BuySafe
Design Description

Version 1.0

Revision History

Date	Version	Description	Author
2002-11-09	0.1	Initial Draft	Juraj Murgić
2012-12-02	0.2	Changes made after the end of the first sprint	Juraj Murgić
2012-01-18	0.3	Updated sections 3 and 4.1.2	Želimir Kompes
2012-01-20	1.0	Final version	Juraj Murgić, Želimir Kompes, Saša Marjančić

Table of Contents

1. Introduction

1.1 Purpose of this document

The purpose of this document is to provide a detailed description of the BuySafe design, including software architecture, external interfaces and detailed software design.

1.2 Intended Audience

This document shall be used in all phases of the project as a design guideline. Intended audience of this project is:

- project supervisor
- project leader
- team members
- course staff

1.3 Scope

This document defines the design of the BuySafe application. It contains information about the following aspects of project design:

- software architecture, including conceptual design and system specification
- external interfaces, including hardware, software and user interfaces
- detailed software design, including modules, data flow, data types, database model

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definitions
BuySafe	The name of the project
JSon	lightweight data-interchange format

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
MVC	Model / View / Controller
TCP	Transmission Control Protocol
IP	Internet Protocol
XML	Extensible Markup Language
SQL	Structured Query Language
ADT	Android Development Tools
GUI	Graphical User Interface
SSL	Secure Sockets Layer

1.5 References

1. http://www.asp.net/mvc
2. http://struts.apache.org/2.x/
3. http://developer.android.com/tools/help/adt.html

2. Software architecture

2.1 Conceptual design

The architecture follows typical client-server model. Client has module necessary for interaction with the user, logic module, persistence module and a module responsible for exchanging data with the server. Server has logic module, SQL module, scheduler module and a module used for interaction with the client.

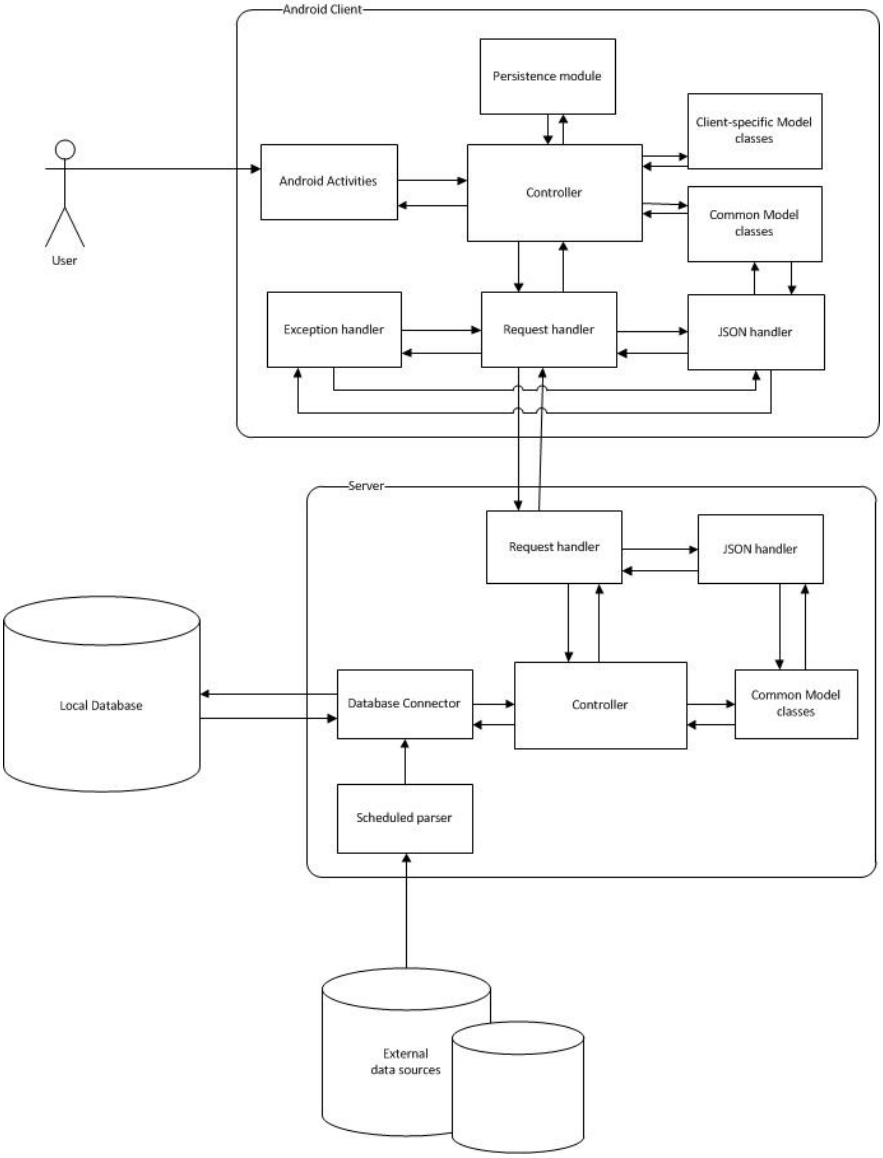

Figure 1: Software architecture

2.1.1 Android Activity

This component manages the GUI and is responsible for interaction with the user.

2.1.2 Client-side Logic

As the name explains, it contains all the manager and entity classes that are required for client-side business logic.

2.1.3 Persistence Module

Persistence module saves and loads data from the android memory.

2.1.4 Request Handler (client)

This component connects to the server. It sends the data to server as name-value pairs and receives data as XML strings.

2.1.5 JSON-parser

It has classes that parse the JSON strings and instantiates the entity classes based on data in the strings.

2.1.6 Request Handler (server)

This module receives data from client and sends it to the required manager classes. It also sends data back to client as XML strings.

2.1.7 Server-side controller

It contains all the manager and entity classes that are required for server-side business logic.

2.1.8 Database Connector

This module is responsible for managing connections with the database. It also contains the stored SQL queries.

