Inf2-SEPP 2023-24

Coursework 3

Software implementation of the university self-service portal

This coursework requires teamwork for SE (1), and individual work for ProP (2).

1 Your SE work (team work)

The aim of this coursework is to implement and test the self-service portal for the University of Hindeburg. It builds on Coursework 1 on requirements engineering and Coursework 2 on design.

The following are the documents that you will primarily need:

- The Development Tools document
- The Requirements and Design document (to be released on 15 March 2024 at 12 pm once all teams with extension will have submitted CW2)
- Mock implementations of the interfaces that handle our system's interactions with the Student Record System and the Email Service.
- CW3-SE high level marking scheme

Please never refer to your CW1 or CW2 solutions in this coursework.

1.1 Requirements and their updates

Your colleagues in AcmeCorp have produced a list of requirements which puts together your conclusions from your previous work (in Courseworks 1 and 2) and also includes further clarifications with the stakeholders (who agreed with the assumptions that were made in Coursework 1 and 2 solutions). This entire list is provided in the Requirements

and Design document- Section 1 (see 1). You should use it as your main source of requirements for this coursework.

Additionally, your company has recently had another meeting with the university execs and has come up with the following decisions, which you should also consider:

External Systems. In the live service version of our system, we would need to interact with the two external systems: Authentication Service and Email Service. However, it has been decided that for this early test version we will not interact with them directly and will just simulate their responses. This will be done via the two provided classes (see 1).

Email Service Response. The external (or in this case, mocked) email service will return an integer code that provides information on whether the email was successfully handled, and if not then what the problem was. The possible codes are:

- 0: Email was sent successfully.
- 1: Email was not sent because the sender email address is invalid (e.g., null or not formatted correctly).
- 2: Email was not sent because the recipient email address is invalid.
- 100: Email was not sent for an unknown reason (i.e. a reason not shown above)

External Authentication Service. To authenticate users, the self service portal system sends the username and password (typed in by a user) to the authentication service, which responds with a JSON object. If the details are incorrect, the JSON response will contain a field `"error"` with an error message. For example:

```
{
    "error": "Wrong username or password"
}
```

Otherwise, it will contain a user object with the fields username, password, email, and role, for example:

```
{
    "username": "Barbie",
    "password": "I like pink muffs and I cannot lie",
    "email": "barb78916@hindenburg.ac.uk",
    "role": "Student"
}
```

The role can be one of “AdminStaff”, “TeachingStaff”, or “Student”. Users’ full names are not returned. The requirement to greet users by name on login is dropped.

Private FAQ topics and Q&As. The requirement to support private FAQ topics and Q&As is dropped, and all FAQ topics and Q&As will be available for everyone in this first release. Pages, however, can still be public or private.

Inquirer information. In the CW2 interview with "students" (lab demonstrators), they may have said that as inquirers they would like to see the email address of the person their inquiry is sent to. However, because inquiries cannot be directed to specific people by inquirers, they all go to the general admin email at first – so it is not possible to show the email address of the eventual inquiry recipient. This requirement is dropped.

1.2 Design documents

After further refinements during the design stage, your team has come up with further documentation that describes the new system. This is provided to you in the form of the Requirements and Design document- Section 2 (see 1). Specifically, this document includes a full class diagram and associated explanations for the entire system, and sequence diagrams for some specific use cases (the ones discussed in CW2). ***IMPORTANT! The code you implement for this coursework should precisely follow these class and sequence diagrams, for the use cases that you are required to cover for your team size.***

In addition, your company still requires you to use the Lucene library.

1.3 Using tools, agile approaches and practices

This coursework is a small-scale opportunity to try out approaches, practices 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
- You are encouraged to use any applicable *practices* that have been described in Weeks 7, 8 and 9 of the course and that have been proposed by one of the following *agile approaches*: Extreme Programming, Scrum, Kanban. Here, you do not need to stick to one approach, but you are free to mix and match practices from different approaches 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!
- You are encouraged to try out software tools that are recommended for your chosen approaches and practices, but also experiment with other tools that may be helpful for construction, testing and teamwork. Overall, apart from the required tools mentioned in the next section, you could try out for example a test coverage tool like the default one provided by IntelliJ IDEA[1] or one of its alternatives, a bug

[1] `https://www.youtube.com/watch?v=QDFI19lj4OM`

tracking tool like Trac[2] or JIRA[3], a static analysis tool like SpotBugs[4] or Infer[5], any tools for support with agile like JIRA, and any tools for team work as presented in the teamwork resources.

