# JEE Application Report

David Kelly B00060572

Institute of Technology Blanchardstown
Institiúid Teicneolaíochta Baile Bhlainséir

# Affable Bean Application

30/11/2015

## Scenario

A small grocery store, the Affable Bean, collaborates with several local farms to supply a community with organic produce and foods. Due to a long-standing customer base and increasing affluence to the area, the store has decided to investigate the possibility of providing an online delivery service to customers. A recent survey has indicated that 90% of its regular clientele has continuous Internet access, and 65% percent would be interested in using this service.

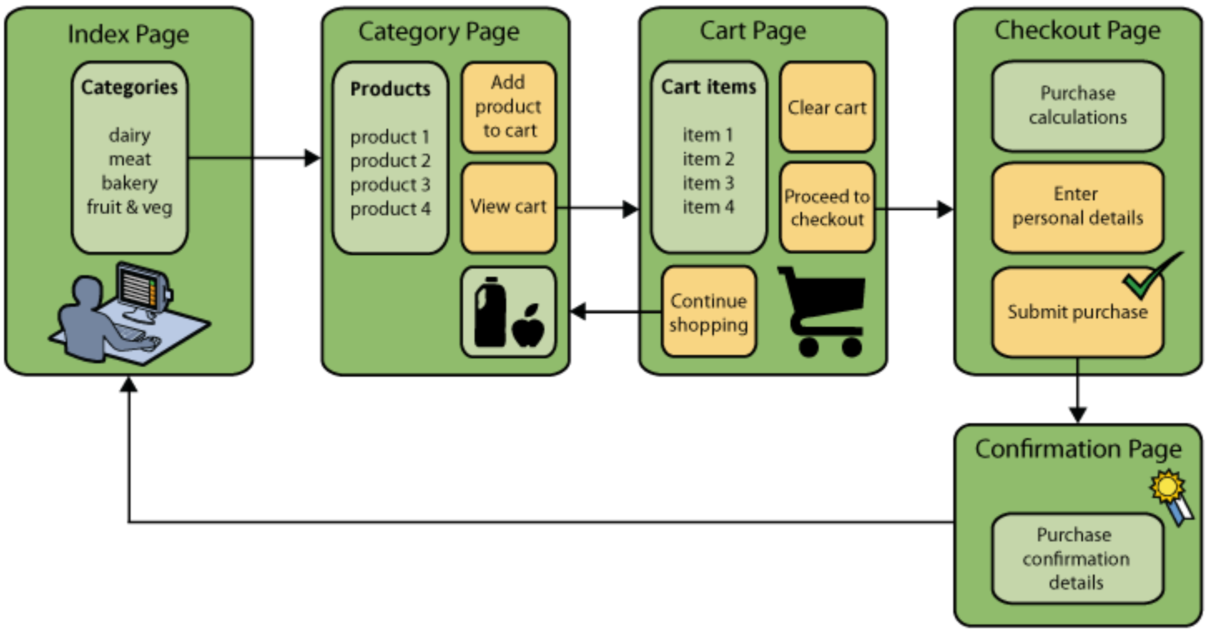
The grocery store staff have asked you, the Java web developer, to create a website that will enable their customers to shop online. They have also asked that you create an administration console alongside the website, which will allow staff members to keep track of orders.

The store's location is in Prague, in the Czech Republic. Because regular clientele are both English and Czech-speaking, staff have requested that the website support both languages.

