



POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Baz Danych

Specjalizacja Bazy Danych

Michał Kulesza, 8537

Julia Osiak, 8732

Arthur Plonski, 8846

Oak Search – Wielobazowa aplikacja internetowa do wyszukiwarki sprzętu elektronicznego.

Praca inżynierska

Promotor: Dr Ida Jokisz

Warszawa, czerwiec, 2014



**POLSKO-JAPOŃSKA WYŻSZA SZKOŁA
TECHNIK KOMPUTEROWYCH**

Faculty of Computer Science

Chair of Databases

Major: Databases

Michał Kulesza, 8537

Julia Osiak, 8732

Arthur Plonski, 8846

**Oak Search - A Multidatabase Web Application for
Electronic Equipment Search Engine.**

Bachelor of Science Thesis

Supervised by Dr Ida Jokisz

Warsaw, June, 2014

Streszczenie

Niniejsza praca stanowi dokumentację systemu do wyszukiwania i licytowania ofert sprzętu elektronicznego. System został stworzony używając wielu różnorodnych technologii i rozwiązań, które zostały dokładnie omówione. Przedstawione zostały także fazy powstawania projektu, zmiany zachodzące w funkcjonalności i możliwościach udostępnionych użytkownikom. System składa się z trzech części przeznaczonych dla poszczególnych użytkowników: administratora, właściciela sklepu i użytkowników przed i po rejestracji.

W pierwszej części publikacji omówiona jest struktura projektu: aktorów, funkcjonalności i wykorzystanych technologii. Najważniejszą i najobszerniejszą częścią pracy jest dokładne omówienie architektury aplikacji, trzech baz danych i ich zależności oraz sposobów komunikacji między nimi.

W kolejnej części przedstawiony jest rozwój projektu z wyszczególnieniem kolejnych etapów, ich początkowych założeń, napotkanych trudności i ich rozwiązania.

Osobny rozdział przeznaczony jest do opisania fazy testowania projektu i otrzymanych wyników.

Następnie znajduje się część dotycząca możliwości przyszłego rozwinięcia projektu a także potencjału projektu do wykorzystania zastosowanych rozwiązań w innych aplikacjach o podobnej architekturze i zapotrzebowaniu rynkowym.

Na koniec zamieszczone są dodatkowe materiały dotyczące stworzonej aplikacji: instrukcja instalacji i obsługi systemu.

Abstract

This Bachelor's thesis is the documentation of a web application designed for searching and bidding of electronic equipment. The system was created using various technologies and IT solutions which will be discussed in great depth. This document also describes the development process with all changes that occurred to functionalities and possibilities presented to the users. The application consists of three parts designed for needs of different users: administrators, store owners and registered users.

First part of this thesis discusses the project's structure: actors, functionalities and used technologies. The most important and extensive chapter is dedicated to thorough explanation of application architecture, the three databases that were used, their communication and dependencies.

Next chapter describes the development process of the project. Specific stages are distinguished with their initial assumptions, problems that occurred and solutions used to overcome them.

Following chapter focuses on the testing phase and obtained results.

Next part of the documentation brings the reader closer to possible future development of the project. It also describes the project's potential for influencing other applications with similar architecture and range of services.

Finally, this document contains additional project-specific materials: installation process and user manual.

Contents

Streszczenie.....	3
Abstract	4
1. INTRODUCTION	7
1. ANALYTICAL PART.....	8
1.1. Description of Use Case Diagram.....	8
1.2. Used Technologies	14
2.2.1 Oracle DB	14
2.2.2 Java and Tomcat	15
2.2.3 Neo4J	15
2.2.4 Cypher.....	17
2.2.4 Mongo DB	18
2.2.5 Ruby on Rails.....	19
2.2.6 Subversion and TortoiseSVN	19
2.2.7 Web Applications dependencies	20
2.3 APPLICATION ARCHITECTURE.....	23
2.3.1 Oak Front Module.....	26
2.3.2 Oak Admin Module	33
2.3.3 Oak Store Module.....	36
2.3.4 Oak DBSync Module.....	38
2.3.5 Oak PriceWatcher	44
2.3.6 Search Engine and Natural Language Parsing.....	47
2.4 Oracle Database	49
2.4.1 Entities Relation Diagram.....	49
2.4.2 Other Database Objects.....	54
2.5 Neo4J	60

2.5.1 Designed Architecture	60
2.5.2 Implemented Solution.....	62
2.6 MongoDB - Statistics in application.....	63
2.6.1 Schema structure	63
2.6.2 Usage.....	64
3. DEVELOPMENT PROCESS.....	66
4. TESTING.....	72
4.1 Performed tests and obtained results.....	73
4.2 Test Results	76
5. FUTURE DEVELOPMENT AND USE	76
5.1 Facilitation and automatisisation of adding new products.....	76
5.2 Store ratings.....	77
5.3 Security.....	77
5.4 Backup plans	78
5.5 Mobile version.....	78
5.6 Admin API	78
6. FINAL NOTE	79
7. INDEX.....	80
8. BIBLIOGRAPHY	81
9. TABLES AND FIGURES	83
List of Code Listings.....	83
List of Figures	84
List of Tables	85
ATTACHMENTS.....	86
USER'S MANUAL	87
Installation Guide.....	120

1. INTRODUCTION

The application OAK Search is a project created as an answer to our own needs. It is a tool that provides users with a unique service of contacting online stores and obtaining a personalized buying offers. Users can search through available products, compare them and after choosing what they want they can create an auction where stores will present their offers to the buyer.

The designed search engine allows users to browse through products that match exactly their selected criteria. However if the 'exact search' option is not selected, the system will show products that have similar features, thus giving the user more possibilities, and providing the stores with more marketing.

The users also have the ability to mark and 'watch' a product that they are interested in. This provides them with an easier access to available information about this product along with statistical data showing fluctuation of the price and popularity of the products based on number of page views.

Another unique feature of OAK Search are the Black Lists. This will prevent the user from contacting a store that they dislike and present them with offers that are provided by stores that seem more reliable and friendly to the user.

All users have the ability to contact each other as well as the stores through the messaging feature, which facilitates obtaining information about products.

The application incorporates various kinds of databases, each specifically designed for different operations in the application.

1. ANALYTICAL PART

1.1. Description of Use Case Diagram

1.1.1. List of Functionalities

The following list is the final version of functionalities implemented in our system. It is divided by user groups to which they are addressed. The users groups will be discussed in more detail later on.

Guests (Unregistered Users):

- Search history held for a week
- Product search
- Registration
- Comparing 2 chosen products

Registered Users:

- Login/logout
- Product search
- Commenting on products and stores
- Personalized search history (the user is able to view it with more details)
- Receive and send messages
- Adding products to 'Watch List'
- Rating products and stores
- Change own profile
- Black-listing stores
- Compare 5 chosen products
- Create an auction

Store Owners:

- Login/logout
- Receive and send messages
- Change own profile
- Create store profile (admin authorization)
- add/remove/edit products
- change company profile
- respond to user requests (auctions)

Administrators:

- everything related to website and its content

DB Administrators:

- total control of all data and data access in the databases

1.1.2. ACTORS

GUEST (Unregistered User)

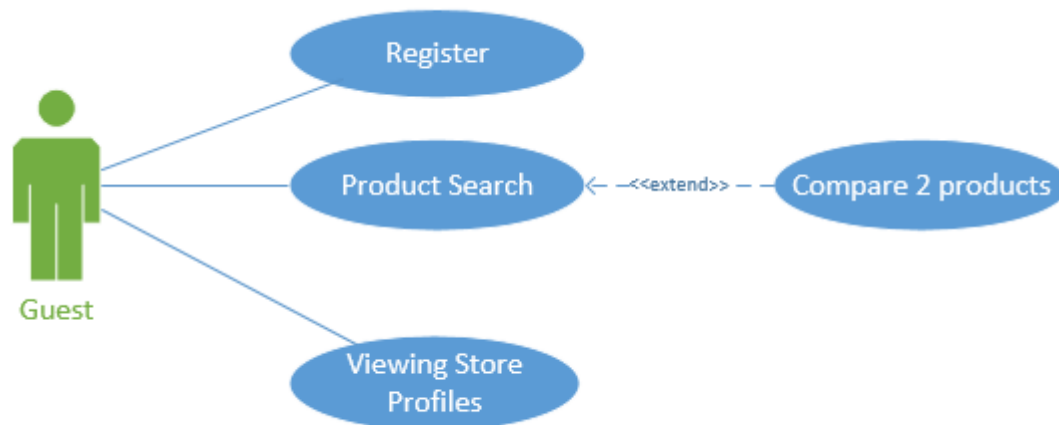


Figure 1. Guest Use Case Diagram

We classified guest as a random person that is directed to our site or stumbled across our site. The guest can browse all products listed and he will be able to compare any 2 products he wishes. He will be able to view any store profiles our registered users put up. This entails viewing and the possibility of purchasing any products the store owner has showcased. Lastly the guest will have the option of becoming a registered user, which will give him immense benefits.

REGISTERED USER

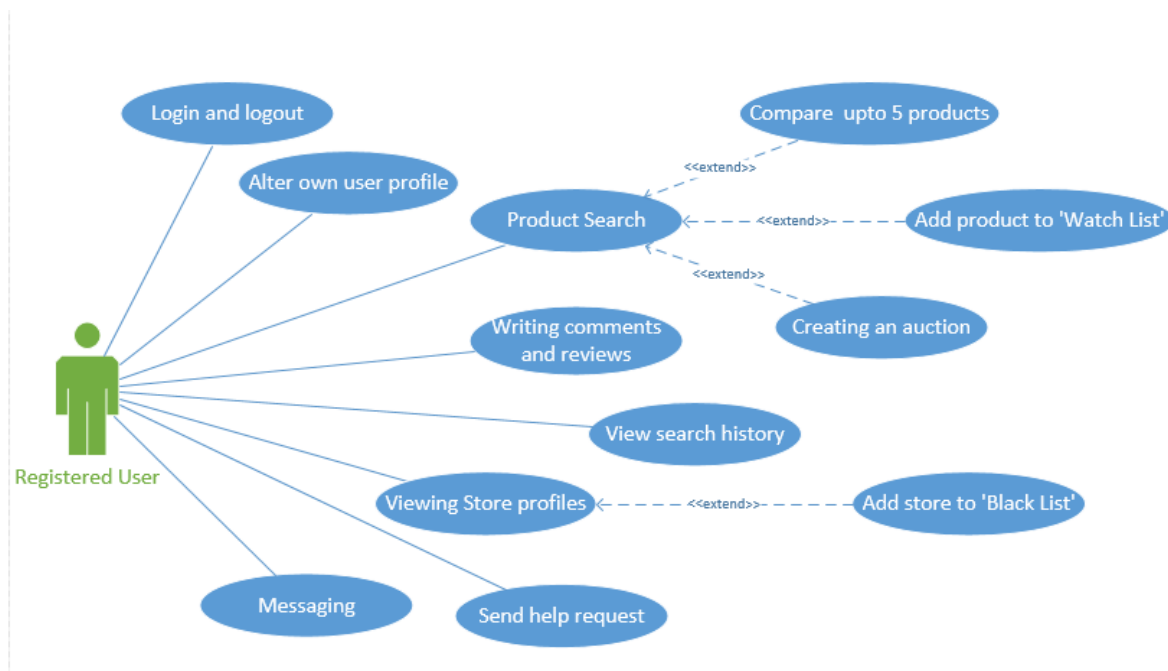


Figure 2. Registered User Use Case Diagram

The first and foremost option that the user will have is a username and a password which he can log into our system with. In this profile he will be able to make any alterations to his preference or liking. During our registered users search he is entailed to compare up to 5 products, add products to his watch list, create an auction for products he wishes to purchase, and write reviews for the products he has purchased or has insight on. Since we will keep track of the users search history he will be able to browse anything they visited previously, the history will be held up to 30 days. While the user browses store profiles on our system he will be able to search for products they offer as well as blacklist the store for his own preference. He is able to contact the admin for any help they may desire in their search process and send messages to store owners or friends that are also registered users.

STORE OWNER



Figure 3. Store Owner Use Case Diagram

This actor will have the ability to log in and out of his own personal profile using a username, email and password, with this he is able to make any desirable changes they may have to their own profile or the stores and create additional store profiles. Being a store owner on our system the user has the ability manage his products in the following ways: adding products, removing products, and updating products. This user may send a help request to our admin with any questions that occur. Just like our registered user the store owner can message any registered user that may have questions and any friends that have logged into the system. Lastly this user is capable of making a product offer for any users that enters an auction.

ADMIN



Figure 4. Admin Use Case Diagram

The admin has full control of the system after his login into his profile with the correct login and password. He may send direct messages to whomever he pleases regarding any promotions or abuses of the system. He is able to alter any products through removing them, adding new ones, or updating old ones. The admin has the power to remove registered users and store owners if they do not follow the guidelines of our system. Along these lines he/she has the authority to remove any stores that do not follow the guidelines or create new ones. The admin has the power to remove any auctions and answer any help questions that may come their way.

1.2. Used Technologies

2.2.1 Oracle DB

The core of our project is the relational database. There are many available, free to use SQL database tools to choose from however we decided that Oracle would be the best choice for us. We decided to use the Oracle Database 11g Express Edition as it is the first and only free version provided by Oracle. It is based on the Oracle Database 11g Release 2 base code and provides the necessary tools to develop a small scale project.

Oracle XE is easily compatible with various developing tools and languages. Java, one of the tools that we were most familiar with, has a built in library that facilitates connecting and operating with Oracle.¹ Even though not all features are available in the XE, we were able to fully implement our ideas. We were able to incorporate function-based indexes, temporary tables, stored functions and procedures. There also many features that will be beneficial in future development, regarding security and backup management.

Another reason for choosing the Oracle XE was the opportunity to learn and discover one of the most commonly used tools in commercial projects. With background knowledge gained from university classes the development process was less time consuming and could be handled with more precision as we became even more familiar. We also took into consideration the fact that if the project was to grow in size and develop into a larger scale system requiring features unavailable in this edition it would be possible to upgrade and move to a commercial edition without significant changes in the architecture.

¹ "Oracle® Database Express Edition." (accessed 10 June 2013.)

2.2.2 Java and Tomcat

Java is programming language with high adoption among enterprises because of its portability and Oracle (and Sun Microsystems before) support. It allows to easy develop and debug applications.

For the web application we used Java compiled and packed to web application archive (WAR). We are deploying it on Java applications server Tomcat.

2.2.3 Neo4J

There were many database platforms to choose from when creating this project regarding the product portion. This portion contains the generic search, product categories, products themselves, and of course description of the product. Therefore we chose to use the world's leader in graph database Neo4j. A quick description of this framework and reasons for choosing it are the proceeding.

Neo4j is a graph database that stores all its information inside nodes that are connect to each other. Before venturing further into the relationships and construction of the database let us take a step back and discuss all the working parts of the database. A good starting point for this is to describe the node. A node is the core building block in order for data to be represented². In other words this is where all the information is stored. Each node has its own unique Id that changes every single time a user exits the platform. A user may assign his own name, Id, and description to every node. This way the user can identify which node he wants to access in later queries. Next there are relationships, which describe how one node communicates with another. This communication can go one way or both, it matters what you are setting up. The relationship should have a name so that anyone using the system may quickly identify why the relationship is set up in such a way.

² "Node (Neo4j Community 2.0.2 API)." (accessed 10 May 2014)

Neo4j is a very stable and strong platform that can grow rapidly and expand to great lengths - the platform can store up to one billion objects. Also it is very compatible with programming languages such as: Java, PHP, Rails, etc. This was very profitable to the project, in the sense that if something didn't work out in one programming language, we will not have to look for another database and all the hard work would go to waste.

One of the biggest factors for using this database is its fast sorting algorithms. Some functions of Neo4j include: shortest path, all path, all simple paths, and Dijkstra algorithm.

Shortest path algorithm works in such a way, that if we want to get from point 'a' to point 'f', we will get there with minimum weight. A graph is composed of vertices and edges. Vertices in a graph are labelled by tokens, for this example we use letters as: 'a', 'b', 'f'. The edges have a weight factor, a numeric value, which symbolizes the cost which one must pay to get from one vertex to the next. In the shortest path algorithm we want this weight to be as low as possible, so we basically pick the 'shortest' route from 'a' to 'f' with the lowest cost. We travel the graph from one vertex to the next picking the lowest cost until we reach our destination.³

All paths algorithm works in a very similar way as shortest path does, except it is much more time consuming. Instead of calculating the weight of each route from one vertex to the destination vertex it shows all possible routes, with the possibility of repeating the same vertex. This means this algorithm returns every single possibility from getting from vertex 'a' to vertex 'f'.⁴ Alongside this comes into play the simple path algorithm. It is almost exactly the same as the all path algorithm, as the name states, with the one change of no vertex can be repeated. This still usually returns a vast number of possibilities and is greatly time consuming.

To complete discussing the algorithms let us focus on the Dijkstra algorithm. Explaining this algorithm using a graph model would be most appropriate. This algorithm starts like any other with the starting vertex. From here, we look at all the possible 'neighbours', which are vertices connected to the starting vertex by a single edge. We look at each specific neighbour and determine which weight of each edge is smallest. We then traverse to this vertex

³ Sedgewick, Robert, and Kevin Wayne. *Shortest Paths*.

⁴ Backhouse, Roland. *All-Paths Algorithm*.

which has the lowest weight cost. This process is repeated until the destination vertex is reached.⁵

Another important factor in choosing Neo4j database was the ability to remove nodes and all the children nodes and their relationships within one query. The ability to do so would be most useful for our application in the case that someone entered a category and some multiple products, and had to remove them all at once. Then he would be able to execute just one query and all the relationships and child nodes attached to that node would be deleted along with the node itself.

2.2.4 Cypher

Cypher is a simple declarative graph language with easy syntax, but at the same time extremely powerful. This language has multiple benefits including very efficient query updates and complicated queries can be expressed with simple syntax. The easiest way to understand a query in Cypher and how it works is to focus on what is retrieved from the database rather than how this information is gathered.⁶

Before focusing on query composition and the syntax used, let's take a step back and describe how the database is laid out within Neo4j. Knowing that Neo4j is a graph database, we define the nodes and edges using Cypher. In our project we defined a central node called search, giving this node a specific id as well as name. From this node following defined edges, signifying relationship between the nodes, we hit all different categories which are similarly attributed as the search node. From each category node you follow the defined edges out to find a specific product node referring back to the category you are in. These specific product nodes have a significant greater number of attributes so the user can identify which product he is searching for very precisely.

⁵ Mehlhorn, Kurt, and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Berlin: Springer, 2008. p. 194

⁶ "The Neo4j Manual V2.1.27.1. What Is Cypher?" 7.1. *What Is Cypher?*" (accessed 10 May 2014)

Nodes are connected by edges in this graph, with representing relationships. Each relation has a name for easier association in which way the nodes communicate with each other. These relationships are later significant in queries to find the specific location of a node or attribute. In our project we used binary relations for the case of executing and deleting recurrent queries. If we haven't done it this way, the delete query would go to the outermost specific product node, and since there is no relationship pointing back at the category node the query would just end there. With our solution we are able to delete products with a specific attribute across the graph with just a single query.

Cypher query language is very similar in syntax and structure to that of SQL. Most key words are directly taken from SQL; others however are borrowed from some semantics from languages as Python and Haskell. This combination brings some great characteristics from these programming languages into Cypher, making it easy to understand and allowing faster development. The main focus in Cypher when constructing a query is how you would want the information to be extracted, point the code directly to the location you want to extract, delete, or update the information to or from.

2.2.4 Mongo DB

MongoDB is NoSQL document oriented database. It doesn't have any structure but concept of documents. Documents can vary so it can't be described as table – because every document can have different properties. MongoDB query language is much simpler than SQL while the advantage are faster read and write operations⁷.

From the beginning it seemed for us that MongoDB is ideal to store our application statistics. Fast writes – that wouldn't slow down our application and ability to selectively read data using simple query language made an ideal connection, almost tailored to our needs.

⁷ Rege Gautam 'Learning Mongoid'

2.2.5 Ruby on Rails

Ruby on Rails is relatively new programming environment. It was created in 2003 by David Heinemeier Hansson. It follows “convention over configuration” rule to make applications easier and faster to develop. The programmer doesn’t have to think how to configure the environment, instead he can focus on developing solutions to real problems.

We used Ruby on Rails for our main web applications appreciating its advantages over other available environments.

2.2.6 Subversion and TortoiseSVN

Throughout the development we needed a tool that would allow us to keep track of all the changes that were made. We decided to use TortoiseSVN as it is a free and open source software that provides an interface for the Subversion (SVN) software. Not being designed for one particular IDE, makes it an ideal tool when developing a project using various different development tools. The software is written in C+ and it is compatible with windows client. It is very intuitive to use and made our communication much easier. TortoiseSVN provides a direct preview of file’s state, allows for easy file movement and renaming. It also has well developer commit dialog, allowing users to write messages in logs for other users to see. Content is shared securely through certificate verification granted by the host user. This improves group’s communication tremendously and keeps the development process’ momentum. If there was a situation where two conflicting commits were made, TortoiseSVN provides a TortoiseMerge tool that helps resolving these situation by clearly defining where in the file is the source of conflict and suggest possible merge options.⁸

⁸ "TortoiseSVN." *About TortoiseSVN*. (accessed 02 June 2014)

2.2.7 Web Applications dependencies

Web applications are written in Ruby on Rails framework and are dependent on some third party libraries, mostly called 'gems' which are special packages of Ruby on Rails plugins.

Table 1 RoR third party libraries

Dependency Name	Description
Twitter Bootstrap http://getbootstrap.com/	Frontend framework
Twitter Bootstrap Flatly theme http://bootswatch.com/flatly/	Frontend framework theme
SB Admin for Twitter Bootstrap	Admin Theme
gem 'ruby-oci8'	Ruby gem: Oracle database connector
gem 'activerecord-oracle_enhanced-adapter'	Ruby gem: ORM adapter
gem 'sass-rails'	Ruby gem: SCSS to CSS compiler
gem 'uglifyer'	Ruby gem: javascript compressor
gem 'coffee-rails'	Ruby gem: Coffeescript to Javascript compiler
gem 'jbuilder'	Ruby gem: JSON API builder
gem 'bootstrap-sass'	Ruby gem: SCSS Twitter Bootstrap styles
gem 'jquery-rails'	Ruby gem: JQuery Javascript library
gem 'haml-rails'	Ruby gem: HAML templating language
gem 'simple_form'	Ruby gem: Form wrapper
gem 'therubyracer'	Ruby gem: Javascript interpreter

gem 'bcrypt-ruby'	Ruby gem: Hashing functions of Bcrypt
gem 'will_paginate'	Ruby gem: pagination of database query results
gem 'mongoid'	Ruby gem: MongoDB database connector
gem 'jquery-datatable-rails'	Ruby gem: JQuery datatables plugin
gem 'treetop'	Ruby gem: Treetop library for grammars
gem 'nvd3-rails'	Ruby gem: NVD3.js library for charts
gem 'neography'	Ruby gem: Neo4J database connector
gem 'breadcrumbs_on_rails'	Ruby gem: Breadcrumbs library
gem 'thin'	Ruby gem: Ruby on Rails HTTP server
Dracula Graph Library	Used for Oak Front module, similar products graph

Oak DBSync dependencies

Database synchronization application is written in Java, runs as standalone and depends only on database connector libraries:

Table 2 Java connector libraries

Dependency Name	Description
JARs: org.neo4j.cypher, neo4-jdbc	Neo4J Java connectors
JAR: com.oracle.ojdbc14	Oracle database Java connector

Price Watcher dependencies

Price Watcher module depends on Apache Tomcat server and simple library for JSON document parsing and generating.

Table 3 Third party libraries used in Price Watcher module

Dependency Name	Description
JAR:com.googlecode.json-simple	Simple Java implementation of JSON document builder and parser
Apache Tomcat version 7	Java Application Server Tomcat, application container

2.3 APPLICATION ARCHITECTURE

OakSearch, apart from the databases, is composed of the following independent components:

1. Oak Front module
2. Oak Admin module
3. Oak Shop module
4. Oak DBSync
5. Oak PriceWatcher
6. Oak Search Engine
7. Statistics

Each of them can work independently of each other, requiring only environment configured to run specified module and database connection(s).

Oak Front module

Oak Front module is a web application written in Ruby on Rails framework. This is the layer presented to website visitor, who is interested in searching for products, comparing them, commenting, writing reviews and comparing prices.

