String translations

Web application for app translations

R. M. Borhem
B. Hizli
E. L. Jimmink
Z. A. van Steijn

Delft University of Technology

TU Delft

String translations

Web application for app translations

by

R. M. Borhem
B. Hizli
E. L. Jimmink
Z. A. van Steijn

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on July 4, 2018.

Project duration:	April 24, 2018 – July 6, 2018	
Members:	R. M. Borhem	
	B. Hizli	
	E. L. Jimmink	
	Z. A. van Steijn	
Thesis committee:	Dr. A. E. Zaidman,	TU Delft, coach
	Ir. W. Van,	bunq, client supervisor
	Ir. O. W. Visser,	TU Delft, coordinator

TU Delft

Preface

This report documents the bachelor's thesis "String translations" as part of the Bachelor Computer Science at the Delft University of Technology. The project ran from April 2018 to June 2018 at bunq B.V. The team worked on an application to speed up and automate the translation process.

We would like to thank everyone who has helped us during the course of the project. A special thanks goes out to Wessel Van who supervised us at bunq and Sander van den Oever who answered many of our questions. Finally, we would like to thank our coach Dr. Andy E. Zaidman for his guidance and feedback throughout the project.

R. M. Borhem
B. Hizli
E. L. Jimmink
Z. A. van Steijn
Delft, June 2018

Summary

bunq is an internationally active digital bank. At the moment, bunq only operates in some European countries, namely the Netherlands, Germany, Spain and Italy. Their goal is to be a worldwide player. To be able to fulfil this goal, their apps should be available in multiple languages. In the past, the translation process took an unnecessary large amount of time. Due to this inefficiency in the process, the client, bunq, gave the team an assignment to create an application which helps to decrease the workload and time spent on translating.

The text of the apps are currently stored as strings in `.xml` and `.strings` files for the Android and iOS app, respectively. Each string has a key which is the string's name. Using these names, the apps' code can refer to a certain string to display them in the apps' interfaces. The product that was created provides the bunq employees a platform where translations can easily be managed. Within the application, the app developers are able to create new strings. These new strings can then be reviewed by bunq's copy writers and thereafter be translated by bunq's translators. Within the application, a functionality is integrated which makes it possible for bunq's developers to map the Android and iOS keys with each other to pair the belonging strings together. This ensures the consistency of the strings between both apps and also initialises the database of the application with the existing strings.

Before starting on creating the application, research had to be done. Certain decisions needed to be made, such as the choice of programming language and frontend and backend frameworks. The project was divided into backend and frontend. The backend is implemented in PHP, whereas the frontend uses JavaScript, HTML and CSS. Laravel is the chosen framework that builds a structure for the codebase. Additionally, SQL is chosen as database structure. Furthermore, Codeception is used as the testing framework.

To be sure the application works as it should, different kind of tests are used. First, unit and functional tests were used to validate that the application gives the correct output with its associated actions. Furthermore, bunq employees were asked to give their feedback. They could give their opinion about how the application works and looks. This feedback is taken into account when making adjustments to the application.

The final product solves the main issues of decreasing the workload and time spent on translating. However, there are some features that are not yet implemented in the final work. These features are recommended for future work. Even though not all features have been implemented, the project is regarded as a success by the development team.

Contents

1 Introduction

bunq is a Dutch internationally active digital bank, which officially received its banking license in 2015 [31]. A digital bank is a bank that eliminates the need to be at a physical location for every banking activity, process and program [14]. It offers the same services as a traditional bank, however, it takes less time and is cheaper. Over the years, bunq has expanded to many European countries, such as Germany, Spain and Italy. The goal of bunq is to eventually become a global player in the banking industry [13]. bunq primarily offers their financial services through its Android and iOS apps. In order to be accessible in multiple countries, the bunq mobile apps need to be available in multiple languages. Hence, each word and sentence in the app needs to be translated into multiple other languages. Currently, all steps needed to translate the sentences, called strings, are executed manually. First, the developers have to create a basic English string. Afterwards, the developers create a ticket for this string to be rewritten into a proper English string by another bunq employee. Next, this proper English string needs to be translated into all available languages and tickets for these tasks are created. bunq employees who fluently speak these languages need to translate all the English strings into their assigned language. Finally, all the translations are copy pasted into the code of both bunq apps.

The purpose of the project is to create a system to automate and speed up the string translation process and use crowdsourcing to gather translations for the app. Crowdsourcing is a practise to outsource your tasks to a crowd [1]. In this case, bunq will ask a crowd to translate their English strings to other languages. To accomplish this, the application should allow users to easily add, update and review strings. Completed strings need to be automatically inserted into the code of the Android and iOS apps.

This report documents the created product and the team's progress on achieving this system. First, the problem definition and analysis are given in Chapter 2, describing the assignment and requirements. Second, the research phase is discussed in Chapter 3. Chapter 4, the product design, describes the architecture of the system and gives an explanation of the database. A description of how the application is implemented follows in Chapter 5. Subsequently, Chapter 6 focuses on the testing techniques used by the team. Chapter 7 contains an evaluation of the final product. Furthermore, Chapter 8 describes the agile methodology and resources the team used. Chapter 9 contains the discussion and recommendations of the team regarding the project. Finally, our report is concluded in Chapter 10.

2 Problem definition and analysis

In order to create a fitting product for the client, it is important to fully understand the client's wishes. The following sections give a description of the problem, the derived assignment to solve the problem and requirements and design goals to meet the client's wishes.

2.1. Problem definition

bunq primarily offers their financial services through its mobile Android and iOS apps. These apps contain many different strings which are visible to the user, such as "Add a new bank account for you or for groups" in the Accounts view. Before a string is visible in these apps, multiple steps need to be taken. At the moment, the string translation process is very costly. The process is handled completely via GitLab[1] issues.

First, a developer creates a basic string in English when implementing a new feature. Thereafter, an issue is created for this new string in their GitLab repository. The string then needs to be reviewed by another employee, a copy writer. The copy writer has the task to create proper English sentences, so that all sentences have consistent terminology and correct grammar. The copy writer comments on the created issue whether the sentence is correct or provides a corrected version of the string. After the English string is corrected, it needs to be translated into all available languages in the bunq apps. The translators comment on the issue to provide the translations for every string. Currently, employees who fluently speak these languages translate the strings in addition to their own work. These translations are handed over to the developers, without being reviewed by another fluent speaker. Finally, the developers need to manually copy the translated strings and paste them into the corresponding `.xml` files of the Android app and `.strings` files of the iOS app. All translations are saved in the mobile app's repositories as `.xml` and `.strings` files.

Both apps make use of i18n (internationalisation) and l10n (localisation). Localisation determines which language to show in the app. In the bunq apps, the language is set to be the same as the device's language. Internationalisation is the process of making your app able to adapt to different languages [4]. In the Android app, for example, the strings for the different languages are stored in different folders, such as `values/` for the default language and `values-nl/` for Dutch [9]. The default language is used when the device's language is not available.

Due to the costly translation process, the following problems arise:

- Translating the strings is time-consuming
- Developers lose time by having to copy and paste the strings into the code
- Inconsistency mistakes between the apps are made when inserting the strings into the code
- Translations might be incorrect
- Fluent speakers are needed for each language

[1] https://gitlab.com/

Taking all the above issues into account, it is obvious that the current approach is inefficient. bunq is a young and ambitious company, eager to be the most innovative bank. This mindset is accompanied by deploying many new features in a short timeframe. In addition, the current workflow of translating strings is not scalable enough to fulfil bunq's ambition to be available worldwide.

2.1.1. The assignment

In order to solve the above mentioned issues, a more automated version of the string translation process needs to be created. More specifically, a web application has to be built where strings can easily be added, updated, translated and reviewed.

As mentioned above, the Android and iOS repositories contain all existing strings in the `.xml` and `.strings` files. Each string in these files has a key, which is different for Android and iOS. First, the different Android and iOS keys with the same string should be mapped, to know that they belong together. This way, when a string is updated using the web application, it will be adjusted in both files and thus in both apps. After the string is translated into all languages, the web application should automatically update the strings in the files so that the developers do not have to do this manually. This will save the developers time and prevent code consistency mistakes. In addition, translations could be obtained by using crowdsourcing. This makes it possible for trusted bunq users to help with the string translations by providing their own. Consequently, bunq employees are able to focus more time on their main work, since they no longer have to translate all the strings themselves. They only need to review the string translations obtained through crowdsourcing.

2.2. Problem analysis

To fulfil the assignment according to the client's wishes, requirements are prioritised according to the MoSCoW model. In addition, design goals are set for the project.

2.2.1. Design goals

In this section, the main design goals of this project are elaborated. The design goals should always be taken into consideration while implementing, to know what to focus on and for good code quality. The following sections contain the design considerations the team will keep in mind during the implementation of the web application:

Scalability
Since bunq is a rapidly growing company, it is crucial to keep scalability in mind. The application must be able to handle bunq's ambition and therefore be able to handle the new languages which bunq wants to launch in their apps.

Usability
Since one of the main goals of the project is to reduce the translations process time, it is important that users can easily perform prescribed tasks. The system should be easy to use, learn and control. This way the bunq employees are inclined to use it and the crowdsource users are willing to contribute to the translations. By performing usability tests, the application can be improved in accordance to the feedback given.

Compatibility
One of the system requirements is that it should be able to run on Safari 11.1 and Chrome 60.0 without problems or conflict. These two browsers are by far the most used worldwide [29]. bunq employees mainly use Safari as they work on Mac computers. Therefore, the system must be compatible with these browser versions.

Performance
Since a large amount of users will be using the system simultaneously when crowdsourcing is deployed, the application must respond quickly and correctly. If, for instance, a user updates a translation, the update should be done fast and not use too much memory. If the action takes too long, a user might refrain from using the application (often). When the user is a crowdsource user, this is unwanted since that user's translations are helpful. When the user is a bunq employee, this is also unwanted, since the application should improve the translation process.

Security
An important design consideration is security. In the first stage of implementing, the application is mainly created to be internally used by bunq employees. However, after deploying the crowdsourcing feature, the application should be secured so that the large amount of users are prevented from attacking the software. We achieve security by implementing the application as secure as possible so that it is resilient against possible attacks. An example of such an attack is a SQL injection. A way to prevent this is to use prepared statements and parameterised queries.

Maintainability
Since the application will be taken over by bunq, the code should be easily modifiable and understandable for bunq's developers. This will make it easy for them to update it in the future. The code written must be well documented and well tested to evaluate where problems may occur. The source code's quality is also important since it can influence many aspect of the code's maintainability. Low quality code might result in a badly performing system, for instance, or a reconstruction of the system. The code must therefore be of high quality. Lastly, in order for the team to easily understand each other's code, a consistent code style should be used by all members. This will increase readability and faster code reviews. Applying these plans ensures that the transferal of the system to bunq from the team will run smoothly.

2.2.2. Requirements
At the end of the project most of the requirements must be completed. The following goals are set:

- 100% of the must haves
- 75% of the should haves

2.2.2.1. Functional Requirements
Must haves

1. The product must have a complete overview of all the strings in both apps (iOS and Android)
2. The items in the overview must store the developer string (English), the reviewed string (English) and the translations for all the languages available in the apps (Dutch, German, Italian, Spanish) in separate fields
3. Each item must be mapped to the correct iOS and Android keys
4. The iOS and Android keys of the item must be stored in fields
5. Items which are not used/applicable in one of the apps must have a special key in that field to show that it is not available for that system
6. Developers must be able to add new items to the system via the GUI of the overview
7. Copy writers must be able to add reviewed strings to items
8. Translators must be able to add translations to items and edit them
9. When a translator edits an item and commits the changes in the system, the improvements must be committed automatically to the code

Should haves

1. A translation reviewer should be able to approve a crowdsourced translation
2. A translation reviewer's approval should be a final approval
3. A translation reviewer should be able to edit a crowdsourced translation
4. The translation should not be available to the developers if it has not been approved yet
5. The final approval of an item's translation should lock that translation and commit them automatically to the code

6. A crowd translator should be able to add new translations to items
7. The system should have a privilege system which determines what the users are allowed to modify in the overview
8. The roles of the privilege system should be: admin, developer, copy writer, translator, translation reviewer, crowd translator, crowd approver
9. Users should be able to have multiple roles
10. An admin should be able to add new languages to the overview
11. An admin of the system should be able to unlock an item
12. An admin of the system should be able to delete an item

Could haves

1. The overview could contain the strings and their translations from the backend (server responses)
2. The backend items could be mapped to their keys from the code
3. The system could save the history of the items and its translations
4. The users could place comments when reviewing a translation
5. Crowd approvers could approve a translation
6. Crowd approver's approvals could not be the final approval
7. A translation could have 2 crowd translator approvals before the translation reviewer can approve it

Would haves

1. Priority would be assigned to items
2. Items belonging to one feature would be grouped together

2.2.2.2. Non-functional requirements

1. The decision-making has to be documented
2. For every feature, tests have to be written
3. At least 70% of the code should be tested
4. Only bunq customers can be used as crowdsourcing translators
5. The product is scalable to new spoken languages
6. The product is able to run on Safari 11.1 and Chrome 60.0
7. The code should be reviewed via SIG
8. The product is available at all times after deployment

3 Research

The first two weeks were dedicated to research. This chapter summarises the findings of this analysis. The goal of the project is to create an application for string translations that automatically updates the translations in the code of the bunq apps and can eventually make use of crowdsourcing. In Section 3.1, an explanation with analysis is given on which language is used to implement the application. Sections 3.2 and 3.3 describe the available solutions and explain the reasoning behind the choice of technology per component of the system. Section 3.4 explains why crowdsourcing was chosen instead of machine translations and illustrates a few examples of already-existing crowdsourcing translation systems. Section 3.5 describes which testing frameworks were considered to test the application. Finally, all decisions are concluded in Section 3.6.

3.1. Language

Before the web application can be created, the team needs to decide which programming language to use. A distinction is made between backend and frontend languages.

3.1.1. Backend

Some of the most commonly used languages for web applications are Java, PHP and Python [22]. The sections below discuss the pros and cons of these languages.

Java [1]

The main reason to choose Java is because the team members are all experienced Java programmers. Java is an object-oriented language used for developing all kinds of applications. As it is widely used, a lot of information and support is available. It is a compiled language, which means that the execution time is faster than for interpreted languages. Furthermore, the Java language is designed to enforce type safety. Finally, Java makes use of JDBC to connect to databases, which makes it possible to connect to a wide-range of databases.

A disadvantage of using Java is that the language is not used by bunq. Therefore, it will be harder to maintain the application after the team has left.

Advantages	**Disadvantages**
All team members are experienced Java programmers	Not used by bunq
Widely used, a lot of support available	
Good database support	
Suitable for all kinds of applications	
Compiled language, execution time is faster	
Type safety	

Table 3.1: The advantages and disadvantages of Java

[1] `https://www.java.com/en`

PHP [2]

PHP is specifically made for web development and is still the most used language for server-side web applications [6]. It is also the programming language bunq uses for their banking backend in an object-oriented way. Since PHP is so widely used, a lot of up to date information and support is available. Therefore, PHP is relatively easy to learn. PHP is a interpreted language, which means no compilation time is needed. Furthermore, the web application needs a database. PHP has built-in support for over 20 databases, including the most popular ones, such as MySQL and PostgreSQL [18].

A disadvantage of PHP being an interpreted language is that the execution time could be slower than that of a compiled language. Moreover, the team is not very experienced with the language.

Advantages	Disadvantages
The main programming language bunq uses	Interpreted language, execution time is slower
A lot of up to date information and support available	The team members are not very experienced
Built-in support for over 20 databases	
Specifically made for web applications	

Table 3.2: The advantages and disadvantages of PHP

Python [3]

Python is an interpreted, dynamic language that requires a relatively small effort to learn, write and read. It is a high-level language for general-purpose programming. Advantages relevant for the project are that Python is widely supported, has many libraries and is object-oriented as well.

Similar to PHP, there is no compilation time but execution time could be slower. Another disadvantage is that Python has limitations with database access because the database access layer is underdeveloped [28].

Advantages	Disadvantages
Quite easy to learn	Not used by bunq
Widely supported	Interpreted language, execution time is slower
Has many libraries	The team members are not experienced

Table 3.3: The advantages and disadvantages of Python

3.1.2. Frontend

The frontend of the web application contains everything the user sees and interacts with. There are a few languages that are essential for every web application. First, HTML is the standard markup language for creating web applications. It provides an overall framework of how the webpages will look. In addition, CSS is used to style the webpages more extensively. Together, these two languages form the basic structure of the static webpages. In order to make the static pages dynamic, a scripting language is needed. Multiple scripting languages are available, however JavaScript is by far the most used alternative (95% of all websites use it) [3], which makes it the best choice for us. It is compatible with all browsers, has the most documentation and the richest feature set.

3.1.3. Decision

The team decided to use PHP as the backend programming language. The final product will be a relatively small web application, where the database plays a crucial role. Therefore, Python is less suitable because its database access layer is underdeveloped. Java and PHP are both appropriate for the application's goal, because they have good built-in support for databases and work well for small-scale applications. The main reason to choose PHP over Java is because bunq uses PHP as their banking backend programming language. This way, it will be easier for bunq to maintain the team's code and integrate it into their own codebase. Furthermore, PHP is specifically designed for web development. Table 3.4 shows an overview of the relevant

[2]`http://php.net`

[3]`https://www.python.org`

criteria for the project of the discussed backend programming languages (++ is a good rating, - - is bad).
For frontend programming HTML, CSS and JavaScript will be used, as these languages are essential.

Backend language	**Java**	**PHP**	**Python**
Used by bunq	- -	++	- -
Level of knowledge and learning curve	++	+	+-
Amount of support available	++	++	++
Database support	++	++	-
Suitable for web applications	+	++	+

Table 3.4: Overview of the criteria for the backend language with ratings

3.2. Web application frameworks

After determining the languages, a choice of frameworks has to be made. For the implementation of the web application, multiple frameworks can be used to support the development. These frameworks provide basic structuring tools needed to create web applications. Using these frameworks enables us to focus more time on implementing the requirements, instead of recreating an already existing structure.

3.2.1. Backend

As described in the previous chapter, PHP will be used as the backend programming language. Currently, multiple PHP frameworks are available, of which CodeIgniter, Laravel and Symfony are the most commonly used [10]. The sections below summarise the pros and cons of each framework.

CodeIgniter [4]

CodeIgniter is a relatively simple, lightweight framework, which makes it easy to learn and to set up. Additionally, CodeIgniter is well-documented. It provides a number of libraries, but not as many as other PHP frameworks such as Laravel and Symfony [8]. However, this is only logical given that it is lightweight.

