Aalto University School of Science

Nintendo Game & Watch Shop

T-106.4300 Web Software Development

Final Report

March 11, 2011

Markku Laine, 48605D, markku.laine@gmail.com
Juha Loukkola, 57929V, jloukko@gmail.com
Kalle Säilä, 64775E, kjsaila@gmail.com

Table of Contents

1 Basic Information

This document is the final report for the course T-106.4300 Web Software Development (3-6 cr). The objective of the course was to develop a web shop using the Django framework. Our web shop, Nintendo Game & Watch Shop, is focused on selling handheld games. All 63 different games ever made in 12 series are available through the web shop. The web shop and its functionality are described in detail in the following Sections.

1.1 Project Size

The original plan was to complete the 5 cr version of the course. However, from the very start we decided that unexpected issues have to be taken into account in one way or another. Therefore, we first implemented all functional requirements for the 4 cr version prior moving on those of the 5 cr version. Personally, we think that this was a wise decision as we actually run into some unexpected issues, such as Kalle's weeklong work trip to the States just before the final deadline and Juha's busy schedule at the Aalto University.

Anyhow, we managed to implement all functional requirements for the **5 cr** version.

1.2 Group Information

Group information—including names, student numbers, and email addresses—can be found from Table 1.

Table 1: Group information

Name	Student Number	Email Address
Markku Laine	48605D	markku.laine@gmail.com
Juha Loukkola	57929V	jloukko@gmail.com
Kalle Säilä	64775E	kjsaila@gmail.com

2 Overall Description

The web shop has been divided into two main areas: Public and Administration. The structures of the Public and Administration areas are illustrated in Figure 1 and Figure 2, respectively.

In the Public area, users are allowed to search and browse products (games) arranged by categories (series), read product details as well as comment and rate them. Products can be added to a shopping cart and when ready, the user can checkout to purchase his/her selected products. Users can also register to the system, which might become handy when shopping often in our web shop, as then they do not need to fill in their user information every time they place an order. Another

benefit from the registration is that the user is able to browse his/her completed orders as well as change other account related information, such as user's shipping addresses. The web shop also has the About and Credits pages, which describe basic information about the web site. Furthermore, for web shop staff, there is a special view for handling order and viewing statistics. Staff can also delete product comments from their respective views.

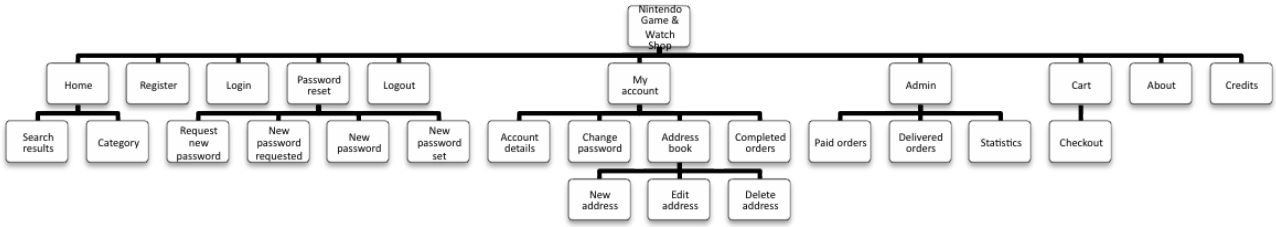

Figure 1: Nintendo Game & Watch Shop conceptual web site diagram of the Public area

The Administration area, on the other hand, is accessible only to users with the staff role. The area contains necessary tools for managing users, their addresses and orders, products, sale items, types, comments, shipping methods, etc. All views have been customized in order to provide better usability/tools for the staff.

