

Aegis Secure – API Testing



IT314 - Software Engineering

Group - 35

❖ Introduction

This report documents the API testing carried out on the Aegis Secure Backend, hosted at:

Base URL: <https://aegis14211-aegissecurebackend.hf.space>

Cyber URL: <https://akshatbhatt515334-aegis-secure-api.hf.space>

The purpose of this testing is to ensure that the backend APIs used in the Aegis Secure (Android application) are functioning correctly, securely and consistently. Testing was performed using Postman. All the positive and negative (edge) cases are considered in the testing.

❖ Test Cases

➤ TC-01 : Register New User (Positive)

Description:

Test registering a new user with a new email to see if the system accepts it.

Output:

The screenshot displays a REST client interface for a test case. The top bar shows the URL `POST /auth/register` and a `Send` button. The `Body` tab is selected, showing a JSON payload:

```
{  "name": "Tester3",  "email": "tester3@gmail.com",  "password": "Test@333"}
```

. Below the body, the response status is `200 OK` with a response time of `4.54 s` and a size of `527 B`. The `Test Results` tab is also visible, showing a `PASSED` status with the message: `Status code is 200 for successful registration`.

Verdict: PASS

The API accepted the new email and created the user successfully.

➤ TC-02 : Register With Existing Email (Negative)

Description:

Try registering with the existing email to check if the system blocks it.

Output:

The screenshot displays a REST client interface for a POST request to `/auth/register`. The request body is a JSON object with the following details:

```
1 {
2   "name": "Jenish",
3   "email": "jenishpatel..@gmail.com",
4   "password": " "
5 }
```

The response status is **400 Bad Request**. The response body is a JSON object:

```
1 {
2   "detail": "Email already registered"
3 }
```

The Test Results section shows two assertions:

- FAILED** Status code is 200 for successful registration | AssertionError: expected 400 to deeply equal 200
- PASSED** Status code is 400 and message indicates email already registered

Verdict: PASS

The API blocked the registration because the email was already used.

➤ TC-03 : Login with Valid Credentials (Positive)

Description:

Log in using correct email and password to check if login works.

Output:

The screenshot displays a REST client interface for a POST request to `POST /auth/login`. The request body is a JSON object with the following structure:

```
1 {
2   "email": "jenishpatel@gmail.com",
3   "password": " "
4 }
```

The response is a 200 OK status, received in 1.65 seconds with a body size of 708 B. The response body is a JSON object:

```
1 {
2   "token": "eyJlbWVpbmI6Implbm1zaHBhdGVsOTk5OTk5OGdtYW1sLmNvbSI6InVzZXJfawQ1OjI2OTFlMWEyNzRhMzdhdnZU2NGI1OGM0ZjMiLCJleHAiOjE3NjYyMjUzMjF9.5YD5ze2Y-S89h6w1SWrBDuifLnFntzDnYacesl7B4dvc",
3   "verified": true
4 }
```

Below the response, the Test Results section shows three passed assertions:

- PASSED** Status code is 200
- PASSED** Response has a token property
- PASSED** Verified property is true

Verdict: PASS

The API logged in the user and returned a valid token.

➤ TC-04 : Login With Wrong Password (Negative)

Description:

Try logging in with a correct email but wrong password.

Output:

The screenshot displays a REST client interface for a POST request to `/auth/login`. The request body is a JSON object with the following content:

```
1 {
2   "email": "jenishpate1@gmail.com",
3   "password": "xyz"
4 }
```

The response status is **400 Bad Request**. The response body is a JSON object:

```
1 {
2   "detail": "Incorrect password"
3 }
```

The Test Results section shows three failed assertions:

- FAILED** Status code is 200 | AssertionError: expected response to have status code 200 but got 400
- FAILED** Response has a token property | AssertionError: expected { detail: 'Incorrect password' } to have property 'token'
- FAILED** Verified property is true | AssertionError: expected undefined to deeply equal true

Verdict: PASS

The API rejected the login attempt because the password was wrong.

