Inf2-SEPP 2021-22

Coursework 3

Software implementation of an events app during the COVID-19 pandemic

March 14, 2022

This coursework requires team work for SE (see 1), as well as working individually for ProP (see 2).

1 Your SE work (team work)

1.1 Introduction

The aim of this coursework is to implement and test the events app requested by the Scottish government, as well as reflect on professional issues related to it.

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

Important! We will provide method signatures and detailed Javadoc documentation for the system you will be building. You must write code that matches that documentation perfectly. The documentation is based on the requirements from CW1. Moreover, the design that you will work with in this coursework is an increment to the one that you can see in the CW2 solutions. However, *if there is any contradiction between the documentation provided in this coursework with anything from the previous courseworks and their solutions, you should follow the documentation from this coursework. The Javadoc documentation is the main thing that your code should be based on. You should also not* *at any point refer to your own requirements document from your CW1 submission or your own design document from your CW2 submission.*

1.1.1 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 [1], a test coverage tool like the default one provided by IntelliJ IDEA[2] or one of its alternatives (if you decide to use this IDE), a bug tracking tool like Trac[3] or JIRA[4], a static analysis tool like SpotBugs[5] or Infer[6], a project management tool like Trello[7], and/or any tools for support with agile like JIRA.

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 reflection which you make in SE Task 6 and your assessment for this section.

1.1.2 Updates to requirements and design

This section contains some clarifications and minor updates to the system.

Note that for this coursework, you will **not** be required to implement anything regarding maps or directions to the venue, transportation (public or taxis), weather, or how crowded the venue is.

[1]see the Git and GitHub tutorial from the Learn course under Other Resources
[2]`https://www.youtube.com/watch?v=QDFI19lj4OM`
[3]`https://trac.edgewall.org/`
[4]`https://www.atlassian.com/software/jira/bug-tracking`
[5]`https://spotbugs.github.io`
[6]`https://fbinfer.com`
[7]`https://trello.com/en-GB`

- An event can have several different performances, i.e. several different instances of the same event at different times / venues. A single performance of an event is uniquely characterised by its date and time, and an event cannot have more than one performance at any one given date and time (meaning it can't take place at different venues at the same time). When a consumer books an event with multiple performances, they need to provide the date and time of the performance (in addition to the event number and performance number) so that it's clear which performance they want to book. Consumers can only book a single performance at a time.
- When entertainment providers create an event / performance on our system, our system shall notify their system, sending them the event name, identification numbers, and total ticket number.
- Numbers that identify objects (such as a booking number, event number, etc) shall be unique across the entire system. Any new object with such an identification number shall be generated by incrementing the last generated number for an object of that class, starting at 1 for the very first object of a class.
- Regarding entertainment providers registering and logging on to the system (R28 and R29 from the requirements document): There can only be one representative with an account in the system, and that is the account that is used to log in as the entertainment provider. The others representative names and emails are provided only as points of contact outside of the system.
- A government representative shall be pre-registered onto the system before it goes live (and hence does not need a Register functionality implemented). They shall be able to login by using an email and password, similar to consumers and entertainment providers.
- When any user registers, they additionally need to state that they have registered to a payment provider system already and need to provide their email that is attached the payment system. The government representative shall have this email by default from the beginning. This is to make sure our system never keeps any record of bank account details for any user.
- A booking confirmation shall be provided for the whole booking, and not for each requested ticket. This means that booking cancellations shall also cancel all the tickets.
- The booking record shall include the price that was paid per ticket and the entertainment provider's payment system username, such that refunds can be correctly made if the entertainment provider cancels an event.
- Events shall have the status 'ACTIVE' or 'CANCELLED'.
- The system shall record bookings with 'ACTIVE' once the booking is paid, 'PAYMENTFAILED' if there is an error with payment, and 'CANCELLEDBYCON-

SUMER' or 'CANCELLEDBYPROVIDER if/when the booking or relevant performance is cancelled.

