Symbian OS: End-to-End Sockets API Example

Version 1.2
May 10, 2006

S60 platform

Legal Notice

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

Change History

July 26, 2005	Version 1.0	Initial document release
October 5, 2005	Version 1.1	JSP included in the document
May 10, 2006	Version 1.2	Example updated to support S60 3rd Edition. Minor editorial changes to the document.

1. Introduction

This example is a reference implementation of an enterprise system that includes a mobile client, server-based database, and a Web portal. The purpose of this example is to demonstrate the following:

- How to incorporate a mobile client into a back-end system.
- How to transfer data between the server and the mobile client in a secure manner.
- How to parse the received data at the mobile end.
- How to inform the mobile client of changes in the server database.

The logic of the example in brief is as follows: A Web portal is used for adding users to the system and for adding tasks to the users. Users can view and complete their tasks through a mobile client. All users belong to a group. There are three different groups: the administrators, managers, and workers. Administrators only manage the users of the system and do not handle tasks in any way. Managers handle tasks by adding them to workers and other managers. Managers can use the mobile client for completing their tasks. Workers only view and complete their tasks through the mobile client.

2. Architecture

The architecture of the enterprise example is displayed in Figure 1 and Figure 2. The three main parts are the Java™ SSL socket server, Symbian client, and a PC.

The Symbian client is implemented using the standard MVC design pattern, which allows the program to be easily ported to other Symbian platforms. The UI interacts with the user while the engine does the actual work. The engine communicates with the server by using Symbian's secure (SSL/TLS) sockets. Data is transmitted between the server and the mobile client securely by using an SSL connection. Using SSL guarantees that all data is encrypted during transport and that both sides of the connection can be verified. The engine is responsible for opening a GPRS connection when one is needed. It also listens to SMS messages sent by the server. When an SMS message is received, the engine automatically reloads data from the server.

The Java server of this example can be roughly divided into two parts. First, there is a communication package that handles incoming SSL connections and offers a certificate to the connecting client. After the connection has been established, the second part of the server handles parsing the incoming client messages and communicating them to a MySQL database. It also receives the reply from the database and parses it into a comprehensible format before offering it to the communications package for sending it as a reply to the client. The SMS message sending feature is not implemented in the server solution of this example due to the fact that no unambiguous solution exists. You can implement it using one of the existing commercial and free solutions suitable for your purposes.

Administration of the database is done through one of two alternative Web applications, a PHP or a J2EE™ application.

2.1 PHP

The PHP version is used with a Web browser and PHP pages. The PHP scripts, used through an Apache Web server, enable the user to connect securely (using HTTPS) to the database and modify the content of the database, using PHP's built-in MySQL communication abilities. Apache handles running the PHP scripts and forming a secure connection between the browser and the server.

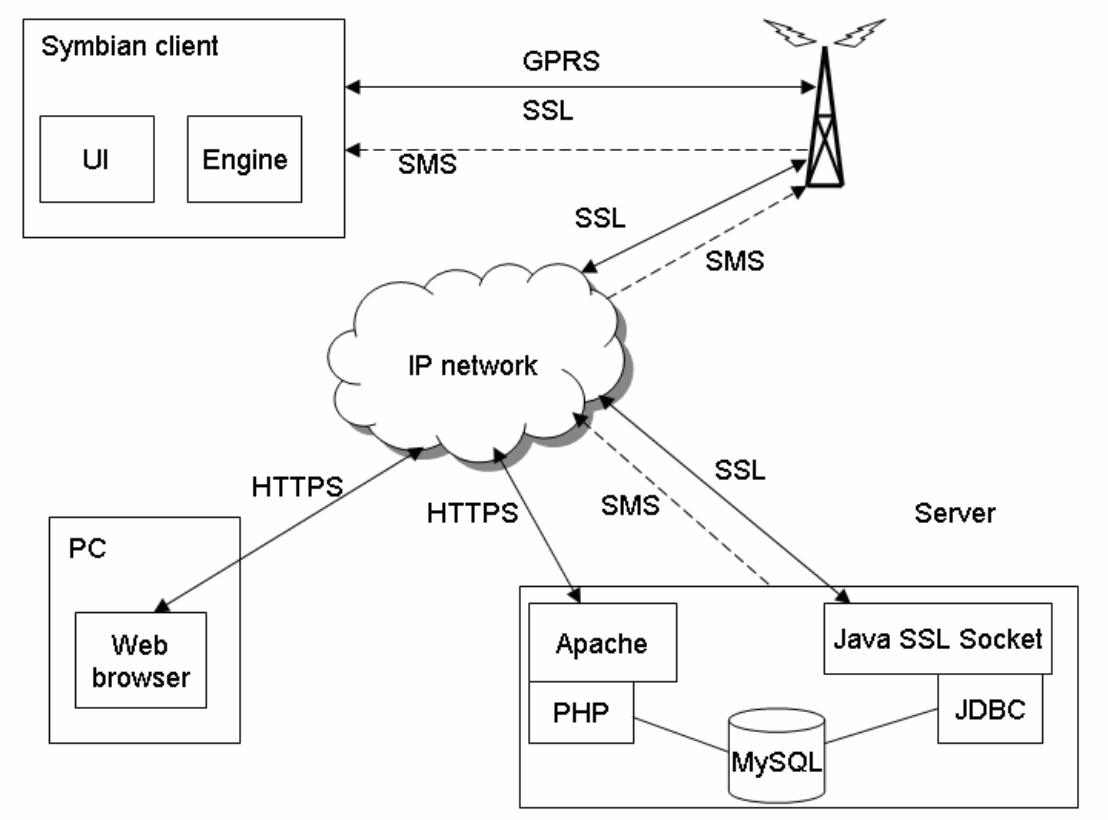

Figure 1: Architecture of the enterprise example using PHP

2.2 J2EE

The J2EE option consists of an Apache Tomcat servlet container and a Java Model 2 implementation using JavaServer Pages, Java Servlets, and JavaBeans. It uses the same MySQL database for data storage but is otherwise different from the PHP version. The Java Model 2 design pattern, which is more or less synonymous with the Model-View-Controller design pattern, presents new challenges for the design and complicates the architecture.

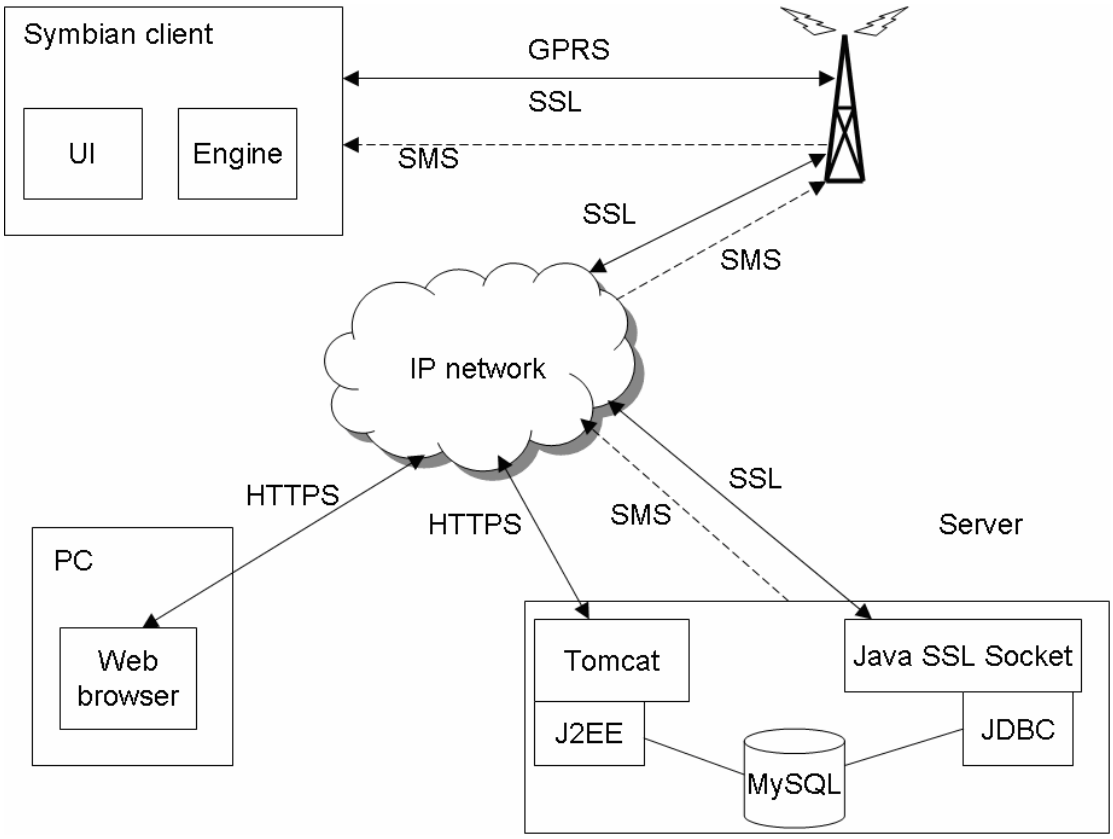

Figure 2: Architecture of the enterprise example using J2EE

2.2.1 Model

The model role is implemented by the JavaBeans component, the database access object (DBAO), and the MySQL database (see Figure 3). Due to the requirements of the JSP 2.0 specification, the JavaBeans themselves cannot make direct references to the database so they need a middleware object (the DBAO) to do the actual queries. The DBAO is also used by the controller when it needs access to the database. The actual beans consist of getter and setter functions that provide data to the view.

2.2.2 Controller

The controller is implemented by a single Java servlet. All client requests go through the servlet; no direct references to a JSP page are allowed (except for the `index.jsp`, the login screen). The servlet decides what page is shown in response to the client request.

2.2.3 View

The view is implemented by a collection of JSP files. This example uses JSP 2.0 and JSTL so there are no “scriptlets,” which can commonly be found in JSP 1.2 files. The Tomcat JSP parser (Jasper) parses the JSP files into an HTML page which is then sent to the client. The JSP files access the database through the JavaBeans which in turn use a persistent DBAO for data retrieval.

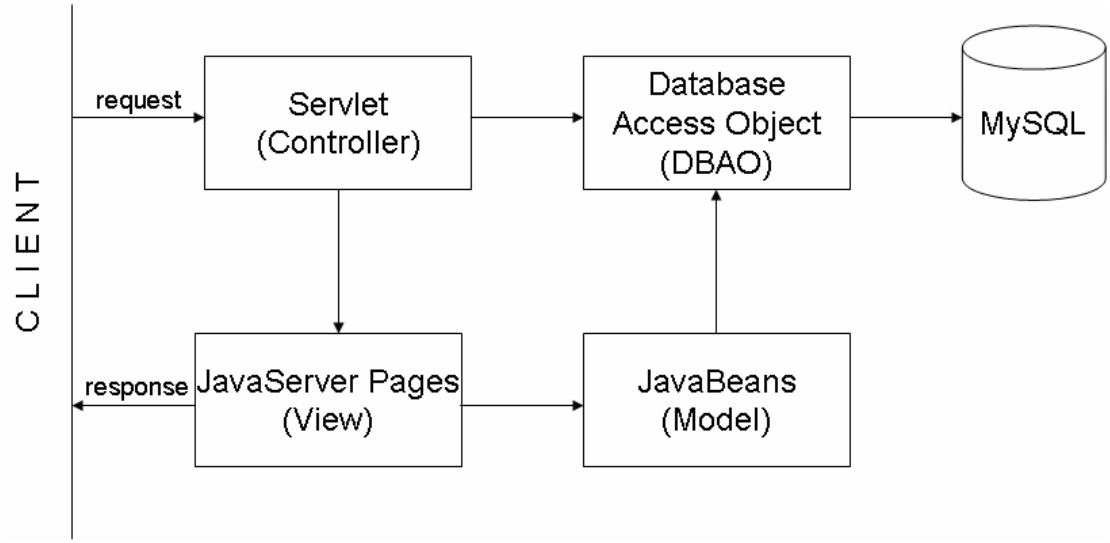

Figure 3: A diagram of the data flow in the system

3. Use cases

This chapter describes the use cases for the mobile client and the Web user interface. The chapter also illustrates with general-level interaction diagrams how the use cases relate to the architecture of the enterprise example.

3.1 Mobile client

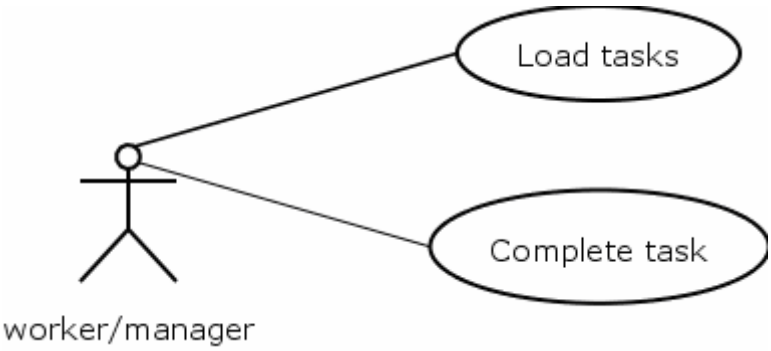

Figure 4: Mobile client use cases

3.1.1 Load tasks

Actors:	Worker or manager
Summary:	Loads the user's undone tasks from the server to the mobile client.
Preconditions:	The following parameters have been set: username, password, IAP to be used, name and port number of the server.
Detailed description:	1. User selects **Load tasks** from the **Options** menu. **OR** 1. User starts the Task Manager. The application will automatically load tasks when started. **OR** 1. The mobile client receives an update SMS message from the server and starts loading tasks automatically. 2. (Optional) User accepts a certificate sent by the server.
Postconditions:	User sees a list of tasks in the mobile client.

3.1.2 Complete task

Actors:	Worker or manager
Summary:	Marks the selected task as done in the server database.

Preconditions:	Tasks have been loaded and user has undone tasks.
Detailed description:	1. User selects a task from the list and presses the **Enter** key. 2. User accepts the **Complete this task** prompt by clicking **Yes**. 3. (Optional) User accepts a certificate sent by the server.
Postconditions:	The completed task is removed from the list. User is left with a list of undone tasks.

3.1.3 Interaction diagram

The interaction diagram displayed in Figure 5 shows on a general level how the mobile client use cases relate to the architecture of the enterprise example (see Figure 1).

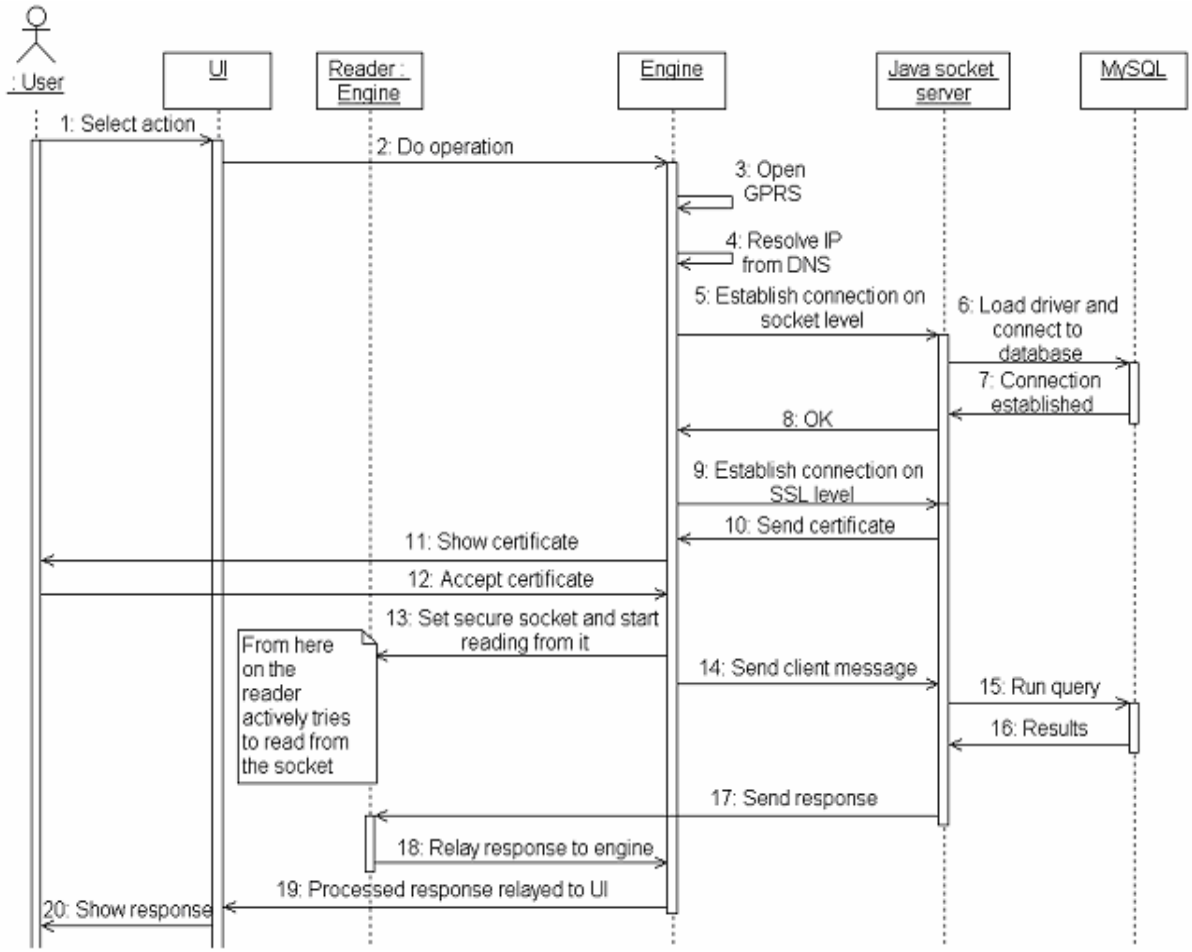

Figure 5: General interaction diagram of the mobile client

3.1.4 Important classes

`TRequest` is a static class that handles building the actual data package that is sent to the server. If at some point you decide to change our message format, you only need to edit this class and `CResponse`. The format of requests is "username#password#operation#messageId" where the final segment is used

only with the operation “mark” (marking a specific task as done). An example request message would be “john#nhoj#mark#15#”.

The `CResponse` class handles parsing the response received from the server into a more manageable form. The server response is in format “size#message”. An example server response to a “fetch tasks” request would be “33#7#File report#8#Prepare meeting#”. If there is a server-side error, the response is “21#Error: Error message”. When marking a task done, the server sends a confirmation message, “3#OK”.

The `CTaskManagerConnForm` class represents the settings form that allows the user to set connection parameters.

`CTaskManagerEngine` is responsible for administering the socket connection to the server and sending data to it.

The `CTaskManagerEngineReader` class reads the data sent to the client from the server. It notifies the engine of a received package.

The `CTimeoutTimer` class is used to prevent the asynchronous function calls from taking too much time.

3.2 Web UI

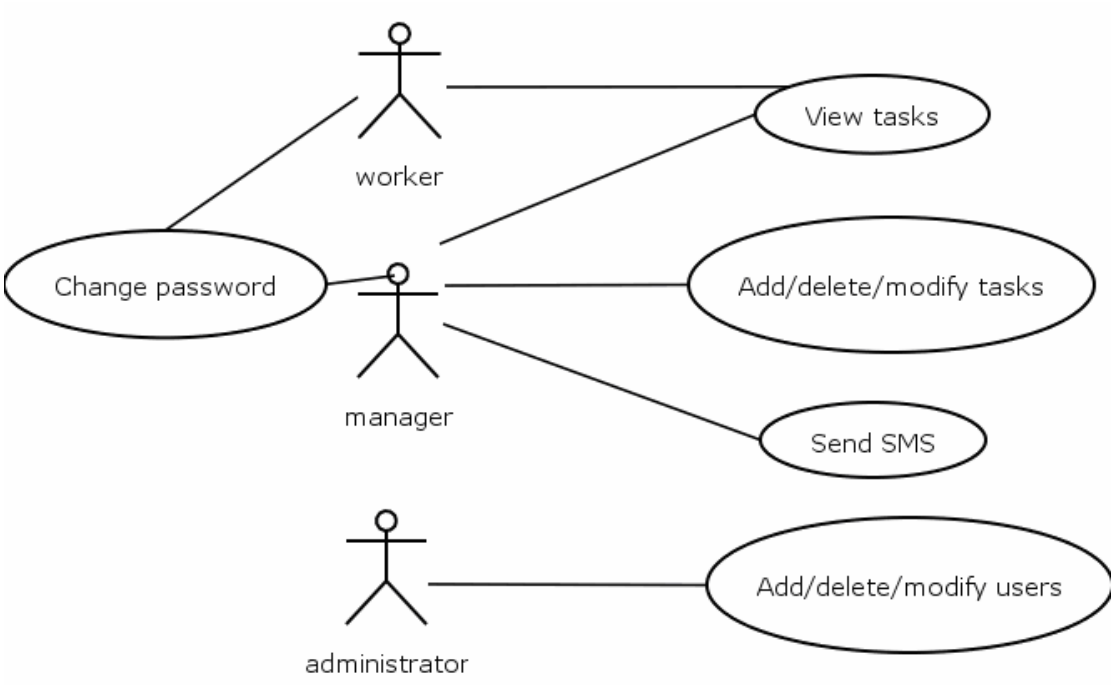

Figure 6: Web UI use cases

3.2.1 View tasks

Actors:	Worker or manager
Summary:	By logging in, a worker will see a list of all of his/her tasks. A manager will see all tasks of every user.
Preconditions:	User has logged in.

Detailed description:	No actions required by the user. A list of tasks is automatically displayed when the user logs in.
Postconditions:	User sees a list of tasks.

3.2.2 Change password

Actors:	Worker or manager
Summary:	User changes his/her password.
Preconditions:	User has logged in.
Detailed description:	1. User types in his/her current password to the **Current password** field. 2. User types in the new password to the **New password** field. 3. User retypes the new password to the **Confirm password** field. 4. User presses the **Change password** button.
Postconditions:	User's password has been changed.

3.2.3 Add task

Actors:	Manager
Summary:	Manager adds a new task to a worker or a manager.
Preconditions:	Manager has logged in.
Detailed description:	1. Manager defines the following parameters for the task: • Owner • Description • Status 2. Manager presses the **Add as new** button.
Postconditions:	A new task appears in the task list.

3.2.4 Delete task

Actors:	Manager
Summary:	A manager deletes a task.
Preconditions:	Manager has logged in and tasks exist.
Detailed description:	1. Manager selects a task from the task list. 2. Manager clicks the **Delete** button.

	3. Manager accepts the **Delete selected task** prompt, by clicking the **OK** button.
Postconditions:	The selected task is removed from the task list.

3.2.5 Modify task

Actors:	Manager
Summary:	Manager modifies task's owner, description, and status.
Preconditions:	Manager has logged in and tasks exist.
Detailed description:	1. Manager selects a task from the task list. 2. Manager modifies task's owner, description, and status. 3. Manager clicks the **Save** button.
Postconditions:	Owner, description, and status of the selected task are updated in the task list.

3.2.6 Send SMS

Actors:	Manager
Summary:	Manager sends an update SMS message to users whose mobile client is not up-to-date with the server.
Preconditions:	Manager has logged in.
Detailed description:	1. Manager clicks the **Send SMS** button. 2. The system sends an SMS message to all users whose mobile client is not up-to-date with the server.
Postconditions:	A list of all users to whom an SMS message was sent is shown.

3.2.7 Add user

Actors:	Administrator
Summary:	Administrator adds a user to the system.
Preconditions:	Administrator has logged in.
Detailed description:	1. Administrator fills in the following parameters for the new user: • Username • Group • Mobile number • Password

	2. Administrator clicks the **Add as new** button.
Postconditions:	A new user appears in the user list.

3.2.8 Delete user

Actors:	Administrator
Summary:	Administrator deletes a user.
Preconditions:	Administrator has logged in and users exist.
Detailed description:	1. Administrator selects a user from the user list. 2. Administrator clicks the **Delete** button. 3. Administrator accepts the **Delete current user** prompt by clicking the **OK** button.
Postconditions:	The selected user is removed from the user list.

3.2.9 Modify user

Actors:	Administrator
Summary:	Administrator modifies user's username, group, mobile number, and password.
Preconditions:	Administrator has logged in and users exist.
Detailed description:	1. Administrator selects a user from the user list. 2. Administrator modifies user's username, group, mobile number, and password. 3. Administrator clicks the **Save** button.
Postconditions:	User's username, group, mobile number, and password are updated in the user list.

3.2.10 Interaction diagram

The interaction diagram displayed in Figure 7 shows on a general level how the Web UI use cases relate to the architecture of the enterprise example (see Figure 1).

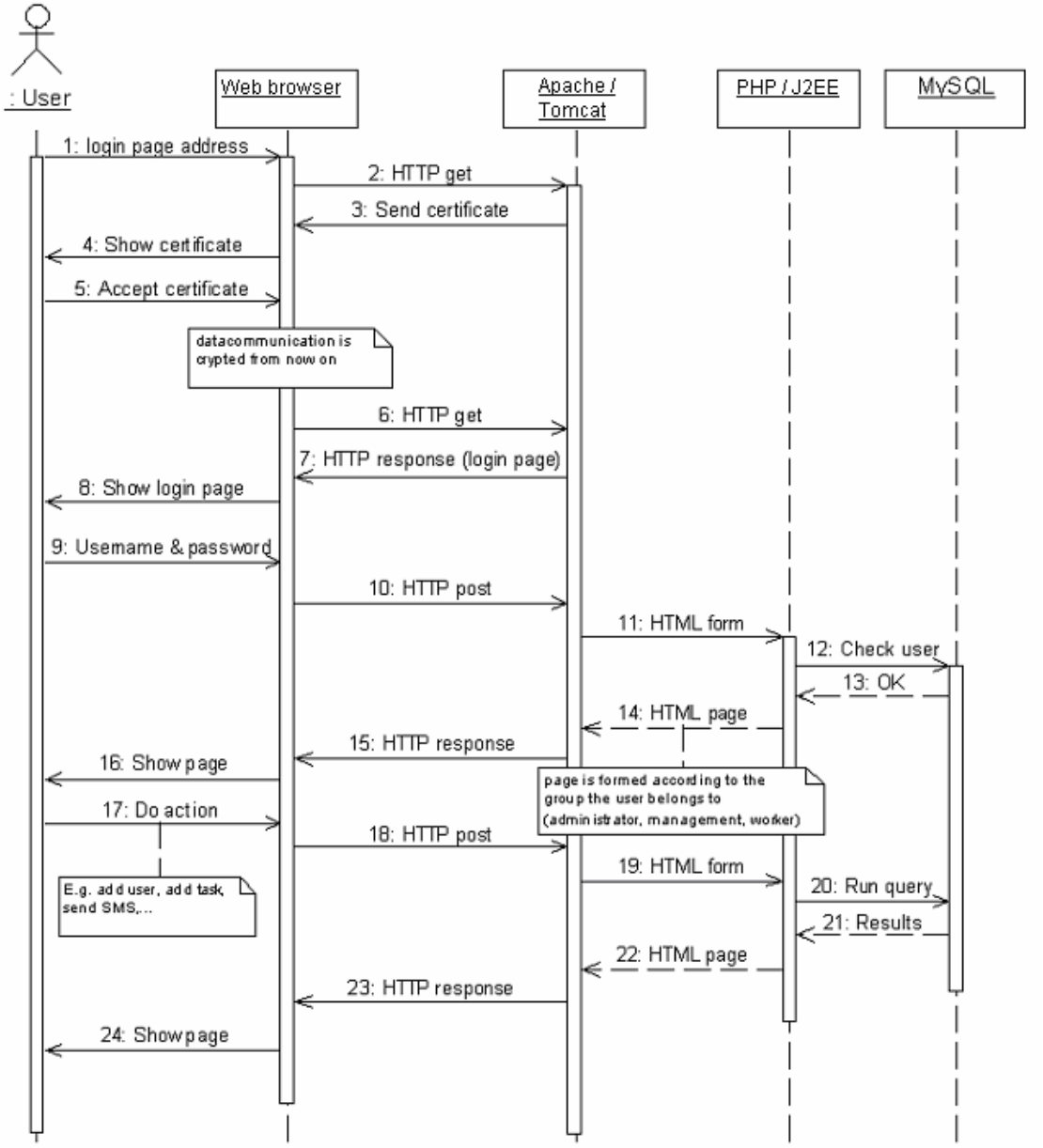

Figure 7: General interaction diagram of the Web UI

4. Installation and configuration

This chapter describes how the client application and the server implementation are installed and configured on a mobile device and server environment, respectively. This example was created and tested in a Fedora Linux server environment, but any server environment could be used. Installing the Linux operating system is beyond the scope of this document.

4.1 Mobile client

This section describes how to get the Symbian OS client application running on a S60 or a Series 80 device.

4.1.1 Installation

Installation, building, and operating instructions for the S60 and Series 80 client can be found in the Release Notes, which are included in `SocketTaskManager_S60.zip` and `SocketTaskManager_Series80.zip`, respectively.

4.1.2 Certificate

The Symbian client prompts the user to accept a certificate every time a transaction with the server is needed. This can be avoided by installing the certificate to the Symbian device. First export the certificate from the keystore with keytool using the following command (in one line):

/usr/java/jdk1.5.0_01/bin/keytool -export -alias mykey -file server.crt -keystore myKeystore

This command exports the certificate to a file named `server.crt`. Then just send the certificate to the Symbian device, for example, with Bluetooth and save the certificate to the device.

The `-alias mykey` parameter specifies the certificate you want to export. By default, a newly generated key is given the alias “mykey”. You can change this when generating the key by adding the parameter “-alias <name of key>” to the command.

4.2 Web server

This section describes how to install the Apache Web server.

4.2.1 Installation

The Apache Web server is very easy to install and configure. Just download the Apache Web server, for example, from ftp://ftp.funet.fi/pub/Linux/images/fedora/linux/core/3/i386/os/Fedora/RPMS/httpd*. Then install it with the command:

rpm -ivh httpd*

No other configuration is needed at this point.

4.3 Java server

4.3.1 Installation

First of all, you need J2SE SDK 1.4.2 or higher (http://java.sun.com). This reference implementation may work with an older version but it is not guaranteed. You also need Apache Ant (http://ant.apache.org), a Java build program. This is not mandatory but significantly eases the compiling and running process. Finally, you need the MySQL Connector/J (J for Java) from http://www.mysql.com. This is a driver for Java's JDBC (Java Database Connector). Refer to the respective installation guides for installation instructions.

Copy the files into a folder on the server. Go to the project root directory (where the Ant build file `build.xml` resides) and run "ant" or "ant compile". However, before running the server you need to create a certificate for the SSL connection.

4.3.2 Configuration

Install Ant and MySQL Connector/J and create the certificate (see Section 4.4.1.1,"Creating a certificate"). Add the server URI, username (if not the default, taskdb), and password to the `TaskManagerServerThread.java` file (change the constant's values defined in the beginning of the file). Recompile the project.

4.3.3 Running the server

If you are running the server from command line, initialize the server with added parameters: `java -Djavax.net.ssl.keyStore=myKeystore -Djavax.net.ssl.keyStorePassword=qwerty com.nokia.forum.taskmanager.comm.TaskManagerServer`. The `keyStorePassword` argument should match the password that you chose for the keystore (see Section 4.4.1.1, "Creating a certificate" or Section 4.4.2, "Configuring SSL for J2EE").

If, however, you are using ant, just type "ant run" to run the server. You need to change the `build.xml` file a bit if your keystore's name is not myKeystore and your password is not "qwerty". Editing `build.xml` should be self-explanatory. The idea is to find the parameters specified above for the command line Java interpreter and replace them with your chosen parameters (keystore password and name of the keystore).

The server should be able to start, assuming that the keystore file is in the right place, but connecting to it will fail if you haven't configured the MySQL server yet. The server runs until you kill the process (for example, by pressing CTRL-C).

4.4 Security (SSL)

This section describes how SSL is enabled for PHP J2EE. Again, you only need to configure either PHP or JSP.

4.4.1 Installing and configuring SSL for PHP

SSL's keys can be created through Java SDK's keytool program, found in `$JAVA_HOME/bin`. You have installed keytool when you have installed the SDK.

4.4.1.1 Creating a certificate

SSL uses asymmetric keys (RSA to be more precise) for enabling data to be transmitted securely between the client and the server. A certificate is used to positively authenticate the server's identity to ensure that data is not sent to an impostor site. A test certificate for this example is created as follows:

keytool -genkey -keystore myKeystore -keyalg RSA

This command creates a new keystore named `myKeystore`, the place of storage for your certificate and keys. It also generates a public/private key pair. You will be prompted for a password for the keystore and for the private key.

When generating the certificate everything else may be left as "Unknown" except for the two-letter country code. If you fail to give a proper code, the certificate will not be valid. For example, for Finland the proper code would be "fi".

Copy the created keystore (which houses the certificate) into the project root directory.

4.4.2 Configuring SSL for J2EE

In the default Tomcat `server.xml` file you should find a ready SSL connector commented out. It should have a preceding comment saying "Define a SSL HTTP/1.1 Connector on port 443" or something similar. Uncomment the <Connector>-tag and change the port according to your preference. You also need a keystore in the user home directory to host an SSL certificate. Alternatively you can define a new location from where Tomcat will search for a keystore by adding the keyStoreFile="<location of the keystore>" attribute to the <Connector>-tag.

You can create a new keystore with the keytool application which can be found from all Java Development Kits, under the /bin directory. To create a new keystore to the present working directory, write:

keytool -genkey -keystore myKeystore -keyalg RSA.

You will be prompted for a password that must be set as "changeit" so that Tomcat can access the keystore. Also make sure you specify a two-letter country code when keytool asks for one in the creation process, for example, "fi" for Finland. Otherwise the certificate may not be accepted by the mobile client or the browser.

More detailed information on enabling SSL can be found in the Tomcat documentation on the Tomcat Web site (http://jakarta.apache.org/tomcat).

4.5 Database

This section describes how to install and configure the MySQL database. It will also describe the database structure of this example.

4.5.1 Installation

- Download MySQL binaries from http://www.mysql.com. Alternatively, using an `.rpm` file is an easy way and it is recommended. After you have downloaded the `.rpm` file, just run `rpm –install filename.rpm`.
- You will need to install the MySQL server and MySQL client packages. The `.rpm` packages are named `MySQL-server-VERSION.i386.rpm` and

`MySQL-client-VERSION.i386.rpm`. The order in which you install these is not important.

- RPM installation also creates a new user named “mysql” into your system. Some versions will also change your system configuration so that the MySQL server will start automatically every time you start your system. Changing these settings is beyond the scope of this document.
- After installing MySQL, run the included setup script which sets up all the necessary MySQL user accounts, tables, and roles. You can execute the script by copying it to your working directory, starting MySQL as the root user, and running the `source setup.sql` command. The script will create a MySQL user account “taskman” that has a password “namksat”. You should change this password to something else either by editing the script before executing it or by hand after sourcing the script.
- The script will also create a new user account “administrator” with a password “admin” that you will need to create new users to the system. The MySQL account is needed for logging into MySQL from the server application and the PHP scripts, whereas this account is used when you add new users from the Web UI.

4.5.2 User table

The user table stores information on all of the system’s users, including those that only administer other users. Each user has the following attributes: an ID as its primary key, a unique login string, a password string, and a string to its mobile phone number. A user also has a state enumeration that specifies whether the user’s tasks in the mobile client are up-to-date or not. All tables also use MySQL’s InnoDB extension that allows more restrictions on foreign keys (more on those later).

4.5.3 Task table

The task table stores all the tasks added to the system. Each task has a unique ID number, a foreign key to the user that owns the task, a description, and a state that specifies whether or not the task has been completed. The foreign key is determined as “on delete cascade” meaning that when a user is deleted from the list, all tasks pointing to it are also removed. This ensures that no ownerless tasks can be found in the task table. This feature requires using the InnoDB extension, since InnoDB-type tables are the only table types in MySQL that support foreign key constraints.

4.5.4 Roles table

The roles table stores information on different roles in the system. The system requires three roles to function: administrator, manager, and user. These roles are set up in the setup script.

4.5.5 Userrolemap table

This table is redundant in the current implementation but if a need would rise to have multiple roles on a single user, it would be needed.

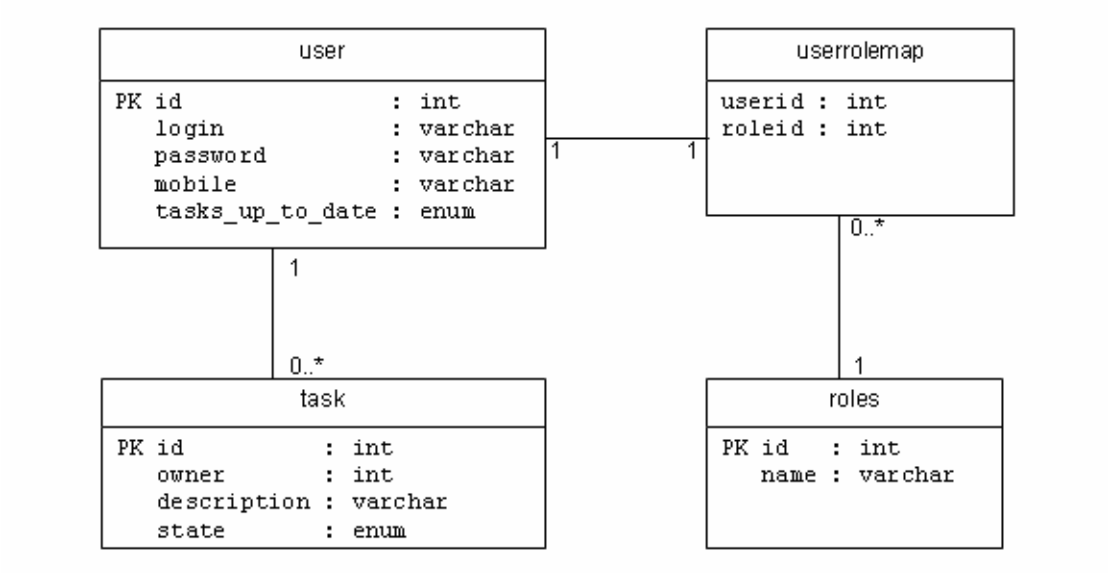

Figure 8: Database tables

4.6 Configuring Web UIs

This section describes configuration details for the two user interface types.

4.6.1 Installing and configuring PHP

PHP is installed as an Apache module so that the whole Apache does not have to be recompiled each time PHP is updated. Download the latest PHP (version 4.3.10 used in this example) from www.php.net. Extract the PHP package:

tar xfz php_4.3.10.tar.gz

In the PHP directory (directories may vary in PHP and Apache builds), type the following commands:

configure --with-mysql --with-apxs2=/usr/sbin/apxs

make

make install

PHP is now installed into the appropriate Apache directories. To enable Apache and PHP to work together, add the following line to the `/etc/httpd/conf/httpd.conf` file:

```
AddType application/x-httpd-php .php
```

If you need to use, for example, iso-8859-1 character set in your PHP pages, make sure that this is defined in the `php.ini` file. If the `php.ini` file does not exist, just create it to `/usr/local/lib` and insert the following line:

```
default_charset = "iso-8859-1"
```

The last thing to do is to restart Apache. The command to do this is:

/usr/sbin/apachectl restart

4.6.1.1 PHP & MySQL

The current version of PHP (4.3.10) does not support the new authentication protocol in MySQL 4.1.1 and higher. Therefore, the MySQL database has to be configured so that it will use the old authentication protocol. Create the `my.cnf` file to the `/etc/` directory, if one does not yet exist, and add the following lines to the file:

```
[mysqld]

old_passwords
```

Also, when a new user is added to the user table (see Section 4.5.2,"User table"), the password for the user has to be generated with the OLD_PASSWORD macro as seen in the PHP files of this example.

4.6.1.2 PHP files

Place all the PHP files of this example to the directory you defined as the SSL directory in the `DocumentRoot` directive (see Section 4.3.1,"Installation").

4.6.2 Setting up J2EE

This section describes how to set up the Java 2 Enterprise Edition version of the Web UI, using the Apache Tomcat 5.5.x servlet container.

To use JavaServer Pages you need a servlet container software. Apache Tomcat is ideal for this use since it is free and is also the official reference implementation of the Java Servlet and JavaServer Pages technologies, developed by Sun. This example was developed and tested using Tomcat 5.5.9. It works with this and newer versions of Tomcat but not with older versions (since the example uses JSP 2.0, to which support was introduced in 5.5.9).

4.6.2.1 Preparation

Get the Java 2 Enterprise Edition SDK if you are going to compile the code files. In case you only wish to run the example, there is an included `.war` file that you can simply deploy to the servlet container in which case J2EE SDK is not needed. You will need a version 5.0 Java 2 Standard Edition Java Runtime Environment (JRE) to run Tomcat 5.5. You can find the JRE at http://java.sun.com/j2se.

4.6.2.2 Tomcat installation and configuration

Download Tomcat 5.5.9 or newer from http://jakarta.apache.org/tomcat. To install Tomcat, simply extract the downloaded archive and read the included RUNNING.TXT for further information. After you have successfully installed Tomcat, you still need to configure it to run in a port of your choosing and to use SSL. You do this by editing the file `$CATALINA_HOME/conf/server.xml` (where $CATALINA_HOME is the installation directory for Tomcat).

4.6.2.3 Installing the Web application to Tomcat

When Tomcat has been successfully installed and is responding from the port to which it was configured, you are ready to deploy the Web application. There are several ways to do this.

1. You can deploy the included `.war` file that includes all the files needed to run the Web application through the Tomcat manager application. The manager application can be found from the Tomcat front page, which in turn is located at the address and port to which you configured Tomcat. Install it from the "WAR file to deploy" section.

2. You can simply copy the `.war` file to the `$CATALINA_HOME/webapps` directory. This way, Tomcat will extract your Web application automatically during startup.

3. You can deploy with ant, assuming that you are deploying to the same machine you are running ant from. Just make the necessary modifications to the ant buildfile (`build.xml`, target "install") found in the project root directory. You need to set a valid manager url, username, and password depending on your Tomcat settings. Installation can be executed with "**ant install**" which will also compile your project, if necessary. Manual compile can be done with "**ant compile**".

To get the application to work with a database, you will need to make changes to the code and recompile it. The most relevant change should be made to the `DBAccess.java` file. You need to redefine the username and password constants so that they match your database.

You also need to make changes to the `build.xml` file if you wish to deploy straight to Tomcat with Ant. The parts that need to be changed are well commented in `build.xml`.

5. Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by rating this resource.