**PRCSA**

Philip Edwards, Richard Imms, Brian Viviers & Dominic Youel

April 2015

Plymouth University



PRCS205 Final Report

Contents

[1. Introduction 5](#_Toc417210514)

[The Scope of this Report 5](#_Toc417210515)

[LETS – A Description of the scenario 5](#_Toc417210516)

[2. Project Management 7](#_Toc417210517)

[Documentation of Problems 7](#_Toc417210518)

[The Initial Plan vs. Project Execution 7](#_Toc417210519)

[Comments on the Appropriateness of the Risk Analysis 8](#_Toc417210520)

[Critical Analysis of Project Conduct 8](#_Toc417210521)

[Project Management - *Pivotal Tracker* 8](#_Toc417210522)

[Version Control - *BitBucket* 9](#_Toc417210523)

[The SCRUM Framework 9](#_Toc417210524)

[3. Requirements Analysis and Design 11](#_Toc417210525)

[The Use Case Model 11](#_Toc417210526)

[User Stories with Acceptance Test 12](#_Toc417210527)

[User Requirements 20](#_Toc417210528)

[Use Case Diagrams 22](#_Toc417210529)

[Use Case 8: Create an Advert 23](#_Toc417210530)

[Use Case 17: Submit a Bid 24](#_Toc417210531)

[Use Case 20: Respond to a Bid 25](#_Toc417210532)

[Use Case 21: Complete Transaction 26](#_Toc417210533)

[Further Demonstration of Use Cases 27](#_Toc417210534)

[Create an Advert: Sequence Diagram (Web Application) 27](#_Toc417210535)

[Create an Advert: State Machine (Web Application) 28](#_Toc417210537)

[Create an Advert: Class Diagram (Web Application) 29](#_Toc417210539)

[Respond to a Bid: Sequence Diagram (Android Application) 30](#_Toc417210540)

[Respond to a Bid: State Machine (Android Application) 31](#_Toc417210542)

[Respond to a Bid: Class Diagram (Android Application) 32](#_Toc417210543)

[Evaluation of Design Changes in Implementation 33](#_Toc417210544)

[The Desktop Application 33](#_Toc417210545)

[The Web Application 35](#_Toc417210546)

[The Android Application 36](#_Toc417210547)

[4. Databases 39](#_Toc417210548)

[Database Evaluation 39](#_Toc417210549)

[Database Searching 39](#_Toc417210550)

[5. Security 40](#_Toc417210551)

[Web Application and Android 40](#_Toc417210552)

[Web Application 40](#_Toc417210553)

[The API 40](#_Toc417210554)

[6. Software Engineering 41](#_Toc417210555)

[The Desktop Application 41](#_Toc417210556)

[The Web Application 42](#_Toc417210557)

[The Android Application 44](#_Toc417210558)

[RESTful Web Services 46](#_Toc417210559)

[7. Appendix 47](#_Toc417210560)

[Data Model Class Diagram (Design Level) 48](#_Toc417210561)

[SQL Modeler Diagram 49](#_Toc417210562)

[SQL CREATE Statements 50](#_Toc417210563)

[Admin Access Table 50](#_Toc417210564)

[Advert Types 50](#_Toc417210565)

[Adverts 51](#_Toc417210566)

[Bids 54](#_Toc417210567)

[Categories 55](#_Toc417210568)

[Item Types 55](#_Toc417210569)

[Members 56](#_Toc417210570)

[Reviews 58](#_Toc417210571)

[Rules 59](#_Toc417210572)

[Sessions 60](#_Toc417210573)

[Transactions 61](#_Toc417210574)

[User Access 62](#_Toc417210575)

# Introduction

This section will first define the scope of the report, before forming a description of the project scenario. Any assumptions made will be detailed here and an explanation to indicate the structure of the system at a user level will be included. An updated list of team roles will be included at the end of this section.

## The Scope of this Report

This report forms an explanation of the systems delivered and includes a critical analyses and evaluation of the systems’ effectiveness. Shortcomings, strong points, extra or lacking functionality and changes made to the design are the main focuses of this report.

Since three applications have been submitted (web, android and desktop), it should be assumed that any discussion is a summarisation of the three systems. Any areas where distinctions need to be made between these applications, subheadings will be used to indicate which application is under scrutiny.

Conclusions will be drawn in each section where relevant and a final conclusion will form a summary of the project as a whole.

## LETS – A Description of the scenario

LETS, a Local Exchange Trading System, allows users to advertise products and services. It forms a locally initiated, member-driven community enterprise. An open policy between members is adopted and as part of the scheme, limited member information is viewable by all registered members. A record of all transactions made between members is an important part of the system. Currency is handled by adopting parity between LETS Credits and the local currency. The systems submitted have been designed to cater for the city of Plymouth and its immediate surroundings only.

Translating these requirements into a system required some definitions to be made. Firstly, since the system must support adverts either looking for or offering something, these will need distinction at both software and user levels. With this in mind, adverts can be labelled as a **request** or an **offer.** Next, the system must be able to handle adverts for **products** or **services**. In this way, advert creation requires the user to specify an advert type (offer or request) and an item type (product or service). These definitions will be referenced throughout.

Another aspect of the advertisement system references the **category** of the product or service being offered. It was decided that categories can be shared between services and products, forming a simple but powerful means to individualise adverts posted to the system. Additionally, this allowed default images to be added – an icon representing the category will be uploaded if the user selects no image for the advert.

Since transport is often an issue in local trading schemes, it has been defined that as part of the advert creation process, the member should indicate as to whether transport is included in their advert. This will be displayed on each advert with the text, “Transport is/is not included.”

Another more difficult problem to overcome is members indicating when they will be available to either deliver their products or carry out their services. This has been resolved by requesting members include a textual explanation of these details in their adverts or messages attached to bids. This was deemed the simplest, but perhaps not the most elegant way to resolve this issue.

The systems facilitating payments between members have seen a lot of development. Adopting an ESCROW-type arrangement, on acceptance of a bid, the full credits are taken from the paying member’s account and held by the system. When the paying member deems the work or items have been delivered, the credits are handed to the payee. This should satisfy both security requirements and reinforce the ideals of the LETS system – this should greatly reduce members violating their agreements and reinforce good member interaction.

Following the use cases demonstrated in section 3 forms a full explanation of the advert creation to transaction completion. The paying and reviewing member is handled differently for offers and requests; an explanation of the payer/payee arrangement can be seen in Figure 1.



Some final definitions and assumptions are as follows. Members can make **bids** on adverts, wherein they specify a time and/or place and submit this text to the advert holder. If the bid is accepted, a **transaction** is formed. At this point, product adverts are completed and no further bids can be made. Service adverts remain active and further bids can be made and accepted. Finally, when the paying member deems the advert’s terms are completed, he/she can finalise the transaction, leaving a textual review and a rating between one and five. The credits are taken from the ESCROW arrangement and credited to the payee’s account at this time, completing the trading process.

# Project Management

This section details the lifecycle of the project execution, primarily from the standpoint of the project manager. Some documentation of problems in the project, comparisons between plans and execution, comments on the risk analysis and thoughts on the SCRUM framework will be presented in this section.

## Documentation of Problems

In the opening weeks of the project, it became apparent that one group member had dropped out, leaving the team as a four. Fortunately, this happened quite early in the project lifecycle, allowing the team to adjust and ensure project deliverables’ delivery. The biggest problem here was a lack of communication from the absent group member – it was not communicated until many weeks later that he had in fact dropped out. Fortunately, since the group member never materialised at any stage of the project, it was simply a matter of delegating work as a group of four.

During the initial planning stage of the project we were required to create an initial planning document. While this document did provide help to the team, especially in the form of requirements and design, no feedback was made available. This caused some confusion between the team as there was no confirmation as to whether the project was going in the right direction. This was, however, later resolved in progress meetings with the tutors.

In relation to the organisation of the project, the team felt the cognitive walkthrough and usability studies were set too early into the project. Despite this, the usability studies provided feedback with which the applications could be improved. The group maintains the opinion that these sessions would have gleaned more useful findings towards the end of the project. The reasoning behind this is that for the first usability study, the team had to organise a false representation of an application to be tested. This meant the results gained from the study were useful in relation to the design but not in relation to a developed application.

## The Initial Plan vs. Project Execution

In following with the agile development process, the plan of work essentially formed a list of user requirements divided into a series of sprints. Pivotal Tracker (PT) was indispensable here; each requirement was written into a user story on PT, weighted with a score indicating the effort required and delegated to the team member(s) responsible for the work. These were then ordered according to the tasks’ priority.

It was planned that new requirements would be added each week, forming a balanced but manageable workload or sprintevery week. In practice though, this was not how the project played out. A group consisting of second-year computer science students, many other deadlines and commitments required servicing on a regular basis. A more realistic outlook for the team was simply *ticking off each requirement where possible*. In line with agile development, working builds were released on each iteration.

Fortunately, this is a dedicated group. Work was steady, but not fast enough to ensure completion at the average working week’s quota. A *crunch* in the closing two weeks was unavoidable. This was not a last minute affair by any means – the work carried out before the crunch was more than significant – but the workload in the final two weeks outstripped any other two week period by some margin. Estimates of group input are in excess of 100 hours per group member. It is worth noting that the primary tasks of this crunch period were focused on project documentation, code commenting and general bug fixing.

Some features planned never made it to the final system, but some requirements were added or others were more fleshed out. Requirements such as “View bids” for example resulted in multiple requirements including view bids by member and view bids on advert. Little details like this cement the fact that the group is still learning, but will be able to exhibit much more experience in future projects.

## Comments on the Appropriateness of the Risk Assessment

The initial risk analysis identified areas in which there was potential for risks to happen. This provided a scale as to how probable the situation was, the severity of the situation and the overall risk factor. Included in the same table were measures to be taken in the case of any issues ocurring.

Specifically, the loss of a team member was documented in the risk assessment. By following the actions set out in the risk assessment, appropriate measures were taken: contacting the module leader and delegating work to other team members. The loss in a team member also resulted in a reflection on the realistic amount of work producible.

In relation to the Android application, the team’s Android leader felt that the online help and documentation for Android Studio (Android IDE) was inadequate. The official Android documentation was out-dated in comparison to the IDE. A number of key methods and patterns were deprecated yet no new documentation was provided. Although the risk was identified in the risk analysis, the controls/action to be taken was not properly stated. Therefore this particular item of the risk analysis provided no guidance on a realistic approach to take.

The risk analysis provided in the initial planning document was appropriate in relation to the project overall. Identifying these risks enabled the team to act accordingly and allowed for early adjustments to be made to mitigate any issues that may have arisen.

## Critical Analysis of Project Conduct

### Project Management - *Pivotal Tracker*

The team made use of Pivotal Tracker as a tool for managing the project. Each user requirement was organised into stories which were added to the Project Backlog. These stories were then worked through in order of importance. Pivotal Tracker provided a visual representation of the work achieved alongside the applications, as well as a representation of the amount of outstanding work.

Once the applications had been mostly developed, and only desirable stories remained, the use of Pivotal Tracker reduced. This reduction happened as team members had other commitments to attend to, core functionality was being worked on and small adjustments were being made to fine tune the applications.

Overall the use of Pivotal Tracker has been beneficial to the team and the project. The ability to keep record of who owned which story enabled the team to easily communicate with each other in relation to specific functionality. This was extremely beneficial in producing related systems in terms of layout, design and functionality.

### Version Control - *BitBucket*

BitBucket was a useful tool throughout the project as it provided the team with a way of managing code and being notified of any changes that had happened. BitBucket also provided visibility of what work had been done by whom. The ability to review work on the BitBucket website and view any changes made through the commit logs enabled team members to comment on whether the code could be improved. As well as providing version control, BitBucket also provided a level of security. Storing code and documentation on a repository ensures the work is always backed up. Considering the possibility that a team member’s work could be corrupted or coding errors committed, a team member could just remove the erroneous code from their machine and replace it with the code from the repository. Although these scenarios never occurred in reality, it provided reassurance that a previous, working application was always available.

