Inf2-SEPP 2020-21

Coursework 3

Creating a software design for a COVID-19 shielding food box delivery system

1 Introduction

This coursework builds on Coursework 1 on requirements capture and Coursework 2 on design. As needed, refer back to the Coursework 1 and Coursework 2 instructions.

This coursework is split into 2 practical parts:

1. A Software Engineering part (Sections 3-4), including tasks and self-assessment. The aim of this part is to implement and test the food box delivery system.

2. A Professional Issues part (Section 5) including tasks and self-assessment.

Both of these parts are assessed summatively (for final marks and feedback).

2 High-level self-assessment instructions

The self-assessment in each part will be part of your mark. For the self-assessment, you should consider to what extent your have met the assessment criteria for that part of the coursework. For each criterion, write one paragraph (a minimum of 2 sentences) discussing how well you think you have met that criterion and why (i.e., strengths and weaknesses of your solution). Moreover, we encourage you to reflect on things like: your experience with the approaches used, progress and work within your team, difficulties you encountered and how you addressed them, parts that you are particularly proud of and why, parts that you are not happy with and you could further improve, how you could improve them. For all of these, you can reference other parts of your solutions to avoid repetition.

Important! You should make a real effort to be reflective (especially for the criteria where you have more to say), as well as honest, here. Please note that only making statements about the criteria without justifications (e.g., "We identified stakeholders excellently well") is not acceptable and will result in no credit for this part.

For both parts of this coursework, the first deadline is the only and final deadline. Therefore, we provide you in the appropriate self-assessment sections with a high-level marking scheme corresponding with the grade intervals of the Extended Common Marking Scheme which we will use to compute your mark for this part.

3 Updates to your SE work

3.1 What you will need to know

In order to carry out this coursework you will need to know how to run a Flask server using Python, create and run tests using JUnit (version 5) and how to use *interfaces* in Java. If you are not comfortable with either of these topics you should review resources online[1] [2] [3] [4].

3.2 Design Details

This section provides some extra directions which further restrict how you should implement the system to help you in this implementation and to make sure you can attempt all the sections of this coursework.

3.2.1 Client Server Architecture

It has been decided to implement the system in a distributed fashion, where the Scottish Government would be hosting a server with the main logic of the system. An initial attempt has been made in order to test a potential implementation of the ordering system on the server. You may see the current state of the OrderingSystem [5] and run it on your machine (at the commandline):

```
export FLASK\_APP=server.py && flask run --host=0.0.0.0 --port=5000
```

which would create a local server (localhost:5000) with an ordering system instance.

[1] https://flask.palletsprojects.com/en/1.1.x/
[2] https://www.vogella.com/tutorials/JUnit/article.html,https://bit.ly/3clJLs6
[3] https://bit.ly/38t1ltn
[4] https://www.w3schools.com/java/java_interface.asp
[5] https://github.com/mocialov/sepp

Mandatory Classes for your Model Your design should contain at least the following classes: ShieldingIndividual ClientImp , CateringCompany ClientImp , Supermarket ClientImp . Skeletons of these classes have been provided, however, you will need to supply implementations of their methods and you will need to introduce new attributes and methods. You will also need to introduce many of your own classes in order to realise your object-oriented design.

- The ShieldingIndividual ClientImp class represents the shielding individual, which should be able to hold its personal information.
- The CateringCompany ClientImp class represents the catering company activities, which should include updating an order and registering with the system. The class should interface with the ordering system, running locally as a separate entity. The application programming interface (API) is provided below.
- The Supermarket ClientImp class represents the supermarket activities, which should include registering with the system.

3.2.2 Mandatory Classes for the Clients

For the purposes of the coursework, the skeleton implements the following 3 client-side implementations, which should interface with the ordering system (at the server side): The application programming interface (API) is provided in the Section 3.2.3.

- The ShieldingIndividualClientImp class represents the shielding individual's application. This class should implement the interface ShieldingIndividualClient, which in turn extends the inteface ShieldingIndividualClientEndpoints.
- The CateringCompanyClientImp class represents the catering company's application. This class should implement the interface CateringCompanyClient, which in turn extends the inteface CateringCompanyClientEndpoints.
- The SupermarketClientImp class represents the supermarket's application. This class should implement the interface SupermarketClient, which in turn extends the inteface SupermarketClientEndpoints.

