

SOA – laboratorium nr 1

Temat: Wprowadzenie do tworzenia aplikacji biznesowych w technologii JavaEE.

Poznanie środowiska serwerów Aplikacyjnych

Cel laboratorium:

1. Zaznajomienie się z środowiskiem potrzebnym do tworzenia Aplikacji webowych tworzonych w technologii JavaEE.
2. Poznanie i konfiguracja wybranego serwera aplikacyjnego.
3. Stworzenie i uruchomienie prostej aplikacji webowej opartej na koncepcji servletów.

Wstęp.

Java EE (nazywana dawniej J2EE) wykorzystuje standardowy zestaw technologii do tworzenia aplikacji uruchamianych po stronie serwera. Technologia Javy EE zawiera servlety, Java Server Pages (JSP), Java Server Faces (JSF), Enterprise JavaBeans (EJB), Context Dependency Injection (CDI), Java messaging Service (JMS), Java Persistence API (JPA), Java API for XML Web Services (JAX-WS) i Java API for RESTful Web Services (JAX-RS), a także kilka innych.

Istnieje kilka serwerów aplikacji umożliwiających programistom uruchamianie aplikacji zgodnych z Javą EE, jedne są komercyjne, inne ze źródłami otwartymi. JBoss WildFly (dawna nazwa AS Application Server) to jedno z wiodących rozwiązań typu open source wykorzystywanych przez programistów. Choć trudno to dokładnie zmierzyć, jest wielce prawdopodobne, że stanowi on najczęściej wykorzystywany serwer aplikacyjny na rynku.

Inne popularne serwery aplikacyjne to Apache Tomcat oraz Glassfish.

Ćwiczenia

Pierwszym krokiem przed przystąpieniem do właściwej nauki serwera aplikacji jest instalacja na komputerze wszystkich elementów niezbędnych do uruchomienia tego serwera.

Serwer aplikacji wymaga tylko środowiska maszyny wirtualnej Javy.

W kwestii wymagań sprzętowych warto wiedzieć, że dystrybucja serwera wymaga jako minimum około 75 MB przestrzeni na dysku twardym i alokuje od 64 MB (minimum) do 512 MB pamięci RAM w przypadku serwera niezależnego.

Przygotowanie środowiska wymagać będzie realizacji pewnych kroków. Oto one:

- Instalacja Java Development Kit pozwalającego na uruchomienie serwera JBoss.
- Instalacja serwera JBoss - proponuje WildFly 10 lub wersje późniejsze
- Instalacja środowiska programistycznego Eclipse / Intelij / NetBeans (według własnych preferencji)

- Instalacja narzędzia Maven lub innego narzędzia do automatyzacji tworzenia oprogramowania.

Ćwiczenie 1. Jeśli korzystasz z komputera z laboratorium sprawdź czy wszystkie wymienione wcześniej komponenty są poprawnie zainstalowane. Jeśli czegoś brakuje – doinstaluj lub skonfiguruj.

Jeśli masz własny laptop – kolejno zainstaluj wymagane elementy (możliwe zamiana serwera JBoss na inny serwer Aplikacyjny podobnie jak i środowiska IDE).

Instalacja JBoss WildFly 10 - Serwer aplikacji JBoss można pobrać bezpłatnie z witryny społeczności pod adresem <http://wildfly.org/downloads>.

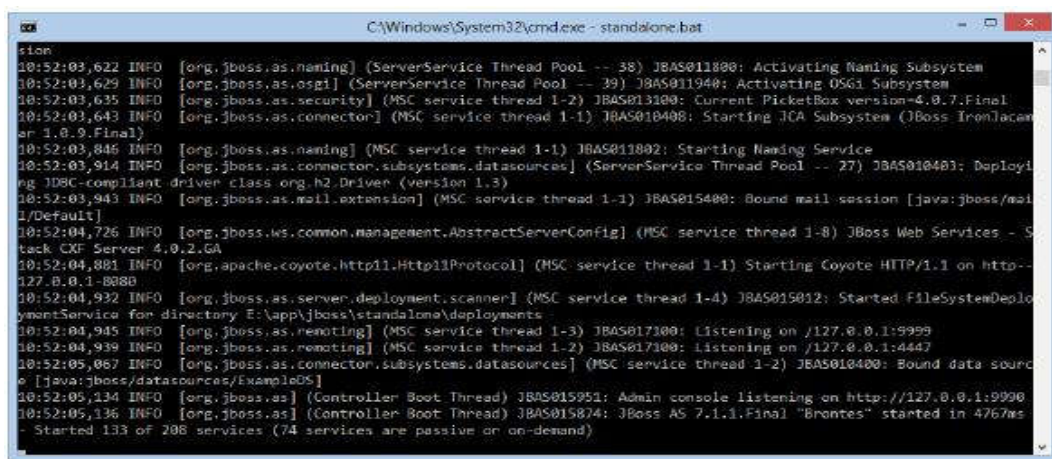
Wykonaj test instalacji JBoss.

