# Requirements and Analysis Document for MailBowser

Version: 2.0

Date: 2015-05-31

Authors: Jesper Jaxing, Oscar Evertsson, Mats Högberg, Filip Hallqvist

**This version overrides all previous versions.**

## 1 Introduction

The application will be a mail client for desktops written in Java.

### 1.1 Purpose of application

The project aims to create a unique mail client. Unique in the sense that mails will be written in the markup language *Markdown* (for more information about Markdown, see *Markdown* by Gruber (2014)).

### 1.2 General characteristics of application

The application will be a desktop, standalone email application, available on both Windows, Mac OS and Linux with a graphical user interface.

It will be possible to add multiple email accounts to the application. All emails from each account will be collected to a single mailbox. When an email is fetched from the server it will automatically be tagged with the name of its folder.

A tag will be an implementation of its real world use case, used for marking and sorting different emails. An email can be tagged with multiple tags, and many emails can have the same tag. The tag system will be essential for the application as it will be the only way of organizing emails. It will be possible to filter the emails by their tags.

The application will also offer search functionality based on the subject, content, receivers and sender of emails.

The application will have functionality to send email. When sending an email you will be able to state recipients of different types (to, cc and bcc), which one of the registered accounts to

send from, a subject and a message body. It will be possible to write the message body in Markdown, and to view a preview of the compiled HTML in real time. When sending an email written in Markdown, only the compiled HTML will be included in the message body.

The application will feature a contact book with contacts. A contact will have a name and a list of email addresses. It will be possible to add new contacts, delete existing contacts, and edit existing contacts. The contact book will also have search functionality for quickly finding specific contacts.

## 1.3 Scope of application

Contrary to many other mail clients, the application will not offer a WYSIWYG-editor. The application code will, apart from fetching and sending emails, only make local changes. This means that emails that are deleted in the application, still remain untouched on the server. The tags will only be stored locally as well.

## 1.4 Objectives and success criteria of the project

1. It should be possible to send, receive, and browse through emails.
2. It should be possible to write emails in Markdown and preview the compiled HTML.
3. It should be possible to categorize emails with tags.
4. It should be possible to use multiple accounts.
5. It should be possible to search for emails containing specific words
6. It should be possible to store contacts in a contact book

## 1.5 Definitions, acronyms and abbreviations

- Markdown — A type of markup language used for formatting text. Markdown is compiled to regular HTML.
- Compile — To convert content from one form to another.
- HTML — A markup language used primarily for the web, but also in emails.
- WYSIWYG — What-you-see-is-what-you-get.
- WYSIWYG editor — A type of content editor where you format text using controls like the ones found in office applications.
- GUI — Graphical User Interface.
- Java — Platform independent programming language.
- JRE — The Java Runtime Environment. Additional software needed to run a Java application.
- User — The person using the application.
- Recipient — A person that receives an email. An email can have several recipients.
- Sender — The person that sends an email. An email can only have one sender.
- Contact — An address that you have added to your contact list.
- Address — An email address, e.g. name@domain.com.

- Email — Electronic mail. A method of exchanging digital messages from a sender to one or more recipients.
- Tag — A tag that categorizes an email. An email can have many tags, and many emails can have the same tag.
- Account — A class that contains or is related to all the data needed to send or receive emails via one specific account on a mail server.
- Fetch — Get emails from a mail server.
- Send — Send an email to its recipients.
- Asynchronous — Performing tasks in a new thread.

# 2 Requirements

## 2.1 Functional requirements

1. Compose email
    a. State recipient addresses
    b. Choose which account to send from
    c. Specify a subject
    d. Write the email in Markdown and preview the compiled HTML in real time
2. Send email
3. Receive emails
    a. Both manual and automatic fetching of new emails that haven't been fetched before
    b. Manual refetching of all emails on the server
4. Browse through and read emails
5. Tag emails
    a. See all emails that have a specific tag
    b. See all tags that a specific email has
    c. Add and remove tags from a specific email
    d. Remove specific tags from all email
6. Contact functionality
    a. Add contact
    b. Edit contact
    c. Delete contact
7. Reply to email
    a. Reply to just the sender
    b. Reply to the sender and all other recipients
8. Forward email
9. Delete email

## 2.2 Non-functional requirements

1. The application should be easy to use for anyone, but it should be more suited for people used to writing code.
2. The application should let the user create an email very quickly and effective.
3. The application should be secure.
4. The user should feel in control of the application, and not the other way around.

