# I. PRZYGOTOWANIE ŚRODOWISKA PRACY

## 1. Instalacja JDK

Pobierz udostępniony pakiet JDK 1.8 https://drive.google.com/open?id=1Jvqqa0479jdqnWas0mXXwRgzIoABNANo (zawiera JRE) lub ściągnij jego najnowszą wersję ze strony https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html, instalacja jest standardowa.

## 2. Instalacja kontenera serwletów

Ze strony https://tomcat.apache.org/download-90.cgi pobierz:

**Binary Distributions**

- Core:
  - 32-bit/64-bit Windows Service Installer (pgp, sha512)

Podczas instalacji podajemy login i hasło użytkownika, który będzie administratorem. Zaraz po instalacji Tomcata pod adresem http://localhost:8080 powinna być dostępna jego aplikacja. Podczas pracy w Eclipse serwer musi być zatrzymany (trzeba kliknąć w zasobniku systemowym prawym przyciskiem myszki na ikonie Tomcata i wybrać *Stop service*), inaczej kiedy rozszerzenie WTP uruchomi serwer na własne potrzeby pojawi się błąd. WTP dodaje ze swej strony obsługę dla własnego środowiska wykonawczego serwera.

## 3. Instalacja bazy PostgreSQL

Należy ściągnąć stabilną wersję bazy PostgreSQL https://www.enterprisedb.com/downloads/postgres-postgresql-downloads (na komputerach laboratoryjnych zainstalowana jest wersja 9.5). Podczas instalacji w systemie Windows zakładany jest użytkownik o nazwie *postgres*, który uruchamia PostgreSQL-a jako serwis. Zakładany jest również wewnętrzny użytkownik baz danych o tej samej nazwie, który ma prawa administratora dotyczące dostępu do baz. Jako hasło dla tego użytkownika można wpisać *postgres*. Dla ułatwienia pracy warto dodać do zmiennej środowiskowej *Path* ścieżkę do katalogu *bin* z narzędziami PostgreSQL-a (przykładowa ścieżka `C:\Program Files\PostgreSQL\9.5\bin`). Pamiętaj, że dodawana ścieżka musi być oddzielona średnikiem od już istniejących wpisów w zmiennej *Path*.
W systemie Windows 10 podczas definiowania ścieżki do podkatalogu *bin* nie należy dodawać średnika (ze względu na sposób edycji zmiennej środowiskowej w tym systemie). Natomiast we wszystkich starszych systemach trzeba podczas wpisywania oddzielić ścieżkę średnikiem od już istniejących wpisów. Po każdej modyfikacji zmiennej środowiskowej zalecane jest ponownie uruchomienie komputera. Ścieżka wskazującą na podkatalog *bin* PostgreSQL-a została poprawnie dodana do zmiennej środowiskowej *Path*, jeżeli komenda *psql* jest rozpoznawana w windowsowym *Wierszu polecenia*.

## 4. Instalacja Eclipse

Należy ściągnąć (https://www.eclipse.org/downloads/packages/) i rozpakować najnowszą wersję *Eclipse IDE for Enterprise Java Developers*. W tej wersji jest już zainstalowane rozszerzenie WTP (Web Tools Platform).

# II. PRACA W ŚRODOWISKU ECLIPSE

## 1. Po pierwszym uruchomieniu środowiska Eclipse

**Dodanie środowiska wykonawczego serwera**
***Window*** `->` ***Preferences*** `->` należy rozwinąć gałąź ***Server*** `->` kliknąć ***Runtime Environments*** nacisnąć przycisk ***Add*** `->` kliknąć na ikonkę folderu o nazwie ***Apache*** i wybrać z listy odpowiednią wersję np. *Apache Tomcat v9.0*. Następnie kliknąć ***Next*** `->` ***Browse***, aby wskazać katalog instalacyjny Tomcata np. `C:\Program Files\Apache Software Foundation\Tomcat 9.0` `->` ***Finish***

**Ustawienie kodowania znaków dla stron JSP**
***Window*** `->` ***Preferences*** `->` ***Web*** `->` ***JSP File*** z listy ***Encoding*** wybrać *ISO 10646/Unicode(**UTF-8**)*.

Eclipse zostało rozbudowane o kontener serwletów, więc można przystąpić do tworzenia pierwszego projektu aplikacji WWW.

## 2. Tworzenie projektu WWW

*(Uwaga: poniższy opis dotyczy perspektywy Java EE)*
***File*** `->` ***New*** `->` ***Other*** `->` rozwinąć ***Web*** `->` wybrać ***Dynamic Web Project*** `->` ***Next***. Podać nazwę projektu np. *project-web-app*, pozostawiamy ustawienia domyślne. Należy zwrócić uwagę czy *Target Runtime* jest ustawione na *Tomcat*, w *Configuration*, gdzie zmienia się aspekty (*facets*) projektu zostawić *default*. Kliknąć ***Next***, pozostawić domyślne ustawienia w *default source* i *output folder* i kliknąć ponownie ***Next***. W ***WebModule*** zaznaczyć checkbox ***Generate web.xml deployment descriptor*** i kliknąć ***Finish***. W utworzonym projekcie znajduje się katalog *WebContent* z określoną strukturą aplikacji sieciowej.