A disadvantage of CodeIgniter is that the database support is rather limited. It does not come with an Object-Relation Mapping (ORM), which is used to easily map database records to objects in the code. Furthermore, CodeIgniter does not make use of Composer as PHP dependency manager, which makes it harder to use third-party database libraries. Finally, it does not have any built-in unit testing tools, thus additional testing tools will be needed.

Advantages	**Disadvantages**
Simple and lightweight, easy to learn and set up	Limited database support, no ORM present
Well-documented	Does not come with Composer
	No built-in unit testing tools

Table 3.5: The advantages and disadvantages of CodeIgniter

Laravel [5]

Laravel is the most popular PHP framework [10], which means that a lot of information and support is available. It is designed to quickly build web applications using the Model-View Controller (MVC) architecture. The learning-curve is small, however CodeIgniter is easier to learn. Laravel is built on several Symfony components, which means that it provides a solid base of well-tested and reliable code. It makes use of its own library Eloquent for ORM features. By using this, Laravel works well with relational databases such as MySQL. In addition, Laravel uses Composer as a PHP dependency manager, which makes it easy to use third-party libraries. Composer is also used by bunq. Finally, Laravel provides accessible, powerful tools, such as an integrated unit testing tool, which help build robust applications.

One disadvantage of Laravel is that it is not as stable as frameworks like Symfony. There is a possibility that version upgrades might cause some problems with the code.

[4]https://codeigniter.com
[5]https://laravel.com

Advantages	Disadvantages
A lot of documentation available	Not as stable as Symfony
Solid base of well-tested and reliable code	
Uses Eloquent as ORM	
Good database support	
Uses Composer as a PHP dependency manager	
Built-in unit testing tools	

Table 3.6: The advantages and disadvantages of Laravel

Symfony [6]
Symfony is another much used PHP framework. It is well-documented and has a large community. The upsides of using this framework are that it is adaptable, scalable and very extensive [24]. Different software components can be selected to be used in the application. These components are prefabricated and can be quickly integrated in the system. Symfony uses the third-party Doctrine as ORM. However, this ORM is more difficult to learn than Laravel's Eloquent. Nevertheless, Symfony has the best database support out of the three. It works well with both SQL and NoSQL databases. Furthermore, Symfony also uses Composer as dependency manager. Finally, unit testing tools are included out of the box.

Symfony has a few disadvantages. First, it has a very steep learning curve because of its size [24]. Furthermore, it is more meant for larger web applications, which is not really applicable for the project.

Advantages	Disadvantages
Well-documented	Very steep learning curve
Adaptable, scalable and extensive	Meant for larger web applications
Uses Doctrine as ORM	
Supports many types of databases	
Uses Composer as a PHP dependency manager	
Built-in unit testing tools	

Table 3.7: The advantages and disadvantages of Symfony

3.2.2. Frontend

Frontend frameworks provide a basis for the web application design. Bootstrap and Foundation are the two most used frameworks [16]. Therefore, a decision will be made between these two based on their pros and cons.

Bootstrap [7]
Bootstrap is the most popular frontend framework, since most companies prefer Bootstrap to other Frontend frameworks [16]. Using Bootstrap as a frontend framework shows to have multiple advantages. First, it has a massive community support because of its popularity. Second, it has a great variety in templates. In addition, it is responsive, lightweight and has a well-functioning grid system, which means that it renders well on many different screen sizes.

One of the disadvantages of Bootstrap is that it requires a lot of effort to deviate from the standard Bootstrap styles. Many Bootstrap applications have similar appearances.

Advantages	Disadvantages
Massive community support	Deviating from the standard Bootstrap styles takes a lot of effort
Great variety in templates	
Responsive	
Lightweight	

[6]`https://symfony.com`
[7]`https://getbootstrap.com`

Has a well functioning grid system	

Table 3.8: The advantages and disadvantages of Bootstrap

Foundation [8]

The second most used frontend framework is Foundation. This framework offers customisation abilities and it possesses a robust grid system. In addition, it is responsive and looks good on all screen sizes. Finally, is easier to create unique webpages with Foundation than with Bootstrap.

A disadvantage of Foundation is that customisation is quite complex. Therefore, Foundation is more difficult to use for unexperienced developers. Furthermore, the community is not as big as Bootstrap's community, which means that less support is available.

Advantages	**Disadvantages**
Easier to create unique webpages	Customisation is quite complex
Has a robust grid system	More difficult to learn and less support

Table 3.9: The advantages and disadvantages of Foundation

3.2.3. Decision

The team decided to use Laravel as the backend framework. Symfony does not seem appropriate for the project, since it has a steep learning-curve and is meant for larger web applications. Since the project only takes 10 weeks, it is important to use a framework that can quickly be learned and used. CodeIgniter has limited database support, no ORM, and no built-in unit testing tools. Since these aspects are important for the project, CodeIgniter does not seem suitable either. Laravel has good documentation, a decent learning-curve, a built-in unit testing tool and works well with SQL databases. The Eloquent ORM makes database interactions easier. Given the amount of time available for the project, the Laravel framework seems to be the right choice. Table 3.10 shows an overview of the relevant criteria for the project of the mentioned backend frameworks (++ is a good rating, - - bad).

Backend framework	**CodeIgniter**	**Laravel**	**Symfony**
Learning curve	++	+	- -
Documentation and support	++	++	++
ORM features present	- -	++	++
Database support	-	+	++
Uses Composer	- -	++	++
Built-in unit testing tools	- -	++	++

Table 3.10: Overview of the criteria for the backend framework with ratings

For the frontend framework, the team has chosen to use Bootstrap. It is currently the most popular framework and provides a lot of support. The disadvantage of Bootstrap is that it is hard to create a unique style. However, this is not very important for this project. Foundation is more complex and harder to learn, which makes it less aappropriate for us. Table 3.11 shows an overview of the relevant criteria for the project of the discussed frontend frameworks (++ is a good rating, - - bad).

Frontend framework	**Bootstrap**	**Foundation**
Learning curve	+	-
Documentation and support	++	+
Grid system	++	++
Responsive	++	++
Create unique style	-	+

Table 3.11: Overview of the criteria for the frontend framework with ratings

[8] `https://foundation.zurb.com`

3.3. Database

To be able to store all information given by the web application, a database is needed. There are different structures for databases. The two main structures are the relational, SQL, and non-relational, NoSQL, data structures [19]. To determine which one is better for the application, a comparison of these two structures is given below.

3.3.1. SQL vs. NoSQL

SQL

Structured Query Language (SQL) databases have a fixed schema and store data in a traditional row and column format in a table [15]. This structure supports vertical scalability. In other words, the performance of a single server can be increased by adding resources like memory [25]. To retrieve data from different tables, they need to be linked together by a join operation. This can be a time consuming process. SQL also supports horizontal scalability if data partitioning is implemented. Sharding is a horizontal partitioning method that splits the data of one table into two tables with the same data structure where the data is separated based on the partition key [33]. These tables can be stored on multiple servers, hence, SQL supports horizontal scalability. However, cross-table joins are inefficient since they could require multiple servers, causing a slow process. SQL is a mature technology, therefore many issues have been fixed. For example, security features are already well incorporated in SQL. A few examples of SQL databases are:

- MySQL
- PostgreSQL
- SQLServer

NoSQL

NoSQL databases, on the other hand, are schema free. Therefore, they have a flexible schema design that can handle a variety of data [15]. This structure supports horizontal scalability, meaning more servers can be added to share the systems load [25]. As for data retrieval, the data is stored in objects. Thus, all the related data is stored into one object. Therefore, this process is time efficient. Finally, NoSQL has not matured yet. As a consequence, security issues may arise. Some popular NoSQL databases are:

- MongoDB
- Redis
- CouchDB

3.3.2. Decision

SQL	**NoSQL**
Fixed schema	Schema free
Vertical (and horizontal) scalability	Horizontal scalability
Retrieving data may be time consuming	Retrieving data is time efficient
Matured technology	Not matured technology

Table 3.12: SQL versus noSQL

A database is needed for the web application. Comparing the SQL and NoSQL structures in Table 3.12, the team came to the conclusion to use a SQL database. Even though the NoSQL retrieval process is not as time consuming as SQL, the relational aspect of SQL comes in handy with its traditional tables. The web application's database will store strings with their iOS and Android keys, and all the current translations. A reason why the team decided to use SQL instead of NoSQL, is that SQL is more mature and it is more secure [25]. Since the data in the application's database will be consistent, there is no need for the flexible scheme NoSQL offers; a fixed scheme can easily be used. Because the decision is to use SQL, one of the examples stated above will be used. MySQL is the most popular one. Therefore, the team has also chosen this option. SQLServer is a data platform of Microsoft. During the project only macOS will be used, making it more suitable to use another option. Furthermore, bunq uses MySQL and might want to merge the data from the application's database into their database in the future. Taking all of the above into consideration, the final decision is to use MySQL as the web application's database.

3.4. Crowdsourcing

At the moment all translations are done by bunq employees. To decrease their work load, either machine translation or crowdsourcing can be used. Machine translations are translations created by computers without any human involvement [30]. Both these options have the characteristics high volume, quick translation output and low cost [12]. Crowdsourcing has the main advantage compared to machine translation when it comes to accessing uncommon languages [34]. Crowdsourcing also has the advantage that it overall results in a higher quality translation than what the machine translation outputs [12]. One disadvantage of crowdsourcing is that it may take some time before a translation is completed. However, bunq does this manually anyway, thus the time they spend on translating the string still takes longer than using crowdsourcing. Therefore, it is in bunq's favour to use crowdsourcing for their translations over machine translations.

3.4.1. Examples

A few systems already exist that implement crowdsourcing for translation. The following paragraphs shortly discuss a few examples of these systems.

Transifex [9]
Transifex is a web-based translation platform. To translate a specific piece of text you can either upload a file or the translation is pushed onto the API. The next phase is the translation management. During this phase the text will be translated, whereafter reviewed. This is done by a team consisting of translators, reviewers and coordinators, who are grouped based on the language they are working on. When the translation is complete, the translated files are pulled in your repository. Now the content is ready for publishing. You can choose if you want your text to be public or private. In other words, everyone or only certain people can work on the translation. Another convenient feature is the ability to prioritise resources. You can give the following three priorities: normal, high and urgent.

MTurk [10]
Another example is Amazon Mechanical Turk, MTurk. MTurk is an online marketplace that enables coordination between programmers with the public to create tasks via crowdsourcing [11]. A 'requester', developer, creates a Human Intelligence Task, HIT. In the case of crowdsourcing translations, a HIT is a task where a turk user is provided with sentences in the source language that needs to be translated into the target language [11]. Rewards are added to make working on specific strings more attractive. A way to reassure quality, is only allowing workers who have historically answered a specific percentage of correct HITs. At the end, the company will have its translated strings which will be verified by the company before payment.

3.4.2. Recommendations

The application could have similar features when bunq decides to use crowdsourcing for the translations and reviews of the strings. To give the users more incentive to help translate, a rewarding system can be added. bunq could for instance give users certain rewards which reflect on the translations help they provided. Furthermore, priorities can be assigned to strings in a visual way on the application. This feature is specifically convenient for strings which should be translated in a short timeframe. This can thus be used to communicate the urgency of translating certain strings to non-employees of bunq. Finally, the users' knowledge could be validated by adding languages tests to the registration process of new users. When the user answers a certain percentage of the test correctly, they are able to contribute. This ensures the quality of the translations for the bunq apps.

[9] https://www.transifex.com
[10] https://www.mturk.com

3.5. Testing

Another aspect that is important to consider is how to test the newly created web application. Not only is it important to verify that the application works as intended, it can also ensure more maintainable code, less security issues and better performance. Since the web application will be written in PHP, the following options are the most popular testing frameworks available for this language [21, 23, 32].

3.5.1. Testing frameworks

The next paragraphs discuss each option's technology and what it is typically used for. The following list contains the types of tests executed with these frameworks:

- Unit tests
- Functional tests
- User acceptance tests

PHPUnit [11]
PHPUnit is the most popular testing framework for PHP applications for unit testing. However, it is not possible to execute other types of tests such as user acceptance tests. This is also the testing framework bunq uses for their backend.

Simpletest [12]
Simpletest is, as the name implies, a simple PHP unit testing framework. It is also known to be easier to set up than PHPUnit. The framework supports SSL, forms, proxies, frames and basic authentication.

Behat [13]
Behat is a testing framework mainly for user acceptance tests. It is inspired by the Cucumber testing framework, where instead of writing code-like tests, the developer writes story-like tests. Therefore, even non-developers are able to write tests, because the tests are human readable describing the desired behaviour of the system.

Selenium [14]
Selenium makes it easier to write user acceptance tests, specially for web applications. One capability of this testing framework is automating browsers. It runs in many browsers and operating systems.

Codeception [15]
Codeception is another popular testing framework for PHP applications which is often used for unit, functional and acceptance testing. It also supports PHP development frameworks such as Laravel and Symfony. The user acceptance tests are executed with the Selenium WebDriver, to make user acceptance testing possible. Not only is Codeception integrated with Selenium, it is also built on top of PHPUnit to execute the unit tests.

[11] https://phpunit.de
[12] http://simpletest.org
[13] http://behat.org/en/latest/
[14] https://www.seleniumhq.org
[15] https://codeception.com

3.5.2. Decision

Framework	Unit tests	Functional tests	Acceptance tests	Notes
PHPUnit	✓	✗	✗	
Simpletest	✓	✗	✗	
Behat	✗	✓	✓	High-level functional tests based on user stories
Selenium	✗	✓	✓	Web browser automated tests
Codeception	✓	✓	✓	Automated tests with and without a webserver

Table 3.13: Test framework comparison

Since Laravel is chosen to be the backend web application framework, it would be a good choice to use Codeception. Not only is the integration with Laravel an advantage, the fact that Codeception supports unit, functional and acceptance testing, gives it a preference because the other popular testing frameworks usually only support one type of tests.

3.6. Conclusions

After the research phase, the following decisions are made. The application will be built using Laravel as backend framework and Bootstrap as frontend framework. The backend will be written in PHP. Whereas, the frontend languages will be HTML, CSS and JavaScript. As for the data storage, a MySQL database will be used. Lastly, the application will be tested with the help of the Codeception testing framework.

4 Product design

This chapter describes the design of the created product. First, the two main components of the system are explained. These components will hereby be known as the 'initialisation feature' and the 'string translations application'. Secondly, the system's architecture will be discussed. Thereafter, the structure of the database will be explained and which objects are present within this structure. Lastly, an overview of the application's interface is given.

4.1. System components

The application has one main purpose: to decrease the workload of the bunq employees for the string translation process. In order for the application to work accordingly, a functionality must be created which inserts the existing strings into the application's database. Due to the complexity of this feature, the application can be divided into two parts.

First, the application should be a translations application, in which the current string base of bunq's mobile apps is extended by either adding new strings or a new language. Furthermore, the application must be able to allow the users (bunq employees) to create translations of these strings, review them and commit them to the mobile apps' repositories. The users should be able to do this without having to temper with the codebase of the apps. The system thus automates a part of the process of translating and therefore decreases the manual labour of extending the string base of the current bunq apps. This corresponds to the main purpose of the application.

Next, the application should contain an initialisation feature, in which bunq employees can create a mapping between the iOS and Android keys. This feature initialises the database of the application. When the mapping is finalised, all the current English strings in bunq's Android and iOS apps are identical to each other. With this initialisation feature, the user is also able to work through the strings of the other available languages and create consistency between both apps. This way all strings of all current available languages are consistent in the apps' code and in the string translations application's database.

4.2. System architecture

The application is divided into components. These components are combined using the Model-View-Controller (MVC) architecture, as shown in Figure 4.1. The application has this architecture due to the Laravel framework[1] on which the application is built. When a user requests a certain page, this request is sent to the controller. The controller then handles the user request and retrieves the needed data from the models. Subsequently, it returns a view to the user, which renders the webpage. The models interact with the database. In order for the users to be able to interact with the application, routes must be added to map the URLs to actions defined in the controllers. The URLs are then accessible to the users via their browser.

The application consists of several models, controllers and views. Table 4.1 shows an overview of all the models, controllers and database tables. The views are represented in a sitemap, shown in Figure 4.4.

[1]https://laravel.com

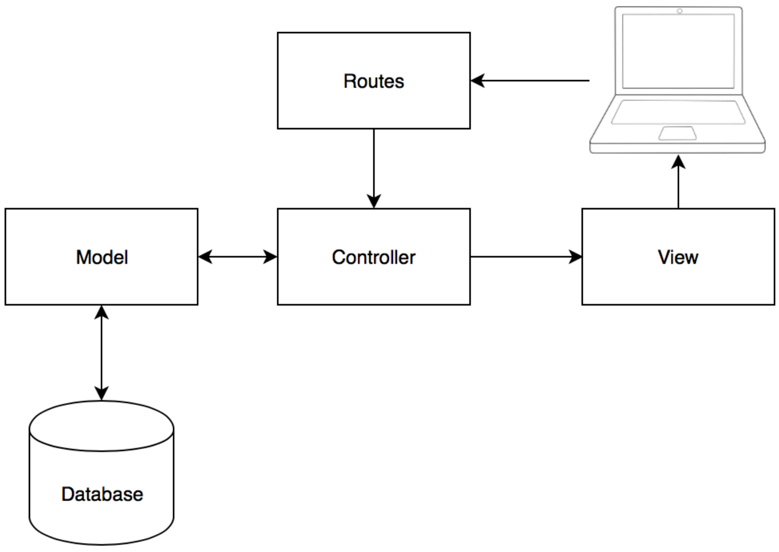


Figure 4.1: MVC model

Models	Controllers	Database tables
Key	KeyController	keys
Item	ItemController TranslationsController ReviewController	items
Language	LanguageController	languages
Path	PathController	paths
Feature	FeatureController	features
Placeholder	-	placeholders
ItemPlaceholder	-	item_placeholder
User	UserController	users
-	HomeController	-
-	InitHomeController	-
-	InitKeysController	-
-	InitTranslationsController	-

Table 4.1: An overview of the associating models, controllers and database tables

4.3. Database schema

The database plays a big part in our system. There is intensive communication between the application and the database. Some views depend on the requested queries output. Before the string translations application can be used, the database has to be initialised, as mentioned in Section 4.1. The initialisation will populate the database with the existing strings of the bunq apps. The string translation feature will search if there are any English strings which are translatable and not yet translated in one of the available languages. An English string is translatable if it can be translated into other languages. For example, a name such as "bunq" is not translatable. To make extension possible, the databases design is scalable for adding new languages and new strings. In Figure 4.2, the database schema can be found. Below all the objects of the database are explained.

