

# API Testing basic

Gam Do

# Agenda

- API overview.
- What should be verify in API testing?
- Our testing API framework.

# What is API?

- API : Application Programming Interface.
- provides a way for systems to interact with each other.
- standardize information exchange: using protocols: SOAP, **REST**, ...
- Http method: Post, put, get, delete

Endpoint	https://apiurl.com/review/new
HTTP Method	POST
HTTP Headers	content-type: application/json accept: application/json authorization: Basic abase64string
Body	{ "review" : { "title" : "Great article!", "description" : "So easy to follow.", "rating" : 5 } }

# What should be verify in API testing

- We will verify the accuracy of the data.
- Will see the HTTP status code.
- We will see the response time.
- Error codes in case API returns any errors.
- Authorization would be check.
- Non-Functional testing such as performance testing, security testing.

Status code	Meaning
200 OK	Indicates success for GET, PUT, or POST requests
201 Created	Indicates a new object, such as a new order, has been created
400 Bad Request	Indicates the request was malformed
401 Unauthorized	Indicates the client is not allowed to access the requested resource and should reissue the request with the required credentials
403 Forbidden	Indicates that the request is valid and the client is authenticated, but the client is not allowed access to the requested page or resource for some reason
404 Not Found	Indicates that the requested resource is not available now
500 Internal Server Error	Indicates that the request is valid but the server is unable to handle it, possibly due to internal bugs
503 Service Unavailable	Indicates the server is down (for example, when undergoing maintenance)

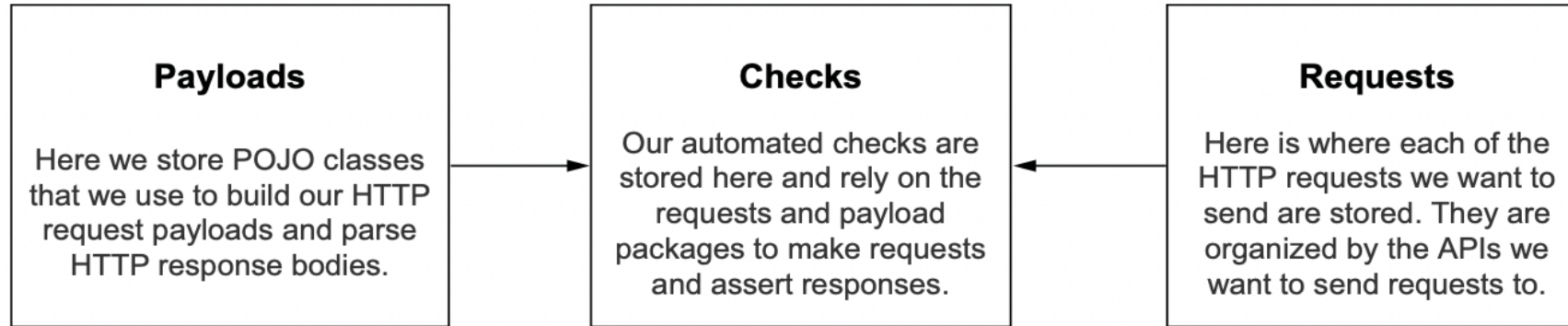
Tool to design test case for exploratory testing:

- Test Heuristic Cheat Sheet
  - <https://www.ministryoftesting.com/articles/ab1cd85c>
- *BINMEN: boundary, invalid entries, nulls, method, empty, negatives*
- *POISED: parameters, output, interoperability, security, exceptions, and data*