## 3. Biblioteki

Przekopiuj z folderu *programowanie-zwinne-cw1\lib* wszystkie udostępnione biblioteki do podkatalogu */WebContent/WEB-INF/lib* utworzonego w Eclipse projektu aplikacji webowej.

## 4. Pierwszy serwlet

**4.1.** Na ikonie projektu trzeba kliknąć prawym przyciskiem myszki i wybrać ***New*** `->` ***Servlet***, wpisać nazwę pakietu np. *com.project* oraz nazwę serwletu np. *TaskServlet* `->` 2x ***Next***. Sprawdź czy są zaznaczone metody *doGet* i *doPost* (*Constuctors from super class* i *Inherited abstract methods* są ustawione domyślnie) `->` ***Finish***.
W `Java Resources/src/com.project/` pojawi się utworzony plik *TaskServlet.java*.

**4.2.** Utwórz metodę *processRequest*, w metodach *doPost* i *doGet* wpisz jej wywołania.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {

}
```

**4.3.** Zdefiniuj zmienną

```
private static final String CONTENT_TYPE = "text/html;charset=UTF-8";
```

**4.4.** W metodzie *processRequest* wpissz:

```
//Ustawiamy typ i kodowanie generowanej strony
response.setContentType(CONTENT_TYPE);
PrintWriter out = response.getWriter();
try{
      //Drukowanie strony HTML
      outHeader(out, "Lab");
      out.println("<h2>Hello World!</h2>");
```

```
        outFooter(out);
    }finally{
        out.close();
    }
```

**4.5.** Aby uruchomić serwlet trzeba jeszcze zaimplementować dwie pomocnicze metody *outHeader* i *outFooter*.

```
    public static void outHeader(PrintWriter out, String title){
        String str = "<html><head>" +
                     "<meta http-equiv=\"Content-Type\" content=\"" + CONTENT_TYPE + "\">" +
                     "<title>" + title + "</title>" +
                     "</head><body>";
        out.print(str);
    }

    public static void outFooter(PrintWriter out){
        out.print("</body></html>");
    }
```

**4.6.** Uruchom klikając prawym przyciskiem myszki na ikonę serwletu i wybierz ***Run As*** -> ***Run On Server***.

## 5. Pierwsza strona JSP

**5.1.** Przejdź do perspektywy ***Java EE*** -> ustaw kursor na katalogu ***WebContent***, nacisnij prawy przycisk myszy i wybierz ***New*** -> ***JSP***, podaj nazwę pliku np. *task_page.jsp* -> ***Next*** -> wybierz szablon np. ***New JSP File (HTML)*** -> ***Finish***. Kreator utworzy nowy plik i otworzy go w edytorze JSP.

**5.2.** Ustaw kursor w sekcji body pliku i nacisnąć kilka razy CTRL + SPACJA, aby podejrzeć podpowiedzi edytora. Ten sam plik można również edytować w *Web Page Editor* (prawy przycisk myszy na pliku -> ***Open With*** -> ***Web Page Editor***).

**5.3.** Dodajemy wyrażenie JSP

```
<%="Hello World!"%><br/>
Data: <%=new SimpleDateFormat("dd-MM-yyyy").format(new Date())%><br/>
ID Studenta: <%=request.getParameter("x_student_id")%><br/>
```

Podczas wpisywania *java.text.SimpleDateFormat* i *java.util.Date* powinny zostać dodane automatycznie importy (atrybuty import są umieszczane wraz z dyrektywą *page*):

```
<%@ page import="java.text.SimpleDateFormat"%>
<%@ page import="java.util.Date"%>
```

**5.4.** Kliknij na pliku JSP prawym przyciskiem myszki i wybierz ***Run As*** -> ***Run On Server***.

Przed uruchomieniem należy zapisać zmiany i sprawdzić czy w menu *Project* jest zaznaczone *Build Automatically*, jeżeli nie to zaznaczyć lub wcisnąć *Build All*.

**5.5.** Po uruchomieniu strony dodaj do adresu parametr x_student_id np. ?x_student_id=12 i naciśnij *Enter*. (http://localhost:8080/project-web-app/task_page.jsp?x_student_id=12)

## 6. Przekierowanie z serwletu do strony JSP

**6.1.** Utwórz serwlet *TaskRedirect* z dodatkową metodą *processRequest*, której wywołania należy umieścić w metodach *doGet* i *doPost*.

W klasie serwletu dodaj zmienną

```
    private static final String CONTENT_TYPE = "text/html;charset=UTF-8";
```

natomiast w metodzie *processRequest* umieść poniższy fragment:

```
response.setContentType(CONTENT_TYPE);
PrintWriter out = response.getWriter();
try{
   request.setAttribute("x_redirect","Jan Kowalski jest zalogowany!");

   ServletContext context = getServletContext();
   RequestDispatcher dispatcher = context.getRequestDispatcher("/task_page.jsp");
   dispatcher.forward(request, response);
}finally{
   out.close();
}
```

**6.2.** W *task_page.jsp* dodaj poniższą linijkę i uruchom serwlet *TaskRedirect*.