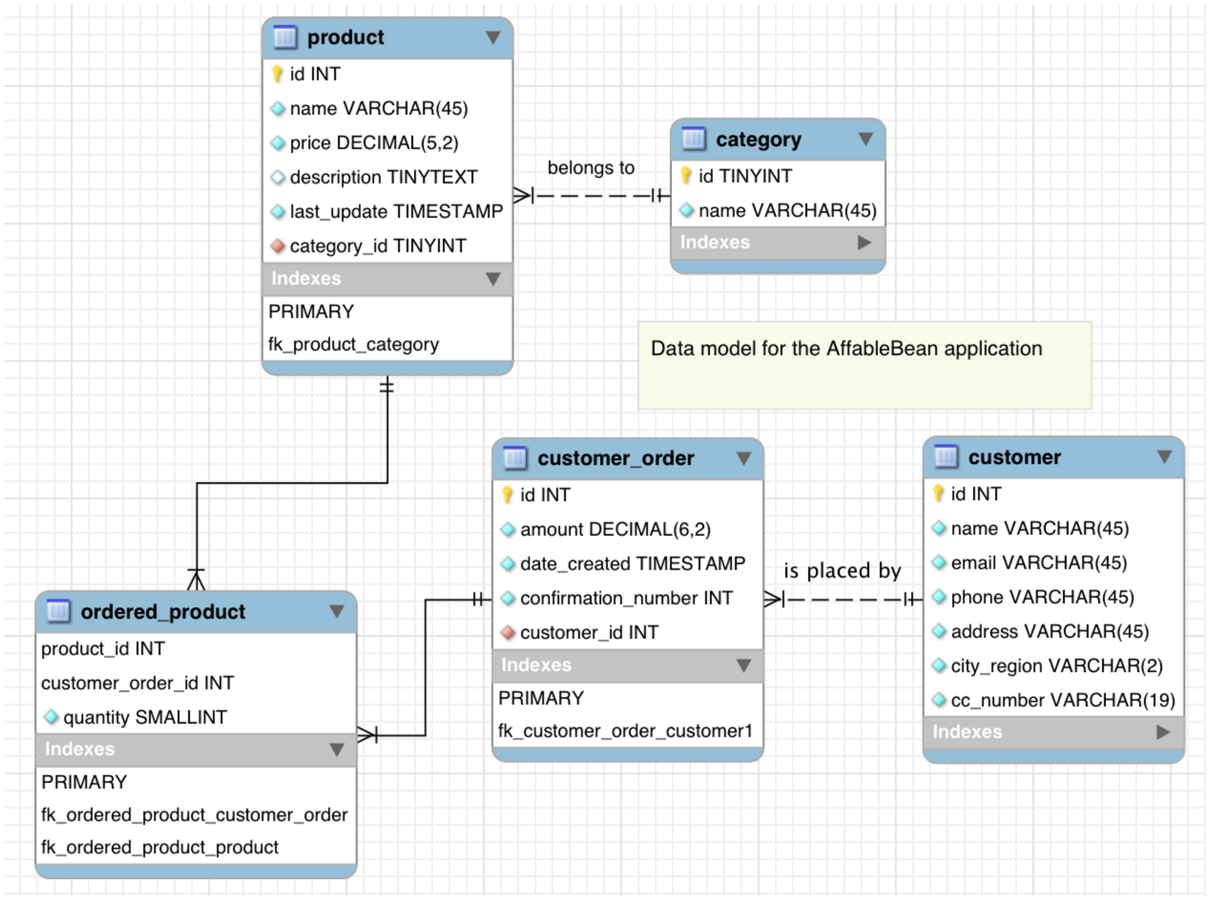
The grocery store has already purchased a domain and web hosting plan that provides a Java EE 6-compliant server and MySQL database server. Staff have indicated that one technically-oriented member is able to deploy the application to the production server once it is ready.

