



**W4**

API Testing

Wednesday, October 23rd, 2019 10:15 AM

# **Advanced Principles of API Testing | Part 1**

Presented by:

**Varuna Srivastava**

ThoughtWorks Canada

Brought to you by:



888-268-8770 · 904-278-0524 - [info@techwell.com](mailto:info@techwell.com) - <http://www.starcanada.techwell.com/>

# Varuna Srivastava

Varuna is a technical tester who's worked on award-winning projects across a wide variety of technology sectors, including retail, travel, financial, and the public sector, and worked with various web, mobile, and IoT technologies. Varuna is a passionate advocate of shipping quality code to production using agile practices. When not working, Varuna likes to get her hands dirty experimenting with her culinary skills. Most of her weekends are spent in cookgraphy—cooking plus photography!



# Advanced Principles of API Testing

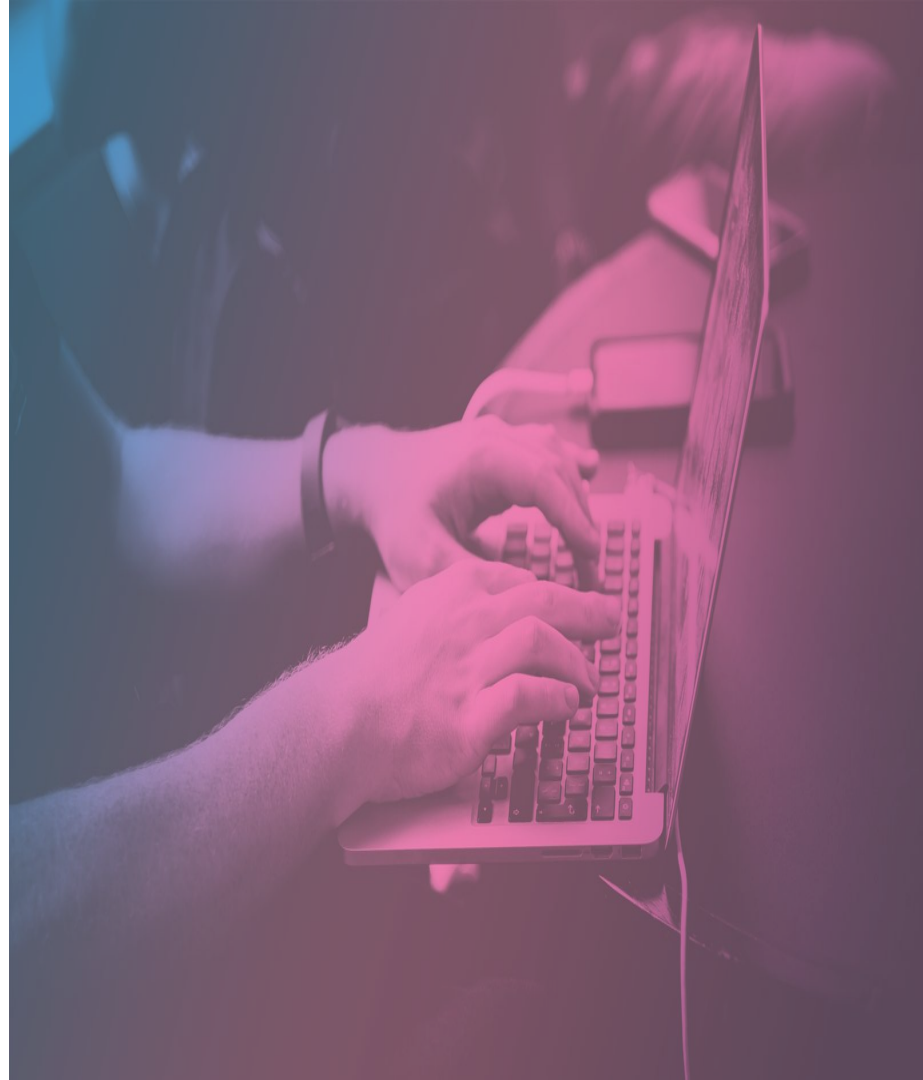
# About Me



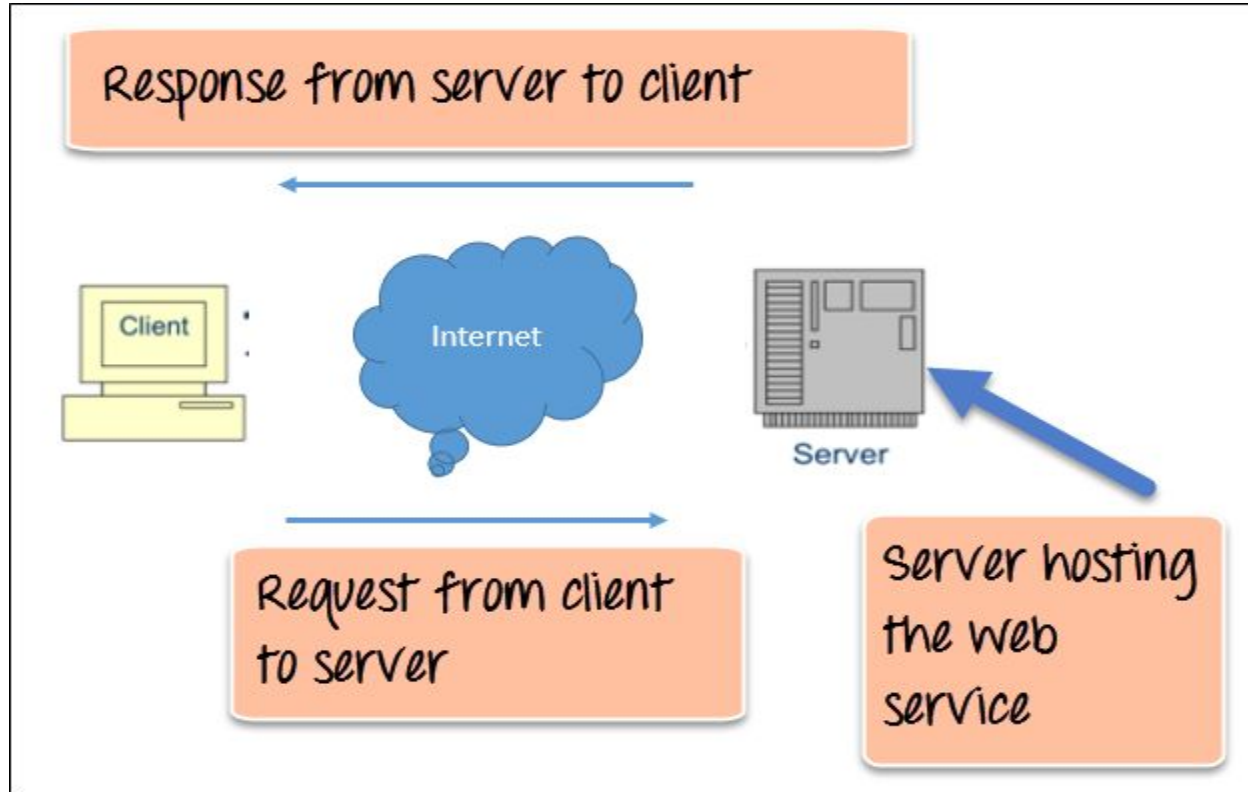
- Lead Quality Analyst at Thoughtworks
- Have been in Test automation for 10 yrs
- Testing traveler
- Conference Organizer @EuroTestConf
- Twitter handle : [@vibrantttester](https://twitter.com/vibrantttester)