It is up to you how many and which tools, agile approaches 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 Task 6 and your assessment for this section.

1.4 Required Development Tools

Your company requires its developers to use certain development tools. We provide a Development Tools document that specifies these tools, their required versions, and how to install them (see 1).

You will also 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 links: `https://www.vogella.com/tutorials/JUnit/article.html`, `https://bit.ly/3clJLs6`, `https://bit.ly/38t1ltn`, `https://www.w3schools.com/java/java_interface.asp`

1.5 Following good practice in construction and testing

Throughout this coursework, 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 of the default size set up by your IDE 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 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.

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 includ-

[2] `https://trac.edgewall.org/`
[3] `https://www.atlassian.com/software/jira/bug-tracking`
[4] `https://spotbugs.github.io`
[5] `https://fbinfer.com`

ing 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.

Use JUnit 5 to your advantage by using the right methods in your test class, the right annotations to test methods, and the right assertion methods (see Lecture 19).

Your system test files will serve as your primary evidence and documentation for the correct functioning of your system. In terms of their documentation, take care with how you structure your test methods, give them intuitive names, add appropriate comments (for example noting particular features being checked) and include a clear message when each test fails.

1.6 Your Tasks

There are several tasks you need to engage in: **these tasks are provided in a certain order, but you may decide to change their order or do several of them in parallel depending on your agile practices**. Moreover, **we recommend you perform tasks 0, 1 and 3 (unit tests for the mock services only) while waiting for the Requirements and Design document released on the 15th of March**. The tasks are described in the following subsections.

1.6.1 Task 0: Lucene Tutorial

This tutorial continues from where the first Lucene tutorial (available here) left off.

Start by opening the project you created previously (or make a new one by following the first tutorial steps again). Then follow the below steps.

Step 5) Create your first class Create a new Java class in your project's `test/java` subdirectory and call it `LuceneTest`. In the `LuceneTest` class, add a method with the signature `public void testCore()`. Copy the sample code provided in the core API documentation page: `https://lucene.apache.org/core/9_9_0/core/index.html` into this method.

At first, there will be lots of errors. You'll need to import Lucene library classes and methods so references to them are not undefined. IntelliJ will offer you the option to do so when you hover your mouse over each red class name or static function. As you do this, be mindful of which libraries you choose to import — you need the ones from Lucene, and it may not always be the first option!

Once you've done this, some unhandled exception errors will be left. You can just add them to the main method signature for now (IntelliJ will offer to do so for you if you hover your mouse over the parts underlined with red squiggles).

After importing all the relevant Lucene classes, you'll notice that two `assertEquals` function calls are still undefined, and there is no class to import for them. This is because

