TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Informatics

IDN40LT

Eduard Gorodnjov 103944 IABB

ANDROID-BASED PROTOTYPE OF E-SHOP ORDER SUBSYSTEM AND ITS RESPONSE TIME OPTIMIZATION

Bachelor's thesis

Supervisor: Tarmo Veskioja
PhD
Research Scientist

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDN40LT

Eduard Gorodnjov 103944 IABB

E-POE TELLIMUSE ALLSÜSTEEMI ANDROIDI-PÕHINE PROTOTÜÜP JA SELLE REAKTSIOONIAJA OPTIMEERIMINE

Bakalaureusetöö

Juhendaja: Tarmo Veskioja
Doktorikraad
Teadur

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Eduard Gorodnjov

15.12.2016

Abstract

The goal of this work is to create an android and web-based prototype of an order subsystem which should perform filtering of the products in constant time on average with 1 concurrent request, i.e. the response time should be independent of the number of products. The need of optimization of response time is due to the fact that for the past few decades the users' requirement for performance of performed operations has greatly increased.

To make requests return results in constant time it has been decided to pre-calculate results and cache them. The proposed method implies the construction of composite keys from all combinations of a set of input parameters and the association of composite keys with prepared results. This method has been compared with the widely used solution where DB response is associated with user's input parameters to identify its strengths and weaknesses [1]. The ways to overcome limitations of implemented method have been suggested. To ensure that the running time of requests for product filtering is constant the measurement of system performance has been performed on different volumes of test data.

The thesis includes a description of the process of prototype development. Web application implements interfaces for 2 workspaces: seller and client. Android chapter contains an explanation of how the responsive design has been implemented, how it is possible to add support for old devices and also the possibility of executing tasks in the background has been selected.

This thesis is written in English and is 50 pages long, including 6 chapters, 22 figures and 3 tables.

Annotatsioon

E-poe tellimuse allsüsteemi androidi-põhine prototüüp ja selle reaktsiooniaja optimeerimine

Käesoleva töö eesmärgiks on luua androidi- ja veebipõhine prototüüp, mis peaks teostama keskmiselt muutumatu aja jooksul toodete filtreerimist 1 paralleelse ühendusega, mis tähendab, et aeg ei tohiks toodete hulgast sõltuda. Vastamise aja kiiruse optimiseerimise vajadus on tingitud sellest, et viimase paari aastakümne jooksul on kasutajate nõudmised teostatud päringute kiiruse vastu tunduvalt suurenenud.

Selleks, et päringute vastuseid oleks tagastatud muutumatu aja jooksul oli otsustatud eelnevalt arvutada vastuseid ja hoida neid vahemälus. Pakutud meetod eeldab komposiitvõtmete konstrueerimist kogu siseparameetrite kombinatsioonide hulgast ning nende vastavusse viimist valmistatud vastustega. Seda meetodit on võrreldud laialt kasutatava lahendustega, kus andmebaasi vastus on sisendparameetritega vastavusse viidud, et tuvastada pakutud meetodi tugevused ja nõrkused [1]. Samuti on ka välja pakutud võimalusi, kuidas implementeeritud meetodi piiranguid ületada. Veendumaks, et päringute käivitamise aeg on muutumatu, olid süsteemi jõudlustestid teostatud testandmete erinevate mahtudega.

Lõputöö sisaldab prototüübi arendamise protsessi kirjeldust. Veebirakendus realiseerib kasutajaliideseid kahele töökohale: müüja ja klient. Androidi peatükk sisaldab selgitust, kuidas sai paidlik kujundus rakendatud, kuidas on võimalik lisada toetust vanadele seadmetele ja kuidas on võimalik käivitada protsessi taustal.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 50 leheküljel, 6 peatükki, 22 joonist, 3 tabelit.

List of abbreviations and terms

ANR	Application Not Responding
API	Application programming interface
CPU	Central processing unit
DB	Database
dp	Density independent pixel, that will result in the pixel with the same physical size on any android device
dpi	Dots per inch
IPC	Inter-process communication
M	Million
ms	Millisecond
MVC	Model-View-Controller design pattern
OOM	Out of memory
OS	Operating system
Page	Smallest unit of data that can be read or written
UI	User interface

Table of contents

List of figures

List of tables

1 Introduction

Over the past few decades, the online shopping has been developing quickly and there are many reasons for this. Firstly, the customer does not need to go to a physical store anymore and can perform the purchase from any location. The delivery of purchased item can also be managed remotely. Secondly, since more and more shops started to create their own representation online, the customer can easily compare the prices for products and delivery conditions from different stores and he does not need to visit every store to gather information on this. Thirdly, the price is usually lower for customers who buy online and there are many reasons for this, e.g. since purchase happen online, the payment is registered inside info system, which results in less paperwork. Because of this, the online shops have become one of the most popular types of web-based information systems.

1.1 Background and problem

One of the most important things that affect user experience with online shopping is the speed of performing operations. During shopping, the users spend most of their time on searching the product. The problem is in reducing the response time of requests that are related to seeking the product.

With the development of mobile technologies, over a half of customers use mobile phones for shopping [2]. This means that if a system does not provide either web or mobile-based interface to interact with them, then it is losing potential customers. As the mobile platform, the android OS has been chosen because in 2015 it was the most popular OS for mobile devices.

Because the creation of the whole internet shop is a hard and time-consuming task, the thesis is limited to the creation of order functional subsystem, which corresponds to the main process of whole infosystem.

1.2 Thesis objectives

The goal of this work is to create an android and web-based prototype of an order subsystem which should perform filtering of the products in constant time on average with 1 concurrent request, i.e. the response time should be independent of the number of products.

To achieve the selected goal, the following objectives has been set.

1. To analyze and design subsystem which should return results of filtering requests in constant time.
2. To implement the prototype with android and web user interfaces.
3. To measure system performance.

1.3 Methodology

The system will be developed using waterfall methodology, where development phases happen sequentially (analysis, design, implementation, etc.), i.e. the next phase does not start before previous is complete. The incremental approach has been taken which implies the creation of one subsystem at a time, which in this case is the order subsystem.

1.4 Thesis structure

In the second chapter, there is an analysis of requirements for order subsystem. The third chapter is dedicated to the design, where a solution for the defined requirements is found. Also, it contains a description of the proposed method for achieving performance requirement. The fourth chapter presents implementation details of the proposed method and the results of system performance measurements. Fifth chapter explains how the android application has been created.

2 Analysis

This chapter contains the functional and non-functional requirements for order functional subsystem as well as conceptual data model.

2.1 Use cases

Figure 1 presents use case diagram where each use case correspond to some feature of the system. Each use case consists of a sequence of steps (functional requirements), which the actor needs to make in order to reach some goal. Below is a brief description of each use case.

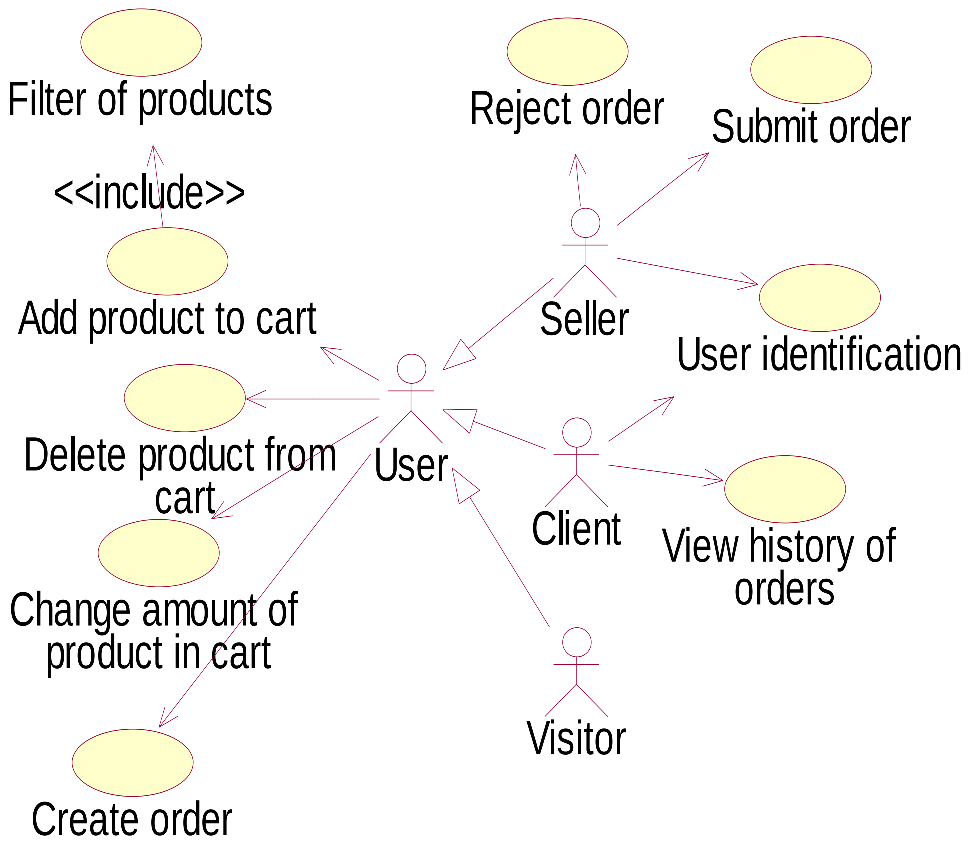


Figure 1. Use case model.

Use case: **User identification**

Actors: Client, Seller (User)

Description: User can enter the system using name and password to perform allowed operations.

Use case: **Filter of products**

Actors: Visitor, Client (User)

Description: User can choose catalog to get a list of products for that catalog. The catalog have its own set of attributes and each attribute can have many attribute values. All products in catalog need to define some value for each attribute. User can select only 1 attribute value from each attribute in order to perform products’ filtering. When user enters catalog, only those attribute values should be displayed for which at least 1 product exists. Products can be sorted by price. User should be able to move between pages. Searching for products by text is not required.

Use case: **Add product to cart**

Actors: Visitor, Client (User)

Description: User can add a product to his cart.

Use case: **Delete product from cart**

Actors: Visitor, Client (User)

Description: User can delete a product from a cart when he observes details of his cart.

Use case: **Change amount of product in cart**

Actors: Visitor, Client (User)

Description: User can set the exact amount of product while observing his cart content.

Use case: **Create order**

Actors: Visitor, Client (User)

Description: User can place his order by specifying recipient’s info. After the order has been created the system should send an email with order details.

Use case: **View history of orders**

Actors: Client

Description: Client can observe the list of previously made orders by him. Can select a particular order to view its details.

Use case: **Submit order**

Actors: Seller

Description: Seller can submit orders by changing the order’s status from unpaid to submitted.

Use case: **Reject**

Actors: Seller

Description: Seller can reject orders by changing the order’s status from unpaid to rejected.

2.2 Non-functional requirements

The properties, which are expected from this system to have, are described below.

- The system should return results of filtering requests in constant time.
- There should be web and android-based user interfaces.
- The Web and Android application should support English, Estonian and Russian languages (only labels).
- Android application should have responsive design, should run on devices starting from Android OS version 2.3.3 and also shouldn’t crash on screen rotation.

2.3 Business rules

A business rule is a constraint statement that defines or constrains behaviour of some subject. In databases, business rules are implemented through the usage of triggers, constraints or stored procedures.

- Person may not have Address, First name, Last name or Phone
- Each order has 0 or 1 client
- Each product have positive non-zero price
- There can be registered only 1 person with certain nick
- Client can be Seller at the same time
- Each Person should be Client or Seller and can be both of them at the same time
- Each Recipient should be associated with 1 Order

2.4 Conceptual data model

Conceptual data model consist of a description of entities (registers), their attributes and relationships between them, which are used by order subsystem. The conceptual model is not tied to any database model, e.g. relational model.

2.4.1 Entity-relationship diagram

To visualize the conceptual model, an entity-relationship diagram has been modeled that will be used in design chapter to construct database logical diagram. Because the ERD doesn't differ much from database diagram that is presented in next chapter, it has been moved to Appendix 1.

2.4.2 Definition of entity types

Table 1 presents clearly defined entities.

Table 1. Definition of entity types.

Name of the entity type	Register of the entity type	Definition
Person_status	Classifier register	Determinates, whether person is active/blocked
Person	Person register	Physical person who is registered in system
Employee	Employee register	Hired person who participates in systems processes
Position_type	Classifier register	Represents the position type of employee in system
Client	Client register	Person who participates in order creation process
Orders	Order register	Summary information for goods that customer wishes to buy
Recipient	Order register	The person who will receive the goods
Order_status	Classifier register	Determinates, whether the order is unpaid, submitted or rejected
Order_item	Order register	Contains additional information for every item in the order
Product	Product register	Item which can be bought
Cart	Order register	Cart stores all products that were selected by user
Cart_product	Order register	Contains additional information for every item in the cart
Detail	Product register	Property of item
Stock	Stock register	Supplies of concrete item
Shop	Stock register	Represents information about one of the affiliates
Shipment_type	Stock register	Defines time boundaries required shipping product
attribute	Attribute register	One of product's characteristics
attribute_value	Attribute register	Value of some characteristic
catalog	Catalog register	Contains list of products that belong to the same group

3 Design

In this chapter, there will be an explanation of the technologies that were used to create the system. The previously created entity-relationship diagram was used to create logical database model that is suitable for creating a database for any relational database, i.e. implementation independent, which then can be used to create a physical design for a specific database. In addition, there is a description of the method for achieving the performance requirements.

3.1 Technology selection and systems architecture

The system should have a client-server architecture, i.e. the client should interrogate with server application and not directly with the database. As programming language for server-side application Java was chosen because it allows to write platform independent code, i.e. it will not depend on CPU architecture, OS libraries and could be run on any lower version of bytecode interpreter than the one that was targeted. Also, in java it is possible to perform remote debugging whereas in some scripting languages like PHP, the debugging is reduced to printing out variables. As a web server, Jetty was used. It was needed to select MVC framework that is used to decouple the code of data access layer from view layer through using controllers. Spring framework was chosen because the whole author's experience of writing web applications in Java was associated with this framework. Because the system deals with payments, there can be some sequence of operations that form single logical operation, which in turn requires a database that supports acid compliant transactions [3]. As such database, PostgreSQL was selected because it is free and open source. Lastly, because the functionality of order creation requires sending an email with order details, it was decided to send them asynchronously. A messaging system based on message queue was used, which is useful in situations where it is possible to postpone some task without requiring from user to wait for it to end, e.g. media file processing or email sending. The choice was between ActiveMQ and RabbitMQ, but because there is a problem with running ActiveMQ on Linux Ubuntu [4], the RabbitMQ was selected. For android, it was decided to develop application using native android’s java framework because the android’s native c++ framework doesn’t provide even half of functionality compared to the first one and all 7 API releases are improving the support for OpenGL library, which was not needed [5]. One of the non-functional requirements was for the prototype to have a web-based interface, i.e. the user should be able to interrogate with the system using a regular web browser. For this, the web application receives HTTP requests and outputs HTML files as a response. The android application receives JSON responses using restful web service. The resulting systems architecture is shown in Figure 2.

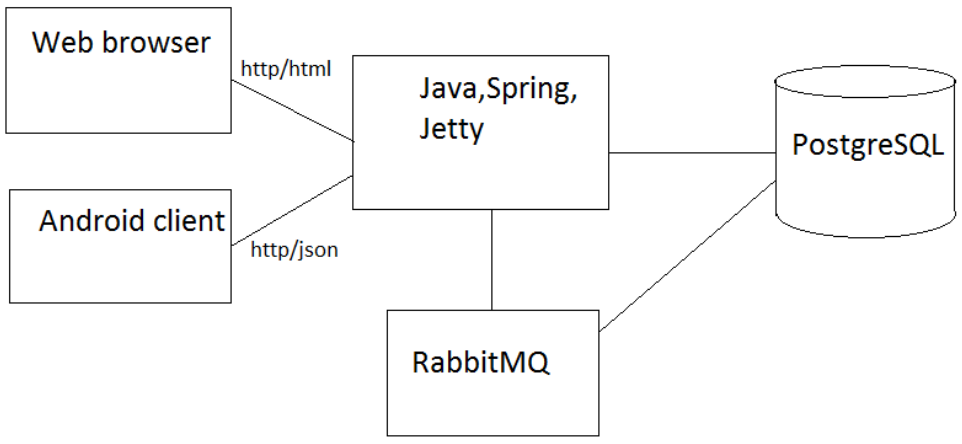


Figure 2. Systems architecture.