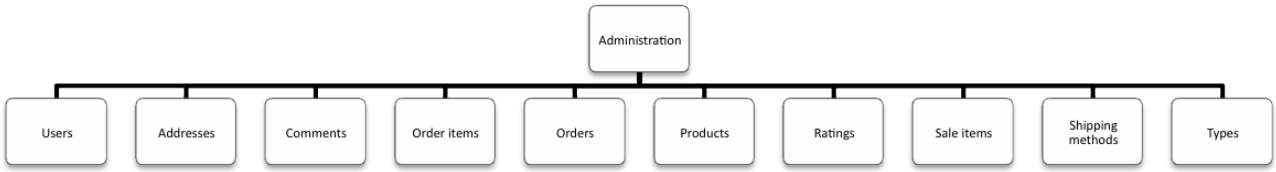

Figure 2: Nintendo Game & Watch Shop conceptual web site diagram of the Administration area

3 Who Did What?

This section describes who did what in the project. In the following subsections, each group member has described his work in the project. More detailed information about who did what and when is available at: https://code.google.com/p/nintendo-gw-web-shop/updates/list

3.1 Markku Laine

As stated in the project plan, my main responsibility was to collect and produce the content for the web shop. This included collecting information about the games and taking photos of each game as well as refining the content (editing texts and manipulating photos). This process took a lot longer than expected! In addition, my main responsibility included designing the layout for the web shop.

What comes to the coding part, my responsibility was to implement authentication (register, login, password reset, and logout) and session related functionality as well as my account and its sub pages. I also implemented admin's paid orders and delivered orders pages. Furthermore, I participated in XHTML, CSS, and JavaScript coding a lot, mainly by modifying/adding CSS rules as well as structuring our project into proper templates.

3.2 Juha Loukkola

My original main responsibility was to design the models. The implementations of these models were done by everyone and changes and additions were made when needed. My other responsibilities were interfacing the web bank and handling a shopping cart. These parts are where I did most of my coding.

As a newbie at browser-side techniques it were great opportunity for me to learn from seasoned web developers like Markku and Kalle.

3.3 Kalle Säilä

My main responsibility was the client-side part of the application (HTML, CSS, and JavaScript/jQuery). Basically this included the bases for the user interface, style and client-side functionality on code level based on the layout designed by Markku Laine. So the lion's share of my work consisted of implementing the whole application related HTML pages, CSS style sheet and JavaScript.

In addition to the whole application related code, I implemented the home, search and category pages for the most parts. Also, like all of us, I did some code and CSS fixing here and there where others needed help. As far as the server-side goes, we all implemented some of the views, URL configurations and models.

4 Project Functionality

4.1 How to Setup the Software

In addition to software listed on the slide 11/14 of the lecture 16, the deployment environment has to support the following external libraries:

- libjpeg 8c[1]: For supporting image uploads (ImageField) in the JPEG format in the administration user interface
- Python Imaging Library (PIL) 1.1.7[2]: For supporting image uploads (ImageField) in the administration user interface

For more detailed information regarding the installation of the aforementioned software on Mac, please refer to the following tutorials: http://paul.annesley.cc/2007/11/django-and-python-imaging-library-pil-on-leopard and http://stackoverflow.com/questions/1398701/problems-with-snow-leopard-django-pil.

The fixtures for the web shop are located in the fixtures folder. The fixtures provide the following initialization data for the web shop: types (product categories), products (games), sale items (mainly product prices), and shipping methods (standard and express). The data in the fixtures are imported to the database, when the *syncdb* command is executed.

4.2 How to Use the Software (and What Kinds of Functionality is There to Use)

The main URLs (and pages descriptions) of the web shop are listed in Table 2. In addition, URLs starting with *ajax/* are used for used for Ajax calls.

