0. Topics

Monday, June 9, 2025 9:46 PM

1. **Importance of Testing APIs**

2. **Types of Tests**

3. **Mock ML Models**

4. **Common API Errors**

5. **Debugging Techniques**

1. Importance of Testing APIs

Monday, June 9, 2025 9:53 PM

- Testing is a non-negotiable aspect of modern API development
- It ensures that the web application functions correctly, is reliable, and behaves consistently even as you scale the applicat ion

1. Correctness of Application Logic:

- Verifies that the API behaves as expected under different conditions
- Reduces bugs in production and catches logical errors early
- Ensures ML models make correct predictions and sensitive endpoints do not expose data due to logic errors

2. Protection Against Regressions:

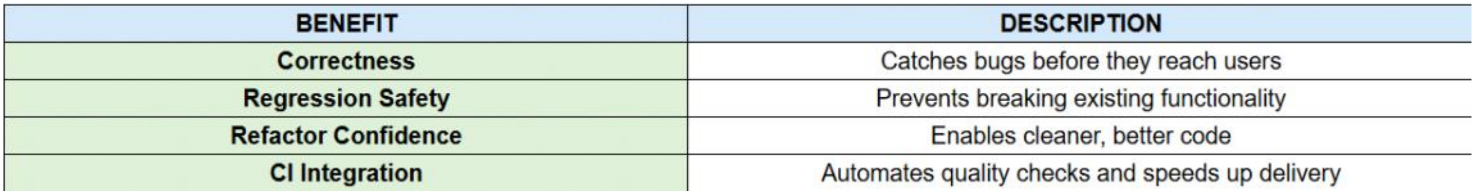
- **Regressions** refer to the bugs introduced into previously working code when new changes are made
- Helps maintain backward compatibility and stability of endpoints
- Use regression test suites to validate critical user flows like authentication, payments, data submissions, etc.

3. Safety Net During Refactoring:

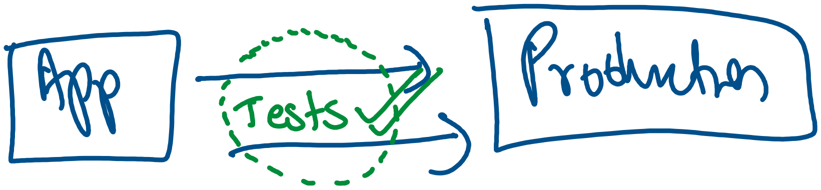
- Helps improve code readability, performance, and maintainability
- Automated test cases verify that the refactored code still behaves the same way as before
- Run the test suite after every major code change or cleanup

4. Enables Continuous Integration Pipelines:

- Tests act as a gatekeeper to production; code won't be merged unless all tests pass
- Ensures stable deployments, boosts team confidence in merging PRs and reduces manual testing effort



BENEFIT	DESCRIPTION
Correctness	Catches bugs before they reach users
Regression Safety	Prevents breaking existing functionality
Refactor Confidence	Enables cleaner, better code
CI Integration	Automates quality checks and speeds up delivery



2. Types of Tests

Monday, June 9, 2025 10:08 PM

- Testing at different layers is essential for ensuring both small units and the system as a whole work correctly
- Each type of test has a distinct purpose, scope, and cost to run

1. Unit Tests:

- Focuses on a single unit of logic—typically a **function** or **method**
- Should not depend on external services like a database, file system, or network
- Fast to run and ideal for **TDD (Test-Driven Development)**
- **When to use:**
  - Testing utility functions
  - Business logic (e.g., tax calculation, discount rules)
  - Schema validation (e.g., Pydantic model validators)
- **How pytest works:**
  - Use the command **pytest tests/**
  - **Pytest** scans the **tests/** folder for any python files that start with **test_** or end with **_test.py**
  - Inside those files, it looks for **functions** that start with **test_**
  - After all tests are run, pytest summarizes the results

