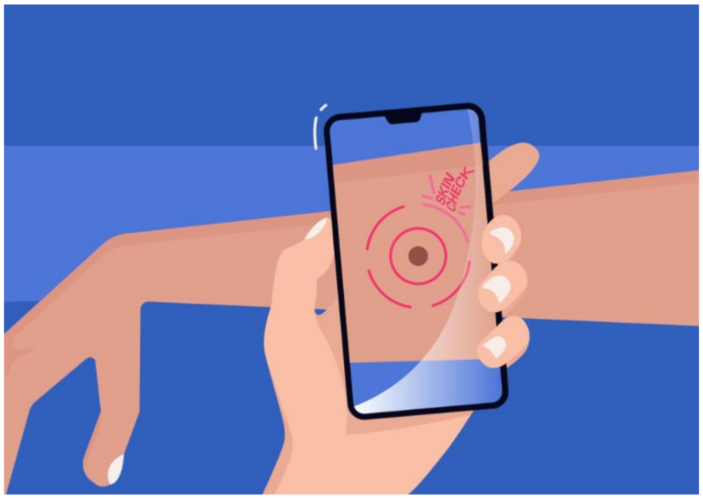
Automated Testing



<u>Project Title:</u> AI-Based Derma Checker Web Application

By

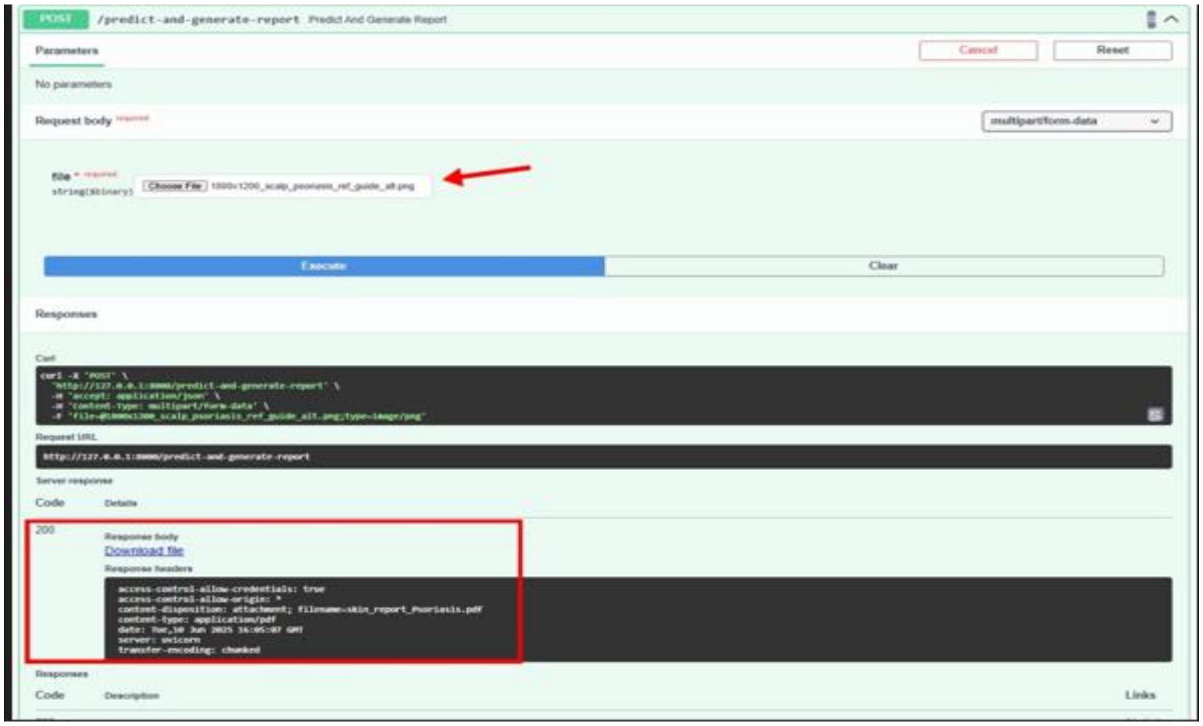
Rafia Arshad	**(210275)**
M. Burhan Ahmed	**(210287)**
Sadia Yaqoob	**(210284)**
Noor ul Huda	**(210294)**
Abdur Rehman	**(210312)**

CE-362 Software Engineering

05 April 2025

1.Automated API Testing Using Swagger UI:

- Swagger UI was used to perform automated testing of the /predict-and-generate-report API endpoint.
- This tool provides an interactive interface for executing and validating REST API calls without the need for external clients.
- By uploading a sample skin image, testers were able to verify the complete backend flow ensuring that the system processes the file correctly, triggers the deep learning model, and returns a downloadable PDF report.
- It also allowed inspection of response status codes, headers (such as Content-Disposition for file download), and real-time system behavior. This approach ensured the API met both functional and data integrity requirements efficiently.