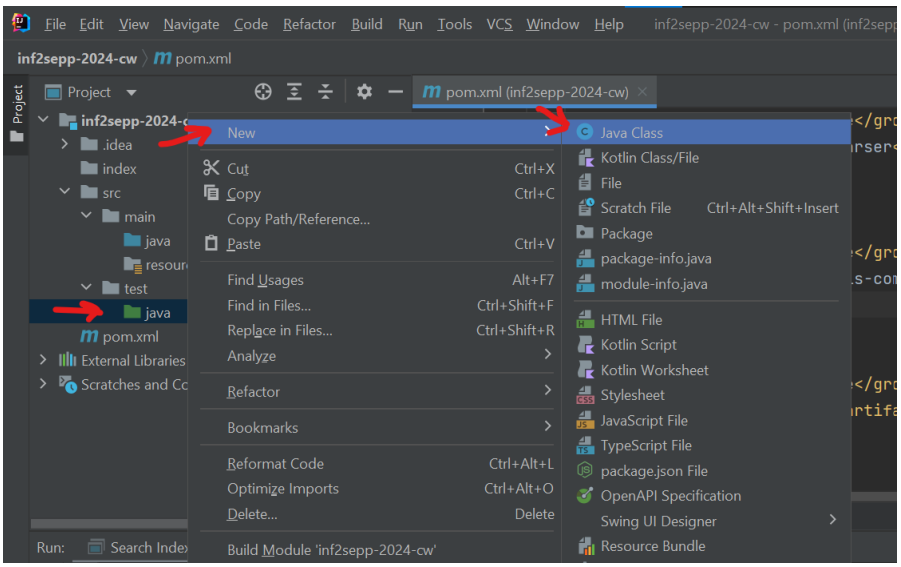

Figure 1: Step 5a: create a new Java Class

Figure 2: Step 5b: import references (Lucene may not always appear on top!)

`assertEquals` is a JUnit library function and we haven't added the JUnit library to our project yet.

Step 6) Add JUnit to the project You'll need to add JUnit to your `pom.xml`, similarly to how we added Lucene earlier. This time, see if you can figure out how to do it yourself following the official instructions: `https://junit.org/junit5/docs/current/user-guide/#running-tests-build-maven`. A hint: you'll need to add 3 things: `junit-bom` under `dependencyManagement > dependencies`, `junit-jupiter` under `dependencies`, and `maven-surefire-plugin` under `build > plugins`.

Once you've added JUnit dependencies, you'll have the option to import the `assertEquals` function. You'll also now be able to add an annotation `@Test` above your `testCore` method.

Some of the method calls may throw exceptions. Just add these to the test method

signature.

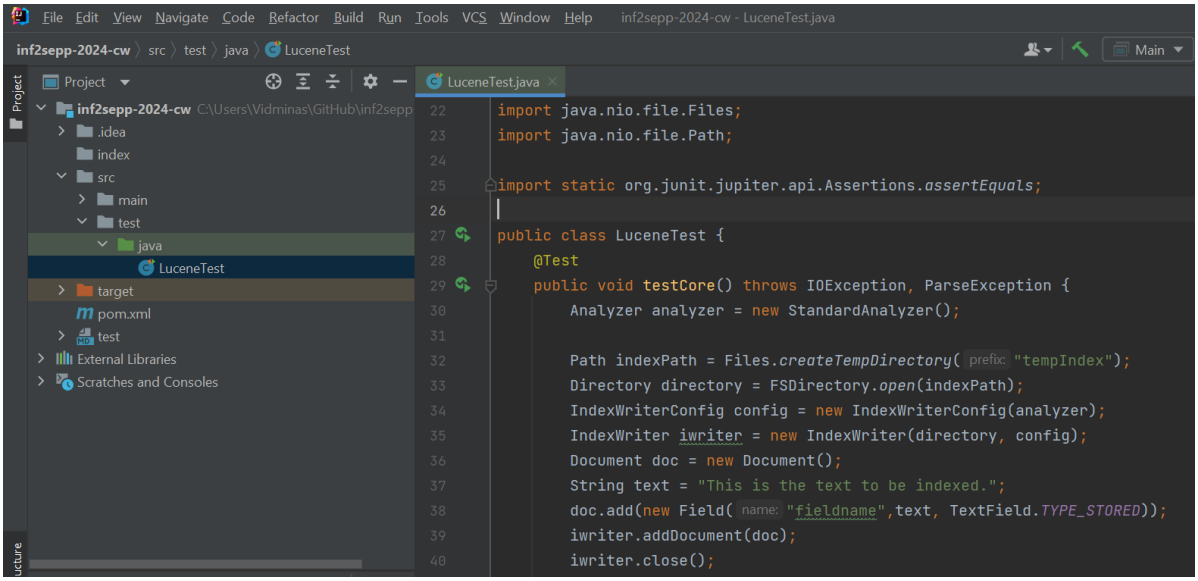

Figure 3: Once all the dependencies are configured and the `LuceneTest` class is completed, there should be no more errors left.

Step 7) Run the test Finally, add a new run configuration. This time, choose JUnit as the configuration type.

Give the configuration a reasonable name, such as "Test Lucene Core API" or similar and set the class to `LuceneTest`.

Run your new configuration and ensure the test passes.

Step 8) Implement Page Search From here you should be well set up and ready to start implementing page search using Lucene. If you need a quick overview of the available queries and their equivalents in Java, see `https://lucenetutorial.com/lucene-in-5-minutes.html`. For further details, explore the official documentation at `https://lucene.apache.org/core/9_9_0/`.