➤ TC-05 : Login With Missing Fields (Negative)

Description:

Try logging in without entering email or password.

Output:

The screenshot displays a REST client interface for a POST request to `/auth/login`. The request body is a JSON object with empty string values for `email` and `password`. The response is a 400 Bad Request with a JSON body indicating the user was not found. Below the response, the Test Results section shows three failed assertions.

Request:

```
POST {{base_url}} /auth/login
```

Body (raw):

```
{
  "email": "",
  "password": "xyz"
}
```

Response (JSON):

```
{
  "detail": "User not found"
}
```

Test Results (0/3):

- FAILED** Status code is 200 | AssertionError: expected response to have status code 200 but got 400
- FAILED** Response has a token property | AssertionError: expected { detail: 'User not found' } to have property 'token'
- FAILED** Verified property is true | AssertionError: expected undefined to deeply equal true

Verdict: PASS

The API gave an error for missing details.

➤ TC-06 : Get User Details with Valid Token (Positive)

Description:

Call /auth/me with a valid token to get user details.

Output:

The screenshot shows a REST client interface for a request to `AegisSecure_API / 01 - Auth / GET /auth/me`. The request method is `GET` and the URL is `{{base_url}} /auth/me`. The `Headers` tab is active, showing 7 headers, with the `Authorization` header set to `Bearer {{auth_token}}`. The `Body` tab is also active, showing a `200 OK` response with a status of `200 OK`, a response time of `495 ms`, and a size of `561 B`. The response body is in `JSON` format and contains the following user details:

```
1 {
2   "name": "Jenish",
3   "email": "jenishpatel[REDACTED]@gmail.com",
4   "verified": true,
5   "avatar_base64": ""
6 }
```

The screenshot shows the `Test Results` tab for the same request, displaying 3/3 tests passed:

- PASSED** Status code is 200
- PASSED** Response body contains required keys
- PASSED** 'verified' is true

Verdict: PASS

The API returned the correct user information.

➤ TC-07 : Get User Details Without Token (Negative)

Description:

Call /auth/me without sending any token.

Output:

The screenshot displays a REST client interface for a test case. The top bar shows the URL `GET /auth/me` and the method `GET`. The `Headers` tab is active, showing a table with one header: `Authorization: Bearer {{auth_token}}`. The `Body` tab is also visible, showing a JSON response: `{ "detail": "Not authenticated" }`. The `Test Results` tab is active, showing three failed assertions:

Key	Value	Description
<input checked="" type="checkbox"/>	Authorization	Bearer {{auth_token}}
	Key	Value

The `Test Results` tab shows the following failures:

- FAILED** Status code is 200 | AssertionError: expected response to have status code 200 but got 403
- FAILED** Response body contains required keys | AssertionError: expected { detail: 'Not authenticated' } to have property 'name'
- FAILED** 'verified' is true | AssertionError: expected undefined to deeply equal true

Verdict: PASS

The API blocked the request because no token was provided.

➤ TC-08 : Get Gmail State Token (Positive)

Description:

Check if Gmail OAuth state token is created properly.

Output:

The screenshot shows a REST client interface with the following details:

- URL:** `{{base_url}} /gmail/state-token?user_id= {{user_id}}`
- Method:** GET
- Params:** user_id (value: `{{user_id}}`)
- Status:** 200 OK
- Response Body (JSON):**

```
{  "state": "eyJ1c2VyX2lkIjoimTc2MzY0NzU2NS42MDM4NDIiLCJpYXQ1OjE3NjM2NDkyMDMsImV4cCI6MTc2MzY0OTUwM30uXxq0RMosQJspvrvq5fm8haY6C0JISy7FYudoYsdwxwHU"}
```

The screenshot shows the Test Results tab with the following assertions:

- PASSED** Status code is 200
- PASSED** Response body is valid JSON
- PASSED** Response contains 'state' property

Verdict: PASS

The API generated the state token successfully.