Language
A Language object consists of a primary key, `id`. Additionally, it has two unique attributes, the `name` and its `abbreviation`. Within the application, the languages are shown in the views so that the users are able to

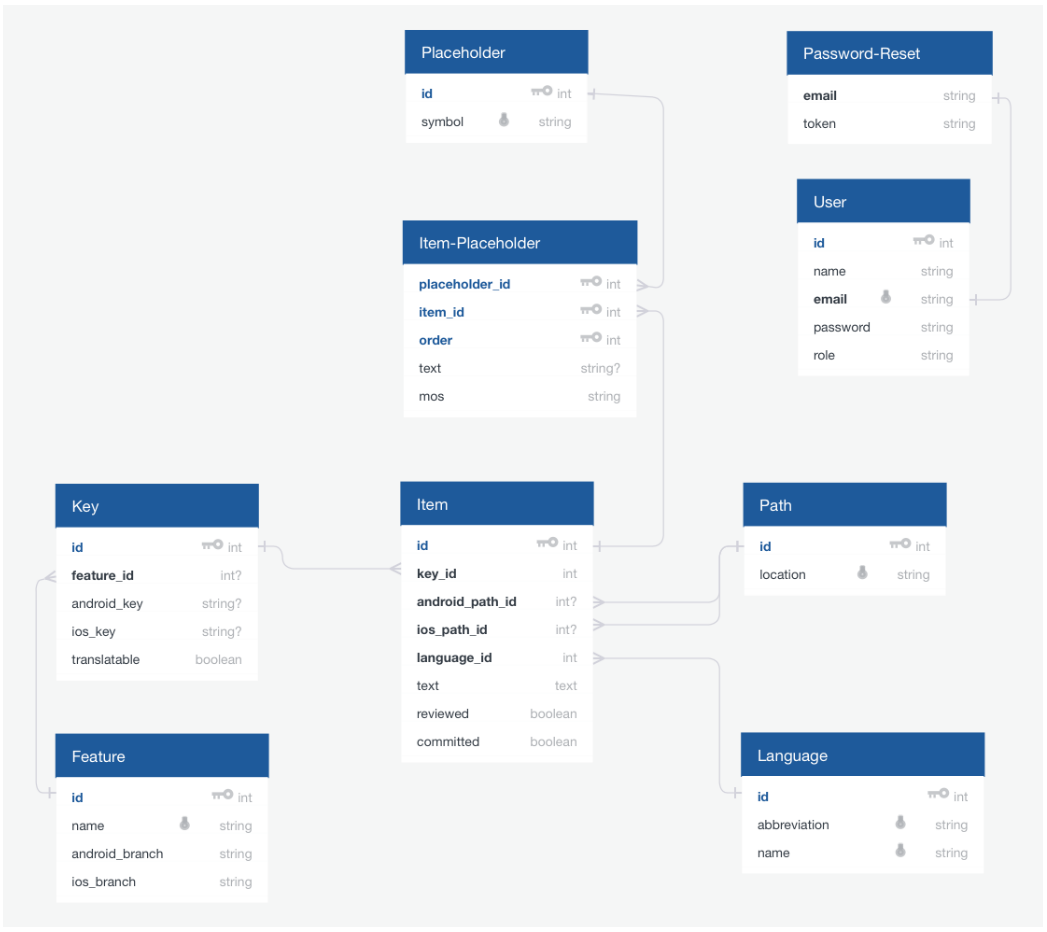


Figure 4.2: Database scheme

choose which language they want to translate. The `abbreviation` attribute is used to determine the path of the files of that Language. In bunq's repository different language files are stored in folders of the according language. In Android these folders are called `values` for the English files and `values-[language abbreviation]` for all the other languages. In iOS the English strings are stored in the Base.lproj folder. The other languages folders, the `[language abbreviation].lproj` is the name of the folder. For example, `values-nl` and `nl.lproj` store all the files containing the Dutch strings.

Path
A Path object has three attributes. To begin with the `id`, which is its primary key. Secondly, it has a unique `location` in the repository of bunq. This location is unique per language and file. When a user adds a string in the application, they have to choose where this string will be saved. Consequently, the string will be saved in the file of the chosen path.

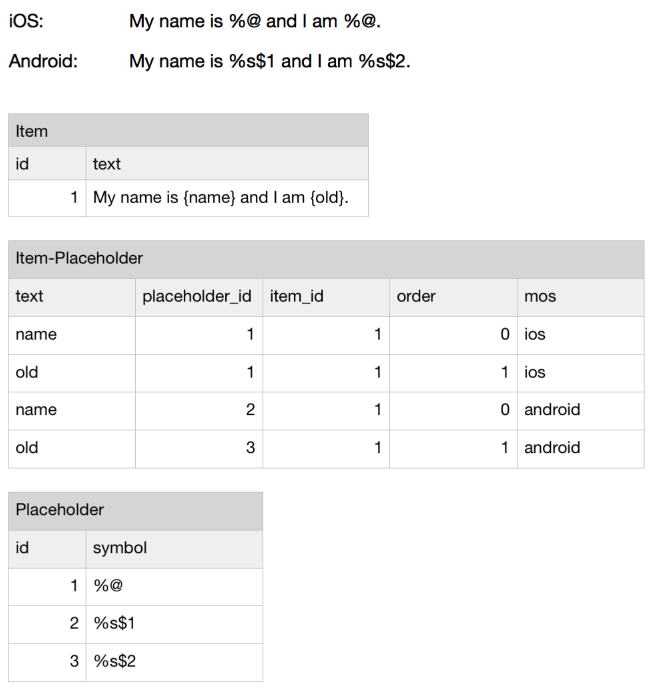
Key
There are strings in Android which are equivalent to strings in the iOS bunq app, and vice versa. These equivalent strings are mapped in the Key object. The primary key of the Key object is its `id`. A Key object has an `android_key` and `ios_key` as attributes. These keys refer to the strings in the bunq app. One of these attributes may be `null`, because there are strings which only exist in one of the bunq apps. Additionally, a Key consist of a `translatable` field, which indicates if the string belonging to that key is translatable in other languages than English, as explained in Section 4.3. Finally, the `feature_id` is a foreign key which refers to a Feature object; read more about this object later on.

Item

Just like the other objects, Item has an `id` attribute as primary key. All translations of a string are stored as an Item object. The `text` attribute contains the string value. An Item object has several foreign keys. To start with a `key_id` which refers, as the name implies, to a Key object. If the Key is translatable, Items can be created for all languages in the system, with the same `key_id`. All these strings have to be written to the bunq apps. Depending on the string's exclusiveness, both or one of the following fields is filled in: the `android_path_id` and the `ios_path_id`. These `path_id`s refer to the Path objects in what file that string occurs. The `reviewed` and `committed` attributes are booleans, stating whether the string has been reviewed and whether the string has been committed, respectively.

Placeholder
Some strings in the bunq app may contain a placeholder. Different formats are used in the code of the Android and the iOS app. These formats are saved in the Placeholder object as `symbol` and they have to be unique. Additionally, it has an `id` attribute as primary key.

Item_placeholder
A pivot table, Item_placeholder, exists to map an item to its corresponding placeholders. This table is added, because of the many-to-many relationship between the Item objects and Placeholder objects. For instance, one Item object may contain multiple placeholders in its text attribute. Additionally, one Placeholder object occurs multiple times in different Item objects or even the same object. A row in this table contains the foreign keys `item_id` and `placeholder_id`. Because there could be more placeholders in one item an `order` attribute is added to identify at what place the placeholder occurs in the item. The `text` attribute matches the placeholder name occurring in the `text` attribute of the Item surrounded by curly brackets. An example can be found in Figure 4.3. Finally, there is a `mos` attribute which indicates the mobile operating system (Android or iOS).



iOS:	My name is %@ and I am %@.
Android:	My name is %s$1 and I am %s$2.

Item	
id	text
1	My name is {name} and I am {old}.

Item-Placeholder				
text	placeholder_id	item_id	order	mos
name	1	1	0	ios
old	1	1	1	ios
name	2	1	0	android
old	3	1	1	android

Placeholder	
id	symbol
1	%@
2	%s$1
3	%s$2

Figure 4.3: Database scheme

Feature
Keys and their corresponding Items can be grouped together in one Feature. This object is created to commit multiple string and key changes that belong together all at once. A Feature object has an `id` as primary key, a unique `name` attribute and an `android_branch` and `ios_branch` attribute. These branches refer to the branch names of the repositories on which the changes should be committed. Keys and Items do not have to

be bound to a feature, a Key without a Feature can be committed as well.

4.4. GUI

An important part of the application is the interface. The interface must be self-explanatory and user-friendly. Therefore, the design must be simple. Figure 4.4 shows a sitemap of all the views. The home screen shows a few buttons, these buttons all refer to another view. The names of the buttons are self explanatory. However, should there be any unclarities, information can be found on the help page.

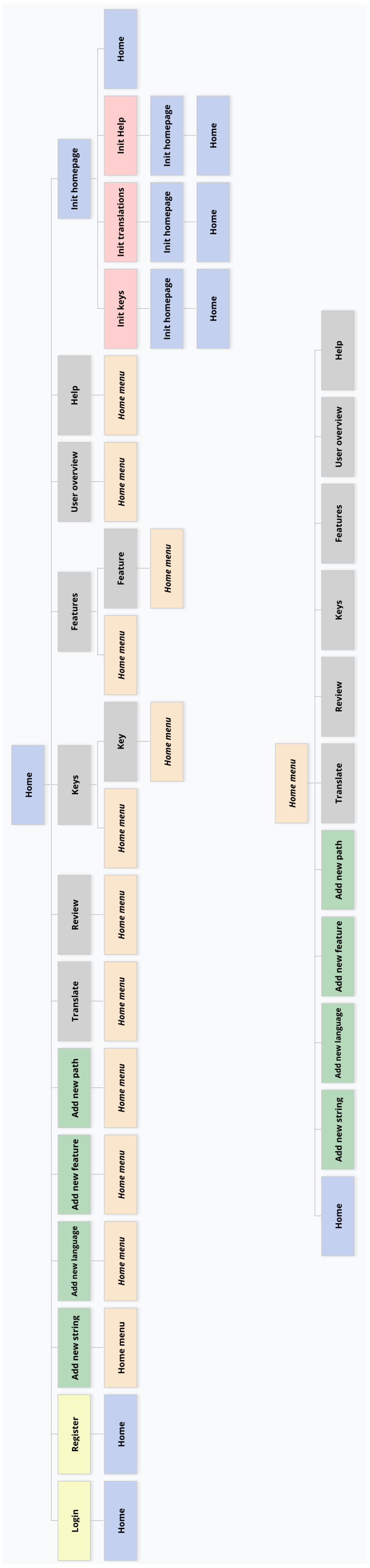


Figure 4.4: The sitemap of the application.

5 Product implementation

Decisions about the product implementation are explained in this chapter. First, the usage of Laravel is explained. Secondly, the decision why Traits are preferred over inheritance is discussed. Moreover, security integration is explained. Lastly, problems that occurred during the implementation phase and their solution are mentioned in the final section.

5.1. Laravel framework

As stated in Section 4.2, the application is built upon the Laravel framework. Laravel provides automated actions so that creating a web application is simplified. The following subsections discuss how the application is built using Artisan Console[1] and how the application was implemented within this MVC architecture. Artisan Console is the command-line interface which is included in Laravel. Several commands are provided to assist the development of the application. These commands can be used to auto-generate specified migrations, models and controllers.

5.1.1. Migrations

A migration is a class that structures a database table. Migration classes contain two methods, namely `up()` and `down()`. The `up()` method is used to add new tables or columns to the database, whereas `down()` is used to reverse the operations of `up()`. These two methods are used to inform Artisan how to create this table. Below is an example of a migration class for the Item table.

```
class CreateItemsTable extends Migration
{
    /**
     * Run the migrations.
     */
    public function up()
    {
        Schema::create('items', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('key_id')->unsigned();
            $table->integer('android_path_id')->unsigned()->nullable();
            $table->integer('ios_path_id')->unsigned()->nullable();
            $table->text('text');
            $table->integer('language_id')->unsigned();
            $table->boolean('reviewed')->default(false);
            $table->boolean('committed')->default(false);
            $table->timestamps();
        });

        Schema::table('items', function (Blueprint $table) {
            $table->foreign('key_id')->references('id')->on('keys');
            $table->foreign('android_path_id')->references('id')->on('paths');
            $table->foreign('ios_path_id')->references('id')->on('paths');
            $table->foreign('language_id')->references('id')->on('languages');
```

[1]https://laravel.com/docs/5.6/artisan

```
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down()
    {
        Schema::table('items', function (Blueprint $table) {
            $table->dropForeign('items_key_id_foreign');
            $table->dropForeign('items_android_path_id_foreign');
            $table->dropForeign('items_ios_path_id_foreign');
            $table->dropForeign('items_language_id_foreign');
        });
        Schema::dropIfExists('items');
    }
}
```

With the help of this migration, the Item table is created when the `php artisan migrate` command is used. This command automates the creation of the tables in the specified database. After running this command, the Item table has eight columns, namely `id`, `key_id`, `android_path_id`, `ios_path_id`, `text`, `language_id`, `reviewed` and `committed`. The constraints of the table columns can be specified such as `android_path_id` is specified to be unsigned and nullable in the example.

The `foreign()` function is used to specify the foreign keys of the table. In the example above, the Key table is connected to the Item table via the `key_id` column, such as the Path and Language table are connected with `path_id` and `language_id`, respectively. The `dropForeign()` function in the `down()` function of the migration ensures that the foreign key is dropped in that table.

Tables with a many-to-many relationship can be connected by a pivot table. A table is used to connect the Item and Placeholder tables in the application. Below is the migration which creates this a pivot table.

```
class CreateItemPlaceholderTable extends Migration
{
    /**
     * Run the migrations.
     */
    public function up()
    {
        Schema::create('item_placeholder', function (Blueprint $table) {
            $table->integer('item_id')->unsigned();
            $table->integer('placeholder_id')->unsigned();
            $table->integer('order')-> unsigned();
            $table->string('text')->nullable();
            $table->string('mos');
        });

        Schema::table('item_placeholder', function (Blueprint $table) {
            $table->foreign('item_id')->references('id')->on('items');
            $table->foreign('placeholder_id')->references('id')->on('placeholders');
            $table->primary(['item_id', 'placeholder_id', 'order']);
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down()
    {
        Schema::table('item_placeholder', function (Blueprint $table) {
            $table->dropForeign('item_placeholder_item_id_foreign');
            $table->dropForeign('item_placeholder_placeholder_id_foreign');
            $table->dropPrimary();
        });
        Schema::dropIfExists('item_placeholder');
    }
}
```

5.1.2. Models

In order for the application to make modifications in the database tables, models are needed. Each model class is connected to a table in the database and with these classes, the controllers (which will be explained in the next subsection) can perform multiple actions upon the data. Models can also be created via Artisan with `php artisan make:model`. An example of such a model is given below:

```php
class Key extends Model
{
    /**
     * All fillable fields of Key.
     */
    protected $fillable = ['android_key', 'ios_key', 'translatable', 'feature_id'];

    /**
     * Get all Items that belong to this Key.
     *
     * @return HasMany
     */
    public function items(): HasMany
    {
        return $this->hasMany('App\Models\Item');
    }

    /**
     * Get the Feature this Key belongs to.
     *
     * @return BelongsTo
     */
    public function feature(): BelongsTo
    {
        return $this->belongsTo('App\Models\Feature');
    }
}
```

Within these models, it is possible to create multiple methods to specify the relationship between the different models. In the example above, the functions `items()` and `feature()` are created to denote the one-to-many relationship between `Key` and `Item` and the many-to-one relationship between `Key` and `Feature`. The methods can be used by the controllers to obtain the items belonging to that key and the feature to which that key belongs, respectively.

Other than these methods, the columns that are allowed to be filled in by the controllers can be specified in the model. In the example, the fields of the Key table which are fillable are `android_key`, `ios_key`, `translatable` and `feature_id`. The controllers are thus able to alter or populate these columns' values in the Key table.

5.1.3. Controllers

All logic of the application resides in the controllers. As briefly mentioned in 5.1.2, the controllers are the classes which handle the creation, reading, updating and deleting of the objects in the database. Therefore, the controllers are given the name CRUD controllers (Create, Read, Update and Delete controllers). The command to create a controller is `php artisan make:controller`. A controller can be created on the basis of a model. Below is an example of a controller created on the basis of the Key model.

```php
class KeyController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index() {}

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create() {}

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request) {}

    /**
     * Display the specified resource.
     *
     * @param  \App\Models\Key  $key
     * @return \Illuminate\Http\Response
     */
    public function show(Key $key) {}

    /**
     * Show the form for editing the specified resource.
     *
     * @param  \App\Models\Key  $key
     * @return \Illuminate\Http\Response
     */
    public function edit(Key $key) {}

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \App\Models\Key  $key
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, Key $key) {}

    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\Models\Key  $key
     * @return \Illuminate\Http\Response
     */
    public function destroy(Key $key) {}
}
```

Laravel provides a controller with standard functions which can be filled in.

- `index()` is intended to display all objects of the model (e.g., all Keys)
- `create()` is intended to show the view in which the application user can create a new object (e.g., a form to create a new Key)
- `store(Request $request)` is intended to create the new object created in the application (e.g., create a new Key in the database)
- `show(Key $key)` is intended to show one specific object of the model (e.g., Key #1)
- `edit(Key $key)` is intended to show the view in which the application can alter an object (e.g., update the Key's name)
- `update(Request $request, Key $key)` is intended to modify the specified object in the database (e.g., update Key #1's name in the database)
- `destroy(Key $key)` is intended to remove the specified object from the database (e.g., remove Key #1 from the database)

5.2. Traits

The built application contains many controller classes. Some of these controllers have to make use of the same method. For instance, both the string translations application and the initialisation feature need to be able to commit changes to bunq's app repositories. Therefore, there needs to be a way to call the same method from multiple different controller classes. One possible way of doing this is by using inheritance. There could be a parent controller class containing all methods that need to be used by more than one controller. However, since there are many of such methods the parent controller would become very large and complicated. This could be solved by having two or more parent controller classes, each with their own functionality. The problem with that is that multiple-inheritance is not possible in PHP. Therefore, a controller that needs two functionalities cannot extend both parent classes. To solve this issue the team decided to make use of traits. Traits have been introduced by PHP to solve the problem of multi-inheritance in PHP. A trait is simply a set of methods that can be used by any controller class. It enables code reuse and thereby decreases code duplication. The `CommitTrait` contains all methods that are needed to interact with the GitLab API. The `InitTrait` contains methods needed for the initialisation of the database, which apply to both the initialisation of the keys and the initialisation of the translations. The `MainTrait` contains other functions that are needed in both the string translations application and the initialisation feature.