3.2 Description of real use cases

Real use cases describe a sequence of steps that user needs to perform in order to use some part of the subsystem. All following real use cases reference the web-based interface of the prototype. The demo of the web application is available at https://testshop11.herokuapp.com/. Credentials for seller workspace: seller01/123456. There are many client accounts that differ only by the last number in their name and all of them have the same password, e.g. test1/123456.

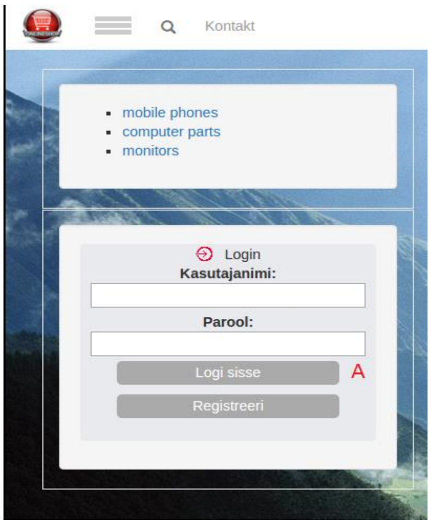


Figure 3. Login form.

Use case: **User identification**

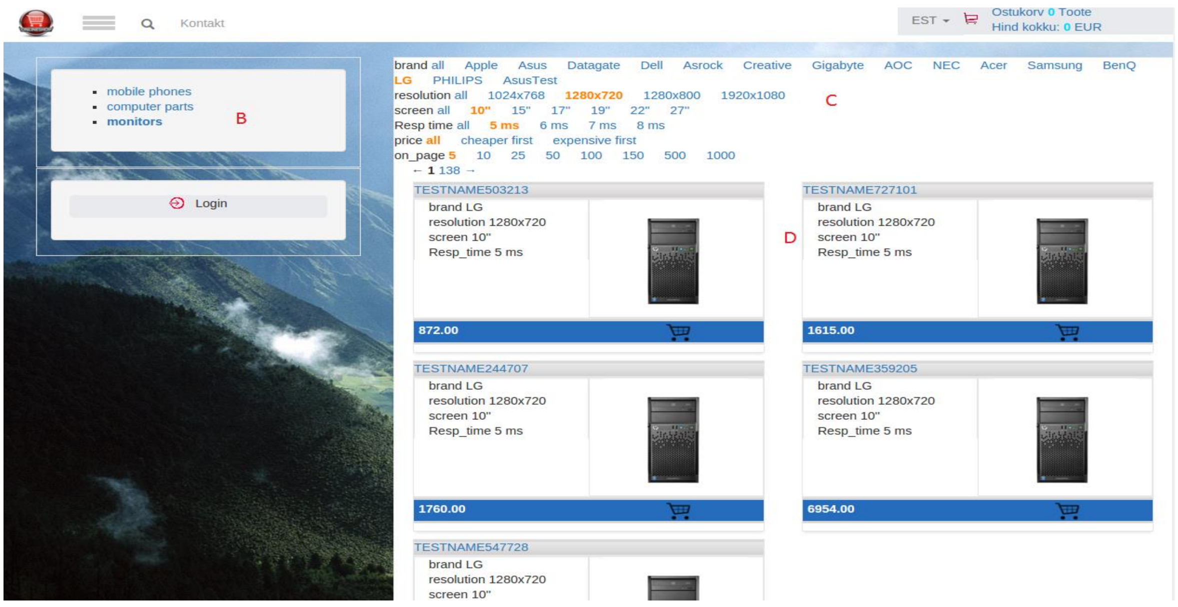
Actors: Client, Seller (User)

Scenario:

1. User enters his credentials and clicks button (A)
2. System grants access to allowed operations

Extensions:

2a. If nick or password doesn’t match the notification is displayed



brand all Apple Asus Datagate Dell Asrock Creative Gigabyte AOC NEC Acer Samsung BenQ LG PHILIPS AusTest
resolution all 1024x768 1280x720 1280x800 1920x1080
screen all 10" 15" 17" 19" 22" 27"
Resp time all 5 ms 6 ms 7 ms 8 ms
price all cheaper first expensive first
on_page 5 10 25 50 100 150 500 1000
← 1 138 →

TESTNAME503213
brand LG
resolution 1280x720
screen 10"
Resp_time 5 ms
872.00

TESTNAME727101
brand LG
resolution 1280x720
screen 10"
Resp_time 5 ms
1615.00

TESTNAME244707
brand LG
resolution 1280x720
screen 10"
Resp_time 5 ms
1760.00

TESTNAME359205
brand LG
resolution 1280x720
screen 10"
Resp_time 5 ms
6954.00

TESTNAME547728
brand LG
resolution 1280x720
screen 10"

Figure 4. Search form.

Use case: **Filter of products**

Actors: Visitor, Client, Seller (User)

Scenario:

1. User chooses one of the catalogs (B)

2. System displays some products from specified catalog along with available attribute values

3. User selects some attribute values for filtering (C)

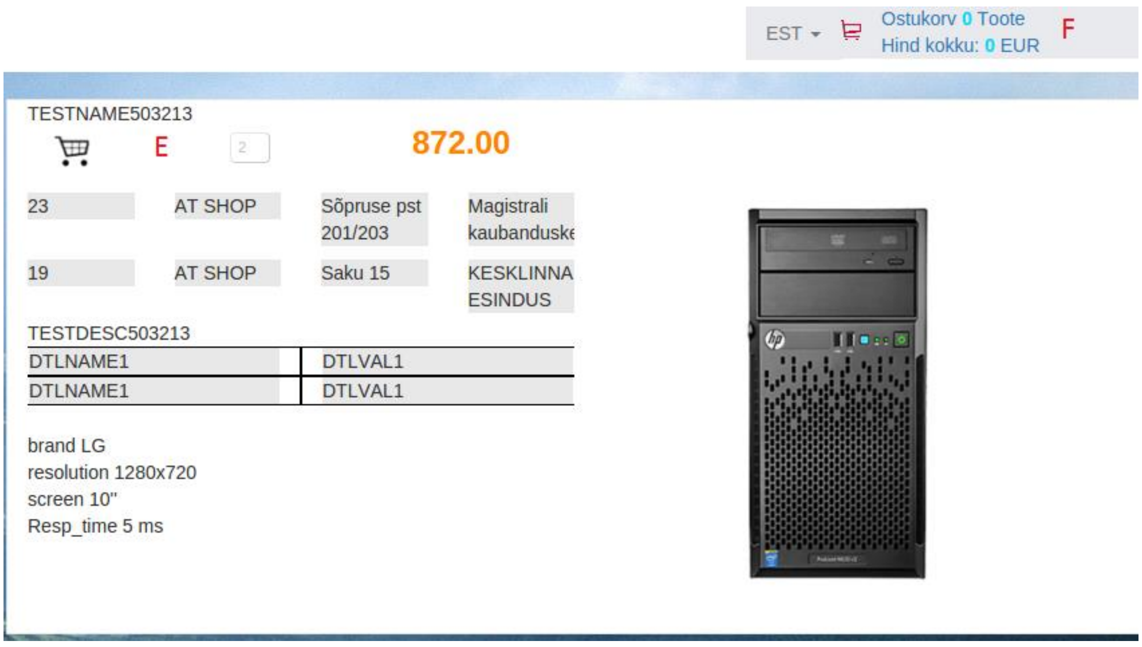
4. System displays products that match criteria (D)

5. User clicks on product to see its details

User can repeat steps 1–4 as many times as he wants

Extensions:

4a. If there is no product that meets all specified attribute values at once then product list will be empty.



TESTNAME503213

872.00

23	AT SHOP	Sõpruse pst 201/203	Magistrali kaubanduske
19	AT SHOP	Saku 15	KESKLINNA ESINDUS

TESTDESC503213

DTLNAME1	DTLVAL1
DTLNAME1	DTLVAL1

brand LG
resolution 1280x720
screen 10"
Resp_time 5 ms

Figure 5. Product’s details form.

Use case: **Add product to cart**

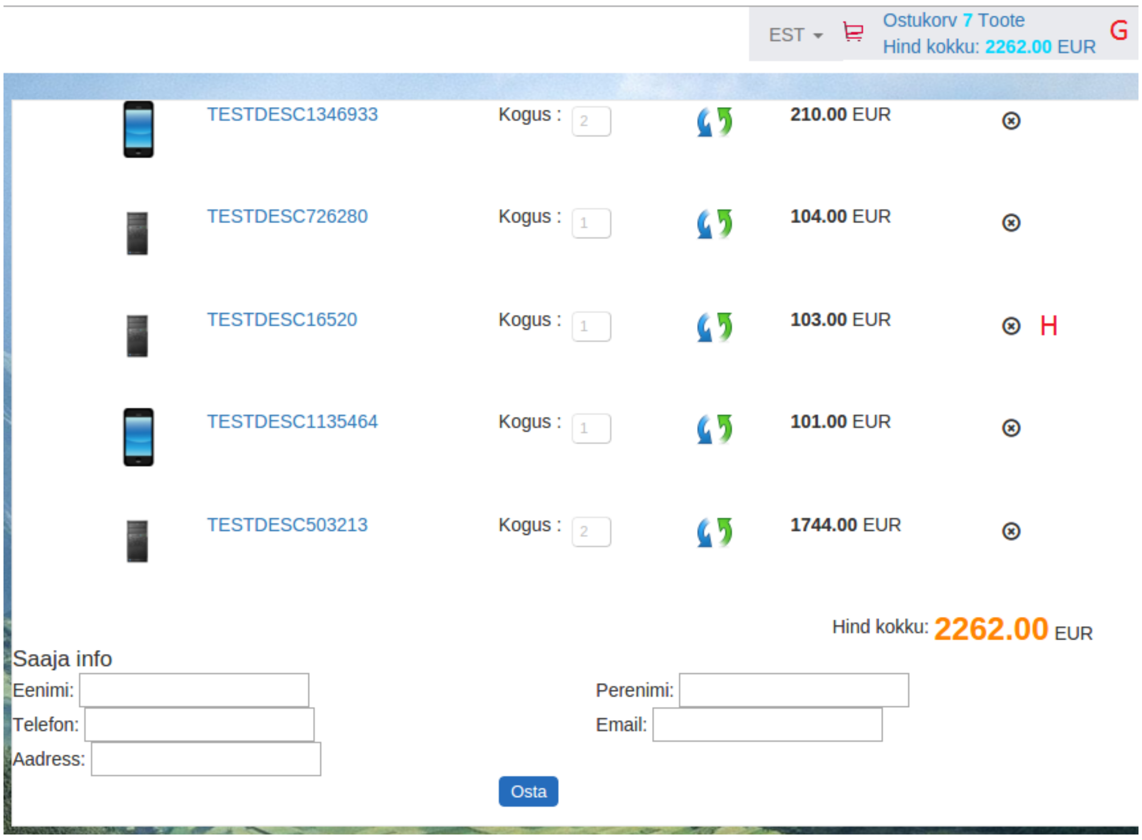
Actors: Visitor, Client, Seller (User)

Scenario:

1. User finds product (starts Filter of products use case)

2. User specifies the amount and hits a button (E)

3. System updates summary of user's cart (F)



EST ▾ Ostukorv 7 Toote G
Hind kokku: 2262.00 EUR

TESTDESC1346933	Kogus : 2	210.00 EUR	
TESTDESC726280	Kogus : 1	104.00 EUR	
TESTDESC16520	Kogus : 1	103.00 EUR	H
TESTDESC1135464	Kogus : 1	101.00 EUR	
TESTDESC503213	Kogus : 2	1744.00 EUR	

Hind kokku: 2262.00 EUR

Saaja info

Eenimi:	Perenimi:
Telefon:	Email:
Aadress:	

Figure 6. Delete from cart form.

Use case: **Delete product from cart**

Actors: Visitor, Client, Seller (User)

Preconditions: User have products in cart

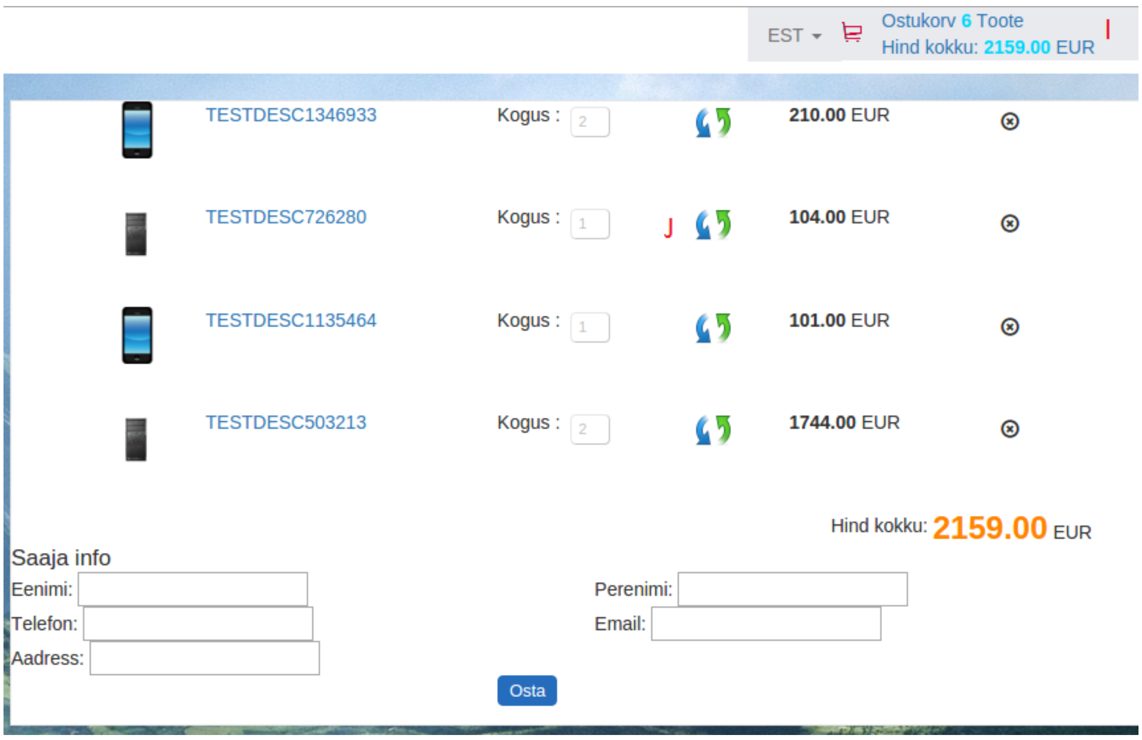
Scenario:

1. User clicks cart at the top right corner (G)

2. User selects product that he wants to delete (H)

3. System updates cart's info

User can repeat steps 1–3 as many times as he wants



TESTDESC1346933	Kogus : 2		210.00 EUR	
TESTDESC726280	Kogus : 1	J	104.00 EUR	
TESTDESC1135464	Kogus : 1		101.00 EUR	
TESTDESC503213	Kogus : 2		1744.00 EUR	

Hind kokku: 2159.00 EUR

Saaja info

Eenimi:

Perenimi:

Telefon:

Email:

Aadress:

Figure 7. Product's update form.