An issue with using BitBucket was a lack of knowledge with general repository usability. This was made apparent when two team members were seen working on the same lines of code. This required a manual merge to resolve any code clashes. Although this issue didn’t significantly impact the state of the project, it manifested in a cost of man-hours which could be better spent elsewhere. This could have been avoided through clearer communication. A valuable lesson has been learnt from this.

Another issue, although not greatly important, became apparent during pair-programming. Dependant on whose machine was being used meant that the BitBucket commits only showed commits from one user. On reflection, the team has noted that the names of the members working in this way should be noted in the commit logs.

Overall, BitBucket has been an invaluable tool to this project. The issues mentioned do not reflect poorly on BitBucket directly, but on some of the team’s level of experience with it. Throughout the project, the team has learnt a lot about the importance of version control and the clear advantages of using a version control system.

## The SCRUM Framework

The team members continued their roles in SCRUM throughout the project. During this time, daily communication was organised to ensure that the team as a whole was kept up to date with project progress. If a member wasn’t available for face to face meetings due to other commitments, another method of communication was organised. A group conversation on Outlook, text messaging and Pivotal Tracker story comments were used for this purpose.

For SCRUM meetings, minutes were recorded. These provided insight into the current progress of the project, significant changes and any issues requiring resolution. Any issues brought up during the meetings were noted and discussed. The discussions lead to the team either finding a solution to an issue or providing their own opinion. Throughout some meetings potential changes to the design of an application or new features to implement were also discussed.

Although every element of the SCRUM framework and the agile methodology were not implemented key aspects of it were. These key aspects allowed the team to work together dynamically. During the initial planning stage this allowed us to organise better as a team. As the project went on other university commitments, deadlines and the Easter break broke up the fluid style of Agile. The team felt that Agile and SCRUM would be better suited to a project in industry rather than one at university. The team also feel that the lessons learnt from adopting a semi-agile process and the tutorials on the subject have improved our understanding of the methodologies available when working on a project.

## Final Team Roles

**Philip Edwards**

* Android Application Lead Developer
* Desktop Developer
* Application Tester
* SCRUM Team Member

**Richard Imms**

* Desktop Application Lead Developer
* Database Lead Developer
* Android Developer
* Application Tester
* Project Manager
* SCRUM Master

**Nathaniel Ovington**

* Absent throughout the project – no roles delegated

**Brian Viviers**

* Web Application Lead Developer
* Security Manager
* API & Software Engineering Lead Developer
* Application Tester
* Product Owner
* SCRUM Team Member

**Dominic Youel**

* Requirements Analysis and Design Leader
* HCI Manager
* Documentation Manager
* Web Developer
* Application Tester
* SCRUM Team Member

# Requirements Analysis and Design

This section of the report will detail the requirements analysis and design at a design level. Use case diagrams will be presented first, showing the system as a whole, before looking in more detail at requirements and the most significant use cases identified. Finally, an evaluation of design changes made during implementation will be discussed separately for the three applications.

## The Use Case Model



## User Stories with Acceptance Test

The user stories below are numbered consistently to match the use cases and requirements list throughout section 3.

|  |  |  |
| --- | --- | --- |
| Use Case | **User Story** | **Acceptance Tests** |
| 1 | As a guest, I should be able to register to the system so I can create and bid on adverts | (i) When clicking the “Register” button, I should be presented with a form containing all member data as defined in the “Members” table.  (ii) This form should present me with an error if I omit any information required to register.  (iii) Upon registration, I should be taken to the login page and upon entering my email and password, I should be able to log in. |
| 2 | As a registered member or admin, I should be able to log into the system. Once logged in, I should be able to log out, too. | (i) Upon clicking the “Login” button, I should be presented with an email and password box.  (ii) If my details are incorrect, I should be presented with an error message.  (iii) If my details are correct, I should see confirmation and be redirected to the homepage or dashboard panel.  (iv) Upon clicking the “Logout” button, I should see confirmation and be redirected to the guest homepage or dashboard panel. |
| 3 | As a member (or an admin), I should be able to see my account details held in the system. Admins should be able to access members in this way. | (i) Upon clicking the “My Account” button, I should be presented with my account information.  (ii) Guests should not be able to see this information and further, members should only be able to see their own account information.  (iii) Upon selecting a member and navigating to “Account,” admins should be presented with said member’s account details. |
| 4 | As a member (or an admin), I should be able to update or edit my account details. This includes my login information – email and password. | (i) While viewing my account information (see use case 3), it should be possible to modify any and all information held in the system.  (ii) Most importantly, I should be able to edit my email and password fields.  (iii) As with registration, any incomplete fields should result in an error message and no database alterations made until these are resolved.  (iv) Admins should be able to edit details held on a member  (v) Guests must not have access to this and members must only have access to their own information. |
| 5 | As a logged in member, I should be able to access a list of all members and see their number of credits and average review. Admins should see all account information. | (i) Upon selecting “Find members,” I should be presented with a list of members’ names. Their number of credits and average review score should be shown here.  (ii) No further personal information should be shown to members.  (iii) The currently logged in member should not be included in this list.  (iv) Admins should be able to see further account information.  (v) Guests must not be able to access this information.  (vi) Links to members’ information should be made accessible when viewing adverts. A link to view the member should be visible on these screens. |
| 6 | Upon selection of a member from use case 5, I should be able to see a list of this member’s reviews. | (i) Upon selection of a member, I should see a list of reviews, including the score out of five and the review text.  (ii) I should also see the member’s balance and average review score on this screen.  (iii) Admins will access this information differently – through selection of a transaction.  (iv) Guests must not be able to access this information. |
| 7 | When viewing the list of members in use case 5, I should be able to search this list to more easily find a member by name. | (i) When viewing the member list, I should be able to type a forename, surname or both and be presented with a list of members matching said query.  (ii) Non-matching members must not be shown in the list after a query is entered.  (iii) I should be able to reset this filter and again see a full list of members.  (iv) If no members are found, I should see an empty list and/or an error message.  (v) Guests must not be able to access this information. |
| 8 | As a member, I should be able to post adverts to the system to offer or request products or services. | (i) When clicking “Create Advert,” I should be presented with method(s) to enter all necessary advert information before being offered a chance to edit, discard or publish the advert.  (ii) The web application should offer an explanatory method for new users and a streamlined method for returning users.  (iii) Any omission of advert data should be made impossible – each piece of data added should “unlock” the next field.  (iv) Any omitted data (if the member erases a field, for example) should result in an error message and no table updates.  (v) If no image is added, a default image indicating the advert item’s category should be uploaded to the system. |
| 9 | As an admin, I should be able to edit an already uploaded advert. | (i) Upon selection of a member, I should be able to select and edit any of the fields in one of their adverts.  (ii) Any incomplete or omitted fields should result in an error message and no table updates being made.  (iii) When updates are made, these should be immediately visible. |
| 10 | As a member holding an active advert, I should be able to delete said advert from the system. Admins should hold the ability to delete any active advert. | (i) I should be asked to confirm before deleting the advert.  (ii) Upon the advert’s deletion, I should see confirmation and no longer be able to access the advert.  (iii) If the advert has already expired, no option to delete the advert should be made available.  (iv) Members should only possess the ability to delete their own advert. |
| 11 | As a member, I should be able to view any adverts I do not own. Admins should be able to view any advert. Guests should be able to view adverts, but be presented with no methods to bid or view members. | (i) All advert details must be shown on this page, including the image.  (ii) Functionality on the view advert page for members should include “Bid” and “View Member.”  (iii) The functionality should differ from Use Case 15 – a distinction must be made between “View Advert” and “View own Advert.”  (iv) Guests must be able to view adverts but have no means to make bids or view members.  (v) Admins should see the advert’s details, but have no means to make bids. |
| 12 | As a guest, member or admin, I should be able to browse a list of all currently active adverts so I can only make bids on active adverts. | (i) Logged in members must not be able to see their own adverts in this list.  (ii) Basic advert information should be shown, including the advert image, title, number of credits, the advert/item type and expiry date.  (iii) If no current adverts are available, an empty list with some explanatory text should be shown. |
| 13 | While browsing adverts, members, guests and admins should be able to search for an advert by a query string. | (i) Upon entering a query, the list of adverts should only show relevant or matching results.  (ii) If no matches are found, explanatory text should be shown with an empty list.  (iii) This filter must have the ability to be reset, again showing the full list of adverts.  (iv) Members must not be able to access their own adverts in this way.  (v) Only active adverts should be displayed in this list. |
| 14 | As an admin, I should be able to filter adverts by active or completed. | (i) Upon selection of the filter, the list of adverts should be updated, only showing relevant matches.  (ii) If no matching adverts are found, explanatory text should be shown with an empty list.  (iii) This filter must have the ability to be reset, again showing the full list of adverts. |
| 15 | As a logged in member, I should be able to view my own active adverts, allowing the extra functionality, “Delete Advert.” | (i) A member’s adverts should be accessible from “My Adverts.” This list should display active and expired adverts.  (ii) It should be made clear to the member that the advert being viewed is their own.  (iii) There must be no means to make a bid or view the member on this screen, as in Use Case 11. Distinctions must be made between viewing an advert and viewing your own advert.  (iv) If the advert has expired, it should be made clear to the user. |
| 16 | As a member, I should be able to see my own completed or expired adverts. No controls to delete the advert should be made available. | (i) When viewing my completed adverts, there must be no means to delete the advert.  (ii) It must be made clear to the user that the advert has expired.  (iii) Admins must be able to view completed adverts, but members must not be able to see these, unless they are the advert owner. |
| 17 | As a member viewing an active advert, I should be able to submit a bid, to begin the trade between two members. | (i) Guests must not be offered the ability to bid on an advert.  (ii) When submitting a bid, the member should be given the ability to insert a (mandatory) message with it.  (iii) Members should be notified that this message is mandatory, and should include information as to the day/time any contact between members can be carried out.  (iv) Failure to include a textual message in the bid should result in an error message and no bid being sent.  (v) Multiple bids can be placed on one advert to allow the offer of different days to trade, for example. |
| 18 | As a member, I should be able to see all open bids on the same screen as one of my adverts. Admins should be able to select an advert and see all bids on it. | (i) Logged in members should be able to navigate to their own advert and view a list of open bids on the same screen. The ability to accept or reject these bids should be offered here.  (ii) If no bids have been made, an empty list should be shown with some explanatory text.  (iii) Admins should see active bids on any active adverts.  (iv) Completed adverts should display no bids, since no further action is required from the member here. |
| 19 | As a member, I should be able to see a categorised list of bids, including active/completed incoming (selling/sold) and active/completed outgoing (buying/bought). | (i) When navigating to the “My Bids” screen, the member should be presented with a series of selectable lists, detailing the four types of bid.  (ii) Any list with no entries should show an empty list with some explanatory text.  (iii) Bids where action is required for the user should include controls to respond.  (iv) Bids which require action from the advert holder (not the logged in member) should include explanatory text as to the status of the bid – “Awaiting response from advert holder” for example. |
| 20 | As a member with an active bid on one of my adverts, I should be able to accept or reject this bid and give a (required) textual response. | (i) This can either occur from the member’s advert screen or the member’s “My Bids” screen.  (ii) Upon clicking accept or reject bid, I should be able to attach a (required) textual response, indicating which day and time the transaction can take place.  (iii) Upon the acceptance of a bid, the credits will be taken from the paying member’s account and held in an ESCROW agreement, until the work or product transfer is deemed complete.  (iv) On accepting a bid, the member’s email and phone number will be appended to the textual response.  (v) The paying member should be notified that their balance has been reduced and the bid accepted.  (vi) Upon rejection of a bid, the buyer is free to make further bids on the same advert.  (vii) Failure to include a textual response should result in an error message and no bid submission.  (viii) A completed bid will place an active transaction in both members’ accounts. |
| 21 | As a member with an accepted bid, I should be able to finalise the transaction when it is deemed the advert has been satisfied for both parties. The payer is always in control of this part. Leaving a numeric and textual review is part of this process and is mandatory. | (i) This functionality should be made available in the “My Transactions” section of the system.  (ii) When clicking “Pay Credits and Submit Review,” the member will be presented with a five-star rating system and a textual review box.  (iii) The numeric star rating will default to three stars and any selection less than 1 or greater than 5 will be impossible.  (iv) Failure to fill out the textual review will result in an error message and no transaction submission.  (v) A valid transaction submission will credit the payee’s account with the credits held in the ESCROW arrangement.  (vi) Both members can access this completed transaction in the “My Transactions” section. |
| 22 | As a member, I should be able to see all my active or completed transactions in the “My Transactions” section. Admins should be able to access these for any member. Transactions are created when a bid is accepted and allow the advert process to complete. | (i) Transactions should be categorised in the same way as bids with 4 selectable lists – incoming active/completed and outgoing active/completed.  (ii) This screen will offer controls to complete transactions by transferring credits and submitting a review.  (iii) Members with no transactions should see an empty list and explanatory text.  (iv) Transactions requiring attention from the logged in member should have controls available to pay credits/submit review.  (v) Transactions requiring attention from other members should be given explanatory text, such as “Awaiting finalisation from the buyer.”  (vi) Members should only be able to access their own transactions. |
| 23 | A member or an admin should be able to see the review attributed to a particular transaction. By selecting a completed transaction, the attributed review value and text should be shown. | (i) When selecting a completed transaction from the “My Transactions” section, the attributed review should be shown, displaying the review value and text.  (ii) Members should only be able to see reviews from their own transactions – access to other member’s transactions is not to be offered. |
| 24 | (Web only) When logging in and viewing the member’s homepage, a list of notifications should be displayed to show whether bids or transactions require action. | (i) These notifications should be categorised in the same way as bids or transactions – only bids or transactions requiring attention from the member should be shown.  (ii) Clicking the notification should direct the member to their bids/transactions page.  (iii) A lack of bids/transactions simply results in no notifications being shown on the member homepage. No explanatory text is required.  (iv) Differing notifications should be easily distinguishable – use of colour and headings should be employed. |
| 25 | Members, guests and admins should be able to view a list of rules, which govern the LETS scheme. | (i) Held in its own section, this is accessible by clicking the “Rules” link or button.  (ii) Forming a simple list, this should always load the most up to date rules.  (iii) If no rules are available, a link to the help page should be displayed instead. |
| 26 | (Web only) Guests and members should be able to view a help page, describing the LETS scheme and how it works. | (i) This should be navigated to by clicking the “Help” button available on all navigation bars.  (ii) The function to contact an admin should be offered on this screen to both guests and members. |
| 27 | (Admin only) Admins should be able to add, edit and remove rules from the system to inform members of new policies or amendments. | (i) The addition of an empty string for a rule should result in an error message and no addition or edit to a rule.  (ii) The ability to remove rules should be handled by a button next to any given rule.  (iii) The removal of all rules should see a notification shown to the admin. The rules list should not be made empty without informing the admin. |
| 28 | (Web only) Guests and members should be able to contact an admin via email to seek help where required. | (i) Accessed from the help page, clicking “Contact Admin” results in a text box for an email message.  (ii) If the member is logged in, their contact details will be appended to the email. Guests will be asked to append their email to allow a response.  (iii) Failure to enter any text into the email field will result in an error message and no email being sent. |

