Hello!

Rafał Misiak

Java Developer dla Stibo Systems (DK) w Ciklum

slack: @rafalmisiak

rafalmisiak@gmail.com

JEE
Techniki Zaawansowane

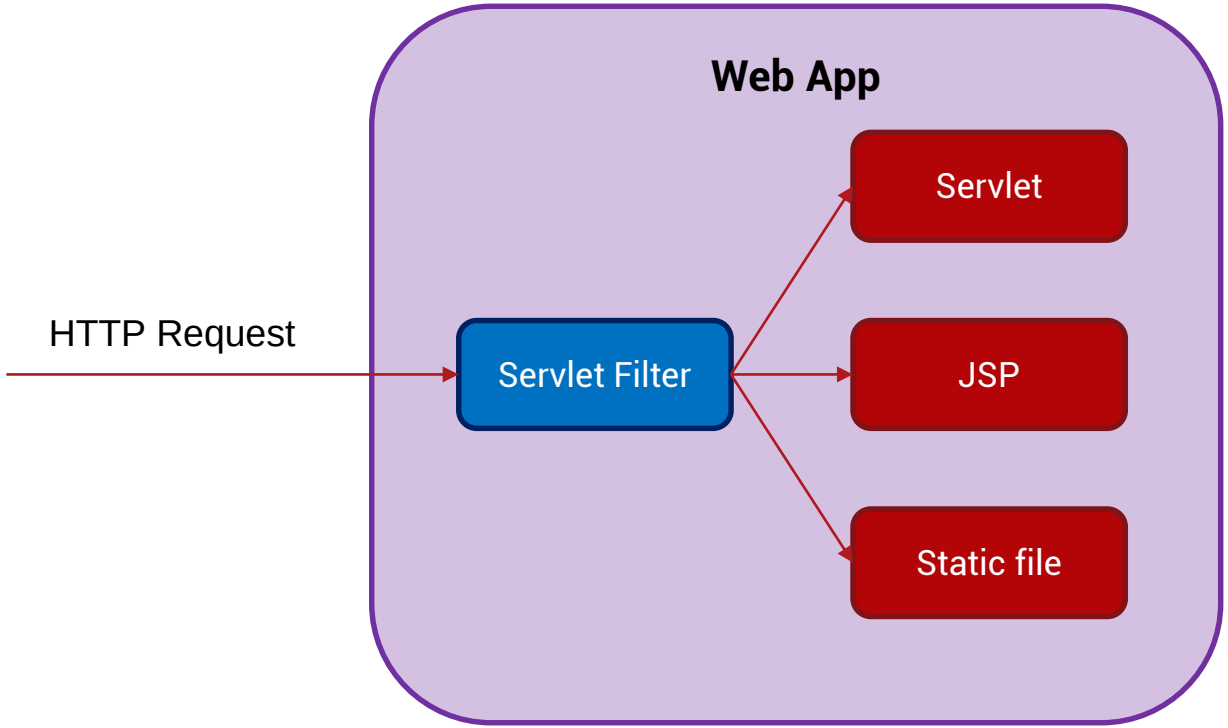
1.
Filters

Filtrujemy komunikację

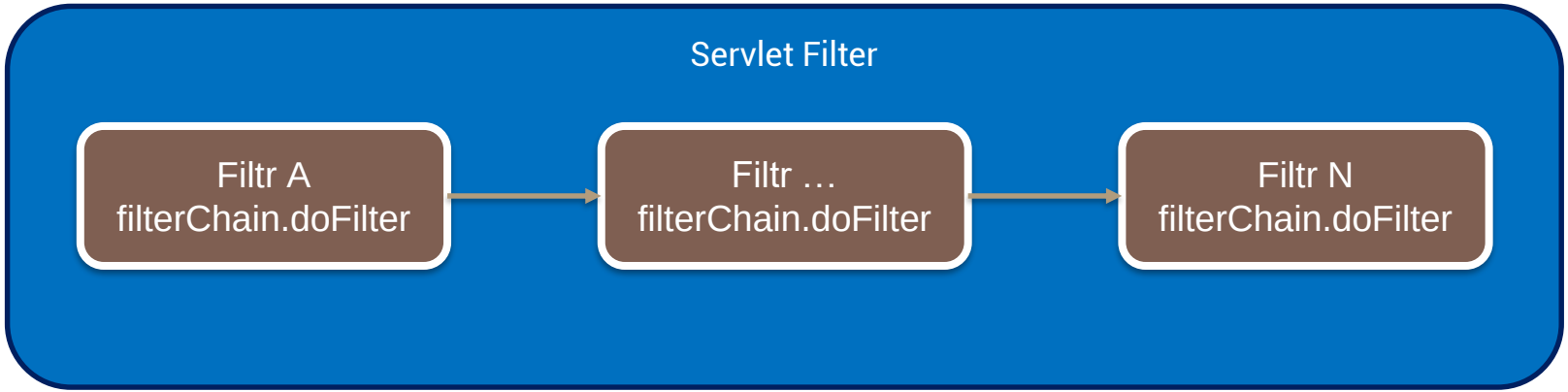
Filter
założenia

Filtry wykorzystywane w servletach oraz JSP są klasami używanymi do filtrowania i podejmowania akcji w komunikacji client-backend.

Filtry
założenia



Filtry
łańcuch wywołań



Filtry
przeznaczenie

Przykładowe/sugerowane przeznaczenie filtrów:

- Autentykacja
- Autoryzacja
- Szyfrowanie
- Kompresja danych
- Weryfikacja i modyfikacja danych zanim trafią do servletu

Filtr
Składnia

```
@WebFilter(
        filterName = "AuthenticationFilter",
        urlPatterns = {"/*"},
        initParams = {
                @WebInitParam(name = "allowedUser", value = "root")
        })
public class AuthenticationFilter implements Filter {

    …

}
```

Filtr
składowe

```
@Override
public void init(FilterConfig filterConfig) throws ServletException {

}

@Override
public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)
                                                                    throws IOException, ServletException {

}

@Override
public void destroy() {

}
```

Zadanie 1.1

- Napisz filtr **SalaryIncrementFilter**, który dla servletu **WelcomeUserServlet** będzie pobierał wartość parametru **salary** z **requestu**.
- Filtr powinien posiadać w konfiguracji parametr **minSalary** o wartości 100.
- Jeśli pobrana wartość będzie mniejsza niż **minSalary**, będzie ustawiał ją na wartość **minSalary** ustaloną w parametrze inicjowanym przez filtr.
- Nowa wartość powinna zostać zaprezentowana automatycznie w istniejącym widoku opartym na szablonie **welcome-user.ftlh**.

Przekierowanie requestu

Istnieje możliwość przekierowania użytkownika na inny widok, jeśli spełnione są pewne warunki.

Istnieje możliwość przekierowania przeładowując (tworząc nowy request) jak również obsłużyć użytkownika w ramach tego samego requestu.