### 2.2.1 Usability

Usability is high priority. The application will have a minimalistic and easy to understand GUI. It is important that the user understands what's happening in the program, and that the user

always feels like he or she is in control. An example of this is that the application will have buttons for manually fetching emails from the server, in addition to the automatic fetching happening in the background.

### 2.2.2 Reliability

As the application will be handling private data, it is very important that it is trustworthy and also expresses this to the user. To fulfill this, the application encrypts sensitive account details before storing them.

### 2.2.3 Performance

The application will only fetch emails that haven't been fetched from the server before. This is very beneficial to network performance, as it would take a long time to fetch all emails on every fetch. As with any application, performance is critical to keeping the application user friendly. It is important the the application feels snappy.

Every request to outside sources will be made asynchronous, so as to not freeze the GUI. Practical examples will be when sending and fetching emails.

### 2.2.4 Supportability

The application will be supported on all of the major operating systems, including Windows, Mac OS X and Linux.

There should be automated tests verifying all functionality. Code related to the GUI will be tested automatically insofar as it is possible. It could also be tested manually.

### 2.2.5 Implementation

The application will be a stand-alone JAR-file, packed with all the libraries required for it to run.

### 2.2.6 Packaging and installation

See 2.2.5.

### 2.2.7 Legal

The "Bowser" name, game character, and other trademarks are property of Nintendo of America Inc (www.nintendo.com). MailBowser is built for educational purposes only and not meant to be a source of income.

## 2.3 Application models

### 2.3.1 Use case model

See appendix for UML diagrams and textual descriptions.

### 2.3.2 Use cases priority

1. Create email
2. Send email
3. Read email
4. Add account
5. Reply to email
6. Forward email
7. Search
8. Delete email
9. View contacts
10. Add contact
11. Delete contact
12. Edit contact
13. Insert contact into mail

### 2.3.3 Domain model

See appendix.

Email, Account and Tag will be persistent. Every time the application is closed the current state of them will be saved, and when the application starts the saved state will be loaded.

### 2.3.4 User interface

See appendix for GUI sketches.

The different parts are for:

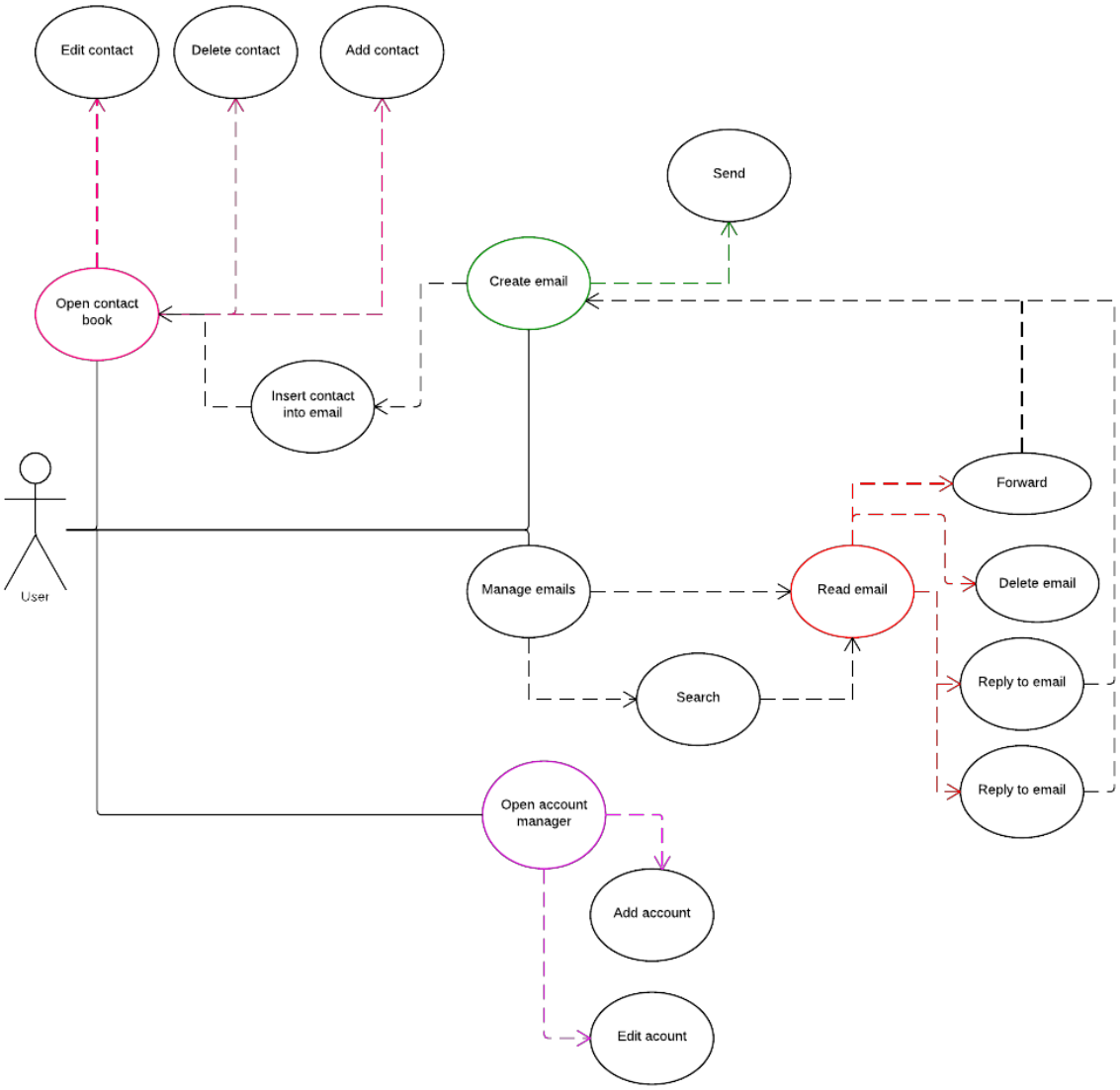
1. A list of all available tags. When a tag is selected all emails tagged with that tag will be shown in 2.
2. A list of all relevant emails. When an email is selected its content will be shown in 5.
3. Action panel with buttons for frequently used commands, like “Fetch”, “New Email”, “Reply” and “Forward.
4. A search field. Emails matching the search query will be shown in 2.
5. The main content area for displaying an emails content.

### 2.4 References

Gruber, John. (2014) *Markdown*
http://daringfireball.net/projects/markdown/

# Appendix

## Use cases:



**Use case texts:**

# Use Case: Open account manager

**Summary**: How the users open the account manager

**Priority**: Medium

**Extends**: -

**Includes**: -

**Participators**: User

## Normal flow of events

1. User opens the account manager

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Clicks on file menu | |
| 1.2 | | Presents menu alternatives |
| 1.3 | Clicks “Accounts…” | |
| 1.4 | | Opens account manager |

# Use Case: Add account

**Summary**: How the users adds an email account.

**Priority**: High

**Extends**: -

**Includes**: Open account manager

**Participators**: User

## Normal flow of events

1. User adds an account to the application

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Clicks add account | |
| 1.2 | | Opens add account setup |
| 1.3 | Fills required input fields | |
| 1.4 | Clicks save | |
| 1.5 | | Creates account with input info |

## Alternative flow of events

1.5 Input is invalid (*excluded*)

| | **Actor** | **System** |
|---|---|---|
| 1.6.1 | | Notifies the user that the input is invalid, and marks the affected fields |

1.6 User clicks cancel

| | **Actor** | **System** |
|---|---|---|
| 1.6.1 | | Exists the add account window |