2.1.9 Scheduled Parser

This module handles the updating of database based on the new information in external data sources. Classes in this module check for updates after fixed intervals of time.

2.1.10 Model classes

This module consists of entity classes used throughout the project.

2.2 System specification

Client side solely uses Android platform, with Java as the programming language. All the modules will be written in java and will be based on Android API. Server will be Linux-based. J2EE is the environment and Java is the server-side programming language too.

2.3 External Components

The server-side will use Apache Tomcat as server. Virtual Machine to be used on server is Turnkey. MySQL will be used as database. Apart from this, Android will use a third party open-source barcode scanning module called ZXing.

3. External interfaces

This section should provide an overview of interfaces for communication between the product and the systems/entities/humans outside the system borders. Communication interfaces among components of the system should not be described in this section.

3.1 Hardware Interfaces

For full functionality application requires a mobile device with integrated camera. Application will have declared camera permission in Android Manifest file. User will need to give stated permission for full functionality.

3.2 Software Interfaces

Application functionalities of the external interfaces will be developed using standard Android Development Tools (ADT) add-on for Eclipse. Application logic will be developed in default Android technologies; business logic will be developed in Java and presentation logic in XML.

3.3 Communication Interfaces

Main functionality of the application is barcode scanning and gathering of product information. Product information will be stored on a remote server. Therefore, application requires continuous Internet access during usage of main functionality. Additional functionality, like shopping list, doesn't require Internet access.

3.4 User Interfaces

Graphical User Interface will be modeled like a classic Android application. User will be able to navigate through application via mobile device touch screen. As shown in figure 2, main screen will consist of one central menu with several options. Immediately after opening the application, user will be able to create or edit profile, search for the product, view personal shopping list, view all products which are safe for him, compare quality of multiple products or simply exit application. This design allows easy implementation of future functionalities. Three-click rule design will be used for design of application. Application user will be able to find any information with no more than three mouse clicks.

Figure 2: BuySafe main screen

4. Detailed software design

4.1 Implementation modules / components

The MVC design pattern helps creating applications that separate the different aspects of the application (input logic, business logic, and GUI logic), while providing a loose coupling between these elements. The pattern specifies where each kind of logic should be located in the application. The GUI logic belongs in the view. Input logic belongs in the controller. Business logic belongs in the model. This separation helps managing complexity when building an application, because it enables focusing on one aspect of the implementation at a time.

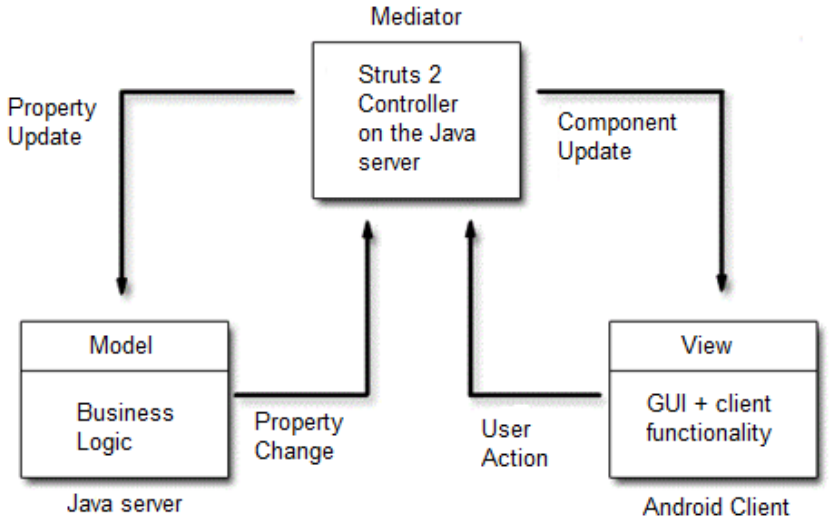

Figure 3: Implementation modules/components

1. The user interacts with the user interface in some way (e.g. presses a button).
2. A controller handles the input event from the user interface, often via a registered handler or callback.
3. The controller notifies the model of the user action, possibly resulting in a change in the model's state, (e.g. controller updates user's shopping cart).
4. A view uses the model (indirectly) to generate an appropriate user interface (e.g. the view produces a screen listing the shopping cart contents). The view gets its own data from the model. The model has no direct knowledge of the view.
5. The user interface waits for further user interactions, which begins the cycle anew.

