

API and API Testing

♦ What is an API?

API stands for **Application Programming Interface**.

It is a set of rules and protocols that allows different software applications to **communicate** with each other.

Example:

Imagine you're ordering food online:

- The app (like Zomato or Swiggy) sends your order to the restaurant's server.
- The server processes the request and sends back a response — order confirmation, ETA, etc.
- The **API** is the **messenger** that carries your order from the app to the restaurant and brings back the response.

♦ What is API Testing?

API Testing is a type of software testing that focuses on verifying that **APIs** work correctly, efficiently, and securely.

It involves sending requests to the API and checking the **response** for:

- Correctness (data, format, status code)
- Performance (speed, reliability)
- Security (authentication, access control)
- Error handling (bad inputs, server failures)

What Do You Test in API Testing?

Test Type	Description
✓ Functionality	Does the API return the correct data?
✓ Status Codes	Are HTTP status codes like 200, 400, 401, 500 used correctly?
✓ Response Time	Is the API fast enough?
✓ Error Handling	How does the API behave for invalid inputs?
✓ Data Integrity	Is the returned data accurate and complete?
✓ Security	Are proper authentication and authorization in place?
✓ Schema Validation	Does the response match the expected format (e.g., JSON schema)?

Tools for API Testing:

Tool	Description
Postman	GUI-based tool for manual API testing. Very popular.
cURL	Command-line tool to make API calls.
SoapUI	Used for testing SOAP and REST APIs.
JMeter	Performance testing for APIs.
REST Assured	Java library for automated REST API testing.
Newman	Command-line runner for Postman collections (used in CI/CD).

Common API Testing Methods:

Method	Purpose
GET	Retrieve data
POST	Create new data
PUT	Update existing data
DELETE	Remove data
PATCH	Partial update

Benefits of API Testing:

- Fast and efficient (no UI dependency)
- Early detection of bugs
- Easy automation and integration in CI/CD
- Helps test business logic independently


Would you like a hands-on demo using Postman or want to learn how to automate API testing using Python or Java?

Why API Testing is Important

API testing is **crucial** in modern software development for several key reasons:

1. APIs Are the Backbone of Modern Apps


APIs connect frontend (UI) with backend services and databases. If APIs fail, the entire application may crash or misbehave — even if the UI looks perfect.

 **Example:** A banking app may load fine, but if the balance-check API fails, the app becomes useless.

2. UI Changes, APIs Stay Stable

APIs are independent of the UI. Testing APIs ensures:

- Core functionality is verified before UI is built.
- Faster, more stable test coverage during frequent UI changes.

 This reduces test maintenance and increases productivity.

3. Faster than UI Testing

- API tests run much faster than end-to-end UI tests.
- Ideal for automation and CI/CD pipelines.

 UI tests might take minutes; API tests can finish in seconds.

4. Early Bug Detection

APIs can be tested **as soon as backend development starts**, even before the frontend is ready.

 Finding bugs early = less cost + faster delivery.

5. Improves Test Coverage

API testing lets you:

- Test edge cases (invalid inputs, large data, etc.)
 - Validate all response types (success, errors, timeouts)
 - Check all endpoints, parameters, headers, and methods
-

6. Ensures Data Integrity

API testing ensures that:

- The correct data is returned from the server
- No extra/missing/incorrect fields are sent
- JSON or XML responses follow proper schema

 Crucial for apps that depend on accurate data like finance, healthcare, e-commerce.

7. Boosts Security Testing

APIs are often public or exposed over the internet. Testing helps ensure:

- Authentication and Authorization are enforced
- Sensitive data is encrypted or masked
- No unauthorized access is possible

 Especially important for APIs using tokens, keys, or OAuth.

8. Enables CI/CD and Automation

API tests can be easily automated using tools like:

- **Postman + Newman**
- **REST Assured (Java)**
- **Pytest + Requests (Python)**

They integrate well into CI/CD tools like Jenkins, GitHub Actions, GitLab CI, etc.

9. Supports Microservices Architecture

Modern systems use **microservices**, each with its own APIs. Testing them individually ensures:

- Each service works independently
- Integration between services is smooth and error-free

10. Better Overall Product Quality

By catching bugs early, verifying business logic, and securing data, API testing helps deliver a **reliable, fast, and secure product**.

Would you like a **real-life example** or mini-project to practice API testing hands-on (manual or automated)?

Great question!

How API Differs from UI

Feature	API (Application Programming Interface)	UI (User Interface)
Definition	A set of protocols for communication between software systems	Visual layer for user interaction with software
Users	Used by developers, testers, and applications	Used by end users (humans)
Purpose	Data exchange, automation, backend logic	Interaction, usability, and user experience
Access Method	Accessed via code or tools like Postman, curl, scripts	Accessed via browser , desktop, or mobile apps
Example	GET /users/123 returns user data in JSON	A webpage showing the user's profile
Testing Focus	Functionality, response, performance, security	Layout, responsiveness, user experience, visuals
Speed	Fast — No rendering needed	Slower — UI rendering adds overhead
Stability	More stable — fewer changes over time	Often changes frequently (new layout, elements)
Error Visibility	Errors returned as codes/messages (e.g., 404, 500)	Errors shown visually (e.g., popups, alerts)
Automation	Easily automated using tools like Postman, REST Assured, or Pytest	Automated using Selenium, Cypress, Playwright
Dependency	Independent of UI	UI depends on APIs to get data from the server