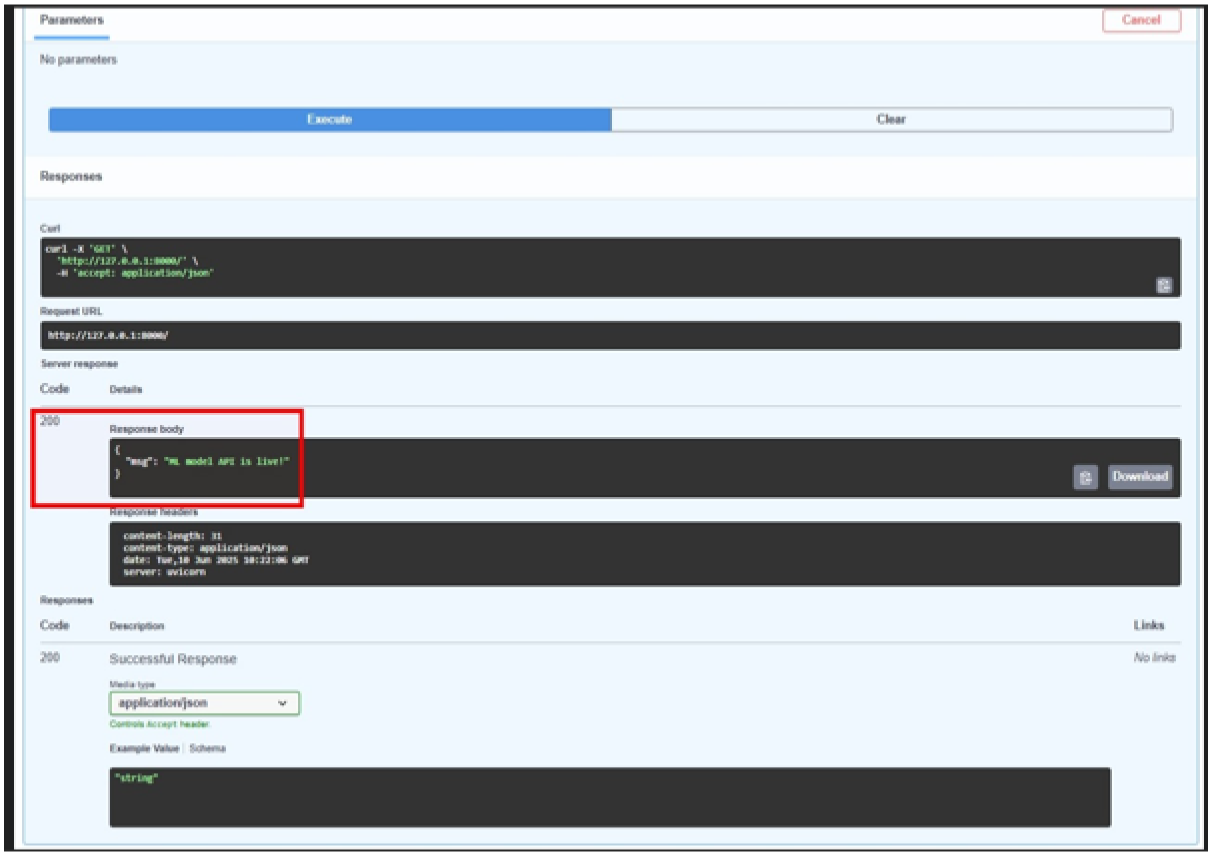


```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict-and-generate-report' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@1000x1200_scalp_psoriasis_ref_guide_alt.png;type=image/png'
```

```
http://127.0.0.1:8000/predict-and-generate-report
```

