

# Software Engineering Group Project

## Final Report

*Author:* L. Jones, M. Goly, T. Mills  
*Config. Ref.:* SE-12-FR  
*Date:* 2016-02-15  
*Version:* 1.2  
*Status:* Release

Department of Computer Science,  
Aberystwyth University,  
Aberystwyth,  
Ceredigion, SY23 3DB,  
U.K.

©Aberystwyth University 2016

## CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>2</b>
1.1	Purpose of this document . . . . .	2
1.2	Scope . . . . .	2
1.3	Objectives . . . . .	2
<b>2</b>	<b>END OF PROJECT REPORT</b>	<b>3</b>
2.1	Management summary . . . . .	3
2.2	Historical account of the project . . . . .	3
2.3	Final state of the project . . . . .	5
2.4	Location of the group repository . . . . .	5
2.5	Performance of each team member . . . . .	5
2.5.1	Adam Neaves . . . . .	5
2.5.2	Josh Mir . . . . .	6
2.5.3	Luke Jones . . . . .	6
2.5.4	Michal Goly . . . . .	6
2.5.5	Will Jones . . . . .	7
2.5.6	Tino Garapasi . . . . .	7
2.5.7	Tom Mills . . . . .	7
2.5.8	Tom Oram . . . . .	8
2.6	Critical evaluation of the team and the project . . . . .	8
<b>3</b>	<b>PROJECT TEST REPORT</b>	<b>10</b>
<b>4</b>	<b>PROJECT MAINTENANCE MANUAL</b>	<b>12</b>
4.1	Program description . . . . .	12
4.2	Program structure . . . . .	12
4.3	Algorithms . . . . .	12
4.4	The main data areas . . . . .	12
4.5	Files . . . . .	12
4.6	Suggestions for improvements . . . . .	12
4.7	Things to watch for when making changes . . . . .	13
4.8	Physical limitations of the program . . . . .	14
4.9	Rebuilding and Testing . . . . .	14
	<b>REFERENCES</b>	<b>15</b>
	<b>DOCUMENT HISTORY</b>	<b>15</b>

## **1 INTRODUCTION**

### **1.1 Purpose of this document**

The purpose of this document is to outline that which we have achieved and gained as a group throughout the software development process, and to reflect on the process as a whole.

### **1.2 Scope**

The document covers the process as a whole, including the state of the software produced, the standards set in our documentation as well as other non-technical aspects such as how we collectively worked as a team and how we could improve. Attached as appendices are the Project Plan, Test Specification and Design Specification.

This document assumes the reader is familiar with the requirements specification[1] and all of the quality assurance standards associated with the project.

### **1.3 Objectives**

The objectives of this document are as follows:

- To provide a concise summary on the state of the software including areas in which it could be improved.
- Discuss the documentation produced and how it was necessary for it to be altered during the process.
- Provide a general summary of the project on the whole from a historical point of view i.e. how?/when?/why? were things done.
- Outline the roles and duties performed by every individual.
- Provide a test log to illustrate where the software passed/failed our tests[2].
- Produce a maintenance manual for the benefit of program maintainers.

## **2 END OF PROJECT REPORT**

### **2.1 Management summary**

In terms of the software produced we believe we have completed what was required[1] to a good standard. Our software adheres to all the necessary functions and requirements as proved during the acceptance testing, with the exception that TaskerMAN does not currently support filtering. This was not due to team members lacking the capability to implement the function, rather it was a misunderstanding when reading the requirements specification, it was only during acceptance testing that we realised we'd made the error. There are areas where the code could be cleaner e.g. the 'Syncer' class within TaskerCLI contains overly complex methods, which make the code more difficult to understand and potentially problematic for a future developer to maintain, though it is all functional. The front-end of TaskerCLI is also not the most eye-catching as we weren't able to design it completely as we wanted due to time constraints, however it's also completely functional.

From a documentation point of view we are also in good stead. On the whole we received positive remarks for both our Test Specification and Design Specification, receiving grades B- and A respectively. Based off the feedback received from both our project manager and project coordinator we made further improvements to the documentation where necessary. The Project Plan was also maintained and updated throughout the entirety of the project by the Group Leader.

The main obstacle of the project were the numerous bugs encountered with TaskerCLI during Integration and Testing Week. In particular, we had an issue where comments made related to the task elements of a task would not save when the task panel was closed, which was a major part of the functionality of the software. The issue stemmed from our naive implementation of merging the remote objects and the local objects. The way that our TeamMemberDB.updateTeamMember method was originally written meant that it would take the remote object, look at the list of tasks it has associated with it, then iterate through the list of tasks. We would then call the sql query UPDATE on tasks to update them, but this didn't work on the local copy of tasks. To fix this we had to change the update method to remove the team member, then re-insert into back into both of the databases, rather than just trying to update it. This solved the issue. There were other issues associated with TaskerCLI such as task reassignment not working, however these issues were much less time-consuming and less frustrating to solve than our main issue. There were also other problems that occurred which were out of our hands e.g. during Integration and Testing Week, we were unable to work for several hours due to a power failure on site, and one morning the remote database was not able to be written to, however such incidents were accounted for in the Risk Analysis of the Project Plan[4].

As a team we worked well together. From a documentation point of view everybody had an input. From a software development point of view there was a bit of an imbalance in terms of work load. Members of the TaskerCLI subteam naturally had more work due to the way in which we designed the system[3] and the more technically proficient members of the group found themselves with the most work allocated to them. Though TaskerMAN was a relatively easy system to design in comparison, there was still the same issue of the most technically proficient member being allocated the most work which was unfair, and this was an error on my behalf (Luke Jones - Project Leader). However as a group the morale was generally good, there were no major personality clashes that hindered the project in any way and we worked effectively to achieve what was required of us.

### **2.2 Historical account of the project**

The initial group meetings were concerned with familiarising each other as a team and the requirements specification. Shortly after this roles were allocated based on team member's strengths/weaknesses.

The first major deliverable was the outline for the 'Interaction and High-Level Design' of the system as part of the Design Specification document. Simultaneously it was required that the project Leader create a Project Plan with an accompanying Gantt Chart and Risk Analysis. After some initial

confusion due to it being the first deliverable and team members being not used to the format of delivery, the 'Interaction and High-Level Design' was completed, reviewed and submitted. As part of the document, we were required to create diagrams and a narrative for the application deployment i.e. applications interface and applications in the system, as well as use-cases for both TaskerMAN and TaskerCLI. It was also necessary for us to design the UI for TaskerMAN and TaskerCLI at this point, and the designs that we created in this period are very similar to the UI which exists on the finished product.

After this deliverable the head developer of TaskerMAN (Tino Garapasi) elected to begin work on prototypes of the website and presented them to the group. We were very impressed with what he'd made and the fact that he worked hard on the system in the beginning meant that our overall workload was significantly reduced later on in the project.

The next major delivery was the 'Test Specification'. The creation of tests and a test table was allocated largely to one individual (Tom Mills). We failed to have a review meeting for this document and consequently received a lower grade than our previous deliverable. Meanwhile members of the TaskerCLI subteam began work on a TaskerCLI prototype. The major implementation in this prototype was the 'Syncer' class related to the synchronisation aspect, which was a difficult class to code, along with all the other necessary functionalities required for the prototype demonstration to our project manager.

The next deliverable required was the complete 'Design Specification', which was a continuation of the 'Interaction and High-Level Design' deliverable in more detail, including sections such as 'Significant Classes' and 'Detailed Design', both of which required several sub-sections and numerous diagrams with pieces of narrative. This document was completed, reviewed and submitted and received positive feedback.

The final delivery of the first term was the actual prototype demonstration to our project manager of a working system. TaskerMAN was already nearly fully-functional at this point, and the TaskerCLI subteam worked hard during the final week to prepare it for demonstration. On the final day we solved several critical bug issues related to the prototype and had several practice demonstration before finally showing our project manager our progress. He was very pleased with the system and praised us on our progress. Up to this point the process strictly followed a plan as there were no major setbacks.

The next time we convened for group project duty was Integration and Testing Week, which was a long time after the prototype demonstration due to Christmas and an exam period. We began the week by creating a list of all known functionality issues on TaskerMAN and TaskerCLI and creating GitHub issues for each individual issue. From a TaskerMAN point of view the major tasks included implementing the optional feature of authentication and cleaning up the website on the front-end. From a TaskerCLI point of view the goal was on fixing any pre-existing bugs and to implement any functional requirements which had not yet been implemented.

The TaskerMAN development ran relatively smoothly and according to plan. We were able to implement everything that we wanted, the front-end was cleaned up and we were able to test the website on a non-developer (Tom Mills) to test it's user-friendliness, and all tests were passed. All tests related to TaskerMAN as outlined in the 'Test Specification'[2] in terms of functional requirements also passed. On the final day we completed the installer and it's associated README.txt and we were very satisfied with the final system.

TaskerCLI development was not as smooth. The target was to spend 3 days fixing critical bugs and implementing a cleaner UI, before moving onto testing and the installer. What actually happened was that the vast majority of the week was spent fixing critical back-end e.g. as comments made to task elements not saving. Because of this there wasn't any opportunity to clean up the UI, although it's still relatively user friendly as it passed the UI tests outlined in the 'Test Specification'. The final system also passed all of the functional requirements associated with it in the 'Test Specification'. On the final day there were difficulties implementing the installer due to the way our system was

designed[3] however it was ultimately completed and it's associated README.txt written. Although such problems as the ones encountered when developing were accounted for in the Risk Analysis TaskerCLI's development was often frustrating due to the fact we didn't achieve 100% of what we wanted.

Acceptance testing began a few days later. Our software passed all of the tests, however as explained earlier failed the filtering requirements of TaskerMAN due to team members misunderstanding the requirements specification.

It was now time for team members to draft their personal reflective report and work on the final deliverable 'Final Report', including the 'End-of-Project Report', 'Test Report' and 'Maintenance Manual'. These were completed and complied shortly before the deadline. At this point we also confirmed that everything was in good order on our repository as outlined in the 'Operating Procedures and Configuration Management Standards' quality assurance document, and confirmed 'nwh-aber' was a collaborator on the repository.

Throughout the entirety of the project team members maintained a weekly blog outlining their progress and contribution.

## **2.3 Final state of the project**

All of the necessary functional requirements of Tasker have been implemented with the exception of filtering in TaskerMAN. TaskerMAN currently supports the sorting of tasks when viewing tasks, either by expected completion date (default), member assigned to or task status, but not full filtering as required [1]. Also in the README.txt file of the TaskerMAN there is no mention of the need to change permissions (if necessary). Finally the TaskerCLI installation script should also be moved from the src folder to TaskerCLI.Installation. All of the documentation is in good standing, has been updated based on the feedback received during the process and is correctly documented with regards to referencing and document change history.

## **2.4 Location of the group repository**

The URL of our repository is <https://github.com/MichalGoly/Tasker>. Within our *src* folder TaskerCLI is a submodule with the URL <https://github.com/MichalGoly/TaskerCLI>. The final branch for delivery is the default *master* branch. The user 'nwh-aber' is both a collaborator on the main Tasker repository and the TaskerCLI submodule.

## **2.5 Performance of each team member**

### **2.5.1 Adam Neaves**

Adam Neaves was a member of the TaskerCLI subteam, specifically the front-end development. Adam was tasked with designing the front-end of the client which he did as seen in the Design Specification[3]. The current working TaskerCLI is pretty similar to the design as seen in the Design Specification, though it's not quite as clean or eye-catching and lacks the colour. This was due to unforeseen errors on the back-end of TaskerCLI meaning that we had to reallocate resources to solving critical issues, though the front-end design is still very user-friendly. During Integration and Testing Week he diverted his focus to fixing errors that existed on the TaskerCLI prototype, as well as single-handedly writing the Javadoc comments for almost all of the TaskerCLI code. From a documentation point of view he played a vital role on the detailed design section of TaskerCLI in the Design Specification, in particular creating a number of extremely informative sequence diagrams with explanation. He was also responsible for the production of the TaskerCLI use-case diagram and narrative.

Adam had a good working relationship with all members of the group and completed all the tasks allocated to him. During group meetings he would always contribute with valuable input and was happy to explain anything to the other group members if they were struggling to understand as he was always aware of what needed to be achieved. In his own words Adam feels he could've played a more instrumental role during Integration and Testing Week, however due to back-end issues with TaskerCLI this wasn't possible and was certainly through no fault of his own.

### **2.5.2 Josh Mir**

Josh Mir was the Deputy Project Leader. He was also a key developer on the TaskerCLI application as he had prior experience working with Java development. From a deputy project leader point of view he wasn't required to do anything as the project leader didn't assign him any tasks associated with the management of the project, and he wasn't required at any point to lead the project in the project leader's absence. Josh worked alongside Michal on the back-end development of TaskerCLI, implementing the most challenging aspects of the design[3] such as synchronisation between the local and remote databases. He was key in solving a number of critical bugs we encountered during Integration and Testing Week for TaskerCLI, as well as helping the TaskerMAN subteam with any issues they were having. He also assisted on the TaskerMAN side of the project during design, suggesting that we use a Bootstrap template for a slick and simple design, which we did. He was the most knowledgeable with operating Git, and was responsible for setting up TaskerCLI as a submodule within the *src* folder of the Tasker repository. He also assisted with the TaskerCLI installer and its associated README.txt.

Josh had a good working relationship with all members of the group, was highly approachable with any issues team members were having and had a high level of technical proficiency related to all aspects of the project, not just Java and TaskerCLI.

### **2.5.3 Luke Jones**

Luke Jones was the Project Leader. He moderated group meetings, booked rooms for group meetings and was the main means of communication between the group and project manager. From a technical point of view he was a member of the TaskerMAN subteam as he had prior experience with HTML/PHP. During Integration and Testing Week he worked mainly on the front-end of TaskerMAN, correcting issues that made the site less eye-catching, as well as spotting a number of bugs associated with the front-end of the site e.g. the date picker wasn't originally responsive in Firefox or Safari browsers. He also had a minor input on the back-end of the website, assisting the main developer (Tino) with any errors which occurred. He also assisted with the installer for TaskerMAN and helped write the README.txt for installation. He played a larger role from a documentation point of view, working largely on the TaskerMAN elements of the Design Specification, as well as being the person responsible for compiling everyone's sections together for delivery. He also maintained a Project Plan throughout the entirety of the process.

Luke was largely organised in what he did, and maintained group harmony to the best of his ability. He never missed a meeting and always had an agenda prepared beforehand. He was very approachable and had a good relationship with the other members of the group. An area in which he could improve is that he could've more equally allocated work during Integration and Testing Week as it was quite imbalanced which led to frustrating and unfair situations.

