Scientific paper kladd

Software engineering assignment 4

## Introduction

## Methods

## Requirements

Functional requirements

Take user input for filling checklist

Upload checklist to database

Indicate errors as alarm for supervisor

Display contents for supervisor

Let supervisor view and acknowledge alarms

Add date and current shift automatically

Allow for user to select different checklists

Non-functional requirements

Usability: User interface needs to be as simple as one on paper. It should be intuitive to select checklist and fill in the boxes.

Reliability: All data should be available for supervisor to view at any time. Checklists should be available for users to fill Friday through Sunday.

Performance: The system needs to operate smoothly and be quick in loading phase.

Supportability: The system needs to be scalable for further expansions.

A screenshot of a computer screen

Description automatically generated

Use case diagram

A diagram of a system

Description automatically generated

Domain model

There should only be one iteration of the elaboration phase. This iteration should take no longer than one week.

## Architecture

There are numerous different architectural structures that a software application can be built upon. Two-tier architectures includes two layers: a presentation layer, which in this case will be the user interface where the firefighters can interact with the checklist and the supervisor can acknowledge alarms. A data layer, database where the checklists, user information etc... will be stored.

Other structures like the tree-tier architecture could improve the system making it more robust and scalable. In the tree-tier architecture an application tier will handle the business logic. For this case that would include managing and validating user inputs, and coordinating interactions between the presentation and data tiers.

Four-tier architecture is commonly used for larger systems. Here we divide the application tier from tree-tier architecture into application and business layer. For this system that would mean the application layer handles the service with GRASP pattern control while the business layer handles the business logic.

For this particular system a two-tier architecture should be sufficient for its purpose. Keeping a simple structure will also lower the workload making it more likely that the system will be implemented.

## Chat GPT Requirements

Functional Requirements:

User Interface:

* Intuitive interface for checklist completion.
* Supervisors can review checklists.
* Support on laptops and tablets.

Checklist Management:

* Pre-made checklist selection.
* Easy completion with comments and signatures.
* Automatic inclusion of date, shift, and user info.
* Archived completed checklists.

Notifications:

* Alert supervisors of deviations.
* Email or in-app notifications.

User Management:

* Authentication and authorization.
* Admin user account management.

Non-Functional Requirements:

Performance:

* Responsive interaction.
* Minimal latency.

Security:

* Robust authentication and encryption.
* Compliance with security standards.

Reliability:

* High availability.
* Data backup and recovery.

Scalability:

* Accommodate growing users and checklists.

Usability:

* User-friendly interface.
* Clear instructions for all users.

Accessibility:

* Accessibility for users with disabilities.
* Compliance with accessibility standards.

Compatibility:

* Support for various browsers and devices.

Maintainability:

* Clean and modular code.
* Documentation for future updates.

ChatGPT provides a thorough list of requirements, encompassing both familiar elements and new additions. The functional requirements are well-detailed, offering comprehensive descriptions. However, some of the non-functional requirements may not align perfectly with the intended system operation. ChatGPT's limitations in fully understanding the system's behavior lead to some requirements being speculative. As such, while ChatGPT serves as a valuable tool for organizing and inspiring requirements, the final specifications should be crafted by the system developers.

When requested for a domain model, ChatGPT provides the model displayed in !fig. The model is short compared to the one created earlier but includes a list of attributes for each class in the domain model. Upon adding these attributes, the two domain models are remarkably similar.

A screen shot of a computer

Description automatically generated

Domain model from ChatGPT

For the elaboration phase a first version class diagram have been made to give a overview of the class structure. As shown in !fig the façade pattern is used to wrap the subsystem of the data access layer. The presentation layer will in this case include classes responsible for user interactions. !link <https://usn.instructure.com/courses/31275/pages/grasp-and-gof-patterns>

A diagram of a structure

Description automatically generated

First version class diagram

!fig, !fig, !fig and !fig shows implementation of the C# program and the user interfaces. The user adds user information in the form of employee number and selects template for checklist. The templates are stored as .csv files. This information is then used to generate a new checklist for the user to interact with. The panels, labels and checkboxes in the checklist gets automatically generated based on the content of the .csv files. This simplifies the process of adding new checklist templates in the future. A user with administrator privileges can open former checklists and acknowledge them.