The methods in each of the {X}Endpoints (where X one of ShieldingIndividualClient, etc.) interfaces should implement the remote communication with the server via the corresponding endpoint (see Section 3.2.3). The rest of the methods in each {X}Client interface are there to allow you to drive the behaviour of each client and also to query their internal states for unit and system-level tests.

Update: There were also individual classes X, that could be used as a guide to encapsulate information in your object model. There are now removed, so it is up to you to decide how and where to store that information.

For example,

```
public boolean
changeItemQuantityForPickedFoodBox(int itemId, int quantity);
```

should change the quantity of a food box item from the user picked food box, before placing an order.

Lastly, each of these applications is driven by their corresponding JUnit test files (e.g., ShieldingIndividualClientTest.java for ShieldingIndividualClientImp).

3.2.3 Server API

The API allows one to specify a communication contract between multiple entities. The API for the ordering system is:

- /registerShieldingIndividual?CHI=x', where 'x' is a CHI number of a shielding individual. This such request returns ~~'registered new'~~ or 'already registered' if the individual has already been registered in the ordering system. The details, such as name, surname, phone number, and the post code are currently collected from fake PHS class that generates random numbers/letters.
  These details are returned upon a new registration.

- /showFoodBox?orderOption=catering&dietaryPreference=x', where 'x' is a dietary preference from the list ['none', 'pollotarian', 'vegan'] as these are the diets currently recorded in a collection of the pre-defined food boxes (see food_boxes.txt)

- placeOrder?individual_id=x
  &catering_business_name=z&catering_postcode=w ~~&dateTime=y~~',
  where 'x' is the registered CHI, 'z' is the catering company's name to fulfill the order and 'w' is the postcode of the catering company.
  ~~and 'y' is the requested delivery date and time~~. The placeOrder is a HTTP POST method and accepts the data in the JSON format. The food items available for the purchase at this moment are: cucumbers, tomatoes, onions, carrots, beef, pork, chicken, bacon, oranges, apples, avocado, mango, cabbage, and cheese. The request returns an order ID.

```
{"contents": [{"id":1,
"name":"cucumbers",
"quantity":1},
{"id":2,
"name":"tomatoes",
"quantity":2}]}
```

Listing 1: HTTP POST placeOrder endpoint JSON data example

- /editOrder?order_id=x ~~&dateTime=y~~', where 'x' is an order ID from placeOrder

~~and 'y' is edited delivery date and time~~. The editOrder is a HTTP POST method and accepts the data in the JSON format. The JSON format is exactly the same as that of the placeOrder. The only restriction is enforced is that it is only possible to reduce the number of items in the ordered food box. This endpoint returns 'True' or 'False' depending on whether the order amendment has been successful or not.

- /requestStatus?order_id=x', where 'x' is the order ID. The end point returns the order status, which can be: 0 if the order has been placed, 1 if the order is packed, 2 if the order has been dispatched, 3 if it has been delivered and 4 if it has been cancelled. -1 is returned only if the order is not found.

- /registerCateringCompany?business_name=x&postcode=y', where x is the name of the business, and y is the post code of the business. The end point returns 'registered new' or 'already registered' if the business has already been registered

- /cancelOrder?order_id=x', where 'x' is the order ID. The end point returns 'True' if the order has been cancelled and 'False' if the order has not been cancelled. Failing to cancel the order can be either due to the fact that the order has been delivered or that it is already cancelled.

- /registerSupermarket?business_name=x&postcode=y', where 'x' is the name of the business, and 'y' is the post code of the business. The end point returns either 'registered new' or 'already registered'

- /updateOrderStatus?order_id=x&newStatus=y', where 'x' is the order ID and 'y' is the new status (packed/dispatched/delivered). This method should be implemented by the catering company in order to change the order status to packed, dispatched, or delivered. The end point returns 'True' or 'False' depending on whether the status has been changed or not

- /recordSupermarketOrder
  ?individual_id=x&order_number=y
  &supermarket_business_name=z&supermarket_postcode=w,
  where 'x' is the CHI number, 'y' is the order ID, 'z' is the supermarket business name and 'w' is the supermarket's postcode. This method should be implemented by the supermarket client in order to record an order via a supermarket. The end point returns 'True' or 'False' depending on whether the order has been recorded or not.

