Table of Contents

1. Introduction

1.1. Purpose

This document serves as a foundation for the further development of the MujMail project. It contains requirements specifications that define the capabilities to be added to the application. Additionally, the document defines interfaces between individual modules of the application.

1.2. Application description

Mobile application is the base part of MujMail project. Its goal is to enable a user to access his/her mail account(s) on a mobile device supporting all the necessary functions, such as reading mails, writing mails, mail forwarding, polling, mail folders and flags manipulation, searching mails downloaded to the device, etc.

1.3. Design

This document contains tables describing interfaces. This information will be used during development, to synchronize between developers.

1.3.1. Interfaces

Each interface table contains methods of one specified class. Description of each method contains method parameters and its return value type (indicated with a slash before type name).

1.4. Overview of the following document

Chapter 2 introduces the project: its goals, competition it faces, history of mujMail, further improvement possibilities and describes the progress of work on the project. Chapter 3 describes the functionality of mujMail which is common to all parts of the code, e.g. multiple languages support and exceptions handling. Chapter 4 defines basic requirements for mujMail mobile and server parts. Next, Chapter 5 defines the requirements for unified data storage and describes the interfaces. In Chapters 6 and 7, network and file system communication, and searching mails are defined, respectively. Chapter 8 deals with user folders. Threading mails is discussed in Chapter 9. Chapter 10 describes the interfaces specified for HTML content display. Finally, Chapter 11 introduces the requirements for tasks definition, execution and management.

2. Project overview

This chapter presents a short introduction to the goals this project is designed to achieve and gives a quick overview of other solutions which aim to solve the same goals.

2.1. Project goals

With the growing accessibility of mobile devices the need for the software designed for these devices increases too. Emailing have became a usual way of communication and as using internet on mobile phones becomes more popular the need for mobile email clients grows too. However, there is a still small number of email clients designed for mobile devices. Thus, there is even smaller number of freeware or user-friendly mobile email clients. Although newer mobile devices have built-in email client applications, they are not open-source and thus, in case those applications have some errors a user does not have the possibility to quickly fix it or enhance it.

The most common software platform for mobile devices is J2ME (Java Micro Edition) from Sun Microsystems. Even the modern PDA devices and smartphones have the support for Java, thus enabling our mobile application to be used on those devices too.

The goal of the project is to enhance MujMail mobile mail client with the features that give the user the possibility to fully control his email accounts including message flags and folders manipulation, synchronize his MujMail settings, communicate with wider range of mail servers through enabling encrypted communications, view his mails as conversations, search mails in mobile device and many more.

2.2. Project competition

There exist a few mobile mail clients, which perform similar tasks as MujMail. The overview of these applications is given in this section.

2.2.1. Gmail

Gmail client (www.google.com/mobile/default/mail.html) is developed by Google for users with Gmail account. It is a user-friendly mail client with very attractive interface. It supports multiple Gmail accounts, saving mails as drafts to send them later, shortcut keys, message autorefreshing, autocompleting addresses, attachments viewing and others. Unfortunately, it has a big con: it is designed only for Gmail accounts. Other mail servers are not supported.

2.2.2. mMail

Polish email client developed by Xtend new media Sp. z o.o. is called mMail (www.mmail.pl). It is able to send and receive simple plain-text email messages. HTML-based emails are converted on the server to plain-text. Images, scripts and attachments are cut. It supports English, German and Polish languages. Unlike Gmail, mMail supports multiple accounts. The application is shareware. The main disadvantage of the application is that it does not support IMAP protocol.

2.2.3. Mailwithme

Mailwithme (www.mailwithme.com) is a mobile email client which supports POP3/IMAP protocols possibly with SSL encryption. In order to function, Mailwithme needs to communicate with its server: a user needs to create an account on the application web. Mailwithme server is used to store mail accounts configuration, to support different email protocols, to preprocess images in order to reduce GPRS consumption and to reduce mobile software complexity to allow service on small devices. The disadvantage of this application is that it does not function without Mailwithme server.

2.2.4. MovaMail

MovaMail (www.movamail.com) is structurally very similar to Mailwithme mail client. It also needs a server to function. This allows a user to download emails from even webmail servers like Yahoo and Hotmail for the price of having an intermediate server. Its key features are support for multiple email accounts, address book integration and synchronization, full support for image attachments, cameraphone support and support for multiple languages.

2.2.5. vimoMail

vimoMail mail client (www.vimomail.com) allows access to user accounts using either POP3 or IMAP protocols. Additionally, it supports SMTP protocol for sending mails, viewing URLs embedded in emails, and to quickly manage user inbox with just a few key presses. It has presets for Gmail, Yahoo Mail and AOL mail. vimoMail address book grows while working with the application, i.e. automatically adds mail addresses to the book. Also, it is possible to switch mailboxes using IMAP protocol. One of the biggest disadvantages of the application is the lacking support for attachments, too few settings entries (i.e. downloading only limited amount of mails), impossibility to stop unread mails retrieving process.

2.3. History of MujMail

MujMail project have been developed for a long time. Firstly, the project was introduced as open-source software in 2003 by Peter Spatka. Then, after some time our project supervisor, Pavel Machek joined the project. The application was designed as a very simple mobile email client allowed working with emails with POP3 and SMTP protocols.