5.3. Mapping of the existing keys

Before the string translations application can be used on the existing strings of the bunq app, the strings of the iOS and Android app have to be mapped. Each string in the iOS and Android app is identified by its unique key. These keys have to be mapped to the key used in the other app. Furthermore, only one string has to be saved and used in both apps to ensure consistency. After all the strings are mapped, the database is initialised with the English strings. The next phase is to fill the database with all the other available languages.

5.3.1. Levenshtein

In order to know which string in bunq's iOS app belongs to which string in the Android app, the iOS and Android keys need to be mapped together. The mapping view shows an unmapped iOS key together with a list of all possible Android keys, from which the user can select the correct mapping. To make the mapping process less tedious, the Android keys should be sorted on their similarity to the unmapped iOS key. That is, the Android string most similar to the iOS string should appear at the top of the page. PHP knows multiple functions that are used to calculate the similarity (distance) between two strings, from which `similar_text()`[2] and `levenshtein()`[3] are the most well-known [18]. `similar_text()` returns the number of matching characters of the two strings, whereas `levenshtein()` returns the Levenshtein distance between two strings. One important difference between the two functions is the complexity; `similar_text()` and `levenshtein()` have a complexity of $O(\max(n,m)^3)$ and $O(n \cdot m)$ respectively. However, the function `similar_text()` returns a more accurate result. Since speed is an important aspect of the initialisation feature and the accuracy of `levenshtein()` seems to be good enough after further inspection, the team decided to choose the `levenshtein()` function. In both the automated tests and manual tests the correct Android key is always displayed at the top of the page using `levenshtein()`.

One issue with the chosen function is that it only works for strings containing less than 255 characters. Therefore, the application only takes the first 254 characters into account when computing the distance, as shown below in the `distanceBetweenStrings()` function.

```php
public function distanceBetweenStrings(string $first_text, string $second_text)
{
    if (strlen($first_text) > 255) {
        $first_text = substr($first_text, 0, 254);
    }

    if (strlen($second_text) < 255) {
        $dist = levenshtein($first_text, $second_text);
    } else {
        $dist = levenshtein($first_text, substr($second_text, 0, 254));
    }
    return $dist;
}
```

[2]http://php.net/manual/en/function.similar-text.php
[3]http://php.net/manual/en/function.levenshtein.php

5.4. GitLab API

An important aspect of the application is the connection to the repositories of bunq's mobile apps. Currently all strings are saved in these repositories as `.xml` files and `.string` files for the Android and iOS app respectively. All strings created in our application must naturally be committed to these repositories after completion. During the initialisation phase, the files must also be retrieved to obtain the most recent version of all string files. Furthermore, the decisions during the initialisation phase, described in Section 5.3, must also be committed to the repositories to create consistency.

In order for the application to obtain the most recent version of the files, the application makes use of the GitLab API[4] and GuzzleHTTP[5]. GuzzelHTTP is a HTTP client for PHP which can be used to easily send requests and integrate with web services such as the GitLab API. GuzzleHTTP allows us to create `GET`, `POST`, `UPDATE` and `PUT` requests within our code. By simply creating a GuzzleHTTP request to the GitLab API, the application is able to either retrieve files or update the files in a specified repository. As seen below, a client is created per repository in GitLab with the correct project id, thereafter a request is created with this client. Below is the code used to obtain all branches from a repository with a `GET` request:

```php
    public function getClient(bool $android): Client
    {
        if ($android) {
            $client = new Client(['base_uri' => 'https://gitlab.bunq.net/api/v4/projects/[android-project-id]/']);
        } else {
            $client = new Client(['base_uri' => 'https://gitlab.bunq.net/api/v4/projects/[ios-project-id]/']);
        }

        return $client;
    }

    public function getAllBranches(bool $android): array
    {
        $client = $this->getClient($android);
        $response =
            $client->get(
                'repository/branches?per_page=100',
                ['headers' => ['Private-Token' => '[token]']]
            );
        $branches = array_column(json_decode($response->getBody()->getContents(), true), 'name');

        return $branches;
    }
```

In order to obtain access to the repositories, an SSH key is needed for the created GuzzelHTTP requests. The private-token can then be created to give GuzzelHTTP the access. The application has its own account within bunq's GitLab environment. This way the bunq employees are able to see which commits were made by the application.

The code snippet below is the response of the example request above. The response is returned by GuzzleHTTP in the structure specified by the GitLab API and contains the branches' information. From this response, the system filters out the name to display in the application.

```
[
  {"name":"develop", ...}
  {"name":"feature/CIT/bugs#1995_user_login_picker", ...},
  {"name":"feature/CIT/bugs#4830_vector_drawables_on_old_devices",  ...},
  {"name":"feature/CIT/bugs#5324_strip_whitespaces_from_document_number", ...}
]
```

With this, the application communicates with repositories to commit the strings, retrieve the string files and retrieve the repositories' branches.

[4]https://docs.gitlab.com/ee/api/
[5]http://docs.guzzlephp.org/en/stable/

5.5. Security

During the implementation of the application, security needs to be taken into consideration. Most of the security issues were solved by using Laravel as framework. More about how Laravel resolves several issues and what problems needed their own approach can be read in this section.

5.5.1. SQL injection

Laravel already protects against SQL injections. Laravel's Eloquent ORM uses PHP Data Objects (PDO) parameter binding to avoid SQL injections. The Eloquent[6] ORM provides a simple ActiveRecord implementation using Models to interact with its corresponding table in the database. The PDO parameter binding ensures a safe input for the database. The input will be quoted and as a result the query will work as expected. For example, a user would like to retrieve the information of a specific user by inserting the mail `'user@mail.com'` in the query `SELECT * FROM users WHERE email='user@mail.com'`. This query will produce a result as expected. However, if the user inserts `'user@mail.com'; drop table users;` into the same query, the query will delete the user table with all its information. Using PDO parameter binding prevents such actions. The input given by the user will be quoted as mentioned before. Thus, the query will look as follows: `SELECT * FROM users WHERE email='user@mail.com; drop table users;'`. This will do no harm to the database.

5.5.2. Input validation

Input validation ensures that the proper input enters the system. This is especially important when the user enters data into the database. There are two strategies when it comes to input validation, namely on syntactical and semantic level [7]. Syntactical validation checks if the input has the correct syntax of the structured fields. On the other hand, semantic validation enforces the correctness of the values in their context. Laravel provides a check for both strategies. For the authentication, the user's email must satisfy the formatting of an email address, such as containing a '@'. Besides Laravel's standard check, the input fields are also checked on whether they contain any curly brackets and, if so, how many. In the application characters between curly brackets are saved as placeholders. Therefore, the number of placeholders of the translations must be equal to the number of placeholders in the English version. Additionally, the translation placeholder names must be equal to the English placeholder names.

5.5.3. Authentication

Several features are not allowed to be visible to all users. Therefore, users are assigned different roles by the admin user. Each role has its own privileges within the application. The admin is privileged to use all features. Next, the developer has almost all the same rights as the admin, except the ability to change the roles of users. Finally, the translator is only allowed to translate strings and review them. To implement this authentication feature, Laravel provides a standard authentication configuration. Users are stored in the database with their `id`, `name`, `email`, `password` and their `remember_token`. This application assigns a role to users, therefore, a `role` is added to the User attributes. Passwords are stored after being hashed. Laravel does this using Bcrypt, which is a password hashing function based on the Blowfish cipher. The Bcrypt function protects against rainbow table attacks and brute force attacks. The hashed string of the password can be divided into several parts. Firstly, the prefix '\$2y\$' indicates that the hashed string is a Bcrypt hash in modular crypt format. The modular crypt format is a standard for encoding password hash strings [2]. Each hashed string has the following prefix: `${identifier}${content}`. As mentioned before, the prefix used in the hash is '\$2y\$' indicating the usage of the Bcrypt scheme and it has crypt_blowfish emit. The prefixes '\$2\$', '\$2a\$' and '\$2x\$' also indicate the usage of Bcrypt. However, these versions are older and contain some insecurities which are fixed in version '\$2y\$'. Furthermore, the prefix can also refer to a SHA* hashing method, '\$5\$', '\$6\$' and \$sha1\$'. These hashes are less secure than using Bcrypt, because SHA* are fast hashes, whereas Bcrypt is a slow hash. With a fast hash, an attacker can compute billions of hashes per second in an offline attack [26]. The content '10' implies the cost, meaning there are 2^{10} iterations, key expansion rounds. The prefix is followed by a 128-bit salt of 22 characters and 184 bits of the resulting hash. As mentioned before, passwords are being hashed and not encrypted when stored into the database. This is preferable, because hashing is a one-way function, whereas encryption is a two-way function. In other words, if the decryption became public, passwords could easily be changed back to their plain text version. Therefore, encryption should only be used over hashing if it is necessary to decrypt the message [20].

[6]https://laravel.com/docs/5.6/eloquent

Using Laravel's Authentication library has many advantages. As mentioned before it offers Bcrypt hashing. Other features are checking active users, protecting against Cross-Site Request Forgery, password resets and encryption [5].

5.5.4. Cross-site request forgery

To prevent cross-site request forgery the defence needs to check the standard headers to verify that the request is from the same origin and to check the CRSF token. Laravel already checks both of them. To begin with, Laravel automatically generates a CSRF token for each active user session. This token is used to verify that the authenticated user is the one making the request to the application and not someone else. This token matches the token from the header. Next, a cookie is added to each request with that token. This check assures that the request is send from the same origin. Therefore, cross-site request forgery, CSRF, tokens are used to ensure that third parties cannot initiate requests they are not authenticated to.

5.5.5. Application protocol

At the moment the application uses the bunq server and is only accessible when the user is connected to bunq's VPN. Therefore, HTTP is sufficient for the interaction between server and client side. However, if the application should go live and, therefore, be used by people other than the employees of bunq, the connection should be more secure. It is then recommended to use HTTPS instead of HTTP. HTTPS uses either SSL or TLS 1.2 as its secure protocol to encrypt communications. Both protocols make use of an asymmetric public key infrastructure.

5.6. Problems

There were several issues during this project. These problems and their solutions are mentioned below.

5.6.1. Session

At the beginning stage of the initialisation feature, it took 8.5 seconds to load each iOS item with its corresponding sorted Android items. This was mainly caused by two nested for loops; one to get all the unmapped keys and one to get all the items of these keys. This problem has been resolved by changing these for loops to an efficient query. This reduced the loading time to 2 seconds. However, the strings were still extracted from the app repositories after every mapping, which is not very efficient. Therefore, the system was changed to store all Android and iOS instances that are not yet mapped in a session instead.

5.6.2. GitLab connection

While implementing the feature where a user can commit a specific string or a collection of strings, the team first opted for a different approach than described in Section 5.4.

The team's first approach: locally clone, copy the files from that directory into the application's directory, alter the strings and thereafter commit the altered version in the cloned directory. By cloning the repositories locally, the application must be able to switch branches and pull the repository files from the remote. Another disadvantage is that a copy of the bunq repositories has to be saved.

After considering the difficulty and redundant copies of the repositories, the team researched another approach and found the GuzzleHTTP solution. The files retrieved with this solution will always be the most recent version on the specified branch and only these string files will be saved temporarily in the application.

6
Testing and code quality

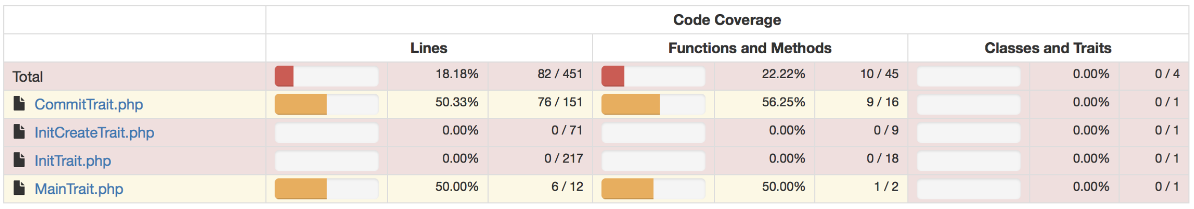
Testing is an important aspect in the creation of any web application. Since the string translation web application will be used in practice, it is essential that all aspects of the application are well-tested and (almost) bug-free. This is important for the application considering that it is able to commit straight into bunq's app repositories. Automated tests should assure that the application functions as expected, both now and in the future. Another important aspect is code quality, so that the code is easy to read and understand. The following sections elaborate upon the different types of tests performed on the system and the quality and structure of the code.

6.1. Testing

The code of the application is tested using the PHP testing framework Codeception[1]. The three subsections below discuss the different types of testing used to test the application, namely unit, functional, acceptance and usability testing. Functional testing is used to *verify* that the application works as expected. Acceptance testing is used to *validate* that the right application is built to meet the customers requirements. Usability testing is used to determine the extent to which the software product is understood, easy to learn, easy to operate and attractive to the users. Finally, unit testing is used to test the individual components of the system. Unit testing is also a type of functional testing. However, since the tests differ a lot from the other functional tests, the unit tests are discussed in a separate subsection.

6.1.1. Unit testing

Unit tests were created to test the small units of code within the various controllers and traits. Almost all methods in the controllers use some sort of connection with either the database, views or bunq's repositories. Therefore, most of the functions within the controller classes are not easy to test with unit testing. More emphasis was put on the other testing types. Nevertheless, unit testing was applied where possible. Figure 6.1 shows the coverage report for the unit tests.



	Code Coverage					
	Lines		Functions and Methods		Classes and Traits	
Total	18.18%	82 / 451	22.22%	10 / 45	0.00%	0 / 4
CommitTrait.php	50.33%	76 / 151	56.25%	9 / 16	0.00%	0 / 1
InitCreateTrait.php	0.00%	0 / 71	0.00%	0 / 9	0.00%	0 / 1
InitTrait.php	0.00%	0 / 217	0.00%	0 / 18	0.00%	0 / 1
MainTrait.php	50.00%	6 / 12	50.00%	1 / 2	0.00%	0 / 1

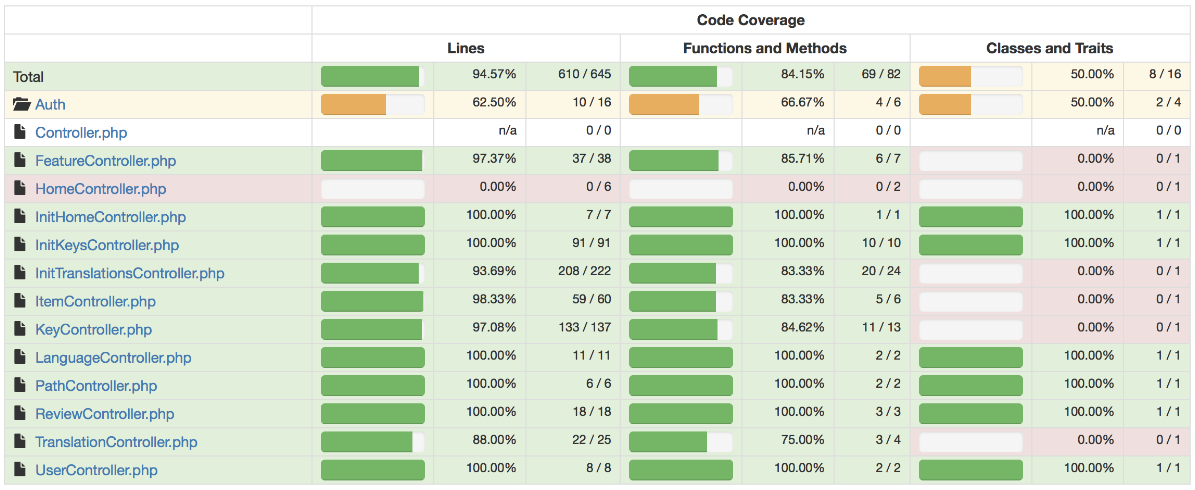
Figure 6.1: Coverage: unit tests

6.1.2. Functional testing

The functional tests created using Codeception assert whether the right actions have been taken after a page has been visited or a button has been clicked. Additionally, the content of the database is examined on its

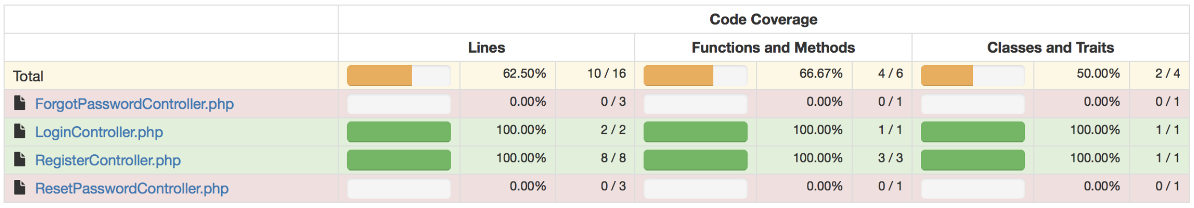
[1] `https://codeception.com`

correctness after each test. Most functional tests are not required to be run in a browser, except for tests using JavaScript. The team decided to use the Selenium web server[2] and ChromeDriver[3] for these functional tests. The other functional tests are not run in a browser because the browser tests run slower. The coverage report of the functional tests can be found in Figures 6.2, 6.3 and 6.4.