- /updateSupermarketOrderStatus?order_id=x&newStatus=y', where 'x' is the order ID and 'y' is the new status (packed/dispatched/delivered). This method should be implemented by the supermarket client in order to change the order status to packed, dispatched, or delivered. The end point returns 'True' or 'False' depending on whether the status has been changed or not

- /getCaterers. This method should be implemented by the shielding individual client in order to assist with the client's decision on finding the closest catering company. The end point returns a list of information about the catering companies in the system
- /distance?postcode1=x&postcode2=y, where 'x' and 'y' are post codes of the format "EH99_2DR", "EH" should always be present since we are only assuming Edinburgh locations. This method should be implemented by the shielding individual client in order to assist with the client's decision on finding the closest catering company. The end point returns a floating point corresponding to the distance between the two provided postcodes.
- /getCateringCompanyforOrder?orde_id=x, where 'x' is ther order number. This method could be implemented by the shielding individual client in order to get the catering company that is fulfilling the order. There is no client endpoint in Java that enforces this at the moment. Use it only if you absolutely require it.

3.3 Using approaches, tools and agile practices

This coursework is a small-scale opportunity to try out approaches and software tools that have been mentioned in the lectures starting with Week 6. For this coursework (only), please assume that you are using an *agile software development process*. As a result, you should pay attention to the *agile principles* from Lecture 2, and you are encouraged to use any applicable *practices* that have been proposed by the agile processes described in Weeks 8 and 9 of the course. Here, you do not need to stick to one process, but you are free to mix and match practices from different processes that would make sense in your view for the system that you are developing and your team. Please be careful about practices that are dependent on one another! Furthermore, you are encouraged to try out software tools that are recommended for your chosen practices.

Apart from the software tools that are recommended for working in an agile process, we also recommend that you experiment with other tools that may be helpful for your development.

Overall, apart from the required tools mentioned in the next section, you could try out for example GitHub and the Git version control system [6], a test coverage tool like the default one provided by IntelliJ IDEA[7] or one of its alternatives (if you decide to use this IDE), a bug tracking tool like Trac[8] or JIRA[9], a static analysis tool like SpotBugs[10] or Infer[11], a project management tool like Trello[12], and/or any tools for support with agile like JIRA.

[6] see the Git and GitHub tutorial from the Learn course under Resources
[7] https://www.youtube.com/watch?v=QDFI19lj4OM
[8] https://trac.edgewall.org/
[9] https://www.atlassian.com/software/jira/bug-tracking
[10] https://spotbugs.github.io
[11] https://fbinfer.com
[12] https://trello.com/en-GB

It is up to you how many and which approaches, tools and agile practices you use, and it should be based on your judgement of what could be beneficial for developing this system. Some of these may prove to be real life savers! The effort you put in will also influence the quality of the discussion which you can make in section ... and your assessment for this section.

In the labs from this coursework, starting with the week 10 labs, the demonstrators will be available to play the role of your customers, and so please ask them to 'take the customer hat' involve them in discussions about their requirements and the validity of your solution. They will all be instructed about what they can advise in these roles, and so do not worry about which demonstrator you approach. Furthermore, please be aware that while playing this role they will be non-technical, and so you will need to pay attention as to how you present and explain things to them.

4 Your Software Engineering Tasks (worth 68.75% of the final mark for this coursework)

There are several concurrent tasks you need to engage in. These are described in the following subsections.

Bear in mind that it is not enough to submit code claiming to implement your system without testing and other documentation demonstrating its effectiveness. As part of this assignment you are asked to implement system tests demonstrating how your system implements the key use cases of the system, and **these tests will be used as one of the primary means of marking this coursework**.

4.1 Construct code

Because of the requirement to now use a Client-Server architecture and the skeleton code we provide for this assignment, the design of the system might have changed from your coursework 2. Please keep your design from coursework 2 as for a monolithic system as originally intended, and only consider updating parts of it which you now find could not work even for a monolithic system, or which the markers have criticised. Make any changes to your design in coursework 2 in green.