Po instalacji serwera JBoss warto przeprowadzić prosty test, by sprawdzić, czy nie istnieją żadne poważne problemy z systemem operacyjnym lub dostępem do maszyny wirtualnej Javy. Aby przetestować instalację, trzeba przejść do folderu bin z instalacji JBoss i wykonać polecenie:

standalone.bat # Windows

\$ standalone.sh # Linux/Unix

Oto wygląd konsoli tuż po uruchomieniu serwera JBoss.



```

C:\Windows\System32\cmd.exe - standalone.bat
10:52:03,622 INFO [org.jboss.as.naming] (ServerService Thread Pool -- 38) JBAS011800: Activating Naming Subsystem
10:52:03,629 INFO [org.jboss.as.osgi] (ServerService Thread Pool -- 39) JBAS011940: Activating OSGi Subsystem
10:52:03,635 INFO [org.jboss.as.security] (MSC service thread 1-2) JBAS013100: Current PicketBox version=4.0.7.Final
10:52:03,643 INFO [org.jboss.as.connector] (MSC service thread 1-1) JBAS010408: Starting JCA Subsystem (JBoss IronJacar
ar 1.0.9.Final)
10:52:03,846 INFO [org.jboss.as.naming] (MSC service thread 1-1) JBAS011802: Starting Naming Service
10:52:03,914 INFO [org.jboss.as.connector.subsystems.datasources] (ServerService Thread Pool -- 27) JBAS010401: Deployi
ng JDBC-compliant driver class org.h2.Driver (version 1.3)
10:52:03,943 INFO [org.jboss.as.mail.extension] (MSC service thread 1-1) JBAS015400: Bound mail session [java:jboss/mail
1/Default]
10:52:04,726 INFO [org.jboss.ws.common.management.AbstractServerConfig] (MSC service thread 1-8) JBoss Web Services - S
tack CXF Server 4.0.2.GA
10:52:04,801 INFO [org.apache.coyote.http11.Http11Protocol] (MSC service thread 1-1) Starting Coyote HTTP/1.1 on http-
127.0.0.1:8080
10:52:04,932 INFO [org.jboss.as.server.deployment.scanner] (MSC service thread 1-4) JBAS015012: Started FileSystemDeplo
ymentService for directory E:\app\jboss\standalone\deployments
10:52:04,945 INFO [org.jboss.as.remoting] (MSC service thread 1-3) JBAS017100: Listening on /127.0.0.1:9999
10:52:04,939 INFO [org.jboss.as.remoting] (MSC service thread 1-2) JBAS017100: Listening on /127.0.0.1:4447
10:52:05,067 INFO [org.jboss.as.connector.subsystems.datasources] (MSC service thread 1-2) JBAS010400: Bound data source
e [java:jboss/datasources/ExampleDS]
10:52:05,134 INFO [org.jboss.as] (Controller Boot Thread) JBAS015951: Admin console listening on http://127.0.0.1:9990
10:52:05,136 INFO [org.jboss.as] (Controller Boot Thread) JBAS015874: JBoss AS 7.1.1.Final "Brontes" started in 4767ms
- Started 133 of 208 services (74 services are passive or on-demand)
  
```

Wskazane polecenie uruchamia niezależną instancję serwera JBoss, która jest równoważna uruchomieniu serwera aplikacji poleceniem run.bat (lub run.sh) we wcześniejszej wersji JBoss.

Jeśli konieczne jest dostosowanie właściwości początkowych serwera aplikacji, trzeba otworzyć plik standalone.conf (lub plik standalone.conf.bat w przypadku systemu Windows) i znaleźć fragment odpowiedzialny za wymagania pamięciowe. Oto odpowiedni fragment dla systemu Linux.

```

if [ "x$JAVA_OPTS" = "x" ]; then
    JAVA_OPTS="-Xms64m -Xmx512m -XX:MaxPermSize=256m -Dorg.jboss.resolver.
warning=true -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.
server.gcInterval=3600000"
fi
  
```

Domyślnie serwer używa minimalnie 64 MB pamięci operacyjnej na stos i maksymalnie 512 MB RAM. To odpowiednie wartości na początek. Jeśli jednak na serwerze mają być uruchamiane bardziej

rozbudowane aplikacje Javy EE, najprawdopodobniej potrzeba będzie minimum 1 GB pamięci, a czasem nawet 2 GB.

Warto sprawdzić, czy uruchomiony serwer jest dostępny przez interfejs sieciowy, wskazując w przeglądarce ogólnie znany domyślny adres strony początkowej serwera aplikacji: <http://localhost:8080>.

Jeśli wszystko jest w porządku powinien pokazać się w oknie przeglądarki następujący ekran:



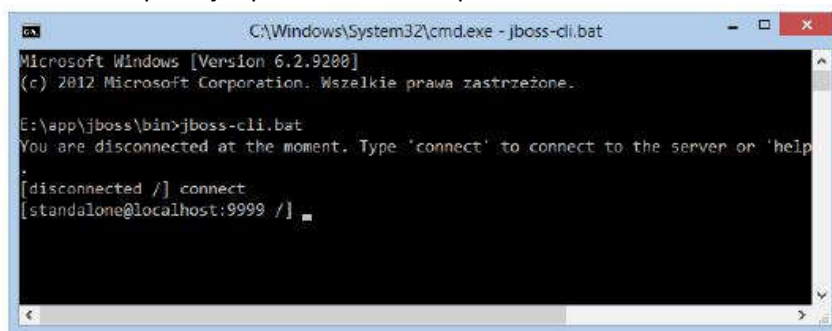
Ćwiczenie 2. Przetestuj zarządzanie serwerem z wiersza poleceń. Wykonaj zatrzymanie i ponowne uruchomienie serwera. Przetestuj za pomocą przeglądarki czy serwer jest aktywny czy nie.

Połączenie z serwerem przy użyciu wiersza poleceń.

Osoby korzystające z poprzednich wydań serwera aplikacji przypominają sobie zapewne narzędzie twiddle, które służyło do odpytywania ziaren zainstalowanych na serwerze aplikacji.

Narzędzie zostało zastąpione przez bardziej wyrafinowany interfejs nazwany Command Line Interface (CLI); jest on dostępny w folderze JBOSS_HOME/bin.

