

Testing Document

Introduction of Testing Tool:

- During the system development process, API testing plays an important role in ensuring that all functions work as expected. The team used **Thunder Client**, an API testing tool integrated into Visual Studio Code, to send HTTP requests (GET, POST, DELETE, etc.) and verify the system responses.
- Thunder Client provides a simple and effective way to test APIs by displaying status codes and response data clearly, helping the team evaluate the correctness of the implemented features.

Test case:

Test Case ID	Function	Test Steps	Input Data	Expected Result
TC01	Register	1. Go to Register page 2. Enter valid information 3. Click Register	Full Name: Nguyen Van A Email: test@gmail.com Password: 123	User is registered successfully and a default AddressBook is created
TC02	Register	1. Go to Register page 2. Enter an email that already exists 3. Click Register	Full Name: Nguyen Van A Email: test@gmail.com Password: 123	Error message " Email already exists " is displayed
TC03	Login	1. Go to Login page 2. Enter valid username and password 3. Click Login	Username: student01 Password: 123456	User logs in successfully
TC04	Login	1. Go to Login page 2. Enter valid username and invalid password 3. Click Login	Username: student01 Password: wrongpass	Error message " Invalid username or password " is displayed
TC05	Add Contact	1. Login to the system 2. Go to Add Contact page 3. Enter valid contact information 4. Click Submit	Name: Nguyễn Văn A Email: vana@gmail.com Phone: 0912345678 Group IDs: [1,2]	New contact is added successfully and appears in the contact list
TC06	Edit Contact	1. Login to the system 2. Go to Contact List 3. Select a contact and click Edit 4. Update contact information	Name: An Phone: 0987654321 Email: an@gmail.com	Contact information is updated successfully

Test Case ID	Function	Test Steps	Input Data	Expected Result
		5. Click Save		
TC07	View Contact List	1. Login to the system 2. Navigate to Contact List page	AddressBook ID: 1	System returns a list of contacts successfully
TC08	View Contact List	1. Login to the system 2. Navigate to Contact List page	User account without any contacts	System displays an empty list or message " No contacts found "
TC09	Add Contact	1. Login to the system 2. Send POST request /api/books/2/contacts	Valid contact data	System returns error message: <i>AddressBook not found</i>
TC10	Add Contact to Group	1. Login to the system 2. Go to Add Contact page 3. Enter contact information 4. Select a group belonging to the same address book 5. Click Submit	Name: Nguyen Van A Email: vana@gmail.com Group: Friends	Contact is added successfully and assigned to the selected group
TC11	Add Contact to Group	1. Login to the system 2. Go to Add Contact page 3. Enter contact information 4. Select a group belonging to another address book 5. Click Submit	Name: David Group: Work (from another address book)	System displays an error message and contact is not added to the group

Test Case ID	Function	Test Steps	Input Data	Expected Result
TC12	Delete Contact	1. Login to the system 2. Send DELETE request /api/books/1/groups/1	Group ID: 01	Contact is deleted successfully
TC13		1. Login to the system 2. Send DELETE request /api/books/1/groups/1	Group ID: 1	

Evidence of using Thunder Client to test the code:

The screenshot shows the Thunder Client application interface. At the top, it displays the URL `C:\Users\ASUS\Documents\CNPM\ProgAndTest_Group12\models.py • 1 problem in this file`. Below the URL, there is a dropdown menu set to `POST` and the target URL `http://localhost:5000/api/register`. To the right of the URL is a blue `Send` button.

Below the URL bar, there is a navigation bar with tabs: `Query`, `Headers 2`, `Auth`, `Body 1` (which is currently selected), `Tests`, and `Pre Run`.

Under the `Body` tab, there is a sub-navigation bar with options: `JSON` (which is selected and highlighted in blue), `XML`, `Text`, `Form`, `Form-encode`, `GraphQL`, and `Binary`. Below this, the `JSON Content` section contains the following JSON data:

```

1  {
2    "fullname": "Nguyen Van A",
3    "email": "test4@gmail.com",
4    "password": "123",
5    "gender": "Male",
6    "address": "Hanoi",
7    "phoneNumber": "0987654321",
8    "date_of_birth": "1995-01-01"
9  }

```

To the right of the JSON content, there is a `Format` button. Below the JSON content, the status of the request is shown as `Status: 201 CREATED Size: 64 Bytes Time: 156 ms`.