You will need to make a class called `PageSearch` with all the functionality for adding and querying pages.

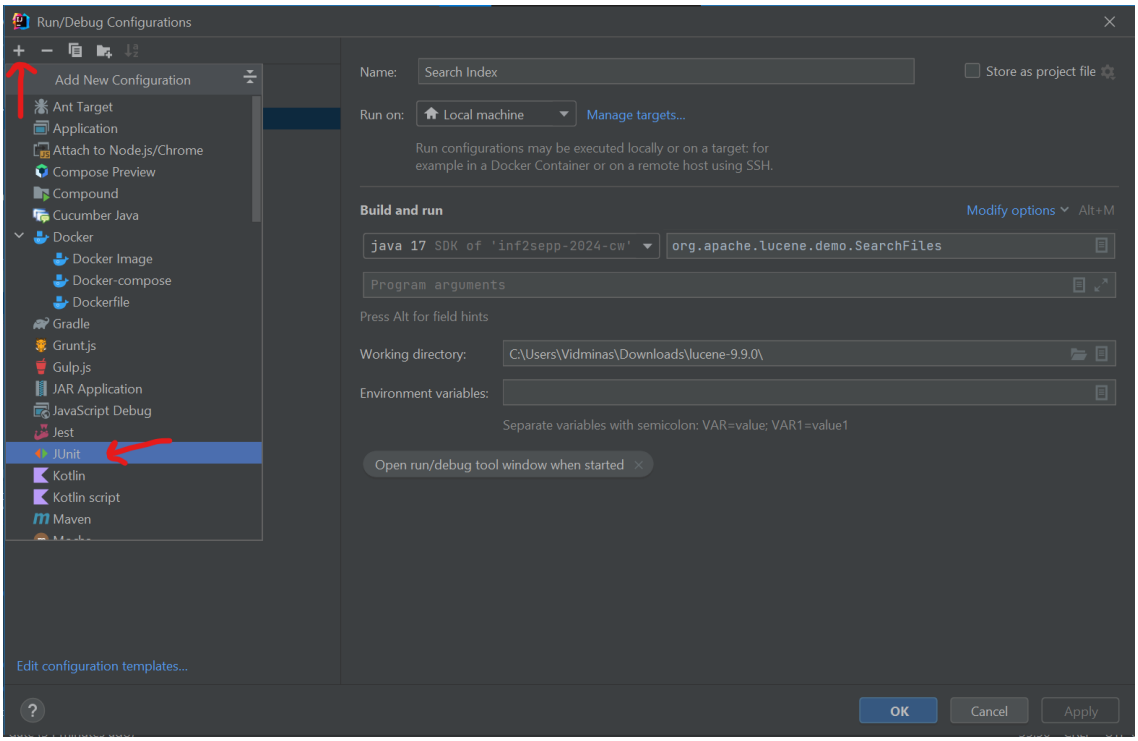

Figure 4: Creating a new JUnit run configuration

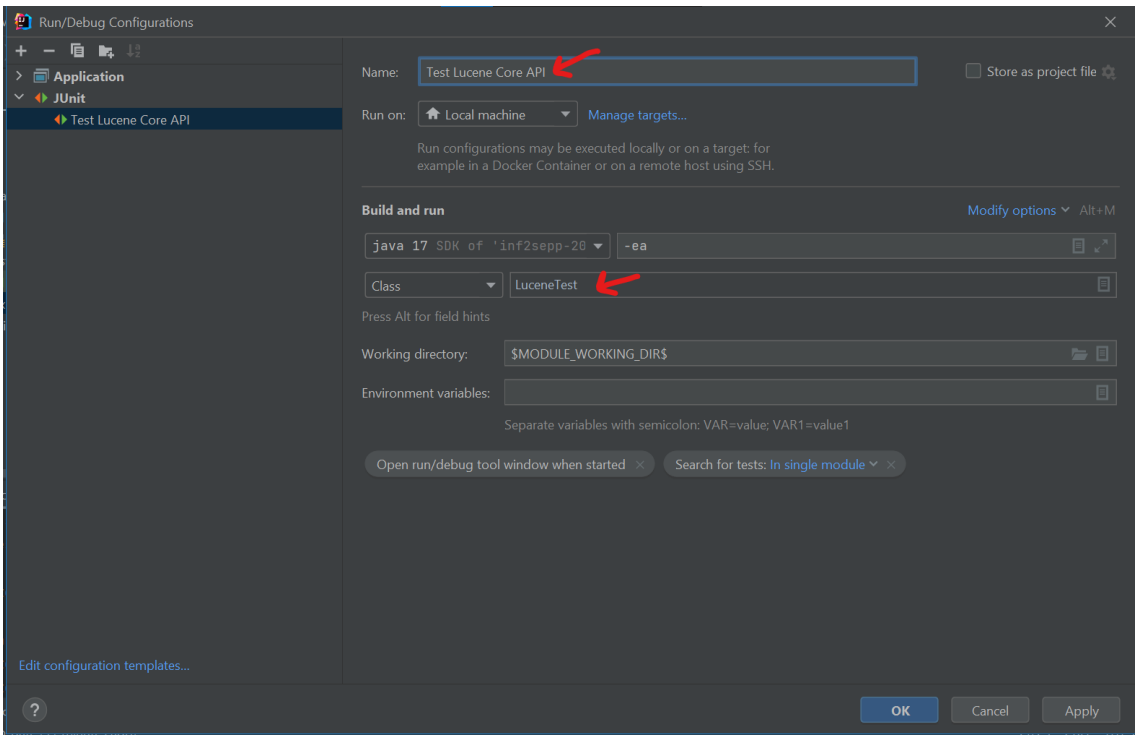