➤ TC-09 : Get Gmail Accounts (Positive)

Description:

Fetch all Gmail accounts connected to the user.

Output:

HTTP AegisSecure_API / 02 - Gmail / GET /gmail/accounts

GET {{base_url}} /gmail/accounts

Headers (7)

Key	Value	Description
Authorization	Bearer {{auth_token}}	

Body

```
{
  "accounts": [
    {
      "gmail_email": "jenishpatel@gmail.com"
    }
  ]
}
```

Test Results (2/2)

- PASSED Status code is 200
- PASSED Response has 'accounts' array with at least one object having 'gmail_email'

Verdict: PASS

The API returned the connected Gmail accounts correctly.

➤ TC-10 : Get Gmail Accounts Without Token (Negative)

Description:

Try getting Gmail accounts without sending a token.

Output:

HTTP AegisSecure_API / 02 - Gmail / GET /gmail/accounts

GET {{base_url}} /gmail/accounts

Headers (7)

Key	Value	Description
Authorization	Bearer {{auth_token}}	

Body

401 Unauthorized • 250 ms • 526 B

JSON

```
{
  "detail": "Invalid or expired token"
}
```

Test Results (0/2)

401 Unauthorized • 250 ms • 526 B

Filter Results

- FAILED Status code is 200 | AssertionError: expected response to have status code 200 but got 401
- FAILED Response has 'accounts' array with at least one object having 'gmail_email' | AssertionError: expected { detail: 'Invalid or expired token' }

Verdict: PASS

The API refused the request because the token was missing.

➤ TC-11 : Refresh Gmail Token (Positive)

Description:

Refresh Gmail access token using the refresh endpoint.

Output:

The screenshot displays a REST client interface for a GET request to the endpoint `/auth/gmail/refresh`. The request includes two query parameters: `user_id` and `gmail_email`, both using placeholder values `{{user_id}}` and `{{gmail_email}}`. The response is a 200 OK status with a response time of 649 ms and a body size of 830 B. The response body is shown in JSON format, containing an `access_token` field with a long alphanumeric value.

Query Params

Key	Value	Description
<input checked="" type="checkbox"/> user_id	<code>{{user_id}}</code>	
<input checked="" type="checkbox"/> gmail_email	<code>{{gmail_email}}</code>	

Body | Cookies | Headers (12) | Test Results | 200 OK • 649 ms • 830 B • Save Response

```
{  "access_token": "3kruvykh079J64qIhm7gAiLwYBN7raeIRRIQe2W5nKBa41sT0U9eRFNmFIrULRSuwcLr"}
```

Test Results (2/2) | 200 OK • 1.58 s • 830 B • Save Response

Filter Results

- PASSED** Status code is 200
- PASSED** Response body contains 'access_token'

Verdict: PASS

The token was refreshed successfully.

➤ TC-12 : Gmail Notifications Endpoint (Positive)

Description:

Send a sample request to the Gmail notification endpoint to see if it works.

Output:

The screenshot displays a REST client interface for a test case. The top bar shows the URL `POST /gmail/notifications` and a `Send` button. Below the URL bar, tabs for `Params`, `Authorization`, `Headers (9)`, `Body`, `Scripts`, and `Settings` are visible. The `Query Params` section is empty. The `Body` tab is selected, showing a JSON response: `{ "status": "ignored" }`. The status bar indicates a `200 OK` response with a response time of 284 ms and a body size of 504 B. Below the response, the `Test Results (2/2)` section shows two passed tests: "Status code is 200" and "Response body contains 'status' property with value 'ignored'".

Key	Value	Description
Key	Value	Description

```
{
  "status": "ignored"
}
```

Test Results (2/2)

- PASSED** Status code is 200
- PASSED** Response body contains 'status' property with value 'ignored'

Verdict: PASS

The API accepted the request and responded correctly.