# Agenda

- Role of API's
- API architecture
- What is API testing
- Types of api testing
- API documentation tools
- API design patterns
- API Test automation



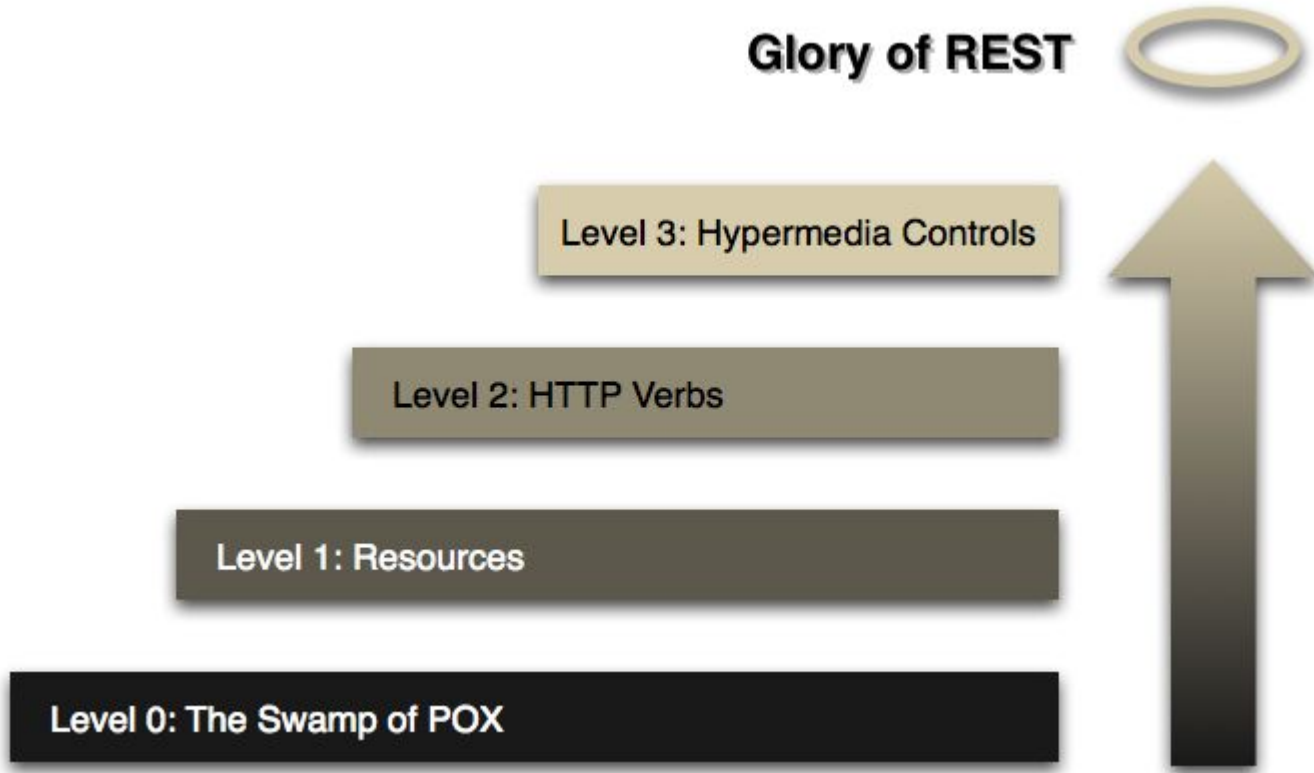
# Web services



# Web services

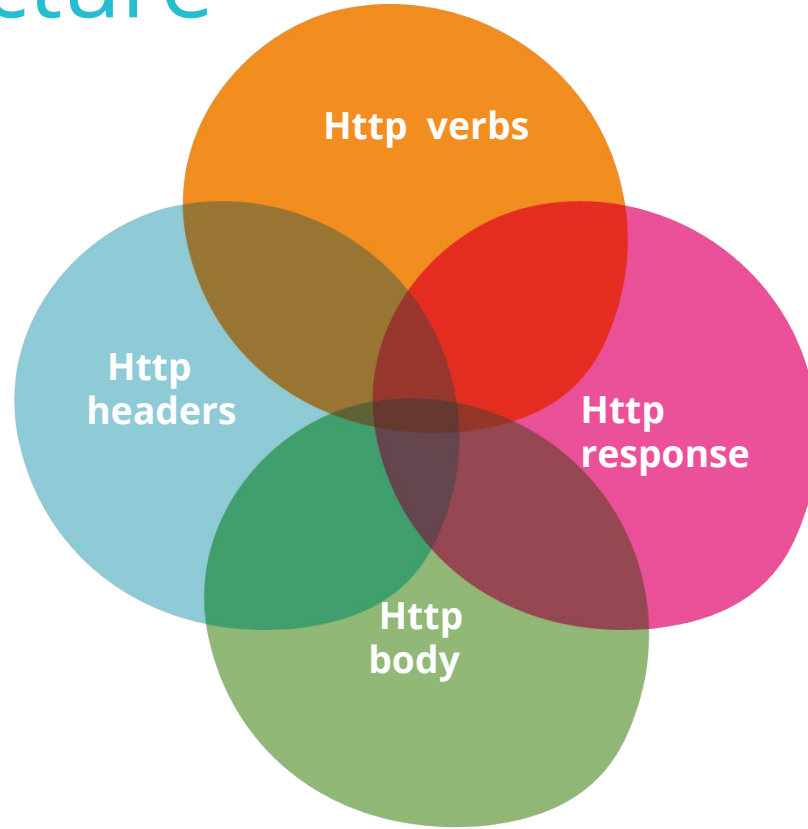


# REST architecture

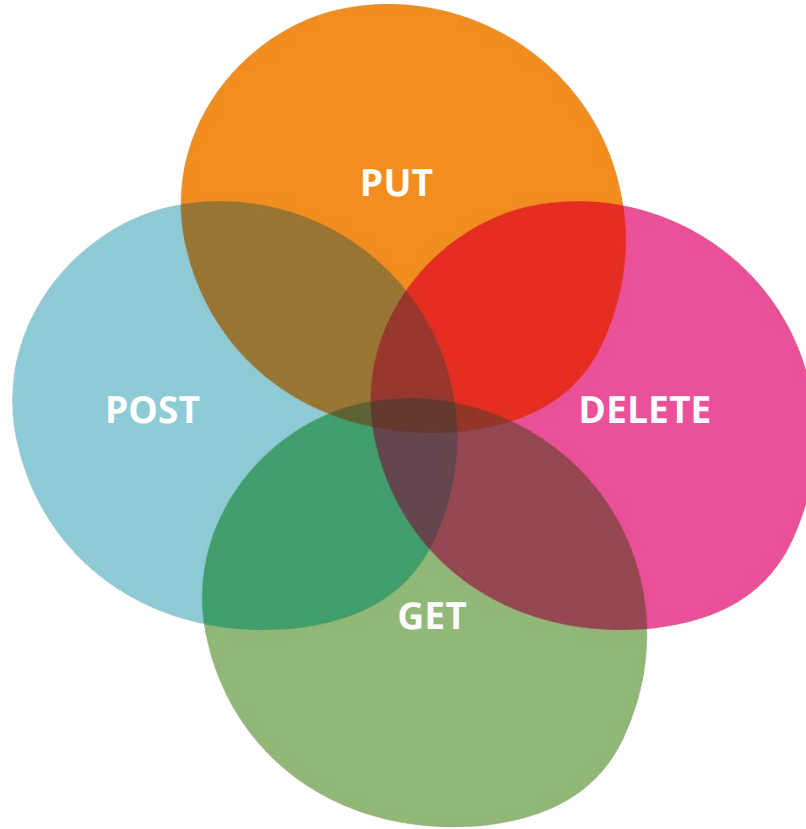




# API architecture



# Http verbs



POST ▼

GET

POST

PUT

PATCH

DELETE

COPY