Wystarczy uruchomić skrypt jboss-cli.bat (lub jboss-cli.sh w systemie Linux) i można zarządzać serwerem aplikacji z poziomu wiersza poleceń.



```

C:\Windows\System32\cmd.exe - jboss-cli.bat
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. Wszelkie prawa zastrzeżone.

E:\app\jboss\bin>jboss-cli.bat
You are disconnected at the moment. Type 'connect' to connect to the server or 'help'.

[disconnected /] connect
[standalone@localhost:9999 /] _

```

Zatrzymanie serwera JBoss

Prawdopodobnie najprostszym sposobem zatrzymania serwera jest wysłanie odpowiedniego sygnału za pomocą kombinacji klawiszy Ctrl+C.

Jeśli jednak proces JBoss został uruchomiony w tle lub działa na innym komputerze, do jego zatrzymania można użyć interfejsu CLI i polecenia shutdown.

```
[disconnected /] connect
Connected to localhost:9999
[localhost:9999 /] :shutdown
```

Prosty skrypt zatrzymujący serwer

Istnieje jeszcze jeden sposób zatrzymania serwera, który bywa szczególnie przydatny, gdy zatrzymanie musi wykonać niezależny skrypt. Rozwiązanie to wymaga przekazania opcji --connect, co spowoduje wyłączenie trybu interaktywnego.

```
jboss-cli.bat --connect command=:shutdown # Windows
jboss-cli.sh --connect command=:shutdown # Unix/Linux
```

Zatrzymanie JBoss na zdalnym systemie

Zatrzymanie serwera aplikacji uruchomionego na zdalnym serwerze wymaga jedynie przekazania do CLI adresu serwera zdalnego, a także — ze względów bezpieczeństwa — nazwy użytkownika oraz hasła (tworzenie użytkowników zostało opisane w następnym rozdziale).

```
[disconnected /] connect 192.168.1.10
Authenticating against security realm: ManagementRealm
Username: admin1234
Password:
Connected to 192.168.1.10:9999
[192.168.1.10:9999 /] :shutdown
```

Ponowne uruchomienie JBoss

Interfejs wiersza poleceń zawiera wiele interesujących poleceń. Jednym z nich jest możliwość przeładowania konfiguracji Serwera Aplikacyjnego lub ich części za pomocą polecenia reload. Użycie polecenia w głównej ścieżce węzła serwera AS zapewnia przeładowanie pełnej konfiguracji usługi.

```
[disconnected /] connect
Connected to localhost:9999
[localhost:9999 /] :reload
```

Cwiczenie 3. Zapoznaj się z poniższym opisem mechanizmu zarządzania serwerem aplikacji. Spróbuj samodzielnie wykonać opisane zagadnienia.

Zarządzanie serwerem aplikacji

JBoss WildFly zapewnia trzy różne sposoby konfiguracji i zarządzania serwerami: są to

- interfejs webowy,
- klient wiersza poleceń
- zestaw plików konfiguracyjnych XML.

Niezależnie od wybranego sposobu edycji, cała konfiguracja jest zawsze synchronizowana pomiędzy poszczególnymi widokami i ostatecznie trafia do plików XML.

Zarządzanie JBoss WildFly przy użyciu interfejsu webowego

Interfejs webowy jest aplikacją GWT (Google Web Toolkit), która może służyć do zarządzania zarówno trybem samodzielnym, jak i trybem domenowym JBoss WildFly. Domyślnie jest uruchomiona w systemie lokalnym na porcie 9990, ale można to zmienić przy użyciu właściwości jboss.management.http.port znajdującej się w pliku konfiguracyjnym serwera (standalone.xml lub domain.xml).

```
<socket-binding-group name="standard-sockets" defaultinterface="public">
<socket-binding name="management-http" interface="management"
port="{${jboss.management.http.port:9990}}"/>
.....
</socket-binding-group>
```

Serwer JBoss jest domyślnie zabezpieczony. Domyślny mechanizm bezpieczeństwa korzysta z mechanizmu loginu i hasła. Powodem, dla którego serwer jest domyślnie zabezpieczony (wymaga uwierzytelnienia), jest możliwość przypadkowego udostępnienia interfejsu zarządzania pod publicznym adresem IP. Z tego samego powodu w dystrybucji nie stosuje się domyślnego użytkownika i hasła.

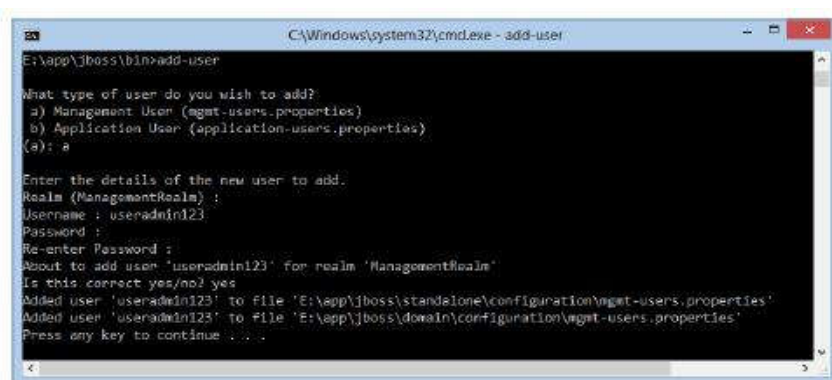
Dane użytkowników znajdują się w pliku właściwości o nazwie `mgmt-users.properties`, w folderach `standalone/configuration` lub `domain/configuration`, w zależności od trybu działania serwera. Plik zawiera informacje na temat nazwy użytkownika, wstępnie przygotowany skrót nazwy użytkownika, a także nazwę mechanizmu uwierzytelniania oraz hasło.

