INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

BANGALORE

SOFTWARE TESTING

CSE 731

Client-Side Web Applications Testing (Bypass Testing) for Academic ERP

Team Members

Sharvari Mishra
MT2023001
Sharvari.Mishra@iiitb.ac.in

Bhumika Jindal
MT2023005
Bhumika.Jindal@iiitb.ac.in

Table of Contents

Introduction

Overview of the Project

The **Academic ERP (Enterprise Resource Planning) System** is a **full-stack web application** designed to streamline academic management processes for universities and educational institutions. The project includes both **frontend** and **backend** components, enabling role-based access for managing courses, departments, and faculties. The primary focus of the application is on **course management**, providing functionalities for CRUD (Create, Read, Update, Delete) operations.

The **frontend** of the system provides a user-friendly interface for updating and deleting courses, while the **backend** is responsible for enforcing data consistency, role-based access control, and managing the relationships between courses and their prerequisites. Some functionalities, such as **course creation** and the **management of faculty and departments**, are handled via **Postman** instead of being implemented in the frontend interface.

Tech Stack Used in Codebase

- **Frontend**: React.js for building the user interface.
- **Backend**: Java Spring Boot for RESTful APIs and application logic.
- **Database**: MySQL for persistent data storage.
- **APIs**: RESTful APIs for course, faculty, and department management.

Purpose of this Testing Document

This document provides a detailed overview of the testing efforts undertaken to validate the **course management** functionalities of the Academic ERP system. It focuses on ensuring the system's robustness in handling user inputs, preventing unauthorized access, and maintaining data integrity, particularly in scenarios where client-side validation mechanisms can be bypassed.

Scope and Importance of Testing

Scope of the Testing in our Web App

The scope of testing for this project focuses on **course management** functionalities, including:

1. **Course CRUD Operations**:
    - **Frontend**: Users can update or delete existing courses.
    - **Postman**: Course creation is handled via API requests in Postman.
2. **Cascading Updates/Deletes**:
    - When a course's prerequisite is updated or deleted, the changes should cascade to all dependent records.
3. **Role-Based Access Control**:
    - Only admin employees are allowed to update or delete courses via the frontend or Postman APIs.
4. **Client-Side Validation Testing**:

- Testing the robustness of client-side validations and identifying potential vulnerabilities when bypassing them.

5. **Faculty and Department Management**:
    - Faculty and department records are added and managed via **Postman APIs**, so testing their operations is out of the scope of the frontend validation testing.

Why Testing is Being Performed

1. **Ensuring System Reliability**:
    - The system must handle user inputs and actions gracefully, maintaining data consistency and application stability.
2. **Securing Data Integrity**:
    - Input data must conform to predefined rules, ensuring that invalid or malicious inputs cannot corrupt the database or bypass security mechanisms.
3. **Verifying Role-Based Access**:
    - The system must enforce strict access controls, ensuring only authorized users can modify or delete sensitive records like course data.
4. **Maintaining Cascading Data Consistency**:
    - When a course or prerequisite is updated or deleted, all dependent records must be updated or deleted appropriately to maintain data integrity.
5. **Identifying Vulnerabilities**:
    - Web applications are prone to attacks such as SQL injection, privilege escalation, or unauthorized data access. Comprehensive testing ensures the system is secure from such vulnerabilities.

Specific Focus on Bypass Testing and Its Importance

Bypass Testing is a specialized testing technique where the tester intentionally bypasses client-side input validation mechanisms to test the robustness of server-side validation. This approach is critical because:

1. **Web Application Vulnerability**:
    - Modern web applications often use client-side scripting (e.g., JavaScript) for quick input validation. However, this validation can be easily bypassed by malicious users using tools like Burp Suite or browser developer tools.
2. **Server-Side Validation**:
    - The server must handle all input validation to ensure data integrity and prevent exploits like SQL injection, unauthorized updates, or cascading failures.
3. **Real-World Threat Simulation**:
    - Bypass testing simulates real-world attack scenarios where users manipulate requests, bypassing client-side checks to exploit vulnerabilities in the backend.
4. **Importance for Academic ERP**:
    - In the context of the Academic ERP system, bypass testing ensures that:
        - Only authorized users (e.g., admin employees) can delete or update course records.
        - Invalid course prerequisites (e.g., self-referencing courses) are rejected.
        - Manipulated data, such as exceeding course capacity or altering credit values, is appropriately handled.

By focusing on bypass testing, this document highlights the importance of robust server-side mechanisms in ensuring the Academic ERP system is secure, reliable, and capable of handling malicious attempts to bypass client-side validations.

Testing Methodologies

Testing the Academic ERP system will be conducted in **three phases** using different tools to cover various aspects of functionality, security, and client-server interactions. Each phase has distinct objectives, importance, steps, and test cases.

1. API Testing in Postman

Objective: To validate the functionality of APIs used for course creation, faculty, and department management.

Importance:

1. Ensures that the APIs are functional and adhere to the defined specifications.
2. Validates the input/output format and error handling mechanisms.
3. Confirms that the APIs can handle edge cases and invalid inputs effectively.

Steps:

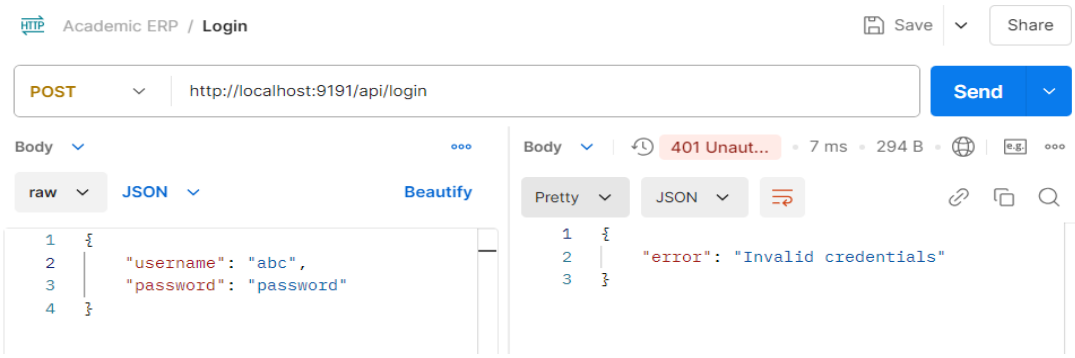
1. Launch Postman and import API endpoints from the provided API documentation.
2. Configure authentication (e.g., API key or bearer token) if required.
3. Execute CRUD operations on course, faculty, and department APIs.
4. Test edge cases by sending invalid or missing data in requests.
5. Analyse API responses for correctness, error codes, and messages.

Test Cases:

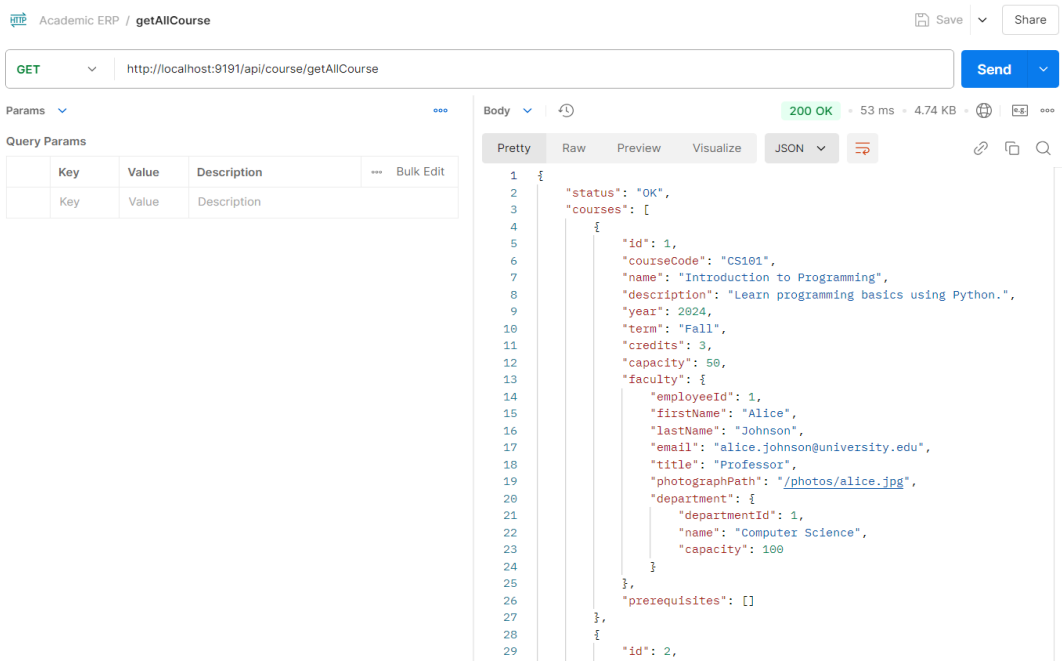
Test Case ID	Test Description	Endpoint	Input Data	Expected Result	Test Result
TC001	Invalid Login Credentials Authentication	**POST** /login	JSON	Status: 401 Unauthorized	Pass
TC002	Viewing all courses, departments and faculties in database	**GET** /course/getAllCourse /employee/getAllFaculty /department/getAllDepartment	None	Status: 200 OK	Pass
TC003	Create a new course	**POST** /course/addCourse	JSON	Status: 200 OK	Pass
TC004	Update an existing course by adding prerequisite courses & update capacity	**PUT** /course/updateCourse/7	JSON	Status: 200 OK	Pass
TC005	Delete a course with prerequisites	DELETE /courses/1	None	Status: 200 OK	Pass

Snapshot of Test Cases:

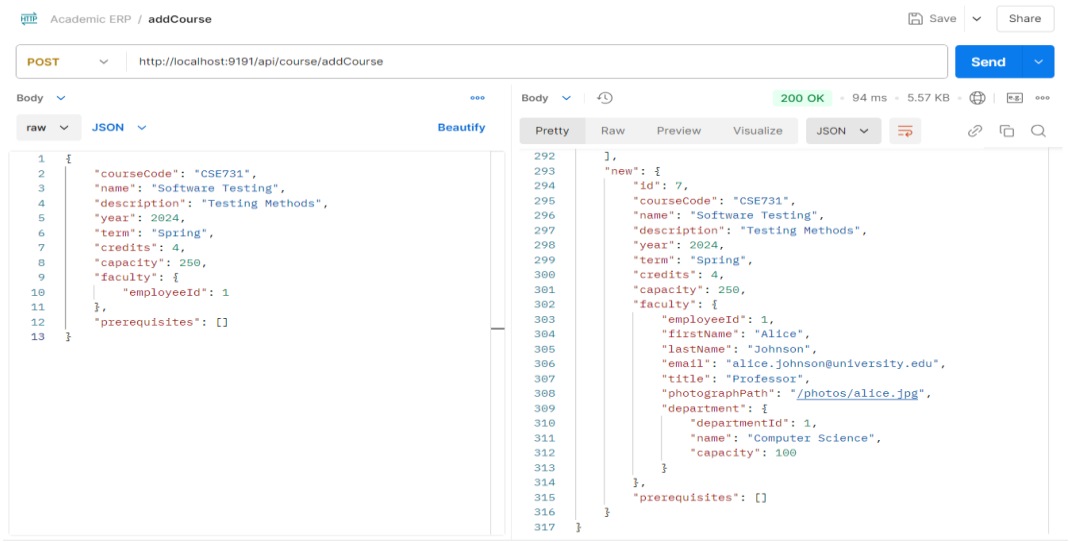
- **TC001:**



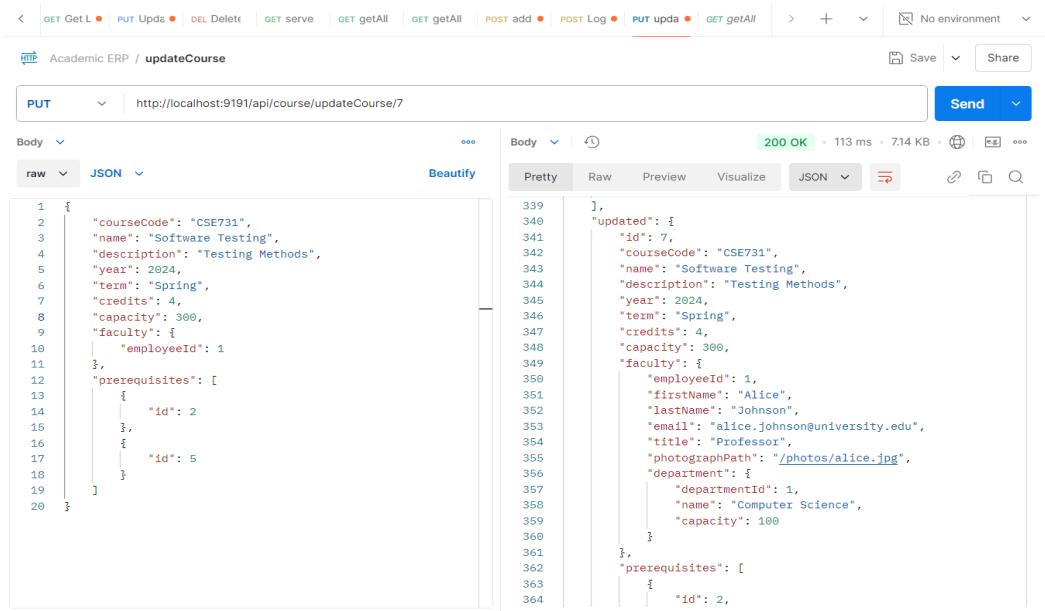
- **TC002**:



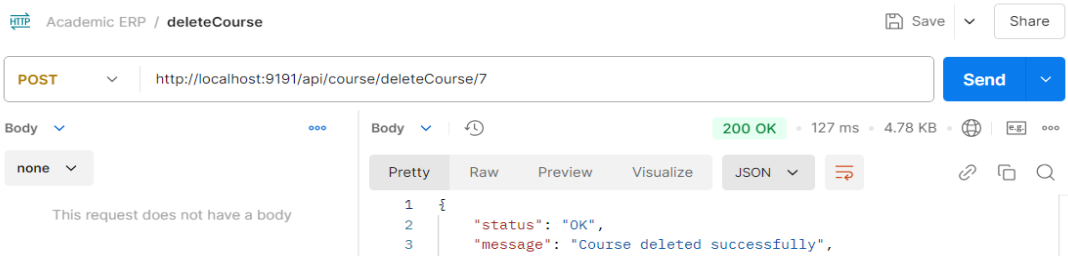
- **TC003:**



- **TC004:**



- **TC005:**



2. Client-Side Testing Using Web Browser (Inspect Tools)

Objective: To verify the frontend's behavior, client-side validations, and its interaction with the backend.