HEAD

OPTIONS

LINK

UNLINK

PURGE

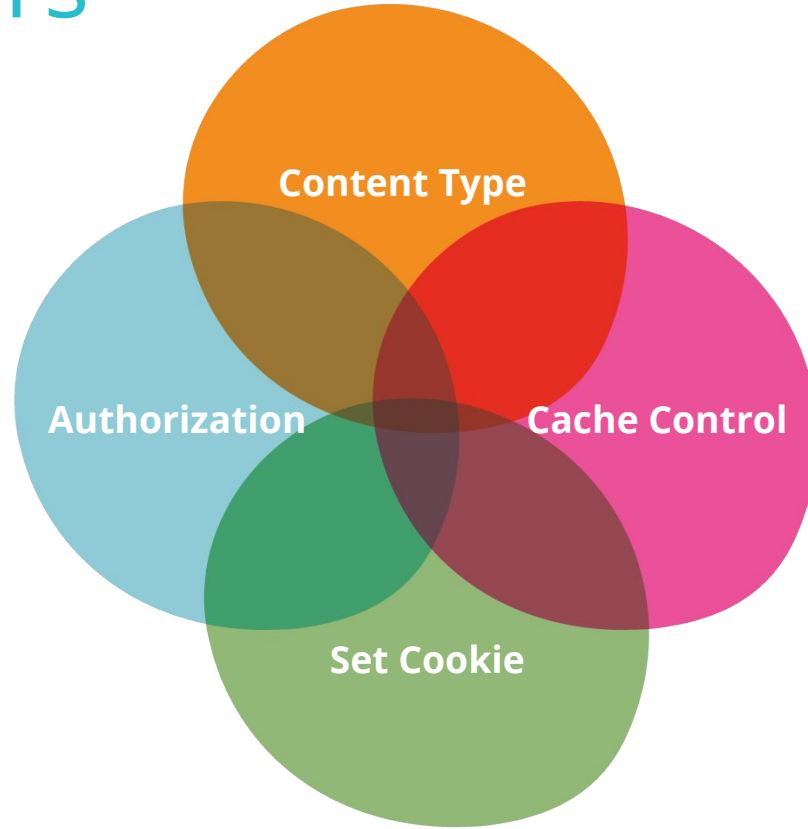
LOCK

UNLOCK

PROPFIND

VIEW

# Http headers



# Http headers → Set headers

```
given().request()  
    .with()  
    .contentType("application/json")  
    .header(headerName: "auth-token", basicAuthToken)  
    .header(headerName: "set-cookie", sessionId)  
    .header(headerName: "cache-control",  
            headerValue: "max-age:604800")
```