Figure 5: Configuring the new JUnit run configuration

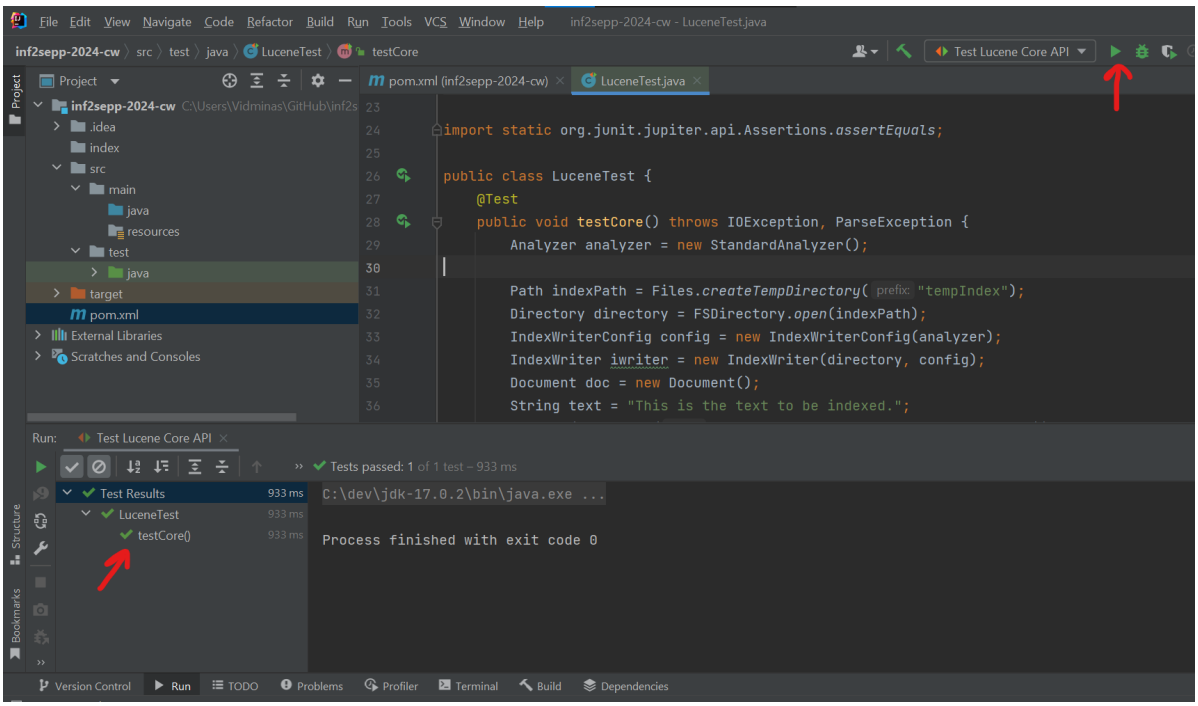

Figure 6: Run the JUnit test

1.6.2 Task 1: Integrating external systems

The external system interfaces and mock definitions are provided for you (see 1). You should add these to your project.