## System Requirements

Since a number of applications are to be developed, a number of system requirements should be included. These will not be discussed in great detail, however these requirements can and do offer extra functionality and usability. Please note that these requirements are all offered in the relevant systems.

* (Web and Android) RESTful web services should be adopted to follow current good practice. See section 6.X for a further discussion of this.
* (Web and Android) Iconography should be present on all controls possible, making use of the same icons across both applications. For consistency, [www.flaticon.com](http://www.flaticon.com) should be the source of all iconography used. This is a free service.
* (Web only) The user interface should resize to allow for different screen sizes.
* (Android only) The screen should only offer portrait orientation, catering to phone or tablet screens of all sizes.
* (Android only) The build should target Android 5, which forms the most current system software.
* (Desktop only) Java 8 should be targeted for development since it is the most current available.
* (Desktop only) The observer pattern should be implemented to ensure the interface is always displaying the most current data. See section 6.X for a further discussion of this.
* (Desktop only) The database connection must automatically refresh every 2 minutes to ensure any changes are reflected within a reasonable time-frame. The means to manually refresh the database connection should be offered also. See section 6.X for further discussion.

## User Requirements

The user requirements are detailed below, with the final three columns indicating the functionality offered in each system’s final build. The *pass* or *fail* mark has been accredited by referencing the acceptance tests outlined in section 3.X. Note that a failure doesn’t necessarily indicate missing features – some requirements are application specific by design.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | **Requirement** | **Use Case** | **Web** | **Android** | **Desktop** |
| 1 | To allow guests to register for the service on web or android | Register | ✓ | ✓ | ✗ |
| 2 | Members should be able to log into the applications | Log in/out | ✓ | ✓ | ✓ |
| 3 | A member’s account details should be viewable | View Account | ✓ | ✓ | ✓ |
| 4 | A member’s account details should have the ability to be updated or edited | Update Account | ✓ | ✓ | ✓ |
| 5 | A list of members should be made visible, and their details shown. | View Members | ✓ | ✗ | ✓ |
| 6 | A list of reviews given to a member should be made visible | View Members’ Reviews | ✓ | ✗ | ✗ |
| 7 | The ability to search for a member by forename, surname or both | Search for Members | ✓ | ✗ | ✓ |
| 8 | Adverts should be able to be added from either web or android applications | Create Advert | ✓ | ✓ | ✗ |
| 9 | An already existing advert should be able to be edited by an admin | Edit Advert | ✗ | ✗ | ✓ |
| 10 | Adverts should have the option to be deleted from the advert owner or an admin | Delete Advert | ✓ | ✗ | ✓ |
| 11 | Adverts should be viewable by guests, members and admins | View an Advert | ✓ | ✓ | ✓ |
| 12 | Adverts should be viewable in a list and made available to all users | Browse Adverts | ✓ | ✓ | ✓ |
| 13 | Adverts should have the ability to be searched by a query string | Search Adverts | ✓ | ✓ | ✓ |
| 14 | Adverts should allow filtering by active or completed in the advert list. (Admin only) | Filter Adverts | ✗ | ✗ | ✓ |
| 15 | Members should be able to view their own advert to allow extra functionality | View own Advert | ✓ | ✓ | ✗ |
| 16 | Members should be able to view their completed adverts | View Completed Adverts | ✓ | ✗ | ✓ |
| 17 | Members must be able to submit bids against active adverts held by other members | Submit a bid | ✓ | ✓ | ✗ |
| 18 | Members and admins must be able to see all active bids on an advert | View bid on advert | ✓ | ✗ | ✓ |
| 19 | Members must be able to see a list of all active bids made and received | View member bids | ✓ | ✓ | ✗ |
| 20 | Members must be able to confirm or reject an incoming bid | Respond to bid | ✓ | ✓ | ✗ |
| 21 | When the trade between two members is completed, the paying member must be able to complete the transaction and submit a numeric and textual review of the transaction. | Complete Transaction | ✓ | ✓ | ✗ |
| 22 | Members and admins must be able to see all transactions past and present. | View transactions | ✓ | ✓ | ✓ |
| 23 | A member or admin should be able to see the review given for a selected transaction | View transaction review | ✗ | ✓ | ✓ |
| 24 | When logging in (web only), the member should be presented with updates as to bids and transactions requiring attention | View advert notification | ✓ | ✗ | ✗ |
| 25 | Members, guests and admins should be able to see the list of rules governing the LETS system | View rules | ✓ | ✓ | ✓ |
| 26 | Guests and members (web only) should be able to navigate to a help page | View help | ✓ | ✗ | ✗ |
| 27 | Admins should be able to add and remove rules to the existing list of rules. | Add rules | ✗ | ✗ | ✓ |
| 28 | Users on the web application should be able to contact an admin via email. | Contact admin | ✓ | ✗ | ✗ |

## Use Case Diagrams

Not all use cases identified will be presented with a use case diagram. Use cases such as “View help” and “View rules” are self-explanatory and little would be gained from the inclusion of a diagram. Only the following use cases which could use some further explanation will be demonstrated:

* Use Case 8: Create an Advert
* Use Case 17: Submit a Bid
* Use Case 20: Respond to a Bid
* Use Case 21: Complete Transaction

These four use cases essentially describe the system and its most relevant interactions, encapsulating the entire buying or selling process.

Each use will be presented with a textual description of the scenario, a use case description and finally, a use case diagram.

### Use Case 8: Create an Advert

Available to members only, members can upload an advert with an image. To be specified by the member is the following:

* Advert Type (Offer or Request)
* Item Type (Product or Service)
* The item category (Accounting, Cleaning, etc.)
* Whether transport is included (Yes or No)
* Number of Credits requested/offered
* Advert Title
* Advert Description (including indication of days available)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Member Actions** |  | **System Response** |
| 1 | Select “Create an Advert” | **2** | Request advert information |
| 3 | Enter advert information | **4** | Request image addition |
| 5 | Upload an image or select “Default Image” | **6** | Show “Preview Advert” button |
| 7 | Select “Preview Advert” | **8** | Present the advert preview with discard, edit and publish advert controls |
| 9 | Member selects “Publish Advert” | **10** | Confirm advert publish |

**Alternate flows**

11a. The user selects “Discard Advert”: Member is returned to homepage

11b. The user selects “Edit Advert”: Member is returned to **5** with all fields filled in, ready for edit.



### Use Case 17: Submit a Bid

If a member has found an advert they wish to make a bid on, they can submit a bid with textual information indicating which day(s) they are available to carry out the trade.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Member Actions** |  | **System Response** |
| 1 | Navigates to an advert | **2** | Advert presented with bid and member controls |
| 3 | Select “Make a Bid” | **4** | Dialog to add a textual message indicating days available |
| 5 | Add textual information and submit bid | **6** | Confirmation of Bid |



### Use Case 20: Respond to a Bid

When an advert holder receives a bid, they can either accept or reject it. A textual response is required, detailing which proposed day(s) and times are acceptable. The member’s email and phone number are appended to this message to facilitate any further communication if necessary. If the bid is accepted, credits will be removed from the paying member’s account in an ESCROW arrangement.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Advert Holder Actions** |  | **System Response** |
| **1** | Navigate to “My Bids” | **2** | Present “My bids” screen |
| **3** | Select the bid requiring attention | **4** | Present “Accept” or “Reject Bid” buttons |
| **5** | Select “Accept” or “Reject Bid” | **6** | Dialog to enter a textual message with bid. The member’s contact details (email and phone no.) are appended to this message. |
| **7** | Add textual information and respond to bid | **8** | Confirmation and, if accepted, removal of credits from payer’s account to ESCROW |

**Alternate Flows**

3a. Click active bids notification from homepage: present “My Bids” screen



### Use Case 21: Complete Transaction

When the paying member deems the trade to be completed, they can complete the transaction. Two stages are required and one cannot exist without the other. The paying member rates the transaction out of five and writes a textual review. Finally, credits are taken from the ESCROW arrangement and credited to the payee’s account.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Paying Member Actions** |  | **System Response** |
| **1** | Navigate to “My Transactions” | **2** | Present “My Transactions” screen |
| **3** | Select the transaction requiring attention | **4** | Present “Pay Credits and Transfer Credits” button |
| **5** | Click “Pay Credits and Transfer Credits” button | **6** | Dialog to enter a review value (1 to 5 stars) and a textual review. |
| **7** | Enter review details and submit | **8** | Confirmation; credits are transferred from ESCROW to payee member. |

**Alternate Flows**

3a. Click active transactions notification from homepage: present “My Transactions” screen



