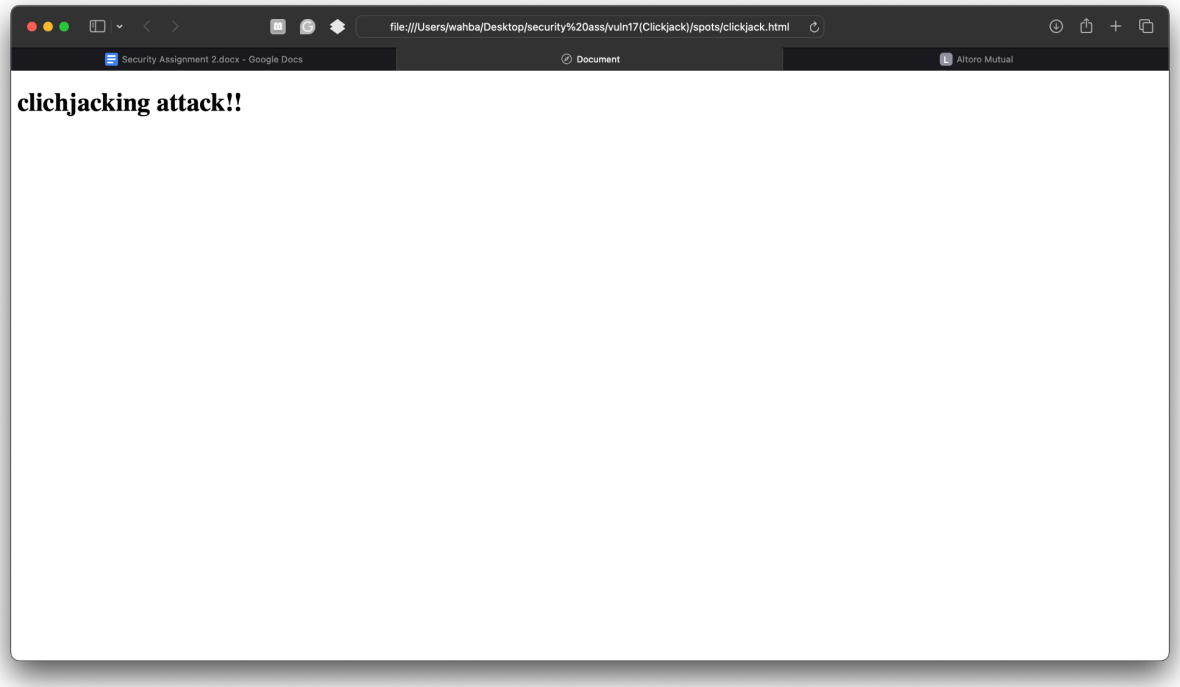



  - additionally we can put the headers in each servlet to make all pages protected against this issue
  - response.setHeader("X-Frame-Options", "DENY");
  - response.setHeader("Content-Security-Policy", "frame-ancestors 'none'");
  - response.setHeader("X-Content-Type-Options", "nosniff");
- Finding RE-TEST steps

- Ensure that these headers are set in response headers lets intercept the request on BurbSuit