Your implementation needs to cover the 8 use cases listed in Section 4.2.

Follow good coding practices as described in the lecture. You should attempt to follow the coding guidelines from Google at

`https://google.github.io/styleguide/javaguide.html`

A common recommendation is that you never use tab characters for indentation and you restrict line lengths to 80 or 100 characters. You are strongly recommended to adopt both of these recommendations because it is good practice and, particularly, in order to ensure maximum legibility of your code to the markers. A good practice is to adopt a consistent formatting style either via your IDE or a standalone tool (which can be used via an IDE) such as:

http://astyle.sourceforge.net/

Add class-level, method-level and field-level Javadoc comments to your classes. For your methods you only need to include @param and @return tags as well as a summary of the role of the method. Browse the guidance at

https://www.oracle.com/technetwork/java/javase/tech/index-137868.html

on how to write Javadoc comments. Follow the guidance on including a summary sentence at the start of each comment separated by a blank line from the rest of the comment. This summary is used alone in parts of the documentation generated by Javadoc tools.

Be sure you are familiar with common interfaces in the Java collections framework such as List, Set, Map, and Queue/Deque and common implementations of these such as ArrayList, LinkedList, HashSet, TreeMap. Effective use of appropriate collection classes will help keep your implementation compact, straightforward and easy to maintain.

You are allowed to assume that the UI has already performed some input validation before calling the methods of your implementation. You should, however, include assertions in at least some of your methods as a means of catching faults and invalid states. You only need one or two assertions to show you know how to use them, but you may want to include more to help you develop and test your system.

4.2 Create system-level tests

Your are expected to use JUnit 5 to create system-level tests that simulate the scenarios of each of the use cases you implement.

Please write these tests within a SystemTests test class.

To get a good mark for this coursework you must test at least the use cases in this section since your tests will provide one of the main means for assessing your implementation. Even if you have implemented a feature, you may not get full marks if you have not tested your system sufficiently to demonstrate your system successfully implements the feature.

You need to provide tests to cover the following use cases that utilize all the available server endpoints (see Section 3.2.3):

1. **Register Supermarket**
2. **Register Catering Company**
3. **Register Shielding Individual**
4. **Place Food Box Order**
5. **Edit Food Box Order**
6. **Cancel Food Box Order**

7. **Request Order Status**

8. **Update Order Status**

You are expected to add enough systems tests to completely test the above use cases.

Few tests are able to test single use cases by themselves. In nearly all cases, several use cases are needed to set up some state, followed by one or more to observe the state. You may want to include tests which perform more than one use cases in turn to check the integration of the modules of the system.

Do not run all your tests together in one large single test. Where possible, check distinct features in distinct tests. Also, when a test involves exercising some sequence of use cases and features, try to arrange that this test is some increment on some previous test. This way, if the test fails, but the previous test passes, you know immediately there is some problem with the increment.

You are encouraged to adopt a test first approach — the tests are just as important a part of the assignment as is your implementation itself. Make use of the provided tests to help you develop your code, and when tackling some feature not already tested for, write a test that exercises it before writing the code itself.

Include all your additional tests as JUnit test methods in the corresponding test files for each client (e.g., ShieldingIndividualClientTest.java for ShieldingIndividualClientImp).

4.3 Unit testing

As well as system tests you may find it useful to create unit tests for specific classes. These should focus on checking whether the implementation of each specific class is correct, rather than considering the correctness of other modules of the system. The tests for a class named MyClass should be in a file named `TestMyClass.java` and should use JUnit 5 assertions and annotations similarly to the examples provided.

For your coursework submission you are only required to provide such unit tests for the provided ShieldingIndividual ClientImp, CateringCompany ClientImp, Supermarket ClientImp classes in the provided files (in addition to the system tests).

4.4 Code review (optional)

This task is optional, but a great opportunity to get some feedback on a small part of your code- your `CateringCompany ClientImp` class and its unit tests- and exercise doing a code review which is a useful skill for a software engineer. Moreover, it contributes towards your engagement for this course.

