

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

TDT4290 — CUSTOMER DRIVEN PROJECT
GROUP 2

**BLOPP - Development of a prototype system for
treatment of asthmatic children, using Android
and Karotz**

Jørgen AABERG
Esben AARSETH
Eirik Skjeggestad DALE
Aleksander GISVOLD
Yngve SVALESTUEN

May 22, 2013

Abstract

This project aimed to create three applications to motivate and remind asthmatic children to take their medication. When children are on a medication plan, taking the medicine might be boring or stressful because they are reminded of their asthma, are disturbed in their routine, or that the medication process itself is scary. The use of an appealing figure like the rabbit robot Karotz provides a way to avoid some of these concerns. In combination with a reminder and distraction application and an adult information and settings application, the complete system could help alleviate the burden of medicating asthmatic children.

The three applications were developed on two different platforms. The guardian application for configuration, teaching and viewing a log, and a children application for teaching, reminding and distracting during treatment were developed for the Android platform. A second application for reminding and distracting the children during treatment was made for Karotz. Through the agile software development technique SCRUM, the project team completed five sprints of iterative study, planning, programming, adaption and testing. The Android applications are written in Java, while the Karotz application is written in JavaScript. A central database is written in MySQL with PHP sites for access through the internet protocol HTTP.

The prototype system is developed for Sykehusapotekene i Midt-Norge as a part of NTNU's course TDT4290 — Customer Driven Project.

Keywords: *BLOPP, Asthma, Gamification, Android, Karotz, SCRUM, Software development*

Jørgen Aaberg

Esben Aarseth

Eirik Skjeggestad Dale

Aleksander Gisvold

Yngve Svalstuen

Contents

1	Introduction	1
1.1	Project Information	1
1.1.1	Project Name	1
1.1.2	Background	1
1.1.3	The task	2
1.1.4	Measurement of project effort	4
1.1.5	General terms	4
1.1.6	Planned effort	4
1.1.7	Schedule of results	4
1.1.8	Report Outline	4
1.2	Customer Information	4
1.2.1	Sponsor	6
1.2.2	Partners	6
1.2.3	Customer contacts	6
2	Project Management	9
2.1	Members	9
2.2	Roles	9
2.3	Responsibilities among roles	10
2.4	Weekly schedule	12
2.5	Work Plan	12
2.5.1	Phases	12
2.5.2	Activities	13
2.5.3	Person-hours per activity and phase	13
2.5.4	Gantt Diagram	13
2.6	Risk Analysis	14
2.6.1	Internal Risks	14
2.6.2	External Risks	14
2.6.3	SWOT analysis	14
2.7	Quality Assurance	19
2.7.1	Language	19
2.7.2	Customer Meeting	20
2.7.3	Advisor Meeting	20
3	Preliminary Studies	22
3.1	Children with asthma	22
3.1.1	Traffic Light Classification of Asthma Condition	23
3.2	Parents with children affected by asthma	24
3.3	The concept of gamification	25
3.4	Karotz	27

3.4.1 Application Platform	28
3.5 Pollen forecast	29
3.6 Design workshop	30
3.6.1 Results	30
3.6.2 What was used in the further development	37
3.7 Frameworks used in the Project	37
3.7.1 Programming Languages, Message Formats and File Formats	37
3.7.2 Database	39
3.7.3 Extra Tools used in the Project	41
3.7.4 Design Principles	44
3.8 Software Architecture	45
3.8.1 MVC - Model View Controller	45
3.8.2 4+1 View Model	45
3.9 Privacy and security	47
4 Development Methodology	50
4.1 Waterfall vs Agile development	50
4.1.1 The Waterfall Method	51
4.1.2 Scrum	52
4.1.3 Kanban	52
4.2 Choice of methodology	54
4.2.1 Sprints	55
5 Requirement Specifications	59
5.1 Use Cases	59
5.1.1 Actors	59
5.1.2 Textual Use Cases for GAPP	61
5.1.3 Textual Use Cases for CAPP	62
5.2 Functional Requirements	62
5.2.1 GAPP - Guardian Application	62
5.2.2 CAPP - Children's Application	71
5.2.3 Karotz Application	72
6 System Design	73
6.1 Architectural Description	73
6.2 Software architecture	73
6.2.1 Logical View	75
6.2.2 Development View	76
6.2.3 Process View	79
6.3 Architecture Rationale	84
6.4 Database	84
6.4.1 Database Implementation	84
6.4.2 Database Access Layer	86
7 Overall Test Plan	88
7.1 Test methods	88
7.1.1 Black-box testing	88
7.1.2 White-box testing	89
7.2 Test levels	89
7.2.1 Unit testing	89
7.2.2 Module testing	89

7.2.3	Integration and System testing	89
7.3	Testing approach	90
8	Sprint 1	93
8.1	Sprint Plan	93
8.2	Sprint backlog	94
8.3	Design and Implementation	94
8.3.1	User Interface Layer	94
8.3.2	Application Logic Layer	95
8.3.3	Data Persistence Layer	95
8.4	Testing and Results	95
8.4.1	Testing	95
8.4.2	Results	95
8.5	Sprint Retrospective	95
8.5.1	Sprint Burndown Chart	96
9	Sprint 2	102
9.1	Sprint Plan	102
9.2	Sprint backlog	103
9.3	Design and Implementation	103
9.3.1	User Interface Layer	104
9.3.2	Data Persistence Layer	105
9.3.3	Database Access Layer	105
9.4	Testing and Results	106
9.4.1	Testing	106
9.4.2	Results	106
9.5	Sprint Retrospective	106
9.5.1	Sprint Burndown Chart	109
10	Sprint 3	113
10.1	Sprint Plan	113
10.2	Sprint backlog	114
10.3	Design and Implementation	114
10.3.1	User Interface Layer	114
10.3.2	Application Logic Layer	114
10.3.3	Data Persistence Layer	116
10.3.4	Karotz	116
10.4	Testing and Results	116
10.4.1	Testing	116
10.4.2	Results	117
10.5	Sprint Retrospective	121
10.5.1	Sprint Burndown Chart	121
11	Sprint 4	126
11.1	Sprint Plan	126
11.1.1	CAPP	127
11.1.2	GAPP	127
11.2	Sprint backlog	127
11.3	Design and Implementation	129
11.3.1	User Interface Layer	129
11.3.2	Application Logic Layer	130

11.3.3 Data Persistence Layer	130
11.3.4 Karotz	131
11.4 Testing and Results	132
11.4.1 Testing	132
11.4.2 Results	132
11.5 Sprint Retrospective	133
11.5.1 What went well?	133
11.5.2 What shall we start doing?	133
11.5.3 What could have gone better?	133
11.5.4 What should we stop doing?	133
11.5.5 Sprint Burndown Chart	134
12 Sprint 5	137
12.1 Sprint Plan	137
12.2 Sprint backlog	137
12.3 Design and Implementation	139
12.3.1 User Interface Layer	139
12.3.2 Application Logic Layer	139
12.3.3 Data Persistence Layer	140
12.4 Testing and Results	140
12.4.1 Testing	140
12.4.2 Results	159
12.5 Sprint Retrospective	160
12.5.1 What went well?	160
12.5.2 What shall we start doing?	160
12.5.3 What could have gone better?	160
12.5.4 What should we stop doing?	160
12.5.5 Sprint Burndown Chart	160
12.5.6 Screenshots	163
12.5.7 GAPP - Screenshots	163
13 Usability Testing	168
13.1 What is usability testing	168
13.2 How to do usability testing	168
13.3 Usability testing in our project	169
13.3.1 Paper prototype test	169
13.3.2 Usability testing of the distraction	172
14 Further Work	179
14.1 Improvements	179
14.1.1 Wifi access and caching of database records	179
14.1.2 Security and privacy	180
14.1.3 Rewardsystem	180
14.1.4 Distraction sequence for children	181
14.1.5 User testing of the guardian application	181
14.1.6 Web application	181
14.1.7 Support for more children	182
14.2 Ideas and minor improvements	182

15 Evaluation	185
15.1 Work Process	185
15.1.1 Development Methodology	185
15.1.2 Development Process	187
15.1.3 Work Load	187
15.2 The Final Product	188
15.3 Functional Requirements completed	189
15.4 Concluding Remarks	191
Appendices	198
A Paper Prototype	199
A.1 About paper prototyping	199
A.2 Usability Testing with a paper prototype	199
A.2.1 Testprocedures	199
A.2.2 The testpersons tasks	200
A.2.3 The testgroups tasks	200
B Document templates	202
B.1 Agenda	202
B.2 Status reports	202
C Karotz Manuscript	204
D Coding Templates	210
D.1 Coding Style	210
D.1.1 Package conventions	210
D.1.2 Indentation	210
D.1.3 Curly Brackets	210
D.1.4 Naming Conventions	211
D.1.5 Android views	211
D.1.6 Code Examples	211
D.1.7 LaTeX folder structure	212
E Class diagram	215
E.1 Class Diagram GAPP	215
E.2 CAPP - Children Application	221
E.3 Karotz Application	227
F Article from Adressa	230
G Abbreviations	234

List of Figures

2.1	Gantt project overview	14
3.1	Ventoline	23
3.2	Flutide	23
3.3	A Nebulizer machine	24
3.4	Karotz: A bunny-shaped robot	27
3.5	Flatnanoz	28
3.6	Nanoztag	28
3.7	Main menu view from design workshop	32
3.8	Change health state view from design workshop	32
3.9	Change health state as pop up from design workshop	33
3.10	Start medication view from design workshop	33
3.11	Pick child view from design workshop	34
3.12	Distraction view from design workshop	34
3.13	Distraction view 2 from design workshop	35
3.14	Rewards view from design workshop	35
3.15	Log view from design workshop	36
3.16	phpMyAdmin screenshot	40
3.17	Postman screenshot	43
3.18	Graphical view of MVC	46
4.1	Graphical representation of the waterfall method[2]	51
4.2	Graphical representation of the SCRUM method[1]	54
5.1	Use Case diagram for GAPP	60
5.2	Use Case diagram for CAPP	60
6.1	Logical View for the system	75
6.2	GAPP package diagram	77
6.3	GAPP package diagram	78
6.4	Karotz package diagram	79
6.5	Sequence diagram for medication completed	80
6.6	Sequence diagram for changing medication plan	81
6.7	Sequence diagram for notification and medication on Karotz	82
6.8	Implemented Database Architecture	85
8.1	Sprint 1 burndown chart	97
9.1	Sprint 2 burndown chart	109
10.1	Sprint 3 burndown chart	122

11.1 Sprint 4 burndown chart	134
12.1 Sprint 5 burndown chart	161
12.2 GAPP main menu	163
12.3 Available plans in GAPP	163
12.4 A medication plan in GAPP	164
12.5 Register treatment in GAPP	164
12.6 Choose among medicines to view more information	164
12.7 Medicine specific information	164
12.8 Medicine log in GAPP	165
12.9 Main menu in CAPP	166
12.10 Start a treatment in CAPP	166
12.11 Amount of stars collected in CAPP	166
13.1 Paper prototype	170
13.2 Usability test CAPP distraction	174
13.3 Usability test Karotz distraction	175
13.4 Usability test CAPP instructions	176
D.1 Java Classes	212
D.2 Java Interfaces	213
E.1 Activities in GAPP	216
E.2 Activity interaction GAPP	217
E.3 JSON parsers in GAPP	218
E.4 JSON posters in GAPP	219
E.5 JSON models in GAPP	220
E.6 Models in GAPP	220
E.7 Adapters in GAPP	221
E.8 Other classes in GAPP	222
E.9 Activities in CAPP	223
E.10 Adapters in CAPP	224
E.11 Available plans in GAPP	224
E.12 JsonModels in CAPP	224
E.13 JsonParsers in CAPP	225
E.14 JsonPosters in CAPP	226
E.15 Misc classes in CAPP	226
E.16 Services in CAPP	227
E.17 Class diagram for the Karotz application	228

List of Tables

1.1	High level functional requirements	3
1.2	Chapters and their respective description	5
1.3	Customer contacts	6
1.4	Stakeholders	8
2.1	Members of developer team	10
2.2	Internal Risks	15
2.3	External Risks	16
3.1	Purpose of the 4+1 View Model	47
5.1	Actors within the system	61
5.2	Use Case 1 for GAPP, log in	62
5.3	Use Case 2 for GAPP, change status	63
5.4	Use Case 3 for GAPP, log	63
5.5	Use Case 4 for GAPP, Pollen feed	64
5.6	Use Case 5 for GAPP, guidelines	65
5.7	Use Case 6 for GAPP, reward	66
5.8	Use Case 7 for GAPP, medication settings	66
5.9	Use Case 8 for GAPP, register medication	67
5.10	Use Case 9 for GAPP, reminder	68
5.11	Use Case 1 for CAPP, log in	68
5.12	Use Case 2 for CAPP, reward	69
5.13	Use Case 3 for CAPP, start treatment	69
5.14	Use Case 4 for CAPP, reward	70
7.1	Test template	90
7.2	List of tests	91
8.1	Backlog for sprint 1	99
8.2	Unit test 1.1, GAPP GUI	100
8.3	Unit test 1.2, CAPP GUI	100
8.4	Sprint 1 burndown chart	101
9.1	Backlog for sprint 2	103
9.2	Unit test 2.1, CAPP distraction sequence	107
9.3	Unit test 2.2, database connection	107
9.4	Unit test 2.3, SQL queries	108
9.5	Sprint burndown chart, Sprint 2	111
10.1	Backlog for sprint 3	115

10.2 Unit test 3.1: Alarm when turned off	117
10.3 Unit test 3.2: Calendar colors	118
10.4 Unit test 3.3: Karotz Notification	118
10.5 Unit test 3.4: Karotz distraction	119
10.6 Unit test 3.5: Several doses in a medication	120
10.7 Sprint Retrospective, Sprint 3	123
11.1 Backlog for sprint 4	128
11.2 Unit test 4.1: instructions	132
11.3 Sprint Retrospective, Sprint 4	135
12.1 Backlog for sprint 5	138
12.2 Unit test 5.1, add_child.php	141
12.3 Unit test 5.2, add_plan_dose.php	142
12.4 Unit test 5.3, dose_is_taken.php	143
12.5 Unit test 5.4, get_available_child_states.php	144
12.6 Unit test 5.5, get_child.php	145
12.7 Unit test 5.6, get_child_state.php	146
12.8 Unit test 5.7, get_doses_for_current_state.php	147
12.9 Unit test 5.8, get_instructions.php	148
12.10Unit test 5.9, get_log_days_for_child.php	149
12.11Unit test 5.10, get_log_for_child.php	150
12.12Unit test 5.11, get_plan.php	151
12.13Unit test 5.12, register_medicine_taken.php	152
12.14Unit test 5.13, remove_plan_dose.php	153
12.15Unit test 5.14, remove_plan_medicine_at_time.php	154
12.16Unit test 5.15, set_child_state.php	155
12.17USABILITY5.1	156
12.18INTEGRATION5.1	157
12.19INTEGRATION5.2	158
12.20INTEGRATION5.3	159
12.21Sprint Retrospective, Sprint 5	162
13.1 The tasks for CAPP	171
13.2 The tasks for GAPP	171
15.1 Functional requirements completion	190
B.1 Status reports	203
C.2 Manuscript for the Karotz	207
C.1 Manuscript actions for Karotz	208
D.1 Naming convention	211

Chapter 1

Introduction

This chapter contains a brief introduction to the project and the layout of the report. It gives an overview of project goals, and the documentation of the development process. This introduction also explains the background for the given project and how the project's success is measured.

In Section 1.1 we give details on the project, including project name, background, task, terms, planned effort and result schedule.

In Section 1.2 we give detailed customer information including information on the sponsor, partners, customer contacts, project group, and an overview of affiliates. The affiliates include a table of stakeholders.

1.1 Project Information

1.1.1 Project Name

The name of this project is “BLOPP”, and was decided by the customer. BLOPP stands for “Barns LegemiddelOPPlevelse” (“Children’s experience with medication”).

1.1.2 Background

Many children today have to take inhalation medicines because of chronic or acute lung disease such as asthma. Children often find it difficult to use the medication correctly, boring or even scary to take them, which means they might object or forget to take them. Parents also sometimes apply the medication incorrectly, apply the wrong treatment, or even forget to give the medication to their children. This may lead to reduced effect of the medication, and the lung disease may worsen and last longer, causing increased pressure on the public health services, increased health related cost and lost working hours for the parents.

1.1.3 The task

Our task was to implement two Android applications, one application for the parents, Guardian Application (GAPP), and one application for children, Children Application (CAPP). In addition, an application for the Karotz platform should be created to assist and to an extent substitute the GAPP and CAPP mobile applications.

The high level functional requirements for these applications are to be found in Table 1.1. The functional requirements for all applications are described in more detail in Section 5.2

#	Description
GHR1	The application must alert the parent(s) when it is time for a medication/treatment for their child.
GHR2	The application must log the health status of their child, according to section 3.1.1.
GHR3	The application should log pollen casts for the area the child is in, and which medications were taken each day.
GHR4	The application must store medical plans for their child. These plans concern asthma medications, and contains which medicines should be taken at which times.
GHR5	The application must provide instructions on how to use different medications. These instructions may be pictures or text, provided by NAAF.
CHR1	The application should distract the children during a treatment.
CHR2	The application should gamify their experience with medication.
KHR1	The application should alert children and parent(s) when it is time for a medication/treatment for the children.
KHR2	The application should distract the children during a treatment.
KHR3	The application should encourage children to take medication through interactivity and gamification.

Table 1.1: High level functional requirements. GHR: GAPP requirement, CHR: CAPP requirement, KHR: karotz application requirement

1.1.4 Measurement of project effort

The customer was seeking a documented prototype of a system which could be used for future development and for getting additional funding for further development of the project. The customer wanted the prototype tested on children suffering from diseases causing breathing problems and their parents, in order to determine whether or not such a system was an adequate solution to the problem. The system should be compatible with Android v4.0 or newer versions, and should be intuitive to use. The resulting prototype should be well documented to ensure that further development would be able to continue development after the end of the project.

1.1.5 General terms

We were to make two applications for Android devices and one for Karotz. Originally the customer wanted the smart phone applications made for iOS, but since we did not have Apple computers and iPhones, and the customer did not have the funding to provide them, we switched to the Android platform. We had at our disposal a Karotz robot with a yellow and a green Nanoz controller, a Github repository, an AgileZen board and could request an Android tablet to be used for testing if necessary.

1.1.6 Planned effort

The course description states an expected effort of 25 hours per week per student. The course lasted for 13 weeks, resulting in a total expected effort of 325 hours per student, at a total of 1625 for the team altogether.

1.1.7 Schedule of results

The applications were scheduled to be completed within November 10th. The time we had left after this, was used for fixing critical errors, and completing the report.

1.1.8 Report Outline

The report is outlined in Table 1.2.

1.2 Customer Information

The customer of this project is "Sykehusapotekene i Midt-Norge".

1.2.1 Sponsor

The sponsor of this project is Extrastiftelsen.

1.2.2 Partners

The Norwegian University of Technology and Science are partner in this project. Norges Astma og Allergiforbund (NAAF) has also been included in the work, both for feedback and helpful information. Table 1.3 shows relevant contacts for the customer.

1.2.3 Customer contacts

Table 1.3 gives an overview of the contact information for the customers.

Chapter	Description
Chapter 1: Introduction	Contains a short description of the project, its goals and purpose and what the report consists of.
Chapter 2: Project Management	Contains a description of how the group is organized and what responsibilities lies on each of the group members. A risk analysis for the project is also included in this chapter, as well as the work plan which describes the different phases, activities and a Gantt Diagram for the project. Quality Assurance techniques are also discussed, which include meetings, coding templates and document templates.
Chapter 3: Preliminary Studies	Contains a documentation of the preliminary studies done ahead of the implementation of the applications, including a report of the design workshop done early in development, development methodology, frameworks and tools used in the project, the Karotz platform and information about asthma.
Chapter 4: Development Methodology	Contains description and discussion about the various development methods considered for the project, and an analysis of SCRUM, the chosen methodology.
Chapter 5: Requirement Specifications	Contains an overview of the requirement specifications for the system, through use cases and functional requirements.
Chapter 6: System Design	Contains a collection of requirements and design choices for the prototype, including use cases, architectural description and documentation of the database.
Chapter 7: Overall Test Plan	Describes how the team will do testing throughout the project.
Chapter 8-12: Sprint 1-Sprint 5	Contains the goals, backlogs, test tables, results and reviews for the respective sprints.
Chapter 13: Usability Testing	Contains a description of usability testing in general, and reports and discussion from the usability tests done in the project.
Chapter 14: Further Work	Contains a description of what has been implemented and what the next logical steps are based on the current state of the system.
Chapter 15: Evaluation	Contains the evaluation and description of how the project was executed.

Table 1.2: Chapters and their respective description

Name	Email
Ole Andreas Alsos	oleanda@idi.ntnu.no
Elin Høien	elin@hoien.no
Marikken Høiseth	marikken.hoiseth@ntnu.no
Hanne Linander	hanne.linander@gmail.com

Table 1.3: Customer contacts

Project Group

- Cand Pharm Elin Bergene, Sykehusapotekene i Midt-Norge (Hospital pharmacies in Central Norway)
- PhD Ole Andreas Alsos, Norsk Senter for Elektronisk Pasientjournal (NSEP) (Norwegian Center for Electronic Patient Journal) and Institute for Computer Science (IDI), NTNU (Ph.D as of 2011). Ole is also working part time at BEKK Consulting.
- Scholarship Marikken Høiseth, Institute for Product Design (IPD), NTNU.
- Bo Alexander Gleditsch, communication advisor NAAF.
- Rose Lyngra, senior advisor NAAF.

Affiliates

The project is in close collaboration with the following affiliates:

- Sykehusapotekene i Midt-Norge (SHAP) will be a test arena for the result of the project. They will work further with the results.
- Norsk Senter for Elektronisk Pasientjournal (NSEP). NTNUs activities within health informatics is gathered at NSEP. The project will take advantage of the academic community and the infrastructure at NSEP (offices and usability lab).
- Department for product design (IPD) at NTNU will consult upon design.
- Department for computer and information science arranges the course and will provide an advisor for the group, Tobias B. Iversen.
- Norges Astma og AllergiForbund (NAAF). The project has been created by NAAF. NAAF will provide expertise about the user groups of the final applications. NAAF will work further with the results of the project.

Stakeholders

A stakeholder is a person, group or an organization that has interest in a project. The different stakeholders of this project are listed in Table 1.4. This table also contains a short rationale for why each party has been listed, and what their main concerns for the applications are.

Stakeholder	Rationale
NAAF	Wants to see whether this is a possible solution to make it easier for children to take their medicine. Also interested in proof-of-concept.
Sykehusapotekene i Midt-Norge	Cooperates with NAAF to find out if there is an easier way to make children take their medicine.
Developers	Wants the applications to be a success, as it is their work. The level of success also affects the degree in one of their courses.
NTNU	Wants the project to be a success to front the research that is done by the university.
Children diagnosed with asthma	Needs something to make it easier to go through with each treatment.
Parents of children diagnosed with asthma	Needs instructions on how to use medicines correctly. Needs reminders about when to give their children their medicines. Needs an organized way to see what medicines have been taken earlier. Wants their children to suffer less during medication, and be happier about taking their medicine.
Extrastiftelsen	Main funder of the project.

Table 1.4: Stakeholders

Chapter 2

Project Management

2.1 Members

Table 2.1 shows an overview of the names, email addresses and phone numbers for the members of the developer team.

2.2 Roles

Early in the planning phase, on August 24th 2012, the group held a meeting to distribute team roles for the project. We discussed which parts were needed for a system fitting the description, and we decided on four main parts: GUI, back-end, database and Karotz. In addition, proper development requires testing so we saw the need for a person in charge of testing, and a person in charge of general quality assurance. We also needed a high-level system architect who could keep the project well-structured and easily maintainable, especially considering that the system spans over at least three different subsystems (database, Karotz and Android applications). Since the group was to keep close contact with both the customer and the group advisor, responsibilities were assigned to these roles as well. There would also be a need to write reports for meetings with these third parties, so a secretary was necessary. At last, the group had already decided to use an agile development model, so a person in charge of this was needed as well (“Scrum master”).

We have identified the following roles for the project.

- Test Master - Eirik
- Scrum Master - Aleksander
- Customer Contact - Aleksander
- Advisor Contact - Esben
- Document Owner - Yngve
- Secretary - Jørgen
- Karotz Developer - Yngve
- Database Manager - Yngve
- System Architect - Esben
- Quality Assurance manager - Jørgen

Name	Email	Phone number
Esben Aarseth	esbena@stud.ntnu.no	48062321
Aleksander Gisvold	aleksg@stud.ntnu.no	46692443
Jørgen Aaberg	jorgeaab@stud.ntnu.no	98866209
Eirik Skjeggestad Dale	eiriksd@stud.ntnu.no	90138539
Yngve Svalstuen	yngvesva@stud.ntnu.no	99101640

Table 2.1: Members of developer team

2.3 Responsibilities among roles

In the following section all roles and their responsibilities are explained.

Test Master The test master is responsible for developing a test plan, initiate testing and follow up on test results. The test master will have the last vote in whether a test is passed or not. The test master will be responsible for making sure all parts of testing is done, including unit testing, integration testing, system testing and acceptance testing with the customer.

Scrum Master Scrum master is accountable for removing impediments to the ability of the team to deliver the sprint goal. The scrum master is no team leader, but is a kind of buffer between the development team and distracting influences. The scrum master is responsible for ensuring that the scrum process is used as intended.

Customer Contact The customer contact is responsible of all contact with the customer outside of the customer meetings. This includes sending meeting invitations, clarifying questions outside of customer meetings and other inquiries to the customer. The customer contact works as a single-point two-way communicator, to reduce amount of communication points.

Advisor Contact The advisor contact is responsible of all contact with the advisor outside of the advisor meetings. Including sending meeting invitations, clarifying questions outside of advisor meetings and other inquiries to the advisor. Advisor contact works as a single-point two-way communicator, to reduce amount of communication points.

Document Owner The Document owner is responsible of finding a suitable tool for writing the report in L^AT_EX, and finding solutions with problems regarding L^AT_EX. The document owner is also responsible for making sure all the correct documents is added to the report, and will let the group know if something is missing.

Secretary Secretary is responsible for taking notes during all internal, customer and advisor meetings. Meeting reports should meet a specific standard, given by the project compendium. It is the responsibility of the meeting report master to ensure this standard is followed.

Karotz Developer The robot bunny, named Karotz, has an API for implementing features for controlling the robot with an application. The Karotz developer is responsible for developing the Karotz specific part of the system.

Database manager The database manager is responsible for selecting a suitable database tool for the applications. The role also includes the responsibility of managing the database architecture and connections towards the database.

System architect The system architect is responsible of the overall architecture of the source code. The architect has final vote in decisions regarding architecture specific problems.

Quality assurance manager The quality assurance manager has the overall responsibility for the applications and reports quality.

2.4 Weekly schedule

The development team had the following fixed weekly schedule:

- Customer meeting: Monday 12:15-13:00
- Advisor meeting: Monday 13:15-14:00

Rooms were reserved on demand. There were also at least one day a week where the development team works together in the same room.

2.5 Work Plan

2.5.1 Phases

Planning phase

In this phase, a lot of time went to researching development methodology, different useful technologies (like L^AT_EX, different frameworks, customer needs, etc.), and deciding upon a template for the software architecture. This phase was scheduled for completion by September 16th.

Development

This phase started as soon as the planning phase was completed and approved by the customer. It included development of the different applications and testing continuously. This phase was scheduled for completion by November 15th.

Report Writing

This phase included writing the necessary documentation of the final code. A lot of work was put into writing the report during the development phase. However, as the report would be large, and we would need the time to make corrections and add content. This phase was scheduled for completion by November 20th.

Planning of presentation

Planning of the presentation was started after this report was completed. This phase was scheduled for completion the day before the actual presentation November 22nd.

2.5.2 Activities

We identified some big tasks that needed to be done during the project lifetime. These tasks were essential to make the project a success.

The identified tasks are:

- Workshop
- Usability tests
- Integration tests

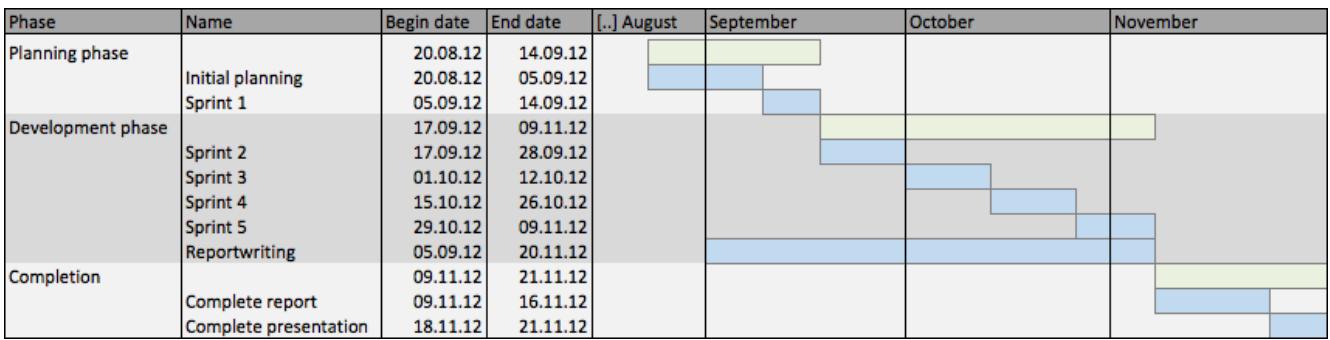


Figure 2.1: Gantt project overview

- Export applications and wrap up the source code
- Project presentation
- Final report correction

2.5.3 Person-hours per activity and phase

We planned the following person hours per phase. These numbers are based on the estimated project effort according to the class staff, and how long we thought each phase would take. As the activities identified in Section 2.5.2 are activities “baked into” the different sprints during development, we have not included estimation for these (for instance, it is hard to know so early how many usability tests we need). Also, we would write the report continuously, and this is considered a subtask of both development and planning. For each sprint we planned to use 175 hours on development.

- Development: 875 hours
- Planning: 425 hours
- Meetings: 125 hours
- Report completion: 30 hours
- Presentation planning: 30 hours

2.5.4 Gantt Diagram

Figure 2.1 shows the Gantt Diagram for the whole project period.

2.6 Risk Analysis

This section contains the risk analysis we did before the project was started. The analysis helped the team detect the most relevant risks for the project, in order to be prepared if such a problem should occur. This allowed the team to make some preemptive measures as well as draw up strategies for handling a number of possible situations. In addition to a listing of internal and external risks, this section contains a SWOT analysis for a deeper understanding of the different relations in the project, both internal and external.

2.6.1 Internal Risks

Table 2.2 contains issues that the group identified as possible internal risks for the project.

2.6.2 External Risks

Table 2.3 contains issues that the group identified as possible external risks to the project.

2.6.3 SWOT analysis

The following section contains an analysis of strengths, weaknesses, opportunities and threats to the project. The analysis is used as a strategic planning method which analyses the internal and external factors in a group. The internal factors are strengths and weaknesses while opportunities and threats represent the external factors.

According to Jackson et. al (2002)[9], the SWOT analysis' intended use is to get an overview of the internal and external factors at the beginning of the project. Later, the analysis was used to map which parts of the project should be relied upon the most, and which opportunities and strengths can help the progress of the project the most.

Keeping the high-risk parts of the project under close watch helped the team catch problems before they evolved. In cases where a diversion was required, the SWOT analysis supported the team.

Strengths

Communication and knowledge were the two greatest strengths in the project group. The fact that all team members spoke Norwegian as their native language, and were fairly competent in English, made all communication and reporting easier. Deciding to do the reporting and programming in English, while all other means of communication in Norwegian led to fewer misunderstandings and made it easier to help each other out when problems occurred. Discussions were also more valuable since everyone were able to participate without the fear of missing out due to lack of understand foreign languages.

When it comes to the level of knowledge, all group members were fourth year Computer Science students. This implied that even though the group members were taking different paths as to which specialization they were following for their masters degree, they all had a common background. It was therefore to be expected that everyone could participate in both coding and writing. The consequences of somebody falling ill were low, since they could be replaced by another member of the team.

The technology was being shared between all team members and every team member took part in each part of the development. In addition to the basic knowledge, team members have chosen their own combination of courses. This made the team more capable of solving a broad array of tasks and finding good solutions to problems. Another strength in the group was that everyone had experience from previous projects, some more than others, but everyone had been involved in IT related projects. This provided the team with the experience needed to avoid some common pitfalls and get a decent start on the projects. The applications is to be used by patients with chronic illnesses, such as asthma. Since there are members of the group with asthma, the group found it easier to take the end users situation into account.

Weaknesses

The team chose to use the document markup language L^AT_EX, even though none of the members had used it before. This led to some problems during the start of the reporting. The team was able to find many different guidelines, research and tutorials to use L^AT_EX, which made it a lot easier to use, but it took time to learn and configure everything. For most people, money is a huge motivating factor. Since this is a university course, the entire group worked for free, making a product someone else may profit from. This meant that the developers had to find some other form of motivation. The group is also given a single grade, based on the overall achievements and results made by the team as one unit. This resulted in the biggest possible gains being experience, knowledge, relations to customers and the final grade. As students, the grade is usually the most motivating factor we get from a course. With a common grade for the group, there is a risk that expectations for the final grade might differ among the team members, and that some members will settle for a lower grade than others. It could prove difficult to get everyone working equally

#	Activity	Risk factor	Consequences	Prob.	Strategy and action	Responsible
IR1	All	Longterm illness among the development team.	M – Decrease of productivity. Lack of expertise.	M	Share information online. The ill developer keeps himself updated.	All
IR2	All	Long-term leave. group member takes a long-term leave.	A M – Reduced capacity.	L	Plan what the member shall do before he is leaving.	The Member
IR3	All	Internal conflicts among the developer team.	M – Bad mood. Less work effort.	H	Bring it up, and handle it right away. Use the advisor.	All
IR4	All	Deadlines not being reached.	H – Unfinished work.	M	Predict work load and set project boundaries.	Project leader
IR5	All	Group members busy with other courses.	L – Reduced capacity	H	Common Google Calendar to plan meetings. The member finds another time to do the work.	All
IR6	All	Group members leaves project.	H – Reduced capacity	L	Reduce the project boundaries.	All
IR7	Meetings	Group members showing up late.	L – Reduced capacity	H	The group member works more next time. Penalties.	All
IR8	Implementation	Lack of knowledge or abilities.	L – More time goes to getting knowledge.	M	Getting the knowledge.	All

Table 2.2: Internal Risks

#	Activity	Risk factor	Consequences	Prob.	Strategy and action	Responsible
ER1	All	External conflicts. One or more of the group members is in conflict with the customer or the group advisor	M – May lead to bad communication, lack of feedback etc.	L	Bring it up and handle it right away. If customer contact is in a really bad conflict, change customer contact.	All
ER2	Design and implementation	Customer changes requirements. The customer is very decisive and demanding.	H – May stress the developers and make them confused on what to prioritize.	H	Force the customer to prioritize tasks. Compromise possible solutions.	Development team and customer
ER3	All	There is insufficient input from the customer regarding the development process.	M – May lead to expectations not being met.	L	Force input from customer.	Development team and customer
ER4	Development	Tools fail. Software tools stop working or are outdated.	M – Development process halted.	L	Do other work. For instance report writing, testing and refactoring. Find other solutions to the problem.	All
ER5	Development	Loss of data. Data connected to the project is lost, or is unavailable for a period of time.	H – Development put back in time.	L	Prioritize tasks according to remaining time.	All
ER6	Meetings, Feedback	One of the customers takes a long-term leave.	L – May delay feedback, input and access to resources.	L	Require more from the other customer contacts.	The customer
ER7	Development	Cannot get the Karott API to work.	H – May lead to removal of this feature for the prototype.	L	Focus on other parts of the applications.	Development team
ER8	Development	Database not working at all.	H – May lead to hard-coding of all features for the prototype.	L	Hardcode necessary parts.	Development team
ER9	Development	Android devices stop working	L – May delay development since emulator has low performance.	L	The group member must switch to emulator.	Group member

Table 2.3: External Risks

hard and as a result, some group members might become frustrated. The team attempted to fight this risk factor by introducing this weakness and discussing what each member wanted to gain from the project.

Opportunities

The gamification concept behind BLOPP has a lot of potential users. There is a significant need for a technological breakthrough in the area of applications regarding medication and motivation.

There exists many different applications for making medication plans, reminding users, logging intake of mediation and similar functionality. A search on Google Play, the application store for Android devices, with the search term “Asthma” gives a result of 214 applications. We downloaded a few of the free ones, and all where very targeted towards adult users, giving information, offering tracking logs and similar.

NAAF is an organization with impact on political decisions regarding health care in Norway. As exemplified by the Norwegian law “Diskriminerings- og tilgjengelighetsloven”[30] (Discrimination and availability law), their work is often referred to in legislative bills and they are considered a very professional and respected organization. The fact that they are backing this project may give the project media attention¹.

NAAF and the BLOPP-project wants to make an application targeting children, but in order to avoid spending money on a poor solution, BLOPP did this as a low-cost project first. Should this project result in a success, NAAF can apply for a financial support from the Department of Health to develop the project further. The concept may also be useful for children with other diseases that require scheduled medication. Rød et al. (2006)[25] writes that 10-12% of the Norwegian school children have asthma, which gives a potentially huge user group.

The final application may also be easily rewritten and target persons with other diseases. This may be done without writing all code from the beginning, but rather change out the parts regarding what medicines are implemented in the solution. This will again give a very huge potential user group, since all people with need to take a medicine regularly may be a target user.

Threats

When making a new product, it is not always clear what the product is going to solve and how it is going to do it. Neither are the opportunities and the limitations. Therefore it is very likely that the product and requirements are going to change during the development process. It is critical that the team makes room for unexpected changes and that adapting to changes is made easy.

To manage this risk, a good working relationship with the customer is necessary. Product tests and demonstrations need to be done iteratively with both users and customers. Failing to do this will be a huge threat to the project and therefore, communication and collaboration will be important.

2.7 Quality Assurance

2.7.1 Language

As a main language for the development project, Norwegian was chosen. The decision was due to all members, customer contacts and the advisor being Norwegian. The report is written in English. All code, including comments were all in English. The language in the applications was chosen to be Norwegian, since the applications is targeted towards norwegian children who do not necessarily understand English.

2.7.2 Customer Meeting

Customer meetings were held once a week. The reason behind this schedule was due to the customers desire of high involvement in the project, and the belief that having a high frequency of meetings would result in a higher quality result. High involvement makes the process of feedback and new ideas easier, and

¹See Appendix F

restricts the possibilities for the development team tracking off course. The meetings were usually held on Mondays. If any unexpected events resulted in meetings being moved, a notice was given at least 24 hours in advance. The first customer meeting was held at St. Olavs Hospital the 21st of August.

2.7.3 Advisor Meeting

The advisor meeting were held each Monday at Campus Gløshaugen. See Appendix B for a typical meeting agenda. The main items of the agenda was the meeting approval of the report from the last meeting, a project status report and discussion on problems and difficulties. A plan for the following week was also presented. During the meetings, the advisor was able give feedback on the status of the report regarding the quality expected from the final result.

Chapter 3

Preliminary Studies

3.1 Children with asthma

Asthma is a chronic inflammatory disease that affects the airways and lungs.[7] It is often more prominent in children, who are more active and easily excited than adults. The asthma is typically triggered by excitement, physical activity, rashes or allergies. The problem is managed by multiple medicines, with different schedules as to when to take them, and how much to take, depending on the condition of the child. There are inhalation medicines that consists of either small dust particles that effect the lungs locally, or saltwater that are inhaled as vapor, either alone or combined with other medicines to loosen up slime. Among these inhalation medicines is Ventoline and Flutide, seen in figure 3.1 and 3.2. There are also pills and liquid medicines that either affects the immune system or affects the body globally, not just locally in the lungs.

The medicines can also be divided into medicines that have immediate effects, and are used during an asthma attack, while other medicines are preventative, and is taken on a regular schedule. These preventative medicines are targeted at bolstering your immune system before going to certain areas, or days where the child is likely to be in contact with materials it is allergic to.

With some of the inhalation medicines it's difficult to time the inhalation with the release of medicine, and here an inhalation chamber, or inhalation mask, is used. When the child has taken the nebulizer this way, usually once or twice a day, they must remember to wash their mouth afterwards.

Figure 13.1 shows an image of a nebulizer machine. The nebulizer machines often makes a lot of noise, and can be scary to children. The nebulizer treatment takes between 2-10 minutes, 1-4 times a day.



Figure 3.1: Ventoline[3] is an inhalation medicine that opens up the airways shortly after inhaling it.



Figure 3.2: Flutide is a steroid keeping the illness checked, but it has no immediate effect.



Figure 3.3: A nebulizer machine.[28]

3.1.1 Traffic Light Classification of Asthma Condition

The traffic light program is a way of classifying condition and resulting medication for asthmatic patients. It is a very simple system that requires very little knowledge to understand, and is therefore well-suited for a project aimed at children. The basic outline can be compared to a traffic light.

Borge et al. (2002)[38] defines the zones as *green*, *orange*, and *red* and describes effects and treatment for each of them.

Green Zone

Green is the normal zone. A patient in the green zone can be described as in “regular condition”. He or she is breathing normally, even when doing light physical exercise.

When an asthmatic is in the green zone, it is normal to take two to three different medicines each day, often with cortisone

Yellow Zone

Yellow is the “ill” zone. When a patient is in the yellow zone, they exhibit moderate signs of illness such as breathlessness and coughing. There may also be allergy reactions, and waking up at night from breathlessness and coughing. A patient may be defined as being in the yellow zone if he or she has a cold.

In the yellow zone, patients typically take more medication than in the green zone. A normal amount is 4 to 6 doses daily. The medication from the green plan is taken as normal, in addition to any new medicines introduced by the yellow state.

Red Zone

The red zone is labeled the “stop” zone. A patient in the red zone will have almost closed airways, making it very difficult to breathe and the person will have to stop any activities.

A patient in the red zone has to fix his or her state immediately. This could be by opening windows, finding a good resting position, taking specific emergency medication or using specific breathing techniques. If these courses of action don’t help, the patient should immediately call the doctor.

3.2 Parents with children affected by asthma

Parents of children with asthma face a series of challenges concerning the medication of their children. One of these challenges is to give the correct amount of medicine, the right type of medicine, at the correct time of the day. Many parents have experienced stressful mornings where they are late for work, their children are unwilling to take their medicine and they either do it in an incorrect manner, reducing the medicines effect, or not giving it at all. The medication plans can be hard to understand, even though they are designed to be easy, and it's typically only one of the parents that have been given the instructions from the doctor directly, making it even harder for the other one to do it correctly.

The children are not always happy about taking their medicine. The inhalation mask might be scary, the medication might interfere with their planned activities that day, or any number of other reasons the child might not want to take his/her medicine that day. Having to force a child to take their medicine could make the child associate taking the medicine with a negative experience, and it becomes increasingly difficult to give the medicine to the child.

3.3 The concept of gamification

Gamification is the concept of applying game-design thinking to non-game applications to make them more fun and engaging. Nick Pelling (2011)[16] writes that the term *gamification* was introduced in 2002, but was not popular before 2010 (Daniels, 2010[17]).

Common techniques applied to introduce gamification to a process include, but are not limited to:

- Achievements/badges
- Levels
- Leaderboards
- Progress bars
- Avatars
- Gifting
- Challenges
- Embedding of minigames

These techniques are all widely used. Here are some examples:

- In games for platforms like Playstation and Xbox, gamification is frequently used to make people finish campaigns.(Hamari et.al, 2011[6]) Trophies and achievements are used to motivate people to complete games 100%. You may have to collect every single item in the game, fight every boss and so on.
- Nike+ introduced gamification to training. (Zichermann et. al, 2011[5]) You can track how fast you run, your maximum pulse and so on, and try to beat this target the next time you are running.
- You can “check in” to places you have been with Google Maps, Facebook, Foursquare, Google+, GoWalla, and similar applications.
- Airline companies uses bonus points as motivation for customers to fly more with a company. (Gamification Wiki, 2012[4]) At a certain level of these points, you may get one more flight for free, get a free upgrade to a better seat and so on.

