
Deployment and Management of the Giraf Project

Developing Complex Software Systems

Aalborg University
Software – 2015
Project group SW605f15

Resumé

We were a four-man group which worked on the Giraf project, which is a collection of apps that helps children with autism and their caretakers. The aim of the project this semester, as it entered its fifth year of development, was to consolidate earlier gains; only a few new features were added and there was a focus on making the project ready for deployment.

Our group worked in the subproject called Build and Deployment (B&D), which is a division of the greater whole of the project focused entirely on making the deployment as smooth as possible and easing the workload for the other developers. We had roles as product owners, and were responsible for managing Google Play and Google Analytics.

As a B&D group we worked on improving the development environment through AAR files. We wrote guides to be shared on Redmine for how to use Google Analytics, how to forward crash reports from Google Analytics, how to use Google Play, how to set up Android Studio, how to rename apps and libraries in Android Studio, how to implement the pictosearch library, and how to use Doxygen for code documentation purposes. In addition to this, we mapped the dependencies of the apps, which was important for our efforts in making the apps standalone. We also changed an app into a library, as a part of a process to make the apps standalone.

As product owners we were a driving force in the organization of sprints. We chaired the sprint planning and the sprint ends for the second, third, and fourth sprint for our subproject. We oversaw the assignment of user stories, which we wrote ourselves upon request and kept up to date throughout the project period. In cooperation with the other product owners we worked out a common standard for the user stories and the backlog and evaluated each other's work together.

As responsible for Google Play and Google Analytics we worked on autoforwarding crash reports and on making the apps send in the reports for us to see in the first place. We added an alpha and beta track to the version control on Google Play. We spearheaded the renaming efforts across the project to ensure consistency in the naming of the apps from their titles to their package-names and repositories.



AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9210 Aalborg East
<http://cs.aau.dk/>

Title:

Deployment and Management of
the Giraf Project

Theme:

Developing Complex Software Systems

Project:

6th semester project

Period:

February - May 2015

Group:

SW605f15

Authors:

Claus Nygaard Madsen
Lukas Nic Dalgaard
Martin Juul Johansen
Sean Skov Them

Supervisor:

Bin Yang

Copies: 6

Pages: 94

Appendix: 32

Finished: 26-05-2015

Synopsis:

Project management was the focus for our group in a sixth semester's multiproject, where we worked on apps for children with autism over the course of 4 sprints. This report details the overall structure of the project along with our work as product owners for a division of the overall project. Our contributions vary from managing the Google Play and Google Analytics aspects of an app, the renaming process of apps across a large developer environment, mapping dependencies between apps and libraries, and maintaining a backlog for the Scrum process. Our efforts bettered knowledge-sharing and improved the workflow for the build process of Giraf.

Signatures

Claus Nygaard Madsen

Lukas Nic Dalgaard

Martin Juul Johansen

Sean Skov Them

Preface

Reading guide

This section is a reading guide which contains a glossary which is containing explanations to words and terms that are not apparent from the context they are in.

In addition, it should also be noted that each chapter and section starts with a short introductory description in italics.

Word list

Our, We: Refers to our own group.

Customer: Refers to the external customers from the institutions the project is working with.

Project: Refers to the Giraf project as a whole.

Subproject: Refers to one of the three parts of the project. The project was split into Build and Deploy (B&D), Database (DB), and Graphical interface (GUI). If no specific subproject is stated it refers to the B&D subproject.

App: Picked in favor of the longer ‘application’.

Launcher: Refers to the launcher, which is the main app in the Giraf-system from which all the other apps are executed.

Glossary

Activity: An activity is a class that takes care of creating a window, which is used to interact with the user through UI elements. An activity can be created or destroyed from other activities. Activities can work in the background or be set to pause when another activity comes to the foreground, to insure that not all activities use memory, when not in the foreground.

Pictogram: A pictogram is a drawing that represent a physical object with some short explaining text, which can then be used by a person with autism to communicate.

Package-name: A package-name is a text-string used by Google Play to uniquely identify an app. We use the newest package names to refer to the apps.

Contents

1	Introduction to Giraf	1
1.1	Status of Giraf	1
1.2	Multiproject, Subprojects, and Responsibilities	1
1.2.1	Database	1
1.2.2	Graphical User Interface	1
1.2.3	Build and Deployment	2
1.3	Project organization	2
1.3.1	Group Organization	2
1.3.2	Roles and responsibilities	3
1.4	Tools used in the project	4
1.4.1	Android Studio	4
1.4.2	Git	4
1.4.3	Jenkins	4
1.4.4	Google Play	5
1.4.5	Google Analytics	5
1.4.6	Redmine	5
1.4.7	Google Docs	5
2	1st Sprint	7
2.1	User Stories and Tasks	7
2.2	Google Play and Analytics	8
2.2.1	Google Play	8
2.2.2	Google Analytics	9
2.3	Central Documentation	10
2.4	Summary	11
3	2nd Sprint	13
3.1	User Stories and Tasks	13
3.2	Dependencies	14
3.2.1	Motivation	14
3.2.2	Applications and libraries	14
3.2.3	Libraries apps are dependent on	14
3.2.4	Cross library dependency	16
3.2.5	Cross app dependencies	17
3.3	Standalone	18
3.3.1	Motivation	18
3.3.2	Features	18
3.3.3	Standalone app or library	19
3.4	Build structure	19
3.4.1	Motivation	19

3.4.2	Android archive	20
3.4.3	Solution	20
3.5	Summary	22
4	3rd Sprint	25
4.1	User Stories and Tasks	25
4.2	Renaming of Package-names	27
4.2.1	Motivation	27
4.2.2	Initial decisions	27
4.2.3	Renaming process	28
4.2.4	Consequences	28
4.3	Improvement of the wiki pages	29
4.3.1	Motivation	29
4.3.2	Process	29
4.4	Core library	30
4.4.1	Pictosearch	30
4.4.2	Download Pictograms	31
4.5	Summary	32
5	4th Sprint	33
5.1	User Stories and Tasks	33
5.2	Package Solution	34
5.2.1	Motivation	34
5.2.2	Approach	34
5.3	Preparations for next semester	34
5.4	Summary	35
6	Roles	37
6.1	Google Play	37
6.2	Google Analytics	37
6.3	Product Owner	38
6.3.1	Backlog	39
7	Collaboration	41
7.1	Build Structure collaboration	41
7.2	Collaboration between product owners	41
7.2.1	Sprint ends and sprint planning	41
7.2.2	Product backlog and release backlog	42
8	Conclusion	43
9	Recommendations for the next semester	45
9.1	Issue handling tool	45
9.2	Renaming	45
9.2.1	Categorygame	45
9.2.2	Various other problems	45
9.3	Core library	46
9.4	Giraf Package	46

9.5	User Story	46
9.6	Backup	46
9.7	Process	47
9.7.1	Leadership	47
9.7.2	Scrum	47
9.7.3	Subprojects	47
9.7.4	Weekly meetings	47
9.8	Server	47
Bibliography		49
A Backlog		51
A.1	Product Backlog	51
A.1.1	GUI	51
A.1.2	Database	57
A.1.3	Build and Deploy	58
A.2	Release Backlog - Sprint 1	60
A.2.1	GUI	60
A.2.2	Database	61
A.2.3	Build and Deploy	61
A.3	Release Backlog - Sprint 2	62
A.3.1	Database	62
A.3.2	Build and Deploy	63
A.4	Release Backlog - Sprint 3	64
A.4.1	GUI	64
A.4.2	Database	65
A.4.3	Build and Deploy	66
A.5	Release Backlog - Sprint 4	67
A.5.1	GUI	67
A.5.2	Database	69
A.5.3	Build and Deploy	69
B Guides		71
B.1	Google Analytics Guide	71
B.1.1	Skaf adgang til data	71
B.1.2	Brugsguide:	71
B.1.3	Tilføj SDK'en til projektet i Android Studio	72
B.1.4	Implementering af Google Analytics i projekt	72
B.1.5	Når der debugges / Application afvikles i Emulator	74
B.1.6	Tracking ID	74
B.2	Google Analytics Crash Reports to Email Guide	74
B.3	How to use Google Play	77
B.3.1	Giving access to alpha and beta-testing	79
B.3.2	Content Classification	79
B.4	Android Studio Setup	79
B.5	App and library renaming in Android Studio	80
B.6	How to implement the Pictosearch library	80

B.7	Javadocs	81
B.7.1	Guidelines	81
B.7.2	Javadocs Example	81
B.8	Dependencies wiki	82
B.8.1	App to library dependencies	82
B.8.2	App to App dependencies	82
B.8.3	Library to library dependencies	83

Introduction to Giraf

1

Giraf is a series of apps for Android, intended to help citizens with autism in their everyday life. Over the past four years Giraf have been under development by students at Aalborg University, with a new group of students taking the mantle every year. As a result of this, it was hard to get an overview of the project in its entirety because we are building upon what others had made. We here give an overview of the project as it was, when we started working on it, and we also give some insight into the organization of the project.

1.1 Status of Giraf

At the start of the project all the apps were already released on Google Play, and most of the apps were executable, except for two of the apps which instantly crash upon launch. Most of the executable apps also had some bugs which in could lead to a crash. All of the apps on Google Play were release-versions, but since some of them could not even be executed, it might have been better to have those on the store as alpha or beta-versions.

1.2 Multiproject, Subprojects, and Responsibilities

The overall goal for the multiproject this year was, to make a working system that is usable and stable. A multiproject in this context being an overall project that several groups, in this case all 6th semester software groups, work on in collaboration. In order to achieve a better workflow, the project was split into three subprojects: Database (DB), Graphical User Interface(GUI), and Build and Deployment(B&D). Our group were a part of B&D, along with 3 other groups. Both the DB and GUI subprojects had 5 groups each. See Figure 1.1 for an overview of the structure of the multiproject for this semester.

1.2.1 Database

DB groups were responsible for handling all things database related, this included making current data accessible and adding new data to the database, as per request from other groups or as needed.

1.2.2 Graphical User Interface

GUI groups were mainly responsible for developing the different apps, this included making general changes in the apps and bug fixing anything related to the apps.

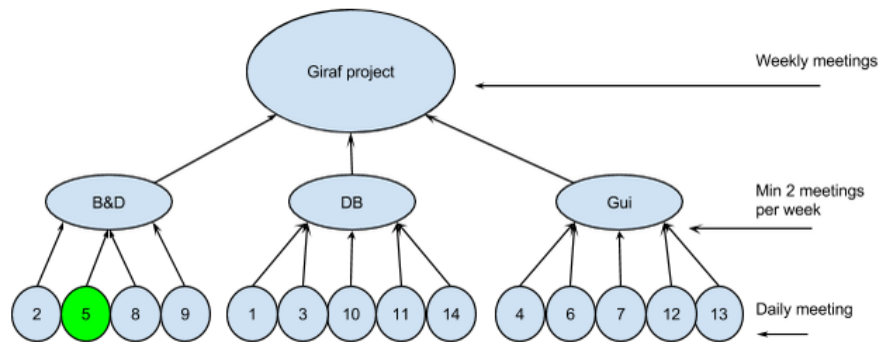


Figure 1.1. Overview of the initial project structure with subprojects and the individual groups depicted, our group is marked with green.

1.2.3 Build and Deployment

B&D groups were responsible for most of the internal project matters, this included maintaining the server services, developing and maintaining the automatic build tools, handling publication of the apps, and administrating the version control. As a B&D group we had to improve the development environment for the project.

1.3 Project organization

The project was organized using Scrum principles, where the individual subprojects held Scrum of Scrum meetings twice a week. The individual groups could follow any development method they wanted to, but Scrum was recommended. In addition, an overall meeting was held every week, where at least one representative from each group were present. At these meetings the overall status for each subproject was given, and any decisions that needed overall consensus were discussed.

As the overall structure was Scrum, the project was split into four sprints. In regards to user stories, it was decided early on at an overall meeting, to operate with two types of user stories, those from the customers, these were called user stories; and those from the other groups, called developer stories. For consistencies sake we will just refer to both as user stories in the report, as the difference was mainly a concern when groups from different subprojects where discussing internally.

1.3.1 Group Organization

Our group used Scrum practices in conjunction with the wider Scrum used in the project. User stories assigned to the group were evaluated using planning poker. Stand-up meetings were held every day. Additionally, we took part in meetings with other groups on subproject and project level. We participated in the project recommendation of being on hand from nine to fifteen every day at the university.

We did not use a burndown chart for our group. Our initial implementation did not pan out for the first sprint, and with our estimations way off and much of our work depending on not being bottlenecked by other groups, we found it to be of too little gain for us. Also,

our tasks related to our user stories were not normal. They varied a lot in complexity and time needed to complete them.

1.3.2 Roles and responsibilities

To delegate responsibility of some critical parts of the project, some groups were given roles, so that everyone knew which groups were responsible for what. This way people knew where to go, if there was anything wrong.

Process: Responsible for ensuring that the overall project follows the scrum principles, and for informing other group about the process used in the project.

Configuration Management: Responsible for keeping track of the different versions of the apps, and for making sure that only the working version of the apps are released.

Google Play: Responsible for managing the apps on Google Play, and for making sure they followed the rules set by Google. This was one of our responsibilities.

Google Analytics: Responsible for managing Google Analytics and ensuring that the crash reports were sent to the groups that requested them. This was also one of our responsibilities.

Product Owner: This role was given to three groups, one from each subproject. The product owners were responsible for keeping track of the user stories that were assigned to their subprojects, and for organizing the sprint planning sessions and sprint ends. We were product owners for the B&D subproject.

Issue Tracker: Responsible for setting up a system on Redmine for keeping track of user stories and for informing group if they do not finish a user story in time or forget to change the status of the user story on Redmine.

Server: Responsible for the server the project used, this included installing software on the server, upgrading the capacity of the server, and making sure the server was running.

Redmine in General: Responsible for keeping track of whether the Redmine website was running and for fixing the site if any problems should occur.

Redmine Wiki: Responsible for keeping the Redmine wiki running and for being the goto group if anyone had problems setting up a wiki page on Redmine.

Redmine Forum: Responsible for keeping the Redmine forum running.

Git: Responsible for the Git repositories used in the project, and for making changes to it if requested.

Webadmin: Responsible for keeping the Giraf website running and for updating information on the site when needed.

Android Guru: This role was given to a person that at the start of the project already had a lot of experience with Android, and worked as a goto guy for anything Android related.

Graphics Guidelines: This group was responsible for making graphical guidelines for the

apps, so that all apps had a consistent look.

Code Style: This group was responsible for making guidelines for how the code should be written and for writing a guide with common code practices.

Customer Representatives: This group was responsible for managing the contact between the project and the customers, so that not all groups send them emails all the time and so that if multiple groups have the same question, the question is only asked once.

1.4 Tools used in the project

In our project we used some tools, to make the whole development process easier and faster. In this section there is a short description of some the the tools used.

1.4.1 Android Studio

Android Studio was an IDE made by Google, for coding apps for android devices [And, 2014]. We used an older version of Android Studio at the start, but already in the first sprintz into the project Android Studio was upgraded to version 1.0.x.

Gradle

To build the project in Android Studio, Gradle was used. Gradle is a project automatic-build tool used by Android Studio, which is designed for use in multiprojects [Gra, 2014].

Artifactory

Artifactory is the maven manager that was used in the project to store and manage versions of the libraries. Because it is able to store multiple versions of every library, it makes it possible to have apps that use an older version of a library and still be able to tell which version is used, and it is also easy to change the version of a library an app is using, by just changing an extension of a line of code in the Gradle build file for the app.

1.4.2 Git

Git was the tool that is used in the project for sharing files internally in the project, primarily code files for the apps. What makes Git good as a file sharing system is that it allows multiple people to code on the same program, and if you commit and another person had committed some changes before you, you can merge the changes between the two versions. Another functionality that Git has that is useful for this project is that it allows the use of “branches”, which allows people to code in a repository that is separate from the original (Master branch).

1.4.3 Jenkins

Jenkins was the tool used for building and testing the apps before release. It was set up so that if anyone pushes changes to the master branch, the app on that branch would be built and then tested by Jenkins. In addition to having tests run every time something is pushed on Git, Jenkins also made an install test and a “monkey test” for all the apps every

night. The install test installs all apps on an emulator to test if any of the apps conflicted when installed. The monkey test starts all the apps one by one and then performs random actions to test the stability of the app. When an app is built and tested successfully the new version of the app is pushed to Google Play as an alpha-version.

1.4.4 Google Play

Google Play is the platform on which all the apps are published. By uploading the apps to Google Play the app become available on the Google Play store where everyone that has an android device can find and download them.

1.4.5 Google Analytics

Google Analytics was used in the project to get crash reports from the apps. When an app crashes a crash report is sent to Google Analytics, where you can go and see them. In addition it is also possible to make Google Analytics send the crash report by mail.

1.4.6 Redmine

Redmine is a web-based project management and issue tracking tool [Red, 2014], and was used as such in the project. Redmine was in the project used to share user stories and for sharing information in general. This information was shared through a forum and a wiki page. The forum was mostly used to arrange social arrangements and other messages where a little wait time was not a problem, and the wiki page was used for sharing guides, guidelines, general information, and summaries from project meetings.

1.4.7 Google Docs

Like the Redmine wiki, Google Docs was used for sharing information, but only for the project backlog and for summaries from meetings. The reason Google Docs was used for these was that it allowed multiple users to edit the same document simultaneously, so that it was not only one person that was forced to write summaries at the meetings, and for the backlog it allowed one screen to display what was being talked about while another could edit the content.