- Two different events can be scheduled at the same address at overlapping times.
- The email address used to register users shall be unique in the system (i.e. The same email address cannot be used to register both a consumer and an entertainment provider account), and help identify the users.
- The email address cannot be changed once an account was created.
- For any action, the system shall check: 1) that it was requested by the correct type of user (given their email address); 2) that the user has legitimate rights to any requested booking (i.e. the booking number corresponds to a booking that belongs to them); 3) that provided event/booking numbers correspond to events/bookings that exist on the system.
- Government representatives shall not be able to change a sponsorship decision for a particular request after making a decision.
- When an already-sponsored event is cancelled, the entertainment provider shall be asked to refund the sponsorship they received back to the government by liaising with the payment provider. The system shall disallow cancelling the event if the refund from the organiser to the government sponsor fails. If some of the bookings are not refunded, this is logged, but the event still gets cancelled.
- Entertainment providers should not be able to cancel an event after any performance of that event has started.
- For the Cancel Event use case, entertainment providers shall not have to provide a message to consumers (for simplicity in this coursework).
- Consumers shall be able to search and filter the list of events for specific event qualities, namely COVID preferences.

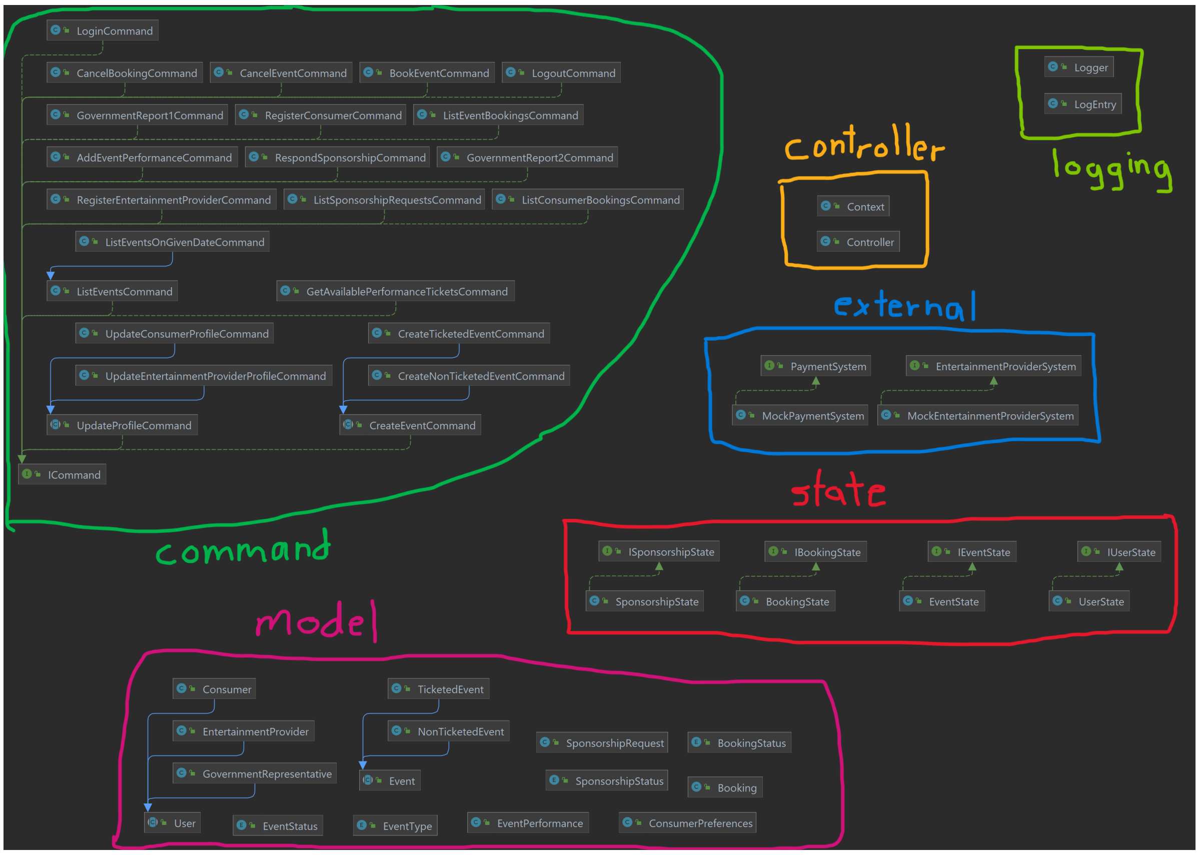
1.1.3 Code specification

The aim of this coursework is to test your ability to implement a large pre-specified piece of software as a team. To that end, we provide a .ZIP archive available here containing the Javadoc documentation of the final project we expect you to deliver. Any code implementation is omitted from the .ZIP archive (other than the method signatures, which are given as part of the Javadoc).

