Praktyczna część seminarium pozwoli zapoznać się z plikami konfiguracyjnymi środowisk Maven i Gradle. W tym celu zostanie wykorzystany już istniejący projekt, dostępny właśnie na tym repozytorium. Jest to prosta aplikacja webowa wykorzystująca technologię Java Servlet oraz dwie biblioteki wykorzystywane do testów: JUnit i Mockito. Naszym zadaniem będzie inicjalizacja projektu dla obu środowisk.

Pierwszą rzeczą jaką należy zrobić, to ściągnięcie z repozytorium (branch master) pliku zawierającego projekt. Później kolejno dla tego projektu zostanie zainicjalizowany Maven, a następnie Gradle.

Pobranie przy pomocy Eclipse:
Window -> Show View -> Other… -> Git -> Git Repositories -> Clone a Git Repository

Następnie podajemy odpowiedni URL do repozytorium, wybieramy tylko branch master. Klikamy prawym przyciskiem na zainicjalizowane repozytorium i wybieramy opcję Import Projects… i zaznaczamy katalog do skopiowania.

1. Maven

Aby zainicjalizować projekt Maven klikamy prawym przyciskiem na katalog projektu i wybieramy: *Configure -> Convert to Maven Project*.

Wpisujemy informacje o artefakcie, czyli *Group Id*, *Artifact Id*, *Version* i *Packaging* (wybieramy war dla projektu web). Klikamy *Finish* aby zainicjalizować projekt Maven.

Po wykonaniu powyższych czynności pojawi się w katalogu projektu plik *pom.xml*. Ze zdefiniowanymi przez nas <*groupId*>, <*artifactId*>, <*version*>, <*packaging*>. Dodatkowo zostal dodany element <*build*> zawierajacy pewne informacje dotyczące wersji Javy i wskazujące na katalog *WebContent* - element <*warSourceDirectory*>. Pierwsza część zawierająca wersje <artifactId>maven-compiler-plugin</artifactId> usuwamy żeby została użyta domyślna (dostępna).

Kolejnym krokiem jest dodanie zależności, które zostały wykorzystane w projekcie aby możliwe było jego poprawne zbudowanie. W tym celu należy stworzyć element dokumentu XML <*dependencies*>, a w nim w blokach <*dependency*> umieszczamy kolejne potrzebne biblioteki wykorzystując ich <*groupId*>, <*artifactId*>, <*version*>. Poniżej zostały umieszczone zależności w postaci <*groupId*> : <*artifactId*> : <*version*>.

- javax.servlet : javax.servlet-api : 3.1.0
- junit : junit : 4.12
- org.mockito : mockito-core : 2.7.19

Podane wyżej biblioteki dostępne są w centralnym repozytorium Maven i właśnie stamtąd zostaną one pobrane. Pamiętajmy aby biblioteki testujące umieścić z odpowiednim tagiem, żeby nie zostały dodane do zbudowanego archiwum.

Po dodaniu zależności należy odświeżyć projekt Maven (jeżeli nie nastąpi to automatycznie po zapisie zmian w pliku *pom.xml*). Klikamy prawym na katalog projektu, następnie wybieramy *Maven -> Update Project...*

Po wprowadzonych zmianach możemy przetestować aplikację przez uruchamianie testów automatycznych przy pomocy Maven'a, co możemy zrobić poprzez kliknięcie prawym przyciskiem na katalog projektu i wybranie: *Run As -> Maven test*.

Aby zbudować projekt postępujemy w analogiczny sposób ale wybieramy opcję *Maven install.* W katalogu target zostanie wygenerowany plik z rozszerzeniem .war, możemy go wrzucić na IBM Cloud bądź uruchomić na lokalnym serwerze WAS Liberty lub Tomcat.

Po wykonaniu ćwiczenia należy dezaktywować środowisko Maven dla tego projektu, co można zrobić przez kliknięcie prawym przyciskiem na katalog projektu i następnie wybranie *Maven -> Disable Maven nature*.

2. Gradle

Uwaga: Gradle nie jest zainstalowany na komputerach ale jest na tyle sprytny, że nie musi być zainstalowany, żeby z niego korzystać ;) Wykorzystamy w tym celu IDE Eclipse, które pozwoli nam zainicjalizować już istniejący projekt (Maven) jako projekt Gradle. Będziemy mogli korzystać z Gradle'a dzięki takiemu narzędziu jak Gradle Wrapper. Wywołanie komendy *gradlew* w katalogu projektu pozwoli wykonywać zadania, buildy, uruchamiać testy i inne rzeczy oferowane przez Gradle.

Aby zainicjalizować projekt jako projekt Gradle musimy kliknąć prawym przyciskiem na projekt i wybrać: *Configure -> Add Gradle nature*, następnie wybieramy *Window -> Show View -> Other… -> Gradle Tasks -> build_setup -> init*. Odświeżamy projekt przez kliknięcie prawym przyciskiem myszy na katalog projektu i wybieramy *Gradle -> Refresh Gradle Project*.

Po wykonaniu powyższych czynności zostały utworzone następujące pliki

- *build.gradle*
- *settings.gradle*
- *gradlew*
- *gradlew.bat*
- *gradle/wrapper/gradle-wrapper.jar*
- *gradle/wrapper/gradle-wrapper.properties*

Tak jak było przedstawione na prezentacji sercem naszego projektu jest plik *build.gradle*. W nim należy zdefiniować zależności projektu oraz inne rzeczy takie jak używane pluginy, itp..