Later, in 2005, two students of Charles University in Prague, Martin Stefan and Nguyen Son Tung have joined to the project. They decided to rewrite MujMail and designed it such way that it complied with their project requirements. Thus, MujMail was rewritten and very little percent of old code from 2003 was left.

The new version of MujMail developed by this project was based on the latest version 1.07. This document specifies the parts of the application developed by our project. Each part which was not fully created by our project will be explicitly commented.

2.4. Future development possibilities

Many new functionalities were added during the development of mujMail 1.08, the whole source code was refactored. All these changes unavoidably increase the size of the application dramatically. Additionally, large size of the application decreases available operating memory at runtime. Typically, unpacked class files, stack and heap share one address space. It makes extending mujMail with a new functionality a really complex task. Thus each new improvement to mujMail had to be implemented with the size of available operating memory in mind.

The next versions of mujMail should have full support for touch screen devices in order to stay up to date. As our project mainly added new features into mujMail, further, the feature-adding should be stopped for some time and the focus should be shifted to stability improvement. Because mujMail has quite larger user base it seems to be a reasonable approach for further development.

Other direction in which the future development can go is to make mujMail even smaller and faster. There is enough space to solve memory inefficiency and thus to improve speed and decrease mujMail size.

Much more useful work can be done in the server part, because then it will be possible to deliver new functions to all users. New version of mujMail can be considered as some base where new functionality can be added. Attachment conversions from even more file types can be supported by the server. Additionally, searching mails is possible only between the mails downloaded to mujMail. But this way only a random subset of the whole user mailbox can be searched. By moving the mails searching to the server this shortcoming can be avoided.

Finally, mujMail should be further maintained as it has a big base of users. The users will be the source of new features. The implementation of mujMail startup lock and basic touch screen support are the examples of such features, which were added to mujMail beyond the specification.

2.5. Development progress

The development progress of mujMail 1.08 can be divided into three phases. In each phase the different tasks are done and the intensity of the work is different too. Shortly the development progress can be described as: creating the team of developers, studying how mujMail works, thinking about extension possibilities, implementing individual features, and last, overall regression testing and writing a documentation.

During the whole development period mujMail user support task was performed and is still performed. All team members have done user support which was based on monthly rotations. During the development also the web pages of the application have been updated, new versions and localizations have been released. Status meetings were held each week.

Creation of the team can be considered as a zero phase. It begun with a user support and looking into mujMail documentation, source comments in order to understand the mujMail functionality. In this phase the team tried to understand how mujMail works, its coding style. As all team members were seeing mujMail for the first time the team was trying it and probing its possibilities. At that time, the web pages were moved to another location and private SVN repository for the work was created.

First phase of development came after. An official specification was created. The possible ways of enhancing mujMail were analyzed and the work to be done has been divided between team members. Each of the developer presented their vision on how to implement the features assigned to them. Also, the team consulted the specification and the suggested solutions with the team lead. Users' needs were taken into consideration while writing the specification. As many users complained about SSL certificates issue, this problem has been implemented first.

In that phase also the server part solution was widely discussed. Discussion was whether to implement a proxy server, which serves as a proxy to mail servers or to implement a standalone server which would directly contain user mails. The latter solution was chosen as it reduces the complexity of the server, and eliminates the need for a central proxy server for all users. The first phase was symbolized by primal eagerness and more intensive work.

Main attribute of the second phase was the separate development. Regular status meetings have been held. Each team member presented the work he made. At that time the problem in team had risen, because one of the team members was leaving the university. New team member had to be found as the minimum team size is four people. The labor division had to be remodified. As the new member had less time to do the work, his tasks were shortened. As soon as the new team member has been found the project was restarted.

During the second phase file system support was added as first feature. Main challenge while implementing this feature was in supporting users without file system extension (JSR75). Because those users cannot simply import these classes, mujMail would not be able to run on mobiles without JSR75 support and such mobiles are not rare even now

and have to be supported (Nokia 6230) as specified. At that time the team thinks about using a preprocessor. But the discovery was, that J2ME supports dynamic class loading and after some testing when the team ensured that this construction works on mobile phones without file system the team started with the implementation. File system browser had to be created to be able to select folders and files.

Then, this functionality was used to implement the feature of storing downloaded attachments into file system. In parallel with storing attachments feature the work started on saving and restoring configuration. Because the team wanted to store configuration into a file the same way as on server the team needed to have file system access implemented.

Sending emails with attachments feature was the next task to implement. Mail sending routines in client had to be modified in order to be able to generate multipart messages and read data from file system of the mobile device. Also, the sending form had to be modified in order to show the attachments and add commands for working with them. With attachments support the team had to refactor the code which was responsible for mail databases manipulation in order to be able to use the file system as the same source of data as RMS databases, i.e. there was the need to have a unified storage interface.

Next, searching mails by given criteria was implemented. In order to preserve a logical structure of mujMail the team had to update internal structure of mailboxes (Inbox, OutBox, SentBox and Trash) and implement a new search result windows and user folders. The team adds Persistent box, where mails are stored in database and NonPersistent box where a new mail is stored only in memory. The team added the functionality needed for searching.

And finally the team added a basic support for touch screen devices. Because mujMail does not use only standard form elements, but uses canvas and paints screen, touch screen device support could not be easily integrated into mujMail as it, as a standard, works only with standard elements. The team introduced a simple solution, that screen regions will be mapped onto keyboard so missing key can be replaced by tapping the screen.