Przekierowanie requestu

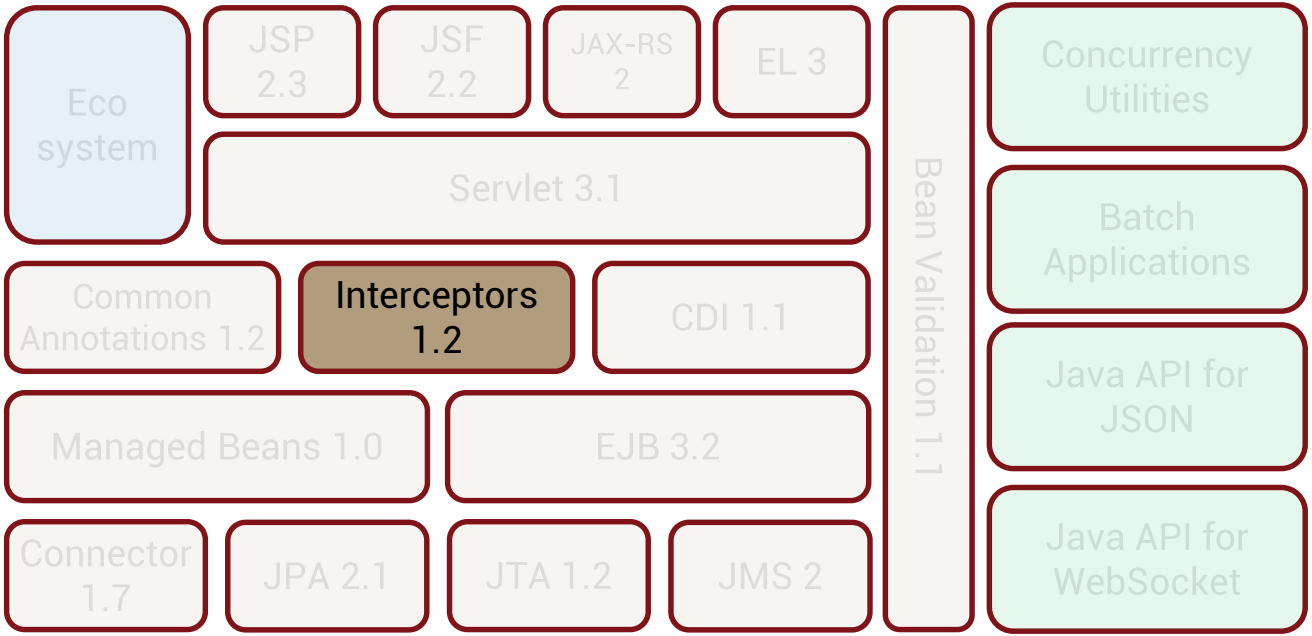
```
resp.sendRedirect("/redirec-to-view");
```

```
RequestDispatcher requestDispatcher = req.getRequestDispatcher("/to-view");
requestDispatcher.forward(req, resp);
```

2.
JEE: Interceptors

Oddzielamy niezależne funkcje

JEE 7


Eco system
JSP 2.3
JSF 2.2
JAX-RS 2
EL 3
Servlet 3.1
Common Annotations 1.2
Interceptors 1.2
CDI 1.1
Managed Beans 1.0
EJB 3.2
Connector 1.7
JPA 2.1
JTA 1.2
JMS 2
Bean Validation 1.1
Concurrency Utilities
Batch Applications
Java API for JSON
Java API for WebSocket

Interceptor
AOP

Aspect Oriented Programming – sposób tworzenia aplikacji polegający na jak najbardziej szczegółowym separowaniu elementów niezależnym względem siebie.

Interceptory w Javie realizują podejście AOP.

Interceptor
obserwator

Interceptor to swoisty **obserwator**. Przejmuje sterowanie zadaniem jak tylko zostanie wywołany w przypadku wywołania obserwowanego bytu (np. metody).