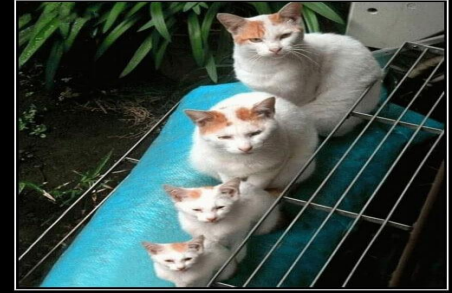
# Http Response



200  
OK



201  
Created



304  
Not Modified



400  
Bad Request



401  
Unauthorized



404  
Not Found

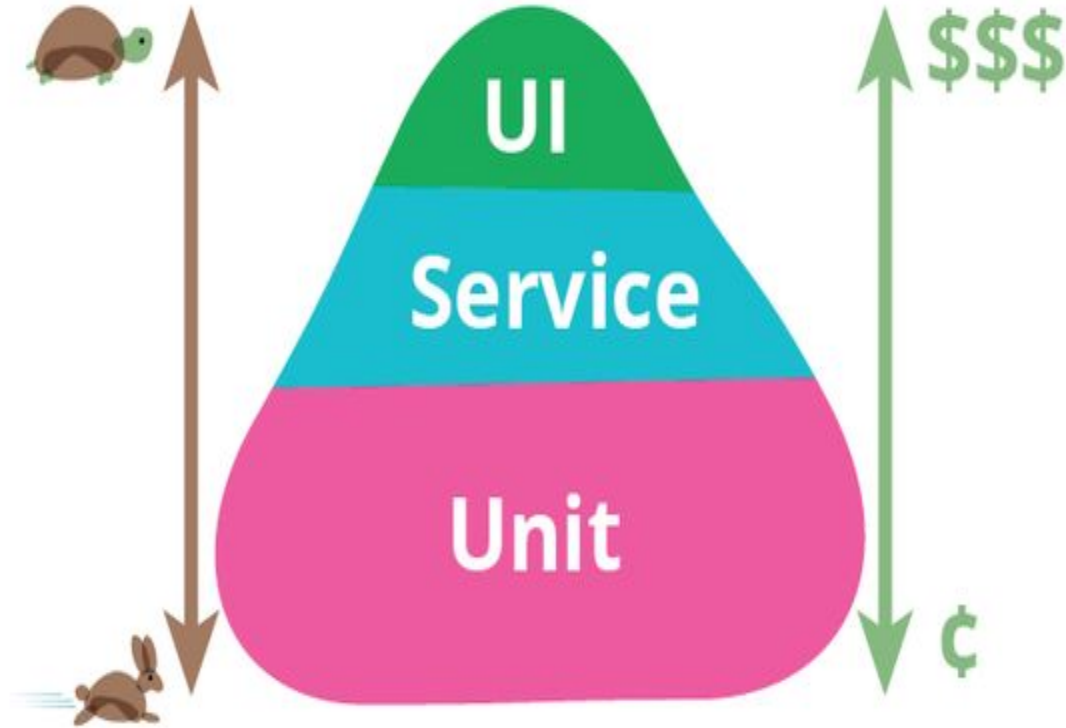
# Rest API structure

```
given().request()  
    .with()  
    .contentType("application/json")  
    .header( headerName: "auth-token", basicAuthToken)  
    .header( headerName: "set-cookie", sessionId)  
    .header( headerName: "cache-control", headerValue: "max-age:604800")  
    .queryParams( parameterName: "format", ...parameterValues: "json")  
    .body(new CreateAddressRequestBuilder().build())  
    .when()  
    .post( path: "http://localhost:8080/addresses")  
    .then()  
    .assertThat()  
    .statusCode(201)  
    .body( path: "addressId", notNullValue());
```

A photograph of a person's hands typing on a laptop keyboard. The image is overlaid with a semi-transparent blue and orange gradient. The text 'API Testing' is centered in white. The background shows a laptop screen and a desk with some papers and a smartphone.

