Webshop

Student: Szekely Attila

**Group: 30236**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

This is a client-server desktop application, which realizes a simple webshop for selling electronic products (min. 3 categories: telephones, laptops, etc). It is multi-user, there are many normal users and admins.

# Functional Requirements

The application will have 2 types of users: normal user and administrator. Login must be implemented.

User operations:

* Show list with all products on the main page
* Show category list on left side
* Click on a category renders that category’s page, where only product from that category are shown.
* Hierarchic list of categories

Admin operations:

* Add/remove products
* Add/remove categories
* Add products to a given category
* All categories can have children. Add/remove subcategories.

Non-functional Requirements

* Use one of the following MVC frameworks:
  + Java Spring MVC
  + ASP .NET MVC
  + (Ruby on Rails)
* Build data access layer based on JDBC/ADO.NET
* Data will be stored in one of the following databases: MySql / Sql Server Express / Oracle Express
* Implement data access layer with an Object Relational Mapping (ORM) framework
* Java: Hibernate
* .NET: Entity framework
* (RoR: ActiveRecord)
* Use composite desing pattern in implementation

Additional requirements in the 2nd iteration:

* Upload excel files which contain informations about products, process them and save in the database.
* Upload images for every product.
* Caching:
  + Use Chain of Responsibility or Strategy pattern to implement caching on categories.
  + Informations in cache expire after 30 seconds.
  + Logic: When informations about categories are requested, firstly are searched in cache, if they don’t exist or are expired, will be read from the database, stored in the cache and cache expiration will be reseted.

2. Use-Case Model

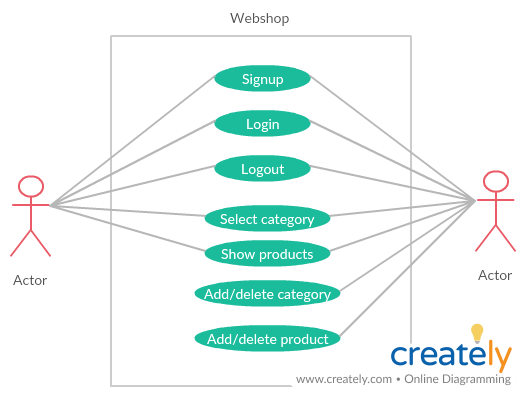
**Use case:** list products

**Level:** user-goal level

**Primary actor:** normal user

**Main success scenario:** login, choose category, the system shows products from that category

**Extensions:** if there are no product in that category, it shows an alert message in the browser.



3. System Architectural Design

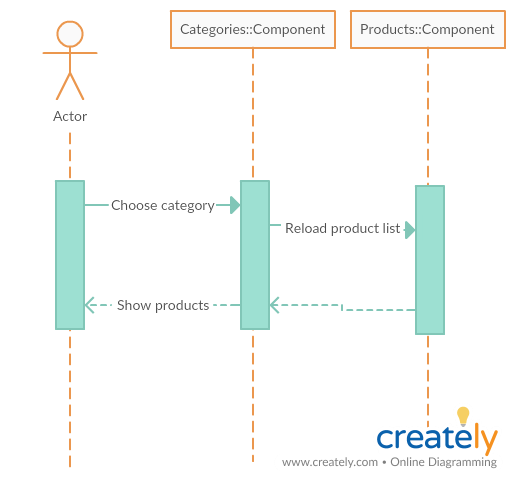
**3.1 Architectural Pattern Description**

The application consists of 2 parts:

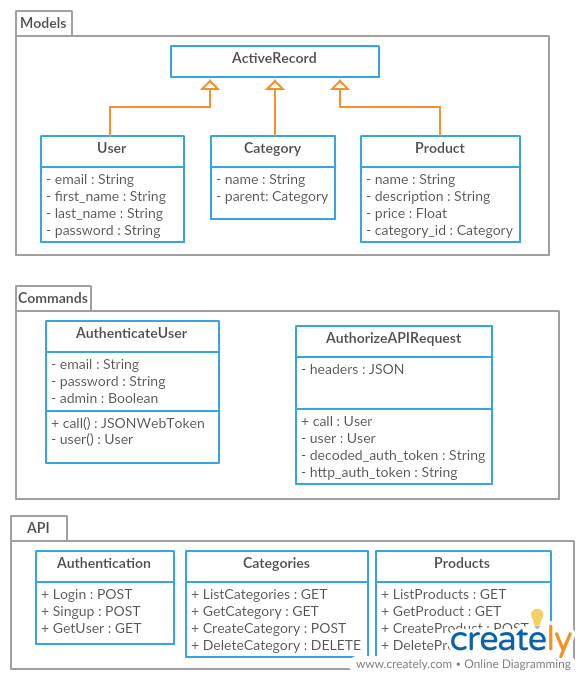
* Server side: Ruby on Rails
  + Database management
  + Model
  + Grape API endpoints
* Client side: React JavaScript
  + View component
  + Access API enpoint with Axios

**3.2 Diagrams**

4. UML Sequence Diagrams



5. Class Design

****

**5.1 Design Patterns Description**

Composite design pattern: hierarchical categories.

Chain of Responsibility design pattern: if information about categories exist in cache they will be loaded from cache, else information will be load from the database and saved in the cache.

**5.2 UML Class Diagram**

6. Data Model

The application database has 5 tables:

* Users (email, first\_name, last\_name, password, admin, image\_b64): contains both the admin and normal users(admin: false – user, true – admin)
* Categories (id, name, parent)
* Products (id, name, price, description, category\_id)

7. Second iteration requirements

1. Caching on categories:

Caching is implemented in the back-end layer: when information about categories is requested, Rails checks if the information exists in the cache. If it exists loads the information from the cache, otherwise loads it from the database, stores in the cache and resets the expiration time (30 seconds).

This type of caching reduces the number of accesses to the database, so it reduces the overall response time of the application.

Information in cache expires automatically after 30 seconds and expires when a new category is created or an existing one is changed.

1. **Upload images to products**

Images are uploaded in the NewProduct React component and they are encoded in Base64.

The database stores a single String, which contains the Base64 encoded image.

1. **Upload products from Excel file**

I used the CSV file format. One line in CSV is composed of 4 field: name, price, description and url to the product image.

CSV file is encoded in Base64 format and sent as a String JSON. On the back-end the file is decoded and parsed. The CSV is read line-by-line and for each line a new product is inserted (the image is downloaded from the internet).

8. System Testing

The system was tested manually. All admin and cashier functionalities were tested.

9. Bibliography

1. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>
2. <https://www.w3schools.com/sql/>
3. https://en.wikipedia.org/wiki/Multitier\_architecture