If interested, please join one of the Week 10 or 11 labs as a team (i.e. both team members), having prepared in advance the `CateringCompanyClientImp` class and its tests. There, one of the lab demonstrators will first ensure that you have the necessary code and that it passes its tests (preconditions for the code review). Then, the demonstrator will pair you up with another team and you will be asked to exchange with them your `CateringCompany ClienImp` class and its unit tests (and only them), and work

in separate breakout rooms to assess each other's work on them and write your notes in a text file, for 20 minutes. At the end of your allocated time, you will be asked to exchange your notes with the other team, and also submit them in an area on Learn which will be dedicated to the code review (see 'How to submit' section), all under the supervision of the demonstrator.

You will be provided with some guidance on what constitutes a good code review in due course by email.

4.5 Experience with the above tasks and with approaches, tools, agile processes

Start this section by describing how you have organised the work for the above tasks, in terms of:

- Using any of the approaches to software development from the lectures starting from Week 6 which you have used
- Using any tools which you have used from the ones we studied (version control, test coverage, bug tracking, static analysis, project management tools).
- Using the agile practices that you have used

Then, comment on your experience with using each. To this end, you could use the following questions as a guide:

- What approach/tool/agile practice have you used?
- What was your reasoning behind choosing this approach/tool/practice? Here, you can also include the reasoning for not choosing an alternative.
- How did the approach/tool/agile practice work for you?
- What would you conclude are the advantages and disadvantages of the approach/-tool/agile practice?
- Would you do anything different next time you develop a system?

In your description of tools and how they helped you (or not), please include some screenshots of your use of those tools.

4.6 How the Client-Server Architecture would impact design

Include in this part a section summarising how the change to a Client-Server Architecture would impact your design from coursework 2. Would your original design or this updated design be better, and why?

In that coursework, keep the design as monolithic (do not make it match with the new architecture), and make only changes to this design if you find anything that could not work there given your experience with the implementation, and considering marker comments. Make any such changes in green in the coursework 2 document.

4.7 Quality attributes

Reflect on the following quality attributes:

- **Security OR privacy**: What kind of security or privacy attacks could happen in our system (try to find at least 2)? What would you do to address them at the level of the design and implementation?
- **Reliability OR availability**: What kind of measure(s) would you use to test the reliability or availability of our system? Why do you think it/they are a good choice?
- **Usability**: After finishing the system back-end together with your team, you are tasked with designing the user interface of a website as well as mobile app for it for shielding individuals (who are ill, elderly, potentially frail, people). State 2 principles of user interface design which would be very important for you to consider, explain why they are particularly important in this context, and briefly describe what you would do to address them in the user interface.

4.8 Software Engineering Task Self-assessment

Please read the high-level instructions for self-assessment from Section 2.

The following are the criteria you need to self-assess on regarding the Software Engineering tasks by writing one paragraph (at least two sentences) for each. For the questions on results and coverage of testing, please provide any screenshots you may find appropriate as proof.

1. Use of Java programming constructs within the provided skeleton code and respecting what can and cannot be altered to construct the functionality of the required 8 use cases for the food box system.
2. Quality of the object-oriented design underlying the implementation.
3. Quality (in terms of readability, maintenance) of the produced code and its documentation.
4. Use of JUnit 5 to construct system-level tests for the 8 provided use cases.
5. Choice of system-level tests (you should try to cover a good mix of classes of inputs: frequent correct inputs, but also unusual/incorrect inputs and edge cases).
6. Results and coverage (of all the scenarios of the 8 use cases, of the code) of the system-level tests.
7. Use of JUnit 5 to construct unit-level tests for at least the 3 required classes.
8. Choice of unit tests (you should try to cover a good mix of classes of inputs: frequent correct inputs, but also unusual/incorrect inputs and edge cases).

9. Results and coverage (of all requirements which are the responsibility of each class, of the code of each class) of the unit tests.

10. Effort and understanding at trying out approaches to software engineering, tools and agile practices for this coursework.

11. Quality of the reflection of the experience with each of these approaches to software engineering, tools, and agile practices (in line with the questions provided).

12. Application of knowledge of good object-oriented design to discuss the impact of using a Client-Server architecture on the design and the design's quality as compared to the design obtained in Coursework 2.

13. Application of knowledge of security or privacy to discuss possible security or privacy attacks to the system, and propose possible solutions.