# Performance test and security test

- TBD

# Automation API testing framework





# Create project we need:

- IntelliJ IDE
- Create maven project
  - <https://mvnrepository.com/>
- Add dependencies:
  - *TestNG*
  - *Jackson binding*
  - *Rest-Assured*

```
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <version>5.2.1</version>
  <scope>test</scope>
</dependency>

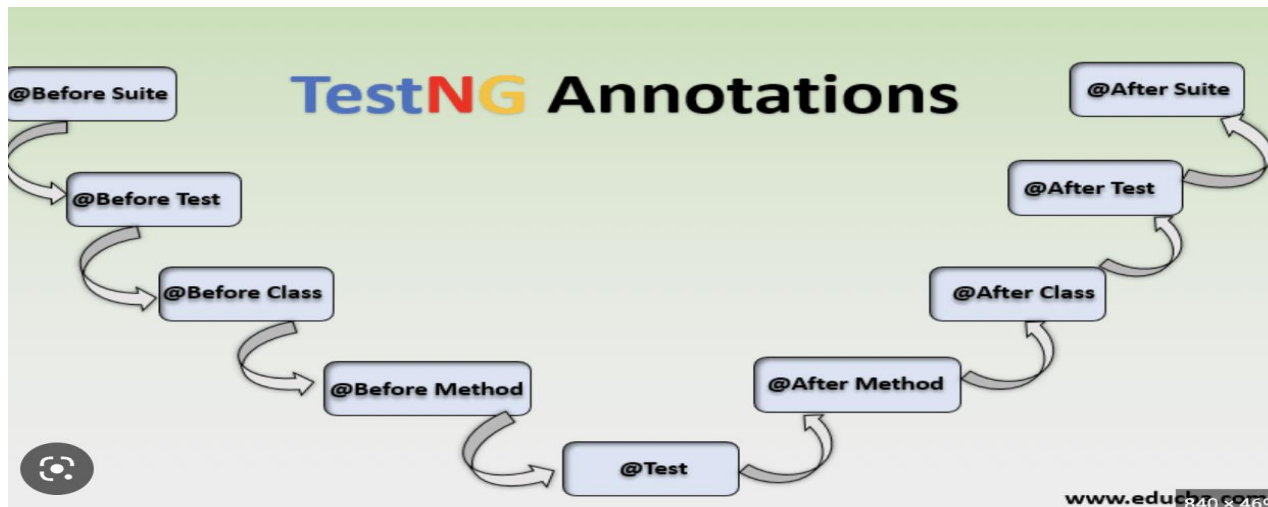
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.14.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.14.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.datatype/jackson-datatype-jsr310 -->
<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-jsr310</artifactId>
  <version>2.14.1</version>
</dependency>
```

# TestNG

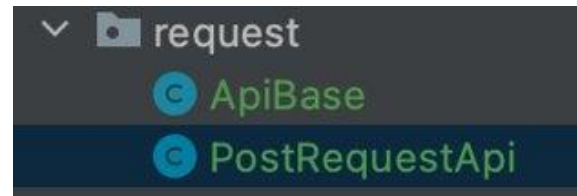
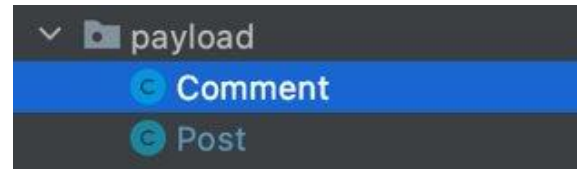
- **TestNG** is a testing framework
  - <https://testng.org/doc/>
- Annotations: @test, @beforeTest...
- Support for data-driven testing (with @DataProvider).



- **Rest Assured** enables you to test REST APIs using java libraries and integrates well with Maven
- **Jackson binding:** convert plain old Java objects into JSON for our HTTP requests and back into usable objects when we receive JSON back in HTTP responses.

# Our testing framework:

- Request structure: URI + base path + path Parameters + query Parameters
- Payload part:
- Request part:
- Check part:



```
//verify status and response
assertEquals(response.getStatusCode(), expected: 200);

Post expectedPost = new Post( userId: 1, id: 1, title: "abc", body: "abc")
Post actualResult = response.getBody().as(Post.class);
Assertions.assertThat(actualResult)
    .as("description: ")
    .usingRecursiveComparison()
    .isEqualTo(expectedPost);
```