We hope gamification will help making children's treatment as enjoyable as possible. Michael Wu has proven that gamification has an impact on human motivation. Wu presents: "Game mechanics and game dynamics are able to positively influence human behavior because they are designed to drive the players above the activation threshold (i.e. the upper right of the ability-motivation axis), and then trigger them into specific actions" (Wu, 2011)[14].

The gamification elements most suited for children, is gifting and embedding of minigames. The gifting needs to be visual, and the two elements can be interwoven to increase the effect, where the minigames changes according to what gifts you've already achieved. Another way that can implement gifting targeted at children, is to implement a currency system, for example coins, and reward them with physical rewards when they reach a certain amount of coins. The rewards could be all from small toys and candy, to larger items, depending on the amount of coins, and the tasks performed to get the coins. A point here is to make sure the currency is very visual, to make sure even young children understand how many coins they have. The implementation of minigames would help greatly in distracting children during tedious tasks like nebulization treatments.

Gamification has a lot of positive sides, but it may have bad sides if gamification is done the wrong way. The fact that we are using gamification to motivate children, makes this an even more serious concern. If children feel that they are stuck in a game, motivation may very easily be broken down. So whatever concept we come up with, it has to be to motivate the children at all time. They cannot feel stuck with our app. This concept is up to the customer to create, while we will be implementing it.

As written above, gamification may have negative effects on users. Anderson-Rainie (2012)[22] show results that users are likely to not be motivated or feel that they are manipulated into completing a task. The research also show that users are likely to feel that gamification is too childish, and that gamification is a trend that will disappear within a few years.

3.4 Karotz

Karotz.com (2012)[15] describes Karotz as a robot shaped as a bunny that can interact with a user through light, ear movement and sound. It can also take input through a button, moving its ears, an RFID (Radio-frequency identification) chip, voice commands and serial (Internet) communication.

The project includes developing an application for the Karotz platform that will serve as an addition to the mobile applications. It is therefore necessary to study its interfaces, development methods and API of the machine.

3.4.1 Application Platform

Karotz application (called "Appz") are installed through an online platform located on the Karotz web site. They can be launched on the Karotz itself either through a scheduler, voice commands or an RFID chip. These RFID chips come in various shapes, sizes and colors. Figure 3.5 and Figure 3.6 show examples of the different kinds of "nanoz", that are small figures with an integrated RFID chip.

Some mentionable applications made by other developers are "At Home", an application that may register that someone checks in at the Karotz, and send an e-mail to a predetermined mail address, so children may tell their parents that they are home. "Twitter for Karotz" may read you tweets and post tweets based on voice-commands. Another mention is "Weather" which may tell you the forecast for the day or the following day. It seems all applications registered at the Karotz website are fairly simple and have little functionality.

As for launching the BLOPP application, the times for the scheduler must be manually set through the Karotz web site, so it cannot be used for notifications directly. The best option for the BLOPP implementation would therefore be to set a scheduler to start every day at 00:00 and stop every day at 23:59. This way it can be ensured that the application is always running, updating itself with medications, status and times, and a timer can be used to schedule notifications.



Figure 3.4: Karotz: A bunny-shaped robot

The Karotz can be programmed in two different ways: either through a web REST (Representational State Transfer) framework, or with JavaScript that runs as an embedded program on the robot itself.

The requirement that a REST program would have to be hosted somewhere, combined with the fact that an embedded program provides more flexibility in terms of local storage to limit the amount of information sent over a network makes the JavaScript framework a more suitable choice for the BLOPP project.

The Karotz has a few ways of providing output to the end-user. It can be asked to

- play sound files;
- move its ears;
- speak, using a TTS (text-to-speech) engine;
- illuminate its stomach in different colors; and
- communicate over the internet with HTTP (Hyper Text Transfer Protocol) GET and POST methods.

For providing user commands, TTS could be an option if the engine supported Norwegian, but since the language options are limited to English, French, German and Spanish, speech will have to be created by recording sound files and playing them with the multimedia engine.

3.5 Pollen forecast

NAAF[8] states that asthma- and pollen treatment needs to be seen in correlation. Children may feel worse during pollen season than the rest of the year. We intend to give parents more analyzing tools by being able to see connections between asthma symptoms and different types of pollen by using pollen data. It may occur that asthma is not the original source of a child's condition, but an allergy against a certain



Figure 3.5: A flat nanoz–flatnanz– a figure with an RFID chip used to provide input to a Karotz.



Figure 3.6: A round nanoz–nanoztag– a figure with an RFID chip used to provide input to a Karotz.

pollen is the source. NAAF hosts a pollen forecast at <http://pollenvarslingen.no/> which we intend to use. We have been given a user key for this cast, even though they do not normally distribute this sort of information to application developers.

3.6 Design workshop

Before the first sprint, a design workshop was held. Hanne Linander, a master student in Industrial Design at NTNU was responsible for the workshop. The goal was to come up with different ideas for functionality and design, and make an early sketch for the layout of the application.

At this stage in the study, we had not yet decided to develop two separate applications, so we created layout suggestions for an application aimed to do both tasks for children and for adults.

Many different exercises were completed throughout the day, in order to make as many creative ideas as possible. Examples of exercises are short time boxed sketch sessions, cross-collaboration without explaining thoughts behind the sketches, different idea-competitions, among others. At the end the ideas were evaluated, resulting in many discards and some being taken into further development. After the workshop the sketches and the ideas were presented to the customer. The development team was very happy about the results and decided to develop a paperprototype from the sketches.

3.6.1 Results

We drew several sketches of how we envisioned the different views in an imagined Android application.

Figure 3.7 shows a suggestion for the main menu in the application, where the action of taking medicine is in focus. The menu includes:

- a log button where an adult can view medication history for the child,
- a manual button for learning how to use medication,
- a settings button for adjusting mediation schedule, alarm ringtone etc.,
- an “about the app” button for learning how to use the application,
- a display where one can view the current health state for the child, and also click on it to change the health state, and

- a count of how many stars the child has collected, with the option to click on that count to see more detailed information about the rewards collected.

Figure 3.8 illustrates how a user would see the health state view separately from the menu. It would consist of three elements represented as colored smileys; one for each health state. The healthy smiley would be green and smiling, the sick one would be yellow and in a mellow mood, while the very sick state would be represented by a red, frowning smiley. The currently active health state could be displayed by illuminating the respective smiley, and clicking on another would change the active state.

Figure 3.9 shows how the health state view could be integrated into the main menu better by injecting it into an overlay. This could make navigating the app more intuitive.

Figure 3.10 is an image of the view a user would first see when clicking the big “medicine” button in the main menu, or the view he or she sees when redirected from a notification. It shows an image of the medicine to be taken, and gives the option to go directly to the medication process through clicking on the medicine image itself. If a user is unsure about how to use the medication, there is illustrated an option where one would directly be brought to the manual page for that medicine.

Figure 3.11 shows a view one would be brought to if more than one child was registered in one application. It shows an icon for each child and the child’s name. Clicking one one or the other would bring the user to the distraction page for that child.

Figure 3.12 is an image of the first part of the distraction animation a child would see. It shows dark clouds and a thunderstorm that represents the child’s state before taking medication. It was thought of as a way to motivate children to take medication in order to clear up the clouds, and as something children could relate to by comparing their progress to the clouds.

Figure 3.13 is also part of the distraction animation. It shows a medicine unit emerging from behind the clouds which are clearing up while the medication is ongoing. The emerging medication was supposed to symbolize the goodness of medication, while the clearing of clouds would symbolize the child’s lungs and airways clearing up.

Figure 3.14 illustrates how a child could view his or her collected stars. There is a big star with a count next to it, which would show the total amount of stars. In addition, there is a scrolling view on the top where each day would have an amount of stars on it, corresponding to how many stars the child had received on that day. Lastly, there is an appealing figure on the page, in this case a “ninja master”, which would be an avatar for the child. It was imagined as something that the child would purchase or obtain after reaching a certain amount of stars, and the object itself would vary, serving as an additional motivational factor to supplement the stars themselves.

Figure 3.15 is a view where adults could check the progress and medication history for a child. It would display graphs of condition and how good they were at following a medication plan. Days would be colored after what condition the child was in on that day. The amount of stars the child collected is also shown in each day, and a percentage of how often a given medicine is taken on the planned time is displayed on the bottom right.

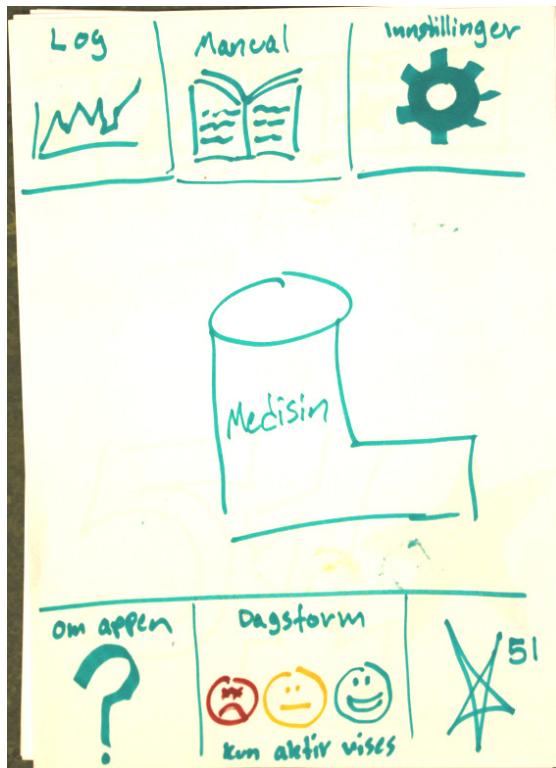


Figure 3.7: View for the main menu of the application.



Figure 3.8: View for changing health state (active medication plan).

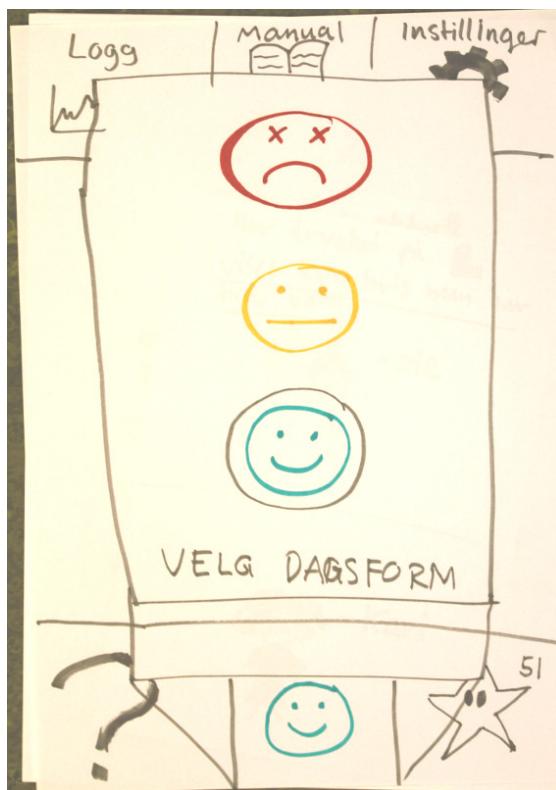


Figure 3.9: View for changing health state as a pop up from the main menu.

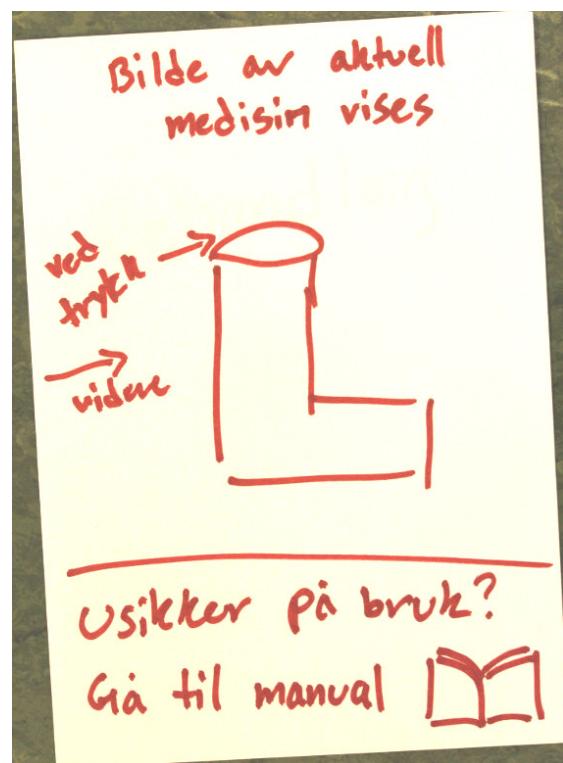
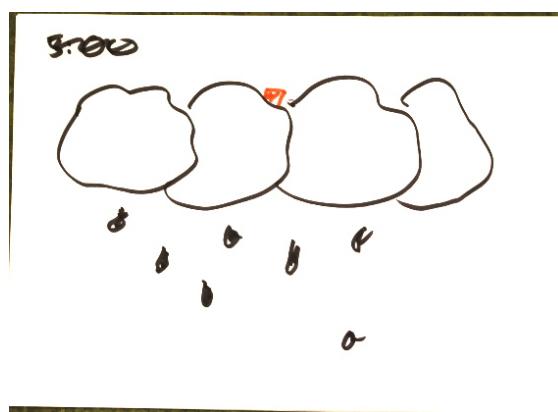
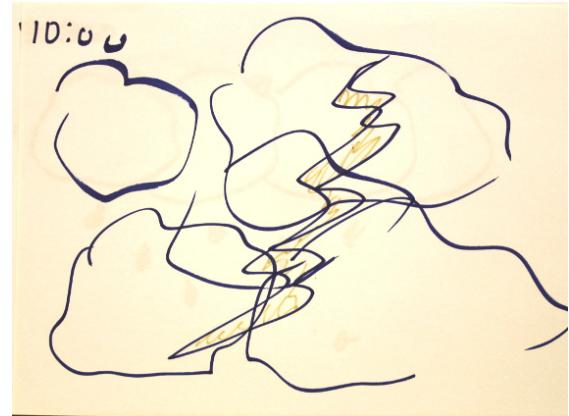


Figure 3.10: View for starting a medication and distraction process.



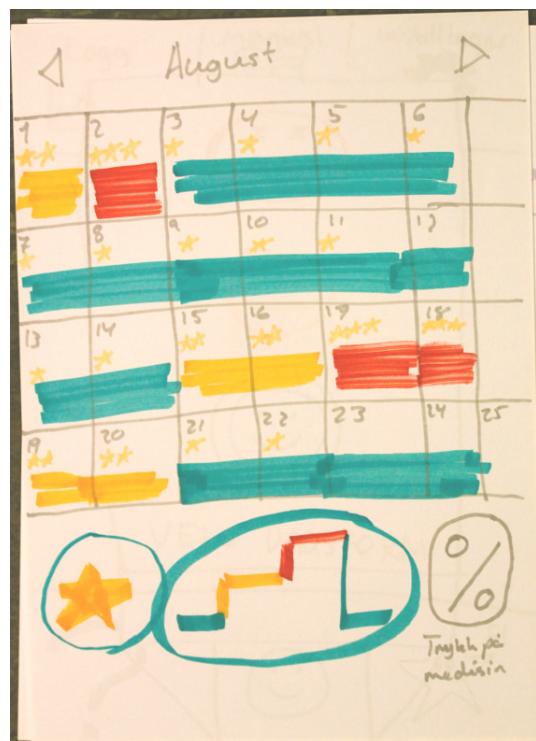


Figure 3.15: View where adults can view the medication and health state history for a child.

3.6.2 What was used in the further development

Several ideas from the design workshop was used later in the development. We realized that we did not have sufficient time to make a very complicated distraction and reward system for CAPP, so we decided to keep the ideas from the workshop about rewarding the child with stars whenever he or she completed a treatment. This is easily implementable, and we believed it would be easy for parents to build on it with physical rewards when the child had accumulated enough stars. For the same reasons we chose to keep and build on the idea of having an animation sequence where a Karotz avatar mirrored what the child had to do, it would equip the inhalation mask when the child had to, making it more exciting for the child to do this as well. This approach have been seen to work with other applications, like for helping children brush their teeth, and we wanted to use this as our base point.

We did the workshop based on the idea of having one application, but later decided on having two. This meant that most of the basic GUI elements we came up with here, had to be redone. For GAPP we implemented a log which kept its general layout, with the coloring showing which plan had been followed the relevant day. We later switched around on the additional information that would be shown on each date and how, but the concept stayed the same.

3.7 Frameworks used in the Project

In this section the different frameworks that have been in use in the project is presented. The frameworks consists mainly of the development model, the different programming languages, the database and server tools, the Karotz API, the Android SDK and the IDE used for development.

3.7.1 Programming Languages, Message Formats and File Formats

The following section will comment on different programming languages, communication protocols and file formats used in the project.

Eclipse IDE

Eclipse [43] is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. The source code is mostly written in Java. Eclipse may be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, Ruby, Python, and many others. Eclipse is owned by the Eclipse Foundation, a non-profit organization focusing on creating and maintaining a community for individuals and organizations who wish to collaborate on commercially-friendly open source software.

Java

Java is a programming language developed by Sun Microsystems in 1995, and is licensed under the GNU General Public License. Java can be run on any Java Virtual Machine (JVM), which means that it can run on any platform. It is an object oriented language and is based upon classes. According to TIOBE Software (2012)[44], it was ranked as the 2nd most popular programming language in the world.

Android SDK

The Android software development kit (SDK) [41] includes a comprehensive set of development tools used when developing Android software. The kit includes an Emulator, documentation of the source code, sample code, tutorials and a debugger. Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK will also support older versions of the Android platform, through downloading extra packages of source code, in case developers wish to target their application towards older

devices. By building the project through the IDE the Android application is packaged in a format which may be distributed (.apk).

Android Developer Tools

The Android Developer Tools (ADT) [42] is a plug in for Eclipse that provides a professional-grade development environment for building Android applications. The ADT makes it easier to use the functions and tools that the Android SDK provides, by giving access through the user interface of Eclipse.

Javadoc

Javadoc is a tool for documenting code. It is integrated in Eclipse IDE, and we will use it document our code. When we use code-completion in Eclipse, the Javadoc of the function selected is shown as its description.

JavaScript

Two parts of the project will make use of JavaScript. The settings page for doctors is web based and will use JavaScript for interactivity. The Karotz can be programmed in two ways; either through a web API on the server side or through stored JavaScript code. In order to maximize stability and power, the client hosted JavaScript technique will be used.

JavaScript is a multi-paradigm, weakly typed and dynamic language that is extensively used on the web today. The main application of JavaScript is to enhance interactivity on a web page through client-side interpretation of the code in a browser. It can be used for a number of purposes such as making something happen when a button is pressed, loading new data without refreshing the page and much more. Even though JavaScript is by far most commonly used on the web there are also applications of it in other areas. Examples of these kinds of implementations are Node.js[18], a network application creation platform for writing JavaScript code as a regular server-side program, and the Karotz API which we will be using.

Karotz API

The Karotz API is based on a Javascript engine. The API is downloadable from the Karotz's homepage and gives access to the Karotz's different functions such as changing the color of the LED light, playing prerecorded sounds, waving its ears and registering contact with an RFID-chip.

A full documentation of the API is given, however most of the example code is commented in French.

3.7.2 Database

Since the project team has decided to use a database to store information relevant to the systems, there is a requirement to use a programming language for creating and maintaining, as well as accessing, the database. Because of technical limitations in the Karotz explained further in Section 6.4.2 we chose to split the database into two parts: the database itself, written and maintained in MySQL, and a set of access modules written in PHP (PHP: Hypertext Preprocessor) and MySQL.

MySQL and phpMyAdmin

MySQL[27] is the world's leading open source database. It offers all the tools necessary to implement and maintain data records for a project such as ours. The advantage of being the biggest contender is that there is a myriad of resources easily and freely available to MySQL developers on the Internet. MySQL is suitable for our project because it is reliable, free and open source, and all the group members have previous experience working with it.

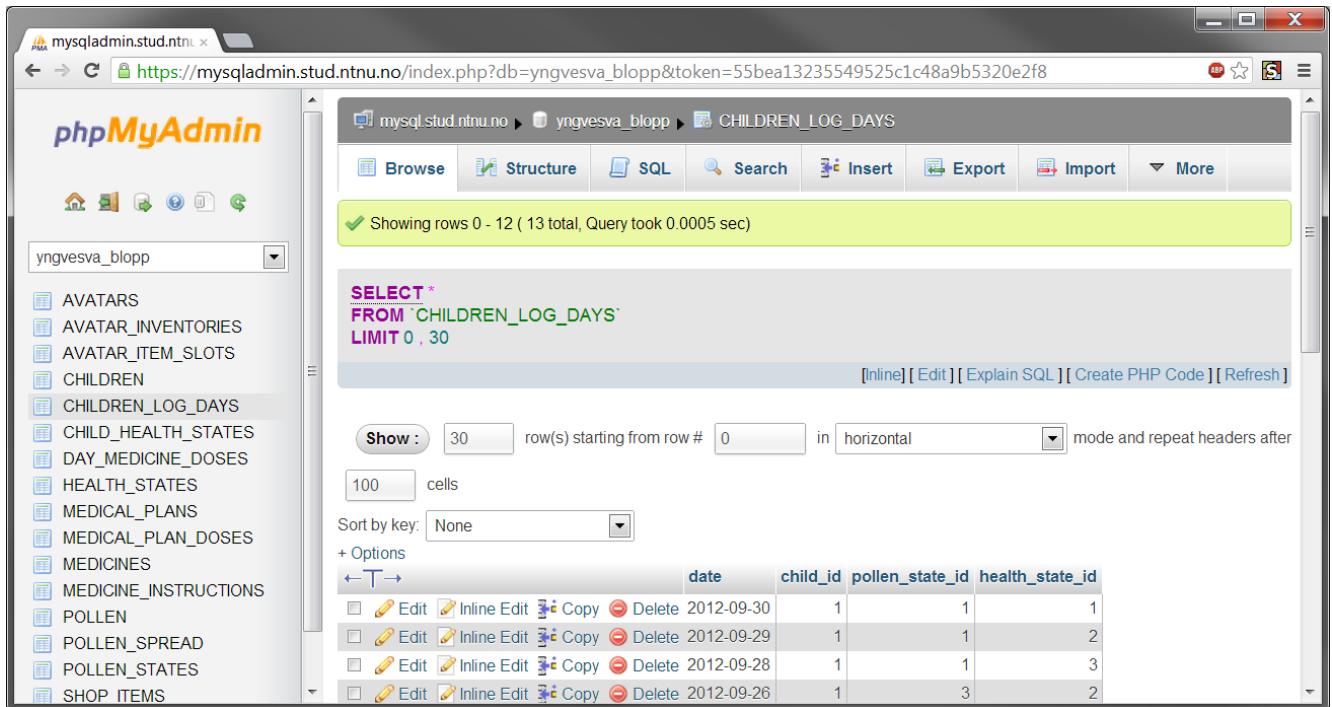


Figure 3.16: phpMyAdmin screenshot

PhpMyAdmin is a graphical MySQL manager that is installed on NTNU’s MySQL server[37]. It provides an easy way to manipulate a MySQL database without having to write any SQL code. Figure 3.16 shows a typical screen from phpMyAdmin, where one can edit, add rows, add information etc. directly within a GUI. Since NTNU’s servers are already integrated with the program, and we use NTNU’s MySQL server for development, phpMyAdmin will be used to administer the database efficiently.

PHP

PHP[20] is a scripting language that is extensively used on the web because of its flexibility, ease of use and popularity. It is mainly used for creating and editing HTML (HyperText Markup Language) pages server-side before sending to the client. In our project, we used PHP to access the NTNU MySQL database from a web server located on a NTNU sub network(<http://folk.ntnu.no/>), and return a JSON file.

JSON

JSON[31] is a format used for storing and sending data in a light, human-readable and easily parsable format. It’s based on JavaScript notation, but uses only a small subset of the JavaScript syntax; only strings, numbers, lists, objects, booleans and the null value are included. We have used JSON for storing configuration data in the Karotz application, and for sending data between the PHP web access modules and the client applications.

3.7.3 Extra Tools used in the Project

The following section contains a description of the tools used for testing of the system, project and task management, team and customer collaboration, communication between participants. The tools chosen were chosen due to familiarity, making the learning curve as flat as possible.

Testing Tools

This section describes tools used for performing unit tests, usability tests, and end-to-end tests.

HTTP Requests Postman[34] is a REST client for performing both basic and advanced HTTP requests to a given URL. It is distributed as an extension for the *Google Chrome* web browser. Postman keeps a history of all sent requests, as well as an easy-to-understand interface for performing GET and POST operations. Since the database was accessed through a web interface that used GET and POST methods, it was natural to choose a REST client for the development process. Postman is also able to format returned JSON, which was useful because the web access modules return JSON formatted data. Figure 3.17 shows a typical screen in Postman when performing a POST operation.

Dropbox

For file sharing between customer and the developer team, and the developer team between each other, we used Dropbox. Dropbox[23] is an online storage which allows sharing of folders and documents between invited partners. Dropbox is asynchronous, meaning only one person may edit a document at a time.

Git

To ensure that every team member was always up to date and no documentation was lost, version control systems were enforced. Git[45] was used as a tool for version control. Git is a distributed source code management system, and was developed by Linus Torvalds in 2005. It comes with a lot of useful features like rollbacks to previous versions, file history and possibility to work on several branches, among others. The repository was hosted at Github (<http://github.com>).

Google Drive

Google Drive (earlier Google Docs), is a file storage and synchronization service made by Google. Google Docs is now housed at Google Drive, and is a free web-based office suite. Google Docs allows several individuals to share and write the same document at the same time, and is ideal to write simple documents concurrently. Google Drive was used for writing agendas and status reports.

Task Management

To ensure task management the team used AgileZen. AgileZen[24] is an online project management tool. You can add user stories, build a backlog and easily add tasks to each story. Unfortunately it is not perfect for projects using Scrum. AgileZen gives no simple way of keeping order of which task is to be done within a certain sprint. Their suggestion is using color coding of tasks, but this was not preferable. There is also no way to assign a single task to a person, you may only sign epics/user stories to a user. Leaving no opportunity for showing that two people may work at the same use story at the same time. At a small project, such as ours this was at times a necessity, which AgileZen did not fulfill. The team used AgileZen in order to keep the customer updated, while the team used Google Docs internally in order to keep track of tasks.

Mockup Tool

For mockups the team chose Balsamiq Mockups. Balsamiq Mockups[32] is a tool designed for easing the collaboration between the GUI developers and the customer. The main advantage to Balsamiq Mockups is the way it ensures no one is too attached to the design. By making sketchy, low-fidelity frames it moves the focus of design conversations towards functionality. Balsamiq also has functionality for making click-through prototypes, which are very nice for demonstration purposes and usability testing. The team was also advised by the customer and the advisor to use Balsamiq Mockups.

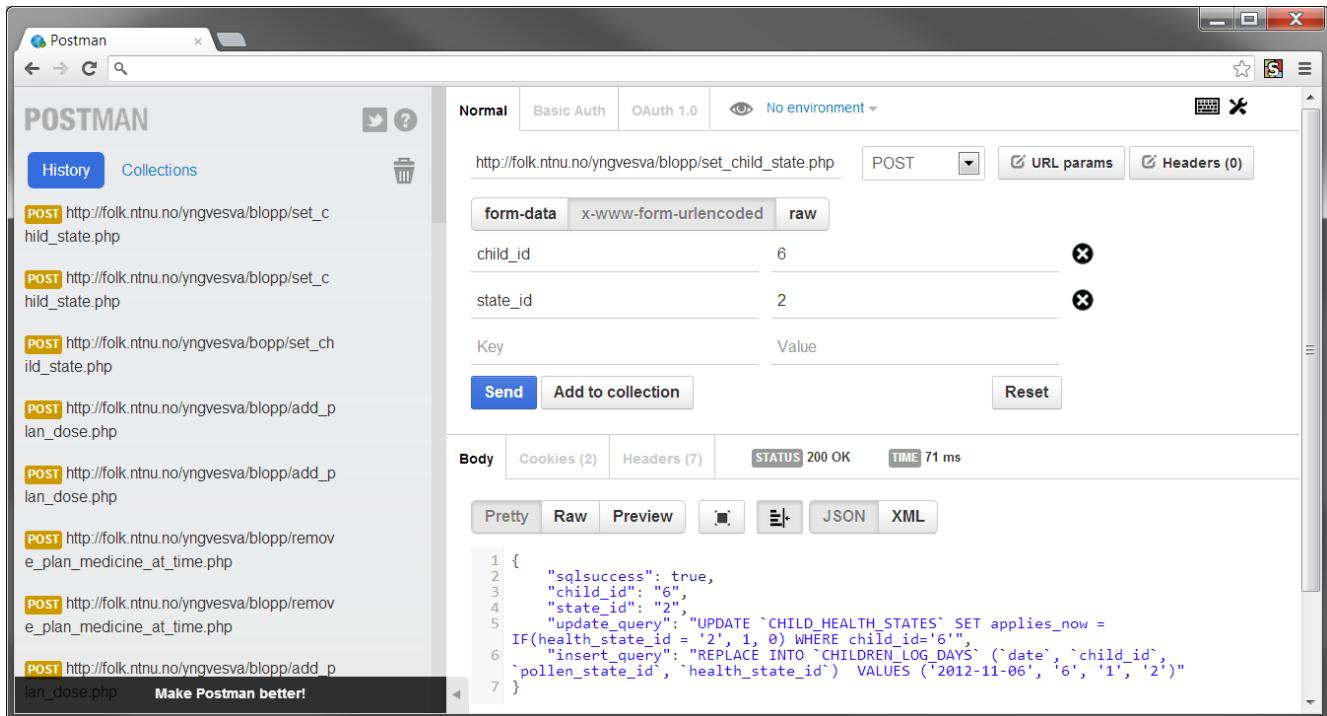


Figure 3.17: Postman screenshot

3.7.4 Design Principles

Google[29], who is developing the Android operating system has some design principles in order to help programmers make better applications. As Google states, quotation: "These design principles were developed by and for the Android User Experience Team to keep users' best interests in mind. Consider them as you apply your own creativity and design thinking. Deviate with purpose."

Many of these principles are guidelines to help the developers make more attractive and better applications, and some of them are rules, meant to follow without deviation. Google does not always state which is which, but refers to their slogan "Don't be evil", meaning developers should not try to scam, trick or hurt users through their applications.

The team as a group has read the design principles and would like to mention the ones that are most central for the applications we are making:

Real objects are more fun than buttons and menus. The principles states that users should directly touch and manipulate objects, rather than buttons and menus. This reduces the cognitive effort needed to perform a task while making it more emotionally satisfying.

Let me make it mine. People love to add personal touches because it helps them feel at home and in control. Provide sensible, beautiful defaults, but also consider fun, optional customizations that don't hinder primary tasks.

Keep it brief. Use short phrases with simple words. People are likely to skip sentences if they're long.

Only show what I need when I need it. People get overwhelmed when they see too much at once. Break tasks and information into small, digestible chunks. Hide options that are not essential at the moment, and teach people as they go.

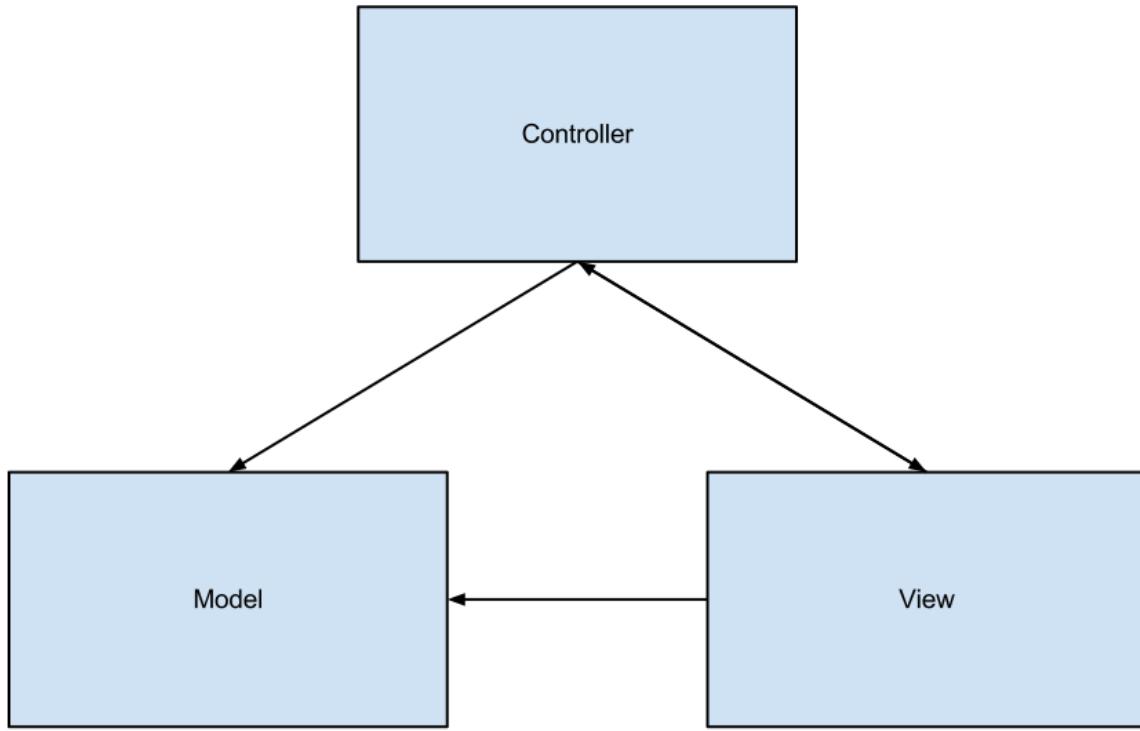


Figure 3.18: Graphical view of MVC

3.8 Software Architecture

In this section, we'll describe some of the software architecture concepts that is applied in the development of the project. “The software architecture of a system is the set of visible structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.” [19]

3.8.1 MVC - Model View Controller

MVC is an architectural pattern used frequently in small software systems and applications similar to CAPP and GAPP. The pattern consists of three components, Model, View, and Controller respectively.

A model consists mainly of “raw” data. A view is responsible to display the data to a user in an appropriate manner, while the controller receives input and manipulates the data models given the input.

We will use MVC as our main architectural pattern to develop CAPP and GAPP.

The advantage of using MVC in applications is that it separates functionality, and it becomes easier to modify the functionality of for instance a single underlying models.

Figure 3.18 shows an textual image of MVC. An arrow from A to B indicates an association from A to B.

View	Purpose
Logical	The logical view is an object model of the design, and is concerned with end user functionality. Addresses end users, customer and development team to give a brief introduction to how the system is implemented.
Process	The process view gives a description of concurrency and synchronization aspects of the design. Reflects upon properties like scalability, performance, and internal processes.
Physical	Describes how the system interacts with different types of hardware. Addresses system engineers, and describes communication protocols and topology.
Development	Describes how the software is organized in a development environment. Addresses programmers and software management.

Table 3.1: Purpose of the 4+1 View Model

3.8.2 4+1 View Model

Kruchten (1995)[35] defines the 4+1 View Model as a model for describing the software architecture for several stakeholders through different views (Please note that “View Model” in this context is something completely different than the views and models described in Section 3.8.1). These views are called Physical View, Process View, Development View and Logical View, respectively, and are built around a series of scenarios, or use cases. We will use this model to describe the architecture in Chapter 6

Table 3.8.2 shows the purpose of the different views.

3.9 Privacy and security

Early in the project we did some research into what information we could be legally allowed to keep track off, and more importantly, what was recommended to avoid doing, to be able to avoid issues later. Specifically we looked into what information we could save regarding how well the medicationplans were followed, and in if this information could be sent to medical staff, with information about who followed the plans well and who did not. We discovered it was not necessarily legal to send this information without consent from the person the data was collected from, or in our case, the guardian of that person. We thought about making this information available to medical staff, but dropped it since it is illegal to send this type of information without consent.

In terms of whether or not we could use the child’s personal number as an identifier, we discovered that this was legal, but only if we had an actual need to save it, that we had pursuant in the law to save it, and finally, that satisfactory identification of the relevant person could not be achieved in any other way. For the system we were making we had none of these criteria in order, but for future expansions this might be information actually required. Hospitals have to be sure they give medicines to the correct people, so the personal number is already written on any prescription medicine used. If the system is expanded towards the hospital nebulization treatment it might be necessary or useful to store the personal number in our database, to connect the user with the person receiving the treatment.

Chapter 4

Development Methodology

This section begins with a comparison between Waterfall development and agile development. Further is contains descriptions regarding the different development methodologies that have been brought up in discussion. Each subsection includes both a short explanation, advantages and drawbacks for each methodology.

4.1 Waterfall vs Agile development

The first formal description of waterfall was published in a 1970 article by Winston W. Royce[10]. This method has been around for decades. The waterfall method is based on the idea of visiting each of the phases, Initiation, Analysis, Design, Construction, Testing, Implementation and Maintenance, only once and finish one before starting the next. The name is given from the idea of progress flowing through each phase, like a waterfall. This results in huge challenges regarding controlling dependencies if the project does reiteration over previous phases at a later stage.

Agile software development is not one certain method, but rather a group of software development methods based on iterative and incremental development. The concept of agile software development was introduced in The Manifesto for Agile Software Development, 2001[11]. Even though iterative and incremental development methods had been around for some years, this manifesto gathered all best-practices in one place. The most used agile methods are, in no particular order, Extreme Programming, Adaptive Software Development, Feature Driven Development, Scrum, Kanban, Lean Software Development and Agile Unified Process.

For the sake of this project, we chose to research waterfall, Scrum and Kanban in order to find a suitable development method.

4.1.1 The Waterfall Method

Figure 4.1 shows a graphical explanation of the sequential design process called the waterfall method.

The main advantage to the waterfall method is that bugs and changes are cheaper to fix if you fix them right away, as it will save you a lot of time and/or money later on.

The main drawbacks are that once the project has moved on to the next phase, the team should not backtrack and edit the previously completed phases, since this might make the further implementation more difficult. The fact that planning has to be done very thoroughly in the beginning to avoid having to reiterate previous phases at a later stage as this can be costly and complex, is also a disadvantage. Leading to a problem with projects where there is no overview to what is to be done and how long time it will take, this method will lead to uncertainty. A roll back to an earlier stage will most likely prove early estimates wrong and might cause complications to the development.

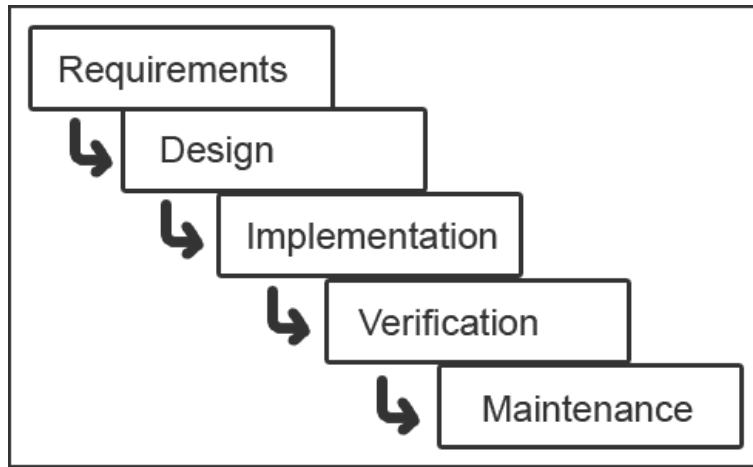


Figure 4.1: Graphical representation of the waterfall method[2]

4.1.2 Scrum

Figure 4.2 shows a graphical explanation of the Scrum method, one of many agile development methodologies. It is an iterative, incremental model which emphasizes on doing several short sprints where the goal is to complete some smaller set of tasks. After a given period of time, usually one to four weeks, the development team summarizes what have been done and what is left from the current sprint, which needs to be completed in the upcoming sprints.

The advantages of scrum is that it makes the software development more versatile, the team can work on all phases and parts of the project at the same time, and update earlier assumptions based on newer discoveries. Meaning that requirements and modelling does need to be finished before starting implementation and because of this, changes are less expensive to do. This is done by having a more relaxed relationship to documentation of source code and the process.

Nothing is written in stone until the product is done, as opposed to the waterfall method, mentioned earlier.

The main drawback of Scrum is the complexity of it. All methods has a certain learning curve at the beginning, leading to stress or less effective work. Scrum has several submethods, all with small differences. This may lead to a learning curve for experienced users.

A more specific explanation of how this project used Scrum is found in Section 4.2.1.

4.1.3 Kanban

The Kanban is, as formulated by Anderson (2003[39]), a method for software development with an emphasis on just-in-time delivery, while maintaining focus on not overloading the developers. It emphasizes that developers pull work from a queue, and the process, from definition of a task to its delivery to the customer, is displayed for all other participants to see.

Kanban is based on some basic principles:

Start with what you do now. The method does not define a specific set of roles for team members, or process steps to follow. There is no such thing as "the Kanban software development process" or "the Kanban project management method". The method bases further work by starting with the roles and processes you have and stimulates continuous, incremental and evolutionary changes to your system.

Agree to pursue incremental, evolutionary change. The organization (or team) must agree that continuous, incremental and evolutionary change is the way to develop improvements and make

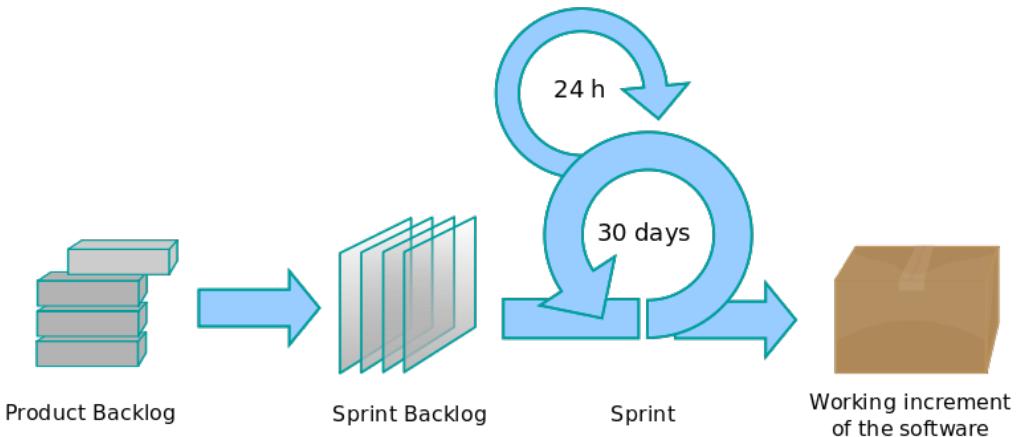


Figure 4.2: Graphical representation of the SCRUM method[1]

the changes stick. The Kanban method encourages continuous small incremental and evolutionary changes to your current system.

Respect the current process, roles, responsibilities & titles. It is likely that the organization currently has some elements that work properly and are worth bringing forward for the future. By agreeing to respect current roles responsibilities and job titles we eliminate initial fear of change. This should enable the team to gain broader support for a Kanban initiative.

Core practices

According to Anderson (2003[39]) there were originally five core practices in the Kanban method.

- *Visualize.* A common way to visualize the work flow is to use a card wall with cards and columns. The columns representing the different states or steps in work flow. This is similar to a scrumboard.
- *Limit WIP.* Limiting work-in-process implies that a pull system is implemented on parts or all of the work flow. The critical elements are that work-in-process at each state in the work flow is limited and that new work is "pulled" in when there is available capacity within the WIP limit of the team.
- *Manage flow.* The flow of work through each state in the work flow should be monitored, measured and reported. By managing the flow continuously, the changes to the system can be evaluated to have positive or negative effects on the system.
- *Make Policies Explicit.* Until the mechanism of a process is made explicit it is often hard to hold a discussion about improving it. With an explicit understanding it is possible to move a more rational, empirical, objective discussion of issues. This is more likely to facilitate consensus around improvement suggestions.
- *Implement Feedback Loops.* Collaboration to review flow of work. Organizations that have not implemented the second level of feedback - the operations review - have generally not seen process improvements beyond a localized team level. As a result, they have not realized the full benefits of Kanban observed elsewhere.

