CHALMERS

Developing a Digital Management System for iOS Devices

Using an Agile Development Method

Master of Science Thesis in the Programme Computer Science – System Development

PETER SÖDERBAUM

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2011

Developing a Digital Management System for iOS Devices
Using an Agile Development Method

PETER SÖDERBAUM

Examiner: CHRISTER CARLSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2011

Preface

This master thesis was carried out at Kredensa AB situated in Varberg and Falkenberg. I would like to thank Kredensa AB for the opportunity to do my thesis work at their restaurant in Falkenberg.

Special thanks to Patric Jernberg for all the help with details, feedback, bright ideas and interesting discussions.

Another thanks to Henrik Bengtsson for the valuable feedback when testing the application at the restaurant.

Last but not least at Kredensa AB I would like to thank Kristian Svensson for all the feedback given when he was testing the application during real operations at the restaurant.

I would like to thank Don Wells at extremeprogramming.org for giving me permission to use the figures from their website in my thesis.

Thanks to Sven Arne Andreasson at Chalmers for the feedback I received when I was designing the database used during the project.

Also big thanks to Christer Carlsson my examiner at Chalmers for the help during the work with the written thesis and for pointing me in the right direction when I needed it.

Abstract

The goal for every McDonald's restaurant is to deliver quality, service and cleanliness to all customers at every visit. To succeed with this the restaurants depends very much on the shift managers responsible for the daily operations. The purpose of this master thesis project is to simplify the work of the shift managers by providing a digital management system. The goal is that the delivered system shall be able to handle the tasks that today's system do in an easier way. It shall also add new functionality simplifying the work. If the work of the shift managers can be simplified their performance will increase. Better performing shift managers will generate better results at the restaurant.

An agile development method was used to develop the digital management system. This method was based on the extreme programming method. Though this project was carried out by a single developer the extreme programming method had to be reworked to fit a single developer environment.

The project resulted in a prototype that were tested at the restaurant during ordinary operations. Overall the results of the prototype were positive and a system like the one developed during this project will be able to help the restaurant to deliver better results.

A digital management system is a viable option for a McDonald's restaurant. The project shows that the possibilities of the digital management system exceeds today's system by far. Further development will be made to the prototype to make it able to replace today's system.

Sammanfattning

Alla McDonald's restaurangers mål är att leverera kvalitet, service och renlighet till alla gäster vid varje besök. För att lyckas med detta förlitar sig restaurangerna mycket på skiftledarna som är ansvariga för den dagliga driften. Syftet med detta examensarbete är att förenkla arbetet för skiftledarna genom att utveckla ett digitalt verktyg för uppföljning av den dagliga driften. Målet är att det levererade systemet ska klara av de uppgifter som dagens system hanterar fast på ett enklare sätt. Det ska också addera nya funktioner som förenklar arbetet. Blir arbetsledarnas arbete enklare kommer de att prestera bättre. Skiftledare som presterar bättre kommer att generera bättre resultat på restaurangen.

En agil systemutvecklingsmetod användes för att utveckla det digitala systemet. Metoden baserades på metoden extreme programming. Eftersom detta examensarbete utfördes av en ensam utvecklare behövde extreme programming metoden omarbetas för att passa ensamutveckling.

Projektet resulterade i en prototyp som testades i den dagliga verksamheten. De övergripande resultaten var positiva och visar att ett system likt det som utvecklats under detta projekt kan hjälpa restaurangen att leverera ett bättre resultat.

Slutsatsen är att ett digitalt system är en gångbar lösning för en McDonald's restaurang. Projektet visar att möjligheterna för det digitala systemet är mycket större än de för dagens system. Prototypen kommer att utvecklas vidare för att kunna ersätta nuvarande system.

Contents

List of abbreviations

API	Application Programming Interface
ERD	Entity Relationship Diagram
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
PDA	Personal Digital Assistant
QSC	See QSC&V
QSC&V	Quality, Service, Cleanliness & Value, the cornerstones in operating a McDonald's restaurant.
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TP	Travel path, a walk thru the restaurant that the shift manager do to control if all areas of the restaurant are up to standards.
VoIP	Voice over IP
XML	eXtensible Markup Language
XP	extreme programming

List of Figures

1 Introduction

1.1 Background

Kredensa AB[1], the company this master thesis project is done at, is in the business of operating three McDonald's[2] restaurants. The foundation of operating a McDonald's restaurant is working with quality, service, cleanliness & value (QSC&V or QSC).

To be able to deliver QSC&V to every customer at every visit McDonald's depends very much on shift managers. There is always at least one shift manager at the restaurant who is responsible for the daily operation of the restaurant. As a shift manager you need to use your skills to manage the crew, equipment and food. To help the shift manager with these tasks among others routines and checklists are used.

The work as a shift manager hasn't changed much during the last 15 years. Of course somethings like cash registers, sales follow ups and more has been computerized. But the basics of managing the shift and deliver perfect QSC&V to every guest hasn't evolved much.

1.2 Purpose

The purpose is to simplify the work as a shift manager at McDonald's by providing a digital management system[3] in a handheld device. By doing this, deliver better QSC&V to all the restaurants guests. It will also make the work as a shift manager at McDonald's easier, more enjoyable and rewarding.

As a shift manager there are a lot of tasks to be done every day at all times during the shift. The purpose is to make some of the tasks that the shift managers do more available and to remind the shift managers of the tasks to be done at the right time. If the system succeeds with this the shift managers can make the tasks right on time and right where they are with their handheld device.

Hopefully it will make the tasks easier and improve the efficiency. If this is the case it will free some time from the shift managers which they can use to lead their shift.

1.3 Objective

The objective is to deliver a handheld computerized management system helping the shift managers to deliver outstanding QSC&V everyday. The system should simplify the everyday work of the shift managers. The customer wants the

[1] Kredensa AB - a company operating McDonald's restaurants in Falkenberg and Varberg on the west coast of Sweden.
[2] McDonald's - the worlds largest chain of hamburger fast food restaurants.
[3] management system - the management system that should be developed during the work of this master thesis. Also referred to as the system.

system to use an Apple[4] iOS[5] application running on an iPhone[6] or iPod touch[7].

The system shall collect data during operations so it can be a source for statistics. Data shall be collected in a manner making it possible to present statistics for each shift manager.

A restaurant manager or someone else at the restaurants must be able to adjust the system to fit a specific restaurant. This adjustments shall easily be made, making the system easily maintained when the conditions at the restaurant change.

Choices in the design of the system shall be made in a fashion that makes it possible to add new features not included during this project. The delivered management system shall be a foundation to develop for more functionality and future needs.

1.4 Scope

A digital management system for a McDonald's restaurant could be a large complex system covering many different areas of the operations.

Because of the purpose to simplify the work as a shift manager, this project will only cover the parts of the operations that the shift managers do as their daily routines. But the design should be chosen in such a way that the system could be updated with new features outside this scope.

By making this delimitation the result will hopefully be a system useful in the reality at the restaurants, with potential to grow.

1.5 Method

This project will use an agile software development method [1]. When using an agile method the system is developed in iterations. For each iteration new functions are added. The agile method makes it easy to change the specifications during the development when new insights are made. This makes it efficient when working with a small and noncritical system [1].

There are several agile methods to choose from for example extreme programming (XP), Agile Unified Process, Open Unified Process and Scrum. All these methods are made for working in a team. This project is made by a single member and non of the agile methods is a prefect match. One of the fundamentals of XP is that no one process fits every project and therefore the different practices should be adjusted to fit the project. Because of this XP [2] was chosen as the starting point to form the method to be used during this project.

Some of the practises of XP that doesn't fit this project need to be reworked and other has to be left out. This project is as stated earlier made by a single

[4] Apple - an American company in the business of consumer electronics.
[5] iOS - Apples operating system for handheld devices.
[6] iPhone - a smartphone made by Apple.
[7] iPod touch - a portable media player and PDA (Personal Digital Assistant) made by Apple.

project member and this will of course make all practises that needs a team impossible to use. Some examples of practises of XP that need a team are pair programming, stand up meetings and collective code ownership.

XP depends much on automatic unit and acceptance tests. The system built during the project will depend much on a Graphical User Interface (GUI) which will make automated testing very difficult to do. But automated testing could be used to some extent.

Other practises of XP fit very well to this project and will be used. These practises will be described in detail in chapter 3 when each step of the method is described in detail.

1.6 Technical research

To understand the agile development process and XP and shape the method to use during the project Sommerville's Software Engineering 8 [1] and the homepage of XP [2] will be read.

Because the customer wants an iOS application there is a need to learn how to develop for the iOS platform. On iTunes U[8] there is a class [3] from Stanford University[9] teaching iOS programming. As technical research all the lectures and all the assignments of this class will be completed. This will hopefully give the basic knowledge to get started. For more in depth research Apples iOS Dev Center[10] and Xcodes[11] developer documentation will be used as research material.

1.7 Existing systems

When the project proposal was first submitted in February 2010, I had never heard of or seen any system like the one purposed to develop during the project. The current methods used in the restaurants are based on pens and paper check-list that are printed out every day. During McDonald's Worldwide Convention[12] in April 2010 I saw a system aiming in the same direction. This system was brand new and in testing in a few specific markets in the US. My experience tells me that it would be several years before this system hits the European market and Sweden if ever.

The functions of this system is similar to the functions the system this project will aim to end up in. A big difference is the platform the systems are developed for. The system showed at the convention used a more robust enterprise handheld device then an iPhone.

[8] iTunes U - Apples web service for managing and distributing educational audio, video and lecture notes.

[9] Stanford University - an university located in Stanford, California, United States.

[10] iOS Dev Center - Apples gathered resources for iOS development.

[11] Xcode - Apples integrated development environment (IDE) for developing iOS and other Apple specific software.

[12] McDonald's Worldwide Convention - a semi annual convention, gathering McDonald's owner operators, suppliers and employees. This is 25,000 person event with sessions, meetings and exhibits by suppliers and McDonald's cooperation.

Choosing the iOS platform will make the system more available because of the smaller size of the device, the manager can keep it in their pocket when not using it and because of Apple shipping their devices almost all over the world. It will use an interface many users already are familiar with. This is very important because the success of the system depends on the response from the shift managers going to use the system. Enterprise systems tend to be more expensive than consumer electronics mostly because of lower sales volumes, making it cheaper to get the devices for the iOS based system.

2 Analysis

QSC&V is the single most important key success factor for a McDonald's restaurant. There is an evident correlation between QSC and sales. As starting point when trying to build sales at a McDonald's restaurant you look at the QSC the restaurants deliver. If the QSC isn't were it should be the first thing to do before doing anything else is to boost the QSC.

It's possible to get new customers to the restaurant working with for example marketing, but if the QSC isn't good these new customers won't come back for another visit. When the QSC is at an appropriate level you can start working with other things like visibility and local marketing.

The McDonald's system is built on high volume sales at each restaurant. It's very hard to make any profit if the volume is low, hence the sales of each restaurant make great impact of the profitability. The restaurant depends on many things to be able to deliver great QSC thus great sales. Probably the most important of these is the performance of the shift managers at the restaurant.

To be a successful shift manager you will need to develop great people and leadership skills to make the most out of and develop your crew. You will also need to work consistent with all available tools. The tools have been refined over the years and are all making a contribution to deliver excellent QSC to all customers every day.

2.1 The systems currently in use at the restaurants

Today the restaurants use many different tools to help the shift managers deliver outstanding QSC. Most of the tools are checklists. Completing these will make the restaurant, crew and shift manager prepared for the shift and remember all things needed to be done during the daily operations. The following tools are some of the most important tools used on a daily basis.

2.1.1 Daily checklist for the shift manager

The daily checklist for the shift manager is a checklist with all the tasks the shift manager in charge has to do during that shift. The checklist contains all tasks needed to be done and what time to do them. It's like a plan for the shift. There could be several versions of these checklists adjusted for different day parts, but one day part has usually the same checklist everyday.

2.1.2 Pre-shift checklist

The pre-shift checklist is one of the tasks at the daily checklist for the shift manager. As the name indicates this checklist should be carried out before the shift manager takes over the responsibility over the shift. It covers all areas of the restaurant some examples are food, equipment and crew.

This checklist should be done as preparations for the shift and especially the rush hours. The shift manager has to walk through all areas of the restaurant to complete this checklist. Doing the pre-shift checklist carefully will make the operations during rush hours run smoothly and result in better QSC during the shift.

Completing the pre-shift checklist will produce a to-do list of things needed to be done during the shift. These things or tasks are then prioritized into four different categories by the shift manager. The priorities are based on how fast the tasks need to be carried out, some need immediate attention, some need to be done before rush hours and some need to be done during the shift.