Importance:

1. Ensures that the user interface handles inputs correctly before sending them to the backend.
2. Validates dynamic elements like form validations and error messages.
3. Identifies weaknesses in client-side validation that could allow invalid inputs to be sent to the server.

Steps:

1. Open the web application in a browser (e.g., Chrome).
2. Use developer tools (F12) to inspect network requests and modify input fields in forms.

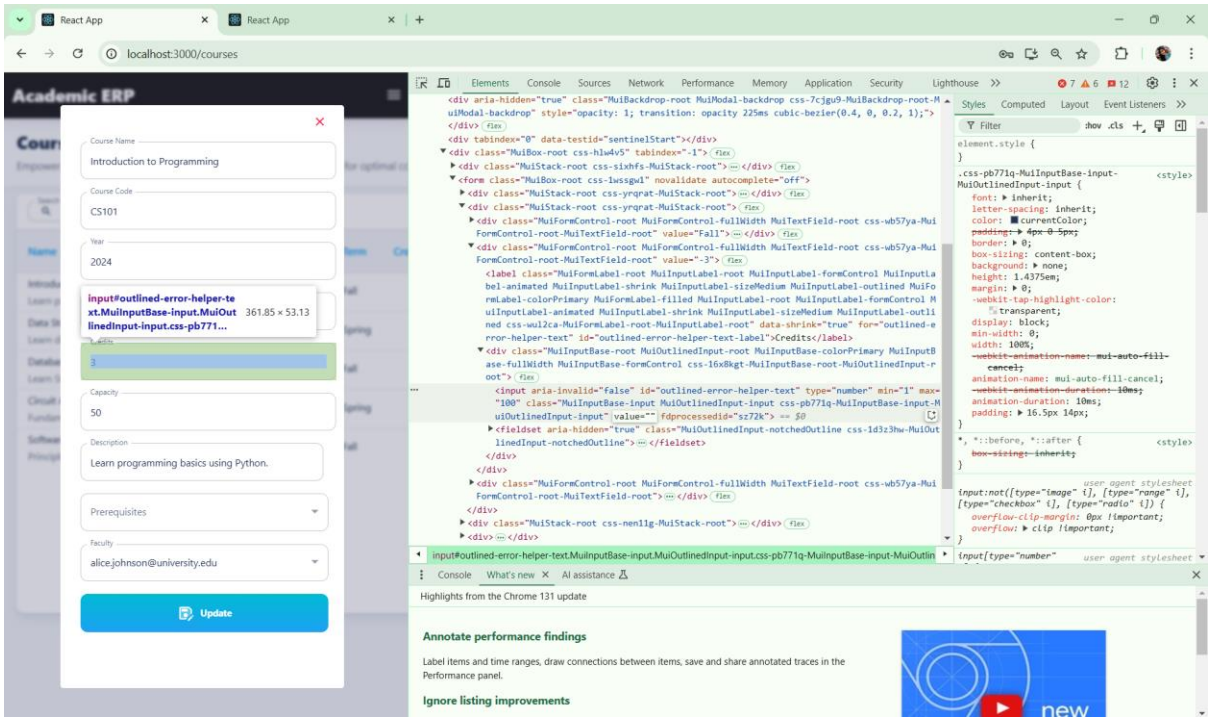
3. Perform the following:
    - Test form field validations by entering invalid or missing data.
    - Analyze how the client-side JavaScript handles errors.
    - Submit forms with deliberately altered fields and observe server responses.

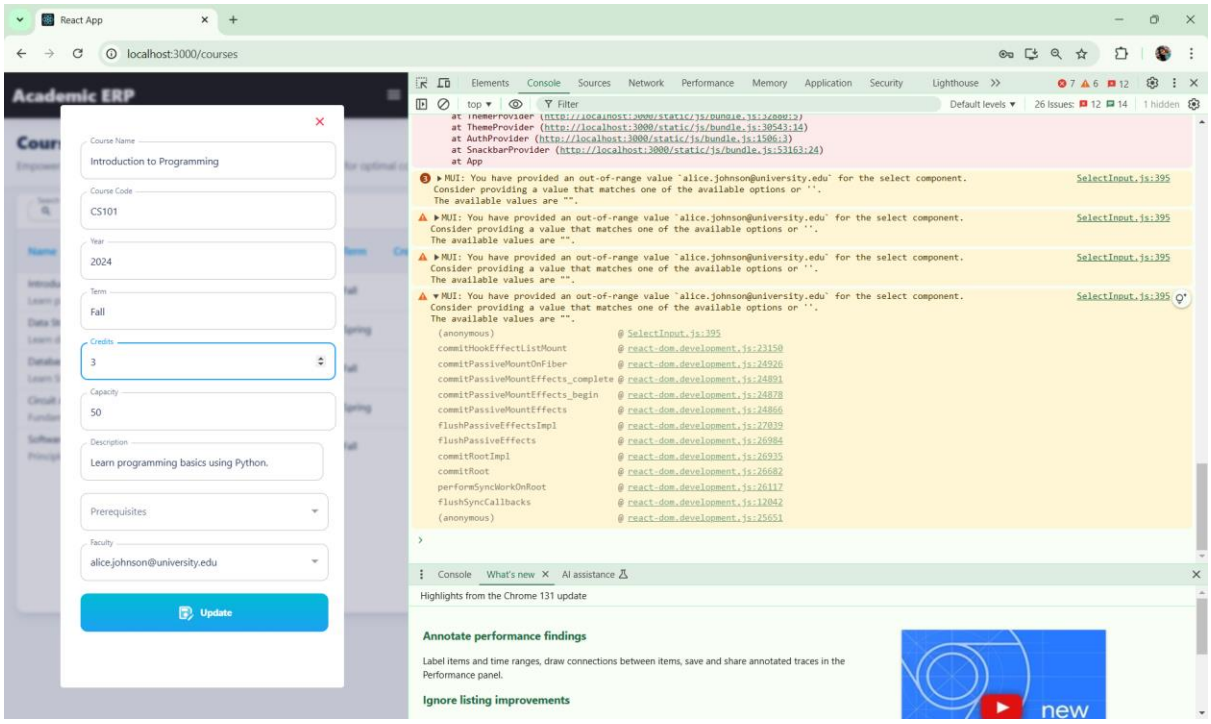
Test Cases:

Test Case ID	Test Description	Steps	Expected Result	Test Result
TC101	Passing null value to non-nullable parameter (like Credit of the Course)	Make the credit course parameter empty and submit.	Error message: "Course Credit cannot be null"	Pass
TC102	Removing range condition on year and giving invalid year	Change the year parameter to 999 and submit.	Error message: "Invalid Year"	Fail

Snapshot of Test Cases:

- **TC101:**





- **TC102:**

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

course_id	capacity	course_code	credits	description	name	term	year	employee_id
1	50	CS101	3	Learn programming basics using Python.	Introduction to Programming	Fall	999	1
2	40	CS102	4	Learn data structures and algorithms.	Data Structures	Spring	2024	1
3	45	IT201	3	Learn SQL and relational database concepts.	Database Systems	Fall	2024	2
4	30	EE301	3	Fundamentals of circuit design and analysis.	Circuit Analysis	Spring	2024	3
5	50	SE401	3	Principles of software development.	Software Engineering	Fall	2024	4
6	25	AI501	4	Introduction to AI concepts.	Artificial Intelligence	Spring	2024	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. Client-side Validation (By-Pass) Testing Using Burp Suite

Objective: To identify vulnerabilities in client-side validations and verify backend security by manipulating requests.

Importance:

1. Ensures server-side validation is robust and protects against bypassing client-side restrictions.
2. Simulates real-world attacks to uncover vulnerabilities like privilege escalation and SQL injection.
3. Validates role-based access control and secure handling of malicious inputs.

Steps:

1. Open Burp Suite and configure our browser to use it as a proxy.
2. Intercept HTTP requests generated by the frontend (e.g., course update/delete operations).
3. Modify intercepted requests to inject malicious or invalid payloads.
4. Forward the modified requests to the server and observe its behaviour.
5. Document vulnerabilities and analyse error messages or system failures.

Types of By-Pass Testing:

Bypass testing systematically evaluates input validation vulnerabilities by simulating invalid inputs, unexpected parameters, and control flow disruptions in web applications. This section outlines how to implement and test **value level**, **parameter level**, and **control flow level** bypass testing.

a. Value Level Bypass Testing

Objective: To verify if the web application correctly evaluates invalid inputs and handles constraints on individual parameters.

Test Design:

1. **Data Type Conversion Violation**:
   - Input data of unexpected types (e.g., string instead of an integer).
   - Example: Inject text into a numeric field like capacity.
2. **Built-in Length Restriction Violation**:
   - Input strings exceeding the max length attribute of fields.
   - Example: For course code with a length restriction of 10, input a 50-character string.
3. **Built-in Value Restriction Violation**:
   - Provide values not defined in pre-configured dropdowns or radio buttons.
   - Example: For a term dropdown with options "Spring" and "Fall," input "Winter."
4. **Special Input Value**:
   - Inject special characters (e.g., <script>alert('XSS')</script>, '; DROP TABLE courses; --) to test sanitization and security.
   - Example: Enter malicious input into the description field.

Steps:

1. Identify all form fields and their validation constraints.
2. Prepare invalid input payloads for:
   - Exceeding length limits.
   - Unexpected data types.
   - Special characters that exploit vulnerabilities.
3. Submit the invalid inputs via:
   - Frontend (manipulated using browser DevTools or Burp Suite).
   - API calls (using Postman or Burp Suite Repeater).
4. Observe server responses and check if errors are handled gracefully.

Expected Results:

- Server should reject invalid inputs with appropriate error messages.
- Application should not crash or leak sensitive data.

Test Cases:

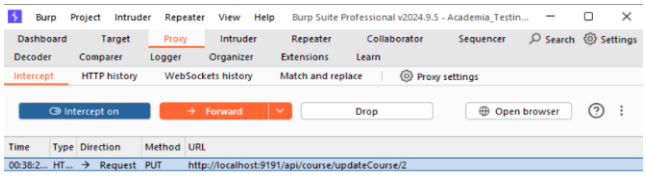
Below is a structured table of test cases based on the provided value-level bypass testing scenarios for the Academic ERP system.

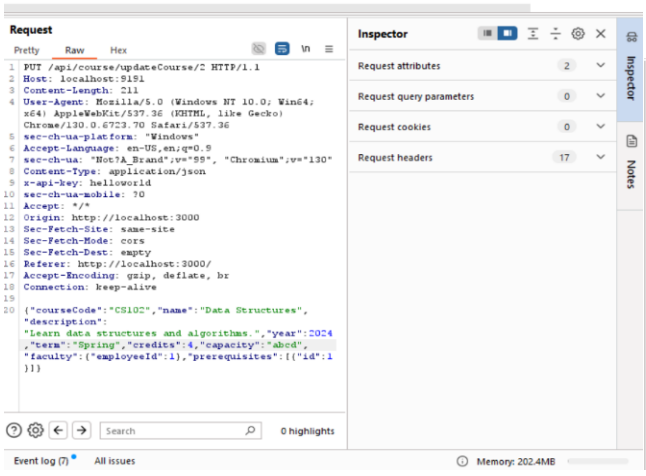
Test Case ID	Test Scenario	Burp Suite Method	Steps	Expected Result	Test Result
TC201	Data Type Conversion Violation	Intercept HTTP Traffic with Burp Proxy	1. Configure browser to route traffic through Burp Proxy. 2. Fill the capacity field with invalid data ("abcd") and submit the form. 3. Intercept the request in Burp Proxy and observe the request payload. 4. Forward the request.	Error message: Invalid Input – Data Type numeric expected.	Pass
TC202	Built-in Length Restriction Violation	Modifying Requests in Burp Proxy	1. Intercept the HTTP request for course Code. 2. Replace the field value with a 10-character string. 3. Forward the modified request to the server and observe the response.	Error message: Course code should be at most of length 6.	Pass
TC203	Built-in Value Restriction Violation	Manually Reissuing Requests with Burp Repeater	1. Capture the request with the dropdown value term in Burp Proxy. 2. Send the request to Burp Repeater. 3. Modify the courses in pre-requisites to same course. 4. Send the modified request repeatedly to test server behaviour.	Error message: Pre-requisite cannot be same as the Course Id	Pass
TC204	Special Input Value (XSS)	Intercept HTTP Traffic & Modifying Requests	1. Submit a form with <script>alert('XSS')</script> in the description field. 2. Intercept the request in Burp Proxy and observe the payload. 3. Forward the request and analyse the response for sanitization or script execution.	Error message: Invalid input, unrelated to parameter	Fail

TC205	Exceeding Numeric Limits	Intercept HTTP Traffic & Modifying Requests	1. Input a very large value (e.g., 99999) in the capacity field. 2. Intercept the request in Burp Proxy and observe the response. 3. Modify the value further (e.g., -1) to test edge cases.	Error message: Parameter value out-of-range.	Pass
TC206	Null Input	Intercept HTTP Traffic	1. Submit a request without filling the mandatory name/code/credit field. 2. Intercept the request and ensure the payload reflects the missing field. 3. Observe the server's response for proper validation.	Error message: Value cannot be empty	Pass
TC207	Invalid Email Format	Intercept HTTP Traffic & Modifying Requests	1. Input an invalid email (invalid-email) in the faculty email field. 2. Intercept the request, forward it to the server, and verify error response.	Error message: Faculty with this email not found	Pass
TC208	Input Overflow	Intercept HTTP Traffic & Modifying Requests	1. Submit a request with a 100,000-character string in the description field. 2. Intercept the request in Burp Proxy. 3. Forward the request and observe server response (e.g., memory or processing issues).	Error message: Parameter value too long.	Pass
TC209	Invalid Characters	Intercept HTTP Traffic & Modifying Requests	1. Input invalid characters (@#$%^&*()) in the name field. 2. Intercept and forward the request. 3. Observe server response for error messages.	Error message: Invalid characters in input	Fail

