Contents

1 Vision

1.1 Introduction

Our goal is to make an interactive document sharing system, Slice of Pie, which allows multiple users to easily share and edit documents both online and offline.

1.2 Problem statement

Sharing and editing documents can be cumbersome.
Sending a document back and forth between multiple users can lead to a lot of errors. Users can overwrite what another user has done, and if they aren't all using the same text editing system this can lead to formatting issues in the document.

1.3 Summary of system features

1. Multiple users must be able to share and edit documents online.
2. Synchronization for offline usage.
3. Merging of documents.
4. History. Which allows the user to see all recent changes made to the document.
5. Documents can be categorized into folders or projects in order to get a better overview when working on a larger project with multiple files.

2 Use cases

1. UC1: Create new document
2. UC2: Edit document
3. UC3: Delete document
4. UC4: Merging documents (resolve conflict)
5. UC5: Offline sync
6. UC6: New folder
7. UC7: New project
8. UC8: Find old version of document
9. UC9: Share Document

10. UC10: Log in

Use case UC1: Create new document
Scope: Slice-of-pie application
Level: User goal
Primary actor: Regular user
Stakeholders and Interests:
- Regular user: Wants to create a new document without any complications.
Preconditions: none.
Postconditions: The document will be created

Basic flow:

1. The user needs a document that can be shared with others.
2. The user chooses a client program (web or desktop client).
3. The user presses the "New Document" button.
4. The client presents a dialogue box.
5. The user types information about the document and press OK.
6. The client sends a request to the server and the document gets stored by the server.
7. The client displays the document for the user.

Extensions:

6. The server is not responding

    (a) The document is saved locally.
    (b) The client requests the server in regular intervals to see if it has recovered.

Special requirements:
- The document should be displayed properly on any devices such as smart phones and tablets.

Use case UC2: Edit document
Scope: Slice-of-pie application
Level: User goal
Primary actor: Regular user
Preconditions: A document must have been created
Postconditions: The document must be edited the way the user desires and the change must be recorded by the server too

Basic flow:

1. The user opens a document.
2. The user edits the content of the document.
3. The client sends a request to the server.
4. The document is changed on the server.

Extensions:

3. The server is not responding

    (a) The document is saved locally.
    (b) The client requests the server in regular intervals to see if it has recovered.

Special requirements:
- The document should be displayed properly on any devices such as smart phones and tablets.

Use case UC3: Delete document
Scope: Slice-of-pie application
Level: User goal
Primary actor: Regular user
Preconditions: A document must have been created. The user trying to delete a document must have sufficient rights to delete selected document
Postconditions: The document is now deleted from the users local repository, and will be deleted when user commits

Basic flow:

1. The user selects a document
2. The user tries to delete the document 3. The client sends a request to the server
3. Server checks for authentication
4. Document is deleted

Extensions:

3. The server is not responding
    - The document is saved locally
    - The client requests the server in regular intervals to see if it has recovered
4. The user lacks the right to delete the selected document
    - User is told that he has not got the rights to delete selected document

Special requirements:

Use case UC4: Merging documents (Resolve conflict)
Scope: Slice of Pie application
Level: User goal
Primary actor: Regular user
Preconditions: A document must exist on the server and another (edited) version of the same document must exist on the stand-alone client
Postconditions: Both versions (client and server) must have matching content

Basic flow:

1. The user opens an existing document on the client.
2. The user edits the document and saves the lates changes.
3. The stand-alone client and the server are synchronized.
4. Both documents merge and contain both the original and the new content.

Extensions:

*4. There is a conflict when merging the documents.
   (a) The user resolves the conflict manually by editing the document.
   (b) The user synchronizes again.

Special requirements:

Use case UC5: Offline synchronization
Scope: Slice-of-pie application
Level: User goal
Primary actor: Regular user
Preconditions: User must be logged in. Client must be connected to the server.
Postconditions: The users offline repository will be synchronized with his online repository

Basic flow:

1. The user selects offline synchronization
2. The client sends a request to the server.
3. The server copies the user's online repository and sends it to the client
4. The client sends verification upon receiving the documents

Extensions:

2. The server is not responding
    - The user gets an error message (no connection)
4. The server is not responding
    - The client tries to re-establish a connection to the server at regular intervals, and then sends the verification.