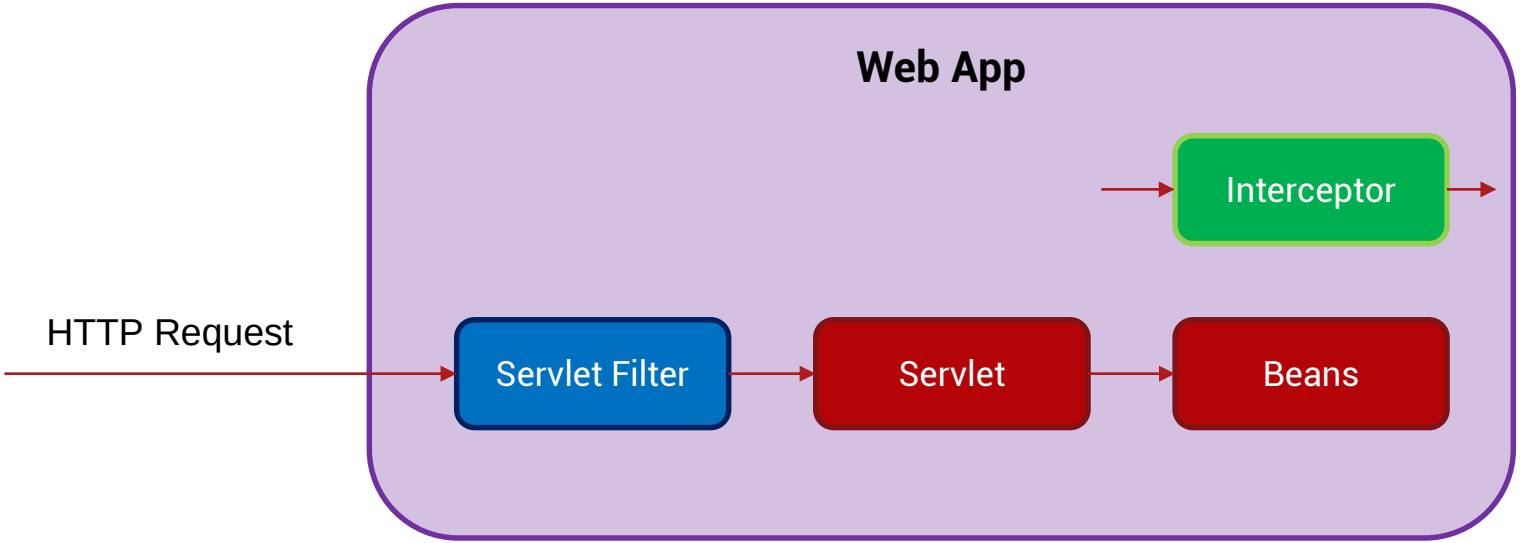
Interceptor
do jakich celów?

Interceptor ma zastosowanie zbliżone do zastosowania filtrów z tą różnicą, że nie jest związany z **ServletContext**!

Co to oznacza?

Możemy na przykład sprawdzić czy zalogowany użytkownik ma dostęp do EJB bez uwzględniania komunikacji servletowej w danej chwili.

Interceptor
założenia



Interceptor
syntax

```
public class AddUserInterceptor {

    Logger logger = Logger.getLogger(AddUserInterceptor.class.getName());

    @AroundInvoke
    public Object intercept(InvocationContext context) throws Exception {
        logger.info("Add user has been invoked!");
        return context.proceed();
    }
}
```

Interceptor
syntax

```
@Override
@Interceptors(AddUserInterceptor.class)
public void addUser(User user) {
    UsersRepository.getRepository().add(user);
}
```

Zadanie 2.1

- Napisz **interceptor** dla dodawania użytkownika, który będzie zgadywał płeć i ustawiał ją automatycznie.
- Załóż obsługę tylko polskich imion, gdzie imiona kończące się na literę „a" to imiona żeńskie, pozostałe to imiona męskie.
- Napisz drugi **interceptor**, który wykonywaną akcję dodania użytkownika będzie logował na poziomie INFO do logów aplikacji.