Your task (see Task 1) is to provide the code that matches the given Javadoc documentation.

It may help to see an updated class diagram. Below is one that splits the system into several sections, which are also briefly explained.

- `Controller` - This is the main external interface of this application. It allows executing commands. `Context` is a wrapper around the entire app state. It keeps references to the external systems and to the internal states.
- `Model` - The classes in this section will be used to make single instances of the respective class. For example, a Booking instance will be made whenever a new booking is made, and it will store the information about only that booking.
- `State` - Stores the full information about all the various objects in the app. For example, IBookingState contains information about ALL bookings on the system. This is opposed to the Booking class, which will be used to represent singular instances of a booking.
- `Command` - Contains individual commands that each user of the system can initiate. For example, BookEventCommand allows the currently logged-in user (assuming they are a Consumer) to book tickets for a specific EventPerformance.
- `External` - This includes the systems that are external to our system, but our system still needs to interact with. See section 1.1.4 for details.
- `Logging` - A singleton that can be retrieved from anywhere to add log entries. This helps to figure out what is happening in a sequence of events and how the application state changes over time.



Note that many of the connections from the above class diagram are omitted. To get a more complete version of the class diagram, see the .svg file provided in the doc-files folder of the Javadoc archive available here.

1.1.4 Server API

There are a number of external systems that our app would have to query or send information to at certain times, but your team does not have access to the code of those systems. In order to simulate this without having to deal with networking (which is a topic for another course), we will be using a mock API that specifies a communication contract between our system and the external ones.

In the Javadoc folder provided earlier, this API is provided in the 'external' folder. For example, `EntertainmentProviderSystem.html` specifies the functionality of `Entertainment ProviderSystem`, which is an interface for the entertainment provider's system. Then `MockEntertainment ProviderSystem` is a mock implementation of the `Entertainment ProviderSystem` interface.

1.1.5 Request Booking Records - Alternative implementations

One of the use cases you will be asked to implement is Request Booking Records, where the government asks for specific information about existing bookings or consumers. For this coursework, we will consider two alternative implementations of this use case. Different teams will implement **one** of them or **one**, and may optionally participate in peer review along with another group with the other implementation (see Task 5). Both of these implementations are different from the description provided in the CW1 solution or the CW2 requirements document - you should ignore that outdated description, and instead use one of the following:

If your group number is *odd*: When the government requests records, they provide two sets of dates and times. Your system should return to them all information about the bookings of performances that are between the two date-times and belong to an active and sponsored event.

Do this by providing the implementation of the `GovernmentReport1Command` class. You must also provide Javadoc for this class (use the Javadoc given earlier as an example). You will be provided with a system test (see section 1.2.2) to test your implementation of the class.

Your group should **not** implement the `GovernmentReport2Command` class at all.

If your group number is *even*: When the government requests records, they provide an entertainment provider's name. Your system should return to them all information about the consumers who have active bookings for that entertainment provider's events that are both active and ticketed.

Do this by providing the implementation of the `GovernmentReport2Command` class. You must also provide Javadoc for this class (use the Javadoc given earlier as an example). You will be provided with a system test (see section 1.2.2) to test your implementation of the class.

Your group should **not** implement the `GovernmentReport1Command` class at all.

Depending on your group number, you should implement one of the above functionalities. If you choose to do Task 4 (i.e. reviewing another group's code), then you will be asked to isolate the relevant class and methods and show them to another group.