4.2 Choice of methodology

The development team chose the Scrum methodology instead of Kanban, due to several reasons. Foremost, the customer asked the team to work in an agile methodology, and preferably Scrum, since they could relate to Scrum. "We want the process to be as agile as possible, to a certain level. Waterfall will not suffice". The customer had many ideas regarding the layout and the functionality of the applications, and were not sure what to include. This leading to a situation where spending time making a detailed requirement specification and locking down all the details was pointless.

The customer was likely to make changes to the initial requirements once the first plan was ready. Secondly, the team was way more eager to try out an agile methodology rather than to use waterfall. In the choice between Kanban and Scrum, we lacked the proper knowledge to set those two apart, and had to do some extra investigation in order to fully understand what they meant. After reading about both Scrum and Kanban, the Kanban method did not make very much sense to us. The simple fact that Scrum is highly recommended by real-life developers, is a good argument for choosing Scrum over any other methodology. We were in the opinion that Scrum would suit our project the best, and therefore chose to use Scrum over the other methodologies.

4.2.1 Sprints

This section gives a short description of how the Scrum development method was used in the project. For a general explanation of Scrum see Section 4.1.2.

Sprint duration

We decided on having 14 day sprints. After discussions with the customer, we agreed that this would be a suitable duration, due to the fact that the documentation needed for each sprint would be time consuming for shorter sprints.

Sprint Planning Meeting

To start each sprint, we held a sprint planning meeting. During this meeting, we discussed which user stories/epics from the sprint backlog should be worked on during the sprint. The reason behind such a meeting is to make sure the team is up-to-date on the goals for the following sprint. To decide what user stories/epics should be chosen, the priorities given by the customer was used as a pinpoint. If the customer wanted to make any changes during a sprint, the changes were noted and discussed during the next sprint planning meeting.

Daily Standup

The daily standups (also commonly known as daily scrum meeting) were held on Mondays, Wednesdays and Fridays. The team decided on this semi-daily recurrence since not all team members were able to work on the project every day. During the standup meetings all team members would answer three questions: what have you done since our last meeting, what will you work on until the next meeting, and what problems did occur since our last meeting?

Answering these questions gave a certain status update, and made it easier to re-assign team members to tasks if needed. During the standups all technical discussions were discouraged. If any technical questions arose, the people involved would discuss this after the meeting, to make sure they were not wasting other people's time. Each standup had a max allowed length of 15 minutes.

Sprint retrospective

The sprint retrospective is the written conclusion of the sprint. A meeting was held at the end of each sprint, discussing the results throughout the sprint, both finished and unfinished tasks. The tasks not completed were moved to the next sprint, and the reason for the task not being completed was stated in the sprint report.

The sprint report also includes an update of the sprint backlog, along with an overview of how much time was spent on each task, making it easy to compare to the time estimate.

The sprint retrospective also contains a burndown chart, giving a visual representation of how the team worked during the sprint.

For each sprint we answered the following questions:

- What went well?
- What shall we start doing?
- What could have gone better?
- What should we stop doing?

Explanation of Sprint Backlog

The sprint backlog is a task management tool to document and ensure the progress of the sprint. Each task the team chooses to focus on in the sprint is enlisted. The task is given an ID and already has a name. The Function number is an hour-independent number telling how difficult the team expects the task to be. The base number represents how many hours the team expects to work to finish one story point. The base number multiplied with the function number for a task gives the estimated work hours needed to finish a task.

The base number may change from sprint to sprint, but not during a sprint. The team did an evaluation of the base number in advance of each sprint, to make good estimates.

The name column is used to keep track of who is responsible for the task. This may change during the sprint, but the sprint backlog should always be showing the correct info.

Based on how many team members are available and how many work hours they may put in, the team gives an expected decrease of the story points left. This is reflected in the sprint burndown chart for each sprint, as a straight decreasing line.

Chapter 5

Requirement Specifications

This chapter describes the requirements for the system. Section 5.1 gives an overview of the relevant use cases for the system, in tabular format. Section 5.2 gives an overview of the functional requirements for the system in a textual format.

5.1 Use Cases

A use case is a description of a functionality of the system and shows the interaction that the users have with the applications. A use case diagram usually consists of an actor representing user or responder to the system, and a use case which represent the interaction alternatives for each user to the system. From these use case diagrams it is easy to capture the behavioural requirements of the system. We chose to have one large use case diagram for each actor (Figure 5.1, 5.2), and then have more detailed textual descriptions for each use case in the diagrams. This is given in Section 5.1.2 and Section 5.1.3.

5.1.1 Actors

An actor is an external participant to the system, doing some kind of action on the system. The actor can both request information and give information to the system, and in a general sense it can both be a regular user and a saboteur or a misuser.

In this case there are the children/patients and the next-of-kin which each interact with their own system interface. The server is also included as an actor even though it is not a part of the system, this is to make visual how the server receives and sends information and which information it is interested in. The different actors are shown in Table 5.1.1

5.1.2 Textual Use Cases for GAPP

In this section, the use cases will be described in more detail. Each textual use case has one main actor, which is the one actor to perform the tasks in each case and is often referred to as the user. The usual flow of events is how the system most commonly will be used. The variations field are different possibilities for more rare flows of events. Some use cases does not have any variations which means there is only one way to perform the specified interaction on that function.

Tables 5.2 through 5.10 are the textual use cases for GAPP.

5.1.3 Textual Use Cases for CAPP

Tables 5.11 through 5.14 are the textual use cases for CAPP.

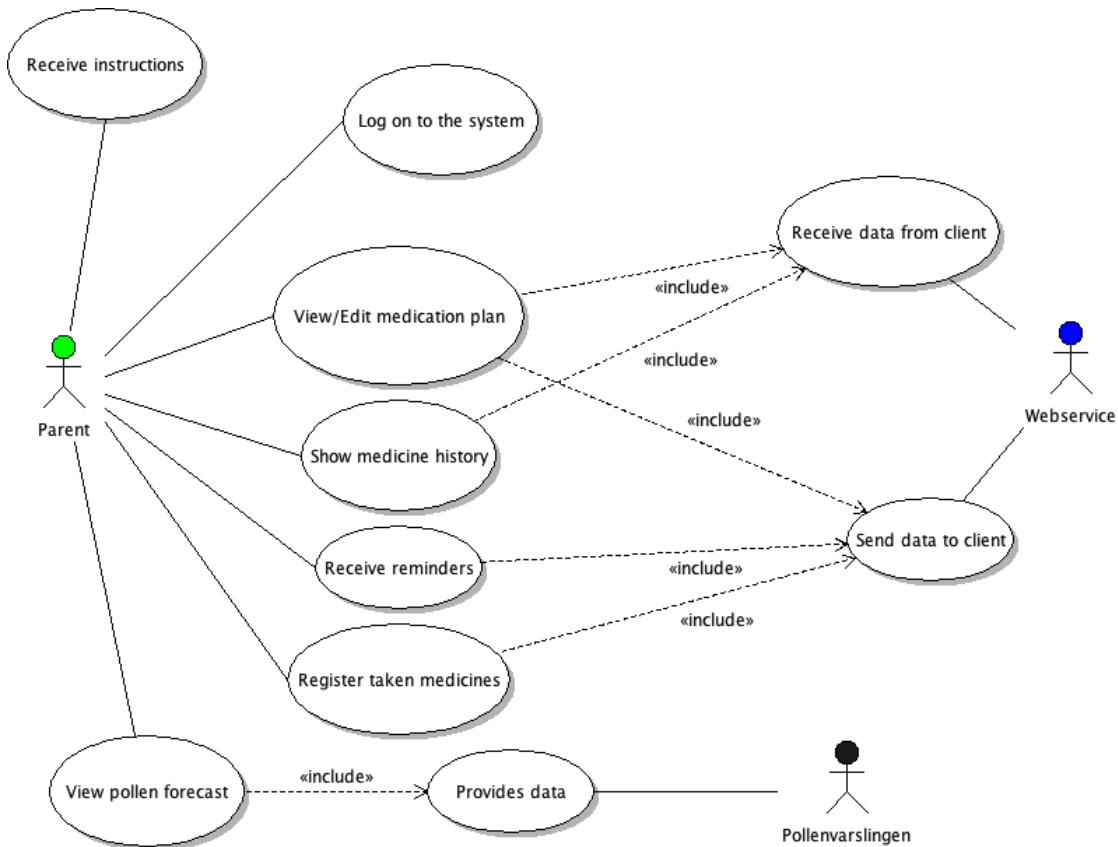


Figure 5.1: Use Case diagram for GAPP

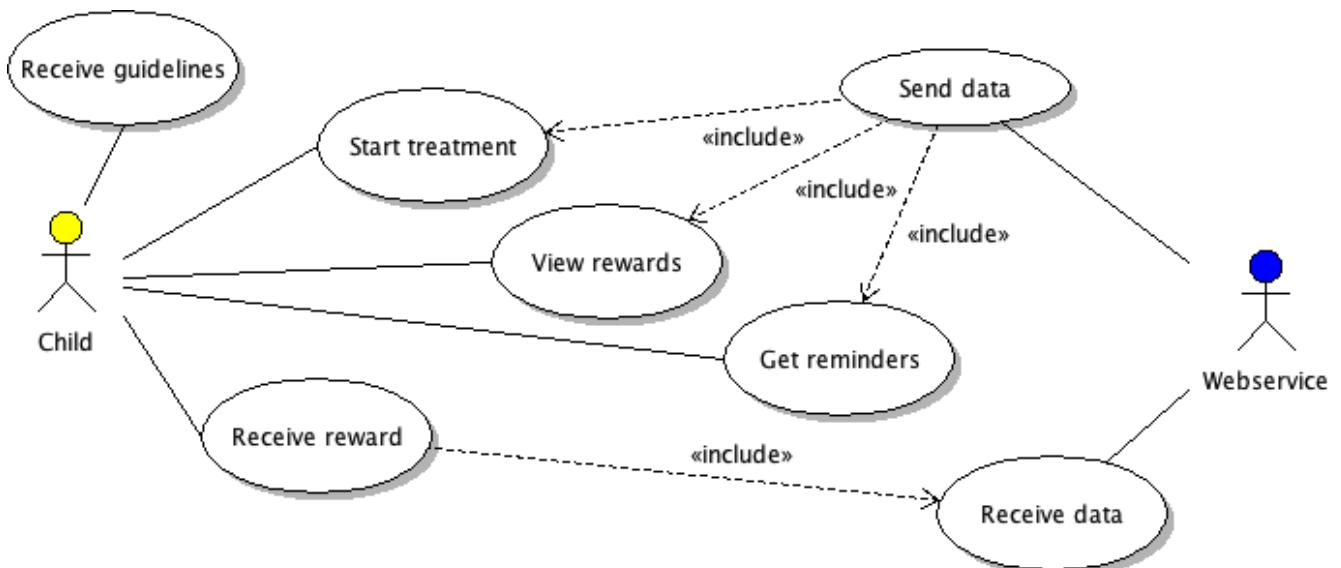


Figure 5.2: Use Case diagram for CAPP

Actorname	Icon	Description
Child	 Child	The child shall interact with CAPP.
Next of kin	 Parent	The next of kind shall interact with GAPP. We have used Parent to denote next of kin in our use case diagrams
Pollenvarslingen	 Pollenvarslingen	“Pollenvarslingen” (“Pollen forecast”) is an external system owned by NAAF that provides GAPP with pollen data
Webservice	 Webservice	Webservice is the connection between the database and the applications, and runs on a server.

Table 5.1: Actors within the system

Use Case	GAPP 1. Log in
Primary actor	Patient, Medical Personnel and Next-of-kin
Trigger	Starting application
Preconditions	User not logged
Postconditions	User logged in
Normal flow of events	1. The user opens the app on an Android device.
Variations	–

Table 5.2: Use Case 1 for GAPP, log in

Use Case	GAPP 2. Change health status
Primary actor	Next-of-kin and patient
Trigger	Pressing the smiley face indicating daily state
Preconditions	Patient's health status has changed
Postconditions	Patient's health status in the application and in real life match
Normal flow of events	<ol style="list-style-type: none"> 1. The user presses the smiley face on the health state screen. 2. The menu for choosing health status pops up. 3. The user presses the according smiley face.
Variations	—

Table 5.3: Use Case 2 for GAPP, change status

Use Case	GAPP 3. See log of daily health status
Primary actor	Patient, medical personnel or next-of-kin
Trigger	Patient, medical personnel or next-of-kin wanting to look at the log for the patient's health status
Preconditions	The application is open and running
Postconditions	The health status log is shown on screen
Normal flow of events	<ol style="list-style-type: none"> 1. The user presses the log icon in the main menu. 2. The log showing the daily health status opens. 3. The log is shown on screen.
Variations	—

Table 5.4: Use Case 3 for GAPP, log

Use Case	GAPP 4. See pollen forecast for today
Primary actor	Patient, medical personnel or next-of-kin
Trigger	The user opens the log feature
Preconditions	The smartphone has an internet connection.
Postconditions	The pollen feed is shown on screen
Normal flow of events	<ol style="list-style-type: none"> 1. The user presses the log icon in the main menu. 2. The log showing daily status, pollen feed and medicine taken opens. 3. The log, with the pollen feed is shown on screen.
Variations	—

Table 5.5: Use Case 4 for GAPP, Pollen feed

Use Case	GAPP 5. Look at guidelines for medication
Primary actor	Patient or next-of-kin
Trigger	The patient or next-of-kin is interested in viewing guidelines and information for a certain medicine
Preconditions	The application is running. NAAF has provided guidelines and information for the medicine. The information is downloaded and stored on the phone.
Postconditions	The guidelines is shown on screen. The user is able to view them.
Normal flow of events	<ol style="list-style-type: none"> 1. The user presses the manual button in the main menu. 2. The manual menu with the different medicines is shown. 3. The user chooses which medicines he/she wants more information about. 4. The adjacent information is shown on screen.
Variations	—

Table 5.6: Use Case 5 for GAPP, guidelines

Use Case	GAPP 6. Look at guidelines for how to do a treatment
Primary actor	Patient and next-of-kin
Trigger	The patient or next-of-kin is interested in viewing guidelines for how to do a treatment.
Preconditions	The application is running
Postconditions	The user has opened the guidelines and was able to scroll through the pictures
Normal flow of events	<ol style="list-style-type: none"> 1. The user presses "Manual" in the main menu. 2. The application opens the Manual and the first picture is shown. 3. The user scrolls through the different pictures.
Variations	—

Table 5.7: Use Case 6 for GAPP, reward

Use Case	GAPP 7. Edit medication settings
Primary actor	Next-of-kin
Trigger	Voluntarily
Preconditions	
Postconditions	Settings has been edited and saved
Normal flow of events	<ol style="list-style-type: none"> 1. The user clicks the "Instillinger" (settings) button. 2. The user is shown a settings page containing means of editing different functionalities. 3. The user clicks "Lagre" (save) button.
Variations	—

Table 5.8: Use Case 7 for GAPP, medication settings

Use Case	GAPP 8. Register medication taken
Primary actor	Next-of-kin
Trigger	The patient has taken medicine without using the application as help. Next-of-kin wants to register the medicine taken
Preconditions	
Postconditions	The medication is logged and saved in the database
Normal flow of events	<ol style="list-style-type: none"> 1. The user opens the log. 2. The user presses the day he/she wants to register a treatment to. 3. The user presses "Etterregistrer" (late-register). 4. The user chooses the medication he/she wants to register. 5. The user presses "lagre" (save).
Variations	—

Table 5.9: Use Case 8 for GAPP, register medication

Use Case	GAPP 9. Remind user to take their medication.
Primary actor	System
Trigger	The predefined time for a reminder is reached.
Preconditions	The user has set up a medication plan and entered when he/she wants to be reminded
Postconditions	An alarm is set off, reminding the user to take his/her medicine
Normal flow of events	<ol style="list-style-type: none"> 1. A reminder is invoked, in the form of an alarm. 2. The user turns off the alarm. 3. The user enters CAPP and starts treatment.
Variations	—

Table 5.10: Use Case 9 for GAPP, reminder

Use Case	CAPP 1. Log in
Primary actor	Patient, Medical Personnel and Next-of-kin
Trigger	Starting application
Preconditions	User not logged
Postconditions	User logged in
Normal flow of events	<ol style="list-style-type: none"> 1. The user opens the app on the Android-based phone.
Variations	—

Table 5.11: Use Case 1 for CAPP, log in

Use Case	CAPP 2. Look at guidelines for how to do a treatment
Primary actor	Patient and next-of-kin
Trigger	The patient or next-of-kin is interested in viewing guidelines for how to do a treatment.
Preconditions	The application is running
Postconditions	The user has opened the guidelines and was able to scroll through the pictures
Normal flow of events	<ol style="list-style-type: none"> 1. The user presses "Manual" in the main menu. 2. The application opens the Manual and the first picture is shown. 3. The user scrolls through the different pictures.
Variations	—

Table 5.12: Use Case 2 for CAPP, reward

Use Case	CAPP 3. Start treatment
Primary actor	Patient and Next-of-kin
Trigger	The patient has an accute asthma attack, or the scheduled daily hour for medication is reached
Preconditions	The patient is in need of medicine
Postconditions	The treatment process is started
Normal flow of events	<ol style="list-style-type: none"> 1. The patient, or next-of-kin, presses the medication button in the application.
Variations	—

Table 5.13: Use Case 3 for CAPP, start treatment

Use Case	CAPP 4. Give reward after treatment
Primary actor	Patient and next-of-kin
Trigger	Treatment is finished
Preconditions	The treatment process is finished.
Postconditions	The user is awarded with a reward as an in-app item
Normal flow of events	<ol style="list-style-type: none"> 1. The treatment finishes. 2. The application gives the user a reward as an in-app item. 3. The statistics log and the rewards counter is updated.
Variations	–

Table 5.14: Use Case 4 for CAPP, reward

5.2 Functional Requirements

As we are developing three separate applications, we found it best to consider the functional requirements separately. We will use the abbreviations PFR (Parent Functional Requirement), CFR (Child Functional Requirement) and KFR (Karotz Functional Requirement).

5.2.1 GAPP - Guardian Application

PFR 1 - Medication plan

The application should make it easier for parents and responsible adults to keep track of the medication plan of their children.

Priority: High

PFR 2 - Notifications

When a child needs to take a preventive medication, the application should send a notification to remind the parents or other responsible adults, either through the smartphone/tablet, through Email, or via Karotz.
Priority: High

PFR 2.1 - Settings for notifications

The user should be able to choose settings for notifications. Example for settings to be made are: Time for notification, Notification appearance, and so on.

Priority: Medium

PFR 2.2 - Notification to change condition

The user should get a weekly reminder to update the condition of the child, if you're outside the default medicationplan.

Priority: Medium

PFR 3 - Families

The application should support several children.

Priority: Low

PFR 4 - Guidelines

The application should provide guidelines for how to use a medicine correctly, and how to do a treatment correctly.

Priority: High

PFR 4.1 - Guidelines from NAAF

The application should provide support for changeable guidelines from NAAF in form of series of pictures, text or animations/movies.

Priority: Medium

PFR 5 - Keep records of condition

The application should be able to keep track of the child's previous medical condition (in terms of days). The condition is to be set by a user, and varies from Green, Orange and Red zone as described in Appendix 3.1.1.

Priority: High

PFR 6 - Pollen forecast

The application should be able to use pollen forecasts to warn parents about possible bad days. The forecast should be included in the log.

Priority: Medium

PFR 7 - Screen sizes

The application should have support for different screen-sizes.

Priority: Low

5.2.2 CAPP - Children's Application

CFR 1 - Distraction

The user should be able to choose a distraction during medication. This could be via an external application, a video or one of the Karotz in the household.

Priority: High

CFR 2 - Rewards

When a child is done with a treatment, he/she should get a reward.

Priority: High

CFR 2.1 - Rewards

The child should feel that the reward gives something back, for instance be able to buy something from an avatar-shop, etc.

Priority: Low

CFR 3 - Screen sizes

The application should have support for different screen-sizes.

Priority: Low

CFR 4 - Avatar

The application should have an avatar for each child, that can be chosen by the child, and be customized through a shop, providing different clothes and items a child could collect. The shop should use the rewards as currency.

Priority: Low

CFR 5 - Child friendly instructions

The application should give instructions to a child, that a child can easily comprehend. Both as part of the distraction, and as a separate part of the application.

Priority: High

5.2.3 Karotz Application

KFR1 - Notification

The application should make children aware of medication-time by playing sound, movement of ears and through the color of the Karotz.

Priority: High

KFR2 - Distraction

The application should be able to distract children when taking medication. This should be through movement of ears and ability to play some sound (music, book reading, etc.)

Priority: High

KFR3 - Reward

The application should reward children by telling how many stars the child earned. The stars should be stored in the database and sync with the application.

Priority: High

KFR4 - Register use of medicine

The application should allow the child to register when it is done taking the medicine, by holding a RFID-chip close to the karotz.

Priority: Medium

KFR5 - Logging

The application should be able to save the health status to the database according to the child's health status.

Priority: Medium

Chapter 6

System Design

In this chapter, we will describe the software architecture of the applications. We will focus on high level abstractions of the system. More detailed class diagrams of the system is found in Appendix E.

6.1 Architectural Description

This project is a prototype to test a concept based on gamification on asthma-treatments. Thus, the main architectural qualities identified for the applications is modifiability and usability. The reasoning behind this decision is that the software should be easy to modify with extended functionality if the concept is proven successful. It should also be intuitive to use, since many of our potential users are children at ages below eight years old. Usability is our top priority.

We chose Model-View-Controller (MVC) as the architectural pattern for GAPP and CAPP, as MVC separates the underlying data models from views, and is proven to enhance modifiability. It is also an integrated part in Android, through a class called `Activity`. We will explain more about activities in Section 6.2.2.

6.2 Software architecture

“The Software Architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.” [19].

In the following section, we will describe the architecture of the system according to the 4+1 View Model introduced in Section 3.8.2.

The relevant stakeholders for the architecture is NAAF, Sykehusapotekene i Midt-Norge, NTNU, in addition to future and present developers. In this section, we will describe the architecture of the system through a Logical View, Development View and a Process View. We have not created a Physical View, as this is not usual for a software that does not run on some sort of special hardware. The different views build upon the use cases defined in Section 5.1

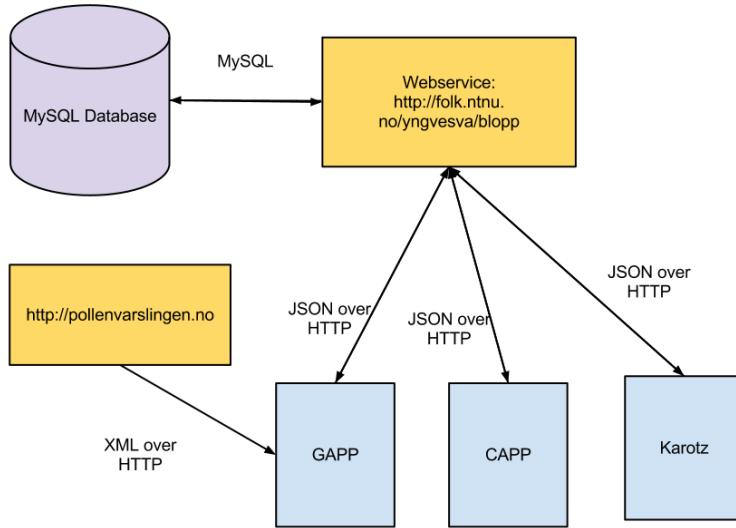


Figure 6.1: Logical View for the system

6.2.1 Logical View

The purpose of the Logical View is to give interested stakeholders an overview of the different modules of the system, and how they communicate between each other.

Figure 6.1 shows the package diagram for the system.

The logical view shows the different modules of the system, and the communication protocols between them.

As mentioned in Section 3.5, NAAF provides a pollen forecast at <http://pollenvarslingen.no> which we intended to integrate with our system. It is currently not connected, because the pollen forecast only operates during pollen season. It is however an important component for GAPP, and should be put in use during further development. We have worked around this fact in order to give the prototype a bit more content. This is described in more details in Appendix E.

The webservice is an important part that ties the different applications together. It required a lot of hours spent on developing it during the early stages, but it definitely paid off later. There were several reasons to provide a webservice for the application:

- Karotz only provides HTTP GET and POST methods for communicating over network,
- Our database server allocated at mysql.stud.ntnu.no is only available from NTNU's LAN (Local area network), which in practice means that a phone that is not connected to eduroam would get access to the database.
- All MySQL queries are allocated at the same place.

6.2.2 Development View

Our development view is represented through a package diagram, and gives a brief overview of the different packages implemented in GAPP and CAPP. We have implemented somewhere around 120 different Java classes, which does not include the Karotz code.

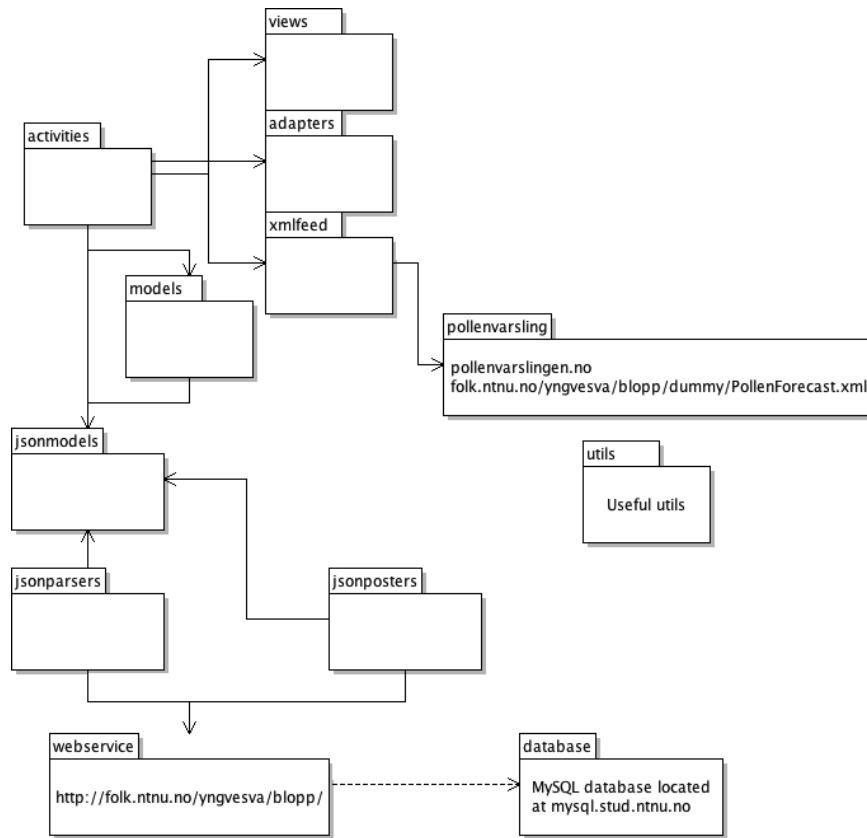


Figure 6.2: Package diagram for GAPP. A line from package X to package Y describes an association from X to Y.

Development View – GAPP

Figure 6.2 show the top level package structure for GAPP. In the logical view, we introduced our webservice. From this webservice, we retrieve data over HTTP GET, using JSON (JavaScript Object Notation). The classes found in jsonposters uses HTTP POST to send data to the webservice, which in turn forwards the information sent to our database.

The diagram also shows a package called `xmlfeed`. This package serves the purpose of parsing XML (Extensible Markup Language) into Java-code. Originally, the idea was to use NAAF's XML-feed. However, this feed is not running during autumn and winter, so we replicated that XML-feed in our own “dummy”-feed¹. This XML-document has similar structure as the original feed, only with test data for the sake of the prototype and proof-of-concept.

The package `activities` contains classes that extends `Activity`, which is an android class that functions as both the View and Controller in MVC. An example of an implemented activity is `MainMenu`.

In order to render listviews and gridviews programmatically, we needed to make a package called `adapters`. An `Adapter` in Android serves the purpose of filling these list- and gridviews with data.

Views in Android can be created in two ways. Either through an XML layout file, or by customising in Java. Most of our views are using XML-files. We have one custom made view (`CalendarView`, used to implement our logging functionality), which is available in the package “views”. The XML-files are not shown in this diagram, but is included in the application’s resource folder, which is standard when working in the Android framework.

We'll explain the content in each package more detailed in the Development View.

¹This XML-file is hosted at: <http://folk.ntnu.no/yngvesva/blopp/dummy/PollenForecast.xml>

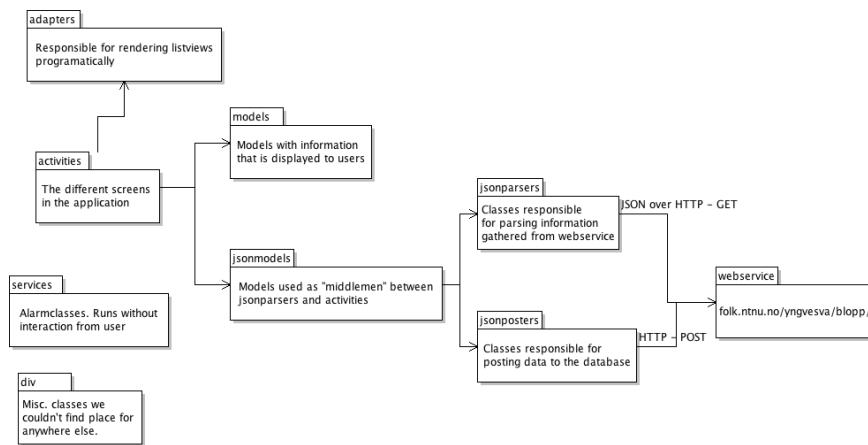


Figure 6.3: Package diagram for CAPP. A line from package X to package Y describes an association from X to Y.

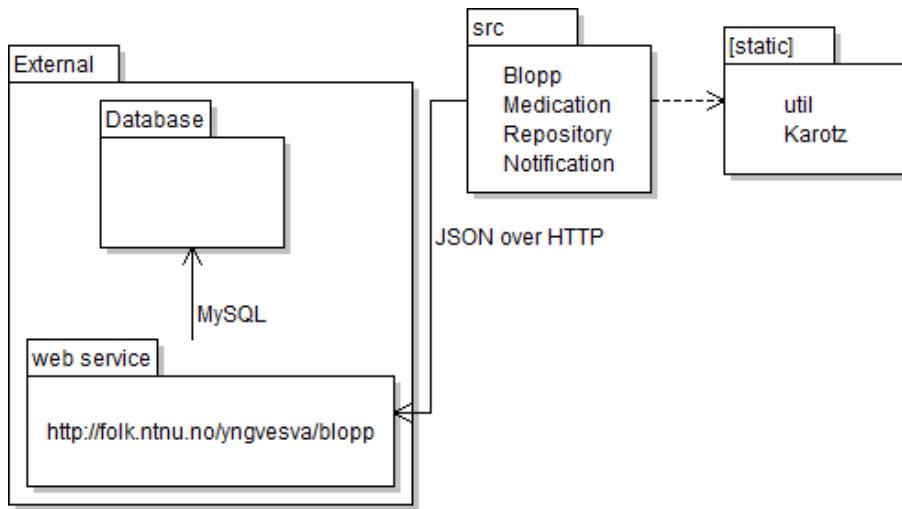


Figure 6.4: Package diagram for the Karotz application

Development View – CAPP

Figure 6.3 shows the package diagram for CAPP. CAPP has very similar architecture as GAPP. The main difference is the “services” package. A service in Android is a task that is executed without any interactions from the user. In our case, we use the services-package to update alarms. CAPP uses the same database and webservice as GAPP, and a lot of the parsers in this application are equal to those found in GAPP.

Development View – Karotz

Figure 6.4 shows the development view of the Kartoz application. The only internal package in the Kartoz application is named “src”. It contains all the main logic for the application, including a repository for connecting to the webservice for connecting to the database, a notification module for setting alarms, a medication module for doing distraction, and a “Blopp” module for keeping track of everything. There is an additional package in the diagram named “[static]”. It contains the Karots class which is inherited from the OS and contains various utilities for controlling output and receiving input for the robot. It has been extended with some helper methods in the implementation. There is also a “util” class which contains additional utility methods that do not belong in “Karotz”. At last, there is indicated an “External” package, which symbolizes all the external services the application connects to. In the case of the Kartoz

application, this includes only the web service that connects to the database.

6.2.3 Process View

The purpose of the Process View is to give the customer an overview of how some of the main functionality has been implemented. We have chosen to do this through Sequence Diagrams. These diagrams shows the main data flow of the system.

We have chosen to include the following functionality:

- Medication completed
- Change of childrens health status

Medication Completed

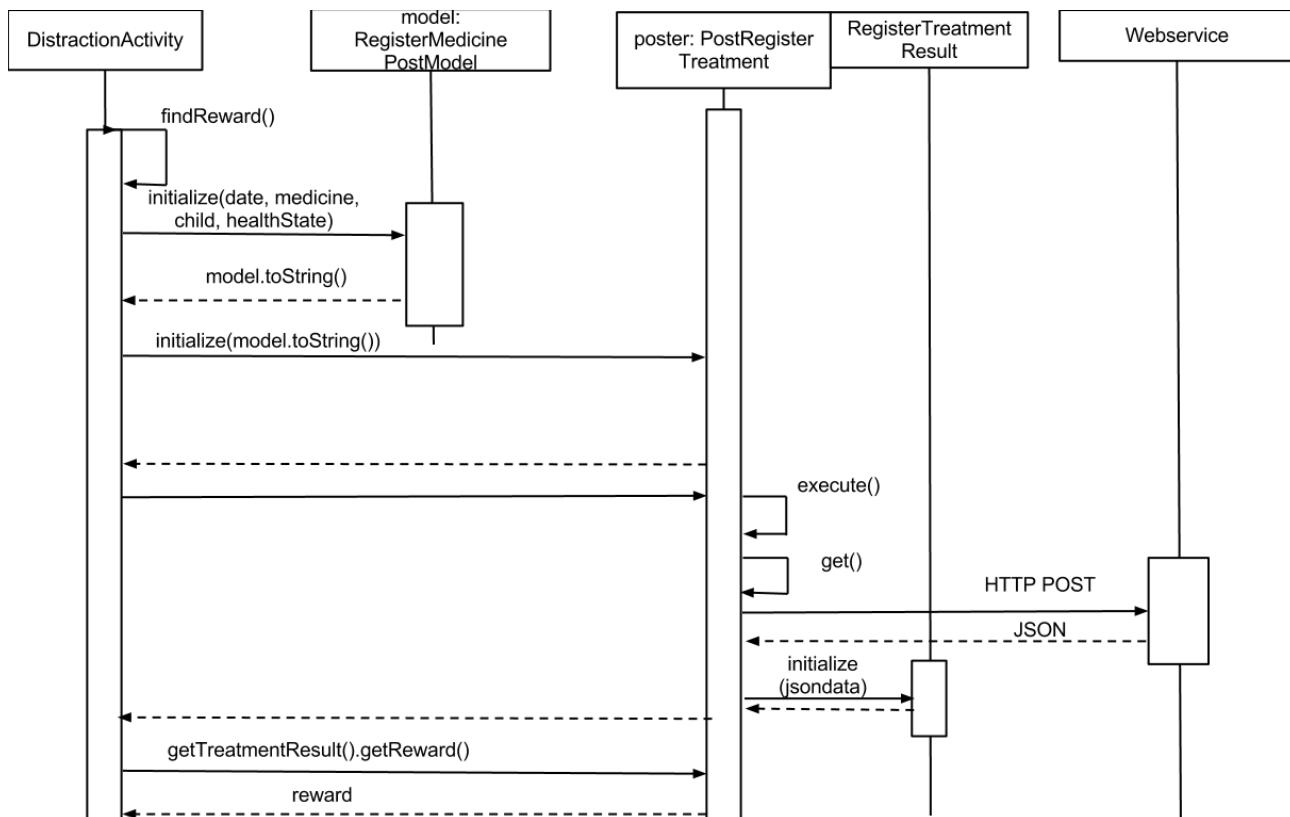


Figure 6.5: Sequence diagram for medication completed

Figure 6.5 shows a sequence diagram for the event of a medication completed. When a medication is completed, `DistractionActivity` creates a new instance of `RegisterMedicinePostModel`, before it uses this instance's `toString()` method to create an instance of `PostRegisterTreatment`. This poster does an HTTP POST to "register_medicine_taken.php", which in turn updates the database. The webservice then returns JSON-formatted data, with the reward included. `RegisterMedicinePostModel` interprets the returned data, and stores the reward in a private variable, which the `DistractionActivity` in turn accesses to show the child the reward collected from the treatment.

Change Of Childrens Health Status

Figure 6.6 shows a sequence diagram for the event of changing a child's health status. The user gives input on one of three checkboxes that `MedicationPlanActivity` receives. The activity then calls creates a new

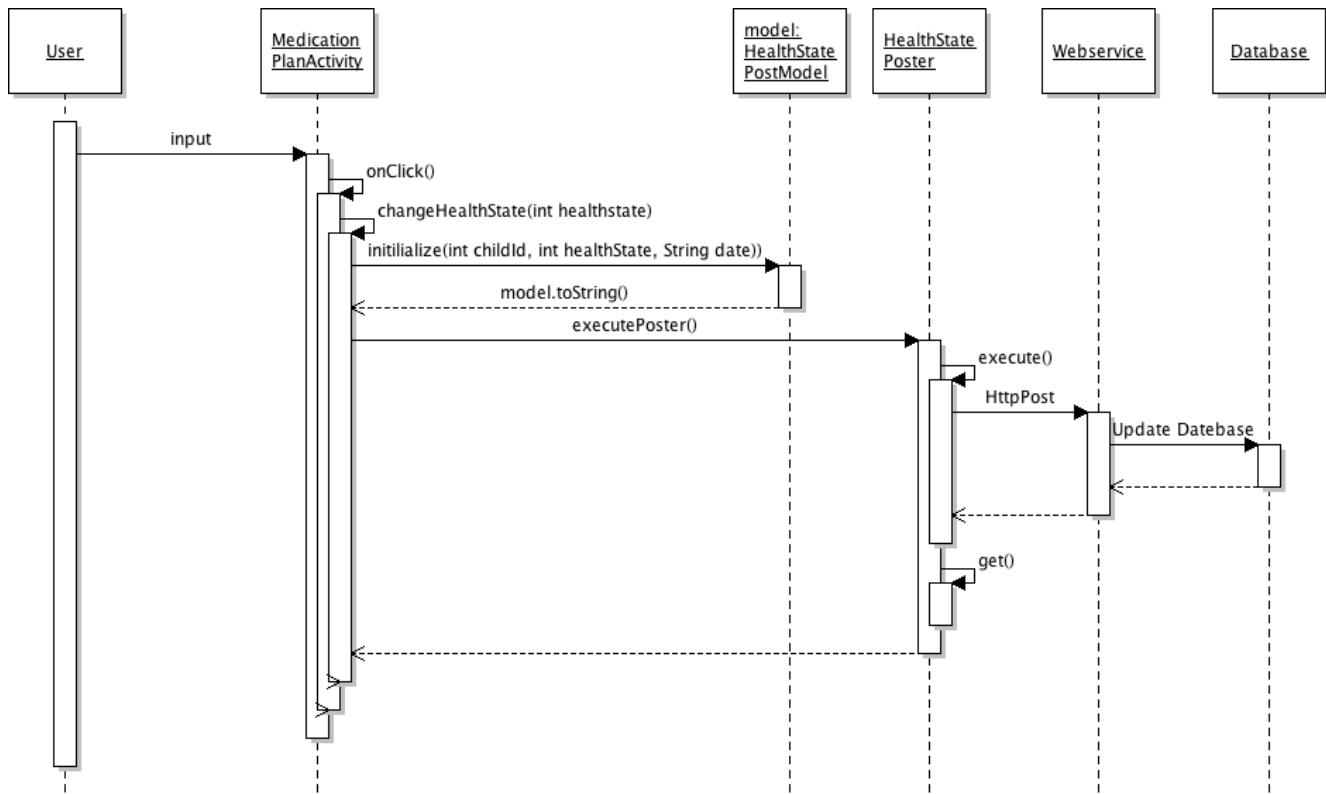


Figure 6.6: Sequence diagram for changing medication plan

instance of `HealthStatePostModel`, and uses this model's `toString()` to execute `HealthStatePoster`. `HealthStatePoster` makes an HTTP POST to “`set_child_state.php`” with appropriate POST parameters. The webservice updates the database, before it returns JSON-formatted data to the `HealthStatePoster`.

Notification and medication on Karotz

Figure 6.7 shows a sequence diagram for notification and medication on the Karotz. The Karotz routinely updates itself by calling a `get` function on the database access layer. The `Repository` object requests that a new notification event is made for the closest medicine dose that should be taken. When the timeout is done, `Notification` calls `startMedication()` in `Medication` in order to start a sequence of events that makes up the distraction sequence. This sequence is represented as a list of actions—called a *manuscript*. The manuscript is interpreted by the `util` module which creates and runs a function based on a premade specification. Each action contains an url to a sound file, and an activator which represents how the next action in the manuscript is triggered. The manuscript is detailed further in Appendix C. When each action in the manuscript is completed, the `Medication` module calls `logMedicineTaken()` in `Repository` once for each dose, and the database is thus updated with the registered medicines.

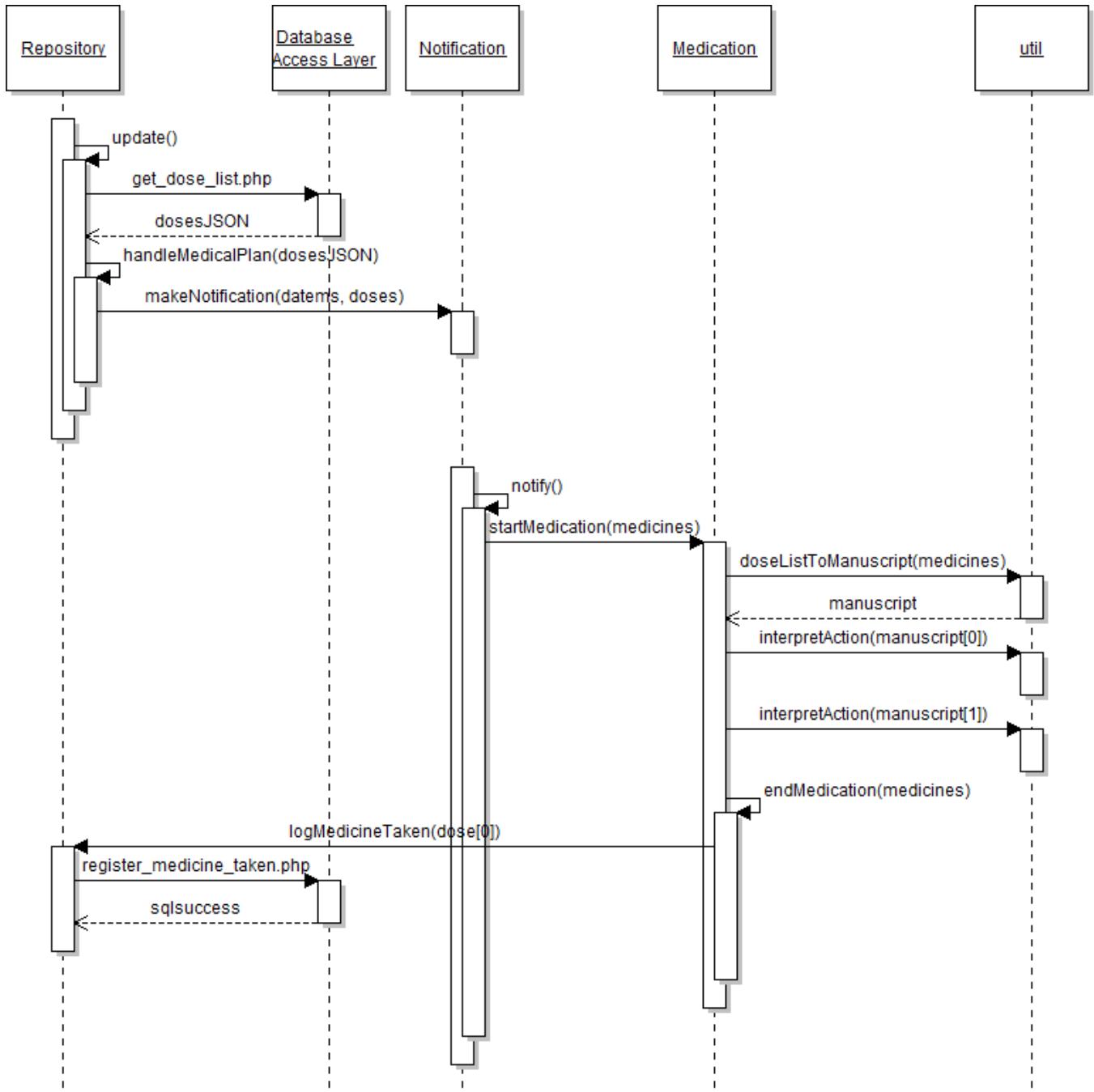


Figure 6.7: Sequence diagram for notification and medication on Karotz

6.3 Architecture Rationale

Choosing MVC has become a somewhat “standard” solution for these types of applications, and it was a successful pattern for our use as well. The choice of making the database accessible through a webservice slowed down performance a bit, but we had to do this, since Karotz only supports access to the database through the webservice. We were also not able to implement proper caching functionality, which could have improved performance severly. More information on this part is found in Chapter 14.