At the bottom of the interface, there is another navigation bar with tabs: `Response` (which is selected and highlighted in blue), `Headers 5`, `Cookies`, `Results`, and `Docs`. Below this, the `Response` section displays the following JSON data:

```

1  {
2    "message": "User registered successfully",
3    "user_id": 2
4  }

```

- The `/api/register` API works as expected
- Data is successfully sent and stored in the database

- The system automatically generates a user_id for the new user

The screenshot shows the Postman interface. At the top, it says "POST" and "http://127.0.0.1:5000/api/login". Below that, there are tabs for "Query", "Headers 2", "Auth", "Body 1", "Tests", and "Pre Run". The "Body 1" tab is selected and has "JSON" selected. The JSON content is:

```

1  {
2    "email": "test4@gmail.com",
3    "password": "123"
4  }

```

At the bottom, it shows "Status: 200 OK", "Size: 114 Bytes", and "Time: 294 ms". The "Response" tab is selected, showing the JSON response:

```

1  {
2    "address_books": [
3      2
4    ],
5    "fullname": "Nguyen Van A",
6    "message": "Login successful",
7    "user_id": 2
8  }

```

- Valid login information
- The system correctly verifies the email and password
- Returns user information and the associated AddressBook list
- The Login function works as expected\

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <http://127.0.0.1:5000/api/books/1/contacts>
- Body Tab:** Selected
- JSON Content:**

```
1 {  
2   "name": "Trần Thị B",  
3   "email": "thib@gmail.com",  
4   "phone": "0988777666",  
5   "address": "456 Đường XYZ, Hà Nội",  
6   "notes": "Đồng nghiệp",  
7   "group_ids": []  
8 }
```
- Status:** 201 CREATED
- Size:** 65 Bytes
- Time:** 20 ms
- Response Tab:** Selected
- Response Body:**

```
1 {  
2   "contact_id": 1,  
3   "message": "Contact added successfully"  
4 }
```

- API works correctly
- Contact is added successfully
- Data is stored in the database
- The system generates a contact_id automatically

The screenshot shows a Postman request to `http://127.0.0.1:5000/api/books/1/contacts`. The **Body** tab is selected, containing JSON data:

```
1  {
2    "name": "Nguyen Van A",
3    "email": "vana@gmail.com",
4    "phone": "0123456",
5    "group_ids": [1]
6 }
```

The response status is **201 CREATED**, size is **65 Bytes**, and time is **31 ms**. The response body is:

```
1  {
2    "contact_id": 1,
3    "message": "Contact added successfully"
4 }
```

- Contact is added successfully to the specified group
- Contact information is processed correctly
- The system returns status **201 CREATED**
- A **contact_id** is automatically generated

The screenshot shows a Postman request to `http://127.0.0.1:5000/api/addressbook/1/groups`. The **Body** tab is selected, containing JSON data:

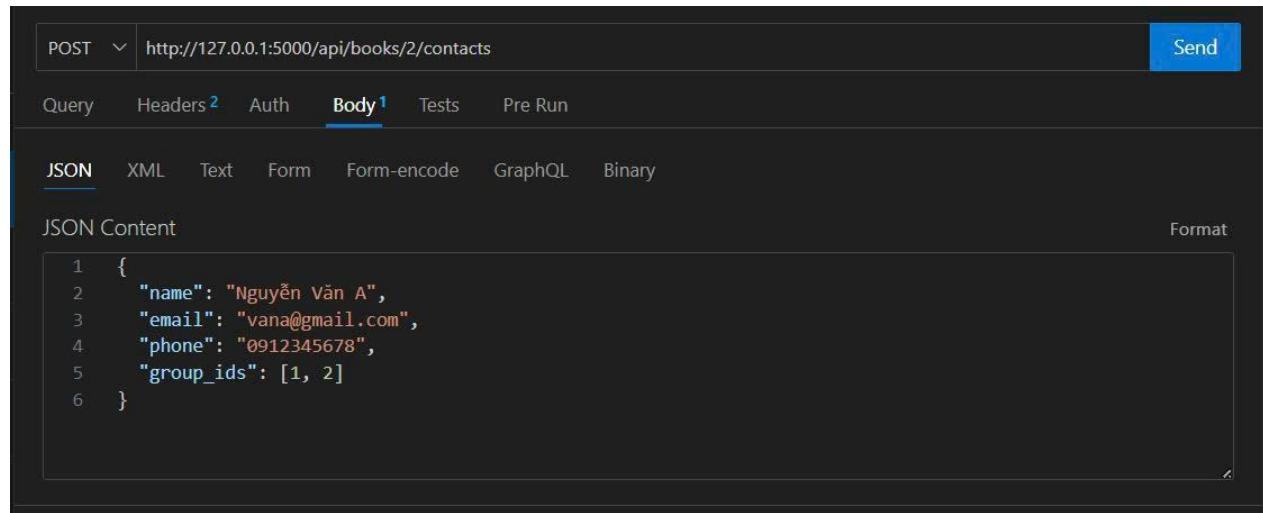
```
1  {
2    "group_name": "Bạn Đại Học"
3 }
```