### **2.5.4 Michal Goly**

Michal Goly was the head developer of the TaskerCLI application as his strengths lied in Java development. He worked incredibly hard on the back-end of the client, both during and prior to Integration and Testing Week, as due to the way our system was designed[3] it was a complex and

error-prone Java application. During group meetings Michal would play a very active role in discussion and being the overall most technically proficient member of the team, always contributed valuable input. During Integration and Testing Week a large part of his time was devoted to solving the numerous issues that existed on the TaskerCLI prototype, as most of the features had already been implemented, as well as working on the installer for TaskerCLI and its associated README.txt. From a documentation point of view he also played a valuable role, in particular in the Design Specification on the deployment description and TaskerCLI detailed design sections, as well as contributing to the maintenance manual in the Final Report.

Michal worked tirelessly for the group and was extremely determined. He was always aware of his duties and would often help others in their own tasks. He arguably had the biggest contribution to the project alongside Josh and Tino. Michal had a good relationship with all members of the group.

### **2.5.5 Will Jones**

Will Jones was a member of the TaskerMAN subteam. Will's roles and duties were identical to mine (Project Leader) without the management work. Early in the process he assisted Tino with the narrative for TaskerMAN UI in the Design Specification[3], and also put a lot of work into the detailed design of TaskerMAN e.g. wrote the entire significant classes section for TaskerMAN. During Integration and Testing Week Will worked on the already functional prototype and made mostly front-end changes, e.g. CSS alterations to make the site more user-friendly and savvy safety features such as changing password fields to print out asterisks instead of plain text. He worked alongside Tino on the authentication aspect helping to create the login/logout functions. He was also tasked with formatting and commenting on our TaskerMAN code.

Will had a good working relationship with all members of the team and was very approachable when asked to complete a task. He seldom missed a meeting and if unsure of something technical then he would work hard to make sure he understood it eventually.

### **2.5.6 Tino Garapasi**

Tino Garapasi was the head developer of TaskerMAN. He was also originally the deputy quality assurance officer however during the process that responsibility was handed to Tom Mills as Tino no longer desired the role. Tino played a massive role in the early stage of TaskerMAN development, creating the majority of the prototype single-handedly. During Integration and Testing Week Tino's work consisted of fixing any back-end issues that existed with the site e.g. protecting the site from SQL injection. He was also the primary person involved with creating the TaskerMAN installer and the associated README.txt. He played a very active role in the documentation, in the Design Specification[3] he created TaskerMAN use-case diagrams and TaskerMAN object diagrams, as well as the TaskerMAN component description.

Tino had a good working relationship with all members of the team and was very easy to work alongside. The hard work he did early on in the process significantly aided us in Integration and Testing Week from a TaskerMAN point of view. He always completed what was asked of him and would often work relentlessly for the benefit of the group, and wouldn't hesitate to help a member of the team who required assistance.

### **2.5.7 Tom Mills**

Tom Mills was the Deputy Quality Assurance. He wasn't originally however after Tino no longer desired the role he volunteered himself. From a quality assurance point of view the only responsibility he had was to take and upload the minutes when the quality assurance officer was absent. Tom's main role in the team was tester. He contributed massively in this aspect, creating most of the Test



Specification by himself maintaining it based on the feedback received. During Integration and Testing Week he was responsible for the testing of the TaskerCLI application, creating unit tests throughout the week, as well as helping the TaskerCLI subteam with issues in any way he could due to Java development being one of his strengths. He was also responsible for the Project Test Report which appears in this report. He worked on the Design Specification[3] writing a narrative with regards to spike programming.

Tom had a good working relationship with all members of the group. He participated in every meeting, offered input where he could and completed everything that was requested of him. It probably wasn't a good idea for such a large aspect of the process (testing) to be allocated to one individual however this was an error on my part (Project Leader).

### **2.5.8 Tom Oram**

Tom Oram was the Quality Assurance Officer. Tom was responsible for taking the minutes of every meeting, and uploading them to GitHub, which after some initial trouble with Git, he did consistently and on time. During Integration and Testing Week he was a part of the testing team, in particular as he was a non-developer of the TaskerCLI application he was able to test the user-friendliness of the application to the average user. He checked the code created by the TaskerCLI subteam to ensure it complied with the Java coding standards as outlined[5]. He also assisted the TaskerMAN subteam with certain CSS issues they were having such as divs not correctly centering. He had the final say on documentation, and approved them before they were ready for release. He assisted in the moderation of review meetings alongside the project leader.

Tom was a friendly member of the group and had a good relationship with all the other team members. He completed every task which was assigned to him, and being an all-rounder assisted others in any way he could. He found himself with less work than expected in terms of documentation as I (Project Leader) would naturally check everything was in order while compiling everyone's sections together, therefore the amount of input he could possibly have here was limited.

## **2.6 Critical evaluation of the team and the project**

On the whole we performed well as a team. We became comfortable with one another early on in the process, we were aware of each others strengths/weaknesses after an initial meeting and there were no major personality clashes. Roles were allocated very early on in the process and these did not change, with the exception of the deputy quality assurance changing from Tino to Tom Mills as he longer wanted the role. This contributed to a productive work environment. Early on in the process work was divided relatively equally amongst the group as a large part of the work was documentation and initial design. During the latter stages of the process the workload became slightly imbalanced as the more technically proficient members of the team were allocated the most work, particularly during Integration and Testing Week. This created a sense of frustration amongst some team members, lowering the overall morale and team chemistry. To combat this in future the work load should be shared more equally to create a less stressful environment, which in turn might lead to better results from a technical point of view. We were however quite fortunate with our group allocation, in that we had specialists in all the necessary areas of the project, meaning team members usually knew exactly what they were and weren't responsible for. At no point during the process was it necessary for a team member to be punished, either by the project leader or the project manager.

There are a few suggestions we have as to how the project process could be improved. Some team members had great difficulty in using Git as they had no prior experience, leading to situations where some team members would have to upload to the repository on behalf of other members. Though there was an introductory session to Git at the beginning of the year, this clearly wasn't enough, so maybe Git tutorials could be provided during the first year in order to give students a head start. Also the hand-in dates for many of the major documentation deliverables coincided with the hand-in

dates for major assignments of other modules. Whilst understandably this may be unavoidable, it meant that we had to compromise on the amount of resources we could dedicate to the group project.

We've learned many valuable lessons about the software development life cycle during this process. The major one being that you must be prepared for setbacks. During Integration and Testing Week, we had numerous issues with TaskerCLI bugs, some critical, which took longer than desired to fix, meaning that less resources could be allocated to other aspects of the process such as system testing. While this was accounted for in the Risk Analysis of the Project Plan[4] it certainly created a sense of frustration. Another is the importance of review meetings to the quality of documentation. For the Test Specification for which we initially didn't have a formal review our grade was considerably lower (B-) than that of our Design Specification (A), for which we did have a review meeting. The final one is to completely understand the requirements specification before proceeding with development, and if unsure then ask the client for clarification. This is the reason why filtering in TaskerMAN isn't fully supported, it was a misunderstanding of what was required.

### 3 PROJECT TEST REPORT

Here is the final project test report of our project made during Integration and Testing Week. All of our tests passed therefore the table is very clear. The tests outlined in this report are derived from our Test Specification[2].

#### Group 12

Test log: 002

Date: 28/01/2016

Task ID	Tester	Pass/Fail	Fail Comments
MAN-001	T. Mills	Pass	N/A
MAN-002	T. Mills	Pass	N/A
MAN-003	T. Mills	Pass	N/A
MAN-004	T. Mills	Pass	N/A
MAN-005	T. Mills	Pass	N/A
MAN-006	T. Mills	Pass	N/A
MAN-007	T. Mills	Pass	N/A
MAN-008	T. Mills	Pass	N/A
MAN-009	T. Mills	Pass	N/A
MAN-010	T. Mills	Pass	N/A
MAN-011	T. Mills	Pass	N/A
MAN-012	T. Mills	Pass	N/A
MAN-013	T. Mills	Pass	N/A
MAN-014	T. Mills	Pass	N/A
MAN-015	T. Mills	Pass	N/A
MAN-016	T. Mills	Pass	N/A
MAN-017	T. Mills	Pass	N/A
CLI-001	T. Mills	Pass	N/A
CLI-002	T. Mills	Pass	N/A
CLI-003	T. Mills	Pass	N/A
CLI-004	T. Mills	Pass	N/A
CLI-005	T. Mills	Pass	N/A
CLI-006	T. Mills	Pass	N/A
CLI-007	T. Mills	Pass	N/A
CLI-008	T. Mills	Pass	N/A
CLI-009	T. Mills	Pass	N/A
CLI-010	T. Mills	Pass	N/A
CLI-011	T. Mills	Pass	N/A
CLI-012	T. Mills	Pass	N/A
CLI-013	T. Mills	Pass	N/A
CLI-014	T. Mills	Pass	N/A
UI-001	T. Oram	Pass	N/A
UI-002	T. Oram	Pass	N/A
UI-003	T. Oram	Pass	N/A
UI-004	T. Oram	Pass	N/A
UI-005	T. Oram	Pass	N/A
UI-006	T. Oram	Pass	N/A
UI-007	T. Oram	Pass	N/A
UI-008	T. Oram	Pass	N/A
UI-009	T. Oram	Pass	N/A

UI-010	T. Oram	Pass	N/A
UI-011	T. Oram	Pass	N/A
UI-012	T. Oram	Pass	N/A
UI-013	T. Oram	Pass	N/A
UI-014	T. Oram	Pass	N/A
UI-015	T. Oram	Pass	N/A
PER-001	T. Mills	Pass	N/A
PER-002	T. Mills	Pass	N/A
PER-003	T. Mills	Pass	N/A
PER-004	T. Mills	Pass	N/A
PER-005	T. Mills	Pass	N/A
PER-006	T. Mills	Pass	N/A
PER-007	T. Mills	Pass	N/A
PER-008	T. Mills	Pass	N/A
PER-009	T. Mills	Pass	N/A

For reference, here are relevant parts that failed on a previous test log where there was an issue connecting TaskerCLI to the remote database.

## Group 12

Test log: 001

Date: 27/01/2016

Task ID	Tester	Pass/Fail	Fail Comments
CLI-001	T. Mills	Fail	Error message popped up. User Not in the Database, connected to local database.
CLI-002	T. Mills	Fail	Error message popped up, incorrect details.
CLI-003	T. Mills	Pass	N/A
CLI-004	T. Mills	Fail	Login details not stored in local database, did not login and sync to the local database.
CLI-005	T. Mills	Fail	Task list not displayed, could not log in.
CLI-006	T. Mills	Fail	See error message for task CLI-005
CLI-007	T. Mills	Fail	Could not delete task, no tasks displayed.
CLI-008	T. Mills	Fail	Could not complete a task, no tasks displayed.
CLI-009	T. Mills	Fail	Could not edit a task, no tasks displayed.
CLI-010	T. Mills	Fail	See error message for task CLI-009
CLI-011	T. Mills	Fail	Could not synchronise, Js@smith.com could not log in.
CLI-012	T. Mills	Fail	See error message for task CLI-011
CLI-013	T. Mills	Fail	Could not view tasks, no tasks displayed.
CLI-014	T. Mills	Fail	Could not log in, could not sync after 5 minutes.

## **4 PROJECT MAINTENANCE MANUAL**

### **4.1 Program description**

TaskerMAN is the website application for Tasker. Users are able to maintain a session on the site as PHP authentication is a feature. Users are able to create both task data and team member data, which is stored in TaskerSRV. Users are also able to edit both task and team member data, as changes are written to TaskerSRV.

TaskerCLI is the Java application for Tasker. Users login with their email and password as set on TaskerMAN. They're then able to view the tasks which have been allocated to them, which are listed in order of their 'taskid' which is a unique attribute given to every task in TaskerSRV upon task creation. Clicking on a task displays elements associated with the task which are displayed in the order they were produced on TaskerMAN. Comments can be made on task elements, which upon synchronisation will write the comment to TaskerSRV.

### **4.2 Program structure**

Information is contained within the Design Specification[3].

### **4.3 Algorithms**

Information is contained within the Design Specification[3].

### **4.4 The main data areas**

Information is contained within the Design Specification[3].

### **4.5 Files**

Information is contained within the Design Specification[3] A local SQLite database will be required for storage of tasks on the local database.

### **4.6 Suggestions for improvements**

- Front end of the TaskerCLI has been developed using Swing. Initially we had designed the list of the tasks for the logged in user, as a list of generated JPanel components, rather than the JTable that we have used eventually. This would have allowed us to create a richer and more interesting user interface.
- When user selects one of the tasks in the TaskerCLI, the TaskFrame dialog is opened to allow the user to comment on his progress on the clicked task. However in order to save newly created comment, he has to manually press enter before closing the window. This behaviour has a negative impact on the user-friendliness, and could be improved by allowing the user to edit the comment box and press the Complete/Close button on the bottom of the screen, without having to press 'ENTER' beforehand.
- Some refactoring of TaskerCLI code may be needed in order to make the system more understandable. Especially the 'Syncer' class which has overly complex methods which could be further split to improve readability. For example the logIn method, not only deals with the logical part of this operation, but also generates the login box for user's credentials. Ideally each

method should only be responsible for a single task, so login box presenting method could be introduced.

- The list of all the tasks in the system in TaskerMAN can be currently sorted by the completion date, the status of the task and the member that has been assigned to the task. Tasks filtering could be implemented to improve the usability of the system. User could then filter tasks by their status or the member assigned, instead of sorting which still requires the manager to scroll through multiple results as stated in the requirements specification[1].
- A friendly message could be implemented in TaskerMAN to inform the user if there is a problem with accessing the remote database. Currently, TaskerMAN user interface is loaded without the data which makes the system appear less professional.
- Currently it is impossible to enter tasks into the system with names containing special characters like apostrophes. This prevents SQL injections, but has an impact on the usability of the system. A more sophisticated approach to preventing SQL injections and XSS attacks could be implemented.
- It's currently not possible to add a task in TaskerMAN with the same name as another pre-existing task due to the task title being a unique attribute in TaskerSRV.
- Finally README with the instructions to install TaskerMAN could be updated with the information about the necessary change of permissions in order for the system to work. TaskerCLI installation script should also be moved from the src folder to the TaskerCLI.Installation.

#### **4.7 Things to watch for when making changes**

In an application of this size, it is inevitable that a change to one part of the system can brake something else. It is therefore essential to back up the original version of the system before attempting to fix or improve anything. Version control system like git seems to be an obvious and recommended approach.