# Use Case: Edit account

**Summary**: How the users edits an added email account.

**Priority**: Low

**Extends**: -

**Includes**: Open account manager

**Participators**: User

## Normal flow of events

1. User edits an an existing account

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Selects account in the list | |
| 1.2 | | Opens edit view for chosen account |
| 1.3 | Updates info | |
| 1.4 | Clicks save | |
| 1.5 | | Saves account with input info |

## Alternative flow of events

1.5 Input is invalid (*excluded*)

| | **Actor** | **System** |
|---|---|---|
| 1.6.1 | | Notifies the user that the input is invalid, and marks the affected fields |

1.6 User clicks cancel

| | **Actor** | **System** |
|---|---|---|
| 1.6.1 | | Discards all input |
| 1.6.2 | | Exists the edit account view |

# Use Case: Create email

**Summary**: This is how the user creates an email.

**Priority**: High

**Extends**: -

**Includes**: -

**Participators**: User

## Normal flow of events

1. The user creates a new email.

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Clicks “new email button” | |
| 1.2 | | Opens edit window |
| 1.3 | Starts adding receiver(s) | |
| 1.4 | | Suggests contacts to add (*excluded*) |
| 1.5 | Adds receiver(s) | |
| 1.6 | Writes email in markdown | |
| 1.7 | | Compiles markdown and shows preview |

## Alternate flows

1.4.1. There are no contacts to suggest (*excluded*)

| | **Actor** | **System** |
|---|---|---|
| 1.4.1.2. | Finishes typing the address | |

1.5.1 Not a valid email (*excluded*)

| | **Actor** | **System** |
|---|---|---|
| 1.5.1.1. | | Informs about invalid email |

| | | |
|---|---|---|
| 1.5.1.2. | tries to correct email | |

# Use Case: Delete email

**Summary**: This is how the user deletes an email.

**Priority**: Medium

**Extends**: -

**Includes**: Read email

**Participators**: User

## Normal flow of events

2.2 User deletes email

| | **Actor** | **System** |
|---|---|---|
| 2.2.1 | Presses delete button | |
| 2.2.2 | | Adds the email to deleted tag |
| 2.2.3 | | Removes other tags on the email |

## Alternative flow of events

2.2.3.1 No next email (*excluded*)

| | **Actor** | **System** |
|---|---|---|
| 2.2.3.1.1 | | Notifies the user that the current tag has no emails. |

# Use Case: Forward email

**Summary**: This is how the user forwards an email.

**Priority**: Low

**Extends**: -

**Includes**: Send email

**Participators**: User

## Normal flow of events

2.3 User forwards email

| | **Actor** | **System** |
|---|---|---|
| 2.3.1 | Selects an email | |
| 2.3.2 | Presses forward button | |
| 2.3.3 | | Include create email with current email data |
| 2.3.4 | | Includes send email |

# Use Case: Read email

**Summary**: This is how the user reads an email.

**Priority**: High

**Extends**: -

**Includes**: -

**Participators**: User

## Normal flow of events

1. User reads an email without taking any action

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Selects an email | |
| 1.2 | | Opens email and presents in read view |
| 1.3 | | Mark as read (*excluded*) |

## Alternate flows

1.3 Already marked as read (*excluded*)

| | **Actor** | **System** |
|---|---|---|
| 1.3.1 | | Nothing happens |

# Use Case: Reply email

**Summary**: This is how the user replies to an email.

**Priority**: Medium

**Extends**: -

**Includes**: Create email, Send email

**Participators**: User

## Normal flow of events

2.1 User replies to email

| | **Actor** | **System** |
|---|---|---|
| 2.1.1 | Selects an email | |
| 2.1.2 | Clicks reply button | |
| 2.1.3 | | Includes create email with current email data. |
| 2.1.4 | | Includes send email |

