Major Project:
Electronic Asset Trading Platform

CAB302 Semester 1, 2021

Date due: 06/06/21 (Sunday, Week 13)

Weighting: 60% (50% group, 10% individual)

Specification version: 1.0

Overview

The unnamed client corporation has contracted your team of developers to create a client-server system for electronic trading of virtual assets within their organisation. Included in this document are a description of the project from the corporation (Appendix A), as well as a collection of user stories from project stakeholders (Appendix B).

Your team's task is to **gather requirements** from the project description and user stories. Your team must produce a **detailed design** of the system you intend to construct to meet these requirements. You must then create an **implementation** of the system in the Java programming language, along with a comprehensive **unit testing** suite. Finally, you will need to **demonstrate** this system to the client, showing how you have met their requirements. Due to the international nature of our client and the present situation with COVID-19, this demonstration will need to be recorded in the form of a **video**. The client also wants individual videos from members of your team, detailing their individual contributions.

These are not sequential steps, and it is expected you will revisit earlier steps as you develop the project and learn more about what you are creating.

In addition to your code and the reports described previously, the client also insists on having access to the full revision history of your project. This means that, from the start, your project's working directory needs to be hosted in a Git repository and you need to be regularly creating commits with useful commit messages. This will also help your team to collaborate with each other. (Note that, if you have a copy of your repository hosted by a provider like GitHub or BitBucket, **make sure that your project is private** to ensure that your code is safe from other competing teams who might try to pass off your hard work as their own.)

The client will naturally want to be updated at various points during the project's development and you will be required to submit videos for that purpose as well.

Project Stages

Requirements (5%)

For this part of the project you must create a **requirements** document, which simply consists of a list of the client's requirements, for use when creating your detailed design. This may be a separate document or part of your main report (along with the detailed design.)

You will find the client's requirements in Appendix A (Project description) and Appendix B (User stories) of this specification. If you require clarification as to the client's requirements, send email to cab302@qut.edu.au – before you do so, however, check the "Responses to Client Inquiries" folder on Blackboard (under Assessment -> Major Project) to see if your question has already been answered.

We have no specific requirements for how this document needs to be formatted- simply that it contains a list of requirements and their priorities (e.g. must have, should have, nice to have) and that those requirements are sensibly grouped (e.g. by priority.) You can also include e.g. very basic user interface mockups if those help to describe the design for the project. The main thing is that this document **needs to contain all the information needed by a project team** to implement the project the client wants without needing to go back to the information in Appendices A and B. In other words, you need to synthesise **all relevant facts** contained within those appendices into your requirements document.

You can search the web for "product requirements document" to get ideas for possible structures for this document, but you are not required to follow any of them.

Expected length: 2-4 pages.

Detailed Design (10%)

Your team needs to create a **detailed design** document that takes the project requirements from your requirements document and describes, in detail, the components needed to create the software you have been tasked with developing.

As with the requirements document, there is no specific format required. However, it is expected that your document will contain at least the following:

- Designs of the (public) Java classes that will comprise your programs:
  - Descriptions of all public methods and fields.
  - Descriptions of the arguments each method takes and what the method returns.
  - Any assumptions (preconditions and postconditions).
  - Any checked exceptions that may be thrown by the method, and the circumstances under which the exceptions will be thrown.

  You may find it useful to use the JavaDoc tool to generate these descriptions from /**-style comments in your source code. This will make it easier to keep the design document up to date with the source code. You are permitted to submit your JavaDoc separately and to reference it from your detailed design document.

  It may be a good idea at this stage to create your project in your Java IDE and start creating these classes and methods. Do not create the actual implementations yet (just make all the methods return e.g. 0 or 'null'), but just creating the classes and methods will allow you to start creating JavaDoc-style /** comments, which you can use as part of your design process.

- How these classes interact with each other (e.g. which classes call which other classes, which classes are passed as arguments to the method calls of other classes.)
    - You may find it useful to create a **class diagram** to document this, but this is not required. Search for "UML class diagram" to see popular examples of this.

      Note that **not every public Java class** needs to be documented here- just the ones that are responsible for the internal work that needs to be carried out, and therefore the classes that would be tested via unit tests. Examples of classes that you do **not** need to include here are:
        - Test classes
        - Exceptions
        - GUI forms
- Designs of the forms that make up the client's graphical user interface, and how these forms are interconnected.
    - These can be e.g., scanned/photographed sketches, but I would suggest using drawing tools to create respectable looking user interface mockups.
    - If you created the server as a GUI program, you should also include designs for the server's GUI forms.
- The design of the **database schema**
    - Tables and table columns (column names and data types)
    - How those tables are connected (primary keys and foreign keys)
      You may find it useful to use Object-Role Modelling to present this information, but it is not necessary.
- The design of the **network protocol** used by the client and server to exchange information.
    - A description of the data that is sent from client to server or from server to client, and how that data should be interpreted.
    - Basically, this part should contain all the information required to create a compatible server or client that could be used to replace the server or client that your team creates.