➤ TC-13 : Google OAuth Callback with Invalid or Missing Code (Negative)

Description:

Call the Google OAuth callback endpoint with an incorrect code to check error handling.

Output:

The screenshot displays a REST client interface for a GET request to the endpoint `/auth/google/callback`. The request URL is `{{base_url}} /auth/google/callback?code=testcode&state=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...`. The request is sent, and the response is a `400 Bad Request` with a response time of 603 ms and a size of 737 B. The response body is JSON, showing an error message: `"detail": "Invalid JWT token: Signature has expired."`.

The Test Results section shows two results:

- FAILED** Status code is 200 | AssertionError: expected response to have status code 200 but got 400
- PASSED** Status code is 400 with expired/invalid JWT token message

Verdict: PASS

The API correctly showed an error for invalid code, confirming proper validation during the callback stage.

➤ TC-14 : Fetch Emails From Gmail (Positive)

Description:

Test the endpoint that fetches emails for the connected account.

Output:

[illegible]

Verdict: PASS

The API successfully fetch the user's emails, confirming that email fetching is working correctly.

➤ TC-15 : Fetch Gmail User Profile (Positive)

Description:

Test the Gmail user profile endpoint to check if user information is fetched correctly from the connected Gmail account.

Output:

The screenshot displays a REST client interface for testing the endpoint `GET /user/me`. The URL is `{{base_url}} /user/me?user_id= {{user_id}}`. The request is a GET method. The Headers tab shows 7 headers, including an Authorization header with the value `Bearer {{auth_token}}`. The Body tab shows the response in JSON format:

```
{
  "name": "User",
  "gmail_email": "jenishpatel[REDACTED]@gmail.com"
}
```

The response status is **200 OK** with a response time of 685 ms and a body size of 564 B. The Test Results tab shows three passed assertions:

- PASSED** Status code is 200
- PASSED** Response has 'name' and 'gmail_email' keys
- PASSED** 'gmail_email' is a valid email address

Verdict: PASS

The API returned the Gmail user's profile successfully, confirming proper connection with the Gmail account.

➤ TC-16 : Sync SMS With Valid Data (Positive)

Description:

Sync SMS by sending a valid list of messages.

Output:

The screenshot shows a REST client interface for a POST request to `{{base_url}}/sms/sync`. The request body is a JSON array of two message objects. The response is a JSON object indicating success.

```
1 {
2   "messages": [
3     {
4       "address": "Adi1",
5       "body": "Test1",
6       "date_ms": 1,
7       "type": "A"
8     },
9     {
10      "address": "Adi2",
11      "body": "Test2",
12      "date_ms": 2,
13      "type": "B"
14    }
15  ]
16 }
```

Body Cookies Headers (12) Test Results (4/4) 200 OK • 1.06 s • 543 B Save Response

```
1 {
2   "status": "success",
3   "inserted": 0,
4   "user_id": "691e1...a7564b58c4f3"
5 }
```

The screenshot shows the Test Results section with four passed assertions:

- PASSED** Status code is 200
- PASSED** Response has 'status' property (string)
- PASSED** Response has 'inserted' property (number, >= 0)
- PASSED** Response has 'user_id' property (string)

Verdict: PASS

The SMS messages were synced successfully.

➤ TC-17 : Fetch SMS (Positive)

Description:

Fetch all synced SMS using a valid token.