1st Sprint 2

For the first sprint our main user story was setting up and continuously support Google Play and Google Analytics. We also had a user story to write guidelines for Javadocs, and we were given the role as Product Owners for B&D.

2.1 User Stories and Tasks

This section describes the sprint planning for the 1st sprint.

In accordance with normal Scrum practices, we started with a sprint planning session. It was held in collaboration with the other groups for our s-ppubproject B&D, which consisted of group 2, 5, 8, and 9. One of the first things the entire project team, not only B&D but the other subprojects as well, did in the project was to scour through the reports from last year's project and construct an initial backlog to work on for the first sprint. After this, the user stories were divided between the relevant subprojects, and then later some of the user stories were divided between the different groups. On Table 2.1 is a list of the user stories that was assigned to our group in the first sprint.

User Stories	Tasks
Setup for Google Play	Update the description of apps Translate descriptions to English Alpha/Beta upload for Google Play
Central Documentation	Test javadocs Construct guidelines
Google Analytics Crash Reports	Create guidelines for implementing Google Analytics Autoforward crash reports

Table 2.1. User stories and related tasks for sprint 1

The main goal of the 1st sprint was to get familiar with how the overall structure of the project was.

2.2 Google Play and Analytics

In order to setup and support Google Play and Google Analytics, we were given the administrator access to both services. In the following we describe our work with the two services.

2.2.1 Google Play

Google Play is a digital distribution platform created by Google. It is used as an app store for Android devices as well as for music and ebook distribution. Since the goal of the project is to make apps for Android devices, we are using the Google Play to distribute the different apps. [Goo, 2014]

As it was our responsibility to maintain and control Google Play for this semester, we started by familiarizing ourselves with the different features provided by Google Play. We found that Google Play provides a lot of statistics for all the apps, like how many times the apps have been installed and how many that still have them installed. Google Play also allowed us to change which apps should be shown on the store and to change the descriptions of the apps. One thing we noticed when we looked at some of the descriptions was that a lot of them were very short and did not provide enough information about the apps, so the user would have trouble knowing the functionalities that the apps provide. Because of this, we decided to update the descriptions, making an English version of the descriptions in the process, and to update some of the pictures showing the apps to the user.

One kind of data that Google Play provided, which was very important, was the statistic about which version of Android the users were using. This allowed us to know that the apps currently should use the Android API version 15, because 22% of the users were using Android version 4.0.3 - 4.0.4, which only supports API version 15 and below. [API, 2014]

Alpha/Beta

At the conclusion of last year's work on the project, all the apps from the project were uploaded to Google Play. This meant that the apps became available for the users, and that data about crashes would be sent to Google Play and Google Analytics. The initial releases were plagued by crash issues and other errors. Some of these issues were attributed to certain applications crashing when trying to use other apps.

To avoid releasing unstable versions on Google Play, while still being able to test the apps through Google Play, it was decided to add an alpha and beta release version to Google Play. The alpha and beta versions fulfill the same role of ensuring that builds were stable before release, but they are updated differently. The alpha version would be updated whenever a new build is uploaded onto the Jenkins server and it passes the automatic tests to ensure it does not break the build. If it passed all of that, this new build will replace the alpha version available on Google Play. The beta version should be updated at each sprint end to the newest alpha build. There were not any plan for when to update the official release, but it follows that stable beta versions can be released officially.

For the work in this implementation of alpha and beta versions, we found a plug-in for

Jenkins that could be used to make the uploading of alpha and beta versions automatic, as well as providing the information the server needs to upload to Google Play. The plug-in and information was then passed on to the group responsible for the Jenkins server, so they could implement the automated upload.

2.2.2 Google Analytics

Google Analytics is a service created by Google. Google Analytics main function is to track visitors, to show the developer how people are using the product. Mobile App Analytics is a branch of Google Analytics, which focus on mobile apps. Mobile App Analytics has Google Play integration which allows for visitors and traffic sources tracking. This traffic can then be used to see who is installing the apps on which platforms. Mobile App Analytics's API provides for tracking events, crashes, exceptions, and user patterns. [Goo, 2015]

Mobile App Analytics and Google Play have a lot of the same functionality for tracking who installed what on which platforms. However Mobile App Analytics has an advantages when it comes to crash and exception rapports, because the API send a crash report directly to Mobile App Analytics, without the user having to do anything, whereas Google Play does not include crashes that are not reported by users. For this reason Mobile App Analytics were used as the main crash report service.

For crash reporting there were three tasks at hand. The first was to automatically forward the crash reports from an app to the groups that worked on that app. The second and third task were to provide a guide on how to implement the API in the applications and to provide a guide for the next years Google Analytics group on how to autoforward crash reports.

Autoforward crash reports

Mobile App Analytics has a built-in email feature, which allows the user to automatically email a crash report daily, weekly, monthly, or quarterly. It is not intuitive how to autoforward crash reports because the user has to first make a report that is able to fit to the time interval the user want. Therefore, a guide was made which can be found in Appendix B.2.

Mobile App Analytics API

In order to get crash reports for an app sent to Google Analytics, the Crash API for Mobile App Analytics has to be implemented in the app. The API makes it possible to make a Tracker which can be used to report information to Mobile App Analytics, when an exception is caught. A great feature about the tracker is that it can be set to report uncaught exceptions. A guide how to implement Mobile App Analytics API in an app can be found in Appendix B.1.

2.3 Central Documentation

This section goes over our central documentation user story, where we work on implementing a code comment generator to make code documentation easier.

As a part of the central documentation user story, we looked into how we could standardize documentation of the code for the project by using Javadoc to provide a clearer overall form of documentation.

Javadoc is a plugin that automatically generates comments for coding blocks. In essence, it generates a comment stub automatically for a code block with a blank field space for all parameters. [Jav, 2015]

The comments can then be automatically gathered and put in order into a html file which partly acts as code documentation.

```
1  /**
2   * Short one line description.                (1)
3   * <p>
4   * Longer description. If there were any, it would be [2]
5   * here.
6   * <p>
7   * And even more explanations to follow in consecutive
8   * paragraphs separated by HTML paragraph breaks.
9   *
10  * @param variable Description text text text.      (3)
11  * @return Description text text text.
12  */
13  public int methodName (...) {
14  // method body with a return statement
15  }
```

Figure 2.1. An example listing of how a javadoc should look. Note the double star at the beginning of the comment.

The motivation for using a Javadoc-like system was to improve the overall comment approach through standardization and ease of use. An added bonus is that it can provide code documentation on the project as a whole, which is a challenge under normal circumstances.

At the start of the project, Javadoc was partially fulfilling its role. It was used in some projects consistently, while in others, comments of any kind were missing.

Our responsibility was twofold, we had to check whether Javadoc were usable across the entire code base and if it were not, then we should make sure it got fully implemented. The other responsibility followed the first. Once it was ensured that Javadoc was properly installed, a guide to Javadoc was distributed to the rest of the project groups and placed on the wiki. After some light testing, we could conclude that Javadoc were fully deployed.

In the same sprint, another group decided to replace Javadoc with another plugin called Doxygen for code documentation purposes. Overall Doxygen serves the same purposes as Javadoc did. Doxygen accepts the same syntax as Javadoc, so migration was smooth.

2.4 Summary

This section is a short description of what we did in the 1st sprint, and which user stories that followed over to the next sprint.

Google Play and Analytics

As it was our responsibility to handle everything related to Google Play and Analytics we first familiarized ourselves with them. For Google Analytics we wrote a guide on how to implement the code to get crash reports on Google Analytics, and made it so that Google Analytics auto forwarded crash reports to all groups that were interested in getting them. For Google Play we updated all the descriptions and made a english translation for the apps, and in cooperation with group 9 we made it so that when an app is build on the Jenkins server the app is released as an alpha version on Google Play.

Central documentation

For code documentation we first did some research to get to know how to use it. After that we then wrote a guide on how to implement the code documentation in the individual apps, which was shared on the Redmine wiki.

For the first sprint we completed all the user stories that were assigned to us, but as Google Play, Google Analytics, and Product owner was our roles, we continued working with them in the 2nd sprint.

2nd Sprint 3

For the second sprint our focus was on Google Play and in looking into making the apps work standalone. This included looking for dependencies.

3.1 User Stories and Tasks

This section describes the sprint planning for the second sprint.

After the end of the first sprint we started sprint planning anew. As the product owners we were responsible for the sprint planning for the B&D subproject. In Table 3.1 the user stories and associated tasks we found for our 2nd sprint for our group can be seen.

User Stories	Tasks
Google Analytics	Provide support
Update Google Play for sprint # 1	Collect APK's and changelogs Sign APK's Upload and update Google Play description
Update Google Play for sprint # 2	Collect APK's and changelogs Sign APK's Upload and update Google Play description
Standalone App	Find dependencies Check the Local database Check GUI design Check how synchronization with database work Ensure consistency between original and Standalone version
Core Program	Isolate dependencies Design and make the core package Publish dependency report

Table 3.1. User stories and related tasks for sprint 2

The main goal for the 2nd sprint was to reform the structure of the project with the goal of improving the build time of apps, looking into making the apps standalone, and improving the structure of the git repositories.

3.2 Dependencies

In this section all the dependencies between apps and libraries are described, with the goal of finding out what is needed to get the apps to run, and which dependencies would have to be removed to be able to make apps standalone.

3.2.1 Motivation

Finding the dependencies for apps and libraries, would help in understanding which dependencies would have to be removed before an app is independent. It would also help the Jenkins group in making tests on apps, since the knowledge of which apps an app is dependent on would help them in knowing which apps would have to be installed before an app can be tested. Lastly, having an overview of the dependencies would make it easier for groups working with apps and libraries, by informing them about which apps and libraries they can be affected by and which they have an effect on.

3.2.2 Applications and libraries

The first thing we did to find the dependencies was to make a list of all the apps and libraries that are used in the project. These apps can be found in Table 3.2.

Apps	Libraries
launcher	giraf-component
sequence	dblib
timer	local-db
pictocreator	pictogram-lib
categorymanager	sequence-viewer
voicegame	category-lib
categorygame	TimerLib
administration	DrawLib
lifestory	Ambilwarna
ugeplan	barcodescanner
pictosearch	
pictoreader	

Table 3.2. List of apps and libraries

3.2.3 Libraries apps are dependent on

After making the list of all the apps and libraries, we found the dependencies for all the apps by opening the project for them in Android Studio, and then look at which libraries were included in the Gradle build files.

From this information we made a list of all the libraries the apps are dependent on. This list can be seen in Table 3.3.

Apps	Dependencies
launcher	→ giraf-component, dblib, local-db, barcodescanner
sequence	→ giraf-component, dblib, local-db, pictogram-lib, sequence-viewer
timer	→ giraf-component, dblib, local-db, pictogram-lib, DrawLib, TimerLib, wheel
pictocreator	→ giraf-component, dblib, local-db, pictogram-lib
categorymanager	→ giraf-component, dblib, local-db, pictogram-lib, catagory-lib, Ambilwarna
voicegame	→ giraf-component, dblib, local-db
categorygame	→ giraf-component, dblib, local-db, pictogram-lib
administration	→ giraf-component, dblib, local-db
lifestory	→ giraf-component, dblib, local-db, pictogram-lib, sequence-viewer
ugeplan ¹	→ giraf-component, dblib, local-db, pictogram-lib
piktosearch	→ giraf-component, dblib, local-db
pictoreader	→ giraf-component, dblib, local-db, pictogram-lib, catagory-lib, Ambilwarna

Table 3.3. List of library dependencies for apps

As can be seen in the list of dependencies, all the apps were dependent on ‘giraf-component’, ‘dblib’, and ‘local-db’. In addition there are a lot of the apps that are dependent on ‘pictogram-lib’.

To get a better overview of the dependencies we made a graph, see Figure 6.1. On the graph the squares represent apps and the ovals libraries. An arrow from A to B signifies that A is dependent on B. Since there are a lot of dependencies between the apps and libraries, we decided to exclude ‘giraf-component’, ‘dblib’, and ‘local-db’ from the graph. This meant that the graph was a lot easier to read, and because all apps were dependent on those libraries, we just added a short note about the dependency.

¹Opens through lifestory

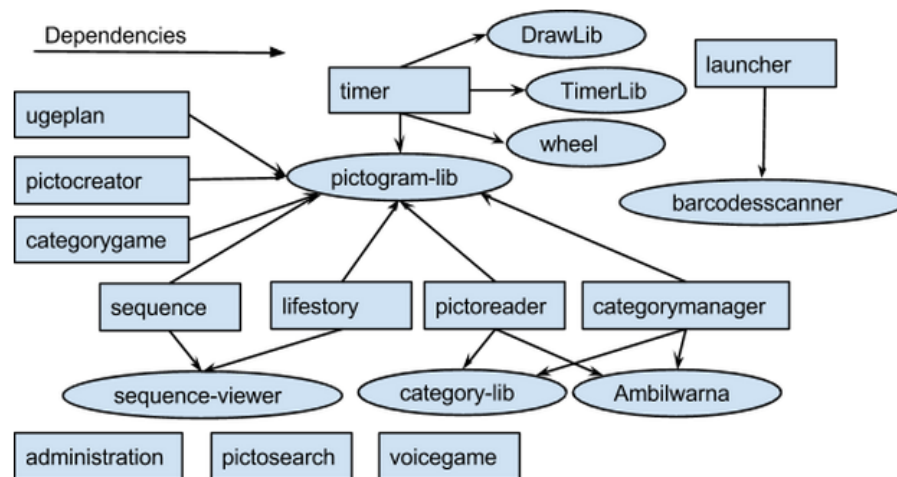


Figure 3.1. Graph of library dependencies for apps. In addition all apps are dependent on dblib and local-db.

3.2.4 Cross library dependency

After having made all the library dependencies for the apps we did the same procedure for the libraries to find out which libraries were dependent on which library. As with the apps we started by looking at the Gradle build files for the libraries, and then made a list of the dependencies, which can be seen in Table 3.4.

Libraries	Dependencies
giraf-component	→ dblib, local-db
dblib	→ local-db
pictogram-lib	→ giraf-component, dblib, local-db
sequence-viewer	→ giraf-component, dblib, local-db, pictogram-lib
local-db	→ dbmetadata
category-lib	→ giraf-component, dblib, local-db, pictogram-lib
Drawlib	→ NULL
TimerLib	→ NULL
Ambilwarna	→ NULL
barcodescanner	→ NULL
wheel	→ NULL
dbmetadata	→ NULL

Table 3.4. List of cross library dependencies

As with apps/library dependencies we also made a graph for cross dependency between the libraries shown in Figure 3.2. The arrows signifies dependencies in the same way as

the previous graph. In this graph we decided to show all dependencies since there was no dependency that was the same for all libraries and because the number of dependencies was low enough to be able to show it all without having too many lines crossing each other to be able to read the graph.

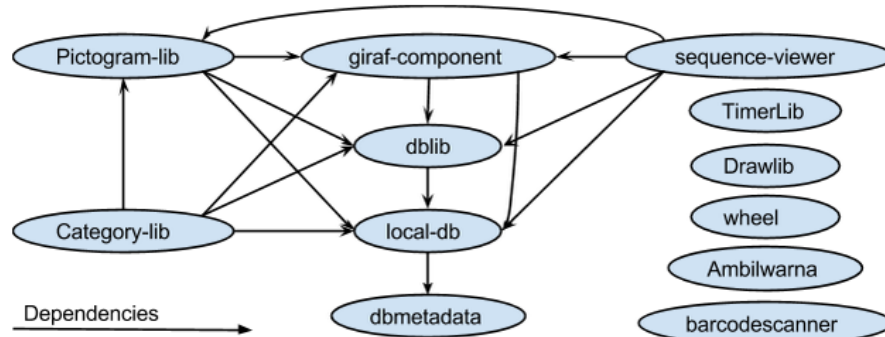


Figure 3.2. Graph of cross library dependencies

3.2.5 Cross app dependencies

As the last step in finding the different dependencies we found the dependencies between the different apps. We did this in two ways, skimming through the code for any calls to other apps, and by trying to run the apps on a tablet and testing if any of the apps stopped working if another app was not installed. From the dependencies we made a graph which is shown in Figure 3.3. The arrows signify dependencies, but in addition this graph also have dotted and red lines. The dotted lines signifies that the app is not directly dependent, but is instead dependent through another app.

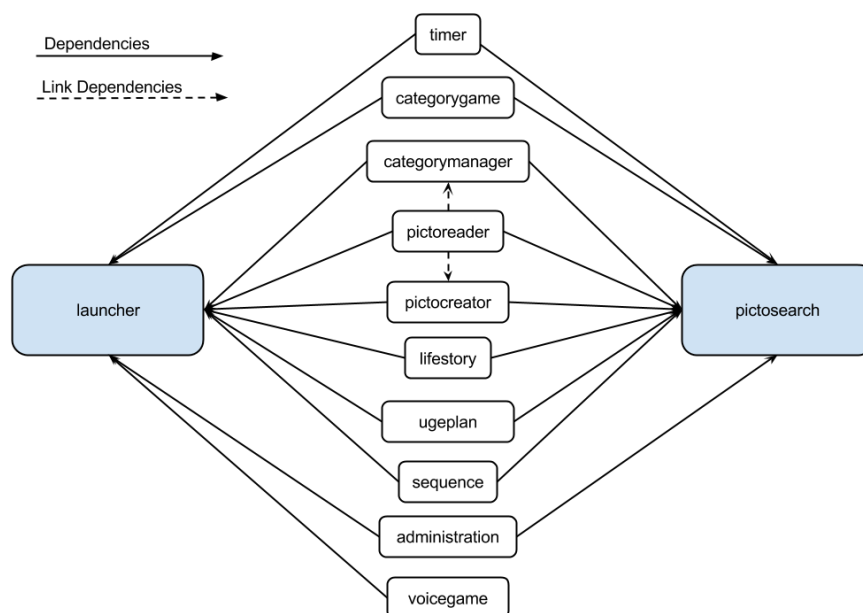


Figure 3.3. Graph of cross app dependencies

3.3 Standalone

The customer showed some interest in having some of the apps available as standalone. This section focuses on our work in this area covering which features would be needed to make the existing apps standalone, and our decision to make a core library.

