Project Report

Web Shop

Group: 15

Personnr:	*Name:*	*Email:*
910106-4039	David Harald	harry.dfh@gmail.com
851202-2016	Martin Bakkelunn	martin.bakkelunn@gmail.com
	Mathias Karlgren	
930528-7113	Viktor Landin	landinv@student.chalmers.se

Git translation table:

Alias:	*Name:*
SuspektZ	David Harald
Arseltomte	Martin Bakkelunn
matkarlg	Mathias Karlgren
Lozzer	Viktor Landin

Table of Contents

1 Introduction

This project has been carried out by four students in the DAT076 course. We decided to create a common web application, a simple web shop. However, instead of using already existing tools such as Drupal or Wordpress to automatically generate the website, we instead programmed it using modern Java technology.

The development of the website was done with support from multiple frameworks. At the bottom sits MariaDB, which is a community developed fork of MySQL. Google, Red Hat, Wikipedia and several other prominent companies have switched from MySQL to this RDBMS solution[1]. MariaDB is a binary drop in for MySQL and should not cause any compatibility problems.[2]

Hibernate ORM[3] is handling the persistent connection to MariaDB. Hibernate solves the problem of using high-level functions by mapping relational database tables to Java objects.

We decided to use a RESTful API design for our backend database solution. With a RESTful API we can easily build applications such as a website, android app, iphone or even a desktop application without modification to our backend. Only requirement is the ability to read/write JSON data.

AngularJS is used for interactions with the users and with the back-end.

1.1 User roles

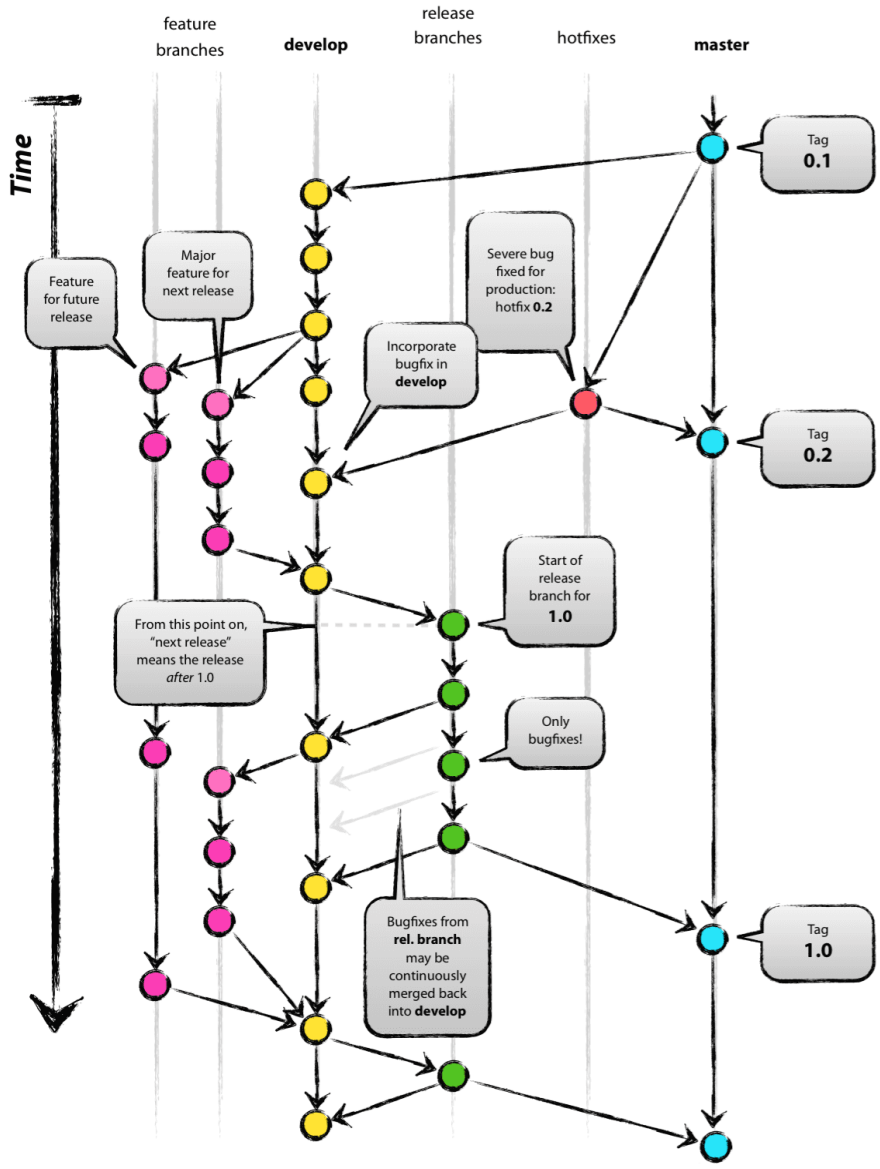
Role	**Permissions**
Guest	View product list, register an account
Customer	View product list, add/remove products to/from shopping cart, purchase the added products, change user credentials, display