```
Informacja przekierowana z serwletu: <%= request.getAttribute("x_redirect")%>
```

## 7. Podłączenie do bazy danych

**7.1.** Utwórz bazę danych o nazwie *projekty*. Możesz skorzystać z instalowanego wraz z bazą PostgreSQL programu *PgAdmin* lub wpisać w windowsowym *Wierszu polecenia* (użycie konsoli wymaga, aby w zmiennej środowiskowej *Path* była dodana ścieżka do katalogu z narzędziami PostgreSQL-a):

```
createdb --username=postgres projekty
```

(Zamiast *--username=postgres* można używać *-U postgres*, a także pomijać wpisywanie użytkownika i jego hasła, jeżeli zdefiniujesz zmienne systemowe PGUSER i PGPASSWORD. Pamiętaj, że w środowisku produkcyjnym korzystanie z takiego rozwiązania jest niedopuszczalne.)

**7.2.** Utwórz plik *context.xml* z poniższą treścią definiującą parametry połączenia z bazą danych. Umieścić plik w podkatalogu *<projekt>/WebContent/META-INF*

```
<?xml version="1.0" encoding="UTF-8"?>
  <Context path="/project-web-app">
       <Resource
              name="jdbc/postgres"
              auth="Container"
              type="javax.sql.DataSource"
              driverClassName="org.postgresql.Driver"
              url="jdbc:postgresql://localhost:5432/projekty"
              username="postgres"
              password="postgres"
              maxActive="20"
              maxIdle="10"
              maxWait="10000"
              removeAbandoned="true"
              removeAbandonedTimeout="60"
              logAbandoned="true"
       />
  </Context>
```

**7.3.** W deskryptorze wdrożenia (*<projekt>/WebContent/WEB-INF/web.xml*, w bloku `<web-app>...</web-app>`) dodaj poniższy fragment

```
  <resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/postgres</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

## 8. Framework Hibernate

**8.1.** W katalogu *<projekt>/src* projektu utwórz podkatalog *META-INF*. Następnie umieścić w nim plik konfiguracyjny *persistence.xml*.

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
		http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
	version="2.2">
	<persistence-unit name="psqlManager"
		transaction-type="RESOURCE_LOCAL">
		<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
		<non-jta-data-source>java:/comp/env/jdbc/postgres</non-jta-data-source>
		<properties>
			<property name="hibernate.connection.datasource"
				value="java:/comp/env/jdbc/postgres" />
			<property name="hibernate.dialect"
				value="org.hibernate.dialect.PostgreSQL95Dialect" />
			<property name="hibernate.hbm2ddl.auto" value="update" />
			<property name="hibernate.show_sql" value="true" />
		</properties>
	</persistence-unit>
</persistence>
```

W pliku deklarujemy użycie utworzonego wcześniej w *<projekt>/WebContent/META-INF/context.xml* `DataSource` o nazwie *jdbc/postgres*, które definiuje parametry połączenia z bazą danych.
Użycie `<property name="hibernate.hbm2ddl.auto" value="update" />` powoduje automatyczne tworzenie struktury bazy danych na podstawie utworzonych klas encyjnych (dostępne są też opcje *create*, *create-drop*, *validate* i *none*).

**8.2.** Utwórz pakiet *com.project.util* i dodaj do niego klasę *HibernateUtil*. Klasa ta jest singletonem – prostym wzorcem projektowym, który pozwala na utworzenie tylko jednej instancji obiektu.

```
package com.project.util;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class HibernateUtil {
	private static final String DATA_SOURCE = "psqlManager";

	private EntityManagerFactory entityManagerFactory;

	private HibernateUtil() {
	}
```

```
	private static class Inner {
		private static final HibernateUtil INSTANCE = new HibernateUtil();
	}

	public static HibernateUtil getInstance() {
		return Inner.INSTANCE;
	}

	public EntityManagerFactory createEntityManagerFactory() {
		if (entityManagerFactory == null) {
			entityManagerFactory = Persistence.createEntityManagerFactory(DATA_SOURCE);
		}
		return entityManagerFactory;
	}

	public EntityManager createEntityManager() {
		return this.createEntityManagerFactory().createEntityManager();
	}

	public void closeEntityManagerFactory() {
		if (entityManagerFactory != null) {
			entityManagerFactory.close();
		}
	}
}
```

**8.3.** Dodaj pakiet *com.project.listeners* i utwórz w nim klasę *ApplicationContextListener*

```
package com.project.listeners;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import com.project.util.HibernateUtil;

@WebListener
public class ApplicationContextListener implements ServletContextListener {

	@Override
	public void contextDestroyed(ServletContextEvent sce) {
		HibernateUtil.getInstance().closeEntityManagerFactory();
	}

	@Override
	public void contextInitialized(ServletContextEvent sce) {
		HibernateUtil.getInstance().createEntityManagerFactory();
	}
}
```

Powyższy fragment deklaruje użycie odbiorcy zdarzeń, którego metody będą wywoływane w momencie uruchamiania i zatrzymywania aplikacji webowej.