200

```
access-control-allow-credentials: true
access-control-allow-origin: *
content-disposition: attachment; filename=skin_report_Psoriasis.pdf
content-type: application/pdf
date: Tue,10 Jun 2025 16:05:07 GMT
server: uvicorn
transfer-encoding: chunked
```



Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/
```

Server response

Code	Details
200	Response body

```
{
  "msg": "ML model API is live!"
}
```

Response headers

```
 content-length: 31
 content-type: application/json
 date: Tue,10 Jun 2025 10:22:06 GMT
 server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

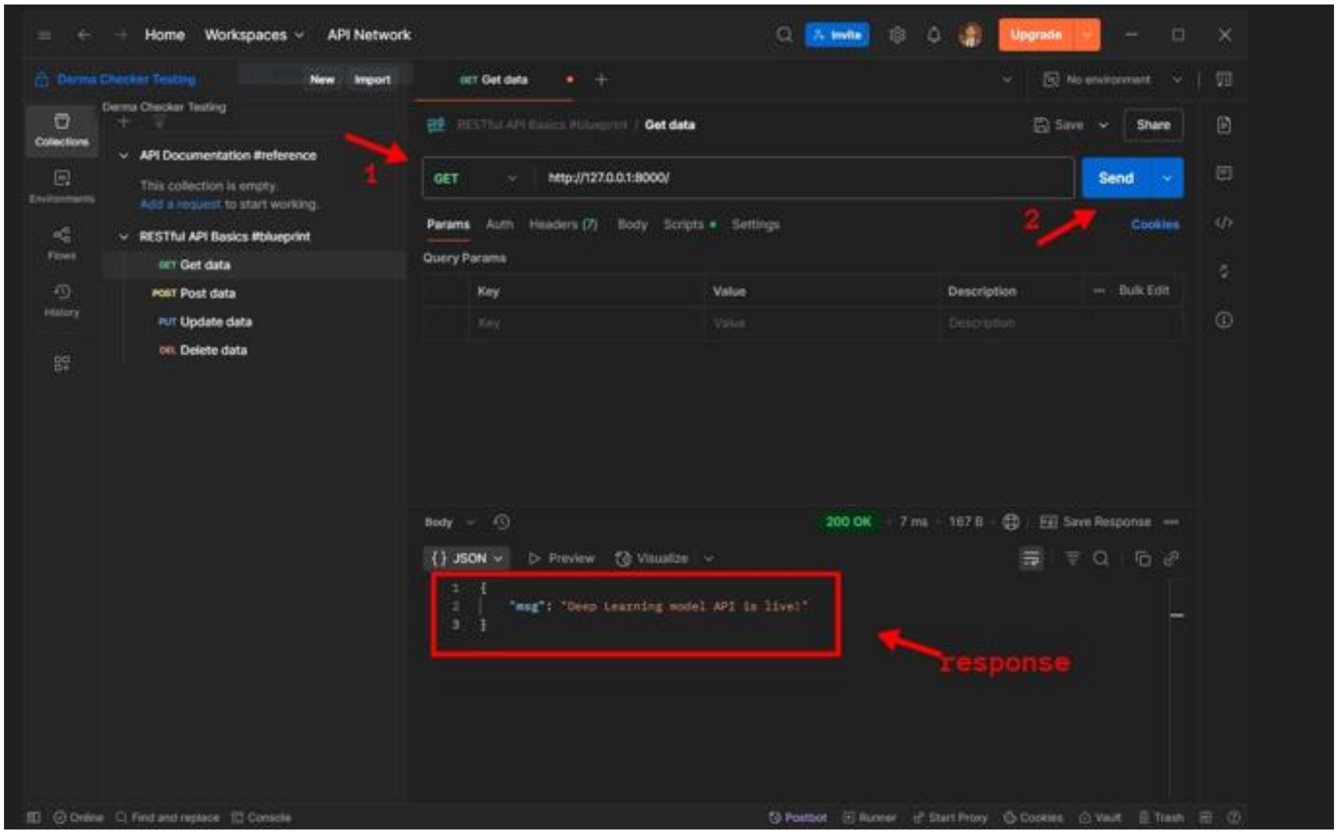
Media type: application/json

Controls Accept header.

```
"string"
```

2.Automated Testing Using Postman:

- Postman was utilized to automate and validate the REST API endpoints of the Skin Disease Detection Web Application.
- Through a series of structured test cases, it helped verify correct request formation, response status codes, and response content including headers, data, and file downloads.
- Postman allowed simulation of both positive and negative test scenarios, such as valid image uploads, invalid inputs, and error handling in payment and report generation flows. This ensured API reliability, consistent performance, and seamless integration with the frontend under various conditions.