A set of the tasks of the pre-shift checklist constitutes a daily security round. Security rounds need to be archived thus the pre-shift checklist needs to be archived. Archiving the pre-shift checklist makes it possible to follow up different shift managers and how they work with the pre-shift checklist.

2.1.3 Travel Path

A travel path (TP) can somewhat be described as a faster version of the pre-shift checklist. When the shift manager does a TP he or she walks through all areas of the restaurant and looks for things that need attention.

Just like the pre-shift checklist a TP will produce a to-do list or more likely the TP will add new tasks to the existing to-do list. The new tasks will be prioritized into the four different categories, so that it's clear which tasks to begin with.

McDonald's standard is that the shift manager should complete a TP every 30 minutes, to minimize that unforeseen problems emerge.

2.1.4 Sales report

The sales report used by the restaurants has several purposes. While filling out the sales report the shift manager needs to collect a lot of different data such as sales plan, actual sales, planned crew hours, actual crew hours and different service measurements.

This helps the shift managers to plan the sales for their shift and follow up sales, staffing and other key success factors. Another part of the sales report is a follow up for the shift manager to fill in when he or she has done a TP. This part makes it easy to follow up if a specific shift manager has done their TPs every 30 minutes or if he or she needs to improve this area of his or her shift management.

2.2 Theory

If the system of this project could make it easier for the shift managers to fill out their checklist carefully and carry out more TPs, it will help the restaurant to deliver better QSC in the long run.

Checklists could easily be digitalized, filled out with a handheld device and then archived automatically. But only making a digital copy of the checklists existing today won't make a great change for the shift managers using the system. For the system to be successful it needs to add extra value to the shift managers using it, without adding extra work. If it doesn't it will not be appreciated by the shift managers.

For the system to be successful it needs the shift managers to like it and to use it because they feel like it helps them during their daily work. Adding new features that will help the shift managers out will make them prefer the new digital system over the old tools. If the system achieves this it has great possibilities to grew and develop.

Only imagination limits the making of a possible features list. A few examples could be to produce and prioritize a to-do list from the pre-shift checklist, give the shift manager reminders when tasks like a TP needs to be done and give the shift manager feedback of current sales and crew hours.

3 Method

The focus when using an agile method like XP is on the code and to deliver a first version of the System to the customer early in the process. Documentation can of course not be omitted, but are restricted and made easier as much as possible. During each iteration of the development issues from earlier versions are fixed and functionality added.

With XP (Figure 1 shows the work flow of the XP method) as a foundation the method described in this chapter was shaped and its practices and tools were used during the project. Each iteration will consist of the steps described. Though the steps must not always be carried out linear in time, like it's done when using a well known method like the waterfall model [1]. When ending an iteration all the steps should have been done anyhow.

As reference when creating the method for this project and writing this chapter Sommerville's Software Engineering 8 [1] and the homepage of XP [2] were used.

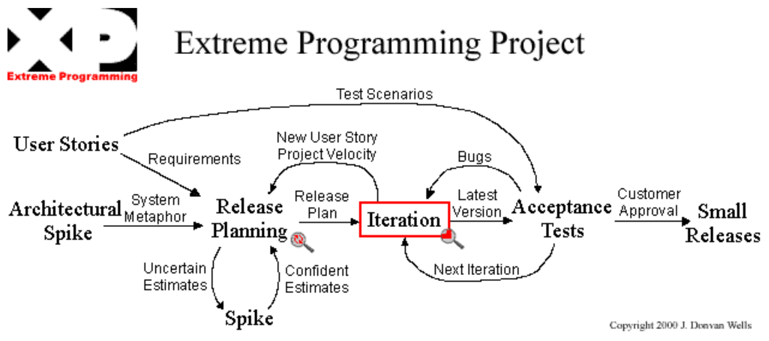


Figure 1: *The work flow of the XP method.*

3.1 Planning

3.1.1 User stories

User stories is a short (about three sentences) description of a task that the customer needs the system to do. The description should be in the language of the customer. It should focus on the needs of the customer and not consider any technical solutions like database layouts or the GUI.

The purpose of the user story is to give the developer the information needed to estimate how much time it will take to implement it. Usually the user stories are made by the customer, but during this project user stories will be made together with the customer Kredensa AB. The user stories acts as the requirements of the system. From each user story at least one acceptance test is derived.

3.1.2 Release planning

When the user stories are finished they are all written on cards so they can easily be moved around. Together with the customer the user stories are then grouped together. Each group represents a release and the story cards in that group are the stories to be implemented during the work of that release. User stories are prioritized and more important user stories are included in earlier releases. The goal for the first release is to choose as few user stories as possible that still makes a usable system.

If new facts arises it could be a reason to change the release plan. It could be things that makes some user stories not prioritized, not needed anymore or on the other hand another story could now be top priority or a new important story could have emerged. When problems like this appear it's necessary to change the release plan and which stories to include during the next release. This change to the plan can easily be done at any time and takes effects as soon as the current release is done.

Usually when using XP somewhere close to 80 user stories is a good number to have when creating a release plan. Because of the time frame of this project and that the development team is one man only this number will be far lower and each release will only cover a few stories.

3.1.3 Iteration planning

Iteration planning normally starts with the customer choosing from the user stories of the current release, which stories to be implemented during this iteration. Failed acceptance tests from previous iterations are chosen to be fixed during the iteration. Because of the fact that each release will cover only a few user stories, this step will be made by the developer instead during this project.

Each user story that will be implemented during the next iteration is then divided into tasks that need to be implemented, to add the functionality of that specific story. These tasks are made together with the customer so that the developer gets the insight needed. Tasks are written in the technical language of the developer.

3.1.4 Evaluation

As mentioned in chapter 3.1.2 an evaluation is done as soon new insights arise. The purpose of the evaluation is to put together the information needed to make a decision what to include in the next release or iteration of the system.

New insights can change, add or remove user stories and change the release or iteration plan. This is not a problem when using an agile method like XP. XP is made to fit projects where the requirements are constantly changing over time.

3.2 Design

3.2.1 Simple design

Simple design is about taking design decisions that make the design as simple as possible. What a simple design is could of course be a topic for discussion. XP recommends to use four different measurements to measure the simplicity of the code. Are the code testable, understandable, browsable and explainable (TUBE) [4].

When starting with a task during an XP project the first thing to do is to write the unit tests supporting that task. When later designing the code for that task it needs to pass the unit test and hence it must be testable. Making it testable will in most cases force the design to be simple. This is because a simpler design is easier to test than a more complex design.

For a project to be browsable it should be easy to find something that you look for in the code. Some things that make the code browsable are a god naming convention and correct use of inheritance.

The code should be understandable. Even if it's understandable to the people working with it, it could be hard to understand the code if you never seen it before. Because of that it should also be explainable to someone new.

3.2.2 Added early

Another way of keeping the code as simple as possible is never to add new functionality early. The meaning of this is to never add more functionality than intended by the user stories and tasks derived from the user stories.

It could be tempting to add extra features with seeing the possibilities, but only 10% of the added extra features will ever be used [5]. Hence 90% of the efforts will be wasted.

Instead try to keep the code as simple as possible and only implement the functionality intended for the specific iteration that is worked on at the moment.

3.2.3 Refactoring

Making a simple design could be hard when not having all the insight in the task that is needed. This is one of the reasons why XP suggests to use refactoring [6] and to refactor as much as possible when there are opportunities.

Many times it can be hard to let go of the code already written, but when insight into the problems is greater you will probably see new and improved solutions.

Simplify the code as much as possible by refactoring. Is there a simpler design use that instead. Express things once and only once, avoiding redundancy in the code will make it much easier to maintain.

Refactoring will take some time when doing the changes, but a simpler code will be easier to understand, modify and expand. In the long run refactoring will be a time saver.

3.3 Implementation

3.3.1 Customer always available

XP requires the customer always to be available as a part of the development team. This makes it possible for the developers always to speak face to face with the customer. Because of the small size of this project this wouldn't be sensible and can not be required of the customer. To keep close to the customer regular meetings will be held between customer and developer. The customer will always be reachable by e-mail or phone.

The customer needs to help out making the user stories describing the project and prioritize the user stories when making the release plan. When dividing the user stories into tasks the customer needs to help out with a detailed description of each user story. There is also a need for the customer to be involved in the acceptance tests during this project.

3.3.2 Unit test first when possible

When ever possible an automated unit test or a set of unit tests should be written before the actual code is written. Writing the test first will give the developer a better knowledge of how to write the code. So writing tests first and then the code will take about the same time it would have just to write the code directly. This practise saves the time it would have taken to write tests after the code [7]. You also get the benefit of knowing exactly when the code is finished, it's when all the unit tests have passed.

Writing tests first will also make the code simpler. The developer needs the code to pass the unit tests. Because of the facts that simpler code is often much easier to test, the developer will probably chose a simpler design.

3.4 Testing

3.4.1 Unit tests

The unit tests are as mentioned made before the code and shall be automated so that it will be easy to run them often. XP requires to write unit tests to test all functionality of all classes. The simple way to manage this is to write a unit test and implement the code passing that test. Then add another unit test and add the code passing that test and all older tests as well and continue expanding the system like this. When finding out that a unit test is missing, this test must be added to the test suite.

It can be hard to see the meaning of writing tests during development when struggling to meet a deadline. Making the tests before the code have many advantages. It's the developers themselves that write the tests when writing tests first. They are the ones knowing what the code does best and will probably write the tests faster than somebody else writing tests at a later stage.

When adding functionality all unit tests are run again and will directly give feedback if something is screwed up. Maybe this feedback forces the developer to make new design decisions. If instead writing tests when all coding is done and finding out that some of the early design needs redesigning. This scenario will probably force major changes to code written later that is depending on the faulty design. So writing the unit tests first will save time in the long run.

The unit tests is also a good help when refactoring. If a part of the code is refactored and the new and improved code passes all the unit tests you know that the code will still meet the requirements.

One of the requirements during this project is that the delivered system will run on Apples iOS platform. The system will because of this be very dependent on user interaction. User interaction is very hard to test by automated tests, but when ever possible automated unit tests shall be written for the underlying functions.

Xcode has built in functionality for making automated unit tests that are run during the build phase of the code. This functionality will be used to carry out the unit testing.

3.4.2 Acceptance tests

From each user story one or more acceptance tests are made to test all functionality of the given user story. As soon a user story is chosen to be implemented acceptance tests shall be written.

XP uses automated acceptance tests for the convenience and ability to run the tests often. In this project the acceptance tests will focus on the usability of the application for the shift managers. It will test the basic functionality but it will also focus much on the experience of using the application and how well the GUI performs. As discussed in chapter 3.4.1 this kind of testing is hard to automate so all acceptance testing will instead be carried out at the restaurant in real environment. When a release feels stable enough it will be tested during ordinary operations.

3.4.3 When a bug is found write a test

The first thing to do when finding a bug is to write a new test that the buggy code wont pass. If there is no test at all catching the bug start with an acceptance test that will catch the bug. When there is an acceptance test that catches the bug dig deeper into the code and make a unit test that catches it. This practise will give you immediate feedback if the same bug is introduced again at a later stage of the development.

4 Results

This chapter will focus on the development of the application. The outcome and results of each step in the development process will be presented.

4.1 Planning

4.1.1 User stories

The first thing done was to discuss the customers vision of the project. From this vision user stories were created together with a representative from the customer. Many user stories were discussed during this step. Because of the slim resources of the project it was decided to focus on a few of the stories.

This discussion with the customer ended up in seven user stories. The customer found these to be the most important stories and that they could make a valuable application by themselves. Six of the stories are about functionality for the shift manager using the application and the last one is about updating and administrate changes to the system, so that the restaurants can administrate changes by their own.

All user stories are shown in Figure 2. From now on the user stories will be referenced as US1, US2 and so forth.

US1: Shift manager carries out a pre-shift checklist
The shift manager of every shift fills out a pre-shift checklist. The checklist is made of tasks to control all areas of the restaurant.

US2: Shift manager makes to-do lists
The shift manager makes to-do lists for the service and kitchen area of all tasks that were not ok when filling out the pre-shift checklist. Other tasks that need attention are also added. The to-do list is prioritized into four categories so that the most urgent tasks are done first. (Generate to-do list automatic.)

US3: Present to-do lists
To-do lists are presented to the staff in service and kitchen area. Staff need to be able to mark tasks that are carried out as done.

US4: Control a pre-shift checklist
The management team at the restaurant must be able to follow up all filled out pre-shift checklists.

US5: Shift manager checklists
The shift manager has many different tasks to carry out during the shift. These tasks are presented as a to-do list. Some should be carried out at specific time. This time must be shown.

US6: Tasks reminders
Every 30 minutes the shift manager shall perform a TP. The application shall remind the shift manager every 30 minutes to do this.