Use case: **Change amount of product in cart**

Actors: Visitor, Client, Seller (User)

Preconditions: User have products in cart

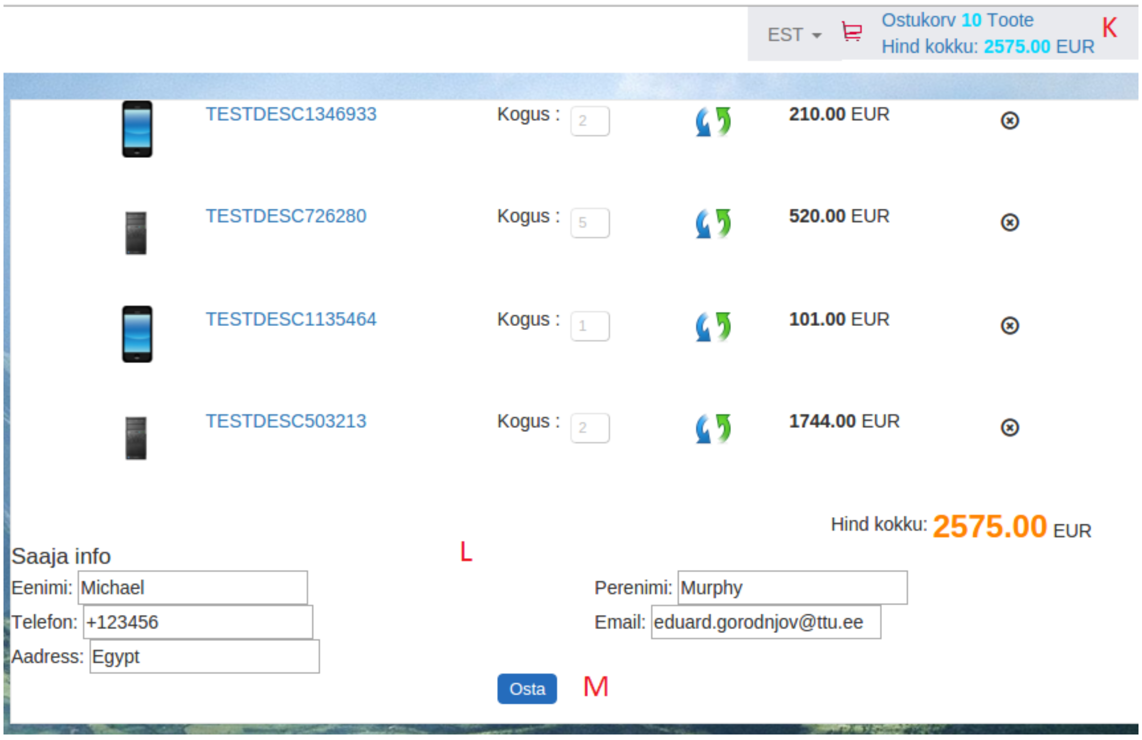
Scenario:

1. User clicks cart icon at the top right corner (I)

2. User enters required amount and presses a button (J)

3. System updates cart's info

User can repeat steps 1–3 as many times as he wants


EST ▾ Ostukorv 10 Toote
Hind kokku: 2575.00 EUR
K

TESTDESC1346933	Kogus : 2	210.00 EUR	⊗
TESTDESC726280	Kogus : 5	520.00 EUR	⊗
TESTDESC1135464	Kogus : 1	101.00 EUR	⊗
TESTDESC503213	Kogus : 2	1744.00 EUR	⊗

Hind kokku: 2575.00 EUR

Saaja info L

Eenimi: Michael
Perenimi: Murphy
Telefon: +123456
Email: eduard.gorodnjov@ttu.ee
Aadress: Egypt

Figure 8. Create order form.

Use case: **Create order**

Actors: Visitor, Client, Seller (User)

Preconditions: User have products in cart

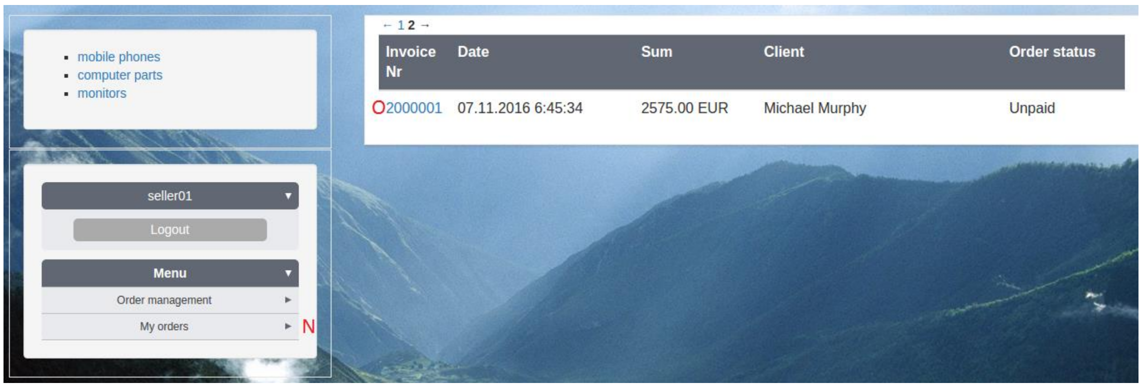
Scenario:

1. User clicks cart at the top right corner (K)

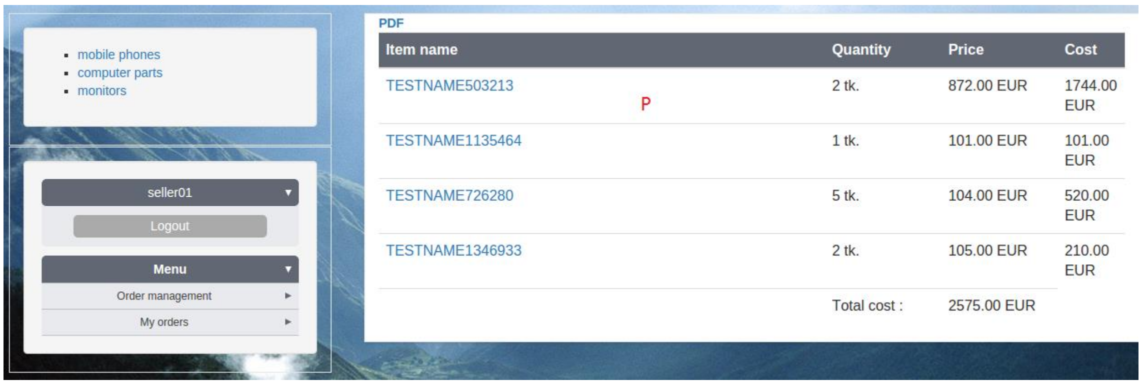
2. System displays order details and total prices

3. User overviews information, enters recipients information (L) and clicks button (M)

4. The system saves order with status “unpaid”, associates this order with user’s account (if any) and sends generated invoice to recipient’s email. Example of generated file with order details is shown in Appendix 2.



Invoice Nr	Date	Sum	Client	Order status
2000001	07.11.2016 6:45:34	2575.00 EUR	Michael Murphy	Unpaid



Item name	Quantity	Price	Cost
TESTNAME503213	2 tk.	872.00 EUR	1744.00 EUR
TESTNAME1135464	1 tk.	101.00 EUR	101.00 EUR
TESTNAME726280	5 tk.	104.00 EUR	520.00 EUR
TESTNAME1346933	2 tk.	105.00 EUR	210.00 EUR
	Total cost :	2575.00 EUR	

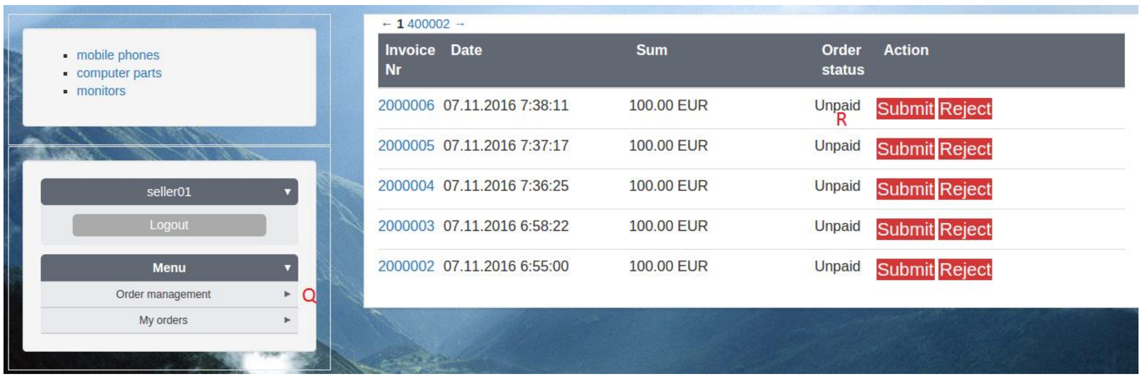
Figure 9. Order history form.

Use case: **View history of orders**

Actors: Client, Seller (User)

Scenario:

1. User asks system to display all his orders by clicking a link (N)
2. System displays all orders of that user
3. User chooses one of the orders (O)
4. System displays order details (P)



← 1 400002 →

Invoice Nr	Date	Sum	Order status	Action
2000006	07.11.2016 7:38:11	100.00 EUR	Unpaid	Submit Reject
2000005	07.11.2016 7:37:17	100.00 EUR	Unpaid	Submit Reject
2000004	07.11.2016 7:36:25	100.00 EUR	Unpaid	Submit Reject
2000003	07.11.2016 6:58:22	100.00 EUR	Unpaid	Submit Reject
2000002	07.11.2016 6:55:00	100.00 EUR	Unpaid	Submit Reject

Figure 10. Order submit form.

Use case: **Submit order**

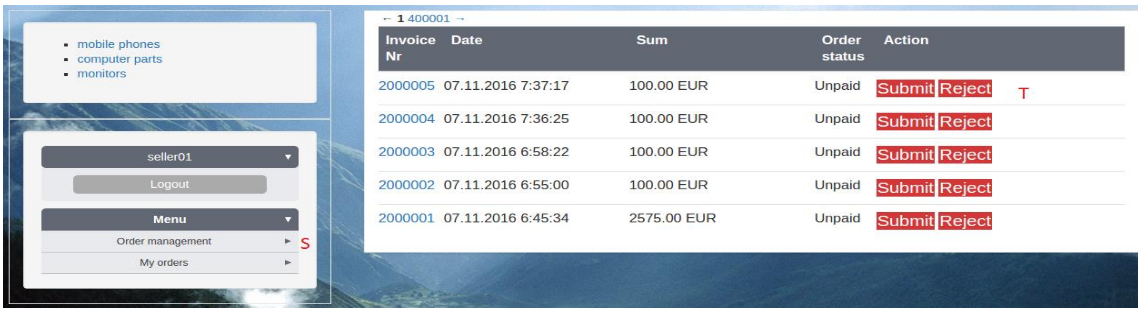
Actors: Seller

Scenario:

1. Seller asks system to display all orders with status unpaid by clicking a link (Q)

2. System displays all orders

3. Seller finds order and clicks a button (R)



← 1 400001 →

Invoice Nr	Date	Sum	Order status	Action
2000005	07.11.2016 7:37:17	100.00 EUR	Unpaid	Submit Reject
2000004	07.11.2016 7:36:25	100.00 EUR	Unpaid	Submit Reject
2000003	07.11.2016 6:58:22	100.00 EUR	Unpaid	Submit Reject
2000002	07.11.2016 6:55:00	100.00 EUR	Unpaid	Submit Reject
2000001	07.11.2016 6:45:34	2575.00 EUR	Unpaid	Submit Reject

Figure 11. Order reject form.

Use case: **Reject order**

Actors: Seller

Scenario:

1. Seller clicks a link (S)

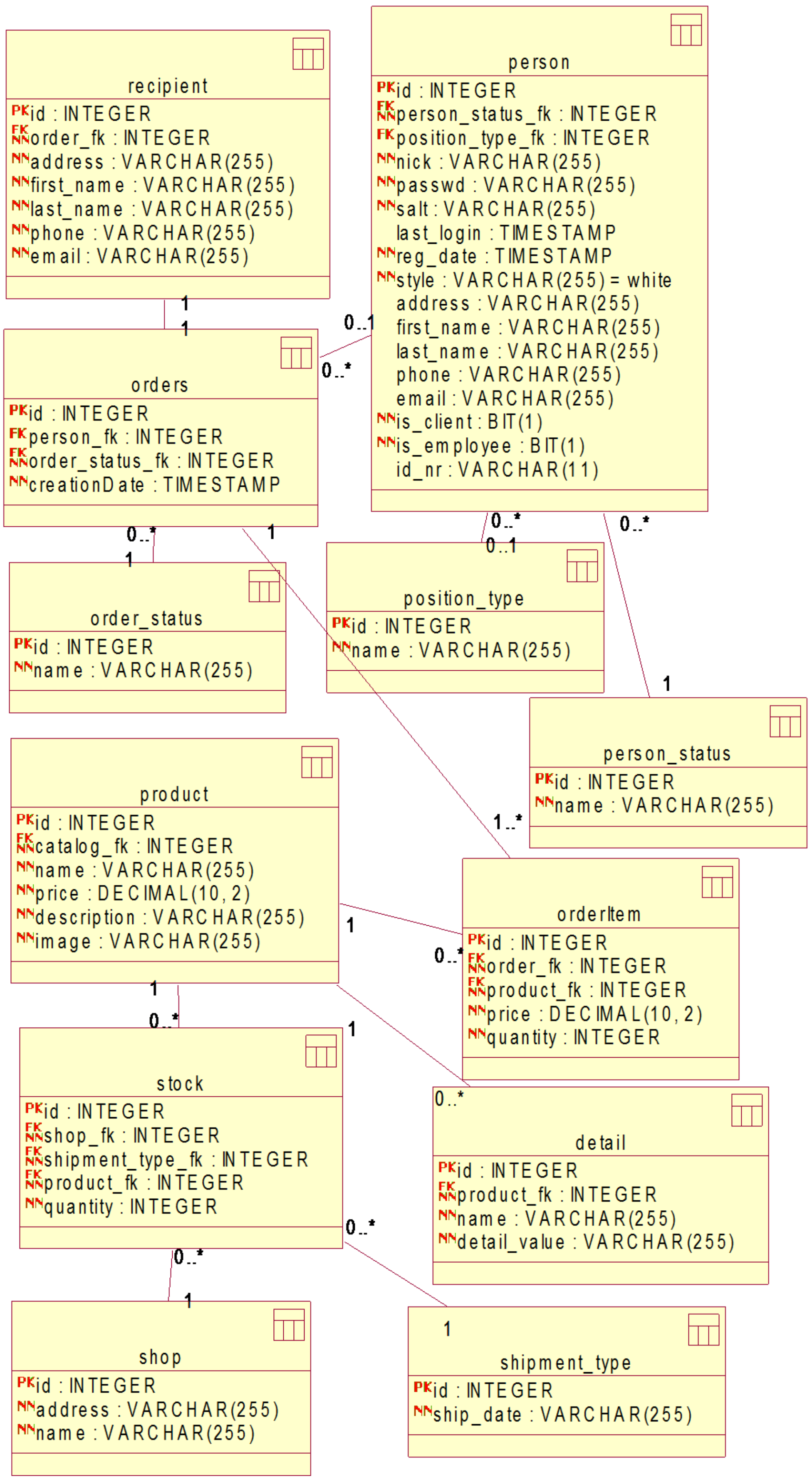
2. System displays all orders with status unpaid

3. Seller clicks a button to reject the order (T)

3.3 Database diagram

Because Rational Rose was used to create database diagram that does not have support for PostgreSQL, the presented logical database diagram in Figure 12 will be ANSI SQL 92 compliant and will not contain PostgreSQL-specific data types. To improve readability, the diagram was divided into 2 parts.

