Automation (Part #1)
All sensitive passwords/tokens are provided as environment variables to prevent access to them and be able to protect their values. They can be encrypted and used as global passwords later on when we execute the tests from a CI server.

Used technologies:
- **JUnit:** To be able to write test cases with proper setup and teardown and have verifications in the code.
- **JUnitParams:** To easily use parameters with Junit 4.x
- **Selenium:** To be able to use the WebDrvier API to interact with local/remote browsers.
- **Log4j:** To be able to easily control the log level and where to output the logs.

Supported browsers:
The framework can run on two browsers: Chrome and Firefox (latest versions). At the time of writing this document, the versions supported are:
- Chrome 71.0.3578.98
- Firefox 64.0

Execution modes:
The framework supports two execution modes:
- **LOCAL:** Local execution on the supported browsers using locally installed browsers.
- **BROWSERSTACK:** Running on the BrowserStack cloud provider VMs. For this to work you need to provide the following environment variables:
  - BROWSERSTACK_USER: The username to access BrowserStack.
  - BROWSERSTACK_TOKEN: The token to access BrowserStack.

How to run a test:
You can run a test using the following Maven command:
```
mvn clean install test -Dtest=WebTests -Dbrowser=Chrome -DexecutionMode=LOCAL
```

Parameters and their values:
Parameter	Usage	Values
test	Run a specific test or a set of tests from a certain class.	<CLASS>#<METHOD> <CLASS>
browser	Specifies the browser to run the test cases on (Case insensitive)	Chrome Firefox
executionMode	Specifies the mode of execution of the test cases (Case sensitive)	LOCAL BROWSERSTACK

Test cases:
ID	TC001	
Test Name	test1GetNumberOfWinsPerConference	
Test Description	checks what was the overall number of wins of teams from Eastern and Western conference during regular season	
Steps / Expected results	Navigating to the website http://stats.nba.com/teams/traditional/?sort=W_PCT&dir=-1	Website opens and shows the combined results for all teams

	Select season “2017-18” from the season combo-box.	Table updates and shows the correct season results.
	Click on the “Advanced Filters” button, choose “East” as a conference and then click on the “RUN IT” button to run the filter operation.	Table results are filtered and only “East” conference teams are shown.
	Sum up the number of wins for all the east teams.	We get the total number of wins for all the Eastern conference.
	Select season “2017-18” from the season combo-box.	Table updates and shows the correct season results.
	Click on the “Advanced Filters” button, choose “West” as a conference and then click on the “RUN IT” button to run the filter operation.	Table results are filtered and only “West” conference teams are shown.
	Sum the number of wins for all the west teams.	We get the total number of wins for all the Western conference.

ID	TC002	
Test Name	test2SumOfTeamPlayerPointsIsCorrect	
Test Description	Checks sum of average points per game for players of two teams - Golden State Warriors and Houston Rockets is correct.	
Steps / Expected results	Navigate to page https://stats.nba.com/leaders/	Page opens and a list of players is displayed according to their performance.
	Search for one of the mentioned teams by typing the team name in the search box in the top right corner of the page.	Team data is displayed.
	Click on the “Team Stats” link and choose “Players” to switch to players data type.	Players information is displayed for each player in the team.
	From the season combo-box, select season “2017-18”	Player data is changed to show the values for the requested season.
	Get the sum of points “PTS” for each player and compare it with the value that you calculated manually.	The two values should be the same (Delta is 0.0001).

ID	TC003	
Test Name	test3PlayerStatusAreCorrect	
Test Description	Verify that the points per game, assists per game, rebounds per game values are correct for the first 3 players in the leaders page.	
Steps / Expected results	Navigate to page https://stats.nba.com/leaders/	Page opens and a list of players is displayed according to their performance.
	Get the values of points per game, assists per game, rebounds per game for each of the first 3 players	Values are fetched from the website along with a link to each player’s profile page.

	Open the profile page of each player by navigating to it in your browser.	Player profile page opens.
	Compare the values you have stored with the values displayed for each player.	Values of points per game, assists per game, rebounds per game should match the values fetched before from the leaders page.

ID	TC004	
Test Name	test4PlayerPageLoadingTimeIsAcceptable	
Test Description	Check if the player page loads the stats segment in under 4 seconds	
Steps / Expected results	Navigate to the player page (https://stats.nba.com/player/201935/traditional/ for example)	Webpage shows the player stats.
	Take the current timestamp (call it start timestamp)	The current timestamp marks the finish of page loading and the start of Ajax requests loading.
	We should wait till all the Ajax requests finish loading.	All Ajax requests have been loaded successfully.
	Take the current timestamp (call it finish timestamp)	The difference between the finish timestamp and start timestamp should be less than 4 seconds

API (Part #2)

For the API testing part, we will be running 3 test cases to verify three different HTTP response codes which are 200, 404 & 429.