US7: Update checklists
A user at the restaurant with the right authority should be able to update the checklists. All possible updates to checklists should be manageable.

Figure 2: *The user stories of the project.*

4.1.2 Release planning

When it was decided which user stories to focus the development on the user stories were divided into releases. When doing the release plan the customer wanted every release to be as small as possible but still adding value. The reason for this was to get the new features into acceptance testing as soon as possible.

Figure 3 shows the release plan used during the project.

Release 1

US1: Shift manager carries out a pre-shift checklist
US2: Shift manager makes to-do lists
US3: Present to-do lists
US4: Control a pre-shift checklist

Release 2

US5: Shift manager checklists
US6: Tasks reminders

Release 3

US7: Update checklists

Figure 3: *Release plan of the project.*

Release 1

The user story that the customer thought was the most important one was US1. Implementing a digital version of the pre-shift checklist wouldn't make sense if not implementing the to-do list showing the tasks that needed attention after the pre-shift checklist. Hence US2 was also needed for this release. The auto generated to-do list from US2 wouldn't do any good if it couldn't be presented to the crew that is going to carry out the tasks. This made up a demand to add US3 into the first release.

There was also a demand on the customer to be able to show all filled out pre-shift checklist for at least a period of one year. This demand isn't something the customer can negotiate and therefore when implementing US1 it's also needed to implement US4.

These demands resulted in US1, US2, US3 and US4 to be the content of the first release of the application.

Release 2

Of the remaining three user stories the customer valued US6 the most. Adding reminders to the application was estimated to be an easy and fast task to do. Therefore the next user story the customer wanted implemented was added to this release, that was US5. The last user story US7 was estimated to be a very heavy and time consuming task, so release 2 ended up containing US5 and US6.

Release 3

As mentioned the only user story left US7, would be very demanding to implement. Because of this release three was decided to be a single user story release.

4.1.3 Iteration planning

Each user story was broken down into detailed tasks together with the customer. There was no need for a careful iteration plan during the development. This was because the development was done by a single developer and hence not depending on other tasks done by other developers.

The iteration plan step ended up in implementing the user stories sequential with the tasks of that user story implemented one by one. Whenever an acceptance test yielded another task for any user story this task was prioritized and implemented as soon as the current task under implementation was finished.

All tasks for each user story are listed in Appendix A.2.

4.1.4 Evaluation

When doing the acceptance testing at the restaurant several problems occurred. These problems were prioritized during development and had to be included in the next iteration, hence effected the iteration planning. In chapter 4.4.2 these problems will be covered in more depth.

There were no such large issues during the project that when evaluated produced a need to change the release plan.

4.2 Design

4.2.1 Simple design

When trying to make use of a simple design the measurements of TUBE (see chapter 3.2.1) where considered.

As will be discussed in chapter 4.3.2 there were large problems writing code that could be tested with automated tests. Only considering the measurement of testing you couldn't say that the design of this project is simple.

The use of an objective programming language with its classes and every class in a header and implementation file makes the structure of the project very browsable. Added to this is the use of the IDE Xcode and the best practices for managing your project in Xcode. For a experienced user of Xcode this makes it very simple to find what you are looking for.

A informative and consistent naming convention was used when naming classes, variables, attributes etc.. This increased browsability by consistent names, understandability and explainability because of the use of informative and self-explaining names.

Further was the Cocoa[13] frameworks and best practises for iOS development used. Well known techniques like these also make the code more browsable and understandable for the experienced iOS developer.

When considering all measurements of TUBE you could define the design of the project as simple but an increase in testability would be preferred.

Taking simple design decisions was many times hard when not knowing exactly what was needed. Faulty design decisions were taken care of using refactoring when found as described in chapter 4.2.3.

4.2.2 Added early

Adding code early was avoided, but in some cases this was forgotten and code was added early despite the fact that it shouldn't have been done. As chapter 3.2.2 states the effort is often wasted. The few occasions during this project when code was added early it was just like that, a waste of time.

It should be mentioned that it was just a few cases and no large time consuming tasks that were added early, but nevertheless adding these features early was wasting time that could have been used wiser.

4.2.3 Refactoring

Thinking of refactoring when developing the application resulted in cleaner code. Common programming patterns were refactored into methods. Insights of the task during development and learning about best practises were used to refactor into a simpler design. When such findings were done the new knowledge was applied to earlier implemented code refactoring it.

Release 3 of the project could be seen as a large refactoring release. To be able to implement the user story of release 3 the design of fetching data in the application had to be redone. In the first releases the focus was to release the application for acceptance testing as fast as possible. It was decided to store the data internally during these releases. This decision made it possible not to waste time on SQL[14] database design and the heavy task of writing a web service connecting the application and database.

The functionality requested by US7 in release 3 made the design with internal data insufficient. It was needed to use the design with an external database and a web service connecting application and database. Instead of updating the application to fit the new database design the application was rewritten from scratch with the old code as a base. Hence the first two releases was used as prototype for testing the user interface and the overall function and usefulness of the application. Making this design decision made it easy to take advantage of all insights and knowledge gained during development of the first two releases, resulting in a updated, refactored and simpler code in release 3.

[13] Cocoa - one of Apples API's for Mac OS X and iOS.
[14] SQL - a language for managing data in relational databases.

4.2.4 Design decisions

This chapter covers the initial design decisions for each user story. Focus during the design was to make the application as user friendly as possible, to limit user input as much as possible and keep it simple. Changes to the design that emerged during the acceptance tests are covered in chapter 4.4.2.

US1

The menu and pre-shift checklist view were implemented using iOS table views (Figure 4) which are designed to show a lot of data. As discussed in chapter 4.2.3 it was decided to store the data locally in the application for the first two releases. When starting a new shift in the application the local pre-shift checklist data was read and stored in a working copy using Core Data[15].

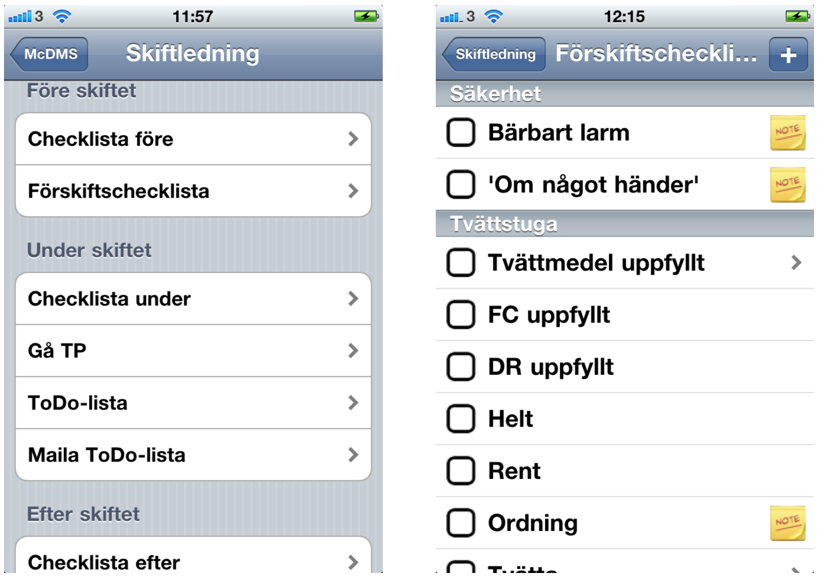


Figure 4: *Menu and pre-shift checklist table views.*

Each task in the pre-shift checklist are represented by a cell in the table view. The cell (Figure 4) shows the tasks name, status (the check box to the left), if it has a note (the note picture to the right) and if it has any child tasks (the arrow to the right). Tapping the check box changes the status of the task. Tapping the cell itself brings up the interface for the note or the child tasks if there are any.

[15] Core Data - a part of Cocoa taking care of storing data in a relational entity-attribute model. Core Data takes care of the underlying communication so that the developer can focus on the model.

US2

US2 adds the function to automatically generate a to-do list from all tasks of the pre-shift checklist that aren't ok. The to-do list are presented as a table view and sorted based on a priority of that task, which is stored in the local database. There are also a view for adding to-dos of the users choice. This view was made using elements minimizing user input. Added to-dos are also stored so if the user want to add the same to-do again it can be picked from a list, minimizing the user input once again. These two views are shown in Figure 5.

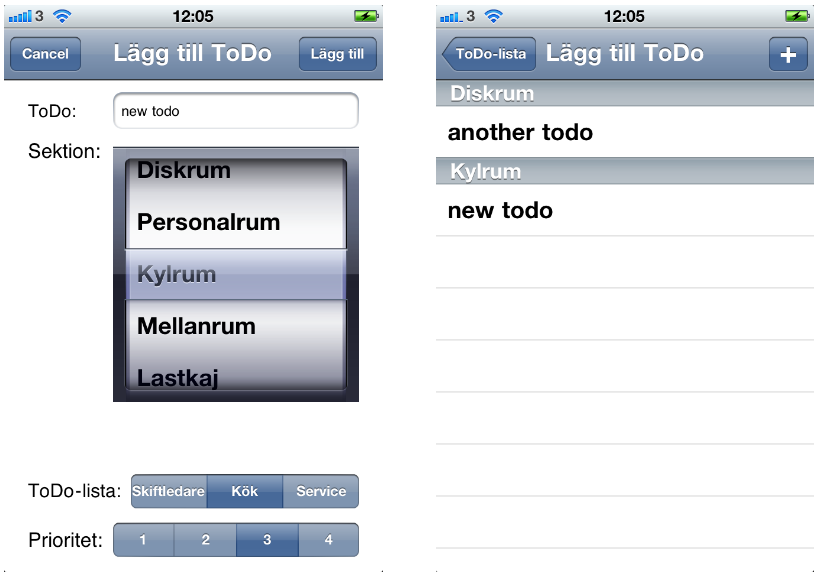


Figure 5: *Views for adding to-dos.*

US3

It was decided that the simplest way to present the to-do list to the crew when using a internal database was to e-mail the to-do list and then print it out. The to-do list generated during US2 was divided into three checklist (manager, kitchen and service) based on data from the local database. These three to-do lists were converted into HTML[16] and sent as an e-mail to the restaurants e-mail.

US4

As in US3 it was decided to e-mail the pre-shift checklist to the restaurants e-mail for archiving. When the shift manager ends the shift the pre-shift checklist is converted to HTML and e-mailed. When looking back at a pre-shift checklist

[16] HTML - a markup language, the most commonly used language for web pages.

there is a need to be able to identify who was responsible during that shift. To manage this the shift manager needs to input his or her name in one way or another. This was done via a log-in screen starting the application (Figure 6) .

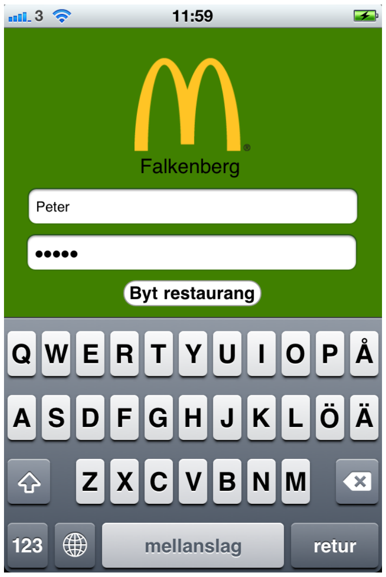


Figure 6: *Log-in screen for the application.*

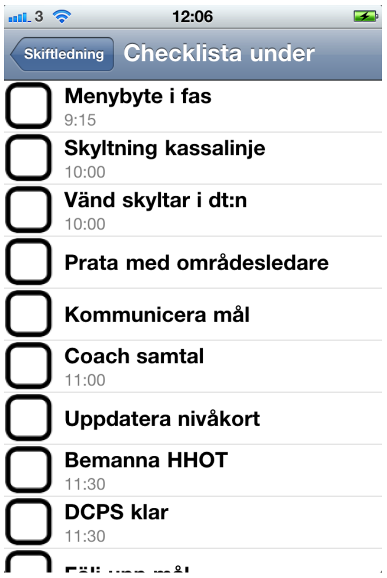
US5

The same table view layout as for the pre-shift checklist was used and also the same structure with an internal database. Though support for showing times when the tasks should be carried out was added to the table view cells in these checklists. Carry out times are stored in the local database and showed in the table view cells as a second line of text in the cell (Figure 7).

US6

Apple doesn't allow applications to run in the background in iOS. Of course the application developed must be able to present a warning to the shift manager even if it's not running in the foreground. To be able to control the device when the application is in the background Apple provides some methods. It is possible to play audio, use location services and VoIP[17] when running an application in the background. These are the only tasks that are allowed in the background, but Apple also provides something called local notifications. An application can set a local notification at a specific time. The user will then be presented a notification at that time even if the application isn't running in the foreground.