14. Application of knowledge of reliability and availability to discuss the choice of measures for testing reliability of availability in the system.

15. Application of knowledge of usability to choose 2 appropriate principles of user interface design which would apply to a web and mobile app interface to the system, and propose how they should be applied to the user interface design.

16. Effort at this self-assessment

You will receive a mark according to the following marking scheme and respecting the Extended Common Marking Scheme (this is only for your information, nothing to do here):

- Pass (40+): An implementation including plausible code and tests for a significant part of the assignment, even if the code does not work correctly (according to your tests and/or ours). Some effort at improving the quality (in terms of readability, maintenance, documentation) of the code. Only some of the SE concepts are correctly understood and applied to the case study, as shown by tasks 5-7.

- Good (50+): An implementation including working code and reasonable tests for most parts of the assignment, even if there are small bugs or omissions (according to your tests and/or ours). The code is reasonably well structured (in terms of good object-oriented design). A good amount of effort at improving the quality (in terms of readability, maintenance, documentation) of the code. A good amount of the SE concepts are correctly understood and applied to the case study, as shown by tasks 5-7. Additionally, some effort at trying out and reflecting on approaches to software engineering, tools and agile practices.

- Very good (60+): An implementation including working code and good and passing tests for all parts of the assignment, even if the code contains small bugs according to our tests (only). The code is well-structured (i.e. good object-oriented design

underlying it) and of high quality (in terms of readability, maintenance, documentation). Most SE concepts are correctly understood and applied to the case study, as shown by tasks 5-7. A good amount of effort at trying out and reflecting on approaches to software engineering, tools and agile practices. Additionally, the self-assessment is honest but only focuses on strengths and weaknesses.

- Excellent (70+): An implementation including working code and passing tests for all parts of the assignment, with no significant bugs. The code is very well structured (i.e. good object-oriented design underlying it), and of very high quality (in terms of readability, maintenance, documentation). All SE concepts are correctly understood and applied to the case study, as shown by tasks 5-7. A good amount of effort at trying out and reflecting on approaches to software engineering, tools and agile practices. The self-assessment is thorough and reflective, going beyond the discussion of strengths and weaknesses

- Excellent (80+): Marks in this range are uncommon. This requires faultless, professional-quality design, implementation and testing, in addition to very well informed and highly reflective answers to the open questions in tasks 5-7 and self-assessment, and a demonstration of an exploratory and reflective stance in trying out approaches to software engineering, tools and agile practices.

4.9 Getting started

You will need access to a Unix command line with the make and git commands (as are available on DICE) and Java 8. Make sure you are able to get the skeleton code and run the provided sample tests as soon as possible so you can get help with any issues in the labs or on Piazza.

Update: All updates in the skeleton code are a single separate commit (i.e., you can use `git diff`) and also searchable since they are titled with the comment:

```
// **UPDATE**
```

Update 2: The second set of updates are also a separate single commit and searchable in the source with the comment:

```
// **UPDATE2**
```

Start by cloning the skeleton code repository:

```
https://github.com/compor/uoe-sepp2021-coursework3-skeleton
```

You can either create your own (**private!**) fork of this repository, download it as a zip file, or clone it using the command,

```
git clone https://github.com/compor/uoe-sepp2021-coursework3-skeleton.git
```

This contains the sub-directories `src/main` for the source and `srs/test` for the JUnit tests.

The repository comes with a simple Gradle file `build.gradle` allowing you to compile the code and run tests from the commandline. Moreover, there is a `settings.gradle` which configures the root directory for the project (you shouldn't need to edit this unless you rename the directory of the repository).

First `cd` to the checked out repository. Then run:

```
gradle build
```

This command also executes the tests. To perform a build without running the tests run:

```
gradle build -x test
```

It is also possible to import this code into an IDE and run the tests within it.

The code is provided in the form of a GitHub repository. You may find it useful to use Git to collaborate with your teammate, but **any repositories you create must be private**.

For more information on installing and configuring Java, Gradle, IntelliJ IDEA, please see the Development Tools instructions from the Learn course page- Coursework- under Coursework 3.

4.9.1 Working with JSON

The transfer of data between the client and the server is using the JSON format to marshal and unmarshal data[13]. For this task we utilize the GSon library[14].