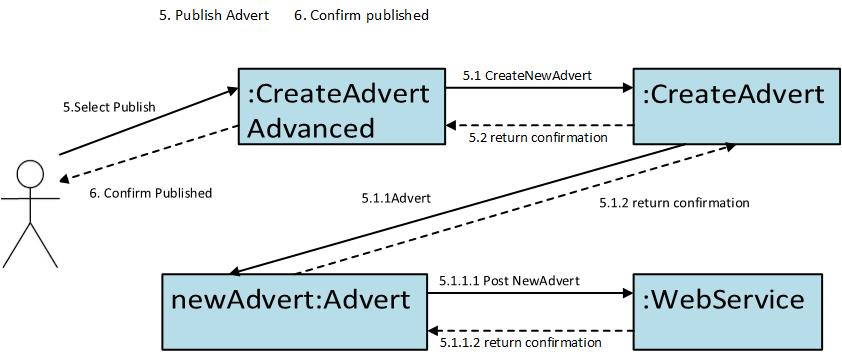
## Further Demonstration of Use Cases

Two use cases have been selected for further demonstration with a design level sequence diagram, a state machine and a class diagram. The use cases selected to demonstrate are as follows:

* Use Case 8: Create an Advert
* Use Case 20: Respond to a Bid

### Create an Advert: Sequence Diagram (Web Application)

### 



### Create an Advert: State Machine (Web Application)

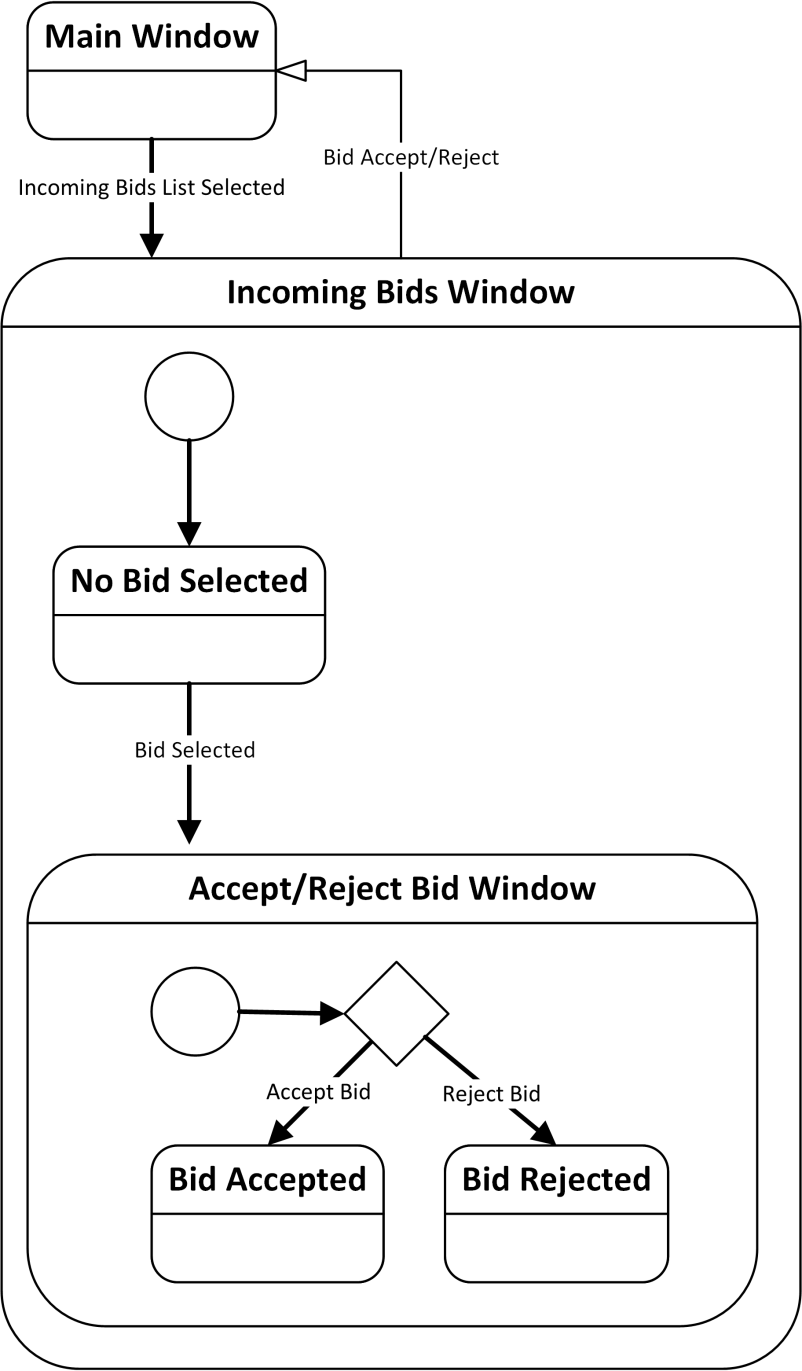
### 

### Create an Advert: Class Diagram (Web Application)

### Respond to a Bid: Sequence Diagram (Android Application)

## 

### Respond to a Bid: State Machine (Android Application)



### Respond to a Bid: Class Diagram (Android Application)

## Evaluation of Design Changes in Implementation

#### Design Documents

The team carried out cognitive walkthroughs, coming up with on-paperdesigns for both the web and android applications. These are available on the project repository and form the interface design specification for the web and android applications. Of course, changes were made to these designs and this section will make reference to this. The desktop application saw no interface design documentation creation beyond basic sketches; the desktop application’s design was not deemed as important as web and android, since training would be supplied for administrators.

To form the design of the systems’ data models, a UML class diagram was designed to facilitate the class design of the three applications. Please see appendix 1 for this diagram. This was designed as a basic template – changes were fully expected, since the team was not experienced in such detailed class design at this stage. Again, this section of the report will detail any changes or problems with the initial data model.

### The Desktop Application

To be implemented using Java and Netbeans, this application is to be used by a trained administrator only. It should provide a rich front-end admin interface with full access to the system.

#### Class Design

In comparison to the initial plan for requirements analysis and design a number of changes have been made that reflect the growth of the application as well as critical areas that were overlooked.

The original class diagram available in appendix 1 has changed during the course of the project, however the fundamental idea of how the classes should relate to each other remains the same. The initial class diagram planned on using an enumeration class for the *ItemType* and *AdvertType* classes. This changed during implementation since it became apparent that any changes to the information provided in the database would not be reflected in the enumeration class until the code had been edited. To resolve this, an *ItemTypeList* and *AdvertTypeList* were added to hold these values. In this way, the addition of item or advert types would require minimal changes to any code. This approach was also used in the *ItemCategory* class. A mistake has been made in accordance to the relationship between these classes and an *Advert*. As each of these classes relate to *Advert,* they should have been included in reference to *Advert*. Instead, the GUI references the list as it populates each of these lists on initialisation. This is bad practise as the *Advert* does not contain an item type, advert type or category but instead contains pointers (ID’s) to the specific type in the related lists.

The review information was not implemented until the final stages of development; the concept of any attributed mechanisms was not finalised until this time. Therefore an independent *ReviewList* and *Review* have been included. This is again populated on initialisation and is referenced by the *Transaction* class. Upon reflection, a *Review* should be an object directly relating to the *Transaction* as opposed to being simply referenced by an ID.

A feature added in to the application after design was the ability to add and edit rules. Since the addition of rules should fall to administrators, it was vital that this was included in the desktop application. This class doesn’t interact with any other classes, so it has been shown in the class diagram as being independent.

A design change unique to the Desktop Application is the Admin Authorisation class. This class allows for the administrator to login to the system. The ability to add new Administrators to the system has been implemented however this goal was not identified during the initial planning phase.

Due to time constraints, these problems or errors discussed have not been resolved. Despite this, the application functions entirely as planned. Given more time, the application would be reviewed and these changes would be implemented.

#### Interface Changes

From the initial design document the desktop application has changed dramatically. This has been due to feedback from the cognitive walkthrough and usability studies. Originally the application was to be arranged in individual frames, each offering the desired information dependant on the toggle selected by the user. At final submission, the application provides a tabbed pane approach. This means the user can define their workspace layout. For instance, in the Member Pane, the user can view the system from the point of view of a member.

Alternatively, the user can make use of the Advert Pane, which sees adverts as the point of focus. This change was developed from feedback from the first cognitive walkthrough. The tabbed pane view works well in allowing the admin to see the application holistically. This approach was considered useful in usability studies as users felt they had a better overview of the system and didn’t have to remember information from another pane.

One disadvantage to this approach to layout is evident on the Transaction Pane. There is a lot of information with regards to Transactions to display, filled the panel with information as to bewilder new users. Efforts to resolve this include information breaking, organising pieces of information into titled panes.

#### Missing Features

A number of features were not included due to time restraints:

* Re-Activate membership or advert:

A user and an advert can be banned due to an offence (breaking a rule) or inactivity for an extended period of time. Once a member is banned they cannot currently have that revoked.

* Loading screen between login and loading the application:

After the administrator has logged in there is not a splash/loading screen between the login and the initialisation of the application.

* Contacting a member:

The ability to contact a member to notify them of any rules they have broken, notifying them of inactivity and other system information would have been useful. This was considered a stretch goal. Due to time constraints the application does not support this feature.

Features not included by design are as follows:

* Not included the API for the desktop application:

The desktop application did not use the API created for the Android and Web application due to the fact that the desktop application has separate functionality. The functionality needed for the desktop application is included within the application and therefore an API would not be necessary.

* The use of Oracle Text :

The desktop application searches through the table model so that once a key has been pressed, the list is filtered. This differs from the Android and Web application, which use Oracle Text. This is beneficial to the Android and Web applications as they are updated from frame to frame. However the desktop application updates dynamically when changes are made. Filtering the list in this application appears the better method.

* No means to create an administrator :

For the desktop application there is already a pre-set administrator with their own login credentials. This administrator was assigned to the database by the team members of the project. If a new administrator was required to join the system, they would have to ask for permission to include this administrator in the database. The reason for this decision was for reasons of security. The team felt that allowing anyone to register as an administrator could cause serious ramifications.

### The Web Application

To be implemented using HTML/CSS, JSP, JavaScript and more, this application should provide the richest front-end system for both guests and members. It was desired that this application should offer the most user-friendly experience, since it forms the most easily accessible system offered.

#### Design Changes

One of these most notable changes manifests in the colour scheme. The colour scheme at the first usability study was blue, black and white. During the study it was commented that the entire application was quite dull and should be made to feel and look more engaging and so the team opted for the current colour scheme of varying shades of blue, green, white and orange.

The member’s home screen underwent a change, removing the display of a member’s details, instead displaying these on the member account page. In place of this, notifications visualising the status of the member’s bids and transactions were implemented. These notifications show the number of bids or transactions requiring attention from the member. An example of this is an advert holder receiving a bid; the member should either accept or reject any bid received.

All the pages showing adverts have changed from the initial design. Previously, adverts were displayed in table format with each advert placed in a row of the table. This was altered so that an advert would inhabit a larger white rectangle consisting of rows showing information summarising the advert. The title of the advert was made larger and bolder while other details were altered to be not so prominent. One major improvement was the addition of an image all adverts. If a member does not include an image, the system will automatically upload an image relating to the category of the advert item.

Another feature which has been added was a tabbed feature for the bids and transactions page. At first, the different types of bids and transactions tables were placed in a list. However, under closer observation, it was noted that a member can accrue a large number of bids, requiring them to scroll through a long list of bids or transactions. Separating bids or transactions into tabbed panels allows much quicker access to the required information. A powerful feature added at this point sees members directed to the relevant bid or transaction table when clicking a notification from the homepage. For example if a member has had one of their bids accepted, then clicking the notification would take them to the tab on the “My Bids” page, showing accepted bids. Observations such as these saw the birth and amendment of more detailed use cases, specifically use cases 18 and 19, whereby bids can be displayed by member or by advert.

The ability to search for members was added very late in development. A member can be searched by either entering their forename, surname or both. It was decided that all members should be able to see other members’ names and balances as well as any reviews given to them. The decision to do was due to the fact that a LETS system should promote a policy of openness regarding other members, while ensuring members do not accrue very high or low balances of credits.