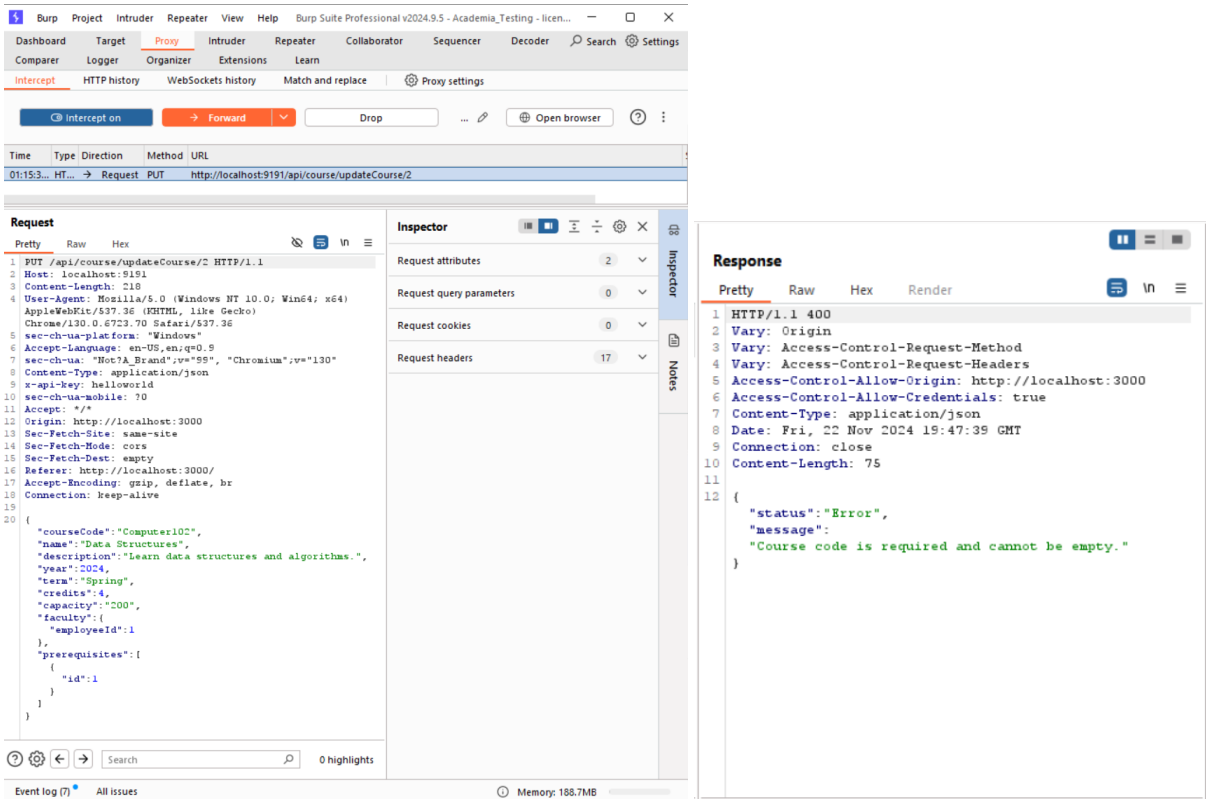
Snapshot of Test Cases:

- **TC201:**

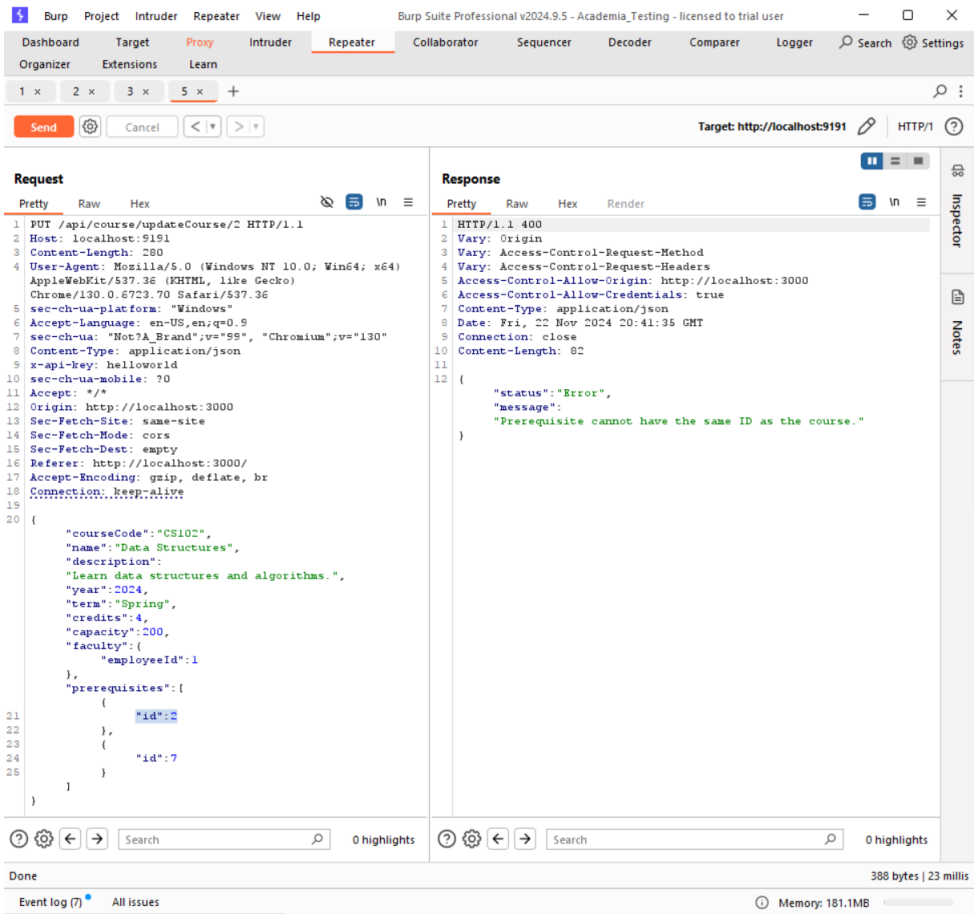




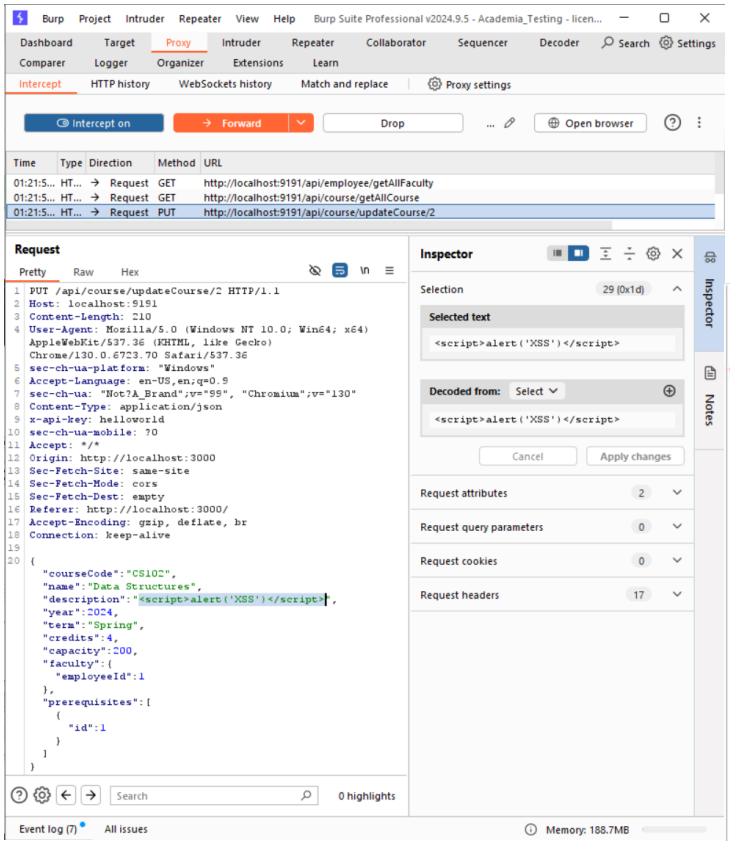
- **TC202:**

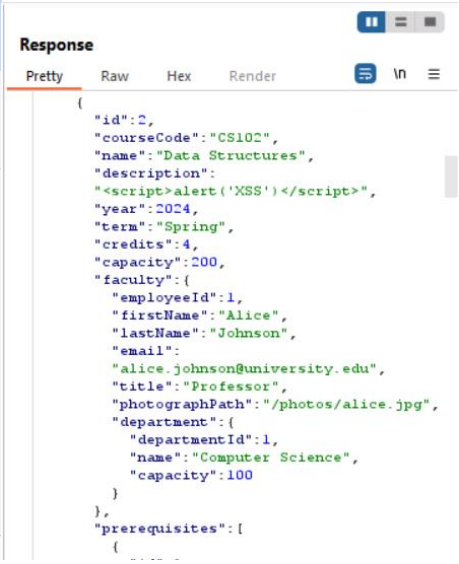

Request
PUT /api/course/updateCourse/2 HTTP/1.1
Host: localhost:9191
Content-Length: 218
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/130.0.6723.70 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
x-api-key: helloworld
sec-ch-ua-mobile: ?0
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
{
"courseCode":"Computer102",
"name":"Data Structures",
"description":"Learn data structures and algorithms.",
"year":2024,
"term":"Spring",
"credits":4,
"capacity":"200",
"faculty":{
"employeeId":1
},
"prerequisites":[
{
"id":1
}
]
}
Inspector
Request attributes 2
Request query parameters 0
Request cookies 0
Request headers 17
Response
HTTP/1.1 400
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin: http://localhost:3000
Access-Control-Allow-Credentials: true
Content-Type: application/json
Date: Fri, 22 Nov 2024 19:47:39 GMT
Connection: close
Content-Length: 75
{
"status":"Error",
"message":
"Course code is required and cannot be empty."
}

- **TC203:**

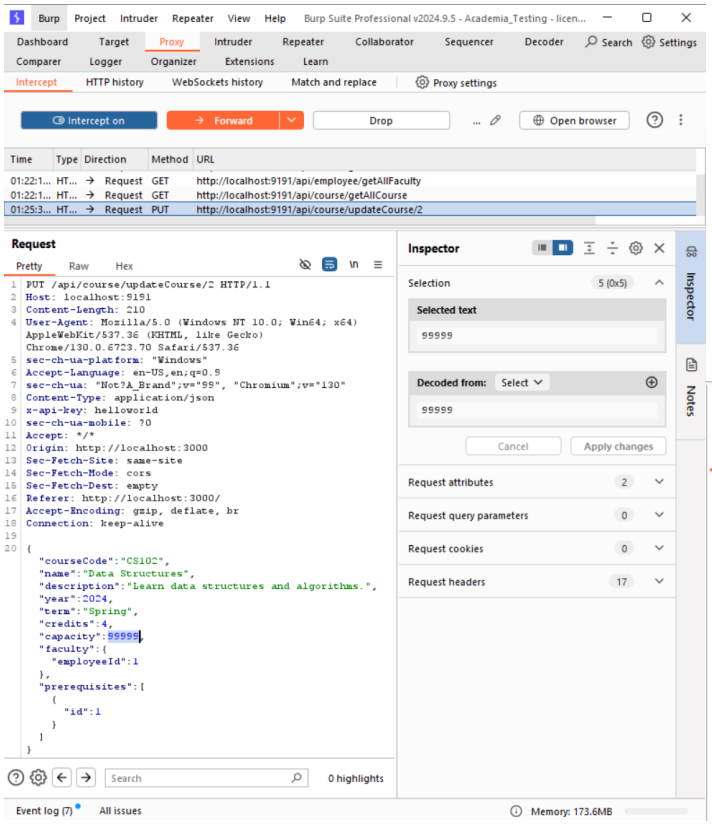

Request
PUT /api/course/updateCourse/2 HTTP/1.1
Host: localhost:9191
Content-Length: 280
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/130.0.6723.70 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
x-api-key: helloworld
sec-ch-ua-mobile: ?0
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
{
"courseCode":"CS102",
"name":"Data Structures",
"description":
"Learn data structures and algorithms.",
"year":2024,
"term":"Spring",
"credits":4,
"capacity":200,
"faculty":{
"employeeId":1
},
"prerequisites":[
{
"id":2
},
{
"id":7
}
]
}
Response
HTTP/1.1 400
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin: http://localhost:3000
Access-Control-Allow-Credentials: true
Content-Type: application/json
Date: Fri, 22 Nov 2024 20:41:35 GMT
Connection: close
Content-Length: 82
{
"status":"Error",
"message":
"Prerequisite cannot have the same ID as the course."
}

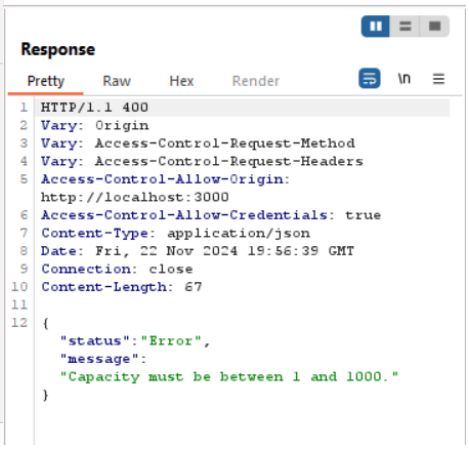
- **TC204:**


Burp Project Intruder Repeater View Help Burp Suite Professional v2024.9.5 - Academia_Testing - licen...
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Search Settings
Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on Forward Drop Open browser
Time Type Direction Method URL
01:21:5... HT... → Request GET http://localhost:9191/api/employee/getAllFaculty
01:21:5... HT... → Request GET http://localhost:9191/api/course/getAllCourse
01:21:5... HT... → Request PUT http://localhost:9191/api/course/updateCourse/2
Request
Pretty Raw Hex
PUT /api/course/updateCourse/2 HTTP/1.1
Host: localhost:9191
Content-Length: 210
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
x-api-key: helloworld
sec-ch-ua-mobile: ?0
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
{
"courseCode":"CS102",
"name":"Data Structures",
"description":"<script>alert('XSS')</script>",
"year":2024,
"term":"Spring",
"credits":4,
"capacity":200,
"faculty":{
"employeeId":1
},
"prerequisites":[
{
"id":1
}
]
}
Search 0 highlights
Inspector
Selection 29 (0x1d)
Selected text
<script>alert('XSS')</script>
Decoded from: Select
<script>alert('XSS')</script>
Cancel Apply changes
Request attributes 2
Request query parameters 0
Request cookies 0
Request headers 17
Inspector Notes
Event log (7) All issues
Memory: 188.7MB