3.3.1 Motivation

Making the apps standalone would increase the usability of the individual apps, as the user would no longer need to install several different apps just to ensure that they can use the app they want. As seen in Section 3.2 all apps in Giraf are reliant on both the launcher and ‘pictosearch’. This dependency was not only annoying for the user but was also causing trouble during the automatic tests, as one of the tests include installing the apps and testing them pseudo-randomly, but the emulation used for this test could not at the time install more than one app at a time, which resulted in this test always failing on apps reliant on the launcher.

3.3.2 Features

In order to make the individual existing apps standalone, we first looked at each app and found out which tasks the apps had to be able to perform when working standalone. In the following section we describe which features that would enable an app to be standalone.

Sequence

Sequence is an app that allow the user to make sequences of pictograms for use in communication. In order to use the app, it needs access to a database with pictograms, and if the database has not been made by the launcher, sequence has to be able to make a database and download the pictograms to the database itself. In order to allow the user to use personal pictograms, the app also need a login feature, so that the user can access these pictograms as well.

Other apps

Like sequence many of the other apps that need to be standalone, need access to pictograms as well as personal pictograms. Therefore, we will not go into details with these apps.

Overall necessities

So from this we have arrived to the following list of features the apps would need to work as standalone apps.

Login: This features will allow the user to access personal pictograms and save pictograms to a personal account.

Make Localdb: This feature has to check if a local database of pictogram has already been made by another app. If no local database has been found the app will then need to make it itself.

Download pictogram: This feature has to download pictograms for the local database from

the server. It also has to get personal pictograms for the user, if the user is logged in to a personal account.

3.3.3 Standalone app or library

As all the apps need the features just mentioned, it would make sense to place them in a common place, either as an app or as a library. That way multiple apps could use the same code without having to rewrite the code in the individual apps. Additionally any new common features needed could be added to this app or library. In general this will be referred to as the core app or core library. In Table 3.5 an overview is shown of the positive and negative aspects (pros and cons) of each solution. The two solutions were evaluated on how easy they would be for the user. This meant that the app-solution would be a bad solution because it would require the user to install another app in addition to the standalone app they wanted to use. The bigger size of the apps in the library-solution is regarded as a minor problem as currently the storage space for the pictograms used far outweighs the storage used by the installed apps and Giraf in its entirety does not use a lot of space to start with.

Core standalone App	Core standalone Library
Pros	Pros
Smaller in size when more than 1 standalone app is installed	True standalone (no other apps needed)
Cons	Cons
Another app needs to be installed	Bigger app size
Other notes	Other notes
Fuse core app and pictosearch (only one service app)	Make pictosearch a part of the core library (no service app)

Table 3.5. Pros and cons of a core standalone app and a core standalone library

3.4 Build structure

This section describes the change made to the build structure of the apps and libraries in the project, in order to get a lower build time when building apps and libraries.

3.4.1 Motivation

When the 2nd sprint started, the apps and libraries recursively downloaded and built all the libraries they were dependent on. This resulted in large repositories, since many libraries were included multiple times in one project. The graph in Figure 3.4 shows the build tree for the sequence app. The darker circles are all local-db. As seen from the graph the library local-db is built 16 times before the sequence app itself is built. All in all the entire project for sequence had to build 31 libraries before it could build the app. Which took a lot of time on the build server.

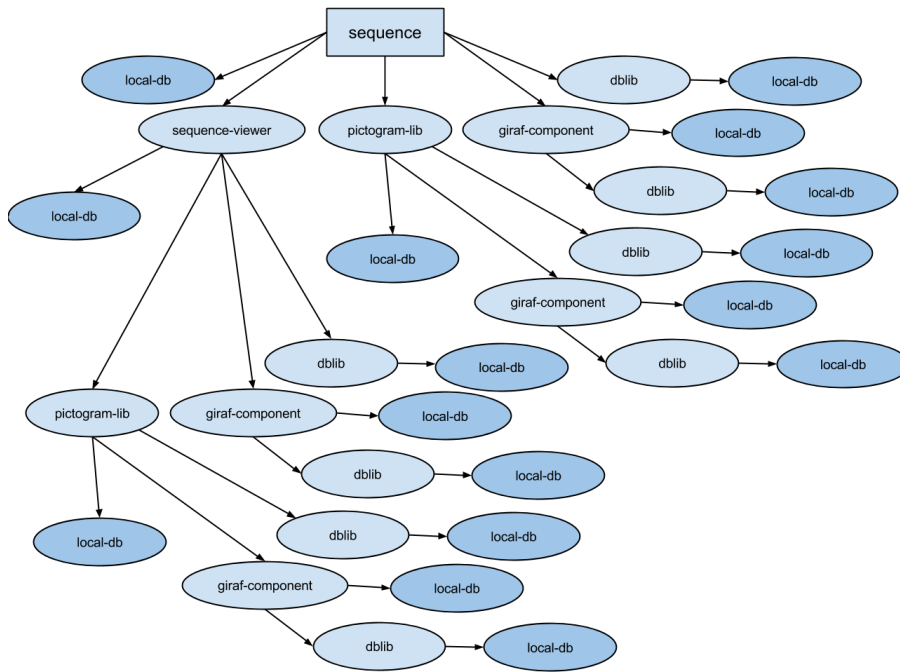


Figure 3.4. Build tree for sequence built recursively

3.4.2 Android archive

As a solution, we wanted each library to build a library file, which could be used in multiple places. This would reduce the build time of all the projects since the apps or libraries would only need to download the pre-built library files. This is where Android ARchive (AAR) files are introduced. AAR files are binary distributions of an Android Library Project, which works like a standard Java Archive (JAR) library. The difference between a JAR library and an AAR library is that the AAR library includes resources, such as images, which allows for visual components like a login screen, which can then be used across multiple apps.

Since more groups were involved in the reformation of the build structure, we refer to Section 7.1 for the collaboration aspect.

3.4.3 Solution

By using a pre-built library, each app only has to build itself with the libraries included in the app as shown in the graph in Figure 3.5. It is important to know that all libraries included in other libraries have to be included by the app. It means that even if sequence did not use dblib directly, it would still have to include it because ‘giraf-component’ is using it. This can be accomplished by using any tree traversal algorithm that returns all unique libraries in the tree from Figure 3.4.

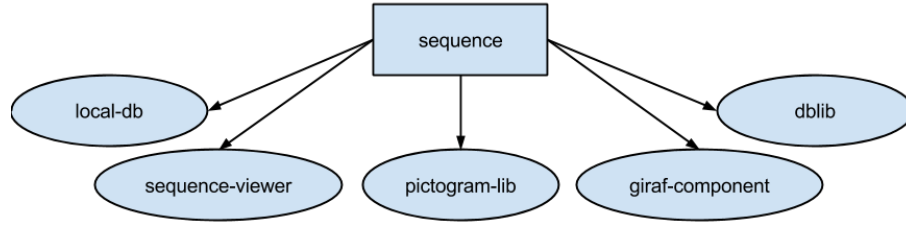


Figure 3.5. Build tree for sequence built using AAR files

Before implementing this solution, libraries could be included once for every other library previously included in the app, and once for the app itself. This meant that if i.e sequence added the library sequence-viewer, then pictogram-lib, then giraf-component, then dblib, and then lastly local-db; then sequence-viewer would be added once, pictogram-lib twice, giraf-component four times, oasis-lib eight times, and 'local-db sixteen times. This followed the arithmetic series:

$$1lib_1 + 2lib_2 + 4lib_3 + 8lib_4 + 16lib_5 \dots 2^{n-1}lib_n$$

This can be summarized to:

$$\sum_{i=0}^{i < n} 2^i$$

Where n is the number of different libraries added to the app.

This was the worst case scenario but as it can be seen from Section 3.2 most apps were in a worst case state before the change. After implementing the solution, the total number of library-builds were reduced to one for every different library. This meant that after the change there was a linear growth between the number of libraries and the number of library-builds.

As it can be seen from Table 3.6, our solution reduced the number of libraries being built drastically, which in turn reduced the build time for each app on Jenkins by a detectable margin.

	Builds Before	Builds After
Launcher:	8 (barcodescanner is only built once)	4
Sequence:	31	5
Timer:	18 (DrawLib, TimerLib, and wheel are only built once each)	7
Pictocreator:	15	4
Categorymanager:	17 (category-lib and Ambil-warna are only built once each)	6
Pictosearch:	7	3
Voicegame:	7	3
Categorygame:	15	4
Pictoreader:	17 (category-lib and Ambil-warna are only built once each)	6
Lifestory:	31	5
Ugeplan:	15	4
Administration:	7	3

Table 3.6. Number of builds before and after

3.5 Summary

This section is a short description of what we did in the 2nd sprint, and which user stories that followed over to the next sprint.

Dependency: We were given the task of finding dependencies for the apps and libraries, for this we found all the dependencies and listed them. To improve the overview of the dependencies we made a graph for app-library, library-library, and app-app dependencies. These graphs were shared with the rest of the project on a Redmine wiki page.

Standalone: In the user story to make app standalone we started by finding out what functionality the apps would need to be able to function on their own. For this we found that we would need to make a library that included these functionalities, which is then included in all the apps.

Build structure: When building apps in the project the build time was very long because all libraries were built recursively. To improve this we incorporated the use of AAR files in the build process. This lead to a change in the number of builds from an exponential number of builds, to a linear number of builds.

For the 2nd sprint we did not complete all the tasks that were assigned to us. The tasks we did not complete were for the standalone app and the making of a core library. These tasks were not finished because they proved to have a lot more work in them than first estimated. We continued working with the core library in the 3rd sprint, but postponed the standalone app since the app would require the core library before any more work could be done.

3rd Sprint 4

The main goal of the third sprint for B&D was to handle the renaming of most apps and some of the libraries. Our group also had to ensure that the various guides used in the project was available on Redmine in a readable format.

4.1 User Stories and Tasks

This section describes the sprint planning for the 3rd sprint.

After the end of the 2nd sprint we started sprint planning anew. As the project owners we were responsible for proper sprint planning for the B&D subproject. The user stories and associated task we found for our second sprint for our group can be seen in Table 4.1.

User Stories	Tasks
Google Analytics	Provide support
Ensure App Name Consistency	Find internal names for apps and libraries Inform the groups when the change is made Make Git and Jenkins rename Rename on Google Play and Google Analytics Update pictures for all Google Play apps
Make pictosearch a Library	Transform pictosearch to a library Modify the code, so that it works on one of the apps If the process is simple, do the same for all the other apps
Code-documentation for ARR files	Find out if it is doable and how Make a guide
Share the knowledge from Guides	Check for missing guides Check the content of the guides and update them Move guides from PDF format to the wiki page Inform groups about which guides exists
Release backlog	Make release backlog for sprint 1, 2, and 3
Backlog	Make a description of items in backlog Insert the backlog into the semester backlog
Core library	Generate a library project Implement the library in an app Implement the library for the remaining apps

Table 4.1. User stories and related tasks for sprint 3

The main goal of this sprint was to handle the renaming of most apps and some of the libraries, as well as ensuring that the various guides used in the project was available on Redmine in a readable format. However many of the tasks proved to be easier than estimated and were finished early, so we added the core library to our sprint backlog.

4.2 Renaming of Package-names

One of our user stories for the 3rd sprint involved the internal renaming of most apps and libraries. In this section we will cover the renaming process including the consequences of the method used.

4.2.1 Motivation

As there was little to no consistency in what the apps were called internally, that is on Git, Jenkins, and the package-names on Google Play, it was highly requested that some consistent names were made. In the project the package-names are on the form: dk.aau.cs.giraf.App, where ‘App’ is replaced by the internal name for the app in question. While in the process of renaming, it was also suggested that the apps got names that relates to what they do, i.e. not many would know what the app ‘croc’ does but when it is named ‘pictocreator’ it makes sense. Since of the descriptive names and the consistency in naming, it would become much easier for new developers to understand what the different apps are.

4.2.2 Initial decisions

Before we started renaming anything, we took a look at the old names, seen in Table 4.2. The names were a mixture of animal references and shorthand notations for parts of the app’s content. The animal names were originally created several years ago and while they might have made sense back then, it was decided to find new names for these. Additionally some of the shorthand notations were changed as well, to become more descriptive.

New names were suggested by us and after some discussion with members of the GUI-subproject, we settled on the names seen in the second column of Table 4.2. Everyone involved in the decisionmaking on the new names, feel that the new names make more sense in the way that they actually say something about the functionalities of the app.

Old app package-names	New app package-names
launcher	launcher
train	categorygame
pictoadmin	categorymanager
tortoise	lifestory
oasis	administration
parrot	pictoreader
pictosearch	pictosearch
pictocreator	pictocreator
zebra	sequence
cars	voicegame
wombat	timer
schedulestarter	ugeplan

Table 4.2. Old and new package-names for the apps.

4.2.3 Renaming process

The most important part of the project to have renamed, was the package-names used by Google Play, as due to Jenkins automatically uploading new versions of the app to Google Play, Jenkins would fail if the package-name was different from the one used on Google Play. Since package-names are unique Google have decided that they should not be changeable after uploading the app, therefore we were subsequently forced to bypass this system by uploading new apps, with the same content, to replace the apps, where we changed the package-name [Pac, 2015]. To easier distinguish the new version from the old, we decided to add the word "retired" as an extension to the name of the old app. This was done to ensure that the other groups could continue working on the apps without being affected by the name change before they were ready to transition to the new name.

During this process we encountered a problem with one app not being uploadable. This was caused by some of the libraries having part of the same package-name, more specifically they were called 'dk.aau.cs.giraf.oasis.LibName', while one of the apps had previously been called 'dk.aau.cs.giraf.oasis'. Due to this problem we decided to rename the package-names of the three libraries causing the problem, in part also to get rid of the 'oasis' part of the name, as it did not tell anything about what the library was used for. Due to some unforeseen events in another group working on one of the libraries in question, we had to wait a few days before the renaming could be continued and finalized. The new and old package-names for the libraries can be seen in Table 4.3.

Old library package-names	New library package-names
oasis.metadata	dbmetadata
oasis.localdb	localdb
oasis.lib	dblib

Table 4.3. Old and new package-names for the libraries.

4.2.4 Consequences

The way the renaming was done was not completely without consequences. As we had to upload essentially new apps, it was not possible to transfer the statistics from the old apps to the new ones. In addition all users at the time would have to uninstall their current versions of the apps and then install the new releases in order to get updates on the apps in the future. Both of these consequences, while potentially very bad if the project had been more established, is not actually too bad. This is because the number of external users are minimal and we have direct contact with them, and the data and statistics gathered from the usage of the apps is not that useful in practice due to the rapid development on most apps. Therefore we see the renaming as an overall success.

4.3 Improvement of the wiki pages

In this section is a description of how we in the 3rd sprint worked on improving the quality of the guides and information pages on the project's Redmine wiki pages.

4.3.1 Motivation

Improving the quality of the information shared on the Redmine wiki, would improve the workflow by shortening the time a group spend on understanding the tools used in the project, by providing useful guides and guidelines.

4.3.2 Process

As the first step we looked through the whole wiki and listed all the guides and pages that provided information. We then looked through the list and discussed in the group if there were any guides that were not on the list that needed to be included. In this discussion we did not find any extra guides that we thought were needed on the wiki page. All the guides and information pages we found are shown on the list below:

- Git guide
- Android guide and guidelines
- Testing
- Issue Guidelines
- Google Analytics
- Javadocs
- Dependencies Management
- Guide for renaming Apps and libraries in android studio
- Continuous Integration Guidelines
- App and library renaming in Android Studio
- How to implement the pictosearch library
- Development method
- Repository names
- Dependencies between apps and libraries
- Database architecture
- Requirements Specification and user stories for Giraf
- Requirements for Database Component
- Links
- Meetings
- Backlog sprint #4 last year projects
- Product backlog
- A list of groups and their responsibilities
- Word list, for terminology when talking about the institutes.

Quality check

We looked through all the guides again, but this time we checked if the content and the formatting of the page were correct and of an acceptable level. For the pages that did not provide good enough content, but where we had the knowledge ourselves to correct it, we corrected them ourselves, and for the one we did not have the knowledge to fix, we contacted the groups that had written them in the first place. For content there were only two instances of pages that did not live up to the standard, the first was “Dependencies management”, where information about how to log in to the maven repository was missing, and the second was the Jenkins guide, which only had titles and no actual content. The login information we inserted to the page ourselves, but for the Jenkins page we had to contact the group responsible for Jenkins in the project. It turned out that the wiki page was not supposed to have been made in the first place, so it was decided that the page got removed, instead of writing the whole guide, because only the Jenkins group would have use of the information, and they already had the knowledge themselves.

Formating

A lot of the pages did not have the correct formatting or did not have a format at all. For those pages we corrected the format so that the pages was easier to read. In addition to the formatting, some of the pages also had most of their content in a PDF file. For those pages we wrote the content of the PDF file into the page and then placed a link at the top of the page where the PDF could be downloaded.

Titles

For some of the pages we also changed the title so that it was easier to understand what content was on the page without having to open the page. After having fixed all the guides and information pages on the wiki we sent an email to all groups informing them that we had fixed all the guides and a list of which guides were available at the moment.