Special requirements:

Use case UC6: Create new folder
Scope: Slice-of-pie application
Level: User goal
Primary actor: Regular user
Preconditions: none.
Postconditions: The folder will have been created

Basic flow:

1. User clicks on "create new folder" button
2. User fills in valid folder information and clicks "OK"
3. Client sends a request to server and server stores the folder
4. Client displays the new folder

Extensions:

2. Information is not valid
   (a) User gets an error message describing what information is invalid
3. Server does not respond
   (a) folder is stored locally
   (b) The client requests the server in regular intervals to see if it has recovered, then resends the request to create a folder.

Special requirements:

Use case UC7: Create new project
Scope: Slice-of-pie application
Level: User goal
Primary actor: Regular user
Preconditions: none.
Postconditions: a Project will have been created

Basic flow:

1. User clicks on "Create New Project" button
2. User fills in valid Project information and clicks "OK"
3. Client sends a request to server and server stores the Project
4. Client displays the new folder

Extensions:

2. Information is not valid
   (a) User gets an error message describing what information is invalid
3. Server does not respond
   (a) Project is stored locally
   (b) The client requests the server in regular intervals to see if it has recovered, then resends the request to create a Project.

Special requirements:

Use case UC8: View old version of document
Scope: Slice of Pie application
Level: User goal
Primary actor: Regular user
Preconditions: A document with multiple changes must exist
Postconditions: None
Basic flow:

1. A user wants to see an overview of all changes made to a document.
2. User opens a document.
3. User presses a button and is presented with a list of all changes made to the specific document.

Extensions:
Special requirements:

Use case UC9: Share document
Scope: Slice of Pie application
Level: User goal
Primary actor: Regular user
Preconditions: A document must be created in the system and at least two users must be registered.
Postconditions: The document is shared between two (or more) users.

Basic flow:

1. The users creates a new document or opens an existing one.
2. The user selects which users he wants to share the document with.
3. The user shares the document with the selected user(s).

Extensions:
Special requirements:

Use case UC10: Log in
Scope: Slice of Pie application
Level: User goal
Primary actor: Regular user
Preconditions: User must be registered in the system *(see "extensions *2").*
Postconditions: User is logged into the system.

Basic flow:

1. When the user starts the system (web client or stand-alone client) he's prompted with a login screen.
2. User must log in by typing username and password.
3. System checks if information matches what is stored in the system.
4. User is logged into the system.

Extensions:

*2. User is not registered in the system.
   (a) User is prompted with a sign up screen (web-client only).
   (b) User fills out required information for signing up.

*4. Username and password are rejected.
   (a) User gets an error message on the screen saying that the username and password don't match the system.
   (b) User gets prompted back to the login screen.

Special requirements:

3 Glossary

- **Response Time:** The time it takes for the system to respond to a request from the user.
- **Document:** A document refers to a complete document, not just a single file. A document contains a owner, id, content and a file.
- **Client:** A client can mean two things. A web client, operating directly on the server, and a stand-alone client, that runs offline, locally on the

end users machine synchronizing with the server.

- **System:** Refers to the core of the application. This includes document handler, user handler etc ..

4 Supplementary Specification (FURPS+)

- Functionality
  - The system must be able to create/edit/delete users.
  - The system must be able to create/edit/delete/share documents.
  - The system must keep a log of all document actions.
  - All system usage requires user authentication.
  - The system must support multiple users.
- Usability
  - The system must be easy to use.
    - Have a clean user interface.
    - 8 out of 10 users must be able to use the system without any training.
  - The system must be easily visible for people with "not perfect" vision. E.g no graphics that blurs the view of the core system functionality.
  - The web client must be easy and quick to navigate. No function should be more than 3 clicks (windows/sub-windows) away.
    - Not counting navigating a users files.
- Reliability
  - It must be possible to use the system without any internet connection.
    - With some limitations.
- Performance
  - The system must respond instantly
    - A request must not take more than a few (2-3) seconds.
    - Not taking external factors (such as bad internet connection) into account.