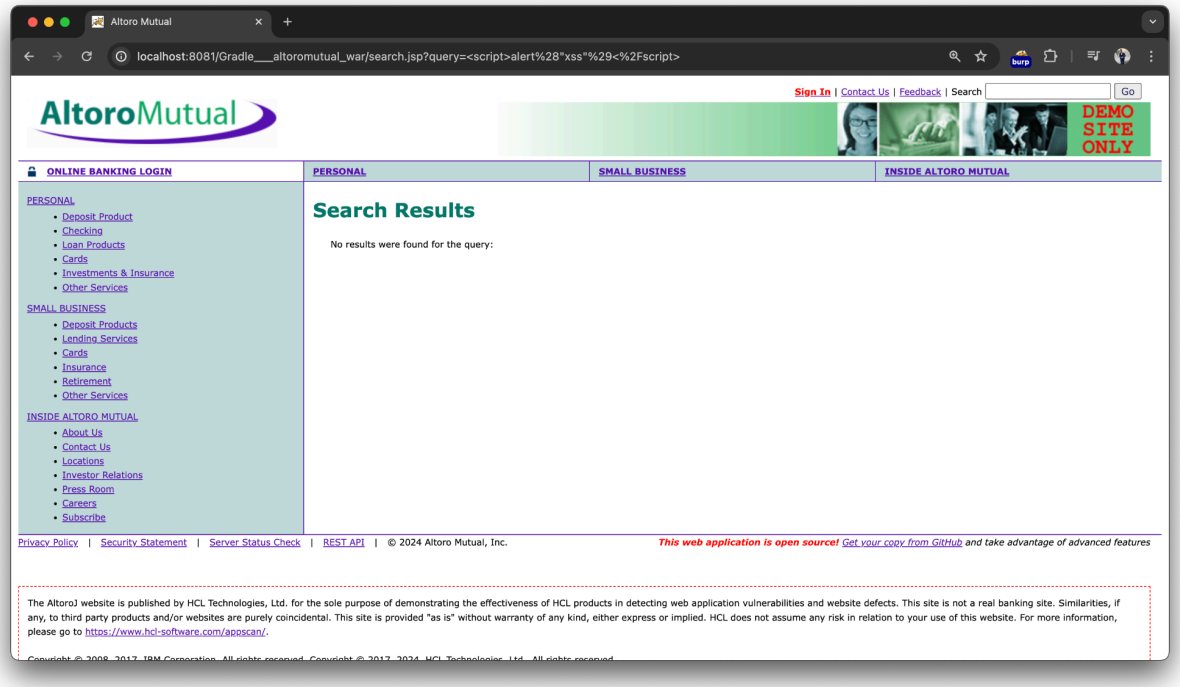

- All right! let’s check the HTML file we created



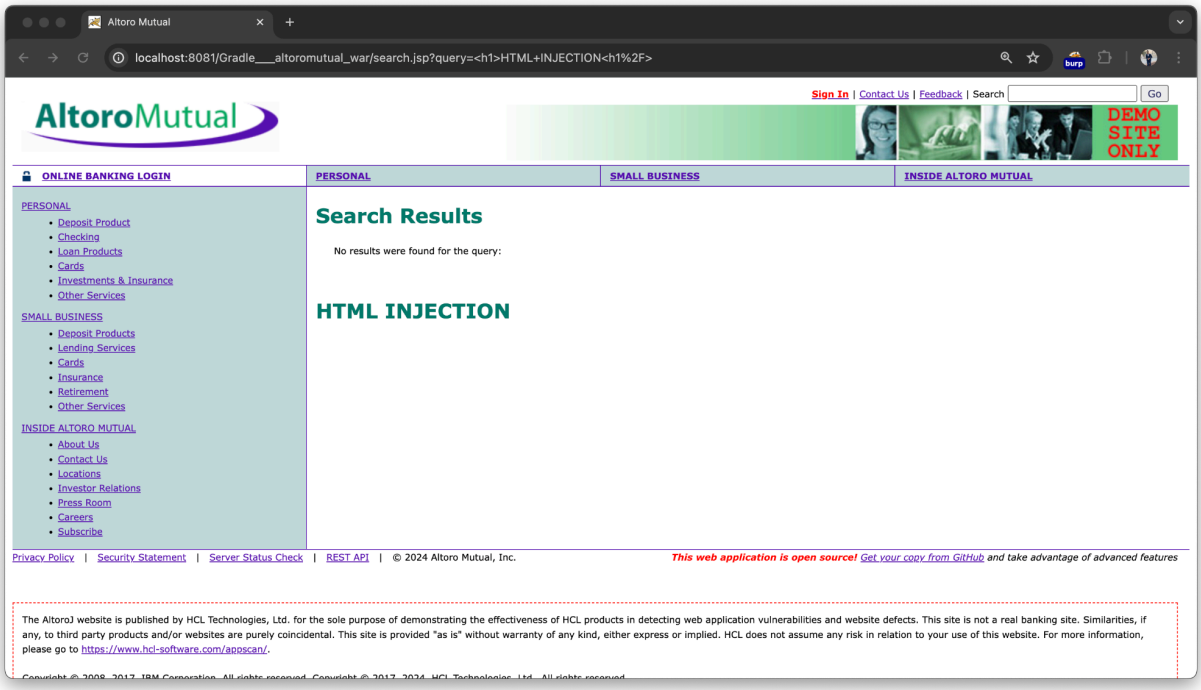

- Resolved Clickjacking issue 😀

# ID18

- Finding test steps
  - In the index page go to the search in the top right of the page