Icons have been added to almost every action in the system to promote dual coding functionality. Examples of this are the home page link has a house next to it and the help page contains a question mark enclosed in a circle.

#### Missing Features

Overall, the application has included many more features than originally intended but there are some features the team failed to realise due to time constraints:

* The ability for a member to edit an already published advert.
* Pagination of adverts. Currently, there exist no limits to the number of adverts shown on the browse adverts page. It would be beneficial limit the display of adverts to around 30 on any page, offering users the ability to see the next 30 on the subsequent page.
* View transaction statuses on the same page as the advert they belong to. Currently only the bids of an advert are shown on the same page as the advert.
* A messaging service for members to communicate with one another to make the process of organising selling and buying easier. Currently, a member’s email and phone number is appended to accepted bids. This is not ideal, however and was put in place to get around this issue.
* A forum where members could post discussion topics and receive help about anything they wish.
* Allow for a member to reset their password if they had forgotten it. Currently they can change their password only when logged in or contact admin to change it if they have forgotten it.
* The ability to filter adverts by date added, expiry date, category, advert type (Offer / Request) or item type (Product / Service).
* The ability to search for adverts posted by a specific member.

### The Android Application

To be implemented for an Android phone or device, this application should provide members access to the system, allowing adverts to be created and bids made. Essentially, this forms a streamlined version of the Web Application in terms of functionality.

From the start of development, the team had a general idea of how the application was to look as well as the general flow of the various pages. However, since the development team were new to Android programming and design, this project was very much a learning experience. Many of the design plans and software structures changed as the team became more proficient with Android and experimented with different layouts and interfaces.

It was decided early on that since this was a university project and not an application to be released, the team would target only Android’s newest software, “Lollipop”. This decision was made as the team desired an experience more relative to the software landscape towards the end of the course.

#### Interface Changes

During development, many changes have been made when compared to the initial planning and cognitive walkthrough documents:

* The application colour palette was changed slightly to a lighter blue as the original caused problems with contrast and readability when displayed on a phone screen
* The home page (displaying a list of current adverts as well as buttons to create new ones) was changed to a “Dashboard” with several icons linking to the main parts of the application as well as the ability to search all adverts from the task bar. This change was made after deciding the original design was too cluttered and awkward to use.
* While the flow of the advert creation process was largely the same to the web application, several changes were made to reduce the number of steps involved and remove any clutter.
* A lot like the website’s separation of different types of bid, these are separated into different tabs, navigated by swiping left and right. This was one of the many subtle changes made in the design to accommodate the limited screen space available to a mobile application.
* The inclusion of a menu bar that is consistently accessible from any page to aid user navigation

These interface changes and others were made as a result of experimentation and user testing. While there are several areas that could see improvement if given more time, the application flows well and maintains a professional look and feel in most areas.

#### Problems

While the android development team have created high quality work in the past they were new to android, which meant that some research and experimentation was required in order to implement almost every feature. It has been noted that although Android is based on Java, there are many differences and quirks in comparison. This translated into the reality that development proceeded much more slowly than was anticipated and a lot of effort was expended fixing bugs and problems without any clear source.

Adding to the problems from an inexperienced android development team, many early interface designs were awkward to use and felt poorly designed. As development progressed, the interface design improved markedly, as did the team’s ability to invent new designs, avoiding the need for further iterations.

Another problem encountered by the design team was a lack of Model-View-Controller (MVC) support within android, making many of the early system designs incorrect. Android requires much more to be done within the “Activity” class that deals with the GUI than the team was used to and this has led to the inclusion of some poorly organised code in the release. However with the experience gained on this project, the team feels confident that new projects would proceed far more smoothly.

#### Missing Features

While most of the major features have been included in the application, there are several that have not made the final release. Most of these features have been dropped due to time restraints; some were not included by design.

A number of features were not included due to time restraints:

* Receive notifications of bids and transactions.
* View a different member’s details.
* Option to delete an advert.
* View the user’s past adverts.
* View bids on an advert via the advert details page.
* Updates to bids and transactions on the dashboard
* Ability to contact admins from inside the application

Further features not included by design are as follows:

* A list of all active members
* The ability to search for a member
* The ability to edit an advert that has been posted
* Filter adverts based on category etc. (This has been replaced by the search feature)

Many of the features not included due to time restraints would be easily implemented if the project were to continue. Unfortunately, due to the difficulties with android and the lack of a 5th group member, they have been dropped for the release.

#### Conclusion

While the Android application has gone through several iterations and is missing some features, it is very close to the team’s initial idea and completes all of the required functionality. Further to this, it looks professional, consistent and is relatively easy to use.

Overall the team is pleased with the outcome of the application and agree that given more time to polish and develop it, it could be used in the real world.

# Databases

This section will see presentation the Oracle Modeler Diagram and the SQL CREATE statements used in the development of the Oracle database used by all three systems. Since these deliverables are best viewed in landscape format, these are available in appendix 2 and 3 respectively. A brief evaluation of the database as a whole is below, with a commentary on the extra Oracle Text integration to follow.

### Database Evaluation

The design of the database was a problem consistently throughout the integration of the project. Losing a group member was a significant problem here; the database lead developer would have been taken up by the group’s fifth member. This group composition has been adopted in previous group assignments and it was unfortunate to have lost this opportunity.

A basic table structure was defined early in the project, but a series of mistakes or omissions were found throughout the project. These problems were minor, but they showed a lack of understanding with exactly the data the database needed to keep track of. Details such as a textual field in the review table and definitions of which member IDs required storing in bids and transactions are examples of omissions in early database planning documents. Table interactions were not seen to change in the implementation, however – the table definition in the design phase was designed well.

An unusual change in design came at the mid-way point of development. The name of the table “Offers” was changed to “Bids.” Since the system deals with offers and requests, this was a confusing definition and caused frequent misunderstandings between the team. A final observation of the table structure shows some suggested poor practice in the bids table. Tracking information such as the advert type and including a field for both members included in the bid, shows duplicated information in the database – these fields could be accessed through the advert ID. However, at a design level, these inclusions of duplicated data made any SQL functions simpler to implement. Details such as the payer/payee scenario, which change with the advert type forms evidence to this unusual scenario. The inclusion of a dedicated database lead developer, without the extra responsibilities detailed in section 1, would have overcome these definitions in table design.

### Database Searching

In order to allow for adverts and members to be searched in the database, the relevant columns were indexed to allow for this.

For the adverts table, this was done by adding the title and description columns into a multi-column data store. Next, the data store was indexed to allow those columns to be searched by members.

The members table was done in the same way, except the columns added to the data store were the member’s forename and surname.

For both tables, an extra column was added called ‘dummy\_indexing.’ This column is what is searched which then links to the index for the table.

Indexing columns allows for a member to enter a phrase such as ‘Van Hire’, or single words such as ‘van’. For members, the search term could be either the forename or surname or both separated by a space.

The inclusion of Oracle Text adds extra functionality to the system – not just allowing the facility of data to be searched through, but the speed and computation costs are both improved dramatically. This sort of functionality is an Oracle special and is well worth making use of, considering the costs attributed to deploying an Oracle Database.

# Security

This section will detail the additional features added to the systems to facilitate security. Since personal data on members are stored and transactions containing theoretical funds are actioned, security is especially important in these systems.

## Web Application and Android

All members need to register and upon doing so, create a password. This password is hashed using *Jbcrypt*, a key derivation function based on the Blowfish Cipher. During hashing, a salt value is added to the password to protect against rainbow table attacks. *Bcrypt* is an adaptive function in that over time, the iteration count can be increased to make it slower. This increases its resistance to brute-force attacks. The *Jbcrypt* function offers an optional parameter that determines the computational complexity of the hashing, which defaults at 10. Our system has this set to 12 to safe-guard against the ever-increasing speed of computerised attacks.

Once a member registers, only the salted and hashed password is saved in the database and the original deleted. This is necessary since without encrypting passwords, administrators could access the passwords manually in their text form, forming potential disputes in the eyes of legislation.

## Web Application

To secure connections between client and server, the system has seen TLS 1.2 implemented. This is used as soon as a member logs in and every web page while logged in to ensure security of personal data.

Once a member is signed in, a session variable is set to hold some of the member’s details. On every subsequent page visit, a filter is run to check if the member is still logged in using this session variable. This ensures that once the member actually clicks the logout button or the session expires, then any of the previous pages visited cannot be accessed again unless they once again sign in. This is especially useful if accessing the website from a shared computer.

## The API

The API uses Basic Authentication with TLS. This means that on every web service request, depending on which web service is requested, the requester needs to be authenticated. To achieve this, the request attaches a header to the https request with the member’s email and hashed password, encoded in base64. To ensure that the member’s email and password cannot be accessed during transport, the connection is secured using TLS 1.2.

Necessary improvements given more time:

* On a web service request, the web service should only allow a member to access their own data. Currently, any member could access any data as long as they are a registered member. Many more checks would need to be implemented to ensure this.
* Basic authentication felt right for this application as the application won’t hold highly personal data but another possibility would be to use OAuth 1.0a, OAuth 2 or Shared Key Authentication such as Amazon’s S3 web services.

# Software Engineering

This section will detail the noteworthy features, shortcomings, problems or issues in the three applications from a software engineering standpoint. Each application will be discussed separately under its own heading, with the RESTful web services seeing discussion at the end of this section.

## The Desktop Application

#### Noteworthy Features

The desktop application has unique features that are not used in the Android or Web application. The reason for these unique features is that the desktop application is specifically created for administrators. Administrators require full access to information stored in the database in case any information is not correct, rules are being broken or an overview of system activity is required.

The desktop application makes use of a recent Oracle implementation that allows for changes made in a database to be used to notify Java applications. The reason for implementing this technology is that it provides the administrators of the system to be notified of any changes made to data in the database. This notification then allows the administrator to have the option to update the information they are viewing. If the administrator does not manually update the application, it will automatically update every 2 minutes. This has been put in place to ensure that the most up to date information is being viewed at any time. An issue implementing this technology was that a specific port was required to be opened to allow the notifications to be sent through. This was an issue within university as permission to allowed ports to be opened is blocked by privileges.

Initially, when dealing with network tasks, the application was noted to take a long time to respond. The assumed reason for this was the fact that the GUI, Data model and Networking were all being processed on the main thread. To speed up the application, all networking tasks were handled on another thread. Once the networking tasks were completed, the GUI and data model would be updated. This dramatically increased the speed of the application.

To ensure that administrators and user passwords are safe they are saved in the database as encrypted, hashed passwords. A more detailed analysis can be found in section 5. When resetting passwords, functionality has been implemented to simply re-assign a new password, rather than send a member their old password. Given more time, a system could be put in place to send members an email suggesting how to reset their email or containing a plain text version of their password. The team has decided that the current way of dealing with passwords is safer for security purposes.

For each of the fields that require administrator input, regular expressions have been assigned to the input of the text fields. These text fields highlight green to hint that the input meets the correct format; if the input is red this indicates the input is incorrect. If any of the text fields contain incorrect information, the administrator will be notified they cannot make any updates until this has been corrected. This front end data integrity is backed up by the database with back-end data validation.

To increase the usability and fluidity of the application, the Observer Pattern has been implemented. This design pattern allows any changes to the data model to update all necessary elements or views of the application front end, automatically. The Observer Pattern means that when a change is made, the information doesn’t have to be manually updated. On a user level, this ensures the administrator is consistently shown the most up-to-date information.

#### Problems

The desktop application planned to use a Singleton class to handle a database connection. While developing the application, it became apparent that this method would not add much value to the application. The ability to open and close the database connection has meant that networking is only used when necessary, freeing up resources. Other advantages of not using a Singleton class involve improvement to the flexibility of the class; Singleton classes are known to be highly coupled across applications, making it difficult to modify in practice.

To provide the user with the ability to *undo* and *redo* changes, the Command Pattern was specified at the design stage. Due to time constraints, however, this was left out of the application due to its complexity. In retrospect, this would form a useful part of the system. Omitting the Command Pattern does not directly affect the goal of the application, but the usability would have been greatly improved within the application if it was offered.

