



Full Stack Application Development

BITS Pilani



Module 9: API Testing

Agenda



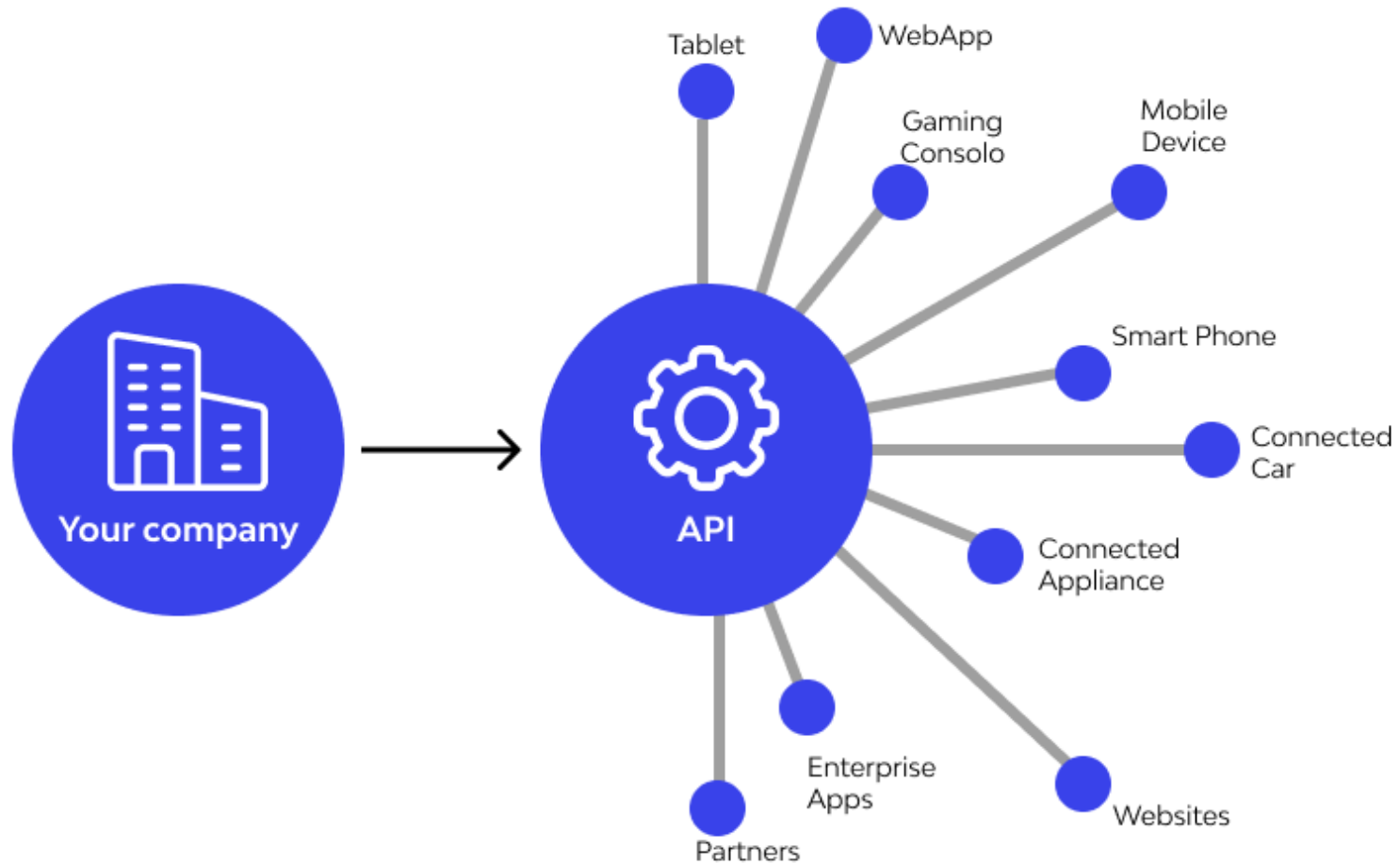
- ☐ API Testing and Types
- ☐ Unit Testing
- ☐ Cross Browser Testing
- ☐ Automated Testing