4.4 Core library

This section goes over the core library user story. Where we found solutions to the three features needed in order to make to apps standalone.

4.4.1 Pictosearch

As part of the core library collaboration, our user story was to make pictosearch a part of the core library. Pictosearch should be implemented in a way which would make it easy to update the code with code from the pictosearch app, because a group were still working on it. This was done by using the code from the pictosearch app, to make an new activity which could be started from any other activity. This would ensure an easy implementation for the apps that used the old pictosearch app. Since the code base is exactly the same as for the pictosearch app, the Core library can easily be updated by copying the pictosearch app code into the Core library.

4.4.2 Download Pictograms

An essential part of the Core library is the ability to download pictograms from a server database to a local database, so that pictosearch has some pictograms to search through without having to be connected to the internet. The first idea was to reuse the download functionality implemented in the launcher, which both creates the the local database and downloads the pictograms. It turned out that the part of downloading pictograms was trivial but there were a big problem when it came to making a local database, which could be managed by multiple apps.

Launcher

When the launcher first creates a local database, it has ownership over that database. This means that other apps can not access the database, and when the launcher is uninstalled that database is deleted. To allow other apps to get data from the database, the launcher has what is called a content provider. The content provider can then access the database for the other apps if these apps have permission, which were set when the content provider was declared.

Solution

The optimal solution would be to have one local database which could be accessed by multiple apps, instead of having every single Giraf app make its own database, which would take up a lot of space on a smartphone or tablet where multiple Giraf apps were installed.

The major problem with having a shared database is that only one app can have ownership over that database and that the database is deleted when that app is uninstalled. The simple solution to this problem would be to make the core library a service app like pictosearch was before it became a library, this way the core library could have a content provider which could be accessed by all the apps that needed pictograms, and then the content provider could be removed from the launcher, since the launcher does not use pictograms anyway and then there would only be one place to maintain access to the local database. This solution would solve all the problems there were with the database. The only problem is that it would give the same problems as pictosearch had before it became a library. Apps would then never be truly standalone, and if the core library is not installed none of the apps using pictograms would work. Though this is an easy solution for the developers, it is not a smart or easy solution for the users.

The idea of having a shared app which could access the database is not a good solution, but having a shared database is still the goal. Since Android has no way of implementing a shared database, it would be of interest to look at alternative data storage options. The other data storage options Android has is file storage. With file storage an application can create, update, and delete files on internal memory, tablet memory, and/or external memory, SD card. Unlike the database, file storage allows an app to store data in a public location which would allow other apps to access that data. And when apps are uninstalled, the data is not deleted. This solves the problem with data sharing. But using this solution also means a loss in functionality that the database has, like the ACID properties, which is

a set of properties to guarantee that the database is reliable. The loss of these properties opens up for a whole new set of problems when it comes to data management. One of these challenges was to find out, if there was any way to secure data when concurrent access to the data is allowed. These are just some of the problems that could occur when going from database to data files. These problems will not be described or solved here since these are lengthy and complex problems, that would involve both DB groups and any groups working on the apps.

It does not make sense to make a login feature before the download feature was implemented, therefore there were no further development on it or the rest of the core library.

4.5 Summary

This section is a short description of what we did in the 3rd sprint, and which user stories that followed over to the next sprint.

Renaming of app and library-package-names: As the naming of apps and libraries were inconsistent, it was decided that they should be made consistent. We were the group responsible for ensuring this consistency overall, and we had to rename the package-names in specific, so the apps could be uploaded with the new package-names to Google Play. To accomplish this we made a detailed guide on how to do the renaming. The renaming was at the end of the sprint finished for the most part but some names still needed to be changed on the Git-repositories.

Guide consistency: We reformatted and to a minor extent expanded or rewrote the guides used in the project, so they were more readable. This included taking several guides in pdf-format and writing them on the Redmine Wiki.

Core library: We presented some solutions to the problems at hand, and why we would not implement them ourselves, because it would create conflicts with the work done in the DB subproject. The solutions are further discussed in Section 9.3.

We completed most of the user stories assigned to us in the 3rd sprint, but some of the renaming was left for the 4th sprint so it would not interfere with the sprint end of the 3rd sprint. Providing code documentation for AAR files was not possible, because it was not supported by Android Studio. The work with the backlog is covered in Section 6.3. We also continued our work with Google Play, Google Analytics, and as Product owners.

4th Sprint 5

For the 4th sprint our main focus was to make it easier for the next year's students to take over the project

5.1 User Stories and Tasks

This section describes the sprint planning for the fourth sprint.

As product owners for the B&D subproject we facilitated the sprint planning for the fourth sprint for the B&D groups. In Table 5.1 the user stories for our 4th sprint can be seen.

User Stories	Tasks
Ensure App Name Consistency	Inform all relevant groups, when an app/library is renamed Find out what is wrong with ugeplan and fix it
Make pictosearch a Library	Transform pictosearch into a library Inform all groups when done and link to guide
Bundle apps together	Find out if it is possible via Google Play If not, make all apps into libraries Implement the libraries in the launcher
Preparation for the next semester	Make guide for using Google Play Ensure we have the guides needed for the next semester (B&D) Combine all the guides and information from B&D in one place
Update Backlog	Make the backlog accommodate Scrum process
Survey on Google Play	Fill out the survey

Table 5.1. User stories and related tasks for sprint 4

The main goal of the 4th sprint was to make the project ready for being taken over by other groups next semester. There are some user stories that have bled over from sprint 3. These user stories were virtually completed but still requires some attention from us.

5.2 Package Solution

The customers requested a simple and easy way to install and run every app in Giraf. This section will cover our approach to the problem, and we present a possible solution.

5.2.1 Motivation

As the target audience are not technically savvy, we should make the download and installation process for the apps as simple and intuitive as possible. One thing that the customers specifically requested was the ability to download and install the entire Giraf-system by just downloading one app. This would also make it easier to use the system for people new to the system as they would not have to download several separate apps to use it.

5.2.2 Approach

The first thing we did was to research whether Google Play had any incorporated methods to combine apps or at least download and install multiple apps from the same Google Play store-page. From our research we found that Google have not made anything that can accomplish this.

As Google Play does not have the functionality to do what we need, we have to combine the apps ourselves. We present two approaches, one would be to code a single app, which would then have the functionality of all current and possible future apps. The other approach would be to modify the current launcher and add the other apps as libraries in a similar manner as to how pictosearch is used. The first approach would make it much harder to split up the work to multiple groups, which would go against the Scrum process used in the project. Using the second approach would solve this problem and as such that would be our approach to the problem.

During a semester meeting with the other groups, it was decided that making a package solution where every app were combined to one, was not a high priority, and it would take more time then there were left for this sprint. The consequence and approaches to having one complete app which include all the apps, will be taken up in Section 9.4.

5.3 Preparations for next semester

For our role as product owner and our work with guide consistency, we took the task of preparing guides from B&D for the next semester.

To make it easier for next year's semester to find the guides written in this semester, a document containing a list of guides, descriptions of said guides, and a reference to the location of those guides were made. The list can be found in Appendix B. Some of the guides on the list were made by our group. Those guides can be found in the same appendix in the sections after the list.

In addition the groups responsible for Jenkins and for the server were tasked with writing relevant guides for their work with those services, so it would be easier to take over their work next semester.

The guides we were directly involved in writing and improving are:

- Google Analytics Guide
- Google Analytics Crash Reports to Email Guide
- How to use Google Play
- Android Studio Setup
- App and library renaming in Android Studio
- How to implement the Pictosearch library
- Javadocs
- Dependencies Overview

5.4 Summary

This section is a short description of what we did in the 4th sprint, and which user stories that followed over to the next sprint.

App name consistency: We managed to make the app names consistent and understandable for all apps except for one, with was the “weekschedule” which instead got the name “ugeplan”, as by the time it could be renamed properly, the sprint was over.

Guides: For the sprint we made a guide for the groups next semester. This included a guide for the use of Google Play and a Starter guide for how to install and run the basic tools in the project. In addition to making these guides we also made a document that described and referenced all the guides from the groups in the B&D subproject.

Pictosearch to library: In the sprint we made the final changes in making pictoseach into its own library, since the core library was discontinued and made it so that all the apps used the library instead of the service app version of pictosearch.

App bundle: We found that Google Play do not support the use of app packages. Because of this it would only be possible to make one place where the user can get all the apps in one place by collecting all the apps into one, but this was decided to be postponed to next semester.

For the last sprint we finished all our user stories.

Roles 6

This chapter consist of descriptions of the different roles we had during the semester.

6.1 Google Play

This section will discuss our role as responsible for Google Play.

As the group responsible for Google Play, we have maintained the store page for all the apps in Giraf. This meant that we made updates to the descriptions of the apps, ensured the right versions were promoted from alpha-versions, to the production-versions, that everyone visiting Google Play can see.

Our daily work with Google Play consisted mostly of being available if other groups needed our help with anything related to Google Play. In the 3rd sprint we did have a minor, urgent task, as Google had made new rules regarding the content-classification, which required us to fill out their new forms, otherwise Google held the right to remove the apps from Google Play.

Most other work have been minor tasks of providing the Jenkins group with information about the current version codes of the apps, so they could auto-upload the alpha-versions.

As described in Section 4.2 we also uploaded essential new apps because of the way the renaming had to be done.

6.2 Google Analytics

In this section we describe the work we did regarding the role as those responsible for Google Analytics. Most of our work in the role of Google Analytics is described in Section 2.2.2.

The implementation of Google Analytics was sporadic and only a few applications had it implemented. Rather than implementing it in all apps, we offered to set it up for any groups interested in getting crash reports. However interest was low and by the end of the first sprint, the advent of continuous integration testing on Jenkins effectively ended development using Google Analytics.

6.3 Product Owner

This section will look into our group's function as the product owners for the semester and how we managed the Backlog.

As more oversight was required than initially expected, our group accepted the role as product owners for the B&D subproject group at the end of the 1st sprint. Our tasks were to manage User stories and manage the transition between sprints.

At the same time as the product owners were made, a hierarchy of who the customers were for each subproject. As the product owners of the B&D subproject, we had the GUI and DB subprojects as our customers.

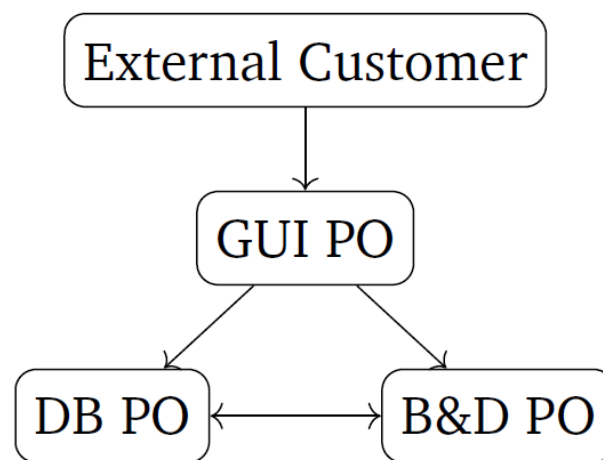


Figure 6.1. Overview of who is customers for who

As product owners, we began to chair the sprint planning for our subproject. When sprint planning representatives from the entire project went through every user story in the backlog, and we had the relevant subprojects prioritize the user stories.

After all the user stories were prioritized, the B&D subproject would by ourselves divide the most important user stories in between us. Given our rather small focus area split between Git, Jenkins, the server, and Google services, most user stories were bound to those roles.

When we were done assigning user stories, each group left to begin work or estimate, depending on their work ethics, after which they could contact us to renegotiate their user stories.

While the sprint was in operation, there was little to do as product owners. We were expected to keep the backlog updated with new user stories received either from other subprojects or from our own. Other groups were expected to come to us with user stories, that were related to the B&D subproject..

Otherwise, we held no special distinction in the weekly group meetings or the Scrum meetings held twice a week with the subproject.

As we began to take over product owner responsibilities, we also began to chair the sprint ends for B&D. Once all were assembled for the meeting, each group in the subproject would present their work they had done for the sprint. Our Scrum process specified that presentation slides were outlawed for these meetings.

6.3.1 Backlog

As product owners we were in charge of creating and maintaining a backlog for B&D. In the beginning we used the Redmine tool to maintain a Backlog, however participation across the project groups were low, so the backlog suffered. Shortly after, the product owner groups started using Google Docs to keep track of user stories because Redmine was unwieldy.

At the closure of the second sprint, the product owners started keeping a shared product backlog.

Release Backlog

In the third sprint we were tasked with making a release backlog for the B&D subproject as a part of the overall Scrum process and to be able to provide a better overview of what has been made during the semester. At this point, there was no release backlog to speak of, given Redmine's lack of use.

A release backlog is a subset of the product backlog used for Scrum. Where the product backlog is a list of features wanted for the product, the release backlog contains the user stories which have been completed for a given sprint.

With the other product owner groups we put together a shared backlog which contained both the product backlog and the release backlog, to have a common place to find the whole backlog for the next semester's students. It can be found in Appendix A.

Collaboration 7

7.1 Build Structure collaboration

When we made AAR files a part of the project build structure, a number of groups were involved. This section will describe the collaboration we had with other groups to ensure that the build structure were implemented to the one described in Section 3.4

In our work with dependencies and making a standalone app, we found that the build structure could be improved by using AAR files. We then shared the idea with our subproject for its use to reform the build structure. It was agreed to implement AAR files and tasks were assigned between the relevant groups. From this point onwards, we acted as consultants for the other groups as it was implemented.

As part of the consultation we found the dependencies for the apps and libraries as described in Section 3.2. This was used to update the Gradle build files used to correct the libraries. The Gradle build files were then updated to use the Maven plugin by ourselves and the Jenkins group, which would allows projects to easily implement libraries, by simply asking for the libraries from Maven. The plugin downloads the implemented library on build time. This means that updated libraries can be used as a project without having to change any of the project's code. Maven also allows for backwards compatibility by using version numbers, which allows the individual projects to use the version of the libraries they need. Furthermore our work with the dependencies helped as consultation with the Git group when the recursive build structure had to be removed.

7.2 Collaboration between product owners

This section covers the collaboration work done between the three product owner groups.

As there were three groups that were product owners in the project, some of the tasks and information were shared in between us. This included user stories, tasks, and the information related to sprint planning and sprint ends.

7.2.1 Sprint ends and sprint planning

During each sprint a number of user stories were given to the product owners, but it was not always the correct subproject that received the right user stories. Therefore, we planned meetings with the other product owners in between each sprint end and the next sprint planning, where we shared the user stories we had, and discussed if there were any other user stories that were needed or if a user story was ambiguous and needed to be reformulated. We then distributed the user stories to the backlog for the subproject they

belonged to, so the user stories were ready to be presented at the sprint planning for that subproject.

7.2.2 Product backlog and release backlog

Since all product owners were responsible for making a backlog and a release log for the user stories, we worked together and collected them all in one document. This required that all the user stories were formulated in a similar way and that at least one from each group needed to be present, so all user stories for the three subprojects were included in the document.

In the process of making the document our group was the main contributor, in the sense that it was our responsibility to make the pdf document that should be included on the Redmine wiki page, for the next semester's students. The backlog can be found in Appendix A.

Conclusion 8

The conclusion has two aspects, one for the overall project, and one for our own work.

Overall the project was in a more refined state than it was, when we started working on the project. The apps now have a more consistent design in order to give the user the feeling that they are using a combined system and not ten different apps. The new design has not been implemented in all parts of the project, as not all apps were worked on during this semester. The database is now able to synchronize between two tablets running the Giraf-system. This is still in an early state and is still not ready to be released to the customers. The synchronization is also still limited to text strings and not pictograms which is one of the end goals for the feature to be capable of.

As for our own work we have helped the overall project by improving the app build structure. This helped reduce the build time for the Jenkins server. Overall, the build time has been reduced from around 25-30 minutes when we started the project, to 5-10 minutes at the end of the project. Internal renaming of the apps was done to make the app names consistent. The renaming was successful, but one of the apps did not work due to a dependency issue, which was not fixed. This means that all but one app has been renamed.

We have also done a lot of work as product owners for the B&D subproject. We helped direct the work for the other groups in the subproject. Our work was also important in making of a combined backlog for the entire project. This backlog will give the students next semester a place to start their work without having to look through every student report from this semester as was done this year just to find unfinished user stories. Our work as a whole was a success and a positive contribution to the project. Our work with the Scrum method fit in with the project nicely, but working in three subprojects caused some problems. Information was not always shared properly between the subprojects, which gave some issues along the way. On the other hand it was nice to have specialized groups that could focus on the more complex problems.

Recommendations for the next semester 9

This chapter is written for the students that take over the project next semester. The section contains recommendations and suggestions our group have for the next semester's students who will continue the project.

9.1 Issue handling tool

At the start of the semester we used a issue handling tool from Redmine, but as no one was using it and because it required too much effort to use, we started using Google Docs as a replacement. For the next year we would recommend that a new and better tool is found, so that issue handling becomes easier and faster to use, especially when used for maintaining the backlog as it is critical that a proper backlog is maintained.

9.2 Renaming

This year we renamed several apps to have more meaningful names in order to make it easier to understand what an app was supposed to do. The process used was tedious, and we do not advise any unnecessary renaming. That being said there are some issues with the some of the current names, which we will present here.

9.2.1 Categorygame

The categorygame currently on Google Play is the version using the wrong package-name of train. The correct version has been uploaded to Google Play, and any push to Jenkins on categorygame will upload the alpha-version to the correct version on Google Play, but that version will crash on startup, because the library metadata is referring to train. Fixing this would require making the app compatible with the methods to handle pictograms that was added in the newer versions of dblib.