Derma Checker Testing

API Documentation #reference

This collection is empty. Add a request to start working.

RESTful API Basics #blueprint

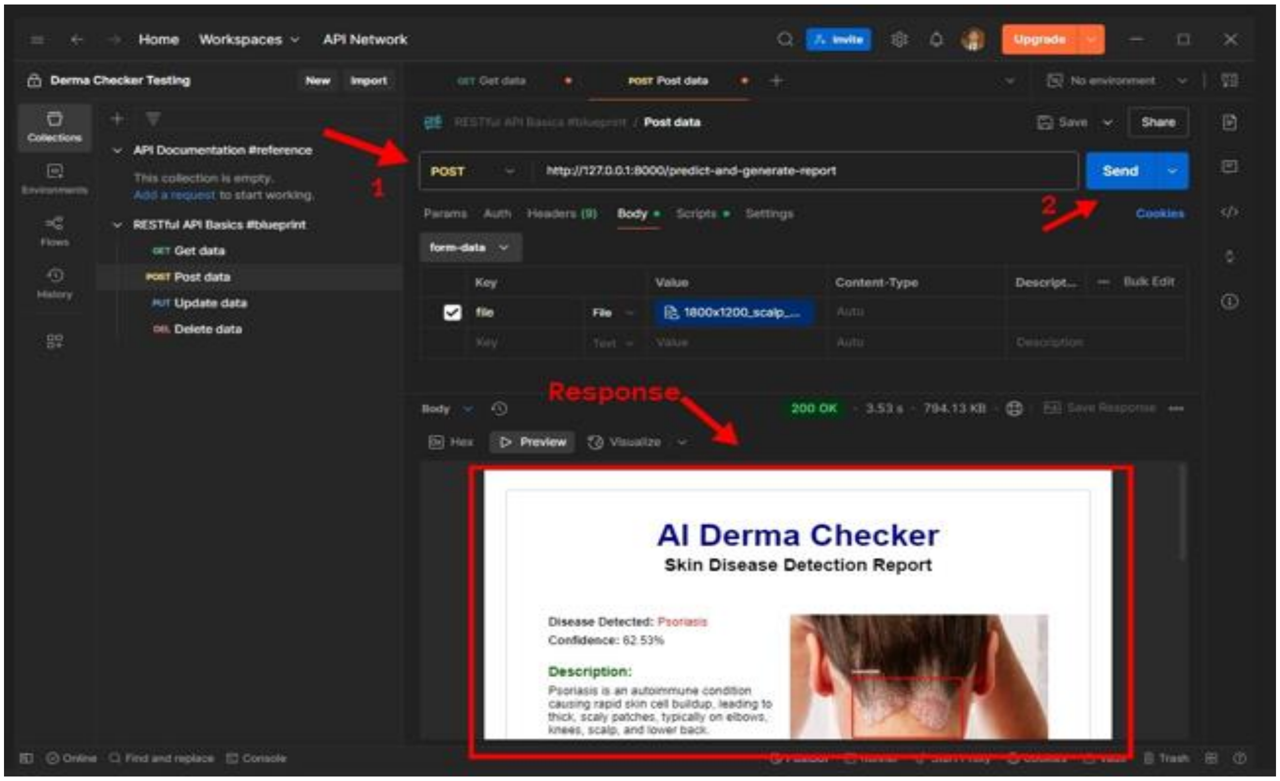
- GET Get data
- POST Post data
- PUT Update data
- DEL Delete data

RESTful API Basics #blueprint / Get data

GET http://127.0.0.1:8000/

200 OK · 7 ms · 167 B

```
{
    "msg": "Deep Learning model API is live!"
}
```



RESTful API Basics #blueprint / Post data

POST http://127.0.0.1:8000/predict-and-generate-report

Body: form-data

Key		Value	Content-Type	Description
✓ file	File	1800x1200_scalp_...	Auto	
Key	Text	Value	Auto	Description