# API Testing

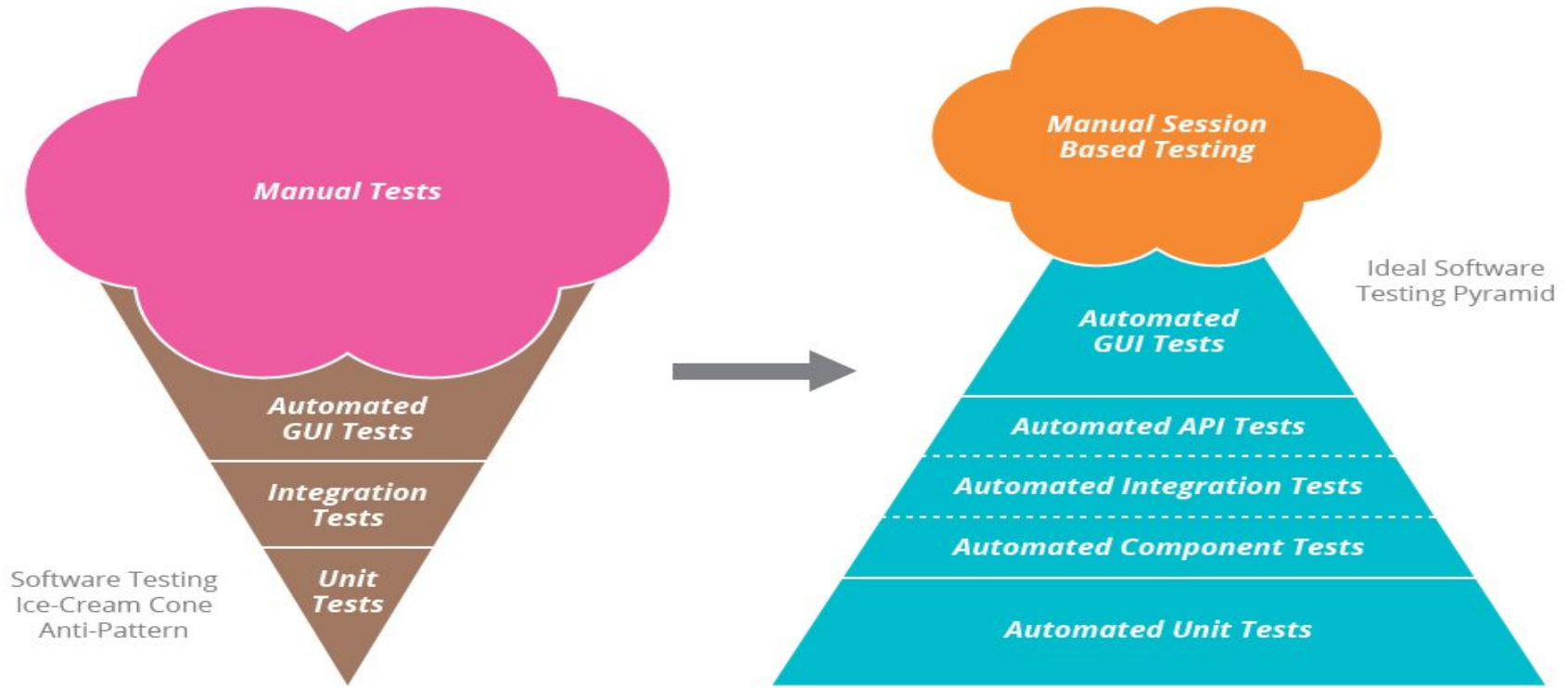
# Test Pyramid





# API Testing

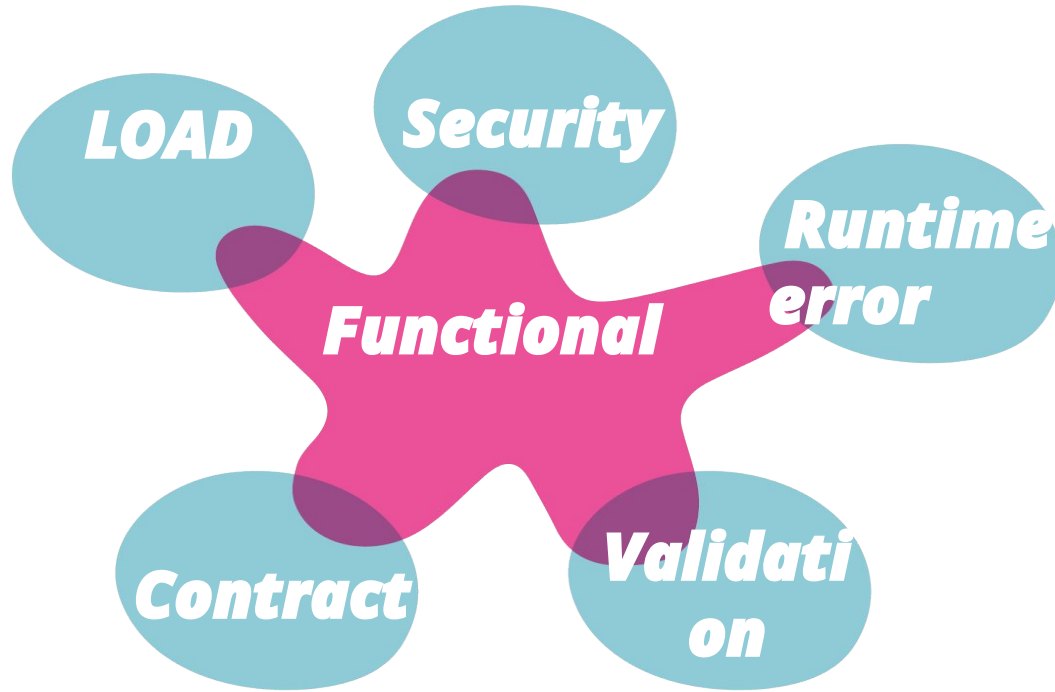
(Adapted from [watirmelon](#) blog)



The background image shows a person's hands typing on a laptop keyboard. The image is overlaid with a semi-transparent blue and orange gradient. The text 'Types Of API Testing' is centered in white.

# Types Of API Testing

# Types of API Testing



# API Functional Tests

- Focus on testing the functionality of respective api with valid inputs  
/searchItem By name By brand

## Responsibility :

- ❖ Define scope of api
- ❖ Verify edge case scenario
- ❖ Verify handled error scenario

**REST-assured**

# API Contract Tests

- Focus on the messages that flow between a consumer and provider /orders

## Responsibility :

- ❖ bugs in the consumer
- ❖ misunderstanding from the consumer about end-points or payload
- ❖ breaking changes by the provider on end-points or payload

PACT 

# API Load Tests

- Focus on verifying whether the theoretical solution works as a practical solution under a given load.

## Responsibility :

- ❖ Verify how **scalable** apis are at maximum user load
- ❖ Verify how quickly apis respond i.e **speed**
- ❖ Verify if the apis are **stable** under varying loads



# API Security Tests

- Focus is to make your data safe from hackers, and ensure that the API is as safe as possible

## Responsibility :







- ❖ Validated external threats
- ❖ Fuzz Testing
- ❖ Penetration testing

The image shows a close-up of a person's hands typing on a laptop keyboard. The scene is overlaid with a semi-transparent blue and orange gradient. The text 'API Documentation' is centered in white. The laptop screen is visible on the right, showing some text. The background is blurred, showing other parts of the laptop and the person's arms.

# API Documentation



# Swagger

pet Everything about your Pets			Find out more: <a href="http://swagger.io">http://swagger.io</a> 
POST	/pet	Add a new pet to the store	
PUT	/pet	Update an existing pet	
GET	/pet/findByStatus	Finds Pets by status	
GET	/pet/findByTags	Finds Pets by tags	
GET	/pet/{petId}	Find pet by ID	
POST	/pet/{petId}	Updates a pet in the store with form data	
DELETE	/pet/{petId}	Deletes a pet	
POST	/pet/{petId}/uploadImage	uploads an image	
store Access to Petstore orders			
GET	/store/inventory	Returns pet inventories by status	
POST	/store/order	Place an order for a pet	