A comprehensive tutorial at

`http://tutorials.jenkov.com/java-json/gson.html`

explains how to use the library.

Moreover, we have provided a dummy and incomplete implementation of the ShieldingIndividualClient interface in DummyShieldingIndividualClientImp which is exercised by the provided test at DummyShieldingIndividualClientImp. This dummy example implements the showFoodBox endpoint to help you with the use of the GSon library.

4.9.2 Working with the HTTP protocol

The client-server communication uses the HTTP protocol. For your convenience we have provided a basic implementation of the required communication primitives in the class ClientIO. You will need to use this functionality in your implementation.

[13] `https://en.wikipedia.org/wiki/Marshalling_(computer_science)`

[14] `https://github.com/google/gson`

Moreover, since we are providing you with a server side implementation which you can run locally, it is useful to be able to configure the URL address of the server endpoints to you localhost. For this, you can edit provided `client.cfg` (under `src/main/resources/`) properties text file to adjust it. We have also included 2 presets under the file names `client-local.cfg` and `client-remote.cfg`, which correspond the localhost and provided remote server URL addresses.

4.10 What can and cannot be altered

- Do **not** alter the provided ClientIO and any of the provided Java interface files.
- The coursework requires you to add Javadoc comments to the provided client classes. You should provide the necessary methods but should not change the interface of the constructors or the provided methods.

It is expected you will introduce several new classes in addition to the provided classes. While it is possible to produce a functioning implementation with very few classes, such an implementation would have very poor object-oriented design.

5 Professional Issues Tasks (worth 31.25% of the final mark for this coursework)

As a team, write a short essay (we will refer to them as 'blog posts') on each of the topics below.

Each blog post should be a maximum of 500 words (this amounts to about one side of A4), and good submissions will not be far under this.

This article gives good advice on how to structure a short essay:

`https://www.wikihow.com/Write-a-Short-Essay`

Remember that reading the first chapter of "A rulebook for arguments" should help here too.

Good essays will:

- Clearly choose a main position
- Clarify important details with reference to other sources
- Consider the terms used and define them where necessary
- Justify arguments with reference to course materials
- Anticipate and address counterarguments

How well you do these things will be the core criteria for marking in your blog posts.

5.1 Topics

Topic 1: "Discuss the extent to which the Boeing 737 Max crashes were a failure of Standards."

The case study is available here: `https://www.bbc.co.uk/news/business-55582496`

This topic is an opportunity to reflect on an event that has already happened, and so lends itself to exploring other sources, as well as reading and referencing others' thoughts on the topic.

Topic 2: "How might Equality or Personal Data concerns be particularly important in the context of your food box system and the wider healthcare domain?"

This question less obviously invites use of external sources, but reference to other cases as examples is still a good way to strengthen an answer.

5.2 Professional Issues Self-assessment

Please read the high-level instructions for self-assessment from Section 1.1.

The following are the criteria you need to self-assess on regarding the Professional Issues blogs by writing one paragraph (at least two sentences) for each (they are the same as for CW1):

1. Answers the question
2. Clarity
3. Appropriate structure
4. Supported by other sources
5. Quality of argument
6. Recognition of counterarguments / alternate views
7. Knowledge and understanding
8. Style and presentation
9. Effort at this self-assessment

You will receive a mark according to the following marking scheme and respecting the Extended Common Marking Scheme (this is the same as for CW1 and CW2, and only for your information, nothing to do here):

- Pass (40+): Essay attempts to address the provided question but is hard to understand and/or makes few clear points.
- Good (50+): Essay is understandable, clarifies key terms, and comes to one or more obvious conclusions. Some course materials and external sources are referenced.

- Very good (60+): Essay has a good structure, flowing between and building upon subsequent points. It identifies possible counter arguments or alternative views and integrates a variety of external sources. The self-assessment is honest, but not very reflective.
- Excellent (70+): Essay reads well and contains well-made arguments which pull together a variety of views and sources. The self-assessment shows extra effort in reflection.
- Excellent (80+): Marks in this range are uncommon. This essay draws the reader in and makes points beyond what would be expected of undergraduate students in Informatics.