API Testing



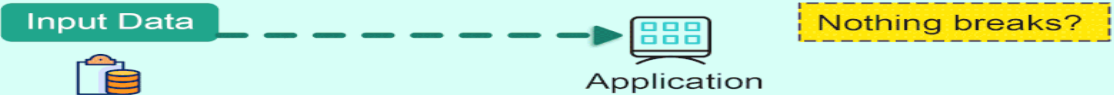






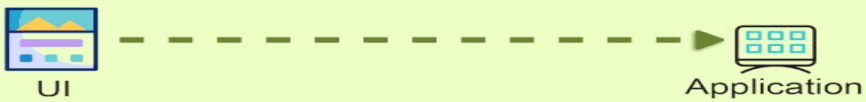

API testing, or application programming interface testing, is a type of software testing that evaluates the functionality, reliability, performance, and security of an API. APIs are the connectors that allow different software systems to communicate and exchange data with each other. For example, the weather app on your phone uses APIs to communicate with the weather bureau's software system and show you daily weather updates

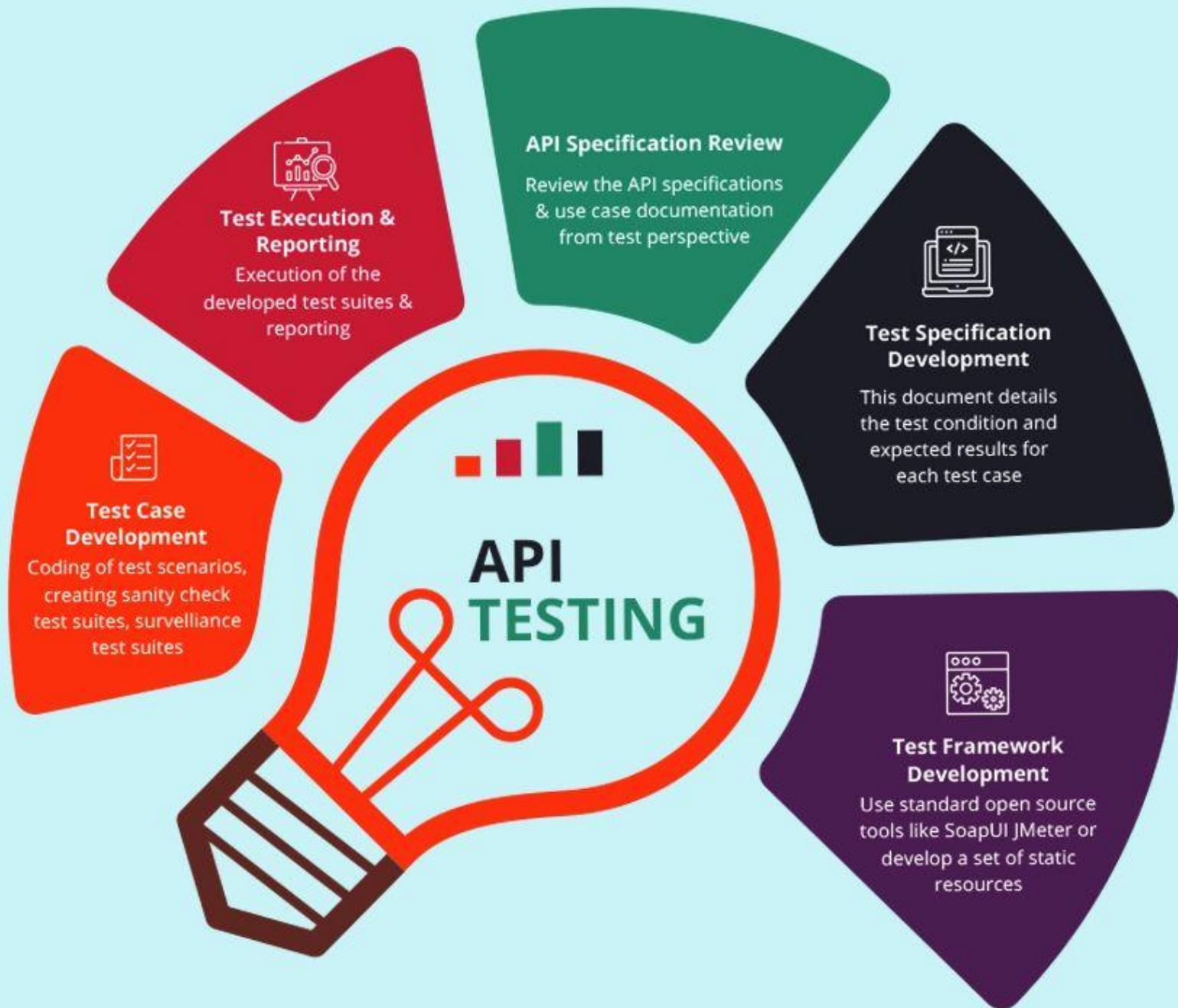
API Varieties



9 Types API Testing

 blog.bytebytego.com

| | | |
|----------------------------|--|--|
| Smoke Testing |  | Simply validate if the APIs are working |
| Functional Testing |  | Test against functional requirements |
| Integration Testing |  | Test several API calls |
| Regression Testing |  | Changes won't break the existing behaviors of APIs |
| Load Testing |  | Test for application's capacity by simulating loads |
| Stress Testing |  | Deliberately create high loads to see if APIs function normally |
| Security Testing |  | Test against all possible external threats |
| UI Testing |  | Test interactions between the UI and the APIs |
| Fuzz Testing |  | Identify vulnerabilities by sending unexpected data into the APIs |



Examples



| | |
|---------------------|--|
| Unit Testing | <ul style="list-style-type: none">• Testing an API's "login" function to authenticate user credentials. |
| Functional Testing | <ul style="list-style-type: none">• Testing an e-commerce API to ensure proper shopping cart functionality. |
| Performance Testing | <ul style="list-style-type: none">• testing an API to measure its speed and responsiveness under different loads. |
| Security Testing | <ul style="list-style-type: none">• Testing an API to validate authentication and encryption methods for data protection |

Examples (cont.)



Integration
Testing

- testing an API that integrates with a payment gateway for accurate payment processing.

Load Testing

- testing an API's performance under high-user traffic.