# Connect to database: using properties files

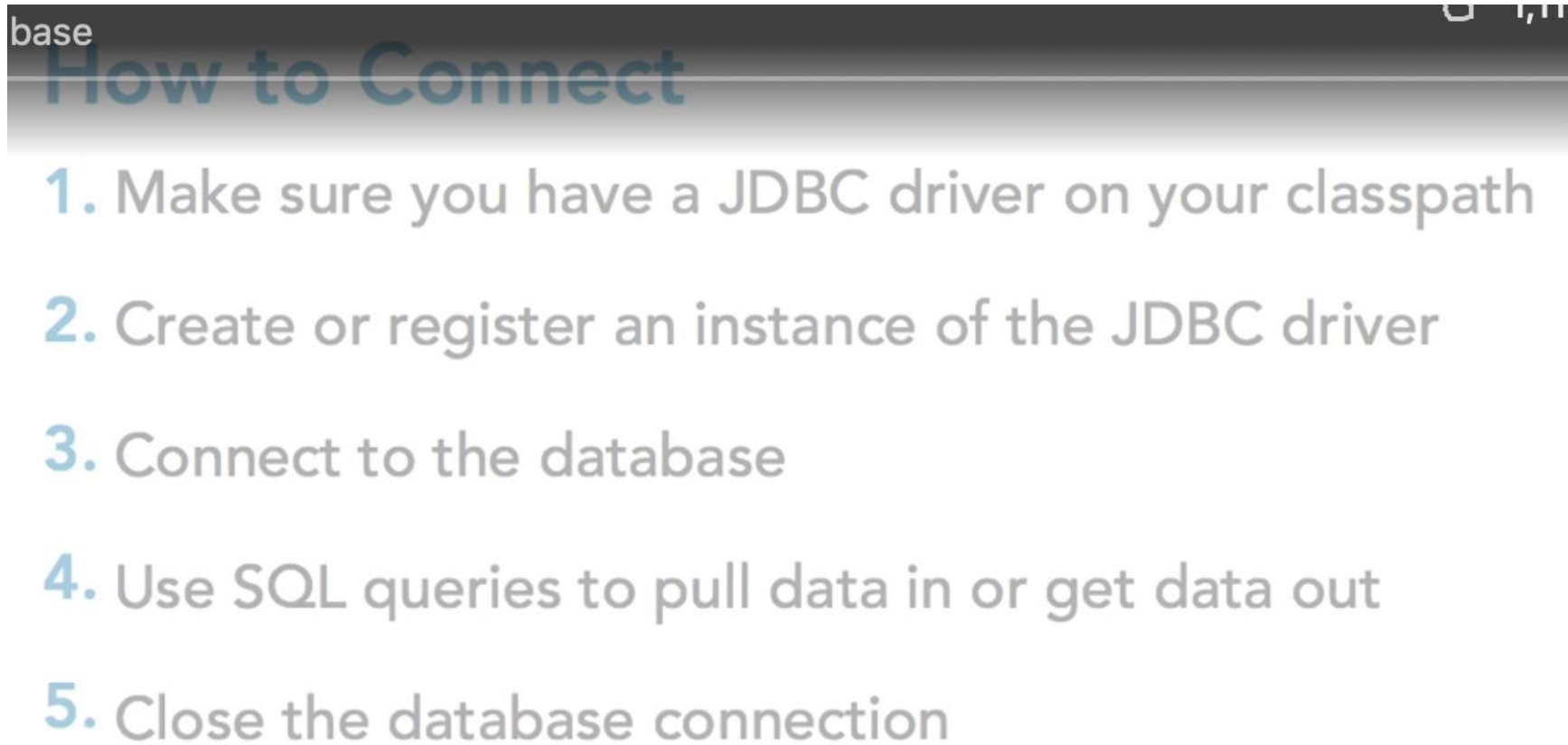
- JDBC
- Properties file structure: key/value pairs

```
public void readProperties() throws IOException {  
    try (InputStream in = new FileInputStream("db.properties")) {  
        Properties prop = new Properties();  
        prop.load(in);  
        dbUrl = prop.getProperty("dbUrl");  
        dbUser = prop.getProperty("dbUser");  
        dbPassword = prop.getProperty("dbPassword");  
    }  
}
```

```
public class DatabaseTest {  
    private static final String DB_PROPERTIES = "db.properties";  
    private String dbUrl;  
    private String dbUser;  
    private String dbPassword;
```

```
public void readProperties() throws IOException {  
    try (InputStream in = new FileInputStream(DB_PROPERTIES)) {  
        Properties prop = new Properties();  
        prop.load(in);  
        dbUrl = prop.getProperty(DB_URL);  
        dbUser = prop.getProperty(DB_USER);  
        dbPassword = prop.getProperty(DB_PASSWORD);  
    }  
}
```

# Connect to database



base

## How to Connect

1. Make sure you have a JDBC driver on your classpath
2. Create or register an instance of the JDBC driver
3. Connect to the database
4. Use SQL queries to pull data in or get data out
5. Close the database connection

# Connect to database

---

- DriverManager – a class that interacts with the driver for creating connections
- DataSource – a modern class that interacts with the driver for creating connections
- Connection – a class that the developer interacts with that manages the actual communication between the client and the server

# Connect to database

- T.B.D



Thank you  
for  
listening!