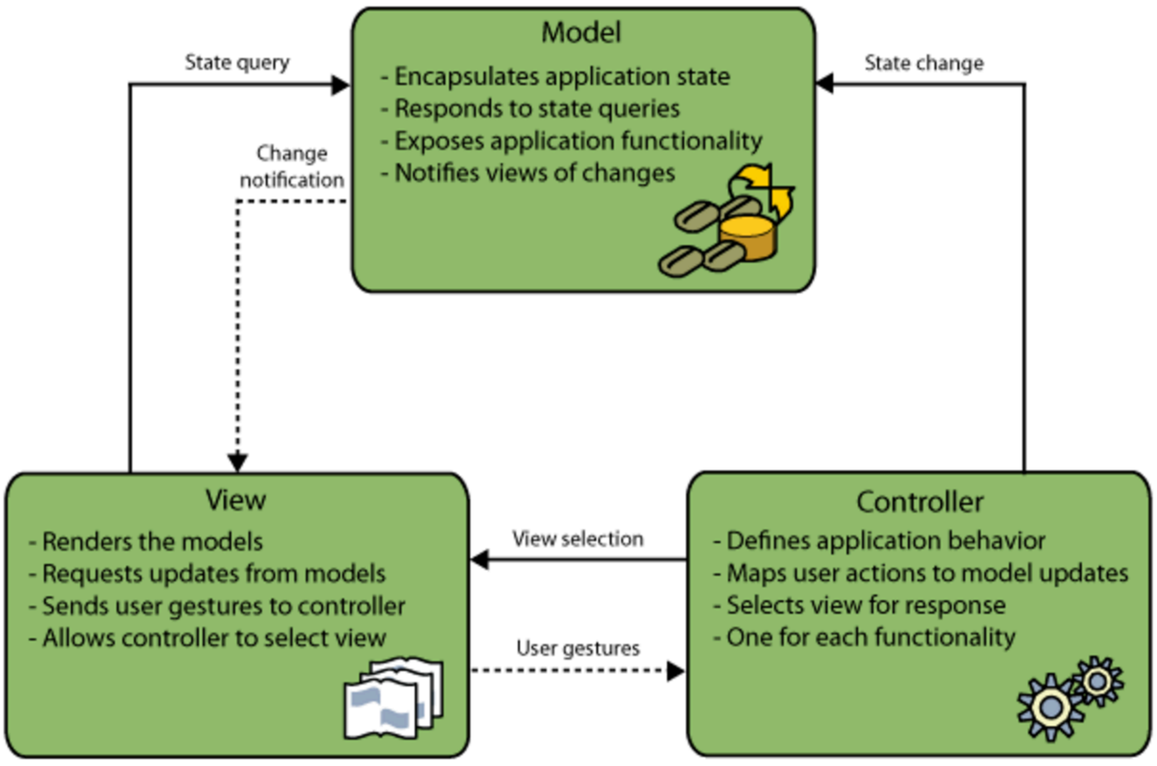
## Use-Case

Customer visits the welcome page and selects a product category. Customer browses products within the selected category page, then adds a product to his or her shopping cart. Customer continues shopping and selects a different category. Customer adds several products from this category to shopping cart. Customer selects 'view cart' option and updates quantities for cart products in the cart page. Customer verifies shopping cart contents and proceeds to checkout. In the checkout page, customer views the cost of the order and other information, fills in personal data, then submits his or her details. The order is processed and customer is taken to a confirmation page. The confirmation page provides a unique reference number for tracking the customer order, as well as a summary of the order.



## Model


product
id INT
name VARCHAR(45)
price DECIMAL(5,2)
description TINYTEXT
last_update TIMESTAMP
category_id TINYINT
Indexes
PRIMARY
fk_product_category
belongs to
category
id TINYINT
name VARCHAR(45)
Indexes
Data model for the AffableBean application
customer_order
id INT
amount DECIMAL(6,2)
date_created TIMESTAMP
confirmation_number INT
customer_id INT
Indexes
PRIMARY
fk_customer_order_customer1
is placed by
customer
id INT
name VARCHAR(45)
email VARCHAR(45)
phone VARCHAR(45)
address VARCHAR(45)
city_region VARCHAR(2)
cc_number VARCHAR(19)
Indexes
ordered_product
product_id INT
customer_order_id INT
quantity SMALLINT
Indexes
PRIMARY
fk_ordered_product_customer_order
fk_ordered_product_product