The generalization between client, employee and person entities can be implemented as single table, where client's and employee's attributes can be *null* [6]. The table for cart entity is absent because it is stored in web server's session. To represent the hierarchy of catalogs, an adjacency list model was used [7]. The advantage of this model is that there's no performance overhead when inserting a new element. The disadvantage is that in order to traverse hierarchical data, it requires to make a slow self-join for every level in a tree. In this case, it is acceptable because the catalog table is small. The data diagram is fully normalized and because of this, an EAV data model is used for catalog and product tables to represent their attributes. EAV model stores columns' names of entity table as a rows in the attribute table and columns' values as rows in the value table. The most typical implementation of EAV uses 3 tables. In this project, 4 tables were used because any attribute as well as any attribute value can be associated with many catalogs. With EAV, it is easy to manage selection, insertion and deletion of entity's attributes. The drawback is a bad search performance of entities by attributes. This is because it inevitably leads to at least one join of relations for each attribute and attribute value pair. The recipient entity will form a single table with order entity because according to business rules, each recipient should be associated with 1 order entity.


recipient
PK id : INTEGER
FK NN order_fk : INTEGER
NN address : VARCHAR(255)
NN first_name : VARCHAR(255)
NN last_name : VARCHAR(255)
NN phone : VARCHAR(255)
NN email : VARCHAR(255)
person
PK id : INTEGER
FK NN person_status_fk : INTEGER
FK position_type_fk : INTEGER
NN nick : VARCHAR(255)
NN passwd : VARCHAR(255)
NN salt : VARCHAR(255)
last_login : TIMESTAMP
NN reg_date : TIMESTAMP
NN style : VARCHAR(255) = white
address : VARCHAR(255)
first_name : VARCHAR(255)
last_name : VARCHAR(255)
phone : VARCHAR(255)
email : VARCHAR(255)
NN is_client : BIT(1)
NN is_employee : BIT(1)
id_nr : VARCHAR(11)
orders
PK id : INTEGER
FK person_fk : INTEGER
FK NN order_status_fk : INTEGER
NN creationDate : TIMESTAMP
order_status
PK id : INTEGER
NN name : VARCHAR(255)
position_type
PK id : INTEGER
NN name : VARCHAR(255)
person_status
PK id : INTEGER
NN name : VARCHAR(255)
product
PK id : INTEGER
FK NN catalog_fk : INTEGER
NN name : VARCHAR(255)
NN price : DECIMAL(10, 2)
NN description : VARCHAR(255)
NN image : VARCHAR(255)
orderItem
PK id : INTEGER
FK NN order_fk : INTEGER
FK NN product_fk : INTEGER
NN price : DECIMAL(10, 2)
NN quantity : INTEGER
stock
PK id : INTEGER
FK NN shop_fk : INTEGER
FK NN shipment_type_fk : INTEGER
FK NN product_fk : INTEGER
NN quantity : INTEGER
detail
PK id : INTEGER
FK NN product_fk : INTEGER
NN name : VARCHAR(255)
NN detail_value : VARCHAR(255)
shop
PK id : INTEGER
NN address : VARCHAR(255)
NN name : VARCHAR(255)
shipment_type
PK id : INTEGER
NN ship_date : VARCHAR(255)
1
1
0..1
0..*
0..*
1
1
0..*
0..1
0..*
1
1..*
1
0..*
1
0..*
1
0..*
0..*
1
0..*
1

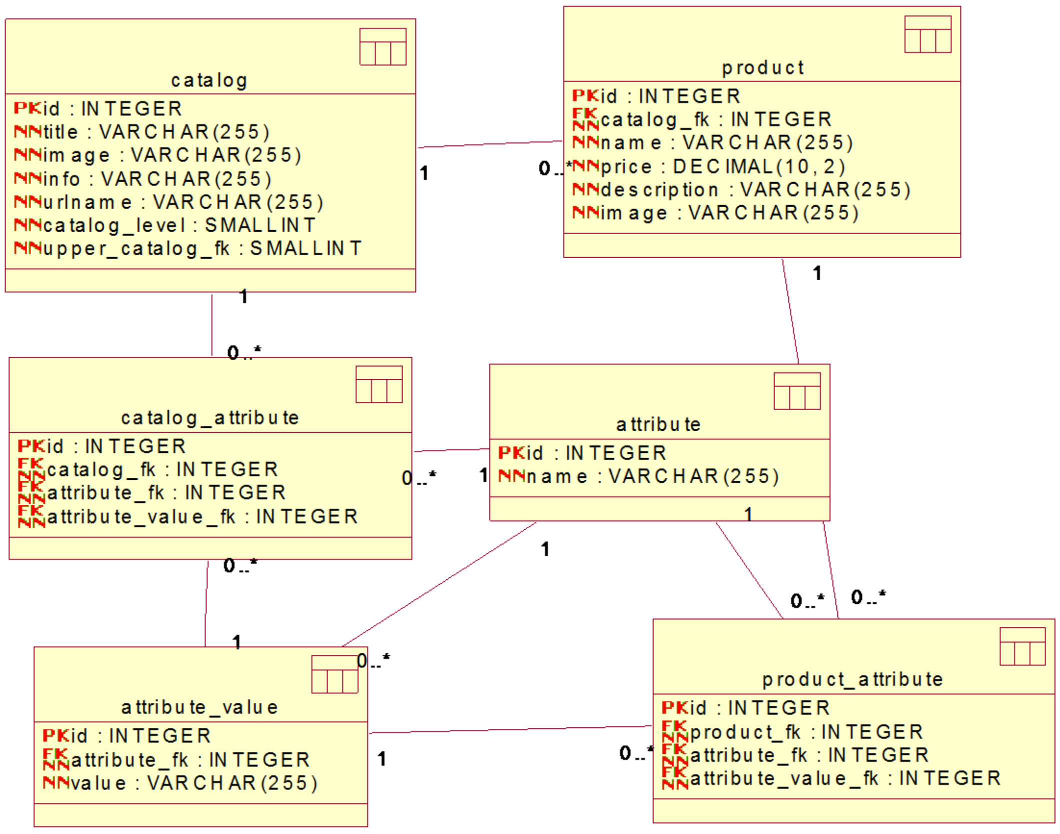


Figure 12. Logical database diagram.

3.4 Denormalization

Denormalization is a technique that implies adding redundant data to the DB and usually is used to increase speed of reads in exchange for reduced write speed. The time of write queries increases because in order to ensure data consistency it is needed to update redundant data when base data have changed [8].

The first denormalized table is catalog_attribute that can be seen in Figure 13, where the *refCount* column was added in order to achieve performance requirement. The filtering requests that should return results in constant time also should return the set of available attribute values. According to one of the functional requirements, when user selects catalog, only those attribute values should be displayed for which at least 1 product exists. A naive implementation would be to scan all products for distinct attribute values that will obviously make the running time dependent on the number of products. This was solved by using refCount column, which is a counter that shows how much products reference particular attribute value. In order to make the update process of the correct row in the refCount column easier, there is a reference from product table to attribute_value table for each possible attribute. The *trigger* for maintaining this column is shown in Appendix 3 and the example of test data insertion for catalogs' attribute values is available at Appendix 4.

The second denormalized table is product table. The flat table design replaced EAV model and the main reason for this was because there is no faster way to query product and its details than keeping all of its data in a single table. One of the disadvantages is the need to store many *null* values because each product type has its own set of attributes. Another problem is the inability to put *constraints* on attribute columns to restrict its values because each product type has its own set of allowed attribute values. Also, such design requires from DB for support of *multi-column indexes* to be able to restrict the index scan by catalog_fk. If single attribute column is indexed, it leads to many unnecessary disk I/Os and additional computations. Another disadvantage is that in PostgreSQL there is a limit for the amount of columns there could be [9]. Using the concrete table inheritance design, i.e. creating a separate table for each product type, the problems with constraints and indexes can be avoided [10].

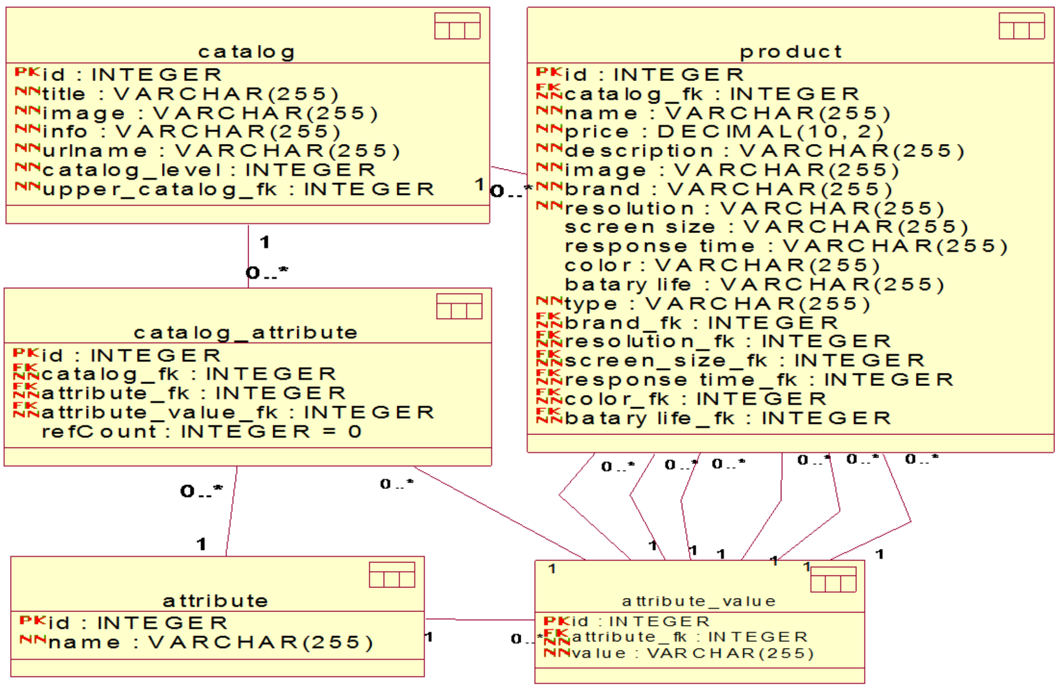

catalog
PK id : INTEGER
NN title : VARCHAR(255)
NN image : VARCHAR(255)
NN info : VARCHAR(255)
NN urlname : VARCHAR(255)
NN catalog_level : INTEGER
NN upper_catalog_fk : INTEGER
product
PK id : INTEGER
FK NN catalog_fk : INTEGER
NN name : VARCHAR(255)
NN price : DECIMAL(10, 2)
NN description : VARCHAR(255)
NN image : VARCHAR(255)
NN brand : VARCHAR(255)
NN resolution : VARCHAR(255)
screen size : VARCHAR(255)
response time : VARCHAR(255)
color : VARCHAR(255)
batary life : VARCHAR(255)
NN type : VARCHAR(255)
FK NN brand_fk : INTEGER
FK NN resolution_fk : INTEGER
FK NN screen_size_fk : INTEGER
FK NN response time_fk : INTEGER
FK NN color_fk : INTEGER
FK NN batary life_fk : INTEGER
catalog_attribute
PK id : INTEGER
FK NN catalog_fk : INTEGER
FK NN attribute_fk : INTEGER
FK NN attribute_value_fk : INTEGER
refCount : INTEGER = 0
attribute
PK id : INTEGER
NN name : VARCHAR(255)
attribute_value
PK id : INTEGER
FK NN attribute_fk : INTEGER
NN value : VARCHAR(255)
1
0..*
1
0..*
0..*
1
0..*
1
0..*
1
0..*
0..*
0..*
0..*
0..*
0..*
1
1
1
1
1
1
1

Figure 13. Diagram with denormalized tables.

3.5 Response time optimization

The response time of filtering requests depends on running time of DB queries because the server application itself is just a thin layer between client and database. One of the most effective solutions for making the running time of queries independent of a number of products is to return cached results.

According to functional requirements, each product in a catalog must define some valid attribute value for each attribute. This means that each product has some number of sets (attributes) where each of them contains a variable number of elements (attribute values) and it is possible to select only 1 element from each set. In other words, each product has a set of its own attribute values and each of them may be selected or not. The idea is in a construction of composite keys where each of them will match certain selection from a set of product's attribute values. These composite keys should then be associated with the results. In order to count all possible selections, it was needed to decide whether to count the selection of elements in different order as unique selection or not. If the order of selections is important, then the number of selections equals to all k-permutations for all k and can be expressed as $\sum_{K=0}^{N}\frac{N!}{(N-K)!}$, where N is the number of attribute values. In another case, the number of selection possibilities equals to all k-combinations (subsets) of a set of attribute values and equals to $(1+1)^N = \sum_{K=0}^{N}\binom{N}{K} = 2^N$ [11]. This equation is a special case of Binomial theorem that is used to expand the power of $(a+b)^N$ into n+1 terms [12]. In combinations, the number of keys grows exponentially, i.e. when the number of parameters doubles the number of keys squares. The second way was chosen because it results in less number of keys that need to be stored and kept up to date. Combinations have strict order of elements, so if the user inputs parameters in different order, then they should be brought to a "valid" order that can be, e.g. hardcoded.

In addition, a user should be able to sort products by price. This requires the creation of 3 sorted versions of every composite key: unsorted, in ascending and descending order. This means, that the number of keys that will be generated by single product is $3 * 2^N$, where N is the number of product's attribute values. Based on this it is assumed that the space complexity of proposed method depends on 2 variables and its growth is proportional to $O(3 * 2^N * M)$, where M is the total number of products because each product will duplicate its data into each of his own composite keys.

As the data that should be associated with composite keys, product's id was selected. If all product's details were stored, it would take up more space without giving any significant performance increase in return. This ids should be stored inside an array because the user should be able to move between pages and this, in turn, requires data structure where the element access complexity is O(1). Strictly speaking, the access to different data items cannot be constant because they should be physically located on different distance from the CPU. The information cannot travel faster than a speed of light so the time will differ. But for simplicity, the speed of element access is thought as constant. Because the difficulty of a write operation at the end of an array is usually O(n), it is not advisable to use it with frequently updated data. The set of composite keys should rely on data structure that allows performing a constant search operation.

It was decided to use PostgreSQL for persisting resulting set of key-value pairs for several reasons. First, the PostgreSQL has an advanced caching algorithm that makes sure that the most frequently used data like *pages* of tables or indexes are not evicted from the memory. To do it, it maintains a usage counter for every page and will not evict any page if its counter is higher than 0 [13]. This will make possible for this method to work effectively in an environment with limited resources. Second, because the base data and cached results are stored in single data storage, it is easy to prevent them from being desynchronized by updating them in single transaction. This type of caching is called write-through caching [14].

One of the most widely used method for improving running time of DB read queries is to associate the query results with user's input parameters and cache them [1]. By this, in the next time when user inputs same parameters, the results can be returned from the cache. If database change occur, the query results should be evicted. This standard method can be implemented using any key-value data storages, e.g. Ehcache, Guava, etc. The proposed solution that implies manual preparation of results and maintenance of them is better in several ways than the method described above. First, it allows to achieve 100% cache hit rate because all the answers are preliminary prepared at product insertion time and immediately updated after database change. In standard method, the cache miss occurs every time when an entry becomes irrelevant. Second, it allows to adjust the number of composite keys that will be generated by 1 product. This can be done through deciding the set of parameters from which the combinations will be generated. For other parameters, the results will be calculated at runtime. In other words, it is possible to decide what data should be cached that may require further processing. For example, it is possible to preliminary prepare sorted results or perform sorting dynamically at runtime. In standard method, such set contains all parameter values (even page number) and this results in a much bigger number of combinations. Third, it allows to enhance the running time of different operations on the cached data. This is possible through selecting the data structure for the cached data that should be processed dynamically, i.e. it allows to decide how to cache the data. For example, for range operator, some type of tree data structure would be more suitable or if the data are constantly changing, the usage of a linked list can be considered. Fourth, the update of results happens faster because in a case of product insertion or update, the changes are merged with 2^N results. For standard method, this means that all DB responses that might have changed need to be evicted, which will proceed with cache misses and repeated execution of slow DB queries. The advantage of the standard method is that it is easier to implement.

The drawback of proposed method is that it is not usable with a big number (30+) of input parameters because the number of all combinations grows exponentially. This will not only consume a big amount of space but also the complexity of a process of updating the composite keys, will be in the order of $O(2^N * e)$, where E is the number of elements in an array and N is the number of product's attribute values. One possible solution is to build an inverted index, which is a mapping between parameter and a collection of corresponding entities. In this case, the number of keys that should be stored is equal to the number of parameters. However, the intersection of 2 or more sets, as well as sorting, are needed to be done at runtime which will make the running time dependent on the number of entities (products) in the set. The time of update process in the worst case will be O(N*e). PostgreSQL has its own implementation of an inverted index that is called GIN and it can be used to index many data types [15]. In current project, the number of attributes should be small, so initial method was selected.

4 Proposed method

In this chapter, there is a description of how the data that should be cached are being prepared. Also, it contains the performance measurement results of improved products' filtering requests.

4.1 Implementation details

As it already has been mentioned, the proposed solution implies preliminary generation of composite keys from all k-combinations of a set of product's attribute values and association them with the arrays of products' ids. These composite keys should be cached in PostgreSQL. The construction of these keys happens at web application layer at product insertion time. There are many recursive and iterative algorithms to generate all combinations, but in this project, to do it, all numbers from 0 to $2^N - 1$ is represented in binary form, where N is the number of product's attribute values. For this example, let product have 4 attributes and attribute values that are showed in Figure 14.

```
1)'brand' => 'Samsung'
2)'size' => '17'''
3)'resolution' => '1024x768'
4)'resp time' => '5 ms'
```

Figure 14. Example of product's attribute values.