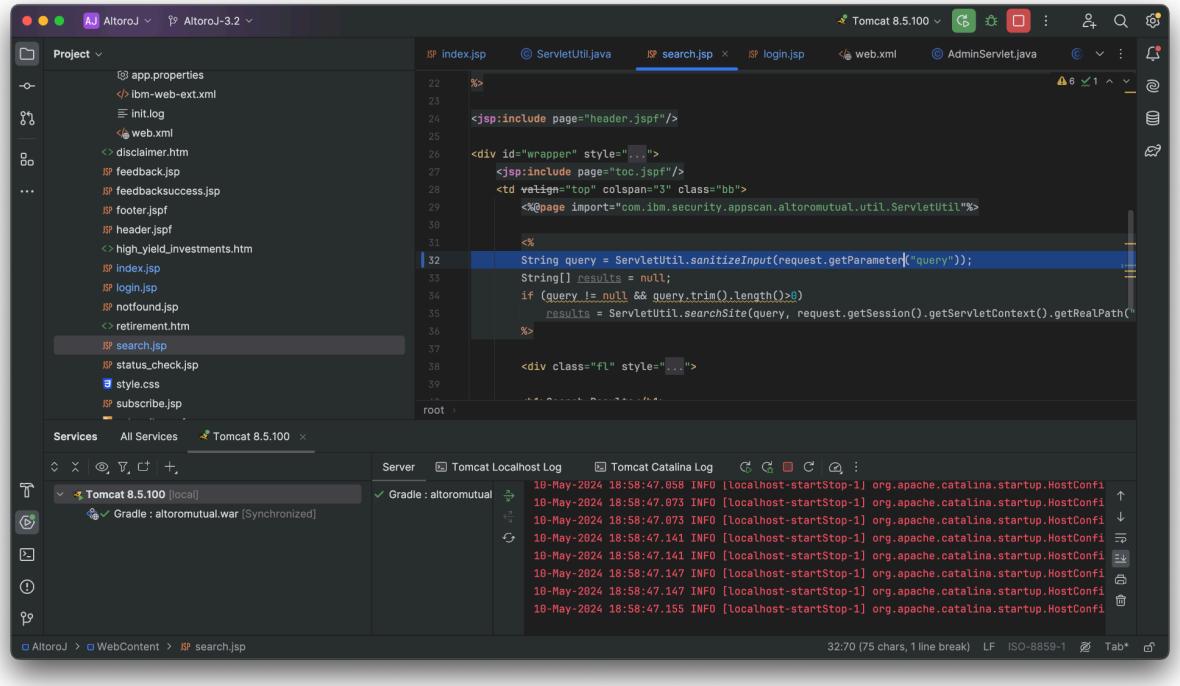



  - Try to write HTML code in search input: `<h1>HTML INJECTION</h1>`



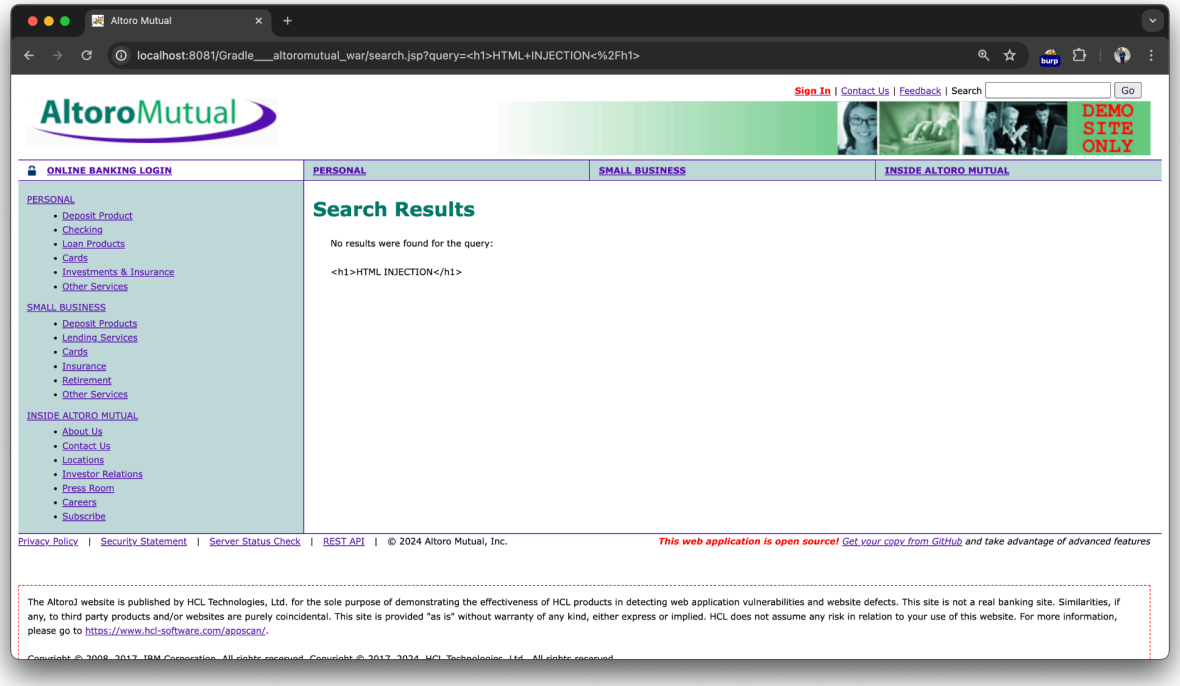
- Injected HTML in the web page

- Fixing steps
  - Go to search.jsp to query parameter
  - Found that we sanitized the input before in ID12 👍