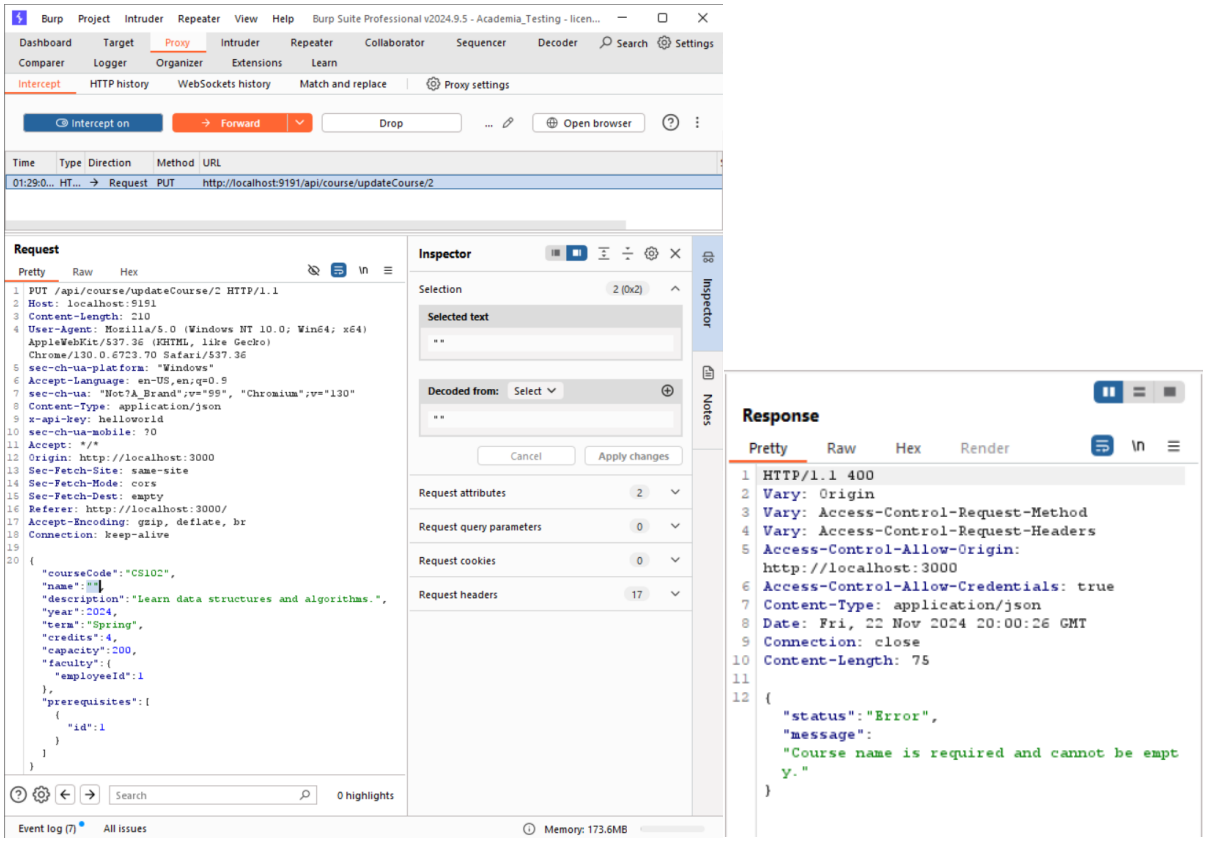

Response
Pretty Raw Hex Render
{
"id":2,
"courseCode":"CS102",
"name":"Data Structures",
"description":
"<script>alert('XSS')</script>",
"year":2024,
"term":"Spring",
"credits":4,
"capacity":200,
"faculty":{
"employeeId":1,
"firstName":"Alice",
"lastName":"Johnson",
"email":
"alice.johnson@university.edu",
"title":"Professor",
"photographPath":"/photos/alice.jpg",
"department":{
"departmentId":1,
"name":"Computer Science",
"capacity":100
}
},
"prerequisites":[
{

- **TC205:**

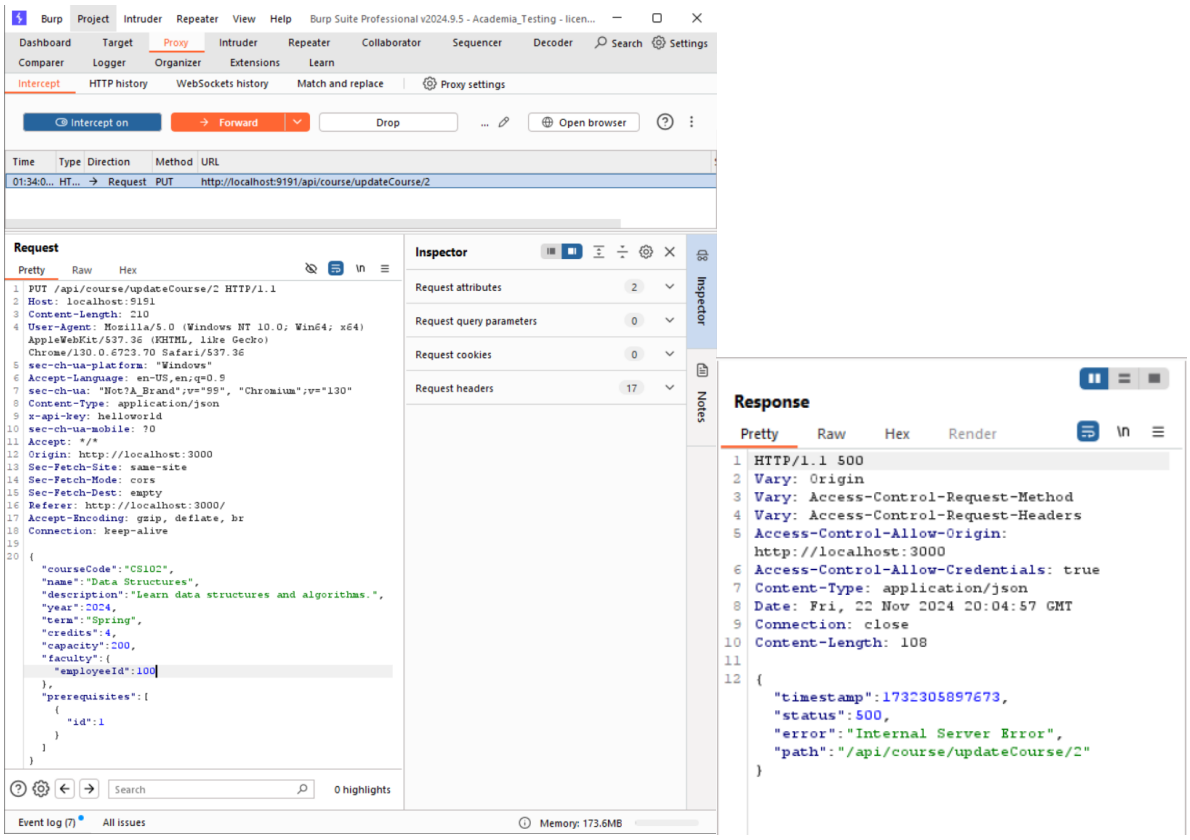

Burp Project Intruder Repeater View Help Burp Suite Professional v2024.9.5 - Academia_Testing - licen...
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Search Settings
Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on Forward Drop Open browser
Time Type Direction Method URL
01:22:1... HT... → Request GET http://localhost:9191/api/employee/getAllFaculty
01:22:1... HT... → Request GET http://localhost:9191/api/course/getAllCourse
01:25:3... HT... → Request PUT http://localhost:9191/api/course/updateCourse/2
Request
Pretty Raw Hex
PUT /api/course/updateCourse/2 HTTP/1.1
Host: localhost:9191
Content-Length: 210
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
x-api-key: helloworld
sec-ch-ua-mobile: ?0
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
{
"courseCode":"CS102",
"name":"Data Structures",
"description":"Learn data structures and algorithms.",
"year":2024,
"term":"Spring",
"credits":4,
"capacity":99999,
"faculty":{
"employeeId":1
},
"prerequisites":[
{
"id":1
}
]
}
Search 0 highlights
Inspector
Selection 5 (0x5)
Selected text
99999
Decoded from: Select
99999
Cancel Apply changes
Request attributes 2
Request query parameters 0
Request cookies 0
Request headers 17
Inspector Notes
Event log (7) All issues
Memory: 173.6MB


Response
Pretty Raw Hex Render
HTTP/1.1 400
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin:
http://localhost:3000
Access-Control-Allow-Credentials: true
Content-Type: application/json
Date: Fri, 22 Nov 2024 19:56:39 GMT
Connection: close
Content-Length: 67
{
"status":"Error",
"message":
"Capacity must be between 1 and 1000."
}

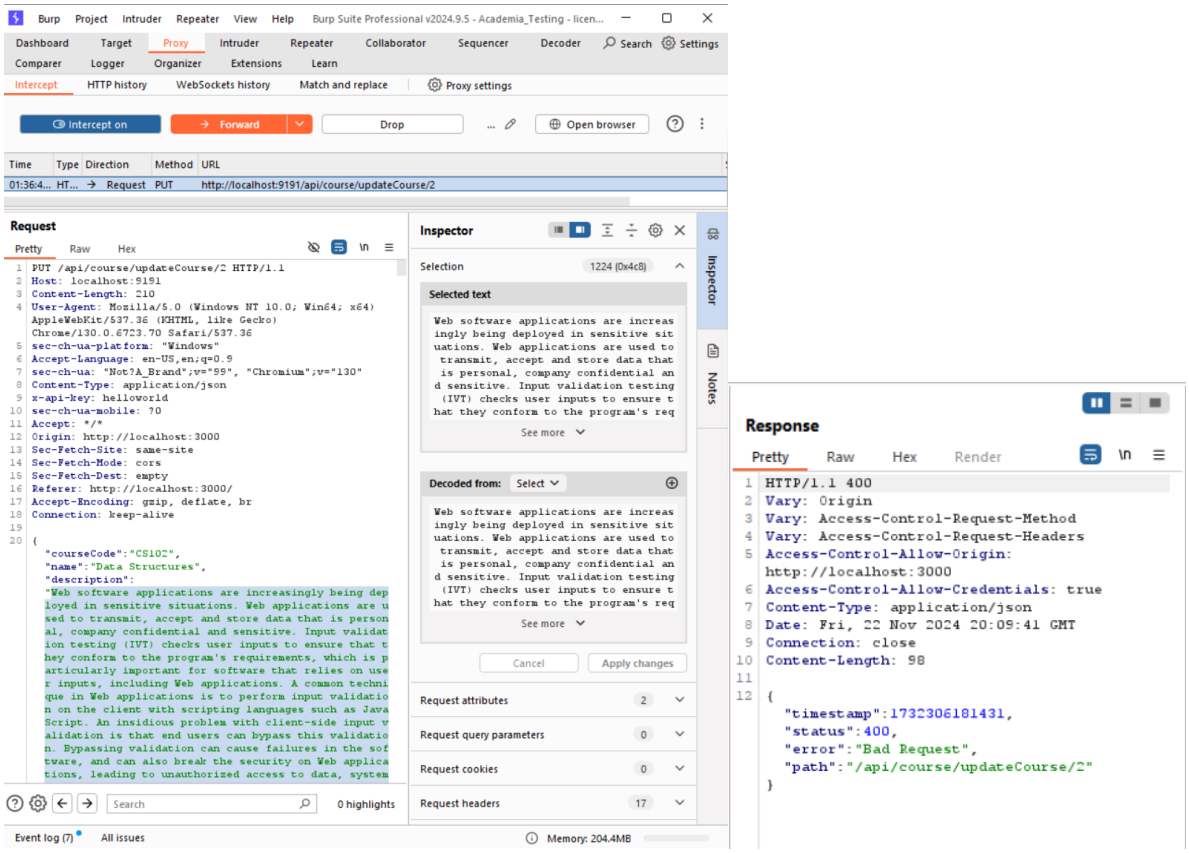
- **TC206:**


Request
PUT /api/course/updateCourse/2 HTTP/1.1
Host: localhost:9191
Content-Length: 210
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/130.0.6723.70 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
x-api-key: helloworld
sec-ch-ua-mobile: ?0
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
{
"courseCode":"CS102",
"name":"",
"description":"Learn data structures and algorithms.",
"year":2024,
"term":"Spring",
"credits":4,
"capacity":200,
"faculty":{
"employeeId":1
},
"prerequisites":[
{
"id":1
}
]
}
Inspector
Response
HTTP/1.1 400
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin:
http://localhost:3000
Access-Control-Allow-Credentials: true
Content-Type: application/json
Date: Fri, 22 Nov 2024 20:00:26 GMT
Connection: close
Content-Length: 75
{
"status":"Error",
"message":
"Course name is required and cannot be empt
y."
}