Output:

HTTP AegisSecure_API / 03 - SMS / GET /sms/all

GET {{base_url}} /sms/all

Send

Docs Params Authorization Headers (7) Body Scripts Settings Cookies

Headers 6 hidden

Key	Value	Description	Bulk Edit	Presets
Authorization	Bearer {{auth_token}}			

Body Cookies Headers (12) Test Results 200 OK 900 ms 2.49 KB Save Response

JSON Preview Visualize

```
1 {
2   "sms_messages": [
3     {
4       "_id": "691f6...3010177ad8727",
5       "user_id": "691e...7a7564b58c4f3",
6       "address": "Adr2",
7       "body": "Test2",
8       "timestamp": 2,
9       "type": "B",
10      "hash": "8781c498...a84f891bc9a70ccae2033f81f6eb43e3f6ffd825d954",
11      "spam_score": 5.0,
12      "spam_reasoning": "The Technical Score is very low (0.01), indicating a likely legitimate me",
13      "spam_highlighted_text": "Test2",
14      "spam_suggestion": "Safe to view.",
15      "spam_verdict": "legitimate",
16      "created_at": "2025-11-20T19:08:47.112000"
17    },
18    {
19      "_id": "691f...d5010177ad8726",
20      "user_id": "691e...37a7564b58c4f3",
21      "address": "Adr1",
22      "body": "Test1",
23      "timestamp": 1,
24      "type": "A",
```

Body Cookies Headers (12) Test Results (2/2) 200 OK 779 ms 2.49 KB Save Response

Filter Results

PASSED Status code is 200

PASSED Response has 'sms_messages' array

Verdict: PASS

The API returned all SMS messages correctly.

➤ TC-18 : Fetch Dashboard with Valid Inputs (Positive)

Description:

Fetch dashboard data using correct mode and day values.

Output:

The screenshot displays a REST client interface for the endpoint `GET /dashboard` with query parameters `mode=both&days=7`. The response is a 200 OK status with a response time of 2.27s and a body size of 768B. The response body is shown in JSON format, containing labels, values, total, and insights.

Query Params

Key	Value	Description
mode	both	
days	7	

Body

```
{
  "labels": [
    "Secure",
    "Suspicious",
    "Threat",
    "Critical"
  ],
  "values": [
    5,
    0,
    0,
    0
  ],
  "total": 5,
  "insights": {
    "fact1": "Use strong, unique passwords for each account and consider a reputable password manager",
    "fact2": "Regularly patch and update software to close security vulnerabilities."
  }
}
```

Test Results (3/3)

- PASSED** Status code is 200
- PASSED** Response body contains expected keys
- PASSED** 'total' matches the sum of 'values' array

Verdict: PASS

The dashboard data was loaded successfully.

➤ TC-19 : Predict Email Content (Positive)

Description:

Test the ML prediction endpoint by sending a valid email (sender, subject, text). This checks whether the model can analyse the email and return prediction details correctly.

Output:

The screenshot displays a REST client interface for testing the `/predict` endpoint. The request is a `POST` with a body containing the following JSON:

```
1 {
2   "sender": "tester3@gmail.com",
3   "subject": "Testing",
4   "text": "API Test is completed",
5   "metadata": {}
6 }
```

The response is a `200 OK` status with a response time of `1.04 s` and a body size of `828 B`. The response body is shown in JSON format:

```
1 {
2   "confidence": 5.0,
3   "reasoning": "The Technical Score is very low (0.01), indicating a low risk. The message content is",
4   "highlighted_text": "API Test is completed",
5   "final_decision": "legitimate",
6   "suggestion": "Safe to view."
7 }
```

Below the response, the Test Results section shows three passed tests:

- PASSED** Status code is 200
- PASSED** Response has all required keys
- PASSED** final_decision is 'legitimate' or 'phishing'

Verdict: PASS

The API successfully processed the email and returned prediction details such as confidence, reasoning, highlighted text, final decision and suggestion. This confirms the prediction model is working properly.

❖ Conclusion

All the API endpoints of the Aegis Secure Backend were tested and everything worked correctly. Both positive and negative test cases passed without any issues. The authentication system handled correct and incorrect login attempts properly. Gmail-related APIs worked as expected, SMS functions synced correctly and the dashboard data loaded without problems.

Overall, the backend is stable and ready to be used. Error handling, validation and authorization are working well, which means the system is secure and the user experience will be smooth.