

# Async Requests

In this lesson, we will learn how to handle async Requests for APIs.

## We'll cover the following



- What is an async request?
- Handling async requests

## What is an async request? #

Synchronous request blocks the execution of server code until the response is received, whereas asynchronous request (async in short), does not block the execution and returns a callback to the client which can be used to receive the actual data once the execution is complete.

## Handling async requests #

**REST Assured** does not support async requests out of the box. We can automate these use cases using a third-party open-source library.

We will be using **asynhttpclient** for the same and to use this, let's add below dependency to our build file.

### Maven

Add this in the *pom.xml* file:

```
<dependency>
  <groupId>org.asynhttpclient</groupId>
  <artifactId>async-http-client</artifactId>
  <version>2.12.0</version>
</dependency>
```

### Gradle

Add this in the *build.gradle* file:

```
compile group: 'org.asynhttpclient', name: 'async-http-client', version: '2.12.0'
```

```
compile group: 'org.asynchttpclient', name: 'async-http-client', version: '2.12.0'
```

Let's now understand how we can handle async requests using the code below:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.testng.annotations.Test;
import org.asynchttpclient.Dsl;
import java.util.concurrent.Future;
import org.asynchttpclient.Response;
import java.util.concurrent.TimeoutException;
import java.util.concurrent.ExecutionException;

public class APIDemo {

    private static Logger LOG = LoggerFactory.getLogger(APIDemo.class);

    static {
        // setting logger to INFO to disable unwanted http client logs
        ((ch.qos.logback.classic.Logger) org.slf4j.LoggerFactory
            .getLogger(ch.qos.logback.classic.Logger.ROOT_LOGGER_NAME))
            .setLevel(ch.qos.logback.classic.Level.INFO);
    }

    @Test
    public void asyncTest() throws InterruptedException, ExecutionException, TimeoutException {

        String url = "https://reqres.in/api/users?delay=3";

        Future<Response> whenResponse = Dsl.asyncHttpClient().prepareGet(url).execute();

        Response response = whenResponse.get();
        LOG.info(response.getResponseBody());
    }
}
```



## Let's understand the code above

- Target URL sends back the response after a delay of 3 secs

```
String url = "https://reqres.in/api/users?delay=3";
```

- Creates an object of `Future` of type `Response` and makes a GET call using the `prepareGet(url)` method

```
Future<Response> whenResponse = Dsl.asyncHttpClient().prepareGet(url).execute();
```

- Using the `Future` `Response` above, it waits if necessary for the computation to

Using the `Future` Response above, it waits if necessary, for the computation to complete, and then retrieves its result until the future returns response

```
Response response = whenResponse.get();
```

To wait with a maximum timeout, we can use the code below:

```
Response response = whenResponse.get(10, TimeUnit.SECONDS);
```

If Future does not complete (or if we don't receive response) with 10 seconds, we get `java.util.concurrent.TimeoutException`.

---

In the next lesson, we will learn about proxy server settings.