4.1.1 Structured view

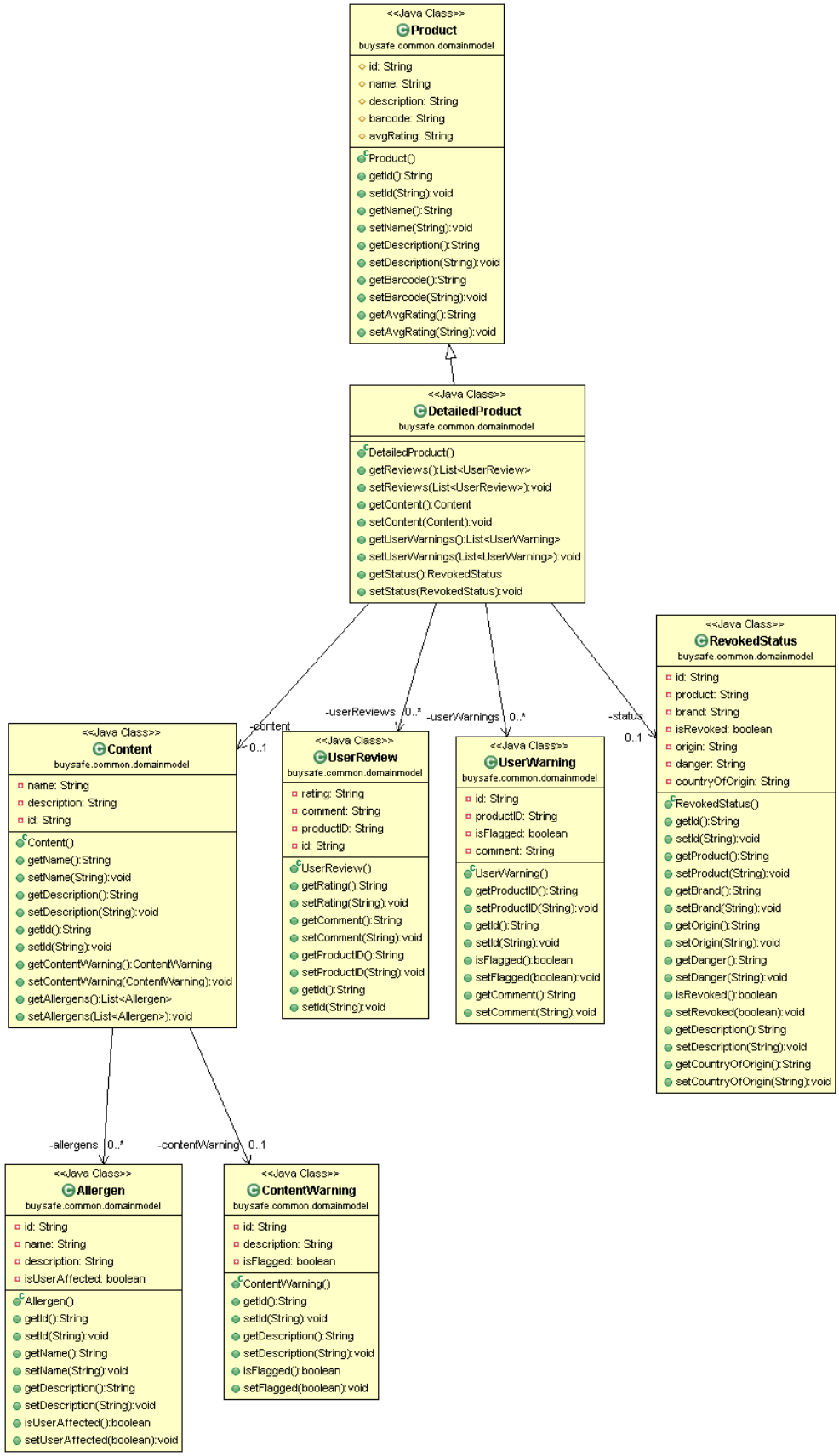

Figure 4: Class diagram - Domain model

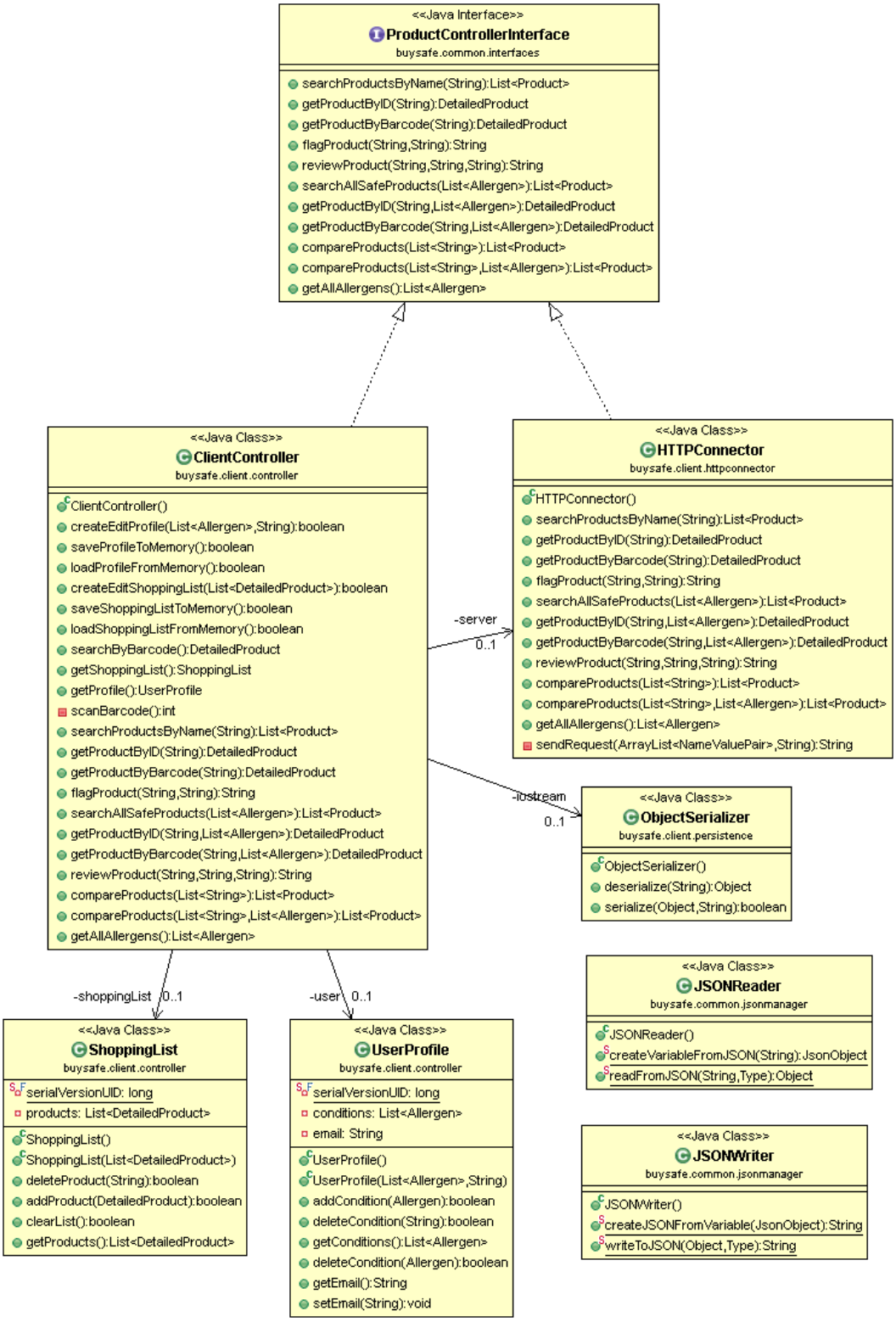

Figure 5: Class diagram - Client

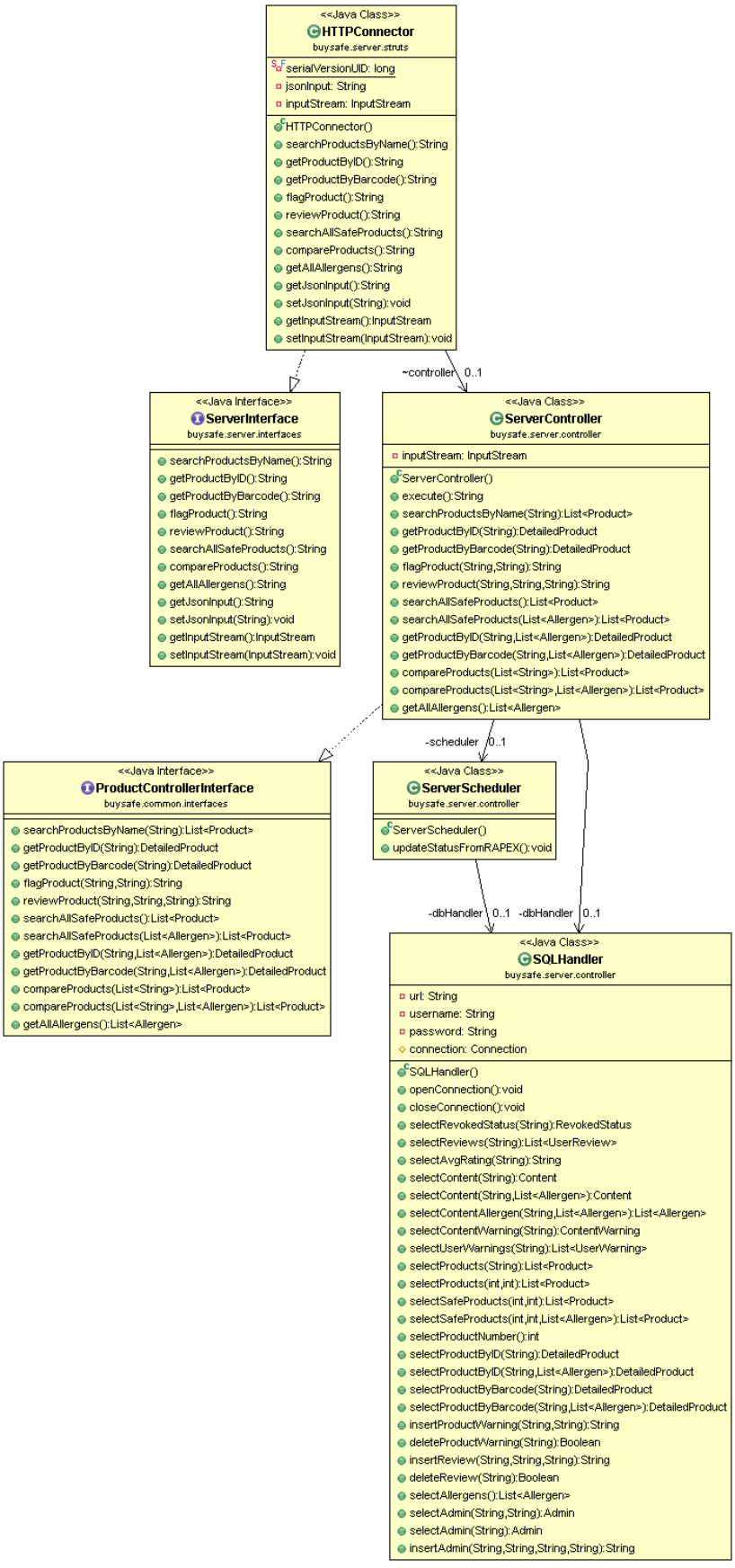

Figure 6: Class diagram - Server

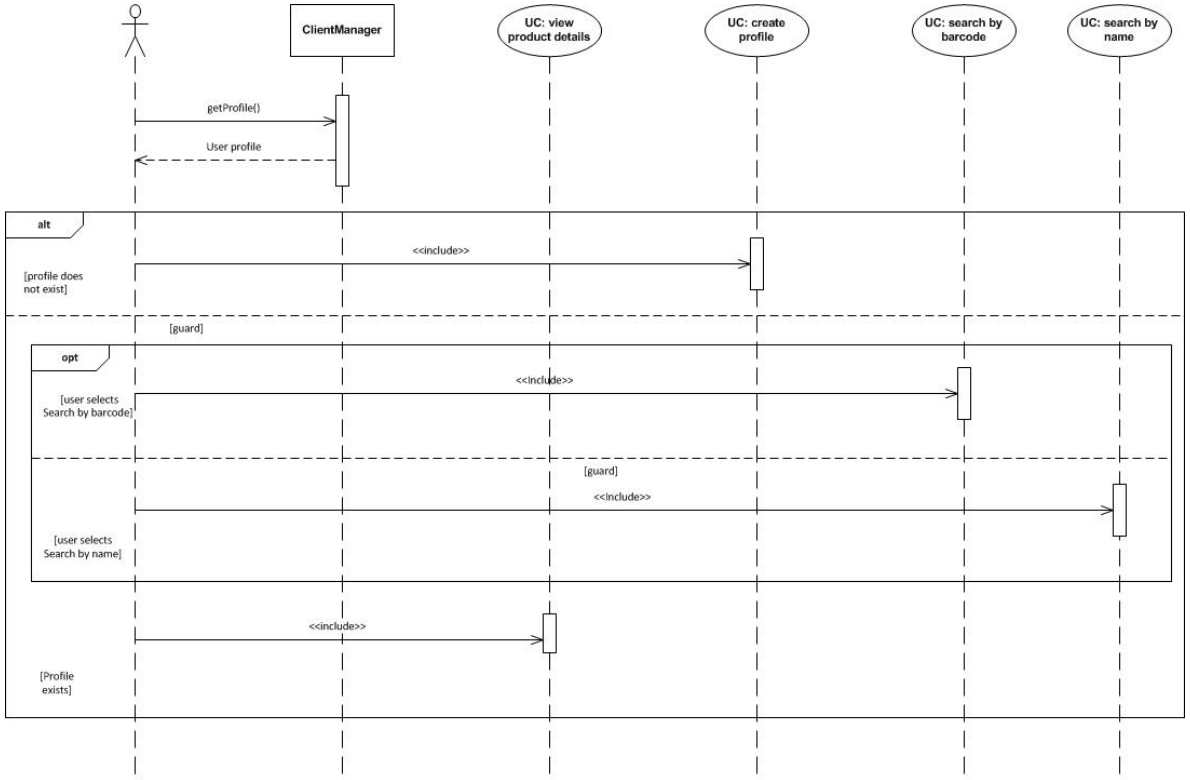

Figure 7: Sequence diagram (1)

This diagram describes how user manages his profile and then checks if the product he/she wants to buy is safe or not. First the user clicks on the "search product" button, and event handler will pass the message to ClientManager and try to get his profile from