In the third phase the team changed project management. Hard deadlines were introduced. This was necessary to finish mujMail project in time. This phase is symbolized by intensive development, on-device testing. Team releases development versions nearly on weekly basis. The team also updates user manual to correspond with current version.

The team coded important missing functionalities. As first the team adds compressing connections to mujMail server, threading, conversions. As last user folders and HTML support were added. Then the team started intensive testing phase removing bugs not found by standard testing. During testing the team checks and improves code and comments quality. Package comments describing how the things work were added.

3. Common features

3.1. Multi-language support

The new version of the application also supports multiple languages and encourages further translations. All the texts used in the application (user interface, error messages) are stored in a separate text file, which is embedded in the application JAR. Individual versions of mobile application with different languages are available on the project web. This ensures the mobile part of the application to be more compact.

Requirements		
ID	**Description**	**Remark**
FR1	Mobile application shall support multiple languages. The user can download mobile application with the language he wants from the project web.	Enhanced

3.1.1 Lang class

All the texts used in the application are managed by the `Lang` class. This class is responsible for the loading of texts for a given language. It provides static string variables that contain the actual texts.

The task of our project was the enhancement of this class to return the texts for the functionality to be implemented by the project.

Interface – `Lang`		
Method	**Description**	**Parameters / Return value**

Interface – `Lang`		
Method	**Description**	**Parameters / Return value**
`get`	Returns the text associated with the given code	`lang_code` – lingual code of the text / `String`

The named constants for all the actual texts used in the application are not included in the previous table.

3.2. Exception handling

3.2.1. MyException class

Individual parts of the application use the exception class `MyException` derived from `Exception` class. This class was not fully developed by our project. The use of this class was enhanced and the module was modified to achieve the goals of the project.

Interface – `MyException`		
Method	**Description**	**Parameters / Return value**
`MyException`	Constructor which initializes the error code.	`int errorCode`
`MyException`	Constructor which initializes the error code and the details string.	`int errorCode, String details`
`MyException`	Constructor which initializes the error code and the details which is initialized to given `Exception`	`int code, Exception e`

Interface – `MyException`		
Method	**Description**	**Parameters / Return value**
	class' description.	
`getErrorCode`	Returns the exception code.	`/ int`
`getDetails`	Returns the description of the exception.	`/ String`
`getDetailsNocode`	Returns the description of the exception without prefixed error code.	`/ String`

Exception codes are defined for individual types of functionality. The error detail is taken from `Lang` class which returns the string in selected language.

3.2.2. Exception handler

All modules in the mobile application use common exception handler to handle problems that the user should be informed of. This handler processes `MyException` class defined in this document. It gets the description of the exception and presents the text to the user. If the application code handles some exceptions internally and it is not a fatal error, it processes it directly. Exception handling is therefore as follows:

- catch and process all internal exceptions, i.e. exceptions that does not terminate the user action
- catch all other exceptions aroused during network communication and call `resolveException` function; action that caused the exception is terminated afterwards.

4. Project scope

4.1. Mobile part enhancement

As mentioned above, the basis of the project is the mail-client for mobile devices. It will allow the user the full capability to work with emails and will communicate with email servers as well as with server-part of the project. It should be note that the mobile application will work also without server-part.

Requirements		
ID	**Description**	**Remark**
FR2	Mobile application shall allow perform such basic operations as forward and bounce.	
FR3	Mobile part shall support JSR-75 functionality when it is possible. JSR-75 specifies filesystem support. The required functionality is the possibility to export emails and their attachments to a device filesystem and attaching files in the device FS to outgoing mails.	
FR4	The application will also support user folders and manipulation with them.	
FR5	There shall also be the support for manipulation with IMAP flags, such as setting a flag and removing a flag.	
FR6	User-friendly visualization of HTML documents shall be supported including opening URLs in phone's native browser.	
FR7	Unsupported attachment formats shall be displayed at least as plaintext files.	
FR8	Support for view of signed emails will be provided.	

Requirements		
ID	**Description**	**Remark**
FR9	Support of attachments for outgoing emails.	
FR10	Mobile application shall also support the threading of incoming messages. Messages will be displayed as GMail conversations.	
FR11	There shall be support for search in mails stored in application databases.	
FR12	Application shall support backing up and restoring its whole configuration to ease the transitions to newer versions of the application by users.	
FR13	Mobile application shall support IMAP Push functionality to immediately download new incoming messages from mail server.	
FR14	It should be possible to modify the size of font of the mobile application.	
FR15	Mobile application will support interrupting feature. It shall be possible to interrupt longer actions, such as downloading new emails, synchronization with servers, etc. The application should also assume limited operating memory.	

4.2. Server part implementation

This part will preprocess emails content for more user-friendly displaying in the mobile device. Additionally, it will allow to backup and restore mobile part configuration for better transition to newer versions.

Requirements		
ID	**Description**	**Remark**
FR16	Emails content processing for better display in the mobile device.	
FR17	Mobile application's configuration synchronization shall be supported to ease the transitions to newer versions.	

5. Data storage

5.1. Overview

The application stores mails, configurations and attachments. It provides a unified interface for all classes that need to access stored data. The storage methods are optimized for speed, while keeping data size limitation in mobile devices in mind.

