Java Enterprise Edition

Gabriele Tolomei

DAIS – Università Ca' Foscari Venezia

Programma del Corso

- 09/01 – Introduzione
- 10/01 – Java Servlets
- 16-17/01 – JavaServer Pages (JSP)
- 23-24/01 – Lab: Applicazione “AffableBean”
- 30-31/01 – Enterprise JavaBeans (EJB) + Lab

AffableBean

UNA VERA APPLICAZIONE JAVA EE

Scenario: Introduzione

- AffableBean è il nome di un piccolo negozio di generi alimentari
- Collabora con agricoltori locali per fornire prodotti di coltura biologica
- Da un sondaggio effettuato sui clienti è risultato che il 65% di essi sarebbe interessato ad un servizio di acquisto/consegna online

Scenario: Obiettivo

- In quanto sviluppatori Java web, vi viene richiesto di realizzare:
  1. l'applicazione web che consenta l'acquisto online ai clienti di AffableBean
  2. una console amministrativa (sempre tramite interfaccia web) che consenta allo staff di AffableBean di tenere traccia degli ordini
  3. Implementazione del modello MVC (Model-View-Controller)

Scenario: Supporto Linguistico

- Il negozio si trova in Italia, ma vista la posizione turistica, serve molti clienti stranieri con cui si interfaccia in inglese
- L'applicazione web dovrà pertanto supportare entrambe le lingue: italiano e inglese

Scenario: Ambiente

- L’ambiente previsto su cui verrà eseguita l’applicazione in fase di rilascio è:
  - Java EE server → JBoss AS 5.1
  - RDBMS → MySQL

Requisiti: Categorie Prodotti

- Rappresentazione “online” dei beni e dei prodotti che sono venduti fisicamente dal negozio
- **4 categorie** di prodotto:
  - Latticini
  - Carni
  - Panetteria
  - Frutta e Verdura
- Ciascuna categoria contiene **4** prodotti
- Per ciascun prodotto (16 in totale) occorre specificare alcuni dettagli, come: **nome**, **immagine**, **descrizione** e **prezzo**

Requisiti: Carrello della Spesa

- Occorre implementare le seguenti funzioni:
  - Aggiungere/Rimuovere prodotti al/dal carrello
  - Aggiornare la quantità di prodotti nel carrello
  - Visualizzare un sommario di tutti i prodotti e le relative quantità presenti nel carrello
  - Eseguire un ordine di un prodotto e procedere al pagamento tramite un servizio “sicuro”

Requisiti: Console Admin

- La console di amministrazione dell'applicazione è ad uso esclusivo del personale di AffableBean
- Consente di visualizzare gli ordini eseguiti dai clienti
- Utilizza un'interfaccia web via browser

Requisiti: Livelli di Sicurezza

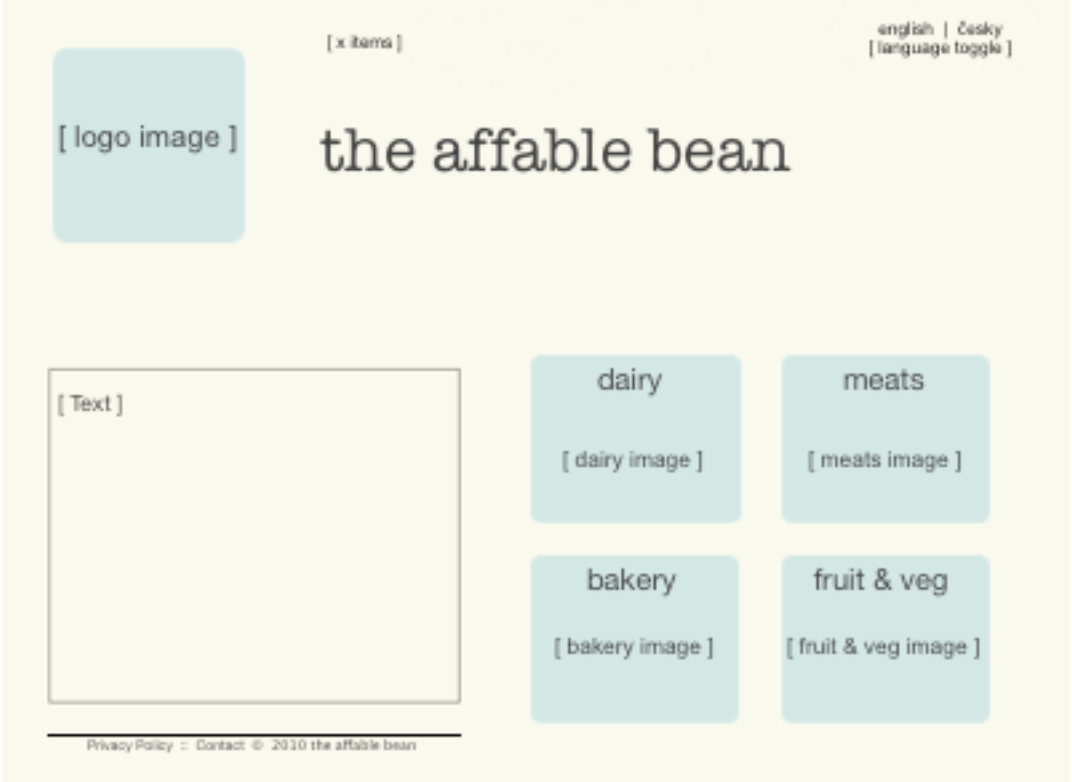
- I dati “sensibili” dei clienti che vengono trasferiti sulla rete devono essere opportunamente protetti
- Occorre prevenire l’accesso alla console di amministrazione da parte di utenti non autorizzati

Casi d'Uso: "Mockups"

- Per poter sviluppare efficacemente l'applicazione è utile disegnare le schermate principali con cui l'utente si interfaccerà durante la navigazione
- Vedremo più avanti come queste "bozze" di interfacce web non siano altro che le "viste" nel modello MVC

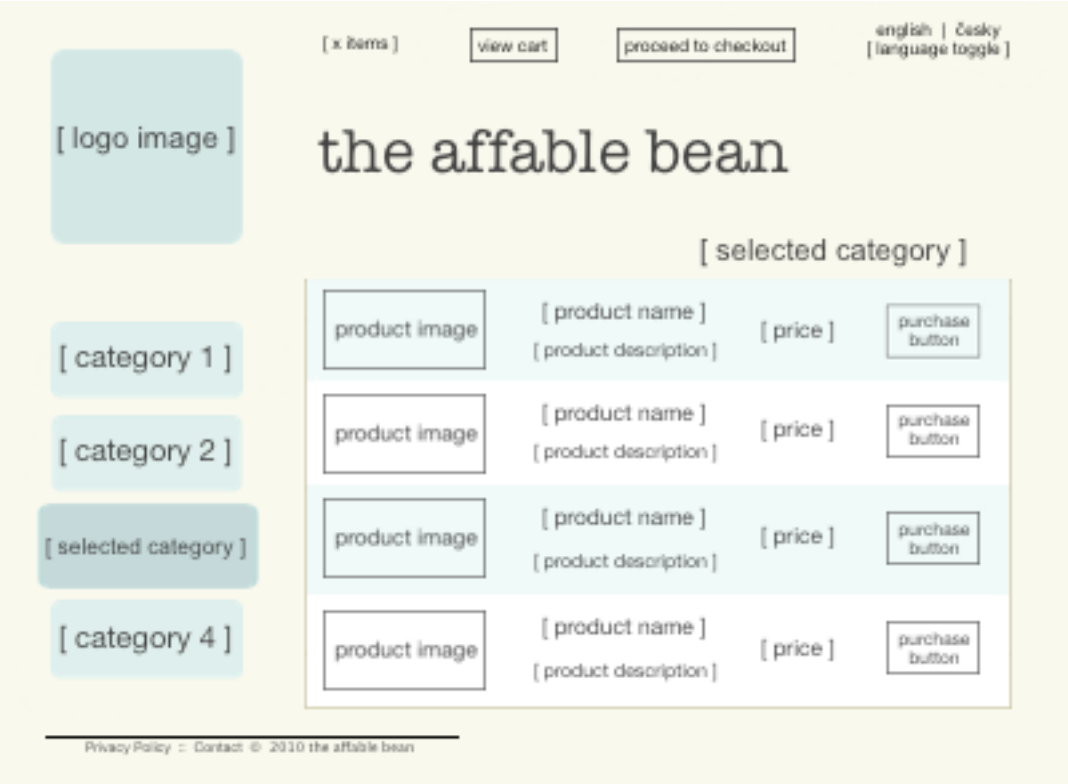
Pagina di Benvenuto

- Costituisce la home page del sito web e il punto d'ingresso principale all'applicazione
- Consente all'utente di iniziare la navigazione all'interno delle varie (4) categorie di prodotti