9.2.2 Various other problems

Ugeplan is named in danish, and could be changed to follow the english names of the other apps. Some of the names on Git have not been changed, which could and should be finished early. The Maven repository has not been completely renamed, as only a few libraries were renamed, but it should be done to keep consistency.

9.3 Core library

The idea of the Core library is to bypass the launcher, while still having some of the launcher's features available for other apps to use. Section 3.3 goes into detail about these features. The one major problem of having the core package is to handle data management. A description of these problems and ideas on how to solve them can be found in Section 4.4. For the next semester we would recommend that the developers look into whether they want to make the whole project into one package, see Section 5.2. If the project becomes one package there will be no use for a core package. So it is important to have to discussion about a combined package before trying to solve the core library.

9.4 Giraf Package

For the last sprint we were given the task to bundle all the apps together into a single download. Our findings concluded that the only way to do this would be to make all the apps in the project into libraries and then include them in the launcher and by that making the whole project into one app. As this would have taken a lot of time and would have affected many of the other groups, it was decided to postpone it to next year. It is therefore our recommendation that if it is decided to make this change, the change should be made as early as possible.

9.5 User Story

To make user stories easier to write and understand a standard structure of how to formulate them were made. This formulation was "As a {type of user}, I want {some goal} so that {some reason}".

This structure made it clear how the main part of the user story should look, but we later decided that it lacked the constraints and conditions for when the user story was satisfactied. This meant that it in some cases was up to the group that had the user story to decide, when the user story was done. We recommend for the next semester's students to improve upon our model by including a standard way of formulating the constraints and conditions for satisfaction.

9.6 Backup

At the end of the semester there were some problems with a disk on the server that hosts the entire project. This meant that there was half a week where the server was down. For the next semester we would recommend having an external backup of critical data, in case something should happen to the server. This backup could be done at the end of each sprint. An idea for a backup solution would be to use Google Drive since there already is 15GB assigned to the Giraf Google account used in the project.(giraflists.aau.dk).

9.7 Process

9.7.1 Leadership

It is our experience that cooperation between students is inherently weak outside of their immediate vicinity and they will often focus on their own projects to the detriment of the greater project if not encouraged otherwise. Thus a structure can help facilitate clear lines of communication and responsibility but such a structure requires overhead to function in the form of meetings and planning. For this reason, we recommend finding a disciplined group with high standards and a good work ethic for overseeing the process run by the project on all levels.

9.7.2 Scrum

The development method that was used in the project and by our group was scrum. We recommend that scrum is also used for the project next semester, because of its flexibility. We would also advise specifying the process early.

9.7.3 Subprojects

For this semester, the project was divided into three subprojects B&D, DB, and GUI. At the start of the semester the number of groups was assigned to these subprojects, but over the course of the semester some of the groups moved to the GUI subproject due to the excessive amount of user stories in GUI. We recommend that the next semester keep the subprojects, but have most of the groups in the GUI subproject. We would also advise not to underestimate the need for further development on the B&D front or the advantages proper project management brings.

9.7.4 Weekly meetings

Since the development method used in the project was scrum we had weekly meetings where we discussed each subproject's status and brought up major decisions for the project. However we noticed that many issues which could have been brought up at these meetings were not. We suspect that the reason for this is that in order for anything to be taken up, a person would have to interrupt the meeting to get his question brought up. For this reason we recommend that a suggestion box is made in some form, where everyone can add notes at any time of the week with questions or issues which can then be brought up at the meetings.

9.8 Server

Over the course of the semester the project had some trouble with the group responsible for the servers. There were often very few of the members present at the university. This meant that often when there was problems with the server it took a long time before the problem was fixed. We therefore recommend that for the next semester two groups are responsible for the server or at least make sure that the group responsible for the server is a group that always is at the university at working hours.

Bibliography

- Android 4.0.3 apis, March 2014. URL <http://developer.android.com/about/versions/android-4.0.3.html>.
- Android studio, March 2014. URL http://en.wikipedia.org/wiki/Android_Studio.
- Google play, March 2014. URL http://en.wikipedia.org/wiki/Google_Play.
- Gradle, March 2014. URL <http://en.wikipedia.org/wiki/Gradle>.
- Redmine, March 2014. URL <http://en.wikipedia.org/wiki/Redmine>.
- Google analytics, May 2015. URL <https://developers.google.com/analytics/devguides/collection/?hl=en>.
- Javadoc tool, May 2015. URL <http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html>.
- Pegi rating, May 2015. URL <http://www.pegi.info/en/index/id/23>.
- Things that cannot change, May 2015. URL <http://android-developers.blogspot.dk/2011/06/things-that-cannot-change.html>.

Backlog A

This is a full list of user stories, including both finished user stories, the Release Backlog, and user stories we have not worked on in this semester, Product Backlog.

A user story is released when it have a working version. In the beginning of the project there was implemented a lot of the features from the user stories, but many of them was not able to run.

Sometimes new user stories are added which are duplicates of old finished user stories. That happens when it is decided they need to be worked on again.

Each user story consist of a title, a priority from 1-5 where 1 is most important, and the story. If the priority contains a hyphen, '-', instead of a number then it hasn't been prioritized yet. Furthermore the user stories are categorized to some extent.

The format of the user stories are as follows:

"As a (end user), I want to (some goal) so that (some reason)".

In the user stories there are different kinds of users used:

Developer People who work on the development of the GIRAF, both current and future developers.

Citizen Children, adults, and elders with an autism spectrum disorder or other developmental disabilities.

Guardian Legal guardians (e.g. parents) or institutional guardians (e.g. pedagogues).

User All kind of users of GIRAF.

A.1 Product Backlog

This section includes all user stories there is not finished.

A.1.1 GUI

General

These user stories should have an impact on all the applications.

1. **Unexpected crashes (1)**

As a user I want the system to work flawlessly and no unexpected crashes occur, as it is very frustrating for the citizens if the systems does not work as expected.

2. **Access (4)**

As a guardian i want closing an application for a citizen to be protected with a

password, so they cannot make any changes or use other application than i have aloud. If closing the application is not approved, then the citizen can return to the application, he was currently executing.

3. **Pictogram sound indication (2)**

As a user of the system, all applications should show, and allow me to play, a pictograms associated sound, if there exist a sound for the pictogram, as the citizens can learn the sound of a pictogram without being able to read.

4. **Change color scheme (4)**

As a guardian I want to adapt the color scheme for a citizen, as this makes it easier to see who is logged into the tablet.

5. **Consistency (1)**

As a user i want that the design is consistent, as this is easier for all applications to learn the icons and design.

6. **Dual languages (1)**

As a developer i want to have the system in danish for the customers and users, and in english for my supervisors.

7. **Icon overview (-)**

As a new user i want an overview of all icons and their meanings, not necessary as a part of GIRAF, so i can easily learn the meaning of each.

8. **Help (2)**

As a guardian i want to have a help function, i can use if i do not know what to do.

9. **Undo (1)**

As a user i want undo/redo in all relevant applications, so i can easily recover from errors.

10. **Change log-in work flow to be compatible with SymmetricDS log-in process (-)**

As a user, I would like the log-in process to be informative and help me log-in when synchronisation is enabled, so I do not have to wait for data to download before I can log-in.

11. **Update all relevant info on sync in all apps (-)**

As a user, I would like all apps to update their views when synchronisation completes, so I do not have to restart the apps to obtain any changes to my data.

12. **Google Analytics (3)**

As a developer i want all modules of GIRAF to report crashes, so potential bugs can be fixed. By implementing Google analytics in the modules, crash reports can be collected.

13. **Commented code (-)**

As a developer I want the code to be well commented and documented, such that I can easily understand it and develop on it.

14. **Unused code (2)**

As a developer I want unused code to be removed, as it is unnecessary and confusing.

15. **Generalising (5)**

As a developer I want the code re-factored, such that similar methods are generalised across applications.

16. **Use new libraries (1)**

As a developer i want, all applications to use the new versions of libraries, so everything is up do date and using the newest components.

17. No GComponents (1)

As a developer, I would like applications to not use GComponents any more, so the old components can be removed from the code.

18. Screen scaling (5)

As a user I want the system to work on tablets of different sizes, ranging from 7" - 10" in size, so i can use GIRAF on normal sized tablets.

Launcher

The launcher is the home screen of the GIRAF system. From here, the GIRAF- and Android applications can be placed, and easily opened. These user stories are specific for the Launcher.

19. Set passwords (4)

As a guardian I want to set the password to different types (pin-code, swipe a pattern or QR code), and be able to turn these on and off for citizens, as citizens above 18 years old should not be restricted access, whereas citizens younger than 18 should be restricted.

20. Error message (-)

As a user i want an error message if there is no Internet, and the GIRAF is trying to download data. When there is a Internet connecting, i should start download data. This ensures i do not have to wait for download, when it is not working.

Profiles

These user stories describes how the profiles are used.

21. Profile breakdown (2)

As a guardian I want to find the group i am working with, as i then can find the citizen I am helping, with a few clicks. If no group is chosen, then the citizens should be listed alphabetically.

22. Settings for multiple citizens (2)

As a guardian I want to be able to save settings for one or multiple citizens, and be able to copy these settings from a citizen to another, making it easier to make settings. The settings should include game settings, week schedules, sequences, pictograms and categories.

Pictosearch

Pictosearch is the application used to find pictograms. This is used whenever the possibility of searching exist. Pictosearch can provide a list of pictograms and categories. These user stories relevant for the searching.

23. Searching strategies (1)

As a developer, i want to improve the searching functionality by implementing a more effective searching strategy, so I can see the search results quicker. The searching strategy should be well documented.

24. Sorting strategies (4)

As a developer, i want to improve the sorting functionality by implementing a more

effective sorting strategy, so the search results will be sorted quicker. The sorting strategy should be well documented.

25. Access to private pictograms (-)

As a guardian, I want to be able to find the private pictograms associated with the current citizen, so I can use them for things like sequences specific to that citizen.

Week Schedule

Week Schedule is used to show the citizens the plan for their week or day. These user stories is covering the features wanted in Week Schedule.

26. Refer to the sequence application (3)

As a citizen, when I click on an activity, it should lead to a saved sequence, so I can get a visual description of how to do the activity. I want that it is visible whether or not the activity has this feature.

27. Guardian for the day (3)

As a citizen I want to see which guardian is attached to me today, as I then can see who I should contact when I am done with an activity.

28. Copy week schedule (1)

As a guardian I want to copy entire weeks and single days, from one citizen to another, because many of the weeks and dags has some similarities, and this would ease the work of creating schedules.

Sequence

Sequence is used to create a series of pictograms, that is used to describe how an activity should be done. The citizens can then see these sequences when performing the activity. These user stories is for features in sequence.

29. Email sequences (2)

As a guardian I want to email sequences, so I can print them, and use them without GIRAF.

30. Copy sequences (2)

As a guardian I want to copy sequences between citizens, so I easy can use the same sequences for multiple citizens.

31. Save edited sequence (3)

As a guardian I want to choose if a change in a sequence should result in the existing sequence being overwritten or if a new sequence should be saved, as this could also be the case.

32. Choice pictogram (4)

As a guardian I want to be able to choose the pictogram illustrating a choice in a sequence myself, so it fits a particular choice, as it makes it easier for the citizen to understand the different choices.

33. Video sequence (5)

As a citizen I want to see a video, that shows me how to execute a sequence, as this can better explain me to complete the sequence.

34. Possibility for citizens to create sequences (4)

As a citizen I want to create sequence myself, because the guardian think I can manage this.

Picto Creator

The Picto Creator is a drawing app, that can be used to create and edit pictograms. It is possible to use the camera and microphone when creating these. These user stories is for Picto Creator.

35. Delete a pictogram (4)

As a guardian I want to delete a pictogram from the system, if a citizen does not need it, or if it gives bad memories.

36. Copy or replace (1)

As a guardian I want to choose whether a pictogram should replace an earlier version or create a new pictogram with a new name, as this choice varies.

Category Tool

The Category Tool is used to create, edit and delete categories of pictograms. Categories makes it easier to find and use pictograms in the other applications. There is no more user stories for this application.

Picto Reader

Picto Reader is an application used by citizens, making them able to create a series of pictograms, that can be read aloud or shown to guardian. The application is used to communicate. These user stories is relevant in Picto Reader.

37. Sentence construction (2)

As a citizen I want to construct sentences with pictograms, for instance [I would like] or [I] [would] [like], because I want to learn to talk using sentences.

38. Blocking (3)

As a guardian I want to make a visual blocking of a category or a pictogram, since there can be certain pictograms or categories that cannot be used in a given situation.

39. Loading (2)

As a citizen, I would like to be informed that the application is loading a newly chosen category, so i know something happens.

Timer

The Timer application can display a timer in the other applications. This is used to restrict how much time a citizen can use the tablet or chosen application. These user stories covers the features needed in the Timer.

40. Pause (3)

As a guardian I want to pause the timer when a citizen is not doing the given task, to be sure that the allocated amount of time is being used on the agreed activity, for instance sitting at the table.

41. Locking the tablet (3)

As a guardian I want to decide if the tablet locks after the time is up or if it can be used freely after, so I can decide if a citizen should move on to another activity or keep using the tablet.

42. Activity after completed time (4)

As a guardian I want to set the timer to decide what activity happens after completed time to make sure the day is progressing as planned.

43. Pre-defined timers (4)

As a guardian I often use 3, 5, 10, 15 and 20 minutes in the timer, so they should be defined per default, such that they can be used faster.

Life Stories

Life Stories can be used to create sequences of a certain activity, e.g. going to the fair. This can be used to show the things they have done. These user stories covers features in Life Stories.

44. At the end of the day: Citizen (5)

As a citizen I want to explain my day through pictograms, because I can use this to support me when telling others about it.

45. Adding text and sound (5)

As a guardian I want to add text and sound to life stories. This will help the user remembering the specifics about the current life story.

Category Game

The Category game is about a train, where the pictograms of some categories are at the first station, and must be moved to the correct station. This is the relevant user stories.

46. Instructions (3)

As a citizen I want a short introduction to the game so that I know how it works without help from a guardian.

Voice Game

Voice game is about changing lanes with a car, where the car is controlled by the users voice. The user is supposed to pick up/avoid object along the way.

47. Standard games (5)

As a guardian I want to have default games for practising high and low voice control, so I can quickly set up the needed game.

Administration

The Administration app is used to create, edit and delete users.

48. Backup (2)

As a guardian I want to be able to take a backup of personal pictograms, categories, sequences, and weekly schedules, as these might be useful later.

49. Manage guardians (2)

As a guardian in an institution I want to create new guardians and edit the current guardians in my institution, as new people might be hired in an institution or information about guardians might change.

Web Administration

Web version of administration app, but with additional features.

50. Administrate profiles (1)

As a guardian I want to administrate the citizen profiles from a homepage, such that I am not limited to using a tablet for doing this.

A.1.2 Database**51. Should be able to track synchronization result(2)**

As a guardian, I would like to be able to check the synchronization status, so I can see if I have unsaved changes, and when I last synchronized.

52. Sync status (2)

As a guardian I would like to be notified of the results of synchronization, so I know if my data have been uploaded.

53. Manual synchronization (5)

As a guardian I would like to be able to manually initialize a synchronization, so I can upload / download changes if they have not been synchronized automatically.

54. DB component usability (5)

As a developer I would like for the DB components to be easy to use, so I can abstract away from the db structure when I write new apps.

55. Security (4)

As a user I would like my confidential data to be secure and for the app to adhere to all laws, so I can safely use the GIRAF app.

56. Mine private data skal ikke kunne ses af andre(1)

As a user I would like my confidential data to be secure and for the app to adhere to all laws, so I can safely use the GIRAF app.

57. DB subset for stand-alone apps (4)

As a B&D developer I would like to download a defined subset of the DB, so that I can run a stand alone version of apps that require pictograms, without having a user or being able to make changes.

58. Save color schema for giraf (5)

As a guardian I would like to save the color schema for certain citizens, so that conflicts regarding ownership of tablets.

59. Private pictograms (2)

As a guardian i would like to have private pictograms, which can be added, removed and edited. I should be able to allow the pictograms for citizens profiles, so that the pictograms can be used only by me and my citizens.

60. Edit public pictograms (2)

As a guardian i would like to use a public pictogram, edit it and save it as a private pictogram, so they can be used as templates.

61. For every pictogram a sound clip must exist (5)

As a citizen I would like to have sound clips for every pictogram, so that I can have them read aloud.

62. Allow the addition and removal of sound from a pictogram (5)

As a guardian I would like to manage the sounds of pictograms so I can remove wrong sounds and add correct ones instead.

63. Pictogram exclusion (3)

As a guardian I would like to be able to exclude certain public pictograms from the apps of my assigned citizens, so I can limit the number of pictograms I have to search through when I set up sequences and other things.

64. Save choice between multi-choice activities and whether it is complete or not. (1)

As a guardian I would like to be able to save the citizens choice in the multi choice, so the actions that have already been taken are shown with the action chosen

65. Save 'Progress' (1)

As a citizen I would like to be able to see how far I have come in my weekly schedule, so I do not have to remember this every time I close the weekly schedule

66. Save 'templates' for week schedule (2)

As a guardian I would like to be able to save templates for weekly schedules for citizens and share them between multiple citizens so I can more easily create new weekly schedules for citizens

67. Save Life Story for citizens (5)

As a guardian i would like to be able to save and manage life stories for citizens, so i can do it for the citizens

68. Save how many pictograms to be shown at a time (5)

As a guardian i would like to be able to set settings for citizens life story app such as how many pictograms are shown at a time, so that the app fits the specific citizen.