Table 2: List of web pages with details

Name	URL	Description
Home	home/	Project root
Search results	home/search/	Show search results
Category	home/category/{type_id}/	Show products of the category
Register	register/	User registration
Login	login/	User login
Password reset	passwordreset/	Password reset root
Request new	passwordreset/requestnewpassword/	Request new password

[1] libjpeg 8c, Software, http://www.ijg.org/files/jpegsrc.v8c.tar.gz

[2] Python Imaging Library (PIL) 1.1.7, Software, http://www.pythonware.com/products/pil/

password		
New password requested	passwordreset/newpasswordrequested/	Success message
New password	reset/{uidb36}-{token}/	Enter new password
New password set	passwordreset/newpasswordset/	Success message
Logout	logout/	User logout
My account	myaccount/	User account root
Account details	myaccount/accountdetails/	Edit user's account details
Change password	myaccount/changepassword/	Change user's password
Address book	myaccount/addressbook/	List user's addresses
New address	myaccount/addressbook/new/	Add a new address to the user
Edit address	myaccount/addressbook/{address_id}/edit/	Edit user's address
Delete address	myaccount/addressbook/ {address_id}/delete/	Delete user's address
Completed orders	myaccount/completedorders/	List user's completed (paid) orders
Admin	admin/	Admin root
Paid orders	admin/paidorders/	List paid orders
Delivered orders	admin/deliveredorders/	List delivered orders
Statistics	admin/statistics/	Show web shop statistics
Cart	cart/	Show shopping cart
Checkout	cart/checkout/	Review the order before paying
About	about/	Information about the web shop
Credits	credits/	List of used resources
Admin	/admin	Django administration UI for managing the web shop

Web shop functionality from the user's perspective has been described in Chapter 2. A couple of screenshots taken from the web shop are presented in the figures below.