Real-Life Example:

♦ API:

A weather API like **GET /weather?city=Hyderabad** returns:

```
json
{
  "temperature": "32°C",
  "condition": "Sunny"
}
```

♦ UI:

A weather app displays:

```
makefile
City: Hyderabad
Temp: 32°C * Sunny
```

Summary

 **API = Brain (Logic + Data)**

 **UI = Face (Design + Interaction)**

- API testing focuses on **business logic**, **data flow**, and **system communication**.
 - UI testing focuses on **look**, **feel**, and **user actions**.
-

Would you like a visual diagram or hands-on comparison using tools like Postman (for API) vs browser/Selenium (for UI)?

Where Does API Fit in the SDLC (Software Development Life Cycle)?

APIs play a **central role** throughout the SDLC — not just during development, but across **design**, **testing**, **deployment**, and **maintenance**.

SDLC Phases with API Involvement:

SDLC Phase	Role of API
1. Requirement Analysis	Define what functionality the API must support (e.g., login, fetch user data).
2. Design	Design API endpoints, request/response formats (JSON/XML), methods (GET, POST), error codes, security mechanisms (auth, tokens).
3. Development	Backend developers build the API logic. Frontend developers may use mock APIs or API documentation (like Swagger) for integration.
4. Testing	Perform API Testing (functionality, performance, security). APIs can be tested before UI is even ready.
5. Deployment	APIs are deployed to dev/staging/production environments, often as part of CI/CD pipelines .
6. Maintenance	APIs are updated with versioning (v1 , v2 , etc.). Bugs are fixed. Monitoring and logging are added (e.g., API gateways, tools like AWS API Gateway or Kong).

API's Key Roles in Each Team:

Team	How They Use the API
Frontend Devs	Consume APIs to show data on the UI
Backend Devs	Build and expose APIs for internal or external use
Testers	Perform API testing before and during UI testing
DevOps	Deploy and monitor APIs as part of CI/CD pipelines
Product/BA	Use API contracts and specs for feature planning

Why API Is So Central in SDLC:

- APIs **decouple** frontend and backend teams: both can work in parallel.
- APIs support **automation** (testing, CI/CD).
- APIs promote **reusability** (mobile app, web app, third-party integrations).
- APIs ensure **scalability** (microservices, cloud-native apps).

Quick Visual (Text-based)

CSS

```

[Requirements]
  ↓
[API Specification/Design]
  ↓

```

```
[API Development] ↔ [Frontend Dev]
    ↓
[API Testing]
    ↓
[Deployment with CI/CD]
    ↓
[Monitoring + Maintenance]
```

Would you like a **hands-on SDLC use case** showing where and how API testing fits with Postman or automation?

Absolutely! Let's walk through a **real-time example of API usage** in a **Banking Application** for two common features:

1. Login Functionality (API-Based)

Purpose:

Authenticate the user and return a **token** to authorize future actions.

API Request:

Method: POST /api/v1/login

Headers:

```
http
Content-Type: application/json
```

Body:

```
json
{
  "username": "john_doe",
  "password": "securePass123"
}
```

API Response:

```
json
{
  "status": "success",
  "token": "eyJhbGciOiJIUzI1NiIsInR... ",
  "user_id": "12345"
}
```

What Happens:

- The server verifies credentials.
- If correct, it returns a **JWT token** (used in headers for next calls).
- If wrong, it returns a **401 Unauthorized** or **403 Forbidden**.

2. Fund Transfer (API-Based)

Purpose:

Transfer money between two accounts securely.

API Request:

Method: POST /api/v1/transfer

Headers:

```
http

Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...
```

Body:

```
json

{
  "from_account": "9876543210",
  "to_account": "1234567890",
  "amount": 5000.00,
  "currency": "INR",
  "remarks": "Rent payment"
}
```

API Response:

```
json

{
  "status": "success",
  "transaction_id": "TXN789123456",
  "balance": 45000.00,
  "message": "₹5000 transferred successfully to 1234567890"
}
```

Validations Tested in API Testing:

Check	Description
Authentication	Token is valid and active
Authorization	User has permission to transfer funds
Balance Check	Sufficient funds before transfer
Amount Validation	Amount is valid (non-negative, max limit)
Transaction Record	Logged in database, retrievable by ID
Error Handling	What if account doesn't exist?
Response Time	Is the transaction fast enough (< 2 sec)?

Error Case Example

If the balance is too low:

```
json

{
  "status": "failed",
```

```
"error": "Insufficient funds",
"code": 402
}
```

Chaining Requests in Postman:

You can use Postman to:

1. **Login API** → Capture token
2. **Use token in header** for next **Fund Transfer API**

Summary:

Feature	Login	Fund Transfer
Method	POST	POST
Auth Required	❌ (initial step)	✅ (requires token)
Request Type	JSON	JSON
Testing Focus	Credential validation, token auth	Auth, balance check, transaction logic

Would you like a ready-made **Postman collection** or a **Python automation script** for this banking API testing workflow?