200 OK · 3.53 s · 794.13 KB

AI Derma Checker

Skin Disease Detection Report

Disease Detected: Psoriasis

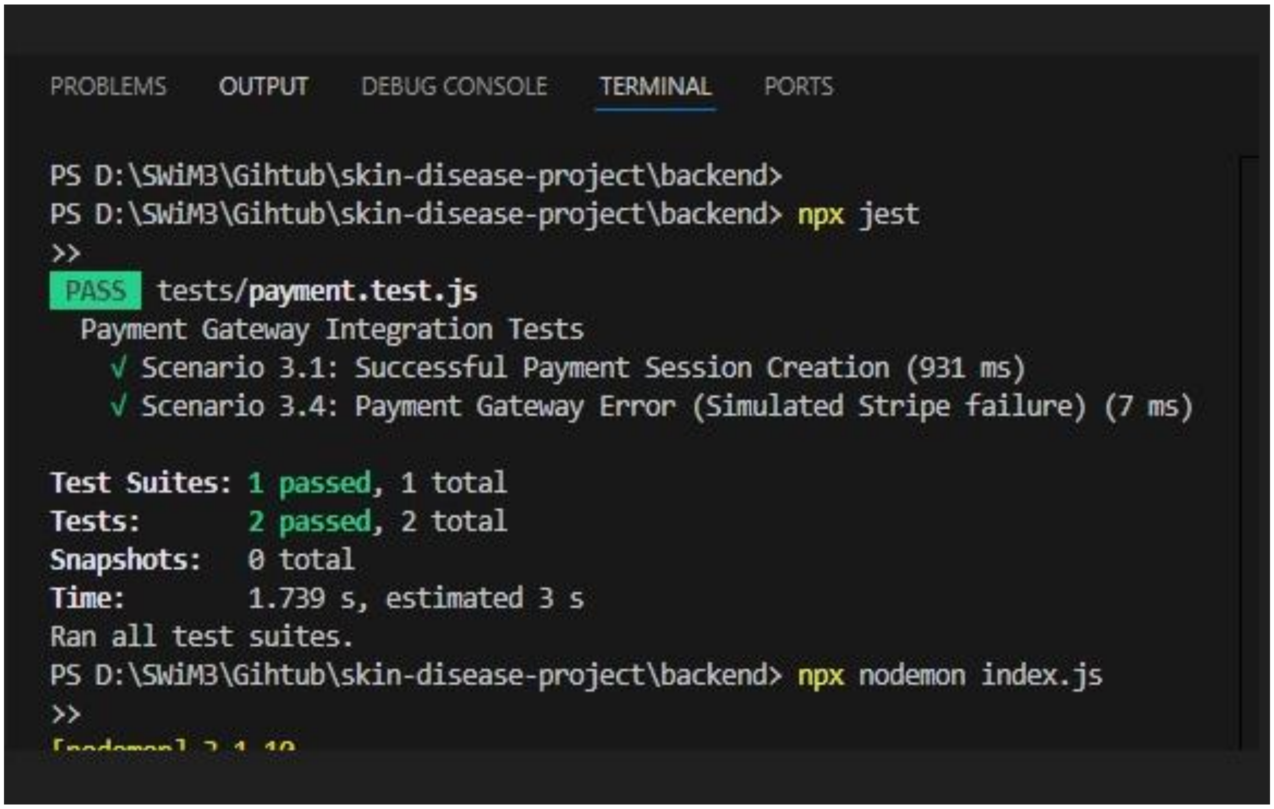
Confidence: 62.53%

Description:

Psoriasis is an autoimmune condition causing rapid skin cell buildup, leading to thick, scaly patches, typically on elbows, knees, scalp, and lower back.

3.Automated Testing Using Jest:

- Jest was used to perform automated unit testing for the backend logic of the application. It allowed developers to write test cases for individual functions such as input validation, API response formatting, and error handling.
- By running tests automatically, Jest helped identify bugs early in the development cycle and ensured that code changes did not break existing functionality.
- This improved the reliability and maintainability of the backend system.