69. Restart of init download(1)

As a user i would like the initial download to continue after a disconnect when the internet is available again so.

A.1.3 Build and Deploy**Structural Changes**

These user stories are related to the overall structure of the project.

70. Replace Redmine with another tool (4)

As a developer I would want to use another tool instead of Redmine so that i could get a more reliable organization tool as the handling of user stories on Redmine is clumsy and is therefore not used by the majority.

Server

These user stories should have an impact on the server or be strongly related to server work.

71. Acquire faster servers (3)

As a developer I would like the servers to be faster, so I do not need to wait too long for Jenkins to build my newest release as others could be waiting for the new release.

72. Security standards (5)

As a developer I would like the server to follow the security standards set by the Government, so that the server can be moved to a server run by the Government.

73. Automatic update of scripts (-)

As a developer i want scripts on the server to be automatically updated when i make changes so that i don't have to do this manually.

Jenkins

These user stories should be related to Jenkins and any work done on Jenkins.

74. Monkey testing screen-capture from last failure (2)

As a developer I would like to have screen-captures taken during the monkey test so that i can get feedback about where the failure happened.

75. Safely disconnecting tablets from testing pool (-)

As a developer I would like to be able to disconnect my tablet from the testing pool safely, without breaking a build on Jenkins.

76. Jenkins Configuration Backup (-) As a developer I want to have a backup of the Jenkins configuration so that it can be restored in case of failure. It must be stored on another machine than the server.**77. Easy test of apps and libraries (-)** As a developer I want apps and libraries to be easily testable so that i can write tests.**78. Prevent push of snapshot version of libraries (-)** As a developer I want to be prevented from pushing snapshot versions of libraries so that i don't mistakingly do so.**79. Automatic notification of failed monkey test (-)** As a developer I want to be automatically notified of monkey test failure so that I an notified when my builds fail.**80. Emulator fallback (-)** As a developer I want emulator fallback if tablets disconnects in the middle of builds so that build jobs don't fail.**Build Release**

These user stories should be related to the build release of GIRAF.

81. Make a package for download on Google Play

As a customer I would like to have a package available on Google Play that would allow me to download just one thing that would install all needed parts for the Giraf-project, so I do not have to download them all individually.

Documentation

These user stories should be related to the documentation of GIRAF.

82. Update and extend descriptions of apps and libraries (2)

As a customer and as a developer I would like to have better and more up-to-date descriptions of every app. This should be in some commonly shared place and on Google Play.

Stand-Alone apps

These user stories should be related to the stand-alone concept.

83. Stand-Alone: Core package (5)

As a customer I would like the apps to be stand-alone so that i don't have to install any additional app to get one specific app working.

Git

These user stories should be related to the Git-repositories.

84. Set up major and minor tags on Git (2)

As a developer I would like to be able to quickly and easily revert to a previous state of the development using tags for every major and minor release.

Analysis

These user stories should be related to analysis of potential new ideas for GIRAF.

85. Future technology analysis (5)

As semester coordinator I would like to know which future technologies could be useful in the project's future.

A.2 Release Backlog - Sprint 1

This section includes all user stories from the release from Sprint 1.

A.2.1 GUI

1. Design document (1)

As a developer I want a design document with guidelines for the design, so I can make GIRAF's design consistent for the users.

2. Opening an application using the launcher (1)

As a user I want to choose the applications that i want to work with through the launcher, so I have a common place to access my applications.

3. Understandable applications (1)

As a guardian I want to read the application name on my native language, and want to conclude from the name, what the application is capable of, so the system is understandable for me.

4. Allow applications (1)

As a guardian I want to choose which applications a citizen has access to, both GIRAF applications and the applications installed on the tablet, so they only have access to relevant application.

5. Show applications (1)

As a guardian I want to be able to decide how many applications is present on each page, for a specific citizen, making this adjustable to the citizens capability.

6. Android settings (1)

As a guardian I want the opportunity to access the android settings, enabling me to change these.

7. Authentication responding to the the user (1)

As a user logging in i want better feedback of successful and failed QR scans, so i know if it works.

8. Log in for developer (1)

As a developer I want a button to skip the authentication for development purposes, as it is annoying to use the QR scanner to log in.

9. Show collections of pictograms (1)

As a guardian I want to find pictograms when I need it, for example when I am creating a week schedule or a category.

A.2.2 Database**10. Unit Testing of DB-lib(1)**

As a developer I want DB-lib to be thoroughly unit tested to ensure functionality when code is changed in DB-lib.

11. Reduce Initial Download Time(1)

As a user I want the initial download to take less time so I do not have to wait for so long.

12. Compress pictograms(1)

As a developer I want the pictograms compressed to reduce the initial download time so the user do not have to wait for so long.

13. Remove the redundancy present in the DB-lib controllers and models(1)

As a developer I want the redundancy present in DB-lib to be removed so it is easier to use and change.

A.2.3 Build and Deploy**14. Setting up Git**

As a developer I would like to have git up and running, so that I can access and share code for the apps in the project.

15. Setting up server

As a developer I would like for the servers to be running, so that I can use all the tools running on them, and so that I can access the files located on the servers.

16. Google Analytics Setup

As a developer I would like to have Google Analytics setup in the apps so that I can get get crash reports for the apps.

17. Setup of Google Play

As a developer I would like Google Play to be used in the project, so that the apps can be released on the app store.

18. Update to Android 1.0.x

As a developer i would like to have the version of Android studio used updated to version 1.0.x, so that I can use the new functionalities in that version.

19. Central Documentation

As a developer I would like to have a place, where all the code in the project is documented, so that I do not have to look several different places or source code files to find the one function I need.

20. Continuous build and integration

As a developer I would like to have an automated build-function running on Jenkins, so that I know the build I upload is attempted to be built and tested.

21. Wiki entry on process

As a developer I would like to have a specification on how the project's overall process is structured, so that I know what my responsibility in the project is and how I should work as a part of the greater unit.

A.3 Release Backlog - Sprint 2

This section includes all user stories from the release from Sprint 2.

1. Product Backlog (1)

As a developer I want a Product Backlog that represents the users need, so I can work based on this in the following sprints.

2. Delete sequences (3)

As a guardian I want to delete sequences for citizens, when they no longer are relevant.

3. Undo (1)

As a user i want undo/redo in Picto Creator, so i can easily recover from errors.

4. Creating a pictogram (1)

As a guardian I want to create a new pictogram, and save this to a specific citizen or the institution, to describe an action or thing that is needed.

5. Searching (1)

As a guardian, when I am searching for pictograms, I want to be searching for both the names, the categories, and the tags assigned to them creation.

6. Day-mode (1)

As a citizen I want to see the plan for one day, by rotating the tablet to portrait mode. This makes me able to see more pictograms and contains no disturbances from the other days.

7. Save and edit week schedules (1)

As a guardian I want to save and edit week schedules such that I can make changes in a week schedule if necessary.

8. Keeping track of the schedule (1)

As a citizen I want to keep track of the schedule by identifying the color of the weekday, and then find the schedule for that day, to give me an overview.

A.3.1 Database

9. One Way Compatibility Between Remote and Local DB(1)

As a developer I want to be able to save everything in remote that i am able to save in local to avoid database errors.

10. Evaluation of DB Design(1)

As a developer I want to evaluate the design of the database to deem if a rework is needed at some point.

11. Save Setting in LocalDB for a Profile(1)

As a developer i want to be able to save a users settings in his profile in LocalDB so they are available next time he log in.

A.3.2 Build and Deploy**12. Binaries instead of submodules**

As a developer I would like to use binary files in the project so that i do not have to use submodules on the git repositories.

13. Snapshots of database/Backups

As a developer I would like to have snapshots of the database, so less data is lost in the event of a server crash.

14. SMTP setup

As a developer I would like to have access to a central mailing system that can automatically notify me of changes, so that I can easily and without effort on my part stay up to date.

15. Logging

As a developer I would like to have logging for the database so that the server follows the requirement set by the Government.

16. Gitlab for the server/GOG

As a developer I would like to have access to a server-side Git-interface, so that it would be easier to use Git.

17. Faster build and test

As a developer I would like faster builds and tests on the jenkins, so that I do not have to wait as long for applications to be built and tested.

18. Code coverage

As a developer I would like to have code coverage on jenkins so that i can how much of the code is tested by Jenkins.

19. Test case installation

As a developer I would like to have Jenkins attempt to install the new builds, so that I can avoid conflicts in installation situations.

20. Release new APK for Sprint 1

As a customer I would like to have access to the newest stable version of the program, so that I can enjoy the new features I have had presented.

21. Auto Upload Beta and Alpha release

As a developer I would like to have a Alpha and a Beta release of the different apps, so that I can test the app from the store without releasing an unstable version to the customer.

22. Dependency

As a developer I would like to know which apps and libraries that are dependent on each other, so that I know which libraries to include in my app when coding.

A.4 Release Backlog - Sprint 3

This section includes all user stories from the release from Sprint 3.

A.4.1 GUI

1. **Create sequences (1)**
As a guardian I want to create sequences for citizens, including choices, so I can show them how to perform tasks.
2. **Edit sequence (3)**
As a guardian I want to replace pictograms in a sequence, change the title, and change the thumbnail, because a citizen for an example got a new jacket and need to use the pictogram with the new jacket instead of the old.
3. **Delete sequences (3)**
As a guardian I want to delete sequences for citizens, when they no longer are relevant.
4. **Add pictogram to category (1)**
As a guardian I want to add an existing pictogram to one or more categories if I think the pictograms is missing from the category.
5. **Remove a pictogram from a category (1)**
As a guardian I want to be able to remove a pictogram from a category, if I have mistakenly added a pictogram to a category where this does not belong.
6. **Create category (2)**
As a guardian I want to create a new category, because I need a new category, and the existing ones does not suffice.
7. **Show categories (1)**
As a guardian I want to see the categories associated to the institution and specific citizens, because I want to see and edit these.
8. **Delete category (2)**
As a guardian I want to delete a category, because I made one by mistake or if the category is no longer necessary.
9. **Manage categories for more than one citizen (-)**
As a guardian i want to create, edit and delete categories for more than one citizen at the time, so i do not need to make the same category more than one time.
10. **Help function (-)**
As a user i would have some help functionality that explains what the various views in the app does and contains, so i can easier can learn the system.
11. **Print (1)**
As a guardian I want to print out pictograms, so i can use them in physical tools.
12. **Eraser tool (1)**
As a user i want to be able to erase parts of a pictogram, so it fits my needs.
13. **Creating a pictogram (1)**
As a guardian I want to create a new pictogram, to describe an action or thing that is needed, and save this to a specific citizen or the institution.
14. **Access category tool (1)**
As a guardian I want to be able to open the Category Tool from Pictosearch, if a pictogram is missing in a category, or if an entire category is missing.

15. Viewing categories (1)

As a guardian when I am searching for pictograms, I want to find entire categories, which I can enter to pick certain pictograms. This will make it easier to find pictograms.

16. Changing to new activity (1)

As a citizen I want to keep track the activities for the day. I expect an indication about which activity I am currently at.

17. “Choose” activity (1)

As a guardian I want to give the citizens the opportunity to choose between many possible activities, so the citizen themselves can choose which activity they want to do. I want to choose up to 10 activities, that the citizens can choose between.

A.4.2 Database**18. Save game settings(4)**

As a guardian i would like to save settings for the category-game, so that it can be used another time without setting it up.

19. Enable storage of more settings, like presets(4)

As a guardian i would like to save presets of settings for the category-game, so that it can easily be used another time.

20. Save Setting in LocalDB for a Profile(1)

As a developer i want to be able to save a users settings in his profile in LocalDB so they are available next time he log in.

21. Change id datatypes(2)

As a developer I would like all ids returned by the DB components to be longs, so that it conforms with the Android standards and I do not have to cast them every time.

22. Naming of apps(1)

As a B&D developer I would like all apps to be consistent in the naming they use, so I can more easily publish apps to google play and so I don not have to know which names correspond to which.

23. Status for large queries(2)

As a developer i would like to be able to get information of the status when doing larger queries, so that it can be used to notify a user of the app.

24. Profiles divided into organizations(1)

As a guardian I would like to divide my department into groups, so I can then view citizens based on a defined subcategorization, e.g. Blue room and Red room.

25. Guardians and citizens can be associated with departments(1)

As a guardian I would like to edit changes for citizens in my department, so I can edit changes for the citizens I am responsible for, and get a list of the citizens I am responsible for.

26. Pictogram categories(1)

As a guardian i would like to save, see, use, add, remove create, modify and change categories for pictograms, so that the pictograms can be grouped.

27. Enable saving of permissions in the DB(2)

As a guardian I would like to be able to save extra permissions for citizens, so citizens

that are able to make their own sequences and such can be allowed to do so without using a guardian profile.

28. **Save sequence settings(5)**

As a guardian i would to save settings for the sequence app, primarily whether a citizen is allowed to create sequences, so that it can be managed which citizens that can create sequences.

29. **Sequences compatible with weekly schedule(3)**

As a guardian i would to be able to use sequences in the weekly schedule, so that it is easy to in both apps.

30. **Save weekly schedule for citizens(3)**

As a guardian I would like to be able to save the weekly schedule for citizens so I can have multiple weeks saved for each citizen.

31. **Tags as attributes of pictograms(1)**

As a developer group I would like the tags of a pictogram to be available as an attribute of a picogram when it is sent to my app.

32. **Feedback regarding connection(2)**

As a guardian I would like to be informed if internet connection is unavailable during initial load, so I do not wait forever.

A.4.3 Build and Deploy

33. **Logging**

As a developer I would like to have logging for the database so that the server follows the requirement set by the Government.

34. **Symmetric DS**

As a developer I would like for the server to support synchronization with the database.

35. **Snapshots of database/Backups**

As a developer I would like to have snapshots of the database so in the event of server breakdown, less data is lost.

36. **Monkey test**

As a developer I want to have monkeys tests, so that the reliability of the apps can be tested.

37. **UI test on Jenkins**

As a developer I would like to have UI test on jenkins so that the UI can be tested for an app when it is pushed on the master branch.

38. **Share the knowledge from Guides**

As a developer I would like for all groups to know all the guides they have access to.

39. **Add a guide on how to do UI test**

As a developer I would like to have a guide on how to do UI tests.

40. **Make guidelines for Continuous Integration**

As a developer I would like to know the specific differences between a major and a minor release.

41. **Specify the Scrum process used**

As a developer I would like to have the Scrum process used by the project specified to better integrate groups into the process.

A.5 Release Backlog - Sprint 4

This section includes all user stories from the release from Sprint 4.

A.5.1 GUI

1. Update design document (1)

As a developer i want the design document to be updated with all the design decisions, so it becomes easy to make a consistent design.

2. Loading progress (2)

As a user i want to see how the loading is going in the launcher, so i can know that it is going forward.

3. Add profiles (1)

As a guardian be able to add a new citizen profile, if a new citizen has started in the institution, so he can have a profile with his personal settings and pictograms.

4. Delete profiles (1)

As a guardian I want to be able to remove a citizens profile from the system. I want to be able to delete his personal pictograms from the system, as these are no longer necessary. I want large degree of certainty about confirming that these should be permanently deleted, as i do not want to lose any pictograms by accident. Because a citizen is no longer a part of my institution.

5. Transfer profiles to new institutions (5)

As a guardian I want to be able to transfer a citizen profile to a new institution, so the citizen can keep his personal data, when he is moving to a different institution.

6. Move citizen to a new group (1)

As a guardian I want to be able to move a citizen from one group to another, as this would be much easier than deleting and creating the profile again.

7. Manage groups (4)

As a guardian I want to create new groups in my institution and edit the current groups, as citizens can be assigned to a new group, or moved from one group to another.

8. Train a citizens voice (3)

As a citizen I want to play a game to practice the level of my voice, as I have a hard time controlling my voice.

9. Collect or Avoid in Voice game(3)

As a guardian I want to set up the track for a citizen, choosing if they should collect stars or avoid boxes, so I can a game best suited for the citizen. When this is set, I want to go directly to a citizen profile, and let that citizen play.

10. Using the Timer (1)

As a guardian I want to set a timer with a visual representation, of how long a citizen have to do an activity, in full screen, because it makes the concept of time easier to understand.

11. Stop timing (1)

As a guardian I want to stop the timer while it is running, because a situation has happened where it no longer makes sense having the timer running.

12. **Timer available in other apps (3)**
As a guardian I want to set the Timer to be seen in another application so the citizen can keep an eye on the time meanwhile.
13. **Show a sequence (1)**
As a citizen I want to see a sequence, that explains me how to perform a certain activity, as I cannot remember and manage the correct order.
14. **Mark activity as done (1)**
As a citizen I can have trouble managing a long sequence. I want to mark activities as done, so I can keep track of the progress in the sequence.
15. **Return pictograms and categories (2)**
As a guardian I want to get one or more pictograms or categories, based on the application that uses Pictosearch, so I can find what I need to use.
16. **At the end of the day: Guardian (5)**
As a guardian I want to help a citizen finding the useable pictograms for him to use for his life story, so that the user will have an easier time creating his life story.
17. **Lock actions (2)**
As a citizen I have to take a choice when i encounter a “Choose” activity, since I can not skip or cancel activities. I cannot use the back button or clicking outside the box to close it.
18. **Showing a specific number of pictograms (1)**
As a guardian I want to adjust the amount of pictograms that is shown in day-mode in Week Schedule, as it differs how many pictograms a citizen can handle.
19. **Edit week schedule (1)**
As a guardian I want to edit a week schedule, marking an activity as cancelled and add new activities, as the daily schedule should correspond with the actual day.
20. **Train citizens understanding of categories (3)**
As a citizen I want to play the category game to get a better understanding of categories and how to divide them.
21. **Create a game from scratch (3)**
As a guardian I want to create a category game from scratch with new categories I specify at the same time, so I can customize the game exactly how I need.
22. **Create a category game with pre-made categories (3)**
As a guardian I want to start a game with some pre-defined categories that I have made previously, so I can quickly set up a game.
23. **Create new pre-made game settings (-)**
As a guardian I want to have a predefined category game ready, so i can let a citizen start the game easily.
24. **Separate Life Stories and Week Schedule (1)**
As a developer i want to have ‘‘Life Storie’’ and “Week Schedule” placed in two separate repositories.
25. **Restructure Week Schedule (1)**
As a developer i want to restructure my use of fragments in the Week Schedule code.
26. **Library (1)** As a user i want Pictosearch to be a library used by the other applications, so i do not need to install it as a separate application.
27. **Screen size (1)**
As a user I want the GIRAF applications to work on a 10” screen, as this is the tablet size we use.