[17] VoIP - a protocol for voice communications over a network.



Checklista under

Menybyte i fas
9:15

Skyltning kassalinje
10:00

Vänd skyltar i dt:n
10:00

Prata med områdesledare

Kommunicera mål

Coach samtal
11:00

Uppdatera nivåkort

Bemanna HHOT
11:30

DCPS klar
11:30

Figure 7: *A checklist showing carry out time for some of the tasks.*

The reminders were implemented using local notification set to a specific alarm time and to repeat at a specific interval. When the clock reaches the set alarm time the application notifies the user via a sound, vibration and a message. The message from the application is shown in Figure 8. At the time the user starts a new shift it's decided when to start the warnings. During day shifts the first warning was set to 10:15 and during evening shifts 17:15. With these times as a starting point the local notification is set up to warn every 30 minutes.

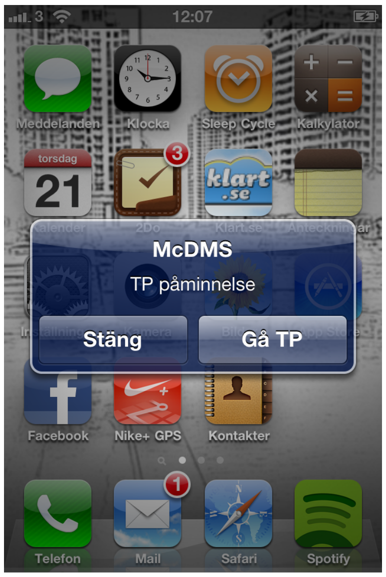


Figure 8: *A warning from the application showing that it's time to do a TP.*

US7

As discussed in chapter 4.2.3 the local database from the previous user stories was reimplemented using a MySQL[18] database. In Appendix A.1 the design of the MySQL database is shown in a ERD (Entity Relationship Diagram). A Java RESTful[19] web service[20] was implemented using NetBeans[21]. NetBeans was chosen for its abilities to generate much of the code needed for the web service.

The web service connects the MySQL database with the iOS application. The application communicate with the web service via the HTTP[22] protocol and XML[23] messages (Figure 9).

[18] MySQL - a relational database management system that is available as open source.
[19] REST - a software architecture for distributed media.
[20] Web service - a software system made for use over a network.
[21] NetBeans - a IDE for Java development.
[22] HTTP - a network protocol used for communication.
[23] XML - a set of rules for encoding messages in a form readable for computers.

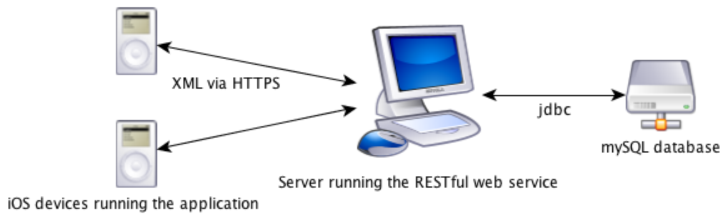

XML via HTTPS
jdbc
mySQL database
Server running the RESTful web service
iOS devices running the application

Figure 9: *Web service, MySQL database and iOS device connections.*

The web service translates XML from the application and collects data from the database via JDBC[24] and sends back to the application or updates the database according to the request. To secure the connection between application and web service the secure version of HTTP which is HTTP's was used. HTTPS provides an encrypted connection making eavesdropping hard. To authenticate the application to the web service basic access authentication[25] was used over the secure HTTPS channel.

User stories 1-6 was then reimplemented using the web service for fetching data instead of the local database. The working copy of the database stored using Core Data was decided to be kept. This allows the application to work properly when not connected to the web service. Once the application and web service are connected again the MySQL database are updated to the state of working copy in the application.

4.3 Implementation

4.3.1 Customer always available

Communication between developer and customer limited to regular meetings and contact by e-mail and phone was in most cases sufficient. In some cases during the development questions aroused that needed to be discussed with the customer when the customer was unavailable. The unavailability of the customer stalled the development process when unable to get answers.

This occasions were few and during this project with a single developer only stalling the work of one person. Considering this the problems were not reason enough to say that the communications chosen for the project were not sufficient enough, but they had their disadvantages. If the project had been larger and done with more developers this part of the method would have to be reconsidered.

[24] JDBC - a standard for connecting Java applications to SQL databases.
[25] Basic access authentication - authentication in the form of user name and password provided via a HTTP request.

4.3.2 Unit test first when possible

Code could be considered to be of different kinds. One kind is logic operations like a plus function. The function takes two integers and adds them together and returns the sum. This type of code could easily be tested with automated unit tests because you can provide it with an input of choice and you know exactly what it's supposed to return. Hooking up the function to a GUI like a calculator could then be done and you now that the underlying function of the GUI is correct. If there is any bugs they must come from the GUI in one way or another.

Most of the underlying code during this project is about fetch and present some data, for example checklists. These checklists are not set in stone and changes are always considered to meet new needs. To test if it's the correct data that is fetched the tests has to be updated according to changes in the database. This isn't a sensible solution. Instead the unit test can test if some data is loaded at all and leave the testing for data correctness to the manual acceptance testing.

To be able to test if the data is presented correct and that user interaction is taken care of the unit tests must include tests for testing the GUI. There is a basically two different methods for making automated GUI tests. The first one records user interaction and then uses the recorded interaction as a test. Another approach is to tag all elements of the GUI and then access the elements via these tags in the tests.

When using the recording method you will have problem when you change the layout of the GUI. As soon changes to the layout is made a new recording of the user interaction has to be done. The other approach tagging all elements manages layout changes with out changes to the test code as long as new elements aren't introduced. Though the code for this kind of GUI tests gets large and takes a lot of time to write.

Creating and maintaining GUI tests whatever method used is a time consuming business. In this project with limited man power and time span there is not enough time to include automated GUI tests. All GUI testing had to be carried out manually during development and acceptance testing.

Even if automated GUI tests had been used the test first approach had been impossible to follow for the GUI parts of the code. This is because both methods needs a finished GUI to set up the tests. Though the few logic operations needed and the underlying functions used by the GUI were developed with the test first (chapter 3.3.2) step of the method in mind.

4.4 Testing

4.4.1 Unit tests

Xcode is delivered with integrated support for unit testing via the SenTestingKit[26] framework which is a unit test suite for automated unit tests. It was

[26] SenTestingKit - an open source framework for automated unit tests.

easy to set up and run the tests automatically during the build phase. As discussed in the previous chapter 4.3.2 automated unit tests for the GUI was omitted. Because of this all functionality of the application isn't tested automatically with every build.

Not being able to fully test the application broke much of the advantage of the automated unit testing. For example you couldn't depend only on the automated unit tests when refactoring. Instead a lot of manually testing also had to be done to be able to test all functionality. The model developed to manage this manual unit testing was to simulate a few steps of an acceptance test.

If automated GUI testing had been used the code hooking up GUI and data could have been tested automatically. Expanding the automated unit tests with automated GUI tests would have made the test suite cover almost all of the code. The test suite would then have been a lot more reliable when refactoring.

4.4.2 Acceptance tests

When writing the user stories with the customer acceptance tests were also agreed upon. Every user story was assigned one or more acceptance tests. As described in chapter 3.4.2 the acceptance tests were carried out manually and focused on the user experience of the application. Every test was given a set of focus areas that were evaluated during the test.

In figure 10 and 11 all acceptance tests are shown. The acceptance tests are also described in more detail in Appendix A.3, there all steps of the tests are included and the focus areas to consider when the tests are carried out are described.

US1: Shift manager carries out a pre-shift checklist

US1A1
Carry out a pre-shift checklist together with a restaurant representative.

US1A2
Let a shift manager test the pre-shift checklist function during real operation.

US2: Shift manager makes to-do lists

US2A1
Control the generated to-do lists with a restaurant representative.

US2A2
Let a shift manager test the to-do list function during real operation.

US3: Present to-do lists

US3A1
Control the presented to-do lists with a restaurant representative.

US3A2
Let a shift manager and the staff at the restaurant test the generated to-do list function during real operation.

US4: Control a pre-shift checklist

US4A1
Let someone in the management team locate and control a specific filled out pre-shift checklist.

Figure 10: *The acceptance tests for US1, US2, US3 and US4.*

US5: Shift manager checklists

US5A1
Control the checklist with a restaurant representative.

US5A2
Let a shift manager at the restaurant test the checklist functions during real operation.

US6: Tasks reminders

US6A1
Let a shift manager at the restaurant test the reminder function during real operation.

US7: Update checklists

US7A1
Let a restaurant representative update the different checklists.

Figure 11: *The acceptance tests for US5, US6 and US7.*

Changes after acceptance tests

When doing acceptance test US1A1 several interface changes were decided to be done. The first change was to the check box that shows if a task in the pre-shift checklist is ok or not. A green check box with a green check mark represents ok and a empty check box not ok. To toggle between ok and not ok the check box picture should be pushed with a finger tip. In the first version the picture was 25 x 25 pixels. This size made touching it with the finger tip to change the status hard to manage. After some reading it was discovered that a button should be 44 x 44 pixels to be easy for most people to push [8]. Figure 12 shows the size of the check box before and after resizing.

There was a problem when carrying out US1A1 to follow where in the pre-shift checklist the user was. To solve this problem another change to the check boxes was decided. This time a new state was introduced. The state is used for showing a checked task that wasn't ok and uses a check box with a red cross as symbol. In Figure 13 the three kinds of check boxes are shown.

A last change about check boxes was decided during US1A1. Some tasks in the pre-shift checklist have child tasks. When trying to push the check box for one of these a warning that the state couldn't be changed popped up. This was changed to bring the user directly to the child tasks when pushing the check box without any warning.

Some tasks of the pre-shift checklist are considered to be of a more general kind. These are needed to be checked over a larger area of the restaurant. The other kind of tasks is more specific and can be checked at a specific point at the restaurant. During US1A1 the general tasks were confusing to the user. It

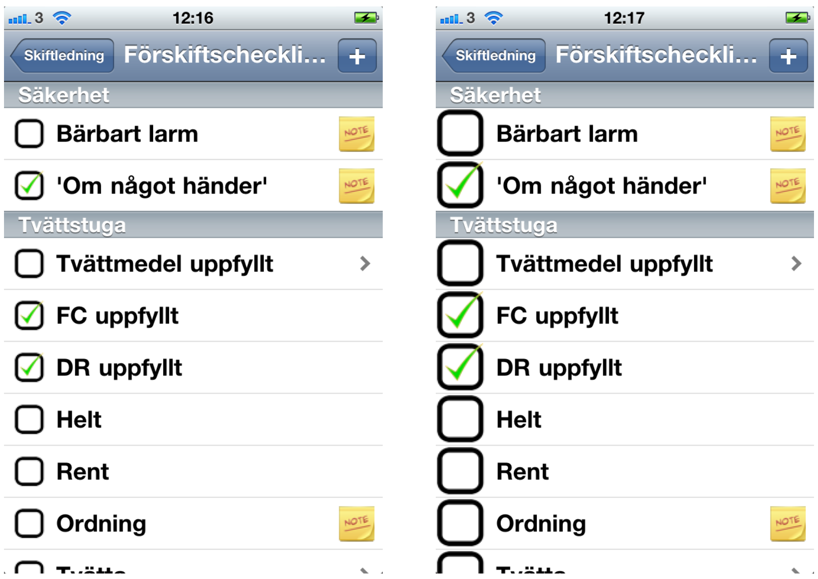


Figure 12: *Check box before and after resizing.*

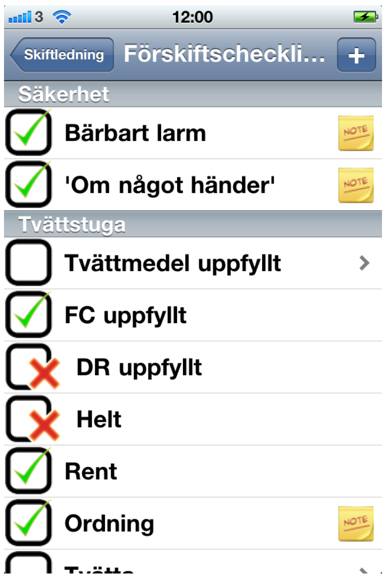
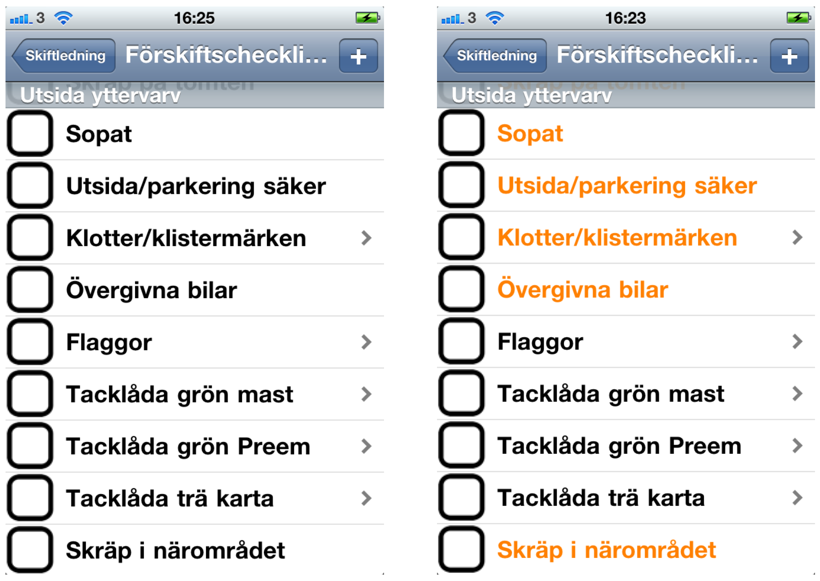


Figure 13: *The three different kinds of check boxes.*

was decided to make it more visible to the user which tasks are of the general kind. This was made by showing the general tasks in orange. Figure 14 shows the difference in the interface before and after this change.