Pagina delle Categorie di Prodotti

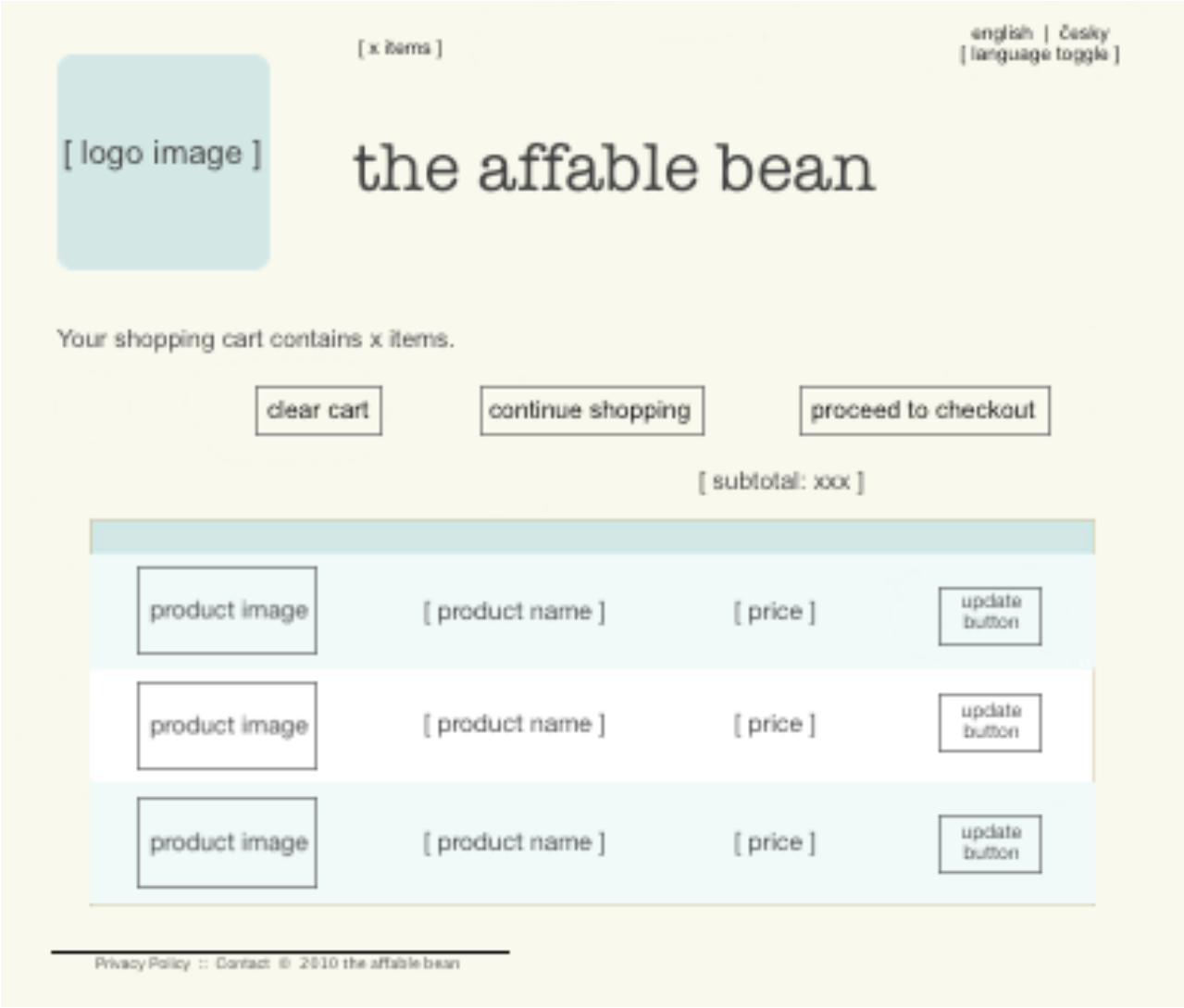
- Elenca i prodotti disponibili per quella categoria (4)
- Da questa pagina l'utente può visualizzare i dettagli di un prodotto o aggiungere elementi al proprio carrello



Pagina del Carrello della Spesa

- Elenca i prodotti presenti nel carrello della spesa
- Mostra i dettagli di ciascun prodotto e il subtotale
- Da questa pagina l'utente può:
  - Cancellare tutti gli elementi nel carrello
  - Aggiornare la quantità di un qualsiasi prodotto provocando, di conseguenza, il ricalcolo del prezzo e quantità (se la quantità viene imposta a "0" la riga relativa a quel prodotto viene rimossa
  - Ritornare alla pagina precedente per proseguire gli acquisti
  - Completare l'acquisto e procedere con il pagamento

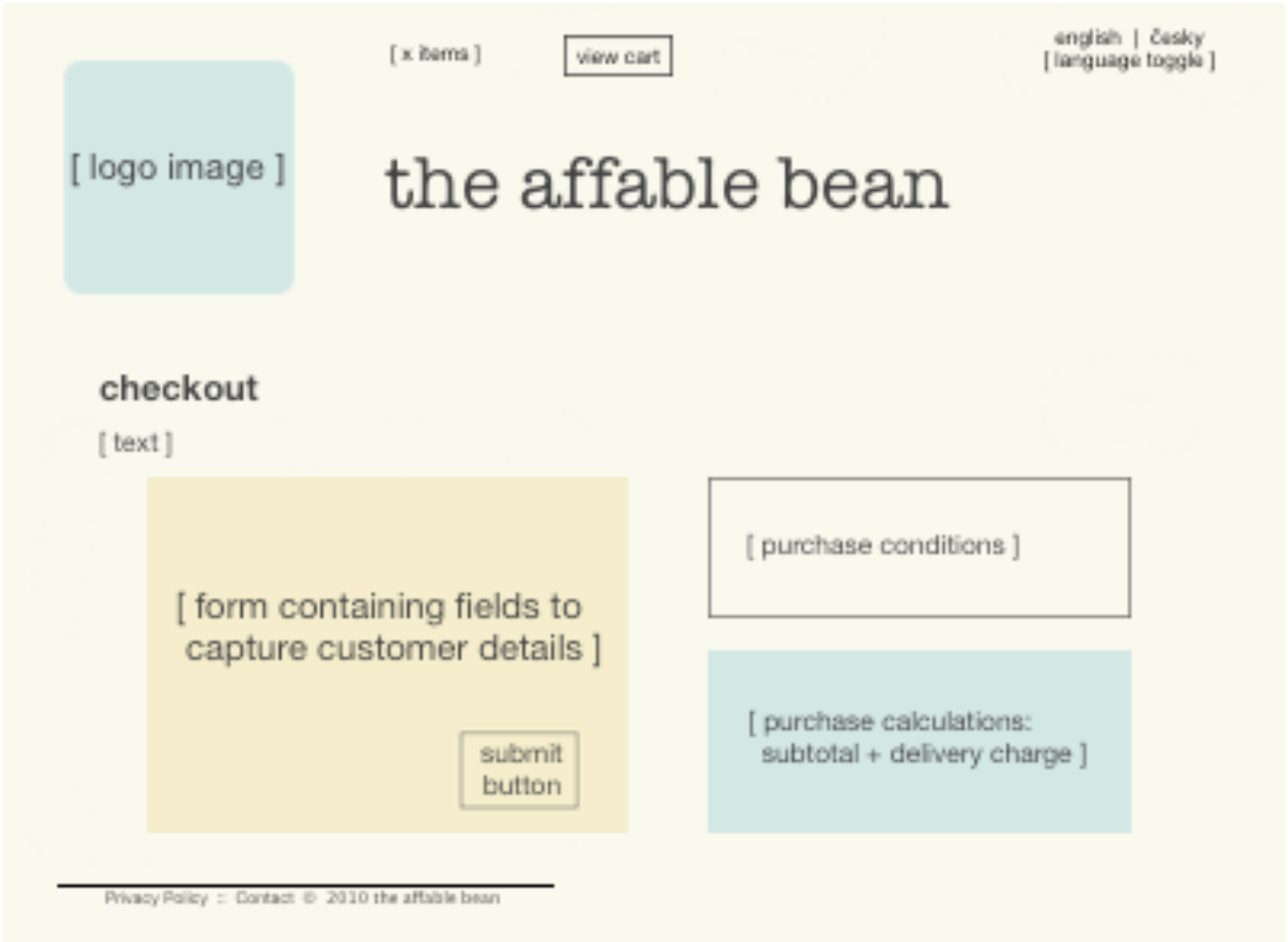
Pagina del Carrello della Spesa



Pagina del Pagamento

- Raccoglie le informazioni di pagamento del cliente tramite apposito form
- Mostra le condizioni di pagamento
- Offre il riepilogo dei beni che si stanno per acquistare
- L'utente deve inviare le proprie credenziali per il pagamento su di un canale di comunicazione sicuro (HTTPS vs. HTTP)