You can find 4 Java files on Learn: `AuthenticationService.java`, `MockAuthentication Service.java`, `EmailService.java`, and `MockEmailService.java`. Put them in a "external" package under `src/main/java/external`.

Note that the mock authentication service uses a JSON library "`json-simple`" (`com.googlecode.json-simple` v1.1.1). Add this library to your `pom.xml` using your knowledge of dependency management from the Lucene tutorial.

Mock user data files for the mock authentication service are also provided together with the Java files. There are separate files for groups with 4 team members and for groups with fewer; you only need one of them. Choose the appropriate one for your group size and place it in your project's `src/main/resources` subdirectory.

You should not modify the provided external system files, but read them carefully to better understand how to use them in your system. With one exception: you should change the file name that data is loaded from in `MockAuthenticationService.java` to match the right mock user data file for your group size. In this task, you will be marked on whether the external systems are integrated and used (also see next task).

1.6.3 Task 2: Code construction

Construct the code that implements the following use cases (these are the exact same as in CW2):

1. **Log in**
2. **Log out**
3. **Add webpage**
4. **Browse webpages (admin)**
5. **Consult webpages**
6. **Add FAQ Q-A**
7. **Browse FAQ (admin)**
8. **Consult FAQ** (where for teams of 2 no consideration of updates/no updates extensions)
9. **Consult member of staff** (teams of 3+)
10. **View received inquiries** (teams of 3+, where for teams of 3 no need to consider teaching staff viewing them)
11. **Answer inquiry** (teams of 3+, where for teams of 3 no need to consider teaching staff answering them)
12. **Redirect inquiry** (teams of 4)

Make sure you closely follow the requirements, class model and sequence diagrams from the Requirements and Design document (see 1), as well as the updated requirements from Section 1.1. Your code should ideally perfectly match the diagrams. However, if you find that something from them cannot work in your implementation, find a solution and justify in the text why you could not respect the design models.

Make sure to add comments (both JavaDoc and inline) to your methods. You should make sure to explain what the various parts of your code are doing, such that a colleague (or a marker) can understand it easily.

Include inline assertions in minimum 2 methods as a means of catching faults and invalid states, but please make sure that you turn these off before submitting.

Advice for writing high quality code was provided in section 1.5.

Unlike CW1 and CW2, this coursework doesn't have a dedicated "ambiguities" task. However, ***make sure to mention any assumptions you make in this task***.

Reaching an "Excellent" mark in this task: If you become confident that your solution to this task correctly respects all of its instructions from above and is of high

quality, and you want a mark of "Excellent", you need to put in effort to go beyond what was required to impress the marker. The following are some possible ways:

- Putting a lot of effort into checking for faults, invalid states, or doing input validation to the code; Briefly mention what you have done in the report.
- Frequently using well-known and high quality algorithms; Include their mention and a justification for using each in the report.
- Looking for "bad smells" in the code and proposing ways that the code could be refactored (in writing only, without modifying the code which will be checked for conformance with the class and sequence diagrams). Here, please include small code excerpts in the report as needed to support the explanation. See Lecture 18.

This, together with excellent tests, can lead to exceptionality marks too.

1.6.4 Task 3: Unit testing

Use JUnit 5 to create unit tests for the following classes:

- Groups of 2: `MockAuthenticationService`, `MockEmailService`
- Groups of 3 or 4: `MockAuthenticationService`, `MockEmailService`, `PageSearch`

These should focus on checking whether the implementation of each specific class is correct, and all pass. Place the tests for a class named `MyClass` in a file named `TestMyClass.java`, and use JUnit 5 assertions and annotations similarly to the examples provided in lectures and Tutorial 6.

In each test class, you should have several test methods for each method 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 6 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.