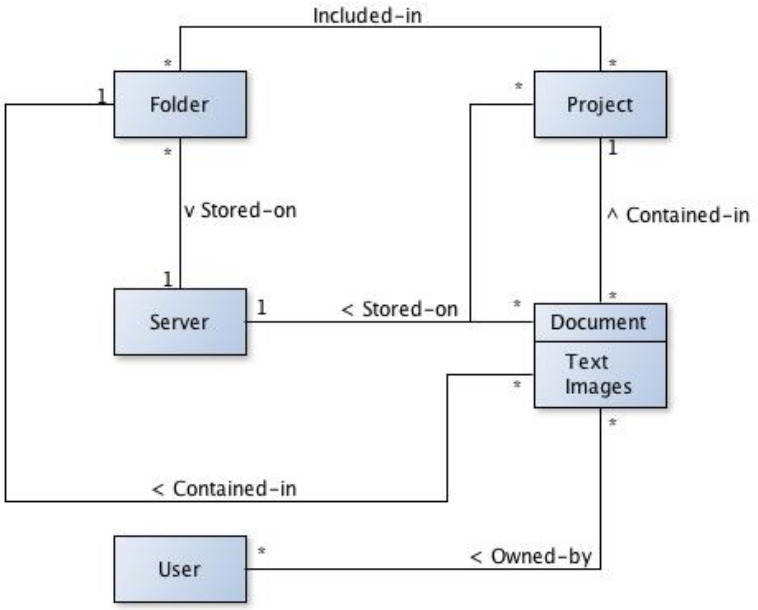
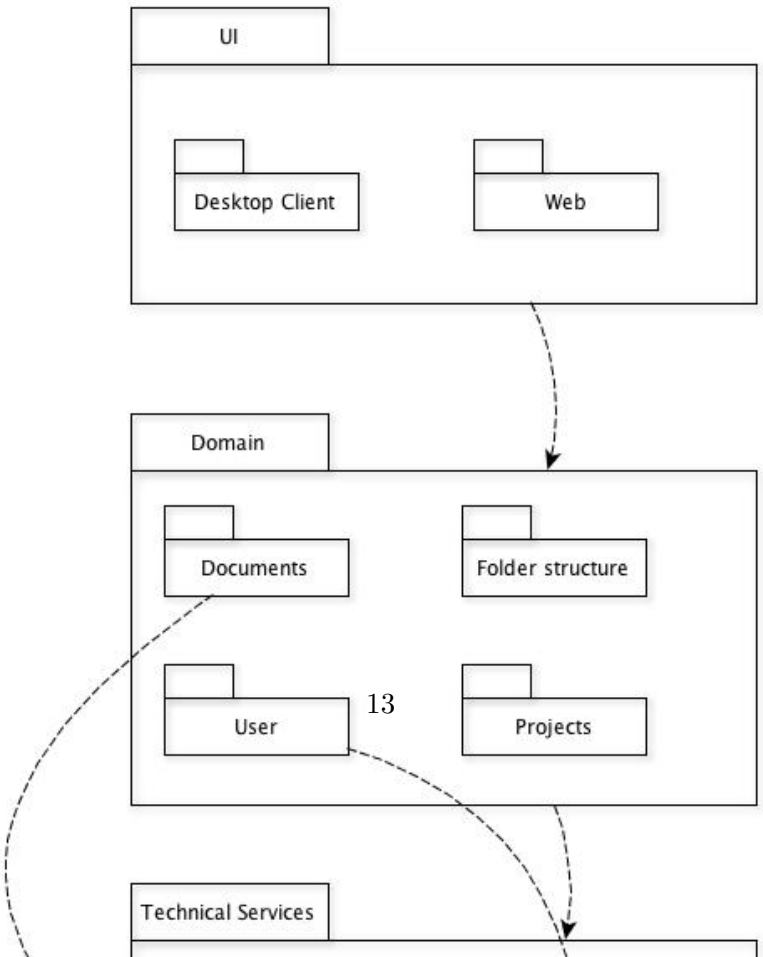