Utsida yttervarv
- Sopat
- Utsida/parkering säker
- Klotter/klistermärken
- Övergivna bilar
- Flaggor
- Tacklåda grön mast
- Tacklåda grön Preem
- Tacklåda trä karta
- Skräp i närområdet

Utsida yttervarv
- Sopat
- Utsida/parkering säker
- Klotter/klistermärken
- Övergivna bilar
- Flaggor
- Tacklåda grön mast
- Tacklåda grön Preem
- Tacklåda trä karta
- Skräp i närområdet

Figure 14: *General tasks before and after making them more visible.*

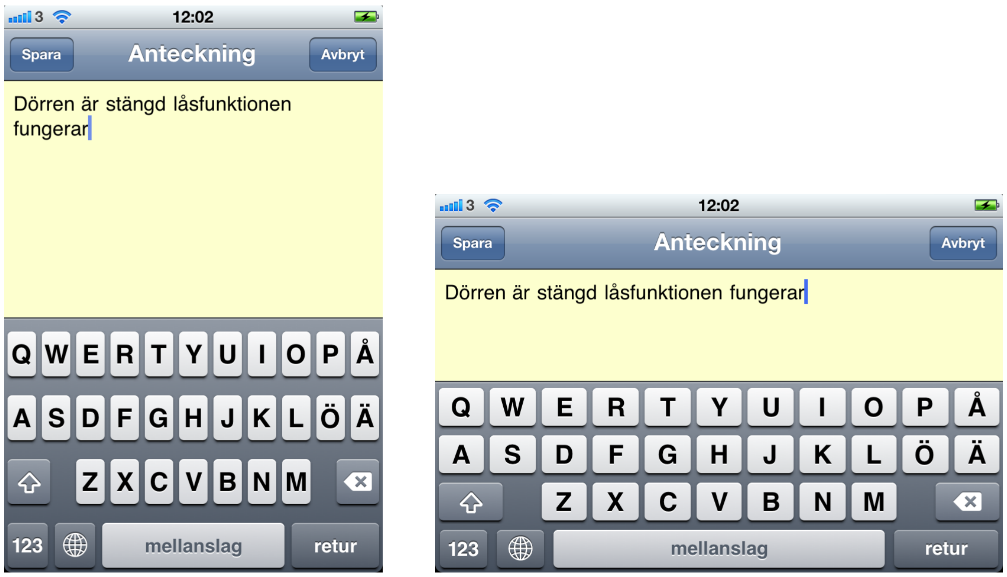
The time consumption when carrying out US1A1 was measured and improved when doing the changes described to the interface. Another interface change to improve the speed to fill out the checklist was to add landscape mode to the note view. The note function was added to be able to describe in more depth why a task is considered not ok. To add a note the touch keyboard on the device is used to input the desired note. When flipping the phone into landscape mode the keyboard buttons will become a little bit larger and writing will be faster. The different note views and keyboard layouts are shown in Figure 15.

Landscape mode was also added to the pre-shift checklist view. This was made to support longer captions to the tasks of the pre-shift checklist as shown in Figure 16.

A result of carrying out US1A2 was even more improvements to speed. The first improvement made was to make the database with tasks as detailed as possible. When the task is so detailed that the task itself describes what to do no notes are needed. This saves the time for writing a note which is rather time consuming compared to just touch a button.

It was also decided to use different versions of the pre-shift checklist for different day parts. In the day shifts the large pre-shift checklist used that far was used as before, but in the evening shifts the pre-shift checklist was slimmed down in favor of speed.

All changes to the user interface combined with the gained experience of



Dörren är stängd låsfunktionen fungerar

Figure 15: *Note view in portrait and landscape mode.*

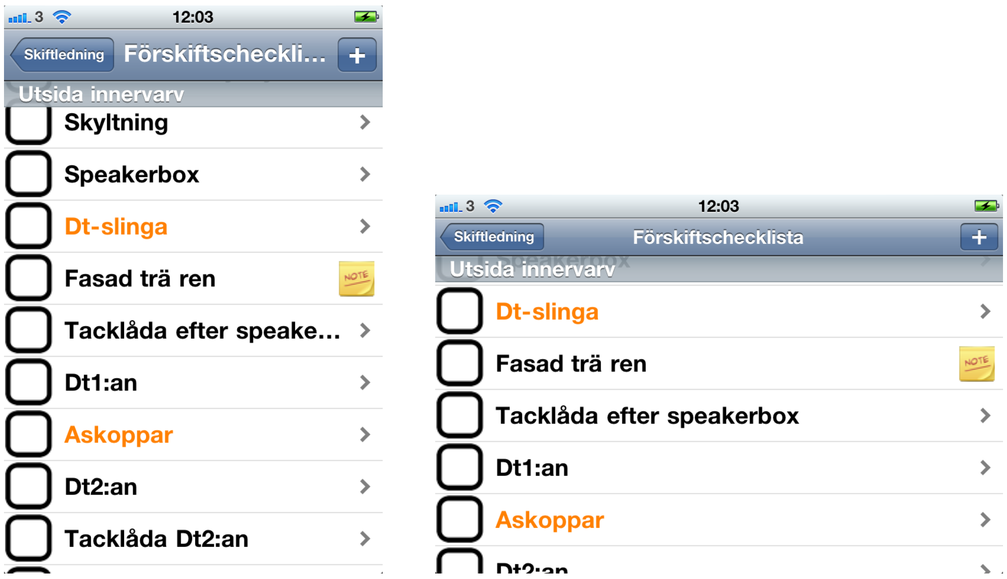


Figure 16: *Pre-shift checklist view in portrait and landscape mode.*

the application drastically lowered the time consumption. The first time when carrying out US1A1 it took 110 minutes for filling out the full version of pre-shift checklist. With all changes made and when carrying out US1A2 it took slightly less then 60 minutes. Hence the time consumption for this task has almost been cut in half. Though when carrying out the pre-shift checklist with the old manual system it takes only 30 minutes. Even if carrying out the pre-shift checklist with the application take almost 30 minutes more then the old method the digital system is considered more efficient. The reason for this is that the carefulness when using the application results in less problems during the rest of the shift.

There were also changes made after the other acceptance tests. US2A2 yielded a change to how the to-do list is sorted for easier reading and US3A2 resulted in a different layout to notes in the mailed to-do list.

A large problem discovered during US6A2 was that it was very difficult to hear the warnings from the application during hours with higher sales volume and higher noise in the restaurant. The first version of the warnings used the standard sound for warnings which is a short beep. For the second version a custom sound was prepared with the maximum allowed length of 29 seconds. In both cases the sound was combined with the built in vibration for notifications which is one short vibration in the beginning of the sound. The length of the vibration can't be changed. None of these warnings was enough to get the shift manager's attention during higher sales volume. Because of the limitations Apple has made to the ability to present notifications to the user, no good solution for this problem has been found.

4.4.3 When a bug is found write a test

If a bug was found and there was no acceptance test catching it one of the existing acceptance tests was expanded. The acceptance test was expanded with all the steps needed to catch the bug.

Then if possible an automated unit test was written that also caught the bug. If there was no possibility to write an automated unit test the steps added to the acceptance test was used as a unit test during development.

The description of the acceptance tests in Appendix A.3 includes the final versions of the acceptance tests with all steps added during the development of the application.

5 Discussion

This chapter will discuss the method used during the project, how well suited the iOS platform was and the result of the system developed during the project.

5.1 Method

During the most part of the project the method felt well suited. Planning went well working with the customer to create user stories and the release plan. The lighter (compared to XP) version of iteration planning used also felt sufficient enough for a single development environment.

XP's practises for design used during the project all worked out well. Simple design could have been driven more by the test first practise but the other parts of simple design was successfully used. Avoiding to add new functions early helped the development to stay focused and sped up the delivering of each release. Refactoring kept the code cleaner and simpler making it easier to work with.

Unit testing was a setback during this project. The tests for logic operations and the code used by the GUI went well and helped out during the development. To be able to meet the objectives of the project automated GUI testing was omitted. All testing of the GUI was instead made manually. This action devalued the unit test practice of the method, but it had to be done to manage as much development as possible. For this project with such a limited time frame and only one developer it probably haven't been worth the time and effort needed to write automated tests for the GUI, but it would have made the unit test practice of the method more successful.

Acceptance testing went very well and improved the application in several ways. The manual way of carrying out the acceptance test were great. If the acceptance tests would be automated the development could be more efficient. To some extent they could have been done using the same technology as for automated unit tests for the GUI. As in the case with the unit tests it would probably not have been worth the effort in a small project like this. It wouldn't have caught all things that the manually testing used during this project did either. Many of the changes done to the interface after carrying out the acceptance tests would never have been considered if only using automated testing.

5.2 Technical choices

5.2.1 The iOS platform

The iOS platform wasn't a choice made during the development. It was a requirement from the customer. This requirements of course affected the project.

Developing for a platform like iOS gives many things for free in the provided frameworks. It's easy and fast to make a GUI based iOS application because of the fact that all ordinary GUI elements are already there to use. You just need to put them together and hook them up. Do this and you made an application with an interface that many of the users already are familiar with. This is of course a large benefit and will shorten the learning curve of the application.

iOS is delivered with many features and has solutions for almost every task needed during this project, but no good solution for the warnings have been found. To solve this problem a work around must be developed or it's just to wait for Apple to increase the capabilities of the notification functions in iOS. Waiting for Apple could take forever so some kind of work around is probably the way to go. If the project instead would have used the Android[27] platform which is open source this problem had never occurred. Using the Android platform could of course have given other problems instead.

Overall iOS is well fitted for the application developed during this project and is very easy to develop for.

5.2.2 The system design

In this section we see the first two release of the project as a prototype for testing the iOS application. These releases are not considered during this section. Instead we focus on the system developed during release three of the project.

The focus of the third release was to make the system manageable by the restaurants, without any help from an administrator of the system. Changes made to the database on one device should be made on all devices. To manage this the local databases stored in the application on each device had to be replaced.

Instead a distributed system design as described in chapter 4.2.4 and Figure 9 was used. As described in chapter 4.2.4 the design uses one server stored database. The iOS frameworks doesn't provide any methods for connecting to external databases. To provide the iOS application possibility to read and write to the database a web service is used as intermediary service. When the application needs to access the external database it sends a request to the web service. The web service responds to the request and returns the information the application asked for or updates the database according to the request. Application and web service talks via XML messages.

To use this kind of design was a simple choice. The need to use only one database for all devices using the system was obvious. Not only for making updates possible. It also allows for other clients to connect to the database in real time and provide its users with meaningful data like the current state of the to-do list. Because of the fact that there is no provided framework in iOS for connecting to external databases there was a need for using a intermediary service like a web service.

As a result of using this design with a web service, it's possible to write clients connecting to the system for any platform as long the platform can make HTTP requests and interpret XML messages.

[27] Android - Googles platform/operating system for handheld devices.

5.2.3 The web service

To write the web service Java was chosen as the language. Some alternative are .NET[28] or PHP[29] . Java was the choice for several reasons. At first it's the language that I the developer of this project is most familiar with. Another reason was that NetBeans is able to automatically generate a Java web service from a connected database.

Another choice you have to make when creating a web service is what kind of web service to use. Two options are SOAP and REST. SOAP is a protocol for defining how a web service and its clients should communicate. When using SOAP you can basically chose exactly how the communication and security should be used. REST is somewhat more lightweight and uses HTTP for the communication between web service and client. When using a RESTful web service you are limited to the capabilities and security of HTTP. But the set up of the a RESTful web service is easier and not as time consuming. A RESTful web service will be able to do all that is needed during this project and will provide satisfying security.