1.1.6 Development tools

In order to carry out this coursework you will need to know how to create and run tests using JUnit (version 5), how to use interfaces in Java. If you are not comfortable with either of these topics you should review resources online. We suggest the following:

JUnit 5 - `https://www.vogella.com/tutorials/JUnit/article.html`

JUnit 5 - `https://bit.ly/3clJLs6`

JUnit 5 - `https://bit.ly/38t1ltn`

Java interfaces - `https://www.w3schools.com/java/java_interface.asp`

We provide a Development Tools document that specifies the tools required for the coursework, their required versions, and how to install them, accessible here.

1.1.7 Working practices

This coursework is a small-scale opportunity to try out ideas that have been mentioned in lectures. For example, you could try writing tests before code and using the tests to drive your design work. You could also try pair programming, whereby, in a session, one of you programs and the other gives feedback.

You are strongly encouraged to take an incremental approach, adding and testing features one by one as much as possible, always maintaining a system that passes all current tests.

Sometimes it is seen as important to have development teams and testing teams distinct. This way there are two independent sets of eyes interpreting the requirements, and problems found during testing can highlight ambiguities in the requirements that need to be resolved. As you work through adding features to your design, you could alternate who writes the tests and who does the coding.

You may also find it useful to collaborate on your code via a shared git repository[8]. In bigger teams, it will likely be particularly important to do so.

[8] `https://guides.github.com/introduction/git-handbook/`

1.2 Tasks

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 tests demonstrating the correctness of some classes, and how your system implements the key use cases of the system, and **these tests will be used as the primary means of marking this coursework**.

1.2.1 Task 1: Construct code

Construct the code that implements the following use cases:

1. **Log in** (for all users)
2. **Register (Consumer)**
3. **Register (Entertainment Provider Representative)**
4. **Request Booking Records** (See section 1.1.5)
5. **Create Event**
6. **Book Event**
7. **Cancel Event** (only for **groups of 3+**)
8. **Cancel Booking** (only for **groups of 3+**)
9. **Accept/Reject Sponsorship** (only for **groups of 4+**)
10. **Search for Events** (only for **groups of 4+**)
11. **View Event** (only for **groups of 4+**)

In doing so, make sure you closely follow the class model given in section 1.1.3. Your code should ideally perfectly match the Javadoc and provided class diagram. However, if you find that something from the provided class model cannot work in your implementation, find a solution and justify in the text why you could not respect the class model. Moreover, say a few words about how you would change the class model as a result to maintain consistency.

Make sure to add comments to your methods. Javadoc is already provided to you, but you should make sure to explain what the various parts of your code is doing, such that a colleague (or a marker) can understand it easily.

Include assertions in at least some of your methods as a means of catching invalid inputs, faults and invalid states.

Some advice

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/`

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.

1.2.2 Task 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 implemented in the last task (still respecting group sizes). This includes your group's specific Request Booking Records implementation (see section 1.1.5).

To get a good mark for this coursework you must test all the use cases you implemented since your tests will provide 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 are expected to add enough systems tests to completely test the relevant use cases.

Some advice

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 `SystemTests.java` class, or as unit tests corresponding to other classes.

Your `SystemTests.java` file should serve as the primary evidence and documentation for the correct functioning of your system. To this end, take care with how you structure your test methods and add appropriate comments, for example noting particular design features being checked.

1.2.3 Task 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 in lectures and Tutorial 5.

For your coursework submission, in addition to the system tests from the previous task, you are required to provide unit tests for only the following classes:

1. `UserState`
2. `BookingState`
3. `EventState` (only for **groups of 3+**)
4. `SponsorshipState` (only for **groups of 4**)

In each test class, you should have several test methods for each method of the class that you are testing. Each test method should test for one combination of inputs as parameters. Make sure that you include frequent classes of inputs, but also more unusual ones, boundary and incorrect inputs for testing (see Tutorial 5 and its solutions for help). Each test method should be appropriately named such that the case you are testing is clear. Each test method should run the method that is being tested only once, i.e. you would normally have one single `assert` statement in each test method. Finally, include a clear message when the test fails.

1.2.4 Task 4: Code review (optional)

This task is optional, but a great opportunity to get some feedback on a small part of your code - specifically, your Request Booking Records implementation and its system 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, having prepared in advance the Request Booking Records implementation 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 implementation and its system 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.