Other advice for testing is provided in section 1.5.

Reaching an "Excellent" mark in this task: If you become confident that your solution to this task correctly respects all of its instructions from above and is of high quality, and you want a mark of "Excellent", you can try to test for all classes of inputs for each method and also use a tool to calculate statement and branch coverage in each of the tested classes and improve your code and retest until they are at least 70% (see Lecture 19). This, together with excellent code and system tests (see below), can lead to exceptionality marks too.

1.6.5 Task 4: System testing

In addition to the unit tests from Task 3, use JUnit 5 to create system-level tests that simulate the scenarios of each of the use cases you implemented in Task 2 (still respecting group sizes). To simplify this task, you are not expected to write tests for notifications being sent when a member of staff is initiating the "Add FAQ Q-A" use case.

To get a good mark for this coursework you must test all the use cases you implemented, with several scenarios each, and make sure that they all pass, 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. For each use case, have each test check one possible scenario.

Few tests are able to test single use cases by themselves. Oftentimes, several use cases are needed to set up some state, followed by one or more to observe the state. For example, you will need to have a user log in and send a inquiry to staff before a staff member can respond to the inquiry.

Include all your system tests for each use case as JUnit test methods in a `[use-case-name]SystemTests.java` class (e.g. `ConsultFAQ.java`, `AnswerInquiry.java`, etc.). Overall, you should have one such class for each use case.

Other advice advice for testing was provided in section 1.5.

Reaching an "Excellent" mark in this task: If you become confident that your solution to this task correctly respects all of its instructions from above and is of high quality, and you want a mark of "Excellent", you can try to cover through your system tests most if not all of the different scenarios for the each case that you are tackling. This, together with excellent code and unit tests, can lead to exceptionality marks too.

1.6.6 Task 5: Quality attributes

Reflect on the following quality attributes, **using only notions from the lectures**:

- 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 them a good choice?

1.6.7 Task 6: Reflection (same as in CW2 apart from size limits, 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. We think such reflection could have an impact on your learning and of your understanding/expectations of how you will be marked.

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.

Start by having a look at the following reflection model (adapted from the Integrated Reflective Cycle (Bassot, 2013)) that you are asked to 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.

1) Reflection on teamwork

Using the above reflection model, write one- two paragraphs summing up to *maximum 250 words* of reflection on your team's teamwork for this coursework. Focus on things such as how you got organised, split up responsibilities between team members, communicated, accommodated different team members' needs, 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 or online tools for managing your team work. You can use the reflection model repeatedly to describe how you improved your teamwork over time.

2) Reflection on the quality of your work (can lead to exceptionality marks)

Using the same reflection model, write one or two paragraphs summing up to *maximum 250 words* of reflection on the quality of your work. Mention how well you think you tackled the work in the different tasks. We recommend you have a look at the marking scheme from this link - making specific reference to parts of it- to help you structure this response, however touching on all marking criteria or marking yourself using it is not expected. You can use the reflection model repeatedly to describe how you improved your solutions over time.

3) NEW! Reflection on tools, agile approaches and practices (can lead to exceptionality marks)

Using the same reflection model, write one or two paragraphs summing up to *maximum 350 words* of reflection on your use of tools, agile approaches and practices for this coursework (see 1.3). 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, include some screenshots of your use of any tools that you mention***.

1.7 Declaration of team work (same as in CW1 and 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 individual mark accordingly. In particular, you are each expected to split up work *fairly*. This means each team member doing:

- **Teams of 2**: around half (50%) of the work
- **Teams of 3**: around a third (33%) of the work
- **Teams of 4**: around 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 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 (ideally, see Terms below).

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

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

Terms: Failure to submit this document or submitting it after the deadline + 15 minutes will lead to our assumption that you have split the work fairly, and you will all receive the same mark without any objections possible. We will accept a document without all signatures included, as long as it is submitted before the deadline + 15 minutes. However, note that 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 team did 15% of the work, this is 60% of the 25% they needed to do, so they will receive 60% of the team'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 marking 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 (excluding the bonus parts which should be maximum 300 words).

As ChatGPT and similar tools are commonplace nowadays, we would love to integrate the results provided by these tools and combine them with your own thoughts.