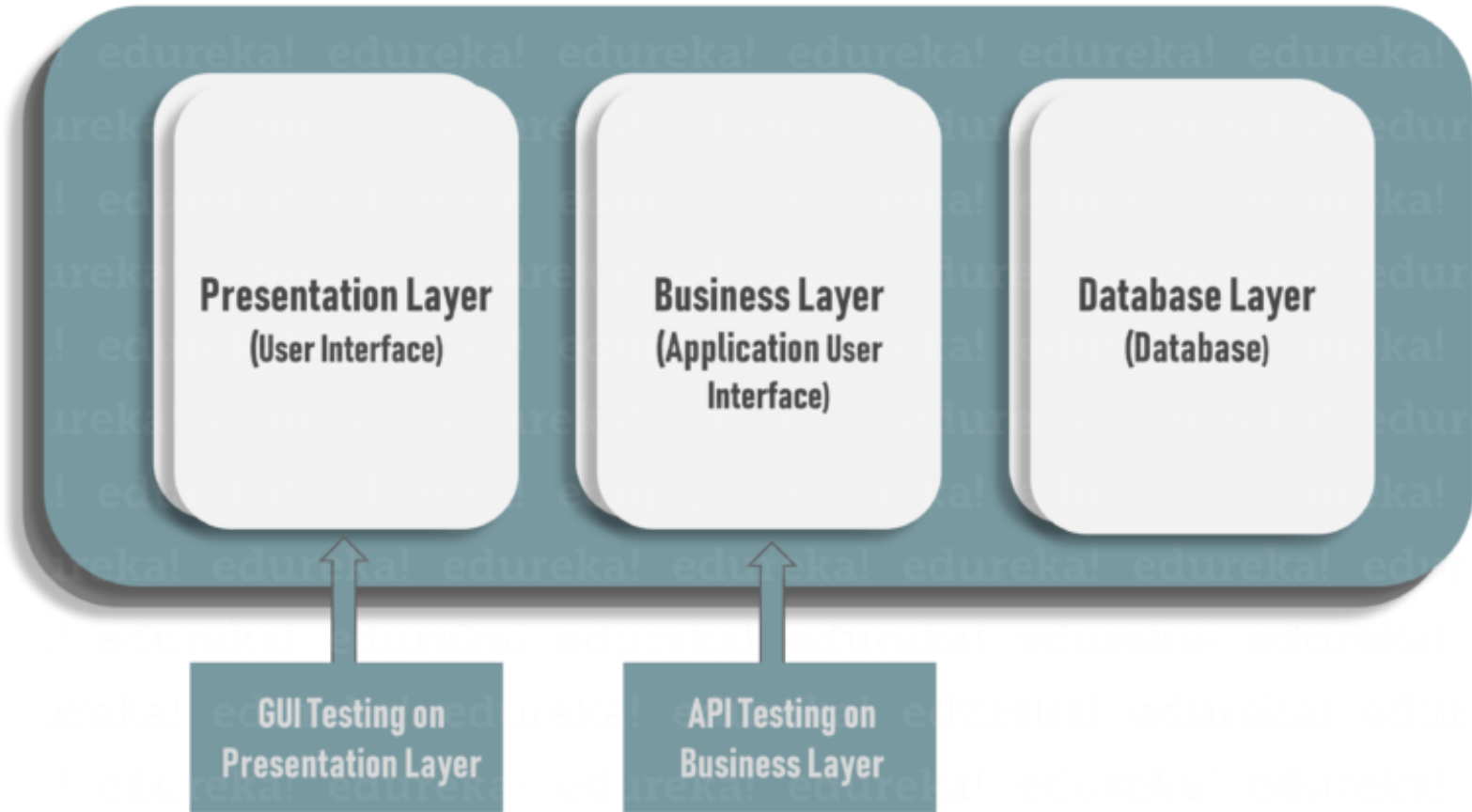
Stress Testing

- testing an API with large data or sudden spikes in user requests.

Fuzz Testing

- Trying different combinations of keys on a keyboard to find unexpected errors (e.g., sending random input to an API's search function to check error handling)

GUI testing



API Testing



API testing can help developers identify errors earlier in the development lifecycle, which can save money because errors can be more efficiently resolved when caught early. The API testing process involves:

1. Sending a request with input data
2. Receiving a response with output data
3. Verifying that the response matches the requirement

Your tests should analyze the following response:

- Reply time
- Data quality
- Confirmation of authorization
- HTTP status code and
- Error codes

API Test Design



1. Test Design Techniques

- **Equivalence Partitioning:** Imagine grouping similar test scenarios together, like sorting clothes into different piles based on their colors.
- **Boundary Value Analysis:** Picture testing the boundaries of valid inputs, such as checking a temperature sensor that works correctly at freezing and boiling points.
- **State Transition Testing:** Think of testing how an API behaves as it moves between different states, just like a traffic light changing colors.

API Test Design



2. Test Coverage & Prioritization

- Make sure to cover all the important parts of the API, like testing different features or functionalities. It's like making sure you explore all the exciting areas of a new city you're visiting.
- Prioritize the most critical test scenarios first, focusing on what can have the biggest impact on the application and its users. It's like dealing with the most urgent things on your to-do list before moving on to less important tasks.

API Test Design



3. Creating Effective Test Cases

- Create test cases that are easy to understand and follow, like writing clear instructions for a game you want to play with your friends.
- Consider positive and negative scenarios, like thinking about different possibilities and outcomes in a fun puzzle or riddle.
- Set up the conditions and environment for your tests, like preparing the stage and props for a play to ensure everything is ready for the actor