Do modyfikacji pliku i dodawania użytkowników służą narzędzia [add-user.sh](#) lub [add-user.bat](#), które poza samym dodaniem użytkowników generują również odpowiednie skróty. Trzeba uruchomić skrypt i postępować zgodnie ze wskazówkami na ekranie.

```
./add-user.sh
What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a):
```

Aby utworzyć nowego użytkownika, należy wprowadzić kilka wartości.

- **Typ użytkownika** (What type of user do you wish to add?) — wybieramy typ Management User, ponieważ użytkownik będzie zarządzał całym serwerem aplikacji.
- **Nazwa mechanizmu uwierzytelniania** (Realm) musi odpowiadać nazwie mechanizmu użytej w konfiguracji, więc jeśli w konfiguracji nie występuje inna nazwa, pozostawiamy domyślną nazwę **ManagementRealm**.
- **Nazwa użytkownika** (Username) — nazwa dodawanego użytkownika
- **Hasło** (Password) — hasło dla użytkownika.



Enter the details of the new user to add.

Realm (ManagementRealm) :

Username :

Password :

Re-enter Password :

Po walidacji wprowadzonych danych pojawi się prośba o ich potwierdzenie. Należy wpisać tekst yes; plik właściwości został uaktualniony.

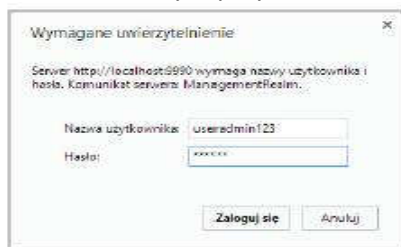
Jeżeli korzysta się z wydania nowszego niż 7.1.1, pojawi się dodatkowe pytanie (Is this new user going to be used for one AS process to connect to another AS process?), które dotyczy dodania podrzędnych kontrolerów hostów wykorzystujących główny kontroler domeny. Krok ten wymaga dodania tajnego klucza do konfiguracji hostów podrzędnych, by mogły się właściwie uwierzytelnić w momencie pobierania danych z głównego kontrolera domeny. (Więcej informacji na temat konfiguracji domeny znaleźć można na stronie dostępnej pod adresem <https://docs.jboss.org/author/display/WFLY10/Admin+Guide#AdminGuide-ManagedDomain>).

Uruchomienie konsoli webowej

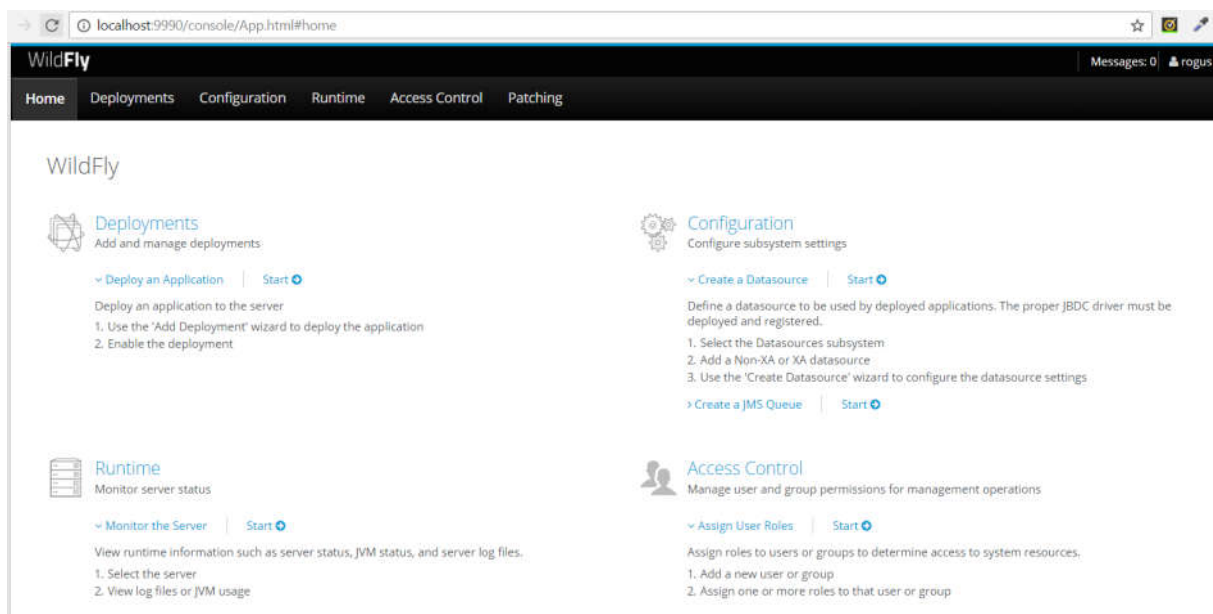
Po dodaniu przynajmniej jednego użytkownika można przystąpić do uruchomienia konsoli webowej dostępnej pod domyślnym adresem <http://<host>:9990/console>.

W naszym przypadku proszę wpisać : <http://localhost:9990/console>

Pojawi się prośba o podanie nazwy użytkownika oraz hasła. Wpisujemy w polach Nazwa użytkownika i Hasło dane użyte przy tworzeniu nowego użytkownika.



Po zalogowaniu nastąpi przekierowanie do głównego ekranu konsoli administracyjnej. Konsola działająca w trybie samodzielnym serwera wygląda tak jak na ekranie poniżej.



składa się z kilku głównych elementów - z których najważniejsze to Deployments, Configuration oraz Runtime.

Zakładka *Configuration* zawiera wszystkie pojedyncze podsystemy stanowiące część modułów serwera. Najważniejsze z nich to moduł do konfiguracji dostępu do baz danych oraz zarządzania kolejkami komunikatów.