#### Coding Practice

The desktop application has been developed with coding practices in mind. For instance, the application reduces the amount of code re-use where possible. This was achieved by splitting the GUI class into multiple classes, each with an individual purpose. The application was initially divided into separate packages but in following the package principles, these were developed into libraries. By splitting the classes into their own individual packages, the code can be re-used over a number of applications.

Where necessary, Interface and Abstract classes have been used to improve the code quality. These class types improve code quality by reducing the amount of repeated code, forcing other classes to implement methods and improve code readability. To improve code readability, the Java code conventions have been followed. Java documents (JavaDocs) were generated for the data model and comments were added to help understand the flow of code.

## The Web Application

Designing a web application using JSP and servlets was a first for the team. The first task was to get familiar with the workings of this technology. Due to our limited understanding, the initial structure of the web application was not as well formed as it is in the final release.

Initially all JSP pages were treated like static HTML pages and each one contained duplicate code, primarily in the form of navigation bars. Over time, the team learnt that large chunks of code could be placed into separate JSP files and then included using the include tags provided by JSP syntax. This was hugely beneficial for further development as it encouraged DRY (Do not Repeat Yourself) coding. As an example, the sidebar was used on in excess of 12 webpages and consisted of about 80 lines of code per page. On release, this code was contained within one file and referenced in every page required. It also meant that when one link’s URL changed, the change could be made in a single file rather than in all 12 pages, reducing the chance of errors.

With this knowledge gained, a lot more code was taken out and put into separate JSP files such as tables of bids or adverts which are exactly the same but in different pages. This method of structuring code was deemed more elegant than repeating code in many pages but on further reflection, it was noted this was *still* not the best approach. Code which called web services was placed in JSP files and it turned out that this was very difficult to debug and track down errors. The next step taken was to take this code and put it into servlets.

This new code structure allowed any required web services to be called before page load. In this way, were an error to occur, then the error would be caught and the user notified. Using servlets also allowed for multi-threaded code. Whenever multiple web services were called (such as getting four tables filled with different types of transaction), then each of the web service calls were run in their own threads. The effect this had on page loading times was very significant. As an example, the ‘My Transactions’ page would normally take 9 seconds to load but due to the multi-threaded code it now takes about 1 second. Of course, this depends on the amount of bids the system has to retrieve.

Another feature worth mentioning is the filter used on almost all of the pages in the application. This filter is run every time a page is requested. If the page is only allowed to be viewed when a member is logged in, the filter will activate and check if a member is actually logged in. This prevents guests or members directly accessing pages not meant for their consumption.

Classes used within the application have been placed into packages according to their functionality to make it easier for future developers to understand and maintain. One such class is the web service connection class which has its own package and could be reused by many other applications if so desired.

The use of a data model within the web application was very different to the desktop application’s data model usage. The entity classes such as member and advert were only used to hold data, before converting the object JSON for transport via web services. There is no explicit relationship between the classes. However, the data model has been placed into a library which both the API and the web app use. This meant that if the data model needed to be changed then it would only be changed in one place and not in both the API and the web application.

#### Problems

One problem related to any called servlets, which in themselves call a web service to insert a new entry into the database. If the page refresh button is pressed, the same data can be inserted twice. For instance, refreshing the page after creating an advert can result in duplicated adverts. This could result in any number of problems for members, so the way the servlet redirects to other pages needed to be changed so that pressing the refresh button on a page did not resend the new data to the database. This is fully resolved in the release.

The application is very stable and there are not many times where an error happens but when such an error does occur there has been code put in place which tries to catch these errors and redirect the user to an error page from which they can login again. This however does not always work and in such cases the web page just goes blank/white and the user needs to open a new page manually.

In order to decrease page load times, details of the currently logged in member are saved into a session variable. This has made the system much more responsive, except in the instance whereby the currently logged in member’s details have been changed by another source. Examples of this include credits transferred to their account or changes made in the Android application by the same member. To overcome this, a request or check to see if any details have changed recently would require implementing. Sadly, this approach would take away the benefit of saving the details in a session variable; page loading times would increase as a result of removing session variables.

#### Conclusion

Learning to program JSP and servlets has been a challenging but welcome task as a lot of new skills have been learnt. Future projects of a similar nature would undoubtedly be much more seamless due to the knowledge gained by undertaking this project.

## The Android Application

Android applications are required to do many processes on separate threads in order to prevent blocking the main thread. If an application blocks the main thread for more than a few seconds it is immediately terminated. Because of this, the team has gained valuable experience working with threads and debugging multi-threaded code. Further to this, a better understanding of thread safety and how to best use threads to speed up applications have been gleaned.

The team is mostly pleased with the software side of the application. It runs smoothly and maintains a clear look with easy-to-understand controls. However if the project was to be undertaken again, it is believed that the overall software structure and design could be greatly improved using the new experience gained.

#### Noteworthy Features

Almost the entire application is multi-threaded, running different threads for encryption, networking and the GUI itself. The application is believed to be thread-safe and no corruption can occur whilst any particular thread is running.

As the team has learnt from previous projects, it is important to design the class structure well to contain any changes that need to be made. Toward this end, the application has been split into packages as much as was feasible. The data model and classes have been separated from the GUI as much as possible to follow MVC, making development and maintenance much easier, as well as improve code navigation.

For several sections of the application the team used the same XML layout form in order to increase the consistency of the layout. The whole application uses the same menu and action bar layout throughout and many of the list layouts have been repeated. Consistency was the focus here; keeping similar components and designs across the application enhances system usability.

For the list of bids and transactions, the team opted to use Fragments as this allows multiple displays to be included in a single activity, allowing the user to quickly swipe between lists. This implementation of a fragment allows the encapsulation of the available bid types, meaning fewer resources are used and the usability is improved as navigation between bids does not involve starting new activities.

#### Coding Practice

Several methodologies have been employed in the development of the Android application including DRY coding where possible and the separation of classes into packages to not only increase productivity during development, but also aid in maintenance of the code in the future. The code has also been extensively commented by the team in order to help readability.

Standard Java naming conventions have been used throughout the application, aiding development and making it easier for other developers to understand how the programme works, however some of the variable names are not ideal. This was mainly caused because many of the features required experimentation and research to implement and sometimes code was not correctly cleaned up after it was fixed.

While it was initially planned to use more abstractions and interfaces to improve the structure of the application, the team has failed to use these effectively due to a combination of being new to Android, the structure of Android itself, and the experimental nature of the development process.

#### Problems

Several problems with Android development had to be overcome during this project, requiring many hours of research and slowing development significantly. One of the most egregious problems the team came across was the amount of deprecated code included in Android and the lack of up-to-date guides provided by Google; many of the official guidelines were out of date or incorrect at best, non-existent at worst.

During development, many software bugs were overcome. Some were formed from simple programming mistakes, while some were caused by the works of Android itself. Some of the more notable issues include:

* The JSON library required to send information to the API encoding a string different in Android and Java: in some cases, the string was seen to be enclosed in square brackets. This is not the behaviour the documentation suggested. In turn, this caused the API to return a failed message. This was fixed by using string manipulation to remove the brackets after the object had been encoded.
* On the Bid and Transaction interfaces, the application was crashing when one of the team was using it. This bug was caused by the way the team member was holding the phone. An item in the list was being pressed as the menu was being swiped. This bug was fixed by delaying the generation of an on-click listener until after the swipe had been completed.

It is believed that the application is stable at release, however there are a few problems still to be worked out, such as the application logging the user out if the user switches to another application and back again. This sometimes causes an infinite login screen that requires the application to be removed from the application history of the device and restarted.

#### Conclusion

While most of the planned features have been included and the application is mostly stable, several more advanced features are missing and some bugs are yet to be resolved. The team feels that given a little more time that the application could be made to be completely stable and any missing or additional features could be quickly integrated.

## RESTful Web Services

The team began with no experience of developing web services and in particular RESTful web services. Initially, the web services were auto-generated within NetBeans using the functionality provided. This was done by linking the database to NetBeans and then generating web services for every table in the database. This approach was only useful in that it helped the team understand how RESTful web services operated. The downside of this approach was that it required extra libraries to be added such as Java Persistence API (JPA). This has drawbacks as the libraries are not always supported in servers such as Apache Tomcat.

Upon achieving an understanding of RESTful services, the RESTful API was completely rewritten because the web services automatically provided by NetBeans did not provide the functionality required by the web and Android applications. Rewriting the API also meant that the API would be compatible within servers such as Tomcat, although the team has opted to use Glassfish Server 4.1. Compatibility across a range of servers can only be seen as a benefit.

The team requested a Glassfish server setup on the system provided by the course instructors. The option to develop this functionality as a team - to be rolled out for use by other groups - was offered. This opportunity was accepted and subsequently delivered by the team’s API developer.

The choice to use web services was adopted as it allowed for both the web and Android application to connect to the database through a single source. This ensures both applications are forced to enter data into the database in the same format. Using web services also extracted the database query implementation out of the web and Android applications, thus making it much quicker and easier to develop these applications, since queries only required implementation in one source.

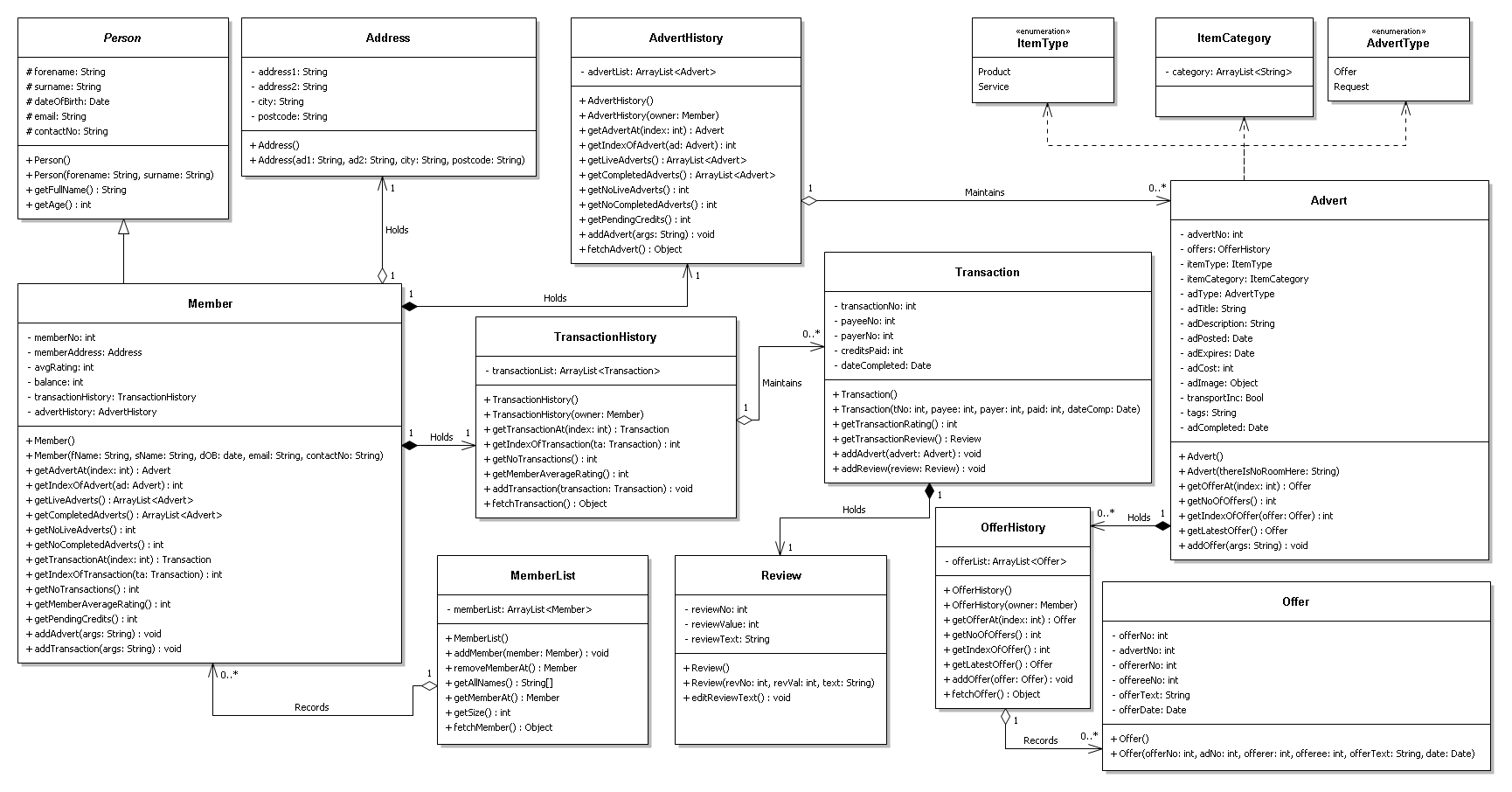
The web services are stateless, thus every request to the server is dealt with and then forgotten. By doing this the server will not be overloaded with keeping sessions alive for all current requests and therefore many more servers hosting the web services can be added to allow for load balancing if required. Further to this, supposing a server hosting the web services needed to be taken down, there is no requirement to transfer any data representing sessions.