2. Integration Tests:

- Involves multiple units working together: **database + app**, **API + ML model**, etc.
- Typically uses in-memory or test database environments
- More realistic than **Unit Tests**, but also slower
- **When to use:**
  - Testing API endpoints
  - Testing FastAPI dependencies (like database or ML models)
  - Validating middleware behavior
  - Ensuring JWT authentication works across endpoints

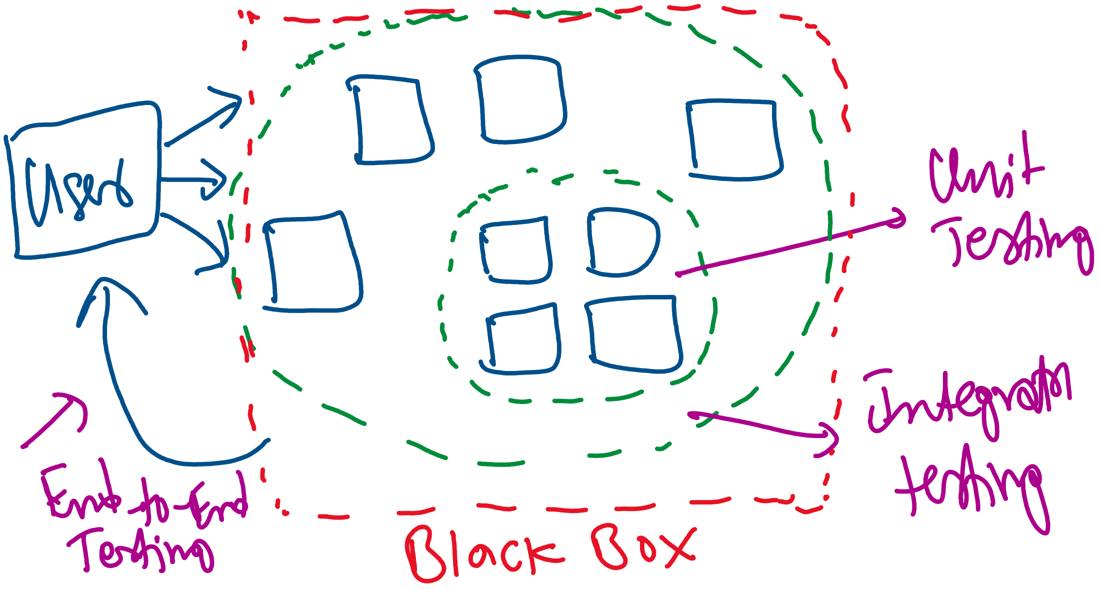
3. End-to-End (E2E) Tests:

- Simulates real user flows like login, submitting a form, or making a prediction
- Runs the system as a black box (i.e., without knowing internal code)
- May interact with UI, frontend, or API
- **When to use:**
  - Pre-deployment checks
  - Validating critical user workflows like sign-up/login/purchase
  - Testing deployment setup and network configurations

Summary:

TYPES OF TEST	SCOPE	SPEED	EXTERNAL DEPENDENCIES	USE CASES
Unit Test	Individual function/class	Fast	None	Business logic, validators
Integration	Multiple modules/components	Medium	DB, ML model, APIs	Endpoint tests, DB interactions, auth logic
End-to-End	Entire system	Slow	Full environment	Login + create order + checkout (user journey)

Unit Test	Individual function/class	Fast	None	Business logic, validators
Integration	Multiple modules/components	Medium	DB, ML model, APIs	Endpoint tests, DB interactions, auth logic
End-to-End	Entire system	Slow	Full environment	Login + create order + checkout (user journey)



3. Mock ML Models

Monday, June 9, 2025 11:14 PM

What is meant by Mocking ML Models?

- Mocking an ML model means replacing the actual ML model with a fake (or simulated) object in your tests that behaves like the real model but doesn't perform any actual computation
- Instead of loading a large model file and calling its predict() method, you pretend (mock) that a model exists and will return a known, controlled value

Why Mock ML Models?