Figure 15 shows the function that is used to get the binary representation of a decimal number. It is visible, that the supported number of attribute values is limited to the number that fits into 4 bytes. At each iteration, it constructs bitmask by doing a right shift on a leftmost bit that is set to 1. This bitmask is used in bitwise AND operation to get the value of n-th bit.

```
private int[] binary_form(int number){
             int[] binaryString = new int[32];
             for(int i = 0;i<32;i++){
                    if(((0x80000000>>>i)&number)!=0){
                          binaryString[i]=1;
                    }else{
                          binaryString[i]=0;
                    }
             }
             return binaryString;
      }
```

Figure 15. Function that returns the binary representation of a decimal number.

Figure 16 shows the resulting bit strings that are build. The first subset is the empty subset that is also required, because the user may not pick any attribute values.

```
0)00000000000000000000000000000000
1)00000000000000000000000000000001
2)00000000000000000000000000000010
3)00000000000000000000000000000011
4)00000000000000000000000000000100
5)00000000000000000000000000000101
6)00000000000000000000000000000110
7)00000000000000000000000000000111
8)00000000000000000000000000001000
9)00000000000000000000000000001001
10)00000000000000000000000000001010
11)00000000000000000000000000001011
12)00000000000000000000000000001100
13)00000000000000000000000000001101
14)00000000000000000000000000001110
15)00000000000000000000000000001111
```

Figure 16. Example of generated bit strings.

There are many unnecessary bits, and they will be cut off leaving only N rightmost bits. The position of bit tells whether the attribute value at this position should be included in the subset or not. As identifiers, the names of attributes and attribute values have been used. The author is aware that it may lead to several problems, e.g. in case if there are 2 attributes in a catalog with the same name and attribute value, it will lead to incorrect results, but for human readability their names are used. Figure 17 shows the results of replacement bits with names of parameters, appendage of catalog id are following.

```
0)catalog:9
1)catalog:9:resp time:5 ms
2)catalog:9:resolution:1024x768
3)catalog:9:resolution:1024x768:resp time:5 ms
4)catalog:9:size:17''
5)catalog:9:size:17'':resp time:5 ms
6)catalog:9:size:17'':resolution:1024x768
7)catalog:9:size:17'':resolution:1024x768:resp time:5 ms
8)catalog:9:brand:Samsung
9)catalog:9:brand:Samsung:resp time:5 ms
10)catalog:9:brand:Samsung:resolution:1024x768
11)catalog:9:brand:Samsung:resolution:1024x768:resp time:5 ms
12)catalog:9:brand:Samsung:size:17''
13)catalog:9:brand:Samsung:size:17'':resp time:5 ms
14)catalog:9:brand:Samsung:size:17'':resolution:1024x768
15)catalog:9:brand:Samsung:size:17'':resolution:1024x768:resp time:5 ms
```

Figure 17. Example of constructed composite keys.

These composite keys will be used to store arrays of unordered products. There will be 2 more sorted versions added for each of them. A full list of resulting 48 keys is presented in Appendix 5.

Because there are much more attribute names and values that can be used by other products, there will be much more keys. The formula that counts the number of keys that will be generated by all products, assuming that there will be at least 1 product for each attribute value is $3 * (a_1 + 1)(a_2 + 1) \dots (a_n + 1)$. As an example, a catalog with 4 attributes that have 3, 5, 4, 13 different values have $3 * (3 + 1)(5 + 1)(4 + 1)(13 + 1) = 5040$ composite keys.

These keys are stored inside table with the schema that is shown in Figure 18. The *total_amount* column is used to avoid counting the size of the array. The hash index type is used to perform searches of keys in O(1) time. Its also possible to zip all results and store them compressed, which is a tradeoff between memory and CPU time [16].

```
CREATE TABLE cachedtb(
	id INTEGER NOT NULL DEFAULT nextval('cachedtb_id'),
	key VARCHAR(255),
	total_amount INTEGER,
	value integer[],
	CONSTRAINT cachedtb_pk PRIMARY KEY(id)
);
CREATE INDEX cachedtb_idx2 ON cachedtb USING hash (key);
```

Figure 18. Table schema for storing cached data.

The function that generates composite keys for products and stores them in DB is shown in Appendix 6. All combinations have single order of elements, so when user inputs attributes in arbitrary order, the “valid” order of parameters is selected from DB by ordering them by identifier. The correctly ordered set of parameters is used to construct valid composite key from user’s input. Figure 19 shows an example of resulting query that uses *subarray* function from *Intarray* module and selects first 5 products.

```
SELECT total_amount, unnest(subarray(value,0,5)) prodids FROM cachedtb WHERE
key='catalog:9:price:asc:brand:Samsung'
```

Figure 19. Example of querying cached data.

4.2 System performance measurement

In order to ensure that the running time of filtering requests is independent of the number of products, their performance was measured using JMeter. All requests were sent sequentially, i.e. 1 request at a time. Requests were sent to the web application that outputs rendered html. The test plan consists of 5 http requests which cover different parts of filtering functionality and 500 samples were made for each request. There are 4 sample sizes: 0,5M, 1M, 2M and 4M. Below is a list of requests.

1. Selection of first 5 products. This happens when a user selects one of the catalogs.

2. Filter of products by 1 attribute. Filtering parameter was set to Resp time=5 ms, which have only 4 distinct values.

3. Sort of all the products by price.

4. Usage of the last page number in conjunction with sorting. The number of page depends on sample size.

5. Selection of 4 filters with sorting by price and usage of latest page.

PostgreSQL was configured with 2 parameters: *shared_buffers* = 2024MB; *work_mem* = 100MB. The first parameter determinates the amount of shared_memory that PostgreSQL is used to store data like tables and indexes. Also, it is used to speed up the write operations by not immediately flushing updates to disk, but waiting until the whole input data is stored in a buffer. The second parameter is responsible for storing intermediate results of operations such as *order by* or *group by*. The work_mem is allocated separately for each request, so in a case of 100 current requests, this will require 10GB RAM. This can cause lots of disk's page in/outs that will result in a slow database. In this case, there is enough of memory and no concurrent requests was made. Hardware specifications are AMD Phenom x4 965, 8 GB RAM, HDD SATA 2 3GB/s 7200rpm, Linux Ubuntu.

Table 2 shows that the average running time of all product filtering requests doesn't not grow with the number of products because everything is read from the cache. An example of results for the sample with 4M products is shown in Appendix 7, where it is seen that the same operation can be 4x slower in the worst case. This may be explained by some freezing effects that were going in the system. The numbers in brackets shows that the space that is required to store all cached data grows linearly. They include the size of rows, indexes and toast tables where the latter is used to store values of wide fields [17]. This numbers correlate with the assumed space complexity because the number of attributes was the same for all sample sizes and only the number of products was growing.

Table 2. Independence of requests for filtering products from data volume.

Amount of products /query	First 5 prodcuts (ms)	Apply 1 filter (ms)	Sorting by price (ms)	Last page + sorting (ms)	4 filters + last page + sorting (ms)
500K(106MB)	21	24	25	26	24
1M(205MB)	22	25	26	26	23
2M(409MB)	25	24	28	28	23
4M(820MB)	24	23	27	27	20

5 Implementation of android application

During the development an android application, the author has followed these requirements: the app should have a responsive design, should support 3 languages, it should run on old devices starting from android OS version 2.3.3 and should not crash when a screen is being rotated. The following is the description of the android application creation process. Appendix 8 contains screenshots of the user interface.

5.1 Responsive design

Responsive design is such design that adds more elements on the screen when enough width is available. This element can be anything like an additional column at the left side of a screen or more elements at toolbar. In order to implement a good-looking responsive design that provides a good user experience, the layout for different screens should be designed and also the images should be handled [18].

5.1.1 Layout design

The layouts for different screen widths were created. For mobile developers, the devices are divided into 2 main groups by their width: phones and tablets. The design of layout should be based on the width of a screen because for the user it is much easier to scroll all the content that does not fit in a vertical direction. The problem comes from the fact, that in 2015, there were registered over 24000 different devices that were using android OS, with their own screen resolutions and screen sizes [19]. Each of these devices has their own dpi (ppi). DPI stands for dots per inch and is the ratio of resolution to screen size [20]. In this project, 2 layouts were adapted for tablets: order list and product list layouts. This was implemented by adding the additional pane with its own layout so that on left side of the screen there is a list of products and on the right side is a pane with product’s details. There are two rules that were followed when designing layout regardless of the width [18].

1) The layout should be stretchable, i.e. a dead space should be added that should fill all available width. The layout of most of UI widgets that were used (Toolbar, TextView, etc.) were made stretchable, i.e. their width became dependent on screen width. In android, the stretching of layout elements can be easily achieved using *match_parent* and *wrap_content* values for *layout_width* and *layout_height* parameters. Match_parent will resize the element to the parent size and wrap_content will make its size depend on child element(s). The stretching helps to avoid creation of different layout for every possible screen width and it will look visually better if all available size is occupied (even when its a dead space). However, not every layout can be stretched. An example is the layout of calculator that have only buttons and space for displaying results. Such layout does not contain stretchable dead space [21].

2) UI elements like buttons, images, checkboxes and even font of text should not depend on screen width, i.e. their size should be physically the same. In a mobile world, when the user owns a larger screen, he does not want to see big elements, but instead he expects for more elements to be visible. The screens that are intended to be read from a certain distance should have UI elements with the same physical size. For example, it is assumed that an element from a phone or tablet will be observed from a distance around 0.5 meters that requires an equal size of elements and font size and, at the same time, the same elements should be physically bigger on TV.

There are several ways how to make elements have same size. First, their sizes can be defined in dp. DP is a virtual pixel that takes physically the same size on any screen. In android, 160 dp approximately equals to 1 inch. The devices with 160 dpi are considered as a baseline, which means that on such devices 1 dp will equal to 1px. The larger the resolution of a screen and the bigger size in inches, the more dp units the device will have.

$$px = dp * \frac{dpi}{160} \tag{1}$$

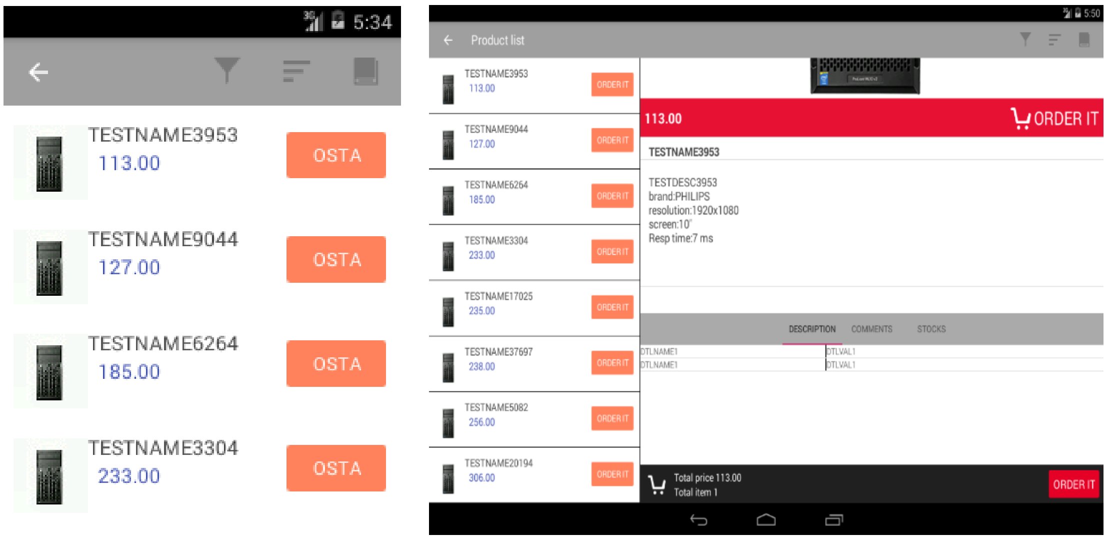
Formula (1) shows that dp can be multiplied by screen dpi in order to be translated into pixels. This results in more pixels for higher densities. The problem is that in Android the size of elements can differ even when using dp. This is because of instead of device's real dpi, a value of predefined density bucket is used in calculations [18], [22].

First reason for using density buckets is that it increases the speed of rendering because the values of a bucket are defined as integer values whereas device’s real dpi is always float. The second advantage is that developer does not need to target every possible screen density and only the range of dpis. From Table 3 it is seen that although, the 1st and 3rd device belong to different density buckets, they have same resulting physical size of an element since they are equally distant from constant that is used as dpi. Device 2 and 4 belong to the same bucket but will have different element length. Each bucket contains around 40 dp unit, e.g. mdpi contains dpis in a range from around 140 to 180, so the physical length of elements which are specified in dp can differ no more than 25%. Second way is to define element size manually through programming code and use device’s real dpi. Because a small inaccuracy in size is acceptable, elements’ sizes were defined in dp.

Table 3. The difference in physical sizes of image.

Resolution(px)	Size(inch)	Density (px/inch)	Image size(dp)	Scale ratio	Result image size(px)	Physical size(inch)
480x800	4,0	233	200x200	1,5	300x300	300/233=1,28
854x480	5,4	181	200x200	1,0	200x200	200/181=1,1
1280x720	4,7	312	200x200	2,0	400x400	400/312=1,28
240x480	3,8	141	200x200	1,0	200x200	200/141=1,4

Figure 20 shows how all mentioned above were used to design layout on the example of product list layout. It is visible that all elements fit into the smallest available screen size (320dp). There are 3 elements and 2 of them (product’s image and button) are not stretchable. The TextView at the middle that displays product’s name was made stretchable using *layout_weight* parameter. Because stretching of this TextView would look too inappropriate on big screens, there has been separate layout created. All static layout elements were defined through XML file, and dynamic elements like basket summary bar at the bottom of the page, are added through code.



(a) (b)

Figure 20. Responsive layout for smartphones (a) and tablets (b).

5.1.2 Image handling

Some UI widgets like buttons can be easily resized by adding more pixels. The images can also be scaled up, but this approach is unacceptable because it results in quality loss. There are two types of images that were handled.

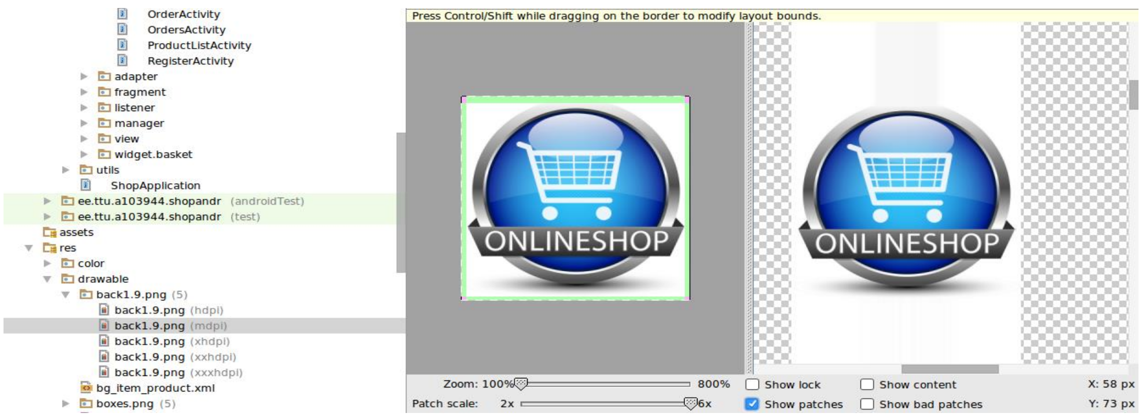
1) Images with known physical dimensions (dp). When a size of an element that contains image is known it is possible to preliminary create an image in higher resolution for different screen densities. This is the case for images that is used e.g. as background for fixed-size buttons.

In this project, there were 4 images with predefined size in use for UI. The general steps that were done in order to deal with such images are the following [23]:

1. The physical size of an image were decided (in dp).
2. Resolution of an image for highest density was calculated, using decided dp and scale factor for highest density (4.0).
3. Image of calculated pixels was produced and inserted into a folder with xxxhdpi resource qualifier.
4. Images for lower densities (mdpi, hdpi, xhdpi, xxhdpi) was produced and inserted into an appropriate folder.

The third step is the hardest because images tend to have their own aspect ratio, i.e. an image can be horizontal or vertical orientation and it cannot change its orientation without being squeezed or stretched. The fourth step is optional and is needed to not let the android scale down the image by himself, which will slow down rendering process. The revert situation where android needs to scale up the image is also possible and if the resolution is high enough the app can crash with OOM exception. It is worth to mention that the more images there are in a project, the bigger app size will be.