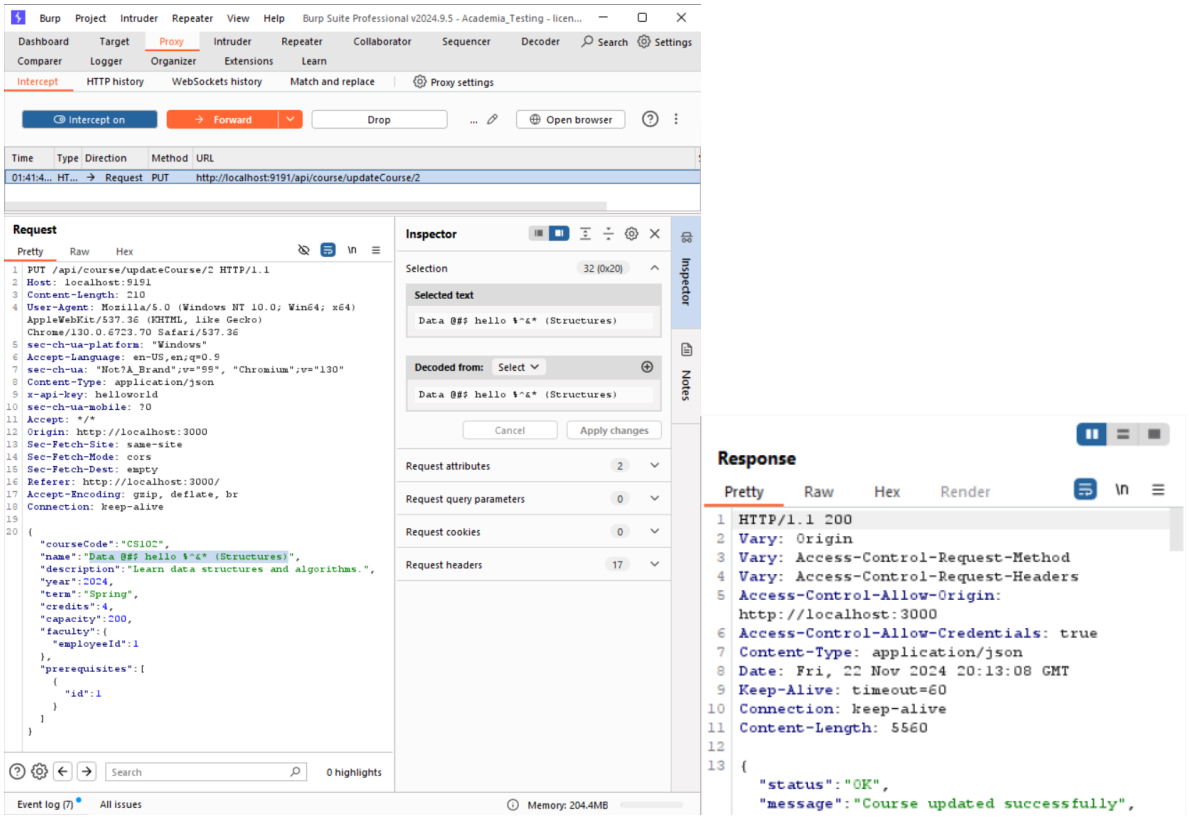
- **TC207:**


Request
PUT /api/course/updateCourse/2 HTTP/1.1
Host: localhost:9191
Content-Length: 210
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/130.0.6723.70 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
x-api-key: helloworld
sec-ch-ua-mobile: ?0
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
{
"courseCode":"CS102",
"name":"Data Structures",
"description":"Learn data structures and algorithms.",
"year":2024,
"term":"Spring",
"credits":4,
"capacity":200,
"faculty":{
"employeeId":100
},
"prerequisites":[
{
"id":1
}
]
}
Inspector
Response
HTTP/1.1 500
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin:
http://localhost:3000
Access-Control-Allow-Credentials: true
Content-Type: application/json
Date: Fri, 22 Nov 2024 20:04:57 GMT
Connection: close
Content-Length: 108
{
"timestamp":1732305897673,
"status":500,
"error":"Internal Server Error",
"path":"/api/course/updateCourse/2"
}

- **TC208:**


Request
PUT /api/course/updateCourse/2 HTTP/1.1
Host: localhost:9191
Content-Length: 210
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
x-api-key: helloworld
sec-ch-ua-mobile: ?0
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
{
"courseCode":"CS102",
"name":"Data Structures",
"description":
"Web software applications are increasingly being deployed in sensitive situations. Web applications are used to transmit, accept and store data that is personal, company confidential and sensitive. Input validation testing (IVT) checks user inputs to ensure that they conform to the program's requirements, which is particularly important for software that relies on user inputs, including Web applications. A common technique in Web applications is to perform input validation on the client with scripting languages such as JavaScript. An insidious problem with client-side input validation is that end users can bypass this validation. Bypassing validation can cause failures in the software, and can also break the security on Web applications, leading to unauthorized access to data, system
Inspector
Selection
1224 (0x4c8)
Selected text
Web software applications are increasingly being deployed in sensitive situations. Web applications are used to transmit, accept and store data that is personal, company confidential and sensitive. Input validation testing (IVT) checks user inputs to ensure that they conform to the program's req
See more
Decoded from:
Select
Web software applications are increasingly being deployed in sensitive situations. Web applications are used to transmit, accept and store data that is personal, company confidential and sensitive. Input validation testing (IVT) checks user inputs to ensure that they conform to the program's req
See more
Cancel
Apply changes
Request attributes
2
Request query parameters
0
Request cookies
0
Request headers
17
Response
Pretty
Raw
Hex
Render
HTTP/1.1 400
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin: http://localhost:3000
Access-Control-Allow-Credentials: true
Content-Type: application/json
Date: Fri, 22 Nov 2024 20:09:41 GMT
Connection: close
Content-Length: 98
{
"timestamp":1732306181431,
"status":400,
"error":"Bad Request",
"path":"/api/course/updateCourse/2"
}

- **TC209:**


Request
PUT /api/course/updateCourse/2 HTTP/1.1
Host: localhost:9191
Content-Length: 210
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
x-api-key: helloworld
sec-ch-ua-mobile: ?0
Accept: */*
Origin: http://localhost:3000
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
{
"courseCode":"CS102",
"name":"Data @#$ hello %^&* (Structures)",
"description":"Learn data structures and algorithms.",
"year":2024,
"term":"Spring",
"credits":4,
"capacity":200,
"faculty":{
"employeeId":1
},
"prerequisites":[
{
"id":1
}
]
}
Inspector
Selection
32 (0x20)
Selected text
Data @#$ hello %^&* (Structures)
Decoded from:
Select
Data @#$ hello %^&* (Structures)
Cancel
Apply changes
Request attributes
2
Request query parameters
0
Request cookies
0
Request headers
17
Response
Pretty
Raw
Hex
Render
HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin: http://localhost:3000
Access-Control-Allow-Credentials: true
Content-Type: application/json
Date: Fri, 22 Nov 2024 20:13:08 GMT
Keep-Alive: timeout=60
Connection: keep-alive
Content-Length: 5560
{
"status":"OK",
"message":"Course updated successfully",

b. Parameter Level Bypass Testing

Objective: To identify vulnerabilities in relationships between multiple input parameters, such as invalid combinations or missing dependencies.

Test Design:

1. **Invalid Pair Testing**:
   - Test for incompatible parameter combinations.
   - Example: Submit a course with term set to "Spring" and prerequisites referencing courses unavailable for the term.
2. **Required Pair Testing**:
   - Omit dependent parameters in the request.
   - Example: Submit a course without providing credits when term is set.
3. **Unexpected Parameter Addition**:
   - Add unexpected parameters to the request payload.
   - Example: Add a non-existent parameter, extraField, with a random value.

Steps:

1. Identify dependencies and constraints between parameters (e.g., required, optional, incompatible).
2. Prepare test payloads with:
   - Missing required parameters.
   - Extra or incompatible parameters.
3. Use Burp Suite to intercept and modify requests or Postman for API payload testing.
4. Analyse responses for proper validation.

Expected Results:

- Server should enforce required dependencies.
- Incompatible or unexpected parameters should trigger appropriate errors.

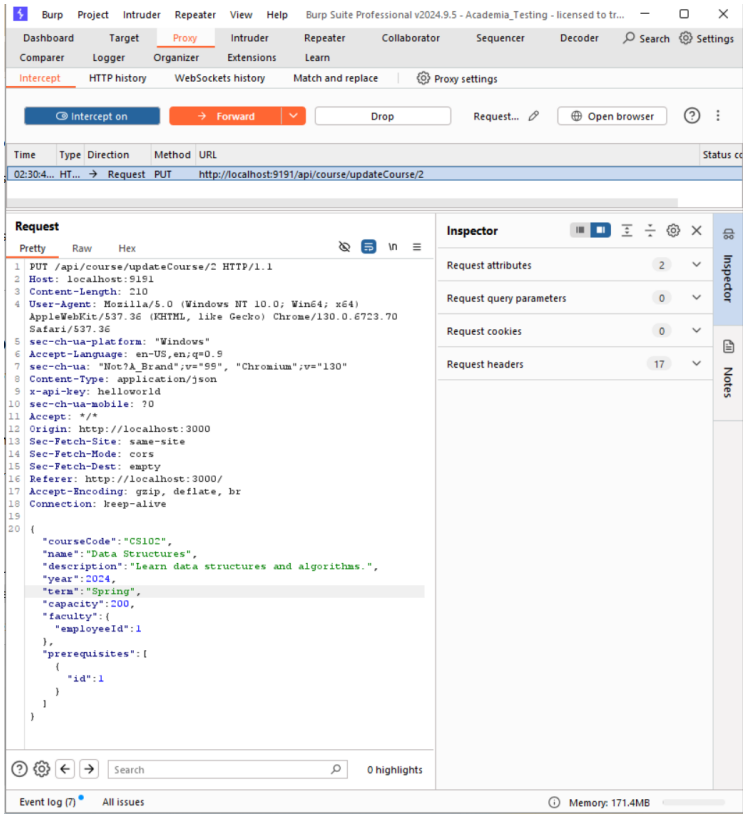
Test Cases:

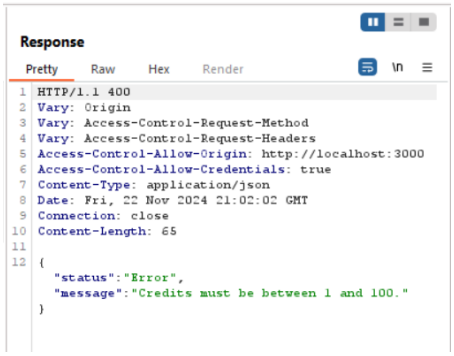
Below is a structured table of test cases based on the provided value-level bypass testing scenarios for the Academic ERP system.

Test Case ID	Test Scenario	Burp Suite Method	Steps	Expected Result	Test Result
TC210	Missing required fields	Intercepting HTTP Traffic, Reissuing Requests	Remove the credits field from the payload and resend the request to test if the server handles the missing required field properly.	Error message: Parameter Missing	Pass
TC211	Unexpected fields in the payload.	Intercepting HTTP Traffic, Modifying Requests	Add an unexpected field, such as "extraField": "randomValue", to test how the server handles additional unexpected parameters.	Error message: Unexpected Parameter	Fail

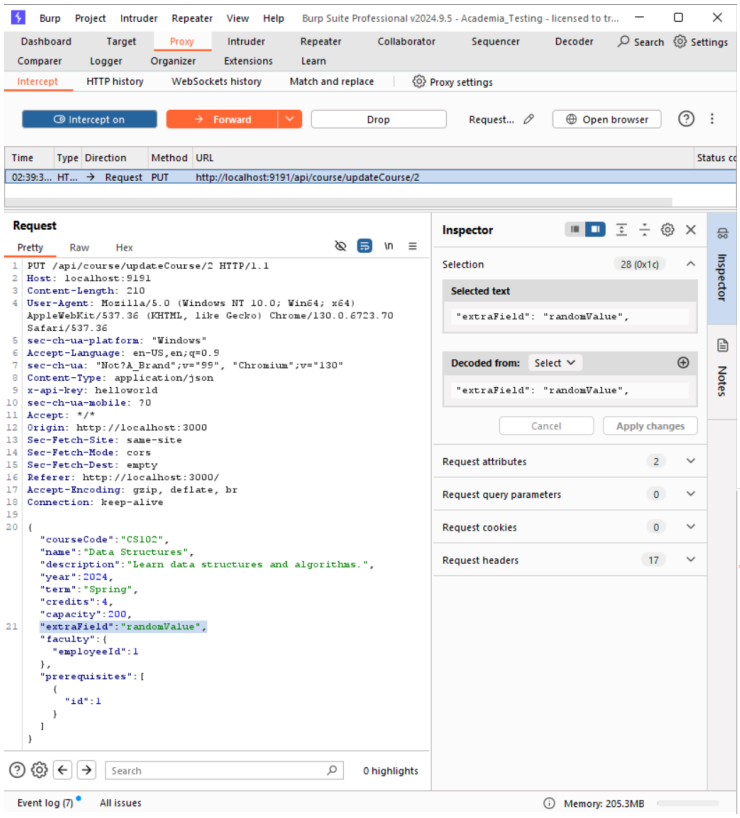
Snapshot of Test Cases:

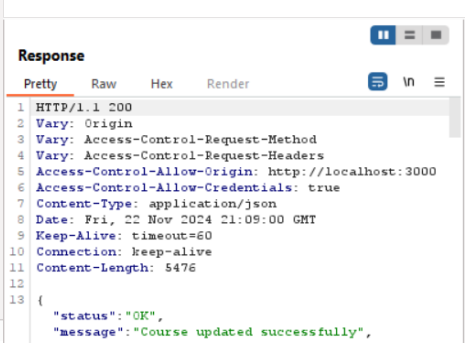
- **TC210:**





- **TC211:**





c. Control Flow Level Bypass Testing

Objective: To test the robustness of the application when users attempt to bypass or manipulate the expected flow of operations.

Test Design:

1. **Backward Control Flow**:
    - Use browser navigation (e.g., back button) to revisit and resubmit forms.
    - Example: Resubmit a course update form after modifying a previously successful request.
2. **Forward Control Flow**:
    - Skip intermediate steps and attempt to directly access restricted actions.
    - Example: Access the course deletion endpoint without completing authentication.
3. **Arbitrary Control Flow Alteration**:
    - Submit requests for actions not permitted in the current context.
    - Example: Attempt to delete a course using an expired session token.

Steps:

1. Map the normal control flow of application interactions.
2. Use Burp Suite to intercept and modify the sequence of requests.
3. Test by:
    - Replaying requests out of order.
    - Modifying session tokens or stateful parameters (e.g., courseID).
4. Observe server responses to ensure that stateful validation is enforced.