We found it somewhat harder to implement good models for the objects that is retrieved from the webservice, which has created a bit of overhead in the architecture. However, it has been easier to connect CAPP and GAPP together by retrieving data through the same webservice, and it was a working solution once we knew how to access it properly.

It is hard to predict how many users the system can handle once proper functionality for several children is implemented. The webservice is developed with the intention of running at one server only, and so, the scalability of the system is very dependent on the capabilities of this server (how many requests it can handle per time unit). If the software is to be deployed on several servers, the webservice needs to be rewritten.

6.4 Database

The database implementation is based on an ER (entity relationship) diagram, which is illustrated in the diagram in section 6.4.1. It is created based on the final implementation. There is also a separate access layer for modifying and viewing the data through the web.

6.4.1 Database Implementation

The final implemented database architecture is displayed in figure 6.8. The tables related to the avatar: AVATAR, AVATAR_INVENTORIES, SHOP_ITEMS and AVATAR_ITEM_SLOTS are not used in the system because the avatar idea was put on hold. The rows location_latitude and location_longitude in the table CHILDREN are also not used because we did not implement updating the pollen feed based on current location.

The arrows symbolize relations, where the big end is the referenced key and the small end is the foreign key.

6.4.2 Database Access Layer

The database access layer consists of 14 PHP files hosted at <http://folk.ntnu.no/yngvesva/blopp/>:

add_child.php Takes a name, personal number (SSN) and a list of states (integer IDs) that the child can have. Creates an entry in the CHILDREN table for the child, and a medical plan entry. Also creates an entry in CHILD_HEALTH_STATES for all the states the child can have. Returns the generated medical_plan_id.

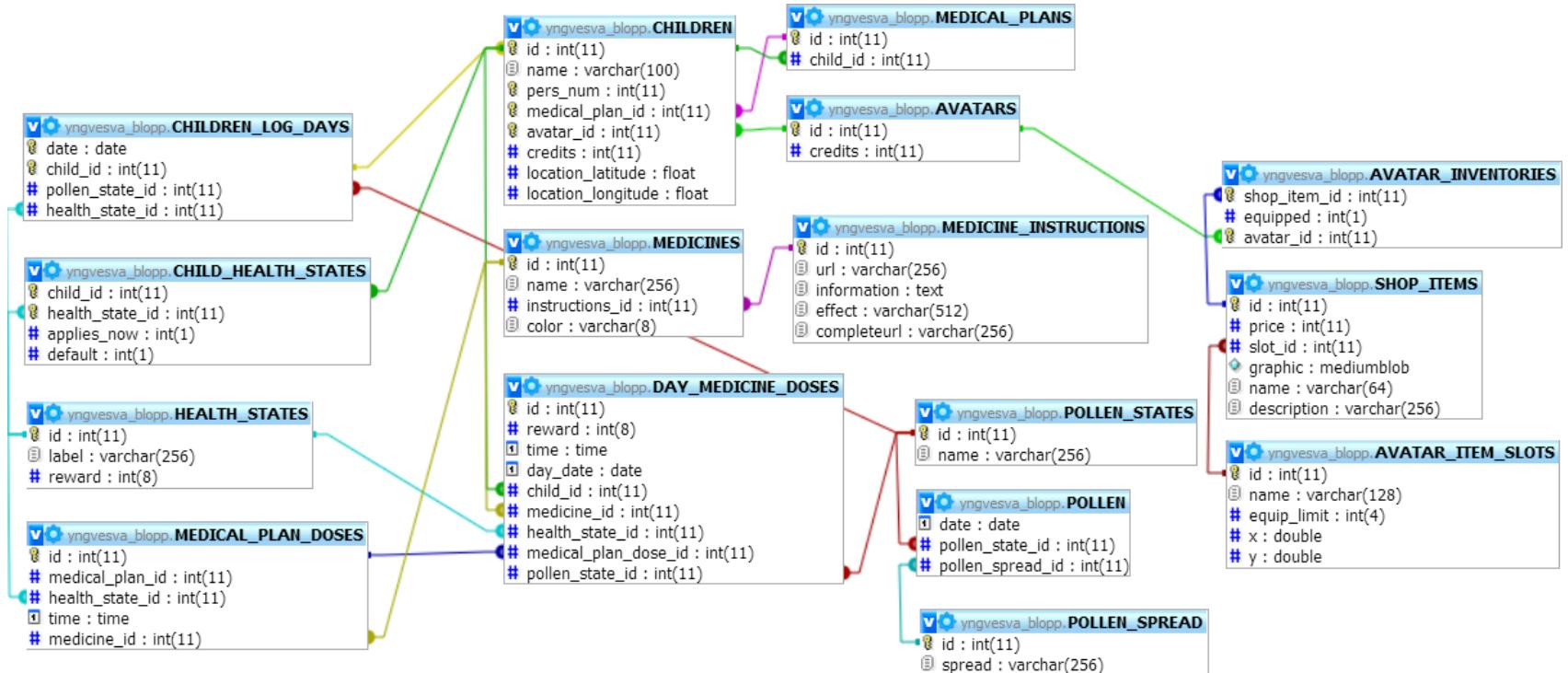
add_plan_dose.php Inserts a new entry in the MEDICAL_PLAN_DOSES table for the given child, with the parameters ‘health state’, ‘dose of the given medicine’ and a timestamp. This is the primary module used to alter medical plans. Returns the ID of the added dose.

dose_is_taken.php Check if a dose of a planned medicine has been taken that day. Takes an id of an entry in the table MEDICAL_PLAN_DOSES as input.

get_available_child_states.php Takes a child_id and returns a list of the labels (colors, names) and IDs of the states the child can have.

get_child.php Takes an ID of a child and returns all the columns for the given ID in the CHILDREN table.

Figure 6.8: Implemented Database Architecture



get_child_state.php Accepts a child ID and returns the ID and label of the current state of the child.

get_doses_for_current_state.php Takes a child ID and returns a list of planned doses of medicines that are not taken that day. The fields of each entry are: `id`, `medical_plan_id`, `health_state_id`, `time`, `medicine_id`, `medicine_karotz_color` and `medicine_name`.

get_instructions.php Get instructions (image, effect description and usage description) for a given medicine by ID.

get_log_days_for_child.php For the calendar in GAPP, it was advantageous to have a database access method that could return a list of days in a month with the child health state for each day, and a list of doses taken on that day. `get_log_days_for_child.php` accomplishes this by using the table `CHILDREN_LOG_DAYS` to find the latest recorded health state before the given month started. Then it iterates through all the days, checking if there are any days in the month where the status changes, and adding all doses, taken from `DAY_MEDICINE_DOSES`, on that day. The method takes a `child_id`, and two optional parameters `month` and `year`. If the month and year are not set, the values for the current days are used.

get_log_for_child.php Returns all registered entries in the table `DAY_MEDICINE_DOSES` for the day for a given child (`id`) during the given month during the given year.

register_medicine_taken.php Register a dose of medicine taken. Accepts a post object with the fields `child_id`, `medicine_id`, `time`, `day_date`, `health_state_id` and `medical_plan_dose_id`. If there is an entry for that dose id that day, the method does nothing and simply returns `unique = false`. Otherwise, it calculates a reward, and updates `DAY_MEDICINE_DOSES` with the entry. Returns the reward for that dose. Then it adds the calculated reward to the child's total credits in the `CHILDREN` table. If no time is given, a default time of 00:00:01 is set.

remove_plan_dose.php Deletes the entry in the table `MEDICAL_PLAN_DOSES` which corresponds to a given id. Returns the number of deleted rows.

remove_plan_medicine_at_time.php During development of GAPP, it was discovered that using a combination of `child_id`, `medicine_id` and `time` as the key for removing elements in the `MEDICAL_PLAN_DOSES` table would be easier than using an ID. Therefore the need for this module arised, and it simply removes all entries in the table that fit the criteria.

set_child_state.php Takes a child ID and a state ID and sets the current state of the given child to the specific state given as input. Also updates the table `CHILDREN_LOG_DAYS` with the given health state.

Chapter 7

Overall Test Plan

The following chapter concerns the overall testplan for the project, and it contains general information about testing and what tests we aimed to perform. The template for our tests can be found in table 7.1, while the tests that were performed can be found in table 7.2.

We test the system to make sure that our applications are easy to use, and works the way they're intended, ensuring that the delivered product fulfills the requirement specifications. We will document the most important tests with a description of the test we performed, what we discovered during the test, and what we did to fix eventual problems that surfaced during the test.

7.1 Test methods

There are different test methods that can be used. The outer points are black-box and white-box testing, but we can also do gray-box testing, a combination of the two.

7.1.1 Black-box testing

This method tests the functionality of the system. Black-box testing means we feed something in and then see if the result we get out is the same as the one we were expecting. This means that knowledge about the code and structure of the system we are testing is unknown and irrelevant to the test. The tests will then be based on external software descriptions like the functional requirements for the system.

7.1.2 White-box testing

This method of testing concerns itself with testing the internal structures of a system. This means knowledge about structure and code is required to run the test, and we require knowledge about the programming to design the test cases. Normally this type of testing is performed at unit level, but can also be used on the system as a whole. The problems detected by white box testing is technical, and it can not detect whether or not a program is fulfilling it's functional requirements.

7.2 Test levels

There are different types and levels of testing. Usability testing focuses on the general, graphical and functional part of the system, how easy the applications is to use for the typical end users. There are low level tests testing the smaller parts of the system, typically classes or methods, and there are high level tests, that tests the system, or parts of the system.

Item	Description
ID	Identifier for the test
Description	Description of the test
Date	Date of the test
Responsible	The person responsible for the test
Subject	The subject being tested (typically a unit or part of the system)
Precondition	The conditions we assume to be in place when the test is started
Steps	The steps to perform
Results	Results after the test was performed

Table 7.1: Test template

7.2.1 Unit testing

The lowest form of testing is unit testing. This is intended to test the smallest units of the system, namely the methods, classes and variables, making sure that each of these parts works as expected.

7.2.2 Module testing

Once the smaller parts of the system has been tested, we also test the coordination between these parts, by testing that the entire module works as we intended.

7.2.3 Integration and System testing

When all of the individually tested modules works as intended, we test that the different parts of the system works together, taking small to medium parts of the system into the test, while the system tests will test the system as a whole. That means that these tests test communication between the different modules and interfaces.

7.3 Testing approach

We decided to use both black box and white box testing for our system. We did two bigger usability tests, where we applied black box testing on both. We would have preferred to do some more usability testing, however, since we had to implement the system from scratch, we decided to start with one paper prototype usability test to get initial thoughts from the potential users. Then we moved on with implementing the system before we again did a bigger usability test, this time with children. We used the implemented applications and the distraction sequence with the Karotz. This meant we had to do a lot of unit and integration testing on the system during the implementation.

All of our tests will follow the template presented in table 7.1.

The tests done during this project is listed in table 7.2, with the testID, brief description of the test and during what part of the project it was done.

ID	Description	Time of test
USABILITY0.1	Paper prototype test	04.09.12 (section 13.3.1)
USABILITY02	Usability testing of the system on children	30.10.12 (table 12.17)
UNIT1.1	Test of GUI for GAPP	17.09.12 (table 8.2)
UNIT1.2	Test of GUI for CAPP	17.09.12 (table 8.3)
UNIT2.1	Test of the CAPP distraction sequence	30.09.12 (table 9.2)
UNIT2.2	Test of the database connection	26.09.12 (table 9.3)
UNIT2.3	Testing of SQL-queries	30.09.12 (table 9.4)
UNIT3.1	Test that alarm is given independently of phone state	14.10.12 (table 10.2)
UNIT3.2	Test that the correct days is colored in the log	09.10.12 (table 10.3)
UNIT3.3	Test that the karotz notification is given at the correct time	09.10.12 (table 10.4)
UNIT3.4	Test that the karotz distraction runs after receiving the notification and starting the sequence	09.10.12 (table 10.5)
UNIT3.5	Test that the notifications with multiple doses makes the correct amount of medications	11.10.12 (table 10.6)
UNIT4.1	Test that the right instructions are downloaded and shown on the instructions menu screen	18.10.12 (table 11.2)
UNIT5.1	Test of the web access module <code>add_child.php</code>	30.10.12 (table 12.2)
UNIT5.2	Test of the web access module <code>add_plan_dose.php</code>	01.11.12 (table 12.4.1)
UNIT5.3	Test of the web access module <code>dose_is_taken.php</code>	01.11.12 (table 12.4)
UNIT5.4	Test of the web access module <code>get_available_child_states.php</code>	01.11.12 (table 12.5)
UNIT5.5	Test of the web access module <code>get_child.php</code> .	01.11.12 (table 12.6)
UNIT5.6	Test of the web access module <code>get_child_state.php</code> .	01.11.12 (table 12.7)
UNIT5.7	Test of the web access module <code>get_doses_for_current_state.php</code> .	01.11.12 (table 12.8)
UNIT5.8	Test of the web access module <code>get_instructions.php</code> .	01.11.12 (table 12.9)
UNIT5.9	Test of the web access module <code>get_log_days_for_child.php</code> .	03.11.12 (table 12.10)
UNIT5.10	Test of the web access module <code>get_log_for_child.php</code> .	03.11.12 (table 12.11)
UNIT5.11	Test of the web access module <code>get_plan.php</code> .	04.11.12 (table 12.12)
UNIT5.12	Test of the web access module <code>register_medicine_taken.php</code> .	04.11.12 (table 12.13)
UNIT5.13	Test of the web access module <code>remove_plan_dose.php</code> .	04.11.12 (table 12.14)
UNIT5.14	Test of the web access module <code>remove_plan_medicine_at_time.php</code> .	04.11.12 (table 12.15)
UNIT5.15	Test of the web access module <code>set_child_state.php</code> .	05.11.12 (table 12.16)
INTEGRATION5.1	Test of CAPPs alarm and distraction sequences.	06.11.12 (table 12.18)
INTEGRATION5.2	Testing that the log updates correctly based on registered medication and pollen feed.	06.11.12 (table 12.19)
INTEGRATION5.3	Testing that the medicationplans is correctly registered to their respective healthstates.	05.11.12 (table 12.20)

Table 7.2: List of tests

Chapter 8

Sprint 1

The development sprints are the most important parts of the scrum development process. Therefore the sprint reports are a vital part of our final report of the project. The following chapter presents an overview of how we planned, worked and completed Sprint 1.

Sprint 1 started on 5th of September and ended on 16th of September. We had not yet decided on how long each sprint should be, and chose a duration of 11 days for Sprint 1. The reason behind this was high chance of changes in the applications.

The chapter is divided into five parts, starting with the overall plan for the sprint in Section 8.1. Followed by the sprint backlog in Section 8.2, which lists up the tasks that have been chosen for the sprint. Section 8.3 will focus on the work made to the GUI, the logic implemented in the applications and the work done to the database. The chapter ends with what have been tested and the corresponding results in Section 8.4 and a sprint review in Section 8.5.

8.1 Sprint Plan

The plan for Sprint 1 was to set up the project, get the main screen up and running and make screens for the distraction, without any animations. Finishes to the preliminary studies were also to be made. We chose such a huge widespread of tasks from the backlog, since we wanted a skeleton of the application up and running. Some connections between the different elements of the application were to be implemented, but mainly we focused on the user interface. During the sprint we mainly did manual testing, since there were not many changes to logical elements.

8.2 Sprint backlog

The table 8.1 shows the sprint backlog for sprint 1, which is a smaller part of the Product Backlog. The goal is to implement all tasks assigned to that sprint, and in this manner implement all task in the product backlog.

We also decided to continuously write on the report, attend lectures and hold advisor and customer meetings during the sprints. These are not included in the backlog.

8.3 Design and Implementation

This section contains a description of the changes done in the respective areas of the development.

In Sprint 1 the main focus was to get the project up and running in Eclipse and learning more about how Android works and getting used to Git. After a customer meeting in the middle of the sprint, we had to change focus for a short period of time, since the customer wanted to see more documentation and

# ID	Task	Story points	Estimated hours	Responsible
1.1	Add navigation between activities in main menu, for GAPP	0.5	5	Esben
1.2	Add element for showing current medication plan	0.5	5	
1.3	Add navigation between activities in main menu, for CAPP	0.5	5	Aleksander
1.4	Make GUI for menu screen in GAPP	2	20	Jørgen
1.5	Make GUI for menu screen in CAPP	1.5	15	Aleksander
1.6	Add the adult app as a project to the repository	0.5	5	Yngve
2.1	Make a screen for starting the treatment	0.5	5	Eirik
2.2	Make a screen for finished treatment	0.5	5	Eirik
2.3	Make a screen for distraction	1	10	Eirik
2.4	Make some kind of distraction through the Karotz	3	30	
2.5	Make a method for saving a treatment to the database	3	30	
2.6	Make a method for starting treatment through Karotz	1	10	
2.7	Make the database for saving treatments, medicine and avatar	10	100	Yngve
SUM		24.5	245	

Table 8.1: Backlog for sprint 1

planning. After delivering a more concrete plan for the development, risk assesments and other important documents, we went back to programming.

We focused on working on smaller parts of the system at the time, rather than starting too broad. This way we may deliver a working system, with some functionality, rather than a system with much not-completely working functionality. During this sprint the only feedback on the user interface was from the customers.

8.3.1 User Interface Layer

Before the sprint we had done one user testing on a paper prototype. The results were used as an inspiration for the GUI for the time being. The project had earlier been divided into two applications, one meant for children, CAPP (Child APPlication) and one meant for adults, GAPP (Guardian APPlication). GAPP should consist of a log, a settings menu, an information menu and an instruction option for treatment. CAPP should have a menu for the avatar, a shop for buying stuff for the avatar and an instruction option for treatment.

We focused on the GUI of the main menu and navigating between different options, in both seperate applications.

Item	Description
ID	UNIT1.1
Description	Test of GUI for GAPP
Date	17.09.2012
Responsible	Esben
Subject	The MainMenu class in the no.blopp.app.activity project.
Precondition	Early version of the application, runable on the android virtual device.
Steps	For each button on the MainMenu: 1. Press the button. 2. Confirm that you are moved to the appropriate screen. 3. return to the main menu.
Results	As expected all buttons on the MainMenu directed us to the correct screen

Table 8.2: Unit test 1.1, GAPP GUI

8.3.2 Application Logic Layer

There wasn't done any changes to the applications logic layers. The focus of the sprint was mainly on GUI and the database.

8.3.3 Data Persistence Layer

During the sprint, the database was created according to the existing specifications. Based on an ER diagram that detailed the structure of tables in the database, it was implemented in MySQL on NTNU's MySQL server.

During the sprint it was discussed whether the database should include internal procedures for adding, updating and removing data. The alternative would be to rely in direct MySQL queries. The advantage of using internal database procedures is that the access procedures and the database itself are strongly related so keeping them separated would cause the system to be more distributed unnecessary, and therefore more difficult to maintain. The advantage of keeping the access in a separate layer, or programming each SQL query directly in the applications is that it would possibly save developer resources since it is easier to write direct MySQL queries than processes. This question was still left undecided at the end of sprint 1.

8.4 Testing and Results

This sections presents the testing done during the sprint and the results of the tests done.

8.4.1 Testing

During this early sprint our amount of testing was limited to two simple tests, where we tested that the graphical user interface acted the way we desired it to, for both CAPP and GAPP, respectively.

8.4.2 Results

The work done in Sprint 1 resulted in a skeleton for both separate applications. It was possible to navigate

Item	Description
ID	UNIT1.2
Description	Test of GUI for CAPP
Date	17.09.2012
Responsible	Aleksander
Subject	The MainMenu class in the no.blopp.app.med.activity project
Precondition	Early version of the application, runable on the android virtual device
Steps	For each button on the MainMenu: 1. Press the button. 2. Confirm that you are moved to the appropriate screen. 3. return to the main menu.
Results	As expected all buttons on the MainMenu directed us to the correct screen

Table 8.3: Unit test 1.2, CAPP GUI

8.5 Sprint Retrospective

This section contains an evaluation of the sprint. The evaluation is done mainly by the us, but feedback from the customers are added to the retrospect.

What went well?

The project is up and running. The developers have learned a lot more about Android and Android programming. The database is running, even though the database is not tested. The documentation for the database and the plan for the project is complete.

What shall we start doing?

We should do more testing. Testing the database and the logic of the applications should be done as the code is written. We have discussed having programming sessions with all developers present, and this is something we should start doing. The customers have asked for more involvement with the documentation and the progress of the development, these requests should be fulfilled during the following sprints.

What could have gone better?

Our team should involve the customer more on different parts of the development. We should have had early sessions to learn Android together, rather than apart. Our team should start sending meeting invitations earlier. The invitations should be sent at least 24 hours in advance of internal meetings, and expectedly 48 hours in advance. For external meetings, our team should send meeting invitations 72 hours in advance.

What should we stop doing?

Nothing went completely wrong during this sprint.

8.5.1 Sprint Burndown Chart

In this section, an overview over the sprint backlog and the time spent on each backlog task in addition to the estimate from earlier. The tasks we haven't worked on are shown without a responsible developer and without time spent. The amount of work estimated to fill this sprint was highly over estimated, due

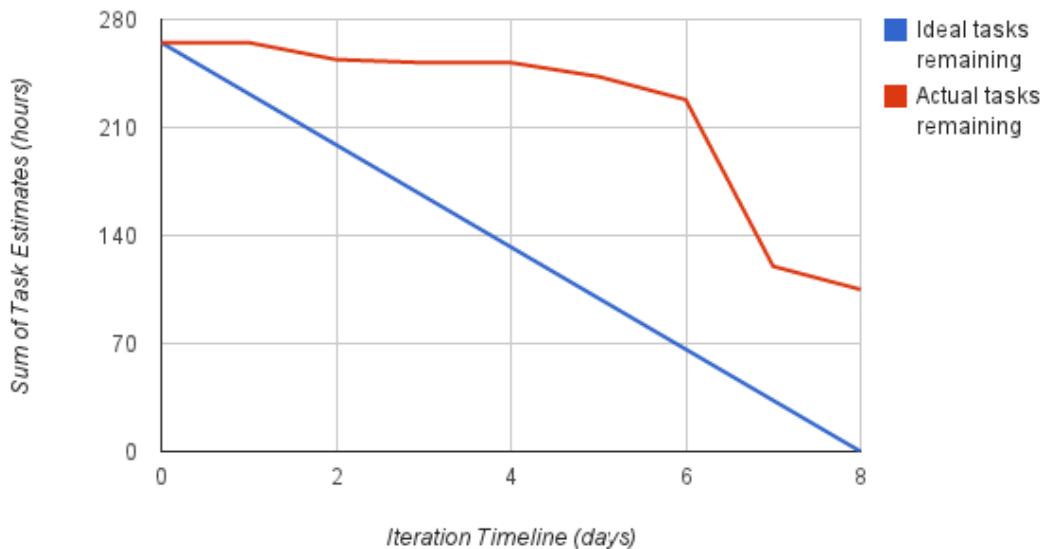


Figure 8.1: Sprint 1 burndown chart

to using a very high hour value per story point. The progression was as expected despite a slow start due to extra needs of documentation from the customer. The use of new technology and a new development methodology did also affect the pace of the development. Advisor and customer meetings were held as usual during this sprint.

Table 8.1 and table 8.4 show the burndown chart from this sprint. A burndown chart is a helpful tool for reflecting upon what have been done and also give a better overview to how the work was done. The burndown chart from this sprint looks bad at first, but then turns out better. The reason to the slow progress was that we were told to work on documentation and other tasks not in the sprint backlog.

Our base goal is to finish 175 estimated work hours during each sprint, which with a base multiplier of 10 corresponds to $175/10 = 17.5$ story points. Even though only 51 work hours were documented in the sprint, there are also 105 work hours not done. This means that out of the total 24.5 story points planned for the sprint, there are $105/10 = 10.5$ story points left. In other words, we completed $24.5 - 10.5 = 14$ of the planned story points, which is only $17.5 - 14 = 3.5$ less than our goal. This is a fairly good result. The result for the dramatic decline in storypoints left midways is due to one task being overestimated. We have learned from this, and believe we will make better estimations in the future.

# ID	Task	Story points	Estimated hours	Actual Hours	Time left	Responsible
1.1	Add navigation between activities in main menu, for GAPP	0.5	5	5	0	Esben
1.2	Add element for showing current medication plan	0.5	5	0	5	
1.3	Add navigation between activities in main menu, for CAPP	0.5	5	5.5	0	Aleksander
1.4	Make GUI for menu screen in GAPP	2	20	3	10	Jørgen
1.5	Make GUI for menu screen in CAPP	1.5	15	3.5	0	Aleksander
1.6	Add GAPP as a project to the repository	0.5	5	1	0	
2.1	Make a screen for starting the treatment	0.5	5	6	0	Yngve
2.2	Make a screen for finished treatment	0.5	5	6	0	Eirik
2.3	Make a screen for distraction	1	10	6	0	Eirik
2.4	Make some kind of distraction through the Karotz	3	30	0	30	Eirik
2.5	Make a method for saving a treatment to the database	3	30	0	30	
2.6	Make a method for starting treatment through Karotz	1	10	0	30	
2.7	Make the database for saving treatments, medicine and avatar	10	100	15	0	Yngve
SUM		24.5	245	51	105	

Table 8.4: Sprint 1 burndown chart

Chapter 9

Sprint 2

The following section presents an overview of how we planned, worked and completed sprint two.

Sprint two started on 17th of September and ended on 30th of September.

The chapter is divided into five parts, starting with the overall plan for the sprint in Section 9.1. Followed by the sprint backlog, which enlists the tasks that have been chosen for the sprint. Section 9.3. will focus on the work made to the GUI, the logic implemented in the applications and the work done to the database and database access in the applications. The chapter ends with what have been tested and the corresponding results in Section 9.4 and a sprint review in Section 9.5.

9.1 Sprint Plan

The plan for the sprint was to add more of the graphical user interface to the applications. The GUI implemented during this sprint, would later on be connected to logic to complete the functionality wanted.

We also wanted to add some functionality through adding logic to the different elements of the applications. The main focus was the log, the medication plan, the notifications and the distraction during a treatment.

We made a change to the sprint plan during the sprint. The problem was that we had focused too much on hardcoded “dummy” data, and needed the connection towards the database working. This lead to a delay on the log, as this part was most dependent of a working database connection.

9.2 Sprint backlog

This section contains a table with the sprint backlog, which is a smaller part of the product backlog. The goal is to implement the entire sprint backlog during the sprint.

We also decided to continuously write on the report, attend lectures and hold advisor and customer meetings during the sprint. These tasks are not included in the backlog, but the hours spent are included in the status reports.

9.3 Design and Implementation

We continued to add graphical user interface elements to represent the different functionality in the applications. During this sprint the focus was fairly broad. The reason for this was the customer’s priorities of what functionality should be implemented, and also that many of the tasks were dependent of each other, and therefore could not be worked on at the same time.

Both we and the customer were also very curious to how the Karotz would be implemented with the applications. We chose to have one developer focus on this task during this sprint.

# ID	Task	Story points	Estimated hours	Responsible
1.1	Make some kind of distraction through the Karotz	8	40	Yngve
1.2	Make method for saving a treatment through Karotz	4	20	Yngve
1.3	Make method for starting treatment through Karotz	2	10	Yngve
1.4	Make the distraction logic class	10	50	Eirik
2.1	Calendar view for log	10	50	Esben
2.2	Backend solution for saving the log	10	50	
3.1	Make the reminder for Android platform	6	30	Aleksander
3.2	Make the reminder for Karotz	6	30	Yngve
3.3	User interface for changing reminder preferences	6	30	Eirik
3.4	Secure that the reminder is giving independently of internet connection and sound level on the phone	2	10	
SUM		64	320	

Table 9.1: Backlog for sprint 2

9.3.1 User Interface Layer

We implemented the log as a calendar view. This was based on open source code, which we modified, to ensure it would have the properties we wanted. At that time, there was still some work that needed to be done to the log. However, it was very hard to do these changes until a backend solution was working, so the rest of this task was being halted until we had the backend system up and running.

During the sprint we added a basic animation to the distraction, simply counting from zero to ten, to assure that the animation-logic was in place, and the counting because the child should take ten breaths from the inhaler during the treatment. After the treatment is finished the user will be rewarded with stars.

To the settings menu we added very simple functionality for making a treatment plan. By using simple menus a user would be able to choose which medicines should be included and what dosage of each medicine. The user should also be able to set the time for reminders. The GUI for this was implemented during the sprint, but the logic and database connections were yet to be implemented.

We added functionality for dressing up the avatar with different costumes. At that time the costumes were not changeable in the GUI, since the shop was yet to be implemented.

To the treatment and distraction part of the applications, we implemented logic to make the treatment understandable for children. We chose to solve this task by using logic to change the GUI.

During the sprint we implemented notifications to remind the user to take his/her medicine. The logic behind this is done by a "Notification Manager" which fires a method for putting a notification on the status bar of the phone.

We found out halfway during this sprint that it was necessary to change the notifications into alarms. Alarms are able to run independently of any other methods, in difference to notifications which are fired by a method internally. To ensure that the user would get the correct reminders it was necessary to change this.

9.3.2 Data Persistence Layer

The development team found a severe problem with the current database server. The problem is that we cannot access the database unless a client (android device) is connected to NTNU's network. The

customer unsuccessfully worked on trying to find another server we could use, therefore we had to change the architecture, and make a separate application on one of our “folk.ntnu.no”-domains, and access the database from this webservice.

We were in a stage where we needed some test-data to see any progress, so we continued to load the database with somewhat relevant testing data.

9.3.3 Database Access Layer

During the sprint, it was discovered that the Karotz cannot directly access the database, and that the NTNU MySQL server cannot be accessed from outside the school network. These two problems lead to the creation of an additional layer to access the database.

The *Database Access Layer* is a set of PHP web pages currently hosted at <http://folk.ntnu.no/yngvesva/blopp> designed to take input in the form of GET and POST parameters, access the database and either get information from or modify it, and return data in JSON format. This layer is further described in section 6.4.2.

The additional work created by the tasks related to the database access layer meant that there was substantially more work to be done in the sprint than was planned before it started.

The modules created this sprint were:

- *add_child.php*: Takes a name, personal number (SSN) and a list of states (integer IDs) that the child can have. Creates an entry in the CHILDREN table for the child, an avatar entry and a medical plan entry. Also creates an entry in CHILD_HEALTH_STATES for all the states the child can have. Returns the generated `avatar_id` and `medical_plan_id`.
- *get_available_child_states.php*: Takes a child ID and returns a list of the labels (colors, names) and IDs of the states the child can have.
- *get_child.php*: Takes an ID of a child and returns all the columns for the given ID in the CHILDREN table.
- *get_child_state.php*: Accepts a child ID and returns the ID and label of the current state of the child.
- *get_doses_for_current_state.php*: Takes a child ID and returns a list of planned doses of medicines for that day. The fields of each entry are: `id`, `medical_plan_id`, `health_state_id`, `time`, `medicine_id`, `medicine_karotz_color` and `medicine_name`.
- *register_medicine_taken.php*: Register a dose of medicine taken. Accepts a post object with the fields `child_id`, `medicine_id`, `time`, `day_date`, `health_state_id` and `medical_plan_dose_id`. If there is an entry for that dose id that day, the method does nothing and simply returns `unique = false`. Otherwise, it calculates a reward, and updates DAY_MEDICINE_DOSES with the entry. Returns the reward for that dose.
- *set_child_state.php*: Takes a child ID and a state ID and sets the current state of the given child to that one.

9.4 Testing and Results

9.4.1 Testing

The testing done during this sprint was mainly concerned with the backend logic of the system, since the basic graphical layout functionality was tested during the previous sprint. We did testing on the database-connection and the sql-queries, as well as the repositories we created for the client side of the system.

Item	Description
ID	UNIT2.1
Description	Test of the distraction sequence
Date	30.09.2012
Responsible	Eirik
Subject	Distraction and DistractionActivity classes in <code>no.blopp.app.med.activities</code> package
Precondition	Working version of CAPP, runnable on the android virtual device
Steps	<ol style="list-style-type: none"> 1. Press "Start Treatment" on the MainMenu. 2. Press the next-button on the screen, after seeing that the avatar is updated, and it shows the right medicine. 3. Watch the animation, press the next-button. 4. Press the shop button or the main menu button to navigate away from the finish-screen.
Results	Application updated the avatar correctly. Pressing the next-button while the animation was still running caused the application to crash.

Table 9.2: Unit test 2.1, CAPP distraction sequence

Item	Description
ID	UNIT2.2
Description	Test of database connection
Date	26.09.2012
Responsible	Esben
Subject	DBConnection
Precondition	—
Steps	Unit test of <code>DBConnection.java</code> . Run the application as Java Application
Results	<ol style="list-style-type: none"> 1. If connected to NTNU's network via VPN (virtual private network): Connection succeeded. 2. If not connected to NTNU's network: Connection failed

Table 9.3: Unit test 2.2, database connection

Item	Description
ID	UNIT2.3
Description	Testing of SQL-queries
Date	30.09.2012
Responsible	Esben
Subject	LogModelRepository
Precondition	Connected to NTNU's network
Steps	Unit test of <code>LogModelRepository.java</code> . Run the application as Java Application
Results	Success. The fields asked for were the same as stored in the database.

Table 9.4: Unit test 2.3, SQL queries

9.4.2 Results

We see that there is a big problem with using NTNU's MySQL server since it cannot be accessed from points external to the NTNU subnet. This problem is currently being discussed by the customer, and we might, get access to a server that is not dependent on the network the unit is at. Other than that, no major errors were found.

9.5 Sprint Retrospective

This section contains an evaluation of the Sprint. The evaluation is done mainly by the developer team, but feedback from the customers were added to the retrospect.

What went well?

Our programming event held thursday 20th of September was a great success. The group works much better when we're working physically together. We feel that we have a good code base, and can start building the applications more efficiently during the upcoming sprints.

What shall we start doing?

We identified four elements that should be added to our sprints:

1. Be more accurate towards deadlines with the customer.
2. Make better conventions for naming of fields in the source code.
3. Make better sprint plans to reflect the need of the backend system.
4. We need a predetermined day to write the reports for each sprint.

What could have gone better?

We must learn to send the meeting reports to the customer within 24 hours. We should have communicated better on who was going to send the report after Jørgen drowned his computer after a customer meeting.

What should we stop doing?

Naming fields (buttons, textviews etc. from the Android framework) inconsistently with the conventions stated in section D

9.5.1 Sprint Burndown Chart

Figure 9.1 and Table 9.5 show the burndown chart for the second sprint. At first glimpse the sprint burndown chart doesn't look too good. We started off at a fairly good pace. At Thursday 20th of September we held a fellow programming session, which was very successful. This resulted in a small dent in the chart, in the positive direction. During the second week of the sprint, we had some difficulties. First off, one of us had to step down on working hours, due to illness in his family. Second, the other team members had assignments to finish in other courses. This stole much of the time which was supposed to be used on programming.

Also, during the sprint the developers discovered problems technical problems they could not foresee in advance. The database is hosted on a NTNU-server, resulting in the need of VPN-connection to work outside of the university's internet network. We, in agreement with the customer, decided to make a web service to handle the traffic to the database. This resulted in an increase in story points for the task.

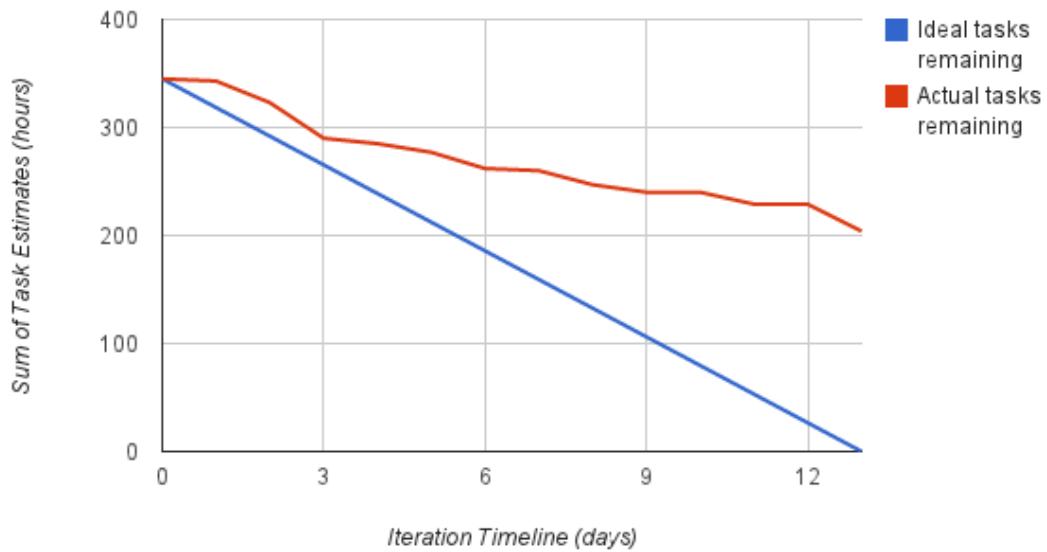


Figure 9.1: Sprint 2 burndown chart

Our goal for each two-week sprint is to finish 175 estimated work hours, which with a base multiplier of 5 correlates to $175/5 = 35$ story points. The backlog contained 64 story points in total, so we had no ambition of completing all the tasks. However, at the end of the sprint there were 243 estimated work hours left, which corresponds to $243/5 = 48.6$ story points. In other words, we only completed $64 - 48.6 = 15.4$ story points in the second sprint. This is less than half of the amount we wanted to complete. We were not satisfied with the progress, and understand we need to step up the amount of work done in the upcoming sprints, in order to finish the prototype.

# ID	Task	Story points	Estimated	Actual	Estimated Left	Responsible
1.1	Make some kind of distraction through Karotz	8	40	18	10	Yngve
1.2	Make method for saving a treatment through Karotz	4	20	—	20	—
1.3	Make method for starting treatment through Karotz	2	10	9	4	Yngve
1.4	Make the distraction logic class	10	50	41	0	Eirik
2.1	Calendar view for log	10	50	20	15	Esben
2.2	Backend solution for saving the log	10	50	4	45	Esben
3.1	Create GUI for settings	1	5	1	0	Jørgen
3.2	Create navigation for settings	1	5	3	0	Jørgen
4.1	Make the reminder for Android Platform form	6	30	14	20	Aleksander
4.2	Make the reminder for Karotz	6	30	7	24	Yngve
4.3	User interface for changing reminder preferences	6	30	0	30	—
4.4	Secure that the reminder is giving independently of internet connection and sound level on the phone	2	10	1.5	10	Aleksander
5.1	Make database access layer	3	15	14	20	Yngve
5.2	Make repositories	10	50	4	45	Esben
SUM		64	320	136.5	243	

Table 9.5: Sprint burndown chart, Sprint 2

Chapter 10

Sprint 3

The following section presents an overview of how we planned, worked and completed sprint.

Sprint 3 started on 1st of October and ended on 14th of October, giving it a duration of 14 days.

The chapter is divided into five parts, starting with the overall plan for the sprint in Section 10.1. Followed by the sprint backlog, which enlists the tasks that have been chosen for the sprint. Section 10.3. will focus on the work made to the GUI, the logic implemented in the application and the work done to the database and database access in the applications. The chapter ends with what have been tested and the corresponding results in Section 10.4 and a sprint review in Section 10.5.

10.1 Sprint Plan

The plan for the sprint was to work on the distraction, the log, the reminders and the database connection. We also planned to do some work on the report, due to an expected preliminary delivery.

Regarding the distraction, the plan was to make some kind of distraction through the Karotz.

Regarding the log, the plan was to work on a better solution of the calendar view and make the backend connections for saving the log in the database.

Regarding the reminders the plan was to finish the work on implementing the reminders and making the reminder work through the Karotz.

Regarding the database the plan was to make different SQL-query methods for medications, reminders, instructions and similiar.

10.2 Sprint backlog

This section contains a table with the sprint backlog, which is a smaller part of the product backlog. The goal is to implement the entire sprint backlog during the sprint.

10.3 Design and Implementation

We continued to add graphical user interface elements to represent the different functionality in the applications. During the sprint the focus has been on the log, the reminders, the Karotz and the database. The reason for this was the priorities given by the customer.

10.3.1 User Interface Layer

During the sprint we made GUI-changes to the log and the instructions. The instructions will now show a list of the different medicines. When a medicine is selected, a new screen with a picture and instructions for correct use is shown. This is downloaded from an external server, which makes it easier to change the pictures without updating the applications.

# ID	Task	Story points	Estimated hours	Responsible
1.1	Make some kind of distraction through the Karotz	2	10	Yngve
1.2	Make method for saving a treatment through Karotz	4	20	Yngve
1.3	Make method for starting treatment through Karotz	2	10	Yngve
2.1	Calendar view for log	3	15	Esben
2.2	Backend solution for saving elements to the log	9	45	Esben
3.1	Make the reminder for Android devices	4	20	Eirik
3.2	Make the reminder for Karotz	5	25	Yngve
3.3	User interface for changing reminder preferences	6	30	Eirik and Aleksander
3.4	Secure that the reminder is giving independently of internet connection and sound level on the phone	2	10	Eirik
4.1	Make SQL-queries for LogModel	1	5	Esben
4.2	Make SQL-queries for essential database fields for CAPP, including avatar, shop, rewards and so on	3	15	
4.3	Make SQL-queries for Medications	2	10	Esben and Jørgen
4.4	Make SQL-queries for Notifications	2	10	Jørgen
4.5	Make SQL-queries for Instructions	1	5	Jørgen
4.6	Make the SQL-queries updaters for Karotz	3	15	Yngve
4.7	Make a generic JSON-deserializer	1	5	Yngve and Esben
4.8	Make JSON-deserializer for logmodel	1	5	Esben
4.9	Make JSON-deserializer for instructions	1	5	Esben
SUM		52	260	

Table 10.1: Backlog for sprint 3

During the sprint the reminders were implemented. When the reminder is set off by the logic layer, the screen will show a message with the text “It’s time to take your medicine”.

During the sprint we, in cooperation with the customer started to work on making better screen elements like buttons, images and similar in order to replace all text. Since CAPP will be used by young children, it will be more understandable for children if we use pictures, rather than text. This process was started so late in the sprint that the results are to be expected during the next sprint.

10.3.2 Application Logic Layer

The reminders have been added, as mentioned previously. At this point, the alarm will be set off by pressing a button inside the application. An alarm manager calculates an offset of ten seconds and then fires an alarm. When the user clicks on the alarm sign, he/she is taken to the treatment screen. The functionality for choosing custom alarm times is yet to be implemented.