2) Images with unknown dimensions like background images. In this case, the resolution of image is not known in advance because image needs to fill the whole screen and resolution of screens differs. There are several ways how this problem can be solved. First, is to provide an image for the largest supported resolution [24]. This solution inevitably leads to cropping, i.e. the image will not be fully visible. This is because the image, as well as the screen, have their own aspect ratios and in order to preserve it and at the same time for the image to fill all width, it should be cropped. Second, is to use method called *9 patch*, which allows to limitlessly stretch the image. It works by defining areas that can be stretched and also the areas that should be static by adding 1px borders to the image. The drawback is that image should have a monotonous background color, so the user would not notice the stretching. The 2nd way was selected, because the logo that was used as background image on the main screen have white background. In Figure 21 it is visible that when an image is stretched in the y-direction, the logo keeps static resolution and the same thing happens when it is stretched horizontally. Of course, for the logo to have an equal physical size it is required to produce an image for different densities in a way it has been described above.


```
OrderActivity
OrdersActivity
ProductListActivity
RegisterActivity
▸ adapter
▸ fragment
▸ listener
▸ manager
▸ view
▸ widget.basket
▸ utils
  ShopApplication
▸ ee.ttu.a103944.shopandr (androidTest)
▸ ee.ttu.a103944.shopandr (test)
assets
▾ res
  ▸ color
  ▾ drawable
    ▾ back1.9.png (5)
        back1.9.png (hdpi)
        back1.9.png (mdpi)
        back1.9.png (xhdpi)
        back1.9.png (xxhdpi)
        back1.9.png (xxxhdpi)
      bg_item_product.xml
    ▸ boxes.png (5)
```
Press Control/Shift while dragging on the border to modify layout bounds.

Zoom: 100% — 800%
Patch scale: 2x — 6x
☐ Show lock ☐ Show content
☑ Show patches ☐ Show bad patches
X: 58 px
Y: 73 px

Figure 21. Stretched background image.

5.2 Support Library

Each new release of Android OS most of the time contains an updated version of java framework APIs, which consists of new classes, functions, etc., which are intended to be used by developers to access android's core features. When an application calls API function that requires a newer version of OS than the one is installed on a device, the application will crash. In order for an application to work on android version 2.3.3, the following was done.

1) Support library was used.

Android Support Library is a collection of libraries, which contain functions from a newer version of framework and also of all UI widgets which are intended to be used to create layout [25]. For example, Toolbar is a new widget that has been added to AppCompat v21 library and now is available on older devices [26]. The functionality of components that are provided by support library may differ from what is presented in java framework.

2) Support library includes most but not all functionality. For the functionality, which had not been presented in support library, the author provided its own implementation. In this case, there is a check for the version of OS that device has. If this version is lower than needed, then the code that does not use that new feature will be executed. For example, it was required to reset the activity and to do it a function *recreate* was used from Activity class that is not included in support library. In order to virtualize this function on older devices, the code that is presented in Figure 22 was written.

```
MainActivity mainActivity = ((MainActivity)context);
if(Build.VERSION.SDK_INT>=Build.VERSION_CODES.HONEYCOMB) {
    mainActivity.recreate();
}else{
    mainActivity.finish();
    mainActivity.startActivity(mainActivity.getIntent());
}
```

Figure 22. Replacement of recreate function on older devices.

Another problem was in the absence of UI widget in the support library called NumberPicker. To overcome this it was decided to use third-party implementation [27].

5.3 Data loading

In android, all long running tasks need to be run in a separate thread, since doing this in process' main thread will cause ANR notification. This separate thread may crash with access violation exception because when a user rotates the screen the activity is recreated and the thread will be delivering results to non-existing activity. In this project, the background task is http request to a server. The choice was between 3 components that can be used to solve the screen rotation problem.

1) *AsyncTask* is almost the same as normal java's thread class. Asynctask provides two methods (onPreExecute, onPostExecute) that are be called in a main thread and doInBackground method are called in a separate thread. In order handle screen rotations there are several steps that need to be done. Firstly, the *Activity* that is about to be killed needs to unset pointer in the AsyncTask, so that it would not try to return results to non-existing Activity. Secondly, it needs to save the pointer to this instance of AsyncTask, so the recreated Activity will be able to communicate with the old instance of AsyncTask. Thirdly, the newly created Activity needs to update the pointer to itself in AsyncTask. It is also theoretically possible so that AsyncTask will finish processing before this pointer is updated which means that the task should probably be repeated. Another disadvantage of using AsyncTask is that all existing AsyncTasks gets executed sequentially, i.e. if *Activity* or *Service* start different AsyncTasks from different threads, only 1 will be executed at a time [28]. One of the advantages of using AsyncTask is that it will not stop the background task if the caller component (Activity/Service, etc.) is stopped. So in a situation when a user switches to another app like an incoming phone call, the AsyncTask will continue executing a task. Of course, if android OS will kill the whole process, the AsyncTask's thread will be also terminated. The biggest problem of using AsyncTask is that the developer needs to manually handle all the issues that might occur during screen rotations.

2) *Services* are one of the standard component of android framework that can mark a process as re-creatable. This means that if a process that was hosting this service will be killed, android will automatically schedule a restart of this process. Of course, the main Activity that loads by default will not be called in recreated process because it is not required. Starting from API 19 (KitKat) the services are required to

draw a notification icon at a status bar. In this way, the user always aware of some foreground services running. There are several ways how service can report results to Activity. One way is using a database. The second way is to use LocalBroadcastManager, but this may cause a problem in a situation when activity unregisters itself from receiving broadcasts and new activity will be too late to register itself to listening for events. Having the ability to restart process makes Services an ideal candidate for tasks that should be running in the background, like music player or keylogger. Using Services for simple http requires is overkill because files and databases are used for achieving IPC.

3) *Loaders* can be used to load any arbitrary data types. Loaders consists of 3 components: LoaderManager, LoaderCallbacks and AsyncTaskLoader. AsyncTaskLoader internally relies on AsyncTask, which means that the task will be done in a separate thread. In contrast to AsyncTask, it is possible to execute many AsyncTaskLoaders in parallel. The disadvantage is that the LoaderManager is bounded to Activity's lifecycle, i.e. it will cancel the task if an Activity is stopped. If an Activity is going to be recreated, the loading will not be cancelled. It is very easy to work with Loaders, because the newly recreated Activity only needs to call initLoader method of LoaderManager. The LoaderManager will deliver results to LoaderCallbacks instance.

   Loaders was chosen to execute http requests because LoaderManager automatically resolves all issues that arise during screen rotation. To perform http requests, the Retrofit library is used. After the data is loaded it is needed to preserve it across rotations and death of process to avoid loading in repeatedly. To do this, an onSaveInstanceState method is used, where all needed data is saved into Bundle that will be serialized and delivered into new app's process if the old one will be killed by android OS.

6 Summary

The speed of an information system has a huge impact on user experience and this is why it is important to keep server's response time low. The goal of this work was to create an android and web-based prototype of an order subsystem which should perform filtering of the products in constant time on average with 1 concurrent request.

The proposed solution for achieving performance requirement implies manual preparation of results for all combinations of product’s attribute values and caching them in PostgreSQL. The results of performance test prove that filtering happens in constant time. This solution has many advantages over the widely used method where database query results are associated with user’s input parameters. Firstly, it is possible to adjust the number of stored results through deciding a set of parameters from which all combinations should be generated, while for other parameters, the results should be calculated at runtime. For example, it is possible to store sorted results or perform sorting on the fly. Secondly, it allows to improve the speed of operations that are performed on the cached data through using different data structure(s) for cached data. In this way, the cached data have been stored in an array to be able to select the products for any page in constant time. The disadvantage of such method is that it is not suitable for use with a big number (30+) of input parameters because the number of combinations that should be stored and maintained grows exponentially. One possible solution is to use a data structure that is called inverted index. This however will make the running time dependent on the number of parameters and products.

During the creation of the android application, attention has been paid to three things. Firstly, the responsive design has been created that requires the creation of separate layout for different screen widths and also maintenance of pixel perfection of images. Secondly, the support for older devices has been added by using support library in conjunction with third-party libraries. Thirdly, the problem with crashes during screen rotation has been resolved through using the Loaders component because the LoaderManager automatically resolves any arising issues.

6 Kokkuvõte

Infosüsteemi kiirus mõjutab kasutajakogemust väga ja seetõttu on tähtis, et serveri vastamise aeg oleks lühike. Eesmärgiks oli luua androidi- ja veebipõhine prototüüp, mis peaks teostama keskmiselt muutumatu aja jooksul toodete filtreerimist 1 paralleelse ühendusega.

Pakutud lahendus jõudluse saavutamiseks eeldab manuaalselt ette valmistamist ja puhverdamist PostgreSQL'isse toodete atribuutväärtuste kõikide kombinatsioonide tulemusi. Jõudluse testi tulemused kinnitavad, et filtreeritakse muutumatu aja jooksul. Sellel lahendusel on palju eeliseid laialt kasutatava meetodi ees, kus kasutaja sisestatud parameetrid viitavad üheselt andmebaasi päringutulemustele. Esiteks, sellega saab reguleerida talletatud tulemuste arvu nii, et määratakse kindlad parameetrid, mille alusel luuakse kõik kombinatsioonid ning ülejäänud parameetrite puhul arvutatakse tulemused dünaamiliselt. Näiteks on võimalik talletada sorteeritud tulemusi või sorteerida töö ajal. Teiseks, tänu erinevatele andmestruktuuridele on sellega võimalik vahemälus olevaid andmeid kiiremini läbi töötada. Näiteks, vahemällu talletatud andmeid hoitakse masiivis, et toodete valimine igal lehel toimuks muutumatu aja jooksul. Ent see meetod ei sobi suure hulga (30+) sisendparameetrite korral, kuna talletatavate ja hallatavate kombinatsioonide arv kasvab eksponentsiaalselt. Üheks võimalikuks lahenduseks on kasutada andmestruktuuri, mida nimetatakse *inverted index*. Ent see paneb tööaja sõltuma parameetrite ja toodete arvust.

Androidi rakenduse loomisel on rõhku pandud kolmele asjale. Esimene on paidlik kujundus, mis nõuab erineva laiusega ekraanide puhul erinevat kujunduse loomist ja piltide haldamist. Teiseks, on *Support Library* ja kolmanda osapoole teekide abil lisatud toetus vanadele seadmetele. Kolmas, ekraani pööramisel tekkinud rakenduse kokku jooksmise probleem on lahendatud kasutades *Loaders* komponenti, kuna *LoaderManager* lahendab kõiki ilmnenud vigu automaatselt.

References

[1] Make your node server faster by caching responses with redis [WWW] http://www.sohamkamani.com/blog/2016/10/14/make-your-node-server-faster-with-redis-cache/ (21.12.2016)

[2] Over half of consumers use mobile for shopping [WWW] http://newsok.com/article/5472139 (14.12.2016)

[3] ACID [WWW] https://en.wikipedia.org/wiki/ACID (10.12.2016)

[4] ActiveMQ fails to start on Ubuntu [WWW] https://bugs.launchpad.net/ubuntu/+source/activemq/+bug/1361831 (09.12.2016)

[5] Android NDK Native APIs [WWW] https://developer.android.com/ndk/guides/stable_apis.html (08.12.2016)

[6] Favre, L. UML and the Unified Process IRM Press, 2003 (09.10.2016)

[7] Storing Hierarchical Data in a Database [WWW] https://www.sitepoint.com/hierarchical-data-database/ (11.10.2016)

[8] Adaptive Server Enterprise 12.5.1 [WWW] http://infocenter.sybase.com/help/topic/com.sybase.dc20020_1251/html/databases/databases233.htm (12.12.2016)

[9] General PostgreSQL limits [WWW] https://www.postgresql.org/about/ (12.12.2016)

[10] Concrete table inheritance [WWW] http://www.slideshare.net/billkarwin/practical-object-oriented-models-in-sql/17-EntityAttributeValue_Concrete_Table_Inheritance_Dene (13.10.2016)

[11] Number of k-combinations for all k [WWW] https://en.wikipedia.org/wiki/Combination#Number_of_k-combinations_for_all_k (14.09.2016)

[12] Binomial theorem [WWW] https://en.wikipedia.org/wiki/Binomial_theorem (14.09.2016)

[13] Inside the PostgreSQL Shared Buffer Cache [WWW] https://2ndquadrant.com/media/pdfs/talks/InsideBufferCache.pdf (15.12.2016)

[14] Write-through, write-around and write-back cache [WWW] http://www.computerweekly.com/feature/Write-through-write-around-write-back-Cache-explained (15.12.2016)

[15] Built-in Operator Classes [WWW] https://www.postgresql.org/docs/current/static/gin-builtin-opclasses.html (07.12.2016)

[16] PostgreSQL Columnar Store for Analytic Workloads [WWW] https://www.citusdata.com/blog/2014/04/03/columnar-store-for-analytics/ (21.12.2016)

[17] TOAST [WWW] https://wiki.postgresql.org/wiki/TOAST (08.12.2016)

[18] Supporting Multiple Screens [WWW] https://developer.android.com/guide/practices/screens_support.html (09.12.2016)

[19] Android Fragmentation Visualized [WWW] http://opensignal.com/reports/2015/08/android-fragmentation (09.12.2016)

[20] Pixel density [WWW] https://en.wikipedia.org/wiki/Pixel_density (09.12.2016)

[21] Automatically Scaling Android Apps For Multiple Screens [WWW] http://www.vanteon.com/downloads/Scaling_Android_Apps_White_Paper.pdf (10.12.2016)

[22] Android: Some Screen Densities, Sizes, Configurations, and Icon Sizes [WWW] http://dbrodersen.com/CIT_238/AndroidScreens.html (11.12.2016)

[23] Multiple screens and scaling Android UI images [WWW] http://dexxtr.com/post/50327457086/multiple-screens-and-scaling-android-ui-images (11.12.2016)

[24] Tips for Designers: from a Developer [WWW] http://vinsol.com/blog/2014/11/20/tips-for-designers-from-a-developer/ (13.12.2016)

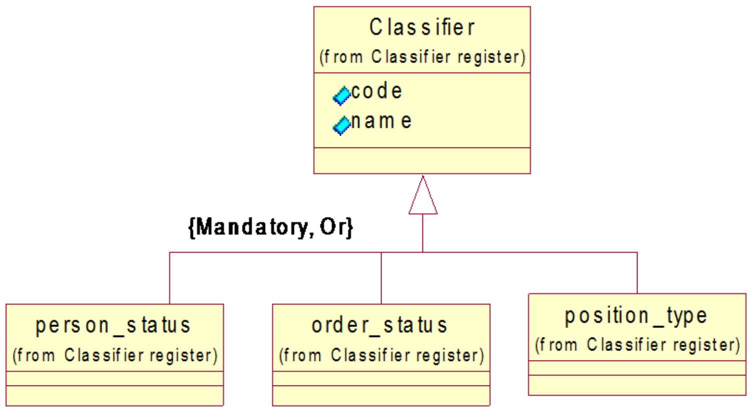
[25] Support Library Features [WWW] https://developer.android.com/topic/libraries/support-library/features.html (14.12.2016)

[26] Using the App ToolBar [WWW] https://guides.codepath.com/android/Using-the-App-ToolBar (17.12.2016)