Due to the way Data Object Access pattern has been used, any changes to the database structure (TaskerSRV) will require significant changes to both the `uk.ac.aber.cs221.group12.taskercli.business` and `uk.ac.aber.cs221.group12.taskercli.data` TaskerCLI packages. Within the business package, JavaBean entities abstract the tables within the database. Therefore if you make a change to the underlying database, you have to make changes in appropriate entities as well. Same problem applies to the data package. Because within the front end of the program, developer can access a given TeamMember object by its email, quite possibly the whole chain of method calls that rely on each other may have to be changed, along with the SQL queries on the top of each file.

Developers have to be especially careful when making any changes to the synchronisation part of the TaskerCLI. Before making any changes it is advisable to refer to the Design Specification[1] to understand how this behaviour is achieved. The main concept is fairly straightforward: you retrieve the remote TeamMember object, retrieve the locally stored TeamMember object, merge them together and finally login and use its data to populate the view. Things however become quickly more complex, when you consider all the possible scenarios like lack of the Internet connection and the precedence of different information about the team member (e.g. task list on the remote database takes precedence over the local version, but comments to any existing tasks have to be taken from the local one).

From a TaskerMAN perspective there exist many variables that span across several .php files e.g. the variable \$Username is present on every page (except for login.php) as it's echoed on the top menu bar of every page. If you were to alter this variable where it's initialised in login.php, then you would have to alter it across all the pages. Developers must also be wary of editing the connector.php file. Since this file is responsible for accessing the TaskerSRV, changing the database login credentials will mean that none of the data present in TaskerSRV will be displayed. This may be desirable if you have a different database you'd rather connect to, however the new database would have to have exactly the

same tables and table names as the ones present in TaskerSRV, or significant alterations would need to be made across both TaskerMAN and TaskerCLI.

#### **4.8 Physical limitations of the program**

Tasker system requirements:

- Apache web server capable of running PHP to deploy TaskerMAN.
- Pre-installed MySQL instance running either alongside the Apache web server, or separately to install TaskerSRV.
- Personal computer running any operating system capable of running Java Runtime Environment 7 or above to install TaskerCLI.

#### **4.9 Rebuilding and Testing**

The whole system can be run and tested locally, provided developer has access to locally running MySQL and Apache. For instance if Ubuntu OS is used for development, both these requirements can be installed after a quick search through the Internet. Having installed Apache and MySQL, developer needs to acquire the whole source code of the program and follow the installation guide provided in order to deploy each of the components.

- TaskerMAN part of the system can easily be edited by changing the contents of PHP source files.
- TaskerSRV can be edited by editing the SQL script which builds the initial structure of the database. Whenever you need to revert the database state back to its original state, you should follow the TaskerSRV installation guide and run the SQL script against the installation of your MySQL database.
- TaskerCLI can easily be edited using any IDE capable of importing Maven projects. You can import the whole project by selecting the `pom.xml` file, which describes the whole build and contains details about TaskerCLI dependencies. After the initial import, it is advisable to select the "build and clean" option within the IDE. This will download all the required dependencies and run all the JUnit tests inside the test package. If you have a problem with accessing either your local or remote database, please make sure the properties files with MySQL and SQLite connection details are pointing at your database and SQLite db file.

## REFERENCES

- [1] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS. 1.1 Release.
- [2] *Software Engineering Group Projects* Test Specification. L. Jones, T. Oram, W. Jones, T. Mills, 1.1 Release.
- [3] *Software Engineering Group Projects* Design Specification. L. Jones, T. Oram, T. Garapasi, M. Goly, W. Jones, A. Neaves, J. Mir, T. Mills, 1.9 Release.
- [4] *Software Engineering Group Projects* Project Plan. L. Jones, 1.2 Release.
- [5] *Software Engineering Group Projects* Java Coding Standards. C. J. Price, A. McManus, SE.QA.09. 17 Release.

## DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2016-02-13	Initial creation	M.Goly [mwg2]
1.1	N/A	2016-02-14	Included End-of-project report and added Project Plan, Design Specification and Test Specification as appendices	L.Jones [luj9]
1.2	N/A	2016-02-14	Included Project Test Report	L.Jones [luj9]



# Software Engineering Group Project Project Plan

*Author:* Luke Jones  
*Config. Ref.:* SE-12-PM  
*Date:* 2016-02-14  
*Version:* 1.2  
*Status:* Release

Department of Computer Science,  
Aberystwyth University,  
Aberystwyth,  
Ceredigion, SY23 3DB,  
U.K.

©Aberystwyth University 2016

## **CONTENTS**

<b>1</b>	<b>INTRODUCTION</b>	<b>2</b>
1.1	Purpose of this Document . . . . .	2
1.2	Scope . . . . .	2
1.3	Objectives . . . . .	2
<b>2</b>	<b>GANTT CHART</b>	<b>3</b>
<b>3</b>	<b>RISK ANALYSIS</b>	<b>4</b>
	<b>REFERENCES</b>	<b>5</b>
	<b>DOCUMENT HISTORY</b>	<b>5</b>

## **1 INTRODUCTION**

### **1.1 Purpose of this Document**

The purpose of this document is to outline our project plan, to provide data on the time frame in which tasks are to be completed and to provide information on risks associated with the project, and how their effects may be mitigated.

### **1.2 Scope**

This document specifies the time frame in which we aim to begin and complete tasks and the team member(s) who will work on each task. This document also highlights any risks involved with the project with regard to delays, and includes instruction on how the effects of such delays can be mitigated.

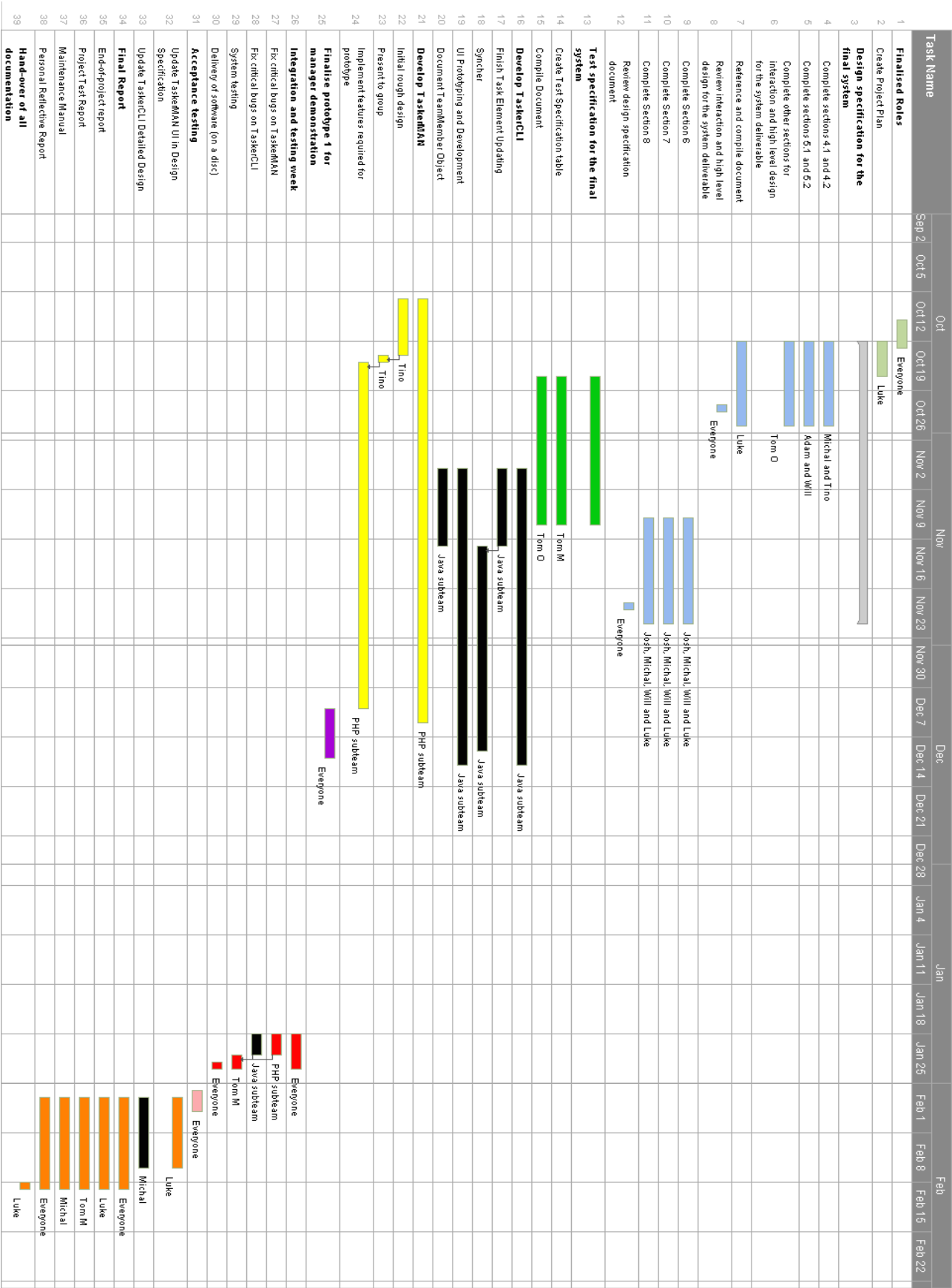
This document should be read and closely monitored by all project members. It is assumed that the reader is already familiar with QA document SE.QA.05b [1].

### **1.3 Objectives**

The objectives of this document are as follows:

- To provide group members with a prior knowledge of when major milestones will be targeted.
- Illustrate the optimal dates for the beginning and completion of tasks/subtasks.
- Outline group member(s) who will be responsible for the completion of each task.
- Identify parts of the plan with potential to cause delay, as well as any outside factors that could make task completion longer than necessary.
- Advise on course of action in the event of a delay occurring.

2 GANTT CHART



### 3 RISK ANALYSIS

<b>Risk</b>	<b>Probability of Risk 0-10</b>	<b>Risk Severity 0-10</b>	<b>Risk Description</b>	<b>Prevention Procedures</b>
Illness to member(s) of group	5	3	One or more members of the group are absent	Agile workload distribution, roles can be reallocated
Members not uploading to GitHub	5	6	A group member fails to upload their work to GitHub	Disciplinary action enforced by project leader
Unit testing not adequate	4	9	Testing falls below specifications	Enforce a testing team with multiple members with a wide skill range
Failure to understand brief requirements	5	5	Member of the group does not understand specific requirements	Have project leader seek clarification from superiors
University network malfunctioning	2	9	The university network is not working	Ensure group members have a backup of work they are currently working on
Members go over 80 hour limit	5	8	Group member goes over the allowed 80 hour limit	Reallocate work to different members of the group with less work time
Skill levels are not adequate	5	6	Members of the group have a low level of skill in specific areas	Assign group members tasks according to their strengths
Members misunderstanding what is asked of them	4	7	Some members of the group may not understand what they have been assigned to do	Have meeting minutes uploaded as soon as possible in clear detail
Internal conflict within group	4	6	Group members have a major disagreement causing a decline in work productivity	Aim to provide a comfortable work environment where moral is high and everyone agrees on what must be achieved.
Functional requirement of TaskerCLI not attainable	4	9	As a team we fail to fully implement a functional requirement of TaskerCLI as required[2].	Discuss the possibility of partial implementation with a project manager.
Functional requirement of TaskerMAN not attainable	4	9	As a team we fail to fully implement a functional requirement of TaskerMAN as required[2].	Discuss the possibility of partial implementation with a project manager.

TaskerCLI/ TaskerMAN interface is not user-friendly	3	8	The interfaces of the TaskerCLI/ TaskerMAN systems are overly complex and not easy to use for non-developers of the software.	Get non-developers to complete a list of spec- ified tasks on the soft- ware without any major difficulty.
--	---	---	---	--

## REFERENCES

- [1] *Software Engineering Group Projects* General Documentation Standards. C. J. Price, N. W. Hardy, B.P. Tiddeman SE.QA.03. 1.8 Release.
- [2] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS. 1.1 Release.

## DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2015-10-22	Initial creation	Luke Jones
1.1	N/A	2015-20-11	Updated Risk Analysis and Gantt Chart	Luke Jones
1.2	N/A	2016-02-14	Updated Risk Analysis and Gantt Chart	Luke Jones

# Software Engineering Group Project

## Test Specification

*Author:* L. Jones, T. Oram, W. Jones, T. Mills  
*Config. Ref.:* SE-12-TS  
*Date:* 2016-0214  
*Version:* 1.1  
*Status:* Release

Department of Computer Science,  
Aberystwyth University,  
Aberystwyth,  
Ceredigion, SY23 3DB,  
U.K.

©Aberystwyth University 2016

## **CONTENTS**

<b>1</b>	<b>INTRODUCTION</b>	<b>2</b>
1.1	Purpose of this document . . . . .	2
1.2	Scope . . . . .	2
1.3	Objectives . . . . .	2
<b>2</b>	<b>TEST SPECIFICATION</b>	<b>3</b>
	<b>REFERENCES</b>	<b>14</b>
	<b>DOCUMENT HISTORY</b>	<b>14</b>



## **1 INTRODUCTION**

### **1.1 Purpose of this document**

The purpose of this document is to outline how we aim to enforce good testing practice, and as such contains a 'Test Specification' which demonstrates how to test against the different requirements as specified in the requirements specification[1], as well as displaying the criteria that must be met in order to pass tests.

### **1.2 Scope**

This document specifically covers the 'Test Specification' section of the Test Procedure Standards Quality Assurance Document[2]

This document should be read by all project members. It is assumed that the reader is already familiar with the Test Procedure Standards [2].

### **1.3 Objectives**

The objectives of this document are as follows:

- Identify where a test will be necessary against the requirements specificationse.qa.rs and give it a unique ID.
- Explain the exact function/requirement that is being tested.
- Explain how a user would undertake a test, as well as the desired output if successful.
- Explain the criteria that must be met by the test in order for it to be deemed as "Passed".