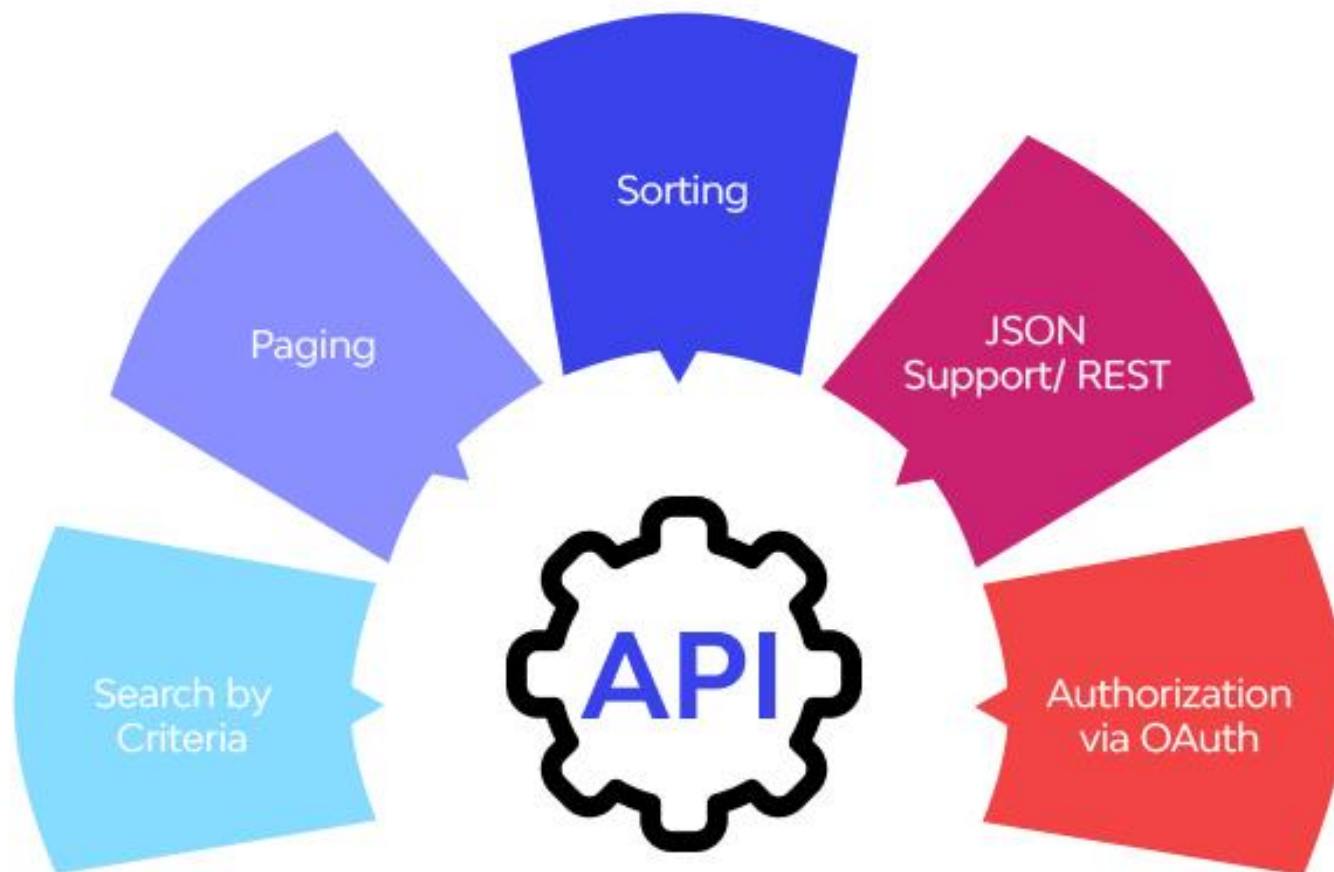
API Test Design



4. Writing Test Code

- Choose a programming language like Python and use libraries such as “requests” to interact with APIs. It’s like having a secret code language to communicate with the API.
- You can write code that sends requests to the API and checks the responses. It’s like playing a game where you make moves and check if you got the desired results.

Features of API



Google Map API Testing?



1. First, you understand your start and end points and any stops you want to make (similar to understanding the API and planning tests).
2. Then, you ensure your device has a good internet connection and GPS signal (setting up the environment).
3. You then follow the route Google Maps provides (executing the tests).
4. Finally, you check if you've arrived at your destination correctly (analyzing the results).