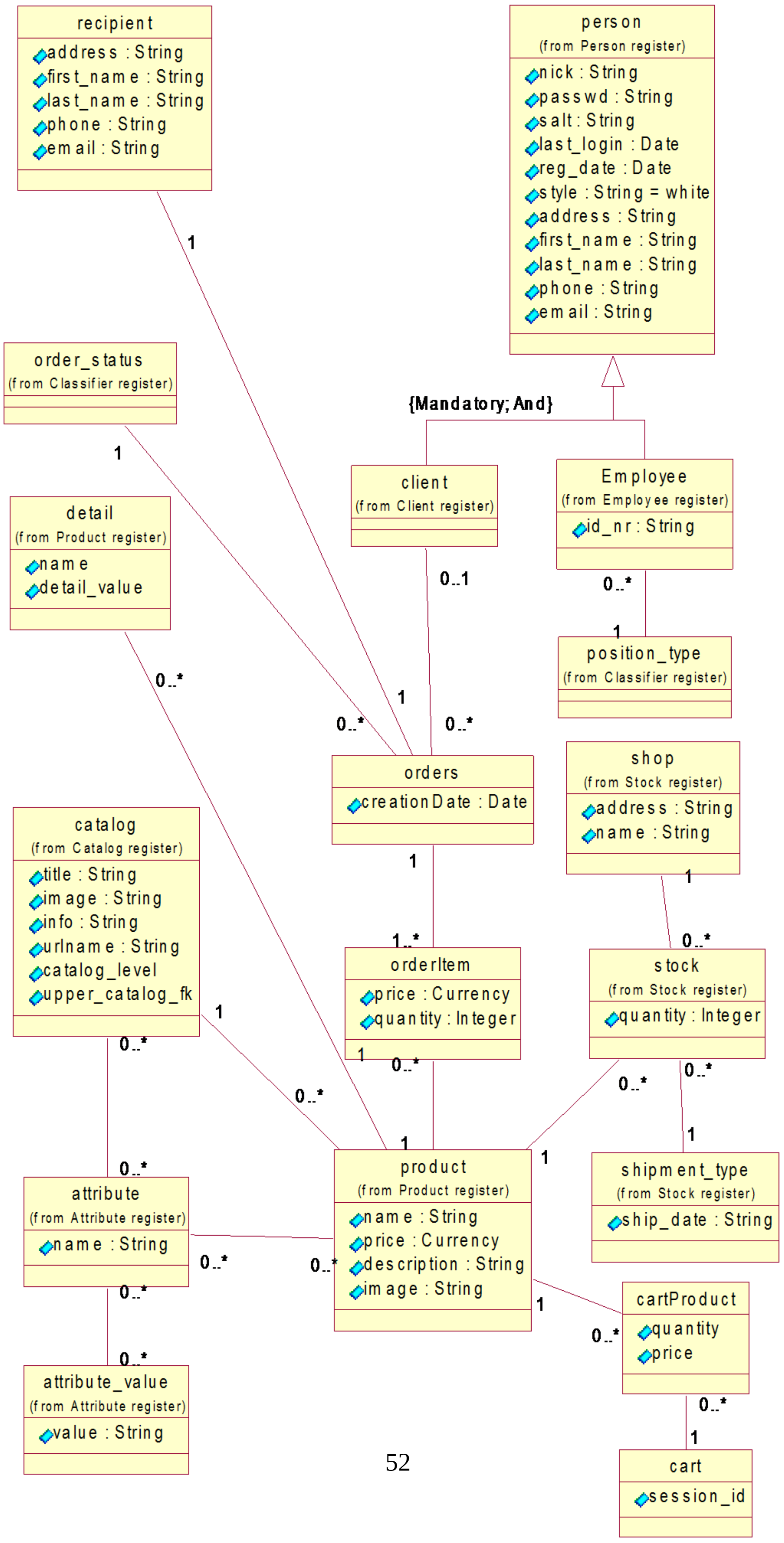
[27] A backport of the Android 4.2 NumberPicker [WWW] https://github.com/SimonVT/android-numberpicker (18.12.2016)

[28] AsyncTask [WWW] https://developer.android.com/reference/android/os/AsyncTask.html (22.12.2016)

Appendix 1 – Entity-relationship diagram of registers that are used by order functional subsystem

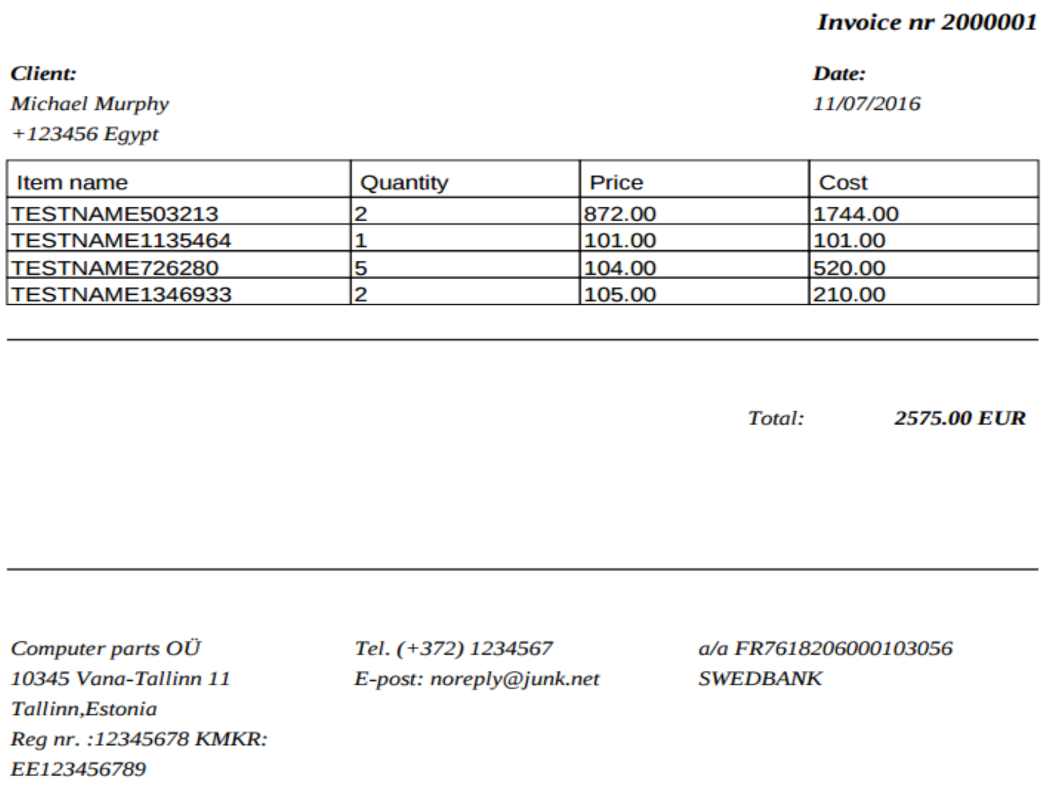


Classifier
(from Classifier register)
code
name
{Mandatory, Or}
person_status
(from Classifier register)
order_status
(from Classifier register)
position_type
(from Classifier register)



recipient
address : String
first_name : String
last_name : String
phone : String
email : String
person
(from Person register)
nick : String
passwd : String
salt : String
last_login : Date
reg_date : Date
style : String = white
address : String
first_name : String
last_name : String
phone : String
email : String
{Mandatory; And}
order_status
(from Classifier register)
client
(from Client register)
Employee
(from Employee register)
id_nr : String
detail
(from Product register)
name
detail_value
position_type
(from Classifier register)
orders
creationDate : Date
shop
(from Stock register)
address : String
name : String
catalog
(from Catalog register)
title : String
image : String
info : String
urlname : String
catalog_level
upper_catalog_fk
orderItem
price : Currency
quantity : Integer
stock
(from Stock register)
quantity : Integer
product
(from Product register)
name : String
price : Currency
description : String
image : String
attribute
(from Attribute register)
name : String
shipment_type
(from Stock register)
ship_date : String
cartProduct
quantity
price
attribute_value
(from Attribute register)
value : String
cart
session_id
1
0..1
0..*
1..*

Appendix 2 – Example of generated pdf file with order details



Invoice nr 2000001

Client:
Michael Murphy
+123456 Egypt

Date:
11/07/2016

Item name	Quantity	Price	Cost
TESTNAME503213	2	872.00	1744.00
TESTNAME1135464	1	101.00	101.00
TESTNAME726280	5	104.00	520.00
TESTNAME1346933	2	105.00	210.00

Total: ***2575.00 EUR***

Computer parts OÜ
10345 Vana-Tallinn 11
Tallinn,Estonia
Reg nr. :12345678 KMKR:
EE123456789

Tel. (+372) 1234567
E-post: noreply@junk.net

a/a FR7618206000103056
SWEDBANK

Appendix 3 – Trigger for updating refCount column

```
CREATE OR REPLACE FUNCTION insrt_amount() RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
      UPDATE catalog_attribute AS ca SET refcount=refcount+1
      WHERE ca.attribute_value_fk IN
(NEW.brand_fk,NEW.resolution_fk,NEW.screen_fk,NEW."Resp time_fk",NEW.color_fk,NEW."Battery
Life_fk")
      AND ca.catalog_fk=NEW.catalog_fk;

      RETURN NEW;
END;
$$;
CREATE TRIGGER tr_insrt_prod
AFTER INSERT ON product
FOR EACH ROW
EXECUTE PROCEDURE insrt_amount();
```

Appendix 4 – Test data for catalogs' attributes

```
INSERT INTO attribute(name)
VALUES ('brand');
INSERT INTO attribute(name)
VALUES ('resolution');
INSERT INTO attribute(name)
VALUES ('screen');
INSERT INTO attribute(name)
VALUES ('Resp time');
INSERT INTO attribute(name)
VALUES ('color');
INSERT INTO attribute(name)
VALUES ('Battery Life');

INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (1,'Apple',1);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (1,'Asus',2);
...

INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (1,'HP',23);
```

```
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (2,'400×800',1);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (2,'1024x768',2);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (2,'1280x720',3);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (2,'1280x800',4);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (2,'1920x1080',5);

INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (3,'10''''',1);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (3,'15''''',2);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (3,'17''''',3);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (3,'19''''',4);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (3,'22''''',5);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (3,'27''''',6);

INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (4,'5 ms',1);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (4,'6 ms',2);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (4,'7 ms',3);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (4,'8 ms',4);

INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (5,'black',1);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (5,'white',2);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (5,'red',3);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (5,'green',4);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (5,'blue',5);

INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (6,'1 hour',1);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (6,'2 hour',2);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
VALUES (6,'4 hour',3);
INSERT INTO attribute_value(attribute_fk,attributeValue,orderby)
```

```
VALUES (6,'111 hour',4);

INSERT INTO catalog_attribute(catalog_fk,attribute_fk,attribute_value_fk)
VALUES (9,1,1);
INSERT INTO catalog_attribute(catalog_fk,attribute_fk,attribute_value_fk)
VALUES (9,1,2);
INSERT INTO catalog_attribute(catalog_fk,attribute_fk,attribute_value_fk)
VALUES (9,1,3);
...
INSERT INTO catalog_attribute(catalog_fk,attribute_fk,attribute_value_fk)
VALUES (9,4,38);

INSERT INTO catalog_attribute(catalog_fk,attribute_fk,attribute_value_fk)
VALUES (3,1,10);
...
INSERT INTO catalog_attribute(catalog_fk,attribute_fk,attribute_value_fk)
VALUES (3,6,47);
```

Appendix 5 – Resulting composite keys from the example

```
0)catalog:9
1)catalog:9:resp time:5 ms
2)catalog:9:resolution:1024x768
3)catalog:9:resolution:1024x768:resp time:5 ms
4)catalog:9:size:17''
5)catalog:9:size:17'':resp time:5 ms
6)catalog:9:size:17'':resolution:1024x768
7)catalog:9:size:17'':resolution:1024x768:resp time:5 ms
8)catalog:9:brand:Samsung
9)catalog:9:brand:Samsung:resp time:5 ms
10)catalog:9:brand:Samsung:resolution:1024x768
11)catalog:9:brand:Samsung:resolution:1024x768:resp time:5 ms
12)catalog:9:brand:Samsung:size:17''
13)catalog:9:brand:Samsung:size:17'':resp time:5 ms
14)catalog:9:brand:Samsung:size:17'':resolution:1024x768
15)catalog:9:brand:Samsung:size:17'':resolution:1024x768:resp time:5 ms
16)catalog:9:price:asc
17)catalog:9:price:desc
18)catalog:9:price:asc:resp time:5 ms
19)catalog:9:price:desc:resp time:5 ms
20)catalog:9:price:asc:resolution:1024x768
21)catalog:9:price:desc:resolution:1024x768
22)catalog:9:price:asc:resolution:1024x768:resp time:5 ms
23)catalog:9:price:desc:resolution:1024x768:resp time:5 ms
24)catalog:9:price:asc:size:17''
25)catalog:9:price:desc:size:17''
```

```
26)catalog:9:price:asc:size:17'':resp time:5 ms
27)catalog:9:price:desc:size:17'':resp time:5 ms
28)catalog:9:price:asc:size:17'':resolution:1024x768
29)catalog:9:price:desc:size:17'':resolution:1024x768
30)catalog:9:price:asc:size:17'':resolution:1024x768:resp time:5 ms
31)catalog:9:price:desc:size:17'':resolution:1024x768:resp time:5 ms
32)catalog:9:price:asc:brand:Samsung
33)catalog:9:price:desc:brand:Samsung
34)catalog:9:price:asc:brand:Samsung:resp time:5 ms
35)catalog:9:price:desc:brand:Samsung:resp time:5 ms
36)catalog:9:price:asc:brand:Samsung:resolution:1024x768
37)catalog:9:price:desc:brand:Samsung:resolution:1024x768
38)catalog:9:price:asc:brand:Samsung:resolution:1024x768:resp time:5 ms
39)catalog:9:price:desc:brand:Samsung:resolution:1024x768:resp time:5 ms
40)catalog:9:price:asc:brand:Samsung:size:17''
41)catalog:9:price:desc:brand:Samsung:size:17''
42)catalog:9:price:asc:brand:Samsung:size:17'':resp time:5 ms
43)catalog:9:price:desc:brand:Samsung:size:17'':resp time:5 ms
44)catalog:9:price:asc:brand:Samsung:size:17'':resolution:1024x768
45)catalog:9:price:desc:brand:Samsung:size:17'':resolution:1024x768
46)catalog:9:price:asc:brand:Samsung:size:17'':resolution:1024x768:resp
time:5 ms
47)catalog:9:price:desc:brand:Samsung:size:17'':resolution:1024x768:resp
time:5 ms
```

Appendix 6 –Function for generation of composite keys and the PLPGSQL functions for maintaining arrays of product ids

```java
private List<String> retoSetOfAttrValues(Product prod) {
	List<String> aVS = new ArrayList<>();
	aVS.add(":price:asc");
	for(String attrKey:prod.getAttrs().keySet()){
		String attrValue = prod.getAttrs().get(attrKey);
		aVS.add(":"+attrKey+":"+attrValue);
	}
	return aVS;
}
```