## 2 TEST SPECIFICATION

Test Reference	Reference Requirement	Test Content	Input	Output	Pass Criteria
MAN-001	FR3	TaskerMAN should support the creation of team member data	In the Create user page of taskerMAN create a new user with name “John Smith” and email “js@aber.ac.uk”	The Show users page in TaskerMAN will now include the user “John Smith”	All data entered should be correct and the show team member’s page displays js@aber.ac.uk
MAN-002	FR3	TaskerMAN should support the creation of team member data	In the create team member page of TaskerMAN try to enter in “hello” as the email address.	Client side form validation should execute.	Form validation should notify user that it is not a valid email address.
MAN-003	FR3	TaskerMAN should support the creation of team member data	In the Create team member page of TaskerMAN, create a team member with their first name as “DROP TABLE User”	An SQL query will be executed with the first name set as “DROP TABLE User?”.	Remote database will still be intact if protection against SQL injection is correctly set up.
MAN-004	FR3	TaskerMAN should support the updating of team member data	Under the edit team member page of TaskerMAN select to edit “js@smith.com” and change its name to “Tom Smith”.	The show team members page in TaskerMAN should now have the updated team member	Form validation should notify user that it is not a valid email address.
MAN-005	FR3	TaskerMAN should support the updating of team member data.	Under the edit team member page of TaskerMAN edit “js@smith.com”’s email to be “hello”.	Client side form validation should execute.	The show tasks page in TaskerMAN will show the updated task with it’s new title
MAN-006	FR3	TaskerMAN should support the deletion of team member data.	Under the edit team member page of TaskerMAN delete the team member with the name “Tom Smith”	The team member will be deleted from the database.	The show team member’s page in TaskerMAN should no longer include the team member “js@smith.com”

MAN-007	FR3	TaskerMAN should support the creation of task data.	In the create team Member page of TaskerMAN create a new team member with name "John Smith" and email "js@aber.ac.uk" (Test subject needed)	Query will be executed to Insert into the team member table the new team member.	Database should now include the team member "js@aber.ac.uk."
MAN-008	FR4	TaskerMAN should support the updating of task data	Under the create task page of TaskerMAN create a task with: Title = "make coffee", Start Date = "30/01/2016", End Date = "1/02/2016", TaskElement1 title = "get Cup", TaskElement2 title= "grind beans", and allocate it to "js@smith.com"	The task should be added to the remote database.	The show tasks page in TaskerMAN will show the task entered and should have the exact input data and have a Task Status of ?ALLOCATED? and will be allocated to "js@smith.com?".
MAN-009	FR4	TaskerMAN should support the creation of task data	Under the create task page of TaskerMAN create a task with: Title = "make coffee", Start Date = "30/01/2016", End Date = "1/01/2000", TaskElement1 title = "get Cup", TaskElement2 title= "grind beans", and allocate it to "js@smith.com"	Client Side form validation should execute.	User should be notified that they entered an end date that was before the start date.

MAN-010	FR4	TaskerMAN should support the creation of task data (Testing against SQL Injection)	Under the create task page of TaskerMAN create a task with: Title = "DROP TABLE Tasks", Start Date = "30/01/2016", End Date = "1/02/2016", TaskElement1 title = "get Cup", TaskElement2 title="grind beans", and allocate it to "js@smith.com"	SQL Query that partly contains the String "DROP TABLE Tasks" will be executed	Database should still be intact.
MAN-011	FR4	TaskerMAN should support the updating of task data.	Under the edit task page in TaskerMAN select the "Make Coffee" task. Choose to add an extra task element: Title="pour water".	The task should be updated in the database	The show tasks page in TaskerMAN will show the updated task with an extra Task Element.
MAN-012	FR4	TaskerMAN should support the deletion of task data.	Under the edit task page in TaskerMAN, select the task "make coffee". Delete this task.	The selected task will be deleted from the database.	The show tasks page in TaskerMAN should now not include the task "make coffee" as it should've been removed from the remote database.

MAN-013	FR5	TaskerMAN should support the re-allocation of a task	Create a new team member, first name: "testy", last name "mctesty", email: "test@test.test" and a password of "123". Create a task with: Title = "make coffee", Start Date = "30/01/2016", End Date = "1/02/2016", TaskElement1 title = "get Cup", TaskElement2 title= "grind beans". Allocate it to test@test.test. Go to edit tasks page, select the task and choose to re-allocate it to "js@smith.com".	The task should be updated in the remote database.	The show tasks page in TaskerMAN should now show the task with "js@smith.com" allocated to it.
MAN-014	FR6	TaskerMAN should support the abandonment of a task	Under the edit task page in TaskerMAN select to edit the task "Make coffee", change its task status to "Abandoned"	The task should be updated in the remote database.	The show tasks page in TaskerMAN should now show the task with a task status of "Abandoned".
MAN-015	FR7	TaskerMAN should support the viewing of tasks.	Create a new task with: Title = "Hey Im a task look at me", Start Date = "12/02/2016", End Date = "12/02/2016", TaskElement1 = "look at me pls" and allocate it to "js@smith.com" (This is so that at least one task exists on the remote database). On TaskerMAN click on the "View Tasks" page	A query should be sent to list all tasks in the database.	The task "Hey Im a task look at me" will be displayed, along with any other tasks should they exist in the remote database

MAN-016	FR7	TaskerMAN should support the filtering of tasks dependant on allocated team member.	On TaskerMAN in the “view tasks” page select the option to filter by team member	A query should be sent to the remote database to list all the tasks by team member	The list of tasks that are displayed should be displayed by member in alphabetical ascending order.
MAN-017	FR7	TaskerMAN should support the filtering of tasks dependant on task status.	On TaskerMAN in the “view tasks” page select the option to filter by task status	A query should be sent to the remote database to list all the tasks by task status	The list of tasks that are displayed should be displayed by task status in alphabetical ascending order.
CLI-001	FR8	TaskerCLI should be able to identify the member upon logging in	Upon login, enter the user credentials: Username: “jsmith@smith.com” Password: “123”.	A query should be sent to the database to locate the user with the data entered	The user will then be logged in and on the main page of TaskerCLI the current user will say “logged in as “Tom Smith””
CLI-002	FR8	TaskerCLI should be able to identify the member upon logging in	Upon login enter in the credentials: Username = “bob”, Password = “1”	Authenticator will check to see if the user in the database.	User should be notified that their credentials were incorrect.
CLI-003	FR8	TaskerCLI should be able to identify the member upon logging in	Upon login enter in the credentials: Email= “DROP TABLE User”	A query will be run that partly contains the String “DROP TABLE User”.	The remote database should still be intact.
CLI-004	FR8	TaskerCLI should store the login details locally for future use.	Upon login enter in the credentials: Userame = “js@smith.com” and the password = “123”, then logout.	The users details will be stored in the local SQLite database (password should be hashed).	“js@smith.com” should have their credentials stored in the local database (password also hashed).
CLI-005	FR8a	TaskerCLI should be able to store data on the user’s computer locally for the user that is logged in.	Log in to the user “js@smith.com” (password: 12345).	This should synchronise and store all data (locally) associated with “js@smith.com”.	The Local SQLite Database should contain the tasks for “js@smith.com”

CLI-006	FR9	TaskerCLI should be able to synchronise to get all the tasks for the user that is logged in at the time and should only be ones that have a task status of "allocated"	Create a new task on TaskerMAN with: Title = "test code", Start date = "30/01/2016", End date = "01/02/2016", TaskElement1 = "module testing", TaskElement2 = "system testing". Allocate it to j"s@smith.com". On TaskerCLI log into the user "js@smith.com"	This should synchronise with the remote database and display the list of tasks including the newly assigned task to "js@smith.com".	JTable displays added task.
CLI-007	FR10	Synchronisation should support the deletion of completed/abandoned tasks.	Using TaskerMAN, select the task "make coffee" and set its Task Status = "ABANDONED". Login to "js@smith.com" (on TaskerCLI)	Upon logging in, TaskerCLI should synchronise with the remote database and no longer contain the "make coffee" task.	The table will no longer contain the "make coffee" task.
CLI-008	FR9	Synchronisation should support the deletion of completed/abandoned tasks.	Log into "js@smith.com" on TaskerCLI, select the task "test code" and select complete task.	After editing the task, TaskerCLI will synchronise with the remote database and should no longer display the "test code" task.	The completed task should be removed from the table.

CLI-009	FR10	TaskerCLI must be able to support the local editing of tasks (Task element comments)	On TaskerMAN create a task with: Title = "Do chores", Start Date = "30/01/2016", End Date= "1/02/2016", TaskElement title = "Make a list of chores", TaskElement2 title= "Find way of procrastinating". Allocate it to "js@smith.com". Then login to TaskerCLI using "js@smith.com". Edit the task "Do chores" and enter the comment "go for a pint" on the Task Element "Find way of procrastinating".	TaskerCLI should synchronise with the database and update comment for the Task Element.	Upon selecting the task from now on the comment "go for a pint" should remain associated with the element "Find way of procrastinating".
CLI-010	FR10	TaskerCLI must be able to support the local editing of tasks (Task element comments)	Login to TaskerCLI with the user "js@smith.com", edit the task "Do chores" and enter the comment "DROP TABLE Tasks" on the Task Element "Make a list of chores".	An SQL query will be executed which will partly contain the String "DROP TABLE Tasks".	Remote database should still be intact.
CLI-011	FR11	Synchronisation must happen on startup	On TaskerMAN proceed to edit tasks, edit the task "Make a list of chores" by changing the Task Element "Find way of procrastinating" to "do chores tomorrow instead". Login to TaskerCLI with "js@smith.com"	TaskerCLI should synchronise with the database after logging in and update the Task Element.	After logging into TaskerCLI select the task "Make a list of chores". The Task Element "Find way of procrastinating" should now be "do chores tomorrow instead".



CLI-012	FR11	Synchronisation must happen before editing a task.	Login to “js@smith.com” on TaskerCLI. On TaskerMAN update the task “Do chores” so that the Task Element “Make a list of chores” is now “Think about chores”. Then select the task on TaskerCLI.	After selecting the task, TaskerCLI should synchronise with the remote database and update the Task Element.	After logging into TaskerCLI select the task “Make a list of chores”. The Task Element should now be called “Think about chores”.
CLI-013	FR11	Synchronisation must happen after editing a task.	Login to “js@smith.com” on TaskerCLI. Select the task “Do chores” and add the comment “think harder” to the task element “Think about chores”, then close the task.	After selecting the task, TaskerCLI should synchronise with the remote database and update the Task Element.	The updated task element and comment should now be visible on TaskerMAN.
CLI-014	FR11	TaskerCLI should synchronise with the database every 5 minutes of it being logged on.	Login to “js@smith.com” on TaskerCLI. On TaskerMAN edit the task “Think about chores” and set it’s status to “abandoned”. Go back to TaskerCLI and wait for automatic synchronisation to occur.	After 5 minutes TaskerCLI should synchronise.	The task should no longer be visible on TaskerCLI after synchronisation.
UI-001	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to create a user (for themselves) in TaskerMAN	User should be able to create a team member without asking for help	Team member has been created without any difficulty
UI-002	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to edit a team member In TaskerMAN	User should be able to edit a team member without asking for help	The team member data can be edited by the user without any difficulty

UI-003	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to delete a team (not themselves) member in TaskerMAN	User should be able to delete a team member without asking for help	The team member data should be deleted without any difficulty
UI-004	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to create a new task in TaskerMAN and set the task to be allocated to them	User should be able to create a new task without asking for help	A task should be created by the user and allocated to themselves without any difficulty
UI-005	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to edit the task that is allocated to them to have a task status of 'completed' in TaskerMAN	User should be able to edit a new task without asking for help	The user should be able to change the status of the task to "Completed" without any difficulty
UI-006	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to delete a task (not the task that was allocated to them) TaskerMAN	User should be able to delete a new task without asking for help	The user should be able to delete the task without any difficulty
UI-007	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to view members of the group	User should be able to view the team members in the group without asking for help	The user should be able to view the team members in the group without any difficulty
UI-008	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	non-developer of the group try to view the tasks	User should be able to view the tasks in the group, without asking for help	The user should be able to view the tasks allocated to group members without any difficulty
UI-009	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to Filter tasks that are allocated to them	User should be able to see the task that was allocated to them, without asking for help	The user should be able to filter tasks that are allocated to them without any difficulty
UI-010	IR1	The user interfaces of TaskerMAN should be easy to use by regular computer users	A non-developer of the group try to Filter tasks that are completed	User should be able to see the task that they created (that has a task status of 'completed')	The user should be able to filter tasks that have been completed without any difficulty

UI-011	IR1	The user interfaces of TaskerCLI should be easy to use by regular computer users	A non-developer of the group try to login to TaskerCLI	User should be able to login without asking for help	The user should be able to login to TaskerCLI without any issue
UI-012	IR1	The user interfaces of TaskerCLI should be easy to use by regular computer users	A non-developer of the group try to view tasks in TaskerCLI	User should be able to navigate to be able to view tasks without asking for help	The user should be able to view the tasks and their respective task elements without any difficulty
UI-013	IR1	The user interfaces of TaskerCLI should be easy to use by regular computer users	A non-developer of the group try to edit a task in TaskerCLI	User should be able to edit a task without asking for help	The user should be able to edit a task without any difficulty
UI-014	IR1	The user interfaces of TaskerCLI should be easy to use by regular computer users	A non-developer of the group try to use the search bar to search for a task	User should be able to search for a task using the search bar without asking for help	The user should be able to use the search bar to search for a specific task without any difficulty
UI-015	IR1	The user interfaces of TaskerCLI should be easy to use by regular computer users	A non-developer of the group try to filter tasks by those allocated to them	User should be able to filter tasks that were allocated to them without asking for help	The user should be able to filter the tasks without any difficulty
PER-001	PR1	Program should respond to user input in a minimum of a second	Enter a user's credentials into the login page of TaskerCLI and click enter	The page should log in in less than a second	The page logged in in less than a second
PER-002	PR1	Program should respond to the user input in a minimum of a second	Press show tasks in TaskerCLI	The tasks should be displayed in less than a second	The tasks were displayed in less than a second
PER-003	PR1	Program should respond to the user input in a minimum of a second	Enter data for a new team member in TaskerMAN	The member should be visible within one second	The user was visible within one second
PER-004	PR1	Program should respond to the user input in a minimum of a second	Enter data for a new Task in TaskerMAN	The task should be visible within one second	The task was visible within one second

PER-005	PR2	TaskerCLI should run on any machine supporting Java	Run TaskerCLI on a Linux machine	Program should execute	Program executed
PER-006	PR2	TaskerCLI should run on any machine supporting Java	Run TaskerCLI on a Windows machine	Program should execute	Program executed
PER-007	PR2	TaskerCLI should run on any machine supporting Java	Run TaskerCLI on a Mac	Program should execute	Program executed
PER-008	PR2	TaskerSRV must run on a suitable web server	Test connection to database using JDBC by logging into TaskerCLI	Should connect successfully	JDBC connected to the database successfully
PER-009	PR2	TaskerMAN should be deployable on an apache web server	Deploy TaskerMAN to an apache web server	Should work correctly	Worked correctly

## REFERENCES

- [1] *Software Engineering Group Projects* Requirements Specifications. N. W. Hardy, SE.QA.RS. 1.2 Release.
- [2] *Software Engineering Group Projects* Test Procedure Standards C. J. Price, N.W.Hardy and B.P.Tiddeman, SE.QA.06. 1.8 Release.

## DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2015-11-12	Initial creation for LaTeX	L. Jones
1.1	N/A	2016-14-02	Updated by Tom M based on Nigel's feedback, final version	L. Jones

# 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:* 2016-02-14  
*Version:* 1.9  
*Status:* Release

Department of Computer Science,  
Aberystwyth University,  
Aberystwyth,  
Ceredigion, SY23 3DB,  
U.K.

©Aberystwyth University 2016

## CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>2</b>
1.1	Purpose of this document . . . . .	2
1.2	Scope . . . . .	2
1.3	Objectives . . . . .	2
<b>2</b>	<b>DEPLOYMENT DESCRIPTION</b>	<b>3</b>
2.1	Applications in the system . . . . .	3
2.2	Applications interface . . . . .	3
<b>3</b>	<b>INTERACTION DESIGN</b>	<b>5</b>
3.1	Use-cases . . . . .	5
3.1.1	TaskerMAN Use-Case . . . . .	5
3.1.2	TaskerCLI Use-Case . . . . .	6
3.2	User Interface design . . . . .	7
3.2.1	TaskerMAN Interface Design . . . . .	7
3.2.2	TaskerCLI Interface Design . . . . .	11
<b>4</b>	<b>Component Description</b>	<b>14</b>
4.1	TaskerMAN Component Description . . . . .	14
<b>5</b>	<b>Significant Classes</b>	<b>15</b>
5.1	TaskerCLI Significant Classes . . . . .	15
5.1.1	Description of each significant Class . . . . .	16
5.2	TaskerMAN Significant Classes . . . . .	18
5.2.1	Description of each significant Class . . . . .	18
<b>6</b>	<b>Detailed Design</b>	<b>21</b>
6.1	TaskerSRV Entity-Relationship Diagram . . . . .	21
6.2	TaskerCLI Sequence Diagrams . . . . .	21
6.3	Spike Programming . . . . .	24
6.4	TaskerMAN Sequence Diagram . . . . .	24
6.5	TaskerMAN Object Diagrams . . . . .	25
6.5.1	TaskerMAN Object Relationship between the ADD_USERS and EDIT_USERS CLASS . . . . .	25
6.5.2	TaskerMAN Object Relationship between the ADD_TASK and EDIT_TASKS CLASS . . . . .	25
	<b>REFERENCES</b>	<b>26</b>
	<b>DOCUMENT HISTORY</b>	<b>27</b>

## **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:

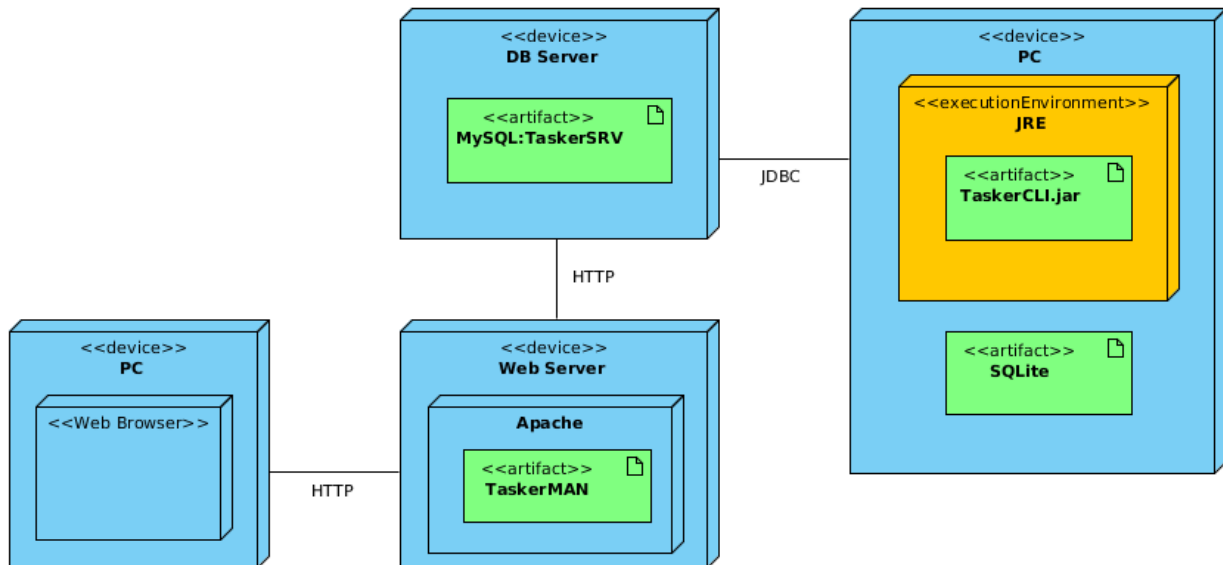
- Explain the way the systems interact with one another and the platforms in which they operate with the assistance of UML diagrams.
- Provide detail on the way in which "actors" are expected to interact with the system through the use of use-case diagrams.
- Present a basic User-Interface design for 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.



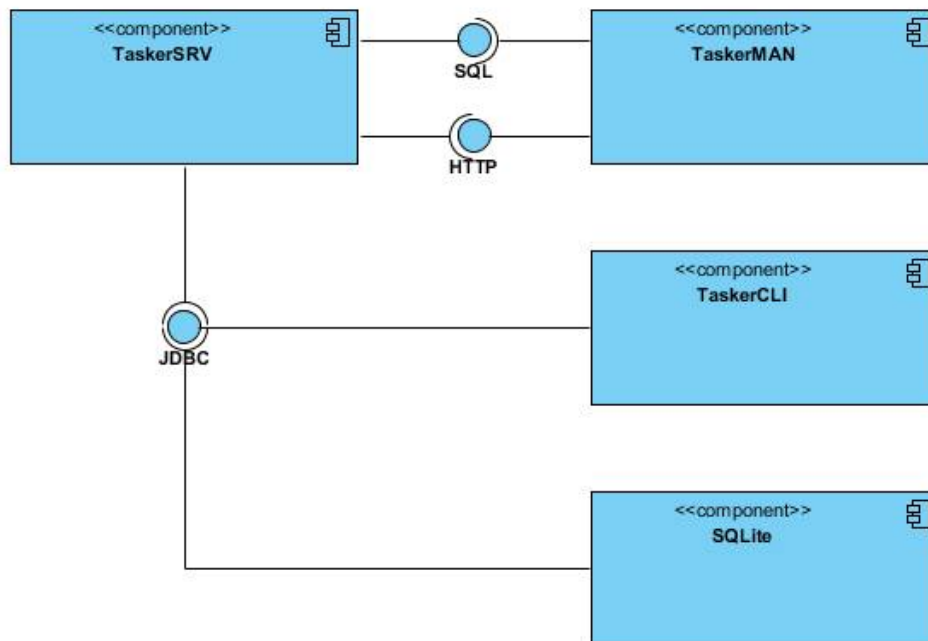
The whole system will be composed of the following three components:

- 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[2], 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.

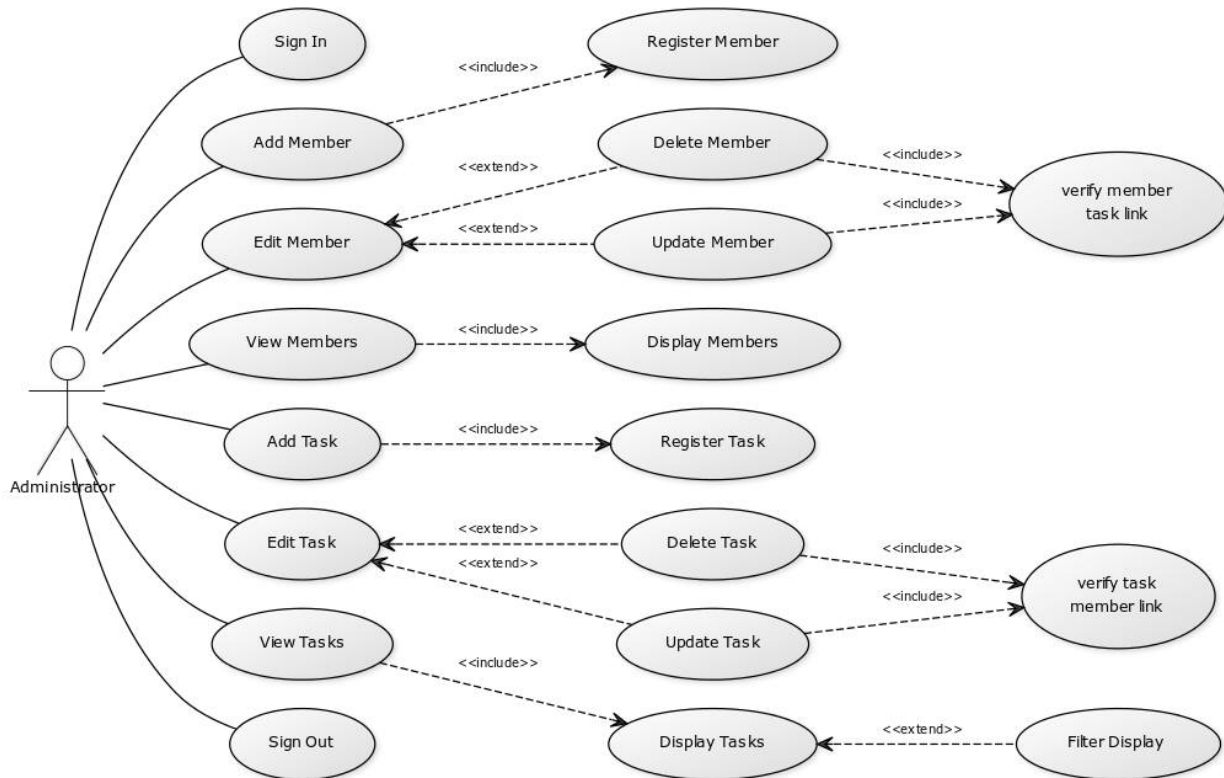
TaskerCLI - This component representing the client has one provided interface (JDBC). Since it is using Java to connect to the database, TaskerSRV as well as SQLite will require that the interface JDBC be implemented on the client or else there will be no database connection.

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



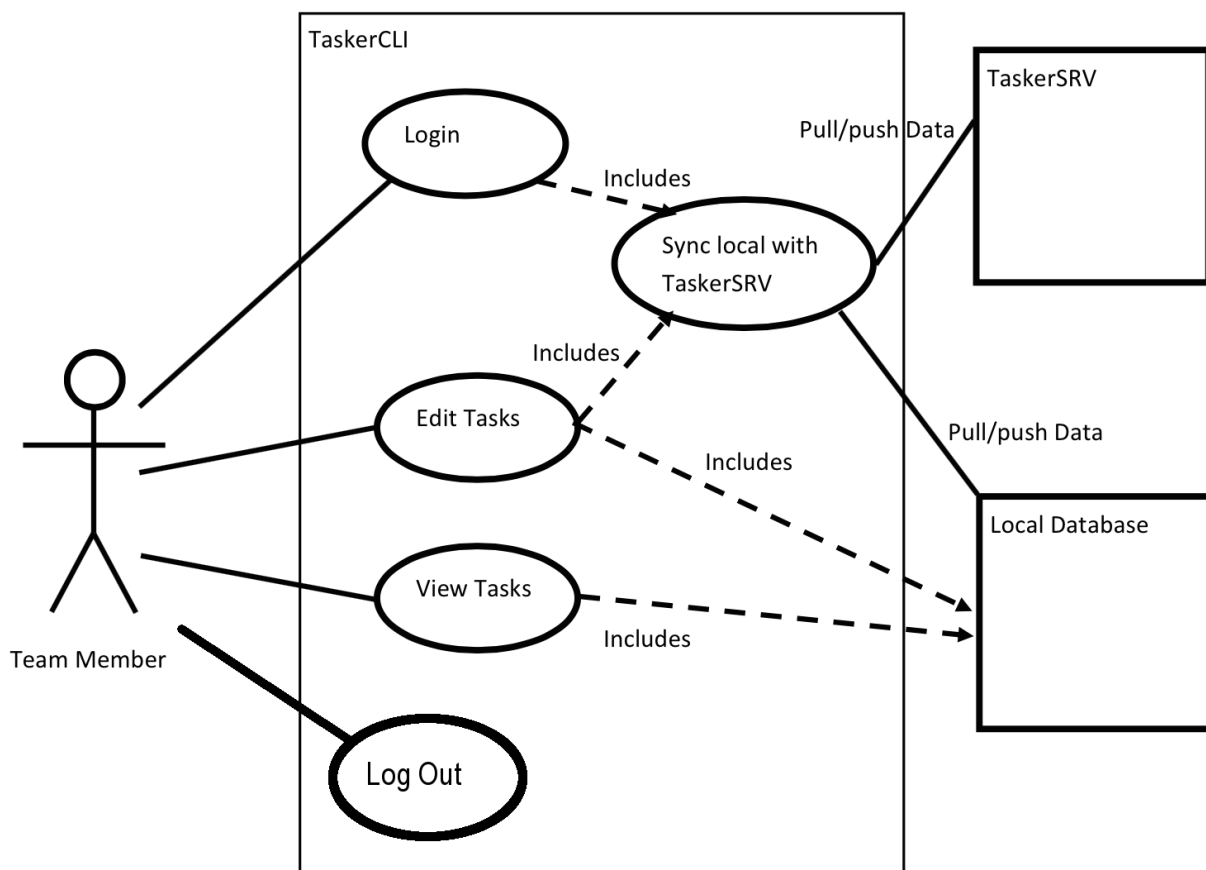
The TaskerMAN use case diagram above depicts clearly how the web administration application will be able to be used by the administrator. The necessary functions depicted in the specification requirements[2] are clearly accessible to the administrator after logging into the system. These functionalities 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



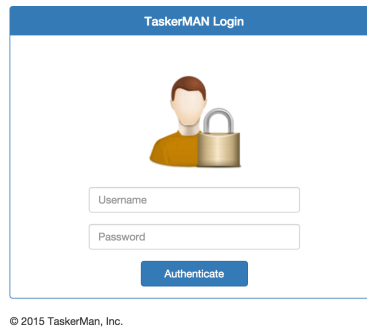
The TaskerCLI use-case diagram above depicts how TaskerCLI will be used by "actors". The diagram illustrates the different functions the application must contain as outlined in the requirements specification [2]. These are user identification (login), local storage of tasks, task synchronisation and 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.