1.2.5 Task 5: 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 mobile app for it for consumers (some of whom may be elderly, or more vulnerable to COVID-19). 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.

1.2.6 Task 6: Reflection (same as in CW1 and CW2 apart from subtask 3 (marked as NEW!) and the marking scheme)

This section asks you to reflect and self-assess your team's progress with this coursework. This is a great opportunity to take the time to learn from your experience with this coursework, and thus develop your analytical thinking skills. It will also help you judge your work before we do, by interacting with the marking scheme and thus gaining an understanding of how you are being marked.

Before you start doing any reflection, here is a useful model (adapted from the Integrated Reflective Cycle (Bassot, 2013)) that you could use to structure your reflection:

1. **The Experience**: Describe what you did, what you tried out.

2. **Reflection on Action**: What were the results? What went well? What didn't? Why?
3. **Theory**: What have you learned from this experience?
4. **Preparation**: What could you have done to make things better, according to the lessons learned? If you have the chance to do this again (e.g. team work), what will you do or try out next time to try to make things better?

You can see an example of this model being put to use at this link.

Write one paragraph with your reflection on each of the following topics (we expect between half a page and one page in total for both topics, with more effort gaining more credit):

1. Teamwork: here, reflect on things such as how you got organised, split up responsibilities between team members, communicated, and managed progress in working towards the deadline for this coursework. Make sure to mention and reflect on the use of and usefulness of any tools that you tried out in this process, e.g. physical tools or online tools for managing your team work.
2. The quality of your work: here, reflect on how well you think you tackled the work in the different tasks. We recommend you have a look at the marking scheme from COMING SOON to help you structure this response, however touching on all criteria from there, or marking yourself using it is not expected.
3. **NEW!** Your experience with any:
   - Approaches to software development from the lectures starting from Week 6
   - Tools which you have used from the ones we studied (version control, test coverage, bug tracking, static analysis, project management tools
   - Agile practices that you have used

   In particular, here it is useful to reflect on your reasoning for choosing each approach/tool/practice (and not choosing an alternative), how the approach/tool/practice worked for you, what you would conclude are its advantages/disadvantages, what you would do different next time you develop a system. **IMPORTANT!** In your description, please include some screenshots of your use of any tools that you mention. of those tools

Important! You should make a real effort to be reflective, as well as honest, in this task. Please note that only making bold statements like "We did this excellently well", with no justification, and (even worse!) not being open to consider that there is always room for improvement, will result in very little, or even no, credit for this part.

1.2.7 Declaration of work

As in CW2, you are also required to declare the amount of work that was carried out by each of your team members, so that we can adjust your group mark accordingly. In particular, you are each expected to split up work *fairly*. This means each team member doing around:

- **Groups of 2**: half (50%) of the work
- **Groups of 3**: a third (33%) of the work
- **Groups of 4**: a quarter (25%) of the work

on this assignment (no matter how you split responsibilities), or to have agreed with teammates how to average out across courseworks in the the case of personal circumstances.

For this task, prepare a separate document (see submission details in Section 5) and do either a) or b) from below:

a) If you consider *your work was split fairly according to the definition from above*, include in it the text "The work was split fairly between our team members", and all sign (see below).

b) If not, write for each team member an estimate (in percentages) of how much work they have carried out. **Important! This estimate must be agreed with the other team members, who must all sign this document (see below)**.

At the end of the document, each team member must include a digital signature or simply a picture of their signature taken with their mobile phone.

Failure to submit this document or submitting it after the deadline will lead to our assumption that you have split the work fairly, and you will all receive the same mark without any objections possible. Submitting it without all signatures will lead to potentially difficult discussions between the course organiser and the team members. Finally, option a) done correctly will result in the same mark for all the team members, while option b) will mean bringing the mark down for anybody with a lower percentage than expected, in accordance with the declared percentage (e.g. if a team member in a 4-person group did 15% of the work, this is 60% of the 25% they needed to do, so they will receive 60% of the group's mark). Individuals who have done more than their share will not have their mark increased according to the percentage, but may respect criteria for excellence and exceptionality for a very high mark (see markings scheme).