As the web services are designed using RESTful, the errors provided map to http error codes such as ‘204 – No Content’. All errors provided when an input fails will be the same for both applications and can provide useful errors to members.

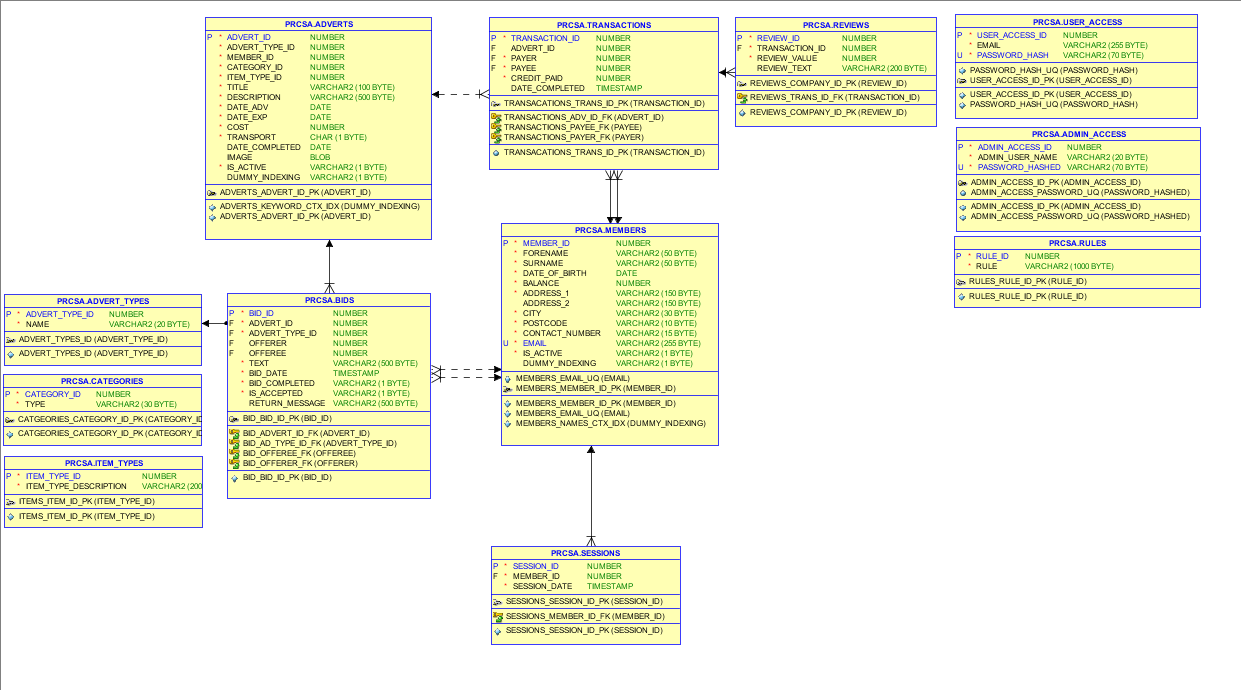
# Conclusion

Blah.

# Appendix A – Data Model Class Diagram (Design Level)

Note: this diagram is not to form a representation of the final system; this is a design document to facilitate the data models’ structure.

# Appendix B - SQL Modeler Diagram



# Appendix C - SQL CREATE Statements

### Admin Access Table

CREATE TABLE "PRCSA"."ADMIN\_ACCESS"

(

"ADMIN\_ACCESS\_ID" NUMBER,

CONSTRAINT "ADMIN\_ACCESS\_ID\_PK"

PRIMARY KEY ("ADMIN\_ACCESS\_ID"),

"ADMIN\_USER\_NAME" VARCHAR2(20 BYTE)

CONSTRAINT "ADMIN\_ACCESS\_USER\_NAME\_NN" NOT NULL ENABLE,

"PASSWORD\_HASHED" VARCHAR2(70 BYTE)

CONSTRAINT "ADMIN\_ACCESS\_PASSWORD\_NN" NOT NULL ENABLE

CONSTRAINT "ADMIN\_ACCESS\_PASSWORD\_UQ"

UNIQUE ("PASSWORD\_HASHED")

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."ADMIN\_ACCESS\_TRIGGER"

BEFORE INSERT ON admin\_access

FOR EACH ROW

BEGIN

SELECT admin\_user\_seq.nextval INTO :NEW.admin\_access\_id FROM dual;

END;

### Advert Types

CREATE TABLE "PRCSA"."ADVERT\_TYPES"

(

"ADVERT\_TYPE\_ID" NUMBER,

CONSTRAINT "ADVERT\_TYPES\_ID" PRIMARY KEY ("ADVERT\_TYPE\_ID"),

"NAME" VARCHAR2(20 BYTE)

CONSTRAINT "ADVERT\_TYPES\_NAME\_NN"

NOT NULL ENABLE

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."ADVERT\_TYPES\_TRIGGER"

BEFORE INSERT ON advert\_types

FOR EACH ROW

BEGIN

SELECT advert\_types\_seq.nextval INTO :NEW.advert\_type\_id FROM dual;

END;

### Adverts

CREATE TABLE "PRCSA"."ADVERTS"

(

"ADVERT\_ID" NUMBER,

CONSTRAINT "ADVERTS\_ADVERT\_ID\_PK"

PRIMARY KEY ("ADVERT\_ID")

"ADVERT\_TYPE\_ID" NUMBER

CONSTRAINT "ADVERTS\_ADVERT\_TYPE\_FK"

FOREIGN KEY ("ADVERT\_TYPE\_ID") REFERENCES "PRCSA"."ADVERT\_TYPES" ("ADVERT\_TYPE\_ID") ENABLE,

CONSTRAINT "ADVERTS\_ADVERT\_TYPE\_NN"NOT NULL ENABLE,

"MEMBER\_ID" NUMBER

CONSTRAINT "ADVERTS\_MEMBER\_ID\_FK" FOREIGN KEY ("MEMBER\_ID") REFERENCES "PRCSA"."MEMBERS" ("MEMBER\_ID") ENABLE,

CONSTRAINT "ADVERTS\_MEMBER\_ID\_NN" NOT NULL ENABLE,

"CATEGORY\_ID" NUMBER

CONSTRAINT "ADVERTS\_CATEGORY\_ID\_FK" FOREIGN KEY ("CATEGORY\_ID")

REFERENCES "PRCSA"."CATEGORIES" ("CATEGORY\_ID") ENABLE,

CONSTRAINT "ADVERTS\_CATEGORY\_ID\_NN" NOT NULL ENABLE,

"ITEM\_TYPE\_ID" NUMBER

CONSTRAINT "ADVERTS\_ITEM\_TYPE\_FK" FOREIGN KEY("ITEM\_TYPE\_ID")

REFERENCES "PRCSA"."ITEM\_TYPES" ("ITEM\_TYPE\_ID") ENABLE

CONSTRAINT "ADVERTS\_ITEM\_TYPE\_NN"NOT NULL ENABLE,

"TITLE" VARCHAR2(100 BYTE)

CONSTRAINT "ADVERTS\_TITLE\_NN" NOT NULL ENABLE,

"DESCRIPTION" VARCHAR2(500 BYTE)

CONSTRAINT "ADVERTS\_DESCRIPTION\_NN" NOT NULL ENABLE,

"DATE\_ADV" DATE

CONSTRAINT "ADVERTS\_DATE\_ADV\_NN" NOT NULL ENABLE,

"DATE\_EXP" DATE

CONSTRAINT "ADVERTS\_DATE\_EXP\_NN" NOT NULL ENABLE,

"COST" NUMBER

CONSTRAINT "ADVERTS\_COST\_NN" NOT NULL ENABLE,

"TRANSPORT" CHAR(1 BYTE)

CONSTRAINT "ADVERTS\_TRANSPORT\_NN" NOT NULL ENABLE,

CONSTRAINT "ADVERTS\_TRANSPORT\_CK"

CHECK (transport IN ('N','Y')) ENABLE,

CONSTRAINT "CHECK\_TRANSPORT"

CHECK (TRANSPORT = 'Y' OR TRANSPORT = 'N') ENABLE,

"DATE\_COMPLETED" DATE,

"IMAGE" BLOB,

"IS\_ACTIVE" VARCHAR2(1 BYTE) NOT NULL ENABLE,

"DUMMY\_INDEXING" VARCHAR2(1 BYTE),

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."ADVERTS\_TRIGGER"

BEFORE INSERT ON adverts

FOR EACH ROW

BEGIN

SELECT adverts\_seq.nextval INTO :NEW.advert\_id FROM dual;

END;

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."IS\_ACTIVE\_ADVERT"

BEFORE INSERT ON ADVERTS

FOR EACH ROW

BEGIN

SELECT 'Y' INTO :NEW.is\_active FROM dual;

END;

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."ADVERTS\_DATE\_TRIGGER"

BEFORE INSERT ON adverts

FOR EACH ROW

BEGIN

:NEW.DATE\_ADV:= SYSDATE;

END;

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."ADVERTS\_DATE\_EXP\_TRIGGER"

BEFORE INSERT ON adverts

FOR EACH ROW

BEGIN

:NEW.DATE\_EXP:= SYSDATE + 14;

END;

BEGIN

ctx\_ddl.create\_preference('adverts\_multi\_col\_datastore', 'MULTI\_COLUMN\_DATASTORE');

ctx\_ddl.set\_attribute('adverts\_multi\_col\_datastore', 'columns', 'title, description');

END;

CREATE INDEX adverts\_keyword\_ctx\_idx

(

ON adverts (dummy\_indexing)

INDEXTYPE IS ctxsys.context

PARAMETERS ('LEXER my\_lexer SYNC (ON COMMIT) DATASTORE adverts\_multi\_col\_datastore');

)

### Bids

CREATE TABLE "PRCSA"."BIDS"

(

"BID\_ID" NUMBER,

CONSTRAINT "BID\_BID\_ID\_PK" PRIMARY KEY ("BID\_ID"),

"ADVERT\_ID" NUMBER NOT NULL ENABLE,

CONSTRAINT "BID\_ADVERT\_ID\_FK" FOREIGN KEY ("ADVERT\_ID") REFERENCES "PRCSA"."ADVERTS" ("ADVERT\_ID") ENABLE,

"ADVERT\_TYPE\_ID" NUMBER NOT NULL ENABLE,

CONSTRAINT "BID\_AD\_TYPE\_ID\_FK" FOREIGN KEY ("ADVERT\_TYPE\_ID") REFERENCES "PRCSA"."ADVERT\_TYPES" ("ADVERT\_TYPE\_ID") ENABLE,

"OFFERER" NUMBER,

CONSTRAINT "BID\_OFFERER\_FK" FOREIGN KEY ("OFFERER")

REFERENCES "PRCSA"."MEMBERS" ("MEMBER\_ID") ENABLE,

"OFFEREE" NUMBER,

CONSTRAINT "BID\_OFFEREE\_FK" FOREIGN KEY ("OFFEREE")