Oak Admin module

Oak Admin module is also a web application written in Ruby on Rails framework. This layer is responsible for maintaining application by administrative users. They can perform CRUD operations on almost every entity in database.

Oak Shop module

Oak Shop module is very similar to the Oak Admin module, it's also a web application written in Ruby on Rails framework, however user can only perform CRUD operations on objects that are in his “workspace” (eg. he or the shop he's assigned to added them).

Oak DBSync module

Oak DBSync module is Java application responsible for synchronizing data from relational, Oracle Database, to our “NoSQL”, graph database – Neo4J. DBSync module also allows to reconstruction of whole graph database (in case of corruption or during maintenance).

Oak PriceWatcher

Oak PriceWatcher module is a Java application. In our application it's responsible for providing product prices. Due to limited options and legal issues with fetching prices from websites we decided to create simple price change simulator so obtained prices are not real, they are based on pseudo random algorithm and the old price.

Oak Search Engine

Search Engine, the most important part of application. This should be the main traffic source to the product subpages.



Figure 5 Search Engine search box

From words, written in the search box, there is query to the database constructed. Depending on the query result search result is displayed. Sample row of search result may look like following:

Compare	Name	Result accuracy
<input type="checkbox"/>	HTC One M8 Matched: Screen Type: Touchscreen Manufacturer: HTC	100%

Figure 6 Exemplary search result

Where the first column contains checkbox used for the product comparison, next is the product name with link to the product and line under the product name there are displayed features matched by the search query, showing why this product is on the results list.

Last column is the result accuracy. It is calculated using following formula:

Equation 1. Accuracy calculations

$$accuracy = \frac{\text{number of matched features}}{\text{number of features in search}} * 100\%$$

By default the constructed search query is permissive and sums up results. To search for the products having all features written in search box the “Exact match” checkbox can be used:



Figure 7 'Exact match' button

It changes the hyphen of criteria queries from OR to and AND narrowing the search results.

2.3.1 Oak Front Module

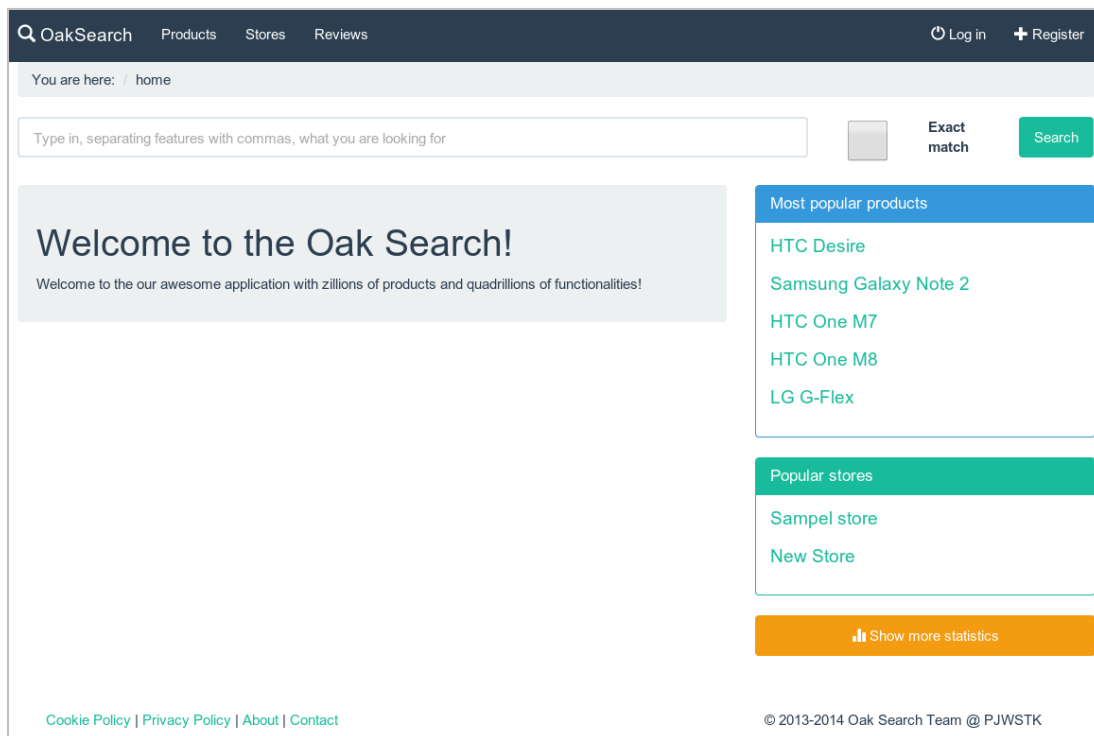


Figure 8. Front Module - website main page

Oak Front module is the main part of our application. This is the layer which is presented to the user. This web application is written in Ruby, Ruby on Rails framework.

Architecture

Oak Front follows Model-View-Controller (MVC) paradigm. That means data layer (model), controller (business logic part) and view (presentation layer) are separated. This separation is good because presentation and model are developed with different technologies and techniques, code gets cleaner and is easier to maintain. Also another advantage of such separation is ability to modify anything in the presentation layer (view) without touching underlying model or business logic. Because of this code can be also reused – it's easier to present same data in different ways creating only additional view.

Business Logic Layer – Application Controllers

There are six controllers allowing users to perform actions:

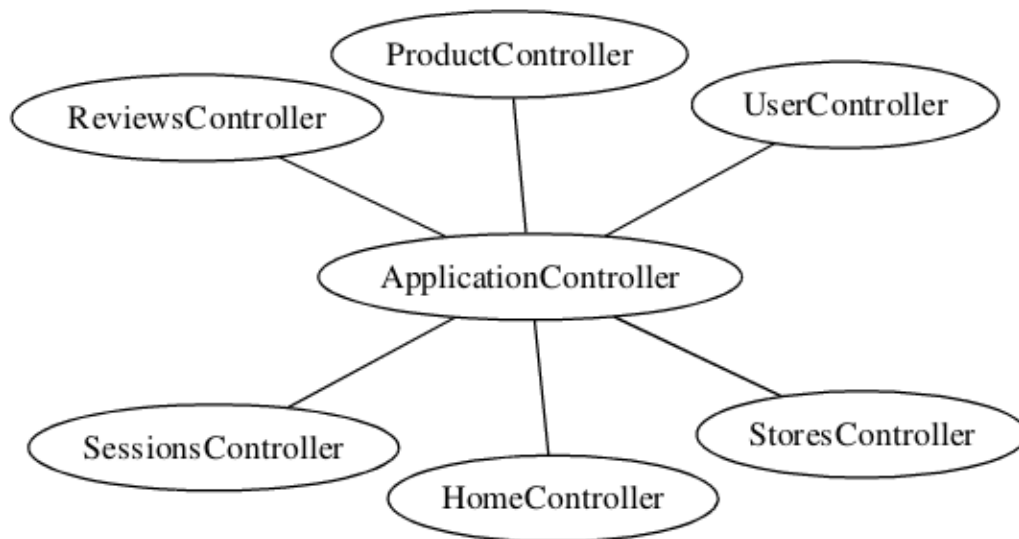


Figure 9 Controls Diagram

All of them inherit from the ApplicationController which holds methods common for others, especially actions to take when any of requested models couldn't be found in database:

```
around_filter :catch_not_found

def catch_not_found
  yield
rescue ActiveRecord::RecordNotFound
  redirect_to root_url, :flash => { :error => "Record not found." }
end
```

Code Listing 1. 'Record not found' action

Or methods connected to user authentication:

```
def authenticate_user
  if session[:user_id]
    @current_user = User.find session[:user_id]
    return true
  else
    redirect_to(:controller => 'sessions', :action => 'login')
    return false
  end
end

def save_login_state
  if session[:user_id]
    redirect_to(:controller => 'sessions', :action => 'home')
    return false
  else
    return true
  end
end
```

Code Listing 2. User authentication method

Such inheritance also allows to avoid code duplication and keep code clean.

HomeController

As it's shown in the picture below, HomeController holds actions for index (welcome) page, static pages such as about and policies, contact page, statistics (stats) and old interface for search using NLP.

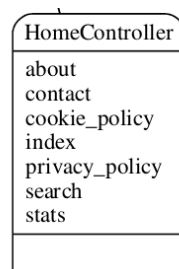


Figure 10 Home Controller

UserController

UserController holds actions for user profile management, receiving price offers, watching products changing password and creating user accounts.

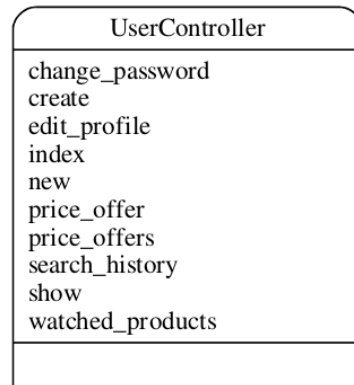


Figure 11. UserController

Actions, that are reserved only for authenticated user are protected using filter, function that is executed before every action or before actions specified in `:only` array:

```
before_filter :authenticate_user, :only => [:show, :watched_products, :index, :price_offer, :price_offers,
                                           :edit_profile, :change_password]
```

Code Listing 3. Authenticate_user Filter

This way `:authenticate_user` method inherited from `ApplicationController` is executed and user is granted or denied access. Such procedure is repeated in other controllers.

SessionController

This controller is responsible for user session management. This is where login and logout takes place.

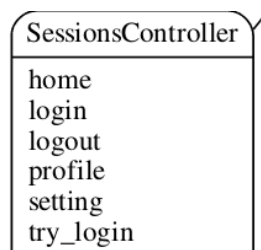


Figure 12. Srsion Controller

ProductController

This controller holds data for all actions that are taken on products. The most important action is search which is responsible for the most advanced feature of the application.



Figure 13. Product Controller

Also action to add review is embedded in this controller, because of straight connection of review with product.

StoresController

Options of browsing stores' offers and products or managing blacklists are contained in StoresController.

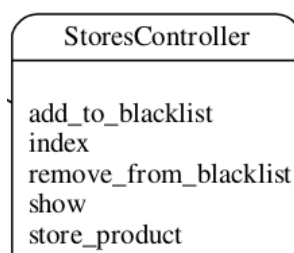


Figure 14. Stores Controller

Actions `add_to_blacklist` and `remove_from_blacklist` are secured with the mentioned authentication filter.

ReviewsController

ReviewsController contains actions responsible for listing all reviews and showing them in single-review view (which is of course separated and easily it can be presented in different way).

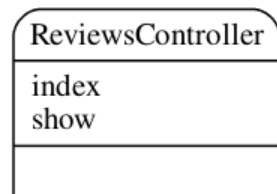


Figure 15. Reviews Controller

Presentation layer – views

Every controller holds a subdirectory in views directory and every action has assigned one view file which is responsible for displaying data. Views are written in HAML, language that compiles to HTML and there are also simple Ruby templates.

Data layer – models

Models are classes with specified table name and relations, for fields Object Relation Mapping (ORM) is used. Fields are assigned during the model initialization and cached for later use. This technology saves time during development (there is no need to add or remove fields from the model when database changes).

```

class Product < ActiveRecord::Base
  self.table_name = 'PRODUCTS'
  self.sequence_name = 'PRODUCT_ID_SEQ'

  validates :name, presence: true, length: {minimum: 2}
  validates :description, presence: true, length: {minimum: 2}

  has_many :productsfeatures
  has_many :features, :through => :productsfeatures

  has_many :reviews
  has_many :userwatchedproducts

  has_many :storeproducts

  has_many :auctions
  accepts_nested_attributes_for :features

  #fill audit data
  before_save :audit_update
  before_create :audit_update
  def audit_update
    if self.created_by.blank?
      self.created_by = 1
    end
    self.modified_by = 2
    datenow = DateTime.now
    self.modified_date = datenow
    if self.created_date.blank?
      self.created_date = datenow
    end
  end
end
end

```

Code Listing 4. Product Model

The code above is an example of more complicated model. It has many relations (`has_many` instruction), validation rules assigned (`validates` instruction). Also, there is trigger-like functionality assigned (`before_save`). It changes created or modified date before CREATE or UPDATE query is performed.

2.3.2 Oak Admin Module

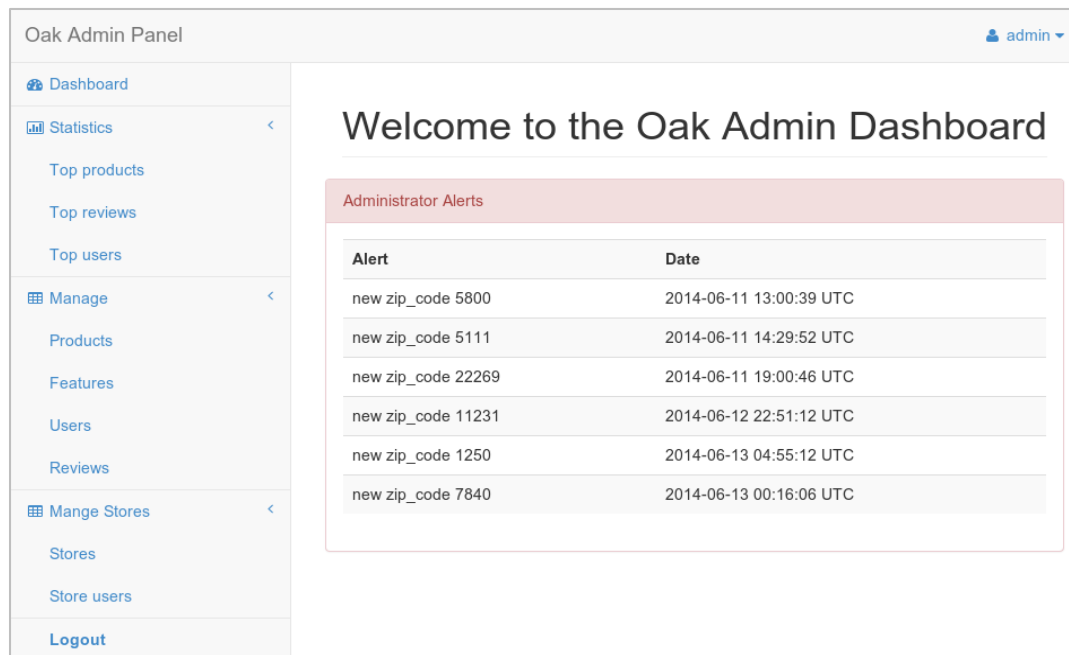


Figure 16. Admin module main page

Oak Admin module is another management console in this project. This one is created for the application administrator. This web application is written in Ruby, Ruby on Rails framework. Its architecture follows the same principles (MVC) as two previous. Also the logic of application is the same – different are controllers and actions and these are described below.

Business Logic Layer – Application Controllers

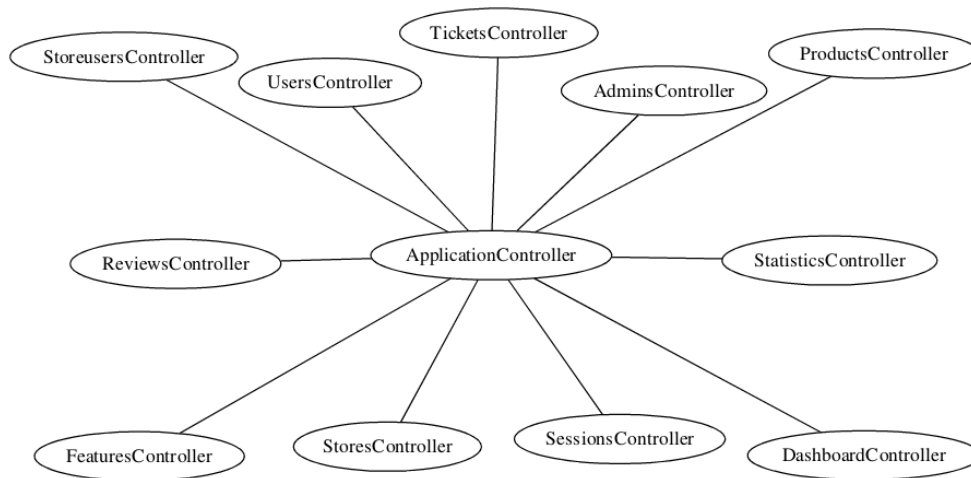


Figure 17. Oak Admin Module Application Controllers Diagram

All application controllers allow CRUD operation on all objects and only example of controller is described below.

ProductsController

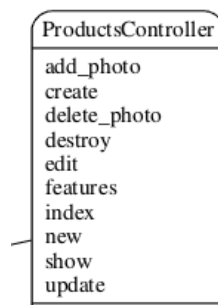


Figure 18. Oak Admin Products Controller

Product controller holds the role of managing products and its features. By actions ‘add_photo’ and ‘delete_photo’ it is responsible for managing products galleries.

This controller actions are accessible only for authenticated administrators so before filter doesn’t specify actions it will be turned on (by default it runs before every action):

```

class ProductsController < ApplicationController
  before_filter :authenticate_user
  before_filter :authenticate_administrator, except: [:index, :show, :new]
end
  
```

Code Listing 5. Admin authentication filter

To prevent uploading malicious files as gallery images, content-type of uploaded file is checked. Only if it is one of expected image content types uploaded file is saved and gallery database is updated:

```
image = params[:uploaded_file]
content_types=["image/png", "image/jpeg", "image/gif"]
if !content_types.include?(image.content_type)
  flash[:error] = "Content type "+image.content_type+" is not allowed!"
  redirect_to "/product_add_photo/#{@product.product_id}"
  return
end
```

Code Listing 6. Uploading picture files security

2.3.3 Oak Store Module

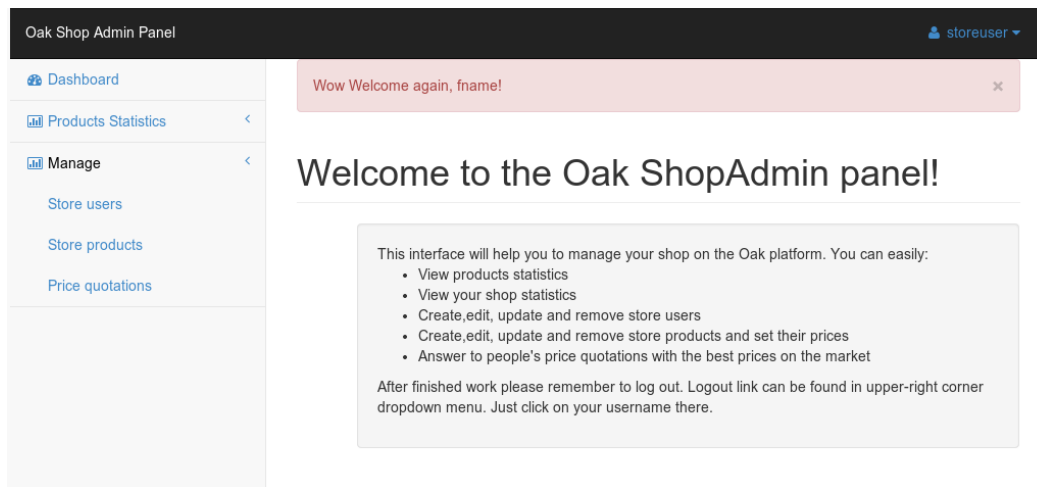


Figure 19. Store User's main page

Oak Shop Admin module is management console created for the store administrators. This web application is written in Ruby, Ruby on Rails framework. Its architecture follows the same principles (MVC) as Oak Front module. Also the logic of application is the same – different are controllers and actions and these are described below.

Business Logic Layer – Application Controllers

In this module there are 3 new controllers introduced.

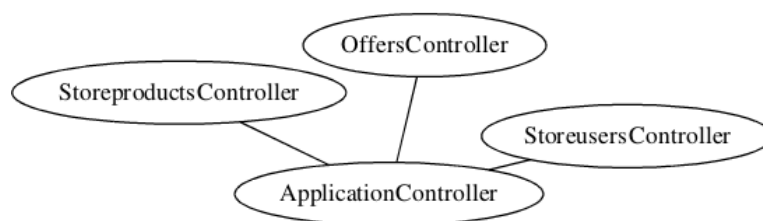


Figure 19. Oak Store User Module Application Controllers Diagram

StoreproductsController

As the name of this controller suggests it's responsible for store products management, that's why all CRUD operations are included. Of course workspace is limited to the store, store user is assigned to – so he can't modify products from not his store.

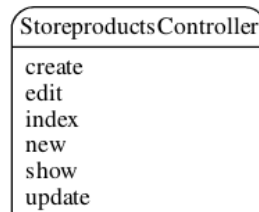


Figure 20. Store Products Controller

OffersController

Offers controller is responsible for offers in users' price quotations.

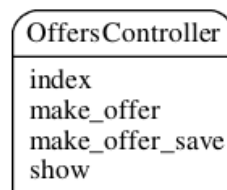


Figure 21. Offers Controller

It allows to list price quotations, show single price quotation with more details and to make offers.

StoreuserController

Store user controller contains actions responsible for managing store users. Here store user can create more store accounts, delete other accounts and change accounts properties. Again, operations are limited to the store user is assigned to.

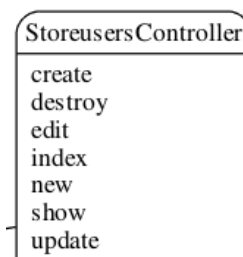


Figure 22. Store Users Controller

2.3.4 Oak DBSync Module

2.3.4.1 Database Synchronization Architecture

In the application we are using 3 different databases: one relational, Oracle Database, and two non-relational, „no-sql” databases: Neo4J for storing graph of products and Mongo for storing user statistics. While Mongo database is almost independent and nothing in relational database depends on it, there exists connection between Oracle and Neo4J.

Products, which are stored in the Oracle database, are also reflected as graph nodes in the Neo4J. To create such structure we were driven by the need of finding related products and products from the same category in search engine of designed system.

At the development level we inserted products simultaneously to both databases. At some point we found out that this approach can make our databases out-of-sync with some products missing in one database while present in another.

To resolve above issue we created service that synchronizes databases automatically:

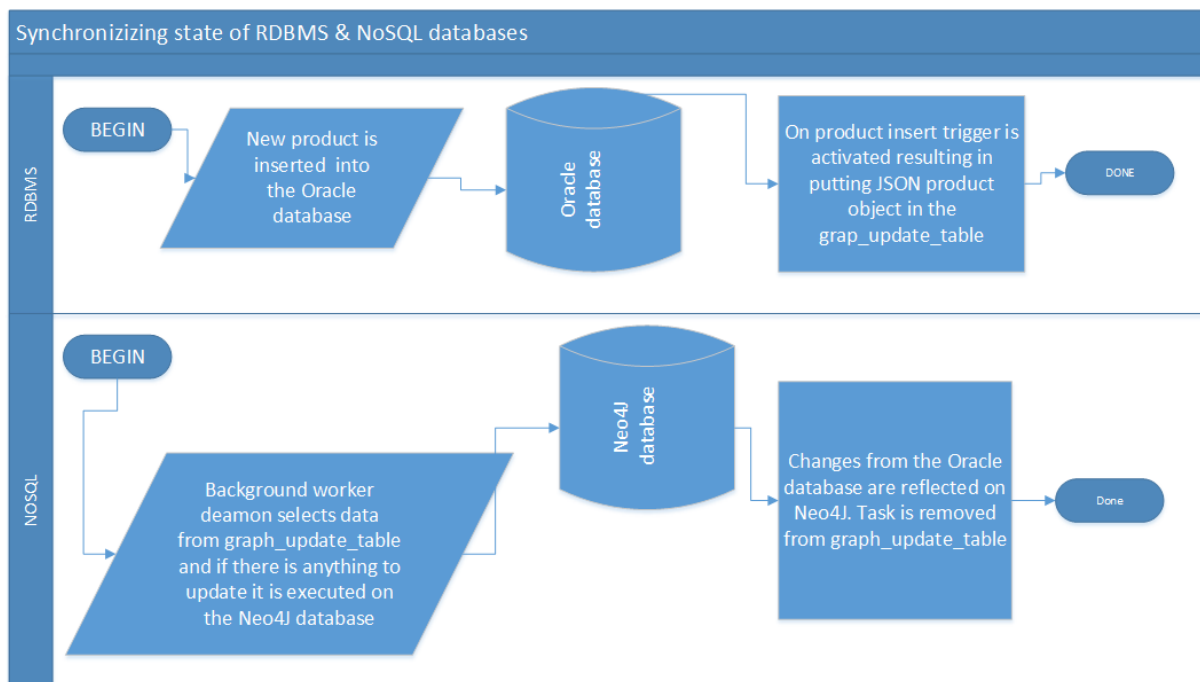


Figure 23. Database synchronization chart

Above graph represents actions taken by the synchronizer service:

1. On product insert/update/delete to the Oracle database trigger is called. Trigger inserts event data (such as event type: insert or update or delete, product_id in case of update or delete, old_data in case of update or delete and new_data in case of update or insert) to the graph_update_table.
2. Synchronizer on the specified time interval polls database for tasks in graph_update_table. If it finds one, the change is reflected on the Neo4J database (query with insert, update or delete is performed).
3. If query didn't return any error task is removed from the graph_update_table.