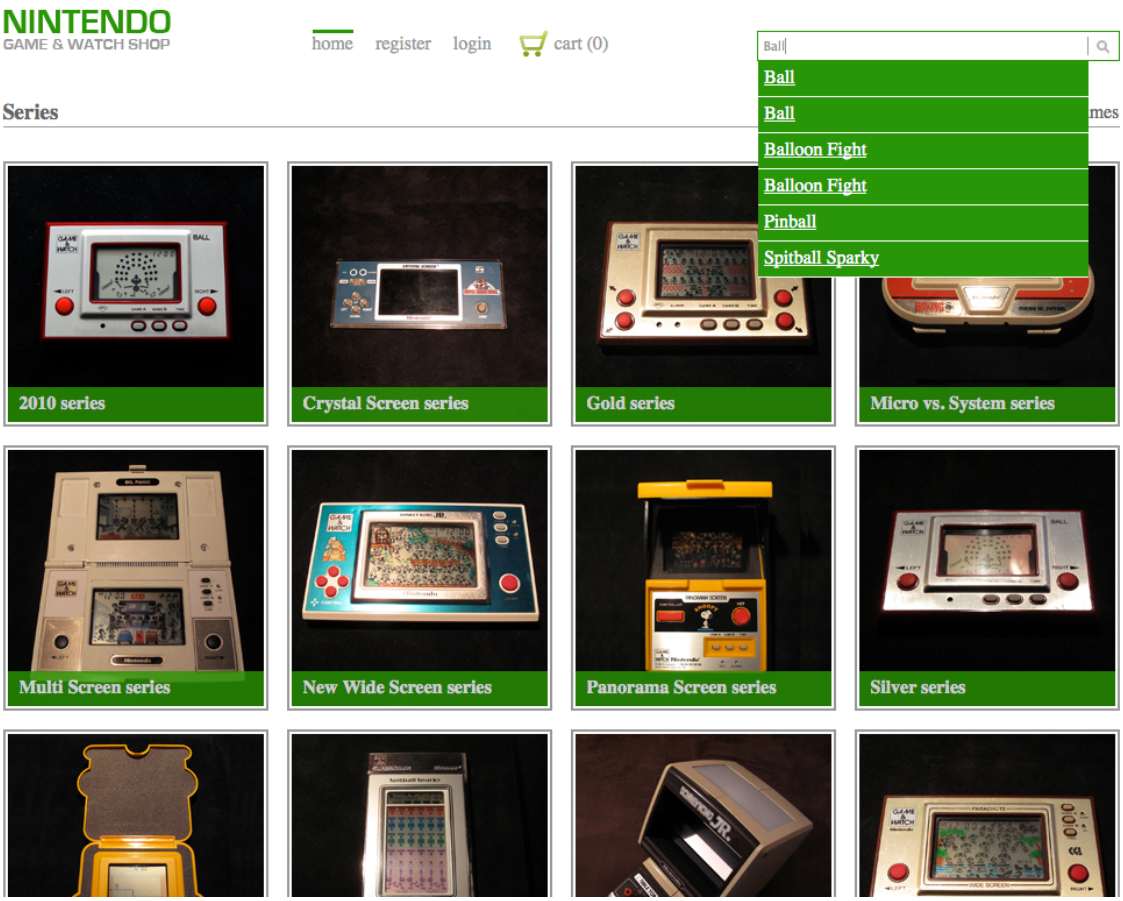

Figure 3: Home page with autocomplete search field

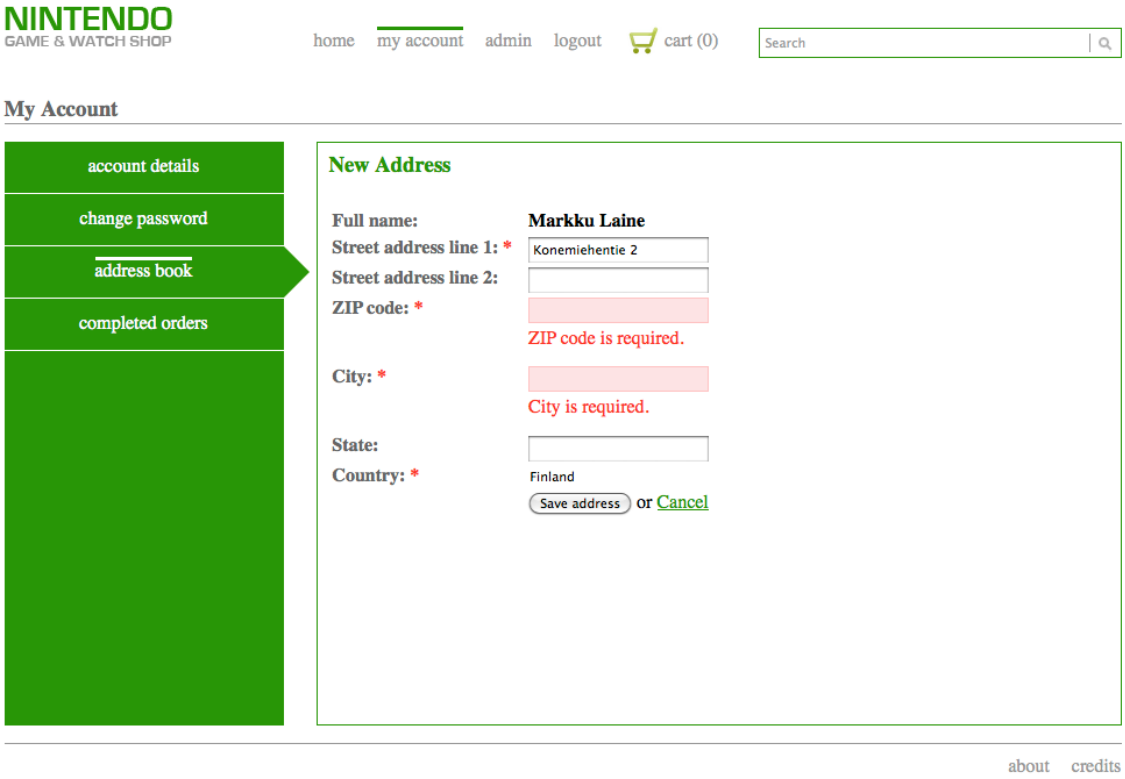

Figure 4: New address page with client-side form validation

Figure 5: Statistics page with products statistics

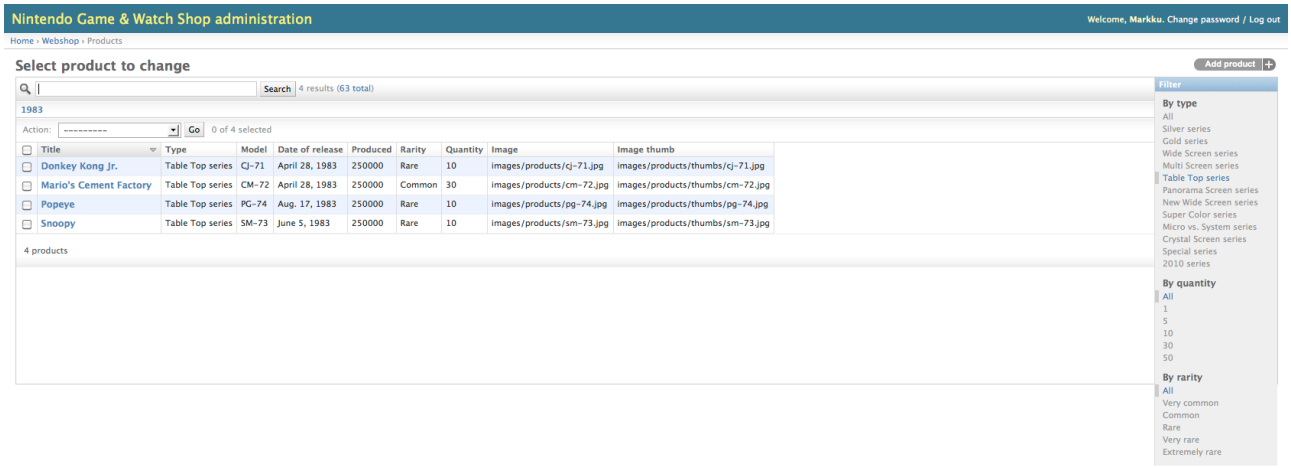

Figure 6: Products page in the Administration area

4.2.1 Usernames and Passwords

- **Administrator:** administrator/admin1234
- **User:** user/user1234

5 Program Structure

Our program, as Django programs usually, is structured to follow the common model-view-controller pattern. The model part consists of the objects needed to represent and manipulate the application data. The view part (called template in Django) defines the user interface structure and the controller part (called view in Django) communicates between model and template by for example creating new objects and retrieving model data to the templates. The final structure of the program is represented in Figure 7 below. The actual application package (webshop) contains only the python files (and the fixtures) needed for the application. The static files, like CSS and images, the templates and the database files each have their own folder at the root level.