3.
Upload plików

Formularze w JEE podejście drugie

Upload plików

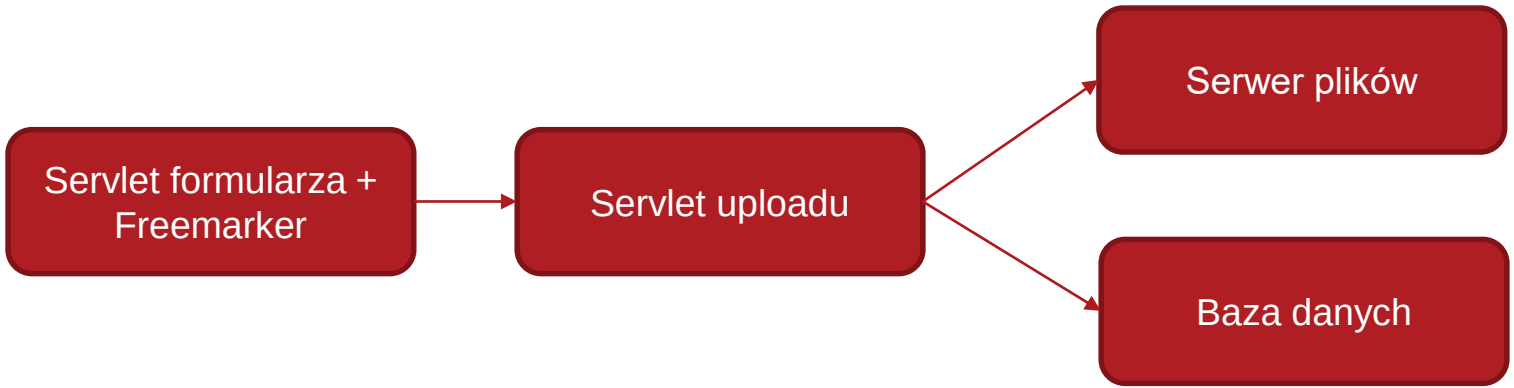
Oprócz zwykłych danych (tekstowych) istnieje możliwość wysłania (uploadu) plików na serwer.

Istnieje możliwość uploadu plików do przestrzeni zdeployowanej aplikacji.

Uwaga! W przypadku usunięcia deploymentu wszystkie dane (pliki) zostają skasowane.

Częsta forma uploadu

Bardzo często architektura uploadu uwzględnia niezależny serwer obrazków pod osobną wydzieloną domeną.



Upload
plików – dysk lokalny

Istnieje również możliwość uploadu plików i zapisanie ich w ramach lokalnego systemu plików (na dysku) poza przestrzenią aplikacji.

Strona www nie jest w stanie jednak wyświetlać bezpośrednio obrazu (udostępniać bezpośrednio pliku) gdyż jest to przypadek naruszenia zasad bezpieczeństwa!

Można stworzyć servlet pośredniczący w serwowaniu plików.

Upload plików
Formularz

Aby aktywować możliwość przesyłania plików na serwer musimy rozpocząć pracę od przygotowania odpowiedniego formularza.

Tag **<form>** musi mieć zdefiniowany atrybut:

```
enctype="multipart/form-data"
```

Oraz zawierać węzeł **<input>** typu:

```
type="file"
```

Zadanie 3.1

- Przekształć formularz dodawania i edycji użytkowników tak aby dodatkowo obsługiwał opcję dodawania plików. Pole opisz jako „Image:" i nazwij je „**image**"
- Utwórz katalog **/home/<user>/uploads** na dysku
- Utwórz plik **settings.properties** w katalogu **resources** aplikacji oraz dodaj do niego klucz **Upload.Path.Images** z wartością wskazującą na katalog **uploads** – pamiętaj aby ścieżka była **absolutna**

Zadanie 3.2

- Przygotuj nowy CDI Bean **FileUploadProcessor** – jaki zakres powinien być zdefiniowany dla tego beana?
- Stwórz w nim nową metodę, wraz z implementacją o sygnaturze **getUploadImageFilesPath()**
- Metoda ta powinna odczytywać plik properties oraz zwracać wartość wcześniej umieszczonego w niej klucza
- Utwórz pakiet **com.isa.usersengine.exceptions** i umieść w nim własny wyjątek **UserImageNotFound**

Obsługa plików
Servlet

Pliki wysłane przez formularz są tak zwanym elementem **Part** requestu.

Plików nie pobieramy za pomocą znanej metody getParameter(name) tylko getPart(name)

```
Part filePart = req.getPart("image");
```

Dodatkowo servlet musi obsługiwać ten typ formularzy:

```
@MultipartConfig
```

Zadanie 3.3
Upload - Bean

- We wcześniej przygotowanym beanie stwórz nową metodę o sygnaturze **uploadImageFile(Part filePart)** oraz zwracającą typ **File**.

- Do klasy **User** dodaj nowe pole **imageURL**. Zapewnij gettery i settery.