6 Some advice (same as for CW1 and CW2)

6.1 Working online as a team

Teamwork is not easy, and this year it is made harder by the fact that we work remotely. However, you can turn this to your advantage if you use your experience as Informatics students, and make use of the wealth of software tools available to help you. Here are some that we would recommend:

1. Microsoft Teams (free through our university) for setting up a team with your colleague, setting up meetings in the calendar, video calls, the chat, editable file sharing, its Tasks By Planner and To Do to organise and split your work.
2. OneDrive or SharePoint under Office365 (free through our university) for storing documents, sharing and working collaboratively on them.
3. The GitHub (`https://github.com/`) online repository and version control system. Please be careful about access permissions (see subsection 6.3).
4. Trello (`https://trello.com/en`) as an excellent (and also free) alternative for splitting up work and recording progress on tasks. We will also use it later in our course.
5. Aww (`https://awwapp.com/`) as an online whiteboard where you can collaboratively sketch your ideas like you would do on paper.

You may want to mention what tools you used for your teamwork in your self-assessments.

6.2 Asking questions

Please ask questions in labs, office hours or on the class discussion forum if you are unclear about any aspect of the system description or about what exactly you need to do. On the class discussion forum, tag your questions using the *cw2* folder for this coursework. As questions and answers build up on the forum, remember to check over the existing questions first: maybe your question has already been answered!

6.3 Good Scholarly Practice

Please remember the University requirement as regards all assessed work. Details about this can be found at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

Please note that we will run a plagiarism checker on your solutions.

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on an online repository like GitHub then you must set access permissions appropriately (permitting access only to yourself or your group).

7 Submission

7.1 For the Software Engineering Tasks in Section 4

7.1.1 Code review (optional)

If you perform a peer code code review in one of the labs by the code review deadline (see Section 8), please save it as a .txt or .doc or .odt document entitled **review** and submit it.

7.1.2 Working code and tests

Produce a .ZIP file of your submission. To do this in IntelliJ, go to
`File - Export- Project to ZIP file`.
Rename this file **submission.zip**.

7.2 Report

Please prepare a PDF (not a Word or OpenOffice document) of your report part of this coursework. The document should be named **report.pdf** and should include **a title page with names and UUNs of the team members who have worked on this coursework**.

How to Submit

As a group, **only one of you** needs to make the submission. If you submit several times, only the last submission will be checked by the markers.

For the compulsory SE part, you will need to submit **both the submission.zip file, and the report.pdf file. Please do so by adding them all in one go**. Ensure you are logged into MyEd. Access the Learn page for the Inf2-SEPP course and go to "Coursework" – "SE Coursework Submission" – "Coursework 3".

For the optional submission of a code review, please submit the review file also on the Learn page under "Coursework" – "SE Coursework Submission" – "Code review".

Submission is a two-step process: upload the file, and then submit. This will submit the assignment and receipt will appear at the top of the screen meaning the submission has been successful. The unique id number which acts as proof of the submission will also be emailed to you. **Please check your email to ensure you have received confirmation of your submission**.

If you do have a problem submitting your assignment try these troubleshooting steps:

- If it will not upload, try logging out of Learn / MyEd completely and closing your browser. If possible, try using a different browser.
- If you do not receive the expected confirmation of submission, try submitting again.
- If you cannot resubmit, contact the course organiser at Cristina.Alexandru@ed.ac.uk attaching your assignment, and if possible a screenshot of any error message which you may have.
- If you have a technical problem, contact the IS helpline (is.helpline@ed.ac.uk). Note the course name, type of computer, browser and connection you are using, and where possible take a screenshot of any error message you have.
- Always allow yourself time to contact helpline / your tutors if you have a problem submitting your assignment.

7.3 For the Professional Issues Tasks in Section 4

Please submit your blog posts for the markers to view as a PDF in 'Coursework" - "PI Coursework Submission" - 'Group Blog for CW3"

8 Deadlines

The deadlines for this coursework are as follows:

- **Code Review Deadline for SE part (optional):**
  16:00, Fri 2nd April

- **Final Deadline** (compulsory with final marks for both the Software Engineering and the Professional Issues parts

  23:59, Sun 11th April EXTENDED!