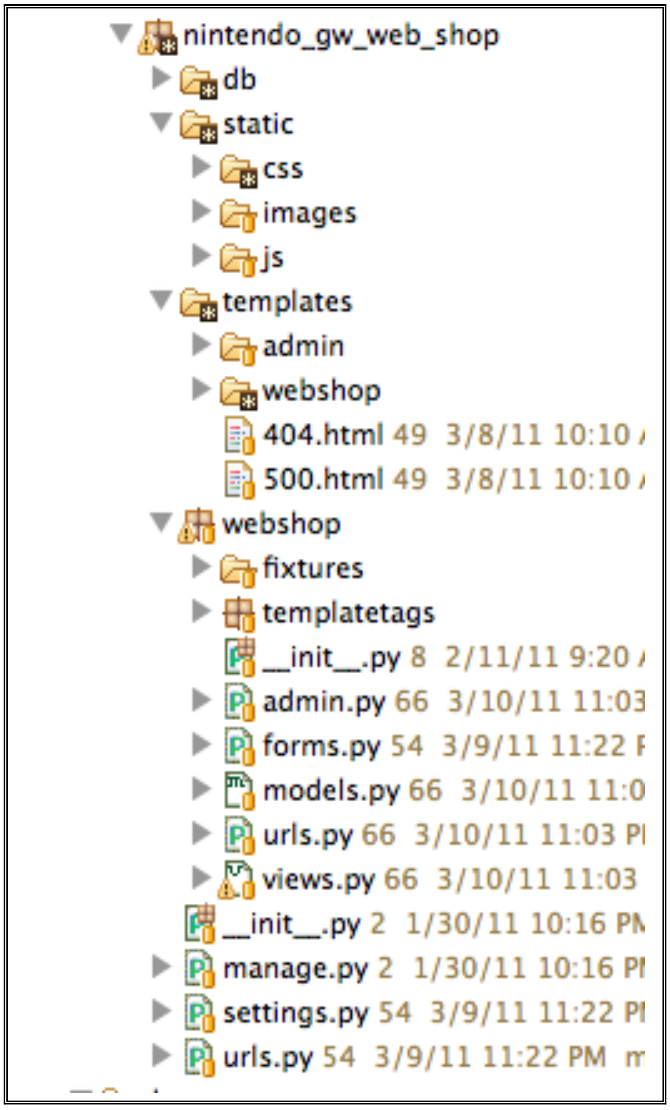

Figure 7: Program structure

5.1 Model

Our model consists of the following objects:

Table 3: List of models

Name	Description
User	Represents a single user (provided by Django).
Address	Represents a single address of a user.
Type	Represents a predefined category of products.
Product	Represents a single product (game).
Comment	Represents a product comment.
Rating	Represents a product rating.
SaleItem	Represents an information block of the current price of a product. One product could have several sale items but only one is valid at a time.
Order	Represents a single order.
ShippingMethod	Represents a shipping method for an order.
OrderItem	Represents a product and quantity of that product in one order.
Statistic	Represents a block statistics collected during a user session.