[1] https://mariadb.com/
[2] https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-compatibility/
[3] Object-relational mapping library for the Java language

	past orders

2 Specification of requirements

2.1 Version control



We use this Git branching solution in our project. The master branch[4] should always have the latest stable release. Develop branch[5] can also be called alpha or nightly and is the primary working branch. All new features should be programmed in a feature branch until it's completed and then merged back into develop. Finally, when we are approaching release we

[4] Stable branch
[5] Can also be called alpha or nightly branch

create a release branch[6], from there we only accept bug fixes. When we deem this branch stable we merge into master and tag it. The project have not followed through with this branching model perfectly, but the underlying structure is there and should make it easier to understand the git repository.

2.2 Requirements

2.2.1 Functional requirements

USER STORIES

As a user...

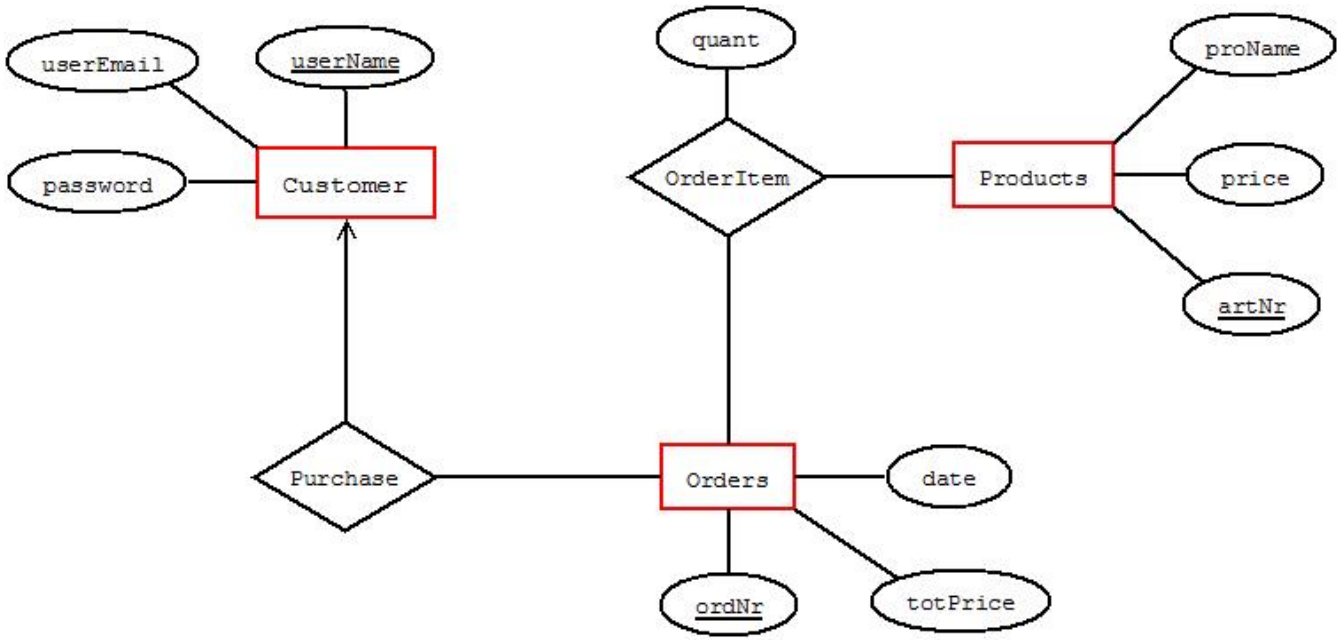
Priority	User Story	Estimate (%)
1	I want to be able to search for products by name, and sort the result by name or price	20
1	Add products to my cart	10
1	I want to be able to checkout and buy my products	10
1	Being able to login to my existing account	10
2	To view a product's details	10
2	I want to be able to contact the company	5
2	Register a new account	10
2	Change my account credentials	10
3	I want to be able to list previous orders I've made	15

