

# MS Wallet

Hannes Tribus - 3941, Irene Moriggl - 3794

Free University of Bolzano

## 1 System Functions

The system is designed for cell phones able to connect to Internet. It provides the following functionalities to the user:

- Creating expense records where the user can specify the following fields:
  - title (30 characters)
  - cost (decimal format)
  - currency of the expense
  - expense date (defaults to creation time)
  - short description (150 characters)
  - expense category (choice group)
  - delete options (see below)
- Editing existing records
- Sorting the expenses according to expense date, title, cost, creation date or last modification date. The sort criteria defaults to expense date (most recently first) and can be changed in the settings.
- Searching for record(s) according to title or description, max. cost, min. cost or expense date
- Store the (changed) expenses to the server (Web4Wallet & database)
- Generating an expense report in form of 2 charts
- Retrieving the expenses form the server (overwrites all non-synchronized records)
- customizing the application with settings: user name, password, sort criteria

The above mentioned delete options are as follows:

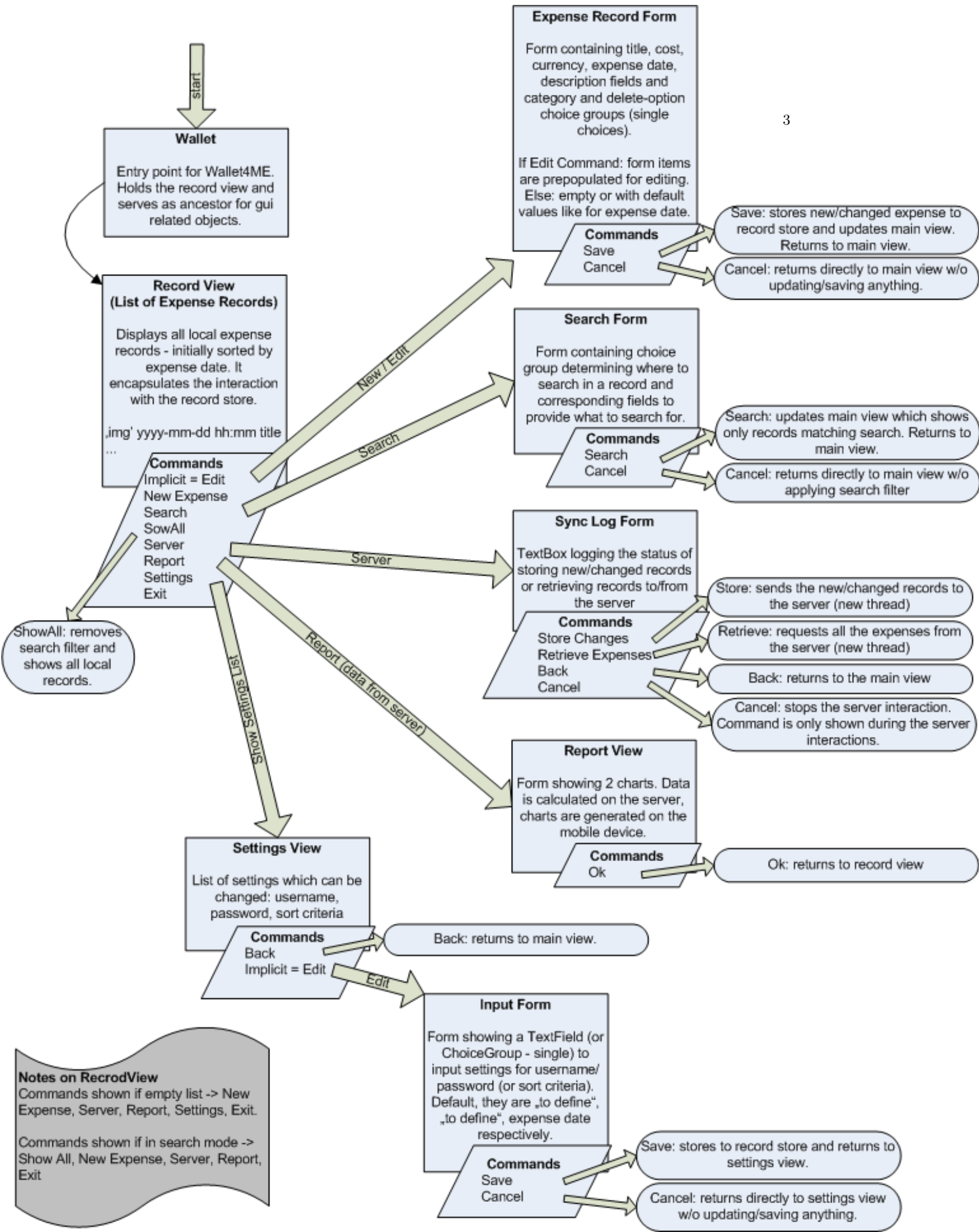
- keep expense: keeps the expense on the cell phone and as soon as it is synchronized it is kept also on the server (default).
- keep only on server: hides the expense from the list and after the synchronization to the server it is deleted on the cell phone.
- delete permanently: hides the expense from the list and after the synchronization it is deleted on the cell phone and on the server. Choosing this option for a newly created expense record won't store this expense at all.

For the features of storing, retrieving expenses and report generation the user must provide authentication information (user name and password) to the server Web4Wallet. These information are automatically taken from the user's settings. Initially they have the value "to define" and can be changed under the settings. The interactions with Web4Wallet and the underlying database is done by means of HTTP requests, which implies that the user must have the possibility to connect to the Internet in order to use these two services.

The application logs the server interactions. In the case of storing and retrieving expenses to/from the server it displays immediately and in any case what is going on and how much expenses are sent through the network. In the case of reporting instead it shows the logged data only if there has been a problem.

## **2 Human Computer Interaction**

The page flow diagram of this document (see next page) shows the possible user interactions. The alerts are not shown. These arise for example whenever the system is not able to process the user requests correctly. This can happen particularly, if the user's record store is not available or accessible.



### 3 Architecture

The system architecture is shown in figure 1. The server is used in order to be able to store the expense records (= one way synchronization), as the initial requirements postulated. In order to give additional meaning to the server, we introduced the feature of retrieving the expenses from it. The user could delete local records to save memory on the phone and retrieve all expenses if necessary. Keeping this in mind, we let the server summarize the reported data so that we are sure that all relevant expense records are taken into account.

The server interacts with an underlying postgres database where it stores beside the expense records also the user's credentials for performing the authentication. The creation and update of the database is done using the hibernate framework.

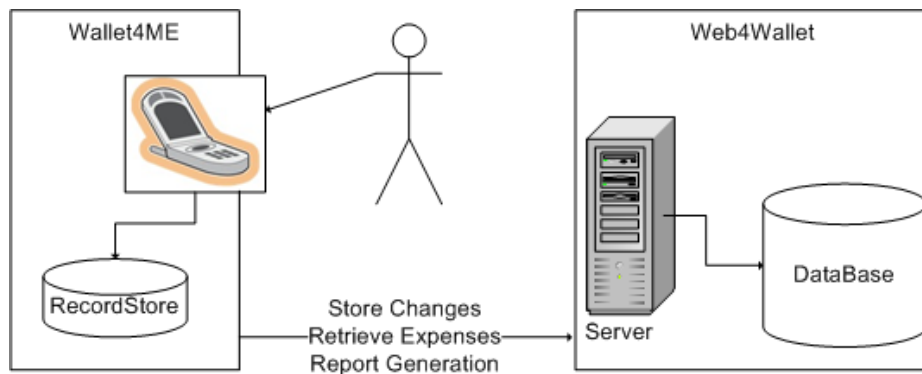


Fig. 1: System Architecture

## 4 Code Structure

The main classes for the graphical user interface were already described in the page flow diagram: Wallet, RecordView, ExpenseRecordForm, SearchForm, SyncLogForm, ReportView, SettingsView, InputForm. These classes are packaged within “gui” together with the class ChartFactory. The latter is responsible for creating the charts displayed on the report view by means of an external library called ChartComponent.

The “gui” package is tightly related to the package “records”. The classes ExpenseRecord and Chart represent the expense records and report charts. The latter uses a list of ChartElement to hold its label-value pairs, which will be passed to the ChartComponent library to draw the chart.

The classes used in combination with the record store and the record view are ExpenseRecordFilter, ExpenseRecordSearchFilter and RecordComparator. The first filter is used to retrieve only those expense records, which have to be sent to the server; that are those having the modify or new date smaller greater than the date of the last transmission. The second filter applies the search criteria so that only matching expenses are retrieved. The comparator enables the different sorting possibilities.

The class XMLHandler instead processes the data from the server and provides it to the gui classes. It uses the RecordXMLHandler for parsing the xml with the records and uses the ChartXMLHandler for parsing the xml with the data to produce the charts.

The third package is called “communication”. Classes related to the http connection can be found here:

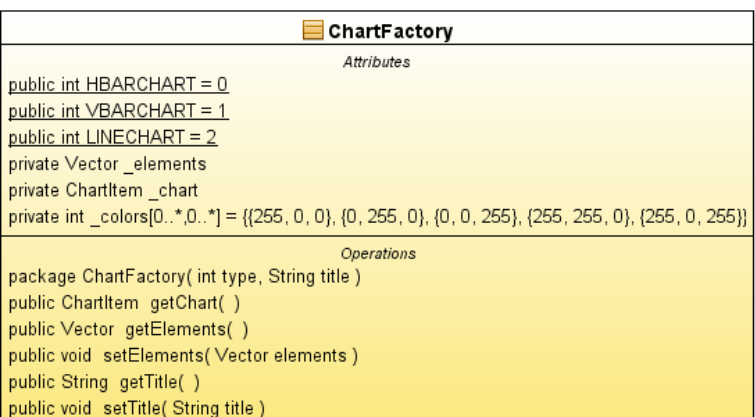
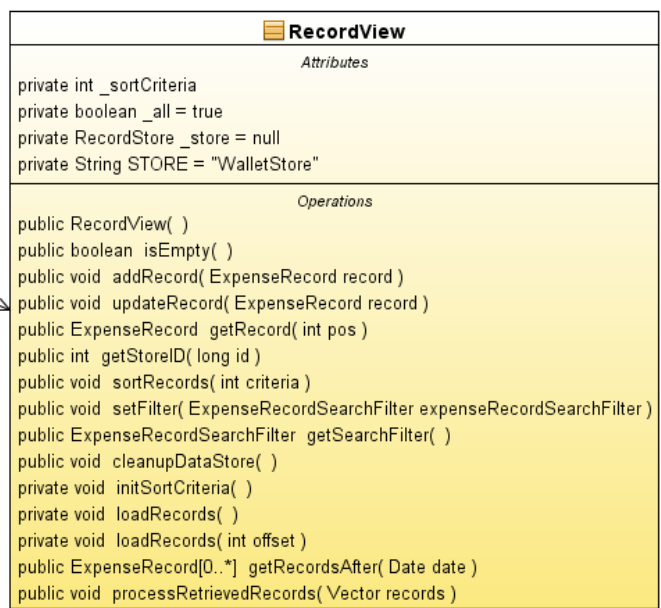
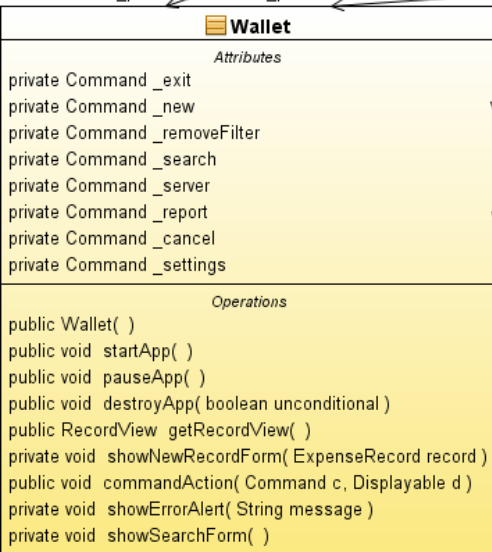
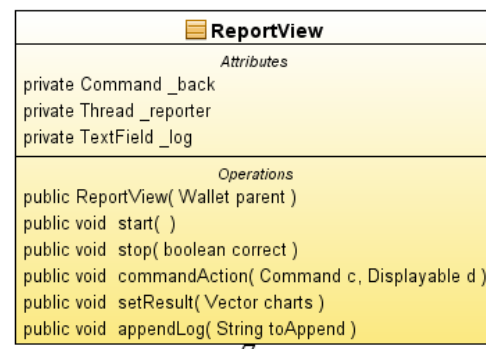
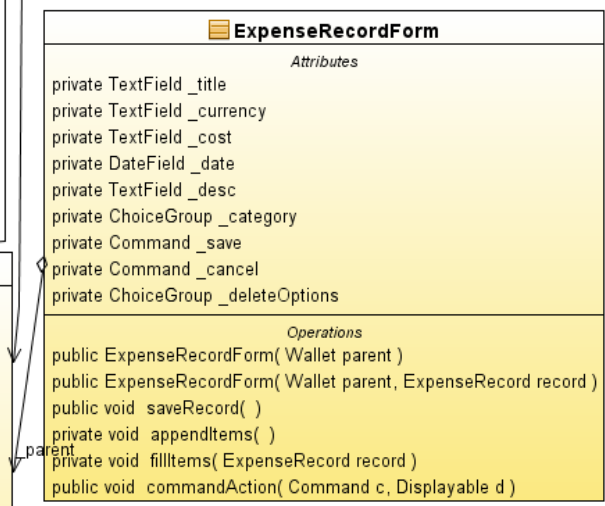
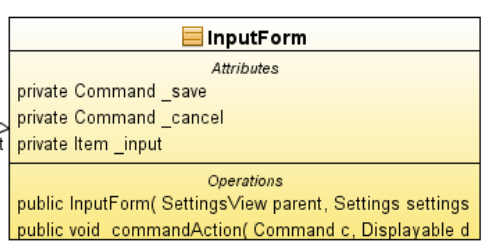
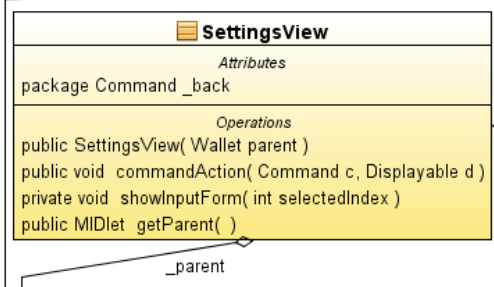
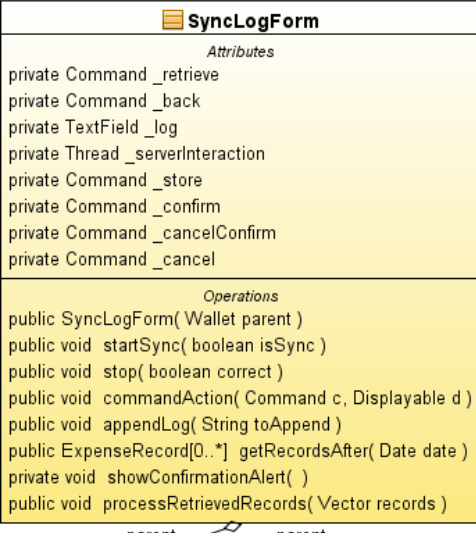
The Reporter is used to let the server generate the summary data. After authentication Web4Wallet generates 2 summaries from all the data on the server and sends it back. The Reporter passes the data to the records package where the charts are then generated. The reporting runs in its own thread and is directly invoked going to the ReportView. Leaving and returning to this view will therefore just result in aborting the interaction and in another call to the server.


The ServerInteraction instead is used for storing changes and retrieving the expenses to/from the server. At storing the changes (=> Server->Store Changes) the user is authenticated and the records having the new or modify data greater than the date of the last storage are transferred via xml string. The deletion on the server is only done if the user set the deletion flag (=> deletion “delete permanently” when edit of record is performed). At retrieving the expenses (=> Server->Retrieve Expenses) the user is authenticated and receives all expenses in form of xml string from the server. The ServerInteraction passes the string to the records package, where it is parsed and all records in the user’s record store are replaced by the new ones. All non-synchronized records are lost. Both interactions run in their own threads. The user can leave the SyncLogForm, he gets informed by an alert when the jobs are finished.


Both Reporter and ServerInteraction extend the abstract class Connection, which performs the common tasks like authenticating.


In the record package we put also the class GlobalConstants. It holds common constants used by all classes. Also the Settings class is here, which provides the user specific settings (like user name and password for the server communication).


The following 3 pages show the class diagrams of these 3 packages (in the order gui, records, communication; if not readable please zoom in with your pdf reader).





 <b>ExpenseRecord</b>	
<i>Attributes</i>	
<pre>private String CATS[0..*] = {"Household", "Food", "Auto/Transport", "Entertainment", "Clothing", "Other"} private Image CATS_IMG[0..*] = null private char SEPARATOR = ',' private String _title private String _desc private String _currency private String _category private Date _date private Date _lastMod private float _cost private long _id private int _delete</pre>	
<i>Operations</i>	
<pre>public ExpenseRecord( byte rec[0..*] ) public ExpenseRecord( ) public void setCost( float cost ) public void setCurrency( String currency ) public void setCategory( String category ) public void setDate( Date date ) public void setDelete( int delete ) public void setDescription( String desc ) public void setLastMod( Date lastMod ) public void setTitle( String title ) public void setNewDate( long date ) public float getCost( ) public String getCategory( ) public String getCurrency( ) public Date getDate( ) public int getDelete( ) public String getDescription( ) public Date getLastMod( ) public String getTitle( ) public long getID( ) public Object get( int criteria ) public byte[0..*] getBytes( ) private String[0..*] split( String record ) public String toString( ) private String convert2Fix( int timePart ) public Image getImage( ) public String[0..*] getCategories( ) public Image[0..*] getCategoryImages( ) public int getIndexForCategory( String cat ) public String getCategoryForIndex( int index )</pre>	


 <b>Chart</b>	
<i>Attributes</i>	
<pre>private String _title private Vector _elements</pre>	
<i>Operations</i>	
<pre>public Chart( String title, Vector elements ) public Chart( String title ) public Chart( ) public String getTitle( ) public Vector getElements( ) public void setTitle( String title ) public void setData( Vector elements ) public void addElement( ChartElement element )</pre>	


 <b>ChartElement</b>	
<i>Attributes</i>	
<pre>private String _label private int _value</pre>	
<i>Operations</i>	
<pre>public ChartElement( String label, int value ) public String getLabel( ) public int getValue( ) public void setLabel( String label ) public void setValue( int value )</pre>	


 <b>ExpenseRecordFilter</b>	
<i>Attributes</i>	
<pre>private Date _date</pre>	
<i>Operations</i>	
<pre>public ExpenseRecordFilter( Date date ) public boolean matches( byte candidate[0..*] )</pre>	

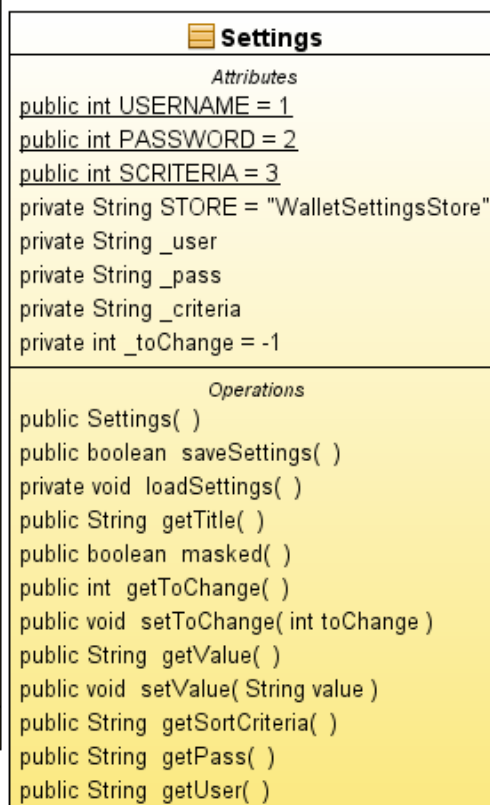
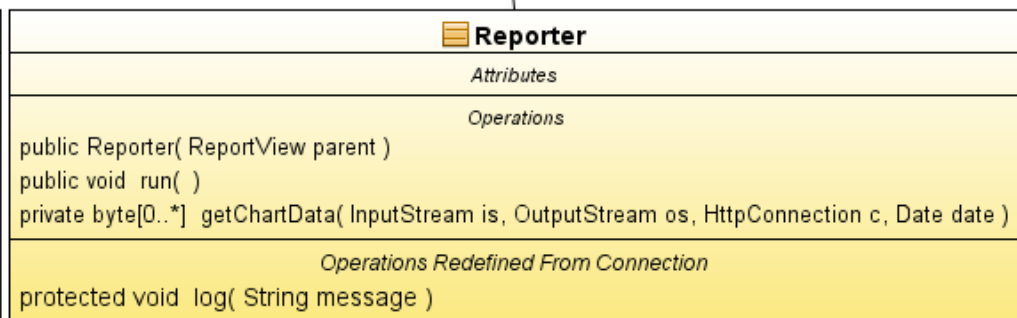
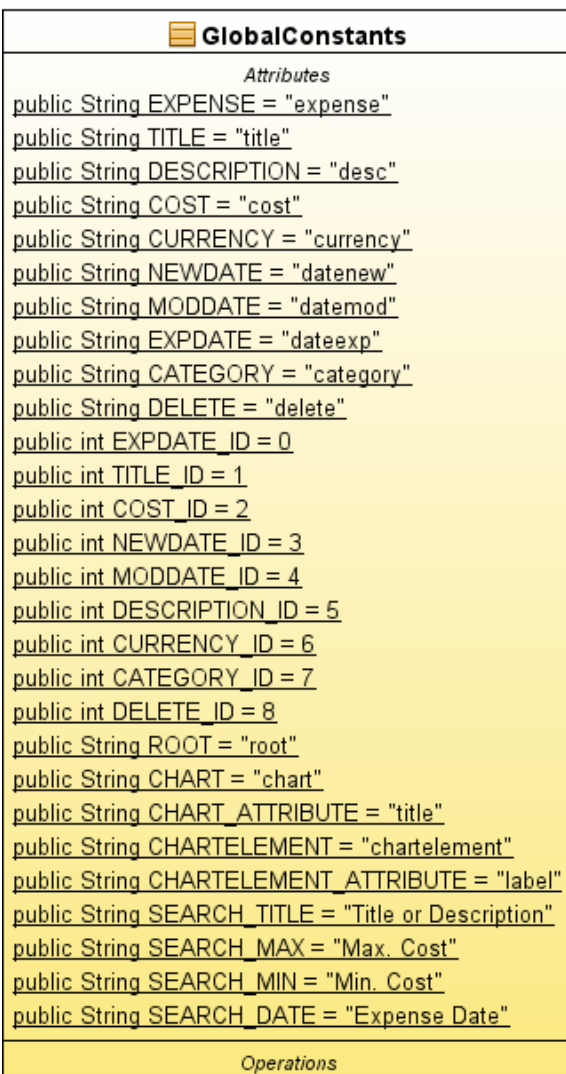
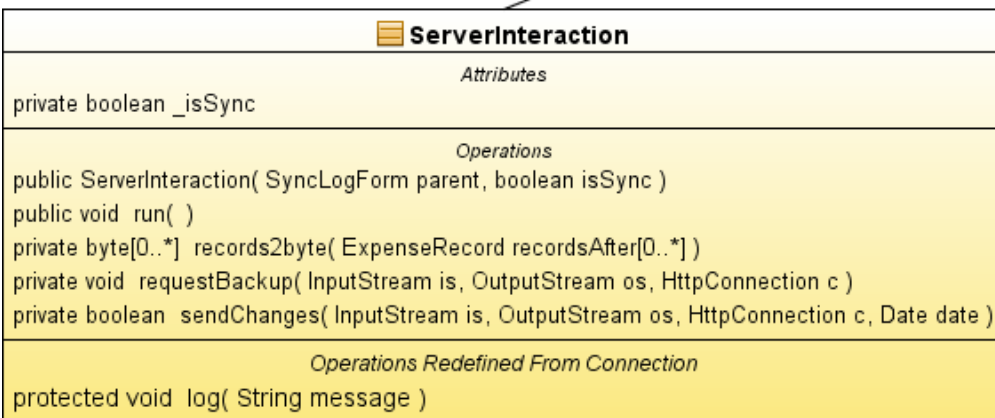
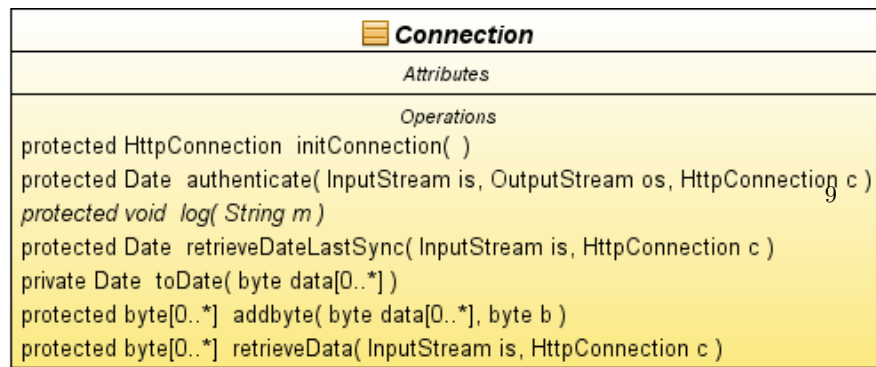
 <b>ExpenseRecordSearchFilter</b>	
<i>Attributes</i>	
<pre>private Object _filter private String _type</pre>	
<i>Operations</i>	
<pre>public ExpenseRecordSearchFilter( String type, Object filter ) public boolean matches( byte candidate[0..*] )</pre>	