memory. If there is no history profile returned, then the user will be required to create a new profile which contains the illness or allergic information. After the profile is done, the user will be provided with two options, which are search by barcode and search by name. Both options will make the ServerMangager check the product that the user input against the database. The database returns the contents of the product to ServerManager which will check if the user profile conflicts with the contents of the product. Besides, ServerManger also checks if the product the user wants to buy is forbidden on the market. Finally the results will be passed to the user.

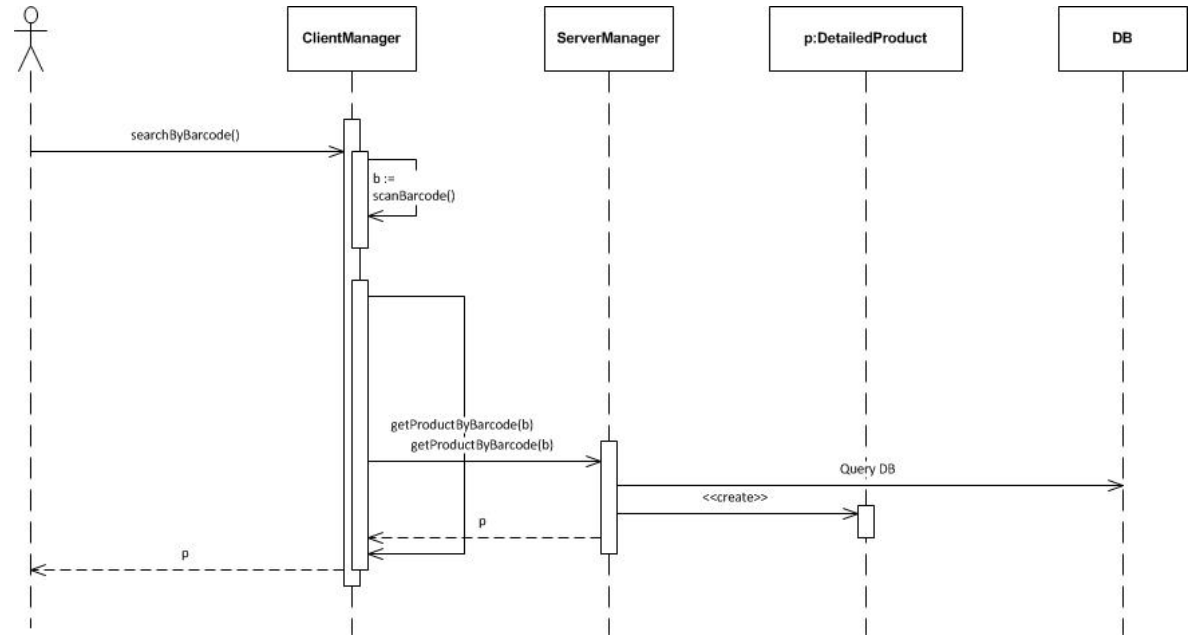

Figure 8: Sequence diagram (2)

This diagram shows how the user could search a product with barcode. First the user selects the search product by barcode option and then the ClientManager will call the scan barcode function. After the barcode is retrieved the ServerManager will use this barcode to make a query against the Product table in database and then return the contents of the product to user.

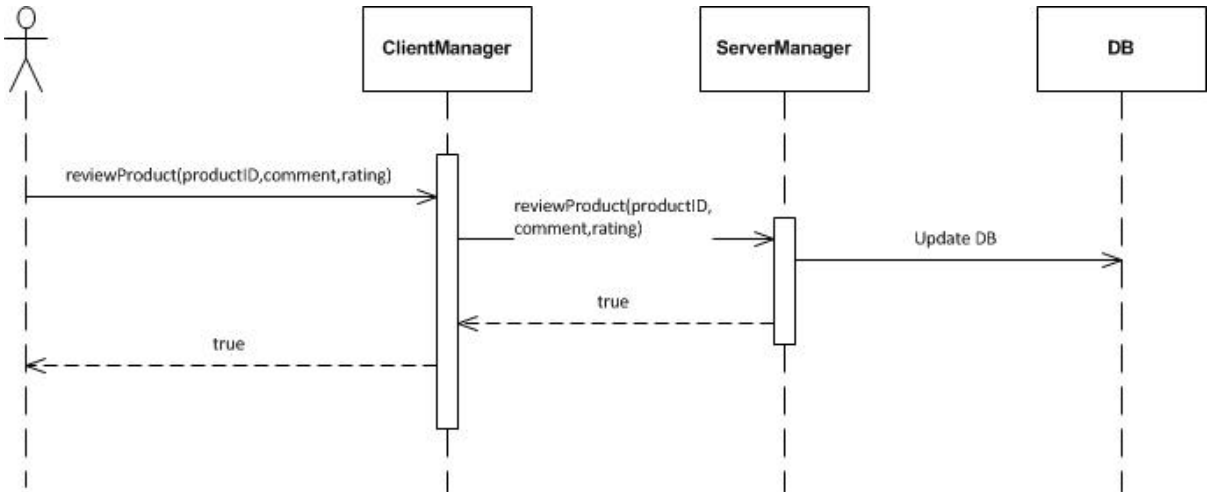

Figure 9: Sequence diagram (3)