Like every other step, it is not expected that you will create this in one go. You will end up refining this document over the course of the project as things change and you learn more about the software you are creating. Make sure it is kept up to date.

You can search the web for "software design document" to see examples of detailed design documents for software projects, but once again, you are not required to follow any particular structure for this document.

The principal litmus test by which you should judge this document is if a skilled team would be able to take it and produce the software (by creating the classes described within the design and developing their implementations) without needing to go back to the requirements document.

Expected length: 6-12 pages, or longer if your detailed class description is included in the page count.

Unit Testing (10%)

For this part of the project, your team will need to take your detailed design document and use the information inside it to prepare **JUnit 5 unit tests** for each of the classes described within.

Initially this will be **black box** unit tests, created before your team has developed the implementations for these classes. These unit tests should check that the classes follow their specifications as described in the detailed design document.

Remember to create unit tests for:

- Normal cases
- Boundary cases (for each equivalence class)
- Exceptional cases

As a general rule, every behaviour described in the specification for that class should be tested for.

After implementation, your team must then come back to this section and create **glass box** unit tests. This time the goal is to complement your black box tests with tests that attempt to achieve full **code coverage**. (You can use IntelliJ IDEA to evaluate code coverage of your unit tests using 'Run with Coverage.') You want to create tests that are deliberately designed to cover as much code as possible.

As with the other stages, your unit tests are not expected to remain static. Changes to requirements and design will result in you needing to modify your unit tests or create new ones.

Some classes, particularly those that deal with external resources (database, network) may need to be **mocked** for them to be tested. Creating mock classes is part of your unit testing stage, and these do not need to be documented in your detailed design.

Remember the mantra of Test-Driven development: **red**, **green**, **refactor!**

Implementation (10%)

In this stage your team needs to actually create the software requested by the client. Ideally your development should be **agile,** and you should reach this stage early on, getting something very basic working from the beginning. It is not necessary or desirable to develop the entire project at once. Get something very simple working, then expand on it.

- Implement classes that you have unit tests for. Follow the Test-Driven Development approach and write very basic implementations designed to pass the unit tests, then **refactor** them to remove duplication and improve the quality of the code.
- Implement classes that you don't have unit tests for. These include the non-mock versions of the mock classes you created during the unit testing stage (classes designed to interface with a database or network connection), exception classes, GUI forms and so forth.

Remember that you are creating **two** applications (a client program and a server program.) We suggest creating just one project (remember that each class can have its own main() method), but dividing your code up into multiple packages to reflect the application that uses each. For example, you might have a 'client' package, a 'server' package, as well, as a 'common' package containing classes that are used by both.

Your team must make use of Java and the JDK (any version, but we will test with 15), which has all the functionality you need for completing this project. Use of external Java frameworks (e.g. the

Spring framework, JavaFX etc.) is permitted, but you must make sure you cite these and provide detailed instructions in your project for getting these working so that we can build your software.

Integration (5%)

Your team should only really consider this part if you are feeling confident and have already made good progress in the other stages, but part of delivering a software project as a professional software team is delivering an established build pipeline:

- Build scripts (in Ant, Maven or Gradle) to automate building and running unit tests.
- A continuous integration pipeline (in Jenkins or GitHub Actions) to build and test your software automatically when commits are made to the project repository.

Make sure you demonstrate your build script(s) and continuous integration pipeline in your final submission video!

Deliverables

As described in the overview, the client wants you to provide video demonstrations of the in-development project, final software etc. No marks are given for the quality of your presentation or fancy video editing work – it is simply important that the videos demonstrate everything that you have been asked to do for this project. **Note that these videos will be used primarily to mark your assignment** – that is, if you do not demonstrate some particular feature or aspect of your work, **you will not get any marks for it**. This applies to the documents you need to submit, your code, your software while it is running etc.

We suggest you read the CRA (available on Blackboard under Assessment -> Major Project) and consider demonstrating each item in the same order they appear in the CRA, to ensure the marker does not miss anything.

You can record these videos however you wish – a simple recording of the screen with voiceover is sufficient, for example.

