

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://akshatbhattacharya515334-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 shows the Postman interface for a POST request to `/auth/register`. The request body contains the following JSON:

```
1 {
2   "name": "Tester3",
3   "email": "tester3@gmail.com",
4   "password": "Test@333"
5 }
```

The response is a 200 OK status with a message: "User registered. OTP sent to email."

At the bottom, a green box indicates the test result: "PASSED 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 shows a POST request to `/auth/register` with the following JSON body:

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

The response is a **400 Bad Request** with the message: `"detail": "Email already registered"`.

Test Results (1/2) indicates one **FAILED** test (status code 200 expected 400) and one **PASSED** test (status code 400).

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 shows the Postman interface with the following details:

- Request URL:** {{base_url}}/auth/login
- Method:** POST
- Body Content:**

```
1 {
2   "email": "jenishpatel_____@gmail.com",
3   "password": "_____"
4 }
```
- Response Status:** 200 OK
- Test Results Summary:** 3/3 passed
- Test Results Details:**
 - 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 shows a POST request to `/auth/login` with the following JSON body:

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

The response is a 400 Bad Request with the following JSON content:

```
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 shows a Postman interface with the following details:

- Request Method:** POST
- Request URL:** {{base_url}} /auth/login
- Body Content:**

```
1 {  
2   "email": "",  
3   "password": "xyz"  
4 }
```
- Status:** 400 Bad Request
- Response Body:**

```
1 {  
2   "detail": "User not found"  
3 }
```
- Test Results:** 0/3 failed (Assertion errors)
- Assertion Errors:**
 - 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 the Postman interface with the following details:

- Request URL:** GET /auth/me
- Headers:** Authorization: Bearer {{auth_token}}
- Response Body (JSON):**

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

- Test Results (3/3):**

 - 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 shows a Postman interface with the following details:

- Request URL:** GET /auth/me
- Headers:** Authorization: Bearer {{auth_token}}
- Response Status:** 403 Forbidden
- Response Body (JSON):**

```
1 {  
2   "detail": "Not authenticated"  
3 }
```
- Test Results:**
 - Assertion failed: Status code is 200 | AssertionError: expected response to have status code 200 but got 403
 - Assertion failed: Response body contains required keys | AssertionError: expected { detail: 'Not authenticated' } to have property 'name'
 - Assertion 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 the Postman API client interface. At the top, the URL is set to `HTTP AegisSecure_API / 02 - Gmail / GET /gmail/state-token`. The method is set to `GET`, and the URL is expanded to `{{base_url}} /gmail/state-token?user_id= {{user_id}}`. Below this, the `Params` tab is selected, showing a table with one row for `user_id` with value `{{user_id}}`. Other tabs include `Docs`, `Authorization`, `Headers (6)`, `Body`, `Scripts`, `Settings`, and `Cookies`. In the main body area, the response status is `200 OK` with a duration of `1.38 s` and a size of `689 B`. The response body is displayed as JSON, showing a single key-value pair where the value is a long string of characters representing the state token.

Below the main interface, a detailed view of the test results is shown. It includes three green `PASSED` status indicators: "Status code is 200", "Response body is valid JSON", and "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:

The screenshot shows a Postman request and its response. The request is a GET to `http://{{base_url}}/gmail/accounts`. The Headers tab shows an Authorization header with the value `Bearer {{auth_token}}`. The response status is 200 OK, and the JSON body contains an array of accounts, one of which has a `gmail_email` field. The Test Results section shows two passed assertions: "Status code is 200" and "Response has 'accounts' array with at least one object having 'gmail_email'".

```
{} JSON
1 {
2   "accounts": [
3     {
4       "gmail_email": "jenishpatel[REDACTED]@gmail.com"
5     }
6   ]
7 }
```

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:

The screenshot shows the AegisSecure API tool interface. At the top, it displays the URL: `HTTP AegisSecure_API / 02 - Gmail / GET /gmail/accounts`. Below the URL, there's a search bar with the query `GET {{base_url}} /gmail/accounts` and a "Send" button. The main interface has tabs for "Docs", "Params", "Authorization", "Headers (7)", "Body", "Scripts", "Settings", and "Cookies". The "Headers" tab is selected, showing a table with one row where the "Authorization" header is set to "Bearer {{auth_token}}". The "Body" tab shows a JSON response with a single key-value pair: "detail": "Invalid or expired token". The "Test Results" tab at the bottom shows two failed assertions: "Status code is 200 | AssertionError: expected response to have status code 200 but got 401" and "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 shows the AegisSecure API tool interface. At the top, it displays the URL: `HTTP AegisSecure_API / 02 - Gmail / GET /auth/gmail/refresh`. Below the URL, there's a search bar with the query: `GET {{base_url}} /auth/gmail/refresh?user_id={{user_id}} &gmail_email={{gmail_email}}`. To the right of the search bar are 'Save' and 'Share' buttons. Below the search bar, there are tabs for 'Docs', 'Params' (which is selected), 'Authorization', 'Headers (7)', 'Body', 'Scripts', 'Settings', and 'Cookies'. Under the 'Params' tab, there's a table titled 'Query Params' with three rows: 'user_id' and 'gmail_email' both set to `{{{user_id}}}` and `{{{gmail_email}}}`, respectively, and an empty row for 'Key' and 'Value'. In the main body area, there's a 'Body' tab showing a JSON response with an 'access_token' key. The response body is partially redacted. Below the body, there's a 'Test Results' section with two green 'PASSED' status messages: 'Status code is 200' and 'Response body contains 'access_token''. The test results show a total of 2/2 passed.

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 shows a Postman interface with the following details:

- Method:** POST
- URL:** {{base_url}} /gmail/notifications
- Headers:** (9) (shown in the Headers tab)
- Body:** (shown in the Body tab)
- Test Results:** (2/2) (shown in the Test Results tab)
 - 200 OK (Status code: 200 ms: 284 B: 504)
 - Response body:

```
1 {  
2   "status": "ignored"  
3 }
```
- Test Results Details:**
 - 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 shows the Aegis Secure API tool interface. At the top, it displays the URL: `HTTP AegisSecure_API / 02 - Gmail / GET /auth/google/callback`. Below the URL, there's a "Send" button and a "Share" button. The main area shows a "Params" tab selected, with a table of query parameters:

Key	Value	Description
code	testcode	
state	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...	

Below the table, the "Body" tab is selected, showing a JSON response:

```
1 {  
2   "detail": "Invalid JWT token: Signature has expired."  
3 }
```

At the bottom, the "Test Results" section shows one failed test and one passed test:

- 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:

The screenshot shows the Postman interface with the following details:

- URL:** AegisSecure_API / 02 - Gmail / GET /emails
- Method:** GET
- Query Params:** account: {{gmail_email}}
- Status:** 200 OK
- Body (JSON):**

```
1 [  
2 {  
3     "gmail_email": "jenishpatel_____@gmail.com",  
4     "gmail_id": "1_____j9e9f140d6",  
5     "body": "<!DOCTYPE html>\r\n<html xmlns='http://www.w3.org/1999/xhtml' xmlns:v='urn:schemas-m  
6     \"from\": \"Uber <noreply@uber.com>\",  
7     \"snippet\": \"Read the latest information about our data practices.  
8     \"spam_highlighted_text\": \"\",  
9     \"spam_prediction\": 5.0,  
10    \"spam_reasoning\": \"Legitimate. The Technical Score is 0, indicating a low risk. The sender is 'U  
11    \"spam_suggestion\": \"Safe to view.\",  
12    \"spam_verdict\": \"legitimate\",  
13    \"subject\": \"We've updated our Privacy Notices\",  
14    \"timestamp\": 1763652056000,  
15    \"user_id\": \"691e1a274a37a7564b58c4f3\",  
16    \"processing\": false,  
17    \"char_color\": \"#90A4AE\"  
18 },  
19 {  
20     "gmail_email": "jenishpatel_____@gmail.com",  
21     "gmail_id": "1_____j9dd3ef640",  
22     "body": "<!DOCTYPE html>\r\n<html lang='en' xmlns:o='urn:schemas-microsoft-com:office:office'\br/>23     \"from\": \"The Postman Team <postman-team@email.postman.com>\",
```

- Test Results (3/3):**

 - PASSED Status code is 200
 - PASSED Response body is an array
 - PASSED Each email object contains required properties

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 shows the AegisSecure API tool interface. At the top, it displays the URL `HTTP AegisSecure_API / 02 - Gmail / GET /user/me`. Below the URL, the method is set to `GET` and the path is `{{base_url}} /user/me?user_id={{user_id}}`. The `Send` button is visible. The `Headers (7)` tab is selected, showing one header: `Authorization: Bearer {{auth_token}}`. The `Body` tab shows a JSON response with two keys: `"name": "User"` and `"gmail_email": "jenishpatel@gmail.com"`. The `Test Results (3/3)` tab at the bottom shows three green `PASSED` status messages: `Status code is 200`, `Response has 'name' and 'gmail_email' keys`, and `'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 Postman test result for a POST request to `/sms-sync`. The request body contains the following JSON:

```
1 {
2   "messages": [
3     {
4       "address": "Adr1",
5       "body": "Test1",
6       "date_ms": 1,
7       "type": "A"
8     },
9     {
10       "address": "Adr2",
11       "body": "Test2",
12       "date_ms": 2,
13       "type": "B"
14     }
15   ]
16 }
```

The response status is 200 OK, with a response body containing:

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

The Test Results section shows 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:

The screenshot shows a Postman request for `GET {{base_url}} /sms/all`. The Headers tab shows a `Authorization` header set to `Bearer {{auth_token}}`. The Body tab displays a JSON response with two SMS message objects:

```
1 {
2     "sms_messages": [
3         {
4             "_id": "691f6...5010177ad8727",
5             "user_id": "691e...7a7564b58c4f3",
6             "address": "Adr2",
7             "body": "Test2",
8             "timestamp": 2,
9             "type": "B",
10            "hash": "8781c498...384f891bc9a70ccae2033f81f6eb43e3f6ffd825d954",
11            "spam_score": 5.0,
12            "spam_reasoning": "The Technical Score is very low (0.01), indicating a likely legitimate message.",
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": "691f6...5010177ad8726",
20            "user_id": "691e...37a7564b58c4f3",
21            "address": "Adr1",
22            "body": "Test1",
23            "timestamp": 1,
24            "type": "A"
25        }
26    ]
27}
```

The Test Results section shows two passed assertions:

- PASSED Status code is 200
- PASSED Response has 'sms_messages' array

Verdict: PASS

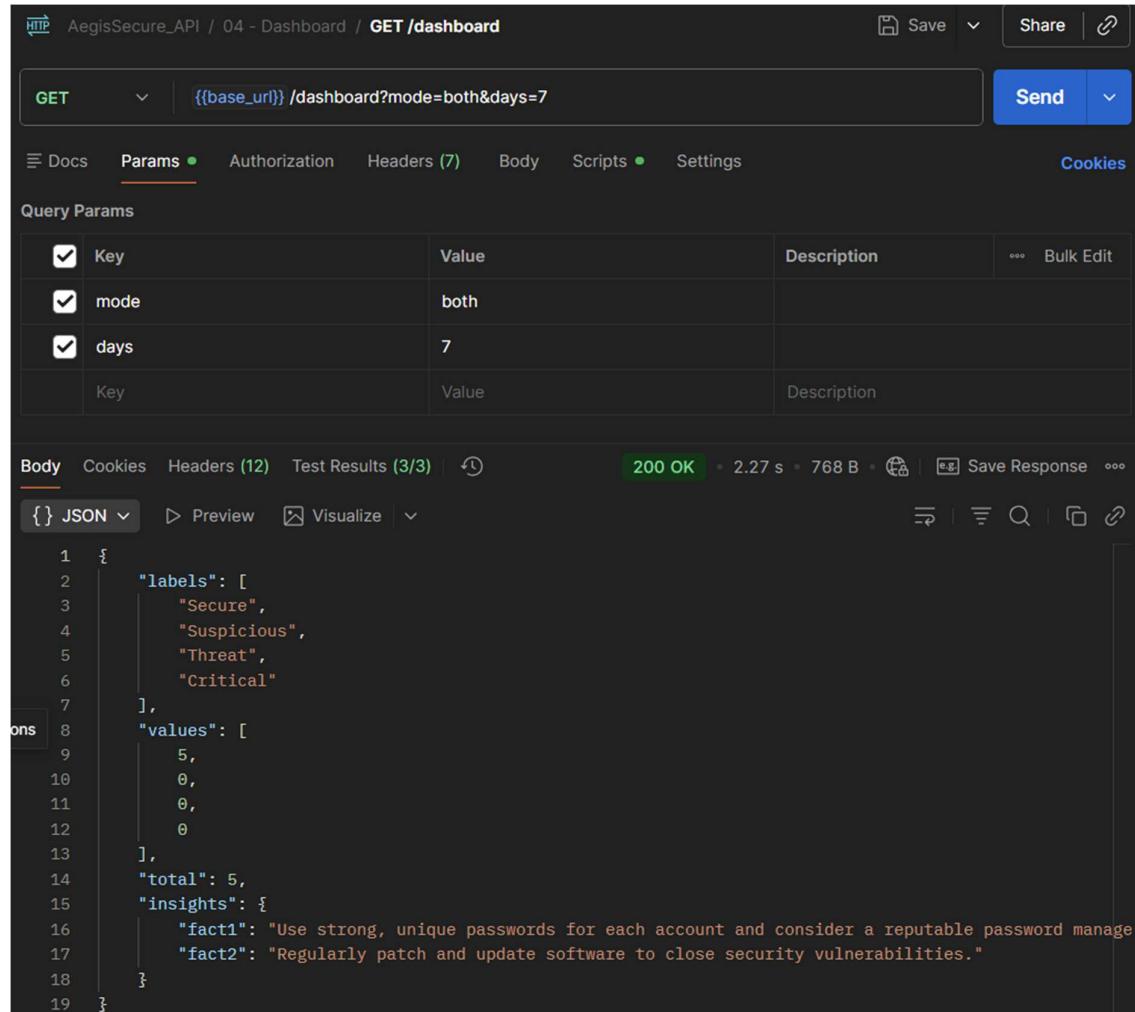
The API returned all SMS messages correctly.

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

Description:

Fetch dashboard data using correct mode and day values.

Output:



The screenshot shows the Postman interface with the following details:

- HTTP Method:** GET
- URL:** {{base_url}} /dashboard?mode=both&days=7
- Headers:** (7)
- Body:** (JSON) { "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:** 200 OK
- Time:** 2.27 s
- Size:** 768 B

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 shows a Postman request to the endpoint `POST {{cyber_url}}/predict`. The request body is a JSON object representing an email message:

```
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 size of 828 B. The response body contains the predicted data:

```
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 }
```

The test results section shows three passed assertions:

- 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.