28. App naming must be consistent (2)

As a user i want the names of all applications to be meaningful, and consistent throughout the system, so i know which application there is referred to.

A.5.2 Database**29. Synchronize Automatically(1)**

As a guardian I would like that my data is accessible on any tablet when logged in, so that I can switch tablet without noticing a difference.

30. Database Namespace(1)

As a developer I would like the database namespace to match the name of the applications.

31. Department-wide Save of Pictograms(2)

As a guardian, I would like to save pictograms, categories, and sequences at the various administrative levels of my department, so I can reuse these when I set up profiles.

32. Reduce Ram use(2)

As a developer, I would like for GIRAF to use less ram so it is possible to run on low-end tablets and to run more smoothly on any tablet.

33. Better feedback on initial download (1)

As a user I would like to get more precise feedback on the initial download so I know how much time there is left.

34. Useless views in remote database

As a future developer I would like the server to be cleaned of all unused views, so I have more control of the access to the database.

A.5.3 Build and Deploy**33. Ensure App Name Consistency**

As a developer I would like all the references to an app and library to use the same name so that it is easier to follow and understand which apps are referenced when reading a name. All names on git, jenkins, and google play should be consistent.

34. Make Pictosearch a Library

As a customer I would like PictoSearch to be a Library so that i don't have to download it separately from other apps.

35. Reset the dummy-database every night

As a developer I would like the dumme-database to reset every night to clean up any mess that might have been made during the day so that these things do not cause any trouble in the future.

36. Prioritization on Jenkins

As a developer I want libraries to have higher priority than other jobs in the build scheduling on Jenkins so that libraries are not bottlenecked when several people are reliant on them.

Jobs must not be starved.

Libraries must be prioritized higher than other job types.

37. **Ensure that every unit-test, monkey-test, ui-test, and maybe integration-tests runs up against the test database**
As a developer, as a future developer, and as a customer I would like the database to not be wiped every time a test is run because I would then lose valuable data.
38. **Monkey tests on debug versions of apps**
As a developer I want tests to run on the debug version of apps so that they can be tested on a test database.
Monkey tests must run on debug APKs.
39. **Monkey testing screen-capture from last failure**
As a developer I would like to have screen-captures taken during the monkey test so that i can get feedback about where the failure happened.
40. **Decrease job build times on Jenkins (Technical work)**
41. **Easy way to download and install all apps**
As a developer I want an easy way to download and install all apps so that it is easy to test the apps combined, and easy to show the external customers.
Users must not install additional software onto their computers.
Old versions of apps must be updated.
Users should be able to merely run a program that then automatically downloads and installs all apps on a device.
42. **Preparations for next semester** As a developer i would like to have guides provided that specify how to use the development tools used in the project, so that is don't have to use as much time researching how to use them before i can start working on the project.
43. **One combined place/report with all relevant information**
As a future developer I would like for all relevant information to be available in one place whether this is a report or some other place I have access to, in order to have an easier time getting in to the project. All relevant information is on giraf.cs.aau.dk.
44. **Add code style check to Jenkins (4)**
As a developer I would like to have Jenkins automatically check whether I follow the specified coding style, because it would allow me to focus on developing content rather than spending time on prettying my code.

B.1 Google Analytics Guide

Info: De fleste apps har fået tilføjet dette sidste år.

Info: Der kan gå op til 24 timer før, dataen der bliver sendt fra enhederne til Google Analytics, er synlig på Google Analytics.

Info: Følg denne guide fra top til bund!

B.1.1 Skaf adgang til data

1. Opret en task på redmine linket til Developer Story #1840 , navngivet "Google Analytics GRUPPENAVN".
2. I descriptionen skriver I den eller de google-konti som skal kunne se dataene. **Det er vigtigt at det er en google-konto, evt. kan I linke jeres aau-mail til en google-konto.**
 - a) Skriv evt. også hvilke specifikke apps I skal have adgang til.
3. Når issuen er oprettet giver vi jer adgang så hurtigt som muligt.
4. Hvis der er yderligere spørgsmål til de data der er, så kontakt os (opret en issue eller stik hovedet ind af døren til 1.1.32).

B.1.2 Brugsguide:

1. Åben <https://www.google.com/analytics/> og log ind med en google-konto med adgang til projektet.
2. Vælg tidsperiode der skal vises data for i øverste højre hjørne. Som standard er det den sidste måned.
3. Vælg hvilken app der skal vises detaljer på listen, ved at klikke på Alle mobilappdata under den enkelte app.
4. Du kan nu se en oversigt over brugen af app'en, men de vigtigste informationer ligger under Behaviour -> Crashes and Exceptions i venstre side. Adfærd -> Nedbrud og undtagelser hvis dit google er på dansk.
5. Under Exceptions kan du se om app'en er crashet inden for den valgte tidsperiode, hvis den er, kan du få detaljer om dem ved at klikke på Exception Description(Undtagelsesbeskrivelse) under graffen.

B.1.3 Tilføj SDK'en til projektet i Android Studio

1. I jeres projekt, opret en mappe libraries i roden af jeres projekt. I Launcher projektet vil det se således ud C:\...\launcher\libraries
2. Download Google Analytics SDK'en her: libGoogleAnalyticsServices.jar here <http://cs-cust06-int.cs.aau.dk/attachments/download/85>
3. Placer libGoogleAnalyticsServices.jar i den nye mappe (libraries).
4. I Android Studio skulle den nye mappe samt sdk vises i project viewet. (Hvis ikke tryk CTRL+ALT+Y for at opdatere). Højreklik på libGoogleAnalyticsServices.jar og vælg Add As Library...
5. Vælg jeres "projekt module"(f.eks. launcher) og OK

B.1.4 Implementering af Google Analytics i projekt

Google Analytics er nemt at sætte op. Alt det kræver er at erklære permissions, og oprette en resource fil der beskriver opsætningen og hvilke data der skal logges. Endelig skal man tilføje en linje i onCreate() og onStop() i alle activities der ønskes logdata på.

Tilføj permissions

Tilføj følgende linjer til AndroidManifest.xml

```
1 <uses-permission android:name="android.permission.INTERNET" />
2 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Opret resource fil

1. Opret en ny resource fil i res/values der hedder analytics.xml (Det er vigtigt at denne fil får dette navn)
2. Kopier nedenstående til analytics.xml:

```
1 <?xml version="1.0" encoding="utf-8" ?>
2
3 <resources xmlns:tools="http://schemas.android.com/tools"
4 tools:ignore="TypographyDashes">
5
6 <!--Replace placeholder ID with your tracking ID-->
7 <string name="ga_trackingId">UA-XXXX-Y</string>
8
9 <!--Enable automatic activity tracking-->
10 <bool name="ga_autoActivityTracking">true</bool>
11
12 <!--Enable automatic exception tracking-->
13 <bool name="ga_reportUncaughtExceptions">true</bool>
14
15 </resources>
```

3. Find jeres tracking id nederst på siden her.
4. Erstat placeholderen i analytics.xml (UA-XXXX-Y) med jeres tracking id.

Log activities

INFO: For det mest præcise data anbefaler Google at man aktiverer Analytics i alle sine activities.

For at logge activities skal der tilføjes to linjer i hver activity der skal logges. Dette er et eksempel taget fra Googles egen guide.

```

1 package com.example.app;
2
3 import android.app.Activity;
4
5 import com.google.analytics.tracking.android.EasyTracker;
6
7 /**
8  * An example Activity using Google Analytics and EasyTracker.
9  */
10 public class myTrackedActivity extends Activity {
11     @Override
12     public void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14     }
15
16     @Override
17     public void onStart() {
18         super.onStart();
19         ... // The rest of your onStart() code.
20         EasyTracker.getInstance(this).activityStart(this); // Add this method.
21     }
22
23     @Override
24     public void onStop() {
25         super.onStop();
26         ... // The rest of your onStop() code.
27         EasyTracker.getInstance(this).activityStop(this); // Add this method.
28     }
29 }

```

Exceptions (Optional)

Uncaught exceptions bliver automatisk sendt til Google Analytics ved at have sat `ga_reportUncaughtExceptions` til `true` i `analytics.xml`. Skulle nogen føle det nødvendigt at sende en caught exception så kan det opnås som i følgende eksempel

```

1 try {
2     // Noget kode som maaske vil kaste en exception
3 } catch (Exception e){
4     // Sending the caught exception to Google Analytics
5     // May return null if EasyTracker has not yet been initialized with a
6     // property ID.
7     EasyTracker easyTracker = EasyTracker.getInstance(this);
8
9     // StandardExceptionParser is provided to help get meaningful Exception
10    descriptions.
11    easyTracker.send(MapBuilder
12        .createException(new StandardExceptionParser(this, null) // Context and
13            optional collection of package names
14
15            // to be used in
16            reporting the
17            exception.
18
19            .getDescription(Thread.currentThread().getName(), // The name of
20                the thread on which the exception occurred.

```

```

14         e), // The exception.
15         false) // False
            indicates a fatal exception
16     .build()
17 );
18
19 // Resten af exception haandteringen
20 }

```

B.1.5 Når der debugges / Application afvikles i Emulator

Det kan være en fordel at deaktivere Google Analytics når der debugges eller en implementation testes.

Tilføj følgende til analytics.xml for at kunne kontrollere om der sendes data.