The automatically generated web service from NetBeans is of the RESTful kind. When generating the web service NetBeans creates several Java web service classes with methods for basic accesses, updates and deletes to database entries. For special cases needed by the system methods had to be added to the generated classes. The methods of these classes are provided to clients as an API and are called via HTTP requests. When a response from the web service is required it's sent as an XML message. The client then interprets the XML message and takes appropriate action.

5.2.4 The database

There are several database management systems to chose from among others Oracle, PostgreSQL, Mimer SQL and MySQL. MySQL was chosen because it's open source and free of cost. It's also easy to connect a MySQL database to a Java web service via the API provided in Java.

The actual design of the database was a complex activity and the design changed and evolved several times during the design process. Sven Arne Andreasson at Chalmers was very helpful during this process and helped out with insights improving the database design.

You could say that the database has several different areas. There are some entities describing checklists and their relations working as a template. Then there are other entities that implements these templates and together constitutes checklists and tasks for a specific shift. The database stores information of the restaurant and the users of that restaurant. User data from the database are used for storing who was responsible during a shift so statistics for each managers could be collected. The user data is also used for authentication

[28] .NET - a software framework by Microsoft that runs on Microsoft Windows.
[29] PHP - a general purpose scripting language used for web development.

when connecting to the web service. In Appendix A.1 a ERD for the MySQL database is presented.

5.3 Delivered system

The application developed carries out many of the tasks of the old system described in chapter 2.1, but is it superior to its older version using paper and pencil? The application is not limited in size like the old paper version is. This made it possible to make an extremely detailed pre-shift checklist. A detailed checklist makes the differences when different shift managers carries out the checklist smaller. It results in fewer problems during the shift because the upcoming problems can be solved beforehand. The detailed pre-shift checklist also fills an educational purpose for new shift managers with less experience in what to look for.

When the application generates and priorities a to-do list it will be done the same way every shift. Not like today when it depends on the shift managers. The checklist will have the same look and layout every shift. These things simplify for the crew using the to-do list.

Release 3 of the project wasn't fully implemented due to lack of time. The functionality of collecting data and the desired maintainability of the system that were included in the objectives in chapter 1.3 were because of this not implemented. The database, the foundation of the web service is done and the reimplementation of the application is begun. What needs to be done is to fully reimplement user story 1-6, implement the update functionality of user story 7 and some changes and added functions to the web service.

The application developed could be seen as prototype. A prototype that already performs well by simplifying the work for the shift managers and adding some advantages. As soon as the function for TP warnings works properly and release 3 is fully implemented the application will fulfill all the objectives. It will also have a great feature in the TP warnings that there is no solution for in the system used today. When these things are solved the new digital system could probably replace today's system.

Even though the application performs well already it has just scratched the surface of the possibilities. Chapter 6.1 discusses some of the possibilities future version can incorporate.

The managers testing the application found it to be more fun to carry out the tasks using the device. That fact by itself makes the developed system a viable option.

6 Conclusions

When doing a user interaction heavy project like this, the used method is a great support and process to use. Though before starting out, the unit testing of the project should be considered. What technologies should be used for unit testing and in what ways. Should the GUI testing be automated and how? If not a method for carrying out manual test must be considered. Making these decisions carefully will make the method a even larger support.

The iOS platform is a good choice for this kind of application because of its rich frameworks. An application of the kind developed during this project is a viable solution for the customer and simplifies the work for the shift managers. In the long run it will probably generate better results for the restaurant. It would be interesting to carry out a more extensive test during a longer period of time, to see if and how using the application affects the results. There are some work to do before the application can replace today's system, but the application has great possibilities to evolve and be even more valuable as it grows.

6.1 Future work

This chapter discuses what the next steps of the system are and some of the possible features future releases can incorporate.

Finishing release 3 of the project is the most important thing to do. The reimplementation of all user stories are not fully done and US7 isn't implemented. When this is done the application has a good foundation to build on.

The functionality for warning the shift manager when it's time to carry out a TP must be solved. It could be made as a work around of the functionality in iOS. Another solution could be to make a client connected to the database presenting useful information to the crew and managers of the restaurants. A computer screen can be mounted on a wall showing this client. It can present information like to-dos that need immediate attention, reminders when it's time to carry out a TP and if connected to the sales system it can present service and sales measurements and much more.

Different user groups can have different permissions and carry out different tasks in the application. As an example a crew user could be added. This user could have permission to see the to-do list and mark to-dos as done. When the shift manager gets a TP warning the application can present a list of tasks to control during the TP. This list can include all to-dos done since last TP helping the shift manager to follow up that the tasks have been done carefully.

The system can send warnings for tasks that haven't been completed at a specific time to for example the restaurant manager and supervisors. This feature will give them immediate information if time critical tasks haven't been done in time.

Another feature that can be included in the system is support for equipment and spare parts. The application can be used to report troubles with the equipment and to search for needed spare parts and their location.

These are just a few of all the features and functions that the system can be used for. The opportunities are endless if taken care of.

Even if this master thesis is ended here the project isn't. I will develop the system further in my spare time and the first two steps mentioned in this chapter is top priority. Time will tell how the system is developed and what it grows into.

References

[1] Sommerville, Software Engineering 8, Addison Wesley, United States of America, 2007.

[2] D. Wells, Extreme programming: A gentle introduction., [retrieved 2011-02-08] (2009).
URL `http://www.extremeprogramming.org/`

[3] Stanford_University, CS 193P iPhone application development, [retrieved 2011-04-24] (2011).
URL `http://www.stanford.edu/class/cs193p/cgi-bin/drupal/`

[4] Z. Spencer, Judging code simplicity? fit it through the TUBE!, [retrieved 2011-02-08] (2010).
URL `http://www.zacharyspencer.com/2010/03/judging-code-simplicity-fit-it-through-the-tube/`

[5] D. Wells, You aren't going to need it. (YAGNI), [retrieved 2011-02-08] (1999).
URL `http://www.extremeprogramming.org/rules/early.html`

[6] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison Wesley, United States of America, 1999.

[7] D. Wells, Test first, [retrieved 2011-02-08] (2000).
URL `http://www.extremeprogramming.org/rules/testfirst.html`

[8] Apple_Inc., iOS human interface guidelines: User experience guidelines, [retrieved 2011-04-18] (2011).
URL `http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/`

A Appendix

A.1 MySQL database ERD

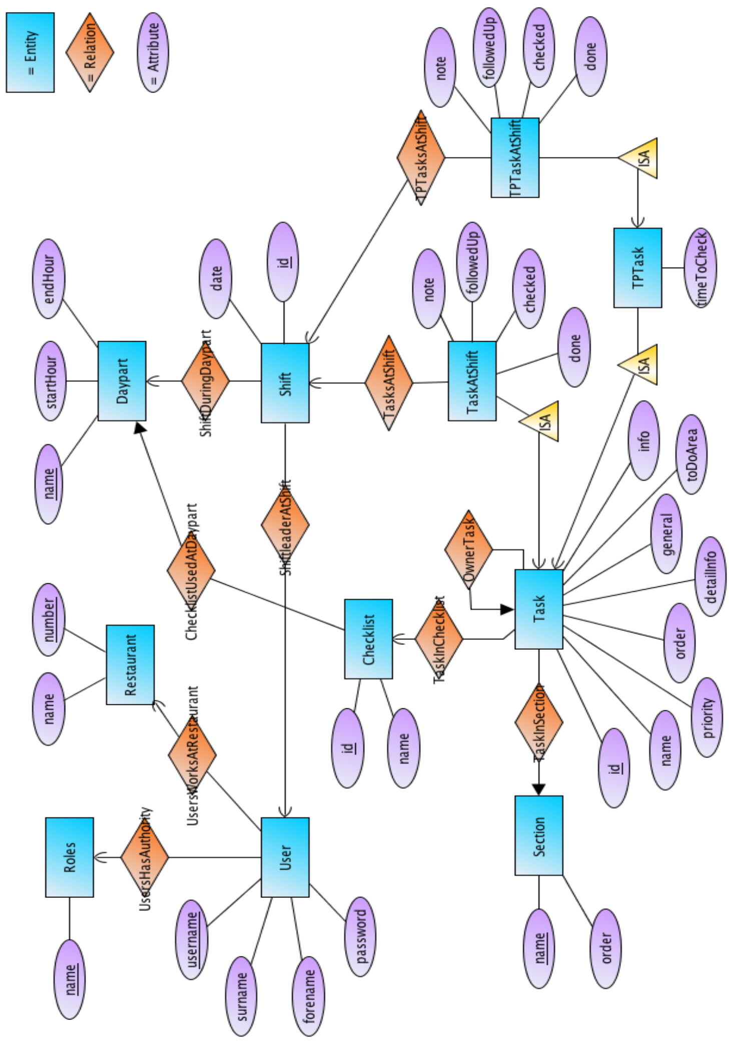


Figure 17: *ERD for the MySQL database.*

A.2 Iteration plan

US1: Shift manager carries out a pre-shift checklist

- make menu system
- make menu item
- read database of tasks
- present tasks in table view sorted by sections then by name
- child tasks
- check/uncheck tasks
- read/write notes

Added after acceptance testing

- larger check box
- when checking task with under tasks go the the undertakes
- checked but not ok
- mark general tasks
- add support for landscape mode to pre-shift checklist
- add support for landscape mode to notes
- different checklist for different day parts

Figure 18: *Tasks for US1.*

US2: Shift manager makes to-do lists

- make menu item
- automatic generate to-do list from pre-shift checklist
- present to-do list as table view
- sort by priority then by section
- check/uncheck tasks
- add to-do tasks
- delete tasks
- read/write notes

Added after acceptance testing
- also sort by name of task
- when mailing to-do print notes added to existing note on own line

US3: Present to-do lists

- make menu item
- make a HTML copy of the to-do list
- divide to-do list into different areas (manager/kitchen/service)
- same sorting as when presented in the application
- mail to-do list

Added after acceptance testing
- when mailing to-do print notes added to existing note on own line

US4: Control a pre-shift checklist

- make a HTML copy of the pre-shift checklist
- sorted as when presented in the application
- add name and date to HTML version
- log-in to app with at least name
- when closing the shift mail the pre-shift checklist

Figure 19: *Tasks for US2, US3, and US4.*

US5: Shift manager checklists

- databases for checklists
- menu items for checklists
- present checklist as table view
- check/uncheck tasks
- read/write notes
- carry out times from database as table view cell detail text

Added after acceptance testing
- different checklist for different day parts
- when checking pre-shift checklist jump to the checklist

US6: Tasks reminders

- start local notifications when starting shift
- should warn every 30 minutes
- starting 10.15 when day shift and 17.15 when evening shift
- when warning present to user that it's time to walk a TP

Added after acceptance testing
- longer music tune

US7: Update checklists

- design MySQL database
- make a Java web service connected to database for presenting and manipulating data
- reimplement US1-US6 using MySQL database instead of local database

Figure 20: *Tasks for US5, US6 and US7.*

A.3 Acceptance tests

US1: Shift manager carries out a pre-shift checklist

US1A1

Carry out a pre-shift checklist together with a restaurant representative.

Steps (see user guide for specific instructions)

1. Log-in
2. Start a new shift
3. Go to the pre-shift checklist
4. Fill out the pre-shift checklist

Focus areas:

-are any bugs discovered?
-are there any needs for interface changes?
-are the presented data correct?
-how much time is needed to fulfill the task?

US1A2

Let a shift manager test the pre-shift checklist function during real operation.

Steps (see user guide for specific instructions)

1. Log-in
2. Start a new shift
3. Go to the pre-shift checklist
4. Fill out the pre-shift checklist

Focus areas:

-are any bugs discovered?
-are there any needs for interface changes?
-are the presented data correct?
-how much time is needed to fulfill the task?

Figure 21: *The acceptance tests for US1 in detail.*

US2: Shift manager makes to-do lists

US2A1

Control the generated to-do lists with a restaurant representative.

Steps (see user guide for specific instructions)
1. Log-in 2. Start a new shift 3. Go to the pre-shift checklist 4. Back out to the shift manager menu 5. Go to the to-do list 6. Go through all to-dos in the list

Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?* *-are the presented data correct?*

US2A2

Let a shift manager test the to-do list function during real operation.

Steps (see user guide for specific instructions)
1. Perform US1A2 2. Back out to the shift manager menu 3. Go to the to-do list 4. Work with the to-do list during the shift

Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?* *-are the presented data correct?*

Figure 22: *The acceptance tests for US2 in detail.*

US3: Present to-do lists

US3A1

Control the presented to-do lists with a restaurant representative.