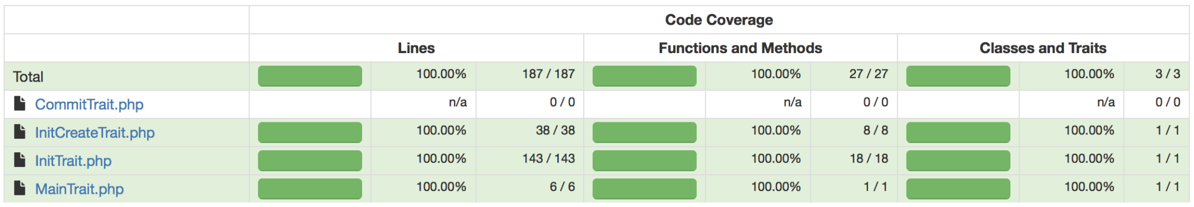
	Code Coverage					
	Lines		Functions and Methods		Classes and Traits	
Total	94.57%	610 / 645	84.15%	69 / 82	50.00%	8 / 16
Auth	62.50%	10 / 16	66.67%	4 / 6	50.00%	2 / 4
Controller.php	n/a	0 / 0	n/a	0 / 0	n/a	0 / 0
FeatureController.php	97.37%	37 / 38	85.71%	6 / 7	0.00%	0 / 1
HomeController.php	0.00%	0 / 6	0.00%	0 / 2	0.00%	0 / 1
InitHomeController.php	100.00%	7 / 7	100.00%	1 / 1	100.00%	1 / 1
InitKeysController.php	100.00%	91 / 91	100.00%	10 / 10	100.00%	1 / 1
InitTranslationsController.php	93.69%	208 / 222	83.33%	20 / 24	0.00%	0 / 1
ItemController.php	98.33%	59 / 60	83.33%	5 / 6	0.00%	0 / 1
KeyController.php	97.08%	133 / 137	84.62%	11 / 13	0.00%	0 / 1
LanguageController.php	100.00%	11 / 11	100.00%	2 / 2	100.00%	1 / 1
PathController.php	100.00%	6 / 6	100.00%	2 / 2	100.00%	1 / 1
ReviewController.php	100.00%	18 / 18	100.00%	3 / 3	100.00%	1 / 1
TranslationController.php	88.00%	22 / 25	75.00%	3 / 4	0.00%	0 / 1
UserController.php	100.00%	8 / 8	100.00%	2 / 2	100.00%	1 / 1

Figure 6.2: Coverage: functional tests of the controllers



	Code Coverage					
	Lines		Functions and Methods		Classes and Traits	
Total	62.50%	10 / 16	66.67%	4 / 6	50.00%	2 / 4
ForgotPasswordController.php	0.00%	0 / 3	0.00%	0 / 1	0.00%	0 / 1
LoginController.php	100.00%	2 / 2	100.00%	1 / 1	100.00%	1 / 1
RegisterController.php	100.00%	8 / 8	100.00%	3 / 3	100.00%	1 / 1
ResetPasswordController.php	0.00%	0 / 3	0.00%	0 / 1	0.00%	0 / 1

Figure 6.3: Coverage: functional tests of the login controllers



	Code Coverage					
	Lines		Functions and Methods		Classes and Traits	
Total	100.00%	187 / 187	100.00%	27 / 27	100.00%	3 / 3
CommitTrait.php	n/a	0 / 0	n/a	0 / 0	n/a	0 / 0
InitCreateTrait.php	100.00%	38 / 38	100.00%	8 / 8	100.00%	1 / 1
InitTrait.php	100.00%	143 / 143	100.00%	18 / 18	100.00%	1 / 1
MainTrait.php	100.00%	6 / 6	100.00%	1 / 1	100.00%	1 / 1

Figure 6.4: Coverage: functional tests of the traits

6.1.3. Acceptance & usability testing

In order to further test the application, acceptance and usability tests were used. As described above, acceptance tests are used to validate that the application meets the customer's requirements. Usability tests, on the other hand, are used to determine the user friendliness of the application. In order to perform these two testing types, a prototype of the application was deployed on the bunq server. This made it possible for bunq employees to examine the application and give their feedback. The team invited five translators, one copy writer and three developers to try out the application. In addition, they were asked to fill out a survey after using the application for the first time. The survey contains twelve questions regarding the expectations, clarity and ease of use of the application. The questions regarding acceptance and usability testing are denoted by the 📋 and 👆 icons respectively.

[2]https://www.seleniumhq.org/projects/webdriver/
[3]http://chromedriver.chromium.org

1. What is your role in the translation process? (Android developer/iOS developer/translator/copy writer)
2. Which browser did you use for the application? (Safari/Chrome/Firefox/other) 📋
3. How clear was the goal of the application? (0-10) 👆
4. How clear was the application's interface? (0-10) 👆
5. Help page 👆
   (a) Did you need the help page? (yes/no)
   (b) If you needed the help page for using the application, please inform us why.
6. Rate the ease of use of the application's features (1-5 + N/A) 👆
   - Adding a new string
   - Adding a new path/language/feature
   - Reviewing a string/translation
   - Translating a string
   - Mapping the existing Android and iOS keys
7. Name at least 1 negative aspect of the application. 📋 👆
8. Name at least 1 positive aspect of the application. 📋 👆
9. Did you have any expectations for the application prior to using it? If so, which ones have not been met? 📋
10. Would you like to use the application in the future? (1-5) 📋 👆
11. Do you have any further comments? 📋 👆

The four obtained responses of the survey can be found in Appendix A. Some of the invited employees gave their feedback in person instead of filling in the questionnaire. These comments were also taken into account. The enumeration below discusses the results of the questionnaire:

1. The obtained feedback is given by three translators and one Android developer.
2. The type of browser is part of the non-functional requirements, which makes it an acceptance test. Three types of browsers were used. The browsers mentioned in the non-functional requirements are all compatible with the application.
3. The clarity of the goal of the application belongs to the usability category, since the question tests whether the goal of the application is understood. The results vary from grade 6-10. The team improved this by updating the homepage to show the application's purpose.
4. The clarity of the application's interface clearly belongs to the usability category. The question scored an average of 6. Therefore, the team decided to improve the application according to the given feedback in the following questions.
5. The help page also belongs to usability testing. When users need the help page to navigate through the application, this indicates that the user interface is not clear enough. As a result, the team improved the GUI according to the given feedback.
6. As can be seen in the results, adding a new string, path/language/feature and mapping the existing Android and iOS keys are considered the most difficult. Adding a new string is considered to be easy by the developer, which matters the most. It is understandable that the translators had difficulties with this, as they have no experience with the terms used. The same applies to the mapping feature, which is confusing for translators for obvious reasons. As adjustment, the terms are made more clear in the interface.

7. The positive and negative aspects of the application belong to both testing categories. It is acceptance testing because the aspects could concern missing requirements. Moreover, it belongs to usability because it could contain remarks concerning the GUI or ease of use of the application. The team revised the application to take the negative aspects into account. This is mainly done by improving the overall flow of the application and moving some important content from the help page to the main application. The third remark will be left for future work, as this would take too much time to implement.

8. The positive feedback was well received.

9. The prior expectations belong to the acceptance category, as the expectations of the user are part of the client's requirements. All the mentioned expectations are present in the application. However, some of them could be more clear.

10. 3 out of 4 users would want to use the application in the future. The team expects the rating to be higher, now the feedback has been implemented.

11. Nothing important was mentioned in this question.

Besides the questionnaire, the acceptance tests are also performed in another way. Since the beginning of the project, the team's client advisor periodically met with the team to review the product. During these meetings, he gave his feedback on some of the implemented functionalities. As a result, the requirements that were initially set were adjusted to meet the client's requirements. For example, the team implemented a feature to add strings to groups and commit them together, instead of committing them individually.

6.2. Code quality

Code quality is an important aspect of maintainable code. Since the application will be used by bunq, even after the internship has ended, it is important that the code is easy to understand. To ensure that the code had a high degree of quality, several measures were taken. First, it was agreed upon that all code has to be peer reviewed by at least two other team members before it can be merged. This measure already helped to get rid of a lot of unneeded code. Next, the code was regularly examined by a code style check tool. PHP Code Sniffer[4] was chosen as code style check tool as it is easy to integrate with PHPStorm. Furthermore, PHP Code Sniffer is also the tool used by bunq, which means it has a rule set that automatically checks if the code complies with bunq's coding style. Finally, the codebase was sent to an independent code evaluation company, the Software Improvement Group (SIG), to assess the code quality. This evaluation was done two times during the project. After the first assessment the code was adjusted according to the feedback. The SIG reviews can be found in Appendix D and are discussed in Chapter 9.

[4]`http://pear.php.net/package/PHP_CodeSniffer`

7 Final product evaluation

During the course of the project a product has been created in accordance to design goals and requirements. This chapter describes the evaluation of this final product compared to the design goals and requirements set.

7.1. Requirements analysis

During the course of the project, it became clear that the original proposed functional requirements were incomplete. The client asked for more features, which originally had a lower priority in the MoSCoW model or were not listed as a requirement. For instance, the mapping of the iOS and Android keys is listed as one requirement. However, this requirement was stated too general, since it represents the whole initialisation of the database feature. Therefore, the project requirements are divided into two parts: the initialisation of the database and the string translations application. Taking this into account, a new MoSCoW model has been created.

7.1.1. Final product requirements

The following subsection describe the final requirements and whether these have been implemented in the final product. These requirements are updated in accordance to the wishes of the client. During the course of this project, certain features of the application have shifted in priority and thus were updated. The second subsection gives an overview of an analysis between the first version of the requirements and the final version.

✓ = Implemented in the final product
✗ = Not implemented in the final product

7.1.1.1. String translations application

Must have(s)

- ✓ The user must be able to add a new string
- ✓ The user must be able to add a new language
- ✓ The user must be able to add a new file path (to store where which string belongs in the repository)
- ✓ The user must be able to add a new feature (to group multiple keys together)
- ✓ Each string must be mapped to the correct iOS and Android keys
- ✓ Items which are not used/applicable in one of the apps must have a null value in that field to show that it is not available for that system
- ✓ Each string must be connected to the correct file path
- ✓ The user must be able to add new strings to a feature

- ✓ The user must be able to review an item by approving them or save another version instead
- ✓ When the user has reviewed an item, that item must be set to reviewed and not show on the review screen anymore
- ✓ The user must be able to translate strings
- ✓ When the user has translated an item in language A, a new item must be created for language A and not show on the translation screens of language A anymore
- ✓ The user must be able to commit a feature (with all its keys and strings) to the app repositories
- ✓ The user must be able to add placeholders to a new string
- ✓ The application must have a complete overview of all key-pairs
- ✓ The application must have a complete overview of all features
- ✓ The application must have a complete overview of all strings of one key-pair
- ✓ The application must have a complete overview of all strings that need to be translated (per language)
- ✓ The application must have a complete overview of all strings that need to be reviewed (per language)

Should have(s)

- ✗ The application should have a complete overview of all the strings in both apps (iOS and Android)
- ✓ The application should have an user log in system
- ✓ The system should have a privilege system which determines what the users are allowed to do in the application
- ✓ The roles of the privilege system should be: developer, translator
- ✓ The user should be able to have multiple roles
- ✗ "Developers" should be warned when they try to commit a feature which is not complete yet
- ✓ "Developers" should be allowed to add new strings
- ✗ "Developers" should be allowed to edit the keys of the strings
- ✓ "Translators" are allowed to translate strings and review the translations
- ✓ "Developers" and "translators" should be able to edit an old string
- ✓ When an old string is edited, the item should be set to uncommitted
- ✓ "Developers" should be able to commit the strings that do not have a feature
- ✓ "Developers" should be able to update an old string and commit the change

Could have(s)

- ✗ The roles of the privilege system could be extended by: admin, copy writer, translation reviewer, crowd translator, crowd approver
- ✗ Only "copy writers" could be able to review the English strings created by the "developers"
- ✓ Only "admins" could be able to assign roles to users
- ✗ Only "admins" could be able to add new languages to the overview
- ✗ "Admins" could be able to delete an item
- ✗ A crowd translator could be able to add new translations to items

- ✗ A "translation reviewer" could be able to approve and decline a crowdsourced translation
- ✗ A "translation reviewer's" approval is the final approval
- ✗ A "translation reviewer" could be able to edit a crowdsourced translation
- ✗ The system could save the history of the items and its translations
- ✗ "Crowd approvers" could approve a translation
- ✗ "Crowd approver's" approvals could not be the final approval
- ✗ A translation could have two "crowd translator" approvals before the "translation reviewer" can approve it

Would have(s)

- ✗ Priority would be assigned to items
- ✗ The overview would contain the strings and their translations from the backend (server responses)
- ✗ The backend items could be mapped to their keys from the code
- ✗ The users would place comments when reviewing a translation

7.1.1.2. Initialisation of the database of the application

Must have(s)

- ✓ The mapping view must show an overview of to be mapped iOS key and all the available Android keys
- ✓ The user's input must not contain any curly brackets if they enter a string in the mapping view without a placeholder
- ✓ The user's input must contain a placeholder name surrounded by curly brackets if they enter a string in the mapping view with a placeholder
- ✓ The user's input must not contain any curly brackets if they enter a string in the initialisation translations view without a placeholder
- ✓ The user's input must contain curly brackets if they enter a string in the initialisation translations view with a placeholder
- ✓ The user's input must contain the same English placeholders as the English string in the initialisation translations view
- ✓ The mapping view must show the Android keys ordered by distance between the iOS string and Android strings
- ✓ The user must be able to store the shown iOS key as an exclusive iOS key (not present in Android app)
- ✓ The user must be able to skip the shown iOS key
- ✓ The user must be able to map the skipped iOS keys later on
- ✓ The user must be able to choose which English string to use when mapping iOS and Android key if their strings differ
- ✓ The user must be able to enter their own English string when mapping the iOS and Android key if their strings differ
- ✓ The user must be able to enter their own placeholder name(s) if the string contains any
- ✓ The system must be able to store all the remaining Android keys as exclusive Android keys (not present in iOS app)
- ✓ The user must be able to choose which translation string to use after mapping the iOS and Android keys

- ✓ The user must be able to enter their own translation after the mapping of the iOS and Android keys if their translation strings differ
- ✓ The user must be able to close the application and after restarting continue where they left off

Should have(s)

- ✓ The application should be able to commit the chosen strings to the bunq apps
- ✓ The user should be able to choose which language they want to initialise

Could have(s)

- ✓ When the system has a user privilege system only "developers" are allowed to map the iOS and Android keys
- ✓ When the system has a user privilege system only "translators" are allowed to choose which translation strings to keep

Would have(s)

- ✗ The application would be able to send the user a notification of when a new translation decision can be made when a new key has been mapped
- ✗ The application would be able to revert a mistake made with mapping the keys

In Section 2.2.2 the team's goals were set on implementing 100% of the must haves and 75% of the should haves. As seen in the overview above, the team has indeed implemented 100% of the must haves and 80% of the should haves.

7.1.2. Requirements comparison

Original version	**Adapted version**	**Priority of adapted version**
Must haves		
The product must have a complete overview of all the strings in both apps (iOS and Android)	The application should have a complete overview of all the strings in both apps (iOS and Android)	✗Should
	The application must have a complete overview of all key-pairs	✓Must
	The application must have a complete overview of all features	✓Must
	The application must have a complete overview of all strings of one key-pair	✓Must
	The application must have a complete overview of all strings that need to be translated (per language)	✓Must
	The application must have a complete overview of all strings that need to be reviewed (per language)	✓Must
The items in the overview must store the developer string (English), the reviewed string (English) and the translations for all the languages available in the apps (Dutch, German, Italian, Spanish) in separate fields		

Each item must be mapped to the correct iOS and Android keys	Each item must be mapped to the correct iOS and Android keys	✓Must
The iOS and Android keys of the item must be stored in fields		
Items which are not used/applicable in one of the apps must have a special key in that field to show that it is not available for that system	Items which are not used/applicable in one of the apps must have a null value in that field to show that it is not available for that system	✓Must
Developers must be able to add new items to the system via the GUI of the overview	The user must be able to add a new string	✓Must
	"Developers" should be allowed to add new strings	✓Should
Copy writers must be able to add reviewed strings to items	The user must be able to review an item by approving them or save another version instead	✓Must
	When the user has reviewed an item, that item must be set to reviewed and not show on the review screen anymore	✓Must
	Only "copy writers" could be able to review the English strings created by the "developers"	✕Could
Translators must be able to add translations to items and edit them	The user must be able to translate strings	✓Must
	When the user has translated an item in language A, a new item must be created for language A and not show on the translation screens of language A anymore	✓Must
	"Translators" are allowed to translate strings and review the translations	✓Should
	A crowd translator could be able to add new translations to items	✕Could
When a translator edits an item and commits the changes in the system, the improvements must be committed automatically to the code		
Should haves		
A translation reviewer should be able to approve a crowdsourced translation	A "translation reviewer" could be able to approve and decline a crowdsourced translation	✕Could
A translation reviewer's approval should be a final approval	A "translation reviewer's" approval is the final approval	✕Could
A translation reviewer should be able to edit a crowdsourced translation	A "translation reviewer" could be able to edit a crowdsourced translation	✕Could
The translation should not be available to the developers if it has not been approved yet		

The final approval of an item's translation should lock that translation and commit them automatically to the code		
A crowd translator should be able to add new translations to items		
The system should have a privilege system which determines what the users are allowed to modify in the overview	The system should have a privilege system which determines what the users are allowed to do in the application	✓Should
The roles of the privilege system should be: admin, developer, copy writer, translator, translation reviewer, crowd translator, crowd approver	The roles of the privilege system should be: developer, translator	✓Should
	The roles of the privilege system could be extended by: admin, copy writer, translation reviewer, crowd translator, crowd approver	✗Could
Users should be able to have multiple roles	The user should be able to have multiple roles Only "admins" could be able to assign roles to users	✗Should ✓Should
An admin should be able to add new languages to the overview	The user must be able to add a new language	✓Must
	Only "admins" could be able to add new languages to the overview	✗Could
An admin of the system should be able to unlock an item	"Developers" and "translators" should be able to edit an old string	✓Should
An admin of the system should be able to delete an item	"Admins" could be able to delete an item	✗Could
Could haves		
The overview could contain the strings and their translations from the backend (server responses)	The overview would contain the strings and their translations from the backend (server responses)	✗Would
The backend items could be mapped to their keys from the code	The backend items could be mapped to their keys from the code	✗Could
The system could save the history of the items and its translations	The system could save the history of the items and its translations	✗Could
The users could place comments when reviewing a translation	The users would place comments when reviewing a translation	✗Would
Crowd approvers could approve a translation	"Crowd approvers" could approve a translation	✗Could
Crowd approver's approvals could not be the final approval	"Crowd approver's" approvals could not be the final approval	✗Could
A translation could have 2 crowd translator approvals before the translation reviewer can approve it	A translation could have two "crowd translator" approvals before the "translation reviewer" can approve it	✗Could
Would haves		
Priority would be assigned to items	Priority would be assigned to items	✗Would

Items belonging to one feature would be grouped together	The user must be able to add a new feature (to group multiple keys together)	✓Must
	The user must be able to add new strings to a feature	✓Must
	The user must be able to commit a feature (with all its keys and strings) to the app repositories	✓Must

Table 7.1: Comparison between the first version of the requirements and the final version of the requirements

The items below only occur in the new version of the requirements. At the beginning, paths were not considered. The path indicates where the string is saved in bunq's repository. During the development of this project it became more clear how necessary it was. Another feature which was left out in the original version, were placeholders. While examining the strings in the code of the bunq apps, placeholders seem to occur in a few strings. Therefore, the system needs to handle placeholders to prevent system failures. Different string format specifiers are used in the iOS and Android app to indicate where a placeholder is placed in the string. Furthermore, originally there was no log in system. This feature is necessary to indicate which role(s) a user has. Finally, the committing feature was not well defined. At the moment before committing, a feature has to be assigned to the to be committed strings. Meaning, one commit contains one feature which consists of multiple strings. The original version also contained a commit feature. However, that version committed each string individually, which is redundant and inefficient. After discussing with the client, the team has opted for a commit function where a complete feature can be committed at once.