5. If you took a wrong turn or encountered a roadblock, you would correct your course and continue (addressing issues and retesting).

API Documentation Testing



This type of testing is used to verify that the API is easy to use and understand and it is performing as mentioned in the API documentation. It ensures that the documentation accurately reflects the APIs capabilities and that all of the features are being properly integrated into the product. With API document testing, testers make sure that the API returns the right data, that the parameters and values are properly set, and that the returned data is in the expected format. This is similar to the functional requirement document/specification document validation against the application behavior.

Question



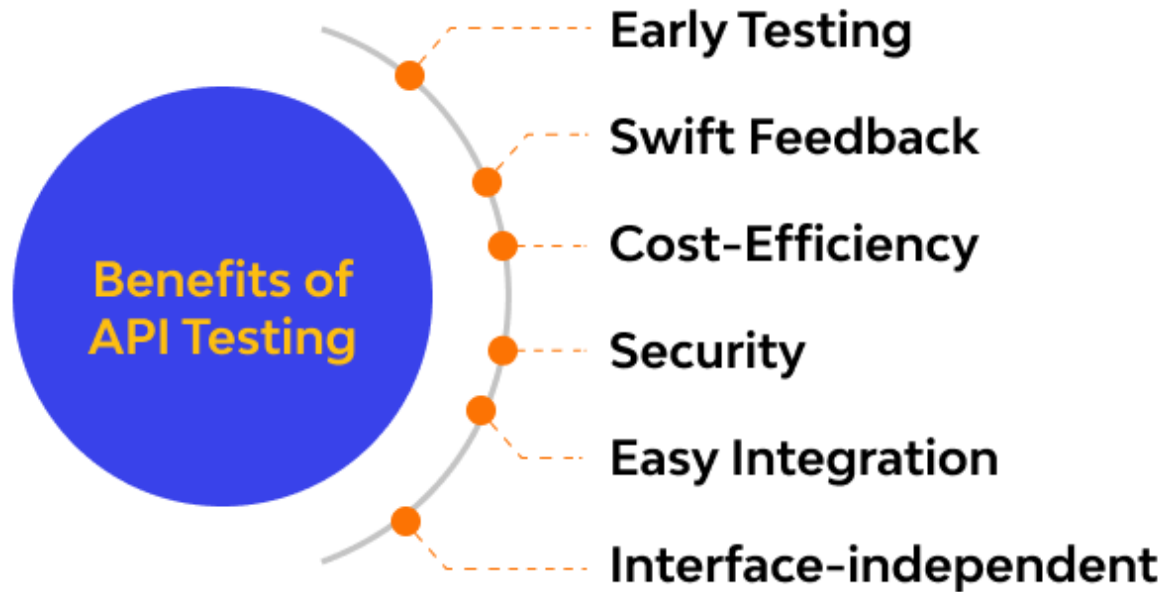
What kind of bugs API Testing will detect?