Message contents in each box (Inbox, Outbox, Drafts, Trash, etc.) are stored in their respective RMS databases which have the corresponding name.

5.2. Message content storage system

The unified content storage interface, represented by the `ContentStorage` abstract class, serves to retrieve message contents stored in the RMS and JAR and store and retrieve any message contents, attachments contents data saved by the user either in the RMS or in the mobile device's filesystem.

A set of high-level functions are available for normal usage, as well as a set of low-level raw data operations in order to access binary data, such as multimedia attachments and others.

The storage system is able to handle requests for data from several threads and avoids race conditions and data corruption.

Requirements		
ID	**Description**	**Remark**
FR18	Mobile application shall be able to access data stored in filesystem or record stores using Record Management System (RMS).	Enhanced for JSR75
FR19	Mobile application shall be able to add, modify and delete data stored in the RMS.	Implemented in previous version
FR20	The storage system shall be able to retrieve all data that was	Implemented in

Requirements		
ID	**Description**	**Remark**
	previously stored by the user.	previous version
FR21	The storage system class shall throw an appropriate exception, in case any error occurs when reading or writing to the storage.	Enhanced for JSR75

5.2.1. RMSStorage class

`RMSStorage` class is the part of `Bodypart` class that stores the information about the content of bodypart that is stored in internal Java RMS database. In order to be able to save the bodyparts with alternative sizes, the contents of bodypart can be divided into several parts.

5.2.2. FSStorage class

This class, being the part of `Bodypart` class, is used to store the alternative bodyparts to filesystem of the mobile device if the device allows access to its filesystem. In order to do this `FSStorage` uses `ConnectionInterface` communication interface.

5.3. Message header storage

5.3.1. Overview

The application stores message headers in the RMS. Besides storing standard fields of message header (From, To, Subject, Cc, Bcc, etc.), it contains a list of bodyparts of the given message. Additionally, it stores the information from which account it was downloaded, message attributes (read/unread, replied, deleted, flagged, etc.) and threading-related fields. The unified storage system is used to store, retrieve and delete the message headers. All the headers are stored in the RMS database "[BoxName]_H".

Requirements		
ID	**Description**	**Remark**

Requirements		
ID	**Description**	**Remark**
FR22	Mobile application shall be able to store message headers. The project will enhance the message header in order to the application to be able to work with message flags, and to display the messages as Gmail conversations.	Enhanced for flags manipulation and message threading.
FR23	The storage system shall be able to list all available message headers and provide means of accessing them directly.	Implemented by previous version.
FR24	The mobile application shall be able to search within message headers by its fields.	Implemented by this project.

5.3.2. Data format

Message header is stored in the corresponding box (Inbox, Outbox, Draft, Trash, etc.). A message header stored in RMS consists of the following records:

Message header RMS record format		
Length	**Type**	**Description**
2 B	`char`	Original box where the message was before being deleted
variable	`String`	“From:” field
variable	`String`	Recipients field
variable	`String`	“Subject:” field of the message
variable	`String`	Value of the boundary parameter of the Content-Type field
variable	`String`	Message ID, a unique primary key that is, among others, used to test

Message header RMS record format		
Length	**Type**	**Description**
		existence of a mail in the mobile device
variable	`String`	Folder name where it is stored in mail server (used for IMAP protocols)
variable	`String`	Account from which the mail was downloaded – must be given for each mail
1 B	`byte`	Indicates message format, whether it has attachments (multipart message) or no (plain message)
1 B	`byte`	Indicates if the message was seen or not
1 B	`boolean`	Indicates whether the message was flagged
1 B	`byte`	Indicates if the message was already stored
1 B	`byte`	Send status of the mail
4 B	`int`	Size of the message
8 B	`long`	“Date:” field of the message header
variable	`String`	“Message ID:” field of the message header, if not found, MujMail message ID is used
variable	`String`	Parent ID – thread parent message ID
4 B	`int`	Size of the parent IDs vector. The vector represents the path to the thread root message
variable	`String`	Parent message ID stored in parent IDs vector
…	…	Remaining message IDs stored in parent IDs vector
1 B	`byte`	Number of bodyparts of the message

All the messages' bodyparts headers are also stored while storing the message header.

5.4. Bodypart header storage

5.4.1. Overview

The application stores bodypart headers in the same RMS database as message headers. Bodypart header contains info details about each message part. The unified storage system is used to store, retrieve and delete the bodypart headers. All the headers are stored in the RMS database "[BoxName]_H".

5.4.2. Data format

Bodypart headers are stored in the same database as message headers. Bodypart header format is as follows:

Bodypart header RMS record format		
Length	**Type**	**Description**
variable	`String`	Bodypart's name or attachment's filename
1 B	`byte`	Bodypart content type (TYPE_TEXT, TYPE_HTML, TYPE_MULTIMEDIA, TYPE_APPLICATION, TYPE_OTHER)
1 B	`byte`	Character set of the bodypart content (CH_NORMAL (ASCII), CH_ISO88591, CH_ISO88592, CH_WIN1250, CH_UTF8, CH_USASCII)
1 B	`byte`	Bodypart content encoding (ENC_NORMAL (7 bit), ENC_BASE64, ENC_QUOTEDPRINTABLE, ENC_8BIT)

5.5. Miscellaneous data storage

Remaining data, such as mobile application configuration, account settings, address book and others are stored/retrieved using standard Java RMS interface.