The response status is **201 CREATED**, size is **63 Bytes**, and time is **19 ms**. The response body is:

```
1  {
2    "group_id": 1,
3    "message": "Group created successfully"
4 }
```

- Group is created successfully

- **Data is saved to the database**
- **The system returns status 201 CREATED**
- **A group_id is automatically generated**



POST http://127.0.0.1:5000/api/books/2/contacts

Query Headers ² Auth Body ¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {  
2   "name": "Nguyễn Văn A",  
3   "email": "vana@gmail.com",  
4   "phone": "0912345678",  
5   "group_ids": [1, 2]  
6 }
```

Status: 404 NOT FOUND Size: 41 Bytes Time: 9 ms

Response Headers ⁵ Cookies Results Docs {} ⌂

```
1 {  
2   "message": "AddressBook not found"  
3 }
```

- **The request failed because the AddressBook does not exist**
- **The system returns status 404 NOT FOUND**
- **Contact cannot be added to a non-existing AddressBook**

```
1  {
2      "name": "Nguyễn Văn A",
3      "email": "vana@gmail.com",
4      "phone": "0912345678",
5      "group_ids": [1, 2]
6  }
```

Status: 201 CREATED Size: 65 Bytes Time: 29 ms

Response Headers 5 Cookies Results Docs

```
1  {}
2      "contact_id": 1,
3      "message": "Contact added successfully"
4  {}
```

- Contact information is valid
- The system successfully adds the contact to the AddressBook
- The API returns status 201 CREATED with a new contact_id

The screenshot shows the Postman application interface. At the top, there is a header bar with a 'DELETE' button, a dropdown menu, the URL 'http://127.0.0.1:5000/api/books/1/groups/1', and a 'Send' button. Below the header, there are tabs for 'Query', 'Headers 2', 'Auth', 'Body', 'Tests', and 'Pre Run'. The 'Body' tab is selected and has a sub-tab 'JSON' which is also selected. A large text area labeled 'JSON Content' contains the number '1'. To the right of this area is a 'Format' button. Below the main interface, there is a status summary: 'Status: 200 OK', 'Size: 48 Bytes', and 'Time: 21 ms'. Underneath this, the 'Response' tab is selected, showing a JSON response with three numbered lines. Line 1 starts with '1 {', line 2 with '2 "message": "Group 1 deleted successfully"', and line 3 with '3 }'. There are also tabs for 'Headers 5', 'Cookies', 'Results', and 'Docs'.

```
1 {
2   "message": "Group 1 deleted successfully"
3 }
```

- The system successfully deletes the selected group
- The API returns status 200 OK
- Group data is removed from the database

The screenshot shows the Thunder Client interface. At the top, there's a header with 'GET' dropdown, URL 'http://127.0.0.1:5000/api/books/1/contacts', and a 'Send' button. Below the header, tabs include 'Query', 'Headers 2', 'Auth', 'Body' (which is selected), 'Tests', and 'Pre Run'. Under 'Body', sub-tabs are 'JSON', 'XML', 'Text', 'Form' (which is selected), 'Form-encode', 'GraphQL', and 'Binary'. A 'Form Fields' section has a table with one row: 'field name' (checkbox) and 'value' (checkbox). Below this, a note says 'Learn more about how to set a custom [content-type](#) for a field.' The status bar at the bottom shows 'Status: 200 OK', 'Size: 187 Bytes', and 'Time: 8 ms'. The 'Response' tab is selected, displaying a JSON array of two contacts:

```

1  [
2    {
3      "contact_id": 1,
4      "groups": [],
5      "name": "Trần Thị B"
6    },
7    {
8      "contact_id": 2,
9      "groups": [
10        "djdj"
11      ],
12      "name": "Trần Thị Q"
13    }
  
```

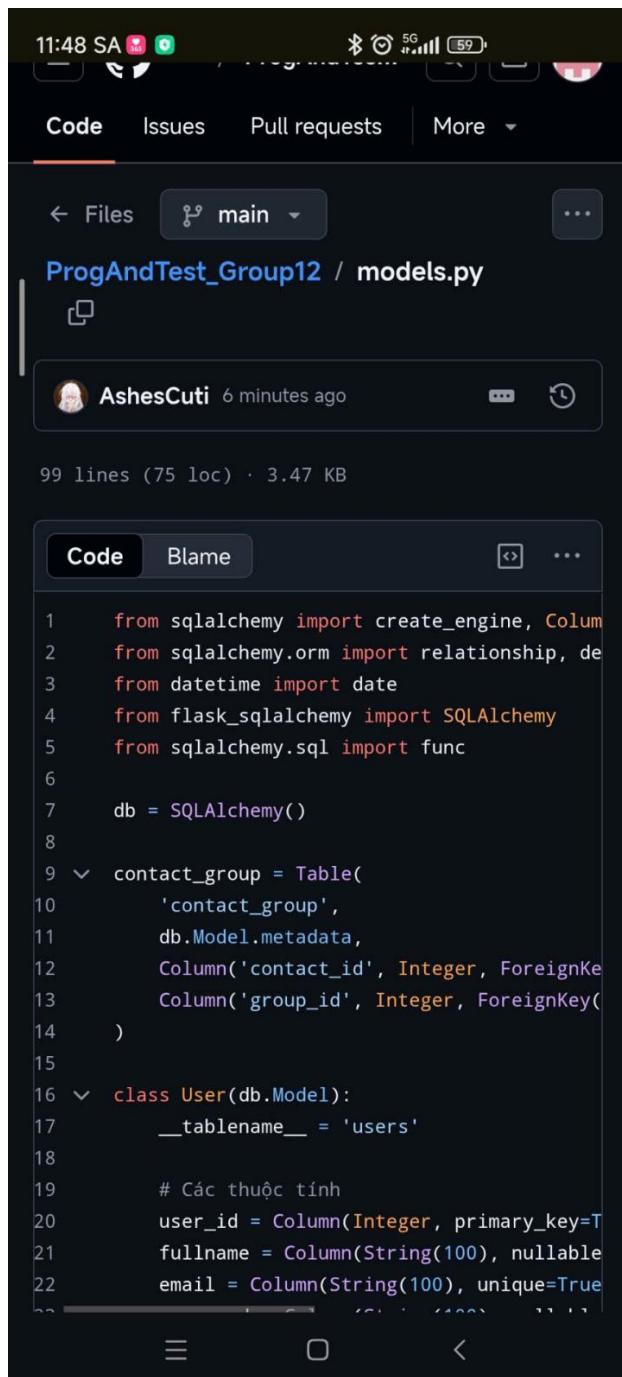
- GET request to retrieve the contact list using Thunder Client, returning a successful 200 OK response with contact data.

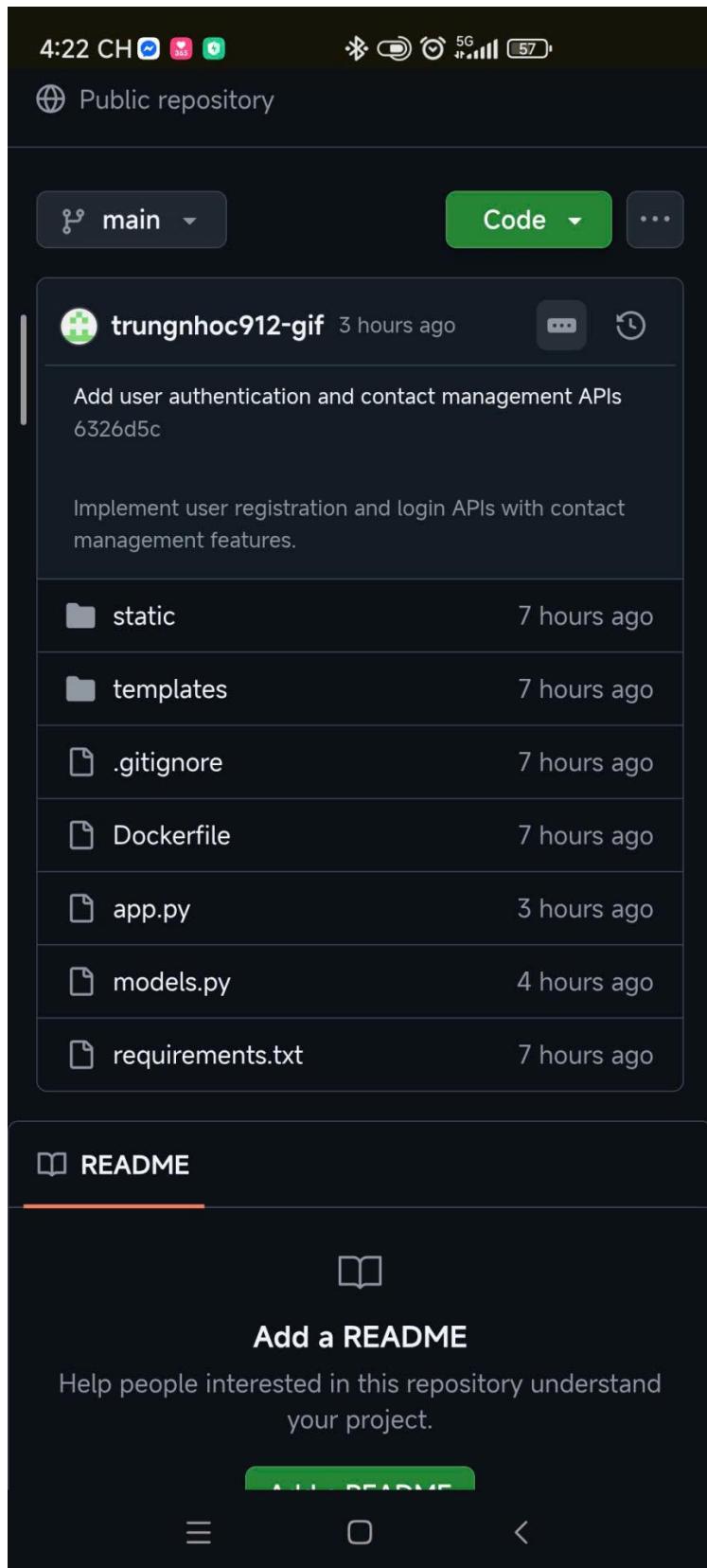
Evidence of using Git for teamwork

The screenshot shows a GitHub commit history for a repository. The commits are as follows:

Name	Last commit message	Last commit date
.gitignore	Update and rename Bin.gitignore to .gitignore	4 minutes ago
Dockerfile	Create Dockerfile	1 minute ago
requirements.txt	Create requirements.txt	1 minute ago

<input type="checkbox"/> Create a new account	Create a new account	Open in new window
<input type="checkbox"/> Add a new account	Add a new account	Open in new window
<input type="checkbox"/> Import accounts	Import accounts	Open in new window
<input type="checkbox"/> Sync accounts	Sync accounts	Open in new window
<input checked="" type="checkbox"/> Import from CSV	Import from CSV	Open in new window





 **ProgAndTest_Group12** Public

[main](#) [1 Branch](#) [0 Tags](#) [Go to file](#) [t](#) [Add file](#) [Code](#)

 **JiluieLatifah** Update models.py f9fb85a · 2 hours ago 20 Commits

 static	Add files via upload	yesterday
 templates	Add files via upload	yesterday
 .gitignore	Update .gitignore	2 hours ago
 Dockerfile	Create Dockerfile	3 days ago
 app.py	Update app.py	2 hours ago
 models.py	Update models.py	2 hours ago
 requirements.txt	Create requirements.txt	3 days ago

 **README**



Add a README

Help people interested in this repository understand your project.