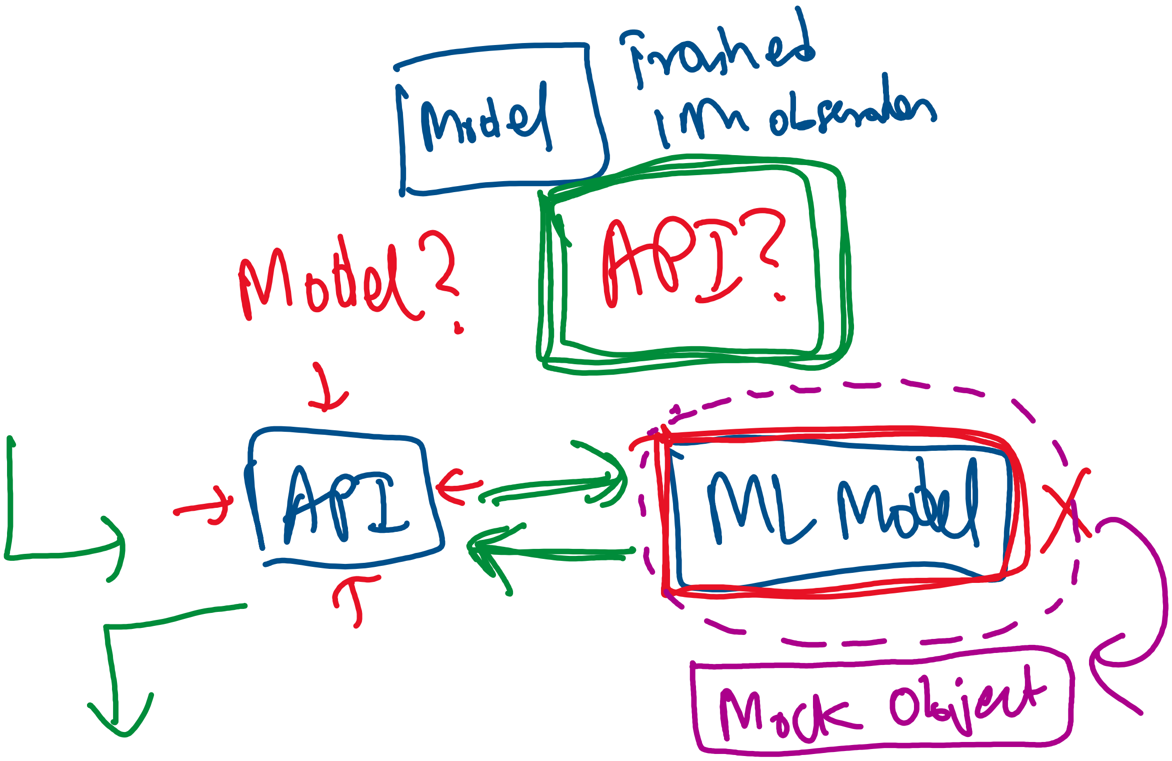
1. **ML Models Are Heavy to Load:**
    - In production-grade ML APIs, models may be trained on large datasets and saved as serialized **.pkl**, **.joblib**, or **.h5** files
    - These files can be hundreds of MBs or even GBs in size and may contain complex architectures (e.g., deep learning models)
    - Deserializing and initializing these models can take several seconds or even minutes
    - Running this load operation for every test case makes the test slow, resource-intensive, and prone to timeouts or memory exhaustion—**not ideal for a CI/CD pipeline**

2. **Tests Should Focus on API Logic:**
    - When testing an API endpoint (e.g., **/predict**), the goal is to check if the endpoint accepts input and returns a response in the correct format
    - We do not care about whether the prediction itself is accurate
    - Hence, using a real model introduces unnecessary complexity and violates unit testing principles