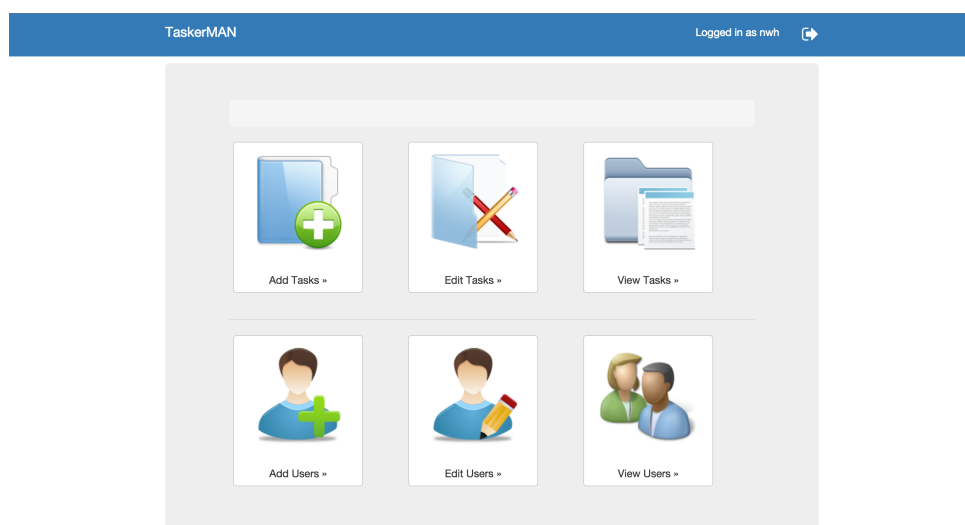
4. The actor confirms changes and clicks 'Edit Task', this will trigger another synchronisation to occur, and the changes will be pushed to TaskerSRV and to the SQLite database.
5. The user has completed their actions and clicks 'Log Out', terminating the system.

## 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.

TaskerMAN

Logged in as nwh

←

**Add User**

First Name

Last Name

Member Email

Member Password

Add User

© 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.

TaskerMAN

Logged in as nwh

←

**Edit Users**

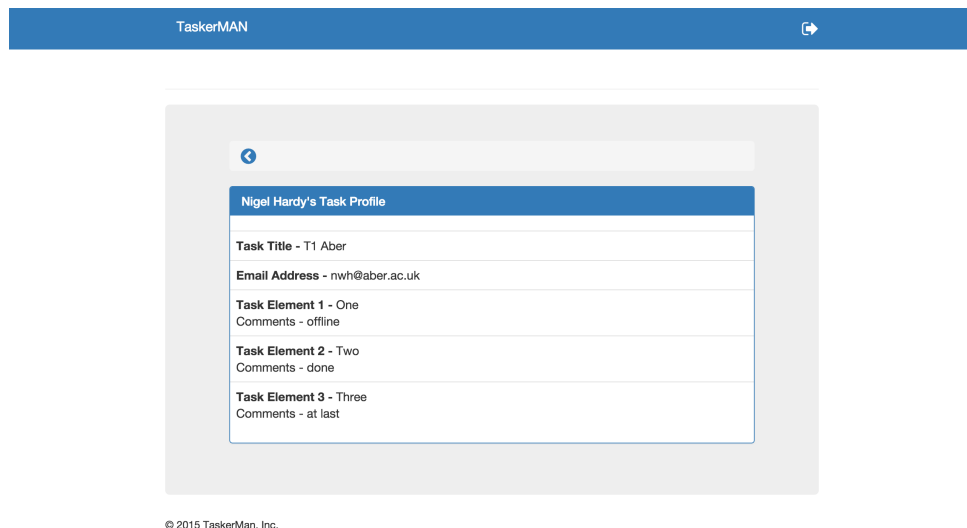
First Name	Last Name	Email	Update	Delete
Bob	Smith	bob@smith.com		
Jane	Adams	jane@gmail.com		
Michal	Goly	m.goly@goly2.com		
Nigel	Hardy	nwh@aber.ac.uk		
Rich	Jones	rj12@aber.ac.uk		

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.

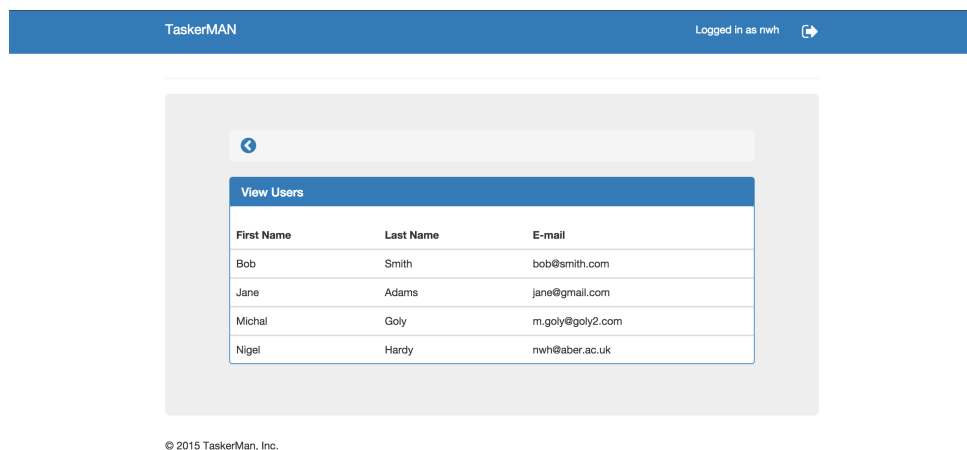
Clicking 'Add Tasks' from the home page will re-direct to the task portal with the address '/tasker-MAN/AddTaskBoot.php'. This is where users can make tasks which will then be allocated to members in the remote database as required [2]. This task portal layout will also be used when updating tasks, the 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.

Expected Completion Date	Title	Member Assigned	Task Status
2015-10-30	Do shopping	m.goly@goly2.com	Allocated <span>More ▾</span>
2016-02-02	t6	bob@smith.com	Allocated <span>More ▾</span>
2016-02-02	t4	bob@smith.com	Allocated <span>More ▾</span>
2016-02-02	t3	bob@smith.com	Allocated <span>More ▾</span>
2016-02-02	t2	bob@smith.com	Allocated <span>More ▾</span>
2016-02-02	t1	bob@smith.com	Allocated <span>More ▾</span>
2016-02-03	T1 Aber	nwh@aber.ac.uk	Allocated <span>More ▾</span>

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.



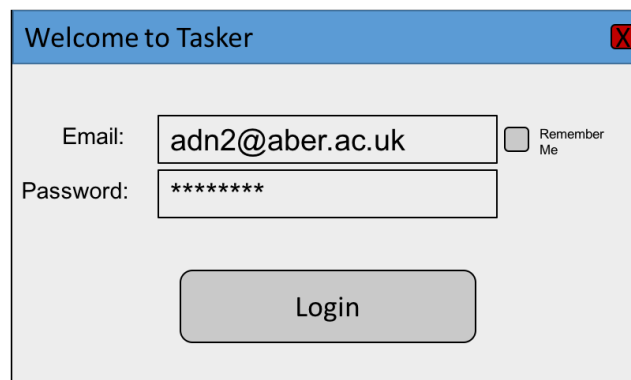
Here is the display of clicking on the 'More' button of a particular task. This view allows the user to view a task, the user assigned to the task, the task elements associated with the task, as well as comments made about the task elements from the TaskerCLI application as required [2]. The task elements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



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



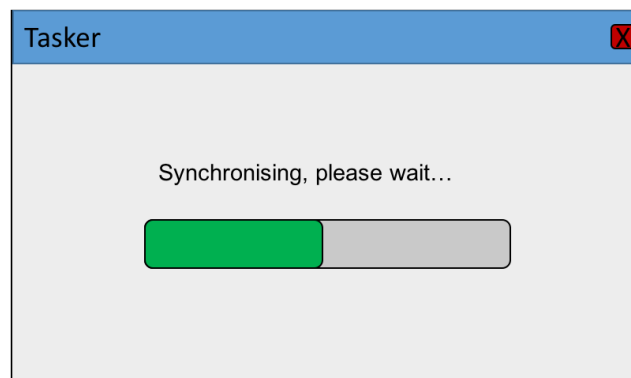
Welcome to Tasker

Email:  ☐ Remember Me

Password:

Login

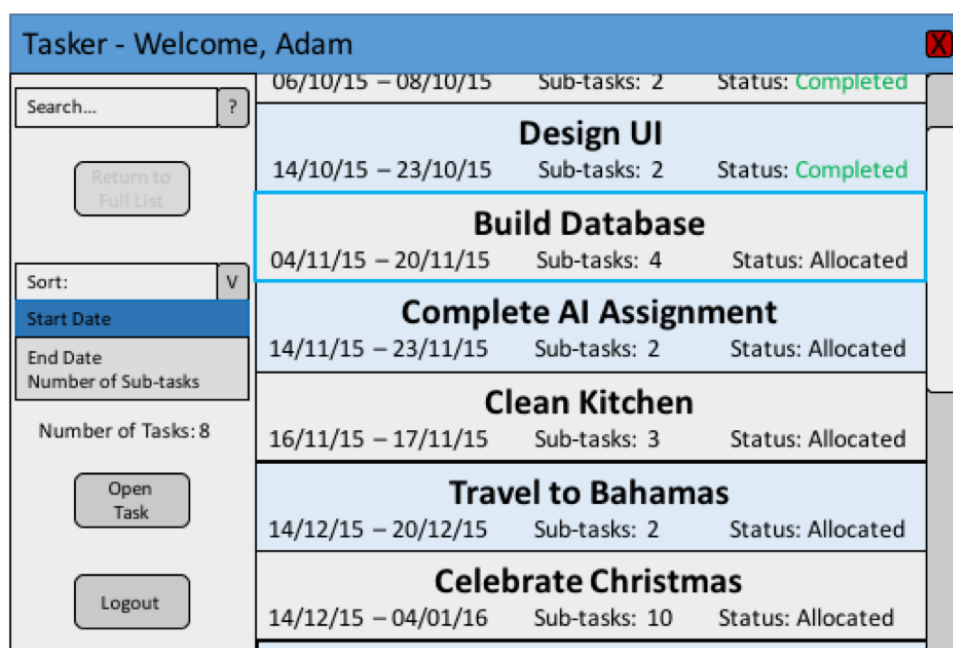
The login screen, the first screen to appear when the client loads. TaskerCLI does not synchronise with the database until the user has logged in as specified in Requirements Specifications.[2]



Tasker

Synchronising, please wait...

The synchronisation screen occurs between the login and main client page. As per the requirements specifications [2] 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.



Tasker - Welcome, Adam

Search... ?

Return to Full List

Sort: V

Start Date

End Date

Number of Sub-tasks

Number of Tasks: 8

Open Task

Logout

Task Name	Start Date	End Date	Sub-tasks	Status
Design UI	06/10/15	08/10/15	2	Completed
Build Database	14/10/15	23/10/15	2	Completed
Complete AI Assignment	04/11/15	20/11/15	4	Allocated
Clean Kitchen	14/11/15	23/11/15	2	Allocated
Travel to Bahamas	16/11/15	17/11/15	3	Allocated
Celebrate Christmas	14/12/15	20/12/15	2	Allocated
	14/12/15	04/01/16	10	Allocated

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 [2], 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.

The screenshot shows a window titled "Tasker - Build Database". It contains the following elements:

- Start Date:** 04/11/15
- Due Date:** 20/11/15
- Status:** Two radio buttons are present. The first is labeled "Allocated" and is selected (filled). The second is labeled "Completed" and is unselected (empty).
- Sub Tasks:** A table with two columns: "Description" and "Comments".

Description	Comments
Design tables	
Assign primary/foreign keys	
Assign relationships	
Design SQL queries	
- Return to List:** A button located at the bottom right of the window.

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 [2].

This screenshot is identical to the previous one, but with the following changes:

- Status:** The "Completed" radio button is now selected (filled), and the "Allocated" radio button is unselected (empty).
- Return to List:** The button remains at the bottom right.

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.

**Tasker - Welcome, Adam**

Design ?	06/10/15 – 08/10/15	Sub-tasks: 2	Status: <b>Completed</b>
<input type="button" value="Return to Full List"/>  Sort: V Start Date End Date Number of Sub-tasks  Number of Tasks: 7 <input type="button" value="Open Task"/>  <input type="button" value="Logout"/>	<b>Design UI</b>		
	14/10/15 – 23/10/15	Sub-tasks: 2	Status: <b>Completed</b>
	<b>Build Database</b>		
	04/11/15 – 20/11/15	Sub-tasks: 4	Status: <b>Completed</b>
	<b>Complete AI Assignment</b>		
	14/11/15 – 23/11/15	Sub-tasks: 2	Status: <b>Allocated</b>
<b>Travel to Bahamas</b>			
14/12/15 – 20/12/15	Sub-tasks: 2	Status: <b>Allocated</b>	
<b>Celebrate Christmas</b>			
14/12/15 – 04/01/16	Sub-tasks: 10	Status: <b>Allocated</b>	
<b>Take Exams</b>			
06/01/16 – 19/01/16	Sub-tasks: 2	Status: <b>Allocated</b>	

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.