- ✓ The user must be able to add a new file path (to store where which string belongs in the repository)
- ✓ Each string must be connected to the correct file path
- ✓ The user must be able to add placeholders to a new string
- ✓ The application should have an user log in system
- ✕ "Developers" should be warned when they try to commit a feature which is not complete yet
- ✕ "Developers" should be allowed to edit the keys of the strings
- ✓ When an old string is edited, the item should be set to uncommitted
- ✓ "Developers" should be able to commit the strings that do not have a feature
- ✓ "Developers" should be able to update an old string and commit the change

As seen in the overview and comparison above, many requirements of the first version are now matched to multiple requirements of the final version. Due to this, the calculation of the implementation percentage achieved is not calculated easily. However, since many requirements have been added and requirements which were first classified as a 'would have' are now implemented as 'must have', it can safely be said that the goals stated in Section 2.2.2, have been achieved.

7.2. Design goals analysis

As described in Section 2.2.1, the product is created with design goals in mind. The following sections discuss per goal how the team has attempted to implement the product as best as possible to achieve this goal.

Scalability

Since the product should be scalable, the team has opted to create a database structure which makes the product scalable for the many languages to be added by bunq. The database structure described in Section 4.3 is made up of multiple tables. An important table is the Item table which stores all strings in the available languages. Since all items are separate objects, it is relatively easier to extended the existing languages with a new one, compared to creating one object per string and store each language in fields of that object. When adding a new language, the system only has to create new Items in that new language. This means that the database is vertically scalable instead of horizontally scalable. The team has also opted for this structure since the database is a SQL database which, as stated in Chapter 3, supports vertical scalability.

Usability
To keep usability in mind, the team has asked multiple developers, copy writers and translators of bunq to test the deployed application. The people who tested the application have filled in a questionnaire created by the team and from this the product's interface was altered to improve the clarity and usability.

Compatibility
Since the system should be compatible with at least Safari 11.1 and Chrome 60.0, the team has tested the application in these browsers. The functional tests are run using Chrome and during development the team has used Safari to validate manually whether the created functionality was correct.

When bunq plans to deploy the crowdsourcing feature, they could tests its compatibility with other popular web browsers such as Firefox.

Performance
Regarding the performance of the system, the team has improved the system's initialisation feature over time since it's functionalities required the most computational power. The improvements are described in Section 5.6.1. The performance of the rest of the application overall did not need improvements since the actions and feedback are processed almost instantaneously.

Security
During this project, an important aspect of the application is its security. Laravel provides security features which helps the teams to keep the application secure. These features are described in Section 5.5. Since the final product is not intended to be deployed publicly yet, the team has depended solely on these features and have only extended them by certain input validation. An example is the placeholder check function which checks whether the given placeholders match the placeholders in the database. This ensures that the user can never enter a mismatched placeholder. The reason for not focussing further on the application's security is because the client has made it clear to assume that the bunq employees will not try to jeopardise the application. The team thus focused on delivering a more complete application versus a more secure one. When bunq deploys the crowdsourcing feature, this is an aspect which they must consider to improve still.

Maintainability
To keep the code maintainable after delivering the product to bunq, the team decided to document clearly what each class' and function's purpose is. This was done by means of PHPDoc. Next to this, the team handed in the codebase to SIG for an analysis of the code's quality. By means of this analysis the team has improved the application's code quality.

8 Process

The process of the project is described in this chapter. The agile methodology used is described first. Secondly, the development resources used in this project are mentioned in the next section.

8.1. Agile Methodology

This project practises an agile development methodology. The agile methodology used is SCRUM [1]. From previous projects, it was clear that using SCRUM is beneficial. The waterfall model, for example, would not suffice. It is not adaptable to change, while this criteria is important because in the course of this project the requirements have been changed. The small cycles of SCRUM, called sprints, enable fast feedback and an adaptation to change. Each sprint has a duration of a week.

At the end of the project the requirements differed from the original requirements created at the beginning of the project, see Section 7.1.2. Due to the realisation that the original requirements were incomplete, new requirements have been added and the original requirements have been adapted to create a new complete version of the requirements, which can be found in Section 7.1.1.

At the beginning of each week, a new sprint starts, thus, the features that have to be implemented that week, are discussed and a SCRUM board is created. This gives the flexibility to deviate from the original requirements.

At the end of the week all features with a deadline within that week have to be done. During the retrospective any issues of that sprint are discussed and fixed in the next sprint. More information about the usage of SCRUM by the team can be found below in Section 8.1.1

Using SCRUM also means that different roles have been assigned to all people involved. In this project there is no official product owner who is a part of the Scrum team. The role will be fulfilled as follows: the supervisor, Wessel Van, communicates the client's wishes to the team. The team then draws up a document on how/what to create the prioritised features. Finally, Wessel approves these documents or asks for changes. As for the team members, responsibilities have been assigned.

During the project, the member should supervise that their responsibility has successfully been fulfilled. In Table 8.1 can be found to whom the responsibilities has been assigned.

Responsibility	**Member**
Scrum master	Maria
Main contact	Maria
Lead programmer	Emma
Lead testing	Zoë
Responsible for documentation	Beyza
Design lead	Beyza

Table 8.1: The responsibilities of each team member

[1] https://www.scrumalliance.org

The standard meetings, for the backlog and retrospective, take place on a weekly basis. The team also discussed on a daily basis what each team member has done that day and what they would do the following day, the daily SCRUM. Additionally, there were several meetings with the coach and the client. These meetings can be found in Table 8.2. When there is an issue, this is immediately asked by either mail or face to face. Therefore, there was no need to wait for the next meeting for questions to be answered. This helped increase the work flow.

Type	**Description**	**Frequency**	**Format**	**Participants**
Backlog	Plan next sprint	Every Friday the week before	In person	Group
Retrospective	Look back on previous sprint	Every Friday	In person	Group
Daily SCRUM	Update team on today's and yesterday's tasks	Every day	In person	Group
Coach	Update coach on progress	Every week (on Monday)	Skype call	Group and coach
Coach	Update coach on progress	Every 2 weeks	In person	Group and coach
Client	Update supervisor on progress	Every week	Skype or in person	Group and supervisor
Progress	Status update for everyone	Every 3 weeks	Skype and in person	Group, coach and supervisor
Midterm	Half way status update	Once	In person	Group, coach and supervisor

Table 8.2: Meetings

8.1.1. Reflection

Looking back to the work process, using SCRUM had many advantages. The team could easily adapt the requirements to the wishes of the client. Furthermore, the weekly meetings created a fine feedback loop. Between team members feedback was given during the retrospectives. The team reflected on the progression of that week to improve the next week. For example, some features were underestimated at the beginning, like mapping the iOS and Android keys. During the first retrospective these issues with this feature were discussed and adjustments were made to the requirements of this feature. For instance, a MoSCoW was created for the initialisation feature. Other features were overestimated, like the committing feature, giving the team more time to work on other features. All these issues were discussed and a resolved as a group for the following week. Thus, SCRUM gave the team the flexibility to adjust the requirements and to change working methods if seemed necessary. SCRUM has a rapid timeline which enables a quick development. Features were split into smaller requirements, which were set on a deadline. If a deadline was not met, the team looked back on why not and made arrangements to include this feature in the product as soon as possible. The team had already worked together with SCRUM as their agile methodology. Therefore, SCRUM only presented advantages and no disadvantages.

8.2. Development resources

During this project, several development resources were used. These resources are mentioned in this section.

8.2.1. Kantree

To visualise which features need to be completed when and by whom, the team uses Kantree[2] as SCRUM board. Kantree is a work management platform. At bunq they also use Kantree as their work manager. Therefore, the team was recommended to also make use of this platform. It gives a good overview of the stage of each task and when it is due. Tasks can be assigned to different team members. In addition to the sprint stages, tasks are also given labels to group them by theme. The tasks can be labelled with bug, fix, test, code

[2]https://kantree.io/index

quality, enhancement, documentation, deployment, GUI and organisation. Looking back on this work flow, it was managed well.

8.2.2. GitLab

GitLab[3] was chosen for version control of the code. The reason behind choosing this hosting facility, is that bunq uses it. Due to confidentiality, all resources used by us are hosted by their own server. Therefore, this project had to make use of GitLab. GitLab is a Git-repository manager. Git[4] is a distributed version control system which handles products no matter what size with speed and efficiency. It assists the development team to organise the code. Each new feature can be created on a new branch and after approval merged into the product. This also ensures maintainability, because the code written has to be approved by fellow developers.

8.2.3. Work place

From the beginning on, the team was expected to work 5 days a week from 9 to 6 at the bunq office. This had a beneficial impact on the work progress. The team members could easily work together and ask questions if needed. Additionally, requirements could easily be adapted to the feedback given from the supervisor at bunq. Finally, when help was needed from a bunq employee, they could easily and quickly offer their assistance.

[3]https://about.gitlab.com
[4]https://git-scm.com

9 Discussion and recommendations

The application already meets most of the requirements, written in Chapter 7. However, not all requirements were met. This chapter describes which features still need to be implemented and which features could be improved. Furthermore, an ethical discussion is held in this chapter and the action taken by the team to implement the feedback of SIG is described.

9.1. Ethical implications

In the current state of the application, there are no ethical issues to consider since the application is now only used by bunq employees. The translations process is merely moved from their GitLab issues to the created application. However, when bunq decides to deploy the crowdsourcing feature it is important to keep the new data privacy law in mind. The user's information should be kept private and only necessary information should be stored within the application. Furthermore, the user should be made aware for what their translations are going to be used for, since bunq will use them in a commercial setting. This means that technically bunq profits off of the crowdsourced translations. This could perhaps be compensated by giving the crowd translators and crowd reviewers rewards.

9.2. SIG

During the course of the project, the team had to upload the code base to Software Improvement Group (SIG), a company which analyses code projects on code quality. Appendix D contains the feedback given. Two feedbacks were given because a mistake was made and an old upload was analysed.

The second feedback is very similar to the first feedback given by SIG. It appears that the team scored 0.1 lower, even though a lot of time was spent on refactoring the functions in order to lower the Unit Size. In both the first and second feedback round, Unit Size and Unit Interfacing seem to be the main issue. The Unit Size example given in the second feedback, `readAndroidFile()`, was indeed too long. However, shortening functions often increases the parameter count, which lowers the Unit Interfacing score. Therefore, for the final version, the team should aim at lowering the Unit Size, without adding a lot of extra parameters. This can be done by combining some parameters that belong together, as stated in the first feedback. Another point of improvement is the Unit Interfacing of the code. The example `Item.newPivot()` was given, however this function was not created by the team. The function was generated by Laravel. Thus, the feedback for this part of the analysis is not applicable for the team. The hard-coded path was indeed only temporary, used when the GitLab API feature was not yet implemented. The current codebase contains no hardcoded paths.

Even though the team understands the given feedback, not all aspects on which the codebase could be improved were specifically mentioned. The second feedback seems to be more positive than the first, with less examples of bad Unit Size and added tests. For the final feedback round, the team took the given feedback into account. The mentioned methods that were either too long or contained too many parameters were refactored. However, since the feedback was not very extensive, the team decided to do some more investigation on how to further improve the code quality. In a previous project, the team worked with Better Code

Hub[1], an application created by SIG. Better Code Hub takes 10 guidelines into account. The guidelines are listed below with the provided information given on Better Code Hub.

1. **Write Short Units of Code**
    - Small units are easier to understand, reuse, and test.
    - When writing new units, don't let them grow above 15 lines of code.
    - When a unit grows beyond 15 lines of code, you need to shorten it by splitting it in smaller units of no longer than 15 lines of code.
2. **Write Simple Units of Code**
    - Keeping the number of branch points (if, for, while, etc.) low makes units easier to modify and test.
    - Try to limit the McCabe complexity, that is number of branch points plus 1, in a unit to at most 5.
    - You can reduce complexity by extracting sub-branches to separate units of no more than 4 branch points.
3. **Write Code Once**
    - When code is copied, bugs need to be fixed in multiple places. This is both inefficient and error-prone.
    - Avoid duplication by never copy/pasting blocks of code.
    - Reduce duplication by extracting shared code, either to a new unit or to a superclass.
4. **Keep Unit Interfaces Small**
    - Keeping the number of parameters low makes units easier to understand and reuse.
    - Limit the number of parameters per unit to at most 4.
    - The number of parameters can be reduced by grouping related parameters into objects.
    - Alternatively, try extracting parts of units that require fewer parameters.
5. **Separate Concerns in Modules**
    - Keep the codebase loosely coupled, as it makes it easier to minimize the consequences of changes.
    - Identify and extract responsibilities of large modules to separate modules and hide implementation details behind interfaces.
    - Strive to get modules to have no more than 10 incoming calls.
6. **Couple Architecture Components Loosely**
    - Having loose coupling between top-level components makes it easier to maintain components in isolation.
    - Do this by minimising the amount of interface code; that is, code in modules that are both called from and call modules of other components (throughput), and code in modules that are called from modules of other components (incoming).
    - You can hide a component's implementation details through various means, e.g. using the "abstract factory" design pattern.
7. **Keep Architecture Components Balanced**
    - Balancing the number and relative size of components makes it easier to locate code.
    - Organize source code in a way that the number of components is between 2 and 12, and ensure the components are of approximately equal size (keep component size uniformity less than 0.71).

[1]https://bettercodehub.com

- For actionable results, you need to define architecture components, for example by configuring a zoom level in the tab on the right of the guideline view. A zoom level recognizes as architecture components the folders at a particular nesting depth.

8. **Keep Your Codebase Small**
   - Keeping your codebase small improves maintainability, as it's less work to make structural changes in a smaller codebase.
   - Avoid codebase growth by actively reducing system size.
   - Refactor existing code to achieve the same functionality using less volume, and prefer libraries and frameworks over "homegrown" implementations of standard functionality.
   - Strive to keep volume below 20 Man-years.
9. **Automate Tests**
   - Automating tests for your codebase makes development more predictable and less risky.
   - Add tests for existing code every time you change it.
   - For small systems (less than 1,000 lines of code), you should have at least some test code and one assertion (currently only checked for Java and C# systems).
   - For medium systems (less than 10,000 lines of code), the total lines of test code should be at least 50% of the total lines of production code, and the assert density (percentage of lines of test code containing assertions) should be at least 1% (currently only checked for Java and C# systems).
   - For large systems (more than 10,000 lines of code), the total lines of test code should be at least 50% of the total lines of production code, and the assert density should be at least 5% (currently only checked for Java and C# systems).
10. **Write Clean Code**
    - Clean code is more maintainable. Proactively search and remove code smells.
    - Remove useless comments, commented code blocks, and dead code. Refactor poorly handled exceptions, magic constants, and poorly named units or variables.

The first, fourth and ninth guideline above were already pointed out by the SIG feedback. The following code snippet which was mentioned by SIG, has been refactored.

```php
    public function readAndroidFile(string $location, string $path_id, int $language_id, string $path):
     Collection
    {
        $xml = simplexml_load_file($path);

        $android_file = collect();
        foreach ($xml->string as $text) {
            if ($text->__toString() != '') {
                $text_dom = dom_import_simplexml($text);

                $total_text = '';
                foreach ($text_dom ->childNodes as $childNode) {
                    $total_text = $total_text . $text_dom->ownerDocument->saveXML($childNode);
                }

                $total_text = strtr($total_text, [
                    '&lt;' => '<',
                    '&gt;' => '>',
                    '&quot;' => '"',
                    '&apos;' => "'",
                    '&amp;' => '&',
                ]);

                $total_text = stripslashes($total_text);

                $placeholders = $this->checkPlaceholders($total_text, true);

                $android_instance = [];
                $android_instance['text'] = $placeholders['text'];
                $android_instance['path_id'] = $path_id;
                $android_instance['language_id'] = $language_id;
                $android_instance['location'] = $location;
                $android_instance['placeholders'] = $placeholders['placeholders'];

                foreach ($text->attributes() as $attribute => $value) {
                    if ($attribute == 'name') {
                        $android_instance['android_key'] = $value->__toString();
                    } elseif ($attribute == 'translatable') {
                        $android_instance['translatable'] = ($value->__toString() == 'true');
                    }
                }

                $android_key = $android_instance['android_key'];
                $key_exists_file = $this->keyExistsInFile($android_key, 'android', $android_file);
                if ($key_exists_file) {
                    // Todo: throw exception?
                    echo 'Two identical iOS keys (' . $android_key . ') exist in same file (' . $location . ').';
                }

                $key_exists_database = $this->keyExistsInDatabase($android_key, $path_id, true);
                if (!$key_exists_database) {
                    $android_file->push($android_instance);
                }
            }
        }

        return $android_file;
    }
```

As seen above, the code was first 57 lines in total. However after refactoring, the method's logic has been split up into three different methods all less than 30 lines each:

```php
    public function readAndroidFile(array $path_info, int $language_id): Collection
    {
        $xml = simplexml_load_file($path_info['file_path']);
        $android_file = collect();

        foreach ($xml->string as $string_instance) {
            if ($string_instance->__toString() != '') {
                $text = $this->getStringText($string_instance);
                $placeholders = $this->getPlaceholdersUpdateText($text, true);

                $android_instance = [
                    'text' => $placeholders['text'],
                    'path_id' => $path_info['path_id'],
                    'language_id' => $language_id,
                    'location' => $path_info['location'],
                    'placeholders' => $placeholders['placeholders']
                ];

                $attributes = $this->getStringAttributes($string_instance);
                $android_instance = array_merge($android_instance, $attributes);

                $key_exists = $this->keyExistsInDatabase($android_instance['android_key'], $path_info['path_id'], true);
                if (!$key_exists) {
                    $android_file->push($android_instance);
                }
            }
        }

        return $android_file;
    }

    public function getStringText(\SimpleXMLElement $string_instance): string
    {
        $string_instance_dom = dom_import_simplexml($string_instance);

        $text = '';
        foreach ($string_instance_dom->childNodes as $childNode) {
            $text = $text . $string_instance_dom->ownerDocument->saveXML($childNode);
        }

        $text = strtr($text, [
            '&lt;' => '<',
            '&gt;' => '>',
            '&quot;' => '"',
            '&apos;' => "'",
            '&amp;' => '&',
        ]);

        return stripcslashes($text);
    }

    public function getStringAttributes(\SimpleXMLElement $string_instance): array
    {
        $attributes = [];
        foreach ($string_instance->attributes() as $attribute => $value) {
            if ($attribute == 'name') {
                $attributes = array_merge($attributes, ['android_key' => $value->__toString()]);
            } elseif ($attribute == 'translatable') {
                $attributes = array_merge($attributes, ['translatable' => ($value->__toString() == 'true')
 ]);
            }
        }

        return $attributes;
    }
```

Moreover, the 3rd guideline seems to be relevant to the team. The codebase did have some duplicate code. As described by SIG, *"a well-designed system should not have more than 5% code duplication. Only exceptionally lean systems show duplication lower than 3%. When duplication exceeds 20%, source code erosion is out of control."* [17]. Therefore, the team focused on avoiding duplication by using trait and inheritance. Traits are PHP classes, which can be regarded the same as abstract classes in object oriented programming. They allow developers to reuse code and thus reduce the limitations of single inheritance. Methods which are needed in logically different parts of the codebase were put into these traits. An example of a trait method:

```
    public function getAllBranches(bool $android): array
    {
        $client = $this->getClient($android);
        $response =
            $client->get(
                'repository/branches?per_page=100',
                ['headers' => ['Private-Token' => '5X9D2iDojRAcVmydcfoU']]
            );
        $branches = array_column(json_decode($response->getBody()->getContents(), true), 'name');

        return $branches;
    }
```

This `getAllBranches()` method is used in two separate actions within the application. First, the user must be able to choose which branches of the app repositories certain features should be committed to and second, the user must be able to indicate on which branches the mapped keys should be committed to. These actions are logically not related to each other within the controller classes. The team therefore chose to make this method a trait method.
Regarding guidelines 5 to 7, most of the module and component structuring of the code is already done by the Laravel framework. Therefore, the team has not focused on improving on these aspect of the analysis. Furthermore, the team did not receive negative feedback regarding these guidelines. Guideline 8 was not taken into consideration since the product was created within a short timeframe. It is not logical to refactor the code since the codebase is still very small. For guideline 9, the team has chosen to test at least 70% of the codebase with unit tests and/or functional tests, hereby adhering to the rules set in Better Code Hub. By using Codesniffer, which has already been set up by bunq developers, the team continues to improve the cleanliness of the code.