- Zintegruj użycie powyższej metody w servlecie **AddUserServlet** – ustaw wartość **imageURL** jako **„/images/" + file.getName()** gdzie file to plik zwrócony przez wywołanie **uploadImageFile(Part filePart)**

Zadanie 3.4
Serwowanie obrazów

- Stwórz nowy servlet **ImagesServlet**, który będzie serwował obrazki w kontekście **images**

4.
@Remote

Remote Interface Call

@Remote
idea

- Idea interfejsów zdalnych to możliwość wykonywania metod na naszym EJB spoza naszego modułu.

Zadanie 4.1

- Stwórz interfejs zdalny dla Repozytorium Użytkowników: **UsersRepositoryDaoRemote**
- Interfejs powinien zawierać tylko jedną metodę, zwracającą listę imion użytkowników: **getUsersNames**
- **UsersRepositoryDaoBean** powinien od tej pory implementować dwa interfejsy: **lokalny** i **zdalny**. Nie zapomnij o zapewnieniu implementacji zdefiniowanej metody

Zadanie 4.2

- Utwórz nowy projekt **Maven** (nie zamykaj dotychczasowego projektu JEE) o nazwie **artifactID:users-engine-standalone**
- W celu utworzenia projektu możesz wykorzystać archetyp: **org.apache.maven.archetypes:maven-archetype-quickstart**
- Zbuduj, uruchom, sprawdź czy aplikacja konsolowa działa poprawnie

@Remote
Biblioteka interfejsu zdalnego

- Aby było możliwe wywołanie zdalne interfejsu należy stworzyć bibliotekę zawierającą zdalny interfejs. Do tego celu tworzymy bibliotekę z interfejsem korzystając z pluginu: maven-ejb-plugin.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-ejb-plugin</artifactId>
    <version>3.0.0</version>
    <configuration>
        <ejbVersion>3.2</ejbVersion>
        <generateClient>true</generateClient>
        <clientIncludes>
            <clientInclude>com/isa/searchengine/dao/UsersRepositoryDaoRemote.class
            </clientInclude>
        </clientIncludes>
    </configuration>
</plugin>
```

Zadanie 6.3

- Stwórz bibliotekę, która będzie zawierała nowo utworzony interfejs komunikacji zdalnej.
- Zbuduj projekt wykorzystując dotychczasowe znane komendy:
  `mvn clean package`
- Uruchom wtyczkę **maven-ejb-plugin**
  `mvn ejb:ejb`
- Zainstaluj bibliotekę w lokalnym repozytorium:

```
mvn install:install-file -Dfile=users-engine-client.jar
-DgroupId=com.isa -DartifactId=users-engine-client
-Dpackaging=jar -Dversion=1.0
```

Zadanie 6.4

- Załącz nową bibliotekę users-engine-client jako zależność mavenową w nowym projekcie standalone