**Tasker – Search: 'Design'**

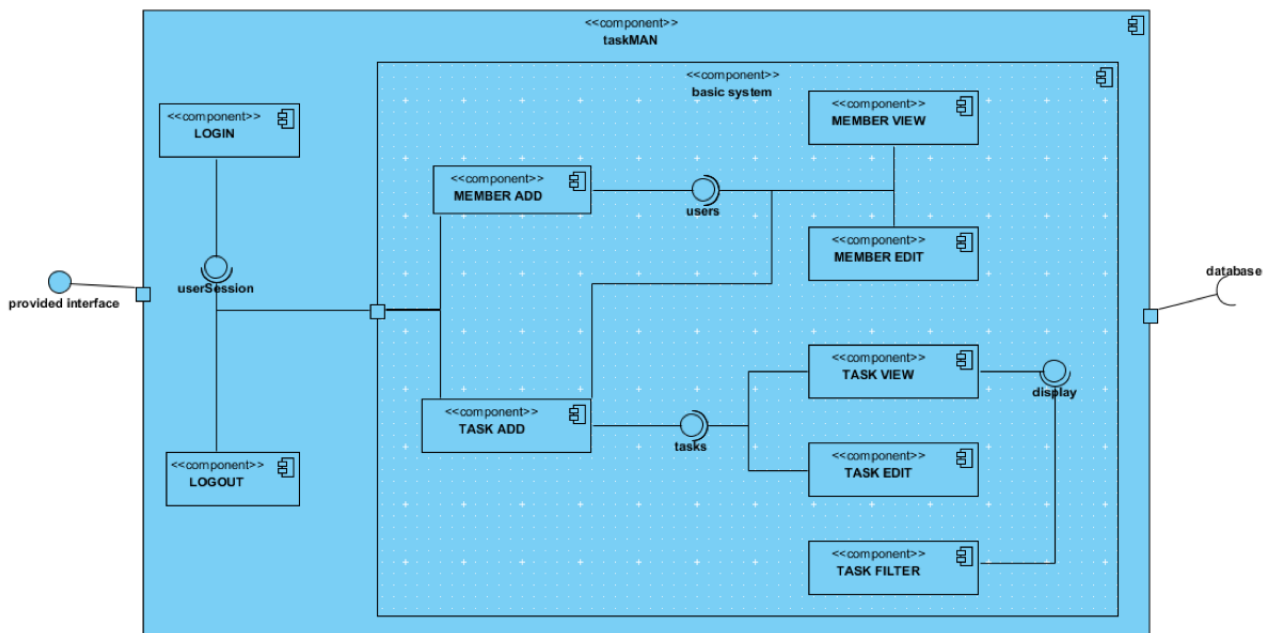
Design ?	<b>Design UI</b>		
<input type="button" value="Return to Full List"/>  Sort: V Start Date End Date Number of Sub-tasks  Number of Tasks: 7 <input type="button" value="Open Task"/>  <input type="button" value="Logout"/>	14/10/15 – 23/10/15	Sub-tasks: 2	Status: <b>Completed</b>
	<b>Build Database</b>		
	04/11/15 – 20/11/15	Sub-tasks: 3	Status: <b>Completed</b>

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 [2].

## 4 Component Description

### 4.1 TaskerMAN Component Description



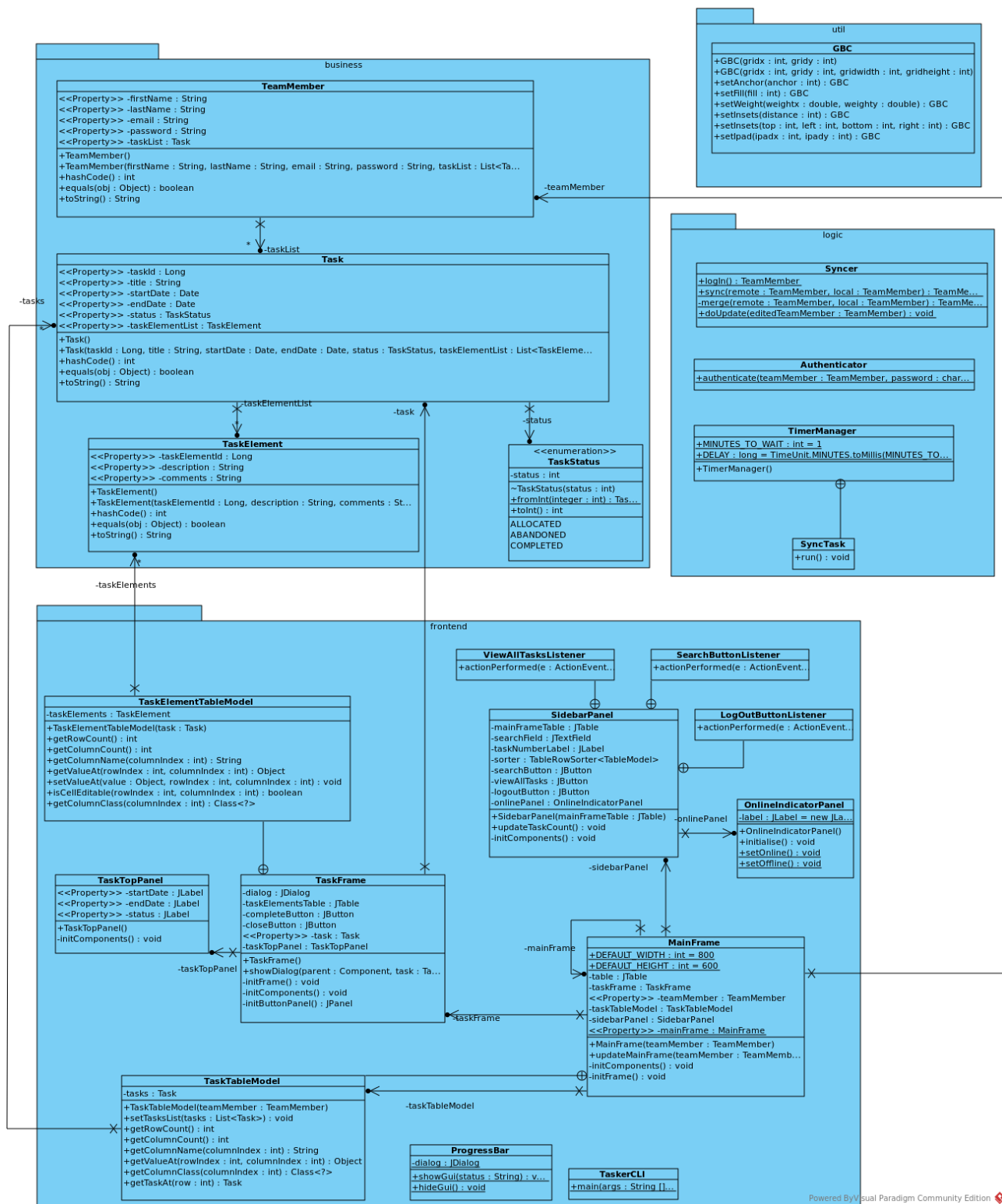
The component diagram above shows a further and advanced decomposed version unlike the previous diagram submitted before. As can be seen, two external interfaces are provided with one connecting to the database and the initial user interface which through a given port lets us use the login component 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.



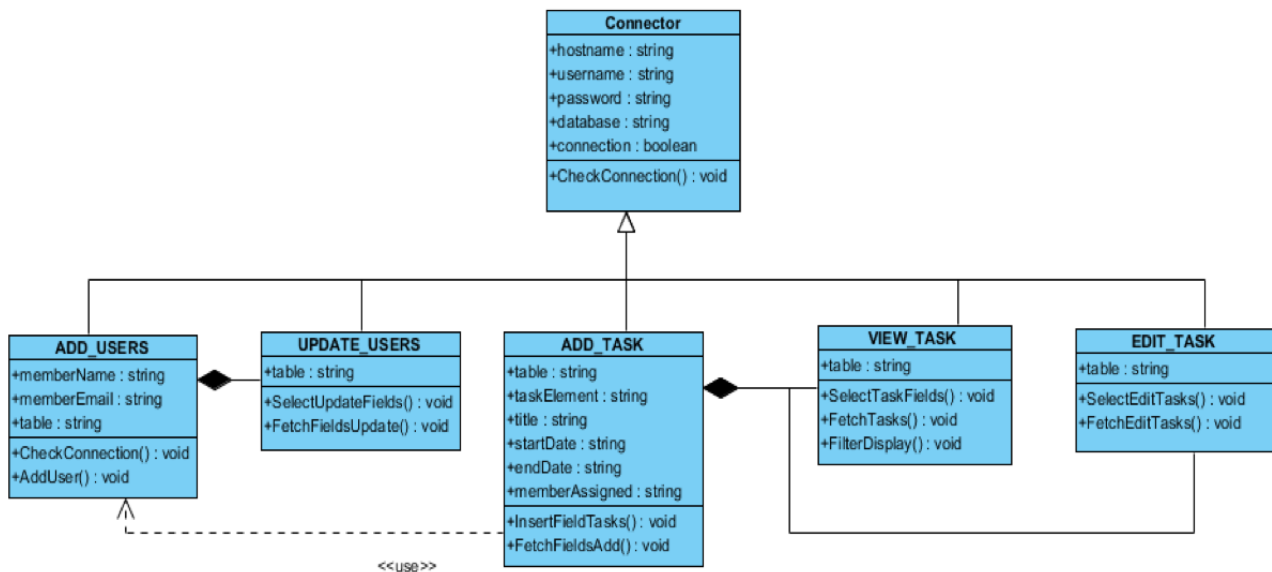
### 5.1.1 Description of each significant Class

- **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 `merge` method. The merge method takes two **TeamMember** objects, generates a canonical TeamMember object, which is supplied to the databases and UI; the merge method is called via the `sync` method, which loads the **progressBar**, and sets the status of the **onlineIndicator**. The Syncer class also contains the `login` method, displaying a dialog for user login, authenticaring using the **Authenticator** class and returning an appropriate **TeamMember** object.
- **TimerManager** Class used to invoke `sync` at regular intervals.
- **Authenticator** provides a static method to provide functionality to the `login` method to check passwords. Using BCrypt the passwords are hashed to keep them secure, and are encrypted in the databases.
- **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

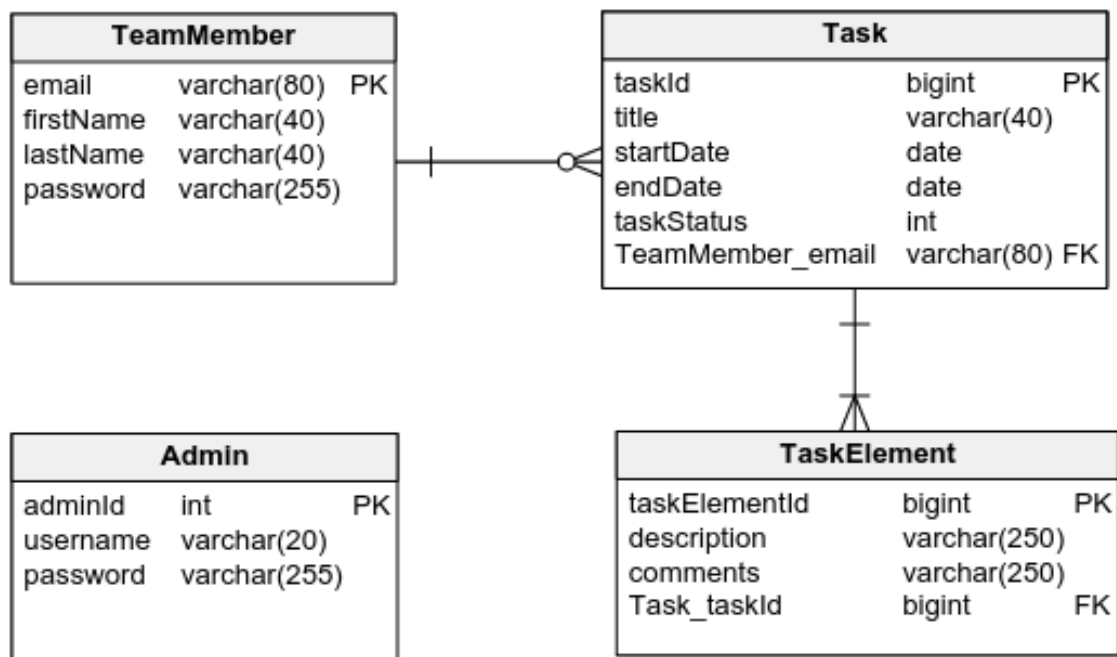
been updated to show all the users they want. To make sure users aren't accidentally deleted we should implement a pop up box to appear and make the user confirm that they want to delete a specific user.

- **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.