Types of Bugs

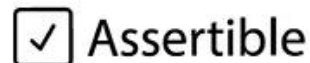


1. **Duplicate or missing functionality:** API testing ensures that the software doesn't have the same feature repeated or any missing features.
2. **Improper messaging:** checks if the messages sent between different software parts are correct and complete.
3. **Error handling problems:** looks for issues with how errors are handled in the software. It ensures errors are dealt with properly and don't cause any unexpected troubles.
4. **Multi-threaded issues:** identify problems that can happen when different parts of the software are trying to do things simultaneously. It checks if the software can handle such situations without conflicts or errors.
5. **Security vulnerabilities:** find security problems in the software, like weaknesses that could allow unauthorized access or attacks.
6. **Performance issues:** checks how well the software performs. It looks for things like slow response times or excessive use of computer resources.
7. **Reliability problems:** helps ensure the software is stable and reliable. It looks for issues like crashes or memory leaks that can cause the software to stop working unexpectedly.

Benefits



Top 15 API Testing Tools



References



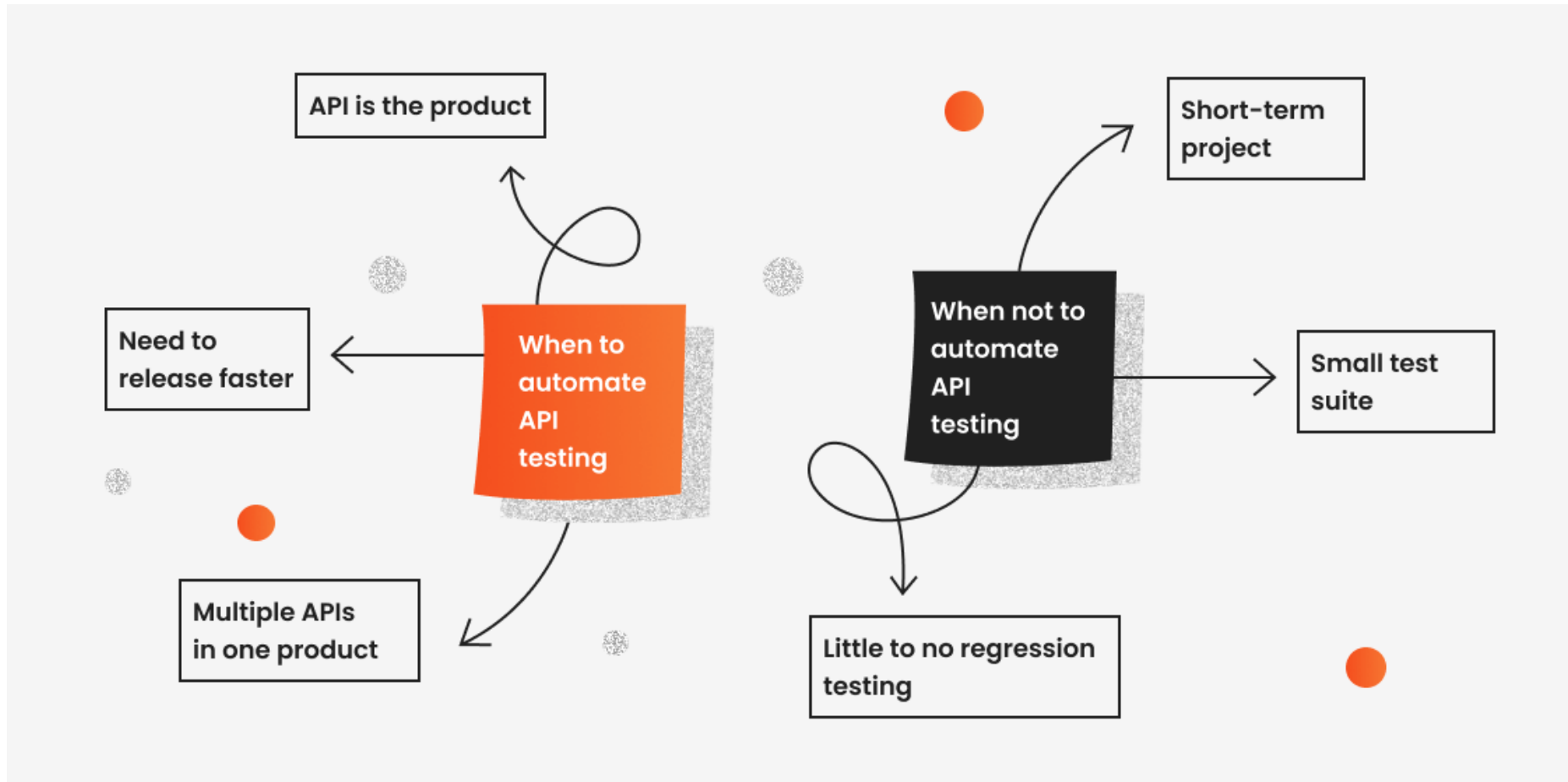
<https://katalon.com/katalon-api-testing>

<https://katalon.com/resources-center/blog/api-testing-tips>

<https://katalon.com/resources-center/blog/cross-browser-testing>

<https://www.soapui.org/learn/functional-testing/api-testing-101/>

API Automation Testing



API Testing



Pros and cons of API testing



Testing is fast

Identifies errors early in development

Technology and language independent

Removes vulnerabilities and guards applications from malicious code

Can be integrated with GUI tests

Parameter sequencing can be difficult

No GUIs available to test the application

Testers need to know how to code

Might test well independently, but not together

Not including API dependencies can lead to overall software issues

1. Difference between API testing and unit testing?

API testing differs from unit testing as it evaluates the holistic performance, reliability, functionality, and security of APIs, while unit testing focuses on validating individual code units within an application.

2. What is the best way to test API?

To ensure comprehensive API testing, it is best to combine manual testing, where requests are manually sent and responses inspected, with automated testing using frameworks like Nightwatch.js.

3. Which methodology is used for web API testing?

The widely adopted methodology for testing web APIs is REST (Representational State Transfer), which emphasizes standard HTTP methods and stateless data exchange in formats like JSON or XML.

4. Why is API used?

APIs facilitate seamless communication and data exchange between software systems or components. They provide a standardized interface that enables integration, promotes interoperability, and empowers third-party developers to extend functionality.



BITS Pilani
Pilani Campus



Testing API Design

Room Booking



CREATE ROOM

/room/

| HTTP method | Description |
|-------------|----------------------------|
| POST | Creates a room for booking |

Request arguments

| | |
|-------------|--|
| roomNumber | The number of the room. |
| type | The type of room. Can be single, twin, double, family, or suite. |
| accessible | Sets whether the room has disability access. |
| description | Sets a description for the room. |
| image | Sets a URL of an image for the room. |
| roomPrice | Sets the price of the room. |
| features | Preset features a room can have including Wi-Fi, TV, safe, radio, refreshments or views. |

What's Wrong?

API Request seems correct



POST /room/ HTTP/1.1
Host: automationintesting.online
Accept: application/json
Cookie: token=r76BXGVy8rlASuZB

```
{  
  "roomNumber":100,  
  "type":"Single",  
  "accessible":false,  
  "description":"Please enter a description for this room",  
  "image":"https://www.mwtestconsultancy.co.uk/img/room1.jpg",  
  "roomPrice":"100",  
  "features":["WiFi"]  
}
```

What's wrong?



The room identifier must always be a **String** because some users use nonintegers to identify rooms.

Dev team missed a series of questions that we could ask: **What if someone sent an integer?** How should we handle that? If the team had allowed time and space for someone to ask that question, it might have resulted in a decision to send a 400 Bad Request HTTP response with additional information to let the client know that a String was required.

A further discussion might have identified that, in fact, **roomNumber is a misleading parameter name** and that renaming it to **roomName would make things clearer**. These additional discussions and questions could have been the difference in whether the person assessing our platform decided to use it or not.

Request corrected

POST /room/ HTTP/1.1
example.com
Accept: application/json
Cookie: token=abc123

```
{  
  "roomName": "102",  
  "type": "Double",  
  "accessible": "true",  
  "description": "This is a description for the room",  
  "image": "/img/room1.jpg",  
  "roomPrice": 200,  
  ["TV", "Safe"]  
}
```

Tools for Questioning?



One common technique is “five Ws and an H”—a phrase that helps us to remember the following six keywords that can inspire different types of questions:

- 1. Who**
- 2. What**
- 3. Where**
- 4. When**
- 5. Why**
- How**

For example:

- 1. What happens when the user isn't authorized?
- 2. What will we do if an API we're dependent on isn't available?
- 3. What information will we send?

Who



Who is going to use this? We would ask this to get a better sense of who or what is going to use the `/room/` endpoint we're discussing. We may discover that another API or a UI library will be consuming the response, or it might be used by an individual. What we learn might inform other questions we ask. For example, I may ask more technical or architecture-based questions if it's another API that is using `/room/`.

Who should have access? We can also consider risks around security during this activity. In the API design, there is mention of a Cookie header with a token, and we might want to learn more about the security controls around this token. Although this is a surface-level question to ask around API security (we'll explore security testing in more depth in chapter 9), answers from this question might allow us to dig deeper into how we're securing our APIs and explore risks that might result in vulnerabilities.

What



- *What if we send incorrect details? What happens?* These questions are reminiscent of the initial example we explored in this chapter. Not all hypothetical questions result in a positive result, and we need to be mindful of how we handle the negative ones, too. At a minimum, we might want to discuss how to minimize the damage that can be done to our systems. But given that we're working with APIs, we might also want to discuss how we provide feedback for errors in a clear, readable manner.
- *What format will the payload be in, and why?* Again, here we're focusing on learning more about potential technical decisions that have been made. On its own, "What format will the payload be in?" could result in a simple response—JSON. But by adding a why in at the end, we're asking for a deeper explanation. We may discover that there are assumptions being made about who might use the response.

Where



- *Where will the room be saved?* A question like this can help reveal implementation details and where they might exist. This might trigger discussions around how the implementation choices we are going to make might fit into architectural conditions we need to meet. For example, if we were working in a microservice architecture, would this work mean we needed to create a new API?
- *Where will the room be used?* A slightly different question from the “Who will use it?” question earlier, this question is attempting to understand how the design choices we make fit into the wider scope of a platform or user features. Learning where something will be used could help reveal new users or APIs that will rely on our design.

When



- *When we make changes to the API contract, will we version the URI?* When we're thinking about how we're going to solve a particular problem, it's important to consider the long-term future of a solution. How might it change over time, and how frequently will we make changes? What might cause it to change? How will we handle said changes? Asking about versioning will have implications not just for our API designs but also for how we document our changes and share them with our users.
- *When this goes live, do we want to cache the requests?* The interesting part of this question is the first part: "When this goes live" Here we can use our understanding of a release process and how our production environments behave in our questioning. This can help us identify additional considerations that need to be made to ensure our designs work in line with existing rules and patterns in our platforms. For example, this question is trying to reveal whether we want to cache the response. If the answer is yes, this could have an impact on our designs (e.g., ensuring no randomized variables are used in URIs that might break caching rules).

Why



- *Why are we building this?* Perhaps a little contrary, but still an important question to ask. If we're to invest our time and money into delivering features, it's best that we are aware of what the value is in the work we're doing. A question like this can help us to understand and empathize with the end user. Any answers we explore from this type of question are going to give us more details about the problems our user might be facing, which will help us make an informed decision as to how we can solve it.
- *Why have we chosen to list features as an array?* This type of questioning is focusing more on design choices. Why are the features in an array? If it's a fixed list of features to pick from, why not have parameters for a sub-object? The goal is not to criticize the decisions that have been made but to explore alternatives and understand the choices that have been made.

How?



- *How will it work?* When asking questions in a collaborative context, there will be times in which you'll have a feeling that the question lacks value. A question like "How will it work?" is a great example because on the surface, it feels like you're opening yourself up to criticism for not understanding what is being discussed. However, a question like this can reveal a lot of misinterpretation between team members. Personally, I've asked this question to get a response from one team member who offers a conflicting view to other team members. Sometimes the basic questions work the best.
- *How will we test this?* Not all questions have to be about understanding the design. We can also question how we are going to respond to the decisions we make. Asking a question like this helps our testing because it can highlight any potential blockers. For example, if we've designed something that relies upon a scheduled task that triggers at specific time intervals, we might ask how we are going to test it and start a conversation around having more control over the time intervals to help speed up our testing.