2.3.4.2 Database Synchronization Implementation

Actual implementation of Database Synchronization is pretty similar to what has been presented as assumption.

DBSync application is written in Java and uses ODBC drivers to connect to databases. It is composed of 3 threads running simultaneously:

1. Main thread, controlling creating the queue and starting other threads
2. dbReaderThread, reading data from Oracle database and putting them into the queue
3. dbWriterThread, polling the queue and inserting objects into Neo4J database.

```

52     n4jConnection = new Driver().connect("jdbc:neo4j://localhost:7474",
53         new Properties());
54
55     System.out.print("Initializing Neo4J connection...");
56     dbWriter = new DBWriter(n4jConnection, this, syncQueue);
57     dbWriterThread = new Thread(dbWriter);
58     dbWriterThread.start();
59     System.out.println(" [OK]");
60
61     try {
62         Class.forName("oracle.jdbc.driver.OracleDriver");
63     } catch (ClassNotFoundException e) {
64         System.out.println("CRITICAL: Oracle JDBC Driver not available");
65         throw new DBSyncException("Oracle JDBC Driver not available");
66     }
67
68     System.out.print("Initializing SQL connection...");
69     sqlConnection = DriverManager.getConnection("jdbc:oracle:thin:@jdbc",
70         "developer", "****");
71     dbReader = new DBReader(sqlConnection, this, syncQueue);
72     dbReaderThread = new Thread(dbReader);
73     dbReaderThread.start();
74     System.out.println(" [OK]");

```

Code Listing 7 - starting threads

As shown in the code above, thread “reading data” from Oracle database is started if, and only if, starting of “writing thread” has succeed. It preserves data to not be taken out from worker_tasks table and lost without being inserted because of Neo4J database issue.

Moreover, because more than one thread operates on the queue the special thread safe implementation of it had to be used. BlockingQueue is example of such interface, and ArrayBlockingQueue of its implementation.

ArrayBlockingQueue is bounded, by the size of underlying array that ensures us that queue will not take all heap space available for Java Virtual Machine and we won’t face the OutOfMemoryException what would lead to graph database data corruption.

However, being bound by the queue size, DBSync application could lose data during many operations on SQL database. That’s why, we implemented waiting for the place in the queue, in such manner, that if inserting thing into the queue wouldn’t succeed, the dbReaderThread will wait for 100 milliseconds and retry until it succeeds.


```

47 EventType eventTypeEnum = EventType.valueOf(eventType.toUpperCase());
48 OakModel oakModel = OakModelFactory.getOakModel(eventTypeEnum, res);
49 EventDTO event = new EventDTO(taskId, eventTypeEnum, oakModel);
50
51 while (false == queue.offer(event)) {
52     try {
53         Thread.sleep(100);
54     } catch (InterruptedException e) {
55         // ignore
56     }
57 }
58 removeTask(event);

```

Code Listing 8 - waiting for the place in the queue

Additionally, what was not covered by the project of DBSync, we decided on creating table, holding the history of performed actions. It's done simply by the trigger copying data before delete operation to the worker tasks history table.

On the database level the process of synchronization is initialized by triggers on related tables. Whenever a DML statement occurs, a group of appropriate triggers is triggered to insert then triggering event's data to worker_tasks table. On each of the three tables – products, features and product_features – there are the following triggers:

- Before insert or update, after insert or update for each row, after insert or update
- Before delete, before delete for each row, after delete

In order to facilitate making changes in Neo4J, the worker_tasks.event_desc attribute is a XML description of the modified row. Figure X shows how this was accomplished, by creating one function using sys_refcursors and built in XML packages.⁹

⁹ Braten, Morten. "REF Cursor to JSON." (accessed 05 January 2014)

```

2 CREATE OR REPLACE function ref_cursor_to_json (p_ref_cursor in sys_refcursor,
3 p_max_rows in number := null, p_skip_rows in number := null) return xmltype AS
4     l_ctx          dbms_xmlgen.ctxhandle;
5     l_num_rows     pls_integer;
6     l_xml          xmltype;
7     l_json         xmltype;
8     l_returnvalue  clob;
9 begin
10     l_ctx := dbms_xmlgen.newcontext (p_ref_cursor);
11     dbms_xmlgen.setnullhandling (l_ctx, dbms_xmlgen.empty_tag);
12
13     -- for pagination
14     if p_max_rows is not null then
15         dbms_xmlgen.setmaxrows (l_ctx, p_max_rows);
16     end if;
17
18     if p_skip_rows is not null then
19         dbms_xmlgen.setskiprows (l_ctx, p_skip_rows);
20     end if;
21
22     -- get the XML content
23     l_xml := dbms_xmlgen.getxmltype (l_ctx, dbms_xmlgen.none);
24     l_num_rows := dbms_xmlgen.getnumrowsprocessed (l_ctx);
25     dbms_xmlgen.closecontext (l_ctx);
26
27     close p_ref_cursor;
28
29     return l_xml;
30 end ref_cursor_to_json;
31 /
32

```

Code Listing 9. Generating XML from sys_refcursor

This function, `ref_cursor_to_json` is called in the after insert/update/delete trigger and processes all rows that were modified before inserting a new row into table `worker_tasks`.

Using three triggers that fire independently, but consecutively prevents the tables from mutating and enforces data integrity.¹⁰ To clearly describe what is happening let us use an example of inserting a new product. Before the new row is inserted the `product_bi` trigger starts. Next the row is inserted into the table. Afterwards, the `product_aifer` stores the new rowid, and if more than one row was inserted this trigger would trigger for each of those new rows. Lastly, the trigger `product_ai` loops through the stored rowids and using `ref_cursor_to_json` inserts processed data into `worker_tasks` table.

The same technique was used for deleting rows, with the difference that instead of storing just the rowid in row level trigger, selected data is copied to an array. In the table level after delete

¹⁰ "Avoiding Mutating Tables." (accessed 17 April 2014)

trigger data from the array is inserted into a temporary table with the same definition as modified table and finally the xml is generated from data in this temporary table and inserted into worker_tasks.

```

80 create or replace trigger product_ad
81 after delete on products
82 declare
83 a_prod_cursor sys_refcursor;
84 json_return xmltype;
85 begin
86     for i in 1 .. product_delete_pkg.oldvals.count loop
87         insert into delete_prod ( name, description, product_id )
88         values
89         ( product_delete_pkg.oldvals(i).name, product_delete_pkg.oldvals(i).description,
90           product_delete_pkg.oldvals(i).product_id );
91     end loop;
92     for i in 1 .. product_delete_pkg.oldvals.count loop
93         open a_prod_cursor for
94         select * from delete_prod
95         where product_id = product_delete_pkg.oldvals(i).product_id;
96
97         json_return := ref_cursor_to_json(a_prod_cursor);
98
99         INSERT INTO developer.worker_tasks
100         VALUES (worker_task_id_seq.NEXTVAL, 'product_delete', json_return, current_timestamp );
101     end loop;
102 end;
103 /
104

```

Code Listing 10. After delete trigger on Products table

2.3.4.3 Drawbacks

Current implementation of DBSync service requires to be running all the time, being highly available. In case of application or server failure recovering the graph database is pretty hard and requires of creating backup of SQL products, product_features and features tables, cleaning Neo4J database, and reinserting whole collection to the Oracle database.

Possibly, as the step of future development, this could be solved by creating additional DBSync module, for a while ignoring worker_task table and recreating graph structure using relations from Oracle database.

2.3.5 Oak PriceWatcher

PriceWatcher is simple interface that is used to update products prices in our application. After being started it serves HTTP request (both GET and POST) with JSON responses. It requires 3 parameters to be present:

1. productId – the Id of the product from the database, may be used for caching of requests
2. productCode – manufacturer product identifier
3. oldPrice – old price of the product

The proper query string of HTTP request should contain all parameters. The example of query string may look like following:

```
?productId=1&oldPrice=200&productCode=3
```

And will result is JSON response with changed product price:

```
{
  timestamp: 1401884206544,
  oldPrice: "200",
  source: {
    sourceURL: "example.local",
    sourceName: "ExamplePriceInfo"
  },
  newPrice: 196,
  productCode: "3",
  productId: "1"
}
```

Code Listing 11. Exemplary JSON response to price change query

The response contains additional information:

Table 4. Additional parameters in JSON response

Parameter name	Description
Timestamp	the current time in milliseconds
Source – sourceURL	URL of the source, the price was obtained from
Source – sourceName	Name of the source, the price was obtained from
newPrice	New price

For the response verification purposes obligatory request parameters are copied into the response.

Data validation

When request parameters are not present or data is corrupted JSON response will contain “error” parameter with simple description of the error. Response will also contain parameters copied and timestamp added.

The following example presents the response for no parameters in the URL:

```
{
  timestamp: 1401884393252,
  oldPrice: null,
  error: "param empty",
  productCode: null,
  productId: null
}
```

Code Listing 12. Exemplary JSON response to query without parameters

Architecture

PriceWatcher is Java web application packed in WAR archive so it can be deployed on any Java web application server which supports Java Servlet 3.0 standard.

Project structure is rather simple, it contains 4 classes and 1 interface - PriceInfo:

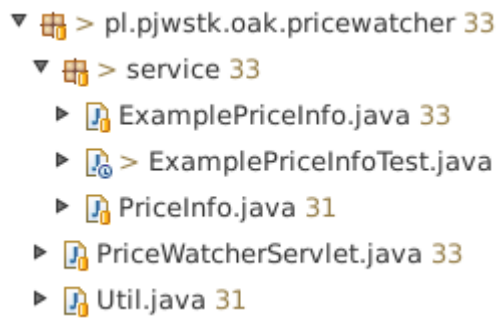


Figure 24. PriceWatcher application structure

Usage of interface structuring price information services allows to easily support more price sources. Current implementation of PriceInfo is based on pseudo-random numbers and the old price:

```

90 private Long getActualPrice(Long oldPrice) {
10     Random rn = new Random();
11     int random = rn.nextInt(100);
12     // price drop
13     if (random < 25)
14         return oldPrice - (oldPrice * random / 1000);
15
16     // price rise
17     if (random > 75)
18         return oldPrice + (oldPrice * (random - 75) / 1000);
19
20     // no price change
21     return oldPrice;
22 }
  
```

Code Listing 13. PriceInfo implementation

This implementation may be easily adapted to application needs, fulfilling the database with data for testing purposes.

2.3.6 Search Engine and Natural Language Parsing

During the development of search engine we came up with idea of creating natural language parser, translating user input straight into database queries. After creating proof of concept grammar example:

```

1 grammar Oak
2
3 rule expression
4   space? '(' body ')' space? <Expression>
5 end
6
7 rule body
8   (expression / gadget / hasfeature / hasfeatureand / feature / space)* <Body>
9 end
10
11 rule gadget
12   "gadget" <Gadget>
13 end
14
15 rule hasfeature
16   "with" space feature <HasFeature>
17 end

```

Code Listing 14. Natural Language Parsing grammar example

And some translation rules, translating to Cypher, Neo4J query language:

```

29 class Gadget < Treetop::Runtime::SyntaxNode
30   def to_cypher
31     puts "GADGET"
32     return { :start => "me = node({me})",
33             :return => "me",
34             :params => {"me" => nil } }
35   end
36 end
37
38 class Feature < Treetop::Runtime::SyntaxNode
39   def to_cypher
40     puts "  Feature"
41     puts self.elements
42     puts self.text_value
43     return { :start => "feature = node:features({feature})",
44             :params => {"feature" => "name: " + self.text_value } }
45   end
46 end

```

Code Listing 15. Translation rules to Cypher

We decided to check some more examples on natural language parsing. It came out that it's not so trivial task¹¹ and generally it shouldn't be done in real time.

¹¹ Bastani, Kenny. "Natural Language Search Using Neo4j." (accessed 05 June 2014)

After testing the search engine with our limited grammar with some random people, seeing their reactions to wrongly written questions (no one could write proper query with condition) we decided on dropping this part.

2.4 Oracle Database

The Oracle Relational Database (ORDB) is the core of the entire project. It stores data about three types of users as well as information about products and all user activity. Since all communication between the RODB and Neo4J begins with a DML statement, Oracle in case of Neo4J failure can come in its place and be used as backup for restoring Neo4J or as a temporary substitution so that the entire application will not fail and users will still be able to perform all or most of the activities.

2.4.1 Entities Relation Diagram

The structure of the database can be divided into sub-modules by grouping the entities by functionalities to which they correspond. We can distinguish five sub-modules: User Actions, Communication, Products' Data Management, User Support and System Operations.

User Actions

This sub-module is responsible for storing user specific data in tables USERS, STORES, STORE USERS, ADMINS. They hold information specific to each account such as names, last names, emails. Next, there are functionality related tables. Tables BLACK LIST and BLACK_LIST_STORES allow users to create their personal list of stores that are not to be notified when a new auction was created or whose products should be omitted in displaying the search results. Table USER_WATCHED_PRODUCTS allows each user to create a list of products that he or she is interested in and that would like to have a quick access to. The table ZIPCODES was created with the intention of future development of the system. Table REVIEWS stores records of product reviews that user has written. He or she will be able to view all posted reviews and they will also be visible on the reviewed product page.



Figure 25. User Actions Entity Relation Diagram

System Operations

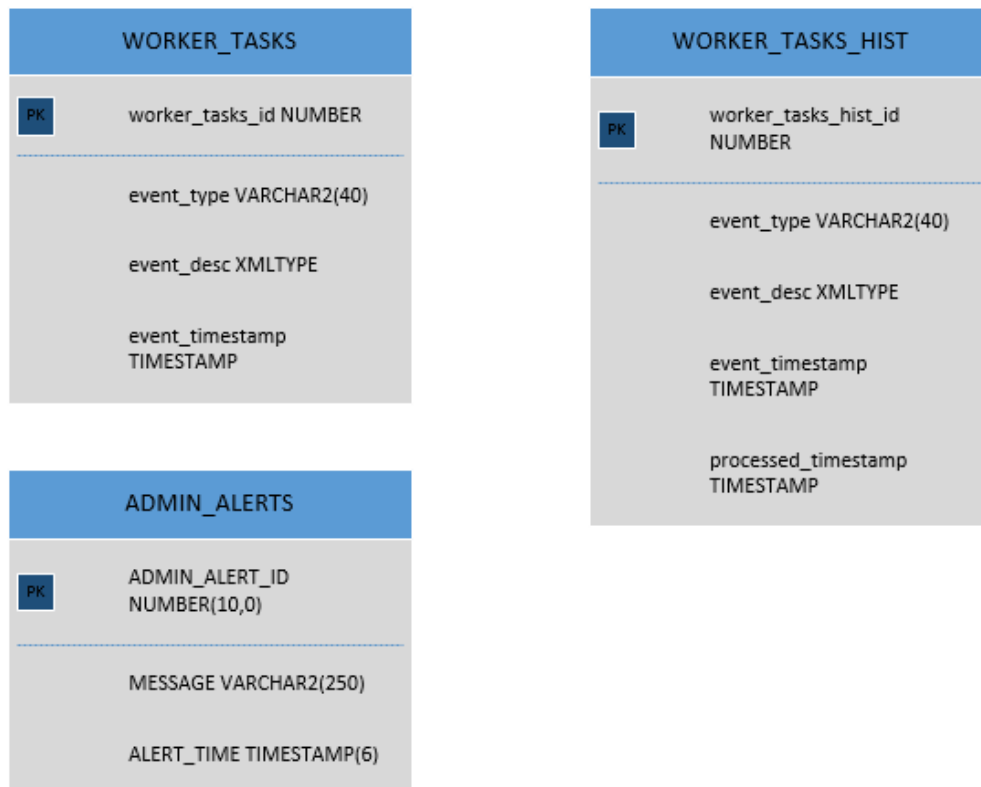


Figure 28. System Operations tables

Two of the entities that are not connected to any other entities play the most crucial role in communication between ODB and Neo4J DB. The records stored in table `WORKER_TASKS` contain information of changes that must be applied in Neo4J for an up to date search results. The table `WORKER_TASKS_HIST` plays a role of an archive – when a record from `WORKER_TASKS` is processed, the data is moved and the time of processing is added. This allows for recreating any modifications that took place in ORDB but were not reflected in Neo4J due to any form of application failure.

The table `ADMIN_ALERTS` is a way for signalling to the admin that there is a task for him or that something unwanted is happening in the application. Currently this is also a substitute for a complete zip codes directory. When a new user registers or a new store is created, if the entered zip code is not yet in the database, then a new row is inserted into this table with time and the new zipcode. Based on this, admin can enter correct city and country into the `zip_codes` table. This will prevent people from entering false or incorrect information, while the table `zip_codes` is not completely filled.

2.4.2 Other Database Objects

TRIGGERS

Most of the created triggers were created for the purpose of smooth communication and dependence between the ODB and Neo4J DB. On each of the three tables, PRODUCTS, FEATURES and PRODUCT_FEATURES there are groups of triggers triggering on insert, update and delete. Each of those trigger groups is composed of four schema objects, for eg. On table PRODUCTS there are:

- product_bi (BEFORE INSERT)
- product_aifer (AFTER INSERT FOR EACH ROW)
- product_ai (AFTER INSERT) and
- product_pkg (additional package for trigger operations)

The basic function of all these triggers is to insert a record into WORKER_TASKS table that consist of description of event that triggered the trigger, XML description of involved record that was inserted, updated or is to be deleted. These changes will then be applied to Neo4J DB. The detailed explanation of those triggers is described in chapter 2.3.4 *Oak DBSync Module*.

Whenever a record in table WORKER_TASKS is processed and then deleted a group of another DELETE triggers is triggered, which adds a new record to the WORKER_TASKS_HIST table.

One group of triggers that is not connected to the DBSync Module is important for the PriceWatcher Module. Just as with the DBSync Module's triggers, in order to avoid mutation of tables a group of co-dependent triggers had to be created. Whenever there is a change of attribute PRICE in table STORE_PRODUCTS a new record is inserted into table PRICE_LIST. This is also where the same technique was used in order to avoid mutating tables.

There are also a two other triggers that work solely inside the database to keep the correct data flow and unity of information inside the tables; they are a mechanism for storing new zip codes while there is no complete official directory of them inserted into the application. They are:

New_store_zip_code_bi – before insert or update on USERS table, adding a new zip code to the zip_codes table

New_user_zip_code_bi - before insert or update on STORES table, adding a new zip code to the zip_codes table

They both add the new zip code to the ZIP_CODES table, but also inform the admin via ADMIN_ALERTS table of this event. This allows the admin to insert the correct city and country for the given zip code into the database, thus preventing users from entering incorrect data.

Triggers are a very useful tool allowing for smooth application operation. They are especially advantageous in OLTP applications such as this one, where each client session operates only on a limited amount of records. If there was a case where a bulk insert was necessary it is sufficient to simply disable these triggers until all bulk operations are completed.

INDEXES

To optimize data retrieval, basing on predicted most frequently used queries, we designed indexes that sped up this process. They will be particularly useful in case of Neo4J database failure as they would allow for faster product searches for users.

All indexes are the default b-tree indexes since this is an OLTP application and all indexed columns have very high cardinality. Indexes were also automatically created on all columns with Primary Key and Unique constraints as an internal method of enforcing integrity constraints.

The different indexes which were not created automatically are:

Function based indexes:

- upper_product_name_idx on table PRODUCTS, column products.name
- upper_feature_name_idx on table FEATURES, column features.name
- upper_feature_type_idx on table FEATURES, column features.type

CONSTRAINTS

To ensure the data integrity some additional constraints were implemented.

Review_rating_check constraint checks if the rating given by a user is in the scale of 0 to 10.

A unique constraint was created on table PRODUCT_FEATURES, columns product_id and feature_id. Each product can have a specific feature occurring only once and this constraint allows for monitoring the uniqueness of these pairings. Although this would also make a good compound primary key, we have decided that an additional dedicated attribute, prod_feat_id, would be more optimal for data processing.

Other unique constraints were created on tables STORE_USERS, USERS and ADMINS on attributes email and nick.

All relations between entities were created with the FOREIGN KEY constraints.

Wherever it was necessary for the business logic of the application, there are additional, not system-created, NOT NULL constraints.

SEQUENCES

Unlike in MySQL database, Oracle does not provide the AutoIncrement column datatype. Instead for each column with NUMBER datatype, on which a Primary Key constraint was created, an independent sequence object was created. They are stored independently of tables and can be accessed through `sequence_name.CURRVAL` or `sequence_name.NEXTVAL` functions. The biggest advantage of using sequences is that they guarantee the uniqueness of values (with NOCYCLE parameter is specified upon sequence creation). To increase Oracle's performance we have specified the CACHE parameter to be 20, which allows the database to store the next 20 values of the sequence in memory for faster access. This will possibly cause gaps between the values; however this does not affect the data integrity.

```

21  PROMPT Creating Sequence 'PRODUCT_ID_SEQ'
22  CREATE SEQUENCE PRODUCT_ID_SEQ
23      INCREMENT BY 1
24      START WITH 1
25      NOMAXVALUE
26      MINVALUE 1
27      NOCYCLE
28      CACHE 20;
29  /

```

Code Listing 16. Exemplary Sequence Definition

PACKAGES

In order to facilitate the future work of DBAdmins, additional sets of functions and procedures were created.

There are three subject-specific packages that group all those additional schema objects: `product_api`, `user_api` and `ticket_api`.

The `product_api` package contains sample functions and procedures related to the `PRODUCTS` table.

```

2 CREATE OR REPLACE PACKAGE product_api AS
3     PROCEDURE new_product(a_name VARCHAR2, a_description VARCHAR2);
4     PROCEDURE delete_product(a_name varchar2);
5     FUNCTION get_newest_prod(a_prod_id out NUMBER) RETURN NUMBER;
6 END user_api;

```

Code Listing 17. Product_api package definition

The procedure new_product is a good example how developer can profit from using sequences. Instead of writing a separate function to get the newest product_id for insert operation, which would be a source of possible errors, it is enough to call the sequence NEXTVAL function. Using procedures and functions also enables easier exception handling.

```

24
25 INSERT INTO developer.products (product_id, name, description, created_by, created_date, modified_by, modified_date)
26 VALUES (product_id_seq.nextval, a_name, a_description, user, current_timestamp, user, current_timestamp);
27
28 END IF;
29 COMMIT;
30
31 EXCEPTION
32     WHEN existing_product THEN
33         DBMS_OUTPUT.PUT_LINE('THIS PRODUCT ALREADY EXISTS');
34

```

Code Listing 18. Example of sequence use

The user_api package contains schema objects related to managing registered users as well as store users. There are two procedures for basic user management and also a function that is of higher business logic character.

```

2 CREATE OR REPLACE PACKAGE user_api AS
3     PROCEDURE new_user(a_name VARCHAR2, a_last_name VARCHAR2, a_email varchar2,
4         a_nick varchar2, a_zipcode varchar2, a_pass_hash varchar2, a_pass_salt varchar2);
5     PROCEDURE delete_user(a_nick varchar2);
6     FUNCTION last_mod_prod(a_nick IN varchar2) return number;
7 END user_api;
8 /

```

Code Listing 19. User_api package definition

The last sample package, ticket_api, is created for managing user support functions delivered via TICKETS tables. If for some reason admins would be unable to perform operations regarding user support on the website, the set of provided procedures would enable them to work directly on the database and make any necessary changes.

```
2 CREATE OR REPLACE PACKAGE ticket_api AS
3     PROCEDURE new_user_ticket(a_user NUMBER, a_category NUMBER, a_topic VARCHAR2, a_content VARCHAR2);
4     PROCEDURE delete_user_ticket(a_user_ticket_id NUMBER);
5     PROCEDURE new_store_ticket(a_store INTEGER, a_category INTEGER, a_topic VARCHAR2, a_content VARCHAR2);
6     PROCEDURE delete_store_ticket(a_store_ticket_id number);
7     PROCEDURE change_user_ticket_status(a_ticket_id NUMBER, a_status NUMBER);
8 END ticket_api;
9 /
```

Code Listing 20. Ticket_api package definition

TEMPORARY TABLES

In the database there are three temporary tables: DELETE_PROD, DELETE_FEAT and DELETE_PROD_FEAT. They were created as an aid for delete triggers on tables PRODUCTS, FEATURES and PRODUCT_FEATURES. When user's session is over the data in the temporary tables is removed automatically but the temporary table definition is kept.