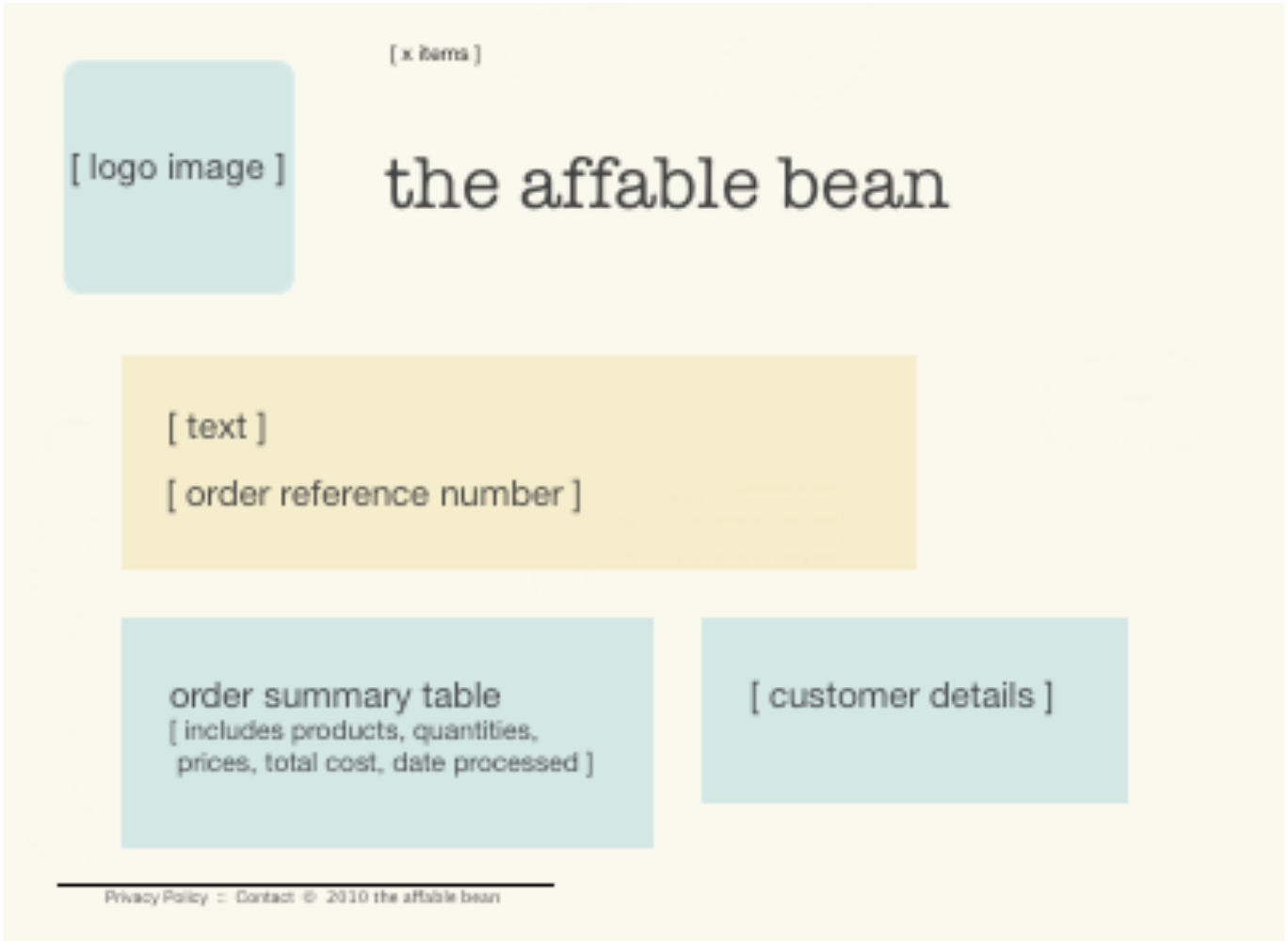
Pagina del Pagamento



Pagina di Conferma

- Restituisce un messaggio al cliente con la conferma che il suo ordine è stato preso in carica
- All'ordine viene associato un identificativo
- Viene visualizzato il riepilogo dell'ordine