REFERENCES "PRCSA"."MEMBERS" ("MEMBER\_ID") ENABLE,

"TEXT" VARCHAR2(500 BYTE)

CONSTRAINT "BID\_TEXT\_NN" NOT NULL ENABLE,

"BID\_DATE" TIMESTAMP (6)

CONSTRAINT "BID\_DATE\_NN" NOT NULL ENABLE,

"BID\_COMPLETED" VARCHAR2(1 BYTE) NOT NULL ENABLE,

"IS\_ACCEPTED" VARCHAR2(1 BYTE) NOT NULL ENABLE,

"RETURN\_MESSAGE" VARCHAR2(500 BYTE)

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."BIDS\_TRIGGER"

BEFORE INSERT ON bids

FOR EACH ROW

BEGIN

SELECT bids\_seq.nextval INTO :NEW.bid\_id FROM dual;

END;

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."BIDS\_DATE\_TRIGGER"

BEFORE INSERT ON bids

FOR EACH ROW

BEGIN

:NEW.BID\_DATE := SYSDATE;

END;

### Categories

CREATE TABLE "PRCSA"."CATEGORIES"

(

"CATEGORY\_ID" NUMBER,

CONSTRAINT "CATGEORIES\_CATEGORY\_ID\_PK" PRIMARY KEY ("CATEGORY\_ID")REFERENCES "PRCSA"."ADVERTS" ("ADVERT\_ID") ENABLE,

"TYPE" VARCHAR2(30 BYTE)

CONSTRAINT "CATEGORIES\_TYPE\_NN" NOT NULL ENABLE,

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."CATEGORIES\_TRIGGER"

BEFORE INSERT ON categories

FOR EACH ROW

BEGIN

SELECT categories\_seq.nextval INTO :NEW.category\_id FROM dual;

END;

### Item Types

CREATE TABLE "PRCSA"."ITEM\_TYPES"

(

"ITEM\_TYPE\_ID" NUMBER,

CONSTRAINT "ITEMS\_ITEM\_ID\_PK" PRIMARY KEY ("ITEM\_TYPE\_ID")

"ITEM\_TYPE\_DESCRIPTION" VARCHAR2(200 BYTE) ,

CONSTRAINT "ITEM\_TYPES\_DES\_NN" NOT NULL ENABLE

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."ITEM\_TYPES\_TRIGGER"

BEFORE INSERT ON item\_types

FOR EACH ROW

BEGIN

SELECT item\_types\_seq.nextval INTO :NEW.item\_type\_id FROM dual;

END;

### Members

CREATE TABLE "PRCSA"."MEMBERS"

(

"MEMBER\_ID" NUMBER,

CONSTRAINT "MEMBERS\_MEMBER\_ID\_PK" PRIMARY KEY ("MEMBER\_ID"),

"FORENAME" VARCHAR2(50 BYTE)

CONSTRAINT "MEMBERS\_FORENAME\_NN" NOT NULL ENABLE,

CONSTRAINT "CHECK\_FORENAME" CHECK (REGEXP\_LIKE(FORENAME,'^[A-Za-z]+$')) ENABLE,

"SURNAME" VARCHAR2(50 BYTE)

CONSTRAINT "MEMBERS\_SURNAME\_NN" NOT NULL ENABLE,

CONSTRAINT "CHECK\_SURNAME" CHECK (REGEXP\_LIKE(SURNAME,'^[A-Za-z]+$')) ENABLE,

"DATE\_OF\_BIRTH" DATE

CONSTRAINT "MEMBERS\_DATE\_OF\_BIRTH\_NN" NOT NULL ENABLE,

"BALANCE" NUMBER

CONSTRAINT "MEMBERS\_BALANCE\_NN" NOT NULL ENABLE,

CONSTRAINT "CHECK\_BALANCE" CHECK (BALANCE > 0) ENABLE,

"ADDRESS\_1" VARCHAR2(150 BYTE)

CONSTRAINT "MEMBERS\_ADDRESS\_1\_NN" NOT NULL ENABLE,

"ADDRESS\_2" VARCHAR2(150 BYTE),

"CITY" VARCHAR2(30 BYTE)

CONSTRAINT "MEMBERS\_CITY\_NN" NOT NULL ENABLE,

CONSTRAINT "CHECK\_CITY" CHECK (REGEXP\_LIKE(CITY,'^[A-Za-z]+$')) ENABLE,

"POSTCODE" VARCHAR2(10 BYTE)

CONSTRAINT "MEMBERS\_POSTCODE\_NN" NOT NULL ENABLE,

CONSTRAINT "CHECK\_POSTCODE" CHECK (REGEXP\_LIKE(POSTCODE,'^[A-Z]{1,2}[0-9][0-9A-Z]? ?[0-9][A-BD-HJLNP-UW-Z]{2}$')) ENABLE,

"CONTACT\_NUMBER" VARCHAR2(15 BYTE)

CONSTRAINT "MEMBERS\_CONTACT\_NUMBER\_NN" NOT NULL ENABLE,

CONSTRAINT "CHECK\_NUMBER" CHECK (CONTACT\_NUMBER > 9) ENABLE,

"EMAIL" VARCHAR2(255 BYTE)

CONSTRAINT "MEMBERS\_EMAIL\_NN" NOT NULL ENABLE,

CONSTRAINT "MEMBERS\_EMAIL\_UQ" UNIQUE ("EMAIL"),

"IS\_ACTIVE" VARCHAR2(1 BYTE) NOT NULL ENABLE,

"DUMMY\_INDEXING" VARCHAR2(1 BYTE)

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."MEMBERS\_TRIGGER"

BEFORE INSERT ON members

FOR EACH ROW

BEGIN

SELECT members\_seq.nextval INTO :NEW.member\_id FROM dual;

END;

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."IS\_ACTIVE\_MEMBER"

BEFORE INSERT ON MEMBERS

FOR EACH ROW

BEGIN

SELECT 'Y' INTO :NEW.is\_active FROM dual;

END;

BEGIN

ctx\_ddl.create\_preference('members\_multi\_col\_datastore', 'MULTI\_COLUMN\_DATASTORE');

ctx\_ddl.set\_attribute('members\_multi\_col\_datastore', 'columns', forename, surname');

END;

CREATE INDEX members\_names\_ctx\_idx

(

ON members(dummy\_indexing)

INDEXTYPE IS ctxsys.context

PARAMETERS ('SYNC (ON COMMIT) DATASTORE members\_multi\_col\_datastore');

)

### Reviews

CREATE TABLE "PRCSA"."REVIEWS"

(

"REVIEW\_ID" NUMBER,

CONSTRAINT "REVIEWS\_COMPANY\_ID\_PK" PRIMARY KEY ("REVIEW\_ID"),

"TRANSACTION\_ID" NUMBER

CONSTRAINT "REVIEWS\_COMPANY\_NAME\_NN" NOT NULL ENABLE,

CONSTRAINT "REVIEWS\_TRANS\_ID\_FK" FOREIGN KEY ("TRANSACTION\_ID") REFERENCES "PRCSA"."TRANSACTIONS" ("TRANSACTION\_ID") ENABLE,

"REVIEW\_VALUE" NUMBER

CONSTRAINT "REVIEW\_VALUE\_NN" NOT NULL ENABLE,

CONSTRAINT "CHECK\_REVIEW\_VALUE" CHECK (REVIEW\_VALUE > 0) ENABLE,

"REVIEW\_TEXT" VARCHAR2(200 BYTE)

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."REVIEWS\_TRIGGER"

BEFORE INSERT ON reviews

FOR EACH ROW

BEGIN

SELECT reviews\_seq.nextval INTO :NEW.review\_id FROM dual;

END;

### Rules

CREATE TABLE "PRCSA"."RULES"

(

"RULE\_ID" NUMBER,

CONSTRAINT "RULES\_RULE\_ID\_PK" PRIMARY KEY ("RULE\_ID"),

"RULE" VARCHAR2(1000 BYTE)

CONSTRAINT "RULES\_RULE\_NN" NOT NULL ENABLE

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."RULES\_TRIGGER"

BEFORE INSERT ON rules

FOR EACH ROW

BEGIN

SELECT rules\_seq.nextval INTO :NEW.rule\_id FROM dual;

END;

### Sessions

CREATE TABLE "PRCSA"."SESSIONS"

(

"SESSION\_ID" NUMBER,

CONSTRAINT "SESSIONS\_SESSION\_ID\_PK" PRIMARY KEY ("SESSION\_ID"),

"MEMBER\_ID" NUMBER NOT NULL ENABLE,

CONSTRAINT "SESSIONS\_MEMBER\_ID\_FK" FOREIGN KEY ("MEMBER\_ID") REFERENCES "PRCSA"."MEMBERS" ("MEMBER\_ID") ENABLE,

"SESSION\_DATE" TIMESTAMP (6),

CONSTRAINT "SESSIONS\_DATE\_NN" NOT NULL ENABLE

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."SESSIONS\_TRIGGER"

BEFORE INSERT ON sessions

FOR EACH ROW

BEGIN

SELECT sessions\_seq.nextval INTO :NEW.session\_id FROM dual;

END;

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."SESSIONS\_DATE\_TRIGGER"

BEFORE INSERT ON sessions

FOR EACH ROW

BEGIN

:NEW.SESSION\_DATE := SYSDATE;

END;

### Transactions

CREATE TABLE "PRCSA"."TRANSACTIONS"

(

"TRANSACTION\_ID" NUMBER,

CONSTRAINT "TRANSACATIONS\_TRANS\_ID\_PK" PRIMARY KEY ("TRANSACTION\_ID")

"ADVERT\_ID" NUMBER,

CONSTRAINT "TRANSACTIONS\_ADV\_ID\_FK" FOREIGN KEY ("ADVERT\_ID") REFERENCES "PRCSA"."ADVERTS" ("ADVERT\_ID") ENABLE,

"PAYER" NUMBER

CONSTRAINT "TRANSACTIONS\_PAYER\_NN" NOT NULL ENABLE,

CONSTRAINT "TRANSACTIONS\_PAYER\_FK" FOREIGN KEY ("PAYER")

REFERENCES "PRCSA"."MEMBERS" ("MEMBER\_ID") ENABLE,

"PAYEE" NUMBER

CONSTRAINT "TRANSACTIONS\_PAYEE\_NN" NOT NULL ENABLE,

CONSTRAINT "TRANSACTIONS\_PAYEE\_FK" FOREIGN KEY ("PAYEE")REFERENCES "PRCSA"."MEMBERS" ("MEMBER\_ID") ENABLE,

"CREDIT\_PAID" NUMBER

CONSTRAINT "TRANSACTIONS\_CREDITS\_NN" NOT NULL ENABLE,

"DATE\_COMPLETED" TIMESTAMP (6)

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."TRANSACTIONS\_TRIGGER"

BEFORE INSERT ON transactions

FOR EACH ROW

BEGIN

SELECT transactions\_seq.nextval INTO :NEW.transaction\_id FROM dual;

END;

### User Access

CREATE TABLE "PRCSA"."USER\_ACCESS"

(

"USER\_ACCESS\_ID" NUMBER,

CONSTRAINT "USER\_ACCESS\_ID\_PK" PRIMARY KEY ("USER\_ACCESS\_ID")

"EMAIL" VARCHAR2(255 BYTE)

CONSTRAINT "USER\_ACCESS\_EMAIL\_NN" NOT NULL ENABLE,

"PASSWORD\_HASH" VARCHAR2(70 BYTE)

CONSTRAINT "PASSWORD\_HASH\_NN" NOT NULL ENABLE,

CONSTRAINT "PASSWORD\_HASH\_UQ" UNIQUE ("PASSWORD\_HASH")

)

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCSA"."USER\_ACCESS\_TRIGGER"

BEFORE INSERT ON user\_access

FOR EACH ROW

BEGIN

SELECT user\_access\_seq.nextval INTO :NEW.user\_access\_id FROM dual;

END;