Besides keeping the aforementioned guidelines in mind, the team also concerned itself with the length of the classes in the codebase. However, no guidelines were given for this and therefore the team emailed SIG to ask what their advice is regarding class length. The following is the answer the team received:

"To shed some light on your question - Our view here is that every class should implement just a single functionality. And from what our consultants usually see, classes that contain more 1,000 lines of code typically contain multiple functionalities and could be split into multiple classes. As a best practice, we therefore recommend each class be kept to 1,000 lines of code maximum."

As the guideline for the class length is 1000 lines, the team did not deem it necessary to refactor the existing classes. The two longest classes are currently, merely 580 and 445 lines and thus are only half as long as the advised maximum length. Furthermore, the team thinks that the functionalities are already grouped logically within the existing classes.

9.3. Future work

The following sections described the most important aspects which are still to be improved or implemented in the application.

9.3.1. Android app strings

The Android app contains multiple types of string resources, however not all of them are processed by the application yet. Below is an overview of the types of strings present in bunq's Android app. The pie chart in figure 9.1 shows an overview of the distribution of the string resource types present in bunq's Android repository. As can been seen in the chart, most string resources in the repository (97.56%) are resources with the `string` tag and `name` attribute, where some strings have an additional `translatable` attribute (lines 1-2 in the code below). Currently, these are the only resource types handled by the application. The remaining 2.44% of the strings, which can be found in the lines 4-22 of the code below, are not yet handled by the application and are recommended for future work.

```
1  <string name="app_name" translatable="false">bunq</string>
2  <string name="title_fragment_recent_contact_picker">Recent</string>
3
4  <string name="contact_group_favorite">@string/title_fragment_recent_contact_picker</string>
5
6  <string name="registration_company_ubo_inform_long" formatted="false">
7      Ultimate Beneficial Owners of a legal entity are natural persons who (indirectly) hold or
8      control a participation of more than 25&#37; in the capital or more than 25&#37; of the voting
9      rights of a company, or who undertake de jure or de facto management of the legal entity.
10 </string>
11
12 <plurals name="days">
13     <item quantity="one">%d day</item>
14     <item quantity="other">%d days</item>
15 </plurals>
16
17 <string-array name="bunq_to_status">
18     <item>Payment sent</item>
19     <item>Payment denied</item>
20     <item>Payment cancelled</item>
21     <item>Payment expired</item>
22 </string-array>
```

The main reason for not implementing these types of strings is the amount of occurrences versus the time-frame. Approximately 2% of the strings in the Android repository are of the string resource type, of which an example can be found in line 4 of the code above. Currently, these strings are treated the same as other strings with the `string` tag. However, for future use the team recommends that the references get treated as a reference and stored in the database as such. This could be done by adding an extra references table to the database, which has a many-to-one relationship with the keys table. By adding this table, the references could be updated whenever the Android keys get renamed.

The resource type containing the `formatted` attribute is only present in one string of the app. It could be implemented easily by adding a `formatted` column to the keys table in the database. However, as discussed with the client, it did not seem necessary to take this single string into account. Besides the `formatted` attribute, plurals and string arrays could also be added to the database quite easily. The `name` attribute could be stored as key and the corresponding `item` tags could be stored as items. An extra column could then be added to the keys table containing the string type. However, it would require quite some changes to the application's code to handle that data. Currently, the application is built to expect one item for each language per key. Plurals and string arrays would contain more items for each language per key. Therefore, a lot of changes in the code would be required to handle these resource types, such as extra write functions for plurals and string arrays. Since these two types only account for approximately 0.3% of the total amount of strings, the team decided that this was not necessary given the timeframe. However, the team suggests that this should be taken into account, when continuing the work of this application.

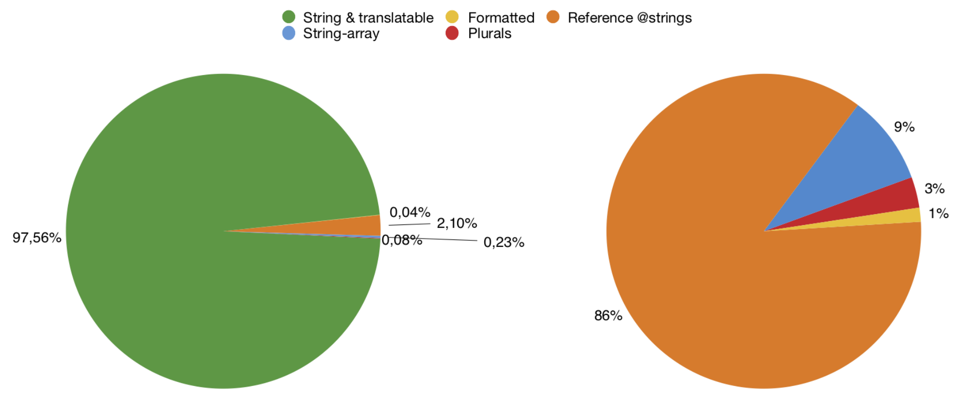


Figure 9.1: Overview of the distribution of the string types in bunq's Android repository.

9.3.2. Security

At the moment the application has some security flaws. This is mainly because the user currently has to access the application from bunq's WiFi and on the VPN, since it can only be used by bunq employees. In the future, bunq may want to expand to using crowd translators and approvers for their translations. Before this is possible, several components of the application must be adapted.

To begin with, the connection protocol needs to be changed from HTTP to HTTPS. This can easily be done. bunq already has a certificate for their website, which means that it can choose to either use the same certificate, if this application uses the same domain name as the bunq website, or buy a new certificate for the application. This will encrypt the website and will secure confidential data. Therefore, including this into the application will make the connection between client and server more secure.

Furthermore, the application uses the authentication feature Laravel provides. At bunq they make use of LDAP user authentication. LDAP is an application protocol that uses an IP network to manage and access the distributed directory information service [27]. Therefore, bunq employees could use their bunq account to log into the application instead of creating a new one themselves. However, users who do not work at bunq do not have a LDAP account and, thus, they still need to create a new account for this application.

9.3.3. Crowdsourcing

The application is at the moment meant for bunq employees. To begin with, the application is not secure enough as can be read in the section above. Most importantly, bunq does not want that everyone is able to submit a translation, only bunq users should be allowed. Therefore, the application should be linked to the bunq app to be able to verify if someone is a bunq user or not. Furthermore, at the moment everyone is able to translate any language they want to. There is no check if the person is qualified to enter a translation in a given language. To fix this problem a small test could be created to ensure someone has a certain level of that language. Additionally, a translator is currently able to submit a translation and to review translations. bunq wants to have the final say if a translation meets bunq's standards. To fix this, a role has to be added: reviewer. A translator is only allowed to submit translations and a bunq employee who has the role reviewer must review the translation before it appears in the code base of bunq apps. Finally, there is no page which informs the translator how the translation should look like. For example, a Design Guide that provides the rules on punctuation, capitalisation and text formatting. Otherwise, bunq employees must still adjust each translation to their preferred format.

10 Conclusion

At the beginning of this project, the team received the assignment to solve an inefficiency problem at bunq. As defined in the problem definition of Chapter 2, the issues were summarised. These issues and how they are fixed are stated below:

Translating the strings is time-consuming
The application provides the bunq employees a platform specifically designed for translating their Android and iOS app. Instead of having to create GitLab issues for every newly added string, the application provides a clear overview of all strings that need to be translated.

Developers lose time by having to copy and paste the strings into the code
The developers do not have to manually copy and paste the strings into the code anymore. Instead, the application automatically inserts the changes into the code when the developer wishes.

Inconsistency mistakes between the apps are made when inserting the strings into the code
As mentioned before, the developers do not have to add the translations in the code manually anymore, because the system always commits the same string to both app repositories. Furthermore, the consistency of the existing strings is ensured by the initialisation process.

Translations might be incorrect
Due to the reviewing feature within the application, the chances of having incorrect translation occurring in the bunq apps are diminished.

Fluent speakers are needed for each language
This issue is not yet resolved as the crowdsourcing feature has not completely been integrated into application. The fluent speakers are still needed to translate the strings.

In short, the application resolves most of the main issues of the current translation process at bunq, as stated above. Adding a new translation is now more efficient. With this application, the developer only needs to add a language and the system will show the English version of all the strings in the overview of that language. A translator can easily submit its translation which will be committed to the code. Therefore, the developer does not need to create a ticket, whereafter they need to copy paste the created translation in both the iOS and Android app. Furthermore, the chance of having an incorrect translation in the bunq apps is decreased by the added review functionality. Additionally, the application provides a structure where all actions needed to be taken to add a new string to the bunq apps can be done in one application.Thus, the application eases creating translations for the bunq apps.

To conclude this report, the team is confident that the application created over the past 10 weeks can be deemed successful, as the most important requirements are all present in the final product.

Glossary

Copy writer A bunq employee who reviews/improves the developer-made strings. This person improves the strings to have consistent terminology and correct grammar, but does not translate any strings.

Crowd approver A bunq user who is appointed to help approve the crowd translated strings (not someone from bunq).

Crowd translator A bunq user who is appointed to help translate the strings (not someone from bunq).

Developer string The original string that is created by the developer. This string is not yet reviewed by copy writers to have consistent terminology and correct grammar.

Keys These words are the names given to the strings in the code. The Android and iOS systems sometimes use different ones for the same item.

Reviewed string The improved developer string with consistent terminology and correct grammar.

Translator A bunq employee who translates strings.

Translation reviewer A bunq employee who reviews the translations from the crowd translators.

Bibliography

[1] What is crowdsourcing? URL https://crowdsourcingweek.com/what-is-crowdsourcing/.

[2] Modular crypt format. URL http://passlib.readthedocs.io/en/stable/modular_crypt_format.html.

[3] Usage of javascript for websites. https://w3techs.com/technologies/details/cp-javascript/all/all.

[4] About internationalization and localization. https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPInternational/Introduction/Introduction.html, Sep 2015.

[5] 9 benefits of laravel application development, Mar 2016. URL http://www.cmarix.com/9-benefits-of-laravel-application-development/.

[6] Programming language usage. https://trends.builtwith.com/framework/programming-language, May 2016.

[7] Input validation cheat sheet, Nov 2017. URL https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet.

[8] Php framework comparison: Codeigniter vs laravel vs cakephp vs yii. https://therightsw.com/codeigniter-vs-laravel-vs-yii-vs-cakephp/, Jul 2017.

[9] Localize your app. https://developer.android.com/guide/topics/resources/localization, Apr 2018.

[10] 11 best php frameworks for modern web developers in 2018. https://coderseye.com/best-php-frameworks-for-web-developers/, Jan 2018.

[11] Vamshi Ambati. *Active Learning and Crowdsourcing for Machine Translation in Low Resource Scenarios.* PhD thesis, Pittsburgh, PA, USA, 2012. AAI3528171.

[12] Dimitra Anastasiou and Rajat Gupta. Comparison of crowdsourcing translation with machine translation. 37:637–659, 12 2011.

[13] R. Bogaarts. Ali niknam: "ik steek zo nodig alles wat ik heb in bunq". https://www.deondernemer.nl/nieuwsbericht/170906/ali-niknam-ik-steek-zo-nodig-alles-wat-ik-heb-in-bunq, 2017.

[14] Dina Djordjevic. Traditional banks vs digital banks?-?challenges and opportunities, Sep 2017. URL https://blog.getzuper.com/traditional-banks-vs-digital-banks-challenges-and-\opportunities-7f5e90b8511d.

[15] H. Fatima and K. Wasnik. Comparison of sql, nosql and newsql databases for internet of things. In *IEEE Bombay Section Symposium 2016: Frontiers of Technology: Fuelling Prosperity of Planet and People, IBSS 2016*, 2017. doi: 10.1109/IBSS.2016.7940198. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85022326280&doi=10.1109%2fIBSS.2016.7940198&partnerID=40&md5=e6dbd3d674b9fbec513a874526f34c4a.

[16] Ivaylo Gerchev. The 5 most popular front-end frameworks compared. https://www.sitepoint.com/most-popular-frontend-frameworks-compared/, Feb 2018.

[17] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, 2007. doi: 10.1109/quatic.2007.8.

[18] Rasmus Lerdorf, Kevin Tatroe, and Peter MacIntyre. *Programming PHP*. OReilly Media, Inc, USA, 2013.

[19] Y. Li and S. Manoharan. A performance comparison of sql and nosql databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 15–19, Aug 2013. doi: 10.1109/PACRIM.2013.6625441.

[20] Security Innovation Europe Ltd. What is the difference between hashing and encrypting, Oct 2016. URL `https://www.securityinnovationeurope.com/blog/page/whats-the-difference-between-hashing-and-encrypting`.

[21] Manmohan. 7 frameworks for automated php testing. `http://evontech.com/what-we-are-saying/entry/7-frameworks-for-automated-php-testing.html`, Jan 2017.

[22] Martin. Top programming languages used in web development. `https://www.cleverism.com/programming-languages-web-development/`, Jun 2015.

[23] Anna Monus. 10 best automated testing frameworks for php. `https://www.hongkiat.com/blog/automated-php-test/`, Oct 2015.

[24] Adnan Raja. Why use symfony? 6 arguments for using symfony php framework. `https://www.atlantic.net/managed-hosting/why-use-symfony-php-framework/`, Sep 2017.

[25] S. Rautmare and D. M. Bhalerao. Mysql and nosql database comparison for iot application. In *2016 IEEE International Conference on Advances in Computer Applications, ICACA 2016*, pages 235–238, 2017. doi: 10.1109/ICACA.2016.7887957. URL `https://www.scopus.com/inward/record.uri?eid=2-s2.0-85018358240&doi=10.1109%2fICACA.2016.7887957&partnerID=40&md5=b22ad7c36e82f0b5e36e90fc75555c8e`.

[26] Frank Rietta. Use bcrypt or scrypt instead of sha* for your passwords, please!, Feb 2016. URL `https://rietta.com/blog/2016/02/05/bcrypt-not-sha-for-passwords/`.

[27] Margaret Rouse. What is ldap (lightweight directory access protocol)? - definition from whatis.com, Nov 2008. URL `https://searchmobilecomputing.techtarget.com/definition/LDAP`.

[28] Mindfire Solutions. Advantages and disadvantages of python programming language. `https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121`, Apr 2017.

[29] StatCounter. Browser market share worldwide. `http://gs.statcounter.com/browser-market-share`.

[30] SDL Trados. What is machine translation? URL `https://www.sdltrados.com/solutions/machine-translation/`.

[31] W. van Noort. Bunq: "whatsapp voor banken". `https://www.nrc.nl/nieuws/2015/11/25/bunq-whatsapp-voor-banken-1559753-a310271`, 2015.

[32] Rohith VR. 10 best automated php testing frameworks 2016. `https://www.ideaherald.com/best-automated-php-testing-frameworks-2195/`, Mar 2016.

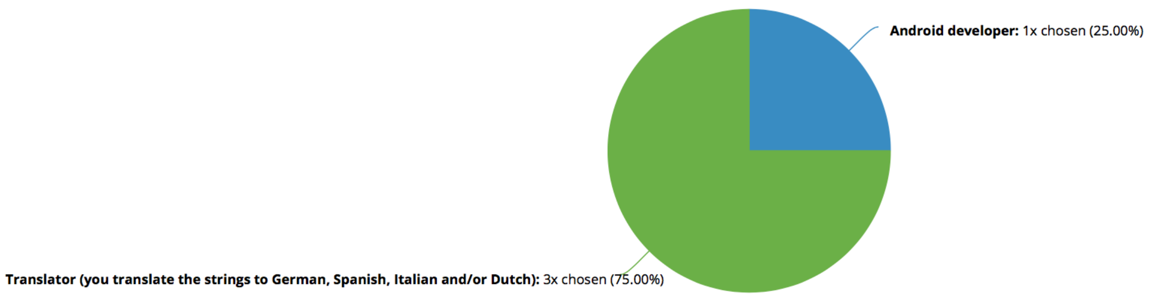
[33] Mike Wasson, Christopher Bennage, Nikita Golovin, Peter Taylor, Micheal Hauss, Bálint Szabó, and Caro Caserio. Data partitioning guidance, Jul 2016. URL `https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning`.