# API Blueprint

FORMAT: 1A

# Dredd example

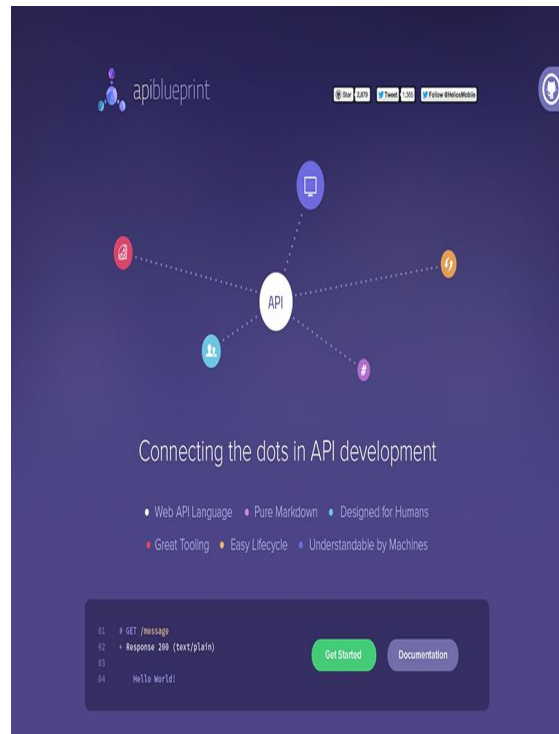
## Addresses [/addresses]

### Create Address [POST]

+ Request (application/json)

```
{
  "addressId": "1",
  "title": "Mrs",
  "firstName": "Varuna",
  "lastName": "Srivastava",
  "line1": "300 Front St West",
  "line2": "Blue building",
  "line3": "Box",
  "city": "Toronto",
  "state": "Ontario"
}
```

+ Response 201 (application/json; charset=utf-8)



# Dredd commands

```
npm install -g dredd
```

```
dredd init
```

```
dredd
```

# Dredd result


Local Development


Continuous Integration

Tutorial

Apiary

Details

Status  failed  
Hostname Varunas-MacBook-Pro.local  
Started a few seconds ago  
Duration 00:00.330  
Results Passes: 0, Failures: 0

 **POST /addresses**  
Create Address

## Create Address

**POST** /addresses

🔄 4 ms

🕒 03 Oct, 2019; 20:27:14 GMT

> Request

▼ Response

Diff · Real · Expected

Content-Type: application/json; charset=utf-8  
content-type: application/json

```
{  
  "addressId": 1  
}
```

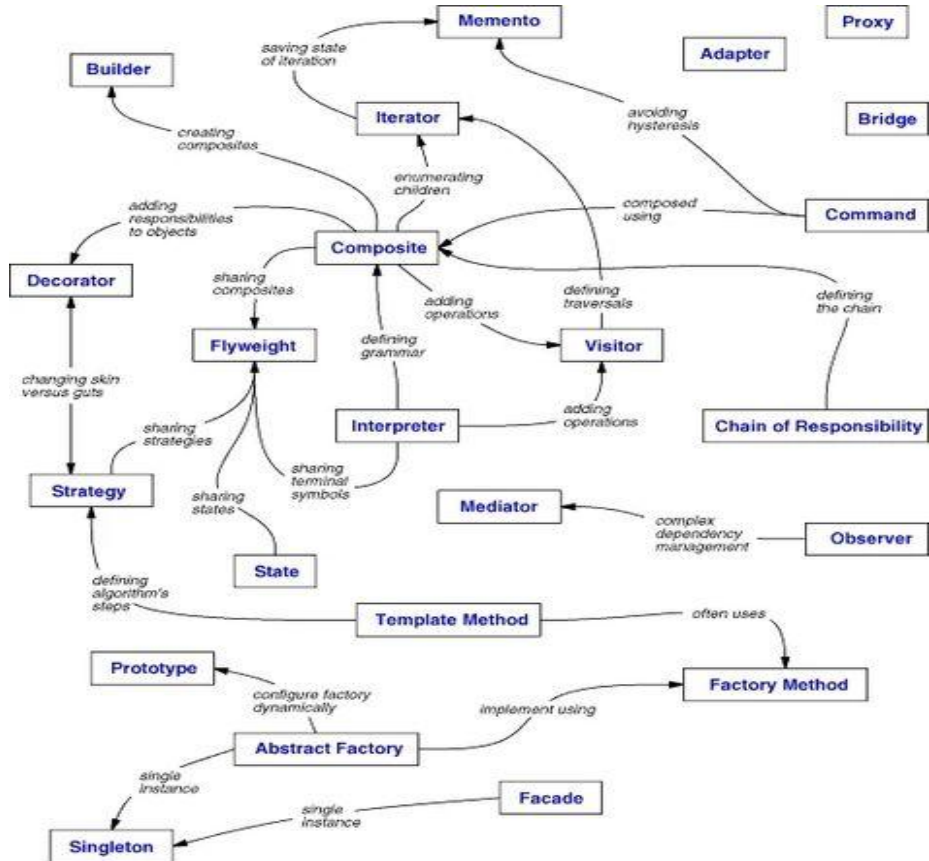
<https://app.apiary.io/public/tests/run/a017f6eb-e89e-4afe-8d49-327559c08d24>

@vibranttester

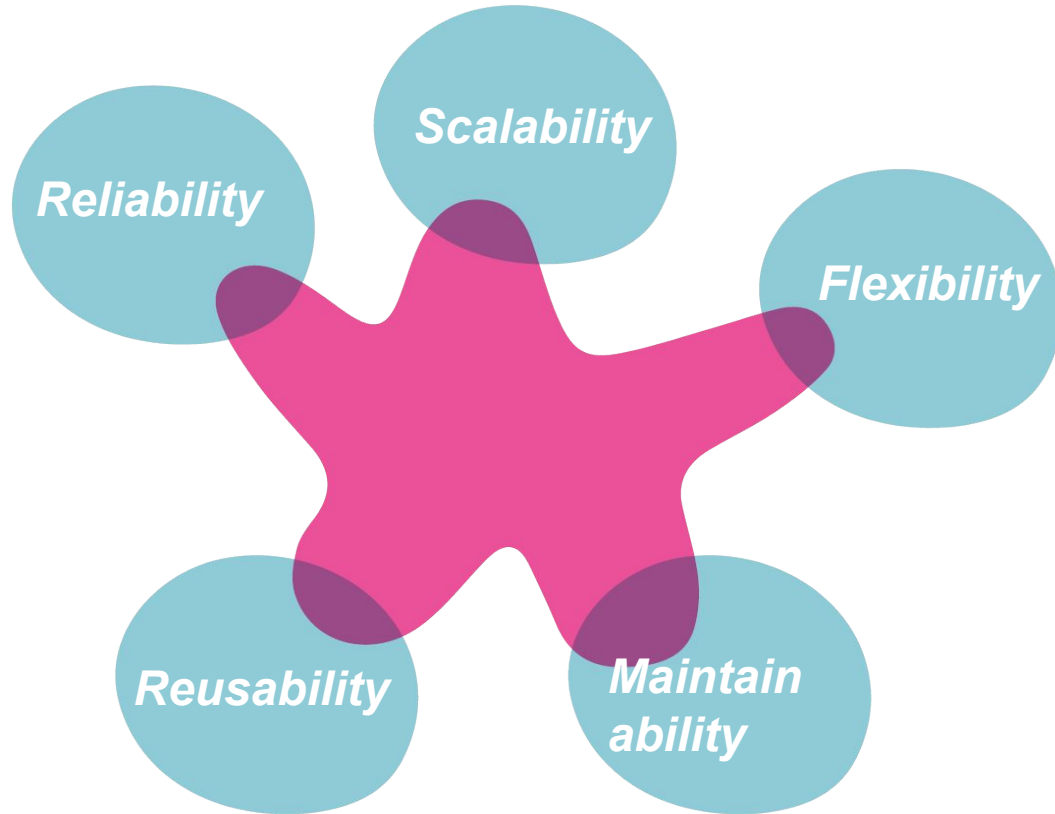
The background image shows a close-up of two people's hands typing on a laptop keyboard. The image is overlaid with a semi-transparent gradient that transitions from a deep blue on the left to a vibrant orange on the right. The text 'API Design Patterns' is centered in a bold, white, sans-serif font.

# API Design Patterns

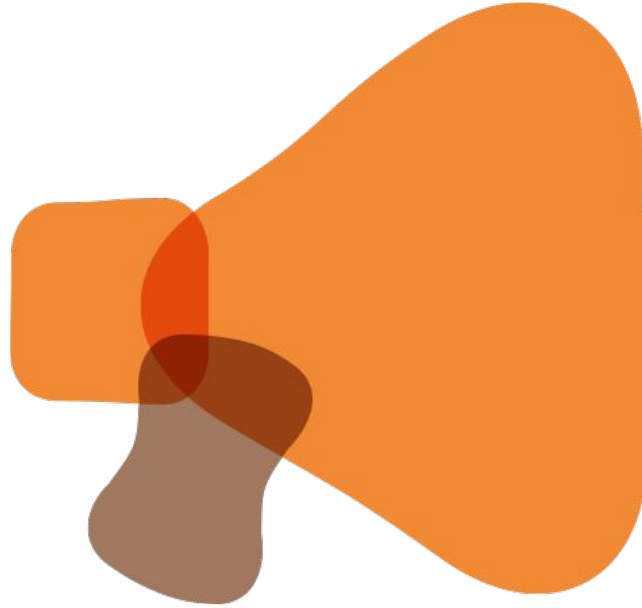
# Design Patterns



# Why Design Patterns in Test Automation?



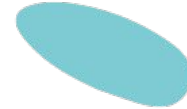
# Types of Design Patterns



Creational



Behavioural



Structural



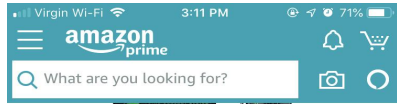


# Structural Design Patterns

# Structural Design Pattern

Structural Design Patterns are used to avoid duplicates in code and increase the **readability** and navigation of code in project

# ◇Page Object Pattern



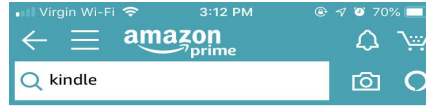
Save up to 40% on a BodyBoss Home Gym 2.0

**\$117.99** List: \$179.00 (34% off)

Ends in 11:43:28

[See all deals](#)

Shop men's clothing



Kindle Paperwhite — Now Waterproof with 2x the Storage + Kindle Unlimited (with...)

★★★★★ 11,892  
\$159.99 (1 new offer)



All-new Kindle - Now with a Built-in Front Light - Black - Includes Special Offers...

★★★★★ 2,356  
\$89.99 (1 new offer)



All-new Kindle Oasis - Now with adjustable warm light - Includes special offers...

★★★★★ 517  
\$279.99 (1 new offer)



Kindle Paperwhite — Now Waterproof with 2x the Storage (International Version)



Amazon

**Kindle Paperwhite — 32 GB, Wi-Fi, Includes Special Offers - Twilight Blue + Kindle Unlimited (with auto-renewal)**

★★★★★ (11,892)



Digital Storage Capacity:  
**32 GB**

Offer Type:

HomePage

Search Result Page

Search Item

Item Search Tests

@vibranttester

# ◇Page Object Pattern

SearchItemTests.java

Tests and  
method  
invocations

SearchItemPage.java

Method  
assertions

SearchItemId.java

Identifier of  
an element

# ◇Page Object Pattern

```
@Test(groups = Categories.SANITY)
public void verifySearchResults() throws InterruptedException {
    JourneyDetails journeyDetails = new JourneyDetailsBuilder().build();
    searchResultsPage = homePage.searchForAOneWayJourneyWith(journeyDetails);
    searchResultsPage
        .verifyCheapestIsSelected()
        .verifySearchResultsAreSortedByPrice();
}
```

# ◇Page Factory Pattern

Page Factory Pattern encapsulates page's attribute by **findby** annotations. It helps to work directly with page fields hiding the low level complexity.