10.3.3 Data Persistence Layer

More web access modules have been added, and some of the previous modules have been altered. The new pages include:

- *dose_is_taken.php*: Check if a dose of a planned medicine has been taken that day.
- *get_instructions.php*: Get instructions (image, effect description and usage description) for a given medicine by ID.
- *get_log_for_child.php*: Returns all registered entries for the day for a given child (id) during the given month during the given year.

In addition to the new modules, *get_doses_for_current_state.php* has been altered to only return doses for the day that has not been taken already.

10.3.4 Karotz

This subsection has not been present in the other sprint reports. Because the Karotz is a separate part of the system from the Android applications and the database, we felt it was important to add it as a separate section.

The Karotz will now play reminders at a specific time set by the user in the database. The reminder starts a distraction process, which makes the Karotz able to act as a distraction during treatment. It will play sound in order to tell the user which medicine to take, how to take them and then count down from 10 to 0 to work as a distraction. There are some problems with the calibration of sound messages and the countdown, so the Karotz may skip messages, making it sound like it skips numbers when counting.

The applications communicate with the Karotz through the database by saving and retrieving data asynchronously and without direct messages between them.

10.4 Testing and Results

10.4.1 Testing

During this sprint we tested the reminder and the log part of the system. We wanted to make sure that the reminder was given regardless of phone state, such as when the phone is asleep, or if the phone is used in a call.

We had had some problems coloring the correct days, and making the log the way we wanted, so we felt we had to make sure it was done correctly, and working in the way we wanted. We therefore did testing focusing on the log.

Item	Description
ID	UNIT3.1
Description	Test that the alarm is given independently of phone state
Date	14.10.2012
Responsible	Eirik
Subject	AlarmReciever and MainMenu (which sets the alarm)
Precondition	Working version of CAPP, runnable on the android virtual device
Steps	<ol style="list-style-type: none"> 1. Press the “notify us” button to set the alarm. 2. Wait until the alarm activates. 3. Observe if the alarm activates if you are on a call. 4. Observe if the alarm activates if you’re in another application. 5. Observe if the alarm activates if the phone have been switched off.
Results	The alarm worked fine, unless if the phone is switched off, which releases all alarms. We found a solution to this by adding an <code>onBootAction</code> .

Table 10.2: Unit test 3.1: Alarm when turned off

Testing the Karotz is essential to the project, so in this sprint the two most important modules of the Karotz app were tested: notification and medication. There was also an additional test to check whether several medications happening at the same time would be handled correctly.

There was a problem with the treatment functionality of the Karotz. The countdown did not work correctly. Often jumping from ten to seven, from seven to three and from three to finished. This is because the Karotz’ built-in media player is slow and takes a little time to start and stop. The solution to this problem is to include the entire countdown in one audio file, compared to previously where each number was stored in a separate file.

In addition to these tests, we are testing the application continuously.

Item	Description
ID	UNIT3.2
Description	Test that the correct days is colored in the log
Date	09.10.2012
Responsible	Esben
Subject	LogModelParser, CalendarView
Precondition	Correct resultset from <code>get_log_for_child.php</code>
Steps	<ol style="list-style-type: none"> 1. Press “log”. 2. Compare the colored days with the resultset from <code>get_log_for_child.php</code>.
Results	Correct days was ultimately colored after a huge effort was made in finding a couple of bugs.

Table 10.3: Unit test 3.2: Calendar colors

Item	Description
ID	UNIT3.3
Description	Test that a notification is given at the correct time through the Karotz
Date	09.10.2012
Responsible	Yngve
Subject	Karotz app's Repository and Notification modules
Precondition	Karotz app starts, planned dose URLs and parameters are correct
Steps	<ol style="list-style-type: none"> 1. Adjust a time for the notification in the table MEDICAL_PLAN_DOSES in the database. 2. Wait until the alarm activates. 3. Observe if the notification is activated at the correct time.
Results	The notification went off at the given time.

Table 10.4: Unit test 3.3: Karotz Notification

Item	Description
ID	UNIT3.4
Description	Test that a distraction is started and works correctly when the button on the Karotz is pressed after a notification is given.
Date	09.10.2012
Responsible	Yngve
Subject	Karotz app's Repository, Notification and Medication modules
Precondition	Notification is given correctly through the karotz
Steps	<ol style="list-style-type: none"> 1. Wait until the notification activates. 2. Press the button to terminate the notification and start medication. 3. Follow the vocal instructions given by the Karotz. 4. Observe that the medication procedure is starting. 5. Observe that the countdown works correctly. 6. Observe that a reward (credits) is given at the end of the medication process. 7. Observe that the reward is saved in the database.
Results	The medication process started, but the countdown started but skipped some numbers. This can however be solved by making the whole countdown in a single sound file. The Karotz expressed audibly that a reward was given, and that was saved to the database.

Table 10.5: Unit test 3.4: Karotz distraction

Item	Description
ID	UNIT3.5
Description	Test that several doses in the same notification will make a medication for each dose, after each other.
Date	11.10.2012
Responsible	Yngve
Subject	Karotz app's Repository, Notification and Medication modules
Precondition	A notification that consists of three doses is given.
Steps	<ol style="list-style-type: none"> 1. Wait for a notification with three doses at the same time. 2. Observe that the first medication process is completed correctly. 3. Observe that when the first medication process is done, another is started automatically. 4. Observe that the second medication process is completed correctly. 5. Observe that when the second medication process is done, another is started automatically. 6. Observe that the third medication process is completed correctly. 7. Observe that when the third medication process is done, no other process is started and the Karotz returns to idle state.
Results	The medication series worked as expected.

Table 10.6: Unit test 3.5: Several doses in a medication

10.4.2 Results

We found out that the alarm was given regardless of the phone state, unless the phone was turned off, since we have no way to turn the phone back on. The alarms will be put back on whenever the phone is turned on, so the only scenario the alarm will not be given is if the phone is turned off at the time for a reminder. Regarding the coloring of the days in the log, the color depends on the medication plan followed that day. We had some trouble discovering what bugs made the wrong days be colored, but after extensive debugging, we found out that one of the SQL-queries was not correct. Upon fixing the query to gather the right ID from the database the log worked as intended.

10.5 Sprint Retrospective

This section contains an evaluation of the Sprint. The evaluation is done mainly by us, but feedback from the customers is added to the retrospect.

What went well?

Karotz is ready for user testing. Eirik was responsible for Aleksanders' tasks when he was gone, and secured progress. Communication with the customer has improved.

What shall we start doing?

We identified one element that we should add to our sprints:

1. Increase amount of programming sessions.

What could have gone better?

We forgot to have daily standups, this shouldn't happen. Better testing of the Karotz in advance to the customer meeting. Be better prepared in advance of customer meeting.

What should we stop doing?

We felt that nothing was done completely wrong during this sprint.

10.5.1 Sprint Burndown Chart

Figure 10.1 and table 10.7 show the sprint burndown chart for sprint 3. The burndown chart looks good at first glance, compared to sprint 1 and 2. During the last two sprints we have learned more about estimation and development, this has resulted in a steadier pace. The burndown charts started out very good, this was due to some functionality being dropped very early in the sprint. This decision was made by the customer.

Also we discovered fairly early that some tasks were easier than expected. Closer to the end of week one the burndown chart flattened out a bit. The reason behind this is that we needed to work intensly on assignments in other courses. At the start of week two, a programming session was held, resulting in a steep decline in the burndown chart. After this the pace was fairly consistent.

We had a goal of finishing 49 story points. There are 60 hours left, which with the multiplier we used, 5, means there are $60/5 = 12$ story points left. Ergo, we finished $49 - 12 = 37$ story points, which is closer to the ideal projected effort than the previous sprints, but not satisfactory.

# ID	Task	Story points	Estimated	Actual	Estimated Left	Responsible
1.1	Make some kind of distraction through Karotz	2	10	3	0	Yngve
1.2	Make method for saving a treatment through Karotz	4	20	8	0	Yngve
1.3	Make method for starting treatment through Karotz	2	10	5	0	Yngve
2.1	Calendar view for log	3	15	15	3	Esben
2.2	Backend solution for saving the log	9	45	5	25	Esben
3.1	Make the reminder for Android Platform	4	20	20	2	Eirik
3.2	Make the reminder for Karotz	5	25	25	0	Yngve
3.3	User interface for changing reminder preferences	6	30	4	28	Aleksander and Eirik
3.4	Secure that the reminder is giving independently of internet connection and sound level on the phone	2	10	10	2	Eirik
4.1	Make SQL-queries for LogModel	1	5	4	0	Esben
4.2	Make SQL-queries for essential database fields for children application, including avatar, shop, rewards and so on	3	15	Task dropped	–	–
4.3	Make SQL-queries for Medications	2	10	7	0	Esben and Jørgen
4.4	Make SQL-queries for Notifications	2	10	0	0	Jørgen
4.5	Make SQL-queries for Instructions	1	5	3	0	Jørgen
4.6	Make the SQL-queries updater for Karotz	3	15	3	0	Yngve
4.7	Make a generic JSON-deserializer	1	5	10	0	Yngve and Esben
4.8	Make JSON-deserializer for logmodel	1	5	5	0	Esben
4.9	Make JSON-deserializer for instructions	1	5	5	0	Esben
SUM		49	245	114	60	

Table 10.7: Sprint Retrospective, Sprint 3

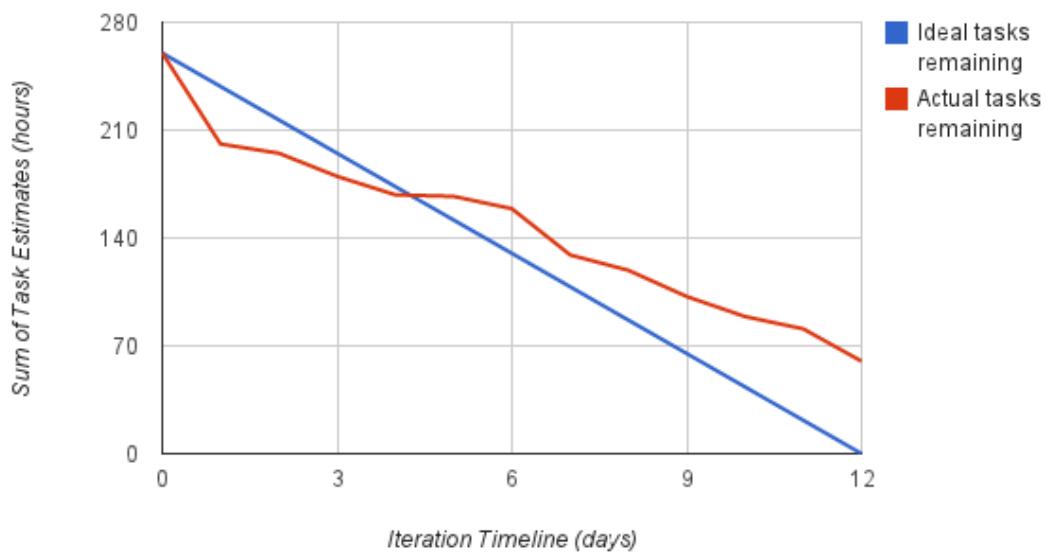


Figure 10.1: Sprint 3 burndown chart

Chapter 11

Sprint 4

The following section presents an overview of how we planned, worked and completed sprint 4.

Sprint 4 started on 15th of October and ended on 28th of October.

The chapter is divided into five parts, starting with the overall plan for the sprint in Section 11.1. Followed by the sprint backlog, which enlists the tasks that have been chosen for the sprint. Section 11.3. will focus on the work made to the GUI, the logic implemented in the application and the work done to the database and database access in the application. The chapter ends with what have been tested and the corresponding results in Section 11.4 and a sprint review in Section 11.5.

11.1 Sprint Plan

The plan for the sprint is to work on a broad spectrum of functionality for the application. With four weeks left to work on implementation, and the last week needed for bug fixing and wrapping up the code, the time is starting to run out. For much of the user stories it was either make it this sprint or break it, since we didn't want to deliver half-finished code.

The general look and feel of both applications was fairly generic and not good-looking. At the start of the sprint, the GUI consisted of mostly standard elements and some pictures we painted ourselves, leaving poor software. Therefore the GUI in both applications were in for a huge upgrade.

11.1.1 CAPP

During the first customer meeting, we made a decision to remove functional requirement CF4: “The application should have an avatar for each child, that can be chosen by the child, and customized through the shop.”. We felt that the shop idea was simply too big of a task to implement during the last four weeks in addition to all the other work that remained. The gamification part of the project was now reduced to a showing of amount of rewards a child had collected, and we decided that this reward would be an amount of stars. However, we kept the database-tables such that it would be easier to extend the application with this functionality.

Regarding the reward system of CAPP we decided to make a more visual overview of how many stars are collected, since children does not necessarily know how to read. We also had to implement a connection between the database and this view.

The distraction was up for refactoring. The solution we had at the start of the sprint was a very simple, not very distracting, countdown. The customer wanted to change this, and told us they would create pictures in order to make a better distraction.

Instructions for correct use should also be implemented during this sprint. Since the instructions are for children, they are implemented as a picture gallery downloaded from an external server.

11.1.2 GAPP

To the log of GAPP we plan to implement functionality for registering a treatment which was taken earlier, without the app present. We also planned to add a pollen forecast to the calendar, as it may warn users about possible pollen spread, since this may effect the child's health state.

To the settings functionality, we planned to improve the general layout of the menu for adding a medication plan. Also this part of the application needs to be connected to the database, which will be done during this sprint.

To the reminder functionality the plan was to make finishing touches in order to make it work.

11.2 Sprint backlog

This section contains table 11.1 with the sprint backlog, which is a smaller part of the product backlog. The goal is to implement the entire sprint backlog during the sprint.

11.3 Design and Implementation

This section will present the changes done during the sprint.

11.3.1 User Interface Layer

CAPP

Huge changes were made to the main menu of CAPP. The buttons from earlier were removed, and instead all buttons throughout the application is image buttons, meaning it is a picture which works as a button when pressed.

During this sprint we added functionality for viewing instructions in the child application. The instructions are accessible from the main menu. Upon opening the instructions the user is met with a picture series where the user may view pictures that give information regarding how to use the medicines correctly.

The distraction for treatment was given a facelift in form of a huge change-out of the pictures. During the sprint we added new pictures and interactive design for the user to interact with the Karotz. We also added more sound files, to make it sound like the Karotz is talking to the user.

GAPP

During this sprint we added functionality for viewing the pollen forecast for today's date and the following day. Using "Pollenvarslingen", an external API, the application can make calls to an external server and receive an XML-feed in return. This XML-feed gives certain info about the pollen forecast. Regarding this there were some problems. Since there is no pollen in the air in October, all calls to the API returned empty XML-feeds. After discussing this problem with the customer, we found out that we solved this by setting up a dummy XML-feed on our own external server. The XML-feed would always return the exact same document, but it still made it look like a real pollen forecast.

The main menu of the app was given some changes. The buttons from earlier are gone, and instead all buttons throughout the application is image buttons, meaning it is a picture which works as a button when pressed.

During this sprint we added functionality for viewing instructions in the application. The instructions are accessible from the main menu. Upon opening the instructions the user is met with a picture series where the user may view pictures giving information regarding how to use the medicines correctly.

The menu for choosing treatment plan was polished with new images and text, to make it more understandable and intuitive.

# ID	Task	Story points	Estimated hours	Responsible
1.1	View for amount of stars collected	1	4	Yngve
1.2	Connect reward view to the database	1	4	Yngve
2.1	Refine the distraction for the children	3	12	Yngve
3.1	Finalize instructions for children	3	12	Aleks
4.1	Make solution for registering a treatment	3	12	Esben
4.2	Implement pollenfeed to log	10	40	Esben
4.3	Finalize view for showing pollen and medications taken at a given day	2	8	Esben
5.1	Finalize instruction for medications	3	12	Aleks
6.1	Make view for viewing existing medication plans	4	16	Eirik
6.2	Import medication list from database to medication plan settings	1	4	Esben
6.3	Improve layout for making medication plan	2	8	Jørgen
6.4	Save the medication plan to the database	3	12	Jørgen
7.1	User interface for changing reminder preferences	1	4	
7.2	Secure that the reminder is giving alarms independently of internet connection and sound level on phone	1	4	Eirik
8.1	Create web interface page for adding medical plan doses	1	4	Yngve
8.2	Create web interface page for removing medical plan doses	1	4	Yngve
8.3	Create web interface for getting log grouped on days	1	4	Yngve
9.1	Refine Karotz app manus, after feedback from customer	3	12	Yngve
9.2	Record new voice messages for the Karotz	1	4	Yngve
10.1	Refine design for registering treatment	1	4	Esben and Jørgen
10.2	Refine design for instructions	2	8	Esben
10.3	Refine GUI for the main menu	1	4	Aleks and Esben
11.1	Refine alarm and distraction regarding database calls	2	8	Yngve
SUM		51	204	

Table 11.1: Backlog for sprint 4

The calendar will now show what health state the child was in, by coloring a bar on top of the given day. The bottom bar of each day will be colored according to the worst amount of pollen on the given day. Since there is no pollen in the winter season, we have hardcoded the values for the amount of pollen, which makes the log show no differences between days.

11.3.2 Application Logic Layer

In the menu for the treatment plan, the application will now create reminders based on what treatment plan is chosen. Different treatment plans will also affect the amount of stars the children will receive upon finishing a treatment.

The menu for registering a treatment taken earlier, will now save to the database, based on what date and what medicine the user chooses from the menu.

The information about medicines will now show a listed menu of the different medicines. When a medicine is chosen by click, it will load a screen with info from the database.

11.3.3 Data Persistence Layer

The following web access pages were added during the 4th sprint:

- *add_plan_dose.php*: Inserts a new entry in the MEDICAL_PLAN_DOSES table for the given child, health state a dose of the given medicine at the given time. This is the primary module used to alter medical plans. Returns the ID of the added dose.
- *remove_plan_dose.php*: Deletes the entry in the table MEDICAL_PLAN_DOSES which corresponds to a given id. Returns the number of deleted rows.
- *get_log_days_for_child.php*: For the calendar in GAPP, it was advantageous to have a database access method that could return a list of days in a month with the child health state for each day, and a list of doses taken on that day. *Get_log_days_for_child.php* accomplishes this by using the table CHILDREN_LOG_DAYS to find the latest recorded health state before the given month started. Then it iterates through all the days, checking if there are any days in the month where the status changes, and adding all doses, taken from DAY_MEDICINE_DOSES, on that day. The method takes a `child_id`, and two optional parameters `month` and `year`. If the month and year are not set, the values for the current days are used.
- *remove_plan_medicine_at_time.php*: During development of the adult app GAPP, it was discovered that using a combination of `child_id`, `medicine_id` and `time` as the key for removing elements in the MEDICAL_PLAN_DOSES table would be easier than using an ID. Therefore the need for this module arised, and it simply removes all entries in the table that fit the criteria.

Changes were made to the following files for web access:

- *add_child.php*: Since we moved away from the avatar idea, this method no longer creates an entry in the AVATARS table, and does not return an ID for that entry.
- *set_child_state.php*: Now updates the table CHILDREN_LOG_DAYS with the given health state.
- *register_medicine_taken.php*: Adds the calculated reward to the child's total credits in the CHILDREN table. If no time is given, a default time of 00:00:01 is set.

The following changes were made to the database itself:

- Since the Avatar idea was at least put on hold, there is no need to store credits in the avatar for a child. An extra column named `credits` was therefore added to the CHILDREN table.

Item	Description
ID	UNIT4.1
Description	Test that the right instructions are downloaded and shown on screen in the instructions menu
Date	18. 10. 2012
Responsible	Aleks
Subject	Medication instructions and external database
Precondition	The application is running, the database is running
Steps	<ol style="list-style-type: none"> 1. Press Instructions button in the main menu. 2. Choose a medication from the list shown. 3. Observe if the correct picture is shown on screen 4. Observe if the correct text (instructions) is shown on screen.
Results	The first few times, a blank screen was shown. After debugging, all worked as expected. More about this in results

Table 11.2: Unit test 4.1: instructions

- To support several medical plans for a child and to be able to save such plans and change back and forth between them, a column `child_id` was added to the table `MEDICAL_PLANS`. It is therefore possible to indicate the “active” medical plan with the id in the `CHILDREN` table, but many can be stored to a child. Subsequently the useless `label` column was dropped.

11.3.4 Karotz

The sound clips for the Karotz were changed in order to be more understandable for children. We had simply not thought of aspects such as the Karotz should count from 0 to 10, since children does not necessarily know how to count backwards. The Karotz will now communicate with the database in order to save rewards earned by the user. At the end of this sprint, the Karotz had all desired functionality for reminders, distraction, instruction and rewards. Testing on users still remained, and was planned for the beginning of sprint 5.

11.4 Testing and Results

11.4.1 Testing

Table 11.2 shows the unit test performed this sprint.

11.4.2 Results

Regarding the test UNIT4.1, we had some problems with showing the instructions. The page returned a blank screen, regardless of what medicine was chosen. We controlled the servers responds by using a browser, to make it easier to read the return values. The return values where correct each time.

After extensive debugging, we discovered that the query done in the application was written in an illegal way. This way, the query was not parameterized as a query and the server did not respond to the request. After fixing the query, everything worked as expected.

11.5 Sprint Retrospective

This section contains an evaluation of the sprint. The evaluation is done mainly by the us, but feedback from the customers are added to the retrospect.

11.5.1 What went well?

We have now finished a lot of the backend functionality for the system, and the system is working well with the exception of a few minor bugs. The system is nearing a finished state and usability testing may begin.

11.5.2 What shall we start doing?

1. Document internal unit tests better.
2. Document time spent on report writing and meetings better
3. Implement user feedback when the application crashes.
4. Code refactoring
5. Add time spent on refactoring to the estimate for a task

11.5.3 What could have gone better?

We could have foreseen many of the bugs that makes crashes, and implemented a try/catch block in the sourcecode, in order to prevent hard crashes. Examples: crashes due to lack of internet connection, lack of input from the user.

11.5.4 What should we stop doing?

We should stop using GUI-based git software, and learn terminal-based git software, as none of the GUI applications are working flawlessly.

11.5.5 Sprint Burndown Chart

Figure 11.1 and table 11.3 show the burndown chart for the fourth sprint. From the burndown chart, the fourth sprint looks like a success. Ahead of the sprint, the function number for the story points was lowered, since we expected to work faster. This was done independently of our estimation, to not make biased results. The team worked faster than expected on a general base. Some tasks were still overestimated. The reason for this was little knowledge towards the task and how it should be solved. For example the implementation of the pollen feed proved to be much faster than expected, since we hardcoded the values.

The sprint contained 204 estimated work hours, and our goal was to complete 175 of those, which corresponds to $175/4 = 43.75$ story points with our base multiplier of 4. While only 131 work hours were recorded, there were only 16 estimated work hours left when the sprint was over. This corresponds to $16/4 = 4$ story points. Therefore, we completed $51 - 4 = 47$ story points, which is $47 - 43.75 = 3.25$ more than our goal. The sprint was therefore considered a success with a satisfactory result.

# ID	Task	Story points	Estimated	Actual	Estimated Left	Responsible
1.1	View for amount of stars collected	1	4	4	0	Yngve
1.2	Connect reward view to the database	1	4	3	0	Yngve
2.1	Refine the distraction for the children	3	12	10	0	Yngve
3.1	Finalize instructions for children	3	12	10	2	Aleks
4.1	Make solution for registering a treatment	3	12	7	0	Esben
4.2	Implement pollenfeed to log	10	40	6	0	Esben
4.3	Finalize view for showing pollen and medications taken at a given day	2	8	5	0	Esben
5.1	Finalize instruction for medications	3	12	7	2	Aleks
6.1	Make view for viewing existing medication plans	4	16	16	0	Eirik
6.2	Import medication list from database to medication plan settings	1	4	1	0	Esben
6.3	Improve layout for making medication plan	2	8	10	2	Jørgen
6.4	Save the medication plan to the database	3	12	6	0	Jørgen
7.1	User interface for changing reminder preferences	1	4	0	4	-
7.2	Secure that the reminder is giving alarms independently of internet connection and sound level on phone	1	4	8	0	Eirik
8.1	Create web interface for adding medical plan doses	1	4	1	0	Yngve
8.2	Create web interface for removing medical plan doses	1	4	1	0	Yngve
8.3	Create web interface for getting log grouped on days	1	4	1	0	Yngve
9.1	Refine Karotz app manus, after feedback from customer	3	12	13	2	Yngve
9.2	Record new voice messages for the Karotz	1	4	2	0	Yngve
10.1	Refine design for registering treatment	1	4	8	0	Esben and Jørgen
10.2	Refine design for instructions	2	8	6	0	Esben
10.3	Refine GUI for the main menu	1	4	3	0	Aleks and Esben
11.1	Refine alarm and distraction regarding database calls	2	8	3	4	Yngve
SUM		51	204	131	16	

Table 11.3: Sprint Retrospective, Sprint 4

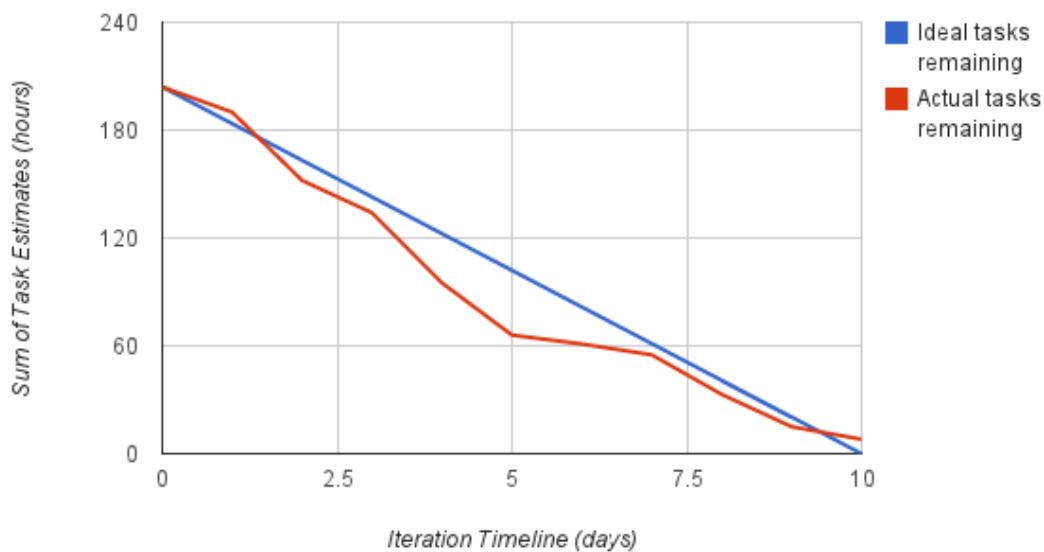


Figure 11.1: Sprint 4 burndown chart

Chapter 12

Sprint 5

The following section presents an overview of how we planned, worked and completed sprint 5.

Sprint 5 started on 29th of October and ended on 11th of November, giving it a duration of 14 days.

The chapter is divided into five parts, starting with the overall plan for the sprint in Section 12.1. Followed by the sprint backlog, which enlists the tasks that have been chosen for the sprint. Section 12.3. will focus on the work made to the GUI, the logic implemented in the application and the work done to the database and database access in the application. The chapter ends with what have been tested and the corresponding results in Section 12.4 and a sprint review in Section 12.5.

12.1 Sprint Plan

The plan for sprint 5 was to finishing the user interface, look for and fix errors and refactor, document and comment code. At the end of this sprint we planned for a code freeze for the sourcecode, meaning no changes were to be made after this sprint. The sprint started with the usability test, and we made the rest of the sprintplan based on the feedback we got from this test.

12.2 Sprint backlog

This section contains a table with the sprint backlog, which is a smaller part of the product backlog. The goal is to implement the entire sprint backlog implemented during the sprint.

12.3 Design and Implementation

Since this was the last sprint, the code had to be refactored and commented. During this sprint we made sure to delete all classes that were not in use, merge classes were possible and logical and mark all deprecated classes. This resulted in a much more understandable and readable code.

After deleting all unused code, we wrote javadoc[21] for all code. Since javadoc is a common use among java developers, this will make the source code easier to read and understand, for people picking up the code in the future.

12.3.1 User Interface Layer

CAPP

CAPP underwent minor changes to the user interface. The main menu had it's icons changed out, to give it a nicer, more colorful and thus more attractive look.

The overview of how many stars the child has earned was given a bigger star at the top, to easily show how many stars the child has earned in total.

# ID	Task	Story points	Estimated hours	Responsible
1.1	Refine distraction for CAPP	2	8	Yngve
1.2	Fix medicine choice before unscheduled medication	2	8	Yngve
1.3	Fix jumping images during child distraction	1	4	Yngve
1.4	Fix correct heading in all pages	1	4	Eirik and Esben
2.1	Bugfix new manuscript on Karotz	1	4	Yngve
2.2	Bugfix Karotz after usability test: double RFID check	1	4	Yngve
3.1	Delete old alarms	2	8	Eirik
4.1	Make better layout for info about medication	1	4	Aleksander
5.1	Write javadoc for all code	3	12	Esben, Eirik
5.2	Remove unused code (imports, classes, etc)	1	4	Aleksander, Eirik and Esben
6.1	Fix Wifi-related crashes in CAPP	0.5	2	Esben
6.2	Fix Wifi-related crashes in GAPP	0.5	2	Esben
7.1	Perform and document unit tests of all we access pages	1	4	Yngve
SUM		19	76	

Table 12.1: Backlog for sprint 5

GAPP

GAPP underwent very small changes in this sprint. To the section where users may view information about medication, we added pages for viewing information about specific medicines, and a link to the instructions images implemented earlier.

Karotz

Based on the usability test, the manuscript for both Karotz and CAPP were updated. The counting action was reworked to include instructions for the child to press the medicine in order to start the medication process. Also, whenever the user was told to hold a nanoz to the rabbit, they were previously told to hold it against the rabbit's stomach. The detector is placed directly beneath the Karotz' nose, so the dialogue was changed to accomodate this. The final manuscript is laid out in detail in appendix C.

12.3.2 Application Logic Layer

The applications still had some bugs at the beginning of the sprint. During this sprint we smashed a lot of them. Mentioning some, we fixed a problem with which medicineID was sent to the database, a problem with deleting alarms was fixed and a problem with the alarm not letting the user turn off the sound was fixed.

12.3.3 Data Persistence Layer

Changes were made to the following files for web access:

- *get_log_days_for_child.php*: This module would return days in the future, with the last recorded health state id. This behavior was changed so that the module doesn't return entries for future days.

12.4 Testing and Results

12.4.1 Testing

During this sprint our main focus was on refactoring, commenting and bugfixing. This meant we did a lot of different tests during this sprint. The sprint started with a usability test, using CAPP and Karotz app to test if children could actually follow the instructions, and to see if our system behaved the way we intended it to. We then did unit tests for all web access modules, to make sure the correct fields were returned from the database. Lastly we did several integration tests to ensure that the changes implemented after the usability test, and the rest of the system, worked as intended

Tables 12.2 through 12.16 describe the unit tests done in sprint 5.

Item	Description
ID	UNIT5.1
Description	Test of the web access module add_child.php
Date	30.10.12
Responsible	Yngve
Subject	add_child.php and database
Precondition	The database is up with the tables MEDICAL_PLANS, CHILDREN and HEALTH_STATES.
Steps	<ol style="list-style-type: none"> 1. Initiate a REST client (POSTMAN) with the URL http://folk.ntnu.no/yngvesva/blopp/add_child.php. 2. Add the POST data <ul style="list-style-type: none"> • name: testname • persnum: 10101012345 • states[]: 1 • states[]: 2 3. Observe the returned JSON data
Results	<p>The JSON data returned was:</p> <pre> 1 { 2 post: { 3 name: "testnavn", 4 persnum: "10101012345", 5 states: [6 "1", 7 "2" 8] 9 }, 10 q: "INSERT INTO 'CHILDREN' ('id', 'name', 'pers_num', 'medical_plan_id', 'credits') VALUES ('', 'testnavn', '10101012345', '0', '0')", 11 medical_plan_q: "INSERT INTO 'MEDICAL_PLANS' ('id', 'label') VALUES ('', 'testnavn')", 12 medical_plan_id: 0, 13 child_id: 12, 14 state_queries: [15 "INSERT INTO 'CHILD_HEALTH_STATES' ('child_id', 'health_state_id', 'applies_now', 'default') VALUES ('12', '1', '1', '1')", 16 "INSERT INTO 'CHILD_HEALTH_STATES' ('child_id', 'health_state_id', 'applies_now', 'default') VALUES ('12', '2', '0', '0')" 17], 18 all_state_ids: [19 "1", 20 "2", 21 "3" 22] 23 }</pre>

Listing 12.1: JSON result from add_child.php

We can see from the JSON result that a child was added successfully, and it got the ID 10.

Table 12.2: Unit test 5.1, add_child.php

Item	Description
ID	UNIT5.2
Description	Test of the web access module <code>add_plan_dose.php</code>
Date	01.11.12
Responsible	Yngve
Subject	<code>add_plan_dose.php</code> and the database
Precondition	The database is up with the tables MEDICAL_PLAN_DOSES and CHILDREN.
Steps	<ol style="list-style-type: none"> 1. Initiate a REST client (POSTMAN) with the url <code>http://folk.ntnu.no/yngvesva/blopp/add_plan_dose.php</code>. 2. Add the post data: <ul style="list-style-type: none"> • <code>child_id</code>: 6 • <code>health_state_id</code>: 1 • <code>medicine_id</code>: 1 • <code>time</code>: 12:34:56 3. Observe the returned JSON data.
Results	<p>The returned JSON data was:</p> <pre> 1 { 2 "sqlsuccess": true, 3 "q": "INSERT INTO 'MEDICAL_PLAN_DOSES' ('id', 'medical_plan_id', 'health_state_id', 'time', 'medicine_id') SELECT '', C. 4 medical_plan_id, '1', '12:34:56', '1' FROM 'CHILDREN' AS C 5 WHERE C.id='6'", 6 "child_id": "6", 7 "health_state_id": "1", 8 "medicine_id": "1", 9 "time": "12:34:56", 10 "id": 40 11 }</pre> <p style="text-align: center;">Listing 12.2: JSON result from <code>add_plan_dose.php</code></p> <p>We see that a planned dose was added successfully by validating the queries and the parameter <code>sqlsuccess</code>, and checking the id 40.</p>

Table 12.3: Unit test 5.2, `add_plan_dose.php`

Item	Description
ID	UNIT5.3,
Description	Test of the web access module <code>dose_is_taken.php</code> .
Date	01.11.12
Responsible	Yngve
Subject	The database and <code>dose_is_taken.php</code> .
Precondition	The database is up with the table <code>DAY_MEDICINE_DOSES</code> .
Steps	<ol style="list-style-type: none"> 1. Initiate a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/dose_is_taken.php?dose_id=40</code> 2. Observe the returned JSON data.
Results	The returned JSON data was:
<pre> 1 { 2 sqlsuccess: true, 3 dose_id: "40", 4 day_date: "2012-11-01", 5 query: "SELECT 'id' FROM 'DAY_MEDICINE_DOSES' WHERE '" 6 "medical_plan_dose_id='40' AND 'day_date='2012-11-01' LIMIT 0,1 7 ", 8 result: false 9 } </pre>	Listing 12.3: JSON result from <code>dose_is_taken.php</code>
	<p>We see from the result that the dose with id 40 (added in UNIT5.2, table 12.4.1) has not been taken on the actual date, and by the query and the <code>sqlsuccess</code> parameter, we see that the GET module works.</p>

Table 12.4: Unit test 5.3, `dose_is_taken.php`

Item	Description
ID	UNIT5.4
Description	Test of the web access module <code>get_available_child_states.php</code> .
Date	01.11.12
Responsible	Yngve
Subject	The database and <code>get_available_child_states.php</code> .
Precondition	A working database with the tables <code>HEALTH_STATES</code> and <code>CHILD_HEALTH_STATES</code> .
Steps	<ol style="list-style-type: none"> 1. Initiate a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/get_available_child_states.php?child_id=6</code>. 2. Observe the returned JSON data.
Results	<p>The returned JSON data was:</p> <pre> 1 { 2 sqlsuccess: true , 3 child_id: "6", 4 query: "SELECT id, label FROM 'HEALTH_STATES' HS WHERE HS.id IN (5 SELECT health_state_id FROM 'CHILD_HEALTH_STATES' CHS WHERE CHS. 6 child_id=6) LIMIT 0,10", 7 rows: [8 { 9 id: "1", 10 label: "GREEN" 11 }, 12 { 13 id: "2", 14 label: "YELLOW" 15 }, 16 { 17 id: "3", 18 label: "RED" 19 } } </pre> <p>Listing 12.4: JSON result from <code>get_available_child_states.php</code></p> <p>We see from the returned data that the child with $ID = 6$ can have all states (GREEN, YELLOW and RED), and that the module works by checking <code>sqlsuccess</code> and <code>query</code>.</p>

Table 12.5: Unit test 5.4, `get_available_child_states.php`

Item	Description
ID	UNIT5.5
Description	Test of the web access module <code>get_child.php</code> .
Date	01.11.12
Responsible	Yngve
Subject	The database and <code>get_child.php</code> .
Precondition	A working database with the table <code>CHILDREN</code> .
Steps	<ol style="list-style-type: none"> 1. Initiate a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/get_child.php?child_id=6</code>. 2. Observe the returned JSON data.
Results	<p>The returned JSON data was:</p> <pre> 1 { 2 sqlsuccess: true, 3 child_id: "6", 4 query: "SELECT * FROM 'CHILDREN' WHERE id=6 LIMIT 0,1", 5 information: { 6 id: "6", 7 name: "Hermann", 8 pers_num: "1010354322", 9 medical_plan_id: "5", 10 avatar_id: "7", 11 credits: "45", 12 location_latitude: "0", 13 location_longitude: "0" 14 } 15 }</pre> <p style="text-align: center;">Listing 12.5: JSON result from <code>get_child.php</code></p> <p>We see from the returned data that the child with $ID = 6$ is named Hermann, some other information, and that the query works.</p>

Table 12.6: Unit test 5.5, `get_child.php`

Item	Description
ID	UNIT5.6
Description	Test of the web access module <code>get_child_state.php</code>
Date	01.11.12
Responsible	Yngve
Subject	The database and <code>get_child_state.php</code> .
Precondition	A working database with the tables <code>HEALTH_STATES</code> and <code>CHILD_HEALTH_STATES</code>
Steps	<ol style="list-style-type: none"> 1. Initiate a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/get_child_state.php?child_id=6</code>. 2. Observe the returned JSON data.
Results	<p>The returned JSON data was:</p> <pre> 1 { 2 sqlsuccess: true, 3 child_id: "6", 4 query: "SELECT id, label FROM 'HEALTH_STATES' HS WHERE HS.id IN (5 SELECT health_state_id FROM 'CHILD_HEALTH_STATES' CHS WHERE CHS. 6 child_id=6 AND CHS.applies_now=1) LIMIT 0,1", 7 state: { 8 id: "1", 9 label: "GREEN" } </pre> <p style="text-align: center;">Listing 12.6: Returned JSON data from <code>get_child_state.php</code></p> <p>We see from the returned data that the child with $ID = 6$ is in the green state, and that the module works.</p>

Table 12.7: Unit test 5.6, `get_child_state.php`

Item	Description
ID	UNIT5.7
Description	Test of the web access module <code>get_doses_for_current_state.php</code> .
Date	01.11.12
Responsible	Yngve
Subject	The database and <code>get_doses_for_current_state.php</code>
Precondition	A working database with the tables MEDICAL_PLAN_DOSES, DAY_MEDICINE_DOSES and CHILD_HEALTH_STATES.
Steps	<ol style="list-style-type: none"> 1. Initiate a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/get_doses_for_current_state.php?child_id=6</code>. 2. Observe the returned JSON data.
Results	The returned JSON data was:
	<pre> 1 { 2 sqlsuccess: true , 3 child_id: "6", 4 query: "SELECT Mpd.id, Mpd.medical_plan_id, Mpd.health_state_id, Mpd 5 .time, Mpd.medicine_id, M.color AS 'medicine_color', M.name AS ' 6 medicine_name' FROM 'MEDICAL_PLAN_DOSES' AS Mpd, 'MEDICINES' AS 7 M WHERE Mpd.medical_plan_id IN (SELECT C.medical_plan_id FROM ' 8 CHILDREN' AS C WHERE C.id=6) AND Mpd.id NOT IN (SELECT DMD. 9 medical_plan_dose_id FROM 'DAY_MEDICINE_DOSES' AS DMD WHERE DMD. 10 medical_plan_dose_id=Mpd.id AND DMD.day_date='2012-11-04') AND 11 Mpd.medicine_id = M.id AND Mpd.health_state_id IN (SELECT HS. 12 health_state_id FROM 'CHILD_HEALTH_STATES' AS HS WHERE HS. 13 child_id=6 AND HS.applies_now='1') LIMIT 0,100", 14 rows: [15 { 16 id: "41", 17 medical_plan_id: "5", 18 health_state_id: "1", 19 time: "01:09:00", 20 medicine_id: "1", 21 medicine_color: "BLUE", 22 medicine_name: "Flutide" 23 } 24] 25 }</pre>
	<p>Listing 12.7: Returned JSON data for <code>get_doses_for_current_state.php</code></p> <p>From the returned data we see that the child with $ID = 6$ has yet to take Flutide today, which should be taken at 01:09:00. We can also validate the SQL query, and see that $sqlsuccess = true$, which means that the module works.</p>

Table 12.8: Unit test 5.7, `get_doses_for_current_state.php`

Item	Description
ID Description Date Responsible Subject Precondition Steps	UNIT5.8 Test of the web access module <code>get_instructions.php</code> . 01.11.12 Yngve The database and <code>get_instructions.php</code> . A working database with the table <code>MEDICINE_INSTRUCTIONS</code> . <ol style="list-style-type: none"> 1. Initialize a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/get_instructions.php?medicine_id=1</code>. 2. Observe the returned JSON data.
Results	The returned JSON data was: <pre> 1 { 2 sqlsuccess: true, 3 child_id: "1", 4 query: "SELECT * FROM 'MEDICINE_INSTRUCTIONS' WHERE id IN (SELECT 5 instructions_id FROM 'MEDICINES' WHERE 'id'='1') LIMIT 0,1", 6 instructions: [7 { 8 id: "1", 9 url: "medicine_flutide_50microg.jpg", 10 information: "Et inhalasjonssteroid som brukes fast slik legen har 11 bestemt. Effekten ses først etter noen dagers bruk, og gjor 12 at betennelsesreaksjonen i lungene til barnet demper seg. 13 Dette hindrer barnets astma/betennelsesreaksjon i a utvikle 14 seg og hindrer utvikling av sykdommen.", 15 effect: "Alle sprayinhalasjoner ma gis pa inhalasjonskammer (16 Aerochamber, Optichamber, Babyhaler eller lignende) for a 17 sikre at barnet far pustet inn medisinien pa riktig mate. Etter 18 inhalasjon med steroider ma barnet alltid skylle munn, drikke 19 eller pusse tenner for a fjerne rester av pulver fra munnen. 20 Blir restene igjen i munnen kan det oppsta en smertefull 21 soppinfeksjon i munnen." 22 } 23 } </pre>
	<p style="text-align: center;">Listing 12.8: Returned JSON from <code>get_instructions.php</code></p> <p>We see from the JSON data that the module works by checking the query and the variable <code>sqlsuccess</code>. We get the information that the medicine with $ID = 1$ has three types of instructions: an image given by an URI, a general information field and a field to explain the effects.</p>

Table 12.9: Unit test 5.8, `get_instructions.php`

Item	Description
ID	UNIT5.9
Description	Test of the web access module <code>get_log_days_for_child.php</code> .
Date	03.11.12
Responsible	Yngve
Subject	The database and <code>get_log_days_for_child.php</code>
Precondition	A working database with the tables <code>DAY_MEDICINE_DOSES</code> and <code>CHILDREN_LOG_DAYS</code> .
Steps	<ol style="list-style-type: none"> 1. Initialize a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/get_log_days_for_child.php?child_id=6</code> 2. Observe the resulting JSON data.
Results	<p>The returned JSON data was:</p> <pre> 1 { 2 sqlsuccess: true, 3 child_id: "6", 4 year: "2012", 5 month: "11", 6 query: "SELECT * FROM 'DAY_MEDICINE_DOSES' WHERE 'child_id'='6' AND YEAR('day_date')='2012' AND MONTH('day_date')='11' LIMIT 0,100", 7 statuses_query: "SELECT 'date', 'child_id', 'health_state_id' FROM 'CHILDREN_LOG_DAYS' WHERE 'child_id'='6' ORDER BY 'date' ASC", 8 days: [9 { 10 date: "2012-11-01", 11 health_state_id: "1", 12 doses: [] 13 }, 14 { 15 date: "2012-11-02", 16 health_state_id: "1", 17 doses: [18 { 19 id: "126", 20 reward: "1", 21 time: "00:00:01", 22 day_date: "2012-11-02", 23 child_id: "6", 24 medicine_id: "1", 25 health_state_id: "1", 26 medical_plan_dose_id: "0", 27 pollen_state_id: "0" 28 } 29], 30 }, 31] } </pre> <p>Listing 12.9: Returned JSON data for <code>get_log_days_for_child.php</code></p> <p>As we see from the returned data, the child with $ID = 6$ has one registered dose in november, and the date was the third. We also see that the health state was changed to $health_state_id = 3$ on the third. It is clear that the module is working because of the returned data, validating the queries and the variable <code>sqlsuccess</code>.</p>

Table 12.10: Unit test 5.9, `get_log_days_for_child.php`

Item	Description
ID	UNIT5.10
Description	Test of the web access module <code>get_log_for_child.php</code> .
Date	03.11.12
Responsible	Yngve
Subject	The database and <code>get_log_for_child.php</code> .
Precondition	A working database with the table <code>DAY_MEDICINE_DOSES</code> .
Steps	<ol style="list-style-type: none"> 1. Initialize a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/get_log_for_child.php?child_id=6</code>. 2. Observe the returned JSON data.
Results	<p>The resulting JSON data was:</p> <pre> 1 { 2 sqlsuccess: true, 3 child_id: "6", 4 year: "2012", 5 month: "11", 6 query: "SELECT * FROM 'DAY_MEDICINE_DOSES' WHERE 'child_id'='6 AND 7 YEAR('day_date')=2012 AND MONTH('day_date')=11 LIMIT 0,100", 8 days: [9 { 10 id: "126", 11 reward: "1", 12 time: "00:00:01", 13 day_date: "2012-11-02", 14 child_id: "6", 15 medicine_id: "1", 16 health_state_id: "1", 17 medical_plan_dose_id: "0", 18 pollen_state_id: "0" 19 } 20] } </pre> <p>Listing 12.10: Returned JSON data from <code>get_log_for_child.php</code></p> <p>We see from the returned data that there was one logged medication and that the module works from checking the query and the variable <code>sqlsuccess</code>.</p>

Table 12.11: Unit test 5.10, `get_log_for_child.php`

Item	Description
ID	UNIT5.11
Description	Test of the web access module <code>get_plan.php</code> .
Date	04.11.12
Responsible	Yngve
Subject	The database and <code>get_plan.php</code> .
Precondition	A working database with the tables MEDICAL_PLAN_DOSES and CHILDREN.
Steps	<ol style="list-style-type: none"> 1. Initialize a web browser with the GET url <code>http://folk.ntnu.no/yngvesva/blopp/get_plan.php?child_id=6</code>. 2. Observe the returned JSON data.
Results	<p>The returned JSON data was:</p> <pre> 1 { 2 sqlsuccess: true, 3 child_id: "6", 4 query: "SELECT Mpd.id, Mpd.medical_plan_id, Mpd.health_state_id, Mpd .time, Mpd.medicine_id, M.color AS 'medicine_color', M.name AS ' medicine_name' FROM 'MEDICAL_PLAN_DOSES' AS Mpd, 'MEDICINES' AS M WHERE Mpd.medical_plan_id IN (SELECT medical_plan_id FROM ' CHILDREN' C WHERE C.id=6) AND Mpd.medicine_id = M.id LIMIT 0,100 ", 5 rows: [6 { 7 id: "41", 8 medical_plan_id: "5", 9 health_state_id: "1", 10 time: "01:09:00", 11 medicine_id: "1", 12 medicine_color: "BLUE", 13 medicine_name: "Flutide" 14 } 15] 16 }</pre>

Listing 12.11: Returned JSON data for `get_plan.php`

We see from the returned data that there is one medication of Flutide planned at 01:09:00. We conclude that the module is working from looking at the query and checking the variable `sqlsuccess`.