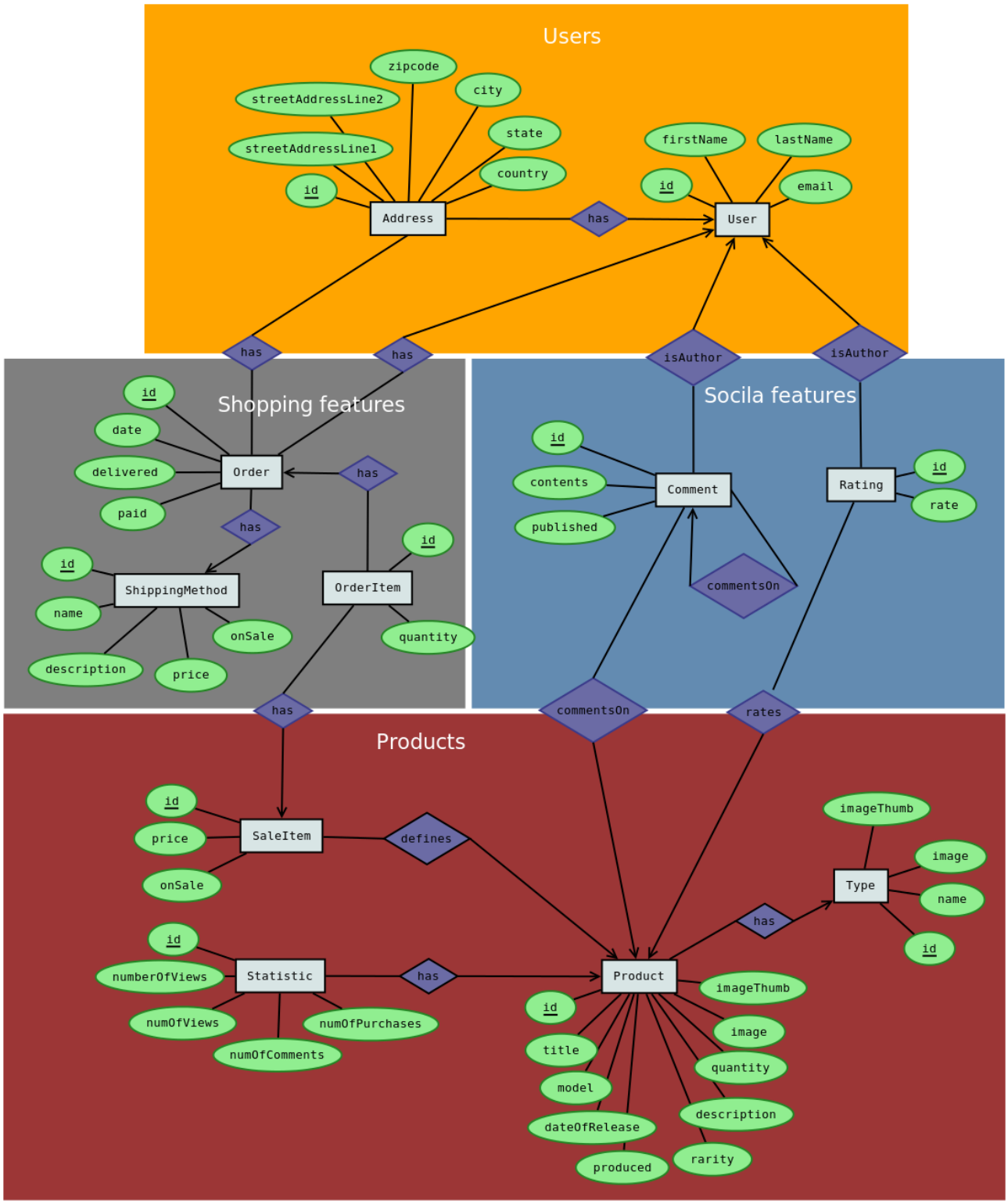

Figure 8: An Er-diagram of the implemented database schema

5.2 Template

Templates, as stated before, define the structure of the user interface. In our application we have one base template (base_template.html) that works as a basis for the other templates. The base template defines the blocks common to all of the templates. For example logo, main navigation and search elements are shown in every page. In addition, we have a couple of smaller templates that define elements used in more than one page. Templates are also organized inside the program in suitable categories. For example, admin templates have their own folder.

5.3 View

The views of our program are all defined in one file (views.py in webshop package). Below is an example of one of our views.

```
@login_required
def myaccount( request ):
  # Handle GET requests
  if request.method == "GET":
    return redirect( "webshop.views.account_details" )
  # Handle other requests
  else:
    raise Http404( "%s method is not supported." % request.method )
```

5.4 Ajax

Ajax is used in several places in our program. The main reason for using Ajax was to increase the usability of our web shop by reducing the amount of page loads. The most important uses of Ajax are updating the order (cart), adding/removing comments, modify ratings and providing auto complete in search field.

5.5 JavaScript

Our program relies on JavaScript in many functions. For example the product displayed in the category page is chosen with JavaScript. The display mechanism uses hashes in the URL, which makes the URLs a bit messy but provides unique URLs for each product even if the JavaScript is disabled. All in all, our program is usable without JavaScript but the user experience is not that nice and some of the not so important functionality, like commenting or rating, is hidden with CSS.

6 Known Faults and Missing Features in the Program (if any)

Below is a list of missing/incomplete features of the web shop:

- Paging would have been nice on “Paid Orders” and “Delivered Orders” as well as “Completed Orders” pages.
- Adding a breadcrumb would have helped navigating within the web shop.

- Adding progressive enhancements to various pages (e.g., Address Book) would have been a nice feature.
- The performance of the JavaScript in category page is far from optimal.
- In category page, clicking an auto complete link of a product in same category does not work.
- [TODO]

7 The Three Best and the Three Worst Parts of the Code

7.1 Markku Laine

The best part of the code: I would say the "Address Book" pages. In my opinion, the implementation of the *AddressForm* form is pretty elegant because it utilizes the *label* and *max_length* information from the *Address* model in its error messages, which again override the default error messages. In addition, the "Address Book" pages allow a user to perform all CRUD operations.

The worst part of the code: I tried to figure out how to properly use Django's built-in views and forms. I never fully understood the logic behind those mainly due to the lack of documentation. In the end, I managed to implement the "Forgot Your Password?" functionality, but still not very proud of how it has been implemented.