2.2.2 Non-functional requirements.

Database: MariaDB will be used as our relational database management system (RDBMS).
ORM: Hibernate will be used for Object Relational Mapping (ORM). This will be used with the DAOs.
String encoding: Java standard UTF-8 will be used.

[6] Beta branch

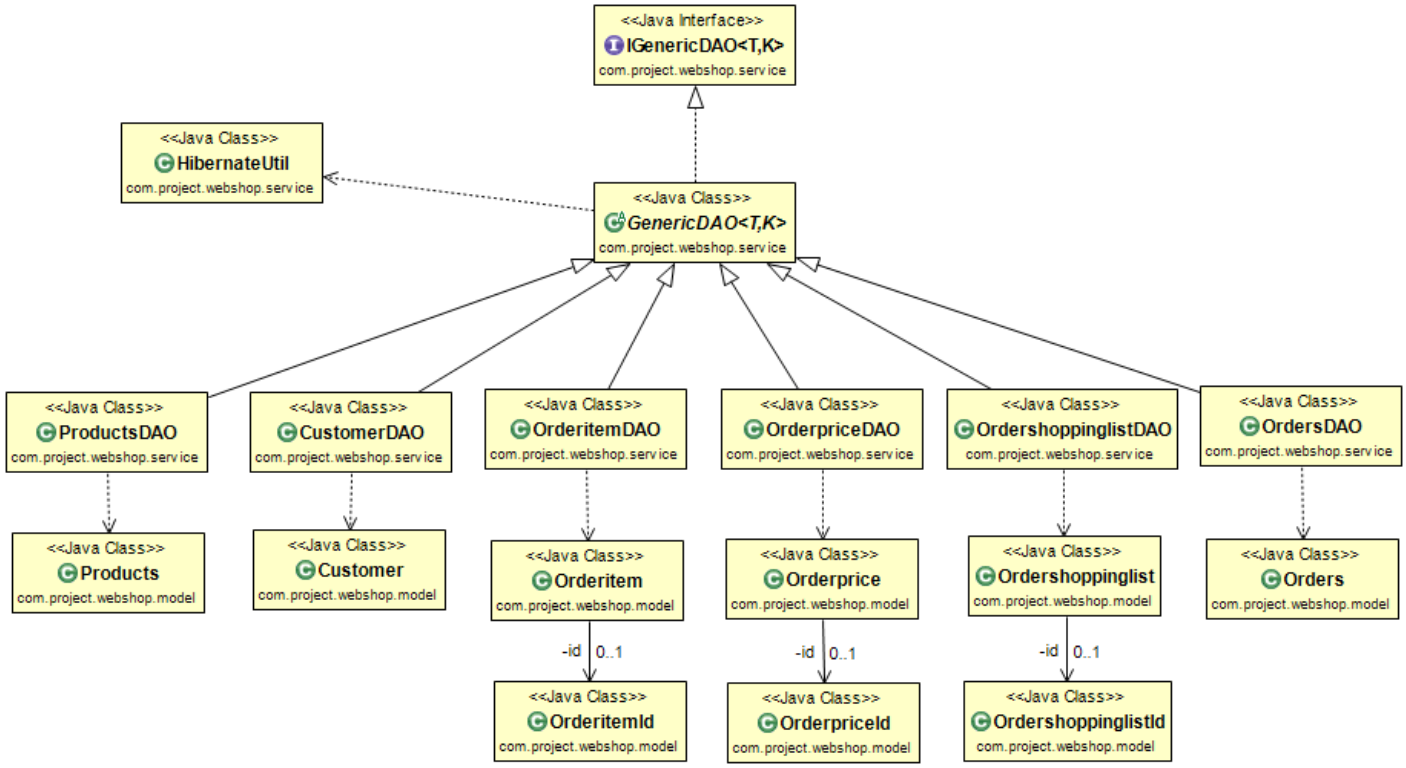
3.0 Domain analysis



Domain description: this database should contain all necessary data for the webshop project. The customer will register a username, a unique email and a password for his/hers account on the site. After that (currently) the customer will be able to buy products. This structure of the database will allow it to store a customer's orders with several products in each order. Products that has been selected by the customer will be then put in OrderItem with the orderNr declaring which order it belongs to and with that, the customer who has bought it. With this database there are also two views to more easily and neatly show orders with its items.

4.0 Design

4.1 UML for the REST-backend



HibernateUtil: Supplies a static method for easily creating a SessionFactory object. This utility class can further be extended with more methods useful for hibernate projects.

IGenericDAO: Interface defining the CRUD operations.

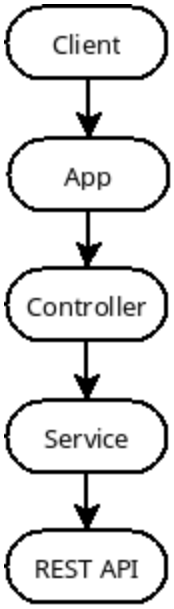
GenericDAO: Implements all CRUD operations and makes them available to the service layer.

Service package DAOs: A DAO object represents either a table or view in the database and makes it available through the RESTful API using Jersey.

Note about DAO_IDs: Hibernate requires each model to have an id field, but OrderItem is a relationship and doesn't contain an id. The other two are views. Our design creates an id field and moves the other attributes into an embedded model called, for example OrderitemId.