3. **Mocking Makes Tests Fast and Deterministic:**
    - Mocking replaces the actual model object with a fake object that simulates the **predict** method
    - This fake object can return predefined outputs for known inputs, simulate exceptions to test error handling and void real computation, making the test lightweight and fast
    - As a result, test becomes deterministic; the same input always yields the same response, which helps maintain stability in test results

Summary of Mocking in ML API Testing:

BENEFIT	DESCRIPTION
Faster Execution	No time spent loading model files or running computation-heavy predictions
Better Isolation	Focus on testing API routes and business logic, not ML internals
Error Simulation	Easily test how your app handles errors like model.predict() throwing an exception
CI/CD Friendly	Lightweight tests that can run on every commit in seconds
Deterministic Results	Avoid randomness introduced by some models (e.g., unseeded probabilistic models)



3.1 - Demo

Monday, June 9, 2025 11:37 PM

File Structure:

- **main.py**
- **model.py**
- **test_main.py**
- **training.ipynb**

Control Flow:

- Uses the **patch()** method from **unittest.mock** to mock the real ML model’s **.predict()** method
- The mocked method always returns **[99]**
- Sends a **POST** request to the **/predict** endpoint using **TestClient**
- Asserts that the endpoint returns a **200 OK** with a prediction of **99**
- Ensures that the mocked **.predict()** was called with the expected input

4. Common API Errors

Tuesday, June 10, 2025 10:13 PM

1. 401 Unauthorized:

- **Meaning:** Authentication has failed — the server didn’t get a valid token or credentials
- **Common Causes:**
    - Missing or expired authentication token
    - Wrong API key or credentials
    - Bearer token not included or formatted incorrectly
- **Debugging Tips:**
    - Confirm if the token is valid and not expired
    - Ensure the **Authorization** header is set properly
    - Double-check API key/token permissions

2. 404 Not Found:

- **Meaning:** The URL or resource requested does not exist on the server
- **Common Causes:**
    - Typo in endpoint URL
    - The route isn't defined on the backend
    - Missing trailing slash (for some frameworks like Django REST Framework)
- **Debugging Tips:**
    - Recheck the API path spelling and method (GET/POST/etc.)
    - Confirm if the endpoint exists in the backend routing logic
    - Use API documentation or tools like **Swagger**/**OpenAPI** to test routes

3. 422 Unprocessable Entity:

- **Meaning:** The server understands the request, but it can't process the data because it doesn't match the expected format or required fields are missing

- **Common Causes:**
  - Missing required fields in the request body
  - Wrong data types (e.g., sending a string instead of an integer)
  - Failing validation checks (e.g., string too short, email not valid)

- **Debugging Tips:**
  - Check the API documentation/schema carefully
  - Validate the payload locally using a schema validator (ex. Pydantic)
  - Use tools like **Postman** or **curl** to test with valid input

4. 500 Internal Server Error:

- **Meaning:** Something went wrong on the server side while processing the request

- **Common Causes:**
  - Unhandled exception in backend code (e.g., division by zero, missing ML model)
  - Database connection issues
  - Misconfiguration in server code or environment

- **Debugging Tips:**
  - Check server logs for traceback or error stack
  - Add proper exception handling (try-except blocks)
  - Ensure all dependencies and models are loaded before request handling
  - Use **logging** to catch unexpected exceptions

5. Debugging Techniques

Tuesday, June 10, 2025 10:30 PM

FastAPI makes Debugging easier through:

- **Structured Logging**
- **Exception Handling**
- **API testing tools like Postman/curl**
- **Development Mode Configurations**

5.1 - Logging

Tuesday, June 10, 2025 10:35 PM

- Logging is a fundamental debugging practice in development
- Instead of using **print()**, we use Python's **logging** module, which is more powerful and configurable
- Logging helps with:
  - Track what part of the app was accessed
  - Log variables and errors
  - Retain logs in files or forward them to monitoring systems

Log Levels:

- **DEBUG:** Low-level system information
- **INFO:** Routine information like successful requests
- **WARNING:** Something unexpected happened
- **ERROR:** A serious problem
- **CRITICAL:** Severe errors that cause premature termination