Zoom can be used for this (just click to start a new meeting, share your screen, start recording and go). OBS Studio is another popular choice but requires more setup work. Use of a mobile phone camera pointed at the screen is discouraged. Editing and artistic license is not marked; however, we do need to be able to see what is going on in your video.

(If you do not have a dedicated microphone but do have a mobile phone that supports Zoom, you can use this as a workaround for getting your voice into the video. Create a Zoom meeting from your computer, join the meeting on your phone, then screen-share from your computer. You can then talk using the phone while recording video using your computer.)

Note that everyone in your team will eventually need to submit a video, so make sure that everyone gets some experience with the process.

Milestone #1 (Week 8) (5%)

The client is keen to see your progress on the **requirements** and **detailed design** documents by the end of Week 8 (April 30) to check that you are on track. These are not expected to cover everything by this point, but just to represent a good start. If you have some early prototype work on the actual software, this would also be a good time to demonstrate this to the client. You should submit the following:

- Your requirements document as it currently stands.
- Your detailed design document as it currently stands.
- Your current plan for the next 2 weeks (sprint planning). Keep this realistic and achievable; and describe what each team member will be doing in that time.
- A video showing your progress (including showing the documents and your plan for the next 2 weeks.) Maximum length: **4 minutes.**

The client may not have time to read the documents, so make sure **everything is covered in the video**. You can then use this submission to get some useful feedback on how you are going, to check that you are on the right track etc.

The maximum length for this video is **4 minutes**. If you submit a longer video, we will watch the first 4 minutes of it and give you marks based on that. You may want to rehearse what you are going to say and present in the video to keep within this time limit.

Note that if you do **not** submit this milestone by Week 8, you can still submit it later (by the final due date of June 6) and get the marks (5%). However, you will not be able to get feedback in time for it to be useful. Submit by the end of Week 8 if you want feedback.

Milestone #2 (Week 10) (5%)

The client is keen to see your progress after the previous two weeks; and would like to see how your requirements/detailed design documents have evolved in this time (if they have), and in general, how the software is coming along. By the end of Week 10 (May 16) submit the following:

- Your requirements document as it currently stands.
- Your detailed design document as it currently stands.
- Your current plan for rest of the project duration.
- A video showing your progress since the Week 8 milestone (including showing the documents and your plan for the rest of the project.) Maximum length: **4 minutes.**

Once again, the client may not have time to read the documents, so make sure **everything is covered in the video**. Use the video to specifically focus on things that have changed since the Week 8 milestone (i.e. it's not important to cover things you covered in the previous video.)

The maximum length for this video is **4 minutes**. If you submit a longer video, we will watch the first 4 minutes of it.

Again, if you do **not** submit this milestone by Week 10, you can still submit it later (by the final due date of June 6) and get the marks (5%), but this will come at the cost of timely feedback.

Group submission (Week 13)

At long last, the project deadline has been reached. The client is keen to see your results. If the project has reached its completion and does everything the client asked for, congratulations! If not, the client still wants to see what you managed to achieve in the time.

By the end of Week 13 (11:59 PM Sunday, June 6) your team must submit the following:

- Your final requirements document.
- Your final detailed design document.
- Instructions on how to deploy your software (including how the database needs to be set up, any external Java frameworks that you have used etc.).

- Your project directory, including full source code, unit tests, .git directory and so on. Submit this as a .zip file.
- A video demonstrating every aspect of the project. Maximum length: **10 minutes**.

Again, everything needs to be covered by the video. The video should be sufficient to mark your entire assignment, so **make sure you cover everything you wish to receive marks for.** We still require you to submit everything else (so that we can check it if we feel it is necessary) but you will not get any marks for anything that is not covered in the video.

The maximum length for this video is 10 minutes. It may be difficult to cover everything in that time. We recommend:

- Rehearsing what you are going to cover in the video.
- Ensuring that the software you are going to demonstrate (your client and server programs) are already open and running before you start so you do not need to wait for them to load up.
- Consider using some simple video editing software (e.g. OpenShot) to cut down on pauses, make jumping between documents faster etc. after you have finished the initial recording.

Note that the .git directory is **hidden by default**. You may need to configure your environment to show hidden files/directories before you zip your project directory, in order to ensure that this is included.

Individual submission (Week 13) (10%)

In addition to what your group has managed to put together, the client also wants to know what you can do individually, and has asked each member of your group to submit videos. These videos have a maximum length of 3 minutes each and should contain the following:

- The parts of the project you worked on by yourself, the parts that you worked on with others etc.
- A quick scroll through the Git history of your project (e.g. on GitHub/BitBucket, or through the 'git log' command from inside Git Bash, or through 'Show Git Log' in IntelliJ IDEA), focusing on your individual commits.