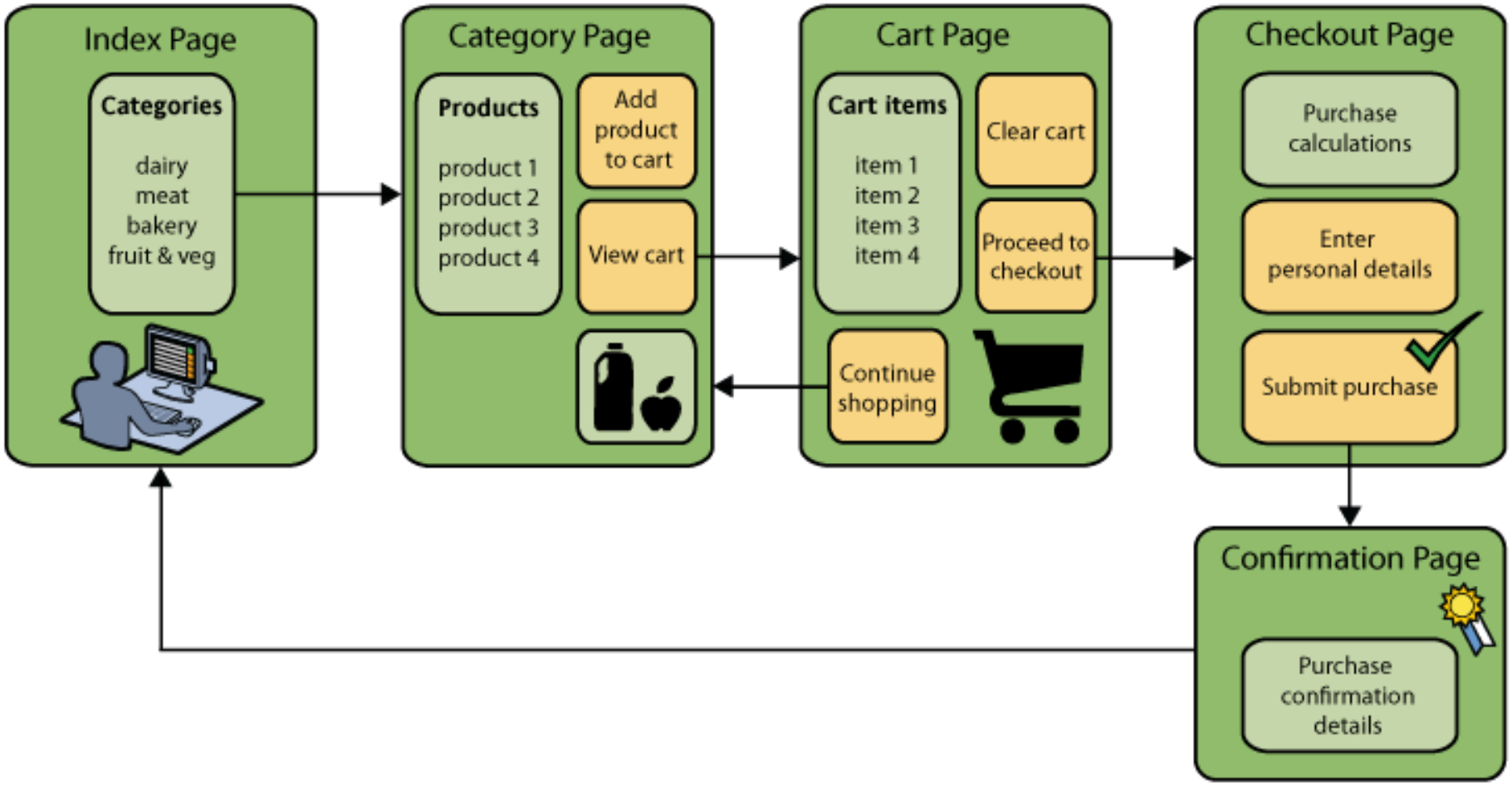
Pagina di Conferma



Altri Requisiti

- L'utente può procedere all'acquisto da qualsiasi pagina a patto che:
  - Il carrello non sia vuoto
  - L'utente non sia già nella pagina di pagamento
  - L'utente non abbia già eseguito il pagamento (ovvero sia nella pagina di conferma)
- Da tutte le pagine l'utente può:
  - Vedere lo stato del proprio carrello (se non è vuoto)
  - Ritornare alla homepage (cliccando sul logo)
- L'utente deve poter selezionare la lingua di qualunque pagina (eccetto quella di conferma)

Business Flow



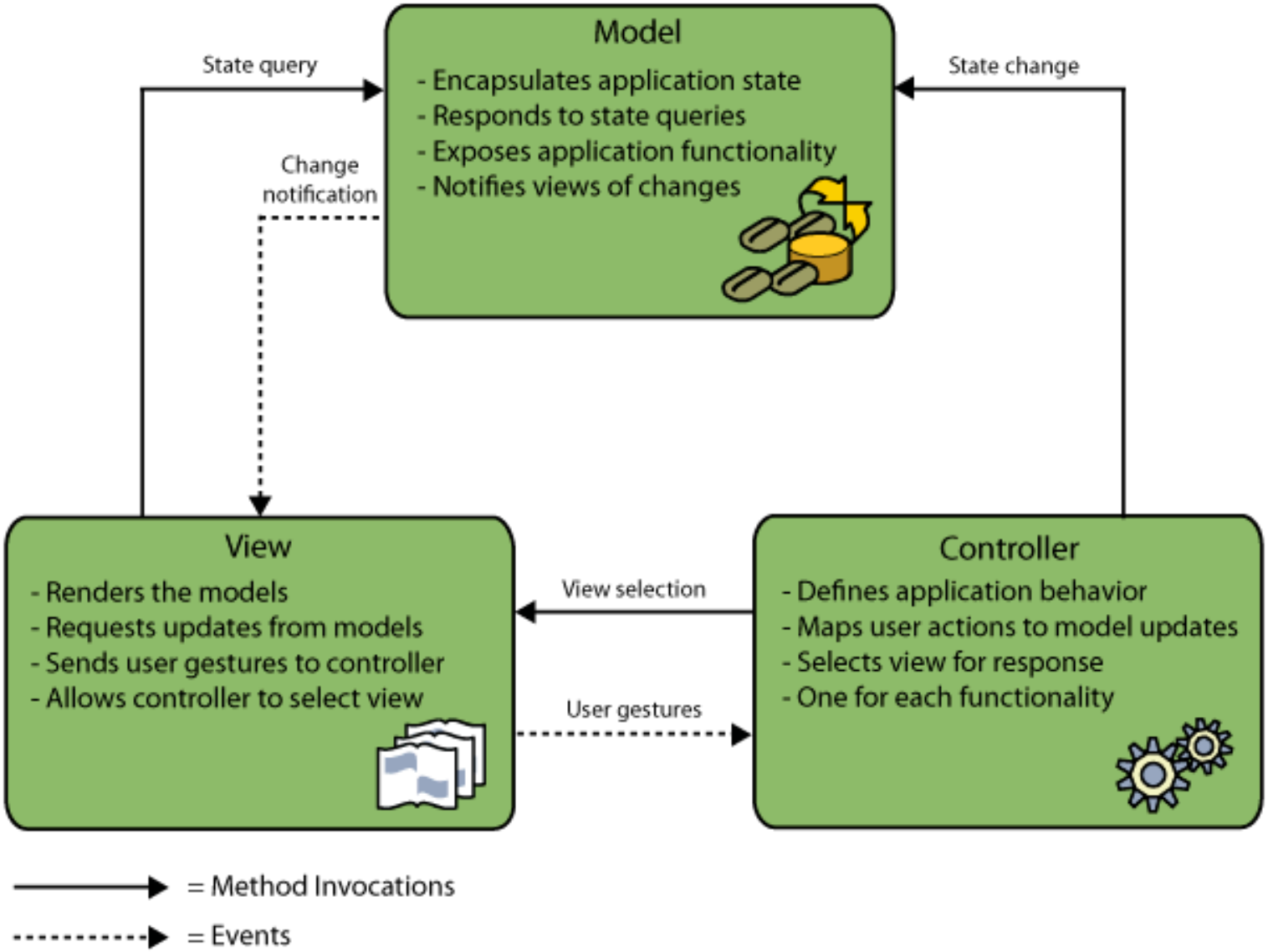
Architettura dell'Applicazione

- Suddividere le responsabilità/funzioni tra le varie componenti e determinare la loro interazione
- Usare **solamente** la tecnologia JSP (Scriptlet) ha alcuni svantaggi:
  - Il codice all'interno delle pagine JSP non è riusabile da altri componenti
    - Ripetizione di logica che deve essere condivista tra più componenti JSP
  - Mix tra logica e presentazione
  - La fase di testing è piuttosto complicata
  - …

Architettura dell'Applicazione: **MVC**

- Paradigma che suddivide l'applicazione in 3 componenti interoperabili:
  - **Model (M)** → rappresenta i "dati" del dominio applicativo su cui opera la business logic
  - **View (V)** → visualizza il contenuto di un modello (di dati) secondo specifiche modalità e smista l'input dell'utente ad un controller
  - **Controller (C)** → definisce il comportamento dell'applicazione interpretando le richieste dell'utente (HTTP GET/POST nel caso web)

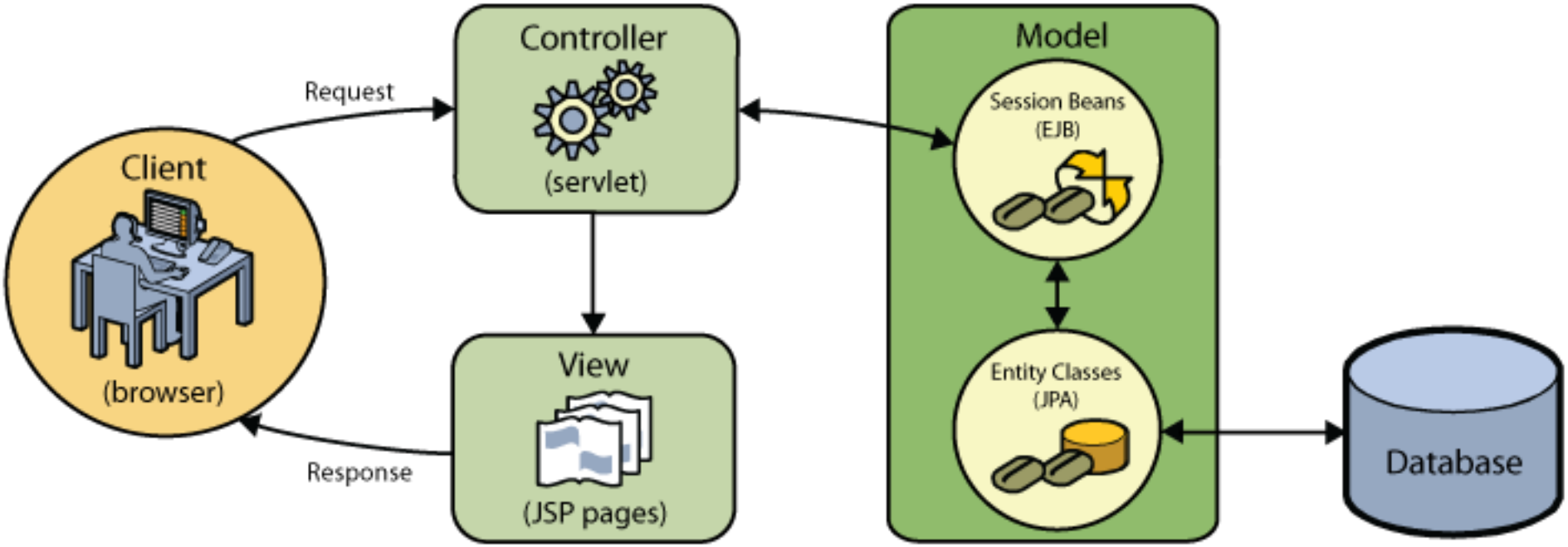
Architettura dell'Applicazione: **MVC**



AffableBean + MVC

- Applicare il paradigma MVC per la realizzazione dell'applicazione AffableBean
- Usare **servlet** per gestire le richieste che provengono dagli utenti → **Controller**
- Le pagine "bozza" mostrate in precedenza diventano le "viste" **JSP → View**
- Infine i dati (memorizzati in un RDBMS) saranno acceduti e modificati tramite **EJB session beans** e **JPA entity classes → Model**

AffableBean + MVC



Pianificazione

- Setup dell'ambiente di sviluppo
- Preparazione del modello dei dati dell'applicazione (DB)
- Creazione/Organizzazione dei file per il front-end (interfaccia) dell'applicazione
- Creazione di una servlet che faccia da "controller"
- Connessione dell'applicazione al DB
- Sviluppo della business logic
- Aggiunta supporto linguistico
- Creazione console di amministrazione
- Gestione della sicurezza dell'applicazione

Setup

Setup Ambiente di Sviluppo

- Abbiamo bisogno di 3 progetti Eclipse distinti:
  - AffableBeanWeb (.war) → Progetto Web dinamico per la realizzazione del web tier (Servlet + JSP)
  - AffableBeanEJB (.jar) → Progetto EJB per la realizzazione dell'enterprise tier (EJB)
  - AffableBeanEAR (.ear) → Progetto Enterprise Application contenitore di AffableBeanWeb e AffableBeanEJB
- NOTA: A partire da Java EE 6 è possibile includere le funzionalità EJB direttamente all'interno di un progetto Web dinamico (.war)

Database

Il Database MySQL

- Scaricare il server MySQL Community Server + MySQL Workbench

  http://dev.mysql.com/downloads
- Prendere confidenza con alcuni comandi di base per l'interazione con MySQL
- Di seguito supponiamo che ${MYSQL_HOME} sia il path del file system in cui si trova il vostro server MySQL

MySQL: Comandi Base

- Tutti i comandi seguenti si trovano in
  ${MYSQL_HOME}/bin
- **`mysqld`** → è l'eseguibile che deve essere lanciato per avviare il DB server
  - su piattaforma Windows NT è possibile configurare il server come servizio di sistema (avvio in fase di boot)
- **`mysql`** → avvia la console interattiva da linea di comando per interagire con il server MySQL
- **`mysqladmin`** → tool di amministrazione del server da linea di comando

MySQL: Avvio

- Portarsi su ${MYSQL_HOME}/bin e avviare il server tramite il comando:

  `> mysqld`

- Per accertarsi che il server si sia avviato correttamente digitare il seguente comando:

  `> mysqladmin ping`

- Se tutto è andato a buon fine comparirà il seguente messaggio:

  `mysqld is alive`

MySQL: Interfaccia da Linea di Comando

- Una volta avviato il server DB è possibile interagire con esso tramite interfaccia da linea di comando digitando:

```
> mysql –u root
```

- L'utente **root** è l'utente di default creato dall'installazione di MySQL ed ha tutti i privilegi di amministrazione
  - È consigliabile aggiungere una password a questo utente in questo modo:

```
> mysql –u root
mysql> SET PASSWORD FOR 'root'@'localhost' =
       PASSWORD('password');
mysql> FLUSH PRIVILEGES;
```

MySQL: Lista di Comandi

- Per una lista di comandi utili, consultare il seguente riferimento:

  http://www.pantz.org/software/mysql/mysqlcommands.html

Il Modello dei Dati

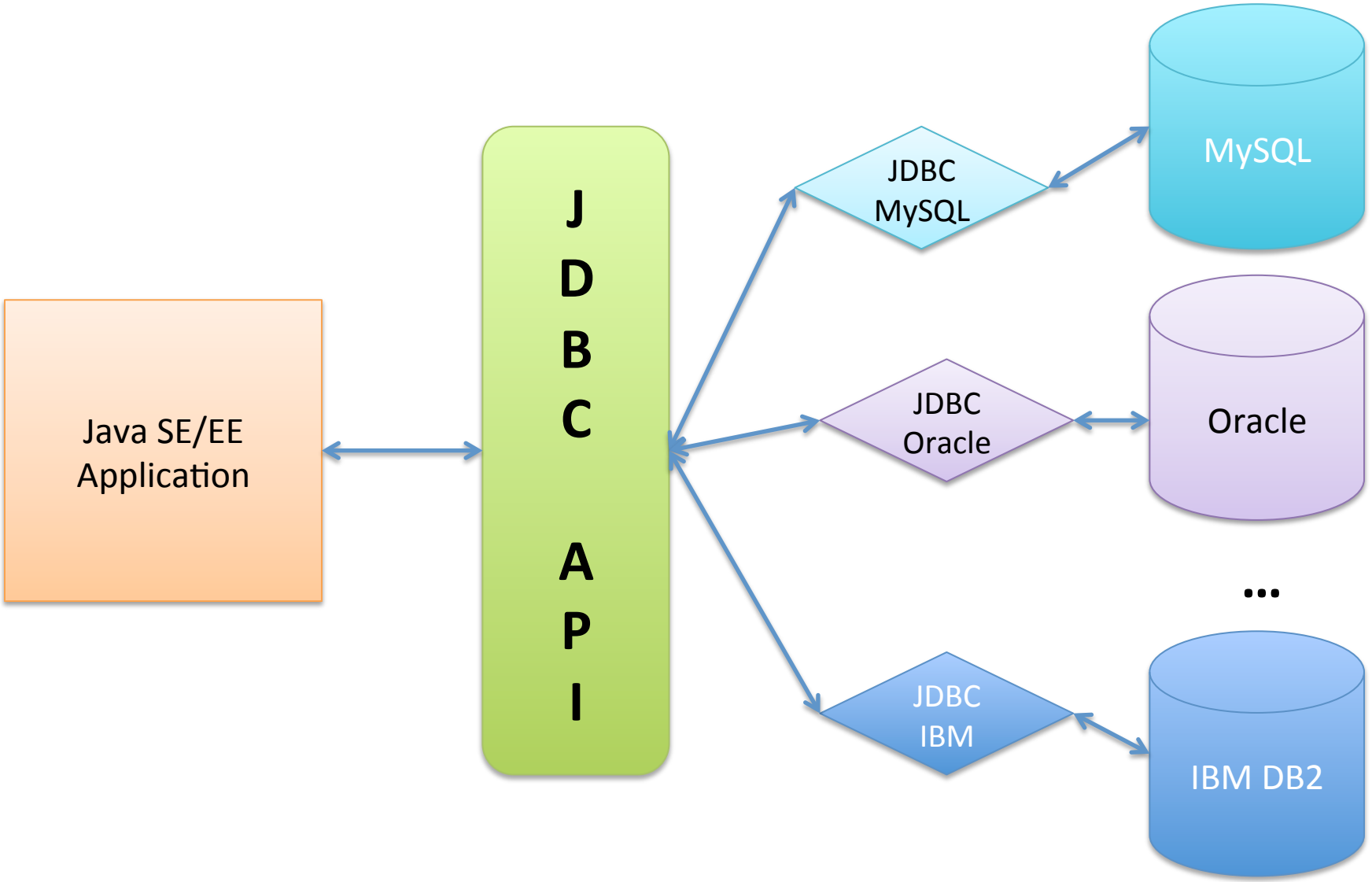
- Usiamo MySQL Workbench per il disegno dello schema del DB
- Questo strumento consente la progettazione dello schema **concettuale**
- Lo schema concettuale verrà poi realizzato concretamente in un'istanza del nostro DB server MySQL (Forward Engineering)
- Una volta create le tabelle "reali" sul DB, creare l'utente che l'applicazione userà per accedere ai dati:
  - `GRANT ALL PRIVILEGES ON affablebeanDB.* TO affablebean@localhost IDENTIFIED BY 'affablebean';`

Java + DB: JDBC API

- La JDBC API fornisce l'interfaccia tra un'applicazione Java ed un **qualsiasi** DB
- In questo modo consente allo sviluppatore di astrarre dal particolare DB con cui si deve interagire
- La maggior parte dei produttori di DB (MySQL, Oracle, IBM DB2, etc.) fornisce un'implementazione di JDBC API
- Per interfacciarsi con MySQL scaricare l'apposito connettore JDBC e renderlo disponibile alla/alle applicazione che ne fa/fanno uso

  http://dev.mysql.com/downloads/connector/j/
- Nel nostro caso, aggiungere il file .jar nella directory del server JBoss ${JBOSS_SERVER}/lib/

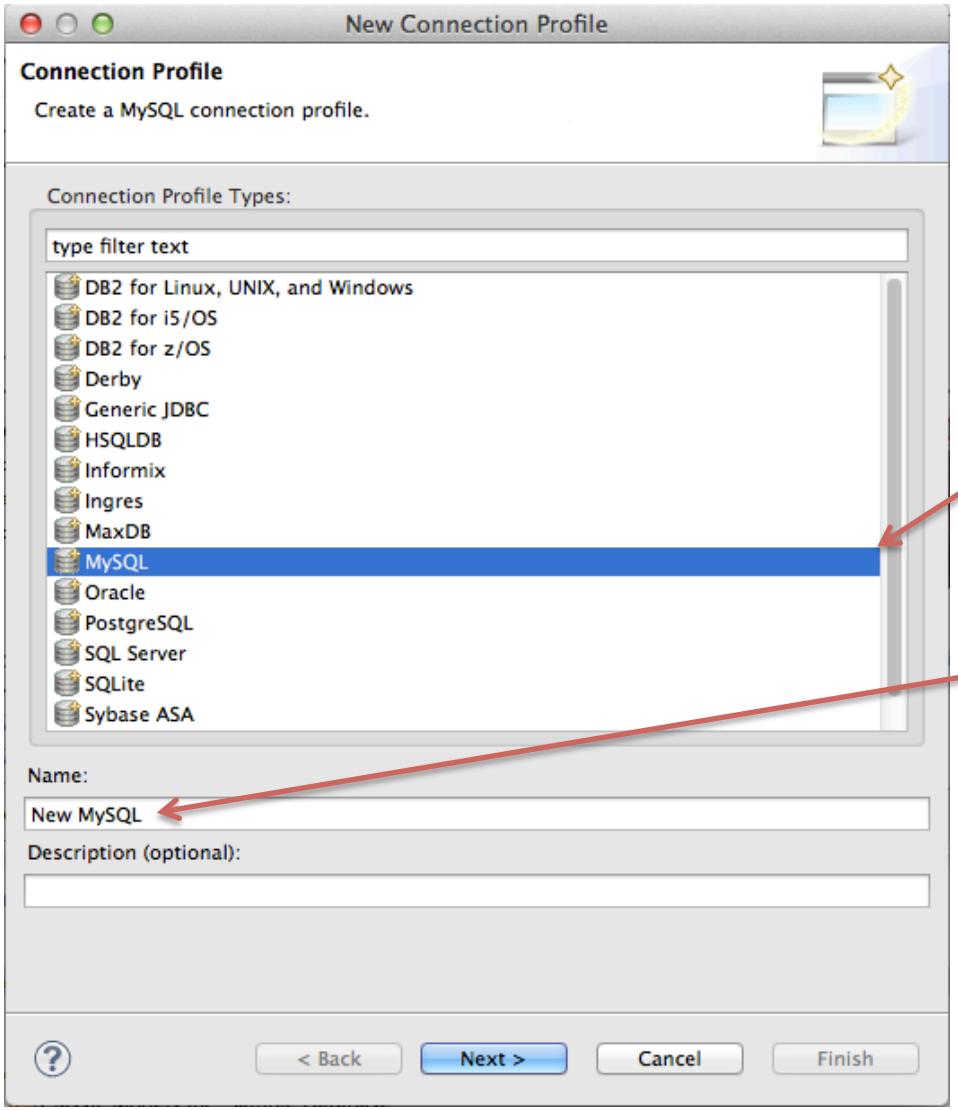
Java + DB: JDBC API



MySQL + Eclipse: Profilo

- Eclipse offre un'apposita "vista" per l'interfaccia con varie sorgenti dati
  - Window → Show View → Data Source Explorer
- Per aggiungere un **profilo** specifico su un DB esistente:
  - Click dx su "Database Connections" → New
  - Seguire le istruzioni del wizard per la creazione del profilo

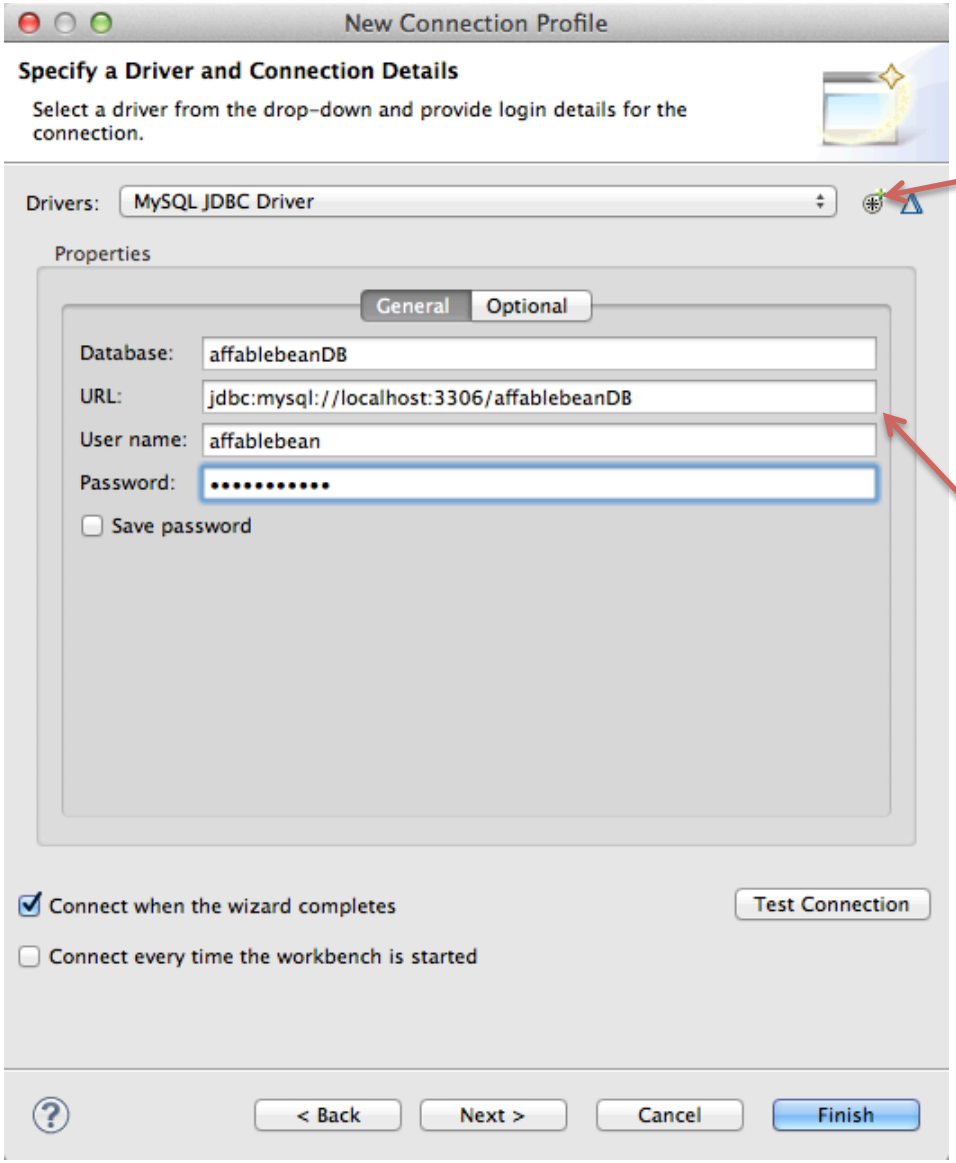
MySQL + Eclipse: Profilo



Selezionare il “tipo” di DB

Specificare
il nome del profilo

MySQL + Eclipse: Profilo



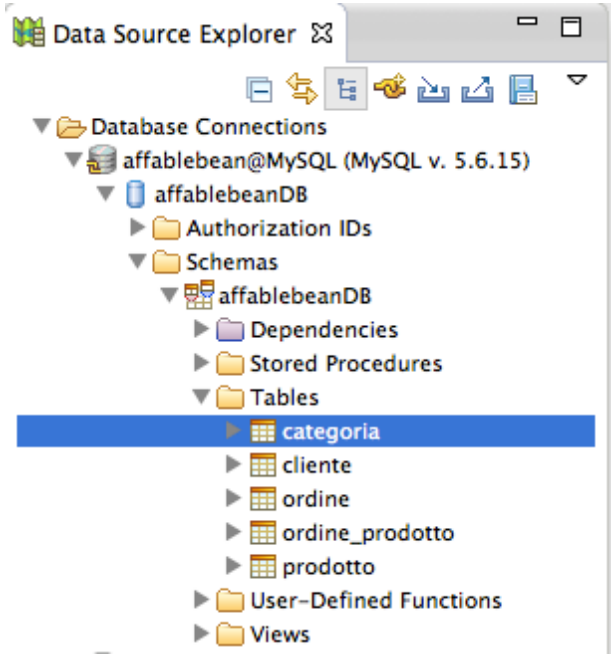
Aggiungere/Editare
il driver JDBC

Specificare i parametri
del profilo DB da creare:

- Nome: affablebeanDB
- URL
- Username: affablebean
- Password: affablebean

MySQL + Eclipse: Profilo

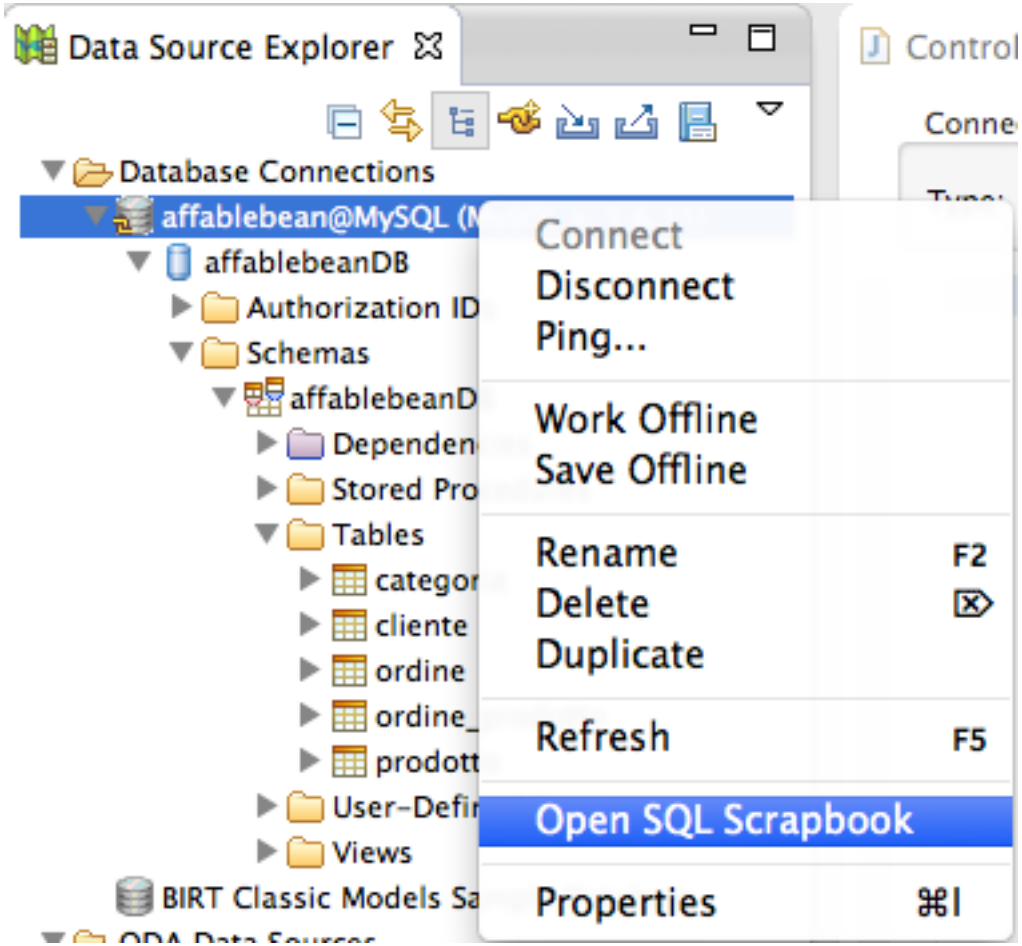
- Prima di testare il corretto funzionamento del profilo ricordarsi di aver avviato il server DB!
- Se tutto va a buon fine, verranno mostrati i dettagli del DB a cui si riferisce il profilo creato



MySQL + Eclipse: SQL

- Interagire con il DB dall'interno di Eclipse tramite il linguaggio standard per RDBMS: SQL
- Click dx sul profilo DB → Open SQL Scrapbook
- Ci sono molti altri strumenti ad hoc per l'interazione via SQL con il DB (extra Eclipse)

MySQL + Eclipse: SQL



Interfaccia Web

Le Viste: Pagine JSP

- 5 viste corrispondenti a 5 pagine JSP:
  - index.jsp (punto di accesso dell'applicazione)
    - All'interno di WebContent/WEB-INF/
    - Accedibile "pubblicamente" via HTTP GET
  - Le altre viste devono essere accedute solo a fronte di scambi di dati col server e per questo sono memorizzate in una sottocartella "view" di WEB-INF
    - view/cart.jsp → carrello della spesa
    - view/category.jsp → categorie di prodotto
    - view/checkout.jsp → inserimento dati pagamento
    - view/confirmation.jsp → conferma ordine

Le Viste: Pagine JSP

- Header e Footer a comune di tutte le pagine vengono inseriti in appositi “frammenti” JSP (.jspf)
- All’interno della directory WEB-INF/jspf
- Evitano la duplicazione di contenuto statico comune a tutte le viste

Le Viste: Pagine JSP

- Utilizzare il link fornito nel tutorial per scaricare le viste
- Il contenuto di queste pagine JSP utilizza un foglio di stile CSS (affablebean.css)
- Inserire il .css all'interno di un'apposita directory del progetto Eclipse (ad es. WebContent/css)

Servlet “Controller”

ControllerServlet.java

- Secondo il modello MVC introdotto implementiamo un'unica Servlet che "smisti" le richieste dei client alle risorse opportune
- Questa non fa altro che testare il path della richiesta HTTP ed eseguire il "dispatching" alle apposite pagine JSP (viste)
- Iniziamo con lo "scheletro" a cui seguirà l'implementazione della logica vera e propria

ControllerServlet.java: doGet

```java
public class ControllerServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ControllerServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub

        String userPath = request.getServletPath(); // restituisce il path associato alla richiesta corrente (ad es. "/addToCart")

        // se la pagina richiesta è "/category"
        if(userPath.equals("/category")) {
            // implementare la gestione delle categorie prodotto
        }
        else if(userPath.equals("/viewCart")) {
            // implementare visualizza carrello
            userPath = "/cart";
        }
        else if(userPath.equals("/checkout")) {
            // implementare checkout
        }
        else if(userPath.equals("/chooseLanguage")) {
            // implementare scelta lingua
        }

        // URL a cui inoltro la richiesta tramite RequestDispatcher
        String url = "/WEB-INF/view" + userPath + ".jsp";

        try {
            request.getRequestDispatcher(url).forward(request, response);
        } catch(Exception ex) {
            ex.printStackTrace();
        }

    }
```

ControllerServlet.java: doPost

```java
/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub

    String userPath = request.getServletPath(); // restituisce il path associato alla richiesta corrente (ad es. "/addToCart")

    if(userPath.equals("/addToCart")) {
        // implementare aggiunta prodotto al carrello
    }
    else if(userPath.equals("/updateCart")) {
        // implementare aggiornamento carrello
    }
    else if(userPath.equals("/purchase")) {
        // implementare acquisto
        userPath = "/confirmation";
    }

    // URL a cui inoltro la richiesta tramite RequestDispatcher
    String url = "/WEB-INF/view" + userPath + ".jsp";

    try {
        request.getRequestDispatcher(url).forward(request, response);
    } catch(Exception ex) {
        ex.printStackTrace();
    }


}
```

Connessione al DB

Integrazione DB su JBoss

- I componenti server “deployati” su JBoss che devono connettersi a RDBMS possono farlo:
  - Direttamente
    - Gestione delle connessioni affidata al singolo componente
    - Una configurazione per ogni applicazione
  - Tramite un “DB connection pool” condiviso tra tutti i componenti e gestito da JBoss
    - Configurazione semplice e manutenibile (un singolo file da editare)
    - Condivisione tra più applicazioni

Integrazione DB su JBoss: I Passi

1. Definire un riferimento alla risorsa (DB) all'interno dell'applicazione
   - Richiede connettività al DB
2. Fornire le risorse all'interno del server (pool di connessioni al DB)
   - Installazione driver JDBC (già visto)
   - Definizione di un DB Connection Pool (DBCP)
   - Mapping tra il DBCP gestito da JBoss e il riferimento alla risorsa specificato nell'applicazione

1. Riferimento alla Risorsa DB

- All'interno del file **web.xml** è possibile specificare la necessità di comunicare con un a risorsa RDBMS gestita dal container:

```
<resource-ref>
 <description>Connection Pool for the AffableBean Application</description>
 <!-- Riferimento (nome simbolico) alla risorsa relativo al contesto JNDI java:comp/env/ -->
 <res-ref-name>jdbc/AffableBeanDS</res-ref-name>
 <!-- Tipo di risorsa (DataSource) -->
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
</resource-ref>
```

NOTA:

il riferimento alla risorsa specificato **jdbc/AffableBeanDS** è relativo al contesto JNDI **java:comp/env** che è unico ed isoltato per ciascuna applicazione

2.a Installazione Driver JDBC

- Abbiamo già visto che i driver JDBC consentono alle applicazioni Java di interfacciarsi con vari DB (MySQL, IBM DB2, …)
- Copiare il driver JDBC di MySQL (.jar) all'interno di una delle 2 seguenti directory:
  - **${jboss.server.lib.url}**
  - **${jboss.common.lib.url}**

2.b Definizione della risorsa DBCP

- Creare un file ***-ds.xml** all'interno della directory deploy del server
  - Ad es., **deploy/affablebean-ds.xml**
- Prendere come spunto il template:

  ${JBOSS_HOME}/docs/examples/jca/mysql-ds.xml

NOTA:

In JBoss AS il riferimento alla risorsa DBCP è relativo al contesto JNDI **java:/** che condiviso ed accessibile da **tutte** le applicazioni in esecuzione sulla stessa JVM

2.b **affablebean-ds.xml**

```
<datasources>
  <local-tx-datasource>
    <jndi-name>AffableBeanDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/affablebeanDB?autoReconnect=true</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>affablebean</user-name>
    <password>affablebean</password>
    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-sorter-class-name>
    <new-connection-sql>SELECT 1</new-connection-sql>
    <check-valid-connection-sql>SELECT 1</check-valid-connection-sql>
    <min-pool-size>5</min-pool-size> <!-- the minimum number of pooled database connections.
    Initialized when the pool is first accessed. Defaults to 0 -->
    <max-pool-size>32</max-pool-size> <!-- the maximum number of pooled database connections.
    Once this limit is reached, clients block. Defaults to 20 -->
    <blocking-timeout-millis>5000</blocking-timeout-millis> <!-- the maximum blocking time (in ms) while waiting on an available
    connection before timing out by throwing an exception. Defaults to 5000 (or 5 seconds) -->
    <track-statements>false</track-statements> <!-- if true, unclosed statements are reported on check-in (via a warning message).
    Defaults to false -->
    <idle-timeout-minutes>15</idle-timeout-minutes> <!--the maximum time (in minutes) before idle connections are closed -->

    <!-- should only be used on drivers after 3.22.1 with "ping" support
    <valid-connection-checker-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker</valid-connection-checker-
    class-name>
    -->
    <!-- sql to call when connection is created
    <new-connection-sql>some arbitrary sql</new-connection-sql>
      -->
    <!-- sql to call on an existing pooled connection when it is obtained from pool - MySQLValidConnectionChecker is preferred for newer
    drivers
    <check-valid-connection-sql>some arbitrary sql</check-valid-connection-sql>
      -->

    <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (optional) -->
    <metadata>
       <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

2.c Mapping tra Applicazione e JBoss

- Occorre mappare il riferimento alla risorsa nel contesto dell'applicazione alla risorsa reale fornita e gestita dal server JBoss
- Creare un file **WEB-INF/jboss-web.xml** che mappa **java:comp/env/jdbc/AffableBeanDS** su **java:/AffableBeanDS**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Binding tra il/i riferimento/i alla/e risorsa/e definite dall'applicazione
e la/e risorsa/e reale/i fornita e gestita dal server JBoss
 -->
<jboss-web>
    <!-- Mapping tra la risorsa di tipo DataSource definita in web.xml
     col nome simbolico java:comp/env/jdbc/AffableBeanDS e quella reale
     definita all'interno di ${jboss.server.home}/deploy/affablebean-ds.xml -->
    <resource-ref>
        <!-- Nome simbolico specificato in web.xml -->
        <res-ref-name>jdbc/AffableBeanDS</res-ref-name>
        <!-- Tipo della risorsa -->
        <res-type>javax.sql.DataSource</res-type>
        <!-- Nome JNDI nel contesto globale del server JBoss (vedi affablebean-ds.xml) -->
        <jndi-name>java:/AffableBeanDS</jndi-name>
    </resource-ref>
</jboss-web>
```

Test Connessione

- Pagina JSP di test che usa JSTL per eseguire una query al DB

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<sql:query var="result" dataSource="jdbc/AffableBeanDS">
SELECT * FROM categoria,prodotto
WHERE categoria.id = prodotto.categoria_id
</sql:query>
<table>
<tr>
    <c:forEach var="columnName" items="${result.columnNames}">
        <th><c:out value="${columnName}"/></th>
    </c:forEach>
</tr>
<c:forEach var="row" items="${result.rowsByIndex}">
    <tr>
        <c:forEach var="column" items="${row}">
            <td><c:out value="${column}"/></td>
        </c:forEach>
    </tr>
</c:forEach>
</table>
</body>
</html>
```

Test Connessione

id	nome	id	nome	prezzo	descrizione	ultima_modifica	categoria_id
1	dairy	1	Latte	1.70	Parzialmente Scremato (1L)	2014-01-24 17:05:25.0	1
1	dairy	2	Formaggio	2.39	Cheddar lieve (330g)	2014-01-24 17:05:26.0	1
1	dairy	3	Burro	1.09	Senza Sale (250g)	2014-01-24 17:05:27.0	1
1	dairy	4	Uova	1.76	media grandezza (6 eggs)	2014-01-24 17:05:28.0	1
2	meats	5	organic meat patties	2.29	rolled in fresh herbs 2 patties (250g)	2014-01-24 17:05:29.0	2
2	meats	6	parma ham	3.49	matured, organic (70g)	2014-01-24 17:05:30.0	2
2	meats	7	chicken leg	2.59	free range (250g)	2014-01-24 17:05:31.0	2
2	meats	8	sausages	3.55	reduced fat, pork 3 sausages (350g)	2014-01-24 17:05:32.0	2
3	bakery	9	sunflower seed loaf	1.89	600g	2014-01-24 17:05:33.0	3
3	bakery	10	sesame seed bagel	1.19	4 bagels	2014-01-24 17:05:35.0	3
3	bakery	11	pumpkin seed bun	1.15	4 buns	2014-01-24 17:05:36.0	3
3	bakery	12	chocolate cookies	2.39	contain peanuts (3 cookies)	2014-01-24 17:05:37.0	3
4	fruit & veg	13	corn on the cob	1.59	2 pieces	2014-01-24 17:05:38.0	4
4	fruit & veg	14	red currants	2.49	150g	2014-01-24 17:05:39.0	4
4	fruit & veg	15	broccoli	1.29	500g	2014-01-24 17:05:40.0	4
4	fruit & veg	16	seedless watermelon	1.49	250g	2014-01-24 17:05:41.0	4