5.2 - Exception Handling

Tuesday, June 10, 2025 10:35 PM

FastAPI lets developers define custom exception handlers to:

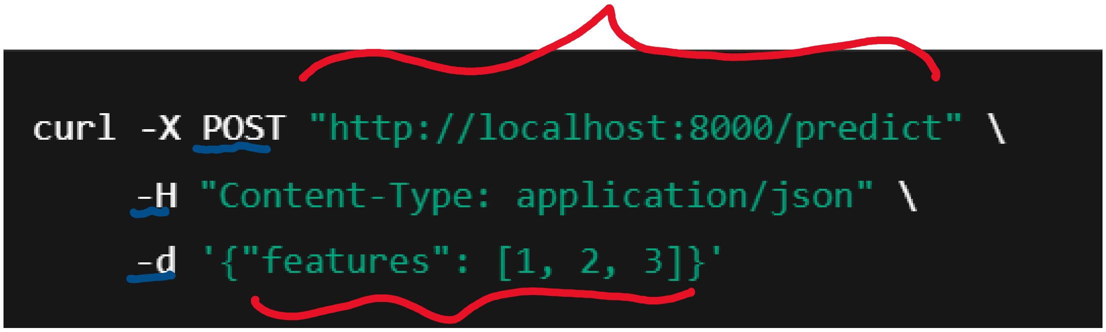
- **Catch unhandled errors**
- **Return consistent and informative error messages**
- **Log errors for debugging and auditing**

5.3 - curl

Tuesday, June 10, 2025 10:35 PM

- Send Requests with headers, payload, methods (GET, POST, PUT, DELETE)
- Inspect responses (status codes, headers, content)
- Helpful in isolating whether issues are in the **backend or client**

Syntax:



```
curl -X POST "http://localhost:8000/predict" \
     -H "Content-Type: application/json" \
     -d '{"features": [1, 2, 3]}'
```

- **-X POST:** Specifies the HTTP method
- **-H:** Adds headers
- **-d:** Sends request body data

5.4 - Configurations

Tuesday, June 10, 2025 10:35 PM

Running **Uvicorn** with **--reload** and **--debug** enables live reloading and better error visibility during development:

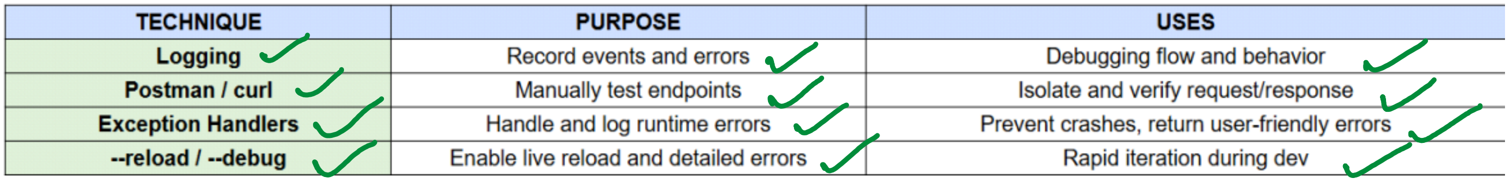
- **--reload:** Automatically restarts the server when you change code (development only)
- **--debug:** Enables verbose output and stack traces in logs (not for production)

Syntax:

```
uvicorn main:app --reload --debug
```

5.5 - Summary

Tuesday, June 10, 2025 11:16 PM



TECHNIQUE	PURPOSE	USES
Logging ✓	Record events and errors ✓	Debugging flow and behavior ✓
Postman / curl ✓	Manually test endpoints ✓	Isolate and verify request/response ✓
Exception Handlers ✓	Handle and log runtime errors ✓	Prevent crashes, return user-friendly errors ✓
--reload / --debug ✓	Enable live reload and detailed errors ✓	Rapid iteration during dev ✓