Note that it is not necessary for your group to collaborate on producing these – the marks awarded for the things demonstrated in these individual videos are awarded **individually**.

Additional Information

Video formats and sizes

Make sure you submit your video files in a format that we can use. Generally speaking, if VLC can open your video, you should be good. The videos created by Zoom and OBS Studio should not have any problems, but if you use some other unusual software there may be an issue.

Your video files should also not be too large. Zoom tends to create nice small video files, while OBS Studio (on default settings) creates pretty large ones. If you are using OBS Studio, it might be desirable to re-encode your videos with a tool like Handbrake (experiment with the quality settings.) Try to aim for 50mb or less.

Another approach you can take is to upload your video to a video sharing site like YouTube (prefer 'unlisted' to 'public' when uploading your video), or a reliable file-sharing service like DropBox or Google Drive, and then submit the link.

Group vs Individual marks

The standard mark breakdown for this assignment is 50% group marks, 10% individual marks. Group marks are awarded to the entire group (everyone will receive the same mark) while individual marks are awarded individually. The group marks come from the Milestone #1, Milestone #2 and Group Submission items above (worth 5%, 5% and 40% respectively), while the individual marks come from your individual submission video (worth 10%.) This is how the assignments will normally be assigned marks.

However, under certain circumstances, **different members of the group may receive a different group mark to each other.** The main reason this might happen is if certain members of the group did not contribute to certain stages of the project (e.g. did not contribute to the detailed design, did not write unit tests, did not write code.) In that case those group members will receive a group mark of 0 for those sections (we cannot give you marks for things you did not do.)

Now, we understand that occasionally group dynamics can turn ugly and things can happen, such as one group member taking over control of a certain part of the project and not allowing others to contribute. We hope this does not happen to your group, but in those circumstances, the solution is to do work on these sections (reports, unit tests, implementation code etc.) and then submit your work separately along with your individual contribution video and an explanation of why you are doing so, just to show that you are capable of doing the work and therefore are entitled to receive the group's mark.

Academic integrity

The process we will be using to ensure the integrity of this assignment is a simple one. At the end of the semester, with everything submitted, we will take everything that has been submitted (all your documents, source code etc.) and throw it into various tools for detecting duplicates, just as Stanford MOSS. If we find teams have submitted substantially duplicate work (either to another team or to material found online) they will be automatically referred to the Faculty Academic Misconduct Committee.

Here are some tips on how to protect yourself:

- If you find some nice code online that does what you want (and there are no legal problems associated with you using it), e.g. on StackOverflow, feel free to copy and paste that into your project. But make sure you put in a comment above the code saying where you got it from. If the code is from StackOverflow, link the StackOverflow post. Cite **everything** you use from somewhere else.
- If you want to work with people who are not in your team, that is okay! You can discuss ideas, discuss requirements, talk about different ways of doing things. However, your documents, source code, software etc. must be kept within your team and you must not submit work that was not created outside your team. In addition, do not record your demonstration videos using another group's software.
- If you have your code in an online repository provider like GitHub or BitBucket (this is **highly recommended**), make sure that you have it set to **private** so that others outside your team cannot see it. Otherwise, unscrupulous individuals can submit your code and you may be penalised by the Misconduct Committee even if you can prove it was your code originally.
- Don't make your repository public immediately after the due date. Last year I had a situation where one student made his repository public after the due date and another student, who was given an extension, downloaded it and submitted it themselves. Wait a few months first. If an employer wants to see your work, you can just give them access to it individually.
- Do not upload your assignment or the assignment specification to any public external sites, including but not limited to StackOverflow, Chegg, social media etc.

Appendix A: Client Project Description

To Whom It May Concern,

RE: Electronic Asset Trading Platform

We have a great many different types of resources that need to be efficiently distributed and shared across the organisation (computational resources, hardware resources, software licenses etc.). After trying a few approaches that we have determined are not satisfactory, we have settled on the following approach, and we want your team to build us a software platform to help us implement it:

We require your team to implement a virtual trading platform for these resources. The idea is that organisational units will be given a budget of a certain number of electronic credits. They can then use those credits to buy access to resources from other organisational units. So, for example, our compute cluster division might sell individual CPU hours for credits, and then take the credits they get and use that to buy other things they need. We do not know, in advance, exactly what assets will be traded on this platform – you should create a flexible enough system that we can add new asset types to.