2.5 Neo4J

2.5.1 Designed Architecture

The Neo4J database is set up that one node is our starting point. This node we named search with an Id of 0. User search will begin here, as he navigates through our database, he will be able to return to this spot. From the search node we have various relationships set up, where each node has a double relationship to the starting node. These nodes are the categories of different products, for example phone, tablet, printer, scanner, and etc. Each of these nodes has a specific name, like the ones listed above and an id. This is so that in any case the user may navigate back and forth through these categories freely through the search node. Lastly, from each category node there are nodes with specific products. Each product node has a name, id, and description along with a double relationship with the product node. The description is the feature of every individual product. Through these features the user will be able to choose which ever product he desires. Again the binary relation applies in this area so that the user can freely navigate the database. The name, id, and description of any node are crucial in the database, because that is what enables selection of a certain product, deleting or updating it. Without these unique features navigation through the database would be impossible. A picture of this whole description is shown below:

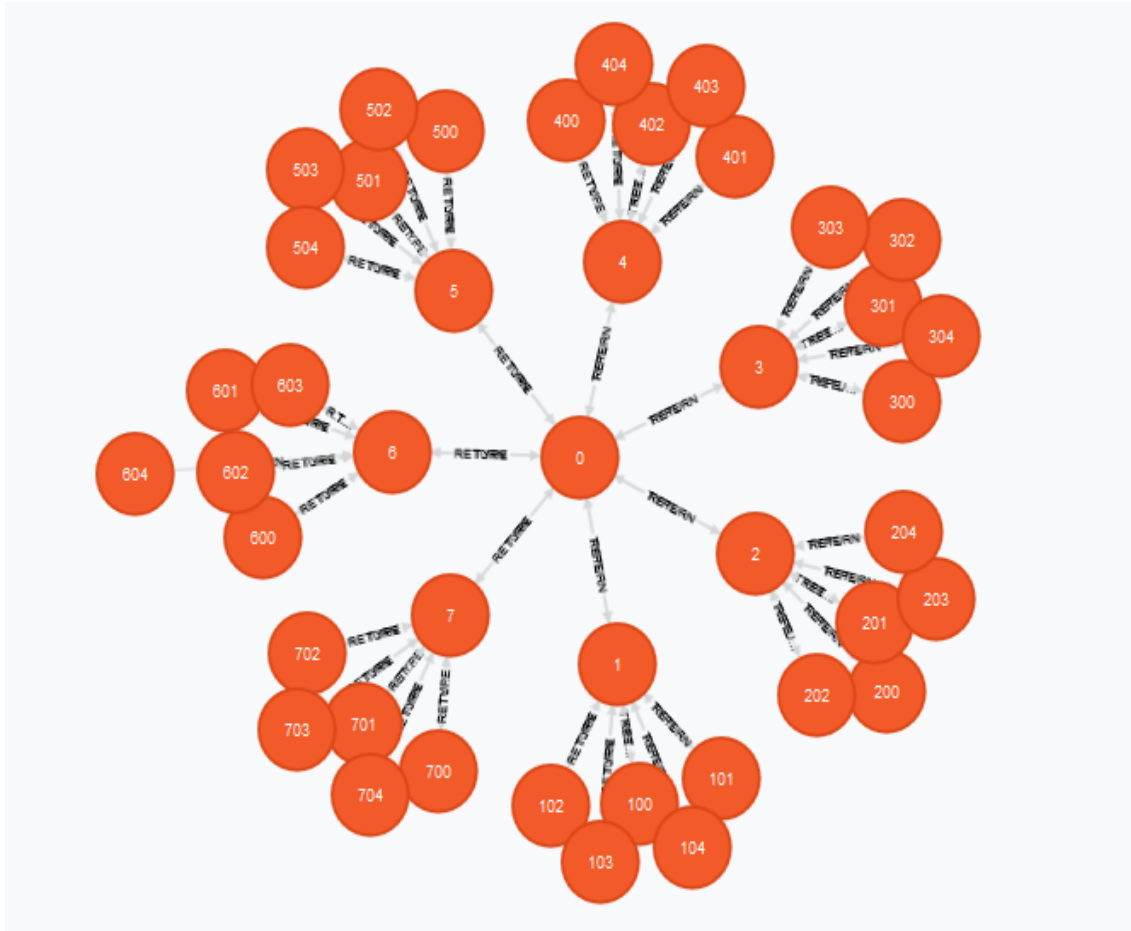


Figure 29. Diagram of Neo4J database

Neo4j uses Apache Lucene, which is a fully text search engine, written in Java, that is compatible with almost any platform. Since we used this graphical database strictly for defining products and their features we would not be dealing much with numeric data. Products are purely strings, and the only numerical data which would be considered for the features could be placed in string format. This will not affect data at all since specific features do not change for a product unless it is a new model, but then the product itself changes name.

2.5.2 Implemented Solution

Neo4J database graph structure

Neo4J database is the supporting database for the application. Additional feature ‘product recommendation’ is build using it.

In database there are nodes representing gadgets and features. Gadgets are connected with features using binary relations.

Simple example of graph fragment can be presented as following:

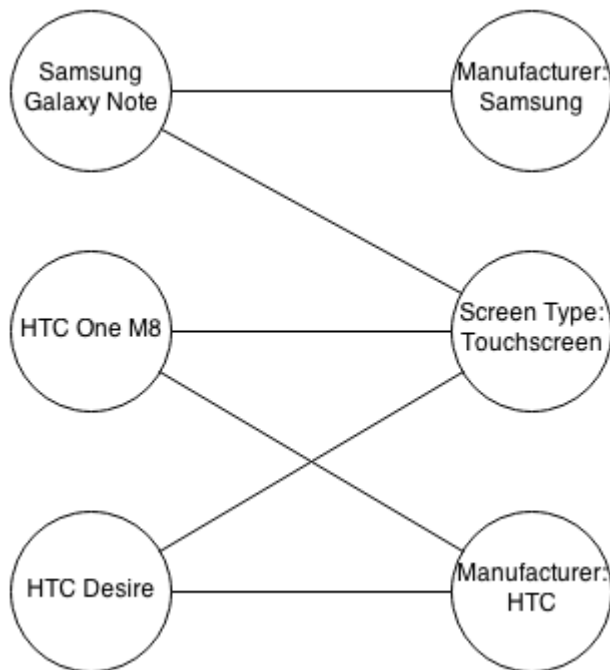


Figure 30. Graph structure example

Algorithm for the similar products is pretty straightforward. Starting from the node representing product, we are looking for other products that share the same features and count of the shared features is similarity factor (higher the factor is – the product is more similar).

As in this example: two HTC gadgets share 2 nodes, so they will be rated more similar to each other, than to Samsung phone, that is only having one common node with them.

2.6 MongoDB - Statistics in application

Application gathers statistics about product popularity, its price history, store product popularity and store popularity. Assumed measure of popularity is number of page views. To gather such data we decided on usage of NoSQL, document oriented database MongoDB.

2.6.1 Schema structure

When it comes to NoSQL we cannot say about data structure of tables because it may vary between documents. We rather say about structure of document. As we gather three kinds of statistics there are 3 types of documents:

1. Product document

Product document (called Productstat) contains three fields: counter called visit, product which represents product_id and date which is a string representing date in format YYYYMMDD.

```
class Productstat
  include Mongoid::Document
  field :visit, type: BigDecimal
  field :product, type: BigDecimal
  field :date, type: String
```

Code Listing 21. Definition of Product document

Additionally, in the model, there is defined MapReduce function allowing to obtain most popular product. Map Reduce is programming paradigm, similar to divide and conquer, for processing large datasets. It was described in 2004 by Google Engineers.¹² Example of such map reduce job looks like following:

¹² Dean, Jeffrey, and Sanjay Ghema W *MapReduce: Simplified Data Processing on Large Clusters*. (accessed 20 May 2014)

```

def self.get_popular
  map = "
    function () {
      emit(this.product, {'visit' : this.visit});
    }
  "

  reduce = "
    function (key, emits) {
      var total = {'visit' : 0};
      for (var i in emits) {
        total.visit += emits[i].visit;
      }
      return total;
    }
  "

  self.all.map_reduce(map, reduce).out(inline: true).to_a
end

```

Code Listing 22. Map Reduce job example

First step is function map, which extracts from data product_id and visits, while function reduce receives pairs of product_id and visits. It sums all visits for the same keys (product_ids) and that's how popular product list is received with popularity factor (in this case: number of page visits).

2. Store document

Store document (called Storestat) similiary contains three fields: counter called visits, field store that holds store_id and date which is a string in YYYYMMDD format.

3. Store product document

Store product document (called Storeproductstat) contains also three fields: counter called visits, field storeproduct for store_product_id and date in format of YYYYMMDD string.

2.6.2 Usage

In applications statistics are saved using Mongoid database driver. Having models defined (as in example of Productstat model) it is relatively easy to store statistics. On product show there is UPSERT (UPDATE or INSERT if not exists) performed:

```

productstat = Productstat.where(:product => @product.product_id, :date => Date.today.to_s(:db)).first_or_create
productstat.inc :visit => 1
productstat.save

```

Code Listing 23. Definition of product UPSERT

And MongoDB “inc” function is used, which performs increment on given field by value given as parameter.

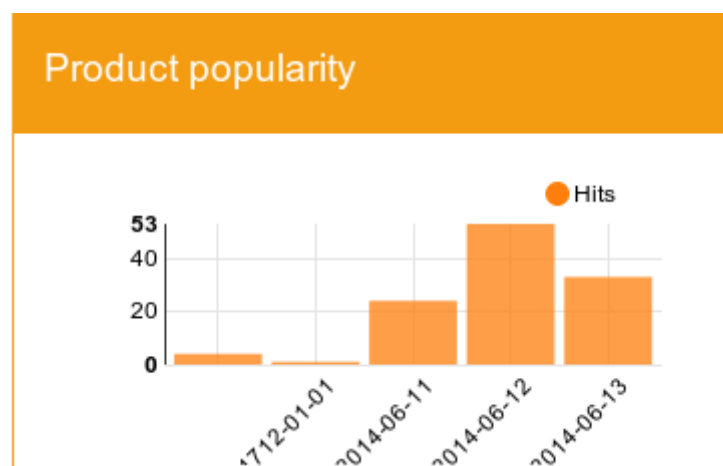
To obtain statistics and display them another simple method is used:

```
@productstats = Productstat.where(:product => @product.product_id)
```

Code Listing 24. Obtaining product statistics

As it can be observed in the code above, where condition looks pretty much the same as if it would be SQL database. ORM allows to perform SQL like queries on NoSQL databases directly from Ruby on Rails.

Later, the data can be presented in form of charts, like product popularity on product information page:



Code Listing 25. Exemplary product statistics chart

3. DEVELOPMENT PROCESS

During the beginning phase of working on our project one of the first decisions that our team has made was the project management and development strategy technique to be used. Taking into consideration the size of our team and size of the planned project some of the development were automatically eliminated from consideration. One of those was scrum, which even though seemed suitable for its project management ideology, our team is definitely too small and some of scrum's positive aspects would not be available. After some thought, we decided to apply the Spiral Model of development.

The Spiral Model of Application Development was first described by Barry Boehm in 1986 in the "A Spiral Model of Software Development and Enhancement" article. It is defined by its author as a '[development process] characterized by repeatedly iterating a set of elemental development processes and managing risk so it is actively being reduced'¹³. In most cases it is a good strategy for developing large projects however smaller applications also can greatly benefit from adapting it and most likely shorten the iterations significantly.

The whole process of developing software using the Spiral Model is divided into the following four stages that together form one iteration: the definition of objectives, identification and resolving risks, development and testing, and planning for next iteration.

¹³ Boehm, Barry W. *A Spiral Model of Software Development and Enhancement*. (accessed 9 June 2014)

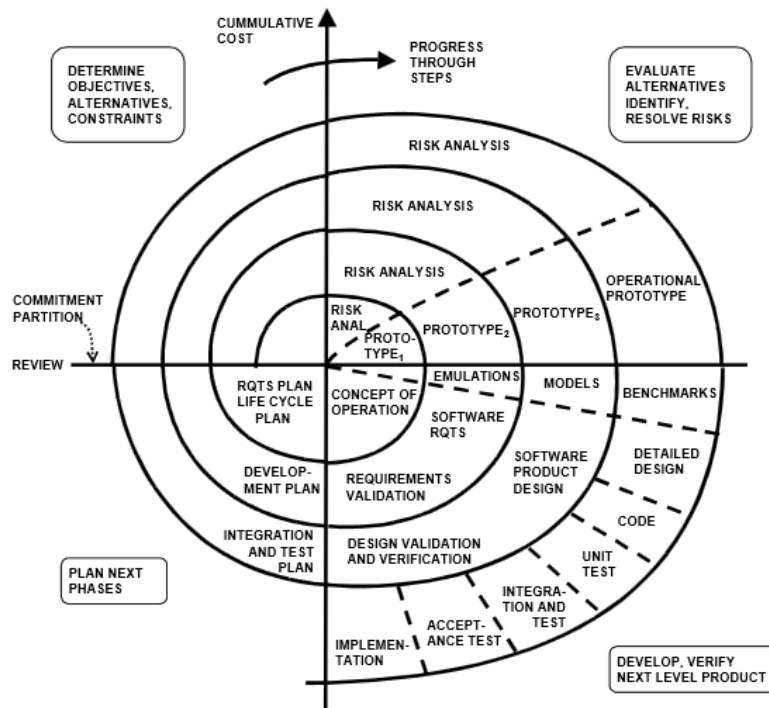


Figure 31. Spiral Model Development Organization

Since our project is small both in terms of planned architecture and team members all iterations were short and number of tasks planned for each was limited. The entire development process was divided into the following parts:

PHASE 1:

This was the initial phase where we defined system functionalities and users. This was also when the created list of functionalities was divided into separate system modules that would be developed in following phases. Each team member chose his or her module to develop and defined first tasks that were to be completed. At this point, we also did research about possible technologies to use and became familiar with new tools and solutions that could benefit our project. After comparison of all possibilities we agreed on the tools that we would be using throughout the development of our project. An initial version of a class diagram was created and it is now clearly visible how much the project has changed and developed since the beginning.

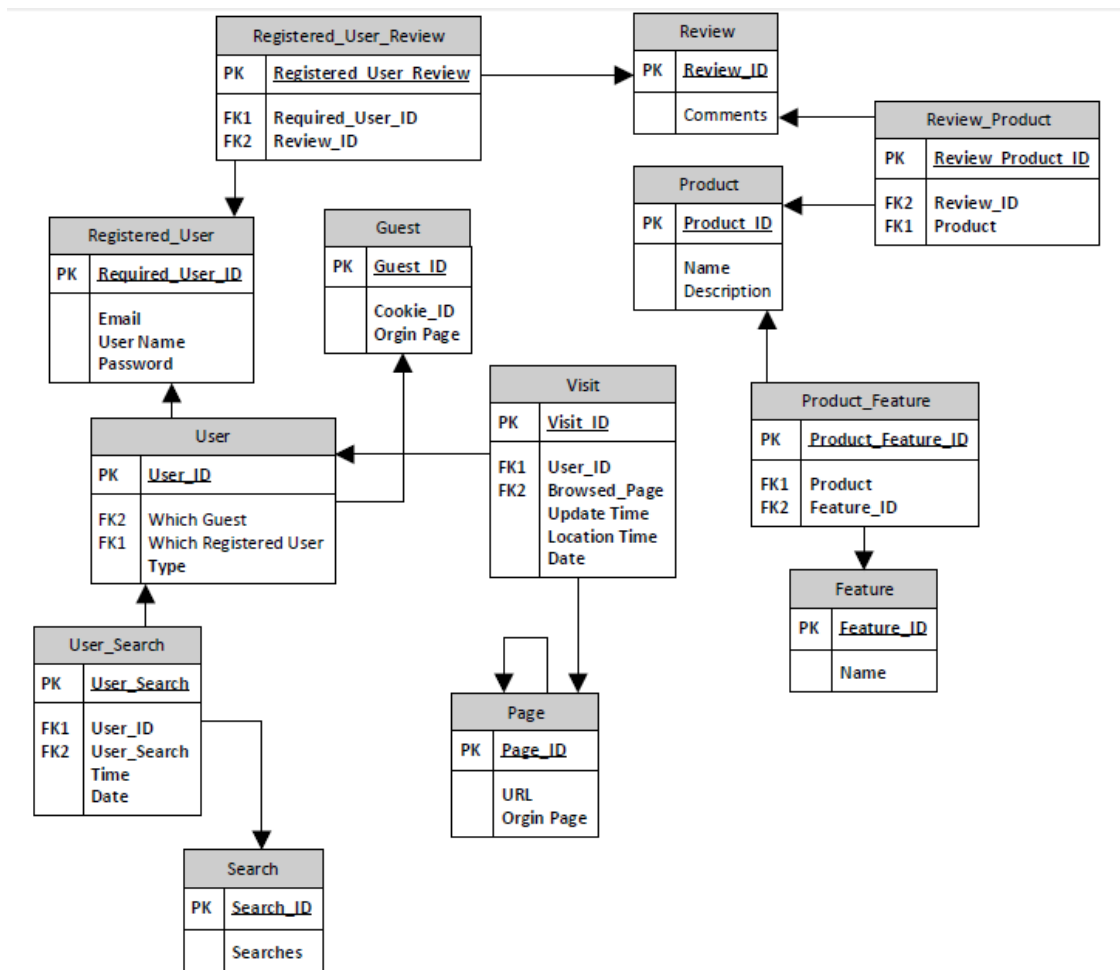


Figure 32. Initial Class Diagram

Objectives:

- Clear definition of the project's idea and goals
- Creating the list of functionalities
- Picking tools and becoming familiar with them

Risks:

- Each member could have many ideas, and not all of them would be compatible if ideas of other team members.
- Limited tool research possibilities - only open source or educationally licensed tools could be used.

PHASE 2:

In this stage we began implementing the most fundamental parts of the system. Having decided upon the databases, we created their instances and most basic database objects. From the list of functionalities an entity-relationship diagram was created.

Objectives:

- Create ER Diagram
- Set up Oracle and Neo4J databases

Risks:

- Detailed and well thought through database architecture could be time consuming
- Further addition of features could force us to change designed architecture or the architecture would not allow for addition of new features the way that we planned to incorporate them.

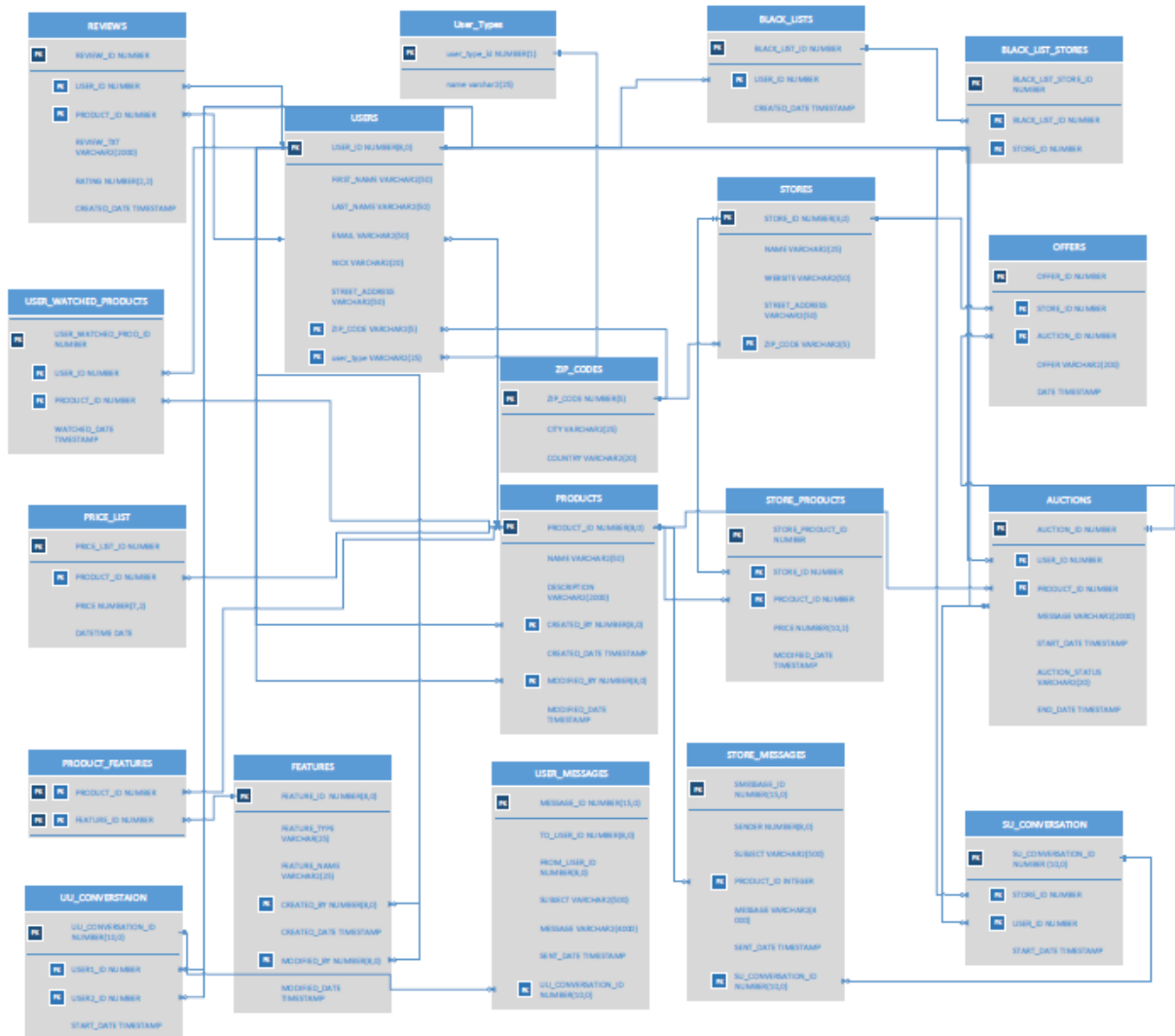


Figure 33 ERD in progress

PHASE 3:

This was the most troublesome and demanding period. Here, we have focused on developing the proper communication between our core database (Oracle) and the search engine that is based on the Neo4j database. In order to achieve our goals we had to look for and try many new ideas for solutions and in the end we came up with an optimal communication method (described in chapter 2.4 Oak DBSync Module). It still is not perfect however for the current size and estimated workload it is satisfactory and can be easily adapted in the future. New functionalities were added to the Oracle DB (messaging, product related media storage and user support).

Objectives:

- Find an optimal and effective method of connecting the databases
- Implement all necessary algorithms and database objects
- Implement new functionalities

Risks:

- No simple solution based on already existing libraries, which would force use to somehow restructure our project
- Communication would be too slow and inefficient

PHASE 4:

In this part, our focus shifted towards UX and development of the website. Using Ruby on Rails we created all panels: user's, admin's and store owner's. Simultaneously we implemented any changes and fixed bugs that came up during testing full communication between the website and databases.

Objectives:

- Design an intuitive and eye pleasing interface of the application for all users.

Risks:

- Implementation of the interface could be very time consuming and take long than expected
- Issues related to the database structure could arise while implementing the interface and additional work would be needed to unify all modules of the project. This would significantly slow down the development process.

PHASE 5:

After implementing functionalities needed for proper operation of the system, the next step was to work on additional features. Here we developed the third, NoSQL, database that supports gathering data about usage, users and data traffic. Gathered information is used for statistic available to administrators or store users. This part of the system has still a great potential for further development and improvement, and solutions used were implemented keeping in mind the possibility of future modifications.

Objectives:

- Design and create MongoDB database.
- Connect it to the user interface and implement functionalities connected to gathering user and product statistics.