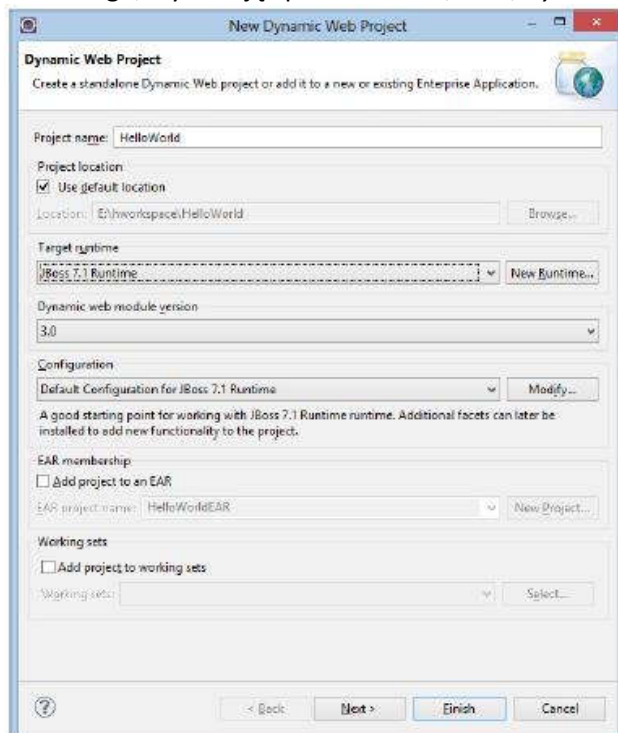
Zakładka *Deployments* pozwala dodać nowe pakiety typu ear lub war do katalogu deployment.

Zakładka *Runtime* pomaga w zarządzaniu wdrożeniami aplikacji i sprawdzaniu charakterystyki serwera.

Ćwiczenie 4. Zapoznaj się z zawartością poszczególnych zakładek.

Ćwiczenie 5. Na podstawie poniższego opisu przygotować i uruchomić pierwszą aplikację na serwerze JBoss AS.

Aby przetestować i uruchomić pierwszą aplikację, wykonamy najprostszą aplikację webową HelloWorld w IDE Eclipse. Uruchamiamy Eclipse i rozpoczynamy tworzenie nowego projektu webowego, wybierając polecenie *File/New/Dynamic Web Project*.



Teraz należy wpisać nazwę aplikacji i włączyć opcję Use default location, jeśli projekt ma powstać w tej samej lokalizacji, co przestrzeń robocza Eclipse. Jeśli nowy serwer JBoss WildFly został poprawnie skonfigurowany w IDE Eclipse, na liście rozwijanej Target Runtime powinna pojawić się opcja JBoss WildFly 10 Runtime. Jej wybór spowoduje automatycznie użycie Default Configuration for JBoss WildFly 10 Runtime na liście rozwijanej Configuration.

Jako Dynamic web module version wybieramy 3.0, co umożliwi łatwiejsze tworzenie kodu dzięki zastosowaniu specyfikacji serwetów w wersji 3.0. Pole wyboru w grupie EAR membership zostawiamy niezaznaczone.

Klikamy Finish, by kontynuować.

Dodajemy do projektu kwintesencję serwetu, czyli bardzo prostą klasę odpowiedzialną jedynie za wyświetlenie komunikatu typu „Witaj, świecie”. Z menu File wybieramy New/Servlet. Wpisujemy sensowną nazwę klasy i pakietu.

Kreator utworzy bardzo prosty szkielet servletu, który wystarczy zamienić na poniższy, minimalny fragment kodu.

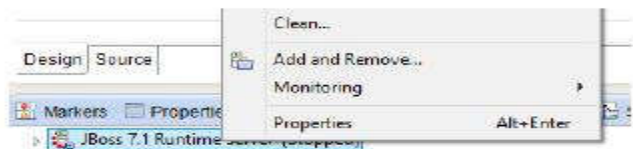
```
@WebServlet("/test")
public class TestServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Witaj, JBoss AS 7");
        out.close();
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
}
```

Warto zauważyć, że ta niewielka klasa zawiera adnotację `@WebServlet`, którą wprowadzono w API Servlet 3.0. Umożliwia ona rejestrację servletu bez użycia pliku konfiguracyjnego `web.xml`. W przedstawionym przykładzie użyliśmy własnej ścieżki URL `/test`. W przeciwnym razie użyta byłaby domyślna ścieżka odpowiadająca nazwie klasy.

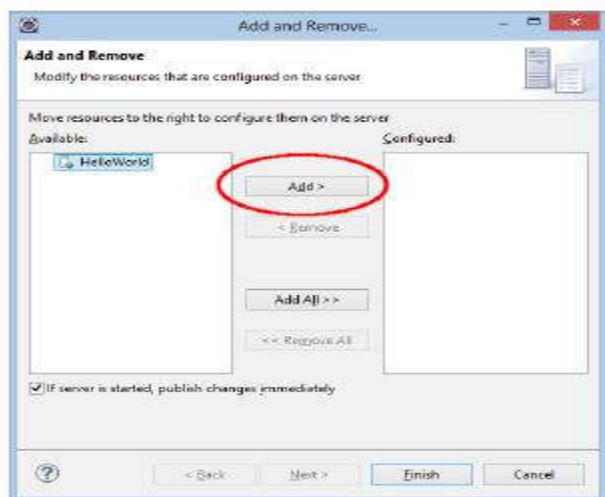
Aplikację trzeba uzupełnić plikiem deskryptora JBoss o nazwie `jboss-web.xml`. Choć nie jest to niezbędne, umożliwia on wskazanie korzenia kontekstu.

```
<jboss-web>
<context-root>/hello</context-root>
</jboss-web>
```

Dodajemy aplikację webową do listy wdrożonych zasobów, klikając zakładkę **Server** w Eclipse i wybierając z menu kontekstowego polecenie **Add and Remove**.