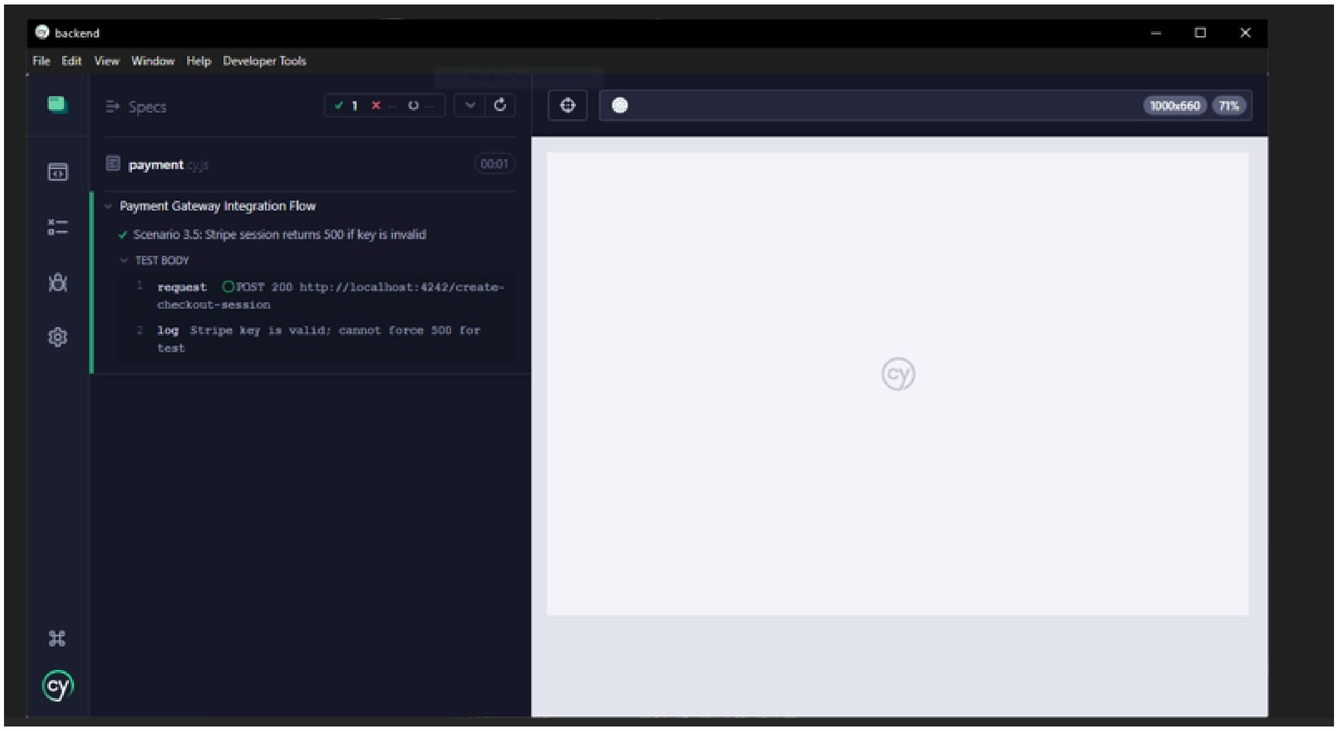


```
PS D:\SWiM3\Gihtub\skin-disease-project\backend>
PS D:\SWiM3\Gihtub\skin-disease-project\backend> npx jest
>>
 PASS  tests/payment.test.js
  Payment Gateway Integration Tests
    √ Scenario 3.1: Successful Payment Session Creation (931 ms)
    √ Scenario 3.4: Payment Gateway Error (Simulated Stripe failure) (7 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.739 s, estimated 3 s
Ran all test suites.
PS D:\SWiM3\Gihtub\skin-disease-project\backend> npx nodemon index.js
>>
```

4. Automated Testing using Cypress:

- Cypress was used to automate the testing of Test Scenario 3.5, which focuses on the payment gateway integration flow.
- The test ensures that users can successfully initiate and complete payments, and that the application handles different outcomes—such as success, failure, or interruption—appropriately.
- Through automated interaction with form fields and real-time assertions, Cypress validated that payment status, confirmation messages, and access to the report were correctly handled. This testing helped ensure a smooth and secure transaction experience for end-users.



Payment Gateway Integration Flow

✓ Scenario 3.5: Stripe session returns 500 if key is invalid

TEST BODY

```
1 request  POST 200 http://localhost:4242/create-checkout-session
2 log  Stripe key is valid; cannot force 500 for test
```

Summary:

Scenario No.	Tool Used	Description
3.1	Jest	Successful payment session creation
3.4	Jest	Simulated Stripe gateway failure
3.5	Cypress	Stripe failure due to missing or invalid API key