- Finding RE-TEST steps

- Write the same HTML code in the input search

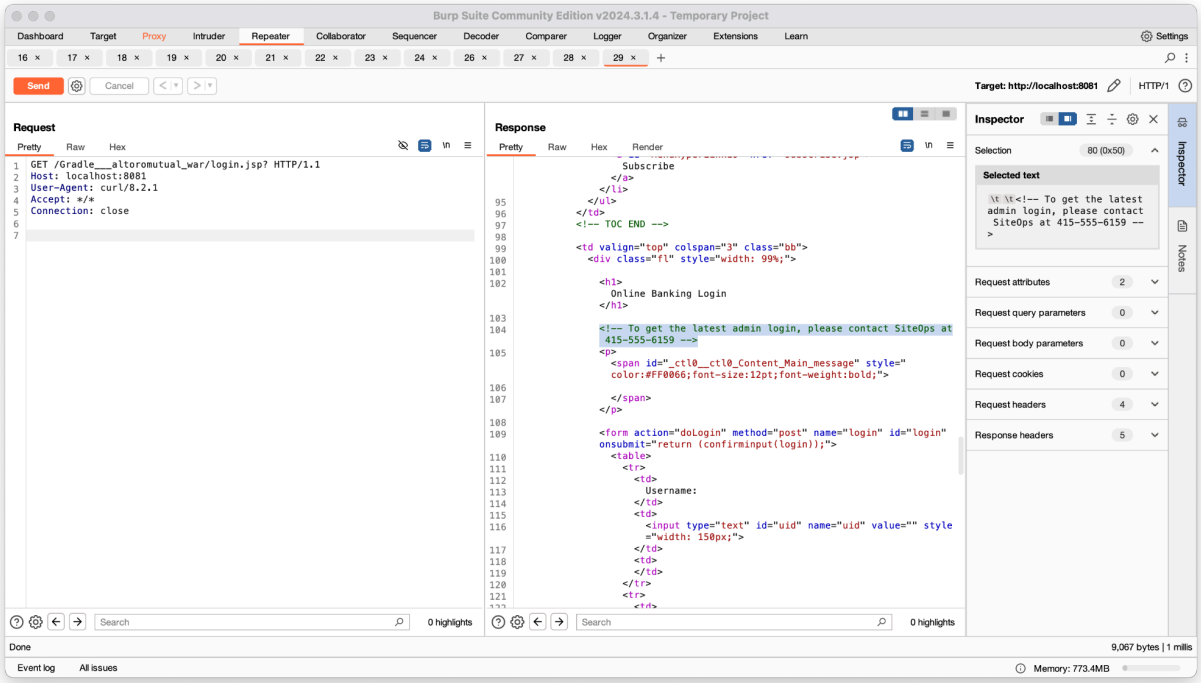



- sanitized input, so resolved issue 😵

## ID19

- Finding test steps
  - Open **login page**

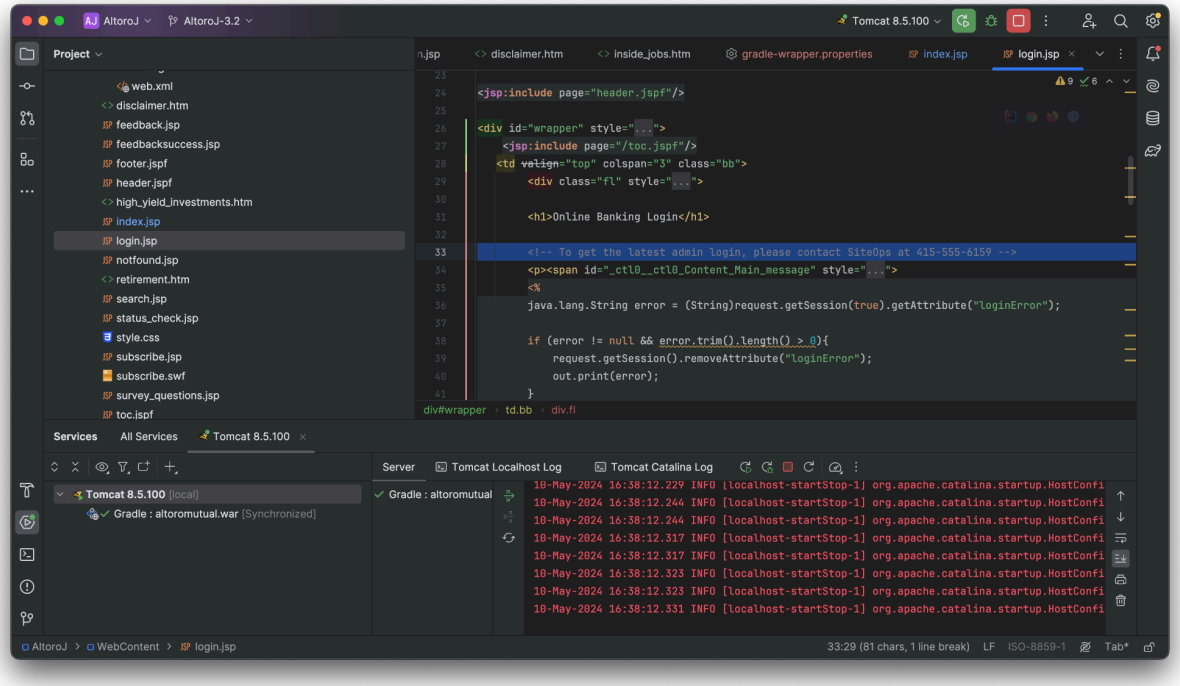
- Using Burp Suite to intercept the request