Zaznaczamy projekt i klikamy **Add**, aby dodać go do listy skonfigurowanych zasobów na serwerze.



Przy założeniu, że serwer JBoss WildFly został uruchomiony z poziomu Eclipse, zasób zostanie automatycznie wdrożony, jeśli włączona będzie opcja *If server is started, publish changes immediately*.

Jeśli jednak serwer został uruchomiony poza Eclipse, pełną publikację można wykonać poprzez kliknięcie zasobu prawym przyciskiem myszy i wybranie z menu kontekstowego opcji **Full Publish**.


```

        throws ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    out.println("<html><head><title>My Servlet</title></head><body>");
    out.println("Hello World!");
    out.println("</body></html>");

    out.close();
}
}

```

Każde żądanie HTTP ma swój typ, definiujący akcję którą serwer powinien wykonać w odpowiedzi. Podstawowe typy żądań to żądania GET i POST. Żądanie GET oznacza, że klient wysyłający żądanie do serwera chce pobrać zasób wskazywany przez URI przekazany wraz z tym żądaniem. Żądanie POST oznacza, że klient wysyła do serwera, a konkretnie do zasobu wskazanego przez URI dołączony do żądania, pewną porcję danych i zleca przetworzenie tych danych. W każdym żądaniu przesyłane są dodatkowo (opcjonalnie) parametry, nagłówki, oraz ciasteczka. Każdy taki element żądania możemy pobrać w kodzie Servletu i wykorzystać zgodnie ze swoją niczym nie skrepowaną wolą.

Implementując klasę servletu implementujemy de facto metody obsługujące żądania HTTP. Aby zaimplementować obsługę metody HTTP GET musimy zaimplementować metodę doGet(...). Aby zaimplementować obsługę metody HTTP POST musimy zaimplementować metodę doPost(...). W powyższym przykładzie zaimplementowaliśmy jedynie metodę doGet(...), co oznacza że servlet ten obsługuje tylko i wyłącznie żądania HTTP typu GET.

Metody doGet(...) i doPost(...) przyjmują dwa parametry wywołania – obiekty reprezentujące żądanie i odpowiedź HTTP. Obiekty te są tworzone i przekazywane do implementowanego przez nas servletu przez serwer aplikacji.

Parametr reprezentujący żądanie HTTP to obiekt typu javax.servlet.http.HttpServletRequest. Obiekt reprezentujący odpowiedź to obiekt typu javax.servlet.http.HttpServletResponse. Z instancji reprezentującej żądanie odczytujemy szczegóły żądania (parametry, ciasteczka, nagłówki itd.) a do bufora związanego z instancją reprezentującą odpowiedź zapisujemy to, co ma być odesłane do klienta, czyli elementy strony WWW (np. HTML). W jaki sposób zapisujemy tę odpowiedź widzimy na powyższym przykładzie.

Żądanie HTTP, zarówno typu GET jak i POST, określa przede wszystkim URL zasobu do którego wysyłamy żądanie, ale może dodatkowo zawierać pewną liczbę parametrów. Wysyłając żądanie GET klient ma możliwość przekazania parametrów poprzez dołączenie ich w odpowiedni sposób do URL. W przypadku żądania POST parametry przekazywane są w treści właściwej żądania (której nie widzimy – tworzy ją za nas przeglądarka WWW).

Ogólna postać URL dla żądania GET jest następująca: `http://{domena}:{port}/{zasób}?{parametry}`
 Zarówno {port} jak i {zasób} są opcjonalne. Opcjonalne są także {parametry}, ale jeśli chcemy je przekazać wraz z żądaniem, to doklejamy je do końca URL po znaku zapytania.
 Składnia definicji pojedynczego parametru jest następująca: `{nazwa}={wartość}`

Element {nazwa} to nazwa parametru a {wartość} to jego wartość. Kolejne definicje parametrów oddzielamy od siebie znakiem &. Przykładowo, aby przekazać 3 parametry napisalibyśmy:
`{nazwa}={wartość}&{nazwa}={wartość}&{nazwa}={wartość}`

Żądania typu POST służą do wysyłania formularzy, a parametry takich żądań to nic innego jak wartości wpisane w pola formularza. Przykładowy formularz poniżej:

```
<form method="POST" action="http://test.pl/kalkulator">
  x: <input type="text" name="paramX">
  y: <input type="text" name="paramY">
  <input type="submit" value="Dodaj">
</form>
```

Wyświetlając powyższy formularz w ramach strony HTML przeglądarka wyświetli dwa pola tekstowe oraz przycisk którego kliknięcie spowoduje wysłanie żądania POST. Żądanie to będzie zawierało dwa parametry: parametr o nazwie paramX oraz paramY. Wartości tych parametrów będą takie, jakie wpisujemy w pola formularza.

Niezależnie od tego czy parametry przekazano w jeden czy drugi sposób, wraz z żądaniem GET czy POST, odczytujemy je w kodzie servletu w ten sam sposób, tzn. za pomocą metody String `getParameter(String name)` obiektu reprezentującego żądanie.