# Use Case: Reply all email

**Summary**: This is how the user replies to an email.

**Priority**: Medium

**Extends**: -

**Includes**: Create email, Send email

**Participators**: User

## Normal flow of events

2.1 User replies to email

| | **Actor** | **System** |
|---|---|---|
| 2.1.1 | Selects an email | |
| 2.1.2 | Clicks reply all button | |
| 2.1.3 | | Includes create email with current email data. |
| 2.1.4 | | Includes send email |

# Use Case: Search for email

**Summary**: This is how the user searches for email.

**Priority**: Low

**Extends**: -

**Includes**: Browse

**Participators**: User

## Normal flow of events

1. User searches for a specific email

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Fills the search input with keywords to search for | |
| 1.2 | | Shows results |
| 1.4 | Browses result | |

## Alternative flow of events

1.2 No results found (*excluded*)

| | **Actor** | **System** |
|---|---|---|
| 1.2.1 | | Notifies the user that no results were found. |

# Use Case: Send email

**Summary**: This is how the user sends an email

**Priority**: high

**Extends**: -

**Includes**: Create email

**Participators**: User

## Normal flow of events

User have an email he/she wants to send.

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Presses send | |
| 1.2 | | Sends email to receiver(s) |

## Alternate flows

1.2.1. Email address does not exist (*excluded*).

| | **Actor** | **System** |
|---|---|---|
| 1.2.1.1 | | Notifies the user by email addres that the email addess does not exist |

1.2.2. User doesn't have any internet connection (*excluded*).

| | **Actor** | **System** |
|---|---|---|
| 1.2.2.1 | | Notifies the user of missing internet connection |
| 1.2.2.2 | | Asks the user if it wants to try to send it later, save it as a draft or delete the email |
| 1.2.2.3 | Chooses an operation | |

| | | |
|---|---|---|
| 1.2.2.4 | | Executes chosen operation |

# Use Case: Open contact book

**Summary**: This is how the user views the stored contacts.

**Priority**: Medium

**Extends**: -

**Includes**: -

**Participators**: User

## Normal flow of events

User opens the contact book from the main view.

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Presses contact book button | |
| 1.2 | | Opens contact book |

# Use Case: Add contact

**Summary**: This is how the user adds a contact.

**Priority**: Medium

**Extends**: -

**Includes**: Open contact book

**Participators**: User

## Normal flow of events

1. User wants to add a contact.

| | **Actor** | **System** |
|---|---|---|
| 1.1 | | Includes open contact book |
| 1.2 | Presses add contact button | |
| 1.3 | | Populates the contacts list with a new list item |
| 1.4 | | Selects the new list item and includes edit contact with no contact information |

# Use Case: Delete contact

**Summary**: This is how the user deletes a contact.

**Priority**: Medium

**Extends**: -

**Includes**: -

**Participators**: User

## Normal flow of events

User wants to delete a contact.

| | **Actor** | **System** |
|---|---|---|
| 1.1 | | Includes view contacts |
| 1.2 | Selects contact in the contacts list | |
| 1.3 | | Populates the right hand side of the contact book with a form containing chosen contact's name and address(es) unless non-existent. |
| 1.4 | Presses delete contact button | |
| 1.5 | | Deletes the chosen contact and clears the form on the right hand side of the contact book. |

# Use Case: Edit contact

**Summary**: This is how the user edits a contact.

**Priority**: Medium

**Extends**: -

**Includes**: -

**Participators**: User

## Normal flow of events

User wants to edit a contact.

| | **Actor** | **System** |
|---|---|---|
| 1.1 | | Includes view contacts |
| 1.2 | | Populates the right hand side of the contact book with a form containing selected contact's name and address(es) |
| 1.3 | Edits the form content and thereafter presses the save button | |
| 1.4 | | Saves the contact |

## Alternate flows

1.3.1. User adds an address field.

| | **Actor** | **System** |
|---|---|---|
| 1.3.1.1 | Presses add address button | |
| 1.3.1.2 | | Adds an address field below the other fields |
| 1.3.1.3 | | If the remove address button was disabled, enable it |

1.3.2. User removes an address field.

| | **Actor** | **System** |
|---|---|---|
| 1.3.2.1 | Presses remove address button | |
| 1.3.2.2 | | Removes the most recently added address field |
| 1.3.2.3 | | If no more address fields exists, disable the remove address button |