Table 12.12: Unit test 5.11, `get_plan.php`

Item	Description
ID	UNIT5.12
Description	Test of the web access module <code>register_medicine_taken.php</code> .
Date	04.11.12
Responsible	Yngve
Subject	The database and <code>register_medicine_taken.php</code> .
Precondition	A working database with the tables <code>DAY_MEDICINE_DOSES</code> , <code>HEALTH_STATES</code> and <code>CHILDREN</code> .
Steps	<ol style="list-style-type: none"> 1. Initialize a REST client with the URL <code>http://folk.ntnu.no/yngvesva/blopp/register_medicine_taken.php</code>. 2. Add the POST parameters: <ul style="list-style-type: none"> • <code>child_id</code>: 6 • <code>medicine_id</code>: 1 • <code>day_date</code>: 2012-09-03 • <code>health_state_id</code>: 2 3. Observe the returned JSON data.
Results	The returned JSON data was:
	<pre> 1 { 2 "sqlsuccess": true, 3 "post": { 4 "child_id": "6", 5 "medicine_id": "1", 6 "day_date": "2012-09-30", 7 "health_state_id": "2" 8 }, 9 "q": "INSERT INTO 'DAY_MEDICINE_DOSES' ('id', 'reward', 'time', " 10 'day_date', 'child_id', 'medicine_id', 'health_state_id', " 11 'medical_plan_dose_id', 'pollen_state_id') VALUES ('', '2', 12 '00:00:01', '2012-09-30', '6', '1', '2', '', '')", 13 "reward": "2", 14 "unique": true 15 } </pre>
	<p>Listing 12.12: Returned JSON data from <code>register_medicine_taken.php</code></p> <p>We see from the returned data, by looking at <code>query</code> and <code>sqlsuccess</code>, that an entry was added to the log. It was given a default time of 00:00:01 since we did not specify it, and the returned calculated reward was 2. We can conclude that the dose was registered successfully and the module works.</p>

Table 12.13: Unit test 5.12, `register_medicine_taken.php`

Item	Description
ID	UNIT5.13
Description	Test of the web access module <code>remove_plan_dose.php</code> .
Date	04.11.12
Responsible	Yngve
Subject	The database and <code>remove_plan_dose.php</code> .
Precondition	A working database with the table MEDICAL_PLAN_DOSES.
Steps	<ol style="list-style-type: none"> 1. Initialize a REST client (POSTMAN) with the URL <code>http://folk.ntnu.no/yngvesva/blopp/remove_plan_dose.php</code>. 2. Add the POST parameter: <ul style="list-style-type: none"> • <code>id: 41</code> 3. Observe the returned JSON data.
Results	<p>The returned JSON data was:</p> <pre> 1 { 2 "sqlsuccess": true, 3 "q": "DELETE FROM 'MEDICAL_PLAN_DOSES' WHERE id='41'", 4 "id": "41", 5 "num_deleted": 1 6 }</pre> <p style="text-align: center;">Listing 12.13: Returned JSON data from <code>remove_plan_dose.php</code></p> <p>We conclude from the query, <code>sqlsuccess</code> and <code>num_deleted</code> that the row with $ID = 41$ was deleted and that the module works.</p>

Table 12.14: Unit test 5.13, `remove_plan_dose.php`

Item	Description
ID	UNIT5.14
Description	Test of the web access module <code>remove_plan_medicine_at_time.php</code> .
Date	04.11.12
Responsible	Yngve
Subject	The database and <code>remove_plan_medicine_at_time.php</code>
Precondition	A working database
Steps	<ol style="list-style-type: none"> 1. Initiate a REST client (POSTMAN) with the URL <code>http://folk.ntnu.no/yngvesva/blopp/remove_plan_medicine_at_time.php</code>. 2. Add the POST parameters: <ul style="list-style-type: none"> • <code>time</code>: 12:34:56 • <code>medicine_id</code>: 1 • <code>child_id</code>: 6 • <code>health_state_id</code>: 3 3. Observe the returned JSON data.
Results	<p>The returned data is:</p> <pre> 1 { 2 "sqlsuccess": true, 3 "q": "INSERT INTO 'MEDICAL_PLAN_DOSES' ('id', 'medical_plan_id', 'health_state_id', 'time', 'medicine_id') SELECT '', C. 4 medical_plan_id, '3', '12:34:56', '1' FROM 'CHILDREN' AS C 5 WHERE C.id='6'", 6 "child_id": "6", 7 "health_state_id": "3", 8 "medicine_id": "1", 9 "time": "12:34:56", 10 "id": 44 11 }</pre> <p>Listing 12.14: Returned JSON data from <code>remove_plan_medicine_at_time.php</code></p> <p>We see from the data that a medication dose was removed, and by the query and <code>sqlsuccess</code> we see that the module works.</p>

Table 12.15: Unit test 5.14, `remove_plan_medicine_at_time.php`

Item	Description
ID	UNIT5.15
Description	Test of the web access module <code>set_child_state.php</code> .
Date	05.11.12
Responsible	Yngve
Subject	The database and <code>set_child_state.php</code> .
Precondition	A working database with the tables <code>CHILD_HEALTH_STATES</code> and <code>CHILDREN_LOG_DAYS</code> .
Steps	<ol style="list-style-type: none"> 1. Initialize a REST client (POSTMAN) with the URL <code>http://folk.ntnu.no/yngvesva/bopp/set_child_state.php</code>. 2. Add the POST parameters: <ul style="list-style-type: none"> • <code>child_id</code>: 6 • <code>state_id</code>: 2 3. Observe the returned JSON data.
Results	<p>The returned JSON data was:</p> <pre> 1 { 2 "sqlsuccess": true, 3 "child_id": "6", 4 "state_id": "2", 5 "update_query": "UPDATE 'CHILD_HEALTH_STATES' SET applies_now = IF 6 (health_state_id = '2', 1, 0) WHERE child_id='6'", 7 "insert_query": "REPLACE INTO 'CHILDREN_LOG_DAYS' ('date', 8 'child_id', 'pollen_state_id', 'health_state_id') VALUES 9 ('2012-11-05', '6', '1', '2')" }</pre> <p>Listing 12.15: Returned JSON from <code>set_child_state.php</code></p> <p>We see from the returned JSON data that two tables were updated: the child's current state, and the log of state changes. We conclude that the module works because of the queries and the variable <code>sqlsuccess</code>.</p>

Table 12.16: Unit test 5.15, `set_child_state.php`

Item	Description
ID	USABILITY5.1
Description	Test to see if the children can follow the instructions given and take their medicine correctly when an alarm is given.
Date	30.10.2012
Responsible	Eirik
Subject	The Karotz application and CAPP
Precondition	<ul style="list-style-type: none"> • Working version of CAPP and the karotz application. • The child registered in the database is in the correct health state. • Child and one parent present. The parent should be familiar with giving asthma medications. • Wireless wpa2 secured connection.
Steps	<ol style="list-style-type: none"> 1. Explain the test for the children, in an easy to understand way. 2. Set up an alarm to go off. 3. Let the parent get the alarm, and see how well the child and parent is able to follow the instructions given by the system.
Results	Generally the system worked as intended. The children successfully took their medicine, and had little trouble following the instructions. We discovered some errors relating to the alarm, aswell as some of the distraction sequence bugging, some parts skipped stages if the child pressed more than once, others had inaccurate instructions (like holding the Nanoz infront of karotz belly, when it's the nose it needs to be in front of).

Table 12.17: USABILITY5.1

Item	Description
ID Description Date Responsible Subject Precondition	INTEGRATION5.1 Test of CAPPs alarm and distraction sequences. 06.11.12 Eirik CAPP
	<ul style="list-style-type: none"> • Working version of CAPP installed on phone or AVD (Android Virtual Device). • Internetconnection for databaseaccess. • Medicationplans for the correct healthstate registered in the database.
Steps	<ol style="list-style-type: none"> 1. Turn on the phone or AVD, and wait until the time of the alarm. 2. Receive the alarm. 3. Press the medicine to start the distraction sequence 4. Follow the instructions and take note if any instructions are skipped or missing.
Results	We found that the alarm set alarms for all healthstates, not just the one the child was currently in. Aside from this, the alarm fired correctly. Some of the changes we had done after the usability test made one of the sound files and animations in the distraction sequence unsynchronized. Since the distractionsequence have checkpoints where the user have to interact with the application, only that specific part of the distraction was out of synch, the rest ran as intended.

Table 12.18: INTEGRATION5.1

Item	Description
ID	INTEGRATION5.2
Description	Testing that the log updates correctly based on registered medication and pollen feed.
Date	06.11.12
Responsible	Esben
Subject	GAPP
Precondition	<ul style="list-style-type: none"> • Working version of GAPP installed on phone or AVD. • Internetconnection for databaseaccess. • Completed medications and pollen feeds registered in the database.
Steps	<ol style="list-style-type: none"> 1. Open GAPP on the phone or AVD. 2. Navigate to the log. 3. Check if the visual representation in the log corresponds to the data in the database.
Results	The log updated correctly.

Table 12.19: INTEGRATION5.2

Item	Description
ID	INTEGRATION5.3
Description	Testing that the medicationplans is correctly registered to their respective healthstates
Date	05.11.12
Responsible	Eirik
Subject	GAPP
Precondition	<ul style="list-style-type: none"> • Working version of GAPP installed on phone or AVD. • Internetconnection for databaseaccess. • Medicationplans registered to various healthstates.
Steps	<ol style="list-style-type: none"> 1. Open GAPP on the phone or AVD. 2. Navigate to medisinplan page. 3. Choose each of the healthstates in turn, and see if the medicines listed under each corresponds to the medicationplans in the database.
Results	The medicationplans is correctly registered to their respective healthstates.

Table 12.20: INTEGRATION5.3

12.4.2 Results

The usability test gave us much feedback on the system, and several errors that had to be fixed. The complete testresults can be found in section 13.3.2, but the main issues were centered around the alarms in CAPP, and the distraction sequence being vulnerable to skipping instructions if the children pressed too fast, which was often the case.

The unit tests performed during this sprint all returned the expected results, which then allowed us to move on to the integration tests we had planned. These uncovered some more issues that either wasn't caught by the usability test, or were a result of the changes we made in response to the usability test. The new dialogue added to the CAPP distraction sequence meant that some of the animations now were not synchronized with the sound of the Karotz counting. To fix this we made separate manuscripts for the android version and the Karotz, making it easier to synchronize. We also discovered a problem where the alarms were registered from all health states, not just the one the child is currently in, and sending many more alarms than necessary. This problem was fixed by a method checking the health state before invoking alarms.

12.5 Sprint Retrospective

This section contains an evaluation of the sprint. The evaluation is done mainly by the us, but feedback from the customers are added to the retrospective.

12.5.1 What went well?

We finished the implementation of desired functionality in time. Even though not all functional requirements were fulfilled, the requirements fulfilled are done properly and without major errors.

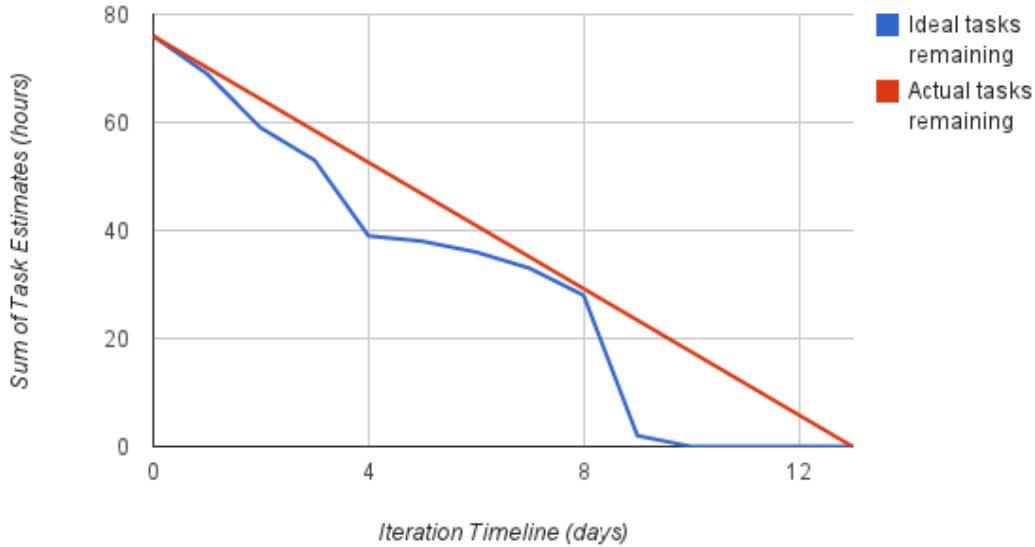


Figure 12.1: Sprint 5 burndown chart

12.5.2 What shall we start doing?

1. Focus only on the report and the presentation

12.5.3 What could have gone better?

All went very well during this sprint.

12.5.4 What should we stop doing?

The sourcecode is now under code-freeze. If errors are found, we should report them in the section for further work.

12.5.5 Sprint Burndown Chart

Table 12.21 and figure 12.1 show the burndown chart for the fifth and final sprint. Since we had a reduced amount of story points for this sprints, everything was completed three days before schedule.

We had planned 19 story points for the fifth sprint, which meant 72 estimated work hours. We worked for a total of 64 hours, and had 0 hours left at the end of the sprint, so all 19 story points were completed. The reduced work speed when compared to other sprints can be explained by a shifted focus from programming tasks which are documented in the sprint, to planning and documentation exercises related to the usability test, the report and the final presentation. These hours are not formalized as sprint tasks and are therefore not included in the estimates. We were satisfied with the sprint result.

12.5.6 Screenshots

Since this was our last sprint, we have included some screenshots of the work that is done.

# ID	Task	Story points	Estimated hours	Actual	Estimated left	Responsible
1.1	Refine distraction for CAPP	2	8	6	0	Yngve
1.2	Fix medicine choice before unscheduled medication	2	8	7	0	Yngve
1.3	Fix jumping images during child distraction	1	4	4	0	Yngve
1.4	Fix correct heading in all pages	1	4	4	0	Eirik and Esben
2.1	Bugfix new manuscript on Karotz	1	4	2	0	Yngve
2.2	Bugfix Karotz after usability test: double RFID check	1	4	1	0	Yngve
3.1	Implement Thobias' changes to the report	1	4	4	0	Yngve
3.2	Document usability testing, add usability test to sprint 4	1	4	4	0	Eirik
4.1	Delete old alarms	2	8	8	0	Eirik
5.1	Make better layout for info about medication	1	4	4	0	Aleksander
6.1	Write javadoc for all code	3	12	7	0	Esben, Eirik
6.2	Remove unused code (imports, classes, etc)	1	4	3	0	Aleksander, Eirik and Esben (etc)
7.1	Fix Wifi-related crashes in CAPP	0.5	2	1	0	Esben
7.2	Fix Wifi-related crashes in GAPP	0.5	2	1	0	Esben
8.1	Perform and document unit tests of all we access pages	1	4	8	0	Yngve
SUM		19	72	64	0	

Table 12.21: Sprint Retrospective, Sprint 5
120

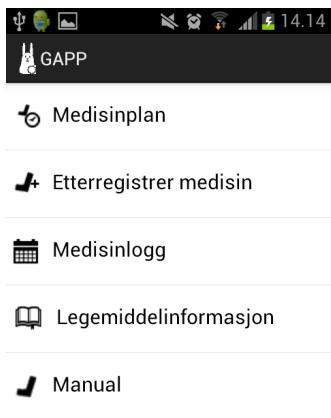


Figure 12.2: GAPP main menu

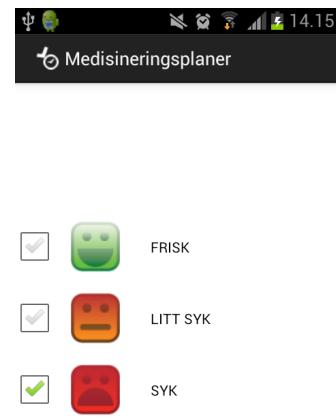


Figure 12.3: Available plans in GAPP

12.5.7 GAPP - Screenshots

Figure 12.2 shows the main menu. The item “manual” is a bit misguiding, as this shows an image gallery of how a child should take the medicine (shake, put on mask, and so on). We did not have time to make a proper manual on how to use the application, but hopefully, the application is intuitive enough.

Figure 12.3 shows the available plans for an adult. By pressing one of the checkboxes, the child’s medical plan is updated appropriately. By touching the name of a plan, you get presented with the screen below.

Figure 12.4 view contains a list of the medicines that are stored in a medicine plan. You can add a medicine through the button, and by touching a medicine, you can delete it.

Figure 12.8 shows the log of a child. The log consists mainly of 3 components. The calendar shows the days of a month, the chosen medicine plan for a child on a given day, indicated by a green/yellow/red line, and the distribution of pollen at that day, indicated by the bottom line of each cell.

This pollen distribution is given by our dummy xml-feed (since pollenvarslingen.no is not currently casting). We have used the same distribution for each day, which explains why every day is brown. Down to the left, one can see how many times the child has taken the given medicine at a day. Down to the right, you can see the pollen spread at the day for every type of pollen that is casted by pollenvarslingen.no. The cells of the calendar is touchable, and the listviews on the bottom of the page is updated according to which day is selected.

It is possible to register a treatment after the medication is taken. This is done by choosing the date (which is autofilled with todays date), and choosing a medicine. The child then gets stars in his/her treasure chest.

CAPP - Screenshots

Figure 12.9 shows the main menu of CAPP. This main menu has three items. To start a treatment, the user chooses the karotz-icon. To see amount of stars, the child picks the treasurechest, which leads the user to Figure 12.11. The book leads to a series of children-friendly images, where the child can have a look at instructions for taking a medicine.

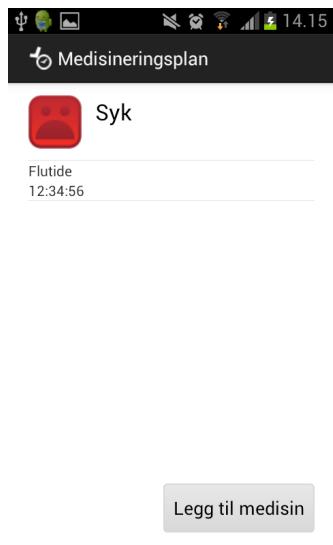


Figure 12.4: A medication plan in GAPP



Figure 12.5: Register treatment in GAPP



Figure 12.6: Choose among medicines to view more information



Figure 12.7: Medicine specific information



Figure 12.8: Medicine log in GAPP

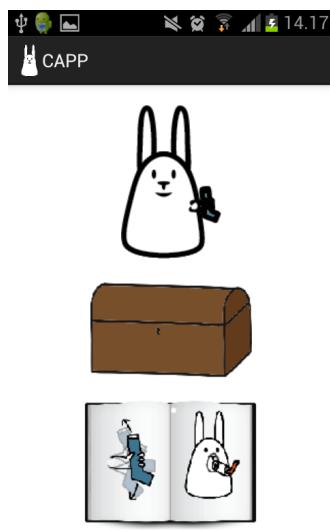


Figure 12.9: Main menu in CAPP

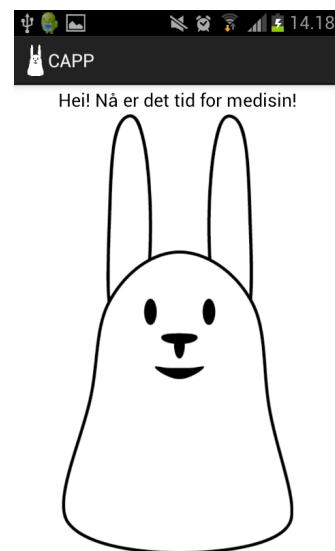


Figure 12.10: Start a treatment in CAPP



Figure 12.11: Amount of stars collected in CAPP

Chapter 13

Usability Testing

This section will explain what usability testing is, and what we did during our project relating to usability testing.

13.1 What is usability testing

Usability testing is a way to test your system on users, using the interaction design of the system. Usually this entails setting up a few tasks for a user, giving the user access to the system and seeing how well the user is able to solve the tasks given in the test with the system provided. The usability testing focuses on the main functionality rather than on the details.

13.2 How to do usability testing

The test usually starts with the tester explaining for the user the different parts of the systems being used, and that it's the system being tested, not the user. The tester also explains that this means the user can't do anything wrong, if there is something the user doesn't understand, that is fine. The tester observe what the user finds hard, or can't do at all, and notes this down as things that might have to be changed on the system.

At last, the user is given an evaluation form of the system, typically a SUS-form (System Usability Scale), and time to talk about how well they felt the system helped them solve the tasks given.

13.3 Usability testing in our project

Our project largely implements user interaction, and usability testing plays a key role in getting a good result, since so much of the project is centered around the user interface. We started the project with an early paper prototype test, to get some feedback on the ideas we got after the workshop we did with the interaction design group. After this we worked for some time on implementing the system, before we late in the project did a full usability test using the hallway method, testing our system on children to see if it helped them take their medicine, and seeing if there was some bugs or things we could implement better.

13.3.1 Paper prototype test

In the early stages of development, it might be useful to do a paper prototype test. This test is done by making a prototype of the user interface on paper, and having one person (the "computer") change between the different paper screens based on what the user being tested clicks on.

We did some early paper prototype testing in our project, on the 4th of September. After our workshop we had made some screen layout mockups, see figure 13.1, and we wanted to test how user friendly these were.

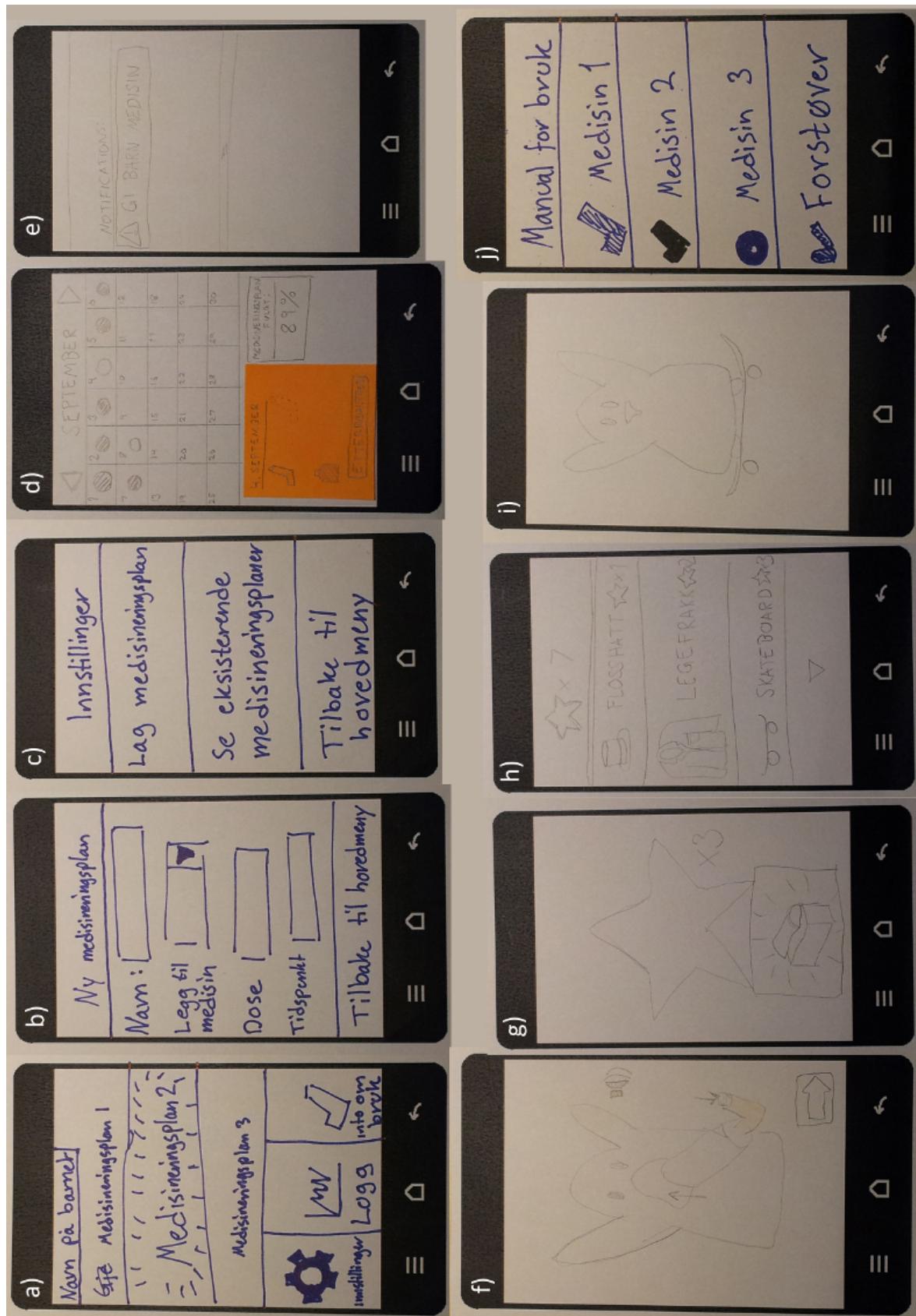


Figure 13.1: The screens of the paper prototype:
 a) The GAPP mainscreen.
 b) GAPP new medication screen.
 c) GAPP settings screen
 d) GAPP log screen.
 e) Notification on phone
 f) One of the CAPP distraction screens.
 g) The CAPP reward screen
 h) CAPP shop screen
 i) CAPP avatar screen, after buying skateboard in the shop
 j) the GAPP manual screen

The test was done with the expert review method. We had a person from NAAF come for the test, to solve the tasks with the system we provided. We chose to do the expert review mainly because our funder was interested in seeing how the project was proceeding. NAAF is also the main provider of information such as instructions and how the medical plans looks, so having one of them see the system was helpful, as we at this point was uncertain about a lot of these things.

From the development group, Eirik and Aleks attended the test. Eirik played the part of the computer, while Aleks introduced the test, explained the system and handled any other talking that needed to be done.

The preconditions we assumed to be present for the CAPP was:

- A paper prototype with the correct screens.
- Basic knowledge of Android devices in the user being tested.
- An active notification for taking medicine on the android device.

for GAPP these precondition were:

- A paper prototype with the correct screens.
- Basic knowledge of android devices in the user being tested.
- The correct medication plans and medicines registered in the system.

We had prepared a set of tasks for each of our two android applications, CAPP and GAPP. We did not have a functional application for the karotz at this point, and it was difficult to test this with a paper prototype, so we chose to leave that part out of this test. The cases can be seen in table 13.1 and table 13.2.

Task	Description
1	There is an active notification for medication on the phone, follow the instructions given by the android device.
2	Go to the shop in CAPP and buy a skateboard for the avatar.

Table 13.1: The tasks for CAPP

Task	Description
1	Open GAPP.
2	Register use of "Medisin 1" on 4th of september.
3	Check for more information on correct usage of "Medisin 2".

Table 13.2: The tasks for GAPP

This test gave us some strong feedback on what parts of the system worked well, and what didn't make sense at all. We also sat down and had an interview with the test person after the test regarding the best way to present the information we would get from NAAF. Some of the problems the test person had might be credited to her being inexperienced with android devices, but we noted down these problems as well. In short the paper prototype test gave us the following results:

- Having the reminder as a notification is insufficient. The test person had trouble noticing it even though she was told it would be there.
- CAPP have to inform how many times you should press the medicine during medication.

- The rewardscreen in CAPP is not intuitive enough.
- The main menu in GAPP needs a complete rework, the menu system was not easy to understand.
- The Log have to be clearer on how to register medicines, and which medicines is already registered.
- There is no back button in the application. This is a problem with android experience, and we didn't implement a back button in the end.
- The Information about correct use of medicines was not easily accessible.

The paper prototype led to a major overhaul of the user interface.

13.3.2 Usability testing of the distraction

Since our system is targeted towards children it was important to test the system properly on children, to see if the effects where as we hoped for. This was difficult to do early in the project since the children could not be expected to deal with paper prototypes, as they were as young as three years old. Because we had to create our system from scratch, it took a long time to get something robust enough to test, there was a long process of unit and integration testing going into making the communication between two Android applications and one Karotz application.

This led to a late usability test, held on 30th of October 2012. At this point we had a working version of the system.

The preconditions for the test were:

- Working version of CAPP and the karotz application.
- The child in the database is registered to the correct health state.
- Child and parent present. The parent should be familiar with giving asthma medications¹.
- Wireless wpa2 secured connection for the karotz.

The test started with some basic introduction, done informally. Trying to run a standard usability test on already shy children was not optimal, so we explained to the parents, and let them tell the children what was going to happen. After giving an introduction, we registered a new medication plan in the database, with alarm set for one or two minute in the future. We had to register it this way to make sure the reward system and log updated properly in response to the treatment.

The test scenario we wanted for CAPP was as follows:

- The Android device receives an alarm. The parent hands the phone to the child.
- After finding the correct medicine, based on the picture on the alarmscreen, the child starts the distraction sequence.
- The child follows the instructions given during the distraction sequence, and does as the karotz on the screen.
- The child, helped where necessary by their parent, successfully takes their medicine.
- After the distraction finishes, the child is done with their treatment, and receives the reward in the application.

For the karotz application, this scenario plays out a little bit differently:

¹The application gives information about this, but it is not what is being tested

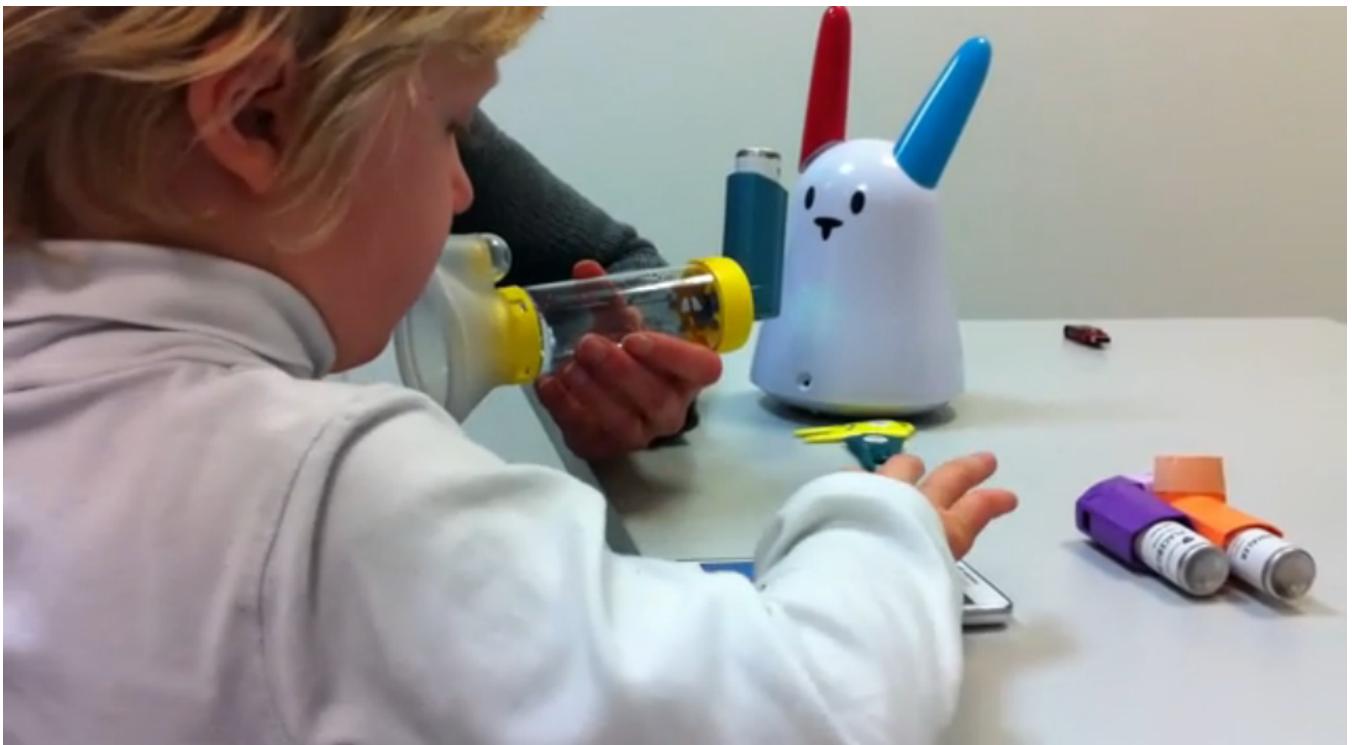


Figure 13.2: Child taking their medicine while following the CAPP distraction sequence. Photo: Elin Høien

- The robot gives a notification.
- The child follows the instructions to turn off the notification and ready the distraction.
- After fetching the parent, the child is given the yellow nanoz by their parent, and holds the yellow nanoz close to the karotz' nose to start the distraction sequence.
- The child and parent follows the instructions given during the distraction sequence, which helps the child successfully take their medicine.
- After the distraction sequence finishes the child uses the green nanoz to collect their reward from the karotz.

Before the test we had already discovered that the Karotz was quite selective in its internet connections security protocols, and it could not connect to for example the "eduroam" network found at the university. The previous day we tried using a wireless router of our own and connecting this to one of the internet cables at a computer lab at NTNU. This approach had worked fine, but we wanted to make sure it was up and running by the time the test were to start. We found out this would not work with the internet at NSEPs usability lab, and we only barely got it up and running by using a smartphone to set up a wireless hotspot for the Karotz. A description of the problems are reported, for future development in 14.1.

The test was done using the hallway method, meaning we let users who had little or no prior knowledge of the system test it. Some pictures from the tests can be seen below:

After the internet connection for the karotz was secured, the test ran without other big problems. We observed the following during the tests: the children managed to follow the instructions given by both CAPP and the Karotz application. Some of the children were reluctant to actually take medicine, but they were eager to see what happened next on the application, and we hope this will motivate them to take their medicine, even for the children who does not really want to take their medicine. We feared that our reward system would be too simple and therefore not rewarding enough, but the children who tested the



Figure 13.3: Child taking their medicine while following the Karotz distraction sequence. Photo: Elin Høien

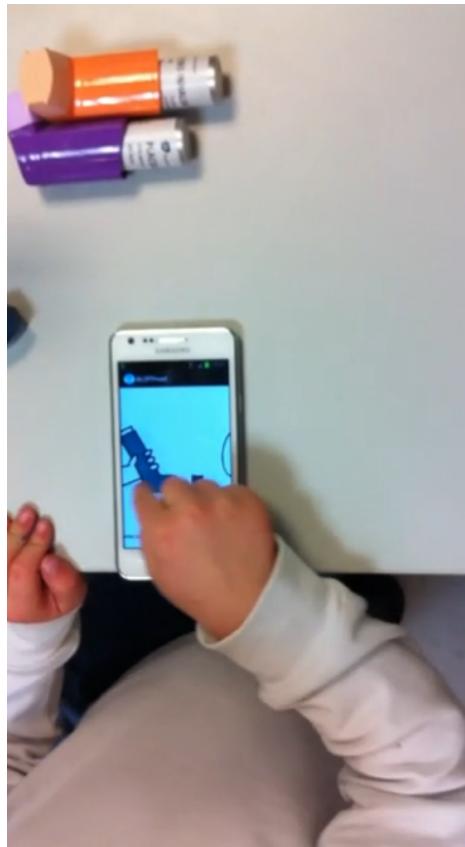


Figure 13.4: Child distributing medicine to the karotz while looking at the instruction manual in CAPP. Photo: Elin Høien

system seemed very interested in it, even though it was just points you gather in the form of stars. One of the children started comparing the amount after each treatment and ran the treatment many times in order to get the most points possible. (We later implemented a time check, blocking the user from doing several treatments in a row). We noted that the children were quick to become friendly with the Karotz, especially when it started talking.

We also uncovered quite a few bugs and parts that needed to be redone. For CAPP, the most important of these was:

- The alarms were not deleted properly. Since we update them regularly to ensure they're fired properly, this resulted in alarms firing at the same time, resulting in a lot of noise, aswell as untimely fired alarms (which should have been moved).
- The alarm sound did not turn off when the distraction sequence ended, or when the "stop alarm" button was pushed.
- After the distraction sequence the children did not understand that the chest was interactable, since the rest of the sequence were spoken, and this was not. However, after seeing the chest in the main menu, they understood that they could click it.
- If the user pressed the screen multiple times, this registered as multiple clicks, and parts of the distraction sequence was skipped. This happened frequently, as the children pressed again because of the delay between the first click register, until the screen updated.
- Part of the instructions had been left out in the distraction sequence, namely the one about rinsing your mouth after taking the medication.
- The instruction about pressing the medicine once, before breathing 10 times, were not clear. One of the children pressed the medicine 10 times.

There were also bugs and problems related to the Karotz, aside from the big issue with network connection:

- The Karotz instructed the user to hold the Nanoz close to its belly, which is incorrect. For the Nanoz to register, they must be held close to the nose.
- The distraction sequence never says to attach the medicine to the chamber.
- Holding the Nanoz close for too long makes the Karotz skip instructions.
- As with CAPP, it was not clear that they had to press the medication once before inhaling 10 times.

Chapter 14

Further Work

This chapter gives an overview of some of the ideas both the customer and the developers had for further development of the application. This includes a description of further development, analysis of the user groups and work towards NAAF and the health department. The main part of the work to be done after the end of this project is connected to requirements that has been taken out of this project due to limitation of time and resources. Other issues remaining is connected to the security and privacy of the patient's treatment log and storing sensitive information. Section 15.3 lists the overall requirements that have not been implemented during the project. These requirements has either been requested early in the process or have been brought up during discussions and meetings with the stakeholders.

14.1 Improvements

The following sections describes the ideas we had for future improvements to the applications. It is parted into subsections for improvements in the fields of database records, the reward system, the distraction and the web application.

14.1.1 Wifi access and caching of database records

The application is currently very dependent on having network access in order to run at all. We are making a lot of HTTP requests, and we never cache the results. This introduces performance issues. This could be avoided by introducing a caching service. The application should store information about when the last database update occurred, and update the cached information appropriately. This is a part of the system that would take a lot of time to implement, and we just did not have enough time.

Once the device loses its internet connection, the application should use these cached results to update the different screens, for example the log. This caching service should also cache information about treatments that is done when the device is not connected. For instance, if a child takes a medication, the device stores this information. Once the device connects to internet, the applications should push these changes to the webservice, which, in turn, updates the database records, and thus we have applications that is not so "tied up" to a device's internet access. This could also introduce the possibility to never connect to internet at all, by just using the stored data.

14.1.2 Security and privacy

The application is, as mentioned, using a webservice to access database records. This webservice is currently not using any form of encryption or protection. Ideally the webservice should be using the HTTPS protocol instead of HTTP, and have proper password protection, to avoid that records of childrens medication history is publicly available. This will have some issues on performance, but if a proper caching system is implemented as mentioned in Section 14.1.1, it would be hardly noticeable.

The only “identifier” we are currently using is the database “id” of a child, which is an automatically incremented number, and can be seen as a random identifier for a child, meaning that it does not have anything to do with a child at a personal level (not a username, name, emailaddress, ssn, etc.). This raises a question whether or not privacy is a real issue here. For one to actually know which child has been taking which medicines, one must either know the stored database id for a child, or hack the system and get access to the name. From our perspective, privacy is not a real issue that needs to be taken care of at the time being.

If however, the application in the future becomes more personalized as mentioned in Section (Minor improvements), these are issues that needs to be taken seriously, as medical history is considered sensitive information.