We require the software platform to facilitate trades via a marketplace model. Instead of having two organisational units agree to trade synchronously, organisational units who want to **buy** something will put in a BUY order, for a certain quantity of a particular commodity at a particular price (e.g. BUY 100 CPU hours at 10 credits each). Organisational units wishing to **sell** something will put in a SELL order, again, for a certain quantity at a certain price (e.g. SELL 500 CPU hours at 10 credits each.) Your software will periodically check and reconcile all outstanding trades. In this particular example, 100 CPU hours will be sold to the first organisational unit for 10 credits each. This will eliminate the first organisational unit's BUY order entirely, while the second organisational unit's SELL order will be reduced to 'SELL 400 CPU hours at 10 credits each'. Trades will not happen if the terms are unacceptable to either party (e.g. more expensive than the buyer is willing to pay, or less expensive than the seller is willing to accept.) If there is some slack in two compatible trades (e.g. 'BUY 10 widgets for 20 credits' and 'SELL 10 widgets for 15 credits') the transaction should be carried out using the lower price (which will always be the selling price.)

We require this to be implemented with a client-server model, where we run 1 server (which keeps track of every organisational unit's assets, credit balance and all trades.) and clients connect to this server to list trades. There should be no (artificial) limit to the number of commodities in the database, no limit to the number of trades that can be listed and no limit to the number of users using the system. We need everyone to have their own usernames and passwords which the user will enter into the client when logging in, to ensure only authorised users from each organisational unit are able to trade.

We have included some user stories from people in the organisation who will be using this software, so you can get a feel for what we want to see. Best of luck!

Regards,

The Manager

Appendix B: User Stories

As a user, I think it’s important to have a nice, friendly GUI interface, not a dusty old command line interface.

As a user, when I am thinking about listing a buy or sell offer for an asset, I want to be able to see what current BUY and SELL offers are currently listed, so that I do not greatly overbid/underbid.

As the leader of an organisational unit, I need all of the members of my team to have access to this system, through their own individual usernames and passwords. I need them all to be trading using the credit balance and assets of the organisational unit they are part of.

As a user, sometimes after adding an offer I might decide that I want to remove that offer, perhaps to relist it again at a different price. I should be able to see all the currently standing offers from my organisational unit in the database and be able to selectively remove them.

As the leader of an organisational unit, I do not want to accidentally put in a BUY offer for more credits than my organisational unit has, or a SELL offer for more of a particular asset than my organisational unit has. We do not want our organisational units to risk going into debt. The system should check for this and stop us from listing more than we have. For example, if I have 50 widgets and I have a SELL offer for 30 of them, I can only create a SELL offer for 20. I will have to cancel the first offer if I want to create another for more.

As a member of the IT Administration team, I require that only we (the IT Administration team) have the ability to create new organisational units. We should also be able to edit the number of credits they have, edit the number of each asset they have and so forth. We should also be able to add new asset types to the database, as well as add new users and assign them passwords and assign them to organisational units. We need to have special types of user accounts that can do this from the GUI client. Members of the IT Administration team should also be able to add new users with the same level of access, so we can give it to anyone new that joins the IT Administration.

As a user, it would be nice if I could change my own password without having to ask the IT Administration team to do it for me.

As a user, when I go to buy or sell an asset, I want to be able to see the price history of that asset – what it has sold for in the past. Actually, it would be nice if you could show this on some kind of graph (with the date on the X axis and the price on the Y axis), like Google shows when you type in a stock name, so that I can see the history of trades of that asset and get a feel for whether the price of something is going up, going down or holding steady. This may be some work, though, so it’s not that important.

Appendix B: User Stories (cont'd)

As a user, when I have the software open and a trade involving my organisational unit is reconciled, I would like it to show a little message somewhere. Nothing too obnoxious, but it's nice to know when a trade is fulfilled.

As a systems administrator, I would like the client to read from some kind of configuration file to get the server IP address and port to connect to. The server should do the same thing to get its port number. Reason is, I may have to move the server to run on a different machine at some point, and I think this is the easiest way to get the new configuration information out to everyone.

As a systems administrator, I require the following things to be stored in a MariaDB or PostgreSQL or SQLite3 database on the server, so they are all kept in one place and they can be backed up easily:

- User information (username, password, account type, organisational unit)
- Organisational unit information (organisational unit name, credits, assets and the quantity of each asset)
- Asset types (asset names)
- Current trades (BUY/SELL, organisational unit, asset name, quantity, price, date)
- Trade history (same as above)

As the head of IT Security, I do not want plaintext passwords being sent over our network. At least hash the password before sending it over. In the same way, I don't want to see plaintext passwords in the database either.