# Use Case: Insert contact into email

**Summary**: This is how the user inserts a contact as receiver of an email.

**Priority**: Medium

**Extends**: -

**Includes**: Create email

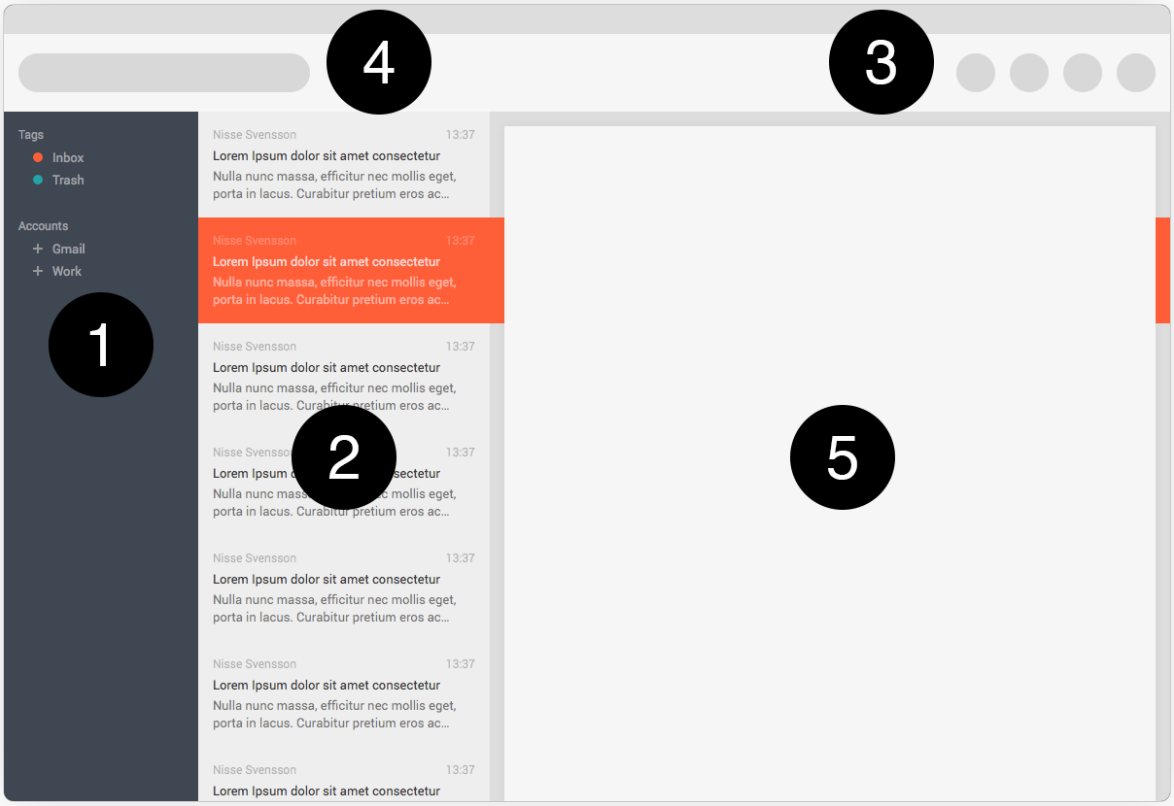
**Participators**: User

## Normal flow of events

User wants to edit a contact.

| | **Actor** | **System** |
|---|---|---|
| 1.1 | Presses create email button, reply button, forward email button or reply all button | |
| 1.2 | | Include create email with current email data, or no data if the create email button was pressed |
| 1.3 | Presses contact book button | |
| 1.4 | | Opens contact book |
| 1.5 | Chooses contact in the contacts list | |
| 1.6 | | Populates the right hand side of the contact book with a form containing chosen contact's name and address(es) |
| 1.7 | Presses the insert contact button | |
| 1.8 | | Inserts the contact as receiver of the email |

## GUI:




## Domain model: