Software Engineering Group Project
Design Specificaion

Author:	L. Jones, T. Oram, T. Garapasi, M. Goly, W. Jones, A. Neaves, J. Mir, T. MIlls
Config. Ref.:	SE-12-DS
Date:	2016-02-12
Version:	1.7
Status:	Release

CONTENTS

1 INTRODUCTION

1.1 Purpose of this document

The purpose of this document is to outline the ways in which our systems work and communicate with one another, explain how users will interact with the system as a whole and to provide a basic user-interface design for TaskerMAN and TaskerCLI.

1.2 Scope

This document covers the design and architecture aspect of the assignment, covering deployment description, interaction design, component description, significant classes and detailed design.

This document should be read by all project members. It is assumed that the reader is already familiar with the Design Specification Standards [1].

1.3 Objectives

The objectives of this document are as follows:

- Explain the way the systems interact with one another and the platforms in which they operate with the asisstance of UML diagrams.

- Provide detail on the way in which "actors" are expected to interact with the system through the use of use-case diagrams.

- Present a basic User-Interface design for the TaskerMAN and TaskerCLI systems with the aid of images to demonstrate how such systems will operate and keep to the requirements set in the Requirements Specification [2].

2 DEPLOYMENT DESCRIPTION

2.1 Applications in the system

The deployment UML diagram below illustrates the division of the software system into separate applications and the platforms on which they will be deployed.
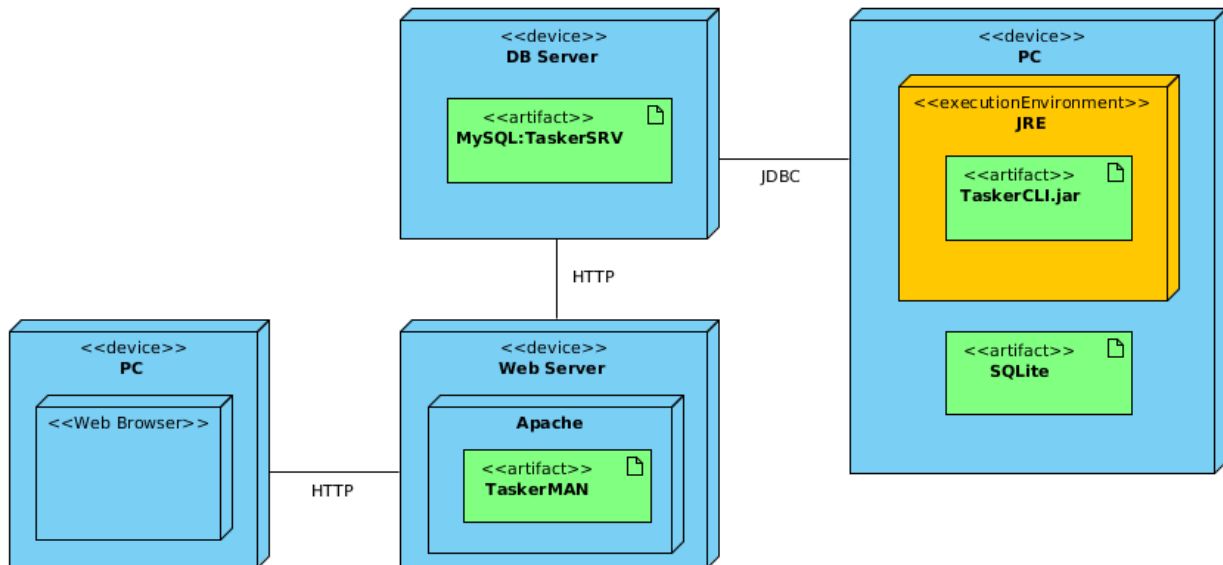
The whole system will be composed of the following three components:

- TaskerSRV (Database)

- TaskerMAN (Web Client)

- TaskerCLI (Desktop Client)

TaskerSRV will be deployed onto a database server with a pre-installed MySQL engine. It will store the critical data of the whole system and make it accessible to both web and desktop clients through the specified protocols. TaskerMAN will be deployed as a PHP web application onto an Apache web server and it will be accessible from the Internet by any client with an installed web browser. Finally, TaskerCLI will be a desktop application written in Java, and it will be deployed as a runnable jar onto a machine running the Java Runtime Environment. Following the requirements specification[3], TaskerCLI will have to operate both on-line and off-line. Therefore it will have a direct access to a local storage provided by the SQLite database.

2.2 Applications interface

Unlike the deployment diagram above, the component diagram below depicts the interaction of different modules or components of the proposed system. There are four major components that form the entire system. Only a black box view is shown here. This means that the components are not further decomposed to reveal the inner components. The opposite of this is a white box view which shows not only the major component but also the inner components that form the big picture. According to the design specification standard[1], only the simple black box view application interaction is relevant for now. A rundown of the actual interactions is as follows:

TaskerSRV - This component, representing the database, consists of one provided interface (solid circle) and two required interfaces (cup shaped). These are:

- SQL(provided interface) : Provides an SQL interface, making it possible for the web application, TaskerMAN, to manipulate the database and carry the necessary operations outlined in the requirements[2] such as adding, editing, deleting users or tasks.

- HTTP (required interface): Without the HTTP protocol the database would not be accessed by the TaskerMAN and no operation can take place.

- JDBC (required interface): The interface is required so that communication to the SQL can be established and give the required services to the client.

TaskerMAN - This component, representing the web application has one required interface (SQL) and one provided interface (HTTP). Without SQL provided by TaskerSRV, TaskerMAN will not be able to do anything. It will remain a useless dummy. Also if it can not provide the HTTP communication interface then it would not be able to access the database.

TaskerCLI - This component representing the client has one provided interface (JDBC). Since it is using Java to connect to the database, TaskerSRV as well as SQLite will require that the interface JDBC be implemented on the client or else there will be no database connection.

SQLite - This component representing the local database on the client has one required interface (JDBC) which makes it possible for the client to connect to it.

3   INTERACTION DESIGN

3.1   Use-cases

3.1.1   TaskerMAN Use-Case
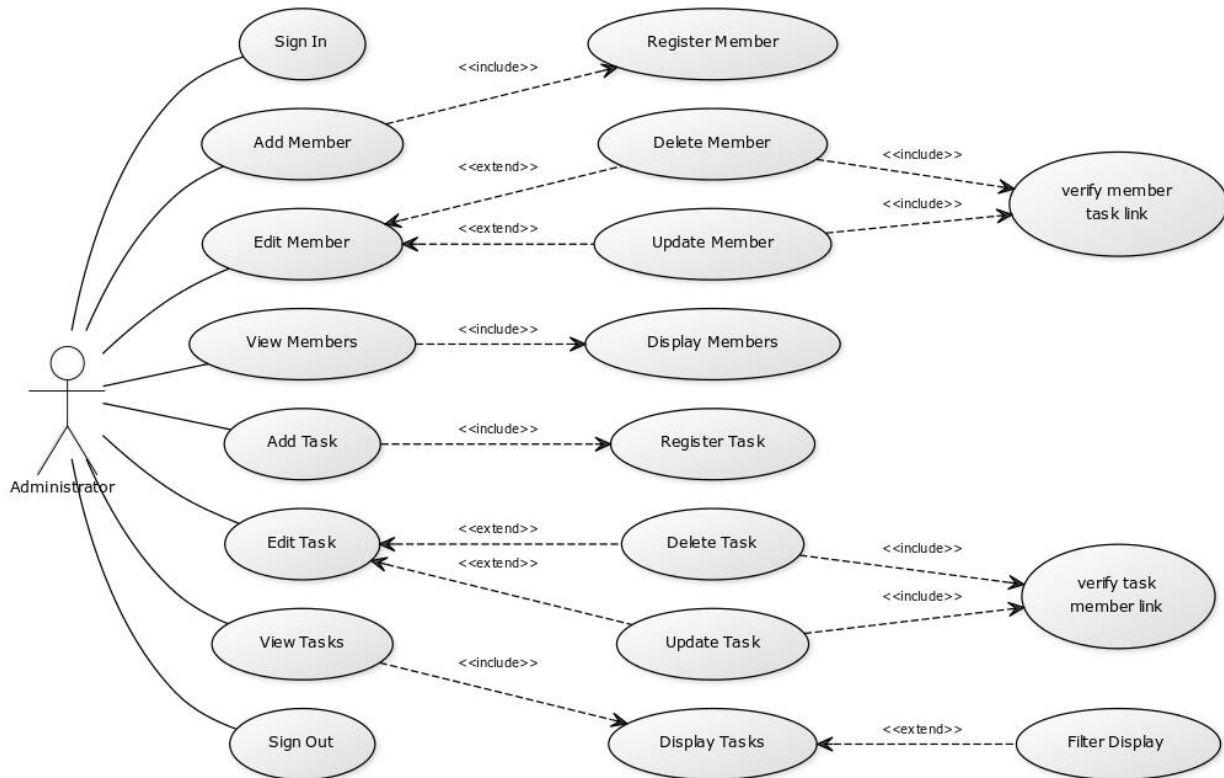
The TaskerMAN use case diagram above depicts clearly how the web administration application will be able to be used by the administrator. The necessary functions depicted in the specification requirements[2] are clearly accessible to the administrator after logging into the system. These functionalities are, add task, edit task, view tasks, filter display, add and edit member. The options update member and delete members as well as update task and delete task are labelled as extending. This is because they are options and an administrator may or may not choose to delete a task or member. Also they include a verification so that the administrator might be aware, for example, that a member to be deleted or updated might be currently assigned a task or vice versa. Similarly the filter display is extended since an administrator may choose to filter display results or not. Example usage scenario:

1. The administrator logs onto the web application system.

2. Selects "add member" and registers the members.

3. Administrator clicks on "view members" to make sure the members are added.

4. Administrator realises that a member name was misspelled. He or she clicks on "edit member" and selects the option "update member". Necessary updates are done.

5. Administrator then clicks on "add task" and a necessary form is displayed where data is entered.

6. Administrator decides that the task was allocated to the wrong member. He or she clicks on "edit task". An appropriate page is loaded where the task can be selected and a reallocation can be carried out.

7. Administrator decides that one more task is actually not needed. He or she clicks on "edit task", selects the task and clicks on "delete task". A prompt is displayed informing that the task is associated with a member. If the administrator is comfortable to delete, then the action is executed.

8. Administrator clicks on "view tasks" and all the tasks displayed. An option is also available for the administrator to filter display results accordingly as specified in the requirements[8].

9. The administrator has finished using the system and he or she clicks on "log out" and the session ends.

3.1.2 TaskerCLI Use-Case

UML USE CASE DIAGRAM: TaskerCLI
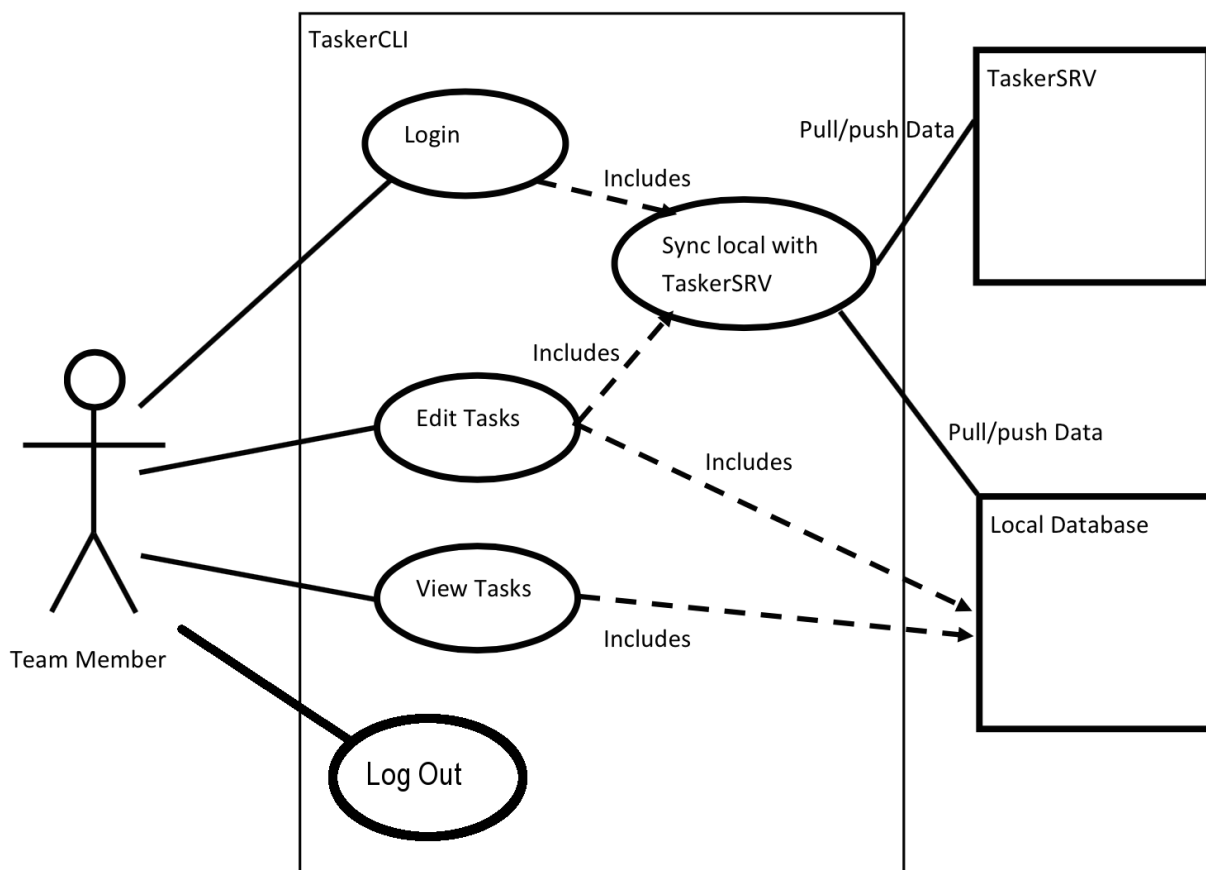
The TaskerCLI use-case diagram above depicts how TaskerCLI will be used by "actors". The diagram illustrates the different functions the application must contain as outlined in the requirements specification [2]. These are user identification (login), local storage of tasks, task synchronisation and the function to edit tasks, both locally and to TaskerSRV. Example usage scenario:

1. The actor logs onto the Java application.

2. The application synchronises with TaskerSRV and pulls the tasks assigned to the actor and stores them to the SQLite database.

3. The actor clicks on 'View Task' and the task information is loaded from the SQLite database and displayed on-screen.

4. The actor confirms changes and clicks 'Edit Task', this will trigger another synchronisation to occur, and the changes will be pushed to TaskerSRV and to the SQLite database.

5. The user has completed their actions and clicks 'Log Out', terminating the system.

3.2   User Interface design

3.2.1   TaskerMAN Interface Design
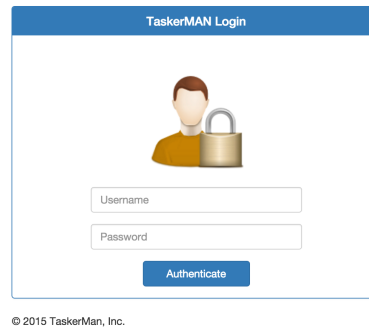
This is the authentication page of our website with the address '/taskerMAN/login.php'. Upon attempting to enter any page of the website, should a local session not exist users will be redirected to this page where they will be required to enter their login credentials. If the credentials entered are incorrect then the screen will simply refresh, and the fields will appear blank again. If the credentials are correct then the user will be redirected to the home page of the website. If at any point in the user's session they click the logout icon in the top right of any of the website's pages, the session will be destroyed and they will be redirected back to this authentication page.
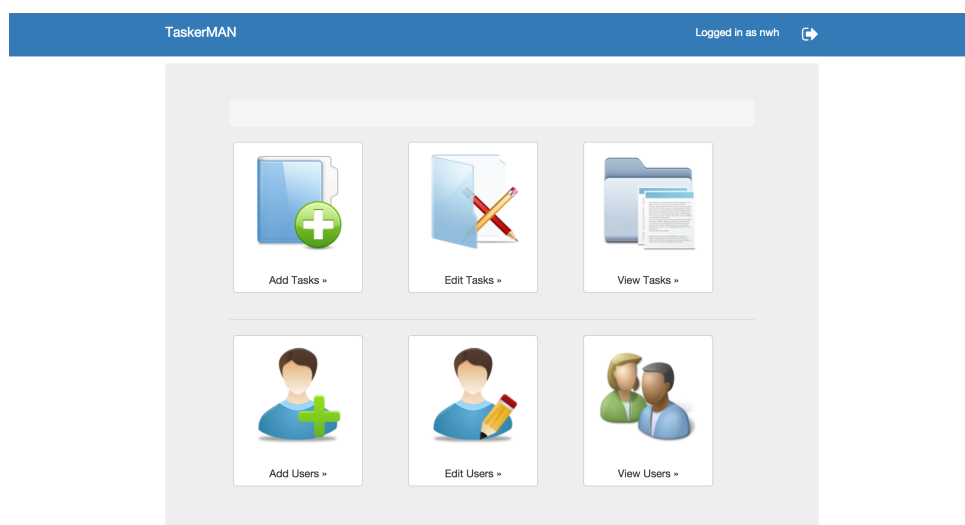
This is the home page of the web interface with the address '/taskerMAN/index.php', it is very easy to use and presents you with six options which provides very quick access to all aspects of the Tasker. To begin we will click on the 'Add Users' button. This will re-direct us to the 'Add User' page.

© 2015 TaskerMan, Inc.

Here within the 'Add User' page with the address '/taskerMAN/adduser.php', you will be presented with the form above which enables you to create a user by submitting their name, e-mail and password. Upon pressing the 'Add User' button the new member will be added to the database as required [2]. Clicking on the 'backwards arrow' in the top-left corner allows users to return to the home page. In fact the 'backwards arrow' exists within all of the sites internal pages, and has the same function as clicking on the 'TaskerMAN' logo in the top-left corner which is to return to the website's home page.

To update or delete user data as required [2] click on the 'Edit Users' button from the home page and you will be presented with this page with the address '/taskerMAN/editusers.php', displaying user's names, e-mail addresses and options to update/delete their information, clicking on the 'update' icon of a user re-directs to a page with the address '/taskerMAN/updateusers.php' which has the same interface as the 'Add Users' page where name and password can be updated, clicking on the 'delete' icon will prompt the user to verify their action with an 'Are you sure you want to delete this user?' message with buttons for 'yes' and 'no'. Clicking on 'no' will redirect the user back to 'Edit Users' portal, clicking on 'yes' will cause a delete command to be submitted to the database to remove the user from the database, a message displaying this action will be displayed on screen.

Clicking 'Add Tasks' from the home page will re-direct to the task portal with the address '/tasker-MAN/AddTaskBoot.php'. This is where users can make tasks which will then be allocated to members in the remote database as required [2]. This task portal layout will also be used when updating tasks, the only difference being that when updating a task there will be an option for the user to set a task as 'Abandoned' as required [2]. 'Task Title' and 'Task Element' will be text fields for user input, whereby clicking on the '+' icon will cause another 'Task element' field to continuously drop below the previous field. The field where you specify which user the task will be allocated to is a drop down field containing all of the members currently present in the remote database.' Start Date' and 'End Date' are drop down calendars created with JavaScript, this allows us to make it impossible to allow users to set an end date which is before the start date. The 'Add Task' button then adds the task to the database and will then sync up with the client, a confirmation screen is then shown to show the user that the task has been successfully added.

Clicking 'View Tasks' from the home page will redirect to this page with the address '/tasker-MAN/viewtasks.php'. Here you can view all the tasks, as well as their title, expected completion date, member assigned to and task status. By default the tasks are sorted by expected completion date as required [2], with the most impending tasks displaying at the top. This can be changed however by clicking the drop down list to filter the results, either by member assigned to or by task status. To view the task elements related to a task you click on the 'More' button of a task.

© 2015 TaskerMan, Inc.
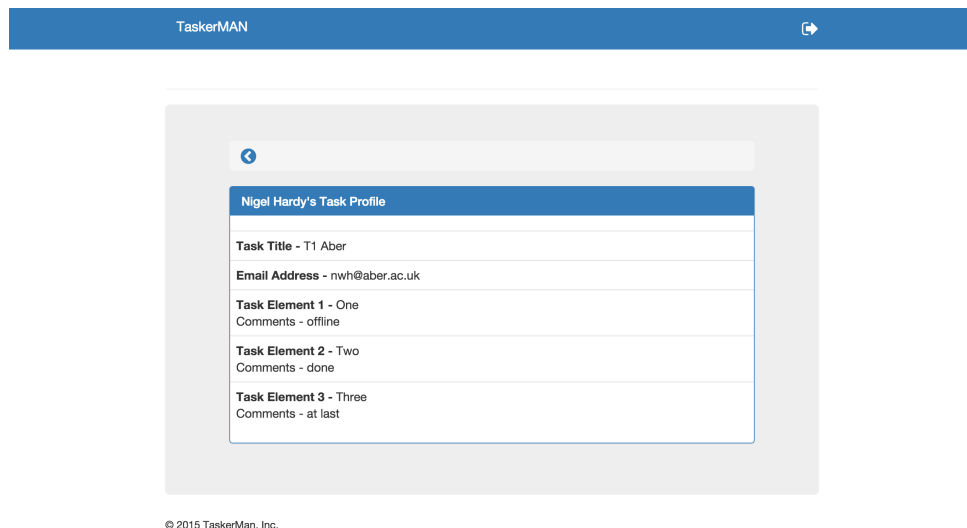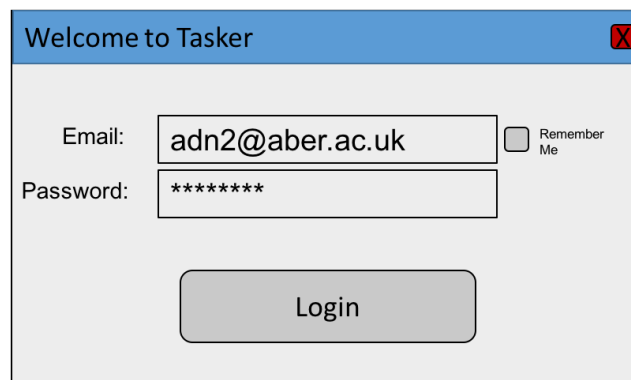
Here is the display of clicking on the 'More' button of a particular task. This view allows the user to view a task, the user assigned to the task, the task elements associated with the task, as well as comments made about the task elements from the TaskerCLI application as required [2]. The task elements and comments are fetched from the remote database.

© 2015 TaskerMan, Inc.
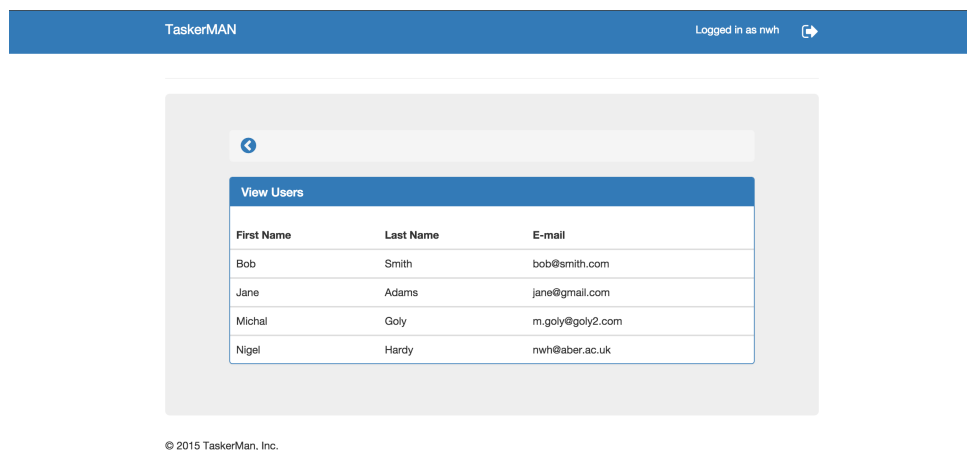
From the home page of the website clicking on 'View Users' will display the view users page with the address '/taskerMAN/viewusers.php'. This page simply fetches all the users currently present in the remote database and displays them on screen with their first name, last name and e-mail.

3.2.2 TaskerCLI Interface Design
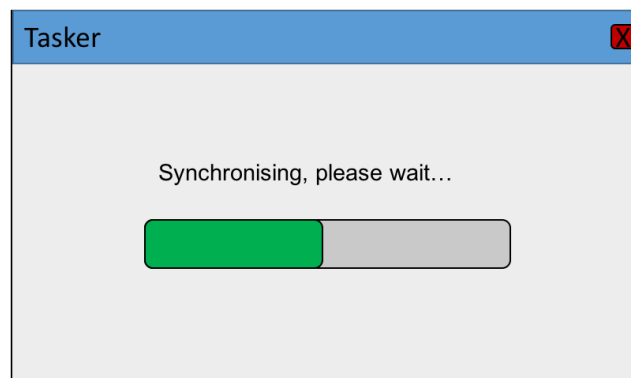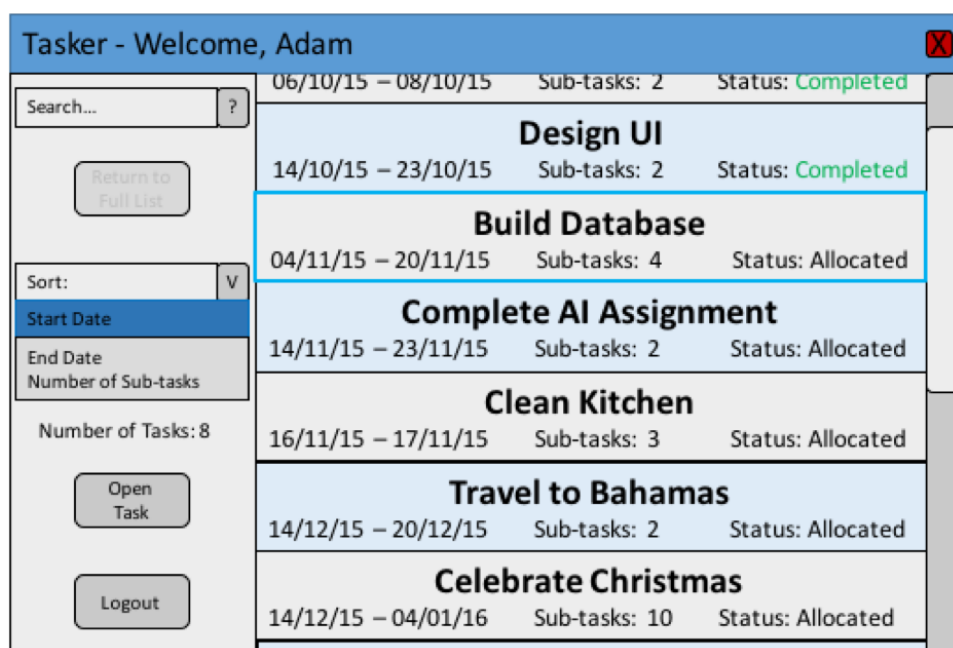
The login screen, the first screen to appear when the client loads. TaskerCLI does not synchronise with the database until the user has logged in as specified in Requirements Specifications.[6]

The synchronisation screen occurs between the login and main client page. As per the requirements specifications [7] this screen will also occur before and after local editing if network access is available, as well as occurring every 5 minutes by default should no local editing take place. This will display the relevant information from the database as it's downloading the data. Ideally this will be a fast enough task that this screen should be seen for as little time as possible.

The 'Main Screen' of TaskerCLI. It shows a list of tasks assigned to the person who signed in, as well

as options for sorting and searching through the list. Clicking on the 'Log Out' button will re-direct the user back to the login authentication screen. Double clicking a task's panel, or selecting a task and clicking the 'open task' button shows more detail about the task [8], as well as the editable parts, such as the completed/allocated attribute and the sub-task list comments. In this example we'll double-click on the "Build Database" task.

This screen shows more details about the "Build Database". It displays all the task's information and allows the user to edit task step comments and change task's status [8].

In this image we've changed the status to 'Completed'. We can press the 'Return to List' button to return to the 'Main Screen'. Doing so will prompt the synchronisation screen to appear and an attempt will be made to connect to the network and update the information on the remote MySQL database. If unsuccessful the user will still be re-directed back to the 'Main Screen', however the client will be now operating locally using the local SQLite database until a connection with the MySQL can be re-established.

Note the changes made are now reflected in this list. Also note that the task named 'Clean Kitchen' has disappeared. This is something that could happen if said task were marked 'Abandoned' via TaskerMAN while the user was editing a task within TaskerCLI, provided there is a network connection.

Searching using the text box searches for the string provided in the task's name, and/or it's sub-tasks. Also note that the 'return to full list' button is only selectable if there is currently a search parameter in the search box.

There is a large focus on designing the interfaces of both TaskerCLI and TaskerMAN to be intuitive to regular computer users [9].

4   Component Description

4.1   TaskerMAN Component Description
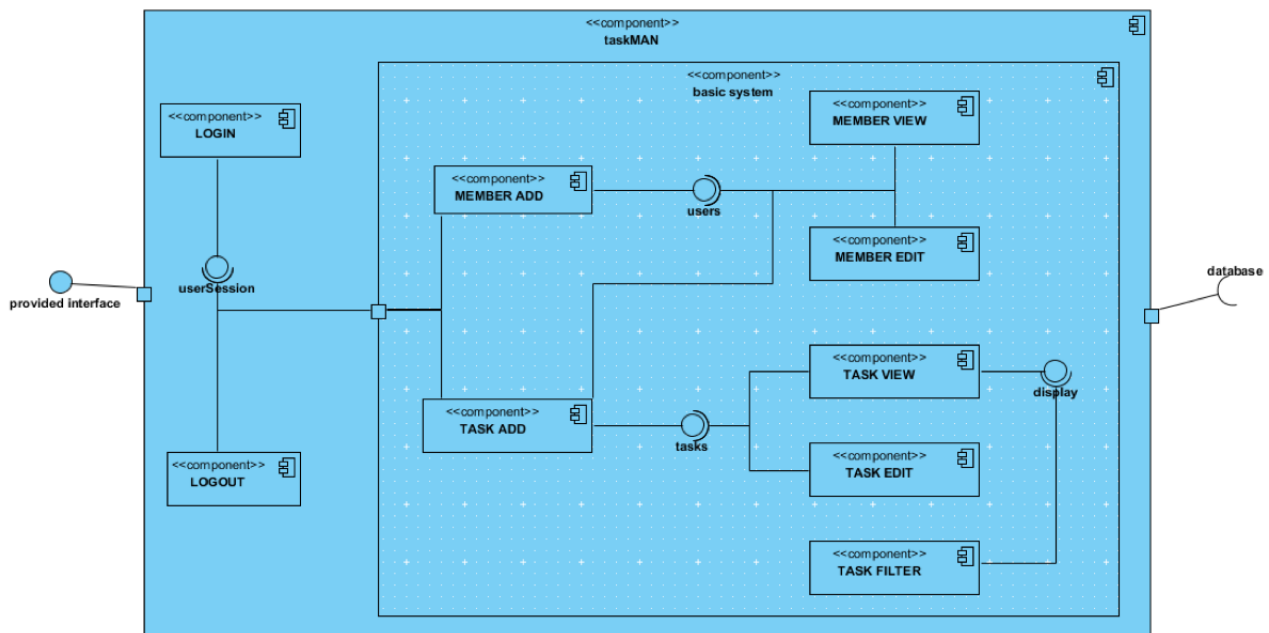
The component diagram above shows a further and advanced decomposed version unlike the previous diagram submitted before. As can be seen, two external interfaces are provided with one connecting to the database and the initial user interface which through a given port lets us use the login component which gives access to the basic system. The first point to using the basic system is either through member add or task add as shown. These two sub components provide respective interfaces to other sub components. For example its clear that the TASK_ADD component requires an interface from MEMBER_ADD, at the same time it also provides an interface which is then used by TASK_VIEW and TASK_EDIT component. TASK_VIEW sub component also provides an interface which is then used by the TASK_FILTER sub component. The LOGOUT sub component is shown as requiring a session established after the LOGIN component has been initiated.

5   Significant Classes

5.1   TaskerCLI Significant Classes

We are using the Data Access Object design pattern, to abstract our database contents and more easily access it from within the Java client. We have abstracted our database tables into Java entities, as an example, in the image below TaskElement class corresponds to the TaskElement table in the databases. In order to access the abstracted data from the database we use a selection of data access classes (see below). These classes apply the DAO pattern, and call methods in order to send properly formatted SQL queries to the appropriate database, and present the result of the queries in the form of the previously described entities. This object is then used by the UI classes in order to generate the view.

util

GBC

+GBC(gridx : int, gridy : int)
+GBC(gridx : int, gridy : int, gridwidth : int, gridheight : int)
+setAnchor(anchor : int) : GBC
+setFill(fill : int) : GBC
+setWeight(weightx : double, weighty : double) : GBC
+setInsets(distance : int) : GBC
+setInsets(top : int, left : int, bottom : int, right : int) : GBC
+setIpad(ipadx : int, ipady : int) : GBC

business

TeamMember

<<Property>> -firstName : String
<<Property>> -lastName : String
<<Property>> -email : String
<<Property>> -password : String
<<Property>> -taskList : Task

+TeamMember()
+TeamMember(firstName : String, lastName : String, email : String, password : String, taskList : List<Ta...
+hashCode() : int
+equals(obj : Object) : boolean
+toString() : String

Task

<<Property>> -taskId : Long
<<Property>> -title : String
<<Property>> -startDate : Date
<<Property>> -endDate : Date
<<Property>> -status : TaskStatus
<<Property>> -taskElementList : TaskElement

+Task()
+Task(taskId : Long, title : String, startDate : Date, endDate : Date, status : TaskStatus, taskElementList : List<TaskEleme...
+hashCode() : int
+equals(obj : Object) : boolean
+toString() : String

TaskElement

<<Property>> -taskElementId : Long
<<Property>> -description : String
<<Property>> -comments : String

+TaskElement()
+TaskElement(taskElementId : Long, description : String, comments : St...
+hashCode() : int
+equals(obj : Object) : boolean
+toString() : String

<<enumeration>>
TaskStatus

-status : int

~TaskStatus(status : int)
+fromInt(integer : int) : Tas...
+toInt() : int

ALLOCATED
ABANDONED
COMPLETED

logic

Syncer

+logIn() : TeamMember
+sync(remote : TeamMember, local : TeamMember) : TeamMe...
-merge(remote : TeamMember, local : TeamMember) : TeamMe...
+doUpdate(editedTeamMember : TeamMember) : void

Authenticator

+authenticate(teamMember : TeamMember, password : char...

TimerManager

+MINUTES_TO_WAIT : int = 1
+DELAY : long = TimeUnit.MINUTES.toMillis(MINUTES_TO...
+TimerManager()

SyncTask

+run() : void

frontend

ViewAllTasksListener

+actionPerformed(e : ActionEvent..

SearchButtonListener

+actionPerformed(e : ActionEvent..

LogOutButtonListener

+actionPerformed(e : ActionEvent..

SidebarPanel

-mainFrameTable : JTable
-searchField : JTextField
-taskNumberLabel : JLabel
-sorter : TableRowSorter<TableModel>
-searchButton : JButton
-viewAllTasks : JButton
-logoutButton : JButton
-onlinePanel : OnlineIndicatorPanel

+SidebarPanel(mainFrameTable : JTable)
+updateTaskCount() : void
-initComponents() : void

OnlineIndicatorPanel

-label : JLabel = new JLa...

+OnlineIndicatorPanel()
+initialise() : void
+setOnline() : void
+setOffline() : void

TaskElementTableModel

-taskElements : TaskElement

+TaskElementTableModel(task : Task)
+getRowCount() : int
+getColumnCount() : int
+getColumnName(columnIndex : int) : String
+getValueAt(rowIndex : int, columnIndex : int) : Object
+setValueAt(value : Object, rowIndex : int, columnIndex : int) : void
+isCellEditable(rowIndex : int, columnIndex : int) : boolean
+getColumnClass(columnIndex : int) : Class<?>

TaskTopPanel

<<Property>> -startDate : JLabel
<<Property>> -endDate : JLabel
<<Property>> -status : JLabel

+TaskTopPanel()
-initComponents() : void

TaskFrame

-dialog : JDialog
-taskElementsTable : JTable
-completeButton : JButton
-closeButton : JButton
<<Property>> -task : Task
-taskTopPanel : TaskTopPanel

+TaskFrame()
+showDialog(parent : Component, task : Ta...
-initFrame() : void
-initComponents() : void
-initButtonPanel() : JPanel

MainFrame

+DEFAULT_WIDTH : int = 800
+DEFAULT_HEIGHT : int = 600
-table : JTable
-taskFrame : TaskFrame
<<Property>> -teamMember : TeamMember
-taskTableModel : TaskTableModel
-sidebarPanel : SidebarPanel
<<Property>> -mainFrame : MainFrame

+MainFrame(teamMember : TeamMember)
+updateMainFrame(teamMember : TeamMemb...
-initComponents() : void
-initFrame() : void

TaskTableModel

-tasks : Task

+TaskTableModel(teamMember : TeamMember)
+setTasksList(tasks : List<Task>) : void
+getRowCount() : int
+getColumnCount() : int
+getColumnName(columnIndex : int) : String
+getValueAt(rowIndex : int, columnIndex : int) : Object
+getColumnClass(columnIndex : int) : Class<?>
+getTaskAt(row : int) : Task

ProgressBar

-dialog : JDialog

+showGui(status : String) : v...
+hideGui() : void

TaskerCLI

+main(args : String [] ...

Powered By Visual Paradigm Community Edition

5.1.1 Description of each significant Class

- **TeamMember** class abstracts the TeamMember table in both databases. Its instance variables correspond to columns inside databases and can be accessed and modified using the getters and setters provided. Instance of this class will be used to populate the user interface with data in the program.

- **Task** class abstracts the Task table in both databases. Its instance variables correspond to columns inside databases and can be accessed and modified using the getters and setters pro-

vided. Each task has an associated list of task elements. The status of a task is stored as an integer in the database and is then transformed to an appropriate **TaskStatus** Enum inside the program.

- **TaskElement** class abstracts the TaskElement table in both databases. Its instance variables correspond to columns inside databases and can be accessed and modified using the getters and setters provided.

- **TeamMemberDB** class is one of the data access classes in the DAO pattern. It provides useful methods which can be used to retrieve and update the **TeamMember** objects in both databases. For example the `selectTeamMemberByEmail` method will as expected acquire the connection to either the MySQL or SQLite database, prepare a suitable query statement, construct the **TeamMember** object and return it back to the caller. This is a really powerful concept which abstracts the raw SQL from the graphical user interface. After we make some changes to the object, we can use a single method `updateTeamMember` to put it back to the database.

- **TaskDB** class is very similliar to the previously described **TeamMemberDB** class. It also is a part of the DAO pattern and provides direct access to the information stored in the task table in both databases. It will be typically only used by the previosuly mentioned **TeamMemberDB** to construct the complete **TeamMember** object.

- **TaskElementDB** class is basically the same as the previous one, but it enables the access to the TaskElement table in both databases.

- **ConnectionManager** class can be used to actually connect to both databases. It enables the caller to acquire a `Connection` object to either the remote MySQL (TaskerSRV), or locally run SQLite. **ConnectionManager** is capable of registering appropriate JDBC drivers depending on the connection required.

- **Syncer** class contains the logic for conflict resolution between the contents of the two databases within its `sync` method. The sync method takes two **TeamMember** objects, generates a canonical TeamMember object, which is supplied to the databases and UI. The Syncer class also contains the `login` method, displaying a dialog for user login and returning an appropriate **TeamMember** object. class also contains a **Timer** object used to invoke syncing at regular intervals.

- **TaskerCLI** class is the main class that is run when the application begins. It calls the **Syncer** classes `login` method and passes the returned **TeamMember** onto the UI Classes.

- **MainFrame** extends **JFrame** class is the primary view after the login menu, and displays a **TeamMember** object's Tasks in a sortable JTable, clicking on this invokes **TaskFrame**. Also displays **SidebarPanel**.

- **SidebarPanel** extends **JFrame** class is used for controls within **SidebarPanel**, such as logging out and searching tasks.

- **TaskFrame** extends **JDialog** class is used to display details of a task when invoked by **MainFrame**. Allows editing of **TaskElement** and also marking of a Task as complete.

5.2   TaskerMAN Significant Classes
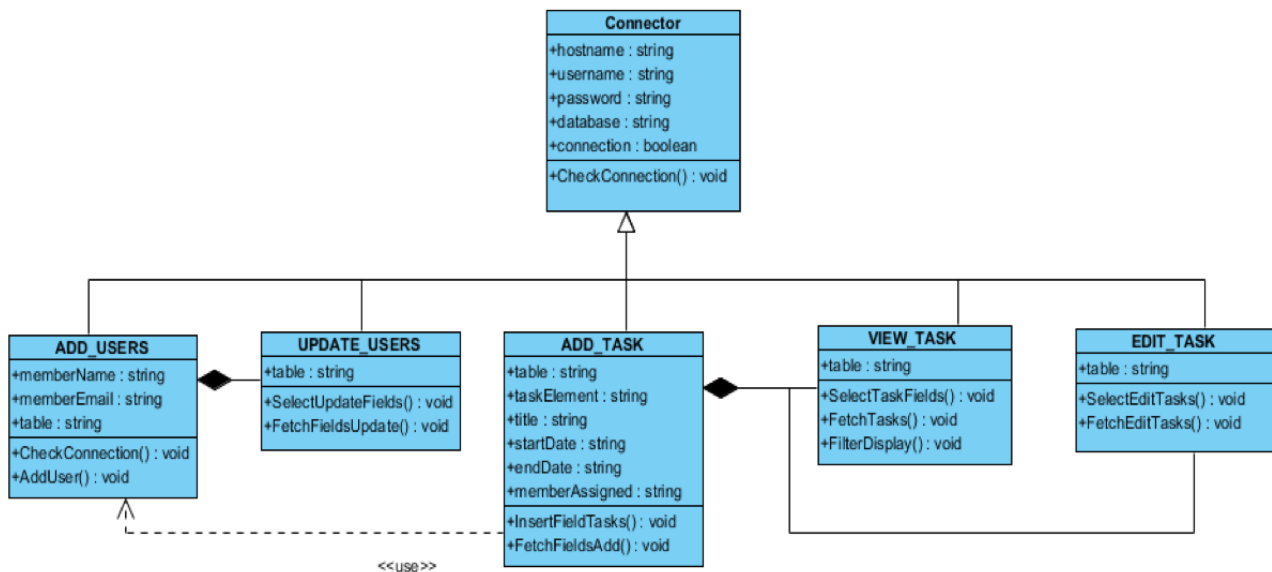
The PHP class diagram above depicts just but a few major collaborating classes in the TaskerMan. All the classes inherit variables and a single method from the Connector.php. These variables and the CheckConnection method have to have a constant connection to the database. UPDATE_USERS class is shown as a composition to the ADD_USERS. This means that it depends solely on the availability of the ADD_USERS class. If the ADD_USERS class is removed then UPDATE_USERS class will also not exist. The ADD_TASK class is show to make use of the ADD_USERS class. This is where we get our users from. The VIEW_TASK and EDIT_TASK classes are shown as a composition to ADD_TASK.

5.2.1   Description of each significant Class

- **Adding a Task**

  One file used for processing data is the addtask.php file. This file firstly allows the user to enter in some information such as the start date of the task, completion date and member assigned. Then the information that is typed in, is processed using another PHP file which inserts what the user has typed in to the database, the file firstly connects to the database, checks the connection and then gets all the data the user typed in and adds it to the database. An if statement is then included to check that the data has been entered into the database correctly, if it has, the web page will display a message to the user confirming that a new task has been added to the database, then the user is presented with 3 options to add another task, log out or go back to the home page. If the data is not added to the database correctly due to a problem an error message will be printed out to the user and they won't be able to go on any further in the process.

- **Edit Tasks**

  The edittasks.php file presents the user with an area to modify the tasks they have set for example, changing who they have allocated the task to. This connects to the database and prints all of the relevant information in a table - all of the tasks are displayed at this point. From here the user can either update the task or delete the task. If they click on delete, a file called connectedb.php will run and depending on the id that it has (each record in the table has an id for reference purposes) it will then delete all the information from the task list via an SQL

command and then closes the database connection. If the user selects the update option from the table they will be taken to a file called updatetask.php this once again is referenced using an ID and once the user clicks this button they are presented with all the information of the task, they can then update the description of the task, change the dates and member allocated. To get this information, the user is connected to the database and a SELECT SQL query is used. Some text boxes are left open and if the information changes then the field is sent to the database that changes. Once you have updated the records, the user will be able to check their changes have been made by viewing all the tasks. If the update has been successful you will be presented with a confirmation message, if you have not, an error message will be displayed. If you have been successful you will have the option to update another record or go back to the home page.

- **View Tasks**

  If you select this option from the home page you will be taken to a file called viewtasks.php. This web page pulls some information from the database using an SQL query to show all of the tasks and then it puts this information into a table to make it easier to read. This webpage also includes a filter that allows the user to view the data based on the member or the status of the task. Once the user has looked at this information they can either log out or navigate back to the home page.

- **Add User**

  If you select the add user option from the home page - you will be presented with two text boxes described earlier in this document. Once you click the submit button you will be taken to a webpage called addusers.php this just adds the information that the user has typed into the database via an SQL command. This once again has an if statement which if unsuccessful will provide the user with a simple error message, if successful in adding the user a confirmation will be shown and then the user will have a few navigation choices.

- **Edit User**

  The editusers.php file presents the user with an area to modify their details if they have typed them in wrong or their details change. Currently the only fields we have here are name and email address. If the user wants to update these they just click on the update link and are then taken to an updateusers.php page where they can then re-enter their name and email address. Once they have done that and are happy with it they click the update button which sends the new data they have put in to the database. They will then be provided with a confirmation message if it is successfully updated, and then you can navigate to other areas of the website. To make sure that the user types in information in the correct format we will be implementing some validation to check that the user is entering data of the correct type. Next you can go back to the editusers.php file and also delete records of users. This will be used for managers so that they can delete users from TaskerMAN if they need to. All that they need to do is look at the table of users that is provided in editusers.php and then click on the delete link of the user that they want to get rid of. Once they do that a confirmation message will be displayed telling them that the user has been successfully deleted. Once this action is completed by the user a file called connectdb.php runs which connects to the database and then a delete SQL command is used to get rid of the user they deleted - each user has a reference number, so this is what is used to delete a specific user. Next time they view the edit user's web page the table will have

been updated to show all the users they want. To make sure users aren't accidentally deleted we should implement a pop up box to appear and make the user confirm that they want to delete a specific user.

- **View Users**

  Once logged in, view users will enable you to look at all the people using the system - this tool will be used for managers only so they can view all the team members they can allocate tasks too. Once they have clicked on the view user's page, a table will be displayed - this will be populated with information about the users in the system (which will just be their name and email) and what tasks they have been allocated to do. To get this information, an SQL query will be used to select all the users and then this information is just output into a table.
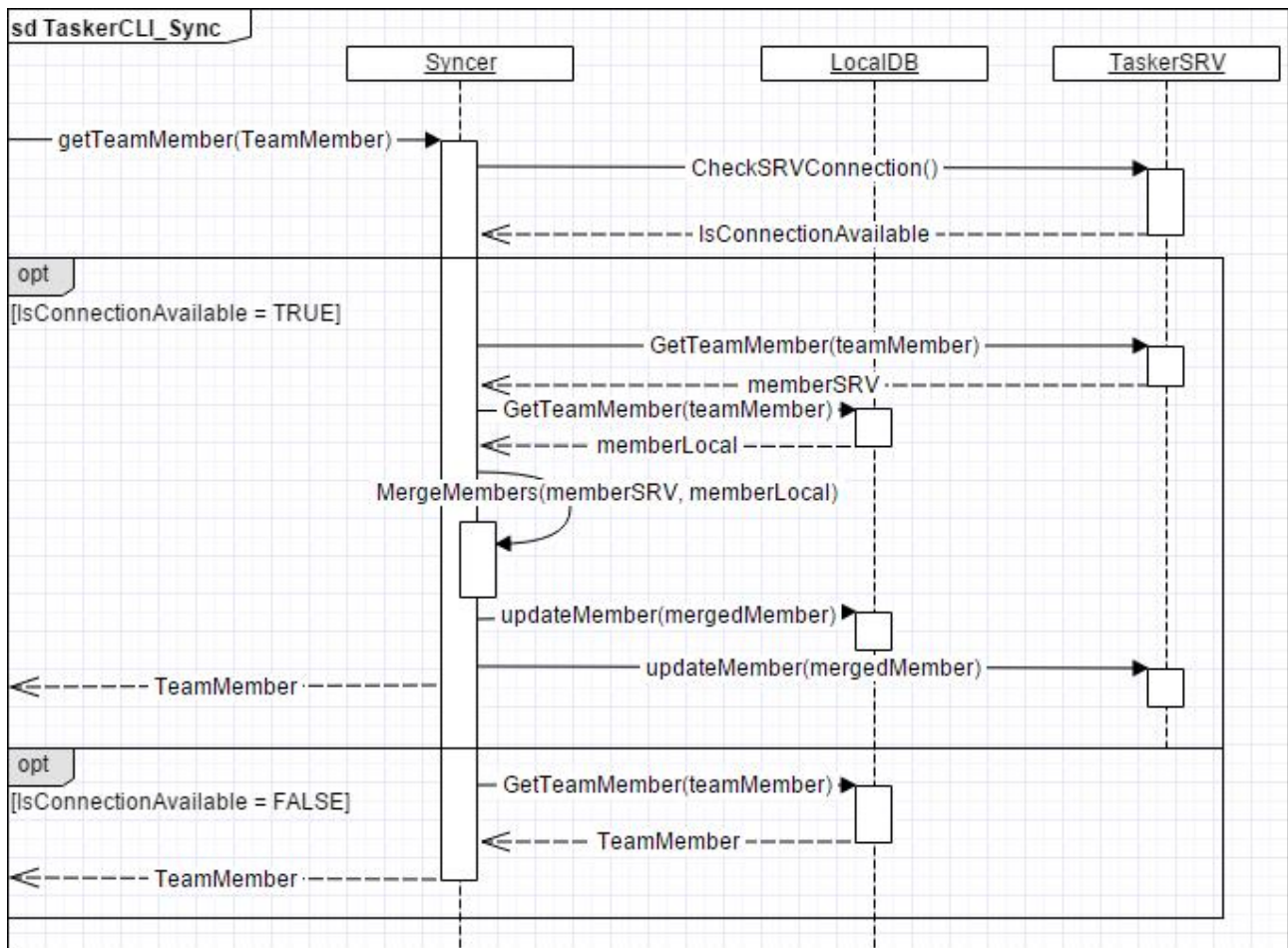
# 6 Detailed Design

6.1 TaskerSRV Entity-Relationship Diagram
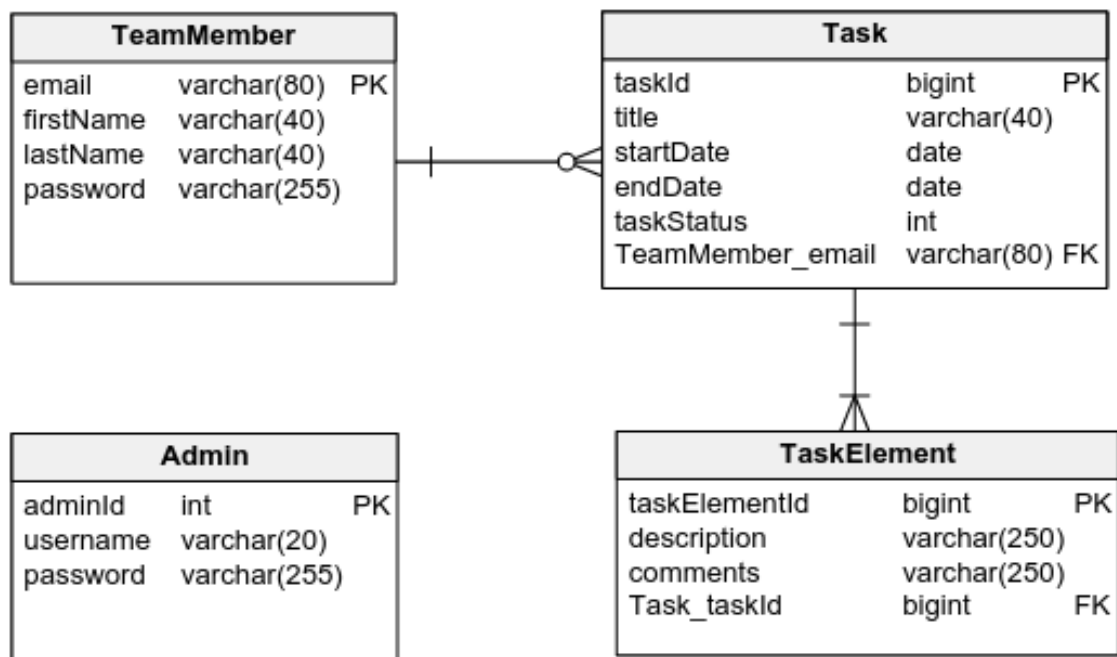
6.2 TaskerCLI Sequence Diagrams

The following are the sequence diagrams describing the major actions taken by TaskerCLI, namly logging in to the system, updating a task, logging out, and the synchroniser syncing the changes made to the local and remote databases.

Above shows the sequence diagram for the action of syncing the two databases. It works by first checking if TaskerSRV can be connected to, then branches depending on the connections availability. If the connection is available, the syncer class gets a copy of the team member from both the local database, and TaskerSRV. It performs a sort of merge on the two copies, updating information depending on the priority of changes made, clarified in FR9 of the specification document. It copies this newly merged teamMember object back to both databases, then returns it.
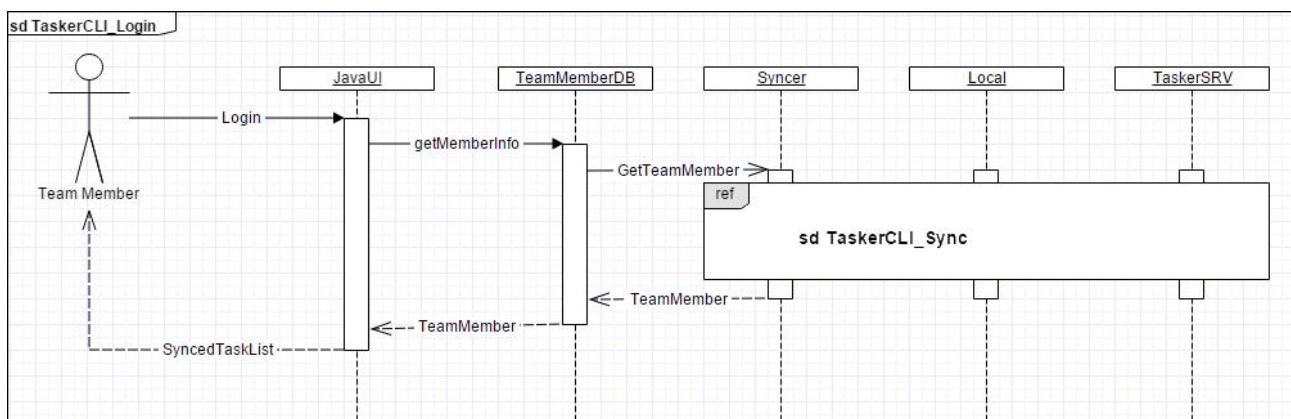

The above sequence diagram describes the action of a user logging in. The user provides their login details, which the UI sends to the TeamMemberDB static class. This class sends a request to the Syncer to receive the merged teamMember from both databases. The syncer runs the actions described in sd TaskerCLI_Sync and returns the teamMember to TeamMemberDB, which passes it on to the UI. The UI pulls the task list from the teamMember, and displays it in the UI for the user.
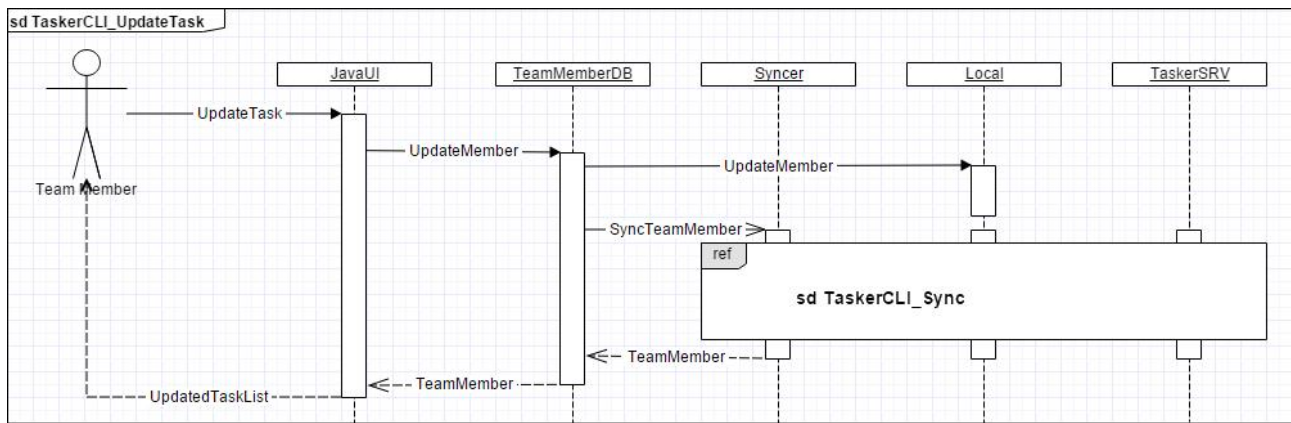
Above describes what happens when a user updates a task, either by completing it or updating the list of subTasks assigned to it. The UI sends an update to TeamMemberDB, which first updates the local database with the newly updated task. It then calls on the Syncer to sync the databases again, using sd TaskerCLI_Sync. Because of the priority, the changes made locally will override TaskerSRV if it is available. The syncer then returns the synced teamMember to TeamMemberDB, which passes it on to the UI. The UI then, like when logging in, pulls the task list from the teamMember and displays it to the user.
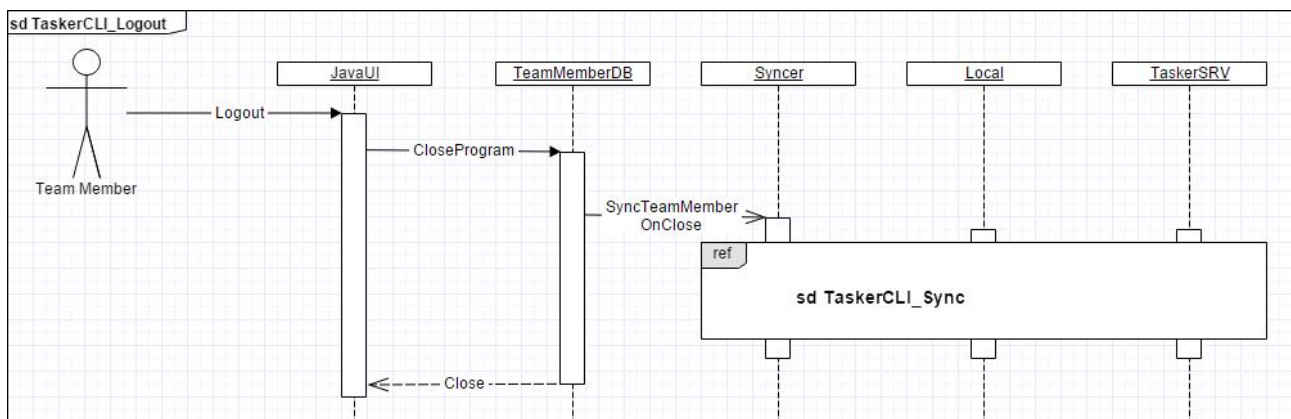

Above describes how TaskerCLI goes about closing when the user logs out. When the user logs out, the UI sends a close command to TeamMemberDB, telling it to perform a sync with the databases in order to save any last changes before shutting TaskerCLI. It requests a sync from the Syncer, which synchronises both databases. TeamMemberDB does not request the teamMember be returned as it is not needed. Instead, once the Sync process is completed, TeamMemberDb sends a message to the UI, telling it that it is ready to shut down. The UI then closes.

6.3   Spike Programming

To be able to explore potential solutions to the implementation for Tasker we have created a spike solution by creating a prototype model for both TaskerCLI and TaskerMAN. The spike prototype for TaskerMAN was used in order to experiment running MySQL queries to the database using PHP and also to experiment keeping sessions for the user of the system. The spike prototype for CLI was used to decrease the threat of it being too technically difficult when it came to the actual implementation of TaskerCLI for example, a quickly made spike solution will help us understand how we will structure our classes and relationships between the classes. Our quick spike solutions did not fully support the functional requirements but helped us understand how we can achieve these by giving us an insight on how to structure the more complex areas of the design.