14.1.3 Rewardsystem

The children’s application (CAPP) is all about changing the children’s view of medication to something positive. It shall be a motivation for the children to take their medication. It is therefore an important task to entertain them and give them some form of reward when they take their medication. As for now, we have given stars to the child after completed medication. The stars are in a treasure chest where the child can see how many stars he or she has. This is a simple reward, but worked fairly well during the user tests. However, it may be boring over time.

The initial idea was to have a shop where the children could buy clothes and other items to their avatar. The stars earned from finishing treatments would serve as credits in the shop. This was not implemented due to time restrictions. It is also possible to take this to the real world, e.g. that the child gets a lollipop for every 10th star, but this would have to be supervised by the parents.

There is an endless line of opportunities for this reward system, and we chose the simplest implementation, so we would have something to test.

14.1.4 Distraction sequence for children

During our workshop, we came up with a lot of ideas for distractions for the children. These would range from simple animation sequences, like what we decided to implement, to more complex things like games that would not require a lot of movement and could therefore help during longer treatments.

The distraction sequence is one of the fields were we feel it has more or less never ending possibilities for improvement, and as more research into what children finds distracting, but not to the point where they can’t take their medicine, this distraction sequence can be evolved.

14.1.5 User testing of the guardian application

GAPP has not yet been user tested on actual parents of asthmatic children. This has to be done to get an understanding of how they interact with the system, and to get knowledge about what they think of an application of this type. This is a system to make it easier for the guardians to give their children medications. While it is important that the children likes the system, it is also important that the parents feel it helps them give their children their medicines, without it being a big time waster.

14.1.6 Web application

There is a possibility of making this application as a web application, as a whole. By extracting the functionality and running it on a web service it would make it easier for people to use it across platforms. Done right, it may run on all devices with an internet connection. This may also give an easier integration with external information such as air pollution forecast, pollen forecast, temperatures, etc. Since our application is written in Java, using Android SDK, it will not run on an internet server as is. Making a web application will require an almost complete refactoring of the source code.

14.1.7 Support for more children

Currently, the application only use one child, but there are implemented support for using more children. Each child has its own id (childId), and support for more children can be implemented without much change of the existing code. There should also be considered using accounts for the guardians connected to the children, in case of the guardians having more than one asthmatic child.

14.2 Ideas and minor improvements

Webinterface The doctors may prefer to set up the users medication plans through a web interface on their computers. This part may be integrated into existing systems.

Other devices The application are fitted for a phone running the Android operating system. For the future it should also be scalable to tablets. There may be more interesting for a child to work on a tablet than a phone. There will also be much more space for content. This extra space gives greater potential of the reward system. It should also be available on other operating systems than Android, e.g. iOS or Windows Phone. This will improve the availability for the users, not limiting them to Android phones.

Overall graphical design The priorities have been to make the major functionality work. We have used lots of time making the applications understandable and easy to use, but there is still a great potential in making the applications interaction design better.

Personalize the system The application may be more personalized. E.g. "It's time to take medication" could be "It's time to take medication, Eric". By involving the users name more in the system, they may feel more appreciated.

Integration of external elements The distraction part of the application may be integrated with a story or other external elements. I. eg. a story where the children will need to take medicine in order to get the next part of the story.

Chapter 15

Evaluation

This chapter contains an evaluation of several aspects of the project. The evaluation discusses the work process the group went through in this project. This includes all work connected to development of the system, how the development methodology and routines worked and how the group experienced the work load of this project. Next follows a description of the state of the project at delivery and a discussion of why it was in this state. The chapter ends with a conclusion of how the group has experienced the project, and how satisfied the group has been with the learning process for this task, and with the way the project was organized by the coordinators. Section 15.1 evaluated the work process of the project. This includes the development process, work routines and an evaluation of the work load. In Section 15.2 the final product is reviewed. This includes a discussion of whether or not the product is finished, and why it is delivered as it is. At last, Section 15.4 gives a conclusion on the evaluation.

15.1 Work Process

During the project there have been several aspects in the work process that is worth a mention. Even though all of us have prior experience in team work and development, we had to learn to work together as a team. As a result, the project could have been done differently.

15.1.1 Development Methodology

During the beginning of the project, we were very eager to use Scrum as a method of development. The reason behind this may be the fact that most IT-companies tend to speak very highly of Scrum and how well it works. Many of the team members had never tried out Scrum before, and were therefore interested in learning how this was done.

During the project we understood early on that Scrum was not as optimal as we first assumed. There were many factors resulting in Scrum not being optimal. One of the major artifacts of Scrum is the daily stand up. This was very difficult to carry out, since each team member had their own lecture plan, and it was therefore difficult to find suitable times for a scrum meeting, even though it only lasts for 10 minutes. We tried to follow the plan of doing semi-daily stand ups as strictly as possible, but many times we used AgileZen or IM clients to update each other.

The use of a scrumboard is yet another important artifact of Scrum. Since it was not possible to find a permanent workroom, we used an online scrumboard via AgileZen, in addition to a spreadsheet in Google Drive. This worked fairly well, but it would have been better to have a physical scrumboard, since it would remind and motivate the team in a higher degree.

The customer was very involved in the process, regarding functionality, prioritizing tasks and giving feedback on results. This helped us focus on what was important to finish. We decided early on a 14 days length. The main reason for this was the fear of wasting too much time on planning, reporting and retrospective if the sprints were only a week long. Sprints of this length lead to us having to take feedback

and design for change in the middle of a sprint, since we met with the customer each week. This was done in an orderly fashion, by us not planning too much in detail for what functionality was to be implemented in each section, but rather what section was up for improvement during each sprint. We were in a way more agile than a usual Scrum project would be. Being more agile turned out for the better, since we quickly eliminated unnecessary tasks, though it was stressful at times, since the customers came with new demands during the sprints.

One aspect we failed at was having a working application at the end of each sprint. At the end of each sprint there was always some parts of the applications that did not work properly, and instead of removing them for the demo, we didn't show these parts at the demo. This worked out OK, but is not the proper way to do things, since the customer may be confused during demos, and ask about nonfunctional parts of the system. Usually in Scrum either functionality is complete, or it's not in the demo application at all. Meaning it's not possible to find it when searching through a demo application.

15.1.2 Development Process

The development process was a dominating part of this project. We used about 50% of our time on programming, which is normal. The reason for this amount spent on programming have many factors. The main reason was that we wanted to spend as much time as possible developing a working prototype, and not delivering a half-finished product. At no point did we rationate the hours available, we rather kept track of what tasks were to be finished, and spent our time thereafter.

The customer was very present during the entire process, and gave much feedback on the functionality implemented, changes in requirements and priorities. The requirements were changed after every customer meeting. This resulted in some functionality being removed, even though it was already implemented. Throwing out some functionality is normal, since opinions may change when seeing the final product, or problems may occur during development.

15.1.3 Work Load

The work load of this project has been intense, but it has resulted in a huge amount of learning and experiences in software development. The focus of this project had two main goals, do the work and document the work done. Since the stakeholder for each of the goals are two separate groups, the workload is not necessarily perfectly balanced. The customer wanted us to implement more functionality, while the our advisor told us to write a better report. As stated in the course description, each team member should use 25 hours a week, to a total of 325 hours on the project. This was a constant struggle, as all team members had at least two other courses to attend, along with exercises to finish in these courses. This was a constant stress factor throughout the project.

The exercises of other courses were at times very time-consuming, effecting our effort on the project. When another course had an exercise up for delivery, we had problems filling the hours demanded for that certain week. This resulted in a very uneven work effort from each of us. A more even work flow would have been more preferable and much crunch-time would have been avoided.

Since all team members had different lecture plans it was hard to coordinate when to work together. Working together is an absolute necessity when programming, and this could have been planned better by the lecturers.

The development lasted until there were ten days until delivery of the report. For the last ten days the focus was directed towards writing the report and documenting the code.

The final source code consisted of almost 11 000 lines of code, which is a lot considering the time and resources we had available.

15.2 The Final Product

The goal of this project was to deliver a fully functional prototype, which later could be used in order to launch a full-scale product. Unfortunately the final product did not include all the functionality wanted from the customer. Yet, we are of the opinion that this is a very functional and well working prototype, and we are very curious as to what this prototype will lead to in the future.

Karotz implementation

The implementation of the Karotz in the project was a good idea, on paper, and that's about it. The children we tested on found the Karotz funny and nice, and by moving the focus away from the medicine and towards the Karotz, it made the mask the children use for taking their medicine seem less frightening.

To work with the Karotz was easier said than done. The documentation of the Karotz API is written in French. A huge problem with the Karotz was connecting it to the internet. In order to make it run the program we wrote for it, we had two possibilities, either deploy to the Karotz website or run the program locally. Deploying the application to the Karotz website was not an option since it would have to wait for approval, and there was no time estimate for how long it would take. When running it locally the Karotz has to be connected to the same network as the computer running the program. Also there is not many possibilities for storing any information on the Karotz and the documentation given by the Karotz documentation was faulty and did not explain how to store programs on the Karotz. This resulted in a solution where the program had to be downloaded for each repeated run. Another huge problem is that the Karotz will update itself every 30 minutes, meaning that the program running on the Karotz will be deleted and will not start itself again, making the use of the Karotz a high-maintenance task.

We also had some problem with the developer website with the documentation of the API. The website was unavailable for a week, during our project. When we reached out to their support desk, they had no time estimates for when it would be back up, leaving us out in the cold.

All in all the implementation of a robot toy is a good idea, since it may appeal to children and make it more enjoyable for them to take their medicine. Using Karotz for this task is not a good idea, and should be avoided. Unfortunately there are very few alternatives as to this. There is reason to believe that the cost-benefit ratio for an end user will be so low that they would have little benefit from using a Karotz.

15.3 Functional Requirements completed

Table 15.3 shows an overview of the functional requirements stated in Section 5.2, and whether they are completed or not. As one may notice, all functionality with high priority is completed. The parts that are not completed can be classified as “nice to have”-features, but are not vital for the prototype.

Functional Requirement	Priority	Completed	Comments
PFR 1 - Medication plan	High	Completed	Supports simple medication plans. Does not support several medicines that is to be taken at the same time. One minute delay is a potential workaround.
PFR 2 - Notifications	High	Completed	An alarm is goes off once it is time to take medicine. The email-notification was not implemented.
PFR 2.1 - Settings for notifications	Medium	Not completed	Due to short time, and because we needed access to file system to find ringtones.
PFR 2.2 - Notification to change conditions	Medium	Not completed	Did not have time, although there is not a whole lot of work to extend the application with this functionality.
PFR 3 - Families	Low	Not completed	-
PFR 4 - Guidelines	High	Completed	-
PFR 4.1 - Guidelines from NAAF	Medium	Not completed	-
PFR 5 - Keep records of condition	High	Completed	-
PFR 6 - Pollen forecast	Medium	Somewhat completed	NAAAF's pollen cast is not running at the moment. We replicated the XML-structure for the purpose of the prototype.
PFR 7 - Screen sizes	Low	Not completed	Only supports screen sizes at 480x800 at the time being. Easily extended, but needs scaling of pictures to fit the screen.
CFR 1 - Distraction	High	Completed	-
CFR 2 - Rewards	High	Completed	-
CFR 2.1 - Rewards	Low	Somewhat completed	-
CFR 3 - Screen sizes	Low	Not completed	Refer PFR 7.
CFR 4 - Avatar	Low	Not completed	The customer had a hard time settling on our gamification concept. In the end there was not enough time. In cooperation with the customer, we decided to scrap the idea.
CFR 5 - Child friendly instructions	High	Completed	Image gallery that shows very simple drawings of how to take a medicine.
KFR 1 - Notification	High	Completed	-
KFR 2 - Distraction	High	Completed	Children needs to interact with the karotz when taking a medicine. This helps the user get distracted.
KFR 3 - Reward	High	Completed	-
KFR 4 - Register use of medicine	Medium	Completed	-
KFR 5 - Logging	Medium	Completed	-

Table 15.1: The original functional requirements, whether they are successfully implemented or not, and comments

15.4 Concluding Remarks

The project has been engaging, educational, stressful and challenging. In retrospect, we all agreed that the experiences has been worth the effort and time we have spent. The weight of what we learned in terms of project management, programming, team-work and customer relations and system development has been huge. Even though the project lead to team members having less time for other courses, we are in the opinion that the time spent has been worth it. Regarding the final product we are proud of what we have delivered, and are very curious about the future of the BLOPP project. The domain of the project has made us feel that we have had the possibility to make changes and actually help children with asthma and their parents. If we would have done the project one more time, we would have done some things differently. First, we would have designed CAPP for multiple users from the beginning. We started implementing CAPP without having a plan for how to implement support for multiple children. At a later stage, when support for multiple users was up for discussion, we had to decline the idea, since it would take too much time to refactor all code already written. We should have arranged programming sessions from the beginning. When working as a team, it is essential to be in close proximity, in order to make communication more effective. The shop functionality was left in the cold for too long. The customer was not certain if they wanted the shop, and at one point the abandoned the shop. We are in the opinion that the shop would have been a very cool idea, if done right. If we had decided to make a shop from the beginning we believe it would result in a great element for motivating children.

Bibliography

- [1] *Scrum process.svg*. October 07 2012. Available at: <www.wikipedia.org>
- [2] *Waterfall Model*. October 07 2012. Available at: <compsci.ca>
- [3] *Riktig bruk av inhalatorar*. November 17 2012. Used with permission. Sjukehusapoteka Vest. Available at: <www.sjukehusapoteka-vest.no>
- [4] *Frequent Flyer Programs*. November 10 2012. Gamification Wiki. Available at: <www.gamification.org>
- [5] Zichermann G, Cunningham C (2011). *Gamification By Design*. O'Reilly Media, Inc. ISBN: 978-1-4493-9767-8
- [6] Hamari J, Eranti V (2011). *Framework for Designing and Evaluating Game Achievements*. Proceedings of DiGRA 2011 Conference: Think Design Play.
- [7] *Fakta om astma*. Published 10.01.2006. Available at: <www.naaf.no>
- [8] *Useful facts on pollen allergy (pollenallergi)*. Published 22.04.2007. Available at: <www.naaf.no>
- [9] Jackson S, Joshi A, Erhardt N (2003). *Recent Research on Team and ORganizational Diversity: SWOT Analysis and Implications*. Journal of Management 2003
- [10] Royce W (1970). *Managing the Development of Large Software Systems*. Proceedings IEEE WECSON 26
- [11] Beck K, Beedle M, van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, Kern J, Marick B, Martin C, Mellor S, Schwaber K, Sutherland J, Thomas D (2001). *Agile Manifesto*. Available at: <agilemanifesto.org>
- [12] *Android API Reference: Service class*. November 13 2012. Available at: <developer.android.com>
- [13] *MySQL Marketshare*. September 20 2012. Available at: <www.mysql.com>
- [14] Wu M (2011). *Gamification 101: The Psychology of Motivation*. Lithosphere's Building Community blog. Available at: <lithosphere.lithium.com>
- [15] *Karotz*. October 09 2012. Available at: <www.karotz.com>
- [16] Pelling N (2011). *The (short) prehistory of “gamification”*.... Nick Pelling's personal blog at WordPress. Available at: <nanodome.wordpress.com>
- [17] Daniels M (2011). *Businesses need to get in the game*. Marketing Week. Available at: <www.marketingweek.co.uk>
- [18] *About NodeJS*. October 10 2012. Available at: <nodejs.org>

- [19] Bass L, Clements P, Kazman R (2003). *Software Architecture in Practice, 2nd Edition*
- [20] *About PHP*. October 13 2012. Available at: <www.php.net>
- [21] *javadoc - The Java API Documentation Generator*. November 13 2012. Oracle. Available at <docs.oracle.com>
- [22] Anderson J, Rainie L (2012). *Gamification: Experts expect ‘game layers’ to expand in the future, with positive and negative results*. Pew Research Facility.
- [23] *About Dropbox*. September 12 2012. Available at: <www.dropbox.com>
- [24] *What is AgileZen?* August 30 2012. Available at: <www.agilezen.com>
- [25] Rød G, Øynar K, Skadberg B (2006, rev. 2009). *Astma Bronkiale*. Helsebiblioteket.no. Available at: <bit.ly/T1f2Dl>
- [26] *Conundra Ltd*. October 20 2012. Available at: <www.nanodome.com/conundra.co.uk>
- [27] *MySQL*. November 03 2012. Available at: <www.mysql.com/why-mysql>
- [28] *IH25*. November 12 2012. Used with permission. Available at: <www.beuler.com>
- [29] *Design Principles*. October 02 2012. Available at: <developer.android.com>
- [30] *Om lov om forbud mot diskriminering på grunn av nedsatt funksjonsevne (diskriminerings- og tilgjengelighetsloven)*. November 09 2012. Available at: <bit.ly/UzZloH>
- [31] *About JSON*. November 05 2012. Available at: <www.json.org>
- [32] *Balsamiq Mockups*. September 07 2012. Available at: <www.balsamiq.com>
- [33] *Personal information and data protection*. November 13 2012. Available at: <www.datatilsynet.no>
- [34] *POSTMan*. November 06 2012. Available at: <github.com/a85/POSTMan-Chrome-Extension>
- [35] Krutchen P, 1995. *Architectural Blueprints—The “4+1” View Model of Software Architecture*. IEEE Software
- [36] Gao C, 2011. *CalendarView*. Available at: <code.google.com/p/android-calendar-view>
- [37] *MySQL on IDI*. November 06 2012. Available at: <drift.idi.ntnu.no>
- [38] Borge C, Engh N, Austegard E, Sandsund C, Hamre H, Slettedal I, Alam H, Erikstad T (2002). *How to live with asthma*. Norges astma- og allergiforbund.
- [39] Anderson D, 2003. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Prentice Hall.
- [40] *About Joda Time*. November 10 2012. Available at: <joda-time.sourceforge.net>
- [41] *About Android SDK*. November 14 2012. Available at: <developer.android.com>
- [42] *About Android Developer Tools*. November 14 2012. Available at: <developer.android.com>
- [43] *About Eclipse IDE*. November 14 2012. Available at: <www.eclipse.org>
- [44] *TIOBE Programming Community Index for November 2012*. November 15 2012. Available at: <www.tiobe.com>
- [45] *Git source code management* Available at: <git-scm.com>

Appendices

Appendix A

Paper Prototype

A.1 About paper prototyping

This section describes the process of paper prototyping. A paper prototype is a low-fidelity prototype made out of paper, post-it notes or similar material. The idea is making a layout in paper, to test the flow of the program, the layout and design to root out early design flaws. By making this out of paper, a low-cost prototype is ensured. The prototype is then tested by an external test person, using different predetermined scenarios.

Usability testing using paper prototypes does have it's limitations. The paper prototype makes it difficult to simulate animations, sounds, scrolling and similar functionalities. The test person must also be able to imagine that the paper prototype is a real program, even though it's made out of paper.

The usability testing done using the paper prototype is explained in Section 13.3.1.

A.2 Usability Testing with a paper prototype

A.2.1 Testprocedures

The usability test is done by making the testperson completing a series of tasks with the help of the paper prototype. The tasks must be very specific, meaning they must be specified in a way which makes the testperson search for specific information, press specific elements on screen or similar. The task shall be realistic and representative for the normal use of the system.

An example of such a task may be: "You wish to change the health state of Ole Olesen from Good to Bad. Please do this via the application".

A.2.2 The testpersons tasks

The testperson shall solve the tasks given, while he/she speaks out loud what he/she is doing and why. The reason for this is allowing the designers to understand the thoughts and the mindset of the testperson's user experience with the prototype.

A.2.3 The testgroups tasks

The team leading the test shall have clearly defined tasks under the test. Testleader gives instructions to the person doing the test and tells what is happening.

"The Wizard of Oz" is responsible for changing out the "frames" (papers representing frames) when the user interacts with the paper prototype.

Observators don't take part in the testing, but observe and take notes throughout the testing.

Neither the testleader, the Wizard of Oz or the observators may answer questions during the testing.

Appendix B

Document templates

B.1 Agenda

Advisor meeting
DD/MM/YYYY

1. Approval of agenda
2. Approval of minutes of meeting from last advisor meeting
3. Comments to the minutes from last customer meeting or other meetings
4. Approval of the status report
5. Review/approval of attached phase documents
6. Sprint X
7. Other issues

B.2 Status reports

Status reports are internal reports that will be sent to the advisor each week. The content and a short description of this report is found in Table B.1

Content item	Description
Status Report - Week WEEKNUMBER	Headline
Work done	Description of where the focus have been the last week. What is the status of the applications
Problems	Description of different problems that has arised during the week. Either internal group problems or problems that occurs due to technical issues.
Next week	Description of activities like meetings and usability testing that will occur next week. Place and time will also appear here.

Table B.1: Content and short description of Status reports

Appendix C

Karotz Manuscript

Table C.1 gives an overview of action sequences the Karotz takes based on a given medicine combination that is supposed to be taken at the same time.

b means blue medicine,

o means orange medicine,

p means purple medicine,

the number represents how many doses should be taken.

Medicine combinaton	1b	1o	1p	2b	2o	2p	1b + 1o
Sequence code	1s						
	2s						
	1b	1o	1p	1b	1o	1p	3s
	2b	2o	2p	2b	2o	2p	1b
	3b1	3o1	3p1	3b2	3o2	3p2	2b
	4s	4s	4s	4s	4s	4s	3b1
	6s	6s	6s	5s	5s	5s	4s
	9s	9s	9s	4s	4s	4s	8s
	7s1	7s1	7s1	6s	6s	6s	1o
				9s	9s	9s	2o
				7s2	7s2	7s2	3o1
							4s
							6s
							9s
							7s2

Table C.1: Manuscript actions for Karotz

Table C.2 gives detailed information on what each code means, with a dialogue statement that the rabbit says and an activator that makes the application go to the next item.

Code	Dialogue	Activator
1s	Hei! Jeg heter BLIPP! Nå er det tid for å ta pustemedisin! Trykk på hodet mitt så forteller jeg deg mer. <i>Hi! My name is BLIPP! Now it's time to take breathing medication! Press my head and I will tell you more.</i>	Button

2s	Hent en voksen som kan se på, og hold den lille gule kaninen rett under nesen min. <i>Fetch an adult that can watch, and hold the small yellow rabbit directly beneath my nose.</i>	Yellow nanoz
3s	Nå skal du ta to forskjellige medisiner. <i>Now you will take two different medicines.</i>	Timeout: 2 seconds
4s	Når jeg sier ifra skal du trykke på sprayen, og du skal puste rolig mens jeg teller til 10. Klar, ferdig, trykk! 1-2-3-4-5-6-7-8-9-10 <i>When I tell you to, you should press the spray and breathe calmly while I count to 10. Ready, set, push! 1-2-3-4-5-6-7-8-9-10</i>	Timeout: 2 seconds
5s	Flott! Bra jobba! Du har tatt den første dosen. Gjør klar for dose nummer to. Sett på deg masken og gjør deg klar til å gjøre det samme en gang til. Trykk på hodet mitt når du er klar. <i>Excellent! Great job! You have taken the first dose. Get ready for dose number two. Put on the mask and get ready to do the same one more time. Push my head when you are ready.</i>	Button
6s	Nå var du flink! <i>You were good now!</i>	Timeout: 2 seconds
7s1	Som belønning får du 1 stjerne til skattekista di. <i>As reward you will get 1 star for your treasure chest.</i>	DONE
7s2	Som belønning får du 2 stjerner til skattekista di. <i>As reward you will get 2 stars for your treasure chest.</i>	DONE
8s	Da kan du hente den andre medisinen du skal ta. <i>Now you can fetch the second medicine you should take.</i>	Timeout: 2 seconds
1b	Hent den blå medisinen og masken du puster i, og trykk på hodet mitt når du har hentet dem. <i>Fetch the blue medication and the mask you breathe into, and push my head when you have fetched them.</i>	Button
2b	Rist den blå medisinen! (ristelyd) Trykk på hodet når du er klar. <i>Shake the blue medicine! (shaking sound) Push my head when you are ready.</i>	Button
3b1	Av den blå medisinen skal du ta 1 puff. Sett på deg masken og gjør deg klar. Trykk på hodet mitt så teller jeg mens du puster inn og ut. <i>You should take one puff of the blue medicine. Put on the mask and get ready. Push my head and I will count while you breathe in and out.</i>	Button
3b2	Den blå medisinen skal du ta to ganger etter hverandre. Vi begynner med 1 puff. Sett på deg masken og gjør deg klar. Trykk på hodet mitt så teller jeg mens du puster inn og ut. <i>The blue medicine you should take twice after each other. We start with 1 puff. Put on the mask and get ready. Push my head and I will count while you breathe in and out.</i>	Button
1o	Hent den oransje medisinen og masken du puster i, og trykk på hodet mitt når du har hentet dem. <i>Fetch the orange medication and the mask you breathe into, and push my head when you have fetched them.</i>	Button

2o	Rist den oransje medisinen! (ristelyd) Trykk på hodet når du er klar. <i>Shake the orange medicine! (shaking sound) Push my head when you are ready.</i>	Button
3o1	Av den oransje medisinen skal du ta 1 puff. Sett på deg masken og gjør deg klar. Trykk på hodet mitt så teller jeg mens du puster inn og ut. <i>You should take one puff of the orange medicine. Put on the mask and get ready. Push my head and I will count while you breathe in and out.</i>	Button
3o2	Den oransje medisinen skal du ta to ganger etter hverandre. Vi begynner med 1 puff. Sett på deg masken og gjør deg klar. Trykk på hodet mitt så teller jeg mens du puster inn og ut. <i>The orange medicine you should take twice after each other. We start with 1 puff. Put on the mask and get ready. Push my head and I will count while you breathe in and out.</i>	Button
1p	Hent den lilla medisinen og masken du puster i, og trykk på hodet mitt når du har hentet dem. <i>Fetch the purple medication and the mask you breathe into, and push my head when you have fetched them.</i>	Button
2p	Rist den lilla medisinen! (ristelyd) Trykk på hodet når du er klar. <i>Shake the purple medicine! (shaking sound) Push my head when you are ready.</i>	Button
3p1	Av den lilla medisinen skal du ta 1 puff. Sett på deg masken og gjør deg klar. Trykk på hodet mitt så teller jeg mens du puster inn og ut. <i>You should take one puff of the purple medicine. Put on the mask and get ready. Push my head and I will count while you breathe in and out.</i>	Button
3p2	Den lilla medisinen skal du ta to ganger etter hverandre. Vi begynner med 1 puff. Sett på deg masken og gjør deg klar. Trykk på hodet mitt så teller jeg mens du puster inn og ut. <i>The purple medicine you should take twice after each other. We start with 1 puff. Put on the mask and get ready. Push my head and I will count while you breathe in and out.</i>	Button
9s	Nå kan du holde den lille grønne kaninen rett under nesen min for å få premien din! <i>Now you can hold up the small green rabbit right beneath my nose to get your prize!</i>	Green nanoz

Table C.2: Manuscript for the Karotz

Appendix D

Coding Templates

D.1 Coding Style

The applications uses Android as platform. As a consequence, most code will be written in Java. The coding style mentioned here only applies to code written in Java.

D.1.1 Package conventions

The package names for GAPP will be on the format “no.blopp.app.X”, where X (lower case) describes the content of the package. The package names for CAPP will be on the format “no.blopp.app.med.Y”, where Y(lower case) describes the content of the package. For instance no.blopp.app.activities, or no.blopp.app.med.activities.

D.1.2 Indentation

All statements, conditional expressions, function declarations and class declarations shall be written on a separate line.

D.1.3 Curly Brackets

The opening curly bracket following a function or class declaration shall appear on the line below the declaration, with equal indentation as the function declaration. The closing bracket shall also appear with this indentation, on the line below the code block.

D.1.4 Naming Conventions

The following naming conventions will be used when writing code in Java. We will use an “I” in front of an interfacename, to better recognize these. That is the only place we have that kind of convention. Table D.1 shows an overview of the naming conventions for Java code.

Type	Convention
Local variable	lowerCamelCase
Class	UpperCamelCase
Interface	<i>I</i> UpperCamelCase
Constants	UPPERCASE

Table D.1: Naming convention

D.1.5 Android views

The android framework has a lot of predefined elements like buttons, textviews and layouts. When creating these elements, they have an id referenced by “android:id= my_id”. These id’s will be a combination of type a_b. “a” will be a constructive word describing the element. “b” will be the component. “a” will be written in lowercase only, separated with underscore. “b” will be in lowerCamelCase. Examples of this can be “back_to_menu_button”, “date_textView” and so on. The reason for this is that android automatically generates R.java, containing these id’s. It should be easy to know exactly which id you are looking for when calling the method findViewById(id).

D.1.6 Code Examples

Figure D.1 shows the coding style we used for classes in Java. As explained the curly braces are beneath the class declarations and method, variables are in lower camel case, while constants are all upper case and constructors are upper camel case.

Figure D.2 shows the coding style for a Java interface. As explained the names of interfaces are prepended with a capital *I*.

```
package Template;

public class TempClass extends SuperClass implements TempInterface
{
    private int tempInt;
    public static final int TEMP_CONSTANT = 5;
    public TempClass tempClass;

    /**
     * Insert Javadoc here
     */
    public TempClass()
    {
        superstring = "Super!";
    }

    @Override
    public void myMethod(int param1, String param2)
    {
        // TODO Auto-generated method stub
    }
}
```

Figure D.1: Java Classes

```
public interface ITempInterface
{
    void myMethod(int param1, String param2);
}
```

Figure D.2: Java Interfaces

D.1.7 LaTeX folder structure

The report will appear as a separate project. In order to keep track of what is what, and in order to not having too many Git-conflicts, we will have a separate folder for each chapter. This folder should be the name of the chapter, without spaces. If we are writing long sections, these should appear in a separate document, with section name as the filename.

Appendix E

Class diagram

The purpose of these class diagrams is to give developers an overview of which classes that is implemented, and the dependencies between the different classes.

Since the class diagram for the three applications are way to big to display on one A4-page, we felt it necessary to split them into several images. While the logical view shows dependencies across packages, the class diagrams shows the internal dependencies between classes. Please refer the logical view (Figure 6.1) for package dependencies.

In the next diagrams the following notation is being used:

- A solid arrow from class A to class B represents that A uses B.
- A dashed arrow from class A to class B represents that A depends on B.
- A solid arrow with an arrowhead from class B to class A represents inheritance, that is, B inherits from A.
- A dashed arrow with an arrowhead from class B to interface I represents that B implements I.

E.1 Class Diagram GAPP

GAPP - Activities Figure E.1 shows the class diagram for the package “activities”. This diagram shows the different activities. An activity is, simplified explained, a screen on the application. In section 6.2.3, the interaction between different the activities is shown. Since all Activities implements one or more of the interfaces displayed, it would be completely unreadable to show which activities implement which interfaces. The interfaces displayed are standard components in the Android framework, and contains functionality for handling input. The activities get the data that is to be displayed from the packages models, jsonmodels and adapters.

Figure E.2 shows the different paths to an activity from a user’s perspective.

GAPP - JSON parsers Figure E.3 shows the class diagram for the package “jsonparsers”. This diagram shows the different jsonparser available in the system. In order to make a call to a server in Android, the parsers needs to extend `AsyncTask`, which makes a separate thread in order to handle the HTTP-calls. The operating system of a device will simply refuse to do network operations on the main thread. What we have made, is an abstract generic parser, `GenericJSONParser`, which extends `AsyncTask` and implements the interface `IInitializeFromJSON`.

When the application needs data from the webservice, `doInBackground()` is called, which executes the request. Once we get a response, `initializeDateFromJSON()` is called with the result from our http call, in the appropriate class. Each of the subclasses to `GenericJSONParser` contains its own implementation of this method.

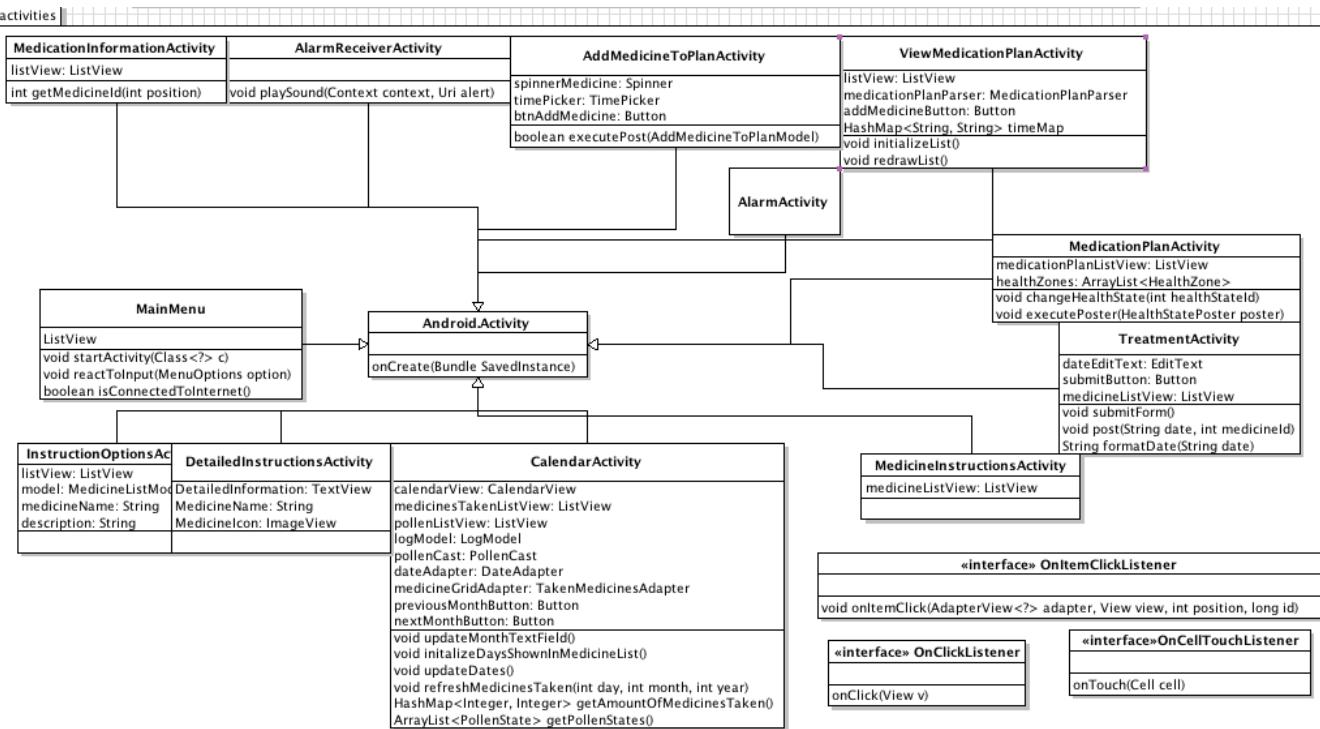


Figure E.1: Activities in GAPP



Figure E.2: Activity interaction GAPP

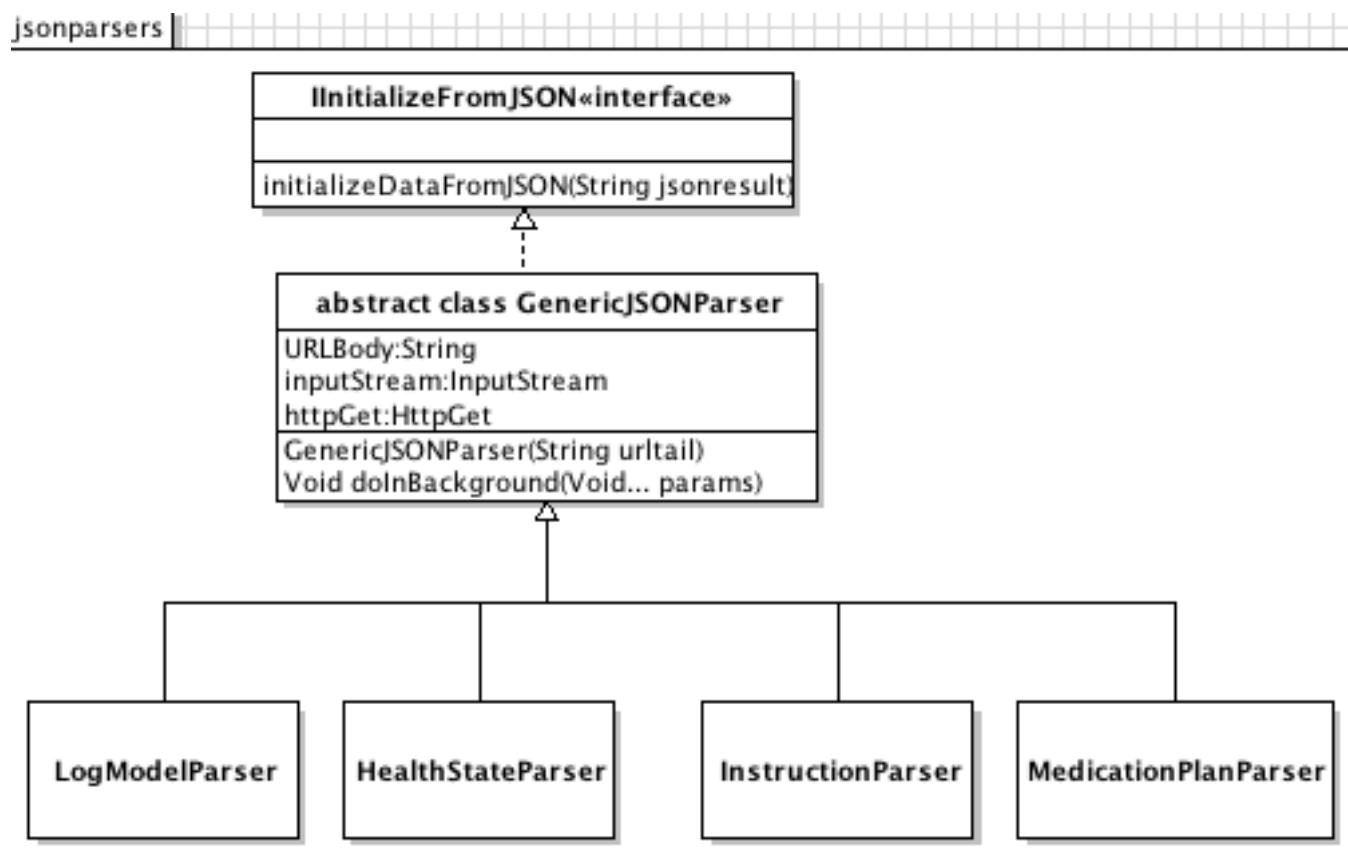


Figure E.3: JSON parsers in GAPP

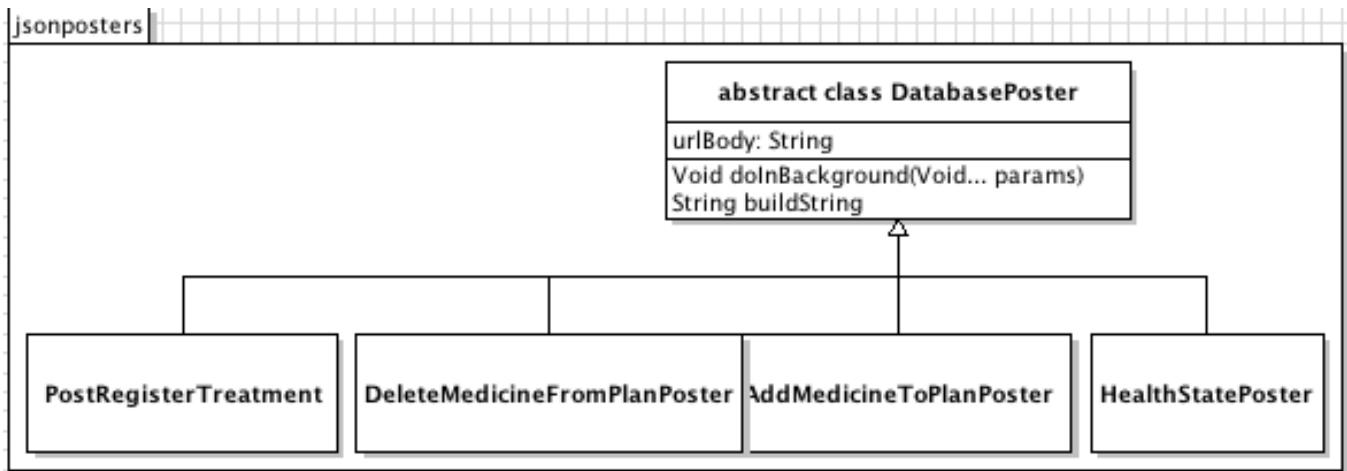


Figure E.4: JSON posters in GAPP

GAPP - JSON posters Figure E.4 shows the class diagram for the package “jsonposters”. This diagram shows the different posters we have made. These classes are responsible for posting information to the database. As with the parsers, the posters also need to extend `AsyncTask`. The class `DatabasePoster` also implements `IInitializeFromJSON`. The reason behind this is that once we have made an HTTP-POST, we get a result on JSON-format. The only useful information here is whether the post was a success or not. There are four subclasses of `DatabasePoster`. These classes represent CREATE or DELETE methods towards the database.

GAPP - jsonmodels The data models we are using can be separated in two categories; “jsonmodels” and “models”. `Jsonmodels` contains classes used to hold information retrieved from our webservice. `Models` contains classes used to hold information that is independent of which child the application displays information about.

Figure E.5 shows the class diagram for the package “jsonmodels”. The different models contain information from database needed in order to display it correctly to the user. This package is used by different activities in order to display data from the database to users. `Jsonmodels` can be split into two categories; Post models and result models. The postmodels contains an important `toString()`-method in order to encode the POST parameters correctly. These models are used by the `jsonposter`-package.

GAPP - models Figure E.6 shows the class diagram for the package “models”. The model package holds three important classes; `LogModel`, `MedicinePlanModel` and `PollenStateAtDayModel`. `LogModel` executes the `LogModelParser`, and is used by `CalendarActivity`. It contains the amount of each medicine that is taken on a day, and the health state of the child. `MedicinePlanModel` contains the different medicine plans, and `PollenStateAtDayModel` contains information about the pollen distribution at a given day. Note that the latter needs some configurations in order to keep up with “pollenvarslingen.no”, since the application at the time being uses dummy data. The pollen cast from NAAF only contains data about today and tomorrow, so the values collected from this service needs to be stored in some manner.

GAPP - adapters Figure E.7 shows the class diagram for the package “adapters”. This package contains some adapters used in order to render `ListViews` and `GridViews` correctly.