```
<dependency>
    <groupId>com.isa</groupId>
    <artifactId>users-engine-client</artifactId>
    <version>1.0</version>
</dependency>
```

Zadanie 6.5

- Zbuduj aplikację w taki sposób aby zawierała wewnątrz wszystkie załączone zależności mavenowe. Do tego celu służy kolejny plugin: **maven-assembly-plugin**.
- Zbuduj aplikację, zajrzyj do katalogu target
- Sprawdź czy wygenerowany artefakt zawiera zależności:
  `jar –tf <name.jar> | grep „isa"`

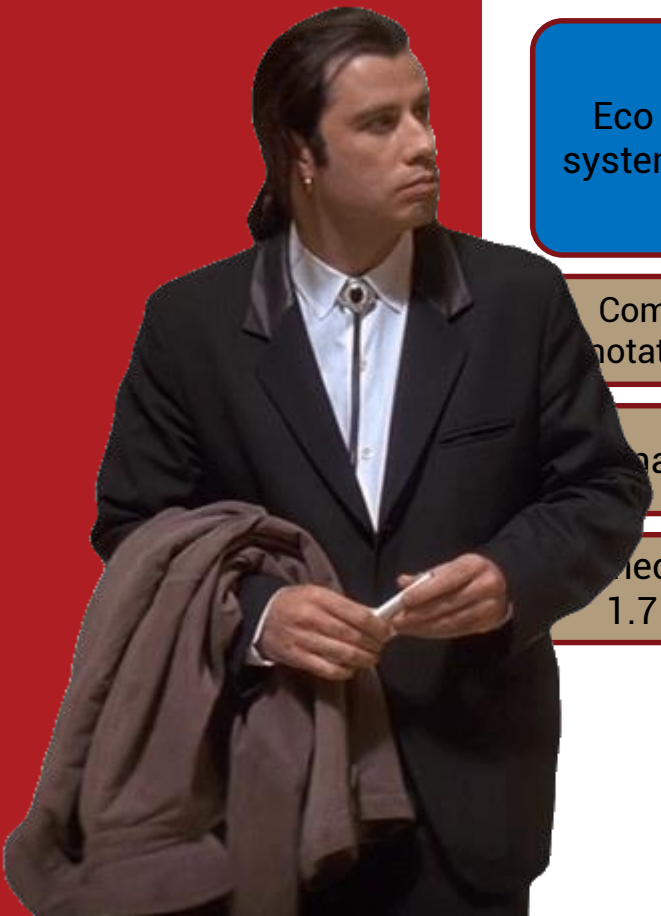
Zadanie 6.7

```
<plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.1.0</version>
    <configuration>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
                <classpathPrefix>libs/</classpathPrefix>
                <mainClass>
                    com.isa.App
                </mainClass>
            </manifest>
        </archive>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

5.
JNDI Lookup

Java Naming and Directory Interface

JEE 7



Eco system

JSP 2.3

JSF 2.2

JAX-RS 2

EL 3

Servlet 3.1

Common notations 1.2

Interceptors 1.2

CDI 1.1

naged Beans 1.0

EJB 3.2

lector 1.7

JPA 2.1

JTA 1.2

JMS 2

Bean Validation 1.1

Concurrency Utilities

Batch Applications

Java API for JSON

Java API for WebSocket

JNDI
Charakterystyka

- JNDI nie znajduje się w specyfikacji JEE ale jest mocno wykorzystywane dlatego kontenery aplikacyjne implementują to API
- JNDI służy do wyszukiwania obiektów po ich nazwach
- Wyszukiwanie odbywa się poprzez klasę **InitialContext** i metodę **lookup(String nazwa)**

Zadanie 7.1

- Przygotuj wymagane zależności w aplikacji standalone. Pamiętaj, że będziemy wykorzystywali implementację dostarczoną przez Wildfly

```
<dependency>
    <groupId>org.wildfly</groupId>
    <artifactId>wildfly-client-all</artifactId>
    <version>11.0.0.Final</version>
</dependency>
```

Zadanie 7.2

- Przygotuj konfigurację, która będzie spełniała wymagania swojej instancji serwera Wildfly

```
Hashtable<String, String> properties = new Hashtable<String, String>();
properties.put(Context.INITIAL_CONTEXT_FACTORY, "org.jboss.naming.remote.client.InitialContextFactory");
properties.put("jboss.naming.client.ejb.context", "true");
properties.put(Context.PROVIDER_URL, "http-remoting://localhost:8080");
properties.put(Context.SECURITY_PRINCIPAL, "<user>");
properties.put(Context.SECURITY_CREDENTIALS, "<password>");
Context context = new InitialContext(properties);
```

Zadanie 7.3

- Zaprezentuj w konsoli listę imion użytkowników wykorzystując nowy kontekst.
- JNDI NAME znajdziemy w konsolce WIldFly: Runtime -> Standalone Server -> Subsystems -> JNDI View -> View -> java:jboss/exported

```
UsersRepositoryDaoRemote generator =
        (UsersRepositoryDaoRemote)
                context.lookup(„<JNDI NAME>");
```

Thanks!!

Q?