Figure 1: Domain Model

5 Domain Model

6 Logical Architecture



7 System Sequence Diagram

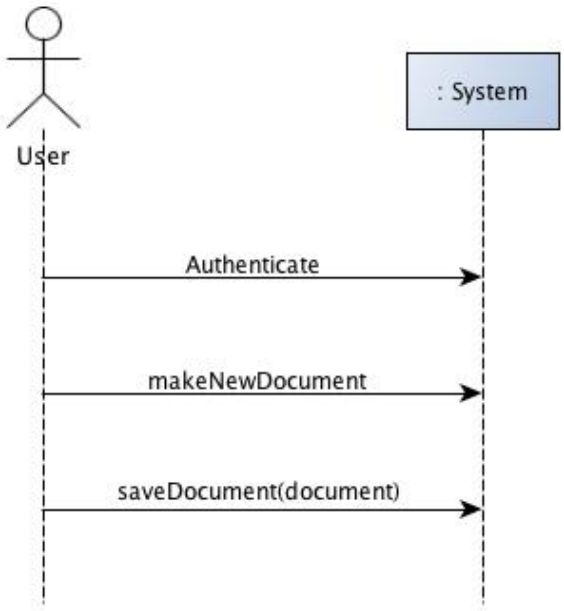


Figure 3: SSD - New document

8 Operation Contracts

Contract CO1: Synchronize **Operation:** Synchronize **Cross References:** UC1, UC2, UC5 **Preconditions:** Some documents and/or must have been created on the system. **Postconditions:** - All documents and folders on the client must be uploaded to the server. - All documents and folders on the server must be downloaded to the client. - Document version must be the same on the client and server.