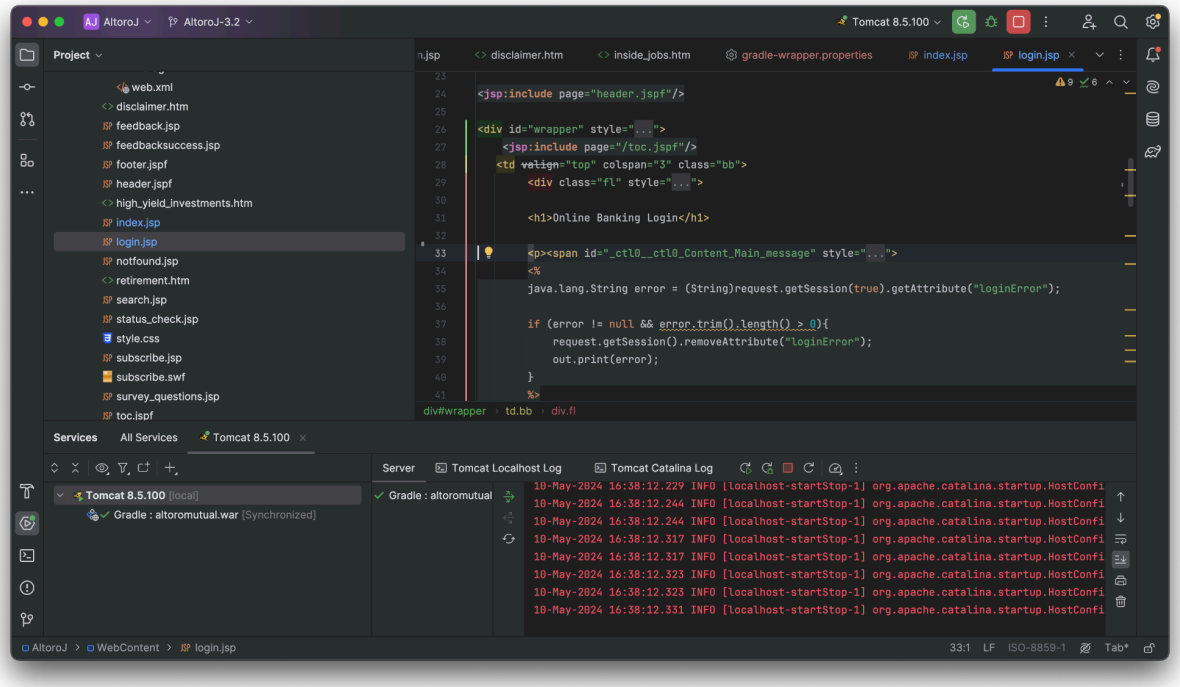

  - Sensitive data exposure in the code was left in a comment by the developer that can be used by the attacker

- Fixing steps
  - Inspect login.jsp file in the project source code