Test cases:

ID	TC005
Test Name	getItalysTop10ScorersTest
Test Description	Get a list of the top 10 scorers in Italy's top league
End-Point	https://api.football-data.org/v2/competitions/SA/scorers
Status Code	200

ID	TC006
Test Name	getNonExistingResourceTest
Test Description	Try to execute an API call using an endpoint that is matching the correct API version.
End-Point	https://api.football-data.org/v1/competitions/SA/scorers
Status Code	404

ID	TC007
Test Name	tooManyRequestsTest
Test Description	Exceeding the number of requests allowed per minute.
End-Point	https://api.football-data.org/v2/competitions/SA/scorers
Status Code	429

Used technologies:

- **JUnit:** To be able to write test cases with proper setup and teardown and have verifications in the code.
- **Rest assured:** To be able to write API test cases easily.
- **Jackson Databind:** To be able to parse the API response body easily to Java objects and use them in the code.

How to run a test:

You can run the test using a JUnit run configuration in Eclipse. Make sure to add an environment variable called "API_TOKEN" with a value matching the exact API token you got when you created an account to use the API service, otherwise you won't be able to execute the API test cases.

Load Testing (Part #3)

Q&A:

A) Explain the test in details
Navigating to the homepage of the website and simulating 1000 users doing this for 15 seconds. The website details are:

i. **Website:** http://www2.cuny.edu/
ii. **Page size:** 114,645 KB
iii. **Response time:** 519.59 ms

B) Did the load test have an impact on web application response time?
Yes, it did. The average response time before the test is 519.59 ms and during the test it was 42,342 ms. The exact test details are:

- **Total test time:** 1 minute 19 seconds
- **Response times:**
  - **Min:** 914 ms
  - **Max:** 69,119 ms
  - **Avg:** 42,342 ms
- **Error rate:** 67.10%

C) What is the optimal application response time for modern day web applications?
The average value for 2018 is 8.66 seconds and the webpage's size is 1.88 MB (depending on the webpage size and the website's industry).

Table showing website response times in 2018 per industry (Source):

Industry	**United States**	**United Kingdom**	**Germany**	**Japan**
Automotive	9.5 sec	12.3 sec	11.0 sec	10.3 sec
Business & Industrial Markets	8.7 sec	8.3 sec	8.2 sec	8.1 sec
Classifieds & Local	7.9 sec	8.3 sec	7.0 sec	8.3 sec
Finance	8.3 sec	8.0 sec	8.6 sec	7.6 sec
Media & Entertainment	9 sec	8.8 sec	7.6 sec	8.4 sec
Retail	9.8 sec	10.3 sec	10.3 sec	8.3 sec
Technology	11.3 sec	10.6 sec	8.8 sec	10sec
Travel	10.1 sec	10.9 sec	7.1 sec	8.2 sec

Table showing webpage size in 2018 per industry (Source):

Industry	**United State**	**United Kingdom**	**Germany**	**Japan**
Automotive	2.1 MB	2.6 MB	2.6 MB	2.5 MB
Business & Industrial Markets	1.6 MB	1.8 MB	1.5 MB	1.8 MB
Classifieds & Local	1.6 MB	1.6 MB	1.2 MB	2.1 MB
Finance	1.3 MB	1.3 MB	1.3 MB	1.7 MB
Media & Entertainment	1.9 MB	1.7 MB	1.4 MB	2.5 MB
Retail	2.1 MB	2.2 MB	2 MB	2.6 MB
Technology	2.3 MB	1.9 MB	1.7 MB	2.7 MB
Travel	2 MB	1.8 MB	1 MB	1.8 MB

D) Analyze few HTTP/S responses

i. Response #1:

The HTTP connection times out here after waiting for the maximum 60 seconds for a reply to arrive from the server.

```
org.apache.http.conn.HttpHostConnectException: Connect to www2.cuny.edu:80 [www2.cuny.edu/128.228.0.52] failed: Operation timed out (Connection timed out)
	at org.apache.http.impl.conn.DefaultHttpClientConnectionOperator.connect(DefaultHttpClientConnectionOperator.java:159)
	at org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl$JMeterDefaultHttpClientConnectionOperator.connect(HTTPHC4Impl.java:331)
	at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.connect(PoolingHttpClientConnectionManager.java:373)
	at org.apache.http.impl.execchain.MainClientExec.establishRoute(MainClientExec.java:394)
	at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:237)
	at org.apache.http.impl.execchain.ProtocolExec.execute(ProtocolExec.java:185)
	at org.apache.http.impl.execchain.RetryExec.execute(RetryExec.java:89)
	at org.apache.http.impl.execchain.RedirectExec.execute(RedirectExec.java:110)
	at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:185)
	at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:83)
	at org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl.executeRequest(HTTPHC4Impl.java:832)
	at org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl.sample(HTTPHC4Impl.java:570)
	at org.apache.jmeter.protocol.http.sampler.HTTPSamplerProxy.sample(HTTPSamplerProxy.java:67)
	at org.apache.jmeter.protocol.http.sampler.HTTPSamplerBase.sample(HTTPSamplerBase.java:1231)
	at org.apache.jmeter.protocol.http.sampler.HTTPSamplerBase.sample(HTTPSamplerBase.java:1220)
	at org.apache.jmeter.threads.JMeterThread.doSampling(JMeterThread.java:622)
	at org.apache.jmeter.threads.JMeterThread.executeSamplePackage(JMeterThread.java:546)
	at org.apache.jmeter.threads.JMeterThread.processSampler(JMeterThread.java:486)
	at org.apache.jmeter.threads.JMeterThread.run(JMeterThread.java:253)
	at java.lang.Thread.run(Thread.java:745)
Caused by: java.net.ConnectException: Operation timed out (Connection timed out)
	at java.net.PlainSocketImpl.socketConnect(Native Method)
	at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
	at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
	at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
	at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
	at java.net.Socket.connect(Socket.java:589)
	at org.apache.http.conn.socket.PlainConnectionSocketFactory.connectSocket(PlainConnectionSocketFactory.java:75)
	at org.apache.http.impl.conn.DefaultHttpClientConnectionOperator.connect(DefaultHttpClientConnectionOperator.java:142)
	... 19 more
```

ii. Response #2:

The socket stayed open for a long time without any response being sent from the website and there was no socket.close() call done during that time, so it eventually timed out.

```
java.net.SocketException: Operation timed out (Read failed)
	at java.net.SocketInputStream.socketRead0(Native Method)
	at java.net.SocketInputStream.socketRead(SocketInputStream.java:116)
	at java.net.SocketInputStream.read(SocketInputStream.java:171)
	at java.net.SocketInputStream.read(SocketInputStream.java:141)
	at org.apache.http.impl.io.SessionInputBufferImpl.streamRead(SessionInputBufferImpl.java:137)
	at org.apache.http.impl.io.SessionInputBufferImpl.fillBuffer(SessionInputBufferImpl.java:153)
	at org.apache.http.impl.io.SessionInputBufferImpl.readLine(SessionInputBufferImpl.java:282)
	at org.apache.http.impl.conn.DefaultHttpResponseParser.parseHead(DefaultHttpResponseParser.java:138)
	at org.apache.http.impl.conn.DefaultHttpResponseParser.parseHead(DefaultHttpResponseParser.java:56)
	at org.apache.http.impl.io.AbstractMessageParser.parse(AbstractMessageParser.java:259)
	at org.apache.http.impl.DefaultBHttpClientConnection.receiveResponseHeader(DefaultBHttpClientConnection.java:163)
	at org.apache.http.impl.conn.CPoolProxy.receiveResponseHeader(CPoolProxy.java:165)
	at org.apache.http.protocol.HttpRequestExecutor.doReceiveResponse(HttpRequestExecutor.java:273)
	at org.apache.http.protocol.HttpRequestExecutor.execute(HttpRequestExecutor.java:125)
	at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:272)
	at org.apache.http.impl.execchain.ProtocolExec.execute(ProtocolExec.java:185)
	at org.apache.http.impl.execchain.RetryExec.execute(RetryExec.java:89)
	at org.apache.http.impl.execchain.RedirectExec.execute(RedirectExec.java:110)
	at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:185)
	at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:83)
	at org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl.executeRequest(HTTPHC4Impl.java:832)
	at org.apache.jmeter.protocol.http.sampler.HTTPHC4Impl.sample(HTTPHC4Impl.java:570)
	at org.apache.jmeter.protocol.http.sampler.HTTPSamplerProxy.sample(HTTPSamplerProxy.java:67)
	at org.apache.jmeter.protocol.http.sampler.HTTPSamplerBase.sample(HTTPSamplerBase.java:1231)
	at org.apache.jmeter.protocol.http.sampler.HTTPSamplerBase.sample(HTTPSamplerBase.java:1220)
	at org.apache.jmeter.threads.JMeterThread.doSampling(JMeterThread.java:622)
	at org.apache.jmeter.threads.JMeterThread.executeSamplePackage(JMeterThread.java:546)
	at org.apache.jmeter.threads.JMeterThread.processSampler(JMeterThread.java:486)
	at org.apache.jmeter.threads.JMeterThread.run(JMeterThread.java:253)
	at java.lang.Thread.run(Thread.java:745)
```

Conclusion

The website managed to handle around 330 users before it started breaking down and showing an increased error rate. The recorded error rate during that test was around 67% which is very high for a modern-day web application.