Master Thesis

Data Quality Assurance in a Mobile Application for Food Science Data

Author(s):
Birenbaum, Diana

Publication Date:
2016

Permanent Link:
https://doi.org/10.3929/ethz-a-010655396

Rights / License:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.
Data Quality Assurance in a Mobile Application for Food Science Data

Master Thesis

Diana Birenbaum
<dianab@student.ethz.ch>

Prof. Dr. Moira C. Norrie
David Weber, Maria Husmann

Global Information Systems Group
Institute of Information Systems
Department of Computer Science
ETH Zürich

6th April 2016
Abstract

In the modern world data can be shared, transferred and modified across different devices as the same applications are developed for multiple platforms. This creates a necessity to unify the data restrictions and limitations across the components and systems in order to support consistency and high data quality.

This master thesis aims at introducing a way to connect two systems based on their constraints. A novel method of constraint injection is proposed in order to transform and transfer a constraint set from the source system to the target system without influencing its work. This method allows to provide consolidated data validation that helps to support data quality on a high level. The constraint injection approach addresses possible constraints changes by suggesting a reusable solution that allows to react on the updates in the source constraint set.

A new flexible adaptive interface approach is proposed that helps to adjust the target system’s input forms to the constraint set it received. For this purpose constraints are assigned with an importance level that allows to determine in what order the input fields should be displayed to the users.

In order to provide the users with the current data quality level in the system a responsive data quality feedback method is introduced. It suggests to distinguish small screens from large ones and to display the feedback in different ways depending on the screen size.

For evaluating our methods a mobile application in food science field was developed and Web API and constraint injection functionalities were added to the existing food science data management desktop system. Three different ways to give data quality feedback were proposed and implemented, including a responsive data quality feedback solution. These feedback representations were used in a user study for comparison and evaluation together with other aspects of the developed application such as usability, validation error depiction and general user feedback. The application received a positive feedback, and the most favorable ways of presenting data quality feedback were determined.
# Contents

1 Introduction
   1.1 Motivation .................................................. 1
   1.2 Thesis structure ............................................ 2

2 Background and Related Work .................................. 3
   2.1 Data Quality .................................................. 3
   2.2 FoodCASE ..................................................... 4
   2.3 Mobile Applications ......................................... 4
       2.3.1 Data Quality and Data Quality Feedback in Mobile Applications .......... 4
       2.3.2 Adaptive GUI in Mobile Applications ........................................... 6
       2.3.3 Constraints injection .................................................. 6
   2.4 Rule engines ................................................... 7
   2.5 Aspect-oriented programming .................................. 7
   2.6 Work positioning ............................................... 8

3 Approach and architecture ....................................... 9
   3.1 Concept ....................................................... 9
       3.1.1 General description ........................................... 10
       3.1.2 Constraints injection ........................................... 10
       3.1.3 Adaptive user interface .......................................... 12
       3.1.4 Data Quality Feedback responsiveness ............................ 14
       3.1.5 General flow ................................................ 15

4 Implementation .................................................. 17
   4.1 Task .......................................................... 17
   4.2 Choice of instruments ......................................... 18
       4.2.1 Mobile platform .............................................. 18
       4.2.2 Rule engines and AOP ........................................ 20
1

Introduction

1.1 Motivation

In our modern world the amount of global smartphone subscriptions grows every day. By now mobile phones are more popular than desktop computers\(^2\); tablets and other portable electronic devices are also gaining weight. Smartphones became an everyday reality, a crucial and essential part of our days; it shapes new ways of accessing information.

While more and more people prefer to perform banking operations, do online shopping, play games and read books on their smartphones, the applications used on the smartphones become more elaborate and sophisticated. There is a high demand for applications, which can move an every-day routine from desktops to tablets and phones. While this demand continues to grow, a new question emerges: with such a substantial amount of applications moved from the classical desktop representation to the mobile format, in which way will they influence the data that a user is inputing and working on?

The answer to this question cannot be given without taking into account what mobile applications should offer to a user: easiness in operations, comprehensive accessibility and absolute user-friendliness. Although mobile development moves in the direction of a simpler interface and easy interaction, it still should take care of the user data on a sufficient level. This means that the controlling of data quality should be performed and a user should receive feedback about the current condition of the data quality when necessary. The usage of a small and portable device must not influence the data input, correctness and quality.

In this thesis we explored the possibilities, opportunities and difficulties which a mobile application brings in regard to the data quality and data quality feedback. We built a mobile application aimed at user support in food purchases for scientific research in the FoodCASE

\(^2\)http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/
project. FoodCASE is a research project at the ETH Zurich as well as a software for the management of scientific food information. The research project deals with the definition and categorization of quality aspects and their measurement for the practical application. The developed mobile application provides the possibility for inputing a significant amount of data that is later used for the scientific purposes, therefore it is important that the data quality is supported on a high level. We investigated several ideas of data quality feedback representation in the application based on the device size as well as using different visual depictions. We evaluated our ideas with the help of a user study, in which participants had to compare different versions of data quality feedback in the mobile application.

Requirements and constraints can change over time, but a system which is built of several distributed parts should stay overall consistent. This means that synchronization between components is necessary in order to help the whole environment develop evenly. Indeed, when a mobile and desktop application collaborate to achieve a shared goal, it is vital that appropriate constraints are known to both applications, they complement each other and must not have any discrepancies. We developed and implemented a constraints transformation approach which allows to keep the constraints in both applications synchronized. Additionally, we provided an adaptive GUI in the mobile application based on this constraints synchronization.

In summary, this work proposes an approach of constraints injection from a desktop application to a mobile application in order to keep the appropriate constraints synchronized without having the developer taking care of it. This method can be used not only for mobile development, but also for connecting different parts of a system through a set of constraints. Further, we present a GUI for a mobile application which adapts itself to this synchronized constraints set and provides responsive feedback about the constraints compliance respectively data quality.

1.2 Thesis structure

Chapter 2 contains a background overview of the data quality terminology and related basic concepts. It is followed by an introduction to the FoodCASE system. Afterwards, some chosen work in the data quality field, validation methods and constraints injection, user feedback and adaptive GUI in mobile applications are described.

In Chapter 3 main concepts and approaches developed in this thesis are presented. It includes constraints injection, adaptive data input and data quality feedback responsiveness.

In Chapter 4 the implementation of the mobile application and its connection with the existing FoodCASE system are described in detail. It discusses the chosen technologies and solutions.

In Chapter 5 evaluation results are introduced, based on the user study performed with participants of various backgrounds.

In Chapter 6 contributions and shortcomings of this work are discussed. We suggest ways to extend our mobile application.

Chapter 7 includes a summary of this work and proposes directions for the future.
2 Background and Related Work

This chapter contains a background research summary related to this thesis. In the beginning it discusses the notion of data quality and basics of the FoodCASE project.

Afterwards, the chapter continues with a detailed look into data quality solutions and constraints usage, adaptive GUI and feedback visualizations in mobile applications. Next, the chapter briefly introduces rule engines and aspect-oriented programming. In the end, the positioning of this thesis is discussed.

2.1 Data Quality

The notion of data is not complete without a notion of data quality. Data is present everywhere: from a personal mobile phone with a list of contacts to a highly sophisticated enterprise information system or a research database. Each time data is input, transferred, modified or deleted, the question of data quality arises: Without proper control the outcome can be disastrous for a person, enterprise and research.

There are different ways to define data quality. In [15] it is suggested to describe data as having high quality if this data meets a user’s needs. At the same time, it is necessary to admit that this definition makes data quality very subjective in one’s eyes and therefore lacking a strict common ground for all users and needs.

There is a way to define data quality more formally. The data is of high quality if it possesses the needed features and is free from defects, according to a rethought definition in [15]. This is still vague and provides a lot of space for debate; a more precise and detailed view on data quality should be found.

Fan and Geerts [5] look at data quality from a database management perspective. They suggest to see data quality as a combination of several components or issues:
• Data consistency: the validity and integrity of data representing real-world entities;
• Data deduplication: identification of tuples in one or more relations that refer to the
same real-world entity;
• Data accuracy: the closeness of values in a database to the true values of the entities
that the data in the database represent;
• Information completeness: whether one database has complete information to answer
queries;
• Data currency: identification of the current values of entities represented by tuples in a
database, and answering queries with the current values.

Although data quality is discussed here from the database perspective, it gives us a rigorous
explanation and is enough for the focus of this thesis.

2.2 FoodCASE

In this section we discuss FoodCASE, which was used for our constraints injection approach
as well as mobile application implementation.

FoodCASE (Food Composition And System Environment) is a food science data manage-
ment system. It was developed at ETH Zurich and is used for the input, administration and
evaluation of food values, such as nutrients and contaminants. It includes information about
various foods from the different participating countries. Its records encompass the nutrient
and contaminant content of the food in order to facilitate calculating the compositional, nutri-
tional and contaminative values. In this thesis we focus on one particular part of FoodCASE:
TDS (Total Diet Studies). This component is responsible for the contaminant analysis and
data storage and is used in the European Project Total Diet Study Exposure (TDS-Exposure)\textsuperscript{1}.

2.3 Mobile Applications

The goal of this section is to discuss existing work in data quality solutions and constraints
definition, data quality feedback and adaptive GUI for mobile applications.

Mobile applications represent one of the most booming fields in computer science. Although
mobile development is very popular, some issues that can be found only in mobile applic-
ations are still here to be addressed. Furthermore, mobile applications are characterized by
features which can introduce a whole new perspective for investigations: They have small
screens, limited battery life, reduced performance capacities and they are always with a user.

2.3.1 Data Quality and Data Quality Feedback in Mobile Applications

Data quality is a very broad field and there are various ways to understand this term. For
example, if a constraint requires that an input field corresponding to a food item name is

\textsuperscript{1}http://tds-exposure.eu/
not empty, adding the food name might not be enough for good data quality: If the name ‘Apple’ is added, but the item is in fact an orange, it is an obvious mistake that leads to low data quality. In this thesis we see data quality as a constraints problem or constraints compliance problem, which means that data quality is implemented only by constraints validation.

The topic of data quality is broadly discussed both in industry and research. Nevertheless, there are only few works which investigate data quality or data quality related issues in mobile applications.

In [4] the authors propose a novel framework for providing Quality of Information evaluation. They investigated how their approach can be used for image applications in constrained networks. Their framework contains several features such as timeliness, accuracy and precision, which can be mapped to data quality attributes. The authors emphasize that they distinguish between information and data notions, but every information feature can be defined through the data attributes and therefore represented using data quality notions. This method of separating the data (an image) from information which this data contains (e.g. number of people on the image) is valuable, as it suggests that high data quality automatically implies the ability to extract useful and meaningful information.