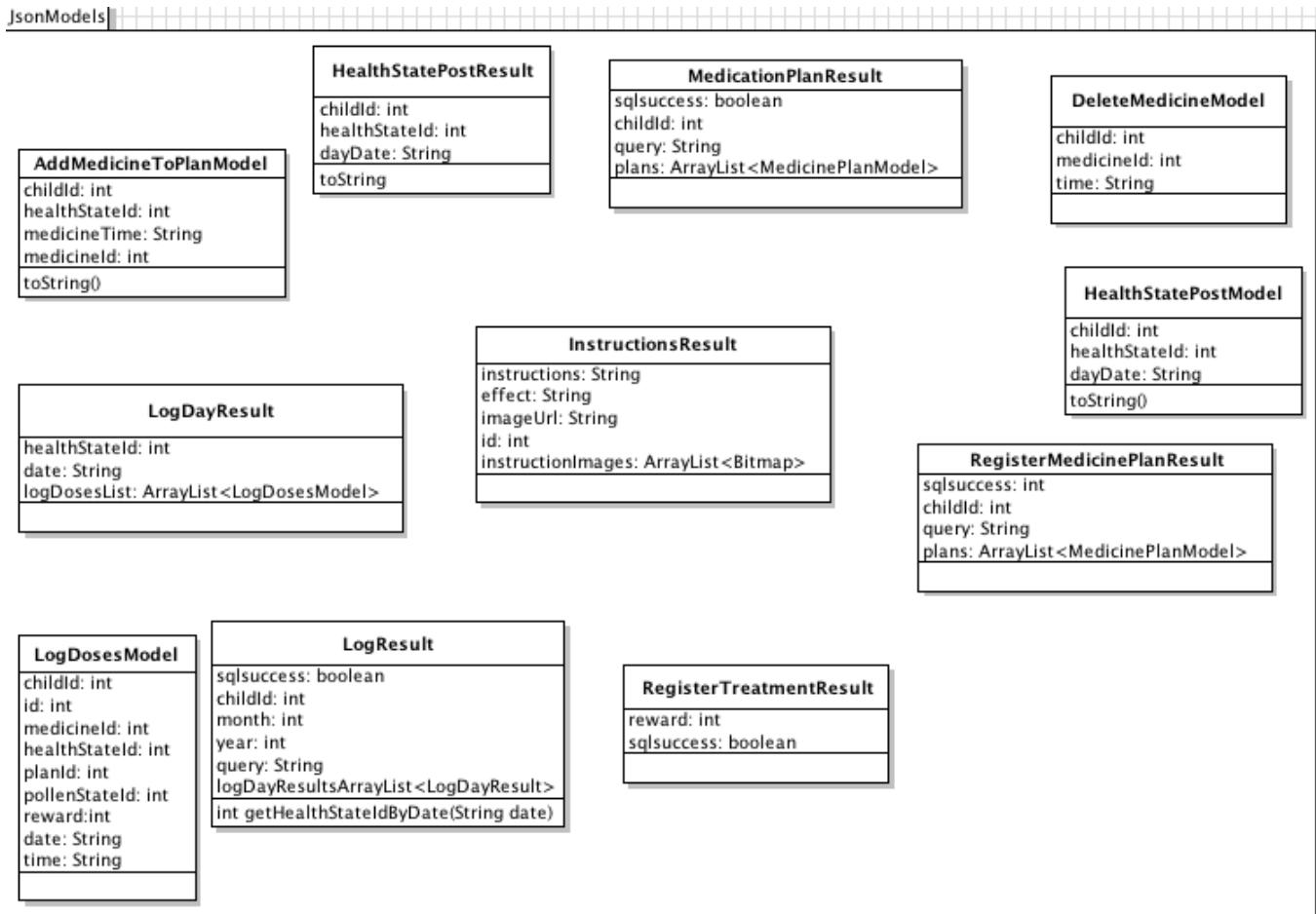


Figure E.5: JSON models in GAPP

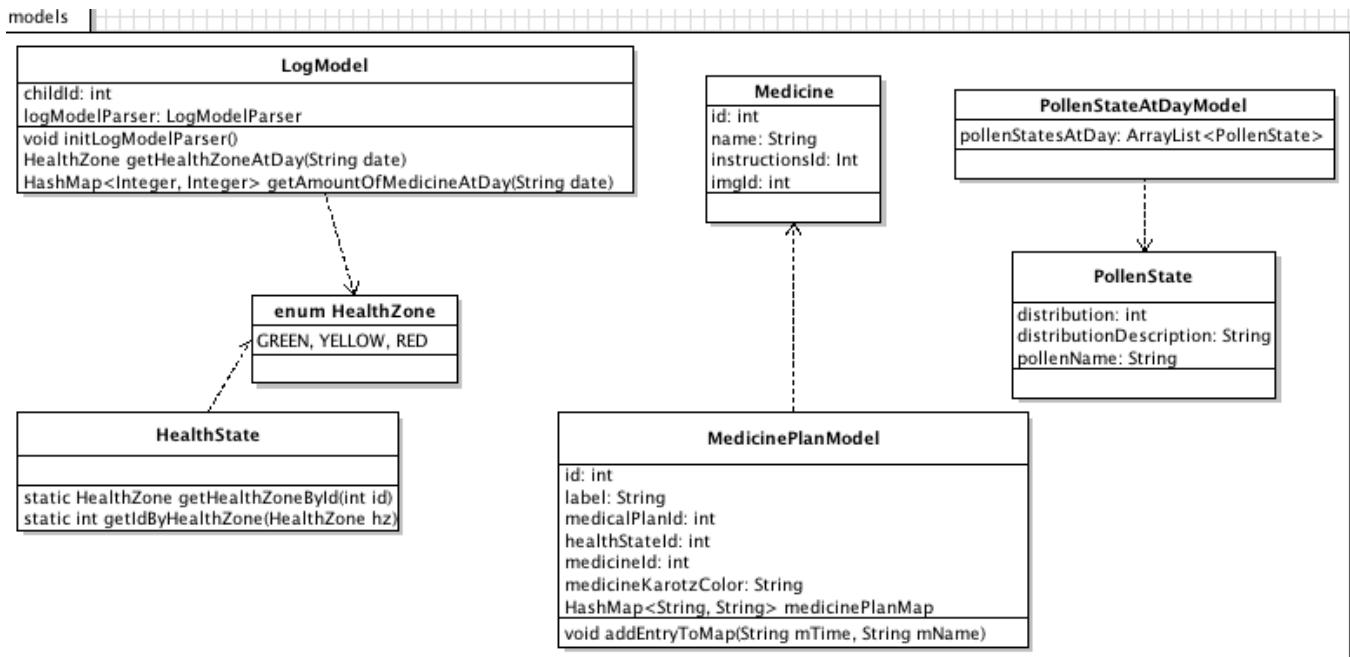


Figure E.6: Models in GAPP

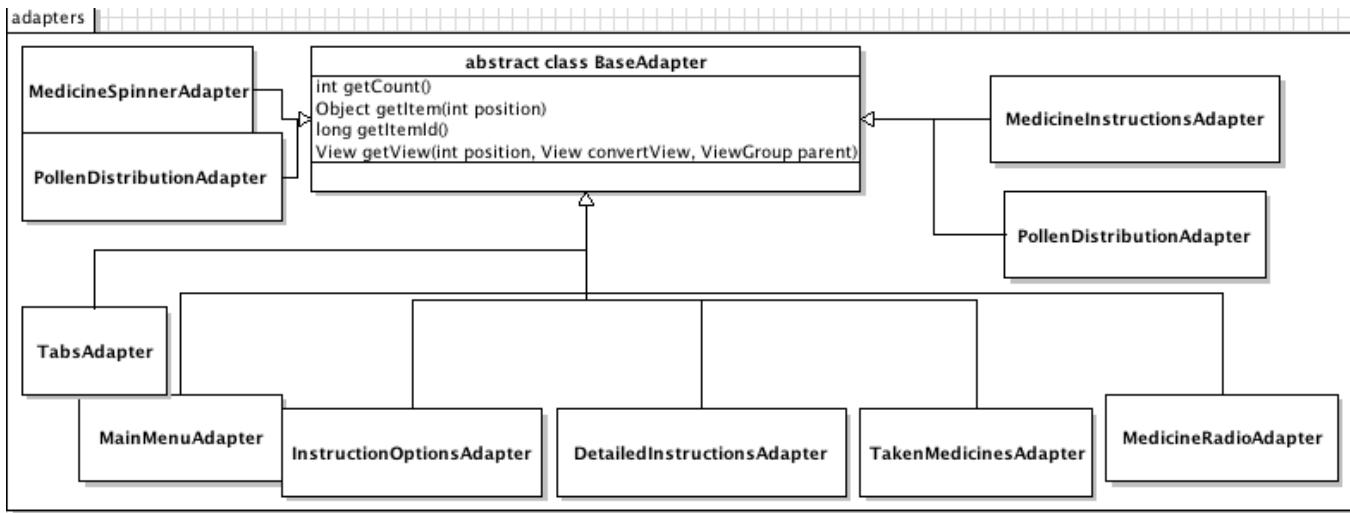


Figure E.7: Adapters in GAPP

GAPP - utils, xmlfeed and views Figure E.8 shows the class diagram for the packages “utils”, “xmlfeed” and “views”. Utils is our “misc”-package, containing classes we did not know where to put. The **DateAdapter** has only one purpose. Convert three integers, day, month and year, to an acceptable MySQL-format. The view package contains our only custom view, **CalendarView**. This view is opensource and is written by Chris Gao (2011)[36], and is configured by the team to serve our purpose. **CalendarView** and **Cell** class is the only views that are programmed. All other views visible to the user is build upon standard Android components like Buttons, ListViews, and so on. These files is on XML-format, and are used by the activities to set the content view (the screen image visible to the user). These layout-files are not included in the architecture, because it won’t actually provide any sorts of useful information to the reader.

E.2 CAPP - Children Application

The children application also contains many classes to display in one image, so we have split this diagram into several images as well.

CAPP - Activities Figure E.9 shows the activities in CAPP. There are four visible activities in CAPP. **AlarmReceiverActivity** is an activity that updates the stored alarms in the system. **MainMenuActivity** displays the main menu. **InstructionsActivity** contains a slideshow of informative images on how to take medicines correctly. **DisplayRewardsActivity** shows the collected stars to the children. **DistractionActivity** controls the views when a child is taking a medicine.

CAPP - Adapters Figure E.10 shows the adapters in CAPP. CAPP has two adapters. **MedicineListAdapter** renders a list of which medications a child can take. **TabsAdapter** is an adapter fitted for **InstructionsActivity**, and contains the logic behind the slideshow that is displayed.

CAPP - Models Figure E.11 shows the models in CAPP. It contains three models. Due to time restrictions, we were not able to extend the application to cover several children. A model for children must thus be made in future development.

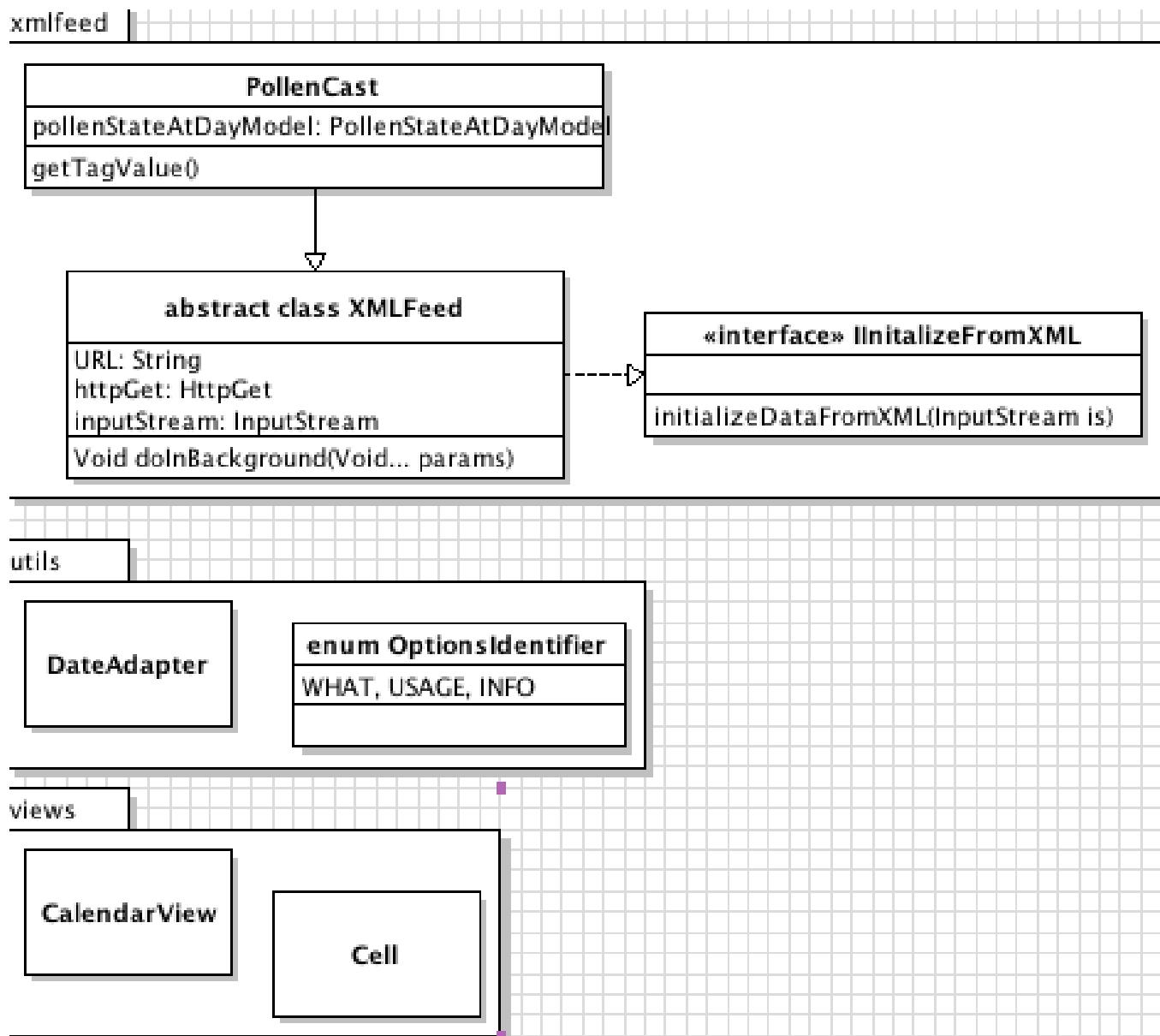


Figure E.8: Other classes in GAPP

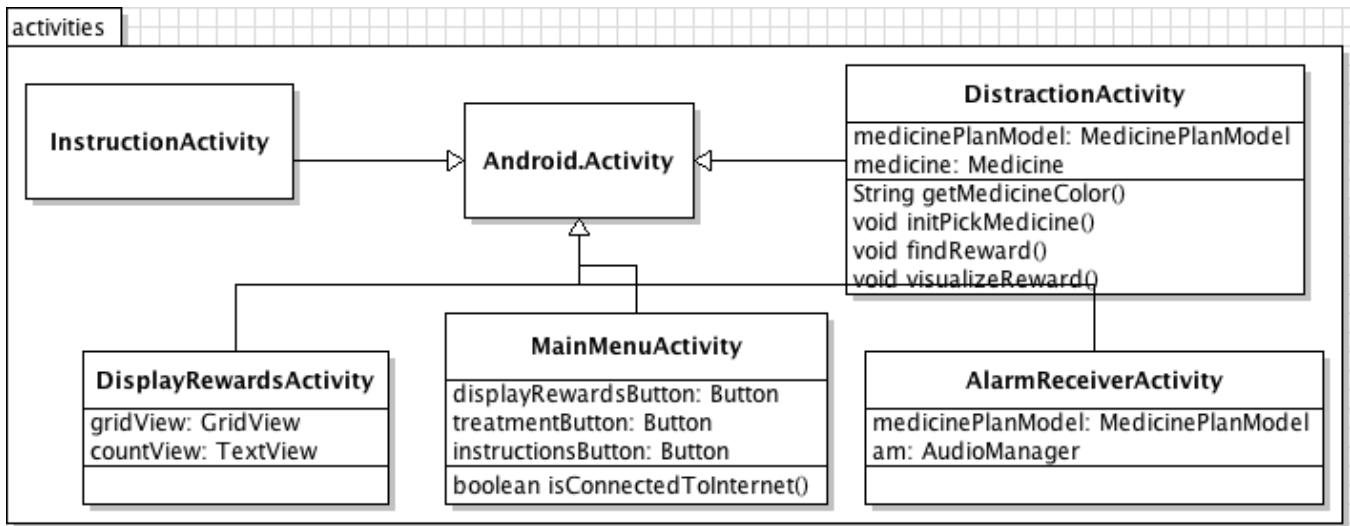


Figure E.9: Activities in CAPP

CAPP - JsonModels Figure E.12 shows the JsonModels in CAPP. `RegisterMedicinePostModel` is used as model for `PostRegisterTreatment` that is shown in figure E.14. The rest of the models reflects the information stored in the database.

CAPP - JsonParsers Figure E.13 shows the JsonParsers in CAPP. It is worth mentioning that `DownloadImageTask` and `InstructionsParser` is not currently used. At the start of the project, the customer wanted easily modifiable instructions for the children. We thought it might be useful to implement classes that could download instructions from the web. However, the customer never created these online instructions. We thought it might be useful for future developers to extend these classes if such functionality is needed one day.

CAPP - JsonPosters Figure E.14 shows the JsonPosters in CAPP. The application does a HTTP POST is when a medication is completed. For consistency among CAPP and GAPP, this package contains similar classes as those found in GAPP.

CAPP - Services Figure E.16 shows the services in CAPP. As defined by Android[12], a `Service` is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use. We have implemented two services, `OnBootReceiver` and `AlarmUpdateReceiver`. `OnBootReceiver` is used to gather alarm information from the database once a device is turned on. `AlarmUpdateReceiver` has similar functionality. It deletes old alarms, and sets the new alarms.

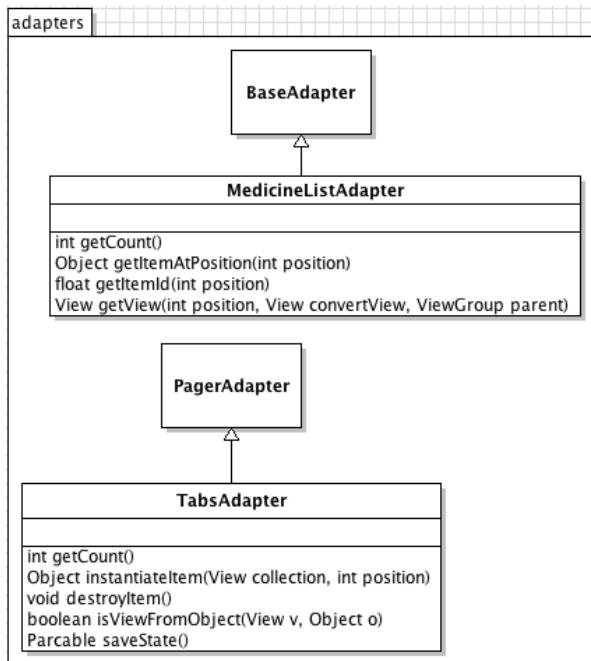


Figure E.10: Adapters in CAPP

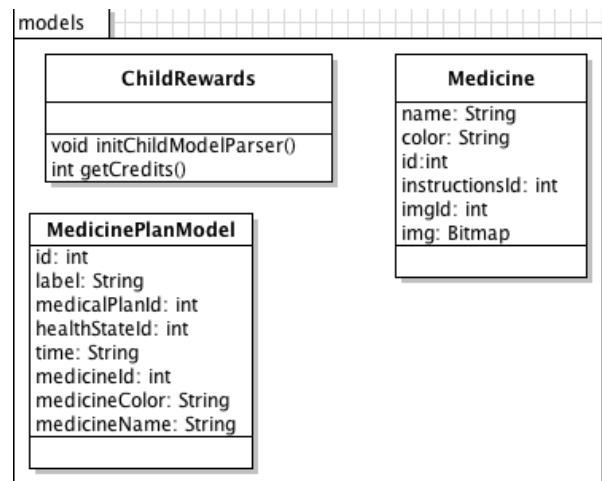


Figure E.11: Available plans in GAPP

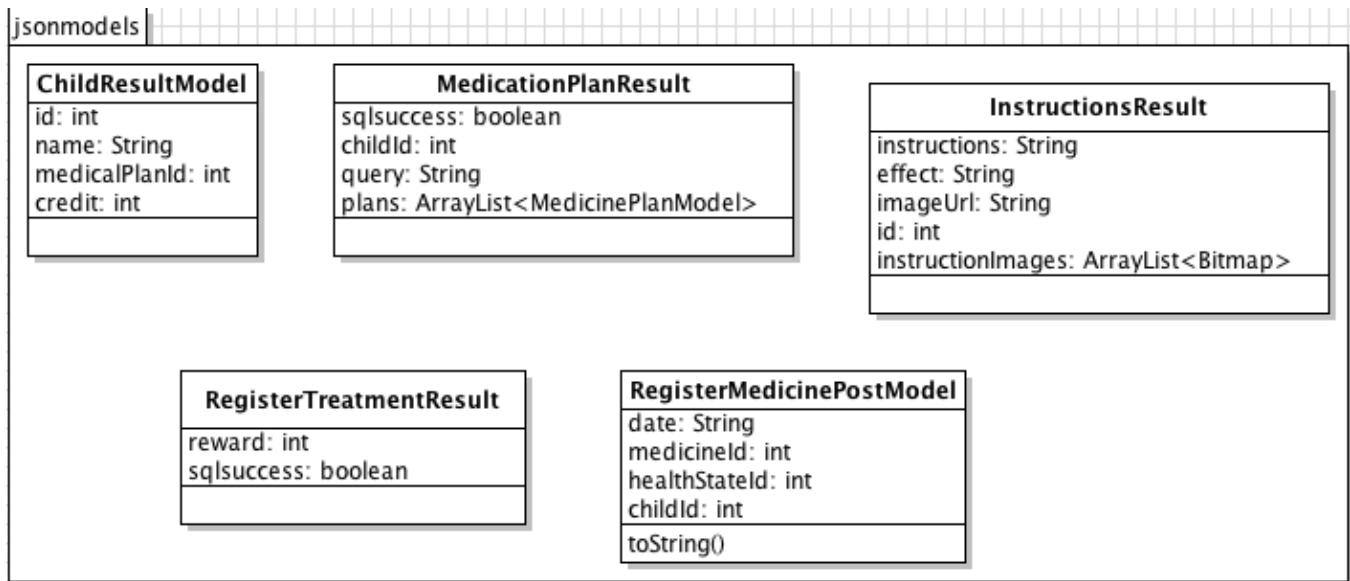


Figure E.12: JsonModels in CAPP

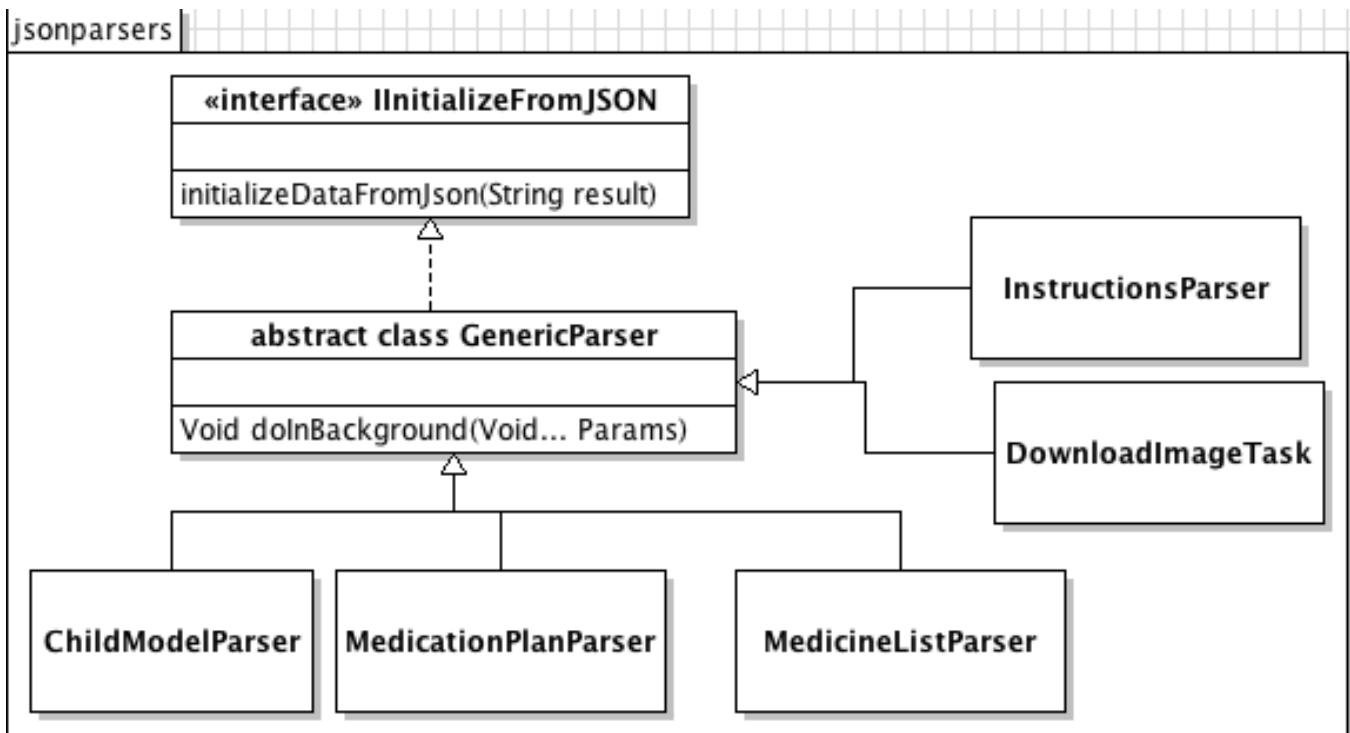


Figure E.13: JsonParsers in CAPP

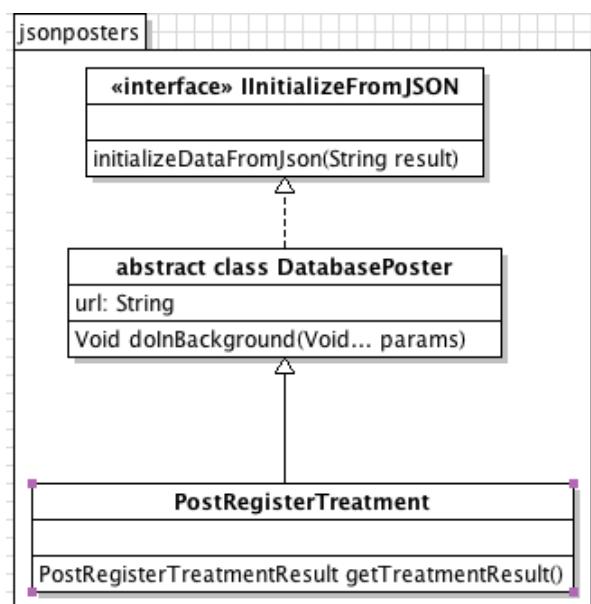


Figure E.14: JsonPosters in CAPP

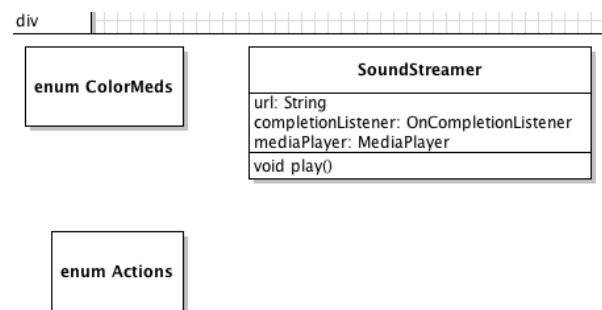


Figure E.15: Misc classes in CAPP

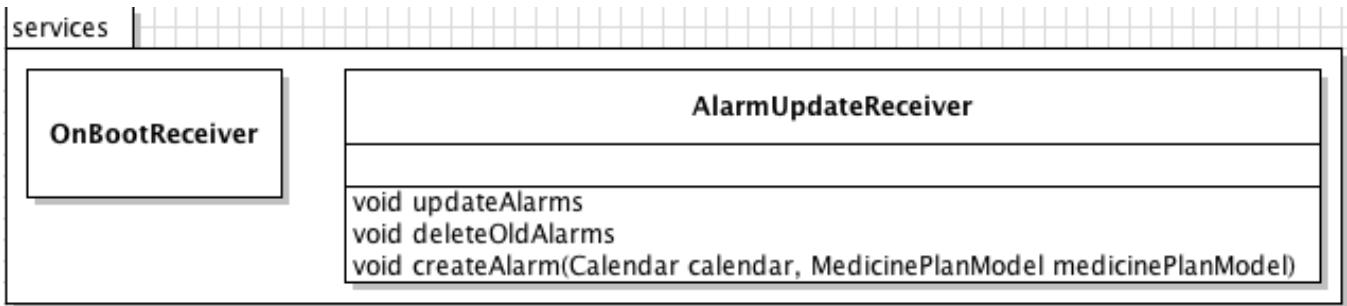


Figure E.16: Services in CAPP

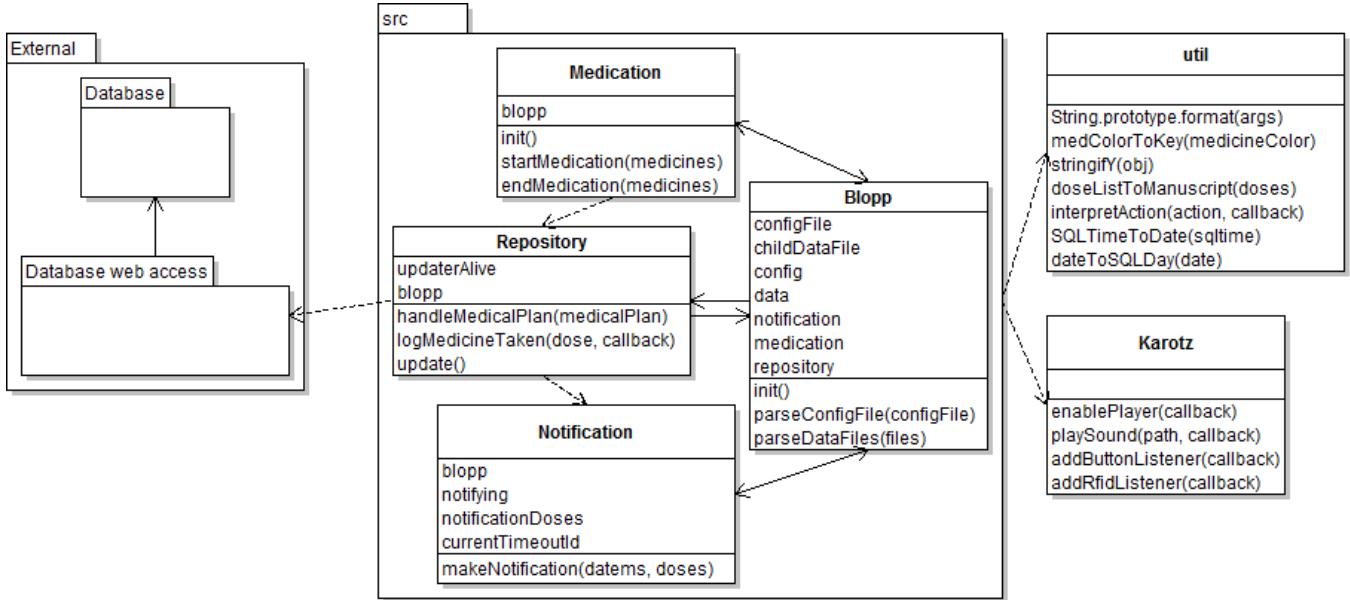


Figure E.17: Class diagram for the Karotz application

E.3 Karotz Application

Figure E.17 shows the class diagram for the Karotz app. There are strong relations between all the modules in the “src” package. The **Blopp** class is instantiated when the application is started and has the responsibility of invoking all the necessary modules. It maintains a **Repository** for connecting to the database. The repository depends on the **Notification** module for calling the method **makeNotification()** which creates a timeout of a given length. The repository ensures that the method is only called for the nearest scheduled medication, and overwrites all the previously set notification timeouts. When a notification event is initiated, the application calls **startMedication()** in the **Medication** module. The distraction process uses the functions **doseListToManuscript()** and **interpretAction()** in the module **util** for creating a list of actions required by the manuscript, and performing them. Action descriptions are stored in an external configuration file, which is loaded in the **Blopp** module in the field **config**. When a medication is finished, **endMedication()** calls **logMedicineTaken()** in **Repository** once for each medicine that was taken.

Appendix F

Article from Adressa

The following article was published in Adresseavisen on Tuesday the 9th of October 2012. The article is about the BLOPP-project, which is the parent project of our prototype. The article is written by Opland, Egil M. for Adresseavisen. Photos taken by Olsen, Kjell A.



Sjefredaktør: Arne Blix

Utviklingsredaktør: Kirsti Husby Nyhetsredaktør: Erlend Hansen Juvik
 Samfunnsredaktør: Stein Arne Sæther Redaktør for nett og mobil: Atle Bersvendsen
 Redaktør og leder medielab: Rolf Dyrnes Svendsen Adm. direktør: Tove Nedreberg

MIDT-NORGES FRIE STEMME



Norge går så det suser

Verdens heldigste finansminister tar i bruk mange milliarder ekstra, men nøyer seg med stø kurs.

Norsk økonomi er en solskinshistorie med vekst og velstand over alt. Finansminister Sigbjørn Johnsen (Ap) måtte til utlandet for å finne noe å bekymre seg over, da han la frem statsbudsjettet for 2013 i år. Det er sterkt vekst i fastlandsøkonomien, og oljesektoren går så det suser. Derfor kan Johnsen både bruke mer penger og sette enda mer penger på bok.

Brukere mer og sparer mer

Reaksjonene på budsjettopplegg er forholdsvis milde. Det synes å være bred enighet om det viktigste balansepunktet som gjelder bruken av oljepenger, med et forutsigbart unntak for Fremskrittspartiet. Regjeringen legger opp til å bruke 125 milliarder oljekroner, noe som er 26 milliarder mindre enn handlingsreglene åpner for. Men Johnsen øker likevel utgiftene med 23 milliarder, og stimulerer dermed en økonomi som allerede er presset, samtidig som han mener opplegget er nøytralt.

Alle kurvene på finansministerens bord går rett væi, med et klart unntak for konkurransevennen som svekket. Sysselsettingen er høy og ledigheten er lav. Renten er rekordlav og lønnsveksten høyere enn for antatt. Samtidig er pristilgangen lav, noe som trekker kjøpekraften ytterligere opp. Både privat forbruk og boligprisene øker. Veksten forsterkes av den høye aktiviteten i oljesektoren.

For sent for Entra

Regjeringen vil øke Nærings- og handelsdepartementets kapasitet og kompetanse til å håndtere statlig eierskap. – Styrkingen vil gjøre oss bedre i stand til å følge opp de investeringene som

De rødgrenne disponerer nå et budsjett på over 1000 milliarder kroner. Det er nesten 400 milliarder mer enn da det startet i 2005. I samme periode er oljefondet blitt firedoblet. Den nye prognosene for fondets størrelse er på 4400 milliarder ved utgangen av neste år. Vi har med andre ord snart fire statsbudsjett på sparekontoen, mens mange europeiske land sliter med blytning gjeld. Men andre land tvinges til dramatiske kutt, har Sigbjørn Johnsen lagt sparekniven i skuffen. I denne situasjonen er kraftig heving av tollsatser på ost og kjøtt et svært dårlig signal til omverden.

Det som kanskje overrasker mest, er den fantasiøse pengebruken som preger det rødgrenne budsjettet. Alle ekstramilliardene regjeringen tar i bruk, går stort sett bare til litt mer av det samme. Ingen nye reformer blir lansert, selv om det er storingsvalg neste år. Derfor er det vanskelig å kalte dette for et valgbudsjett.

Kommunesektoren og samferdsel er å anse som klare budsjettvinnere. Tapere er det ikke lett å finne, siden det ikke finnes kutt. Men noen områder, som for eksempel forskning, har fått mindre enn andre. Siden dette er det siste rødgrenne budsjettet for valget, synes det klart at de ikke makter å innfri fattigdomsloftene eller løftet om full sykehjemssdekning. Det er oppsiktvekkende i lys av alle ekstramilliardene de har hatt og har til disposisjon.

departementet forvalter som eier, skriver næringsminister Trond Giske (Ap) i en pressemelding. Men tiltaket kommer først til å avverge skandalen i Entra.

Nyheter

Tips oss! - Telefon 07200

nyhet@adresseavisen.no • MMS/SMS: Kodeord TIPS

Vil hjelpe

Barn sliter med å ta medisin, og foreldrene opplever ofte en kamp. En faggruppe i Trondheim har tatt fantasien i bruk for å hjelpe barn og foreldre.

Togfører Mulle spiller en viktig rolle i et nytt dataprogram som faggruppen har utviklet. Det retter seg spesielt mot barn som trenger inhalasjonsbehandling.

Når ungene drar på en digital fantasireise sammen med togføreren og vennen hans, skal ungene bli så oppslukt at behandlingen går som en lek.

– Barn som stritter imot av alle krefter når de skal ta medisin, er en kjent situasjon for mange foreldre. Medisinene smaker kanskje vondt, barna skjønner ikke hvorfor de må ta den og noen ganger tar det lang tid å bli ferdig, sier farmasøy Elin Høien Bergene ved Sykehusapoteket i Trondheim.

Både egne og andres erfaringer med slike situasjoner dannet bakgrunnen da Bergene tok initiativ til et prosjekt som skal gjøre medisineningen av små barn lettere. Med eksperter fra NTNU ble det etablert en faggruppe, og nå har gruppen utviklet prototypen til en app (applikasjon) som kan lastes ned på et lesebrett.

– Vi ønsker å motivere barna i stedet for å tvinge dem. Barn er oversett som legemiddelbrukere, mener Bergene.

Skal testes ut

I tillegg til Bergene deltar både industridesignere og eksperter på brukervennlig datateknologi i faggruppen. Prototypen på appen er foreløpig ikke testet ut på mange syke barn, men det skjer på St. Olavs Hospital over nytår. Målgruppen er barn mellom ett og fire år gamle.

– Vi ser fram til å starte tester i samarbeid med sykehuset, og har stor tro på at dette vil gi en positiv effekt. Hensikten er at barnet tar medisinen sin på riktig måte og til riktig tid. Appen er ikke ment som et leketøy, sier Ole Andreas Alsos.

Han arbeider med brukervennlighet ved Institutt for da-

tateknikk og informasjonsvitenskap ved NTNU. Alsos synes det er spennende å utvikle dataløsninger tilpasset små barn.

– Små barn gir ikke uttrykk for sine behov og ønsker på samme måte som voksne, derfor er det krevende å finne løsninger som passer for barn, sier han.

På sykehus

Appen er utviklet for barn som får inhalasjonsbehandling hjemme og på sykehus. I tillegg arbeider gruppen med ytterligere en variant for hjemmebruk.

– Hvis barnet bare hyler når det skal puste inn medisinen, kommer den ikke ned i lungene som den skal. Behandlingen får dårligere effekt, sier Bergene.

Underveis har faggruppen innhentet råd og synspunkter fra foreldre, helsepersonell, barnehageansatte og andre fagfolk. Gruppen har fått økonomisk støtte fra Extrastiftelsen, som bevilger penger til helse og rehabilitering.

– Et av fem norske barn har astma, så vi har også fått flaglig støtte fra Astma- og allergiforbundet, sier Bergene.

Ikke noe lignende

Det mangler ikke på råd til foreldre som sliter med å få gitt medisin til ungene sine. Faggruppen mener likevel den har utviklet et hjelpemiddel som er unikt.

– Vi har ikke sett noe lignende spesielt tilpasset små barn, sier industridesigner Jonas Asheim i firmaet Nice.

Han begynte å arbeide med appen da han tok mastergrad i industridesign ved NTNU.

– Vi har laget en historie som bygger på at barnet skal ta medisin, og at dette er bra for barnet. Til slutt får barnet den viktige belønningen i form av en skinngullstjerne, sier han.

EGIL M. OPLAND 951 986 89
 egil.opland@adresseavisen.no

– Hvis barnet bare hyler når det skal puste inn medisinen, kommer den ikke ned i lungene som den skal. Behandlingen får dårligere effekt.

ELIN HØIEN BERGENE, farmasøy ved Sykehusapoteket i Trondheim

Filmpris

Vinneren av Nordisk råds filmpris offentliggjøres. Årets norske kandidat er Kompani Orheim, representert ved regissør og manusforfatter Arild Andresen og manusforfatter Lars Gudmestad.

i dag

Vinneren av nobelprisen i fysikk kunngjøres i Stockholm kl. 11.45.

DEL 1

- **Nyheter** side 2-18
- **Trondheim** side 23-25
- **Utland** side 26-27
- **Økonomi** side 28-32
- **Folk** side 37-39
- **Været** side 40

DEL 2

- **Kulturnyheter** side 2-9
- **Tegneserier** side 11
- **Meninger** side 12-13
- **Ordet fritt** side 14-15
- **Snakk ut** side 16

DEL 3

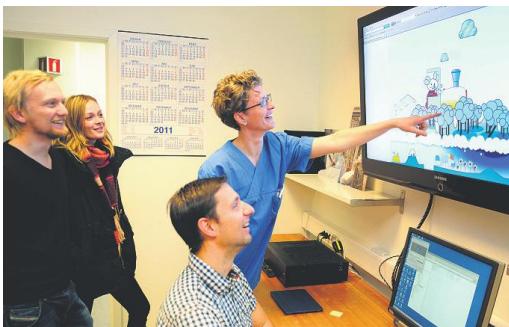
- **Sportsnyheter** side 2-10
- **Bredden** side 8-9
- **Bil** side 18-21
- **Radio og tv** side 22-23
- **Spor Adressa** side 24
- **Leserbilder**

barn med å ta medisin



Erlend ble rolig: Erlend sprellet på mor Lene Hejna Pedersens kne da han skulle få behandling. Så fanget farmasøy Elin Høien Bergene oppmerksomheten hans på dataskjermen. Sykepleier Siv Anita Myhre følger med. Foto: KJELL A. OLSEN

Mulle tok Erlends oppmerksomhet



Tverrfaglig gruppe: Industriedesigner Jonas Asheim (f.v.), stipendiatur Mariikken Høiseth, ekspert på brukervennlighet Ole Andreas Alsos og farmasøy Elin Høien Bergene er fornøyd med det nye hjelpemoduleIdet.

Gråten stilnet da Erlend (ett og et halvt år) dro på fantasireise sammen med togfører Mulle.

Erlend strittet imot og hylte da mor kom med inhalasjonsapparat, men det glemt han da fortellingen om Mulle begynte å rulle over dataskjermen.

– Det tok ikke lang tid å fange Erlends oppmerksomhet om det som foregikk på skjermen. Han satt stille hele tiden, sier mor Lene Hejna Pedersen fra Trondheim.

Flere ganger i døgnet

Erlend er innlagt på infeksjonsavdelingen ved barne- og ungdomsklinikken på St. Olavs Hos-

pital. Her behandles barn for ulike infeksjoner, blant annet astma.

– Han må bruke inhalasjonsapparat flere ganger i døgnet enten vi er hjemme eller på sykehuset. Humøret avgjør om Erlend er samarbeidsvillig. Vi gjør vårt beste for å avlede oppmerksomheten, men det er slitsomt å komme i mål, sier Pedersen.

Hun ønsker nye hjelpemoduleIder velkommen.

– Det er kjempebra at noen viser interesse for dette, sier hun.

Absolutt behov

Avdelingssykepleier Guro Karlsholm har fulgt med på utviklingen av det nye hjelpemoduleIdet.

– Inhalasjonsbehandling kan være tidkrevende, og det er ikke uvanlig at behandlingen må gjentas hyppig. Men det er vanskelig å forklare et lite barn hvorfor det er viktig å ta medisinen, og at det kan ta tid. Samarbeidet mellom barn, forelder og sykepleier i slike situasjoner kan være utfordrende, sier Karlsholm.

– Vi prøver å finne på morsomme ting for å avlede barnas oppmerksomhet, og viser blant annet videosnutter. De er imidlertid ikke tilrettelagt for slike situasjoner, og fungerer ikke like godt. Det er så absolutt behov for hjelpemoduleIder som kan gjøre behandlingen til noe mer positivt for barna, sier Karlsholm.

Appendix G

Abbreviations

BLOPP	Barnas legemiddelopplevelse	TTS	Text-to-speech
GAPP	Guardian Application	HTTP	Hypertext Transfer Protocol
CAPP	Children Application	IDE	Integrated development environment
NAAF	Norges astma- og allergiforbund	SDK	Software development kit
OS	Operative system	JVM	Java Virtual Machine
NSEP	Norsk senter for elektronisk pasientjournal	ADT	Android Development Tools
NTNU	Norwegian University of Science and Technology	SQL	Structured Query Language
IDI	Department of Computer and Information Science (at NTNU)	PHP	PHP: Hypertext Preprocessor
SHAP	Sykehusapotekene	HTML	HyperText Markup Language
IPD	Department of Product Design (at NTNU)	MVC	Model, view, controller
GUI	Graphical user interface	PFR	Parent functional requirement
SWOT	Strengths, weaknesses, opportunities and threats	CFR	Child functional requirement
API	Application program interface	KFR	Karotz functional requirement
IT	Information technology	LAN	Local area network
RFID	Radio-frequency identification	JSON	JavaScript Object Notation
REST	Representational state transfer	XML	Extensible Markup Language
		ER	Entity relationship
		VPN	Virtual private network
		SUS	System usability scale