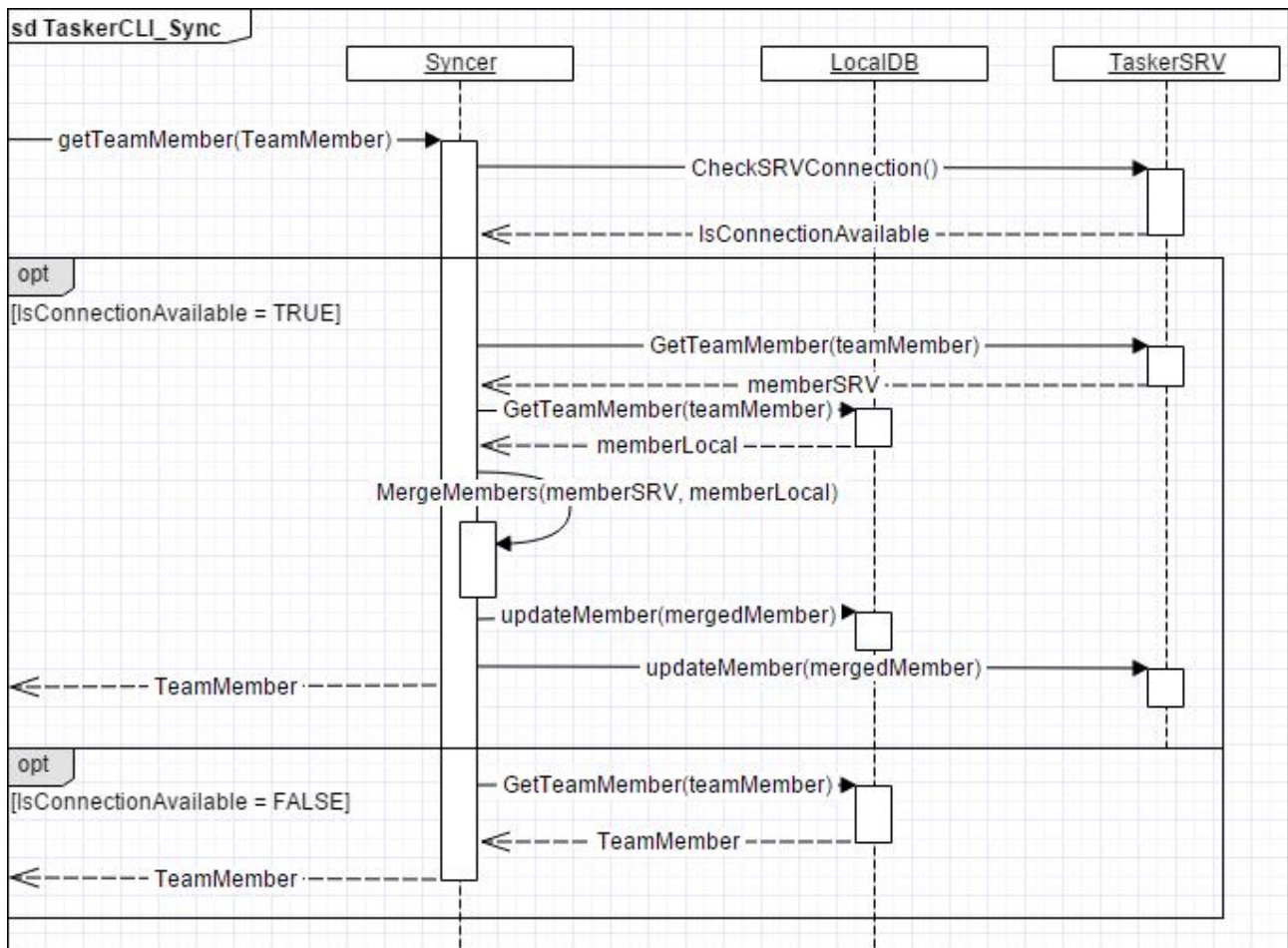
## 6 Detailed Design

### 6.1 TaskerSRV Entity-Relationship Diagram

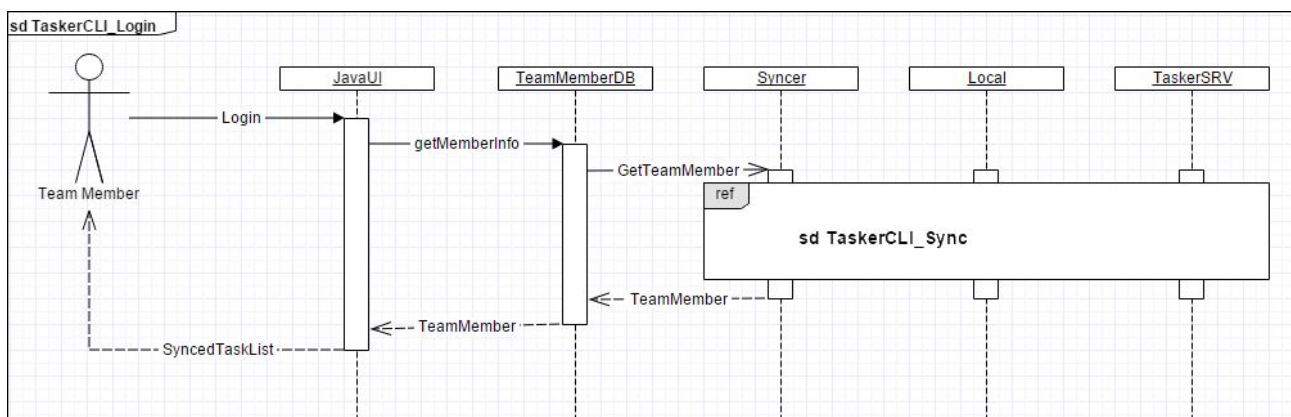


### 6.2 TaskerCLI Sequence Diagrams

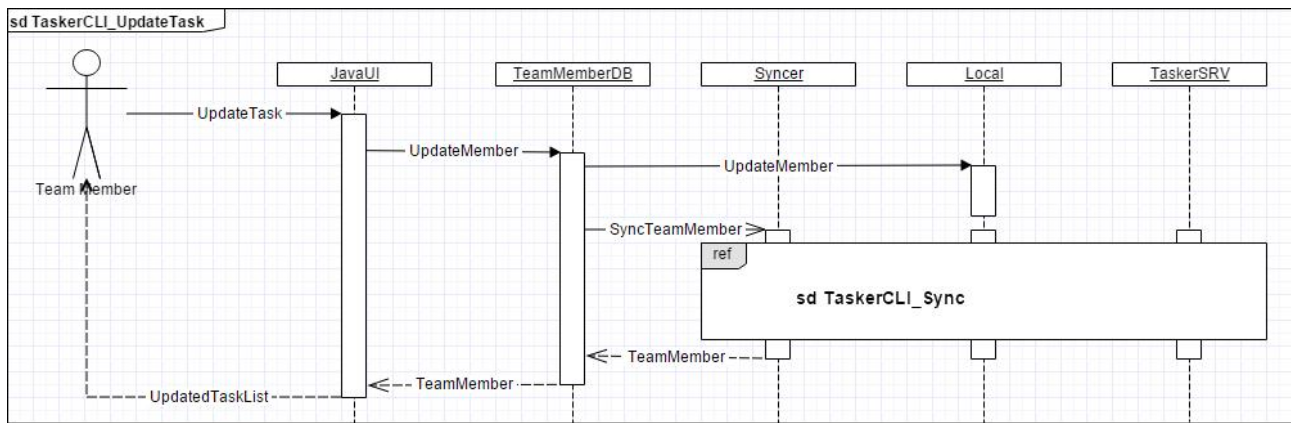
The following are the sequence diagrams describing the major actions taken by TaskerCLI, namely logging in to the system, updating a task, logging out, and the synchroniser syncing the changes made to the local and remote databases.



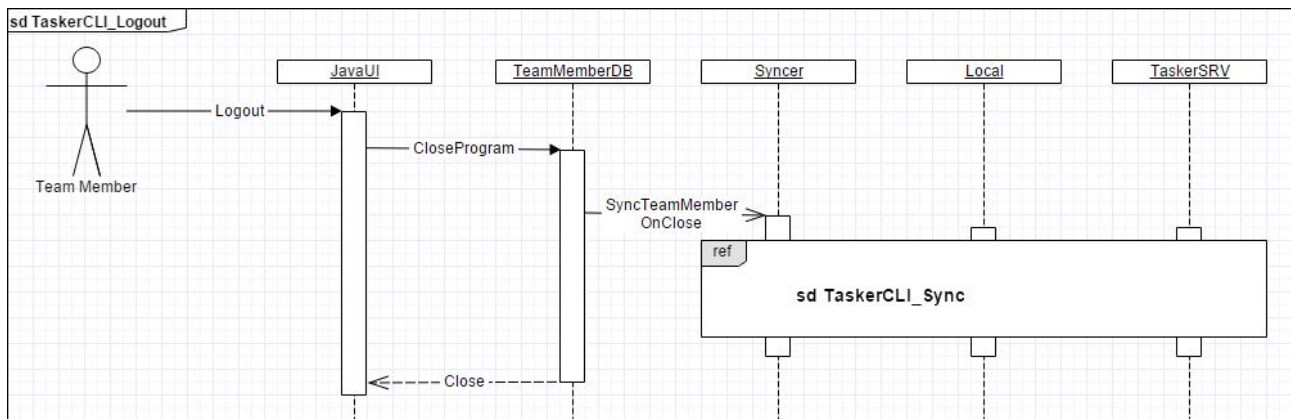
Above shows the sequence diagram for the action of syncing the two databases. It works by first checking if TaskerSRV can be connected to, then branches depending on the connections availability. If the connection is available, the syncer class gets a copy of the team member from both the local database, 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 team member 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.

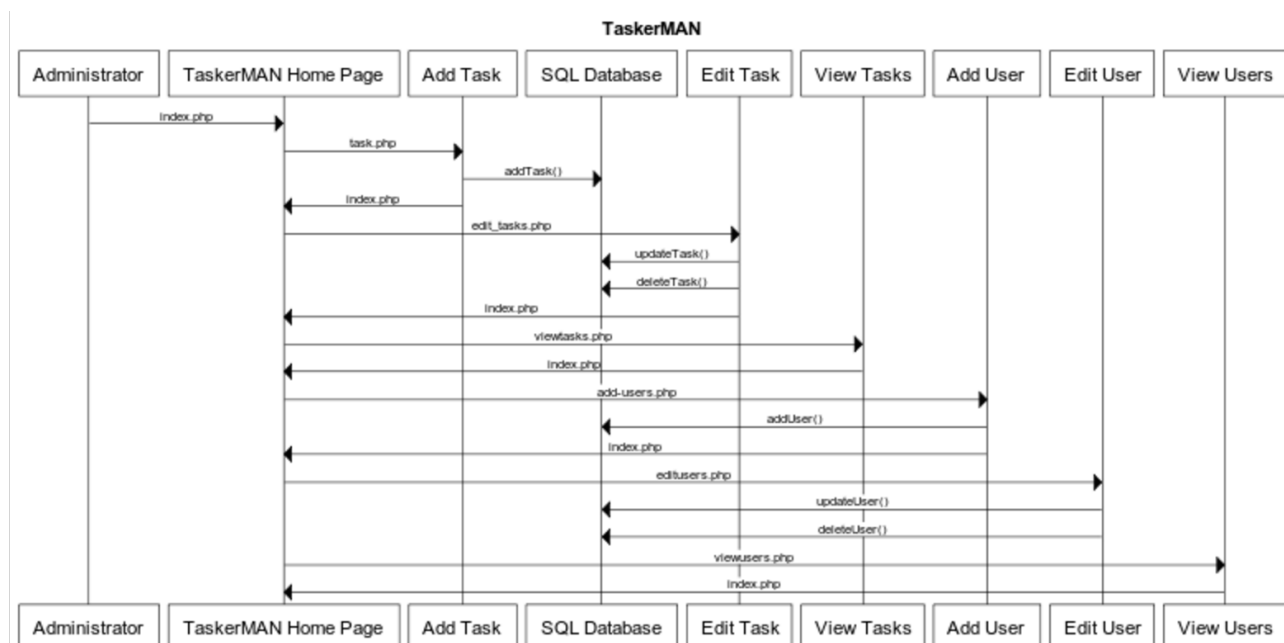


Above describes how TaskerCLI goes about closing when the user logs out. When the user logs out, the UI sends a close command to TeamMemberDB, telling it to perform a sync with the databases in order to save any last changes before shutting TaskerCLI. It requests a sync from the Syncer, which synchronises 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.

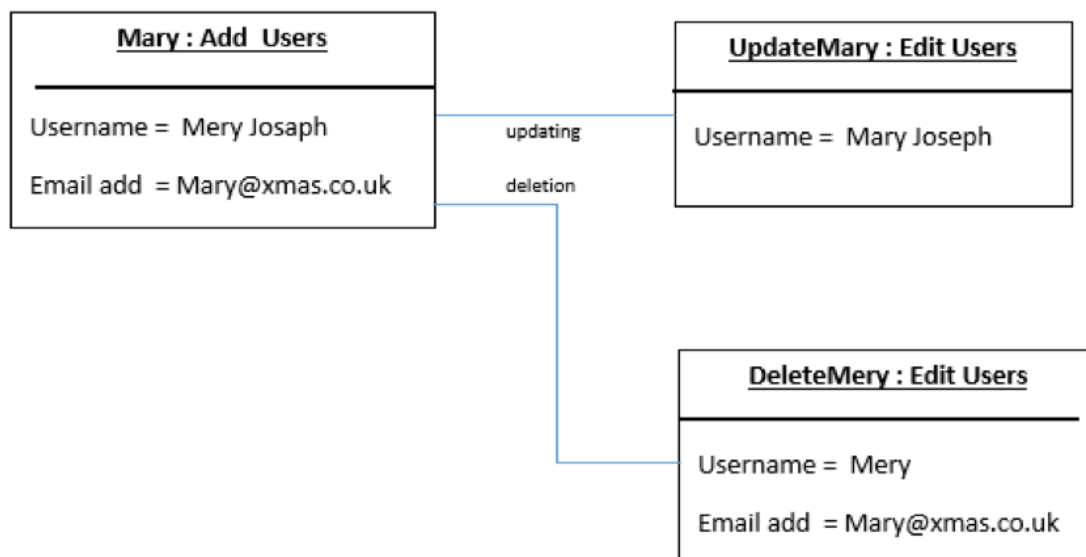
### 6.4 TaskerMAN Sequence Diagram



## 6.5 TaskerMAN Object Diagrams

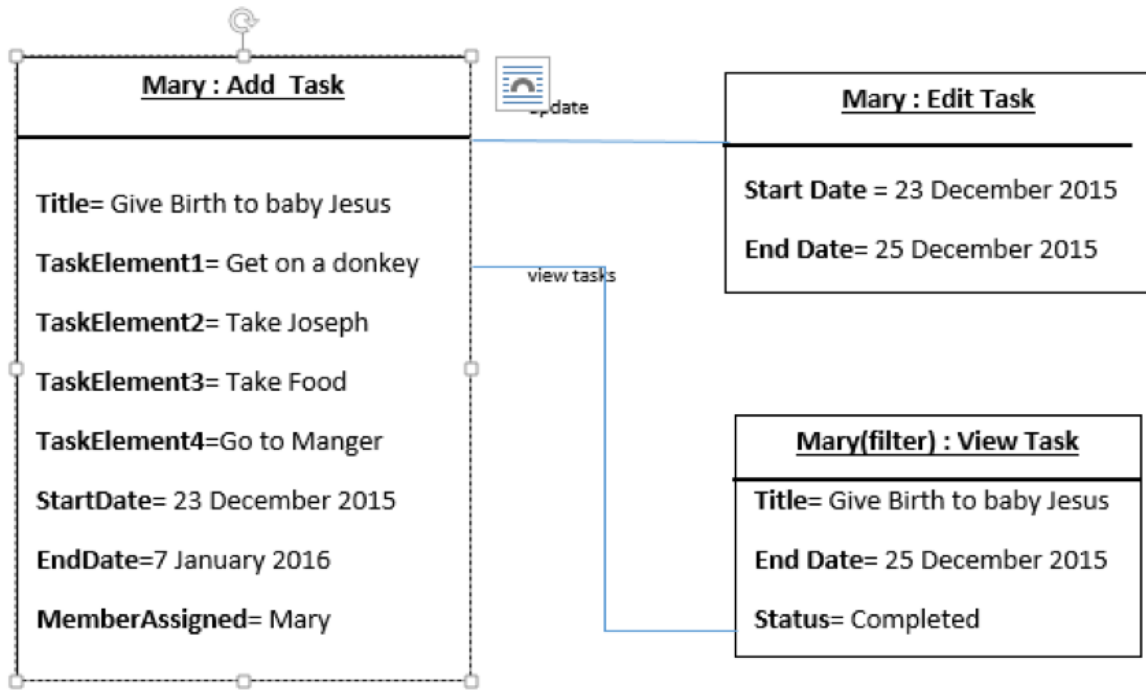
### 6.5.1 TaskerMAN Object Relationship between the ADD\_USERS and EDIT\_USERS CLASS

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, FR10. 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
1.8	N/A	2016-02-14	Updated Java significant classes based on implementation changes for release	J. Mir
1.9	N/A	2016-02-14	Final version, updated TaskerMAN UI and updated TaskerCLI significant classes diagrams	L. Jones