Model package: The database design mirrored with model files using hibernate. Orderprice & Ordershoppinglist are views and therefore read-only (only GET requests).

4.2 Flow chart for AngularJS powered web application



Services
WebShopProxy, CustomerProxy, OrdersProxy and OrderitemProxy provide proxies for interaction with the rest back-end.

CartService handles the shopping cart by maintaining a list of products as a cookie.

Auth handles user authentication

Controllers
Various different controllers for the web pages control pretty much everything related to user interaction.

5.0 Results

5.1 Errors encountered

Hibernate
The hibernate framework has support for creating all tables automatically. As part of this process hibernate also supports importing a SQL-file with custom commands. We experienced problems with UTF-8 encoding not working correctly for this file. The reason was because hibernate used an internal Java function for importing and this function used the default encoding on the system. This was only a problem for computers not defaulting to UTF-8 (Windows). Swedish letters didn't work and the solution was to create our own method and override the default, but we didn't follow through with this solution. Instead we changed the file to a windows friendly encoding.

This means all other systems[7], will have to change this file back to UTF-8 to avoid broken Swedish letters during the testing phase.

The hibernate function to automatically create and populate tables is only for testing purposes anyway. Of course, this feature will be removed when moving to a production environment.

AngularJS
Encountered strange behaviour when clicking the Login button, it must be clicked twice for the ng-if to react, does not occur when clicking Logout.

[7] UTF-8 configured systems. (Most *nix / Mac PCs and servers.)