Risks:

- Not enough time to implement all functionalities.
- Problems with connecting yet another database to the application.
- Slower operating application which would be unattractive and even frustrating for the potential user.

4. TESTING

After finishing developing our project, the next step was to conduct testing of the application. We showed the user module to a couple of testers and allowed them to explore the application as if they encountered it on their own. This gave us insight into flaws in our design and allowed us to improve the user experience by making it more intuitive or by providing clear instructions

or comments. Part of the testing was also performed using Selenium, a web browser automation software.

Selenium is a very useful tool for developers to make their test cases. It is a basic plugin for the Firefox browser that lets you record any actions that you may partake in. In addition a user may first select a UI element from the current browser display and then select a command with a pre-defined parameter that fits within the context of the UI element. This is a convenient tool for testers, as every step is not only recorded it may as well be played back as the starting URL is defined correctly. We used Selenium IDE to test whether our system behaved in the way it was supposed to. We found this tool very useful in testing whether the correct data would arrive upon insertion, the right link would open upon click, and if the data displayed was free from error.¹⁴

4.1 Performed tests and obtained results

While testing the reviews of products in our system, we found that the system works flawlessly with one review per user per product. We tested this using selenium, in which we set the string value we wished to appear after submitting the review and the value of the review. As expected the values of both were exact to as the user input them. After playing back the recording, an error would appear on the webpage signalling that the user had already had written a review for the given product which was expected. This way we would prevent any spammers on our page. We strongly believe that one review per a product per a user is enough to get out any disgruntle or praise the person may have for the product and or acquirement of it.

During the testing of Log in credentials, we tested if for an existing user - given the correct data they should be able to log into their personal profile. Through selenium we passed the correct login and password credentials and played the test. The effects were positive, and the user after creating his personal profile with his login and password could easily log in and out of the system.

Starting off the testing section is the ability to register. The Selenium IDE test looks like the following:

¹⁴ "What Is Selenium?" *SeleniumHQ Browser Automation*. (accessed 09 June 2014)

User First Registration		
open	/	
clickAndWait	link=Register	
type	id=user_nick	person
type	id=user_first_name	Peter
type	id=user_last_name	Piper
type	id=user_zip_code	05100
type	id=user_email	rock@solid.ap
type	id=user_password	arthurplonski
type	id=user_password_confirmation	arthurplonski
clickAndWait	name=commit	
verifyAlert	id=flash_notice	You signed up successfully
verifyAlert	id=flash_notice	valid

Table 5. Testing schema for user registration

From our home page the user will click on register in order to create a profile. This command is: clickAndWait on the target: link=Register. Next the user has to fulfill all the credentials in order to create a unique profile. In the above test you can see that the command for all the field types is: type and the target is described by id= and its respected field name. In the example the user sets the value of his username/user_nick to person, first_name to Peter, last_name to Piper, zip_code to 05100, user_email to rock@solid.ap, id_userpassword to arthurplonski, and the same is repeated for user_password_confirmation. The next command in the test is clickAndWait for when the commit button is pressed. If everything is successful the two messages should pop up “You signed up successfully “and “Valid”. The only errors we had here was when the user would input a zip-code starting with anything but 0 for example 80003. We quickly however fixed the code allowing for a non-zero value as the first number in the sequence. Other fields that have a strict structure for the user to follow are email with a string@string.string format, password having to be at least 8 characters, and password confirmation which the same rules apply. If we were to run the test above it would fail because each user is supposed to be unique so the error would pop up that the username and email are already taken. In order to check for regression tests the last two lines of the test have to be removed.

Regression tests are very valuable to developers, in that they show errors when changes are made. This is very helpful when you are working on separate parts of your project. Once you have made changes to a certain part you can go back and run all the tests you previously catalogued as working. If an error shows up, then you will know that an error was made while you were working on the different section of the project. We found Selenium Ide to be very helpful in this, as we tested every working portion of our project separately. After each change was made to our system, we would quickly run all the tests that we have saved and see if errors would occur. Thanks to this, we never had to second guess where the error in coding was made.

During the testing of log in credentials we tested, that if for an existing user given the correct data, they should be able to log into their personal profile. Through selenium we passed the correct login and password credentials with similar commands to the registration and played the test. The effects were positive, and the user after creating his personal profile with his login and password could easily log in and out of the system. The last part of the test was to check if the message “Thank you for logging in first_name last_name (user_nick)” was displayed. The log in test for the user Arthur Plonski is displayed below.

LogIn Check		
open	/	
clickAndWait	link=Log in	
type	id=username_or_email	arthur@usetheox.com
type	id=login_password	arthurplonski
clickAndWait	name=commit	
type	id=flash_notice	Thank you for logging in Arthur Plonski (aplonski)!

Table 6Testing schema for Logging in

While testing the reviews of products in our system, we found that the system works flawlessly with one review per a user per a product. We tested this using selenium, which we set the string value we wished to appear after submitting the review and the value of the review. As expected the values of both were exact to as the user input them. After playing back the recording, an error would appear on the webpage signaling that the user had already had written a review for the given product which was expected. This way we would prevent any spammers on our page. We assume that one review per a product per a user is enough to get out any disgruntle or praise the person may have for the product and or acquirement of it.

4.2 Test Results

After finishing all functionalities, application has been tested for the last time to ensure that invented functions didn't bring any regression. During the last test also security issues, from the OWASP Top 10 list¹⁵, have been taken into consideration. Following conclusions have been made:

1. Tested application parts are robust to CSRF (Cross Site Request Forgery attack).
2. Displaying of user input is protected against XSS (Cross Site Scripting) by internal Ruby on Rails mechanisms.
3. Application is protected against SQL-Injection attack by usage of prepared statements and parameters binding.
4. Passwords in database are salted and hashed using the Bcrypt algorithm.
5. Application is not protected against "sniffing" of login credentials. To protect it SSL certificate should be used.
6. Application is not protected against uploading malicious files (Executable can be uploaded with expected content type HTTP header when uploading product photos).
7. Database passwords should be moved to environment variables rather than kept in files.
8. Session management should be improved – when cookie is set it's enough to authenticate user.

Found security issues should be fixed in the future, however these are not the issue to consider application as insecure.

5. FUTURE DEVELOPMENT AND USE

5.1 Facilitation and automatisisation of adding new products

Currently the process of adding new products is time and resource consuming. The forms and admin approval provides security and does not allow incorrect or redundant data to be inserted into the system, however a more user friendly and more efficient method would be preferred. Also it would be beneficial if all users could make additions to stored data – store users with

¹⁵ "Category:OWASP Top Ten Project." (accessed 12 June 2014)

products that they have available, with related features and registered users with products that they are interested in but are not in the system.

5.2 Store ratings

This additional feature would work well with already implement one the Black Lists. Each store has its profile and it would be very informative for the user to see how others feel about making purchases from this seller. Users would be able to give a numerical rating, possibly in couple of different categories, along with a written comment. The average of numerical ratings could be displayed. This could be taken into consideration in displaying statistics or products recommended for a user (offers from stores that were highly graded by the user could be displayed at the top of the page).

5.3 Security

Security is always an aspect that can be improved in an application. It would be important to decrease any chance of unwanted activity such as insertion of not existing products or changing store information. It is also necessary that offers made by stores during an auction are not altered.

5.4 Backup plans

As the system grows, it will store more information and with additional features implemented losing any of that information would cost not only time to recollect it but also, in case of commercial use, it would be a cause of possible unwanted money loss. If the Oracle database is upgraded to a more advanced edition, the built in backup features would allow restoring data to exact point of time when a failure occurred. Creating a detailed and well thought out back up plan should be the next step in development of this project. Automatisation would guarantee that the data is always backed up and that the administrator will not have problems with database recovery.

5.5 Mobile version

In the era of smartphones, every respectable website should have a well-designed mobile version. Further development of the mobile version for our application would bring us closer to our users, allowing them to use our services at any given time and place. This will have a great marketing impact as the more accessible the website will be, the more stores would be interested in working with us. It is important to remember that mobile versions have to be thought through to make the user experience intuitive, simple and pleasant. It would also be greatly beneficial to develop and release a native application. People would be more likely to use our application on their smartphones through a downloadable application instead of using a mobile version of the website, as the downloaded app could provide additional features.

5.6 Admin API

Thinking also about the administrators a further development of provided APIs would be recommended. This provided the admins with additional and friendlier tool for managing the data inside the database. Instead of having to write complex procedures or queries each time, the admin will be able to call a built in function or a procedure and get necessary work done much faster. The API should contain basic procedures for managing tables' contents as well as analytical tools for gathering additional statistics and generating reports. Of course DBAdmins would still have to work directly with the database for maintenance purposes and problem solving, but the content management would be significantly improved and made easier.

6. FINAL NOTE

Working on this project has been a great learning opportunity as well as a demanding test of our abilities. Throughout the development process we have encountered a number of problems that required creative thinking along with gathering new knowledge.

Comparing our initial expectations for the final product with the project presented in this paper we find that it has met our goals. All of the users are presented with a fully functional and well operating application that meets the desired criteria. Oak Search provides a unique service with available features that are not present in other similar applications that we have met with. It is also ready for further development without significant changes to the existing code if there would be a demand for incorporating and providing more functionalities.

7. INDEX

All-paths algorithm -----	p. 14
Dijkstra algorithm -----	p. 14
Graph -----	p. 14
Graph database -----	p. 13
Guest -----	p. 8
MVC -----	p. 24
Node -----	p. 13
Oak Admin Module -----	p. 21
Oak DBSync Module -----	p. 22
Oak Front Module -----	p. 21
Oak PriceWatcher -----	p. 22
Oak Shop Module -----	p. 22
Shortest path algorithm -----	p.14

8. BIBLIOGRAPHY

Publications

1. Mehlhorn, Kurt, and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Berlin: Springer, 2008. Print.
2. Rege Gautam, *Learning Mongoid.*, PacktPublishing, 2013

Web Sources

3. "Avoiding Mutating Tables." *Avoiding Mutating Tables*. Ask Tom Oracle, <https://asktom.oracle.com/pls/asktom/ASKTOM.download_file?p_file=6551198119097816936>.
4. Backhouse, Roland. *All-Paths Algorithm*. 22 Oct. 2002. PDF.
5. Bastani, Kenny. "Natural Language Search Using Neo4j." *Natural Language Search Using Neo4j.*, 30 Oct. 2013. <<http://www.slideshare.net/KennyBastani/natural-language-search-using-neo4j>>.
6. Boehm, Barry W. *A Spiral Model of Software Development and Enhancement*. 1988. PDF. <<http://csse.usc.edu/csse/TECHRPTS/1988/usccse88-500/usccse88-500.pdf>>.
7. Braten, Morten. "REF Cursor to JSON." Web log post. *ORA-00001: Unique Constraint Violated*. 11 Feb. 2010. <<http://ora-00001.blogspot.com/2010/02/ref-cursor-to-json.html>>.
8. "Category:OWASP Top Ten Project." - OWASP. <https://www.owasp.org/index.php/Category%3AOWASP_Top_Ten_Project>.
9. Dean, Jeffrey, and Sanjay Ghema W *MapReduce: Simplified Data Processing on Large Clusters*. Google Inc., 2004. PDF.
10. "Node (Neo4j Community 2.0.2 API)." *Node (Neo4j Community 2.0.2 API)*. <<http://api.neo4j.org/2.0.2/org/neo4j/graphdb/Node.html>>.
11. "Oracle® Database Express Edition." *Oracle® Database Express Edition*. Oracle, 2012. Web. <http://docs.oracle.com/cd/E17781_01/license.112/e18068/toc.htm#XELIC119>.
12. Sedgewick, Robert, and Kevin Wayne. *Shortest Paths*. Princeton University, 2007. PDF.

13. "The Neo4j Manual V2.1.27.1. What Is Cypher?" *7.1. What Is Cypher?*
<<http://docs.neo4j.org/chunked/milestone/cypher-introduction.html>>.
14. "TortoiseSVN." *About TortoiseSVN*. <<http://tortoisesvn.net/about.html>>.
15. "What Is Selenium?" *SeleniumHQ Browser Automation*. <<http://docs.seleniumhq.org/>>.

9. TABLES AND FIGURES

List of Code Listings

Code Listing 1. 'Record not found' action.....	27
Code Listing 2. User authentication method.....	28
Code Listing 3. Authenticate_user Filter	29
Code Listing 4. Product Model.....	32
Code Listing 5. Admin authentication filter	34
Code Listing 6. Uploading picture files security	35
Code Listing 7 - starting threads.....	40
Code Listing 8 - waiting for the place in the queue	41
Code Listing 9. Generating XML from sys_refcursor	42
Code Listing 10. After delete trigger on Products table	43
Code Listing 11. Exemplary JSON response to price change query	44
Code Listing 12. Exemplary JSON response to query without parameters.....	45
Code Listing 13. PriceInfo implementation.....	46
Code Listing 14. Natural Language Parsing grammar example	47
Code Listing 15. Translation rules to Cypher.....	47
Code Listing 16. Exemplary Sequence Definition	57
Code Listing 17. Product_api package definition.....	58
Code Listing 18. Example of sequence use	58
Code Listing 19. User_api package definition.....	58
Code Listing 20. Ticket_api package definition	59
Code Listing 21. Definition of Product document.....	63
Code Listing 22. Map Reduce job example.....	64
Code Listing 23. Definition of product UPSERT	64
Code Listing 24. Obtaining product statistics.....	65

Code Listing 25. Exemplary product statistics chart	65
-----------------------------------------------------------	----

List of Figures

Figure 1. Guest Use Case Diagram.....	10
Figure 2. Registered User Use Case Diagram	11
Figure 3. Store Owner Use Case Diagram.....	12
Figure 4. Admin Use Case Diagram.....	13
Figure 5 Search Engine search box.....	24
Figure 6 Exemplary search result	25
Figure 7 'Exact match' button.....	25
Figure 8. Front Module - website main page.....	26
Figure 9 Controlles Diagram	27
Figure 10 Home Controller.....	28
Figure 11. UserController	29
Figure 12. Srssion Controller.....	29
Figure 13. Product Controller	30
Figure 14. Stores Controller.....	30
Figure 15. Reviews Controller.....	31
Figure 16. Admin module main page	33
Figure 17. Oak Admin Module Application Controllers Diagram	34
Figure 18. Oak Admin Products Controller.....	34
Figure 19. Oak Store User Module Application Controllers Diagram	36
Figure 20. Store Products Controller	37
Figure 21. Offers Controller	37
Figure 22. Store Users Controller	37
Figure 23. Database synchronization chart.....	38
Figure 24. PriceWatcher application structure.....	46

Figure 25. User Actions Entity Relation Diagram.....	50
Figure 26. Communication Entity Relation Diagram.....	51
Figure 27. Product Management Entity Relation Diagram	52
Figure 28. System Operations tables	53
Figure 29. Diagram of Neo4J database.....	61
Figure 30. Graph structure example.....	62
Figure 31. Spiral Model Development Organization.....	67
Figure 32. Initial Class Diagram.....	68
Figure 33 ERD in progress	70

List of Tables

Table 1 RoR third party libraries	20
Table 2 Java connector libraries	21
Table 3 Third party libraries used in Price Watcher module	22
Table 4. Additional parameters in JSON response	45
Table 5. Testing schema for user registration.....	74
Table 6Testing schema for Logging in	75

ATTACHMENTS

Users' Manual

Installation Guide

USER'S MANUAL

Table of contents:

I.	Guest User.....	89
	a. Registration.....	90
	b. Browsing Products.....	91
	c. Comparing Products.....	91
	d. Browsing Stores.....	93
	e. Browsing Reviews.....	94
	f. Statistics	96
	g. Legal Information.....	96
	h. Contacting application owners.....	97
II.	Registered User.....	98
	a. Profile Management.....	99
	b. Messages	101
	c. Viewing a product.....	102
	d. Price Offers.....	103
	e. Reviews.....	104
	f. User Support – Tickets.....	105
III.	Store Owner/Admin.....	107
	a. Managing Users.....	108
	b. Managing Store’s Products.....	110
	c. Making a Price Offer.....	112
IV.	Admin.....	113
	a. Admin Dashboard.....	113
	b. Statistics	113
	c. Manage Options.....	114
	d. Manage Stores.....	116
V.	Mobile Version	118

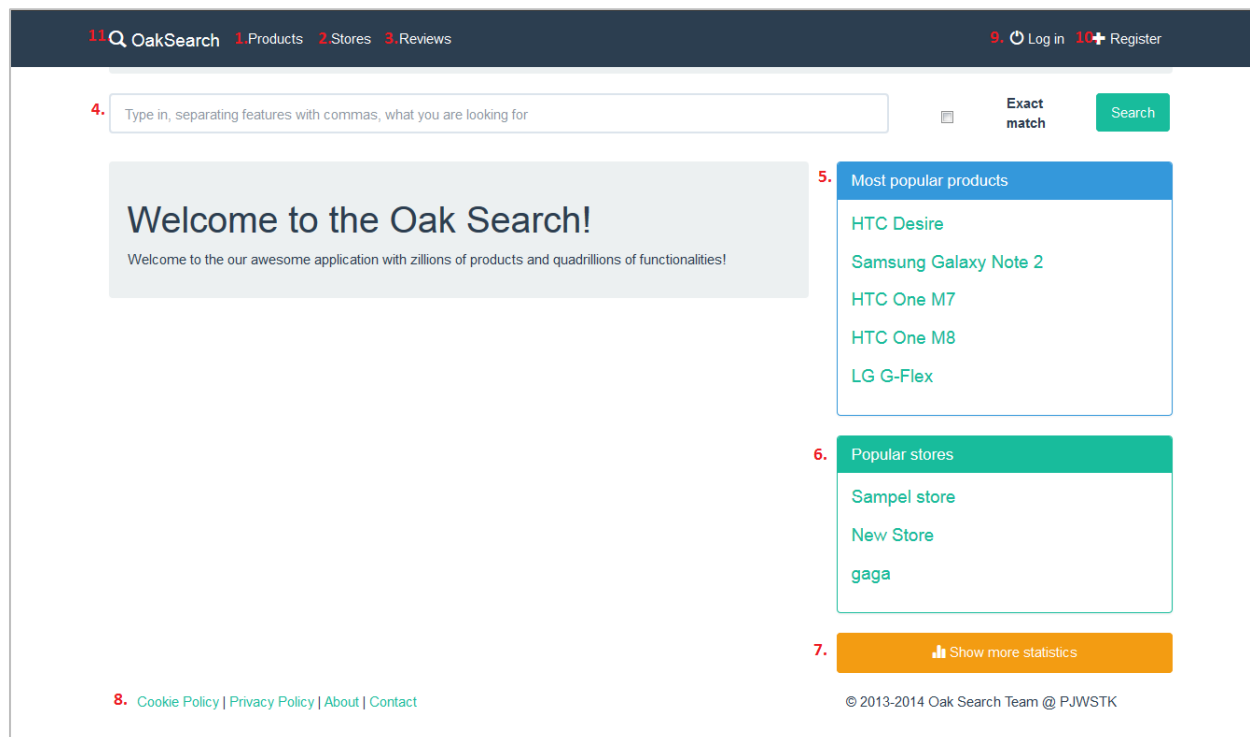
I. Guest User

The image below presents the main page of the front module of the application. This is the page that users see first when using our application. The options available from this page are:

- browsing products(1), stores(2), and reviews(3)
- conducting a product search based on parameters entered into search text box(4)
- viewing the most popular products(5)
- viewing the most popular stores(6)
- viewing more statistics (7)
- obtaining legal information about the application (8)
- logging in(9) or registering(10)

The difference between the two searches is that an exact match search will find any product with the exact words you enter in separate strings. While the regular search will only work with the order of key words you input into the search box. In order to conduct an exact match search the radio box has to be checked before clicking the search button.

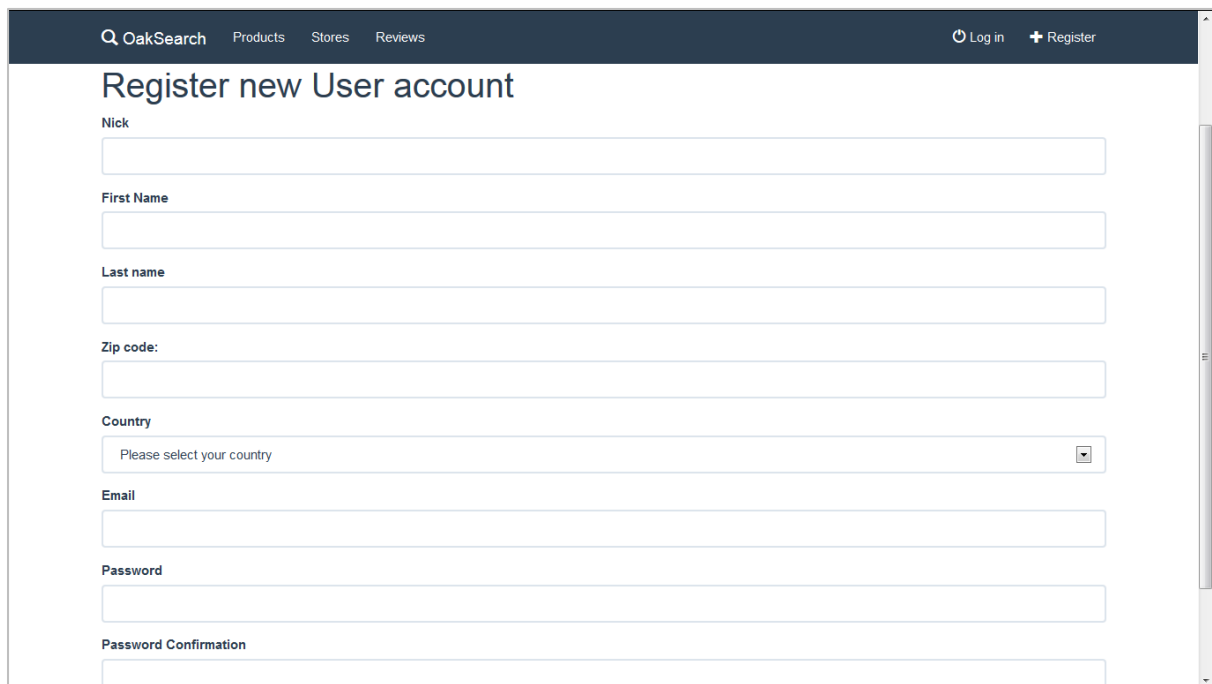
The “OakSearch” logo (11) may be pressed at any time during the browsing period in order to return the user back to this home page.



If the user is logged in, in this section the “logout” option will appear, which allows the user to log out of his personal profile.

Registration

To sign up a guest has to press the “Register” button on the main page (9). A form shown below will appear, where user will have to fill out all his information correctly and then click ‘Signup’.