Storage interfaces		
Class/Method	**Description**	**Parameters / Return value**
`Settings / loadSettings`	Used to load the configuration settings from the Settings database. This method is called only at the start of the mobile application.	
`Settings / saveSettings`	Stores the configuration settings to Settings database. Parameter indicates whether to setup the settings from Settings form.	`boolean init`
`AccountSettings / loadAccounts`	Loads accounts from persistent Accounts database. Executed in a separate thread.	
`AccountSettings / saveAccount`	This method is used to save given new or edited account from the form to the recordstore or the account retrieved using Restore mechanism.	`String accountId, MailAccount account`
`AddressBook / saveContactForm`	Saves a contact from the form.	
`AddressBook / saveContact`	Saves the given contact to the AddressBook database and containers. Used, e.g. for restoring the addressbook from either local or remote source.	`Contact contact`

6. Connection interfaces

Connections have been refactored in mujMail 1.08. Connections contain (network) transmission related code, code for connections management, encryption, compression and buffering. Connections are bidirectional. Connections in mujMail are based on the two basic interfaces: `ConnectionInterface` and `ConnectorInterface`.

`ConnectionInterface` is an outer interface that uses all other components of MujMail (mainly, in protocol parsing classes). Creates the illusion of line oriented buffered input and string accepting output.

Interface – `ConnectionInterface`		
Class/Method	**Description**	**Parameters / Return value**
`available`	Returns `true` only if the input stream can be read without blocking, i.e. it contains non-empty input.	`/ boolean`
`clearInput`	Skips all data in input stream until the end of stream.	
`close`	Closes the connection with the server.	
`getLine`	Gets a line from the input stream.	`/ String`
`isConnected`	Checks if the connection was already opened.	`/ boolean`
`open`	Creates a new connection to specified server.	`String url, boolean ssl, byte sslType`
`quit`	Marks the connection as closed but does not close it. No more data can be read or written after the call.	

Interface – `ConnectionInterface`		
Class/Method	**Description**	**Parameters / Return value**
`send`	Sends the specified data to the server.	`byte[] command`
`send`	Sends (multi-)line data to the server.	`String command`
`sendCRLF`	Sends (multi-)line data to the server with ending CR/LF.	`String command`
`unGetLine`	Puts the last read line to the connection input buffer.	
`unQuit`	Reverts the effect of `quit` method, so the connection can be used again if it was not closed before.	

`ConnectorInterface` is an internal interface to be used only in `mujmail.connections` package. The interface is intended to create unified vision on different streams for sending data (into file, over network). It is the most low-level communication component. Most typically, it is used for communication with mail servers.

Interface – `ConnectionInterface`		
Class/Method	**Description**	**Parameters / Return value**
`available`	Returns the number of bytes which can be read from the input stream without blocking.	`/ int`
`close`	Ends connection with the server.	

Interface – `ConnectionInterface`		
Class/Method	**Description**	**Parameters / Return value**
`flush`	Sends all data in the buffer to the server.	
`open`	Creates a new connection to specified server.	`String url, boolean ssl, byte sslType`
`read`	Reads data from given input stream and fills given buffer.	`byte[] data, int off, int len`
`write`	Sends data to the server from the given buffer.	`byte[] data, int off, int len`

7. Searching mailboxes

7.1. Overview

Search package implements mailbox search capability. It searches for given keyword among messages downloaded to the mobile device. Classes in this package can be divided into two groups:

- User interface classes that enable user to enter search settings or display search results.
- Classes used to find messages that fulfill given criteria.

7.2. User interface

All user interface functions are accessible using static methods of class `SearchWindows`. It provides methods both for displaying dialog where user can enter search settings and for displaying results of the search. Settings dialog remembers message parts selected to search in and boxes selected to search in using `SaveableSelectedState` and `WasSelectedReminder`. The results of the search are displayed using class `SearchBox`.

Each matched message contains information about occurrences of search phrases found in the message. This information is accessible via method `getSearchResult`. This information can be later used for better display of search results in `SearchBox`.

Interface – `SearchWindows`		
Method	**Description**	**Parameters / Return value**
`SearchWindows`	Constructor.	`MujMail mujmail`
`displaySearchWindow`	If the search box is not empty, displays it. If it is empty, displays the search settings window.	
`newSearch`	Displays the dialog window to configure the search settings and start	

Interface – `SearchWindows`		
Method	**Description**	**Parameters / Return value**
	the search.	

7.2.1. Running search

Searching is initiated by executing method `SearchCore.search`.

7.3. Search algorithm

Method `SearchCore.search` enumerates all messages in all boxes contained in search settings. If the message matches search criteria specified in search settings, it is added to search box.

The message matches if the date of the message is in interval specified in search settings and if the message matches given search phrases.

Search phrases are represented by class `SearchPhrase`. This class provides method `SearchPhrase.findFirstMatch` that finds first occurrence of the phrase in given message. This method lists all message parts that should be searched and looks for the phrase there.

Message parts that should be searched are represented by class `SearchMessagePart`. This class provides method `SearchMessagePart.findFirstMatch` that finds first match of given search phrase in this message part. This method is `abstract`, this means that every concrete searchable message part must implement searching itself.