2 Your ProP Work (individual)

As part of CW3 (ProP) you will be required to write an essay choosing one of the topics below.

The length of the essay should be around 1000 words.

2.1 Topics

Topic 1: “Identify one or more related accessibility problems that specific target users face when using your “Managing and Booking Events” system and propose an approach to address the problem(s).”

With the digital transformation, more and more people benefit from technology. However, an increasing number of people encounter accessibility problems, because of a permanent (e.g. low vision), temporary (e.g. a broken arm) or environmental disability (e.g. noisy location). For this essay, you have to focus on a certain target group and identify one or more related problems they face when using your “Managing and Booking Events” system.

Your essay should:

- describe the problem(s),
- explain further implications of the problem(s) for the target population (to what extent this limits/discourages them from using the system)
- propose an approach to address the problem and ensure inclusion
- justify your approach among others
- discuss your approach from various perspectives, e.g. how that affects various stakeholders' interests, at the level of organisations, potential risks (if any) in case the problem(s) is not addressed, to what extent the lack of approaching the problem falls under the Equality Legislation (i.e. leads to discrimination), to what extent it complies with ISO 9001:2015 principles.

You are not expected to cover every aspect of the system or task users need to perform – just focus on a few aspects/tasks. Also, you don't need to tackle the problem from all perspectives, but you are expected to discuss it from at least two perspectives.

Topic 2: “Discuss to which extent the scandal around the Dalí17 Museum regarding the use of Dalí's name and likeness in its logo, as well as reproduction and display of artworks online, is an infringement of copyright.”

The case is explained: here

You are expected to reflect on this case, but also to explore other cases to support your argument. You are also expected to approach multiple perspectives, e.g. the purpose of use, the amount and substantiality of the portion used in relation to the copyrighted work as a whole.

2.2 Writing the essay

When writing the essay, you should think about the assessment criteria for this coursework – see section 3.3. Read carefully the criteria definitions and use them to guide your writing.

Here is a possible approach for writing your essay:

1. **Revisit relevant (course) materials**. Revisit tutorial 1, Lecture 3.2, Lecture 16.1, Lecture 21.1, 21.3, 21.6, and other useful sources (e.g., the first chapter of "A rulebook for arguments"). Carefully read the coursework description, particularly section 4 Assessment. Also, you may want to use the following article as a guide for writing your essay: https://www.wikihow.com/Write-a-Short-Essay
2. **Brainstorm**. Take some time to write down your premises, conclusion, potential evidence. Write down ideas that support your conclusion but also opposing views. Think about various perspectives and write down ideas. Do not care about your writing style or structure/organisation of ideas, just write them down, preferably using bullet points.
3. **Prepare**. Collect all the evidence and write down an outline of your essay. Take enough time to gather the evidence you need and check if it was reliable. This is the phase when you organise your ideas written down in phase 2 and think about how to unfold them (e.g., in which order).
4. **Draft**. Write a rough draft of your essay. Don't spend time checking how correct your sentences are. Just write and try to include any data/direct quotes as early as possible.
5. **Revise**. Polish your rough draft, optimise word choice, make sure your sentences are concise, and restructure your arguments if necessary. Make sure your language is clear, and double-check that you effectively made all your points and rebuttals. Also, remove redundant sentences and sentences which do not bring any value to your argument. Do not expect to write your essay in one go, but develop it iteratively.
6. **Proofread**. Read through your essay and focus exclusively on fixing mistakes. Use Grammarly to identify and fix grammatical errors.

2.3 Assessment for the ProP Tasks

2.3.1 Criteria

The criteria for assessing this coursework are as follows:

C1. Comprehensibility/Clarity – The essay is clear and understandable. The terms are defined for the non-expert and used in a consistent way. It is clear what the main position of the author is.

C2. Structure of the essay – The essay has an introduction clearly stating the focus of the essay, a body that unfolds the ideas in a natural and coherent way, and a conclusion that restate the main point(s) of the essay. The essay includes a list of references. Citations and references consistently follow a set of rules (depending on the chosen style).