Therefore, for each of the topics below you will find the corresponding ChatGPT 3.5 answer here and your task will be to write an essay outlining:

- What are your reflections on the answer? Not all which is good might be good and the other way round
- Find references for what you write
- Find evidence in literature for the statements
- Describe what your opinion is
- Find examples (if applicable)
- Missing information (if applicable)

The ChatGPT answer mostly is very common-place and not specific. There is plenty of option to dig into details, find examples in literature, give pros and cons (sometimes not necessary). Yet always your consideration (so, what you think important) is questioned.

This essay-task is not to test your factual knowledge derived by lectures or reading, yet the understanding and ability to reflect on information presented and apply this to your own context in a coherently manner.

There are no "right" answers, perhaps some "wrong" ones – the main achievement is to see your thoughts and reasoning based on evidence.

2.1 Topics

Topic 1: "Discuss different organizational forms of companies and their implication on agile projects and software engineering in general. As an optional bonus describe a hypothetical optimal organization form of a company for agile projects"

Topic 2: "Discuss the impact of software engineering standards for software project implementation and which meaning it has for a production of high-quality code. As an optional bonus briefly describe how a software engineering standard which address functional safety would impact your software engineering process"

2.2 Writing the essay

When writing the essay, you should think about the assessment criteria for this coursework – see the next section. 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 lectures 1, 3, 8 and 17, 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 this article as a guide for writing your 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

Please find here a marking scheme for ProP.

3 Some advice (Same as for CW1 and CW2)

3.1 Working as a team (for SE tasks only)

To get organised and work effectively as a team, you may want to have a look at the Teamwork Resources, and follow any advice on teamwork from guest lectures. You may also want to mention what you have done and used for teamwork in Task 6.

3.2 Asking questions

Please ask questions in labs or on Piazza if you are unclear about any aspect of the system description or what tasks. On Piazza, tag your questions using the *cw3* 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

Please remember the University requirement as regards all assessed work:

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

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

You are also required to take reasonable measures to protect your assessed work from unauthorised access. E.g., if you put any such work on an online repository like GitHub then you must set access permissions to allow access only to you and your team.

4 Submission

Please make **three** submissions for this coursework:

1. **As a group**, the SE task solutions, under "Assessment" - "Coursework 3"- "SE"- "Submit Code and Report". This should include:
    - A .ZIP file of your implementation, including all your tests and Javadoc as well. To do this in IntelliJ, go to: `File - Export - Project to ZIP file`. Rename this file **CW3SEImplementationGroupX.zip**.
    - A PDF (not a Word or OpenOffice document) of your report. The document should be named **reportTeamX.pdf**, where you replace the X with your team number. Please **do NOT include the names and/or UUNs of the team members** within this document, so that we can mark anonymously. Only one of the team members needs to submit this document, and the last submission will be considered for marking if several team members submit.
2. **As a group**, The SE team declaration of work, under "Assessment" - "Coursework 3"- "SE"- "Submit Declaration of Teamwork". This should include a PDF

(not a Word or OpenOffice document) of your 'Declaration of work', entitled **CW3SEDeclarationGroupX.pdf**, where you replace the X with your group number. This document will only be accessed by your course organiser. This submission should also only be made by one of the team members, but this team member can be different to the one who submitted the requirements document.

3. **Individually:** Your individual ProP essay, under "Assessment" - "Coursework 3"- "ProP". This should include a PDF (not a Word or OpenOffice document) named **CW3ProP[exam number].pdf**, where [exam number] is replaced by your own exam number. Please **DO NOT include your name** in the document.

How to Submit

Submission is a two-step process: (i) upload the file, (ii) 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.
- 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 ask for help if you have a problem submitting.

5 Deadline

Please submit both your SE and ProP solutions by **12:10, Fri 5th April 2024**. You are allowed to submit the declaration of work 15 minutes later.

This coursework is worth 63% of the total coursework mark: 38% for the SE tasks, and 25% for the ProP tasks. We estimate it should take each team member around 26 hours of work (20 for SE tasks, 6 for ProP task).Extension rule 3 is used for this coursework.

Borislav Ikonomov, Vidminas Vizgirda, Cristina Alexandru, Michael Glienecke 2024.