9 Use Case Diagram

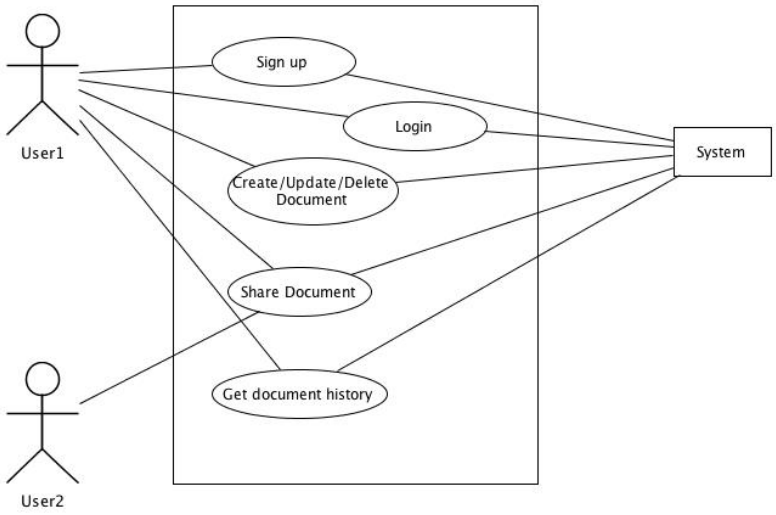


Figure 4: Use case diagram

10 UML Package Diagram

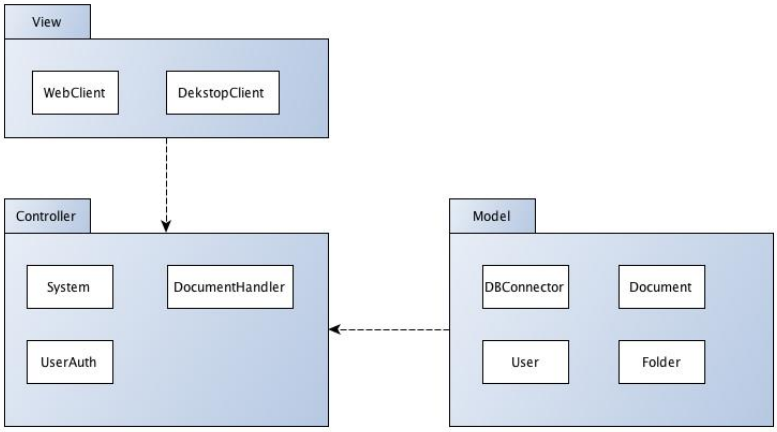


Figure 5: Package Diagram Overview

11 UML Class Diagram

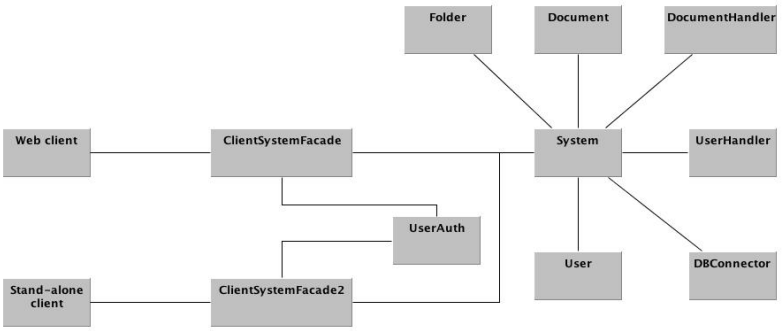


Figure 6: Class Diagram Overview

12 Communication Diagram

Diagrams/Delete Communication Diagram.jpg

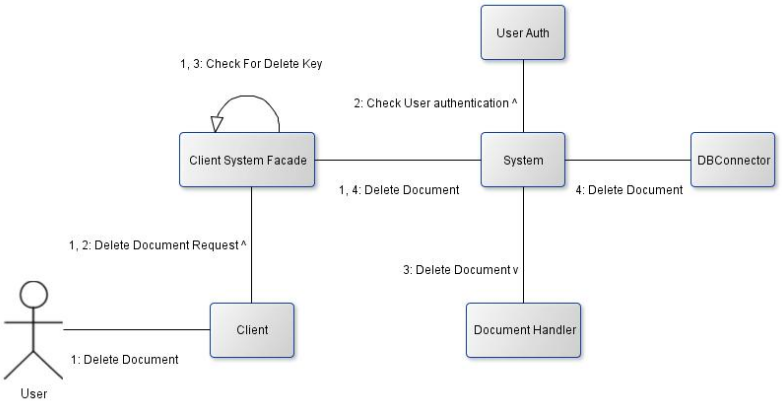


Figure 7: Communication Diagram

13 Technical Memo

Technical Memo
Issue: Files format - Which file format to use

Solution Summary: Use HTML for our file format.
Factors

- Must be able to contain both text and images.

Solution
We chose to use HTML for our file format because it's simple to construct, and can contain text and images seamlessly.
Motivation
We needed a file format that can contain images and text as well as being easy to construct. in addition, HTML can easily be extended to other content. Lastly, HTML can be opened with any browser, so the users isn't tied to SliceOfPie if he just want's to view the content of a file.
Alternatives considered
We considered using a .txt file format, but .txt can only contain plain text. We also considered using our own file format (since the format itself isn't important to the application). But if we use our own format the user is stuck with using SliceOfPie, so he can't view the content of a file with any other application.

Technical Memo
Issue: Merging two versions of the same document.

Solution Summary: Git-hub inspired merge.
Factors

- Merging two versions of the same document without overwriting existing changes.

Solution
Our merging algorithm reads the two documents and stores them, line by line in an array.
Then the algorithm compares each line in the two arrays, if the lines are the same, insert the line into a new array. If the two lines aren't identical, insert the new line into the new array + insert the line from the old array in the next line. This line will be encapsulated with <<< TEXT >>> which shows the user where there is a conflict which the user can solve later on.
If the new version of the document contains lines that aren't in the old array, they are simply added to the new array.
Motivation:
There are other, more advanced, merging algorithms available. Because of time constraint we chose to use this one. It isn't the most advanced/complete algorithm but it does the job quite well considered it's simplicity.
Unresolved issues:

- Our algorithm doesn't 100% solve the conflict. In the end the user must manually chose which version to keep, and which version to discard.
- If two identical lines exists in both versions but the lines is at another line number in the old document, this might cause a conflict <<< TEXT >> that could be avoided.

Alternatives considered
An algorithm that analyses every line in the file keeps the one that the user wants.