Expected Results:

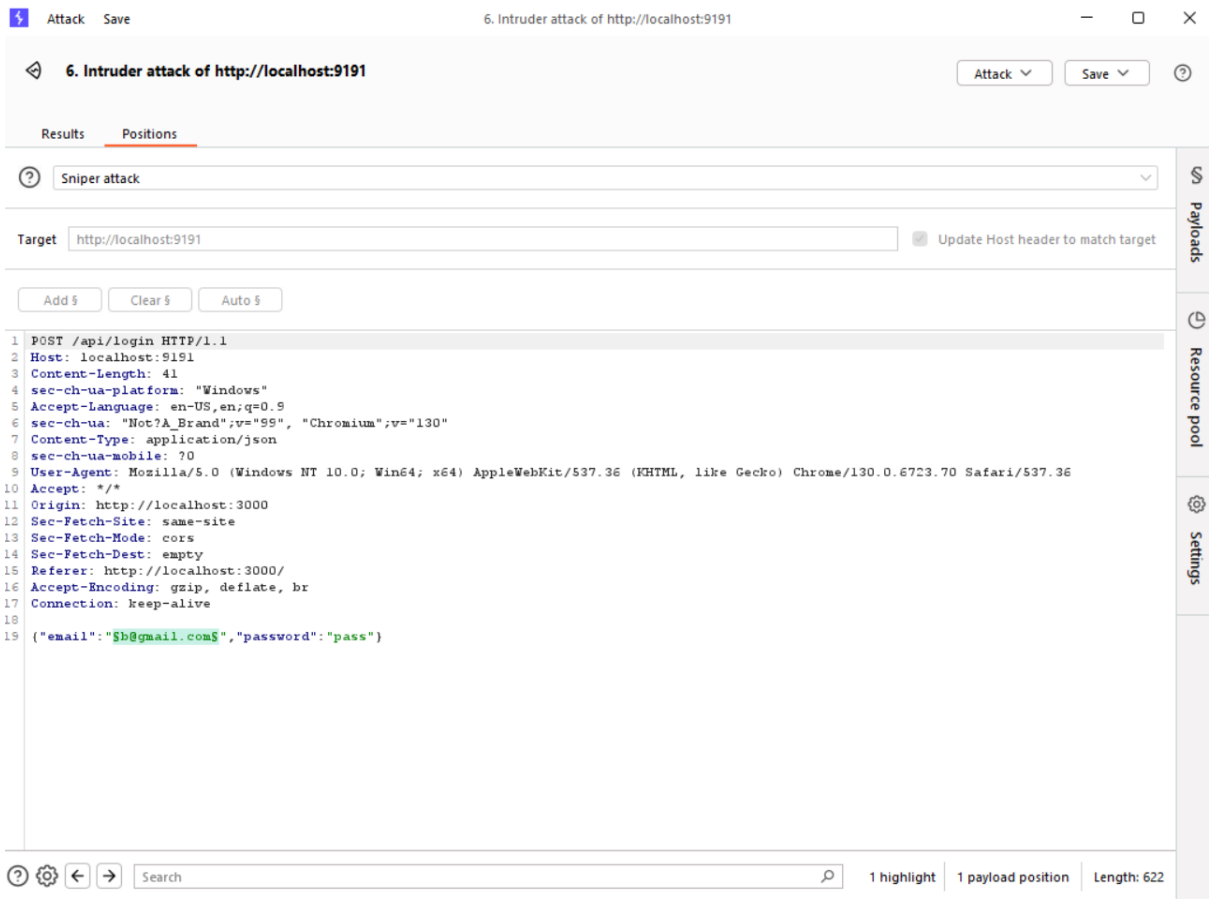
- Actions requiring prerequisites (e.g., authentication) should fail if skipped.
- Unauthorized access attempts should be denied with 403 Forbidden or equivalent responses.

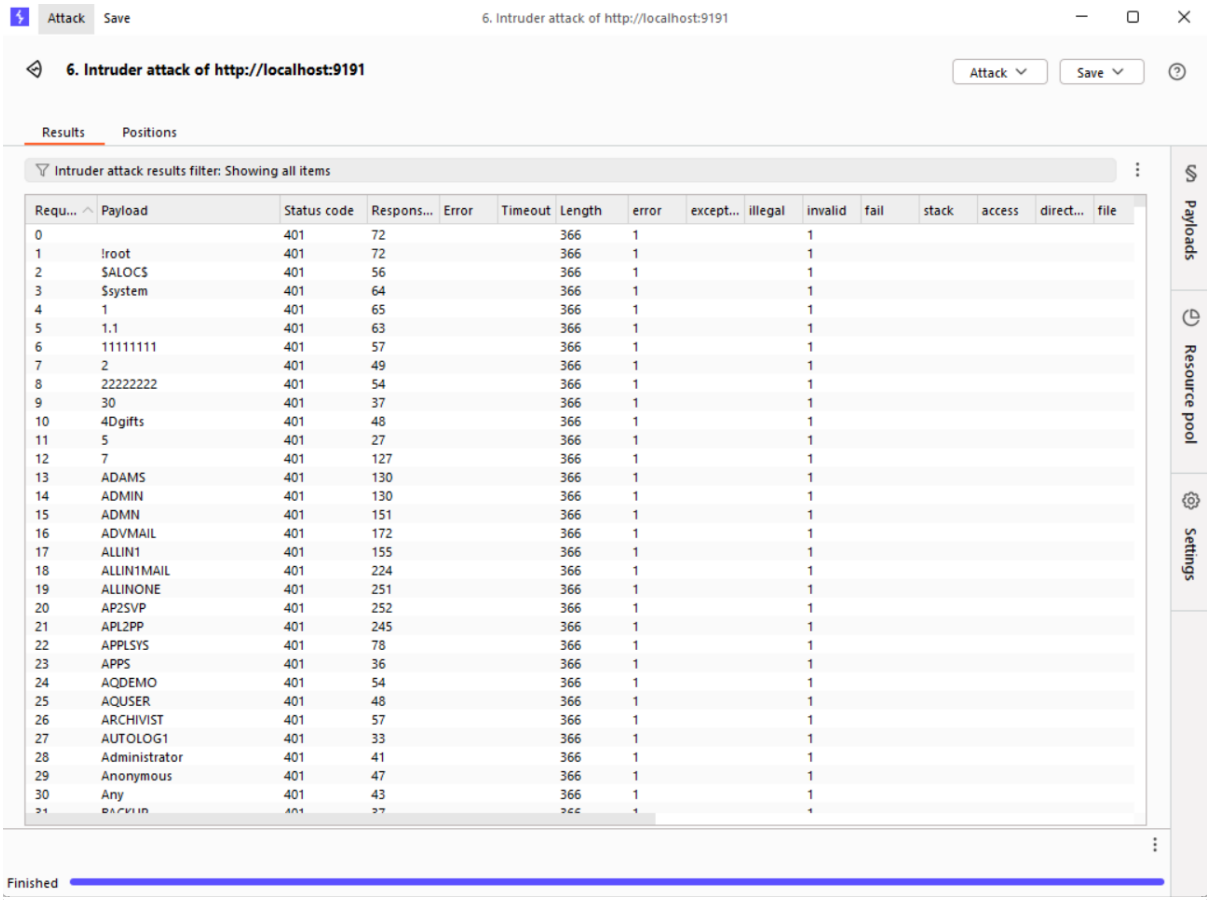
4. Testing Authentication Mechanism in Burp Suite

Burp Suite is a comprehensive web vulnerability testing tool used to intercept, modify, and analyse HTTP traffic for security testing. It helps identify vulnerabilities like SQL injection, XSS, and authentication flaws to enhance application security.

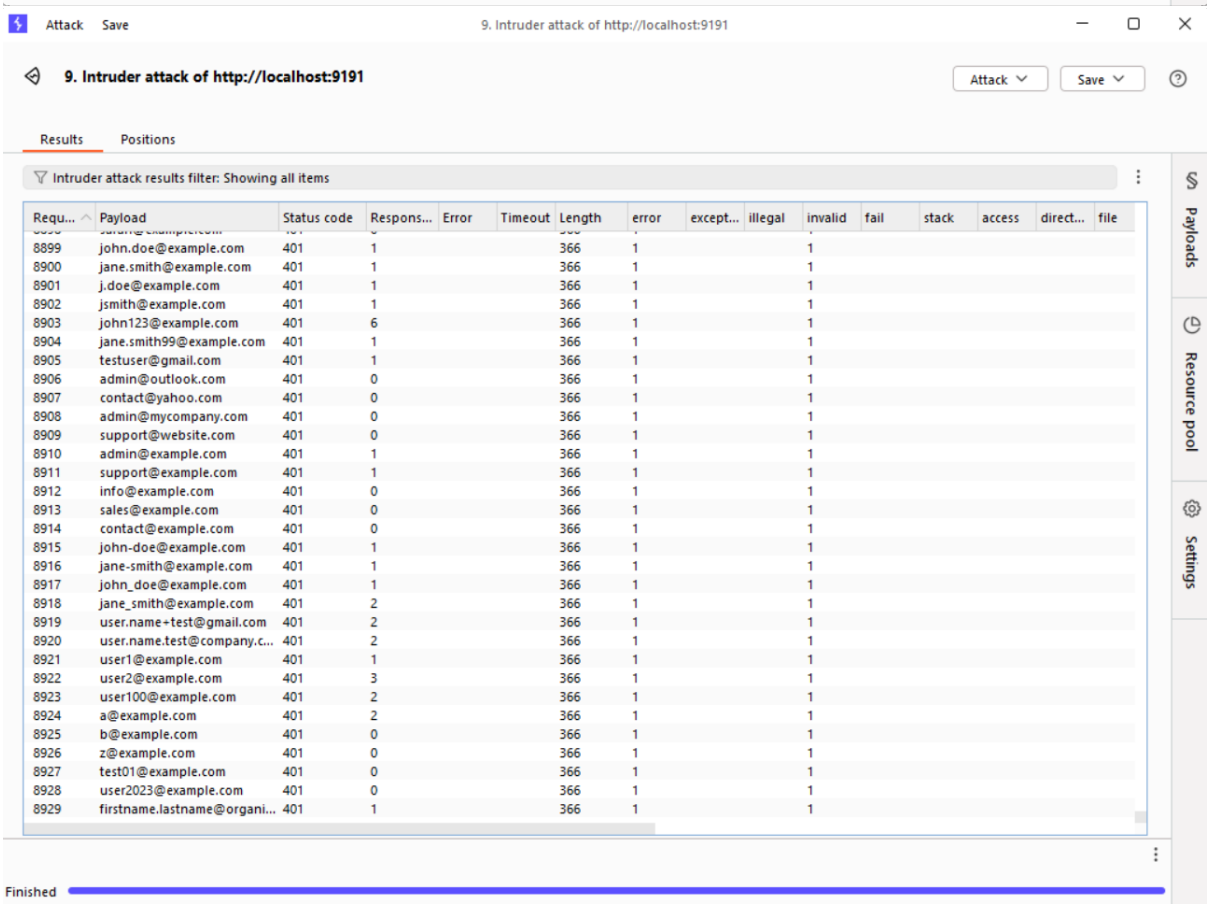
a. Enumerating usernames

Using Burp Intruder, usernames can be enumerated by injecting a list into authentication forms, with valid usernames identified through distinct responses (e.g., 401 Unauthorized for valid ones vs. 404 Not Found for invalid ones). Intruder simplifies this process, making it easy to exploit valid usernames for password-guessing, session hijacking, or social engineering. This greatly reduces attackers' effort and enhances the risk of targeted breaches.





Requ...	Payload	Status code	Respons...	Error	Timeout	Length	error	except...	illegal	invalid	fail	stack	access	direct...	file
0		401	72			366	1			1					
1	!root	401	72			366	1			1					
2	$ALOC$	401	56			366	1			1					
3	$system	401	64			366	1			1					
4	1	401	65			366	1			1					
5	1.1	401	63			366	1			1					
6	11111111	401	57			366	1			1					
7	2	401	49			366	1			1					
8	22222222	401	54			366	1			1					
9	30	401	37			366	1			1					
10	4Dgifts	401	48			366	1			1					
11	5	401	27			366	1			1					
12	7	401	127			366	1			1					
13	ADAMS	401	130			366	1			1					
14	ADMIN	401	130			366	1			1					
15	ADMN	401	151			366	1			1					
16	ADVMAIL	401	172			366	1			1					
17	ALLIN1	401	155			366	1			1					
18	ALLIN1MAIL	401	224			366	1			1					
19	ALLINONE	401	251			366	1			1					
20	AP2SVP	401	252			366	1			1					
21	APL2PP	401	245			366	1			1					
22	APPLSYS	401	78			366	1			1					
23	APPS	401	36			366	1			1					
24	AQDEMO	401	54			366	1			1					
25	AQUSER	401	48			366	1			1					
26	ARCHIVIST	401	57			366	1			1					
27	AUTOLOG1	401	33			366	1			1					
28	Administrator	401	41			366	1			1					
29	Anonymous	401	47			366	1			1					
30	Any	401	43			366	1			1					

