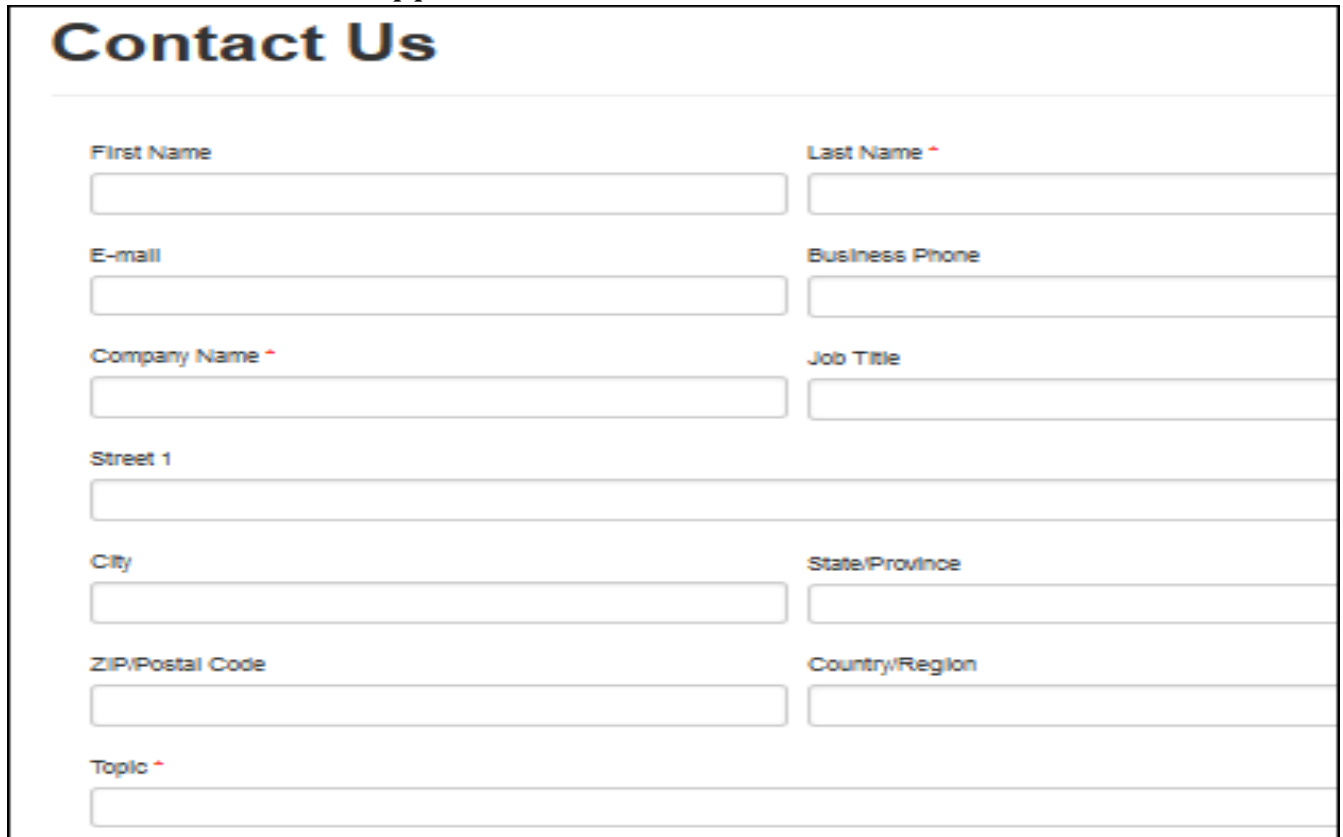


## CHAPTER 2.

Write test case for this application form.



The image shows a 'Contact Us' form with the following fields:

- First Name
- Last Name \*
- E-mail
- Business Phone
- Company Name \*
- Job Title
- Street 1
- City
- State/Province
- ZIP/Postal Code
- Country/Region
- Topic \*

Assignment Test the Below Form.

Write Test Cases for Below Form.

Name :

Mobile:

Test Cases for NAME text Box.

Verify input is Text

Verify input is Not Numeric

Verify the test box is 50 Characters

Test Cases for Mobile input.

## **What is API (Application Programming Interface) testing ?**

APIs, or Application Programming Interfaces, are the connecting tissue between different systems or layers of an application. Applications often have three layers: a data layer, a service (API) layer, and a presentation (UI) layer. The API layer contains the business logic of an application - the rules of how users can interact with services, data, or functions of the app. Because the API or service layer directly touches both the data layer and the presentation layer, it presents the sweet spot of continuous testing for QA and Development teams. While traditional testing has been focused on the UI, the advantages of API testing are becoming well known.

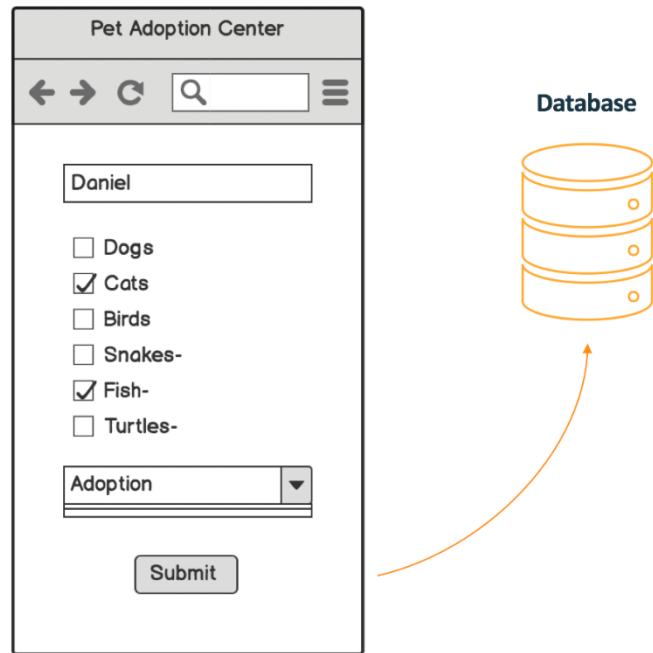
While there are many aspects of API testing, it generally consists of make requests to a single or sometimes multiple API endpoints and validate the response - whether for performance, security, functional correctness, or just a status check. While UI testing may focus on validating the look and feel of a web interface or that a particular payment button works - API testing puts much more emphasis on the testing of business logic, data responses and security, and performance bottlenecks.

### **An Example:**

The below example is a fairly simple and common functional test occurring at the UI level. We're heading to a website, filling out a form, submitting the form, and verifying that we are brought to the next screen.

### Steps:

1. Go to site
2. Fill in name
3. Find Cats & Click
4. Find Fish & Click
5. Find Dropdown & Click
6. Select Adoption
7. Click Submit
8. **Verify Next Screen**



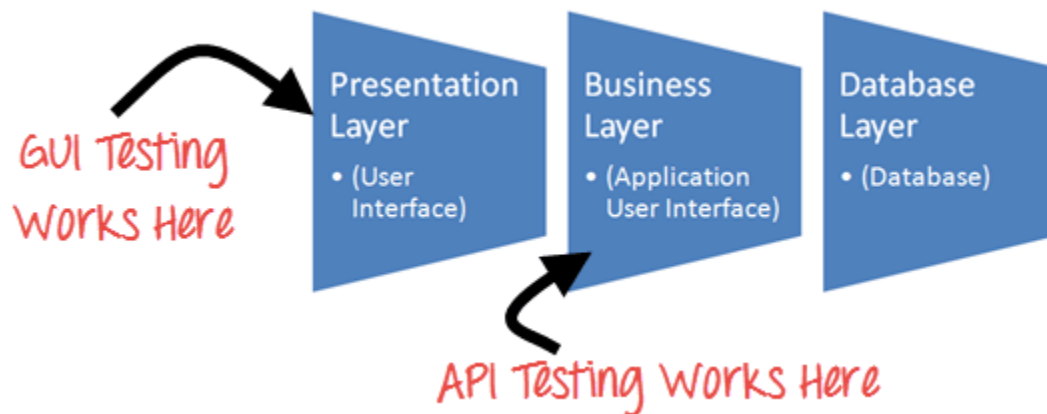
At the UI level, this simple test can present us with a couple of challenges. First, we are hampered by the physical limits of our browser and network connection, having to load the browser each time we want to run an iteration of this test. Second, any of these elements could change on the screen and our test would fail - if the "Dogs" entry is covering the "Cat" entry, we wouldn't be able to click it and our tests would fail. These challenges are annoying on their own - now try driving 10,000 different names and combinations through this form and see your build time grind to a halt.

With API Testing, this entire testing scenario can, and should, be boiled down to one step:

### Steps:

1. Hit endpoint
2. **Verify Response**





### Test Cases for API Testing:

Test cases of API testing are based on

- Return value based on input condition: it is relatively easy to test, as input can be defined and results can be authenticated
- Does not return anything: When there is no return value, a behavior of API on the system to be checked
- Trigger some other API/event/interrupt: If an output of an API triggers some event or interrupt, then those events and interrupt listeners should be tracked
- Update data structure: Updating data structure will have some outcome or effect on the system, and that should be authenticated
- Modify certain resources: If API call modifies some resources then it should be validated by accessing respective resources

API Testing Approach is a predefined strategy or a method that the QA team will perform in order to conduct the API testing after the build is ready. This testing does not include the source code. The API testing approach helps to better understand the functionalities, testing techniques, input parameters and the execution of test cases.

Following points helps the user to do API Testing approach:

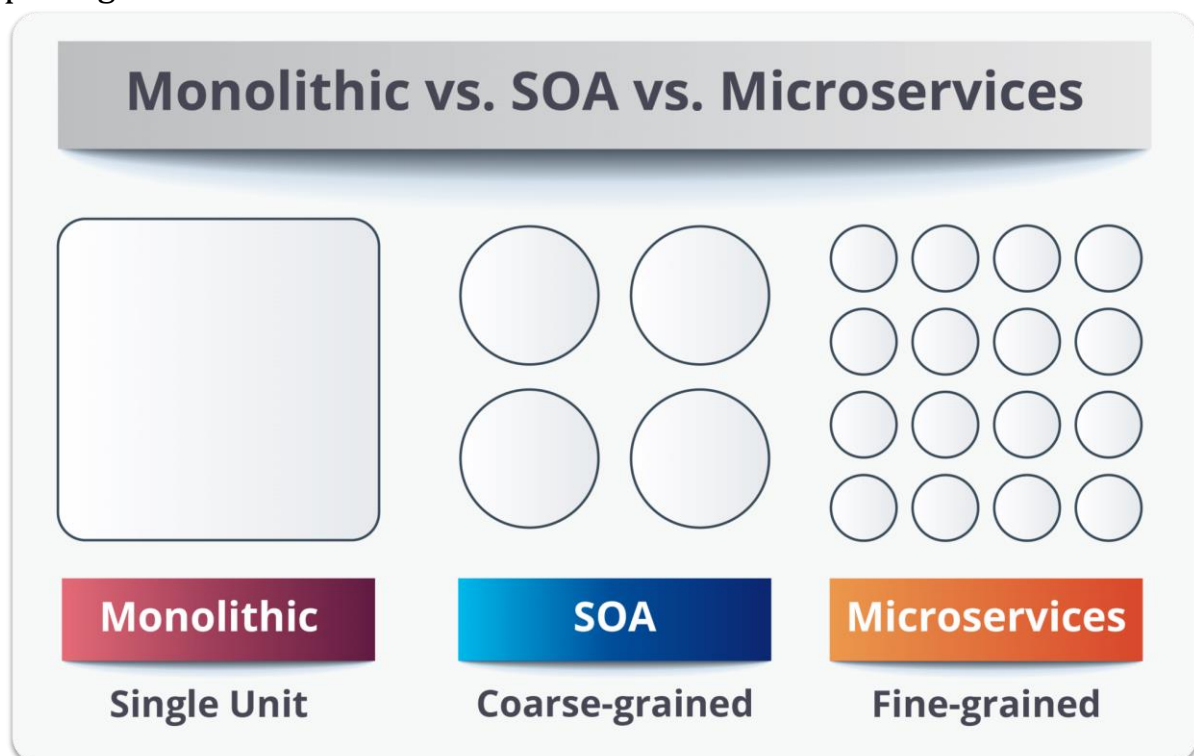
- Understanding the functionality of the API program and clearly define the scope of the program
- Apply testing techniques such as equivalence classes, boundary value analysis, and error guessing and write test cases for the API
- Input Parameters for the API need to be planned and defined appropriately
- Execute the test cases and compare expected and actual results

## LEARN ARCHITECTURE OF APPLICATION

SOA and microservices, which have distributed architectures, offer significant advantages over monolithic architecture. I will explain layered-based architectures and tell you the difference between microservices and SOA architecture.

Before we dive into the differences between microservices and SOA, let me just tell you the basic differences between monolithic architecture, SOA, and microservices:

In layman's terms, a monolith is similar to a big container wherein all the software components of an application are assembled together and tightly packaged.



Service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or two or more services coordinating some activity. Some means of connecting services to each other is needed.

Microservices, aka microservice architecture, is an architectural style that structures an application as a collection of small autonomous services modeled around a business domain.

### **Monolithic Application**

You can think of a monolithic application as a container which is basically hosting a number of software components. There can be a number of software components as part of your software application and if they are all hosted together and delivered together, then that's called a monolithic application.

Now there are various challenges with any software application which is implementing the monolithic architecture. Let's see the challenges:

First of all, they are not flexible. Monolithic applications can't be built using different technologies. That's the problem.

Second challenge is that, they are unreliable. Even if one feature of the system doesn't work, then the entire system will not work.

Third Challenge, these monolithic applications are not Scalable. They can't be easily scaled and even if the application needs to be updated, then the complete system has to be rebuilt.

The next problem is that, with these monolithic applications, they block continuous development. All the features can't be built and deployed at the same time. They will have to be done separately.

With monolithic applications, development is very slow. They take a lot of time to be built since each and every time, every feature has to be built one after another.

Most of all, monolithic applications are not fit for complex architecture. Because you can't use different technologies. You are very limited and very constrained.

## SOA - Service Oriented Architecture

With SOA, these services or features, they are broken down. Entire software application is not built as one but different features and different services are broken down into smaller components. That's why it's called Coarse Grained Architecture.

Let's say one software application which provides four features, then all those four features are delivered by four different services. That is what you can see in below illustration.

Each of these services would have multiple tasks inside them.

And these smaller tasks would rather be delivered as one particular feature.

and the whole software application comprises of a number of these features.

## MSA- MicroServices Architecture

But comparing to SOA, with microservices it is little more different. In Microservices, these features or services are further broken down into task-level-services



### SOA ORCHESTRA vs BALLERINA MSA

Any software application need not to be developed on the same programming language. In your core software application, you might have smaller applications and each of those smaller applications could be written on Java or Python or .NET or C# etc.

But when it's written in different programming languages it's tough for them to interact with one another. If you want these communications happen, then you have to bring a particular platform.

That's why we have this **messaging Middleware in SOA**.

In case of SOA, there are something called a messaging middleware which acts as a communication point or the interaction point between different applications which are in a different language.

In the above example, One application is on **C#/.NET** and other is on **classic JAVA**. All of these communications, they go through one particular messaging middleware.

But in case of Microservices, we don't have any broker/messaging middleware here.

Here, communication happens directly between application to application even if they are in different programming languages [structurally different and built on different grounds].

**Question.** In the above example, A C#/.NET application is communicating with a JAVA application. How that is happened?

It happens with the help of a REST API (REpresentational State Transfer).

Even if the applications of similar type (same language) wants to communicate, they do it through REST API.

See, No broker/messaging middleware here.

Difference with respect to Component Sharing

In SOA- this is very similar to previous section. [The Same Order Service]

Here it would be the same order service which is interacting with different smaller applications i.e Customer Management, Warehouse Management and Order Fulfillment.

Data would be stored in different databases.

But in MSA, for each and every application, [Separate Order Service]

i.e. for Customer management we have a separate order service, that order service will be accessing its own database for data.

Similarly for Warehouse management we have a separate order service with different database and similar for Order Fulfillment Application.

So, we can summarize it as

**In SOA - Share as much as Possible.**

**In MSA - Share as little as possible**



**API TESTING-TRAINING BOOK**  
**Trainer-Himadri Sen.**