14 ER-Diagram

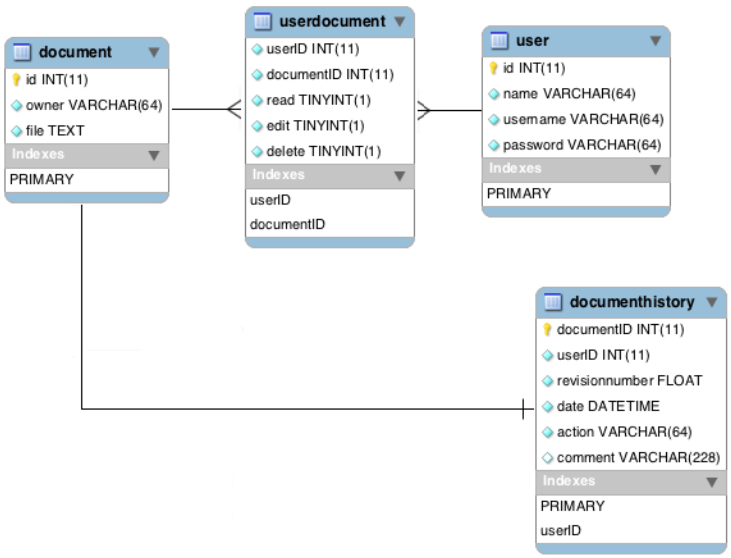


Figure 8: ER-Diagram

15 Slice of Pie user manual

15.1 Starting the application

15.1.1 Running the application from Visual Studio

Before starting the application, you need to start Visual Studio with administration priviledges. The reason for this is that the application will need to create files and folders for the documents and in order to do so the program needs to be run with administrator priviledges which will give the application write access.
Since the web client runs in the browser, the browser needs to allow pop up windows for localhost. This can be done when the application is run for the first time.
When starting the application, you need to set the WebClient project as startup project (if it's not already set), and then run the program (f5 for debug mode, ctrl + f5 for the release version).
The web client will start up with your default internet browser. When the main page has loaded, you are ready to use the application.

15.1.2 Signing up

As a first time user there won't be any user registered in the system, so the first time you need to do is to sign up to use the system.
To sign up, click the "Sign up" button. This will open up a new window with the sign up form.
Fill out the form and click the "Sign up" button. A message will appear to say if the signup was successfull or not.
If the signup was successfull you are now registered in the system, and are ready to use it. Close the sign up window and go back to the main window.

15.1.3 Logging in

On the main screen there are two text boxes at the top of the window named "username" and "password". Enter the newly created username and password into the boxes and click the "Login" button.

15.2 Using the system

After you have logged in, press the "Get Files" button on the left page of the window. This will show you all the files that belong to the current user. Since you are a new user you don't have any documents, so you should only see the root folder (the one with your username).

15.2.1 Create a new document.

To create a new document, click the "New Document". This will clear all text boxes, and you are ready to write a new document.
Creating a new document doesn't save the document, so before you go too far you should save your document. Write a filename in the filename box, and click the "Save Document" button. If you wish to save the document in a sub folder, just write the: "foldername/filename.html" in the filebox.
The system doesn't require that you save the document as a HTML file, but the system is built around it. Not doing so won't make it able to add images to you document.

15.2.2 Deleting a document

Deleting a document is very simple. Select a document from the list on the left. Make sure that the filename of the file is entered into the filename box (this can also be done manually). Delete the document by clicking the "Delete Document" button.

15.2.3 Sharing a document

Sharing a document is very simple as well. Open the document you wish to share. Enter the username of the user you wish to share the document with in the text box next to the "Share Document" button, and click the share document button.

15.2.4 Showing a document

Since the document is built around the HTML format, the text area can't show any images or text formatting. In order to see the document (with images, formatting etc) you need to open it in another page in the browser. Select the document you wish to view and click the "Show Document" button. This will open a new window with your document in parsed HTML.

16 Revision Table

Table 1: Revision Table

Revision	Changes	Section	Author
21-11-12	Created Vision	1	Bergar
21-11-12	Created Use cases	2	Bergar
21-11-12	Use case 1 and 2	2	Filip
21-11-12	Created Glossary	3	
21-11-12	Created Supplementary Specifications	4	Bergar
21-11-12	Created Domain Model diagram	5	Bergar and Filip
21-11-12	Created Logical Architecture diagram	6	Bergar and Filip
22-11-12	System Sequence Diagram	7	Bergar
22-11-12	Operation Contracts	8	Bergar
22-11-12	Use cases 3, 4, 6 and 7	2	Morten
22-11-12	Use case diagram	9	Filip
23-11-12	UML Package Diagram	10	Bergar
23-11-12	UML Class Diagram	11	Bergar
27-11-12	UML Communication Diagram	12	Morten
04-12-12	Use case 10	2	Bergar
04-12-12	Update class diagram	12	Bergar
04-12-12	Technical memo for file format	13	Bergar
04-12-12	ER-Diagram	14	Bergar and Morten
16-12-12	Use case 4, 8 and 9	2	Bergar
16-12-12	Updated supplementary requirements	4	Bergar
16-12-12	Updated use case diagram	9	Bergar
16-12-12	Merging Technical memo	13	Bergar
16-12-12	Updated class diagram	11	Bergar
16-12-12	Application user manual	16	Bergar