Steps (see user guide for specific instructions)
1. Log-in 2. Start a new shift 3. Go to the pre-shift checklist 4. Back out to the shift manager menu 5. Send the to-do list as an e-mail 6. Print the e-mail with the to-do list 7. Go through all to-dos in the list

Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?* *-are the presented data correct?*

US3A2

Let a shift manager and the staff at the restaurant test the generated to-do list function during real operation.

Steps (see user guide for specific instructions)
1. Perform US1A2 2. Back out to the shift manager menu 3. Send the to-do list as an e-mail 4. Print the e-mail with the to-do list 5. Present the printed to-do list

Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?* *-are the presented data correct?*

Figure 23: *The acceptance tests for US3 in detail.*

US4: Control a pre-shift checklist

US4A1

Let someone in the management team locate and control a specific filled out pre-shift checklist.

Steps (see user guide for specific instructions)
1. Decide which pre-shift checklist to control 2. Open the restaurant e-mail account 3. Find the pre-shift checklist 4. Control the pre-shift checklist

Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?* *-are the presented data correct?*

Figure 24: *The acceptance test for US4 in detail.*

US5: Shift manager checklists

US5A1

Control the checklists with a restaurant representative.

Steps (see user guide for specific instructions)
1. Log-in 2. Start a new shift 3. Go to the checklist before 4. Control and test the contents of the checklist 5. Back out to the shift manager menu 6. Go to the checklist under 7. Control and test the contents of the checklist 8. Back out to the shift manager menu 9. Go to the checklist after 10. Control and test the contents of the checklist
Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?* *-are the presented data correct?*

US5A2

Let a shift manager at the restaurant test the checklist function during real operation.

Steps (see user guide for specific instructions)
1. Log-in 2. Start new a shift 3. Carry out a shift using the checklists for the parts before, under and after
Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?* *-are the presented data correct?*

Figure 25: *The acceptance tests for US5 in detail.*

US6: Tasks reminders

US6A1

Let a shift manager at the restaurant test the reminder function during real operation.

Steps (see user guide for specific instructions)
1. Log-in 2. Start new a shift 3. Carry out a shift with the device by your side

Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?* *-do the shift manager notice the reminders?*

Figure 26: *The acceptance test for US6 in detail.*

US7: Update checklists

US7A1

Let a restaurant representative update the different checklists.

Steps (see user guide for specific instructions)
1. Log-in with the role of at least restaurant manager 2. Go to the update menu 3. Select a checklist to update 4. Update a task of the checklist 5. Change place of a task 6. Add a new task 7. Delete a task

Focus areas:
-are any bugs discovered? *-are there any needs for interface changes?*

Figure 27: *The acceptance test for US7 in detail.*

A.4 Application user guide (in Swedish)

McDMS
Användarmanual

Peter Söderbaum

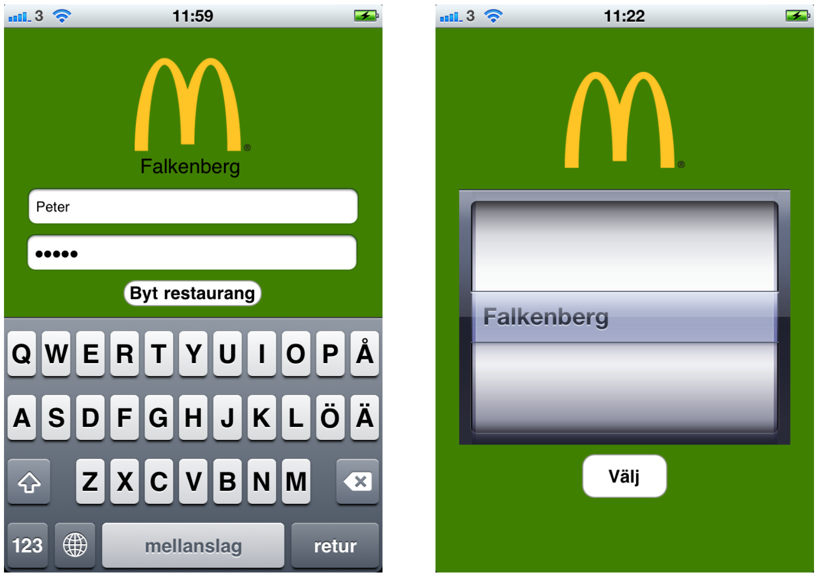
23 april 2011

Innehåll

1 Användarmanual

1.1 Logga in

För att logga in börja med att välja vilken restaurang du vill logga in till. Det görs genom att trycka på knappen “Byt restaurang” (Figur 1, vänster). En ny vy visas, rulla fram restaurangen du vill logga in på och tryck på “Välj” (Figur 1, höger). Fyll sedan i ditt användarnamn och lösenord. För att logga in tryck “retur” på tangentbordet.



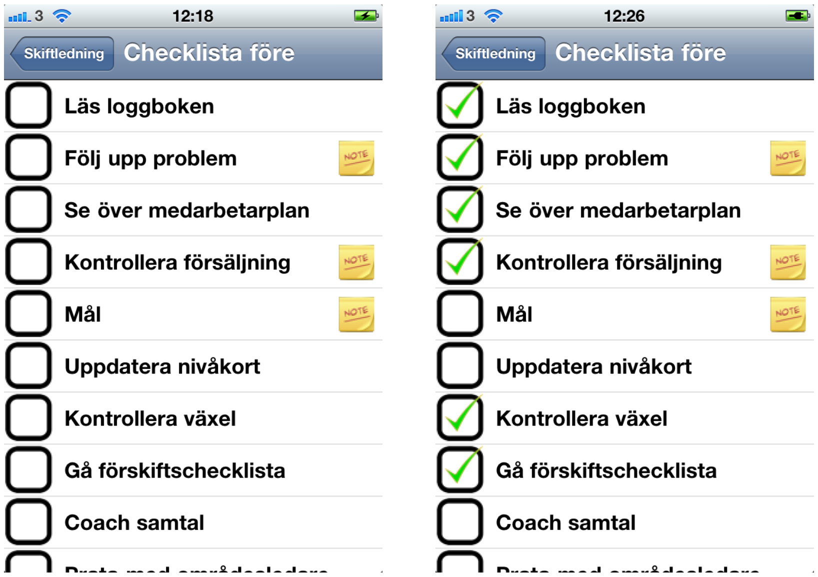
Figur 1: *Inloggnings skärm och Restaurang väljar vy.*

1.2 Starta ett nytt skift

Starta ett nytt skift genom att trycka på “Starta nytt skift”. Du kommer att visas skiftledarmenyn. Applikationen kommer även att börja ge påminnelser när det är dags att gå en TP. Första påminnelsen kommer 10.15 vid dag skift och 17.15 vid kvällsskift. Efter det påminner applikationen var 30 minut fram tills skiftet avslutas.

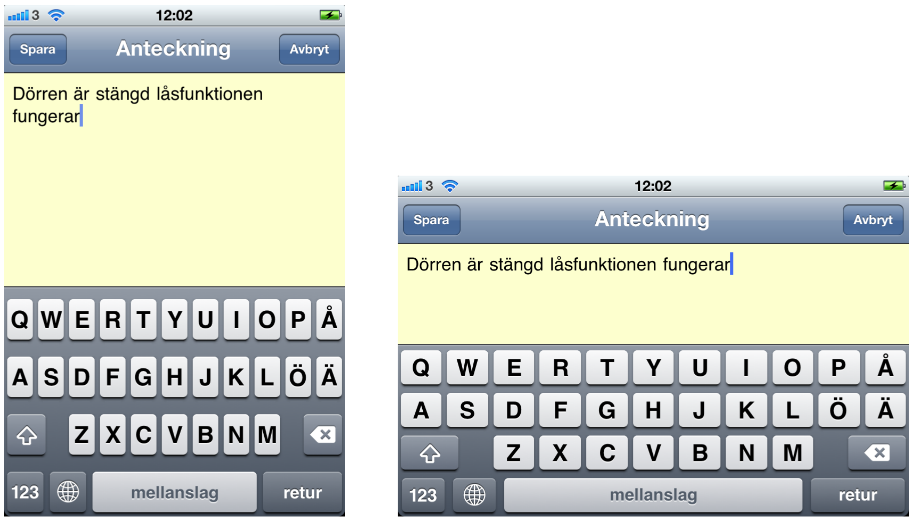
1.3 Checklistor

I skiftledarmenyn finns tre checklistor med de uppgifter som ska utföras under skiftet. Uppdelningen är före, under och efter skiftet. För att se en av checklistorna klicka på den. När en uppgift på checklistan är klar bocka av den genom att trycka på rutan till vänster om uppgiften, en grön bock kommer synas i rutan (Figur 2).



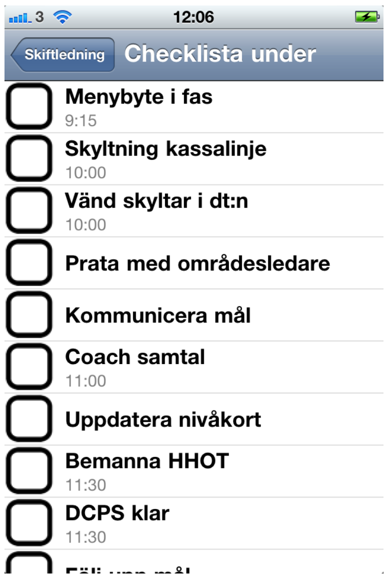
Figur 2: *Checklista med och utan genomförda uppgifter.*

Vissa uppgifter har en anteckning som vidare beskriver uppgiften. Dessa uppgifter har en gul post-it lapp till höger om uppgiften. För att se och redigera en anteckning klicka var som helst på uppgiften. Det går även att klicka på uppgifterna utan post-it lapp för skriva en egen anteckning. För att skriva en anteckning använd tangentbordet och trycka på “Spara”-knappen längst upp till vänster när du är klar. Det går också att avbryta utan att spara genom att trycka på “Avbryt”-knappen längst upp till höger. För att få ett större tangentbord går det bra att vända telefonen till landskapsläge. Figur 3 visar porträtt och landskapsläget för anteckningar.


Dörren är stängd låsfunktionen
fungerar

Dörren är stängd låsfunktionen fungerar

Figur 3: *Porträtt och landskapslägen för anteckningar.*


3 12:06

Skiftledning Checklista under

Menybyte i fas
9:15

Skyltning kassalinje
10:00

Vänd skyltar i dt:n
10:00

Prata med områdesledare

Kommunicera mål

Coach samtal
11:00

Uppdatera nivåkort

Bemanna HHOT
11:30

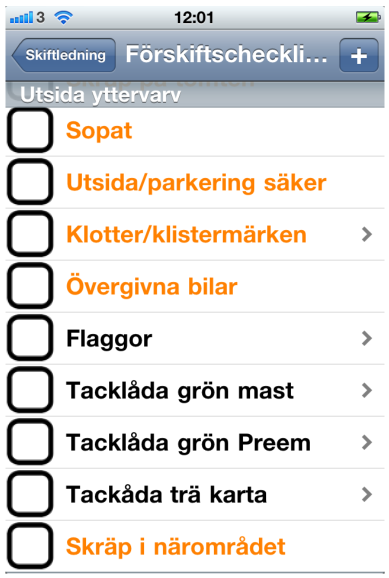
DCPS klar
11:30

Figur 4: *Uppgifter med en tid.*

Några uppgifter på checklistorna visar även vilken tid de ska utföras. Tiden visas i sådana fall i en textrad under själva uppgiften (Figur 4).

1.4 Förskiftschecklista

För att komma till förskiftschecklistan klicka antingen på “Förskiftschecklista” i skiftledarmenyn eller “Gå förskiftschecklista” i checklistan före skiftet. Förskiftschecklistan är uppdelad i sektioner och är upplagd så att man ska gå den punkt för punkt med undantag för några generella punkter. De generella punkterna är skrivna med orange text (Figur 5) och innebär att dessa punkter ska kontrolleras under hela sektionen. Viss uppgifter har även underuppgifter dessa är markerade med en pil (Figur 5) till höger om uppgiften. För att se underuppgifterna tryck på uppgiften.