Model
- Encapsulates application state
- Responds to state queries
- Exposes application functionality
- Notifies views of changes
State query
State change
Change notification
View
- Renders the models
- Requests updates from models
- Sends user gestures to controller
- Allows controller to select view
View selection
User gestures
Controller
- Defines application behavior
- Maps user actions to model updates
- Selects view for response
- One for each functionality

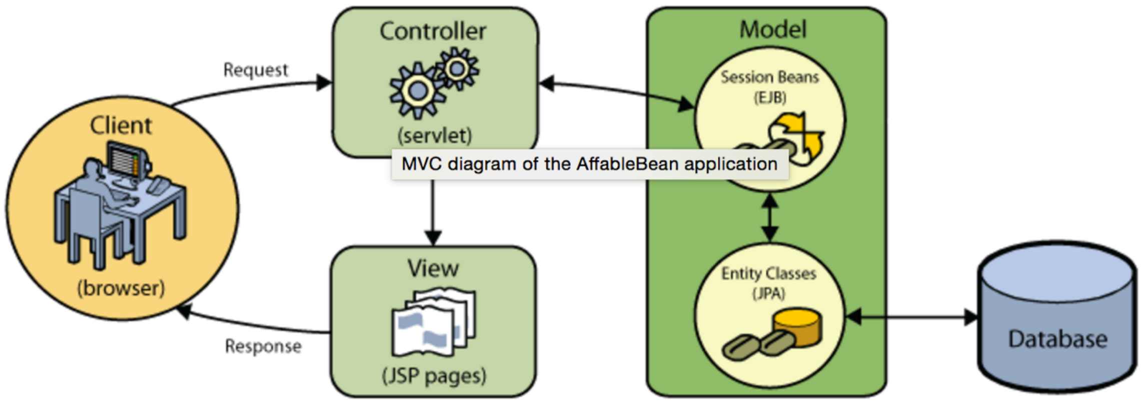
## Components

**WEB Pages folder**

The web pages folder contains the JSP files related to the html display. Each JSP contains code specific to the display of each webpage. Within the JSP files JSPF (fragments) are called. These fragments contain code that is common across many webpages and can simply be plugged in to the JSP file and retrieved. Commonalities across this application include the header and footer on each webpage. The file affable.css provides the style sheet for the entire website. Using tags and classes to define components within the HTML code to change the style on.

**Source Packages folder**

The controller package contains the ControllerServlet which handles incoming requests by initiating actions needed to generate the model for the request, then forwarding the request to the appropriate view on the browser.



MVC diagram of the AffableBean application

The entity package contains the entity classes for each selected database table, complete with named query annotations, fields representing columns, and relationships representing foreign keys. You can see that all entity files that exist in the package exist in the database model except for OrderedProductPK.java. While creating the entity classes through NetBeans, the wizard generated this additional class. This is because the ordered_product table uses a composite primary key that comprises the primary keys of both the customer_order and product tables. Because of this, the persistence provider creates a separate entity class for the composite key, and *embeds* it into the OrderedProductentity.

The session package contains the entity class facades to create an EJB session facade for each entity class with basic access methods. A session facade abstracts the underlying business object interactions and provides a service layer that exposes only the required functionality. Thus, it hides from the client's view the complex interactions between the participants. The session bean (representing the session facade) manages the relationships between business objects. The session bean also manages the life cycle of these participants by creating, locating, modifying, and deleting them as required by the workflow. All of the generated session facades instantiate an EntityManager using the @PersistenceContext annotation. The @PersistenceContext annotation is used to inject a container managed EntityManager into the class. In other words, GlassFish EJB container opens and closes EntityManagers as needed.

**Transactional Business Logic**

In the AffableBean application the OrderManager.java file acts as the transaction manager. The transaction manager handles multiple write actions to the database as a single unit of work. It ensures that the work-unit is performed either in its entirety or, if failure occurs at some point during the process, any changes made are rolled back to the database's pre-transaction state