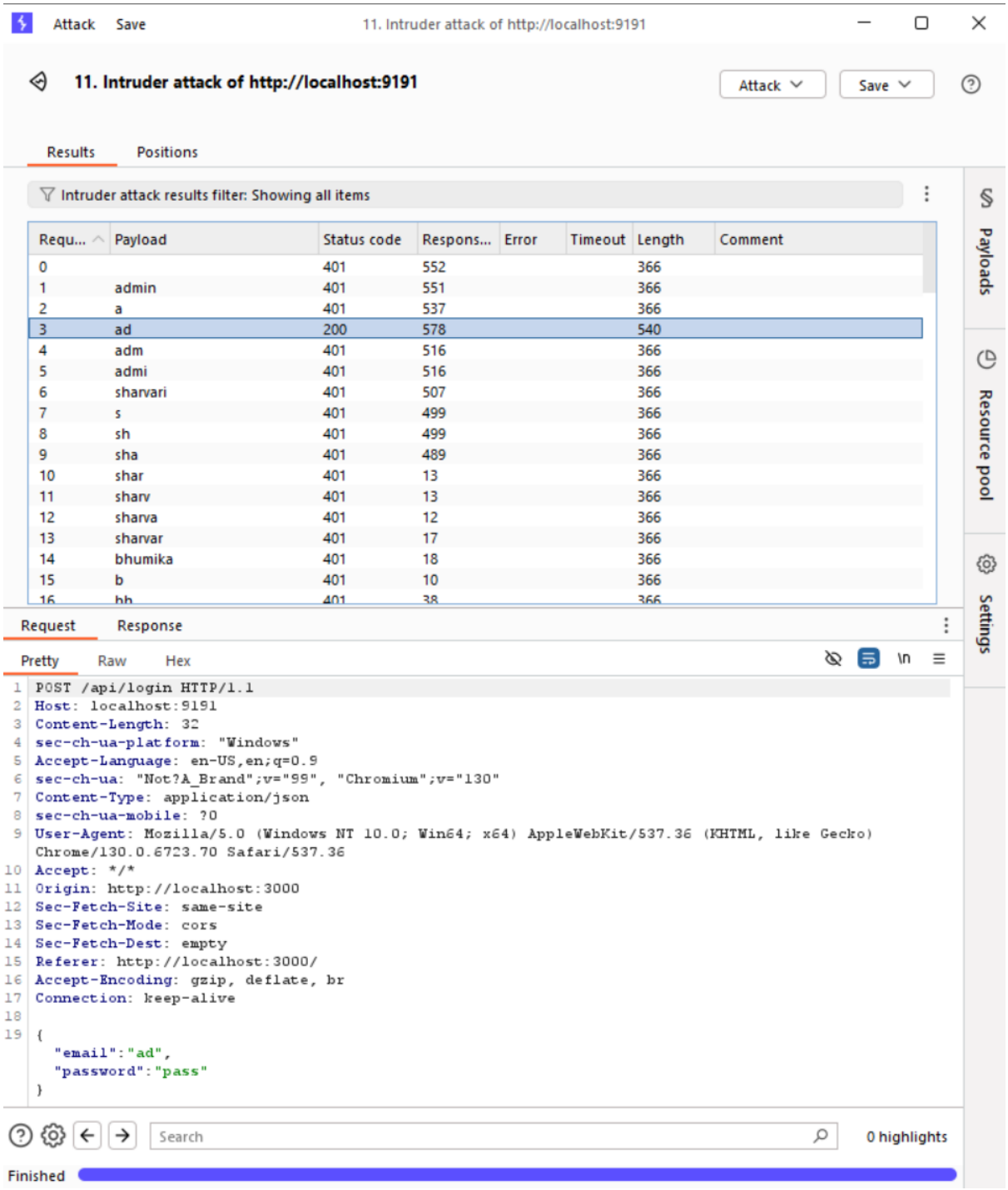


Requ...	Payload	Status code	Respons...	Error	Timeout	Length	error	except...	illegal	invalid	fail	stack	access	direct...	file
8899	john.doe@example.com	401	1			366	1			1					
8900	jane.smith@example.com	401	1			366	1			1					
8901	j.doe@example.com	401	1			366	1			1					
8902	jsmith@example.com	401	1			366	1			1					
8903	john123@example.com	401	6			366	1			1					
8904	jane.smith99@example.com	401	1			366	1			1					
8905	testuser@gmail.com	401	1			366	1			1					
8906	admin@outlook.com	401	0			366	1			1					
8907	contact@yahoo.com	401	0			366	1			1					
8908	admin@mycompany.com	401	0			366	1			1					
8909	support@website.com	401	0			366	1			1					
8910	admin@example.com	401	1			366	1			1					
8911	support@example.com	401	1			366	1			1					
8912	info@example.com	401	0			366	1			1					
8913	sales@example.com	401	0			366	1			1					
8914	contact@example.com	401	0			366	1			1					
8915	john-doe@example.com	401	1			366	1			1					
8916	jane-smith@example.com	401	1			366	1			1					
8917	john_doe@example.com	401	1			366	1			1					
8918	jane_smith@example.com	401	2			366	1			1					
8919	user.name+test@gmail.com	401	2			366	1			1					
8920	user.name.test@company.c...	401	2			366	1			1					
8921	user1@example.com	401	1			366	1			1					
8922	user2@example.com	401	3			366	1			1					
8923	user100@example.com	401	2			366	1			1					
8924	a@example.com	401	2			366	1			1					
8925	b@example.com	401	0			366	1			1					
8926	z@example.com	401	0			366	1			1					
8927	test01@example.com	401	0			366	1			1					
8928	user2023@example.com	401	0			366	1			1					
8929	firstname.lastname@organi...	401	1			366	1			1					

b. Guessing username for known users

Burp Intruder has a built-in username generator that takes an input and produces a list of potential usernames using common patterns. For example, if we were to provide the input Carlos Montoya the generator would return carlos.montoya, mcarlos, and similar combinations.

This is useful in circumstances where we know details of a specific user (for example, their name or email address) but don't know their exact username. It provides a more targeted way of enumerating usernames than a generic name list.

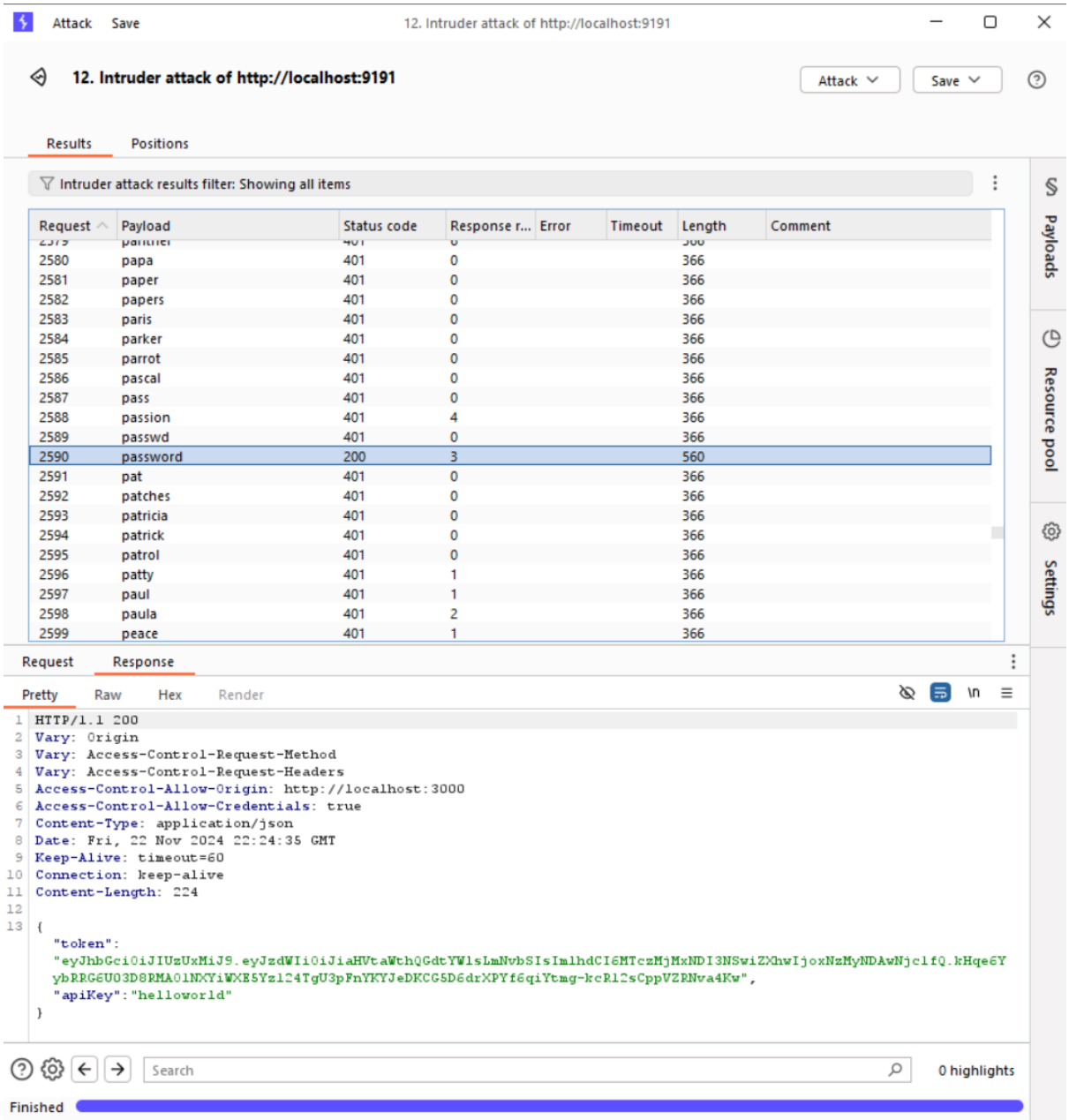


Requ...	Payload	Status code	Respons...	Error	Timeout	Length	Comment
0		401	552			366	
1	admin	401	551			366	
2	a	401	537			366	
3	ad	200	578			540	
4	adm	401	516			366	
5	admi	401	516			366	
6	sharvari	401	507			366	
7	s	401	499			366	
8	sh	401	499			366	
9	sha	401	489			366	
10	shar	401	13			366	
11	sharv	401	13			366	
12	sharva	401	12			366	
13	sharvar	401	17			366	
14	bhumika	401	18			366	
15	b	401	10			366	
16	bh	401	38			366	

c. Brute-forcing passwords with Burp Suite (Dictionary Attack)

Burp Suite provides a number of features that can help us brute-force the password of a given user, gaining access to their account and additional attack surface. For example, we can use a list of common passwords. This is commonly known as a dictionary attack.

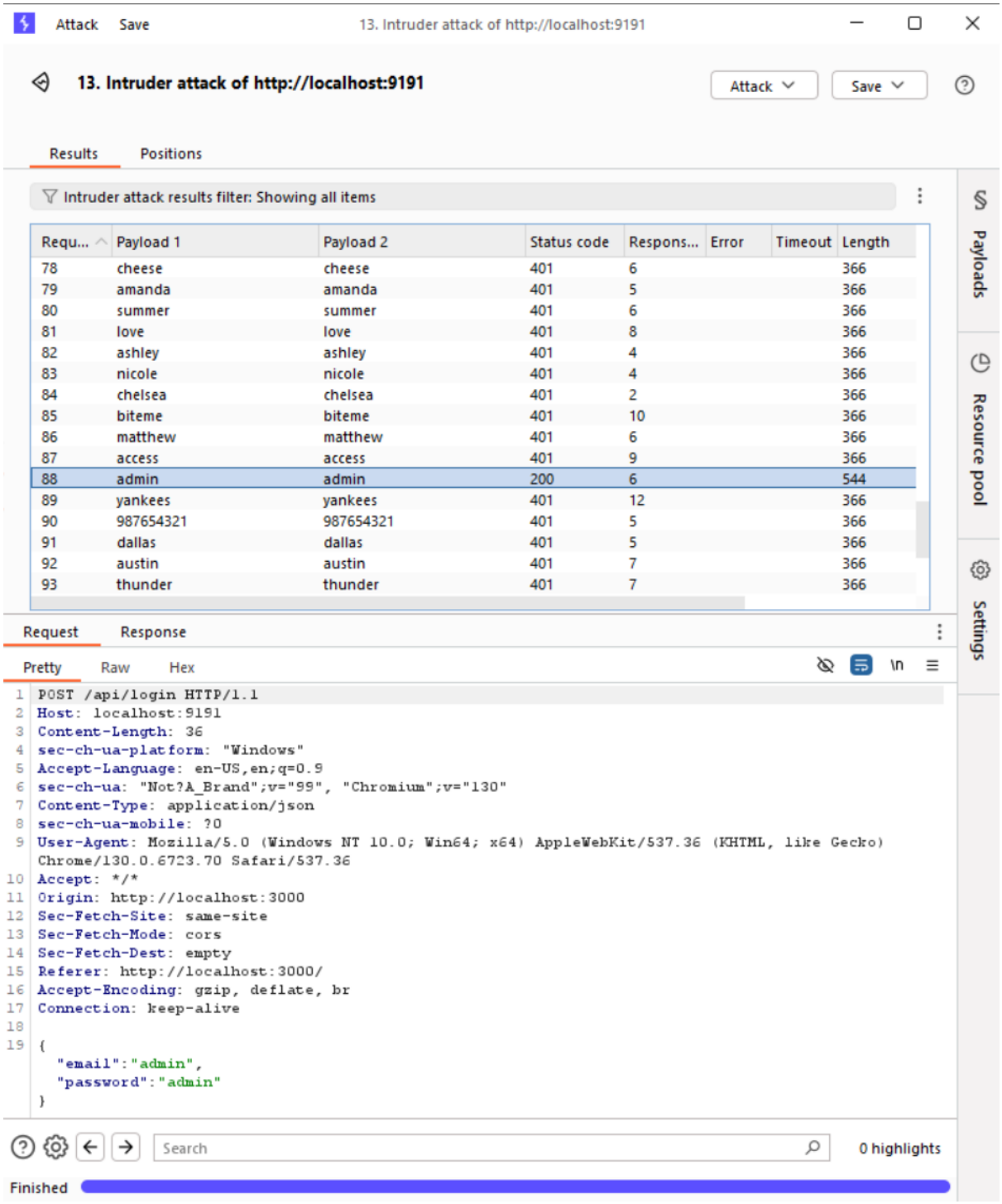
Trying every permutation of a character set. Burp Suite provides a number of features that can help us brute-force the password of a given user, gaining access to their account and additional attack surface.



d. Credential stuffing with Burp Suite

Credential stuffing is a form of brute-force attack in which we attempt to log into a website using known username and password combinations from other websites. These sets of credentials are usually collated from earlier data breaches.

These attacks rely on the fact that users often reuse the same credentials across multiple different sites. Crucially, as we're only attempting to access each account once, defense mechanisms such as account locking are effectively powerless against this kind of attack, although we may still need to bypass any rate limiting that's in place.



Requ...	Payload 1	Payload 2	Status code	Respons...	Error	Timeout	Length
78	cheese	cheese	401	6			366
79	amanda	amanda	401	5			366
80	summer	summer	401	6			366
81	love	love	401	8			366
82	ashley	ashley	401	4			366
83	nicole	nicole	401	4			366
84	chelsea	chelsea	401	2			366
85	biteme	biteme	401	10			366
86	matthew	matthew	401	6			366
87	access	access	401	9			366
88	admin	admin	200	6			544
89	yankees	yankees	401	12			366
90	987654321	987654321	401	5			366
91	dallas	dallas	401	5			366
92	austin	austin	401	7			366
93	thunder	thunder	401	7			366

5. Web Vulnerability Testing Using Burp Suite

Burp Suite Scans: Overview and Utility

Burp Suite Scans automate security testing for web applications by analysing their structure and behaviour. Scans detect vulnerabilities like SQL Injection, XSS, weak session management, and security misconfigurations.

Types of Scans:

1. **Crawl Only**: Maps application structure without testing vulnerabilities.
2. **Audit Only**: Tests specific requests or endpoints for vulnerabilities.
3. **Crawl and Audit**: Combines mapping and vulnerability testing for comprehensive security assessment.

How to Perform a Scan:

1. Start Burp Suite and define the **Target Scope**.
2. Choose a scan type (Crawl, Audit, or both).
3. Configure the scan (e.g., CORS policies, authentication settings).
4. Run the scan and monitor progress in the **Dashboard**.
5. Review issues categorized by severity and confidence.

Common Vulnerabilities:

- Injection attacks (SQL, XSS).
- Security misconfigurations (e.g., missing headers).
- Weak authentication or session management.

Advantages:

- Comprehensive automation of repetitive tasks.
- Real-time issue detection with remediation advice.
- Integration into CI/CD pipelines for continuous testing.

Limitations:

- False positives require manual validation.
- Limited handling of complex workflows like CAPTCHA.