 <b>RecordXMLHandler</b>	
<i>Attributes</i>	
<pre>private Vector _expenses = new Vector() private String _currentElement</pre>	
<i>Operations</i>	
<pre>public void startDocument( ) public void startElement( String uri, String localName, String xmlElement, Attributes attributes ) public void characters( char ch[0..*], int start, int length ) public Vector getExpenses( )</pre>	

 <b>XMLHandler</b>	
<i>Attributes</i>	
<pre>private String HEADER = "&lt;?xml version='1.0' encoding='UTF-8'?'&gt;\n"</pre>	
<i>Operations</i>	
<pre>public byte[0..*] transform( ExpenseRecord records[0..*] ) private String clean( String toClean ) public Vector prepareReportData( byte reportXML[0..*] ) public Vector prepareBackupData( byte reportXML[0..*] )</pre>	

 <b>ChartXMLHandler</b>	
<i>Attributes</i>	
<pre>private Vector _charts = new Vector() private String _currentLabel private String _currentElement</pre>	
<i>Operations</i>	
<pre>public void startDocument( ) public void endDocument( ) public void startElement( String uri, String localName, String xmlElement, Attributes attributes ) public void characters( char ch[0..*], int start, int length ) public Vector getCharts( )</pre>	

 <b>ExpenseRecordComparator</b>	
<i>Attributes</i>	
<pre>private int _criteria = GlobalConstants.EXPDATE_ID</pre>	
<i>Operations</i>	
<pre>public ExpenseRecordComparator( ) public ExpenseRecordComparator( int sortCriteria ) public int compare( byte rec1[0..*], byte rec2[0..*] ) private int compare( float toCompare1, float toCompare2 ) private int compare( Date toCompare1, Date toCompare2 )</pre>	



## 5 Faced Technical Problems

### 5.1 Server

The major problems arose due to the usage of a server. We experienced, that it represents a huge overhead to mobile applications. Besides setting up such a server and an underlying database and keeping both running, the communication to and from the server were quite error prone. Although we could use the SAXParser, it took effort to wrap and unwrap the transmitted information from the expense objects to an xml string and vice versa. This shows also our code structure, which could be much smaller without the usage of a server. For such kind of mobile application we would not use a server again, if we are not forced to.