7.3.1. Full-text search and search modes

Most of searchable message parts use full-text search in string. Object that provides full-text searching is accessible by calling method `SearchCore.getFulltextSearcher`.

Method `FulltextSearcher.searchInString` uses instance of class `FulltextSearchAlgorithm` for searching the string and `FulltextSearchModes` stored in `SearchPhrase` to check whether the location of

the string matches the `FulltextSearchModes`. This means that the location of string meets given condition - for example it is whole word etc.

7.3.2. Implementing new full-text search algorithm

To implement new algorithm for full-text searching, implement the interface `FulltextSearchAlgorithm`. To make new search algorithm used while searching, create instance of class `FulltextSearcher` with new search algorithm as the parameter and make method `SearchCore.getFulltextSearcher` to return this instance.

7.4. Searching interfaces

All search functions within the message headers are provided by the `SearchCore` class.

Interface – `SearchCore`		
Method	**Description**	**Parameters / Return value**
`search`	Returns vector of messages (instances of the class `MessageHeader`) that match given search settings. Search settings, mailbox to search in and progress painting task are given as input parameters.	`SearchSettings settings, SearchBox searchBox, StoppableProgress progress / Vector <MessageHeader>`

If no message meets given criteria, the method returns zero.

8. User folders

8.1. Overview

User folders serve as new separate storages for emails. User can specify which account to retrieve into selected user folder. Main part of user folders functionality is implemented in `mailboxes` package which currently takes care only about user folders.

8.2. Mail accounts

Mail accounts store user account configuration and information needed for connecting to them (mail server address, port, login, password, etc.). Mail accounts are stored in MujMail object and can be retrieved by `MujMail.getMailAccounts` method. Mail accounts were implemented by previous version of the application. Our project enhanced mail accounts functionality to allow accounts synchronization and user folders creation.

There are two types of mail accounts in MujMail: primary and derived. Primary accounts are stored in RMS database called ACCOUNTS. They are a primary source of information about accounts. These accounts are visible to user. Derived accounts are used only by user folders to customize primary account's behavior and are invisible to user.

Interface – `AccountSettings`		
Method	**Description**	**Parameters / Return value**
`AccountSettings`	Constructor.	`MujMail mujmail`
`deleteAccount`	Removes specified mail account from ACCOUNTS RMS database and shows a confirmation dialog if `sure` parameter is `false`.	`String accountID, boolean sure`
`deleteAll`	Removes all mail accounts from ACCOUNTS RMS database and shows a confirmation dialog if `sure`	`boolean sure`

Interface – `AccountSettings`		
Method	**Description**	**Parameters / Return value**
	parameter is `false`.	
`getNumAccounts`	Opens ACCOUNTS recordstore and returns the number of accounts in the database.	`/ int`
`isBusy`	Returns `true` while loading accounts from database, otherwise `false` is returned.	`/ boolean`
`loadAccounts`	Loads accounts from persistent Accounts database. Executed in a separate thread.	
`saveAccount`	This method is used to save given new or edited account from the form to the recordstore or the account retrieved using Restore mechanism.	`String accountID, MailAccount account`
`showAccount`	Shows a form for editing an account.	`String accountID`
`waitForAccountsLoading`	Blocks a caller until all accounts are loaded.	

8.3. Implementation

Main part of work is performed by `BoxList` class. `BoxList` class contains the list of user boxes. It takes care about basic mailbox operations like creating, loading, removing. User folders are implemented as `InBox` class instances. Their names are stored in a special database. Each user folder has (and also loads and saves) list of accounts to retrieve.

Interface – `BoxList`		
Method	**Description**	**Parameters / Return value**
`BoxList`	Constructor.	`MujMail mujmail`
`createPersistentBox`	Creates new user folder. Shows user folders settings form to customize the mailbox.	
`deleteAllMailsFromAllUserBoxesAndDB`	Removes mails from all user boxes. This function is intended for clearing databases and freeing space for new mails.	
`editUserMailBoxSettings`	Shows edit form for user mailbox. Allows setting user folder properties like name of retrieved accounts. Parameter indicates the index of mailbox in mailboxes vector.	`int index`
`getBoxList`	Returns vector of user created mailboxes.	`/ Vector<InBox>`
`loadBoxes`	Loads all mails stored in persistent storage into user folders.	
`removeUserMailBox`	Remove mailbox from system. Remove also all its databases.	`int index`
`sort`	Sorts user folders by their names for displaying purposes.	
`spaceOccupied`	Returns the space occupied by all user folders databases.	`/ int`

9. Threading mails

9.1. Overview

MujMail application supports the basic type of threading that is equivalent to Gmail conversations. Mails threading can be viewed as a special type of sorting emails where mails which are logically relevant are grouped together.

The goal of mails threading, shortly, threading is ordering mails to groups, which are logically dependent. Two mails should be in one group if the second one was the reply for the first one, i.e. if it is possible to interconnect them into logical chain (in which some mails can be missing).

9.2. Concept

Threading is based on message headers, which typically contain Message-ID and, in case the message is a reply to an email, the message header also contains fields In-reply-to, which indicates to what mail this message is replying and a list of references containing the whole chain (history of replies).