[34] Omar F. Zaidan and Chris Callison-Burch. Crowdsourcing translation: Professional quality from non-professionals. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1220–1229, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1122`.

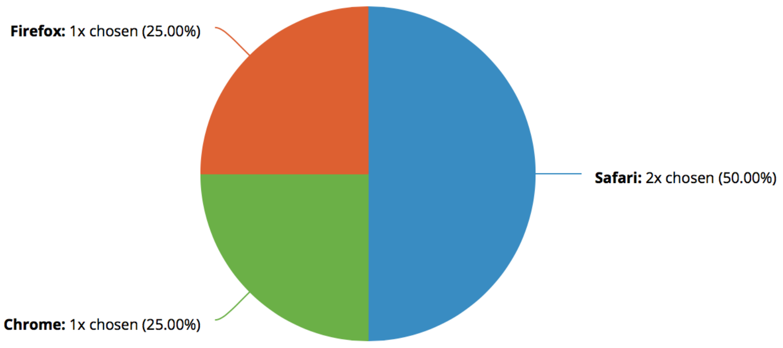
A

Appendix: Questionnaire results

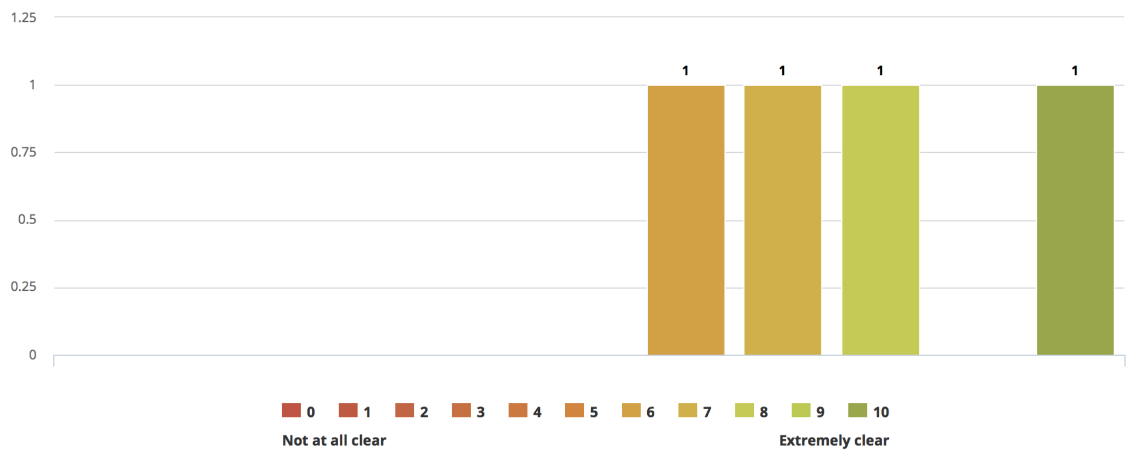
1. **What is your role in the translation process?**



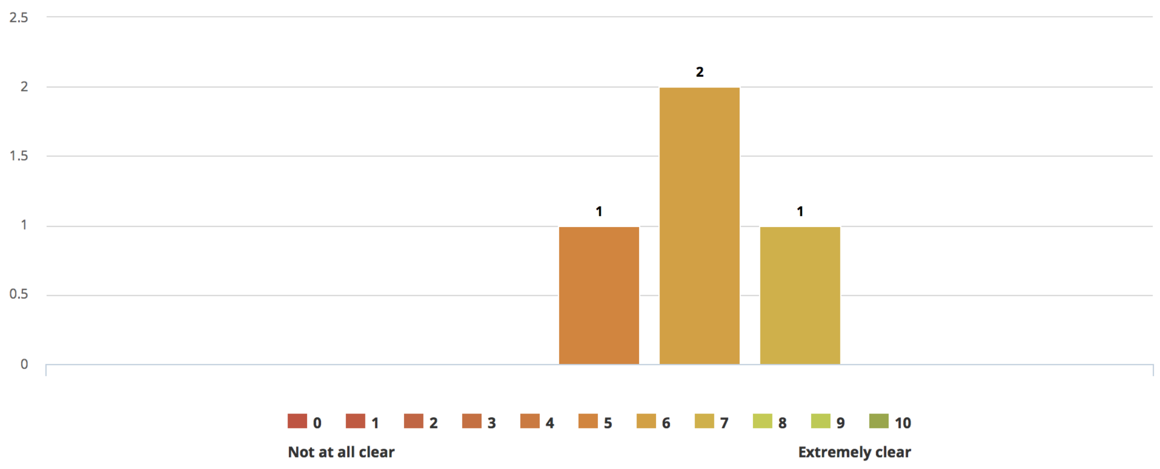
2. **Which browser did you use for the application?**



3. **How clear was the goal of the application?**

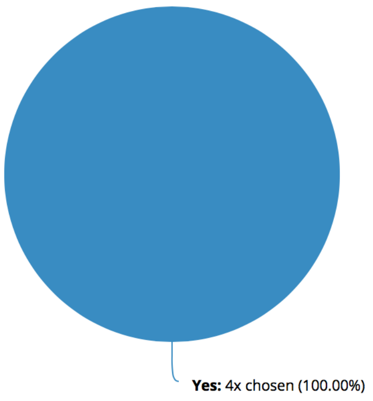


4. **How clear was the application's interface?**



5. Help page

   (a) **Did you need the help page?**

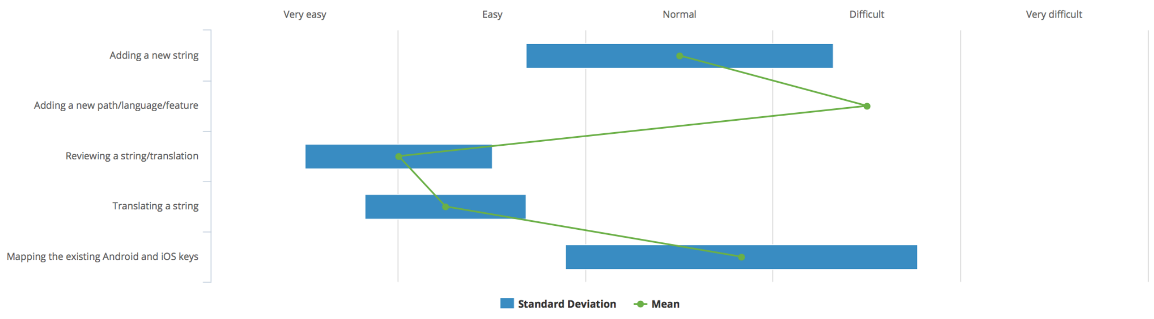


   (b) **If you needed the help page for using the application, please inform us why.**

   - At first, when I tried adding a new string I could not find it in the translate page
   - I had not noticed the icon to be fair, but would've been really useful for terminology. What's a key? What is considered a string? etc.

- - check how to submit a translation
  - if it was possible to see the name of the string
  - check if I can see a translation memory
  - check if I can see also similar strings already translated
- The order/flow of how/when/why to do things did need a bit of explanation the first time I saw the home screen.

6. **Rate the ease of use of the application's features**



7. **Name at least 1 negative aspect of the application.**

   - Some functionalities are not completely clear without using the help page
   - Some parts of the interface assume context knowledge of the users.
   - there is (I think) no way to
     - see existing similar strings and terms
     - see the translation memory
   - The way file paths and feature branches are selected/added can be improved still. The overall UI and 'flow' of doing things could be improved still to make it a bit more user friendly (also for non-technical people) and in such a way that the help page isn't needed anymore ;)

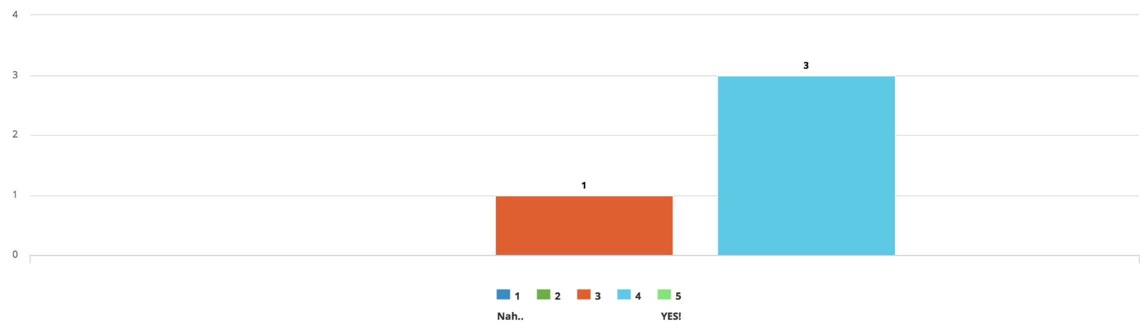
8. **Name at least 1 positive aspect of the application.**

   - The basics for reviewing and translating strings exists and work well
   - It looks really nice and solid! Nice!
   - Easy interface
   - It does what it should and integrates nicely with our existing codebase and way of working. It also doesn't require us starting to use this immediately for every string we add.

9. **Did you have any expectations for the application prior to using it? If so, which ones have not been met?**

   - I expected to be able to translate and easily define new ones (as a developer). This turned out to be more difficult than I expected. :)
   - I didn't expect all features to be implemented but almost all of them are. The stability could be a bit better at some times though (less errors), but that was expected a bit.

10. **Would you like to use the application in the future?**



11. **Do you have any further comments?**

    - Nice work girls!

B Appendix: Project infosheet

Title of the project: Crowdsourcing <> String Translations
Name of the client organization: bunq
Date of the final presentation: July 4, 2018
Name and affiliation of the client: W. Van, Lead of Android development, bunq
Name and affiliation of the project coach: A. Zaidman, Software Engineering Research Group, Delft University of Technology

B.1. Description

Over the course of 10 weeks we worked on an assignment from bunq, a Dutch bank which received its banking license in November 2015. Currently, the bunq employees have to manually create the translations for the two mobile apps they have, which is a costly process. The core challenge of the project was to create an application from scratch to help bunq with their translation process. The application's goal is to simplify, automate and speed up the translation process.

The first two weeks we only focused on research whereafter we concluded which technologies suit this project best. During this phase, decisions were made on the structure of the code. From this research, we have decided to implement our application with the coding language PHP. Furthermore, the application is built on the Laravel web application framework in combination with a MySQL database. The components were tested using the PHP testing framework Codeception, which allowed us to create unit and functional tests. In addition, CodeSniffer was used to improve the code quality.

We used SCRUM as our software development framework and stored our code in a bunq hosted GitLab repository. During the course of the project, the client's wishes changed and thus the problem tackled by the team also altered. As a result, the team had to reevaluate the product's requirements and design goals. Furthermore, the application's initialisation with the existing records of bunq's apps turned out to be more extensive than anticipated. This is what led to the project being split into two separate components of the product and each component was worked on by two members of the team.

The application we created currently allows the user to add new strings, translate them, review them, update them and thereafter commit them. This can all be done in the application's web interface. Since it is important to have a complete database of the already existing strings in both apps, the bunq developers are also able to map these strings and their keys together. The system reads the files in which the strings are stored from the repositories and commits the changes back to create consistency between the two apps.

Currently, the application is not in use yet as it is partially incomplete. We have given recommendations on how they can expand the application, mainly on the crowdsourcing aspect. Furthermore, small features that are already implemented but not yet integrated into the web application are ready to be added. We expect the client to make use of the product as the feedback we received is positive.

B.2. Team

R. M. Borhem: mborhem@gmail.com
Contributions and role: acted as SCRUM master and main contact person, mainly worked on the string translations component.

B. Hizli: beyzahizli@gmail.com
Contributions and role: acted as design lead and was responsible for documentation, mainly worked on the string translations component.

E. L. Jimmink: emmajimmink@live.com
Contributions and role: acted as lead programmer, mainly worked on the initialisation component.

Z. A. Van Steijn: zoevansteijn@hotmail.com
Contributions and role: acted as lead tester, mainly worked on the initialisation component.

All team members contributed to preparing the report and the final project presentation.

The final report for this project can be found at: http://repository.tudelft.nl/.

C Appendix: Original project description

Italian, Spanish, English, German, French, Dutch. We're an international company, with great apps to match that ambition: everything's translated in local languages for markets we are active in (or aim to be active in). Right now, this is all done manually. We're on the lookout for a smarter system in which we might even use the power of the crowd to insert specific translations in our app. Translations that better match the cultural nuance that is sometimes needed when you go abroad! This is where this project comes in: can we build an automated system that gathers input through e.g. Together (together.bunq.com) or from within our apps to improve the quality of our translations, or even automate translating of our apps altogether?

C.1. Potential research questions

- Are crowdsourced translations better than those supplied by centralised translators?
- What is the most effective way of gathering string translations from a crowd and assessing quality?
- The aim is to develop a (crowdsourced) string translation insertion system for our entire platform.

C.2. About us

Some years ago a bunch of coders decided they would challenge the status quo. Together we set out to create the true alternative to traditional banking. A bank that would be different. The bank of the free.

You will join a team of over 80 passionate bunqers who've come from all over the world (13 nationalities and counting) to build something amazing together. Waking up with brilliant ideas and going to bed knowing that you've made them happen. That's working at bunq. We are using the unlimited power and possibilities of code to bring innovation to a place where stagnation is the status quo: the finance industry. We don't limit ourselves to what has been done or could be done: we make shit happen.

C.3. What we offer

- A monthly internship compensation of 500,-
- Reimbursement of travelling expenses
- Freedom and responsibility from day one
- The best food (and drinks!) to keep your engine running
- The bunqr - our spacious office located at walking distance from train station Amsterdam Sloterdijk

D Appendix: Software Improvement Group: feedback

D.1. First feedback

D.1.1. Dutch

De code van het systeem scoort 3.8 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Size en Unit Interfacing vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld *[no example given]*, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes.

In jullie project is `store()` in `KeyController.php` een voorbeeld waarbij je al snel kunt zien dat deze methode meerdere verantwoordelijkheden heeft. De "blokken" functionaliteit worden nu door middel van commentaar van elkaar gescheiden. Door deze indeling met codestructuur aan te geven wordt het makkelijker om die blokken onafhankelijk van elkaar te kunnen testen.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden.

Bij jullie is `createPh()` in `MappingController.php` een goed voorbeeld. Die heeft nu 6 parameters, wat nog niet enorm veel is, maar `$ph` en `$ph_names` zijn nu aparte parameters terwijl ze eigenlijk bij elkaar horen. Jullie nemen in de code nu aan dat de lengte van beide arrays altijd hetzelfde is. Die aanname is niet onmiddellijk duidelijk voor de persoon die de methode aanroept. Het is daarom beter om een parameter-object te introduceren, zodat de naam van de placeholder en de tekst samen in één object zitten. Het lijkt overdreven om zo'n klein object te maken, maar juist dit soort kleine abstracties zorgen er vaak voor dat de code op termijn leesbaar blijft en duidelijk is wat er met bepaalde parameters bedoeld wordt.

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

D.1.2. English

The code from the system scores 3.8 stars on our maintainability model, which means that the code is market average maintainable. Unit Size and Unit Interfacing can be seen as possible areas of improvement, due to the lower scores on those parts.

Unit Size is concerned with the percentage of code that has an above average length. Splitting these types of methods into smaller parts ensures that each segment becomes easier to understand, easier to test and therefore easier to maintain. Within the longer methods of this system, for instance *[no example given]*, separate parts of functionality can be found, which can be refactored into separate methods.

In your project the `store()` method in `KeyController.php` is an example in which you can easily see that this method has multiple responsibilities. The "blocks" of functionality are now separated by comment lines. By denoting this division in the code structure, it becomes easier to independently test these blocks of code.

Unit Interfacing is concerned with the percentage of code in units with an above average amount of parameters. An above average amount of parameters generally indicates a lack of abstraction. Furthermore, a large amount of parameters often leads to confusion when calling the method and in most cases also leads to longer and more complex methods.

In your case `createPh()` in `MappingController.php` is a good example. Currently, the method has 6 parameters, which is not yet an enormous amount, however `$ph` and `$ph_names` are separate parameters while they actually belong together. In the code you assume that the length of both arrays is always equal. This assumptions is not immediately clear for the caller of the method. Therefore, it is better to introduce a parameter-object, so that the name of the placeholder and the text are contained in one object. It seems exaggerated to create such a small object, however these types of abstractions often ensure that the code remains readable and that the meaning of certain parameters is clear.

One final remark is that no (unit)test-code was found in the code-upload. It is recommendable to at least define automatic tests for the most important parts of the functionality, to make sure that no unwanted behaviour is introduced by certain adjustments in the code.

D.2. Second feedback

D.2.1. Dutch

Unit Size en Unit Interfacing blijven als aandachtspunten staan, maar de specifieke voorbeelden zijn inderdaad niet meer geldig.

Voor Unit Size is `readAndroidFile()` in `InitAndroidTrait.php` nu een goed voorbeeld. Volgens mij is die methode ook nog niet helemaal af, want er staat nu een hard-coded pad naar "/Users/zvansteijn" in. Desondanks zou ik nog eens goed kijken naar de verschillende functionaliteiten binnen deze methode, daar zou een duidelijker scheiding tussen kunnen.

Voor Unit Interfacing blijven de eerder genoemde voorbeelden staan, en je zou daarnaast naar `Item.newPivot()` kunnen kijken.

Wat betreft de testcode ziet de nieuwe upload er inderdaad een stuk beter uit. Zorg wel dat het niet bij een eenmalige actie blijft, idealiter zouden we in de volgende upload naast nieuwe productiecode ook weer nieuwe testcode willen zien.

Jullie score is met 3,7 ongeveer hetzelfde als voorheen. Dat is ook expres zo, de score kijkt naar alle code, en het is daarom moeilijk om er snel verandering in te brengen, zeker als het volume tussendoor aan het groeien is.

D.2.2. English

Unit Size and Unit Interfacing are still points of consideration. However, the specific examples are no longer valid.

`readAndroidFile()` in `InitAndroidTrait.php` is a good example for Unit Size. I think that the method is not finished yet, as there is currently a hard-coded path to "/Users/zvansteijn". Nevertheless, I would take a good look at the different functionalities within this method, as there could be a more clear division between these parts.

For Unit Interfacing the examples mentioned earlier are still relevant. Furthermore, you could look at `Item.newPivot()`.

With regards to the test code, the new upload indeed looks a lot better. Make sure that it is not a one-off action. Ideally, we would like to see new test code besides the new production code in the next upload.

Your score of 3.7 is about the same as last time. That is done on purpose, since the score takes all code into consideration. For that reason, it is difficult to quickly bring changes to the score, especially when the volume is growing in the meantime.