Parametry są zawsze typu String, tak więc aby np. zaimplementować servlet sumujący dwie liczby przekazane za pomocą formularza jako parametry żądania musielibyśmy sparsować je, tak aby otrzymać zmienne typu int. W tym celu możemy posłużyć się metodą statyczną `int parseInt(String)` z klasy Integer. Poniżej przykładowa implementacja takiego servletu:

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = resp.getWriter();

    String paramX = req.getParameter("paramX");
    String paramY = req.getParameter("paramY");

    out.println("<html><body>");

    int sum = Integer.parseInt(paramX) + Integer.parseInt(paramY);

    out.println("Suma liczb x i y wynosi " + sum);
    out.println("</body><html>");

    out.close();
}
```

Jeśli chcemy aby nasz servlet obsługiwał także żądania wysyłane metodą GET to wystarczyłoby do powyższego dopisać:

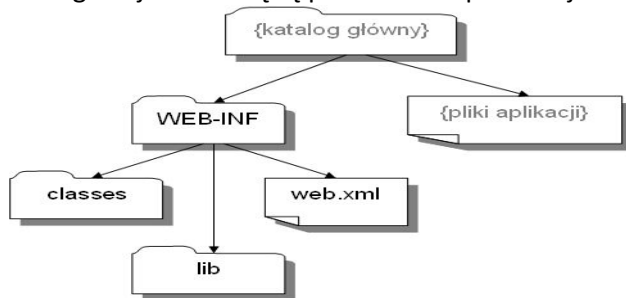
```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

    doPost(req, resp);
}
```

Tym samym przekazujemy obsługę żądań typu GET do metody `doPost(...)`. Jest to możliwe, jako że parametry, niezależnie od tego jakiego typu żądanie HTTP otrzymaliśmy, obsługujemy w ten sam sposób. Także w ten sam sposób generujemy stronę HTML która odsyłana jest jako odpowiedź na żądanie.

Typowa aplikacja WWW składa się z wielu różnego rodzaju komponentów: plików konfiguracyjnych, plików zasobów statycznych (np. HTML, JPG, CSS), plików JSP, katalogów i oczywiście plików zawierających skompilowane klasy, w tym servlety.

Poszczególne komponenty aplikacji WWW muszą być umieszczone w odpowiedniej strukturze katalogowej. Strukturę tą pokazano na poniższej ilustracji:



Element oznaczony jako *{katalog główny}* to katalog grupujący wszystkie elementy naszej aplikacji. Jego nazwa może być dowolna, jednak domyślnie nazwa ta często używana jest przez serwer aplikacji do określenia adresu URL pod jakim aplikacja ta jest serwowana.

W takim wypadku adresem URL naszej aplikacji będzie: **http://{adres serwera}/{katalog główny}**

Jeśli np. katalog główny naszej aplikacji nazwiemy *myApp* i aplikację zainstalujemy na serwerze aplikacji dostępnym pod adresem *http://test.pl:8080*, to aplikacja będzie dostępna pod adresem *http://test.pl:8080/myApp*.

Bardzo ważnym katalogiem obecnym w każdej aplikacji jest katalog o nazwie *WEB-INF*, który musi nazywać się dokładnie tak i musi być umieszczony bezpośrednio w katalogu głównym.

Katalog *WEB-INF* ma tę własność, że wszystkie umieszczone w nim pliki i katalogi są chronione przed bezpośrednim dostępem, tzn. nie jest możliwe pobranie żadnego z umieszczonych w nim plików przy pomocy żądania HTTP. Dla odmiany, wszystkie pliki umieszczone poza tym katalogiem serwer aplikacji zwróci w odpowiedzi na żądanie typu GET, jeśli do adresu URL aplikacji dodamy poprawną ścieżkę dostępu do tego pliku (względem katalogu głównego aplikacji).

Np. jeśli w katalogu głównym utworzymy katalog o nazwie *html* i w nim plik o nazwie *strona.html*, to będziemy mogli pobrać ten plik (i wyświetlić w przeglądarce) wysyłając żądanie na adres *{adres aplikacji}/html/strona.html*, przy czym *{adres aplikacji}* to np. *http://test.pl:8080/myApp*.

W katalogu *WEB-INF*, w podkatalogu o nazwie *classes* umieszczamy w odpowiedniej – odpowiadającej pakietom – strukturze katalogowej skompilowane klasy, w tym servlety. Jeśli do implementacji aplikacji używamy bibliotek spakowanych w archiwa JAR to biblioteki te umieszczamy w podkatalogu *lib*.

W katalogu *WEB-INF* umieszczamy główny plik konfiguracyjny aplikacji WWW, tzw. deskryptor wdrożenia. Jest to pokazany na powyższej ilustracji plik *web.xml*.

Deskryptor wdrożenia to główny plik konfiguracyjny aplikacji WWW. Jest to plik XML o nazwie *web.xml*, który umieszczamy bezpośrednio w katalogu *WEB-INF*.

W deskrypcie wdrożenia konfigurujemy między innymi adresy URL servletów. Wiemy już, z wcześniejszego opisu jak skonstruować URL dla zapytań, dla których w odpowiedzi odsyłane są pliki umieszczone bezpośrednio w katalogu głównym aplikacji, ale w jaki sposób wysłać żądanie HTTP do

servletu? Otóż to, które żądania zostaną obsługane przez servlet konfigurujemy właśnie w deskrytorze wdrożenia, tzn. w pliku web.xml. Poniżej przykład:

```
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"