9.3. Algorithm

9.3.1. Input

Input to the threading algorithm is a list of mails to be threaded, which is represented in MujMail as `Vector<MessageHeader>` structure. While retrieving mails it is controlled, whether a given email contains Message-ID field and in case, it does not contain the field a unique message ID is generated for this email, which guarantees that the emails not containing Message-ID won't be grouped together to a thread.

9.3.2. An execution cycle

Threading algorithm takes the messages sequentially and does not assume any ordering of input mails. Internally, the algorithm creates a map of IDs of messages (keys) containing a list of messages with the same ID. The situation when several messages have the same Message-ID can occur, e.g. when a mail is sent to several recipients and in MujMail the accounts are created for these recipients and when the mails from these accounts are retrieved the sent mail will occur twice in InBox with same MessageID.

An execution cycle is as follows:

- Get a message from input and check if there is already a mail with this ID in the map.
- If not, then the message is added to the map as root message for a thread when parent ID field does not exist in the message header otherwise a thread is created with empty root and the message is added to this thread.
- Otherwise (the case when some messages already exist for given ID), we differ two possibilities:
    - If the message is a root for some thread then map contains this thread with empty root – add the message as root.
    - Otherwise a message is added to an existing thread as a new message for the thread.

At the end of the algorithm execution a structure of threaded emails is returned as the instance of the class `ThreadedEmails`.

Interface – `Algorithm`		
Method	**Description**	**Parameters / Return value**
`getAlgorithm`	Returns the instance of `Algorithm` class.	`/ Algorithm`
`invoke`	Executes a threading algorithm on the input `Vector<MessageHeader>`.	`Vector<MessageHeader> / ThreadedEmails`

9.4. Implementation details

In general, mails dependency creates a tree data structure, but for the project it was decided not to show the whole structure that could be rather deep. Instead, mails are displayed only with 1 level of depth. This decision is reasonable for the mobile devices as their display can be rather small and thus often the whole tree structure cannot be displayed in the device. In order to implement mail threading runtime mail storing system had to be refactored.

MujMail 1.07 uses `Vector<MessageHeader>` structure which contains the mails stored in given mailbox. Thus, it does not count with mails threading. In order to be able to thread the mails in given mailbox the new interface `IStorage` had to be introduced.

This interface is created in order to be able to sort only given set of mailboxes. Such mailboxes as Draft, Trash, etc. are not threaded. Class `ThreadedEmails` implements this interface in order to provide the tree structure of threads.

Interface – `IStorage`		
Method	**Description**	**Parameters / Return value**
`addMessage`	Adds the given message header to the storage.	`MessageHeader messageheader`
`getEnumeration`	Returns all the message headers in the storage as an enumeration.	`/ Enumeration`
`getMessageAt`	Returns the message in storage with the given index.	`int index / MessageHeader`
`getSize`	Returns the number of message headers in the storage.	`/ int`
`isEmpty`	Returns `true` only if the storage is empty.	`/ boolean`
`removeAllMessages`	Removes all messages from the storage.	
`removeMessage`	Removes first occurrence of given message header from the storage.	`MessageHeader messageheader`
`removeMessageAt`	Removes the message in storage with the given index.	`int index`
`sort`	Sorts messages in the storage in the order defined by given comparator.	`Comparator comparator`

10. HTML content display

10.1. Overview

The mobile application is able to display an HTML bodypart. Parsing is implemented directly in the application in order to simplify the display procedure and speed up the overall HTML visualization process.

10.2. HTML parser

`Parser` class is responsible for parsing HTML stream. There are some requirements for Parser functionality:

- It should be the as simple as possible in meaning of robustness - we do not require handling of each error in HTML stream (like missing closing tag, incorrectly paired tags). Parser is not validating the input, just wants to highlight some subset of HTML tags.
- Parser has to process incomplete HTML source. It's possible in mujMail to limit the number of downloaded bytes/lines of e-mail so generally we cannot expect correctly paired tags not even closed (HTML can end, e.g. with "<b" and we do not know if that had to be <b> or
 tag).

When stream is parsed, vector of elements is returned.

Interface – `Parser`		
Method	**Description**	**Parameters / Return value**
`Parser`	Constructor, which creates a new instance of parser with HTML content to be parsed.	`String html`
`parse`	Parses the HTML source passed in the constructor and returns parsed	`/ Vector`

Interface – `Parser`		
Method	**Description**	**Parameters / Return value**
	elements.	

10.3. Displaying HTML content

`Browser` class is responsible to displaying the vector of elements returned by the parser. For drawing of these elements instance of the `Browser` class calls a `draw` method on those elements as it assumes that the vector of elements is a vector of instances implementing `Drawable` interface.

Interface – `Browser`		
Method	**Description**	**Parameters / Return value**
`Browser`	Constructor, which takes as input a vector of objects of class `Drawable`.	`Vector<Drawable>`
`getActualBrowser`	Returns the current browser.	`/ Browser`
`getActiveLink`	Returns the selected hyperlink index.	`/ int`
`setLink`	Sets the selected hyperlink.	`String link`
`increaseLinksNumber`	Increases hyperlinks number on the page. Returns the amount of all links after the operation.	`/ int`

Interface – `Drawable`		
Method	**Description**	**Parameters / Return value**
`draw`	Draws the element on given position. The returned position should be the position where the next element can be drawn.	`Graphics g, int x, int y / Point`

11. Tasks

Package `mujmail.tasks` provides classes and interfaces for creating, running and managing tasks. Basically, a task is an action that is executed in a new thread and that is registered before execution and unregistered after the execution is done. This enables various management possibilities of such tasks. Tasks also support displaying a progress of the action to user. User can cancel the displaying of this progress by pressing Back button. User can also see all running tasks and their progresses in Tasks manager. If the task is descendant of `StoppableBackgroundTask`, progress contains Stop button that allows stopping the task.