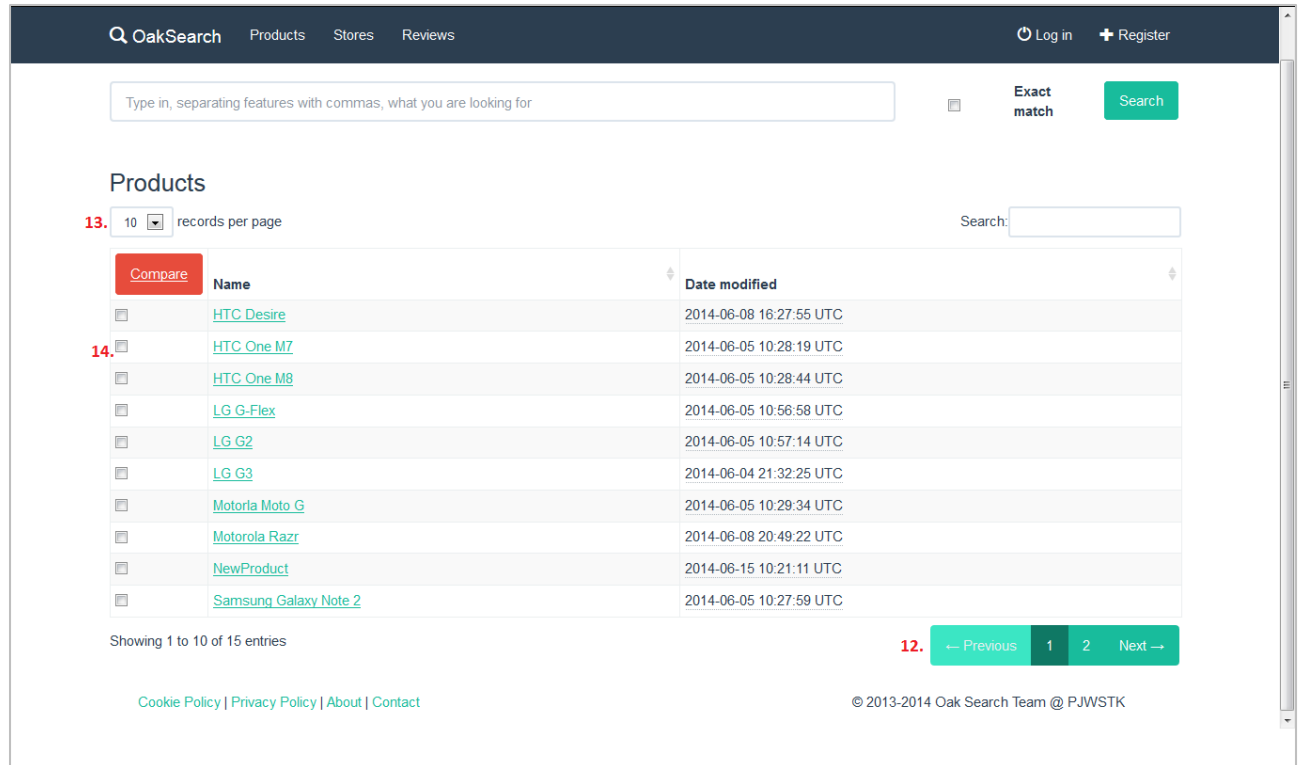


The screenshot shows a web application interface for user registration. At the top, there is a dark blue header bar containing a search icon and the text 'OakSearch', followed by navigation links for 'Products', 'Stores', and 'Reviews'. On the right side of the header, there are links for 'Log in' and '+ Register'. Below the header, the main content area has a title 'Register new User account'. The form consists of several input fields: 'Nick', 'First Name', 'Last name', 'Zip code:', 'Country' (a dropdown menu with the placeholder text 'Please select your country'), 'Email', 'Password', and 'Password Confirmation'. Each field is represented by a white rectangular box with a thin border. A vertical scrollbar is visible on the right side of the form area.

A short message box will alert the user of his success in creating an account.

Browsing Products

When the products option is chosen (1), the new page will look as shown below.



A list of all the products is given by their name and modification date. The user may browse each product by clicking the name of the product which will redirect them to that item.

Only 10 entries are shown per page, but a navigation panel is located at the bottom right of the screen (12).

If the user wishes to view more products per a page this is possible through a simple drop down right below the “Products” label (13). The number of products viewed can be of 10, 25, 50, or 100 items whichever is to the users’ request.

Comparing Products

The user may also compare a number of products (this amount differs for registered users and guests). To compare items with each other the user must check the check boxes located to the left of the product title (14). After the selection has been complete a simple click of the button

‘Compare’ will lead to opening a pop-up window which shows the comparison being made between the items.

The comparison window between two products is shown in the image below. A guest may only compare two items at a time and a registered user may compare up to five items.

The screenshot displays a web application interface with a dark header. The header includes a search bar labeled 'OakSearch', navigation links for 'Products', 'Stores', and 'Reviews', and user options for 'Log in' and 'Register'. Below the header, a breadcrumb trail shows 'You are here: / home / products'. A search input field is present with the placeholder text 'Type in, separating features with comma'. The main content area is titled 'Products' and features a 'Compare' button and a 'records per page' dropdown set to '10'. A list of products is shown, including HTC Desire, HTC One M7, HTC One M8, LG G-Flex, LG G2, LG G3, Motorola Moto G, Motorola Razr, NewProduct, and Samsung Galaxy Note 2. A comparison window is open, titled 'Comparison results', showing a table with columns for 'Feature', 'LG G-Flex', and 'LG G3'. The table lists 'Screen Type' as 'Touchscreen' and 'Screen Technology' as 'OLED' for LG G-Flex and 'LED' for LG G3. A 'Close' button is located at the bottom right of the window. The footer shows 'Showing 1 to 10 of 15 entries' and pagination controls for 'Previous', '1', '2', and 'Next'.

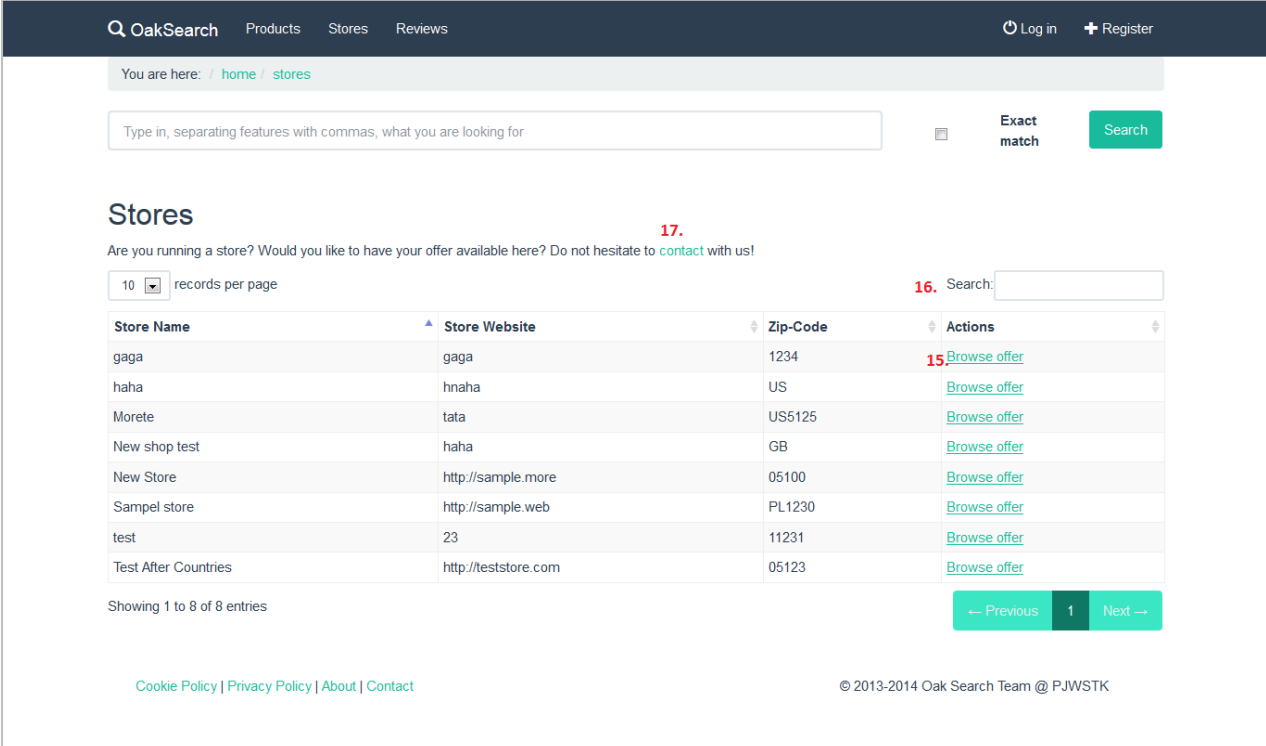
	LG G-Flex	LG G3
Feature		
Screen Type	Touchscreen	Touchscreen
Screen Technology	OLED	LED

Browsing Stores

The page listing all stores is shown in the picture below. The user may view all the stores available and access any of them through a simple click of the “Browse Offer” link (15). Analogically to browsing products, the amount of entries per a page can be manipulated through the drop down menu, and navigation is possible through the panel on the bottom right hand side.

A search bar (16) is also introduced on the right hand side in order to ease the user browsing the stores if he already knows what they are looking for.

In order for a user to create his own online store in our system he has to proceed with the contact us link located centrally on this page (17).



OakSearch Products Stores Reviews Log in Register

You are here: / home / stores

Type in, separating features with commas, what you are looking for

Exact match Search

Stores

Are you running a store? Would you like to have your offer available here? Do not hesitate to [contact](#) with us!

10 records per page

16. Search:

Store Name	Store Website	Zip-Code	Actions
gaga	gaga	1234	15. Browse offer
haha	hnaha	US	Browse offer
Morete	tata	US5125	Browse offer
New shop test	haha	GB	Browse offer
New Store	http://sample.more	05100	Browse offer
Sampel store	http://sample.web	PL1230	Browse offer
test	23	11231	Browse offer
Test After Countries	http://teststore.com	05123	Browse offer

Showing 1 to 8 of 8 entries

← Previous 1 Next →

[Cookie Policy](#) | [Privacy Policy](#) | [About](#) | [Contact](#)

© 2013-2014 Oak Search Team @ PJWSTK

Once a store page is opened it will look like the image below. User may view the different products, their price, and any actions concerning them. This page is very similar to the OakSearch products page in look and functionality. The official store website is provided if the user feels that is a better way for them to go about his search (18).

OakSearch
Products
Stores
Reviews
Log in
Register

You are here: / [home](#) / [stores](#)

☐
Exact match

18. **Sampel store**

This store is with us since 2014-06-08 21:13:33 UTC

This store's official website is: <http://sample.web> And the street address is:
Sampel address

Store's offer:

10 records per page
Search:

Product Name	Price	Actions
HTC Desire	126.0	Ask about this product
HTC One M7	629.0	Ask about this product
HTC One M8	799.0	Ask about this product
iPhone 6	120.0	Ask about this product
LG G-Flex	123.0	Ask about this product
Motorola Razer	120.0	Ask about this product

Browsing Reviews

When the review tab is selected a window similar to the one below is displayed. All the reviews are visible with specifications for each. The user may go directly to the reviewed product by clicking on the link, or may view the review itself. Amount of views can be manipulated as well as easy navigation is provided. A search bar is also present in order to make any browsing easier to the client.

OakSearch
Products
Stores
Reviews
Log in
Register

You are here: / home / reviews

Type in, separating features with commas, what you are looking for
Exact match
Search

Reviews
10 records per page
Search:

Rating	Author	Product	Date added	Actions
1	test	HTC Desire	2014-06-09 11:31:03 UTC	Show review
2	test	HTC Desire	2014-06-08 11:06:42 UTC	19. Show review
3	test	LG G2	2014-06-09 11:22:27 UTC	Show review
5	aplonski	iPhone 6	2014-06-11 05:05:48 UTC	Show review
5	aplonski	LG G-Flex	2014-06-11 06:03:03 UTC	Show review
5	test	HTC Desire	2014-06-10 23:49:46 UTC	Show review
5	aplonski	Samsung Galaxy Note 2	2014-06-11 23:57:08 UTC	Show review
5	aplonski	HTC One M8	2014-06-11 03:29:20 UTC	Show review
6	test	HTC Desire	2014-06-10 23:52:45 UTC	Show review
6	aplonski	HTC Desire	2014-06-09 19:33:32 UTC	Show review

Showing 1 to 10 of 10 entries
Previous
1
Next

[Cookie Policy](#) | [Privacy Policy](#) | [About](#) | [Contact](#)

© 2013-2014 Oak Search Team @ PJWSTK

When a customer clicks ‘show review’ (19) a window will appear that is almost exact to the one below. The content of the submitted review is shown with all its details.

OakSearch
Products
Stores
Reviews
Log in
Register

You are here: / home / reviews / review of Samsung Galaxy Note 2 by aplonski

Type in, separating features with commas, what you are looking for
Exact match
Search

Review of Samsung Galaxy Note 2

Rating: 5
Testing

written by aplonski on 2014-06-11 23:57:08 UTC

More reviews of Samsung Galaxy Note 2

Other reviews by aplonski

[Cookie Policy](#) | [Privacy Policy](#) | [About](#) | [Contact](#)

© 2013-2014 Oak Search Team @ PJWSTK

Statistics

Showing more statistics from the homepage will provide a page which shows all: recently added products, recently updated products, recent reviews, recent comments, recent price raises, and recent price drops. To navigate to a specific field of interest the user just has to click the link and they will be taken directly to that spot.

Products recently added

Name	Date added
NewProduct	about 12 hours ago
michaltest	about 12 hours ago
Samsung Galaxy s5	2 days ago
Testmemore	5 days ago
iPhone 6	5 days ago
Motorola Razr	10 days ago
HTC Desire	10 days ago
Motoria Moto G	10 days ago
HTC One M8	10 days ago
HTC One M7	10 days ago

[View more »](#)

Products recently updated

Name	Date updated
NewProduct	about 11 hours ago
michaltest	about 12 hours ago
Samsung Galaxy s5	2 days ago
Testmemore	5 days ago
iPhone 6	5 days ago
Motorola Razr	7 days ago
HTC Desire	7 days ago
LG G2	10 days ago
LG G-Flex	10 days ago
Motoria Moto G	10 days ago

[View more »](#)

Recent reviews

Product	Rating	Date
View more »		

Recent comments

Name	Date
View more »	

Recent price rises

Product	Date
View more »	

Recent price drops

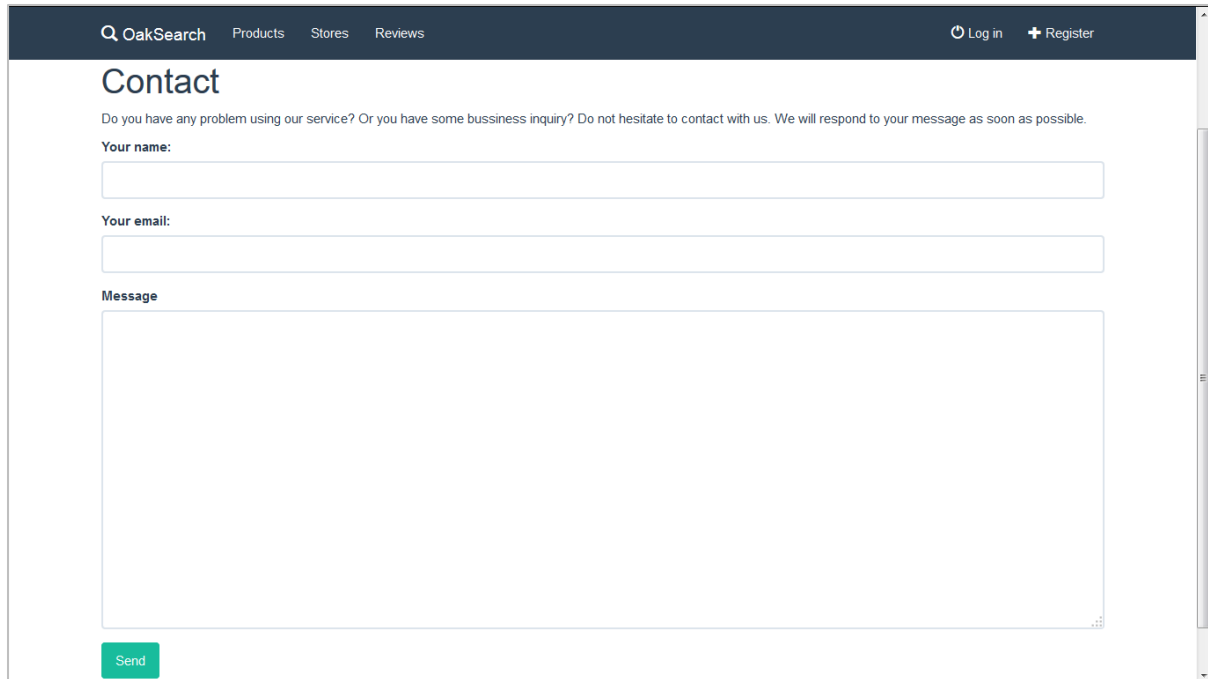
Product	Date
View more »	

Legal information

When the user sees the menu below he may take advantage of it to view the cookie policy, privacy policy, about us, and go to contact information. The first three links take you to a page that explains the meaning of each and educates the user on what we do.

Contacting application owners

When the user selects “Contact” (20) a screen appears exactly like the one below. After filling out all the credentials the user may submit his entry and it is sent to the admins email.



The screenshot shows a web browser window displaying the 'Contact' page of the OakSearch website. The page has a dark blue header with the 'OakSearch' logo and navigation links for 'Products', 'Stores', and 'Reviews'. On the right side of the header are links for 'Log in' and 'Register'. The main content area is white and titled 'Contact'. Below the title is a message: 'Do you have any problem using our service? Or you have some bussiness inquiry? Do not hesitate to contact with us. We will respond to your message as soon as possible.' The form consists of three input fields: 'Your name:', 'Your email:', and a larger 'Message' text area. A green 'Send' button is located at the bottom left of the form.

Contact

Do you have any problem using our service? Or you have some bussiness inquiry? Do not hesitate to contact with us. We will respond to your message as soon as possible.

Your name:

Your email:

Message

Send

II. Registered User

In order to log in as an existing user click the “Log in” button (9) and fill out your username or email and password. The screen below shows what the user will see as the login page.

The screenshot shows the OakSearch website's login page. At the top, a dark blue header contains the 'OakSearch' logo, navigation links for 'Products', 'Stores', and 'Reviews', and buttons for 'Log in' and 'Register'. Below the header, a light gray breadcrumb trail reads 'You are here: / home'. A search bar with the placeholder text 'Type in, separating features with commas, what you are looking for' is positioned above a 'Search' button. To the right of the search bar, there is a small icon and the text 'Exact match'. The main content area features a 'Log in' heading, followed by input fields for 'Username or Email:' and 'Password:'. A green 'Log In' button is located below the password field. At the bottom of the page, a footer contains links for 'Cookie Policy', 'Privacy Policy', 'About', and 'Contact', along with the copyright notice '© 2013-2014 Oak Search Team @ PJWSTK'.

Q OakSearch Products Stores Reviews Log in Register

You are here: / home

Type in, separating features with commas, what you are looking for

Exact match Search

Log in

Username or Email:

Password:

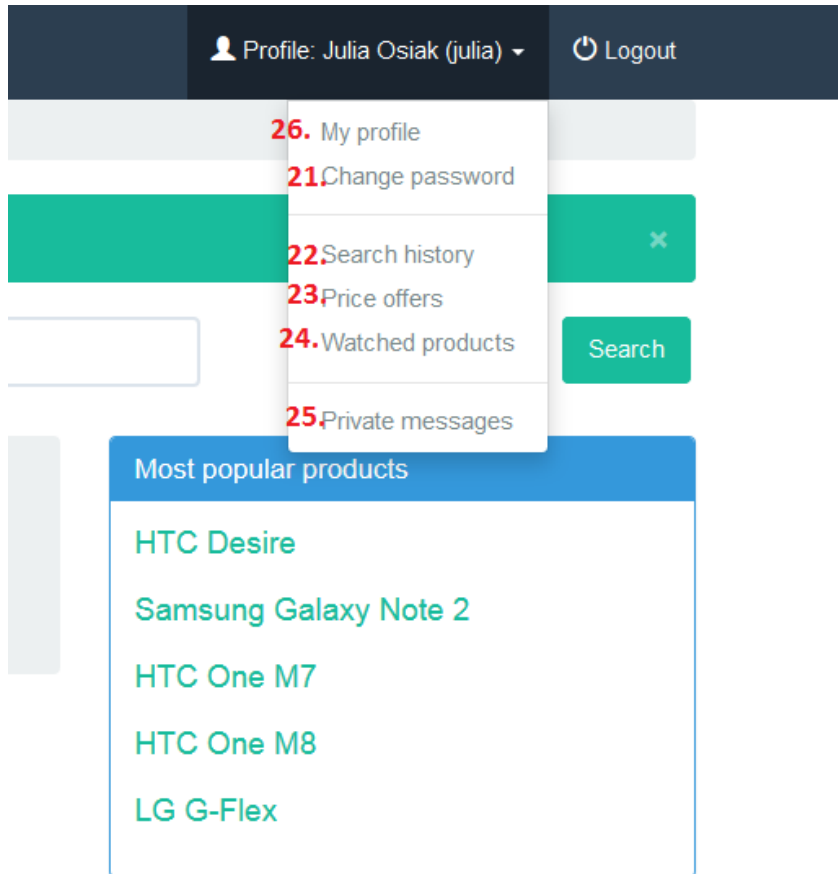
Log In

[Cookie Policy](#) | [Privacy Policy](#) | [About](#) | [Contact](#)

© 2013-2014 Oak Search Team @ PJWSTK

Profile Management

The drop down menu available to all registered users is displayed below.



“Change password” (21) option will redirect the user to a page where he can change current password.

“Search history” (22) will display the users browse history involving any products he may have viewed.

“Price offers” (23) will show the client any auctions he may have created.

“Watched products” (24) will show any products the user selected to watch.

“Private messages” (25) takes the user to history of his messages with other users, store’s or admins.

“My profile” (26) option the user will be displays a similar page to the one below. The user can view his profile information, edit any of it by clicking the “Edit Profile” button (27), and change his password (28).

OakSearch

ProductsStoresReviews

Profile: Julia Osiak (julia)Logout

You are here: / home / profile

Type in, separating features with commas, what you are looking for

Exact matchSearch

julia Profile

First NameJulia

Last NameOsiak

Nickjulia

Emailjulia@osiak.me

Address

Zip CodePL05800

27. Edit ProfileChange Password28.

Cookie Policy | Privacy Policy | About | Contact

© 2013-2014 Oak Search Team @ PJWSTK

Messages

While writing a personal message, the window will appear similar to the one below. After adding the recipient's e-mail, writing the message content and submitting, the message will be displayed on the page.

OakSearch

Products

Stores

Reviews

Profile: Julia Osiak (julia) Logout

Private Messages

Newest messages are on top.

>> Hey there!
What's up? This page is AMAZING!!!
2014-06-15 22:09:50 UTC from test to julia Reply

Hey there!
2014-06-15 22:09:03 UTC from julia to test Reply

hi!
2014-06-15 22:08:42 UTC from julia to aplonski Reply

Write new

Recipient

Message

Submit Query

Viewing a product

Upon selecting a chosen product the user will be taken to a page that looks similar to the one below. The user can view the gallery, features, description, and prices of the specific product. Along the right hand side the price history (29) and popularity of the item (30) can be seen as well. Any reviews written about the product are at the bottom of the page (31). A registered user has the perk of watching a product (32), getting price offers (33), and writing reviews (34).

HTC Desire [Watch this Product](#) [Get price offers](#) product information page

Gallery
There are no photos of this product yet, sorry.

Similar products graph [Redraw](#) [Click node to proceed to product](#)

Graph showing connections between products:
 - Samsung Galaxy S5 (2 features) → HTC Desire (3 features)
 - HTC One M8 (3 features) → HTC Desire (3 features)
 - LG G3 (2 features) → HTC Desire (3 features)
 - HTC One M7 (3 features) → HTC Desire (3 features)
 - Motorola Razer (2 features) → HTC Desire (3 features)

Features

Feature	Value
Screen Type	Touchscreen
Screen Technology	LED
Manufacturer	HTC

Description
HTC Desire1

Prices
Average price: 126 from 1 offer(s)

Price Source	Price
Sampel store	126

29. Price history

Line chart showing MIN, MAX, and AVG price history over time.

31. Reviews [+ write new](#) 34.

Review 1:
 Rating: 2
 test review
 Written by test on 2014-06-08 11:06:42 UTC

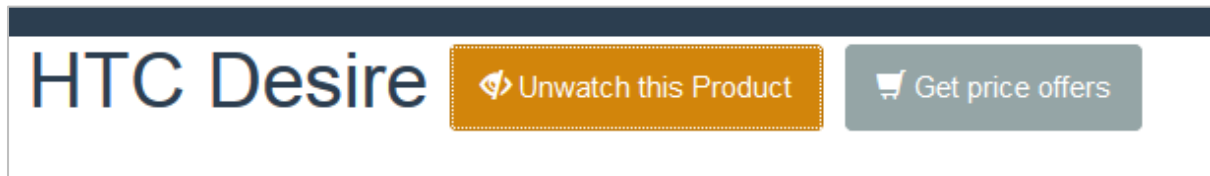
Review 2:
 Rating: 5
 xyz
 Written by test on 2014-06-10 23:49:46 UTC

Review 3:
 Rating: 6

30. Product popularity

Bar chart showing Hits over time.

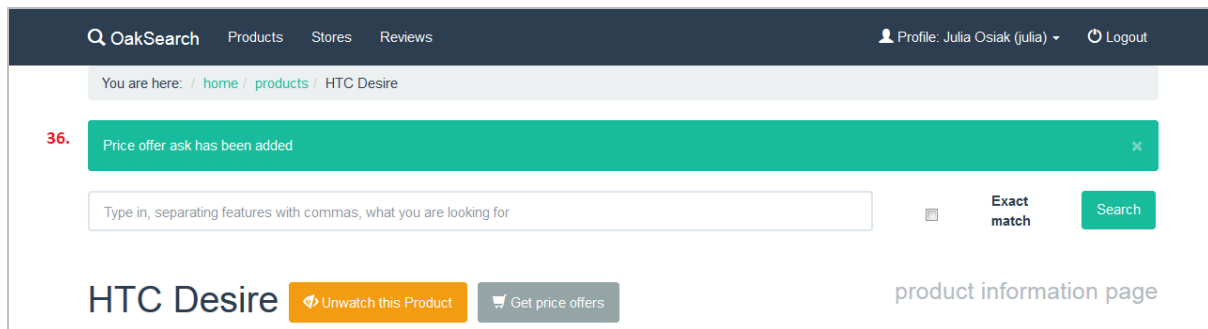
Watching a product is simply done by clicking the “Watch this Product” button (32) which turns to “Unwatch this Product” and the icon changes as well. Basically by this action the user will receive any information regarding changes to the specific product he is watching through email.



Price offers

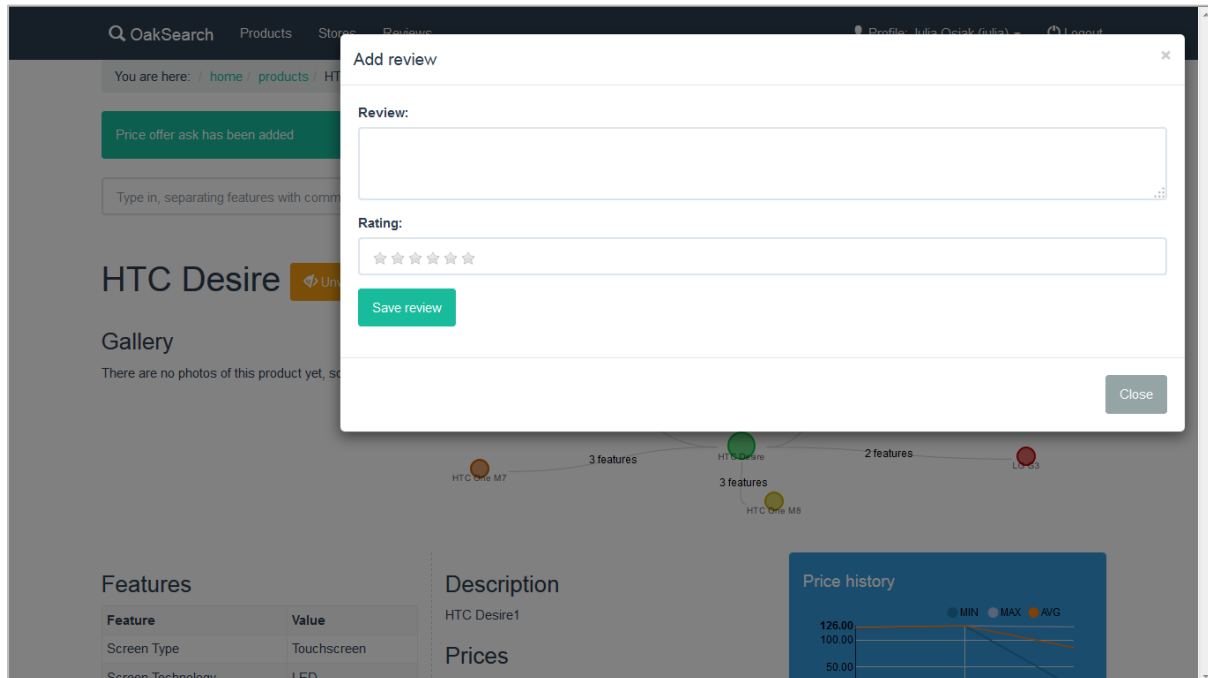
When the registered user clicks on “Get price offers” button (33) a window similar to the one below will appear. The product name is visible as well as a field where user can write his inquiry.

The user also has the option of filling out how long he is willing to hold the offer (35). Save the offer with the press of the button “Submit” and a message box will appear regarding acceptance of the request (36).



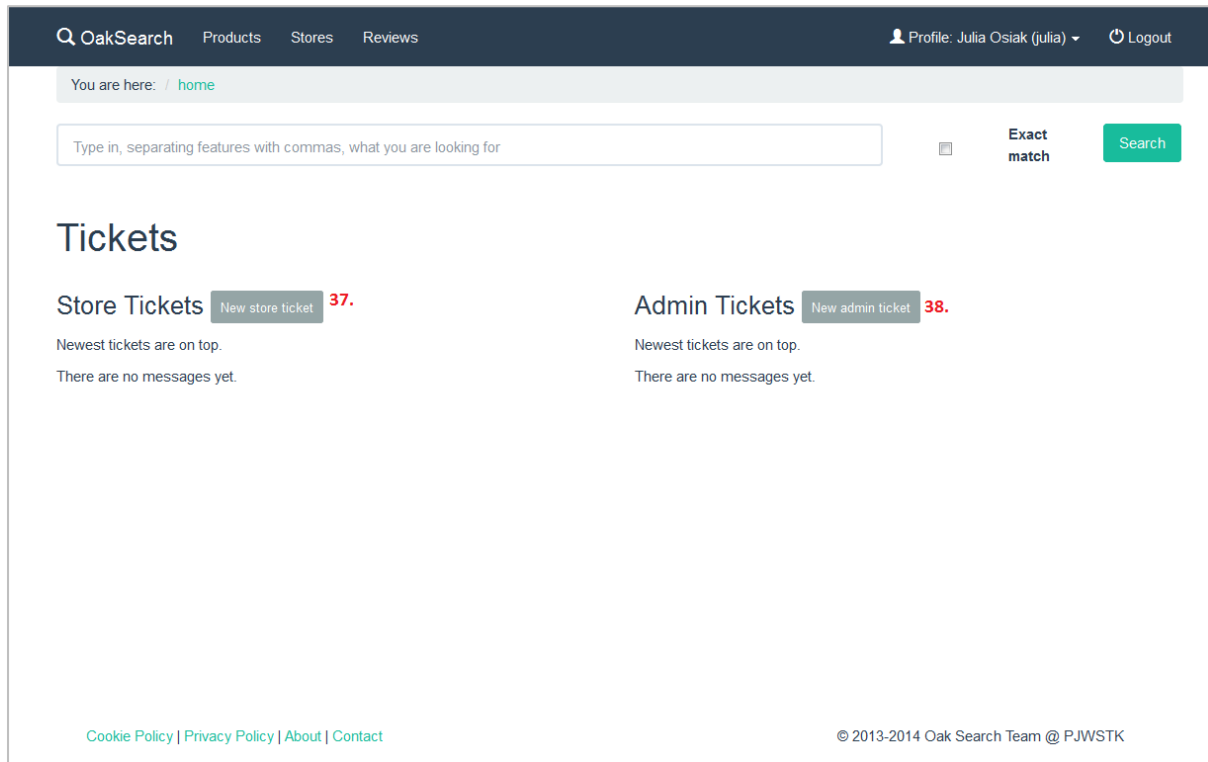
Reviews

When the user wishes to add a review he should click the “write new” button (34) next to the Review label. A window like the one displayed below will be appear. The user has the option submitting his opinion in the review section of the form as well as giving the product a rating from 1-6 depending on his preference. To share the review simply click the “share review” button, and a dialog box should follow confirming the request.



User Support – Tickets

If the registered user will need any support from the website admin or if he will want to communicate with a store he is provided with the feature of tickets. Choosing “Tickets” from the user’s drop down menu he will be redirected to the dedicated page which is shown below.



In order to send a message to the store he should click on the “New store ticket” button (37). Analogically to contact one of the admins he should select the “New admin ticket” button (38).

Considering the first scenario of contacting a store, the user will be then taken to a form where he can choose a store from the drop down list and write the desired message. This form is presented below.

OakSearch

Products

Stores

Reviews

Profile: Julia Osiak (julia)

Logout

You are here: / [home](#)

Type in, separating features with commas, what you are looking for

Exact match

Search

New Store Ticket

Store

Message

Send

[Cookie Policy](#) | [Privacy Policy](#) | [About](#) | [Contact](#)

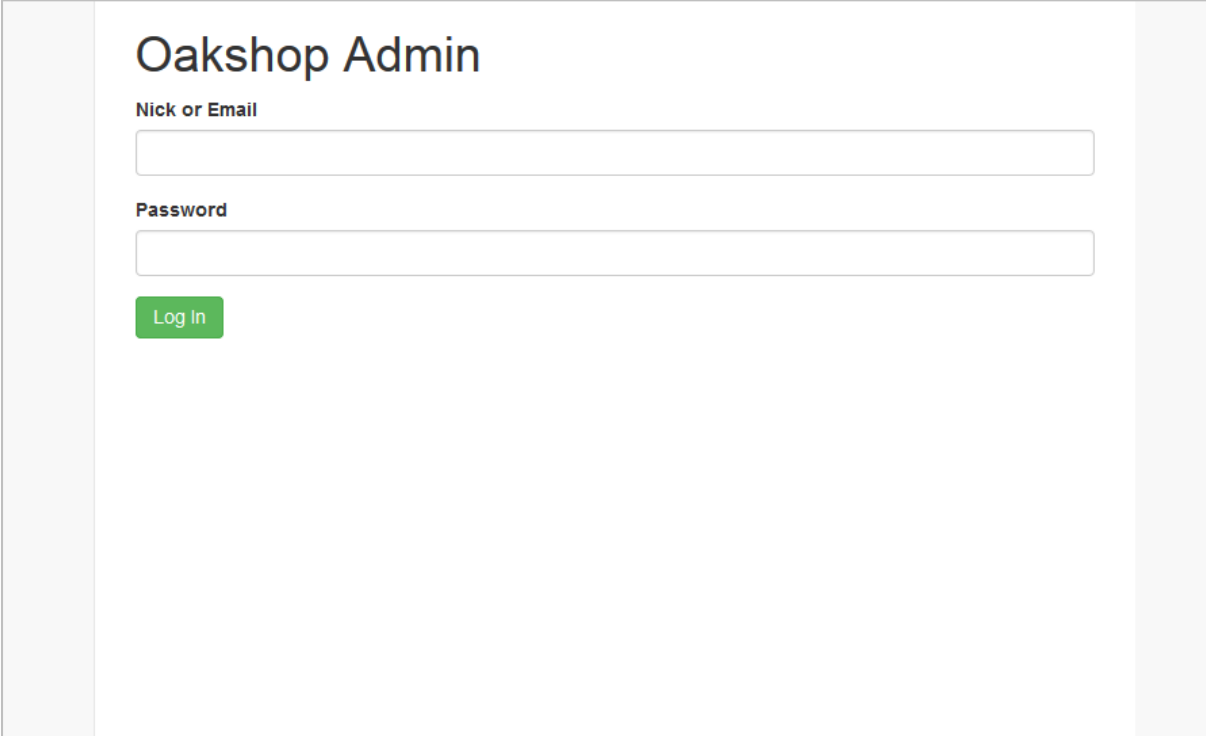
© 2013-2014 Oak Search Team @ PJWSTK

Analogical form is provided for creating the admin tickets.

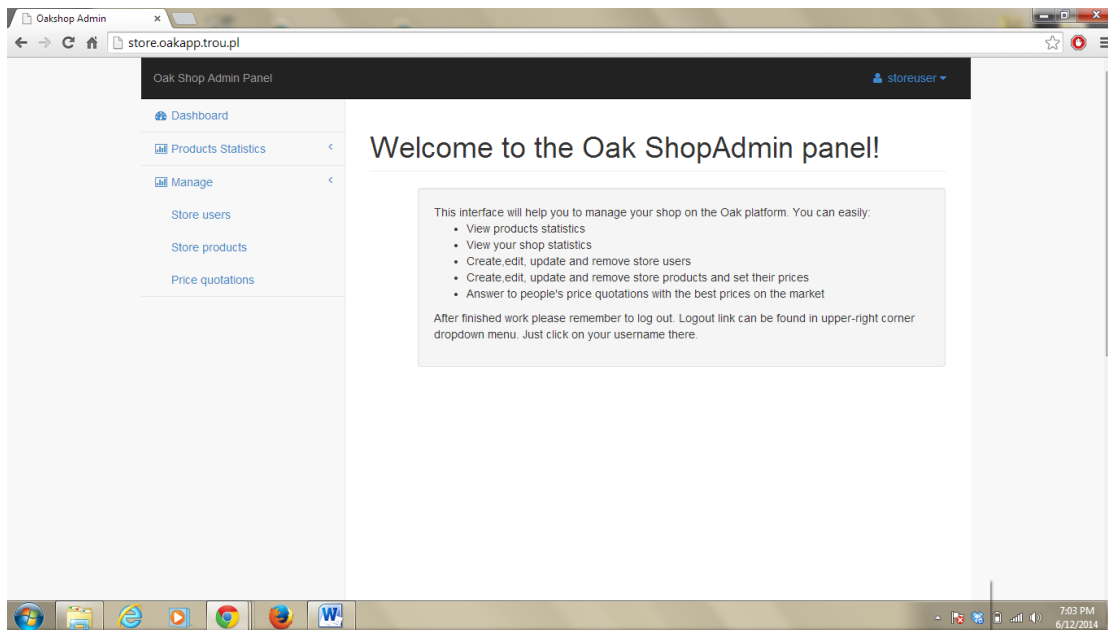
III. Store Owner/Admin

When a user contacts us to become a store owner he will be granted a login and password to a shop owner interface.

The image below shows the log in page where the given credentials have to be entered.

The image shows a login page for 'Oakshop Admin'. The page has a light gray background with a white central area. At the top, the text 'Oakshop Admin' is displayed in a large, dark font. Below this, there are two input fields. The first is labeled 'Nick or Email' and the second is labeled 'Password'. Both labels are in a small, dark font. Below the 'Password' field, there is a green button with the text 'Log In' in white. The entire form is enclosed in a thin gray border.

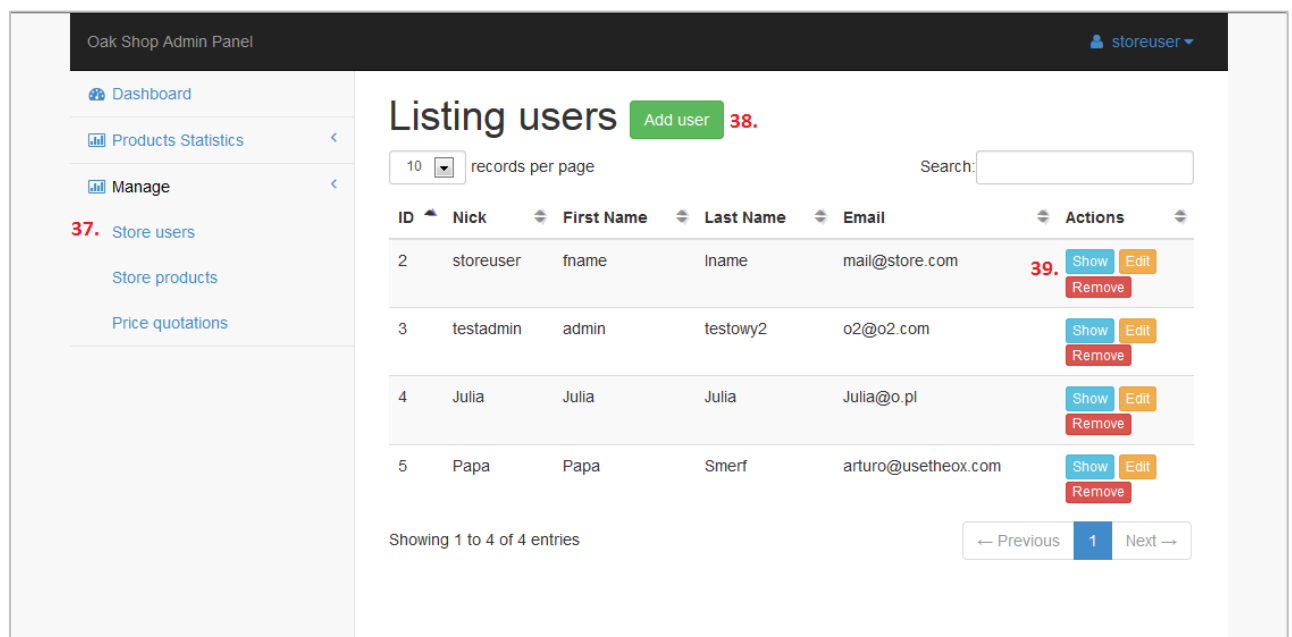
After a successful log in the user will see the main page presented below:



Managing Users

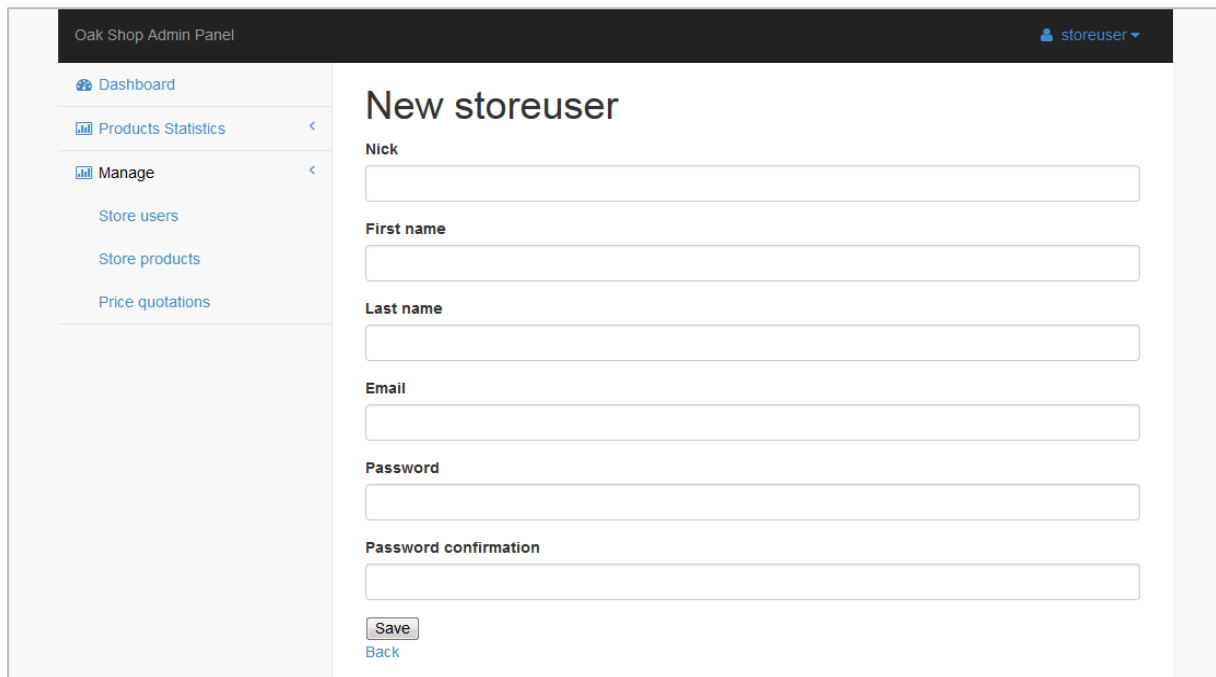
Below is the view for the store owner option “Store Users” (37). Navigation is simple through the search bar, navigation panel, and drop down for amount of records shown.

The store owner may add any accounts they want by clicking the “add user” button (38) and in addition view, edit, or remove any existing ones (39).



Adding new store user

When clicked the “Add user” button (37) a form appears requesting entering new user’s information. Submit and back buttons are located at the end in order to commit the entry or go back to the store user page.



The screenshot displays the 'New storeuser' form within the 'Oak Shop Admin Panel'. The interface features a dark header bar with the panel name and a user profile icon labeled 'storeuser'. A left-hand sidebar contains navigation links: 'Dashboard', 'Products Statistics', 'Manage' (with sub-links for 'Store users', 'Store products', and 'Price quotations'), and 'Products Statistics'. The main content area is titled 'New storeuser' and contains a form with the following fields: 'Nick', 'First name', 'Last name', 'Email', 'Password', and 'Password confirmation'. Each field is represented by a text input box. At the bottom of the form, there is a 'Save' button and a 'Back' link.

Field	Type
Nick	Text input
First name	Text input
Last name	Text input
Email	Text input
Password	Text input
Password confirmation	Text input
Save	Button
Back	Link

Managing Store's products

Next is the view of the store products as displayed below.

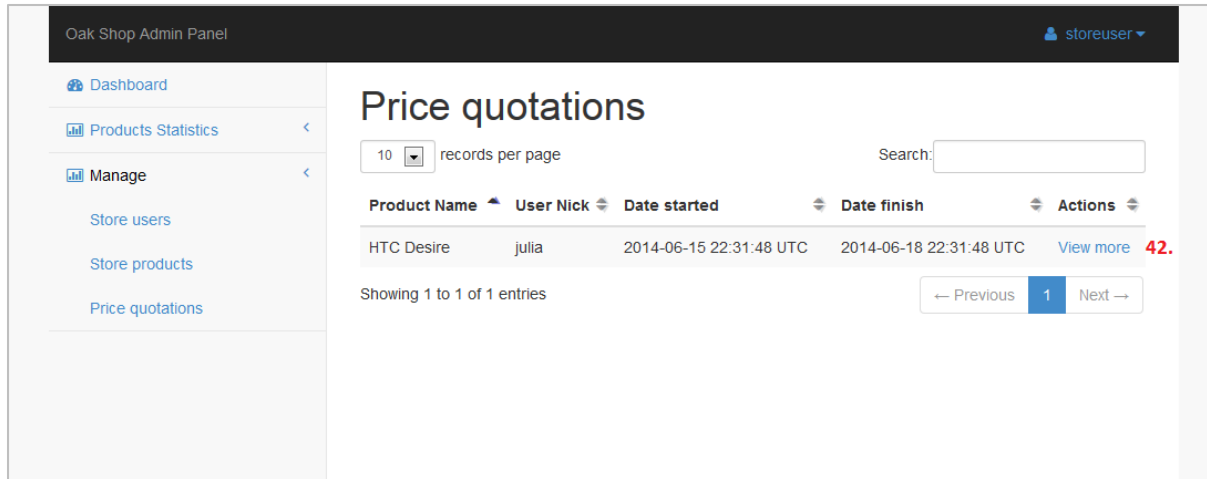
The screenshot displays the 'Oak Shop Admin Panel' interface. The top header shows the user 'storeuser'. The sidebar on the left contains navigation links: Dashboard, Products Statistics, Manage (selected), Store users, Store products, and Price quotations. The main content area is titled 'Listing products' and features a green button 'Add store product' with a red '40.' next to it. Below this is a search bar and a dropdown for '10 records per page'. The product listing is a table with columns: Product, Price, and Actions. The table contains 7 entries, each with 'Change price' and 'Remove' buttons. The first entry, 'HTC Desire', has a red '41.' next to its price of 126.0. The footer of the table shows 'Showing 1 to 7 of 7 entries' and pagination controls for 'Previous', '1' (current page), and 'Next'.

Product	Price	Actions
HTC Desire	126.0	Change price Remove
HTC One M7	629.0	Change price Remove
HTC One M8	799.0	Change price Remove
iPhone 6	120.0	Change price Remove
LG G-Flex	123.0	Change price Remove
Motorola Razr	120.0	Change price Remove
Testmemore	111.0	Change price Remove

The owner may add products (40), remove products, and change the price of any existing products (41).