## 6.4   TaskerMAN Sequence Diagram
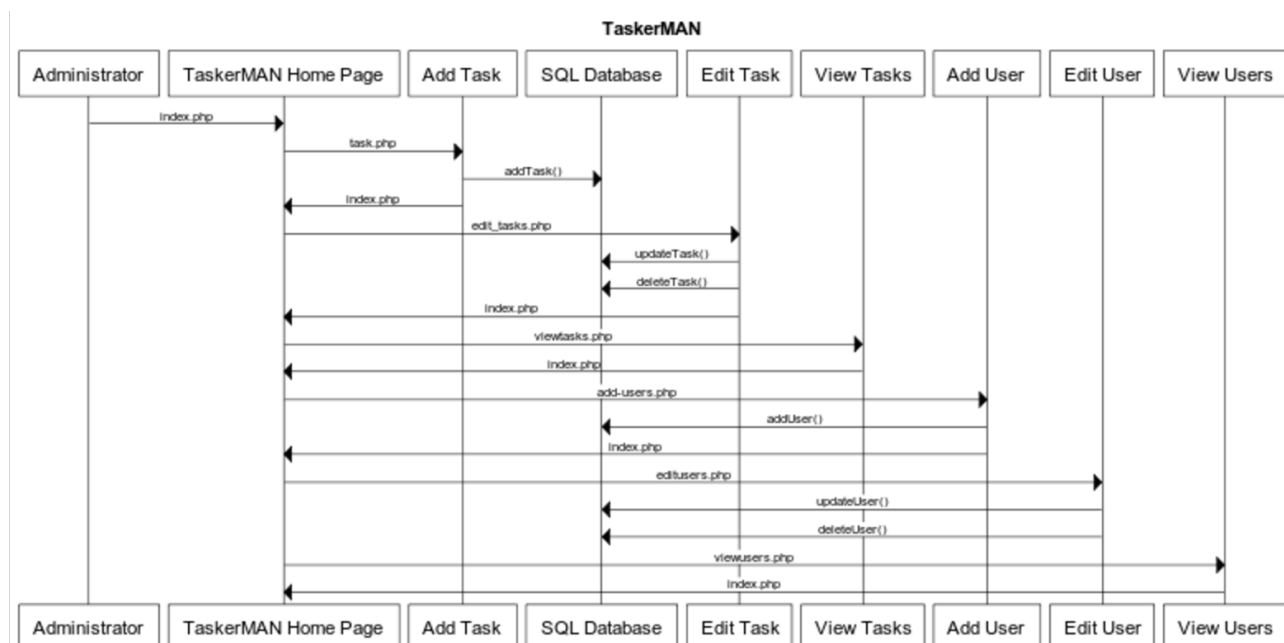
6.5   TaskerMAN Object Diagrams

6.5.1   TaskerMAN Object Relationship between the ADD_USERS and EDIT_USERS CLASS

Below is an example of the object instances and association in the TaskerMAN at a particular point in time. In this case an instance is created with the user and email of a new member. The two other instances updateMary and DeleteMary are both possible as a result of the Mary instance in the Add_Users class. An example is the diagram showing a wrong spelling name entered in the Mary Instance. Since the Add_Users class has an association with the Edit_Users class we can instantiate the UpdateMary instance and correct the name to Mary Joseph. Similarly we can instantiate the DeleteMary instance and delete the member with the wrong name if we wish to.
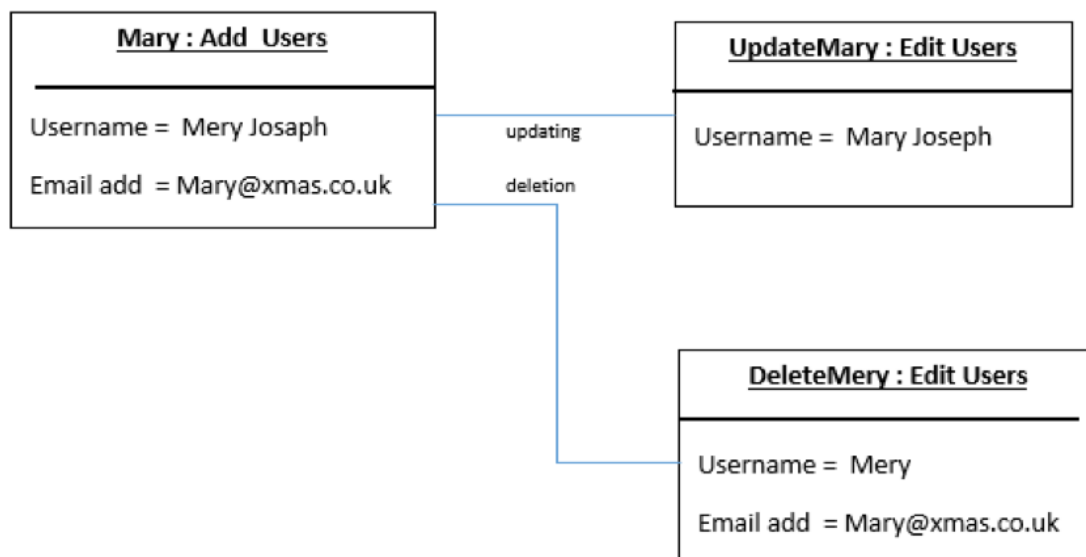
6.5.2   TaskerMAN Object Relationship between the ADD_TASK and EDIT_TASKS CLASS

The logic of the object interaction below is essential the same as in above. Here the Mary instance is established and its values are clearly depicted. All the values in MaryBirth object are correct except the EndDate. We therefore instantiate the MaryEdit object and make the necessary changes as shown. We can instantiate another object MaryFilter and view the task.
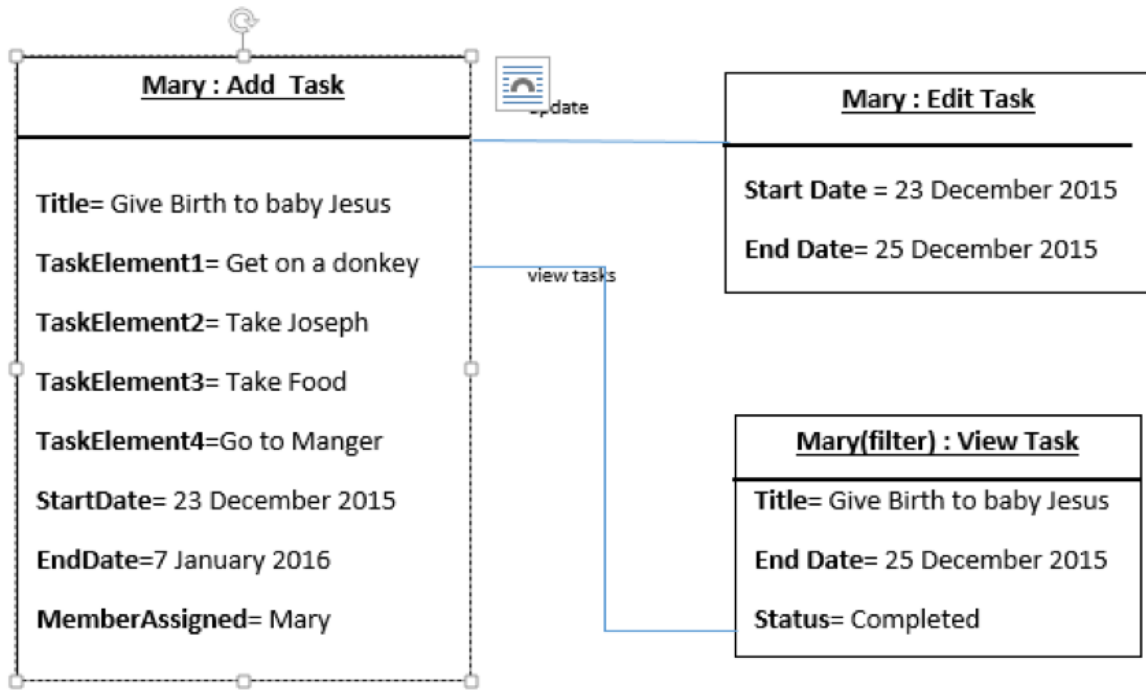
REFERENCES

[1] *Software Engineering Group Projects* Design Specification Standards. C. J. Price, N.W.Hardy and B.P.Tiddeman, SE.QA.05A. 1.8 Release.

[2] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS. 1.1 Release.

[3] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS, FR8 *Local Storage of Tasks.* 1.1 Release.se.qa.rs

[4] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS, FR5. 1.1 Release.

[5] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS, FR6. 1.1 Release.

[6] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS, FR8 *User Identification.* 1.1 Release.

[7] *Software Engineering Group Projects* Requirements Specifications.se.qa.rs N. W. Hardy, SE.QA.RS, FR11. 1.1 Release.

[8] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS, FR10. 1.1 Release.

[9] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS, IR1. 1.1 Release.

DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2015-10-27	Initial creation for review	L. Jones
1.1	N/A	2015-10-28	Updated the TaskerCLI user interface section	M. Goly
1.2	N/A	2015-10-28	Created sub-subsections for Use-cases and User-interface design	L. Jones
1.3	N/A	2015-10-29	Added TaskerCLI Use-Case diagram and narrative	L. Jones
1.4	N/A	2015-10-29	Updated version from review to release	L. Jones
1.5	N/A	2015-11-24	Added Sections for Design Specification Review	L. Jones
1.6	N/A	2015-11-27	Compiled team's Component Description, Significant Classes and Detailed Design sections for release	L. Jones
1.7	N/A	2015-11-27	Added TaskerSRV diagram and updated the TaskerCLI sequence diagram narrative for release	M. Goly