# ◇Page Factory Pattern

```
public class HomePage {  
    WebDriver driver;  
    SearchResultsPage searchResultsPage;  
    private By searchTextBox=By.id("twotabsearchtextbox");  
    private By submitText=By.className("nav-input");  
  
    public HomePage(WebDriver driver) { this.driver = driver; }  
    public SearchResultsPage searchItem() {  
        driver.findElement(searchTextBox).sendKeys(...keysToSend: "Kindle");  
        driver.findElement(submitText).click();  
        return searchResultsPage=new SearchResultsPage(driver);  
    }  
}
```

# ◇Page Factory Pattern

```
public class HomePage {
    WebDriver driver;
    SearchResultsPage searchResultsPage;

    @FindBy(id = "twotabsearchtextbox")
    private WebElement searchTextBox;

    @FindBy(className = "nav-input")
    private WebElement submitText;

    public HomePage(WebDriver driver) {
        this.driver = driver;
    }

    public SearchResultsPage searchItem() {
        searchTextBox.sendKeys(...keysToSend: "Kindle");
        submitText.click();
        return searchResultsPage=new SearchResultsPage(driver);
    }
}
```



# ◇Page Factory Pattern

```
private void launchApplicationUnderTest() {  
    PropertyReader reader = new PropertyReader();  
    String applicationURL = reader.readProperty( key: "applicationURL");  
    driver.get(applicationURL);  
    SearchResultsPage searchResultsPage = new HomePage(driver).searchForTheJourney();  
}
```



```
private void launchApplicationUnderTest() {  
    PropertyReader reader = new PropertyReader();  
    String applicationURL = reader.readProperty( key: "applicationURL");  
    driver.get(applicationURL);  
    homePage = PageFactory.initElements(driver, HomePage.class);  
}
```

# ◇Chain of Invocation Pattern

Chain of invocation helps to avoid repeating **object** again and again before method invocations and makes code **pretty!!**

# ◇Chain of Invocation Pattern

```
ReviewOrderResponse roResponse = reviewOrder();  
roResponse.assertShippingAddress(addressId, shippingAddress);  
roResponse.assertBillingAddressss(shippingAddress);  
roResponse.assertPaymentMethod(piId, card, viewOrder().getGrandTotalAmount());  
roResponse.assertRootLevelAttributes(viewOrder(), userType: "G");
```



```
ReviewOrderResponse roResponse = reviewOrder();  
roResponse.assertShippingAddress(addressId, shippingAddress)  
    .assertBillingAddressss(shippingAddress)  
    .assertPaymentMethod(piId, card, viewOrder().getGrandTotalAmount())  
    .assertRootLevelAttributes(viewOrder(), userType: "G");
```

# ◇Chain of Invocation Pattern

```
addItemToCart() AddItemsToCartResponse  
    .addShippingAddress() AddShippingAddressResponse  
    .addPaymentInstructionWithBillingAddress() AddPIResponse  
    .submitOrder() OrderSubmitResponse  
    .assertAttributes(orderId);
```

# ◇Chain of Invocation Pattern

```
addItemToCart()  
    .addShippingAddress()  
    .submitOrder()  
    .addPaymentInstructionWithBillingAddress()  
    .assertAttributes(orderId);
```

# ◇Strategy Design Pattern

Strategy pattern is used whenever we want to have more than one implementations of the same action differently. It makes code more **flexible** and **maintainable** by using separation of concepts.

A photograph of a person's hands typing on a laptop keyboard. The image is overlaid with a semi-transparent blue and orange gradient. The text "Data Design Patterns" is centered in white.

# Data Design Patterns

# Data Design Pattern

Data Design Patterns are used to separate data and test logic. It **reduces** amount of data related code from test class.



# ◇Value Object Pattern

Value Object makes code more **readable** and it reduces amount of **repeatable** constructions. It is immutable which avoid modifications and extensions.

# ◇Builder Pattern

Builder pattern makes process of building **complex object** easier. We don't have to create multiple constructor for different scenario.

## ◇DataProvider Pattern

DataProvider pattern used to provide **parameters** to a test. A test method will be executed using the same instance of the test class to which the test method belongs.

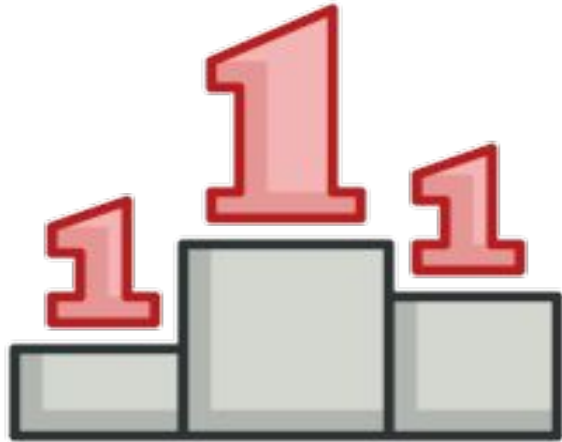
A photograph of a person's hands typing on a laptop keyboard. The image is overlaid with a semi-transparent blue and orange gradient. The text "Creational Design Patterns" is centered in white.

# Creational Design Patterns

## ◇ Singleton Pattern

Singleton class has only one instance, which provides a global access point to this instance. Singleton object is **initialized** only when it's requested for the first time.

# ◇ Singleton Pattern



*Example?*

# ◇ Singleton Pattern

```
synchronized static SingletonClass getInstance() {  
    if (instance == null)  
        instance = new SingletonClass();  
    return instance;  
}
```

```
String getAddress() { return "Address of star canada"; }
```

A background image showing a person's hands typing on a laptop keyboard. The image is overlaid with a semi-transparent blue and orange gradient. The text is centered in the upper half of the image.

**Let's try this out !!**  
**API Test Automation**



# Comments..?Doubts..?Complains..?

*Drop a note [@vibranttester](#) to continue this conversation*

**Varuna Srivastava**

LEAD QUALITY ANALYST

[varunas@thoughtworks.com](mailto:varunas@thoughtworks.com) | [thoughtworks.com](http://thoughtworks.com)

A close-up photograph of a person's hands typing on a laptop keyboard. The image is heavily stylized with a color gradient overlay, transitioning from a teal/cyan on the left to a magenta/pink on the right. A semi-transparent dark blue rectangular box is positioned diagonally across the center, containing the text 'Thank you' in a white, sans-serif font. The background shows the laptop screen and various keyboard keys, including a '#' key and an 'End' key. The overall aesthetic is modern and digital.

Thank you