Et bilde som inneholder tekst, skjermbilde, Font, nummer

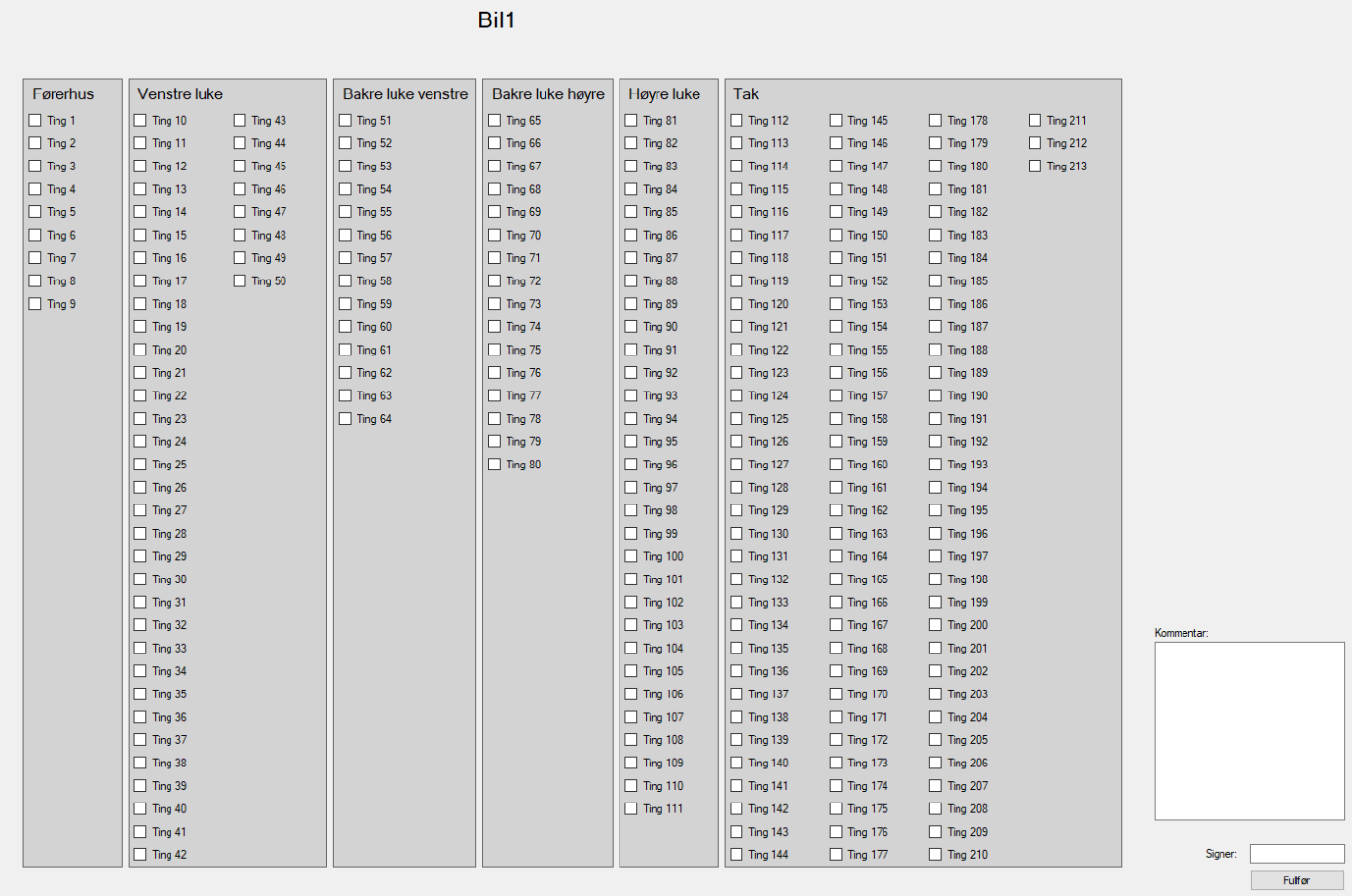
Automatisk generert beskrivelse

Login GUI (UserInfo)

Et bilde som inneholder tekst, skjermbilde, display, diagram

Automatisk generert beskrivelse

Checklist selector GUI (Template)



Checklist generated based on CSV file (Presentation layer)

Et bilde som inneholder tekst, skjermbilde, Font, programvare

Automatisk generert beskrivelse

Admin page (Presentation layer)

The data layer stores data in a Microsoft SQL Server database. The structure of the database is created with Erwin database modeler. The model from Erwin is shown in !fig

A screenshot of a computer

Description automatically generated

Erwin database model

The most important use case is “Fill checklist”. This is the origin of the system. The operator should get a user friendly and self-explanatory checklist based on a template. A fully dressed use case diagram of the use case can be seen in !fig

Et bilde som inneholder tekst, skjermbilde, Parallell, Font

Automatisk generert beskrivelse

FDUCD for case: Fill checklist