Gradle powinien być na tyle inteligentny żeby dodać automatycznie zależności ale jeśli tego nie zrobił możemy je dodać postępując w sposób przedstawiony poniżej.

> JEŚLI GRADLE POPRAWNIE DODAŁ ZALEŻNOŚCI MOŻNA POMINĄĆ TĄ SEKCJĘ <

Należy dodać centralne repozytoria (to co potrzebujemy powinno dodać się domyślnie przy tworzeniu pliku *build.gradle*) - w bloku *repositories* dodajemy *jcenter()* (czemu nie *mavenCentral()*? *jcenter()* ponoć szybsze i większe ale wszystkie potrzebne biblioteki w tych wersjach dostępne są zarówno na JCenter jak i na Maven Central Repository).

Kolejną rzeczą jaką należy zrobić jest dodanie zależności, w tym wypadku również możemy to zrobić na kilka sposobów ale przedstawiony zostanie tylko jeden. W sekcji *dependencies* dodajemy potrzebne trzy biblioteki. Pamiętajmy, że biblioteki testowe dodajemy ze specjalnym oznaczeniem. W tabeli poniżej zostało przedstawione dodawanie zarówno biblioteki aplikacji jak i testów. Wykorzystane zostało nazewnictwo takie jak dla Maven:

compile '<*groupId*>:<*artifactId*>:<*version*>'	compileTest '<*groupId*>:<*artifactId*>:<*version*>'

> KONIEC SEKCJI ZALEŻNOŚCI PROJEKTOWYCH W GRADLE <

Po dodaniu zależności aby zbudować plik war należy dodać plugin o takiej samej nazwie - 'war'. Aby dodać plugin możemy postępować następująco (lewa forma preferowana dla dużej ilości pluginów):

plugins {
id 'nazwa_naszego_pluginu'
}
	apply plugin: 'nazwa_naszego_pluginu'

Dodatkowo należy wskazać katalog *WebContent* przez podanie w pliku build.gradle: *project.webAppDirName = 'WebContent'*.

Po dodaniu zależności, pluginu oraz wskazaniu katalogu *WebContent* można uruchomić build. Czyli otwieramy okno *Gradle Tasks* i uchuchamiamy *build*. Po zbudowaniu projektu plik war powinniśmy znaleźć w katalogu gradle\build\libs. Żeby sprawdzić, że wszystko się zgadza można przenieść plik do katalogu C:\Programs\apache-tomcat-8.5.28\webapps i uruchomić serwer - C:\Programs\apache-tomcat-8.5.28\bin\strartup.bat.

Uruchomienie zadania *build* to nic innego jak uruchomienie predefiniowanego przez Gadle pewnego zadania. Uruchamia on jeden po drugim konkretne zadania, takie jak testy, stworzenie archiwum, itp.. Teraz zdefiniujemy swój własny kopiujący task. Jego zadaniem będzie skopiowanie wygenerowanego pliku archiwum .war na serwer Tomcat, czyli do katalogu C:\Programs\apache-tomcat-8.5.28\webapps.

```
task copyWarToTomcat(type: Copy) {
        description "Copies war archive into local Tomcat server"

        from "build/libs"
        into "C:/Programs/apache-tomcat-8.5.28/webapps"
```

```
	doFirst {
		println "Copying war into server..."
	}
	doLast {
		println “war file has been copied"
	}
}
```

Aby uruchomić task wchodzimy w Gradle Tasks i wybieramy nasz nowo zdefiniowany task (pewnie okazało się, że nie ma tam dostępnego tak nazwanego taska…) Jeśli nie pojawił się tam należy uruchomić terminal i ręcznie odpalić zadanie. Czyli wchodzimy *Window -> Show View -> Other… -> Terminal*. Otwieramy nowe okno terminalu (cmd). Możemy wyświetlić wszystkie dostępne taski poleceniem *gradlew task --all* aby skopiować wygenerowane archiwum war na serwer uruchamiamy task w następującym poleceniem *gradlew copyWarToTomcat*.

Możemy pójść dalej w procesie automatyzacji zadania, czyli uruchomić serwer Tomcat przy pomocy skryptu Gradle. Stwórzmy więc kolejny task (wykonywalny), którego zadaniem będzie automatyczne uruchomienie serwera Tomcat.

```
task startTomcat(type: Exec) {
	description 'Starts local Tomcat server'

	workingDir 'C:/Programs/apache-tomcat-8.5.28/bin'
	commandLine 'cmd', '/C', 'startup.bat'

	doFirst {
		println "Starting Tomcat..."
	}
	doLast {
		println 'Tomcat has been started.'
	}
}
```

Proszę sprawdzić, czy plik jest kopiowany i czy serwer się uruchamia po wykonaniu skryptów.

Kolejnym zadaniem będzie stworzenie taska zależnego. Będzie on łączył proces kopiowania i startowania serwera w jedno zadanie. Aby tego dokonać należy połączyć ze sobą oba wcześniej stworzone taski w następujący sposób:

```
task deployOnTomcat() {
	finalizedBy startTomcat
	dependsOn copyWarToTomcat
}
```

Możemy też zamknąć automatycznie serwer przy pomocy Gradle'a, czyli zdefiniujmy task, który uruchomi polecenie *shutdown.bat*.

```
task stopTomcat(type: Exec) {
	description 'Stops local Tomcat server'

	workingDir 'C:/Programs/apache-tomcat-8.5.28/bin'
	commandLine 'cmd', '/C', 'shutdown.bat'

	doFirst {
		println 'Stopping Tomcat...'
	}
	doLast {
		println 'Tomcat has been stopped'
	}
}
```