- Found the sensitive data in line **33**

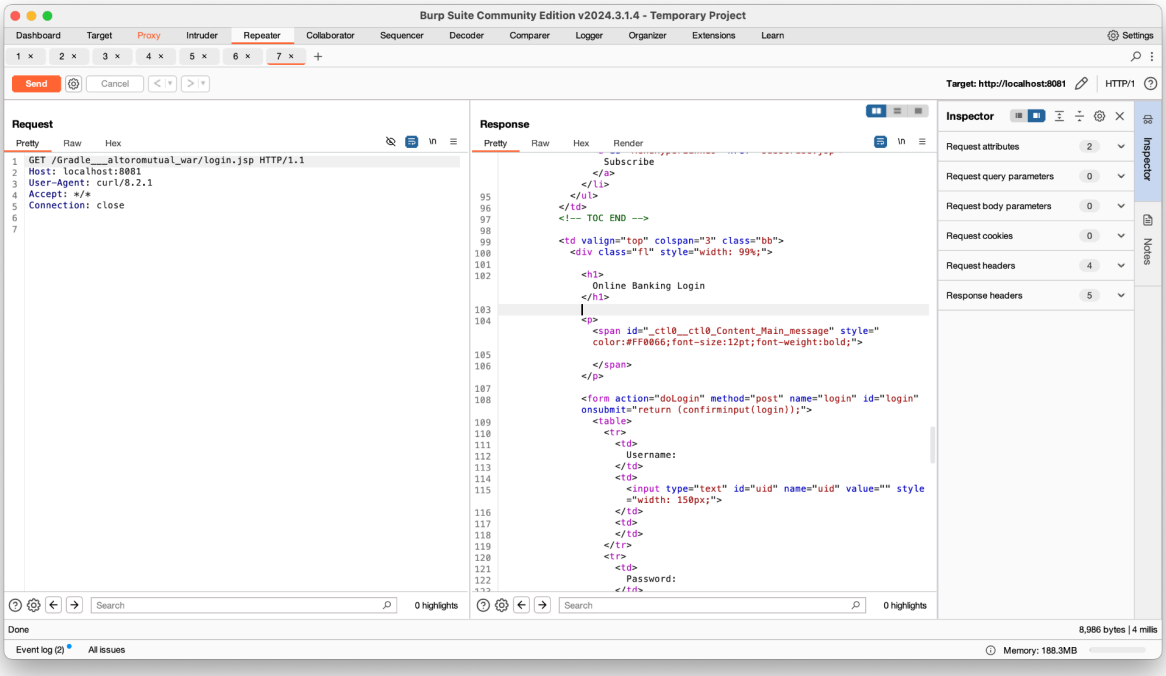



- Remove the comment




- Finding RE-TEST steps
  - Open login page

- Use Burp Suite to intercept the request




- Sensitive data is gone 😀