This diagram describes how a user could review a product. After the user viewing all the details of a product, he/she clicks on the review product button, which will lead he/she to a review form. The user can submit the review by clicking on confirm button. The event handler will ask ClientManager to pass the review to ServerManger which will update the database with the new review. After it's updated, the GUI will show the user a successful message.

4.1.2 Deployment

Client side application will be developed for Android mobile devices and will require continuous connection to the server for full functionality. On the server side Apache Struts2 web framework will be used for handling client requests for resources. For the database, MySQL will be used because of the simplicity and because all team members are familiar with usage of the database. The goal of Struts is to separate the model (application logic that interacts with a database) from the view and the controller (instance that passes information between view and model). Struts2 provides the controller (a Servlet known as ActionServlet). After the user sends a request to the server for some resource, Servlet filter (Filter Dispatcher) looks the request and then as per the mapping of URL, request is forwarded to appropriate Action Class. Selected action is executed to perform the requested operation. Depending on a request from application, selected action performs an SQL query to database and extracts required data. After data processing, JSON response will be generated and sent to the client. Thereafter, client receives response and processes the received data.

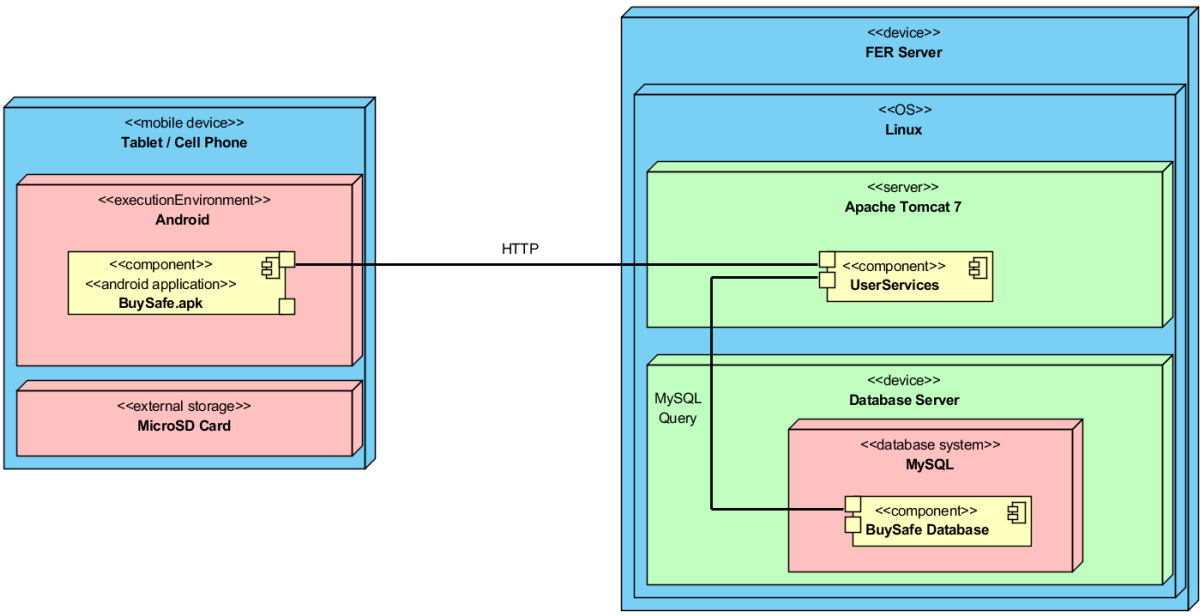

Figure 10: Deployment diagram

4.2 Data flow / Interactions / Dependencies

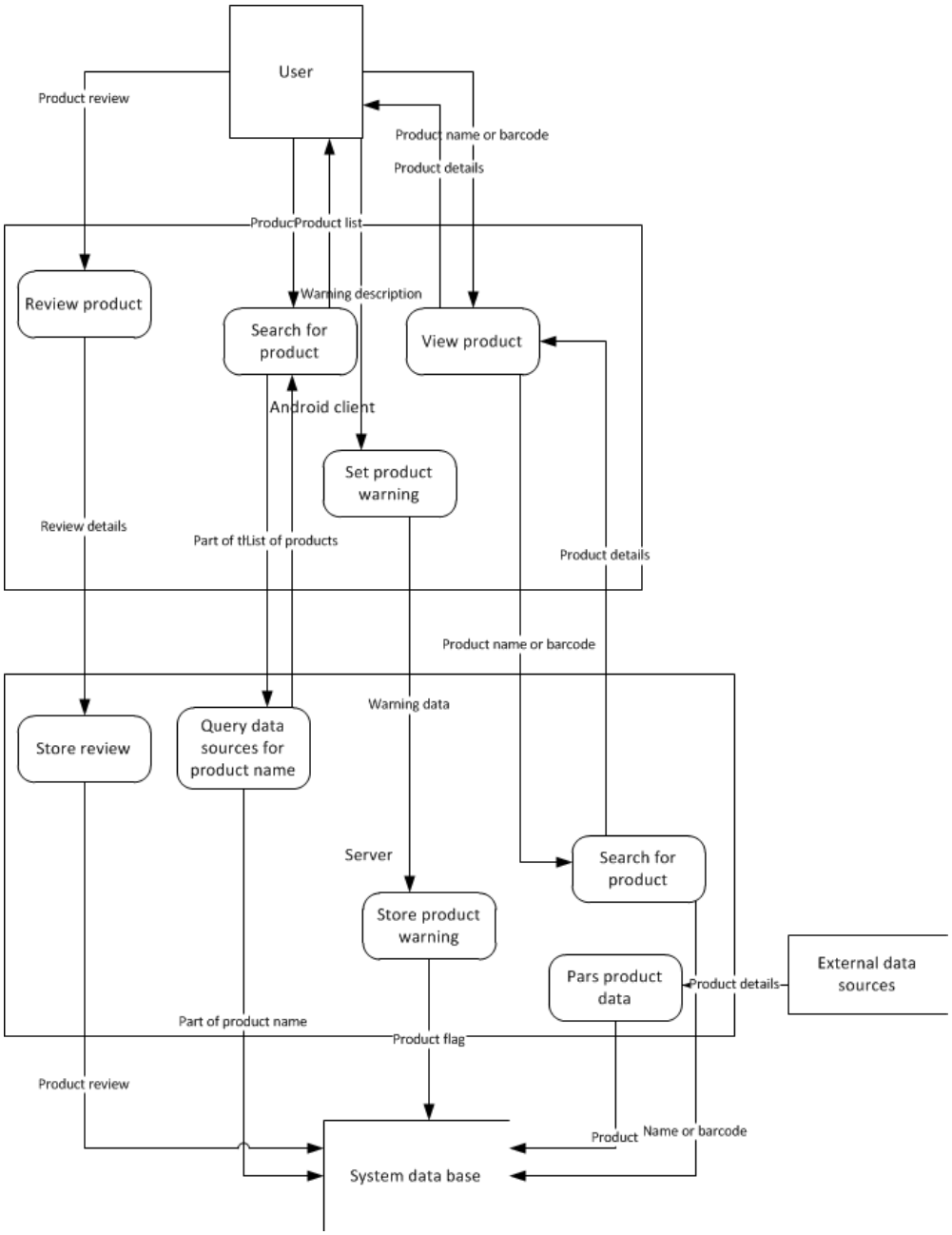

Figure 11: Data flow diagram

For communication mechanisms refer to sequence diagram in section Structured view.

4.3 Data Types / Formats

For the configuration of the server interface we will use an xml document that is configured to the struts2 dtd. The client server communication will be done using JSON objects sent as strings.

4.4 Database Model

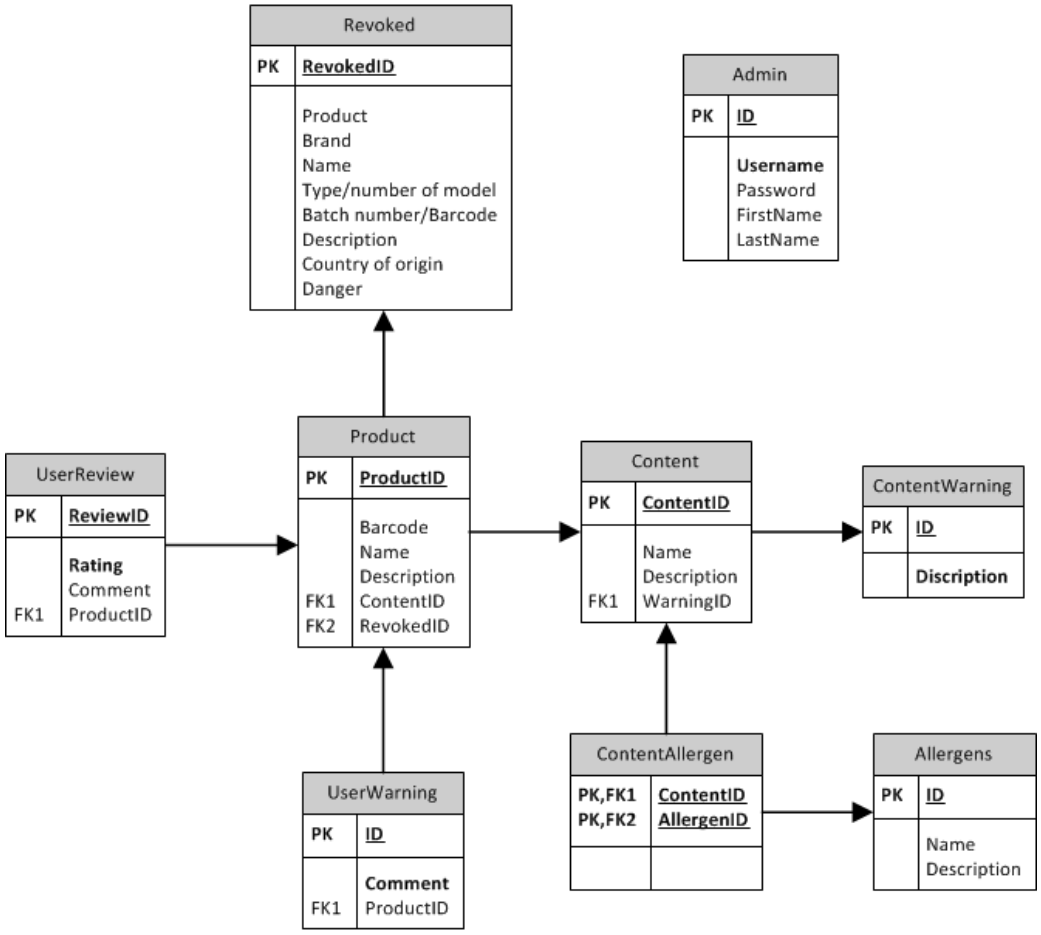

Figure 12: Database model

4.4.1 Database tables

Product describes basic information on the product that the user is researching.
Columns:

ProductID – automatically generated ID of the product
Barcode – the number representation of the product barcode (optional) [indexed]
Name – name representation of the product (optional) [indexed]
Description – short description about the product usually contains country of origin and manufacturer (optional)
ContentID - foreign key relation to the Content table
RevokedID - foreign key relation to Revoked table

Content describes individual content that can be a part of any product.
Columns:

ContentID – automatically generated ID of the content
Name – name of the content (optional)
Description – description of the content (optional)
WarningID – relation to the ContentWarning table that is on that specific content [the warning

describes why the content is dangerous to the public] – if there is no link then the product is not harmful

ContentWarning describes why is the content dangerous or harmful to people.
Columns:

ID - automatically generated ID of the ConetntWarning
Description – describes why is the product content dangerous to people

Allergen describes the allergen that a person can have.
Columns:

ID - automatically generated ID of the allergen
Name – name of the allergen (optional)
Description – short description about the allergen (optional)

ContentAllergen describes a relation between the content that can have problematic impact on a person who has a allergen (like when gluten has an impact on a person that is allergic to gluten)
Columns:

ContentID - relation to the identifier of content
AllergenID - relation to the identifier of allergen
Description – describes the impact of the content on the person with the allergen[like induces through swelling] (optional)

UserReview describes the review of the product given by the user.
Columns:

ReviewID - automatically generated ID of the review
Rating – describes the product quality on a scale of one to five (required)
Comment – describes the user input on the product (optional)
ProductID – relation to the product that the user rated (required)

UserWarning describes the user input on a product that he thinks is dangerous or harmful
Columns:

ID - automatically generated ID of the flag
Comment – description on why is the product harmful (required)
ProductID – relation to the product that is flagged

Revoked describes why and where is the product revoked.
Columns:

RevokedID - automatically generated ID of the revoked
Product - type of product
Brand - brand of the product
Name - name of the product
Type/NumberOfModel - number of model or type
BatchNumber/barcode - batch number or barcode
Description - description of the product
CountryOfOrigin - country in which it is created
Danger - description of the reason why this product is so dangerous

Admin describes details about admin users who have access to the admin page.
Columns:

ID - automatically generated ID of the admin table
Username - username of the admin user
Password - password of the admin user
FirstName - first name of the admin user
LastName - last name of the admin user

4.4.2 Users and permissions

The database will have the following users:

1. Administrator user – user that will handle all conflicts that can arise in the database
2. Parser user- will have read and write permissions on all tables expect Review and UserFlag
3. Server input user – will have read and write permissions on Review and UserFlag tables
4. Server service user – will have read permissions on all tables

4.5 Web site organization

We are not using a web application but an Android native application. The Android App will communicate with the server over SSL encryption. Server interface will be described in the struts.xml document (see 3. Data types and formats). Server methods will be invoked as struts 2 actions with JSON parameters and results will be returned also as JSON object (see 3. Data types and formats). All serve error will be handled on the server. The client will just display the error message to the user.

4.6 Protocols

Messaging:
Communications between the client and server occurs over TCP/IP messaging.

Message Exchange:
The client and a server interact through a sequence of synchronous requests and responses. A client requests a service from the server and blocks itself. The server receives the request, performs the operation, and returns a reply message to the client. The client then resumes its execution. Logically, the clients communicate directly with the servers.

In case of loss of connection, the client will then have a max time of 10 seconds to reconnect to the server. If that time is reached with no success, then a message is displayed informing the user that communication with the server is down.

Error Handling:
We intend on extending exception classes and defining our own subclasses for additional error specificity as and when needed. We also plan on using multiple catch blocks for a single try block, each handling a different type of exception. In case of an error, the Server will return a description to the client to display to the user. When connection between Client and Server is lost, Server will wait 10 seconds till it issues an error message.

4.7 Interfaces to External Systems

Currently no life feeds are used. We gained access to "OPC" but decided not to use it because information provided was in Swedish which required translation that would made our work much more complicated. The kind of information provided by this feed was also far much less than what we could parse from other sources on the internet. We currently depend entirely on parsed info.

4.8 Algorithms

We have no need to specify any routing algorithm or algorithms. Having stated that we will strive to avoid:

- A complex solution to a simple problem.
- A simple, incorrect solution to a complex problem.
- An inappropriate, complex solution to a complex problem.

Our application is a client-server mobile application. Different information about the products are gathered from two websites FoodFacts and Amazon and put into one table in the DB. We have an option of gathering further allergy information from FoodStandards, but that is not necessary currently as we get similar info from FoodFacts.

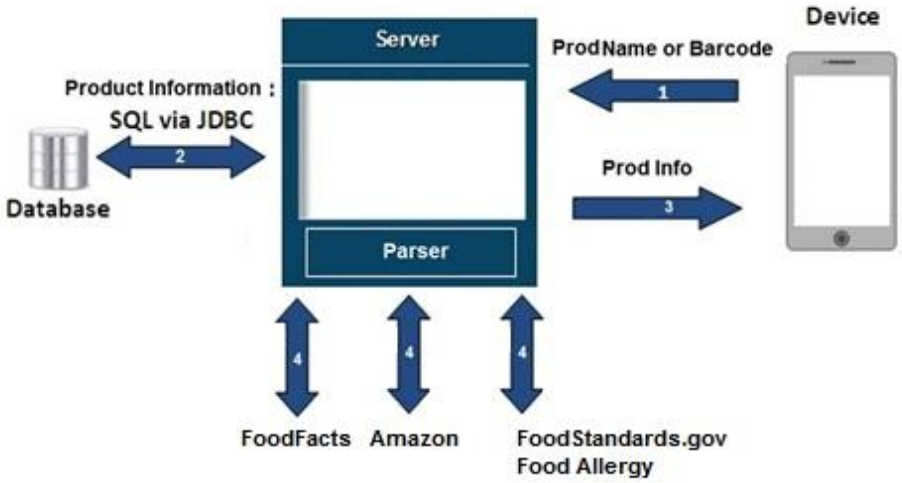

Figure 13: Product information flow

Remark: Step 4 will be performed periodically to update the Database.