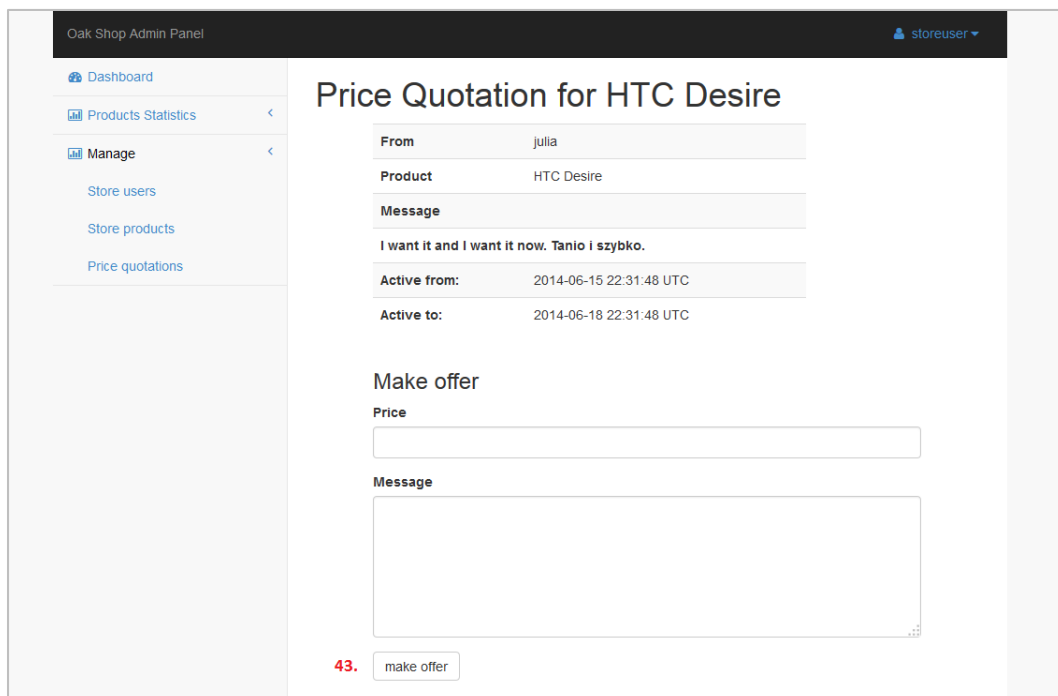
Making a price offer

During the view of the price quotations the screen will display something similar to the image below.



Any user may view the different quotations and if wishes to respond to one or see more details click “View more” (42).

When “View more” (42) is clicked a window will appear that will show the user all specifications of the offer and give him the option to reply to the request. The store user can submit his response with the click of the button “make offer” (43). The example of this is shown below.



IV. Admin

Admin Dashboard

The admin has his separate control panel, verified through username and password. The image below shows what the main page of the panel looks like.

The dashboard(44) shows any changes that have been made in the systems code. Admin can securely log out of his personal profile by clicking one of the two buttons found in two places: bottom left (45) and upper right (45).

Oak Admin Panel

Wow Welcome again, Julia!

Logout 45.

Welcome to the Oak Admin Dashboard 44.

This is the Oak Admin Dashboard. It allows to manage almost all aspects of this web application. Most important things from the recent application events are shown on this page. To proceed, please choose an option from the sidebar. **Happy working!**

Administrator Alerts

Alert	Date
new zip_code 100	2014-06-14 21:22:19 UTC
new zip_code PL1234	2014-06-14 21:24:27 UTC
new zip_code PL11234	2014-06-14 21:24:35 UTC
new zip_code PL111234	2014-06-14 21:24:42 UTC
new zip_code PL1230	2014-06-14 21:27:19 UTC
new zip_code PL123401	2014-06-14 22:32:37 UTC
new zip_code AF981	2014-06-14 22:39:51 UTC
new zip_code PL555	2014-06-14 22:41:16 UTC
new zip_code PL05800	2014-06-15 18:02:17 UTC
new zip_code 5800	2014-06-11 13:00:39 UTC
new zip_code 5111	2014-06-11 14:29:52 UTC
new zip_code 22269	2014-06-11 19:00:46 UTC

Statistics

Under the statistics section (46) the admin can add, view, edit, and remove all of the options under the category. The list entails: top products, top reviews, top users, and live users.

Manage Options

Under manage options (47) the admin may alter, add, or remove any products (48), users (49) and or reviews (50). Features (51) and zip codes (52) can only be removed or edited. The following four screen shots show each of these categories with the options they provide.

Products' Management

Oak Admin Panel

Listing products

New Product

10 records per page

Search:

#ID	Name	Created by	Created date	Modified by	Modified date	Actions
42	LG G-Flex	admin	2014-06-02 14:01:34 UTC		2014-06-05 10:56:58 UTC	Gallery Show Edit Destroy
61	LG G2	admin	2014-06-04 20:54:00 UTC		2014-06-05 10:57:14 UTC	Gallery Show Edit Destroy
62	LG G3	admin	2014-06-04 20:54:50 UTC		2014-06-04 21:32:25 UTC	Gallery Show Edit Destroy
63	Samsung Galaxy Note 3	admin	2014-06-05 10:27:32 UTC		2014-06-05 10:27:32 UTC	Gallery Show Edit Destroy
64	Samsung Galaxy Note 2	admin	2014-06-05 10:27:59 UTC		2014-06-05 10:27:59 UTC	Gallery Show Edit Destroy
65	HTC One M7	admin	2014-06-05 10:28:19 UTC		2014-06-05 10:28:19 UTC	Gallery Show Edit Destroy
66	HTC One M8	admin	2014-06-05 10:28:44 UTC		2014-06-05 10:28:44 UTC	Gallery Show Edit Destroy
67	Motorola Moto G	admin	2014-06-05 10:29:34 UTC		2014-06-05 10:29:34 UTC	Gallery Show Edit Destroy
68	HTC Desire	admin	2014-06-05 10:50:37 UTC	admin	2014-06-08 16:27:55 UTC	Gallery Show Edit Destroy
69	Motorola Razr	admin	2014-06-05 10:51:30 UTC	admin	2014-06-08 20:49:22 UTC	Gallery Show Edit Destroy

Showing 1 to 10 of 15 entries

Previous 1 2 Next

Features' Management

Oak Admin Panel

Listing features

New Feature

10 records per page

Search:

#ID	Type	Value	Created by	Created date	Modified by	Modified date	Actions
2	Screen Type	Touchscreen		2014-06-04 18:30:12 UTC	1	2014-06-14 21:52:31 UTC	Edit Destroy
3	Screen Technology	OLED		2014-06-04 18:30:24 UTC		2014-06-04 18:30:24 UTC	Edit Destroy
4	Screen Technology	LED		2014-06-04 18:30:34 UTC		2014-06-04 18:30:34 UTC	Edit Destroy
5	Keyboard	Physical		2014-06-04 18:30:58 UTC		2014-06-04 18:30:58 UTC	Edit Destroy
6	Manufacturer	Samsung		2014-06-05 10:26:34 UTC		2014-06-05 10:26:34 UTC	Edit Destroy
7	Manufacturer	LG		2014-06-05 10:26:42 UTC		2014-06-05 10:26:42 UTC	Edit Destroy
8	Manufacturer	HTC		2014-06-05 10:26:51 UTC		2014-06-05 10:26:51 UTC	Edit Destroy
9	Manufacturer	Motorola		2014-06-05 10:29:14 UTC		2014-06-05 10:29:14 UTC	Edit

Users' Management

Oak Admin Panel

julia

Dashboard
Administrators
Statistics
Top products
Top stores
Top users
Manage
Products
Features
Users
Reviews
ZIP codes
Mange Stores
Stores
Store users
Tickets
Logout

Listing users

10 records per page

Search:

ID	Nick	First Name	Last Name	Email	Actions
1007	test	Name	Last	test@test.com	Show Edit Remove
1021	test123	Name	Morename	mi@mi.pl	Show Edit Remove
1025	aplonski	Arthur	Pionski	arthur@usetheox.com	Show Edit Remove
1031	artur	Peter	Piper	ap@ap.com	Show Edit Remove
1034	alamakota	ala	Makota	ala@kota.ma	Show Edit Remove
1035	test12300	uno	dos	test@o.com	Show Edit Remove
1036	person	Peter	Piper	rock@solid.ap	Show Edit Remove
1037	StefanKa	Stefan	Krol	st3finkri@gmail.com	Show Edit Remove
1038	michal123	Michal	Kulesza	michal@michal.com	Show Edit Remove
1039	michal000	michal	michal	mm@o2.com	Show Edit

Reviews' Management

Oak Admin Panel

julia

Dashboard
Administrators
Statistics
Top products
Top stores
Top users
Manage
Products
Features
Users
Reviews
ZIP codes
Mange Stores
Stores
Store users
Tickets
Logout

Listing reviews

10 records per page

Search:

ID	Product	Author	Rating	Date	Actions
4	HTC Desire	test	2	2014-06-05 10:50:37 UTC	Show Edit Destroy
5	LG G2	test	3	2014-06-04 20:54:00 UTC	Show Edit Destroy
6	HTC Desire	test	1	2014-06-05 10:50:37 UTC	Show Edit Destroy
7	HTC Desire	aplonski	6	2014-06-05 10:50:37 UTC	Show Edit Destroy
8	HTC Desire	test	5	2014-06-05 10:50:37 UTC	Show Edit Destroy
9	HTC Desire	test	6	2014-06-05 10:50:37 UTC	Show Edit Destroy
10	HTC One M8	aplonski	5	2014-06-05 10:28:44 UTC	Show Edit Destroy
11	iPhone 6	aplonski	5	2014-06-11 03:54:29 UTC	Show Edit Destroy
12	LG G-Flex	aplonski	5	2014-06-02 14:01:34 UTC	Show Edit Destroy
13	Samsung Galaxy Note 2	aplonski	5	2014-06-05 10:27:59 UTC	Show Edit Destroy

Managing Stores

The next section for the admin to review is the manage store category. The admin may add, edit, or remove any stores (53) or store users (54) in this section.

The screenshot shows the Oak Admin Panel dashboard. The left sidebar contains navigation links: Dashboard, Administrators, Statistics (with sub-links for Top products, Top stores, and Top users), Manage (with sub-links for Products, Features, Users, Reviews, and ZIP codes), **Manage Stores** (highlighted), Tickets, and Logout. The 'Manage Stores' section shows 53 stores and 54 store users. The main content area displays a welcome message and a table of Administrator Alerts.

Alert	Date
new zip_code 100	2014-06-14 21:22:19 UTC
new zip_code PL1234	2014-06-14 21:24:27 UTC
new zip_code PL11234	2014-06-14 21:24:35 UTC
new zip_code PL111234	2014-06-14 21:24:42 UTC
new zip_code PL1230	2014-06-14 21:27:19 UTC
new zip_code PL123401	2014-06-14 22:32:37 UTC
new zip_code AF981	2014-06-14 22:39:51 UTC
new zip_code PL555	2014-06-14 22:41:16 UTC
new zip_code PL05800	2014-06-15 18:02:17 UTC
new zip_code 5800	2014-06-11 13:00:39 UTC
new zip_code 5111	2014-06-11 14:29:52 UTC
new zip_code 22269	2014-06-11 19:00:46 UTC
new zip_code 11231	2014-06-12 22:51:12 UTC
new zip_code 1250	2014-06-13 04:55:12 UTC

The following two images show these possibilities.

The screenshot shows the 'Listing stores' page in the Oak Admin Panel. It includes a 'create store' button, a search bar, and a table of store listings. The table has columns for ID, Name, Website, Zip Code, Created date, Modified Date, and Actions. The Actions column contains 'Show', 'Edit', and 'Remove' buttons for each store entry.

ID	Name	Website	Zip Code	Created date	Modified Date	Actions
103	Sample store	http://sample.web	PL1230	2014-06-08 21:13:33 UTC	2014-06-14 19:27:07 UTC	Show Edit Remove
104	New Store	http://sample.more	05100	2014-06-11 04:56:10 UTC	2014-06-11 04:56:10 UTC	Show Edit Remove
105	test	23	11231	2014-06-13 02:51:02 UTC	2014-06-13 02:51:02 UTC	Show Edit Remove
108	Test After Countries	http://teststore.com	05123	2014-06-14 19:02:24 UTC	2014-06-14 19:02:55 UTC	Show Edit Remove
109	New shop test	haha	GB	2014-06-14 19:03:39 UTC	2014-06-14 19:03:39 UTC	Show Edit Remove
110	haha	hnaha	US	2014-06-14 19:04:15 UTC	2014-06-14 19:04:15 UTC	Show Edit Remove
111	gaga	gaga	1234	2014-06-14 19:05:10 UTC	2014-06-14 19:05:27 UTC	Show Edit Remove
112	Morete	tata	US5125	2014-06-14 19:08:10 UTC	2014-06-14 19:08:10 UTC	Show Edit Remove

Showing 1 to 8 of 8 entries

← Previous 1 Next →

Oak Admin Panel

Dashboard

Administrators

Statistics

Top products

Top stores

Top users

Manage

Products

Features

Users

Reviews

ZIP codes

Mange Stores

Stores

Store users

Tickets

Logout

julia

Listing store users

create store user

10 records per page

Search:

ID	Nick	First Name	Last name	Email	Actions
2	storeuser	fname	lname	mail@store.com	<div>Show</div> <div>Edit</div> <div>Remove</div>
3	testadmin	admin	testowy2	o2@o2.com	<div>Show</div> <div>Edit</div> <div>Remove</div>
4	Julia	Julia	Julia	Julia@o.pl	<div>Show</div> <div>Edit</div> <div>Remove</div>
5	Papa	Papa	Smerf	arturo@usetheox.com	<div>Show</div> <div>Edit</div> <div>Remove</div>
6	aplonski123	person	aa	aa@o2.pl	<div>Show</div> <div>Edit</div> <div>Remove</div>
7	admin	admin	admin	admin@admin.com	<div>Show</div> <div>Edit</div> <div>Remove</div>
8	Fanatic	optic	fiber	Fanatic@oak.com	<div>Show</div> <div>Edit</div> <div>Remove</div>

Showing 1 to 7 of 7 entries

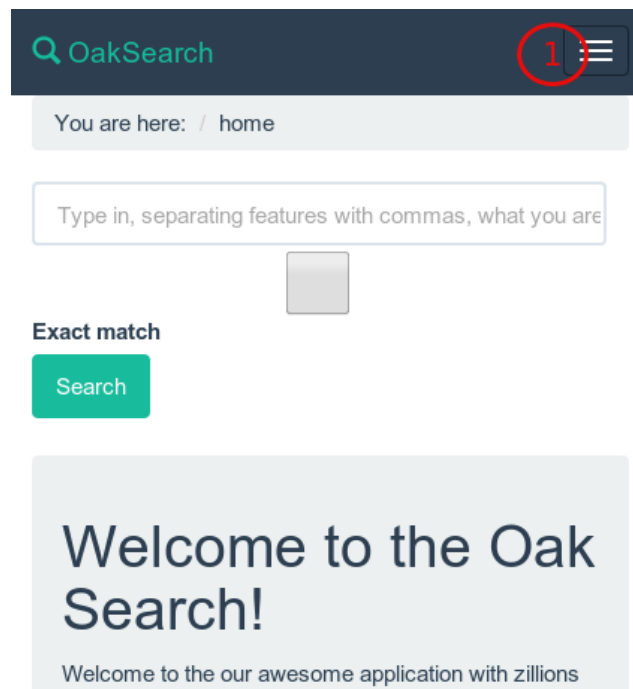
← Previous

1

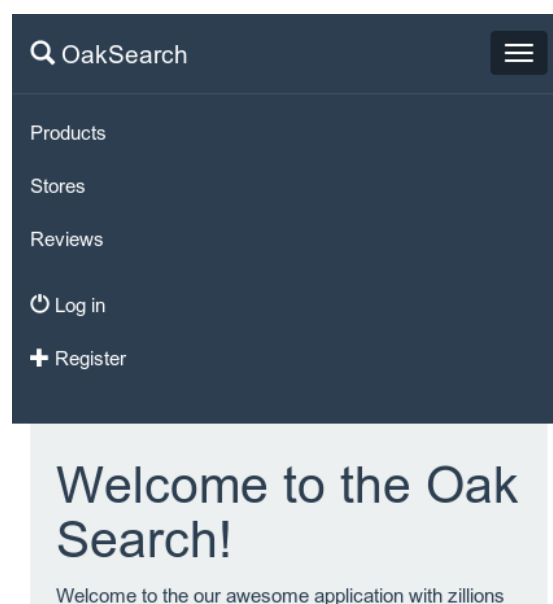
Next →

V. Mobile version

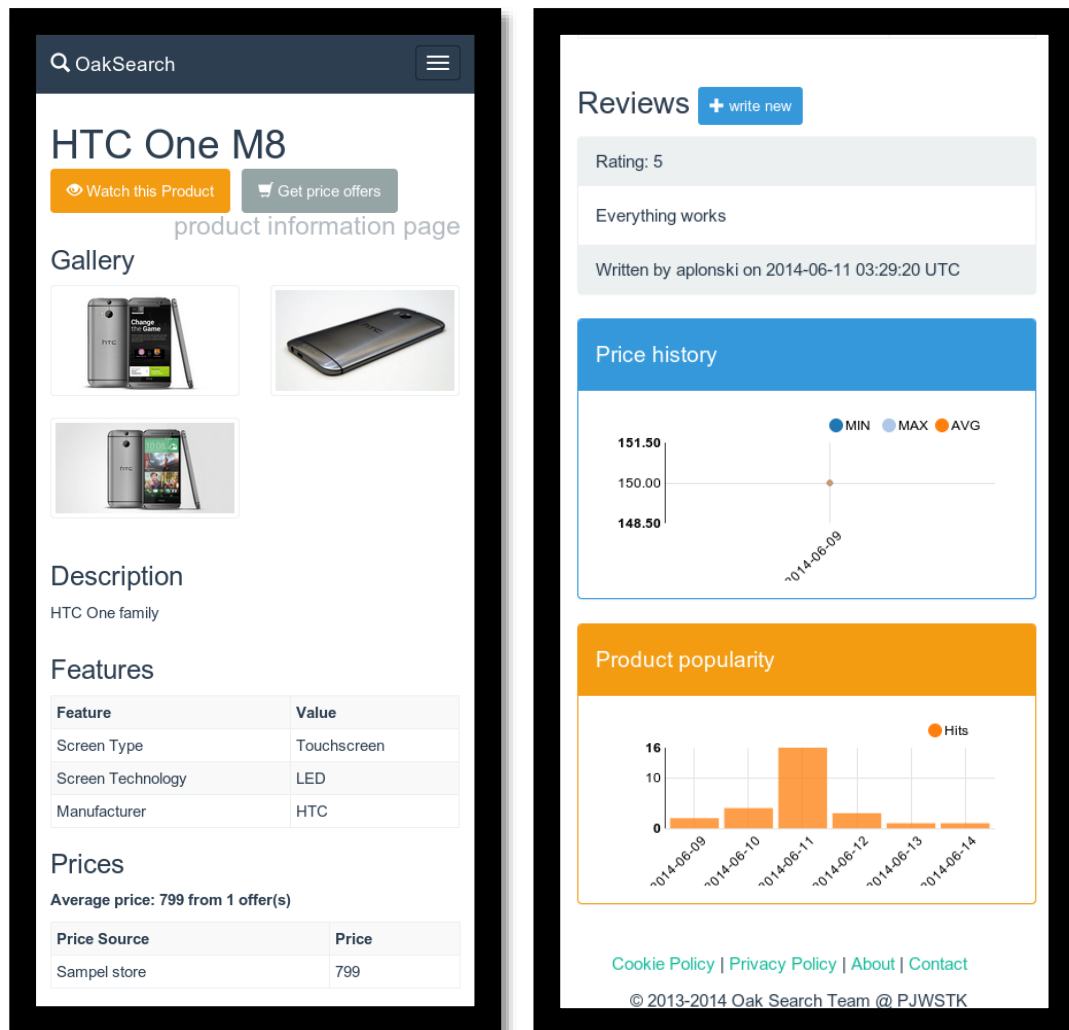
Thanks to the usage of Twitter Bootstrap fronted framework our application is responsive and adapts to smaller screens and lower resolutions automatically.



As it can be observed, the menu bar is collapsed into small icon indicating menu, that after activation (click or touch on the screen of the smartphone) is revealed in form of dropdown menu:



Also other pages, like product page are adapted automatically:



That makes the web application available for users ‘on the go’ that are for example checking prices of products during shopping in stationary shops.

Installation Guide

This installation guide expects that application will be installed on 2 machines – one for databases and one for application servers and applications. It is assumed that database server runs under control of Linux CentOS 6.5 x64 version and Oracle database 11g Express Edition is installed and configured, and application server runs under control of Linux Debian 7 x64.

Application server configuration (Debian)

Application server needs installation and configuration of following services and application:

- Oracle Java 7, required by DBSync and PriceWatcher applications
- Apache HTTP server with Ruby on Rails support, required by OakFront, OakAdmin and OakShop modules

Installation of Oracle Java 7

Oracle Java 7 on Debian 7 operating system can be installed using repository provided by webupd8.org¹⁶:

```
# su -
# echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu precise
main" | tee /etc/apt/sources.list.d/webupd8team-java.list
# echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu
precise main" | tee -a /etc/apt/sources.list.d/webupd8team-java.list
# apt-key adv --keyserver keyserver.ubuntu.com --recv-keys EEA14886
# apt-get update
# apt-get install oracle-java7-installer
```

Additionally, Java environment variables should be also set, by installing special package:

```
# apt-get install oracle-java7-set-default
```

Installation of Apache HTTP server with Ruby on Rails support

First, Ruby with Ruby on Rails framework should be installed using following command:

```
# \curl -L https://get.rvm.io | bash -s stable -rails
```

¹⁶ Source: <http://www.webupd8.org/2012/06/how-to-install-oracle-java-7-in-debian.html>

Next, RVM script should be executed to set environment variables:

```
# source /usr/local/rvm/scripts/rvm
```

Next, Ruby 2.1.1 should be installed:

```
# rvm install ruby-2.1.1
```

and it should be activated:

```
# rvm use ruby-2.1.1
```

Now Apache HTTP server with mod_proxy should be installed using following command:

```
# apt-get install build-essential libapache2-mod-proxy apache2
```

Next, apache VirtualHosts should be configured. Create file /etc/apache2/sites-enabled/oak with following contents:

```
<VirtualHost *:80>
    ServerName oakappdomain.com
    ProxyPreserveHost Off
<Location />
    ProxyPass http://localhost:3000/
    ProxyPassReverse http://localhost:3000/
</Location>
</VirtualHost>

<VirtualHost *:80>
    ServerName admin.oakappdomain.com
    ProxyPreserveHost Off
<Location />
    ProxyPass http://localhost:4000/
    ProxyPassReverse http://localhost:4000/
</Location>
</VirtualHost>

<VirtualHost *:80>
    ServerName store.oakappdomain.com
    ProxyPreserveHost Off
<Location />
    ProxyPass http://localhost:5000/
```

```
ProxyPassReverse http://localhost:5000/  
</Location>  
</VirtualHost>
```

Now, applications files should be copied to the `/opt/oakapp/` directory. In this directory following starting script, `oak-run.sh`, should be created:

```
#!/bin/sh  
source /usr/local/rvm/scripts/rvm  
cd oakfront && bundle install && rails s &  
cd ../oakadmin && bundle install && rails s -p 4000 &  
cd ../oakshopadmin && bundle install && rails s -p 5000&
```

now execution rights should be added

```
# chmod +x oak-run.sh
```

Additional databases configuration

Despite of Oracle database, Neo4J and MongoDB databases should be installed. It is assumed that Neo4J database server will also run on application server machine together with MongoDB database.

Advantage of these databases installation is fact that they don't need any prior configuration and will work just after the install.

Neo4J database installation

Neo4J can be installed using instructions from the official guide:¹⁷

```
# Import our signing key
wget -O - http://debian.neo4j.org/neotechnology.gpg.key | apt-key
add -
# Create an Apt sources.list file
echo 'deb http://debian.neo4j.org/repo stable/' >
/etc/apt/sources.list.d/neo4j.list
# Find out about the files in our repository
apt-get update
# Install Neo4j, community edition
apt-get install neo4j
# start neo4j server, available at http://localhost:7474 of the
target machine
neo4j start
```

MongoDB database installation

MongoDB can also be installed using instructions from the official guide¹⁸:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit
dist 10gen' | tee /etc/apt/sources.list.d/mongodb.list
apt-get update
apt-get install mongodb-org
```

²¹ <http://debian.neo4j.org/>

²² <http://docs.mongodb.org/manual/tutorial/install-mongodb-on-debian/>

Starting the application

Current working directory should be changed to application directory:

```
cd /opt/oakapp
```

and application should be started using created script:

```
# ./oak-run.sh
```

Now, Apache web server should be restarted to reflect configuration changes:

```
# service apache2 restart
```

Application can be accessed under following URLs:

- <http://oakappdomain.com> – for application
- <http://admin.oakappdomain.com> for administrator panel
- <http://store.oakappdomain.com> for store administration panel