```

1 <!--Enable debug mode - Useful when doing debugging and testing new
   implementation - true = do not send / false = send data -->
2 <bool name="ga_dryRun">true</bool>

```

B.1.6 Tracking ID

Application name	Tracking id
Launcher	UA-48608499-1
Voicegame	UA-48608499-2
Categorymanager	UA-48608499-3
Pictocreator	UA-48608499-4
Pictosearch	UA-48608499-5
Sequence	UA-48608499-6
Timer	UA-48608499-7
Lifestory	UA-48608499-8
Categorygame	UA-48608499-9
Ugeplan	UA-48608499-10
Administration	UA-48608499-11
Pictoreader	UA-48608499-12

Table B.1. Overview of Tracking IDs for the apps

B.2 Google Analytics Crash Reports to Email Guide

This is a step for step guide for the setting up crash reports from Google analytics to email, for the google analytics group of 2016.

1. The first thing to do is to go to the google analytics website at <http://www.google.com/analytics/>
2. Then log in to the google account using the google account information provided.
3. The page should then look like shown below on figure B.1.

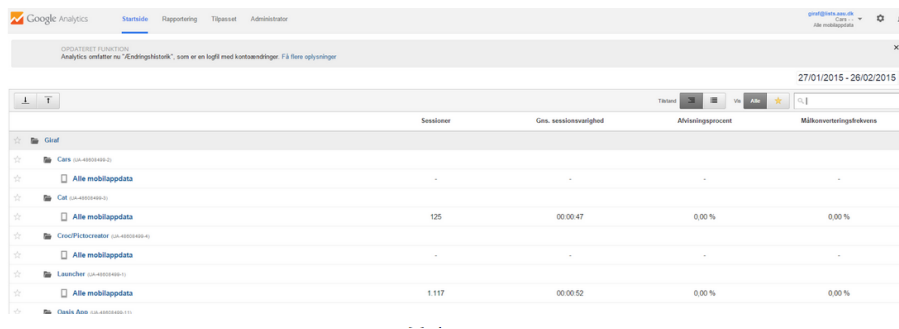


Figure B.1. Main page when logged in to Google analytics

4. Select the app you want to have a crash report from by pressing “Alle mobile appdata” Right below the app.
5. This should then lead you to a page like on figure B.2 the one below.

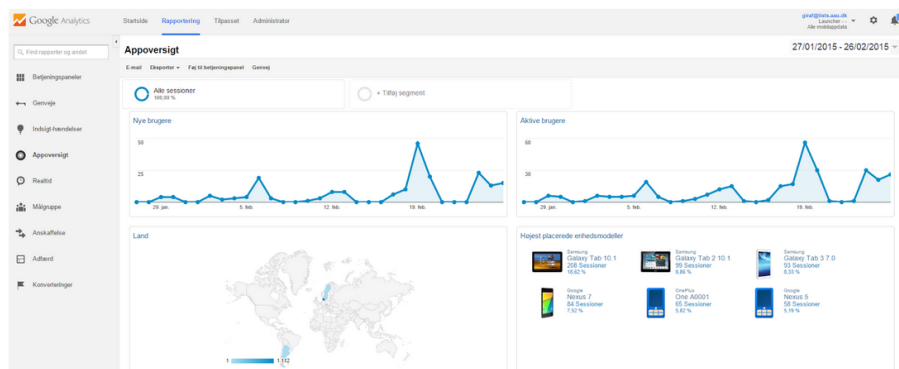


Figure B.2. App overview page

6. At the left most sidebar, press on “Adfærd”/”Behavior” and select “Nedbrud og Undtagelser”/”Crashes and Exceptions” in the dropdown menu.
7. The Page should then look like shown below on figure B.3.

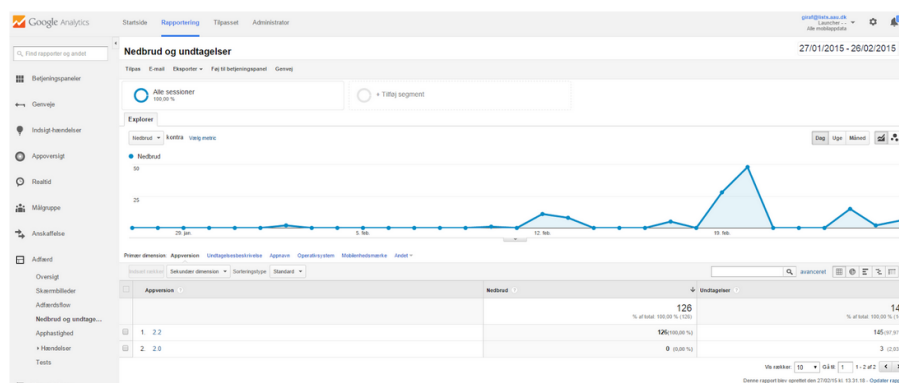


Figure B.3. Crashes and Exceptions page

8. At the bottom of the page click on the newest version of the app.
9. The page should then show all the crashes at the bottom of the screen.

10. After this go to the top right of the page and click on the date interval.
11. This should open a window like the one shown below on figure B.4.

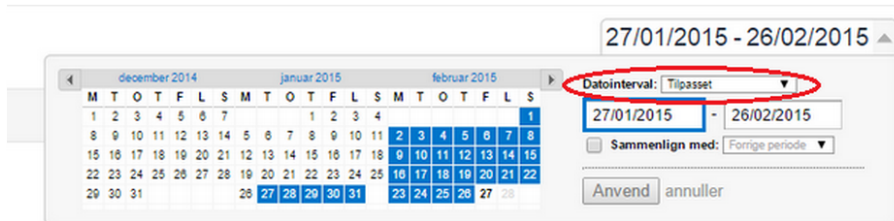


Figure B.4. Calender window

12. In the window select the time interval that you want the crash report to be sent from in the highlighted dropdown menu.
13. Press “Anvend”/”Apply” to close the window.
14. In one of the top menus there is a button named “Email”, press it.

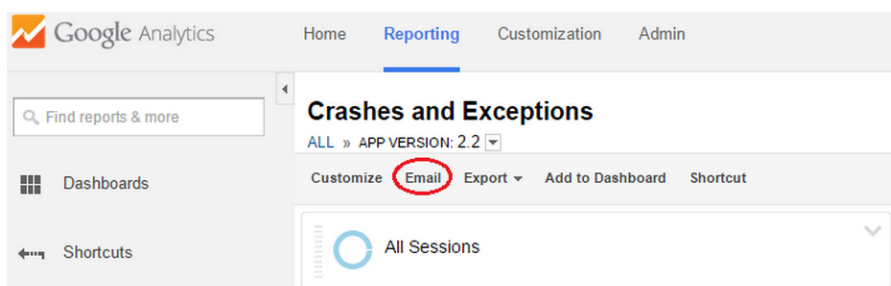


Figure B.5. Email button

15. In the window that opens select whom to send the mail to, what format the crash report should be in, and when the email should be send.

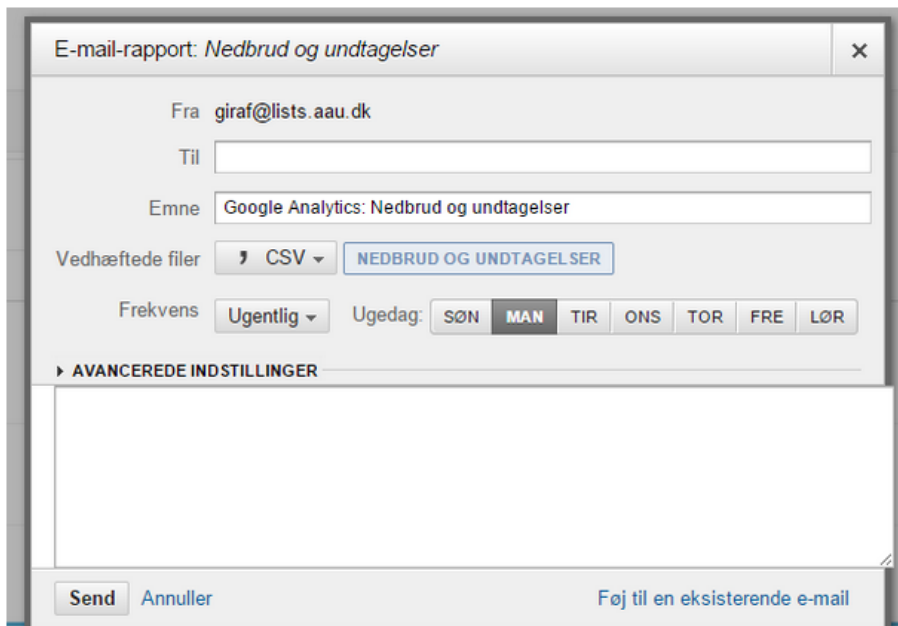


Figure B.6. Mail report window

B.3 How to use Google Play

This section will describe how to use Google Play.

To use the administrative side of Google Play you first and foremost need to be logged in at: <https://play.google.com/apps/publish> using an administrative account.

When logged in you have access to a list of the currently published apps sorted alphabetically on their name (the title of the app). The list also shows the current version, the rating, and when the apps were last updated, as well as a few other pieces of information.

When clicking on the name of an app, you are taken to the shop description, where you can change the description of the app and the various screenshots being used when the app is presented in Google Play. On Figure B.7 an overview of the most important tabs is given. These include: the APK-tab (1), where version management is handled; the shop description-tab (2); the content-classification-tab (3), where the content declaration can be changed, see Section B.3.2; and the price and distribution-tab (4), where the app's availability can be changed. In addition to this, there are other more self-explanatory tabs.

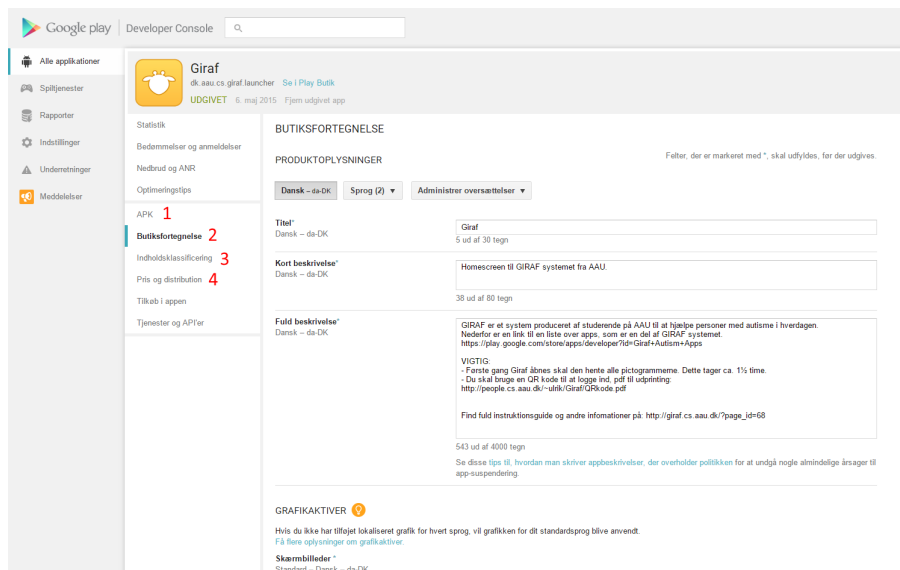


Figure B.7. Main page when logged in to Google analytics

Going to the APK-tab, see Figure B.8, where all APKs for the app can be seen. This is separated into the production-version, a beta-version, and an alpha-version. The production-version is the one shown in Google Play, unless the one browsing the Play store are logged in on an account with test rights to the app. I.e. the production-version (1) might be 25, the beta-version (2) 28, and the alpha-version (3) could be 32. The alpha-version is always newer than the beta-version which in turn is newer than the production-version. An alpha or a beta-version can be promoted to become the production-version, also the alpha-version can be promoted to become a beta-version (4). It is also possible to change a previous version to be the active alpha, beta, or production-version, by first switching to the advanced state (5), and then click the button (6) to move it to the version needed.

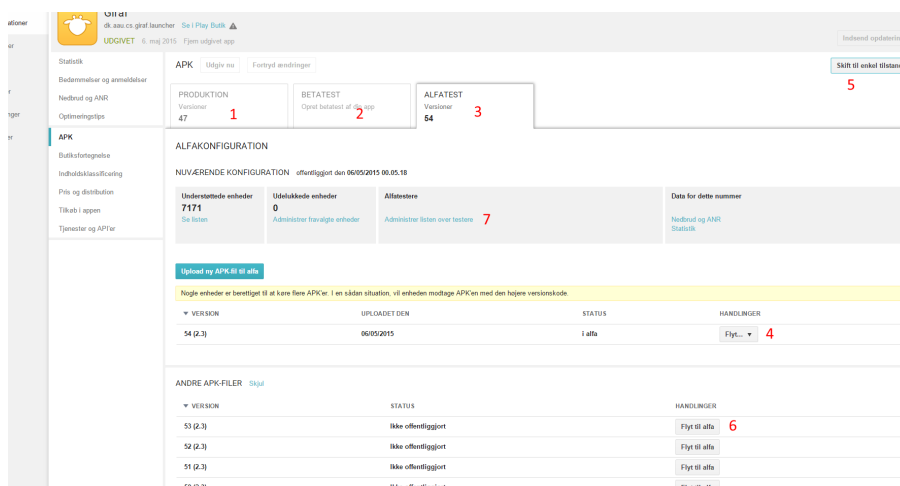


Figure B.8. Main page when logged in to Google analytics

B.3.1 Giving access to alpha and beta-testing

To give others access to alpha and beta versions, they need to be added to a Google Group that is assigned as a tester, either alpha or beta, on the app. The group ‘alfa-og-beta-testing-af-giraf@googlegroups.com’ was made for the purpose. When a person have been added to the group they will then have to go to ‘<https://play.google.com/apps/testing/dk.aau.cs..giraf.PackageNameOfApp>’ and accept to become tester on that app. Then if that person is logged in on a tablet, they will automatically download the alpha or beta-version of the app. If a new app is added to Google Play, then the testing group will have to be added to the app as well. This is done by pressing the administrate the list of tester button (7) on Figure refVersionHandlingGooglePlay, and then add the group.

Likewise there is a group specifically for beta testers, ‘beta-testing-af-giraf@googlegroups.com’, which grants beta versions but no alpha versions. The procedure for this group is identical to the other. It is to be noted that it is only necessary to be a member of the first group, alfa-og-beta-testing-af-giraf, to have both alpha and beta testing available, though you will always use only the newest version, so if an alpha version is uploaded, then any beta version you are using will be replaced on the tablet.

B.3.2 Content Classification

The content classification is a rating that shows which age groups are allowed to see the app in Google Play. In Denmark and the rest of Europe, except for Germany, the rating to follow is the PEGI-rating [PEG, 2015]. Google automatically assigns a rating to the app, after the developer, us, fills out a survey, where we declare what the app contains of content. Here it is important to distinguish between content that has been made by the developers and user-generated-content. The pictograms used in the apps we release can be counted as user-generated for the purpose of the content classification. This allows us to obtain a PEGI 3 rating, which means our apps are ‘safe for use’ by anyone from the age of three and upwards.

B.4 Android Studio Setup

1. Download and intall git from <http://git-scm.com/>
2. Download Android Studio 1.0.1 and install it <http://tools.android.com/download/studio/canary/1-0-1>
 - a) If you do not have Java JDK get it here <http://www.oracle.com/technetwork/java/javase/download>
3. Download Stand alone SDK here <http://developer.android.com/sdk/installing/index.html?pkg=tools>
4. Open Android SDK Manager
5. Install these packages
 - a) Android SDK Tools
 - b) Android SDK Platform-tools
 - c) Android SDK Build-tools ver 19.0.1
 - d) Android API 15, 17 og 19
 - e) Android Support Repository
 - f) Android Support Library

- g) Google USB Driver
- 6. Download gradle 2.2.1 here <https://services.gradle.org/distributions/gradle-2.2.1-all.zip>
- 7. Go to <http://cs-cust06-int.cs.aau.dk/git>, to see the list of projects you have access to, using your student login
- 8. In Android Studio, choose Check out from Version Control, using Git with path: <http://cs-cust06-int.cs.aau.dk/git/project-name>
- 9. Download a project through AS using option git.
- 10. A good start could be the launcher
- 11. Point Projects Gradle to your downloaded gradle 2.2.1 folder, using local distribution Point Android Studio to installed SDK
- 12. In the project folder run (using CMD or Git Bash)
 - a) git pull
- 13. From here you should be able to complete the import
- 14. The first time Android Studio start up the project, there could be some first-time processing, which could take quite a long time, like indexing. This will be a one-time only processing.
- 15. Try to compile a project

Troubleshooting; If you get error: bad class file magic (cafebabe) or version (0034.0000)

Solution: Make Sure you use Java JDK 7

B.5 App and library renaming in Android Studio

1. Open the app or library project in Android Studio
2. Under src -> main -> java the path shown is dk.aau.cs.giraf.CurrentName
 - a) Rightclick above path and under refactor click rename and change CurrentName to NewName.
3. After renaming the above, press Ctrl+Shift+R (or Cmd+Shift+R on Mac) and search for giraf.CurrentName and replace it with giraf.NewName.
4. In case the app or library calls other apps or libraries that are being renamed, then do step 3 again for each case, replacing CurrentName and NewName with the ones from the app or library being called.
5. Commit the changes to Git and Jenkins will attempt to build it.

B.6 How to implement the Pictosearch library

1. a) **add:** compile(group: 'dk.aau.cs.giraf', name: 'pictosearch', version: '2.0.6', ext: 'aar') to dependencies
2. Update Intent
 - a) **add:** import dk.aau.cs.giraf.pictosearch.PictoAdminMain;
 - b) **replace:** intent.setComponent(new ComponentName("dk.aau.cs.giraf.pictosearch", "dk.aau.cs.giraf.pictosearch.PictoAdminMain"));
 - c) **with:** intent = new Intent(getApplication(), PictoAdminMain.class);

B.7 Javadocs

Javadocs is a comment generating tool. It should already be installed in your Android Studio Application and be ready for use. Most of it is auto-generated.

Here is a link to the in-depth <http://en.wikipedia.org/wiki/Javadoc>

Why do we use this? If used consistently, we can generate a html page with documentation for all of our code.

If you wish to generate this html page for yourself, you can do this inside Android Studio in just seconds.

1. Open the Android Studio Project
2. Open the tools menu in the bar
3. Pick 'Generate JavaDoc...'

From here on you just have to specify details such as output folder and etc.

B.7.1 Guidelines

Javadocs should be used for all public functions and classes. The aim is to cover as much of the code as possible. However, it is not recommended that you go out of your way commenting code already written, unless you're refactoring.

How to use

A Javadoc comment is set off from code by standard multi-line comment tags `/**` and `*/`. The opening tag has an extra asterisk, as in `/**`.

Simply put this opening tag right above the code block in question, and Javadocs will auto-generate a comment-stub for the code block. Then you fill in the blanks. In short:

1. Provide a short description of the functionality or role of the code block.
2. Elaborate on any parameters or otherwise auto-generated stubs made by Javadocs.

B.7.2 Javadocs Example

Self-explanatory comment example ripped from the official wiki page. Notice that a description longer than the initial short line isn't mandatory.

```
1  /**
2   * Short one line description.                (1)
3   * <p>
4   * Longer description. If there were any, it would be [2]
5   * here.
6   * <p>
7   * And even more explanations to follow in consecutive
8   * paragraphs separated by HTML paragraph breaks.
9   *
10  * @param variable Description text text text.    (3)
11  * @return Description text text text.
12  */
13  public int methodName (...) {
```

```

14  // method body with a return statement
15  }

```

B.8 Dependencies wiki

B.8.1 App to library dependencies

This shows the dependencies between Apps and libraries.

NOTE: ALL apps are dependent on 'giraf-component', 'oasis-lib', and 'local-db'. They were left out of the graph, to give a better overview of the rest of the dependencies.

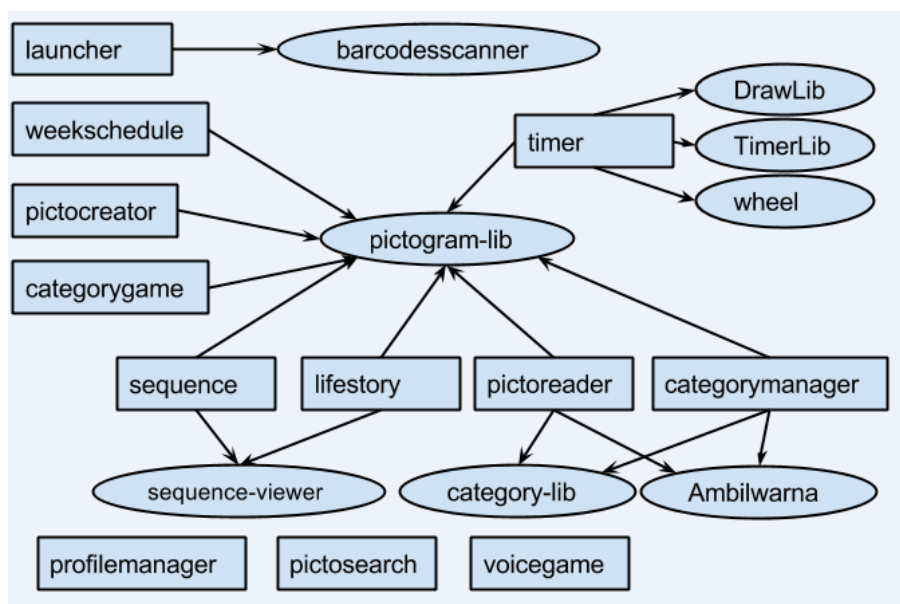


Figure B.9.

B.8.2 App to App dependencies

This shows the dependencies between Apps and other Apps. The link dependencies are caused by buttons in the Apps that tries to open other Apps. The uncertain dependencies are only because the Apps crash on start-up.

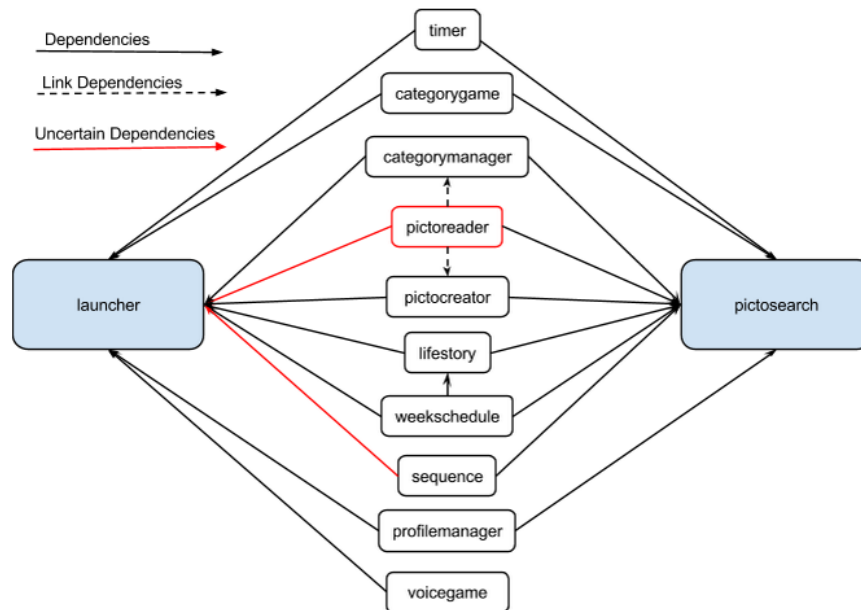


Figure B.10.

B.8.3 Library to library dependencies

This shows the dependencies between libraries and other libraries.

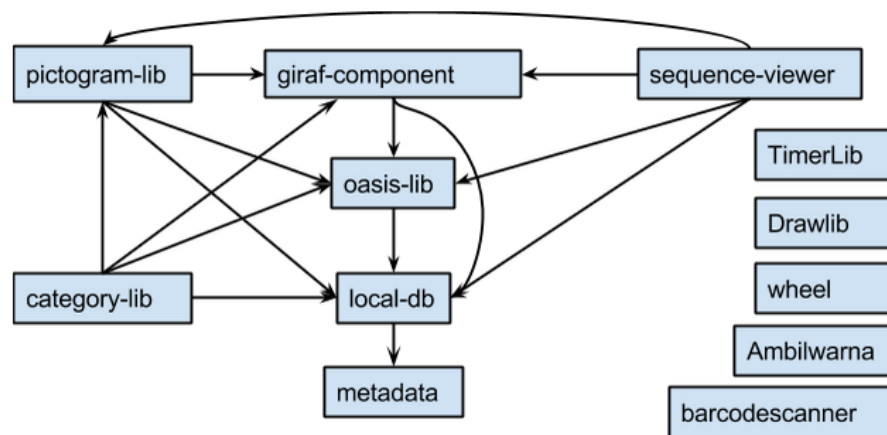


Figure B.11.