Best Practices:

- Scope scans effectively and monitor performance.
- Validate results, fix issues, and re-scan.

Burp Suite scans save time, ensure compliance, and strengthen application security when combined with manual testing.

A full security scan was conducted using **Burp Suite** to identify potential vulnerabilities in the Academic ERP web application. The scan assessed the application's frontend and backend for common vulnerabilities, focusing on transport security, cross-origin resource sharing (CORS), clickjacking, and information disclosure risks.

Detailed Findings:

The application has **no critical or medium-severity issues**, but **12 issues** were identified, primarily in the **Information** and **Low severity** categories.

1. **Unencrypted Communications**
    - **Severity**: Low, **Confidence**: Certain.
    - **Description**: Communications over HTTP are vulnerable to eavesdropping and man-in-the-middle attacks.
    - **Remediation**: Use HTTPS with Strict-Transport-Security headers.
2. **CORS Misconfigurations**
    - **Severity**: Information, **Confidence**: Certain.
    - **Description**: Permissive CORS policies and lack of Vary: Origin expose the app to cache poisoning and cross-origin attacks.
    - **Instances**: /, /manifest.json, /robots.txt, /static/js/bundle.js.
    - **Remediation**: Use a trusted domain whitelist and avoid * in Access-Control-Allow-Origin.
3. **Clickjacking Vulnerability**
    - **Severity**: Information, **Confidence**: Firm.
    - **Description**: Missing X-Frame-Options or Content-Security-Policy headers allow potential clickjacking.
    - **Affected Pages**: /, /courses.
    - **Remediation**: Add X-Frame-Options: DENY or SAMEORIGIN and use Content-Security-Policy.
4. **Presence of robots.txt File**
    - **Severity**: Information, **Confidence**: Certain.
    - **Description**: Accessible robots.txt discloses directory information, aiding attackers in mapping the application.
    - **Remediation**: Secure sensitive directories with authentication and authorization.

Insights:

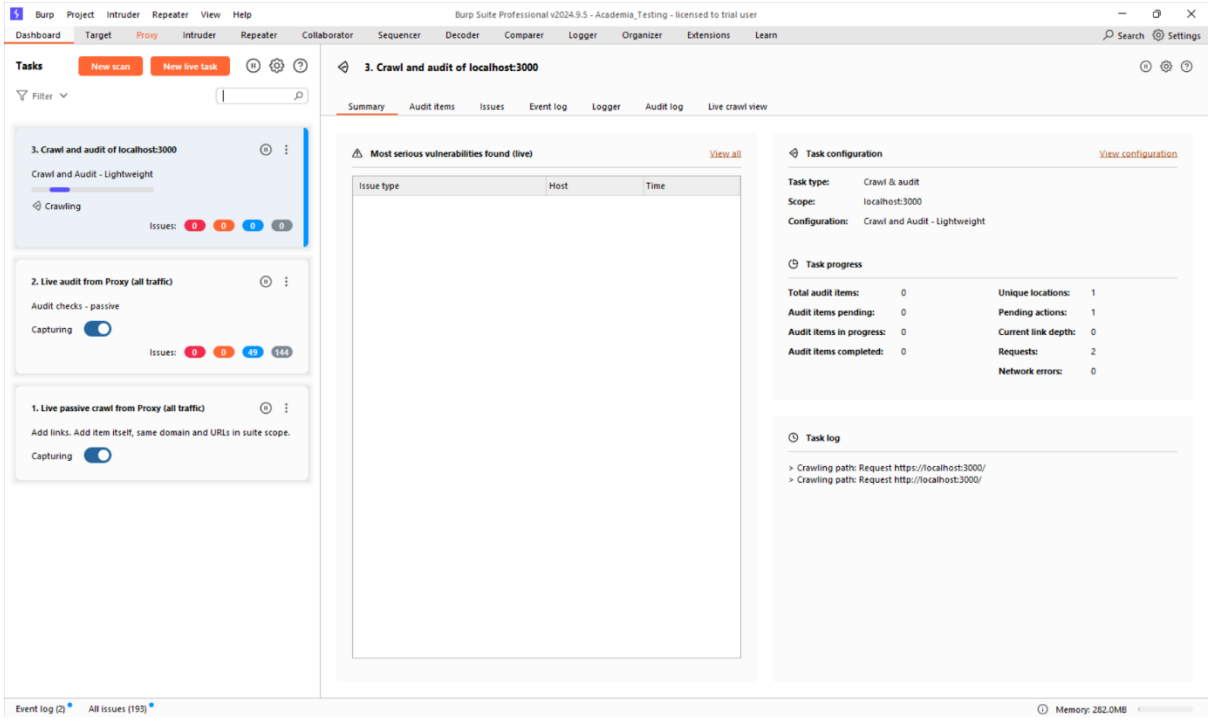
- **Strengths**: No high or medium-severity vulnerabilities; good baseline security.
- **Weaknesses**: Permissive CORS, unencrypted communication, and missing security headers pose risks.

Remediation Plan:

1. **Transport Security**: Implement HTTPS and enforce Strict-Transport-Security.
2. **CORS Policy**: Restrict origins and include Vary: Origin headers.
3. **Clickjacking Protection**: Add X-Frame-Options and Content-Security-Policy.
4. **Sensitive Information**: Restrict access to directories disclosed in robots.txt

The Academic ERP application demonstrates a good security baseline with no critical vulnerabilities. However, several low-severity and informational issues, such as unencrypted communication and permissive CORS policies, should be addressed to enhance overall security. Implementing the recommended remediations will help mitigate potential risks and improve resilience against attacks.

Snapshot of the Full Scan Testing and its Report



Burp Scanner Report

Burp Suite Professional

Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low, Information or False Positive. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

		Confidence			
		Certain	Firm	Tentative	Total
Severity	High	0	0	0	0
	Medium	0	0	0	0
	Low	1	0	0	1
	Information	9	2	0	11
	False Positive	0	0	0	0

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.

Please refer to the Full Scan Report PDF attached in the submission folder for more details.
https://github.com/sharvari-00/Software_Testing/blob/main/Burp%20Scanner%20Report.pdf

6. Client-Side Validation Testing Using Selenium

Selenium is an open-source framework for automating browsers. It enables developers and testers to write Java programs to simulate user actions like filling forms, clicking buttons, and navigating websites in browsers such as Chrome, Firefox, and Edge.

Why Use Selenium (Java) for Client-Side Validation Bypass Testing?
Selenium with Java is ideal for bypassing client-side validations because:

- **Direct Interaction:** It can directly manipulate input fields, bypassing JavaScript-based validations in real-time.
- **Automation:** Automates repetitive testing scenarios and edge cases.
- **Input Manipulation:** Tests with invalid or unexpected data, verifying that the server handles all validation and security checks correctly.
- **Real Browser Context:** Runs tests in a real browser, mimicking user behaviour.

a. Testing the Login Page Using Selenium

We developed LoginTests.java to automate comprehensive testing of a login page using Selenium, ensuring functionality and security. The file includes tests like Valid Login Test for correct credentials, Invalid Login Test for rejecting incorrect inputs, and Empty Fields Test to validate mandatory fields. Advanced security tests like SQL Injection and XSS are implemented to check against common vulnerabilities. Additionally, Long Input Test and Boundary Value Testing evaluate the system's handling of maximum input lengths and special characters. These tests run automatically, simulating real user interactions, verifying error messages, and ensuring a secure and user-friendly login experience.

Test Cases for Login:

Test Case ID	Test Name	Steps	Expected Outcome	Result
LGT001	Valid Login Test	1. Navigate to Login Page. 2. Enter valid credentials (email and password). 3. Click "Submit".	User is redirected to the dashboard or courses page after successful login.	Pass
LGT002	Invalid Login Test	1. Navigate to Login Page. 2. Enter invalid credentials. 3. Click "Submit".	An error message, "Invalid credentials", is displayed.	Pass
LGT003	Empty Fields Test	1. Navigate to Login Page. 2. Leave the email and password fields empty. 3. Click "Submit".	An error message, "Email is required", is displayed.	Pass
LGT004	SQL Injection Test	1. Navigate to Login Page. 2. Enter SQL injection code in the email field (e.g., ' OR 1=1--). 3. Click "Submit".	The application should not redirect to the dashboard; instead, it should display "Invalid credentials".	Pass
LGT005	XSS Test	1. Navigate to Login Page. 2. Enter XSS payload in the email field (e.g., <script>alert('XSS')</script>). 3. Click "Submit".	The application should not execute the JavaScript. The login should fail with an "Invalid credentials" error.	Pass

LGT006	Long Input Test	1. Navigate to Login Page. 2. Enter very long strings (e.g., 5000 characters) in both fields. 3. Click "Submit".	The application should reject excessively long inputs with a validation error or truncate them safely.	Fail

b. Testing the Course Update Page Using Selenium:

We developed CourseUpdateTests.java to automate comprehensive testing of the Course Update Page using Selenium. The file includes a Valid Course Update Test to ensure updates are saved correctly with valid inputs, and an Empty Fields Test to verify error messages for missing data. Advanced tests like Invalid Data Type Test validate numeric fields against incorrect inputs, while Special Characters Test ensures proper handling of special characters. Additionally, Max Length Input Test checks the system's behaviour with excessively long inputs, and Dropdown Selection Test validates the saving of dropdown selections (e.g., faculty). These automated tests verify robust validation, functionality, and a seamless course update experience.

Test Cases for Course Update

Test Case ID	Test Name	Steps	Expected Outcome	Result
UCT001	Valid Course Update Test	1. Navigate to Courses Page. 2. Click "Edit" for a course. 3. Fill in all fields with valid inputs. 4. Click "Update".	The course is updated successfully, and a success message is displayed.	Pass
UCT002	Empty Fields Test	1. Navigate to Courses Page. 2. Click "Edit" for a course. 3. Leave all fields empty. 4. Click "Update".	An error message, "Can't be empty", is displayed for all required fields.	Pass
UCT003	Invalid Data Type Test	1. Navigate to Courses Page. 2. Click "Edit" for a course. 3. Enter text in numeric fields (e.g., "invalid_data" in Credits). 4. Click "Update".	An error message, "Invalid input", is displayed for fields where invalid data types were entered.	Fail
UCT004	Special Characters Test	1. Navigate to Courses Page. 2. Click "Edit" for a course. 3. Enter special characters (e.g., "!@#$%^&*()") in all text fields. 4. Click "Update".	Special characters are either accepted (if valid) or rejected with a proper validation error.	Fail
UCT005	Max Length Input Test	1. Navigate to Courses Page. 2. Click "Edit" for a course. 3. Enter very long strings (e.g., 256 characters) in text fields. 4. Click "Update".	An error message, "Input exceeds max length", is displayed for fields that exceed maximum length.	Fail

Screen Recording of Test Cases:
Please refer to this video for the detailed execution and results of all test cases, including the validations and functionality tests in Selenium:
https://drive.google.com/file/d/1BshJ0n2cT4LoON8Sc0LcK-JVe6thvIXa/

Selenium is highly effective for testing client-side validations and functionality due to its ability to automate real browser interactions, directly manipulate input fields, and bypass JavaScript-based checks. It ensures thorough testing of edge cases, security vulnerabilities (like SQL Injection and XSS), and input validations for both login and course update functionalities. Further, Selenium can be extended for continuous testing in CI/CD pipelines, integrated with reporting tools for detailed results, and used with parallel execution frameworks to test across multiple browsers and platforms.

Conclusion

The testing process for the Academic ERP system highlights the critical role of comprehensive validation, secure input handling, and functional correctness for key features like login and course updates. Utilizing Selenium for automation allowed us to simulate real user interactions within browsers, ensuring a thorough assessment of functionality, validation rules, and security mechanisms. Client-side validation bypass testing emphasized the importance of secure coding practices, with tests uncovering vulnerabilities like improper handling of long inputs, special characters, and invalid data types.

In addition to **Selenium**, we incorporated **Burp Suite**, a robust tool for web application security testing, to analyze and mitigate backend vulnerabilities. Burp Suite was instrumental in intercepting and analyzing HTTP requests and responses, identifying potential weaknesses such as insecure APIs, improper session handling, and data leakage. This complemented the automated Selenium tests by providing deeper insights into server-side vulnerabilities and enabling targeted fixes.

The development of LoginTests.java and CourseUpdateTests.java files facilitated efficient and repeatable testing of edge cases, error handling, and security threats, such as SQL Injection and XSS. Combined with tools like **Postman** for API validation and Burp Suite for security analysis, this testing process ensured the application's robustness, integrity, and security.

Future enhancements in testing frameworks, leveraging advanced tools like OWASP ZAP, Selenium Grid, and AI-driven test generation, can significantly improve security, scalability, and cross-platform consistency. Prioritizing automation, real-time monitoring, and accessibility testing ensures reliability and a seamless user experience. These advancements provide a robust foundation for reliable software deployment and streamline integration into CI/CD workflows. Continuous monitoring and automated testing will enhance application quality and security, enabling organizations to maintain high standards while adapting to evolving challenges.

Codebase Link

By-pass Testing on Academic ERP - https://github.com/sharvari-00/Software_Testing

Team Contribution

Together:

- Collaborated on the **documentation**, test case tables, and overall test strategy.
- Explored **Selenium** and **Burp Suite** for testing client-side validation and automating processes.
- Used **developer browser inspection tools** for analyzing and testing field validations, input constraints, and debugging.
- Planned and structured testing for client-side, server-side, and API validations.

Sharvari's Contribution:

- **Login Testing:** Developed and automated **LoginTests.java** with Selenium, covering Valid Login, Invalid Login, Empty Fields, SQL Injection, XSS, and Long Input Validation. Conducted developer tools testing for login parameter inspection.
- **Burp Suite:** Focused on client-side validation bypass for login fields, testing maximum length, invalid characters, and data type checks.
- **API Testing:** Tested login-related API endpoints using Postman and Burp Suite, ensuring proper error handling and secure access.

Bhumika's Contribution:

- **Course Update Testing:** Automated **CourseUpdateTests.java** with Selenium for Valid Course Update, Empty Fields, Invalid Data Type, Special Characters, Max Length Input, and Dropdown Selection. Performed developer tools testing for bypass validation on the Course Update page.
- **Burp Suite:** Conducted parameter and scripting validation bypass for Course Update fields, focusing on data format, inter-field dependencies, and preset field constraints.
- **API Testing:** Verified Course Update API endpoints for authentication, validation, and error handling.