  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <servlet>

    <servlet-name>HelloServlet</servlet-name>

    <servlet-class>pl.test.HelloWorldServlet</servlet-class>

  </servlet>

  <servlet-mapping>

    <servlet-name>HelloServlet</servlet-name>

    <url-pattern>/hello</url-pattern>

  </servlet-mapping>

</web-app>
```

Element o nazwie web-app to element nadrzędny, grupujący wszystkie elementy konfiguracji. Zawiera on niezbędne definicje w postaci atrybutów, w których znaczenie nie potrzebujemy póki co wnikać – z naszego punktu widzenia jest to element stały każdego pliku web.xml i zawsze wygląda on tak samo (dla specyfikacji servletów w wersji 2.5).

To co nas szczególnie interesuje, to element servlet oraz servlet-mapping.

Element servlet to definicja servletu; w pod-elementcie servlet-class podajemy pełną (poprzedzoną nazwą pakietu) nazwę klasy która implementuje servlet, zaś servlet-name to dowolna, wymyślona przez nas nazwa. Nazwa servletu którą wpisujemy jako wartość pod-elementu servlet-name służy w zasadzie tylko do tego, aby powiązać definicję servletu z definicją mapowania na URL. Element servlet-mapping to właśnie definicja wzorca adresów URL, dla których żądania będą obsługiwane przez powiązany servlet. Uwaga! Element servlet-name wiąże definicję servletu z definicją mapowania i jego wartość musi być w obydwu tych definicjach taka sama.

W powyższym przykładzie skonfigurowaliśmy aplikację WWW w ten sposób, że wszystkie żądania wysłane na adres URL postaci {adres aplikacji}/hello zostaną przekazane do servletu pl.test.HelloWorldServlet i w odpowiedzi na takie żądanie zostanie do klienta odesłana treść wygenerowana w tym właśnie servlecie.

W deskrytorze wdrożenia konfigurujemy także szereg innych parametrów aplikacji. W szczególności, możemy umieszczając w nim odpowiednią konfigurację zabezpieczyć aplikację przed nieautoryzowanym dostępem.

Ćwiczenie 7. Samodzielnie wykonaj przykłady opisane w Rozdział 3 z podręcznika

<https://docs.oracle.com/javaee/7/JEET.pdf>

lub

<https://javaee.github.io/tutorial/toc.html>

Ćwiczenie 8. Zarządzanie wdrożeniami z poziomu konsoli webowej. Wykorzystując servlet z ćwiczenia 5 uruchom go z poziomu konsoli webowej.

Wdrażanie aplikacji z poziomu Eclipse jest prostym zadaniem i najprawdopodobniej pozostanie domyślnym trybem w trakcie programowania aplikacji. W tym punkcie opisane zostanie użycie konsoli webowej do wdrożenia aplikacji, bo warto znać również ten sposób uruchamiania aplikacji.

Uruchamiamy konsolę webową i przechodzimy do zakładki Runtime. Z opcji po lewej stronie okna wybieramy Deployments/Manage Deployments.

W panelu głównym do zarządzania wdrożeniami służą przyciski Add, Remove, Enable, Disable i Update. Klikamy przycisk Add Content, aby dodać nową jednostkę wdrożenia. Na następnym ekranie wybieramy plik do wdrożenia (na przykład archiwum HelloWorld.war) z lokalnego systemu plików.



Kończymy pracę z kreatorem, weryfikując nazwę wdrożenia i klikając przycisk Save.

Wdrożenie pojawiło się na liście Deployments. Nie jest jednak domyślnie włączone. Klikamy przycisk Enable, aby włączyć wdrożenie aplikacji.



Ćwiczenie 9. Wdrażanie aplikacji przy użyciu narzędzia CLI. Wykorzystując servlet z ćwiczenia 5 uruchom go z poziomu linii komend.

Innym sposobem wdrożenia aplikacji jest użycie narzędzia CLI (Command Line Interface), które można uruchomić za pomocą pliku jboss-cli.bat (lub jboss-cli.sh w przypadku użytkowników systemu Linux).

Jak nietrudno się domyślić, do wdrożenia aplikacji niezbędne będzie użycie polecenia deploy.

Polecenie bez argumentów powoduje wyświetlenie listy aktualnie wdrożonych aplikacji.

```
[localhost:9999 /] deploy
```

```
ExampleApp.war
```

Jeśli pierwszym parametrem polecenia będzie archiwum z zasobami (na przykład lokalizacja pliku .war), zostanie ono od razu wdrożone na lokalnym, samodzielnym serwerze.

```
[standalone@localhost:9999 /] deploy ../HelloWorld.war
```

Jak można się domyślić z powyższego zapisu, narzędzie CLI używa standardowo folderu, w którym zostało uruchomione, czyli JBOSS_HOME/bin. Nic jednak nie stoi na przeszkodzie, by do wskazywania lokalizacji archiwów używać pełnych ścieżek; obsługa przez narzędzie funkcji automatycznego uzupełniania (klawisz Tab) dodatkowo ułatwia całe zadanie.

```
[standalone@localhost:9999 /] deploy c:\deployments\HelloWorld.war  
'HelloWorld.war' deployed successfully.
```

Domyślnie wdrożenie powoduje również aktywację aplikacji, więc jest ona od razu dostępna dla użytkowników. Jeśli chce się jedynie wdrożyć aplikację i aktywować ją później, konieczne jest przekazanie opcji --disabled.

```
[standalone@localhost:9999 /] deploy ../HelloWorld.war --disabled  
'HelloWorld.war' deployed successfully.
```

Aby uaktywnić już wdrożoną aplikację, ponownie trzeba użyć polecenia wdrożenia bez opcji --disabled, a zamiast lokalizacji archiwum podać nazwę wdrożenia.

```
[standalone@localhost:9999 /] deploy --name=HelloWorld.war  
'HelloWorld.war' deployed successfully.
```

Ponowne wdrożenie aplikacji wymaga użycia w poleceniu wdrożenia dodatkowego znacznika.

Używamy opcji -f, by wymusić ponowne wdrożenie.

```
[localhost:9999 /] deploy -f ../HelloWorld.war  
'HelloWorld.war' re-deployed successfully.
```

Usunięcie wdrożenia realizuje polecenie undeploy, które jako parametr przyjmuje nazwę aktualnie wdrożonej aplikacji.

```
[localhost:9999 /] undeploy HelloWorld.war  
'HelloWorld.war' undeployed successfully.
```