The classes `ProgressManager` and `StoppableProgressManager` provide the user interface for displaying of the progress of `BackgroundTask` and `StoppableBackgroundTask`, respectively.

11.1. Defining a task

In order to define a new task, new class has to be created that inherits either from `BackgroundTask` or `StoppableBackgroundTask` if the task should be stoppable. Then the action that should be performed in the task has to be defined in the method `BackgroundTask.doWork` that is `abstract` in parent class.

It is not possible to stop a task preemptively in J2ME. That is why cooperative multitasking must be used. If the task is the descendant of `StoppableBackgroundTask`, it must control whether it should terminate. `StoppableBackgroundTask` implements interface `StoppableProgress` that contains method `StoppableProgress.stopped`. If this method returns `true`, the task should terminate.

While running the method `BackgroundTask.doWork`, it is possible to display progress of the action to user. Methods of interface `Progress` that `StoppableBackgroundTask` implements can be used to do this.

It is also possible to disable displaying progress of task to user by calling method `BackgroundTask.disableDisplayingProgress` before starting the task.

Interface – `BackgroundTask`		
Method	**Description**	**Parameters / Return value**
`BackgroundTask`	Constructor, input is task name.	`String taskName`
`cancelStartingTask`	If this task is waiting to start, cancels starting the task.	
`disableCheckBeforeStarting`	Disables the check if the limit of the same type of tasks is reached. It means that the task will start immediately.	
`disableDisplayingProgress`	Disables displaying progress after start of the task.	
`doWork`	This method defines the background task.	
`showProgress`	Shows progress of the task. If the task is not running, shows next screen.	
`isRunning`	Returns `true` only if the task is still running.	
`setTitle`	Sets the title of the progress bar which is associated with this task.	`String title`
`start`	Starts the background task in new thread and displays progress bar.	
`start`	Starts the background task in new thread and displays progress bar. When the task is finished the screen will be switched to the input parameter.	`Displayable nextScreen`

Interface – `BackgroundTask`		
Method	**Description**	**Parameters / Return value**
`start`	Starts the background task and shows the progress bar. When the task is minimized the screen will be switched to `prevScreen` parameter. When the task is finished the screen will be switched to the `nextScreen` parameter.	`Displayable nextScreen, Displayable prevScreen`
`updateProgress`	Updates progress of the progress bar associated with this task.	`int total, int actual`

Interface – `StoppableBackgroundTask`		
Method	**Description**	**Parameters / Return value**
`StoppableBackgroundTask`	Constructor.	`String taskName`
`stopped`	Returns `true` only if Stop button was pressed.	`boolean`
`stopTask`	Sets the state of the task to stopped.	

11.2. Task execution

11.2.1. Starting a task

Method `start` has to be called in order to start the task. Before the task is started, it is checked whether number of running tasks of the same class is less than given limit. If it is not possible to start the task immediately, the dialog where user can cancel running the task is displayed. It is possible to disable displaying this dialog by calling method `BackgroundTask.disableDisplayingUserActionRunnerUI` before starting the task. The task is placed between waiting tasks and it is started when the number of tasks of the same class is less than limit.

To start the task immediately, without the check if there are less tasks of the same class than given limit started, call method `BackgroundTask.disableCheckBeforeStarting` before starting the task.

11.2.2. Observing tasks progress

It is possible to register to receive notifications every time the progress of the given task is changed. The list of events that are received is described in enumeration class `TaskEvents`. Classes `BackgroundTask` and `StoppableBackgroundTask` are descendants of class `Observable` that provides methods for registering and unregistering objects that want to listen to these events.

Interface – `Progress`		
Method	**Description**	**Parameters / Return value**
`getActual`	Returns the value of current progress of the progress bar.	`/ int`
`getTitle`	Returns the title of the progress bar.	`/ String`
`getTotal`	Returns the value of the whole progress.	`/ int`
`incActual`	Increments current progress of the progress bar.	`int increment`
`isDisplayed`	Returns `true` only if the progress bar is currently displayed.	`/ boolean`

`setTitle`	Sets the title of the progress bar.	`String title`
`updateProgress`	Updates the progress bar.	`int total, int actual`

Interface – `StoppableProgress`		
Method	**Description**	**Parameters / Return value**
`stopped`	Returns `true` only if the Stop button was pressed.	`/ boolen`

11.2.3. Managing tasks

There is a class `TasksManager` for managing the tasks. Class `TasksManagerUI` provides user interface for some task management features such as displaying a list of running or waiting tasks or displaying progress of running tasks.

11.2.4. Running actions on task start or task end

It is possible to receive notifications every time some task starts or terminates. To register for receiving such events there are methods `TasksManager.addEndTaskObserver` and `TasksManager.addStartTaskObserver`. Then it is possible to check given condition every time some task starts or terminates and if it is true, run given action using class `ConditionalActionRunner`. For displaying user interface to users in case that it is possible to use class `ConditionalActionRunnerUI`.