One of the most popular application domains for data quality research is medicine and medical records. Medicine and human’s health put a high demand for top-quality data. De La Harpe [7] compares the data quality of paper care data versus mobile care data for medical records from a semiotic point of view. This creates a lot of features of data quality which can be taken into account. The author evaluates several parameters, which are important for medical data quality and comes to the conclusion that mobile health solutions show promise, although only if all features of data quality as well as the human involvement are considered.

There are different ways to adapt a data quality framework for mobile devices. But apart from tracking data quality, it is important to inform users about some changes or notify about improvements that they can perform. For a mobile device, which possesses some very special characteristics mentioned earlier, it opens an interesting field for examination. In [13] the authors investigate possibilities for a participatory sensing using mobile devices and develop a framework for work evaluation. This framework does not correspond entirely to the classical data quality attributes, but offers an interesting insight into the possibilities of the feedback given to users: Prompts, positive reinforcement, adaptive interfaces and social validation. These methods use not only already mentioned characteristics of the mobile devices, but also exploit the social behavior of mobile users, encouraging them to interact with other people or be more active.

Another work [16] discusses the quality of spatial data in GIS applications. Data quality in GIS applications plays a very important role, as user experience and precision of the presented information fully depend on the quality of data; these applications require a very high level of data checks and constant support of high quality of data. Poor service may significantly decrease the number or suitableness of offered services. The authors address visualization possibilities for data quality assessment, as it is a convenient and easy way for the user to understand the current data quality level. They emphasize that the visualization should be easy enough for understanding for a person not specialized in GIS applications. Indeed, the usage of an application does not necessarily imply how professional users are in that field.

A more general answer to the question how to give feedback on data quality in sophisticated
2.3 MOBILE APPLICATIONS

information systems is proposed by [17]. They introduce two types of feedback: Direct feedback on individual entities during data input and feedback from an analysis of the system’s database. The authors also suggest to distinguish between different levels of problem severity, and informing users about it with different colors; textual feedback is given in a single location below the input area of a system.

2.3.2 Adaptive GUI in Mobile Applications

There is a lot of research done in the area of adaptive interfaces addressing multiple domains and various needs. In [13] the authors suggest an adaptive interface among other approaches as a tool for passive persuasion of participants. They hope to make feedback continually effective through the usage of an adaptive interface. Other examples include [2], [6], [9] and [8].

A specific analysis for small screens was performed in [12]. The authors focused on using ideas from rule-based programming. It allowed them to match several transformation rules for any source pattern and to use conflict resolution to determine which rules to apply. The presented approach includes the usage of several models, subsequently transforming one to another. Various proposed rules match different interface parts; at the end of matching, all parts are attempted to fit a predefined screen size. The authors suggest to take into account three main optimization objectives: maximum use of the available space, minimum amount of navigation clicks and minimum scrolling.

Nakamura et al. [10] investigated the adaptive user interface of a personalized transportation system. They suggest to build an interface based on the usual routes that users have and extend it when a new route is added. The authors show that the usage of an adaptive interface decreases click costs and time costs in comparison with two other already existing solutions.

At the same time, in [14] a more general method to the adaptive interface is investigated. The authors propose a model-based approach, which allows to adapt the user interface at runtime to numerous (also unknown) contexts-of-use; the system dynamically handles context information from the environment and uses it for interface generation. This solution combines a user interface runtime architecture and runtime user interface models.

2.3.3 Constraints injection

One of the significant contributions in the topic of constraint transformations is made through investigating model transformations. In [3] a method for model transformations is proposed that allows to carry out construction of new models based on constraints in an existing model. A proposed approach uses generated constraints, written in the Object Constraint Language (OCL), to guide designers in creating new models, rather than suggesting steps to perform direct model transformation. This method helps to reveal inconsistencies, which can be corrected later.

An algebraic approach to data schema transformation in [1] works closely with constraints modifications and transformations. The authors suggest to use enhanced data refinement laws to formalize the propagation and introduction of constraints during schema transformation. This allows not only to perform a desired data schema modification, but also to preserve or
modify constraints in accordance with a new data schema. With the help of this approach an example of data schema mapping from XML to SQL is completed.

At the same time, the work in [11] focuses on database constraints and investigates how they can be modeled platform-independently and later generated to a suitable code. The authors suggest two steps of constraints transformation: from model-independent representation through relational model to the SQL/DDL code for several popular database platforms. It is stated that this approach provides better maintainability and portability.

### 2.4 Rule engines

One of the ways to represent and work with constraints is with help of a rule engine. The rule engine is a system that uses rules in any form, that can be applied to data to produce results, for instance, to validate input forms. Usually the rule engine has one or more sets of rules, against which the given facts are compared. Examples of such rule engines are Java-based Drools\(^1\) and JavaScript-based nools\(^2\). Drools is a Business Rules Management System (BRMS) solution and provides a business rules engine, a web authoring and rules management application and an Eclipse IDE plugin for core development. Nools is a simple rule engine aimed at JavaScript development and can be used directly in browsers or with Node.js\(^3\).

There are several advantages in using a rule engine for work with constraints:

- A rule engine separates logic (rules) from data (objects). When rules are changed over time, it does not influence the objects;
- All knowledge can be gathered in one place;
- The declarative nature of rules is easy to understand.

### 2.5 Aspect-oriented programming

Another approach to constraints representation and integration is the usage of aspect-oriented programming (AOP). AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It allows to add additional behavior to existing code without modifying the code itself, instead separately specifying which code is modified via a 'point-cut' specification. This feature is very suitable for the constraints implementation, because constraints can be treated as this additional behavior, therefore they can be added, modified and deleted without influencing the main flow of a system.

The basic terminology of AOP includes the following concepts:

- Cross-cutting concerns: A functionality that is identical among several models (or classes) but cannot be clearly separated from these models;

\(^{1}\)http://www.drools.org/
\(^{2}\)https://github.com/C2FO/nools
\(^{3}\)https://nodejs.org/en/
• Advice: An additional piece of code that should be applied to the existing model;
• Join point: A point during the execution of a program, such as the execution of a method or the handling of an exception;
• Point-cut: A predicate that matches join points;
• Aspect: A combination of the point-cut and the advice.

There are different types of advices, depending on the time of execution. Example of the aspect-oriented programming modules are AspectJ\(^1\) or Spring AOP\(^2\).

2.6 Work positioning

To the best of our knowledge, there is no work done in the field of constraints injection. Constraints transformation is used for various purposes in multiple works, both as an investigation goal and as a side-problem for main topics. Although constraints formats can change from one application to another, there are no works that specifically address the process of transforming constraints from one representation to another and adding them into an application, or performing constraints injection. One goal of this thesis is to explore this topic and to develop an approach, which allows to transform constraints to a new format and directly insert them in a working system.

Another goal of this thesis is to explore opportunities that an adaptive GUI provides in regard to constraints: We developed a method which would allow to adapt user interfaces to a given set of constraints. Furthermore, we looked into different ways to give data quality feedback and instruments to motivate users to support high quality data. Both questions are important for mobile devices due to their characteristics and were only partly addressed before.

\(^1\)http://projects.eclipse.org/projects/tools.aspectj
\(^2\)http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html
In this chapter we will look in detail into the structure of the suggested approaches and methods, specifically into three main parts: constraints injection, adaptive interface and data quality feedback responsiveness.

3.1 Concept

One of the main goals of this thesis is to investigate data quality representation and feedback in mobile applications. But it is impossible to evaluate data quality without knowing what parameters determine the level of data quality; in other words, we need data constraints for defining limitations and restrictions and helping to measure data quality. Resulting data quality levels can be presented to users through a data quality feedback, which will be described in Section 3.1.3.

This thesis explores not only the data quality issues in mobile applications, but also addresses the problem of unified constraints for multiple applications. In order to ease constraints synchronization between different applications we propose a new concept of constraints injection in Section 3.1.2. Following the need for possible change of constraints over time, we suggest an adaptive GUI solution in Section 3.1.4 based on the current set of constraints.

The methods and approaches described in this chapter can be adapted and implemented for different systems. In case an example is needed, the FoodCASE desktop or mobile applications will be used.

We start with a broad description of the concept, discussing its general overview and structure, and then focusing on the separate parts.
3.1.1 General description

Our approach includes several main components: constraints injection, adaptive user interface and data quality feedback. All these parts are tied together to interact within one environment, which is created by the common and most important part of the system: constraints.

There are two types of systems that we distinguish: a source system and a target system. The source system includes an original set of constraints, which is created specifically for this system, and modified and stored independently. The source system is not limited to a desktop implementation. It possesses a constraints set that is modified and prepared for the target system by performing constraints injection.

The target system can be represented by another application or a component. It also uses constraints, but possibly in a modified version that is received from the source system after constraints injection. This new adjusted constraints set is used for two purposes within the target system:

- As a usual set of constraints, for supporting data quality and regulating input. Based on this set the data quality level is determined, and feedback is created and presented to users;
- For adaptive interface generation.

The format, which is used to give users data quality feedback, is chosen based on the device size.

3.1.2 Constraints injection

Constraints are vastly used in modern information systems in order to limit possible data formats, control input and take care of adhering to a defined data schema. There are multiple ways to define constraints: With different levels, addressing various data types, platform-independently, with help of various languages and models.

A key idea behind our constraints injection method is to enable transformations of constraints into a new representation, suitable for the target system, and insert it into this system without influencing its work.

In Fig. 1 you can see a structure of our constraints injection method. The source system on the left in Fig. 1 contains a set of constraints that might be divided into groups based on their importance. A triangle colored from red to green on the left side of the source system represents the different levels of importance of these constraints. You can see an example in Fig. 2: in our FoodCASE application each food item should have a name, so the constraint ‘Name is present’ is highly important and marked in red color; another strict rule is that the food weight should be more than 100 grams, so this also has high priority and is again colored in red. On the other hand, the origin of this food item is only of medium importance; this means that users are not forced to have this information, but it is still recommended, and this constraint is marked in orange color. The approach does not limit the number of such groups.
The original set can contain additional information apart from importance such as type of constraint (text, number, etc.). For our example it means that name and origin are of type text and weight of type number, this information is marked after a field’s name in Fig. 2. This constraints set is the original set, and all subsequent operations and modifications are performed based on it.

The next step in the constraints injection process is to generate a set of constraints for a target system, that is represented by an arrow from the source system to the target in the center of Fig. 1. This generation process includes adaptation, modification and transformation of the original set, based on the requirements and target system implementation. All additional information or only some part of it can be forwarded to the modified set. The generation process can utilize this information in different ways, for instance to extract only constraints of some type or certain importance level. Generally, the constraint set can contain as many details as is necessary for the given purposes. In our example with three different constraints, we use information about their importance to generate two separate files: The first for the highly important constraints (name and weight, in red color in Fig. 2) and the second for
other (origin, in orange color in Fig. 2). We do not keep information about their importance anymore, but we keep the types of constraints. The number of generated files should correspond to the number of importance groups that does not have a limit and can be more than two. The result is used by the target system, which is seen on the right in Fig. 1. The approach assumes that this constraints set is stored in a way that does not influence the stability of the system’s work in case it is modified; for example it is stored in a separate file.

An important feature of the synchronized generation is the ability to re-render the set in case the original one was changed, meaning that this generation is reusable. This aspect of our approach is represented in Fig. 1 by two circling arrows in the middle at the bottom of the scheme. This can be achieved either manually by running the generation process again, or automatically upon receiving an update in the original constraint set. For example, if a new highly important constraint ‘Brand Name is present’ is added to our example set, we can generate the target set again and the result will contain all four constraints.

### 3.1.3 Adaptive user interface

The constraints set can change over time. An application that displays a considerable amount of detailed information and contains a lot of input fields might fail with a new set of constraints: The mapping between constraints and input can get broken, introducing challenges and creating errors. There are four possible transformation cases that might happen with a constraint set:

- A constraint importance might change;
- A constraint might be deleted;
- A constraint might be added;
- A constraint itself might be changed.

For example, if the target system’s constraints set contains a medium-importance rule about an origin of a food item, after some time the situation can change and the origin can become strictly important as name and weight were before. For the user interface it implies that the error representation and system behavior now have to be changed. Otherwise, users who do not input a food origin will still be facing a warning that they can ignore, or in even worse situation, no information at all.

As a result, constant adjustment is needed. To avoid performing manual code modification, we suggest to build the user interface based on the set of constraints which that application is using. In Fig. 3 an outline for the adaptive interface generation method is displayed.

We suggest to use the constraint set to build convenient interfaces for data input. This part of an interface is the most dependent on the constraints, because users make mistakes and might not know restrictions which are imposed on data. Therefore, they need to receive immediate feedback in the form of errors and warnings if their input does not satisfy existing constraints in the system.
Our adaptive interface generation works as follows. For the start, there must be a constraint set present, which is shown on the left of Fig. 3. During the first step the target system reads a new set of constraints and extracts information about them: separation into types, groups and possible features. This process is represented by an arrow from the constraints set towards the inside of target system in Fig. 3. After the reading process is finished, the target system generates an interface that will be presented to users (the arrow inside the target system in Fig. 3). For this purpose constraints that were read before are used. Afterwards, the resulting interface can be demonstrated to users.

Constraints describe various aspects of a system. They define if data is mandatory or not and which logical, numerical or textual restrictions are imposed on this data. In our example with three constraints the ‘Name is present’ constraint describes that this parameter must exist; at the same time ‘Weight \( \geq 100 \) grams’ constraint does not declare if weight must be present, but specifies that if so, its value must be equal or over 100 grams. For our approach it is important to look into the details of a special case - mandatory and non-mandatory input constraints. This is not a necessary requirement, but the presence of such a separation can facilitate interface generation in the target system, especially for data input. This new constraint set is presented schematically in Fig. 4.
When an application contains a significant amount of data input, it is important to guide users through it. Our approach helps users to distinguish between very important and less important data by generating an interface for mandatory input constraints first. For example, if the presence of some data field such as the name of a food item is required, it can be offered to the users first. This solution can be effective when users are in a hurry and have time to add only the most important information. Depending on the case, another design of using importance information can be elaborated.

For this special case we propose to mark constraints in the source system as requiring a mandatory input (and therefore checking for the presence of this input) and less important input constraints. In Fig. 4 it is displayed as additional information boxes on the very right of each constraint. This example demonstrates a separation into two categories ‘Mandatory input’ and ‘Non-mandatory input’, but our concept does not impose any restrictions on the number of groups. The step of extraction does not differ for this special case, but the generation suggests that we first evaluate if mandatory input constraints are present. If so, they are used to generate an interface for mandatory input, and only after that interfaces for other input fields depending on their importance level group. This method makes sure that users do not skip input, which is required by the system.

For instance, a food item name will appear for users first, and only after successfully adding a value for it a user will be guided to the next step with weight and origin of the item. This helps to focus on main data input fields and in case of time shortage or other reasons skip parts which are not highly important to the system.

An important case of such data separation is data fields that belong together but received different importance levels. For instance, it can be a price and a currency: They are linked together by their meaning, but the price is more important because the currency can be presumably derived based on the shop place. In such cases our approach suggests to adapt the interface in a way that a less important field is pushed ‘up’ to the category of the more important linked field.

### 3.1.4 Data Quality Feedback responsiveness

As it was already stated in Section 2.3.1, data quality in this thesis is understood as *constraint compliance*. But just the presence of constraints in a system is not enough to reach high data quality. It is crucial to inform users about the data quality state and stimulate them to improve it. In order to achieve it, we suggest to use data quality feedback enhanced with the capability of adapting to the device size.

If we already adapt an interface based on the set of constraints, we can take into account the device size as well. This idea contributes to the users’ convenience, as we utilize the screen size as much as possible.

Our approach divides screens of mobile devices into two types: small and big, as depicted in Fig. 5. The first group (on the left in Fig. 5) encompasses tablets, or big screen devices. The second group (on the right in Fig. 5) contains devices with a small screen size and usually consists of mobile phones. This separation can be determined depending on an application domain, because different content, mission and design of applications influence significantly the way a feedback can be given.
For small devices we employ left and right panels that are revealed on left and right swipe gestures. Therefore, during usual interaction with an application users can use the full screen size and refer to data quality feedback upon need by simply opening either of the side panels. In case a more detailed feedback is needed, it is possible to navigate using a button to a separate view with detailed statistics. This is made based on the assumption that not all users need the detailed feedback for their work.

For big devices a part of the main screen can be used. Representation of data quality feedback in both versions might differ, depending on the type of feedback. For example, information can be presented in plots, tables or both.

For both types of devices we suggest to give validation feedback directly on the data input fields. This can help to make an input process quicker, because users see possible errors and warnings immediately.

### 3.1.5 General flow

In Fig. 6 the overall concept is depicted, which integrate the three approaches: constraint injection, adaptive user interface and data quality feedback responsiveness.
In Fig. 6 there are two interacting systems: the source system and the target system. The source constraint set (on the left) is modified and prepared for the target system. Then a constraints injection process that is displayed as an arrow between two systems is performed. The target system receives its target constraint set. It is used to generate an adaptive interface (the upper right corner in the target system in Fig. 6) and to validate data, which allows to give feedback about data quality. This feedback is designed in a responsive way and depends on the device size (displayed in the bottom of the target system).

Our three introduced approaches, when integrated together, provide the possibility for developing a flexible system that can react quickly on any changes applied to the constraint set. It allows to reduce the amount of adjustments required each time the constraint set is altered. The responsive feedback heeds the device size, therefore a necessity to tune the solution manually depending on the device is eliminated.
In this chapter we discuss the implementation of our methods, based on the FoodCASE application. We explain the task, choice of instruments and look into details of implementation such as architecture, classes and methods.

4.1 Task

In Section 2.2 we described the FoodCASE project and its part Total Diet Studies, which is a key component of this thesis. An important part of TDS is purchasing food items in shops. Food samples should be bought according to shopping lists in multiple shops in different places. All foods must be tracked and information about it must be noted. After that all items are collected in a laboratory for performing analysis and adding information to the TDS database.

The process of purchase is long and includes tracking a significant amount of information, as items must be bought according to their precise descriptions. A mobile application can significantly improve and facilitate this process.

One of the main requirements for this mobile application is a connection and synchronization with the existing desktop application. Our goal was to implement approaches that we had developed earlier and investigate, with which difficulties and challenges users can be faced.

In summary, the mobile application had to fulfill the following requirements:

- It should be connected to the desktop application through a Web API;
- It should implement the pre-defined business processes;
- It should be cross-platform, in other words to work on multiple mobile operational systems;
• It should implement data quality measurements and provide data quality feedback;
• It should utilize constraints.

4.2 Choice of instruments

Before the process of implementation started, several instruments for the development had to be chosen:

• Mobile development platform;
• Constraining specification framework or library in order to separate constraints from the application itself;
• Database system for persistent storage in our application.

4.2.1 Mobile platform

In the modern mobile world several different mobile operating system exists, each of them requiring different languages and technologies for implementation. To overcome the problem of possible repetitive implementation of the same business logic, it was decided to work with a framework for cross-platform mobile development.

There were several cross-platform frameworks available at the time of this thesis. Among them were:

• PhoneGap\(^1\)
• Sencha\(^2\)
• Appcelerator\(^3\)
• Crosswalk\(^4\)

We had to eliminate Sencha and Appcelerator as they are not open-source and free of charge platforms. The choice between PhoneGap and Crosswalk was made in favor of the former, as it provided a bigger community and more plugins were available.

PhoneGap is the original distribution of Apache Cordova\(^5\), an open-source mobile development framework, and it allows to create a mobile app using HTML, CSS and JavaScript.

Any cross-platform mobile development tries to recreate the native user interface experience. It is a goal achieved with difficulty, because the instruments that cross-platform development utilize consist of web-technologies, not native tools. To overcome this problem, the technologies used for creation of user interfaces should be chosen carefully.

\(^1\)http://phonegap.com/
\(^2\)https://www.sencha.com/
\(^3\)http://www.appcelerator.com/
\(^4\)https://crosswalk-project.org/
\(^5\)https://cordova.apache.org/
One of the most common and popular solutions for implementation of user interfaces is jQuery Mobile\(^1\). Initially it was used for web mobile development, but recently found its application in cross-platform development as well. The advantages that jQuery Mobile offers are a stable release, large community and a broad set of features and possibilities. Unfortunately, it also has disadvantages, and one of the main is a very old last release - at the moment of this thesis it was dated by October 2014, that is more then a year ago. In the world of the mobile development, where devices and technologies evolve constantly and incredibly quickly, this time period creates a significant gap in the level of advancement, so we had to take that fact into account.

A prototype of a mobile application was developed in order to evaluate potential difficulties in our future work and to examine a possible outlook. The result showed to be stable in work although it did not feel like a native application. Bringing an application to a native look would require a very long and thorough work with CSS and possibly with additional frameworks. The jQuery interface version is presented with Fig. 7a and Fig. 8a.

\[\text{(a) jQuery Mobile} \quad \text{(b) Ionic Framework}\]

Figure 7: Start page in prototypes

In order not to limit ourselves with the jQuery Mobile option, tools and frameworks that Apache Cordova is connected with were explored. We discovered the Ionic Framework\(^2\), an HTML5 SDK that helps to build native-feeling mobile applications using web technologies like HTML, CSS, and JavaScript. Ionic is built using Cordova and AngularJS\(^3\). Although this framework was only recently developed and introduced to developers, it has already quite a big and active community and is regularly updated. Comparably high performance characteristics and active usage of AngularJS helped to make a decision in favor of the Ionic Framework. It was decided to try this framework and compare the results with our jQuery Mobile prototype. The prototype which was built with Ionic demonstrated superb

\(^1\)https://jquerymobile.com/
\(^2\)http://ionicframework.com/
\(^3\)https://angularjs.org/
user experience; for the same amount of functionality less of code had to be written. The Ionic interface version is presented in Fig. 7b and Fig. 8b. Our final conclusion was to use PhoneGap and Ionic.

4.2.2 Rule engines and AOP

One of the choices that we had to make before starting the implementation process was which framework or library for constraints should be used. In order to understand our motivation, let us have a look at the facts:

- We use a cross-platform development framework, which means that we operate only with JavaScript, HTML and CSS;

- We assume that users might not be connected to the Internet all the time during application usage; they are rather off-line almost all the time with no connection to the main database;

- We needed to have a light weight solution, as mobile development puts several restrictions on the size of the application package, amount of memory and the duration of battery life.

As our implementation was going to follow our developed methods as constraints injection as described in Section 3.1.2, it was vital for us to find a solution that satisfies several requirements. First, as we want to synchronize constraints and be able to change them over time independently from our application, an implementation solution should offer a way to easily
de-couple the constraints definitions from the source code of an application and if possible be stored even in a separate file. Second, we were inspired by aspect-oriented programming and wanted to have a solution which would give opportunities for extending in a way of aspects. This means that our solutions should be flexible and offer a broad functionality or at least have possibilities for extending. And third, it was important to adopt a framework or library which would be well-supported, have extensive documentation and examples, offer a stable version and be easy to debug; we decided that an open-source solution would be the best.

There are two different approaches that were taken into consideration: aspect-oriented programming (AOP) and rule engines. AOP can be successfully used for integrating constraints: By using point-cuts we can define when and where constraints are checked. A rule-based system can also be used for constraints integration and support; this system will be using constraints as a set of rules, which are checked and an action is performed depending on if a rule is satisfied.

We started from exploring current solutions in these two fields. All solutions which are difficult to integrate with our mobile application were put aside, for example, AspectJ\(^\text{1}\). Although it has a lot of advantages such as popularity, good documentation and features themselves, for our thesis it had one and only drawback: it is a Java-based solution. While in case of a web mobile application with constant Internet connection it is a possible option, our case did not offer an opportunity for that: No proper Internet connection does not allow to use AspectJ as a server-side instrument. An option to embed it in the application itself was also not suitable: cross-platform mobile development means it should run on all platforms. The presence and characteristics of the iOS operating system limited our choice because of the different programming languages required: AspectJ is a Java-based solution and iOS applications are Swift\(^\text{2}\) or Objective-C\(^\text{3}\) based.

It was decided to have a look at libraries and frameworks which were made to use with JavaScript: js-aspect\(^\text{4}\), noools\(^\text{5}\) and dojo\(^\text{6}\).

- js-aspect: This aspect-oriented framework provides simple but powerful functionality. On the downside, all code responsible for cross-cutting concerns should be integrated within application code, therefore any changes should be made in the application code itself. Moreover, by offering wide opportunities this framework also requires to define many things manually.

- noools: This rule engine includes several features that in our case are very important: Rules in noools can be defined in a separate file using the noools language and have almost no limitations in their sophistication; rules can be changed independently from an application, as they can be compiled and used on-the-go; vast functionality is available. On the other side, noools does not contain predefined pre- and post-constructions for specifying conditions.

\(^1\)https://eclipse.org/aspectj/
\(^2\)https://developer.apple.com/swift/
\(^4\)https://github.com/antivanov/jsAspect
\(^5\)https://github.com/C2FO/nools
\(^6\)https://dojotoolkit.org/reference-guide/1.10/dojo/aspect.html
• dojo: This toolkit contains a module, named dojo/aspect, which provides aspect oriented programming facilities to attach additional functionality to existing methods. Although it contains some predefined useful methods, it has several drawbacks in regard to this thesis’ implementation: All aspects code should be integrated within an application and it also lacks some important functionality.

With each of the three frameworks a small application version was developed and results were compared. The nools rule engine fitted our requirements the best and it was decided to use it.

4.2.3 Database

According to the requirements, our application had to be able to store data persistently. This data included photos of food items, data received from the server and input data. Modern mobile phones and tablets have high quality cameras that produce photos with a high resolution, resulting in big picture sizes. Therefore, a data storage solution for big data sizes was required.

Several different solutions for data storage in our application were investigated. First, as a PhoneGap application is in fact a HTML5 application and therefore runs in the browser, browser-based approaches were checked. They included Local Storage, Web SQL and IndexedDB that are described in the PhoneGap documentation\(^1\). Unfortunately, neither of them were suitable for our purposes:

• Local Storage: a simple solution, with an API providing synchronous key/value pair storage, is available in underlying WebView implementations; in other words, it stores data in the user’s browser. Unfortunately, its upper limit is 5MB of data.

• Web SQL: This API is available in the underlying WebView and supports Android, iOS and BlackBerry operational systems. Unfortunately, this API was deprecated, although it is still supported on iOS and Android browsers.

• IndexedDB: This API is a NoSQL solution such as a key-value storage. Unfortunately, supported mobile platforms are limited to BlackBerry and Windows Phone.

The next step was to evaluate existing mobile database solutions. As a PhoneGap application does not have access to built-in phone capabilities as storage, we had to consider only databases, to which we could connect through the usage of PhoneGap plugins.

There are currently three major database models: relational (RDB), object-oriented and NoSQL.

We reviewed several different database systems.

• Relational database systems:

  – Realm\(^2\): This database system has a dual nature: although mostly it uses the RDB approach, it also supports storing data in a format of JSON files. It supports

\(^1\)http://docs.phonegap.com/en/3.6.0/cordova_storage_storage.md.html
\(^2\)https://realm.io/
various conditions which are equal to constraints. On the down side, this database system is still under development and it does not support the NULL notation yet. Realm only supports native development and at the moment of this thesis was not available for work with Phonegap.

- SQLite\(^1\): This wide-spread database system offers support for iOS and Android operational systems through a special plugin. The storage capacity upper limit is about 140 terabytes\(^2\), which was enough for our purposes. Moreover, SQLite has a large developer community, detailed documentation and stable releases. The plugin needed for supporting SQLite in a PhoneGap application receives regular updates, is open-sourced and includes more than 20 contributors.

  • Object-oriented database systems:

    - ObjectDB\(^3\): although this database system provides a wide range of features, it lacks compatibility with PhoneGap: it requires an environment with Java SE support and at the moment of this thesis no solutions to integrate it were provided;

    - db4o\(^4\): This database system has such advantages as easy embedding and supporting a data schema evolution. On the other hand, it does not offer any solution to integrate it with PhoneGap.

  • NoSQL:

    - BerkeleyDB\(^5\): This database system is in fact a key-value store. It should be used in a combination with Oracle Mobile, and therefore is incompatible with PhoneGap.

    - CouchBase Mobile\(^6\): This is a document-oriented store, where documents are stored in the format of JSON files. The server does not impose any constraints on the database schema itself. CouchBase can be used together with PhoneGap with a help of a plugin\(^7\), providing support for iOS and Android operational systems. This plugin does not have extensive documentation or support itself.

    - LevelDB\(^8\): This is a key-value store developed by Google that does not support the notion of transactions at all. At the moment of this thesis there were no plugins which would provide a simple and quick way to integrate this storage with a PhoneGap application.

After consideration, we decided to choose SQLite database as it is a well supported, popular, stable and provides a place for storing large amounts of data.

\(^1\)https://www.sqlite.org/
\(^2\)http://www.sqlite.org/limits.html
\(^3\)http://www.objectdb.com
\(^4\)http://www.db4o.com/
\(^6\)http://www.couchbase.com/nosql-databases/couchbase-mobile
\(^7\)https://github.com/couchbaselabs/Couchbase-Lite-PhoneGap-Plugin
\(^8\)https://github.com/google/leveldb
4.3 Implementation

4.3.1 Implementation domain

Our mobile application is built for the food science domain. It is necessary to learn and understand the basic entities in this domain, connections between them and how mobile users can influence them.

In Fig. 9 the domain structure is presented.
All work and analysis with food is organized in the following way. There are studies (in Fig. 9 in the left upper corner), which usually represent one study in a certain year. A study is separated into four campaigns (on the right from the Study), usually they are associated with different seasons. Each study contains some amount of samples that have to be analyzed (below the Study in Fig. 9). For instance, a sample can be represented by apples. Our main entity for work is a subsample (in the middle in Fig. 9). A subsample is a usual food item that can be purchased in a shop. For example, if a sample is depicted by apples, different subsamples are various sorts of apples such as Antonovka or Glockenapfel.

Both study and campaign cannot be changed by a person who buys food items through the mobile application. The possibility to create, modify and delete these two entities exist only for the users of the FoodCASE desktop application; we use these entities in this thesis only for extracting and structuring data in our mobile application. At the same time, a subsample is an entity that mobile users work directly with and can influence its state.

As our mobile application was developed to facilitate shopping, the thorough attention was paid to situations that might occur during the purchasing process. For example, a listed item is not presented in the shop, or there is only a similar item. These situations are described by the status entity (in the upper right corner in Fig. 9). Each subsample can have an optional status, meaning that before visiting a shop no statuses are assigned to food items. To continue our example, the assumption can be made that some food items can replace each other. In Fig. 9 it is displayed as an arrow from the subsample to itself.

For the scientific purposes various foods are analyzed: Both raw items such as apples or tomatoes as well as composite foods, or recipe-based, such as pizza or soups. The raw items...
are called *ingredients* and composite foods are called *recipes*. The *subsample* entity can be represented by either of these food types.

### 4.3.2 Architecture

In this section the final architecture of the system is discussed.

In Fig. 10 a schematic representation of the architecture is given. There are two main parts: the FoodCASE desktop application (on the left) and the mobile application (on the right). Once the mobile application is started, its SQLite database (in the bottom right part in Fig. 10) must be synchronized before any further steps are performed. The application sends AJAX\(^1\) requests to the FoodCASE Web API (represented by arrows between the applications) and asks to send all necessary data back. The FoodCASE application queries required data from its PostgreSQL database (in the bottom left part in Fig. 10) and returns it in a format of JSON formatted data to the mobile application. It saves received data in the SQLite storage, communicating with it through the SQLite Plugin\(^2\) (the block inside the mobile application in Fig. 10). From this step the mobile application is ready for further use.

![Figure 10: Application architecture](image)

### 4.3.3 FoodCASE

Within the FoodCASE desktop application two features were implemented: a Web API and a source system side of constraints injection.

\(^{1}\)[http://www.w3schools.com/ajax/ajax_intro.asp](http://www.w3schools.com/ajax/ajax_intro.asp)

\(^{2}\)[https://github.com/litehelpers/Cordova-sqlite-storage](https://github.com/litehelpers/Cordova-sqlite-storage)
Web API

For the communication between the mobile application and the desktop one a Web API was developed, including methods for querying and posting data.

In Fig. 11 a simplified schema of the Web API is displayed. It consists of two classes \texttt{TdsSamplingWebService} and \texttt{TdsDisseminationSamplingBean} and one interface \texttt{TdsDisseminationSamplingRemote}. In Fig. 11 not all methods are displayed for the sake of convenience.

The mobile application interacts with the \texttt{TdsSamplingWebService}. It calls one of the Web API methods, sending data and defining query parameters where necessary. Each method in this class calls a method in the interface \texttt{TdsDisseminationSamplingRemote}, which is implemented in the \texttt{TdsDisseminationSamplingBean} (in the bottom in Fig. 11). The methods in this class work with the FoodCASE main database using a Hibernate\textsuperscript{1} mapping of the database tables to entities. After execution of the called method is finished, the result (if it exists) is returned to the mobile application through the \texttt{TdsSamplingWebService} class.

As it was mentioned earlier, the database tables are mapped to the entities within the FoodCASE application using Hibernate. This allows to work with the tables directly from the Java code: modify, add new records, delete records and query information.

Initially the existing mapping was provided for the majority of the required tables. Apart from our needs, those entities were serving the rest of the FoodCASE desktop application. During the development process a big challenge arose: these existing entities, represented as

\footnote{http://hibernate.org/orm/}
classes, were producing errors when used in our Web API implementation. Upon research, it was discovered that the interfaces that the entities were implementing were in contradiction with the Web API features and characteristics. Therefore, for our own purposes a new set of entities was created which were extending the Serializable interface.

Following is the list with all Web API methods and their descriptions.

- public Boolean login(@QueryParam("user") String user, @QueryParam("password") String pass): Provides sign-in functionality;
- public List<TdsSamplingSubsample> getSampleList (@QueryParam("studyId") Integer[] studyId): Pulls all samples from studies, which are passed as a parameter;
- public List<TdsSamplingStudy> getActiveStudyList(): Pulls all active studies;
- public List<TdsSamplingUnit> getUnit(): Pulls all amount unit representations such as weight;
- public List<TdsSamplingSubsampleIngredient> getSubsampleIngredients(): Pulls all information from the linking table between recipes and ingredients;
- public List<TdsSamplingCampaign> getCampaignList (@QueryParam("studyId") Integer[] studyIds): Pulls all campaigns from the specified studies;
- public List<TdsSamplingBuyingStatus> getBuyingStatusList(): Pulls all possible buying statuses;
- public void putAllPictures(List<TdsSamplingSubsampleImage> images, @QueryParam("user") String user): Pushes all photos from the local database to the FoodCASE central database;
- public void putAllSamples(List<TdsSamplingSubsample> samples, @QueryParam("user") String user): Pushes all subsamples from the local database to the FoodCASE central database.

Constraints injection in the FoodCASE application

For performing constraints injection it was necessary to collect constraints across the system. In our case constraints were already implemented directly in the FoodCASE code as different classes such as MandatoryValidator or IntegerFieldValidator. There were two classes which contained constraint definitions required for our mobile application: TdsSubsampleDetailFrame and TdsSubsamplePlanningPanel (Fig. 12 in the top row). These classes are responsible for a user interface with the food item details and an interface with a planning panel for purchases. They contain definitions of the mandatory constraints MandatoryValidator for different fields in a food item.
As different constraints types were presented in the system by different classes, a general interface `SamplingValidator` was implemented. This interface introduced three functions which were aimed at facilitating the constraint injection process:

- `public Integer getImportance():` Returns the importance of the constraint;
- `public String getValidatedField():` Returns the name of the field for which this constraint was set;
- `public String getType():` Returns the type of the field such as Integer or String.

This interface was implemented by a sample class `TdsSamplingMandatoryValidator` (Fig. 12 in the middle-right), which extended the functionality of the original `MandatoryValidator`. This multi-tier approach allowed to replace original `MandatoryValidator` objects in the code with new `TdsSamplingMandatoryValidator` objects. The usage of the interface provides an opportunity for extending the implementation with more constraint classes.

![Figure 12: Constraints injection implementation in the FoodCASE application](image)

In order to create new constraint files for the mobile application the `ValidationGenerator` class was created. An object of this class is created in the `generator()` method of the `TdsSubsampleDetailFrame` class. It uses a list of existing constraints `constraintList` (Fig. 12 on the top left) to prepare them and write to a noools file.

Fig. 12 demonstrates the connections between newly implemented (green) and existing (blue) classes and interfaces. The newly implemented method `generator()` and
constraintList in the TdsSubsampleDetailFrame are also marked in green. To summarize the implementation of constraints injection the final flow is described. When the TdsSubsampleDetailFrame object is created, constraints defined with it are initialized as TdsSamplingMandatoryValidator objects and added to a list of constraints constraintList. Afterwards, the TdsSubsamplePlanningPanel object is created and constraints belonging to it are initialized as TdsSamplingMandatoryValidator objects and are added to the list of constraints. On the last step the ValidationGenerator object is created and uses the constraint list to generate noools files.

4.3.4 Mobile application

In this section a full description of the mobile application is presented.

Database

Our choice was to use an SQLite database. In order to integrate it with our application, the Cordova-sqlite-storage plugin was used. This plugin allows to get access to all SQLite functionalities.

After installation of the plugin, it is possible to create and open a database and perform transactions.

Listing 4.1: Opening a database

db = window.sqlitePlugin.openDatabase({name: "my.db"});
db.transaction(function(tx){
  tx.executeSql("CREATE TABLE IF NOT EXISTS subsample (id INTEGER, name TEXT)", [],
   function(tx, results){}, //success callback for the SQL statement
   function(error){} // error callback for the SQL statement
  )
});

All SQL statements must be executed within a transaction; SQL statements (queries) have success and failure callback functions. An important characteristic of all transactions is that they are in fact JavaScript promises: They are used for deferred and asynchronous computations. All operations that should be performed after the database queries are placed in the success callback functions of the transactions.

1https://github.com/litehelpers/Cordova-sqlite-storage
Local database synchronization

In order to synchronize the local database in the mobile application with FoodCASE, AJAX\(^1\) requests of the Web API are performed. There are several requests, each of them synchronizes one specific table of the database; this is performed in the order that the tables’ relations are defined.

After a response with JSON data is received, database \texttt{INSERT} statements are executed.

Each operation of the table synchronization was implemented in a separate AngularJS \textit{factory}. Each factory returns a \textit{promise}, which is resolved during the run either successfully or with a failure - depending on the result of synchronization. All factory calls are chained together, so that after a successfully resolved promise the next factory can be called. All calls are performed in the \texttt{HomeTabCtrl} controller upon pressing the ‘Synchronize db’ button in the main home view \texttt{templates/home.html}.

General functionality and controllers

As our mobile application was developed with the Ionic Framework, all concepts and functionality mainly came from AngularJS\(^2\).

The entire application is a set of views, or \textit{templates}. All templates are mapped to their controllers and URLs in a configuration block, which is executed in the initial configuration phase.

Controllers are used to control the data in the application. Each view in our mobile application has a corresponding controller. For some special functionality such as taking pictures with the camera or preparing a subsample item for sending to the FoodCASE central database factories were used. A factory is a special AngularJS module that can have other dependencies and upon invocation returns a value.

The main functionality of our mobile application is listed below and its implementation is briefly described.

Shop view

In order to see a list of items for purchase, a shop should be chosen. It can be achieved either by choosing a specific date or browsing all possibilities, which includes all available studies and their campaigns.

Choosing a date is implemented in the \texttt{StudyCtrl} controller, which also is responsible for processing an option without a date. Depending on the choice, this controller shows a button which guides users further until they reach a list of available shops. The list of shops is created in the \texttt{ShopCtrl} controller that queries the local database and extracts all suitable shops. With the Ionic directive \texttt{collection-repeat} the view \texttt{templates/shop.html} is instantly populated with the shop list and presented to the users. An example of the shop view is displayed in Fig. 13.

Item details

Once users choose a specific shop, they can observe a list of items that they need to buy in

\(^{1}\)http://www.w3schools.com/ajax/ajax_intro.asp
\(^{2}\)https://angularjs.org/
this particular shop (Fig. 14a). The SamplesCtrl controller is responsible for this list representation. It queries the local database and extracts the food items that are assigned for a purchase in this specific shop. With the AngularJS directive ng-repeat the view template templates/sample.html is populated with the list of food items.

When a particular item is chosen, the database is queried and detailed information view is displayed based on the extracted item data (Fig. 14b). Apart from the item’s detailed information this view also offers barcode scanning and photo functionalities, which are discussed later.

**Operations with food items**

There are 6 different actions that can be performed with a food item.

- **Buy an item**: Once this option is chosen, the editBuyingStatus function in the DetailsCtrl is called and the item status is updated to ‘Bought’ in the local database;

- **The second best option**: This action is needed if the designated item cannot be bought for some reason. This option is displayed only if there exists an item in the database that can replace the current one. It is achieved by querying the database in advance and checking the existence. In case this option is chosen, the user is redirected to a new view templates/secondbest.html. This view demonstrates detailed information about the second best choice item and gives the user the possibility to buy it;

- **Postpone to another shop (Fig. 15)**: This option allows the user to move an item to another shop where it can be bought later. This option redirects the user to the
4.3. IMPLEMENTATION

(a) List of items

(b) Detailed item view

Figure 14: List of items and detailed item information views

It is possible to undo an item purchase and removal. In both cases the item status in the database will be reset to NULL and an item will be displayed again as available for all actions.

Additional information after purchase
As soon as the first item is bought, a new button appears at the bottom of the shopping list.
Figure 15: List of additional actions available for this item

This button introduces the functionality of adding more details about the purchased item. The view templates/detailsadditional.html is responsible for editing and updating information. This time all already existing details are blocked from editing, but empty fields are available for changes. Once the result is submitted, the item record in the database is updated.

4.3.5 Constraints injection in the mobile application

After the source side constraints injection is finished, the rule files are ready for the constraints injection process in the target system.

There are three cases when constraints are needed in our mobile application:

- Editing of an existing item before purchase;
- Adding a new item for purchase;
- Input of additional information after purchase.

For each of the cases our constraints injection method was implemented in the following way. When a user navigates to either of these options, a controller of the corresponding view is invoked. Using the Cordova-plugin-file\(^1\) the data from the rule files is extracted and a flow

\(^1\)https://github.com/apache/cordova-plugin-file
is created after compiling. Inside this flow a model can be extracted from the constraints file and a local model object is created.

For each of the operations that required a constraint check, the local model is used. The checking process is implemented as follows:

1. Upon a field change in the views templates/editDetails.html, templates/detailsadditional.html and templates/newItem.html a check function is called. The local model in each of the cases are created and populated with the received field value;

2. The match function for the model is executed, returning either a set of errors if any discrepancies were found or an empty set of errors if all values were proved correct.

3. If a non-empty set of errors was returned, an error message corresponding to the input field is extracted based on the field name. In case no errors were found, this step is the last and no more actions are performed.

4. When a corresponding error message for the field was found, it is added to the error object for this view. The user interface immediately displays an update, represented by an error message in a red box.

In Listing 4.2 an example of creating a flow, a model and performing a check is given.

Listing 4.2: Example constraint check

```javascript
$templateScope.flow = noools.compile($templateScope.result, {name:"mandatoryrules"});
$var Model = $templateScope.flow.getDefined("model");
$var MandatoryModel = new Model({});
$templateScope.flow.getSession(MandatoryModel).match().then(
    // actions performed on the error set
);```

The implementation specifies that the flow is created only once, if the check function is called multiple times.

4.3.6 Data quality feedback

In total three different versions of the data quality feedback were proposed and implemented. The first version is an adaptive feedback implementation based on the earlier developed responsive data quality feedback approach in Section 3.1.3. This feedback representation version was named basic.

The size of the device is determined with help of AngularJS's $window service and its property innerHeight. The result is returned in pixels and for our application it was decided that 650 pixels would be separating a small screen size from a large screen size. The example of the code you can see in Listing 4.3.
For the large screens the feedback is positioned at the bottom, taking 35% of the screen. This size is sufficient to display a pie chart in the feedback area as well as a table with additional information for our mobile application (Fig. 16a).

```
Listing 4.3: Adaptive Feedback

if ($window.innerHeight < 650){
    // set flag for small screens
} else{
    // set flag for big screens
}
```

For small screens the feedback is positioned on the side panel of the application, which becomes visible when a user swipes right (Fig. 16b).

(a) Big screen data quality feedback  
(b) Small screen data quality feedback

Figure 16: Basic version

In both cases there is a possibility to see more detailed statistics on a separate page that is available upon a click on the button ‘Detailed statistics’ (Fig. 16). It contains the bar chart demonstrating the data quality levels in each shop. Data quality is calculated as a percentage of filled data fields of the food items in each shop, eleven different data fields per item were taken into account. These fields were chosen based on the assumption which information could be found on the packaging of a food item and easily added by the users in the mobile application.

The second version, named *weather*, introduces a metaphorical representation of data quality in the application. Different levels of data quality are mapped to five different weather conditions such as a thunderstorm, rain, cloudy, intermediate and sunny weather. This mapping consists of two steps:
4.3. IMPLEMENTATION

(a) Side data quality feedback
(b) Icon data quality feedback

Figure 17: Weather version

1. Evaluation in percentage of eleven fields completeness per food item, for all items existing in the mobile application at the given moment;

2. Summarizing the evaluation results and mapping the final data quality level to a weather condition.

The weather information is shown to the users with a little icon on the top panel (Fig. 17b) and as a picture in the side menu (Fig. 17a). Additional short statistics in the table format are available.

The third version, named *modal* because of the implementation solution it contained, presents reduced information about data quality in the mobile application to the users. The users are able to learn statistics only concerning the amount of bought, deleted, edited and added items in the system. These statistics are shown in a special pop-up as a pie chart and a table (Fig. 18). The pop-up appears only when a status of the item was changed: For example, when it was bought. During the application usage the status change is the only option to see this feedback. No detailed information about data quality in the items themselves is provided.

4.3.7 Barcode scanning

The barcode scanning is implemented in the mobile application with help of the PhoneGap Barcode Scanner\(^1\). This plugin works with multiple operational systems and can read a wide range of different barcodes, QR-codes and EAN-codes.

The scanning is available in the application in the detailed view of an item. Upon a click on the barcode scanning button a function `scope.scanBarcode` is called. The function

\(^1\)https://github.com/phonegap/phonegap-plugin-barcodescanner
uses the barcode plugin to get access to the camera and recognition functionality. In case of success a success callback function is called, in case of failure an error callback function is executed. A code snippet introducing the barcode scanning function is presented in Listing 4.4.

When a barcode is detected and scanned, a check of this barcode is executed. In case the item for which scanning was performed does not have any barcode, the newly scanned one is added and the database record corresponding to this item is updated (Fig. 19).

**Listing 4.4: Barcode scanning**

```javascript
$scope.scanBarcode = function(){
  cordova.plugins.barcodeScanner.scan(
    function(result){}, //success callback
    function(error) {}, //error callback
  );
}
```

If the item already has a barcode, the newly scanned one is compared with the existing one. In case they are the same, no changes are applied to the item and the database remains in the same state. If the barcodes differ, users are asked if they want to replace the existing barcode with the new one. If they agree, the database record for this item is updated with the new barcode and new information is displayed.
4.3. IMPLEMENTATION

4.3.8 Photos

The functionality to take pictures is implemented with the Cordova Camera Plugin\(^1\). This plugin allows to get access to the full camera functionality in the mobile phone that is inaccessible otherwise.

A call to the camera functionality is implemented in a separate factory MyCamera that is called each time the photo process is started. The camera works with several predefined configurations which are subject to tuning and possible changes. These parameters in our implementation include:

- **quality**: the quality of the saved image, expressed as a number between 0 and 100. The full resolution is represented by 100;
- **targetWidth**: a width in pixels to scale the photo;
- **targetHeight**: a height in pixels to scale the photo;
- **saveToPhotoAlbum**: a boolean value that specifies if the image should be saved to the photo album on the device after capture;
- **cameraDirection**: a number ‘1’ or ‘0’, specifying if the front or back camera should be used;
- **destinationType**: a type that specifies the format of the return value.

\(^1\)https://github.com/apache/cordova-plugin-camera
The destinationType parameter is an important part of the camera configuration. In our application the DATA_URL format is used. It returns a base64 encoded string depicting an image. Another option was to use FILE_URI type that represents a file URI and store it in the database. The final choice of DATA_URL was based on several reasons:

- Security reason: with DATA_URL an image string is saved to the database. In case of a memory card malfunction the image still can be extracted. Moreover, if users have direct access to the images through a photo gallery on their mobile phones, they might delete or modify an image which can result in image extraction failure.

- Device independent storage: the photos can be transferred between devices without any dependence on a platform;

- Reducing the number of steps required to show an image: in case of the chosen solution only the image record from the database should be extracted. Afterwards, the image can be displayed on the screen without any transformations. In case of the FILE_URI it would be necessary to extract the database record first and then use it to find a file in the system and display it.

- Synchronization convenience: when the photos should be synchronized with the FoodCASE central database, our chosen type implies that the records can be extracted from the local database and sent without any further transformations. Moreover, the FoodCASE central database stores the images in base64 encoding, meaning that in case of the FILE_URI type a conversion before synchronization would have been required.

The mobile application has additional functionality for creating new pictures of the food items. This functionality can be accessed through the ‘Make photos’ menu in the home view. The list of all items existing in the local database is presented to users. It is possible to filter items based on different parameters such as a buying status or a campaign. When a required item is chosen, its detailed information is displayed. The users can create new photos or delete existing ones. Each of the operation leads to the database changes: either new records are added or existing ones are deleted.

4.3.9 Location and navigation

A simple navigation functionality is implemented in our mobile application. It uses the Google Maps JavaScript API.1

The developed functionality includes:

- Positioning on the map based on the geographical location (Fig. 20a);
- Positioning the shops on the map based on their geographical location (Fig. 20a);
- Suggesting a driving route from the current position to the chosen shop (Fig. 20b).

At the moment of this thesis no coordinates were available in the actual data. The navigation was tested based on an artificial data set with locations of shops in Zürich.

1https://developers.google.com/maps/documentation/javascript/
4.3. IMPLEMENTATION

(a) Locating shops and the user

(b) Route calculation based on the chosen shop

Figure 20: Navigation

4.3.10 Pushing data to the FoodCASE database

At any moment of the application usage users can decide to send the current data to the central database. This functionality is represented by the button ‘Push to the server’.

There are two major types of data that can be pushed to the FoodCASE database: The photos and general food item information. In our implementation it was decided to use this separation due to the potentially large size of the photo files. The use case of the application implies that users might make several pictures per food item, resulting in a significant amount of data.

Our solution pushes data fully or partially to the central database based on type of the Internet connection. Using the plugin Cordova Network Information\(^1\) the type of connection is determined: Wi-Fi or mobile Internet. In Listing 4.5 a code snippet for determining the connection type is presented.

Listing 4.5: Determining the type of Internet connection

```javascript
if(window.Connection) { // some Internet connection exists
    if (navigator.connection.type != Connection.WIFI) {
        // Mobile network actions
    } else { //wi-Fi network actions
        //Mobile network actions
    }
}
```

In case of mobile Internet presence, only general information is pushed. Pushing the photo files via a mobile network can result in significant expenses and possibly a long time of

\(^1\)https://github.com/apache/cordova-plugin-network-information
execution, therefore they are sent to the FoodCASE database only when the application has access to the Wi-Fi network. If there is no Internet connection, users are informed about it. All necessary data is extracted from the local database and sent in a JSON format.
This chapter consists of two parts. The first part describes a user study that was carried out in order to evaluate the mobile application. The second part focuses on the assessment of this study.

5.1 Study description

In order to evaluate the developed mobile application a user study was carried out. The goal of the study was to investigate what types of data quality feedback the users prefer, assess the usability of the application and learn what the users expect from such an application.

For the study three different versions of the data quality feedback were used as described in Section 4.3.6. Each of them was embedded in a separate instance of the mobile application.

The study was simulating a shop visit, offering the users an opportunity to look at various food items and perform different actions in the application. For the study five food items’ descriptions were added to the database and the corresponding food items were placed on the table in front of the participant. The descriptions and the food items were chosen in a way such that participants would be required to perform various actions such as buying, editing or deleting an item.

Upfront the study participants were given the possibility to familiarize themselves with the application. For this purpose a version without any data quality feedback was demonstrated. Participants were instructed about the goal of the application and the purchase process. They were asked to navigate to the required shop and try to perform some operations with any food item. This helped to prepare participants for the navigation in the application and fulfilling the study.

As there were three different application versions with which participants had to work, it was decided to shuffle their order from one participant to another in order to avoid a learning
5.2. RESULTS

The study questions were divided into four parts: Data quality feedback, usability, validation and a personal user feedback. During the whole study participants could make any notes about the applications in the personal user feedback sheet. The full list of the questions can be found in Appendix A.

The data quality feedback part consisted of three sections. For each section the participant was presented with one of the application versions and was asked to purchase the items positioned on the table. After the participant was finished with all necessary actions, three questions about the data quality feedback were asked.

The usability part contained questions about the general usability of the applications. The questions covered all three versions as the larger part of the interface was common for all of them, but participants were suggested to mark any questions if they would have significantly different answers to them. In case the question was mentioning an interface part or an option that the participant has not seen, an option ‘Not applicable’ was available in the questionnaire.

The validation part contained three screenshots of the different error messages that could happen in the application. The participants were asked if they could correct the errors and if so in which way. Afterwards, they had to answer three questions about validation in the application.

The last part of the study gave the participants the opportunity to give the feedback they wanted in a freely written way. They could describe what application version (or a combination of versions) they liked the most and suggest ideas for the application improvements.

5.2 Results

13 participants took part in the study. The majority of them are studying or working in the computer science field.

One of the participants gave negative or strongly negative feedback with a strong personal bias in most questions. This participant was not eliminated from the study evaluation.

A Likert-type scale was used for the answers representation. In the usability part of the questionnaire some questions received ‘Not applicable’ answers. These answers were excluded from the total evaluation results and the questions were specified as receiving not all participants’ answers.

5.2.1 Data quality feedback

There were three statements that were presented to the participants:

- Statement 1: The data quality feedback representation was clear for me;
- Statement 2: I found the feedback useful;
- Statement 3: The feedback was helpful in improving the data quality.

The first statement received generally a very positive response. Fig. 21 presents the distribution of answers. Although the basic feedback representation version was not clear for one
participant, it received totally the same amount of positive responses as the modal version. The weather version had the highest amount of the best possible answer.

![Figure 21: The data quality feedback representation was clear for me](image)

![Figure 22: I found the feedback useful](image)

![Figure 23: The feedback was helpful in improving the data quality](image)

The second statement received positive feedback. The least useful version was the modal one with four participants being neutral towards it. The basic and weather versions were found
5.2. RESULTS

by the participants comparably good in regard to the feedback usefulness.
The third statement revealed the notable differences in how helpful the feedback was for improving the data quality. The most successful results were shown by the weather version. The least helpful in improving the data quality was the modal version, receiving even one a strongly negative response. The basic version obtained generally a good response.

5.2.2 Usability

In this evaluation the usability questions are separated in two parts: the first one consists of the questions that did not receive any ‘Not applicable’ responses. In the second part the questions with different numbers of ‘Not applicable’ responses are presented.

<table>
<thead>
<tr>
<th>Question</th>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1: The buttons and link labels were clear</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>#2: The navigation was clear and consistent</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>#3: The order and organization of the information on the screen was clear</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#4: The application provided enough information to complete the task I needed</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#5: It was clear how to use this application</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>#6: I was able to complete the tasks efficiently using this application</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#7: It was easy to use this application</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 5. USER EVALUATION

5.2.3 Validation

There were three questions in this study part.

- Question 1: It was easy for me to understand where incorrect data was entered;
- Question 2: It was easy for me to understand how I need to change the input data in order to satisfy the validation constraints;
- Question 3: The validation errors were displayed clearly.

The received answers are very positive. All participants found it easy to understand where on the screenshots the errors were made. The large majority of the participants agreed that the validation errors were represented clearly. They also did not have any problems with understanding how the input data needed to be changed.

The results are presented in Fig. 26.

5.2.4 Personal user feedback

The users were asked which application version (or a combination of versions) they liked the most and why. Eight out of 13 participants chose the weather version or a combination of it
with some additional features. Some participants specified that the *weather* version with a pie chart statistics or a pop-up with a possibility to turn it off in the configurations would be the best option for them.

The pure *modal* version was named as the best only by few participants. Other participants noted that this data quality feedback solution interrupts their work and that they would not like to use it.

The *basic* version was regarded by some participants as the best one because of the detailed statistics it contained. Other participants indicated that not all information available in this version would be useful for them during a purchase process. Therefore, they suggested to choose the most informative representations in order to use them together with another application version.

The users’ feedback revealed that there are several suggestions for the application improvements seen by the different participants.

Three participants noted that the shopping list is cumbersome or inconvenient. They suggested that the bought items should be grouped together in order to free more space for not yet bought items in the list. Another option would be to make the shopping list generally smaller and more compact.

Three participants mentioned in their feedback that the button icons were not clear or obvious. They suggested to change the icons and make the buttons’ purpose more explicit.

Two users noticed that the data quality feedback should be presented in a non-intrusive way. One of the users suggested that the *weather* version was the best illustration of it, another one stated the *basic* version as non-invasive and convenient. In both cases the *modal* version was mentioned as disturbing and interrupting.

Two users stated in their feedback that the editing process of an item should be changed. Although there were only two participants who clearly specified this concern, some other users had issues while trying to edit an item. A way to edit an item without an additional step from the detailed item view was proposed in the feedback.

Three participants each gave a suggestion for further improvements. For example, one participant suggested that all fields of an item should get some description displayed on a ‘Help’ icon click. Another participant proposed that the ‘distance’ until the next weather
condition should be shown. One more original suggestion was to show all data quality feedback after a button ‘Finished shopping’ (this button was proposed by the same participant) was pressed.

5.3 Key findings

The list of the most interesting findings from our user study is presented below.

- All data quality feedback representations were clear for the majority of the users (Fig. 21). This shows that pie charts, tables and a metaphorical weather depiction are easy to understand and therefore are generally good visualization solutions;

- The basic and weather versions were found more useful than the modal version (Fig. 22). It can be explained by the simple representations that were used as well as by the non-invasive nature of the solutions.

- The weather version was found the most helpful in improving the data quality (Fig. 23). It can be explained by its simplicity and immediately visible effect as soon as some data quality changes occur. The participants noted that the motivation to improve the data quality with this version was the highest. The modal version did not receive much support as it was rated as not motivating the users and also did not show enough information to perform possible data quality improvements.

- Fig. 24 indicates that all participants were generally happy with the usability of the application. Question #4 demonstrated the most controversial result: All users except for one agreed with the statement, one user disagreed and no one took a neutral position. This can signify that some additional details should be provided. There is still space for improvements in the button labels and icons (#1) as it was mentioned earlier and the order of the information on the screen (#3).

- It is important to see in Fig. 24 that the vast majority of the participants found the application easy to use (#7) and were able to complete the tasks efficiently (#6).

- In Fig. 25 it is clearly seen that the help provided by the application is not enough (#3) as two out of 11 participants disagreed with the statement. This answer could be caused by the difficulties in locating the help in the application. On the another side, all 11 participants who encountered the error messages found them helpful in fixing the problems.

- The validation feedback results (Fig. 26) showed that the error representation in the application was good and enough for the majority of the participants to cope with the issues.

- Based on the personal feedback it can be concluded that the less invasive and easier to understand data quality representations are a good solution. Moreover, the motivation that the data quality feedback creates is an important part of the data quality improvement. A well motivating solution makes the process of the data quality improvement more interesting for the users.
• Some original users’ suggestions can be used to improve the application. For example, a gamification idea was proposed: Develop ways to actively engage the users and motivate them to support the data quality on a high level.

In general, the results of the study are inspiring for our mobile application. The ways the data quality feedback can be given were compared within three different application versions and the most helpful version in improving the data quality was found. The way the best data quality feedback representation could look like was determined as a possible combination of the weather version and the basic one. Finally, the majority of the participants indicated that the current interface of the application has a good usability and suggested new ways to improve it further.
Contributions and shortcomings

In this chapter the contributions made by this thesis are discussed first. Afterwards, the shortcomings of this work are presented. Finally, possible ways to extend the mobile application are introduced.

6.1 Contributions

The list of the main contributions of this thesis is summarized below:

- A novel constraint injection concept was developed. This concept allows to:
  - Synchronize the constraints including all information between different applications or components;
  - Generate a constraint set of a different size based on the source set;
  - Use different importance thresholds for the constraint set injection;
  - React quickly on possible changes in the constraint set by generating a new set for the injection;

- An adaptive interface generation approach based on the constraint set was introduced. This approach helps to:
  - Generate automatically an interface based on the constraint set;
  - Avoid adjustments in the code in case of changes in the constraint set;
  - Use the additional information about constraints for building a more intuitive and user-friendly interface;

- A responsive data quality feedback method was proposed. It allows to:
– Adjust feedback position and structure depending on the device size automatically;

• All the mentioned concepts were implemented within the mobile application and the FoodCASE desktop application; In addition to it, the application was equipped with additional functionality such as:
  – Camera: taking and storing pictures;
  – Barcode scanning: scanning, recognizing and storing barcodes;
  – Navigation: locating the current position and calculating the driving route to the chosen shop;

• Three different data quality feedback representations were proposed and implemented;

• A user study was carried out in order to compare the different data quality feedback representations and to evaluate the general usability of the application. The received feedback was positive and offered interesting insights into the motivation that the data quality feedback can provide.

6.2 Shortcomings

Some shortcomings can be found in this thesis. The main ones are presented below:

• The *adaptive interface* method does not specify how the interface should be generated in case no constraints were received from the source system;

• The *responsive feedback* method does not introduce a method for defining a threshold between large screens and small ones;

• The implemented mobile application strongly depends on the third party plugins and their stability;

• The mobile application might have some compatibility issues on different mobile platforms and different versions of the same operational system due to the PhoneGap and Ionic frameworks’ characteristics;

• The mobile application functionalities can vary among mobile platforms. For example, the possibility to connect a printer depends on the mobile platform: For Android the majority of printers can be connected using a proper plugin and for iOS only printers with the AirPrint ability can be used.

• The mobile application does not provide a solution to potential network problems during the database synchronization.
6.3 Extensions

There are different ideas that can be used for extending the mobile application. It can be enhanced with new features and functionality that are listed below. This list can be continued as new ideas or requirements are introduced.

- Extending the constraints existing only in the mobile application with new ones. They might include:
  - Limiting a shopping trip duration;
  - Establishing a maximum purchase price for an item and for all items planned for the shopping trip together;
  - Restricting the purchase of perishable foods to the last moment before bringing them to the laboratory;
  - Checking the expiry date of a food item.

- Integrating a food item check and automatic field filling based on its barcode. For these purposes a proper barcode database with food items information should be found:
  - The mobile application compares the data based on the scanned barcode with the existing data and notifies a user in case of discrepancies;
  - The application scans the barcode and fills in the empty data fields based on the extracted information.

- Implementing automatic ordering of the shops based on the current user location. The shops that are located closer are offered for the visit earlier than the remote ones. This feature can help users to do the shopping quicker;

- Introducing an automatic data push to the FoodCASE central database as soon as good Internet connection is available;

- Implementing a special ‘Assign to another person’ option if a food item cannot be bought. This option assumes that other people using the same application are known. The option suggests to move an item to a shopping list of another user who can potentially buy this item. This option requires that several additional constraints are introduced in the system such as a number limitation of such ‘assigned’ items, proximity check for users or a coinciding shops check. Moreover, it is necessary to keep track on such items in order to avoid moving back and forth;

- Implementing general usability improvements such as adding a shopping cart for the bought items, a drop-down list for predefined values such as currency or editing an item in one click. The latter idea was popular among the user study participants and included a possibility to edit an item information directly where this information is displayed without additional steps as there are currently implemented.
This chapter provides a summary of the work that was presented and proposes some possible directions for future developments.

7.1 Work summary

In this thesis three novel approaches were introduced: constraint injection, an adaptive interface based on the constraint set and responsive data quality feedback.

The constraint injection method proposes a way to inject constraints from the source system into the target system without influencing the work of the latter. Constraints of different importance levels can be collected together, modified and prepared for future usage. Afterwards, the target system can use these constraints. In case a change in the constraint set in the source system occurs, the injection process can be repeated without influencing the system’s work: It makes this method reusable.

The adaptive interface method that was developed offers a new approach for creating an adaptive interface. This approach uses the constraint set for generating input forms. Constraints in the set can be separated into several groups based on their importance, the number of such groups is flexible. For example, a separation in two groups of mandatory (very important) and non-mandatory (less important) input field constraints. The interface components are generated based on this importance partition: The more important a constraint is, the earlier in the interface a corresponding element is placed. For input forms this solution makes sure that users do not skip the most important input fields, which are presented in the very beginning of the input process.

The responsive data quality feedback method proposes a solution for different sizes of screens. This approach suggests to distinguish large and small screens in mobile devices, which allows to position data quality feedback in different places of the interface depending
FUTURE WORK

on the screen size. For small screens the feedback can be placed on the left hidden panel, for large screens it can be located at the bottom. The developed method allows to automatically select data quality feedback positioning and therefore the need to manually adjust the location is excluded.

All three methods can be successfully integrated together, creating one flow. This flow starts in the source system by collecting and preparing the existing constraints for the constraint injection into the target system. The target system uses the received set for creating an adaptive GUI for the data input. The same constraint set is used for checking the input values. In our thesis data quality is understood as constraint compliance, so its level is estimated based on how well the constraints are satisfied. The data quality feedback representing these estimations can be displayed depending on the screen size of a device.

All three concepts were implemented in the mobile application and partially in the FoodCASE desktop application. The data quality feedback was proposed and implemented in three different versions. The mobile application was implemented with the usage of the PhoneGap and Ionic frameworks; for the constraints definition and validation the rule engine nools was used. In the FoodCASE application the Web API and constraints injection were implemented in Java.

The three developed data quality feedback representations were included in a user study. The participants were asked to compare and evaluate these versions, give usability feedback and answer questions about input validation in the application. The assessment of the study revealed a positive user feedback and determined the preferences in the data quality feedback representation.

7.2 Future work

The three developed approaches can be extended further in different directions. Below the main ideas for future work are listed. This list does not limit the possible future improvements and developments. Certainly, new ideas can emerge and suggest interesting extensions for this work.

7.2.1 Constraint injection

This approach can be extended with opposite direction synchronization. If there are constraints added to the target system the source system must be notified and existing data should be checked for these new constraints (if applicable). This would help in avoiding data discrepancies.

Another possible extension includes developing an approach for the automatic synchronization of the constraint set once it is changed in the source system. Currently the constraint injection method allows to perform it both manually and automatically without detailed description of the process. For the automatic synchronization a source and a target constraint definition methods should be determined. When they are known, it is possible to develop a framework for the automatic constraint transformation from one representation to another. Also a more general case can be investigated, where constraint definition methods are not
specified.

### 7.2.2 Adaptive interface

Adding more possible components of the interface that can be generated with the adaptive interface method is a possible extension of this method. This can be achieved by using types of constraints different from the input ones. For example, temporal constraints where one action is allowed only after another. At the same time, the problem of distinguishing what constraints are responsible for which part of an interface should be solved.

One more question that is interesting to address is adapting an interface to the data quality level. For instance, if data quality in the mobile application is decreased, all next input forms become more strict and increase the number of mandatory fields. Although this idea can help to increase the data quality on average, it has also a drawback: The data quality across different food items might fluctuate.

### 7.2.3 Responsive data quality feedback

Enhancing the responsive feedback approach with an automatic choice of the feedback representation among different ones depending on the available screen space. For instance, a pie chart for large space, a bar chart for medium or a table for small.

Another interesting idea is to propose a method for the data quality feedback when a mobile device is connected to a device with larger screen: For example, to a desktop or a television screen. In this case a way to display the feedback on this new screen should be found. The feedback presented on the larger screen can contain additional statistics that are not shown on a mobile device.
7.2. FUTURE WORK
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Constraints injection structure</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Example constraints set</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Adaptive interface generation</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Special case of the example constraints set</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>Data Quality Feedback responsiveness</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>General flow structure</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>Start page in prototypes</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>Shopping list page in prototypes</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>Domain structure</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>Application architecture</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>Web API</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>Constraints injection implementation in the FoodCASE application</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>Shop view</td>
<td>31</td>
</tr>
<tr>
<td>14</td>
<td>List of items and detailed item information views</td>
<td>32</td>
</tr>
<tr>
<td>15</td>
<td>List of additional actions available for this item</td>
<td>33</td>
</tr>
<tr>
<td>16</td>
<td>Basic version</td>
<td>35</td>
</tr>
<tr>
<td>17</td>
<td>Weather version</td>
<td>36</td>
</tr>
<tr>
<td>18</td>
<td>Modal version</td>
<td>37</td>
</tr>
<tr>
<td>19</td>
<td>Adding a barcode to the food item information</td>
<td>38</td>
</tr>
<tr>
<td>20</td>
<td>Navigation</td>
<td>40</td>
</tr>
<tr>
<td>21</td>
<td>The data quality feedback representation was clear for me</td>
<td>45</td>
</tr>
<tr>
<td>22</td>
<td>I found the feedback useful</td>
<td>45</td>
</tr>
<tr>
<td>23</td>
<td>The feedback was helpful in improving the data quality</td>
<td>45</td>
</tr>
<tr>
<td>24</td>
<td>The first group of the usability questions</td>
<td>46</td>
</tr>
<tr>
<td>25</td>
<td>The second group of the usability questions</td>
<td>47</td>
</tr>
</tbody>
</table>
26 Validation study responses ........................................... 48
Acknowledgments

Foremost, I would like to express my profound gratitude to my supervisors, David Weber and Maria Husmann, for their active engagement and continuous support. Our regular discussions and their openness for all my questions were incredibly helpful during my work. I want to thank David especially for his patience during past six months.

Furthermore, I would like to express my sincere thanks to Prof. Dr. Moira C. Norrie and the GlobIS group for allowing me to work on my master thesis in such an interesting topic. I especially thank Dr. Karl Presser for his valuable help with the FoodCASE system.

I would like to thank my family and friends for their constant support and alacrity to talk with me about my thesis work. A special big thanks goes to Nadia for sharing a work office with me and creating a cheerful and optimistic mood. And of course I am exceptionally grateful to my boyfriend, Cyril, for his infinite support, patience and not giving up in explaining me the correct way of using English articles.
Appendices
User Study
**FoodCASE Mobile Application**

**Description**

The main purpose of the FoodCASE Mobile Application is to assist a person who buys food items for the scientific research. The amount of various items that must be bought during one shop visit might reach several dozens. As all items are bought for the scientific research, it is necessary to keep track on the purchases as well as on additional information that should be stored together with a purchase. High quality of data must be supported.

The goal of this study is to evaluate different versions of the data quality feedback on a mobile device within the food research domain.

Before the study begins, you will have an opportunity to familiarize yourself with a mobile application version, which does not contain any data quality feedback features. Your goal at this step is to explore the interface and the flow of the application.

Please follow the steps below:

1. Open the application;
2. Synchronize the database;
3. Chose “Shopping” option;
4. Choose a study;
5. Choose a campaign;
6. Choose a shop Coop Shop;
7. Observe a list of food items which needs to be bought in the shop;
8. Look for the details of any item.
**Study**
Now you will be presented with three different application versions. For each of them please follow the scenario. After each version you will be asked several questions. After you are finished with all three versions, you will be asked questions about all versions in general.

**Scenario**
You are going to the shop Coop Shop to make purchases. You will see the list of the food items you need to buy in the shop, as well as items themselves (presented in front of you). Your goal is to buy items and keep the data quality as high as possible.

Please notice that in case when you cannot buy a specified item, you have the following options (not all of them are used in the current study, please see the comments below):

1. Choose the second best option: in case of existence of such an item in the database, it will be offered you (not applicable in the scenario);
2. Move an item to another shop (not applicable in the scenario);
3. Edit an item in order to adapt an existing item;
4. Add a new item instead of the existing one;
5. Delete an item.
Part 1. Data Quality Feedback

Application 1: ____________________________________________________

1. The Data Quality Feedback representation was clear for me.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

2. I found the feedback useful.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

3. The feedback was helpful in improving the data quality.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
4. The Data Quality Feedback representation was clear for me.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. I found the feedback useful.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. The feedback was helpful in improving the data quality.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. The Data Quality Feedback representation was clear for me.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

8. I found the feedback useful.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

9. The feedback was helpful in improving the data quality.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
**Part 2. Usability**
Please notice that the questions in this part are referring to the all versions of the application.

1. **It was easy to use this application.**
   
<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. **I was able to complete the tasks efficiently using this application.**
   
<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. **It was clear how to use this application.**
   
<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. **The application’s error messages were clear.**
   
<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. **The application’s error messages helped me to fix the problem.**
   
<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. **I was able to recover from all mistakes I made (return to the previous state).**
   
<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. **The help provided by the application was enough.**
   
<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8. **The help provided by the application was clear.**
   
<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9. It was easy to add the data.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

10. The application provided enough information to complete the task I needed.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

11. The order and organization of the information on the screen was clear.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

12. The navigation was clear and consistent.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

13. The buttons and link labels were clear.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Not applicable</th>
</tr>
</thead>
</table>
Part 3. Validation
You will be presented with screenshots of the input fields and potential validation errors in the application. Please correct them in a way that you would correct them in a real situation. After that answer the following three questions.

1. It was easy for me to understand where incorrect data was entered.
   - Strongly disagree
   - Disagree
   - Neutral
   - Agree
   - Strongly agree

2. It was easy for me to understand how I need to change the input data in order to satisfy the validation constraints.
   - Strongly disagree
   - Disagree
   - Neutral
   - Agree
   - Strongly agree

3. The validation errors were displayed clearly.
   - Strongly disagree
   - Disagree
   - Neutral
   - Agree
   - Strongly agree
Part 4. Additional feedback and paper prototyping

1. Please specify, which application version you like the most and why:

2. Please add any feedback that you think might be valuable for the application development.


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Data Quality Assurance in a Mobile Application for Food Science Data

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s): Birenbaum

First name(s): Diana

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the ‘Citation etiquette’ information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Zürich, 06.04.2016

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.