C3. Quality of argument – There is a logical connection between the premises and conclusions. The premises are relevant (in favour of) and sufficient for the conclusion. The essay considers also alternative views (counter-arguments) showing why they are flawed. In other words, there is a clear refutation for each counter-argument.

C4. Quality of evidence – The premises are reliable with solid evidence to support them.

C5. Exceptionality – This criterion refers to aspects of the essay which are well beyond the level expected of a competent student at their level of studies, such as surprising/novel ideas or publishable essay quality in its current form.

2.3.2 Grade description

The grade below will be used only for criteria C1 to C4.

Unacceptable – No attempt or attempt at criterion which is not relevant.

Poor (this is not a passing mark) – There is very little attempt to address some of the aspects.

Fair (this is a marginal pass) – The essay attempts to address some of the aspects in the criterion with minor problems, or most of them but some have major problems.

Good – The essay addresses most of the aspects in the criterion with no major problems. Alternatively, the essay addresses all aspects of the criterion, but there are major problems in a few of them.

Excellent – The essay addresses all the aspects of the criterion; there are no major issues or omissions.

For **C5 (Exceptionality)** you can get:

- 8 marks if your essay brings strikingly novel perspectives/ideas that surprises the reader OR it is of publishable quality in its current form
- 16 marks if your essay brings strikingly novel perspectives/ideas that surprises the reader AND it is of publishable quality in its current form Look at the CW3 (ProP) Rubric for more details.

3 Some advice (same as for CW1 and CW2)

3.1 Working online as a team (For SE tasks only)

Teamwork is not easy, and you may need to sometimes 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 colleagues, setting up meetings in the calendar, video calls, chat, editable file sharing, its Tasks By Planner and To Do to organise and split your work.
2. OneDrive (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, which you will also use later in this course. Please be careful about access permissions (see subsection 4.3).
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. Miro (`https://miro.com/online-whiteboard/`) 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 Task 6.

3.2 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, user interviews 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!

3.3 Good Scholarly Practice and Respecting the Law

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).

Finally, please be warned that forging someone's signature is a criminal offence.

4 Submission

Please make **three** submissions for this coursework (**IMPORTANT!** Some are group submissions while for ProP it's individual):

- **As a group**:
  1. SE task solutions, under "Coursework" – "Coursework Submission" – "CW3 Software Engineering". This should include:
     - A .ZIP file of your submission, including all your Javadoc as well. To do this in IntelliJ, go to:

       `File - Export - Project to ZIP file`.

       Rename this file **submission.zip**.
     - A PDF (not a Word or OpenOffice document) of your report. The document should be named **CW3SEreportGroupX.pdf**, where you replace the X with your group number. Please **DO NOT include the names and/or UUNs of the team members, or the declaration of work** within this document, so that we can mark anonymously. This submission should be made by one person and is used for the whole group.
  2. The SE team declaration of work, under "Coursework" – "Coursework Submission" – "CW3 Software Engineering". This should includeA PDF (not a Word or OpenOffice document) of your 'Declaration of work', entitled **DeclarationGroupX.pdf**, where you replace the X with your group number. This document will only be accessed by your course organiser. This submission can be made by a different person in the group.
- **Individually:** Your individual ProP report, under "Coursework" – "Coursework Submission" – "CW3 Professional Practice". This should include a PDF (not a Word or OpenOffice document) of your report with the essay, named **CW3ProPreport GroupX.pdf**. Please **DO NOT include your name, or the declaration of work** within this document, so that we can mark anonymously.

How to Submit

Ensure you are logged into MyEd. Access the Learn page for the Inf2-SEPP course and go to "Coursework" – "Coursework Submission". See in the previous section where to submit each part.

Submission is a two-step process for each: 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.

5 Deadline

Please submit all parts of this coursework, and both documents, for a final mark and feedback by:

16:00, Fri 8th of April 2022.

This coursework is worth 63% of the total coursework mark: 38% for the SE tasks, and 25% for the ProP tasks.

Important! Similarly to the last two courseworks, CW3 follows Rule 3 on late penalties as detailed here: https://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests, with 3-day calendar extensions permitted.

Borislav Ikonomov, Vidminas Mikucionis, Cristina Alexandru, 2022.