Utsida yttervarv

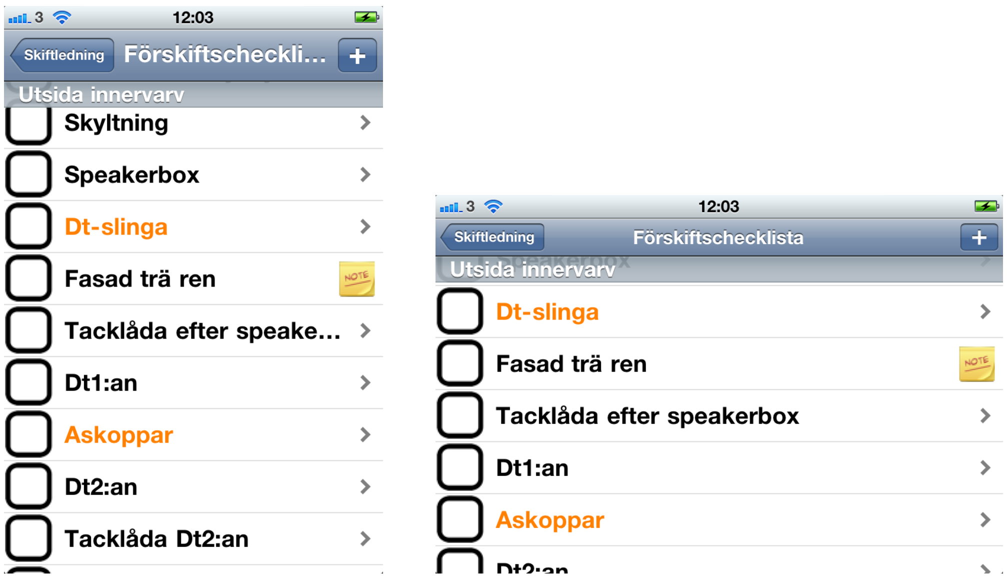
- Sopat
- Utsida/parkering säker
- Klotter/klistermärken
- Övergivna bilar
- Flaggor
- Tacklåda grön mast
- Tacklåda grön Preem
- Tackåda trä karta
- Skräp i närområdet

Figur 5: *Generella punkter och pilar markerande underuppgifter.*

Funktionerna gällande anteckningar beskrivna i kapitel 1.3 gäller även för förskiftschecklistan. Det går också att vända till landskapsläge för att se långa uppgiftsnamn (Figur 6).

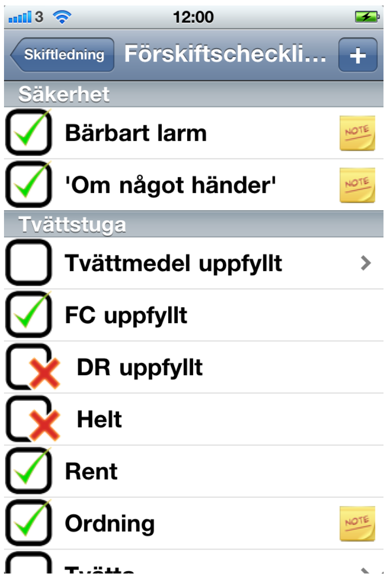
När förskiftschecklistan genomförs markeras de uppgifter som är ok med en grön bock genom att klicka på rutan till vänster om uppgiften. De uppgifter som inte är ok markeras med ett rött kryss genom att hålla nere fingret på samma ruta. I Figur 7 visas de olika markeringsalternativen.

Plustecknet uppe i högra hörnet används för att lägga till egna todo-uppgifter om något saknas. Denna funktion beskrivs mer ingående i kapitel 1.5.


3 12:03
Skiftledning Förskiftscheckli... +
Utsida innervarv
Skyltning
Speakerbox
Dt-slinga
Fasad trä ren
Tacklåda efter speake...
Dt1:an
Askoppar
Dt2:an
Tacklåda Dt2:an

3 12:03
Skiftledning Förskiftschecklista +
Utsida innervarv
Dt-slinga
Fasad trä ren
Tacklåda efter speakerbox
Dt1:an
Askoppar

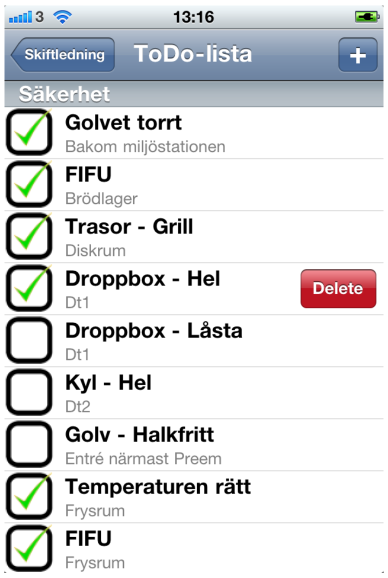
Figur 6: *Förskiftschecklista i porträtt och landskapsläge.*


3 12:00
Skiftledning Förskiftscheckli... +
Säkerhet
Bärbart larm
'Om något händer'
Tvättstuga
Tvättmedel uppfyllt
FC uppfyllt
DR uppfyllt
Helt
Rent
Ordning

Figur 7: *Markeringsalternativ i förskiftschecklistan.*

1.5 ToDo-lista

När förskiftschecklistan är genomförd genereras automatiskt en todo-lista. Den genererade listan prioriteras och sorteras efter prioritetsordningen och sektion. För att se todo-listan klicka på "ToDo-lista" i skiftledarmenyn. När en uppgift är uppföljd och klar markeras den med en grön bock genom att klicka på rutan till vänster om uppgiften. För att ta bort en uppgift helt från todo-listan svep med fingret över uppgiften från den ena sidan till den andra och tryck på "Delete"-knappen (Figur 8).

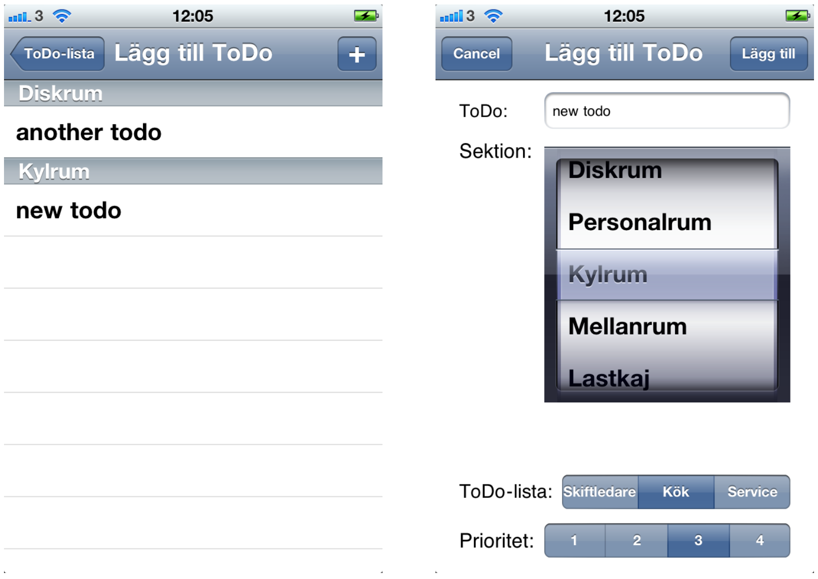


Säkerhet

- ☑ **Golvet torrt** Bakom miljöstationen
- ☑ **FIFU** Brödlager
- ☑ **Trasor - Grill** Diskrum
- ☑ **Droppbox - Hel** Dt1
- ☐ **Droppbox - Låsta** Dt1
- ☐ **Kyl - Hel** Dt2
- ☐ **Golv - Halkfritt** Entré närmast Preem
- ☑ **Temperaturen rätt** Frysrum
- ☑ **FIFU** Frysrum

Figur 8: *Ta bort en todo-uppgift.*

Finns det behov av att lägga till en ny todo tryck på plusknappen längst upp till höger. Det kommer visa vyn för tidigare egentillagda todo-uppgifter (Figur 9, vänster). Är det någon av dessa som ska läggas till tryck på den uppgiften som ska läggas till. Applikationen lägger till todo-uppgiften och tar dig tillbaka till todo-listan. För att lägga till en ny uppgift tryck återigen på plusknappen i övre högra hörnet. En ny vy visas (Figur 9, höger) fyll i namn på uppgiften, välj eller skapa en ny sektion, välj sortering och prioritet. Tryck sedan på “Lägg till” i övre högra hörnet. Applikationen lägger till todo-uppgiften och tar dig till vyn för todo-uppgifter.



ToDo: new todo

Sektion: Kylrum

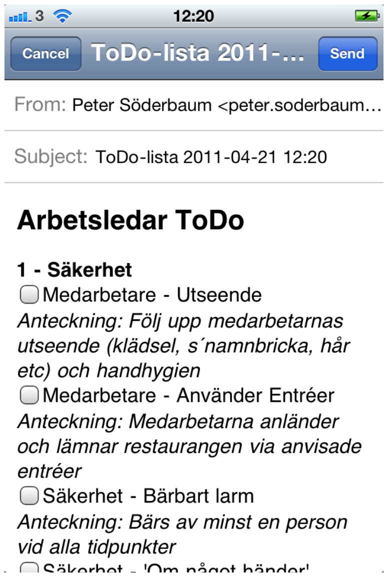
ToDo-lista: Kök

Prioritet: 3

Figur 9: *Vyer för att lägga till egna todo-uppgifter.*

1.5.1 Skriva ut todo-listan

För att kunna skriva ut todo-listan måste den skickas som ett mail till restaurangens email och sedan skrivs mailet ut. Tryck på "Maila ToDo-lista" i skiftledarmenyn. Ett mail förbereds och visas, tryck på "Send" i övre högra hörnet för att skicka todo-listan till restaurangens email (Figur 10). Todo-listan sorteras in i tre kategorier arbetsledare, kök och service inom varje kategori sorteras todo-uppgifter efter prioritet och sedan sektion. Öppna restaurangens email och skriv ut mailet.



From: Peter Söderbaum <peter.soderbaum...

Subject: ToDo-lista 2011-04-21 12:20

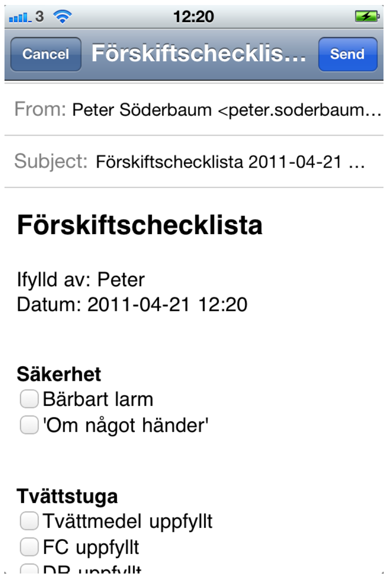
Arbetsledar ToDo

1 - Säkerhet
▢Medarbetare - Utseende
Anteckning: Följ upp medarbetarnas utseende (klädsel, s´namnbricka, hår etc) och handhygien
▢Medarbetare - Använder Entréer
Anteckning: Medarbetarna anländer och lämnar restaurangen via anvisade entréer
▢Säkerhet - Bärbart larm
Anteckning: Bärs av minst en person vid alla tidpunkter
▢Säkerhet - 'Om något händer'

Figur 10: *Vy för att maila en todo-lista.*

1.6 Avsluta skiftet

När hela skiftet är genomfört är det dags att avsluta skiftet i applikationen. Det är viktigt att vara säker på att allt är avklarat när skiftet avslutas för datan går ej att få tillbaka efter det. För att avsluta skiftet tryck på “Avsluta skiftet” i skiftledarmenyn. Det kommer upp en varning, tryck på “Ja” om du är säker på att du vill avsluta skiftet. Förskiftschecklistan visas som ett email, tryck på “Send” för att maila förskiftschecklistan till restaurangens email för arkivering och för att rensa datan från skiftet (Figur 11). Du tas till vyn för att starta ett nytt skift. Det går här att logga ut användaren genom att trycka på “Logga ut” uppe i vänstra hörnet (Figur 12).



From: Peter Söderbaum <peter.soderbaum...

Subject: Förskiftschecklista 2011-04-21 ...

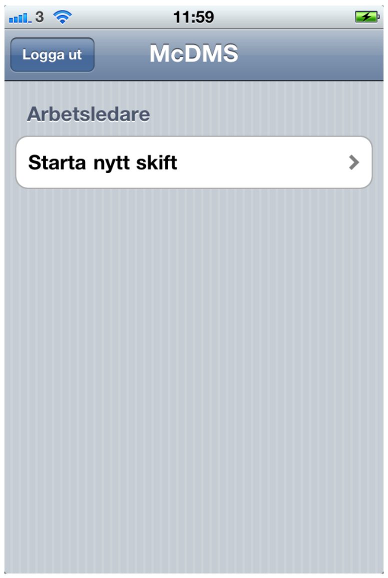
Förskiftschecklista

Ifylld av: Peter
Datum: 2011-04-21 12:20

Säkerhet
- ☐ Bärbart larm
- ☐ 'Om något händer'

Tvättstuga
- ☐ Tvättmedel uppfyllt
- ☐ FC uppfyllt

Figur 11: *Vy för att avsluta skiftet och maila förskiftschecklistan.*



Figur 12: *Vy för att starta ett nytt skift.*