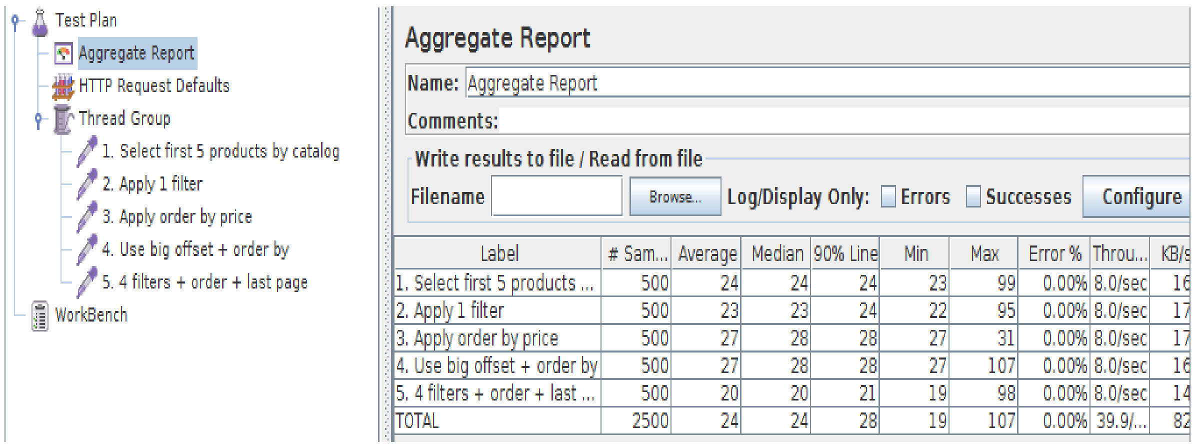
```
private int[] binary_form(int number) {
	int[] binaryString = new int[32];
	for (int i = 0; i < 32; i++) {
		if (((0x80000000 >>> i) & number) != 0) {
			binaryString[i] = 1;
		} else {
			binaryString[i] = 0;
		}
	}
	return binaryString;
}

private int[] cutNulls(int[] binaryString, int nbImportantBits) {
	int[] newBinaryString = new int[nbImportantBits];
	for (int i = 0; i < nbImportantBits; i++) {
		newBinaryString[i] = binaryString[32 - nbImportantBits + i];
	}
	return newBinaryString;
}


public void genSetOfKVpairsForProducts(List<Product> products) {

	Map<String, List<Integer>> cachedResults = new LinkedHashMap<>();
	for (Product product : products) {
		List<String> aVs = retoSetOfAttrValues(product);

		List<List<String>> subsets = new ArrayList<List<String>>();
		int N = aVs.size();
		int nbOfAllSubsets = (int) Math.pow(2, N);

		//gen all subsets by getting binary representation of numbers between
0 and (2^n)-1
		for (int i = 0; i < nbOfAllSubsets; i++) {
			int[] binaryString = binary_form(i);
			binaryString = cutNulls(binaryString, N);
			List<String> subset = new ArrayList<>();
			int found = 0;
			for (int j = 0; j < binaryString.length; j++) {
				if (binaryString[j] == 1) {
					subset.add(aVs.get(j));
					found = 1;
				}
			}
			if (found == 0)
				subset.add("");
			subsets.add(subset);
		}

		String compositeKey = "";
```

```
			for (List<String> subset : subsets) {
				String compositeKeyDescPrice = "";
				//construction of composite key and creation of 3 sorted versions of the
key
				compositeKey = "catalog:" +
Integer.toString(product.getCatalog().getId());
				for (String element : subset) {
					compositeKey+=element;
				}

				if(compositeKey.contains("price"))
					compositeKeyDescPrice = compositeKey.replace("price:asc",
"price:desc");


				if (!cachedResults.containsKey(compositeKey)) {
					cachedResults.put(compositeKey, new ArrayList<>());
					if(!compositeKeyDescPrice.isEmpty())
						cachedResults.put(compositeKeyDescPrice, new
ArrayList<>());
				}
				//associate product ids with composite key
				List<Integer> prodIds = cachedResults.get(compositeKey);
				prodIds.add(product.getId());
				if(!compositeKeyDescPrice.isEmpty()){
					prodIds = cachedResults.get(compositeKeyDescPrice);
					prodIds.add(product.getId());
				}
			}
		}

		String SQL = "SELECT insrt_or_append(?,?);";

		//save in DB
		jdbcTemplate.batchUpdate(SQL, new BatchPreparedStatementSetter() {

			@Override
			public void setValues(PreparedStatement ps, int i) throws SQLException {
				String key = (String) cachedResults.keySet().toArray()[i];
				Array results = ps.getConnection().createArrayOf("integer",
cachedResults.get(key).toArray());
				ps.setArray(1, results);
				ps.setString(2, key);
			}

			@Override
			public int getBatchSize() {
				return cachedResults.size();
			}
		});
```

```
}


CREATE OR REPLACE FUNCTION insrt_or_append(i_pids INTEGER[],i_key TEXT)
RETURNS VOID AS
$$
DECLARE
exist TEXT;
BEGIN
        SELECT key INTO exist FROM cachedtb WHERE key = i_key;
       IF CHAR_LENGTH(exist)>0 THEN
             UPDATE cachedtb SET value = value || i_pids WHERE key = i_key;
       ELSE
             INSERT INTO cachedtb(key,value) VALUES (i_key, i_pids);
       END IF;

END
$$
LANGUAGE 'plpgsql' SECURITY DEFINER;


CREATE OR REPLACE FUNCTION sort_keys()
RETURNS VOID AS
$$
DECLARE
tb_row cachedtb%ROWTYPE;
BEGIN
       FOR tb_row IN
       SELECT *
       FROM cachedtb
       LOOP
             IF tb_row.key LIKE '%price:asc%' THEN
                   UPDATE cachedtb SET value =
                         array(SELECT p.id pid
                         FROM product p
                         WHERE p.id IN (SELECT unnest(tb_row.value)  )
                         ORDER BY price ASC )
                         ,total_amount = icount(tb_row.value)
                    WHERE key = tb_row.key;
             END IF;

             IF tb_row.key LIKE '%price:desc%' THEN
                   UPDATE cachedtb SET value =
                         array(SELECT p.id pid
                         FROM product p
                         WHERE p.id IN (SELECT unnest(tb_row.value)  )
                         ORDER BY price DESC )
                         ,total_amount = icount(tb_row.value)
                    WHERE key = tb_row.key;
             END IF;
```

```
            IF tb_row.key NOT LIKE '%price:desc%' AND tb_row.key NOT LIKE '%price:asc%' THEN
                  UPDATE cachedtb SET total_amount = icount(tb_row.value)
                   WHERE key = tb_row.key;
            END IF;

      END LOOP;

END
$$
LANGUAGE 'plpgsql' SECURITY DEFINER;
```

Appendix 7 – Example of test results with 4M products



Aggregate Report

Name: Aggregate Report

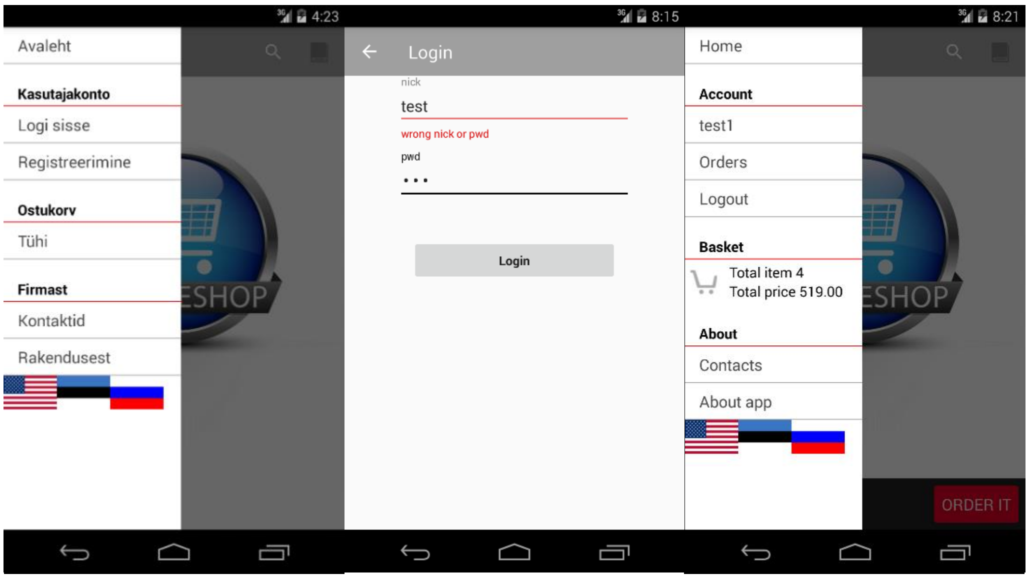
Comments:

Write results to file / Read from file

Filename | Log/Display Only: ☐ Errors ☐ Successes

Label	# Sam...	Average	Median	90% Line	Min	Max	Error %	Throu...	KB/s
1. Select first 5 products ...	500	24	24	24	23	99	0.00%	8.0/sec	16
2. Apply 1 filter	500	23	23	24	22	95	0.00%	8.0/sec	17
3. Apply order by price	500	27	28	28	27	31	0.00%	8.0/sec	17
4. Use big offset + order by	500	27	28	28	27	107	0.00%	8.0/sec	16
5. 4 filters + order + last ...	500	20	20	21	19	98	0.00%	8.0/sec	14
TOTAL	2500	24	24	28	19	107	0.00%	39.9/...	82

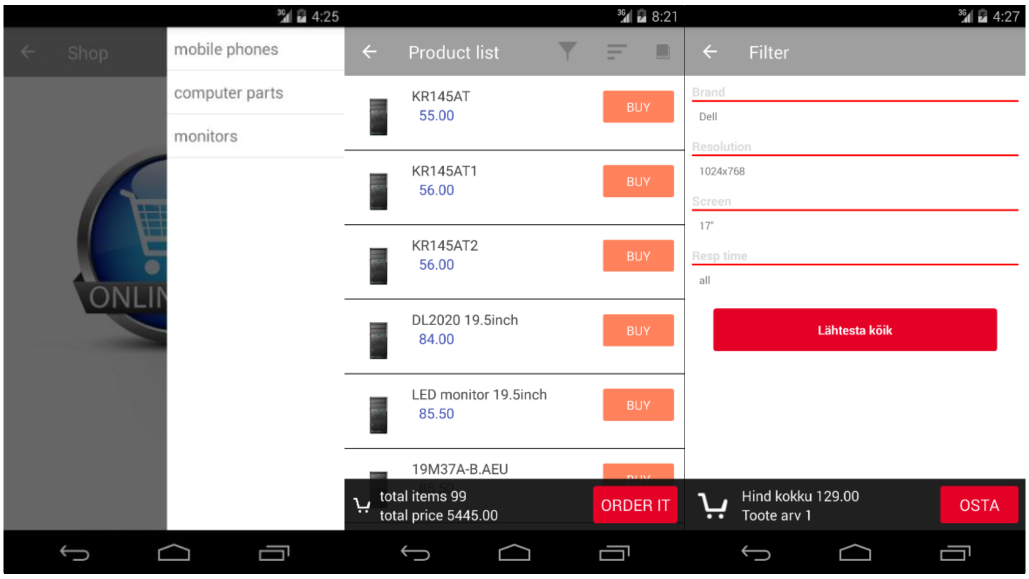
Appendix 8 – Android user interfaces



nick
test
wrong nick or pwd
pwd
•••

Total item 4
Total price 519.00

Signup form.

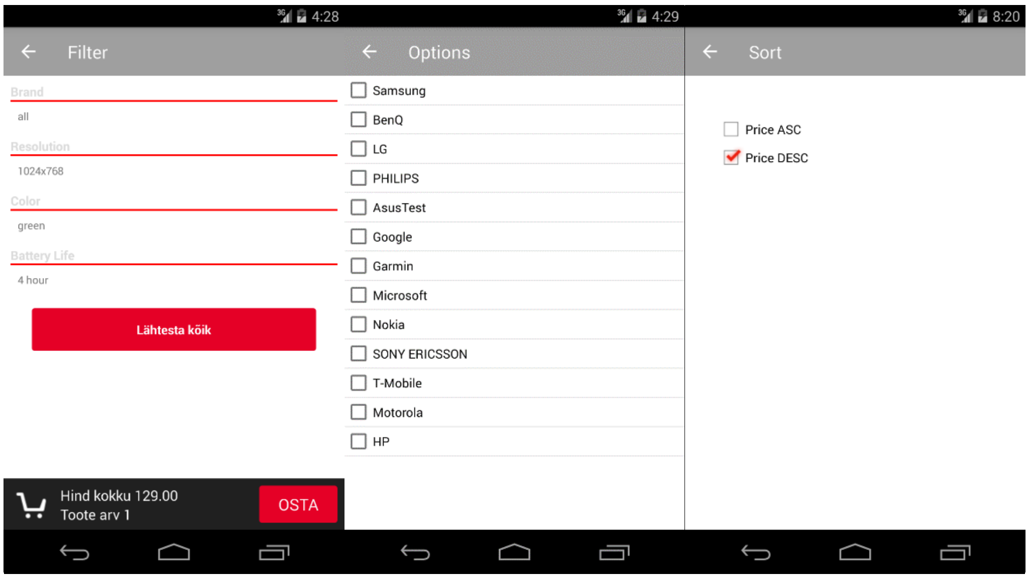


KR145AT 55.00
KR145AT1 56.00
KR145AT2 56.00
DL2020 19.5inch 84.00
LED monitor 19.5inch 85.50
19M37A-B.AEU

total items 99
total price 5445.00

Brand
Dell
Resolution
1024x768
Screen
17"
Resp time
all

Hind kokku 129.00
Toote arv 1



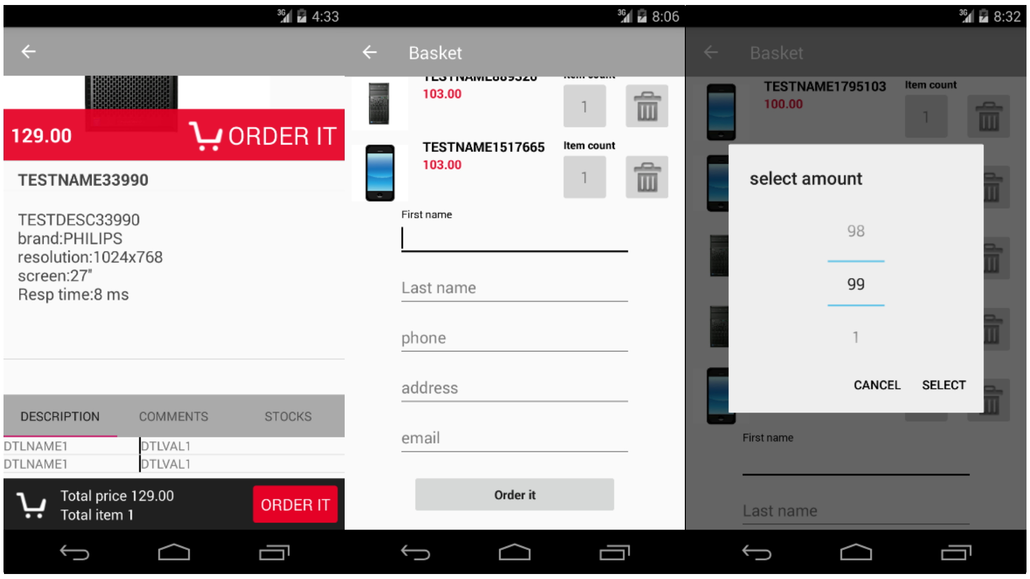
Brand
all
Resolution
1024x768
Color
green
Battery Life
4 hour

Hind kokku 129.00
Toote arv 1

☐ Samsung
☐ BenQ
☐ LG
☐ PHILIPS
☐ AsusTest
☐ Google
☐ Garmin
☐ Microsoft
☐ Nokia
☐ SONY ERICSSON
☐ T-Mobile
☐ Motorola
☐ HP

☐ Price ASC
☑ Price DESC

Product search flow.



129.00 ORDER IT

TESTNAME33990

TESTDESC33990
brand:PHILIPS
resolution:1024x768
screen:27"
Resp time:8 ms

DESCRIPTION	COMMENTS	STOCKS

DTLNAME1	DTLVAL1
DTLNAME1	DTLVAL1

Total price 129.00
Total item 1
ORDER IT

Basket

103.00

TESTNAME1517665 Item count 1
103.00

First name

Last name

phone

address

email

Order it

Basket

TESTNAME1795103 Item count 1
100.00

select amount

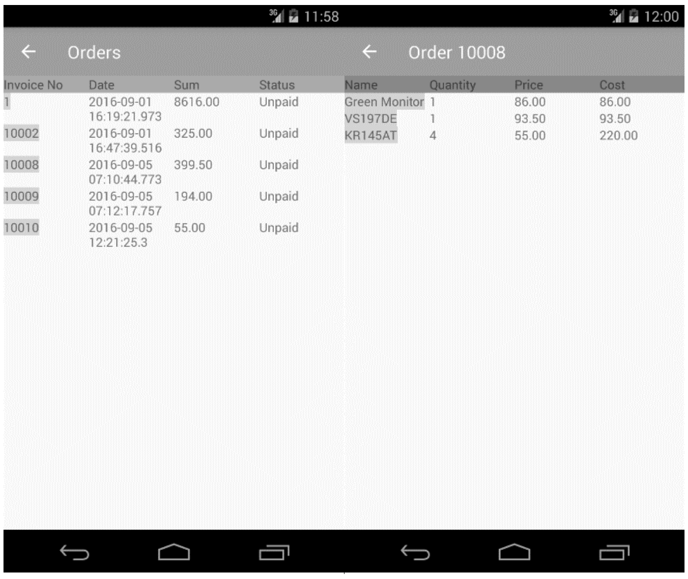
98
99
1

CANCEL SELECT

First name

Last name

Order creation form.



Orders

Invoice No	Date	Sum	Status
1	2016-09-01 16:19:21.973	8616.00	Unpaid
10002	2016-09-01 16:47:39.516	325.00	Unpaid
10008	2016-09-05 07:10:44.773	399.50	Unpaid
10009	2016-09-05 07:12:17.757	194.00	Unpaid
10010	2016-09-05 12:21:25.3	55.00	Unpaid

Order 10008

Name	Quantity	Price	Cost
Green Monitor	1	86.00	86.00
VS197DE	1	93.50	93.50
KR145AT	4	55.00	220.00

Order overview form.