7.2 Juha Loukkola

[TODO]: Each group member should describe one best and one worst part of his code.

7.3 Kalle Säilä

The best part of the code: I would say the CSS layout of the application as a whole and especially the category page since it contains quite a lot of elements that need to be displayed nicely.

The worst part of the code: The worst part of my code is the jQuery code in the category page. Although, I'm satisfied about the functionality achieved by this code and it works quite smoothly most of the time, but performance vis it is far from optimal and should be refactored.

8 Diversions from the Original Plan

We followed the original plan until the check-point submission. However, after that we realized that our plan had not taken into account some of the views/functionality needed, such as managing user's addresses and profile information to name a few. It was also unclear how administration user interface was supposed to be implemented, i.e., what was actually required/allowed to do in order to fulfill the requirements. Anyway, after the check-point submission we divided the rest of the work into three parts, each group member being responsible for his own part. The parts were as follows: authentication and access control as well as my account (Markku); home and category/product pages including commenting and rating (Kalle); and cart & checkout including payment (Juha).

From that on, the work continued as stated above. Markku finished his part earlier than Kalle and Juha so he had extra time to concentrate on implementing administration user interfaces. Kalle had to postpone some of his work due to a weeklong work trip. Juha also had to postpone his work to a later time due to other courses. All in all, the amount of work required for the project and tight timetable were very surprising for us.

9 References

Books and web sites:

- Course web site: https://noppa.tkk.fi/noppa/kurssi/t-106.4300/etusivu
- Python v2.6.6 Documentation: http://docs.python.org/release/2.6.6/
- Django Documentation 1.2: http://docs.djangoproject.com/en/1.2/
- The Django Book 2.0: http://www.djangobook.com/en/2.0/
- Nullege: http://nullege.com/codes/show/src%40p%40y%40python_blog_engine-HEAD
- Google Code: https://code.google.com/
- Eclipse: http://www.eclipse.org/
- Pydev: http://pydev.org/
- Subversion: http://subversion.tigris.org/
- Base 2: http://menial.co.uk/software/base/

- jQuery: http://jquery.com/

JavaScript libraries:

- jQuery 1.4.2[3]: For simplifying web development on the client-side
- jQuery UI 1.8.9[4]: For advanced effects and widgets in the user interface
- jQuery Validation plugin 1.7[5]: For validating forms on the client-side
- jQuery Quicksand plugin[6]: For sorting series
- jQuery Flip! plugin[7]: For flipping the product information
- jQuery Simple Dialog[8]: For displaying bigger product image
- Modernizr 1.5[9]: For better browser support
- HTML5 JavaScript shiv[10]: For recognizing and styling the HTML5 elements in Internet Explorer
- Highcharts JS[11]: For statistics

Snippets of code:

- Nicer Django Messages Output[12]: For rendering nice Django messages

Content:

- Mike's Nintendo game&watch forum - FAQ[13]: For gathering/double checking information about Nintendo Game & Watch games

[3] jQuery 1.4.2, Software, http://jquery.com/
[4] jQuery UI 1.8.9, Software, http://jqueryui.com/
[5] jQuery Validation plugin 1.7, Software, http://bassistance.de/jquery-plugins/jquery-plugin-validation/
[6] jQuery Quicksand plugin, Software, http://razorjack.net/quicksand/
[7] jQuery Flip! plugin, Software, http://lab.smashup.it/flip/
[8] jQuery Simple Dialog, http://www.mudaimemo.com/p/simpledialog/
[9] Modernizr 1.5, Software, http://www.modernizr.com/
[10] HTML5 JavaScript shiv, Software, https://code.google.com/p/html5shiv/
[11] Highcharts JS, http://www.highcharts.com/
[12] Nicer Django Messages Output, Snippet of code, https://github.com/mrben/nicer_django_messages_output
[13] Mike's Nintendo game&watch forum – FAQ, Web site, http://mpanayiotakis.proboards.com/index.cgi