Include a cover page with the names and IDs of all your team members.

1. Introduction

An overview of the project, purpose/motivation (i.e. the problem you are trying to solve), targeted users. You can take this information from your project proposal.

2. Requirements

Backlog

All IDs will have a link to the corresponding user story in ZehHub/GitHub or JIRA. You can also group user stories into EPICs in the table below. This table should be consistent with your Backlog in ZenHub or JIRA at any time during the project. To indicate that a user story has been deleted you can use strikethrough. To indicate a change in the user story points strikethrough the old value and add the new value next to it. Newly added user stories will be highlighted in green color font. **Please link the ID with the corresponding JIRA or ZenHub issue.**

ID	Name	USP	Priority
ABC-1	As a user I want to X so that Y.	8	1
ABC-2	As a user I want to A so that B.	3	1
ABC-5	As a user I want to G so that H.	1	1
ABC-3	As a user I want to C so that D.	3	2
ABC-4	As a user I want to E so that F.	5	2
Total		20	

3. Release Planning

This will be repeated 3 times for the 3 Sprints of each Release. So, after each release there should be 3 Sprint summaries in the document.

Sprint 1 Summary

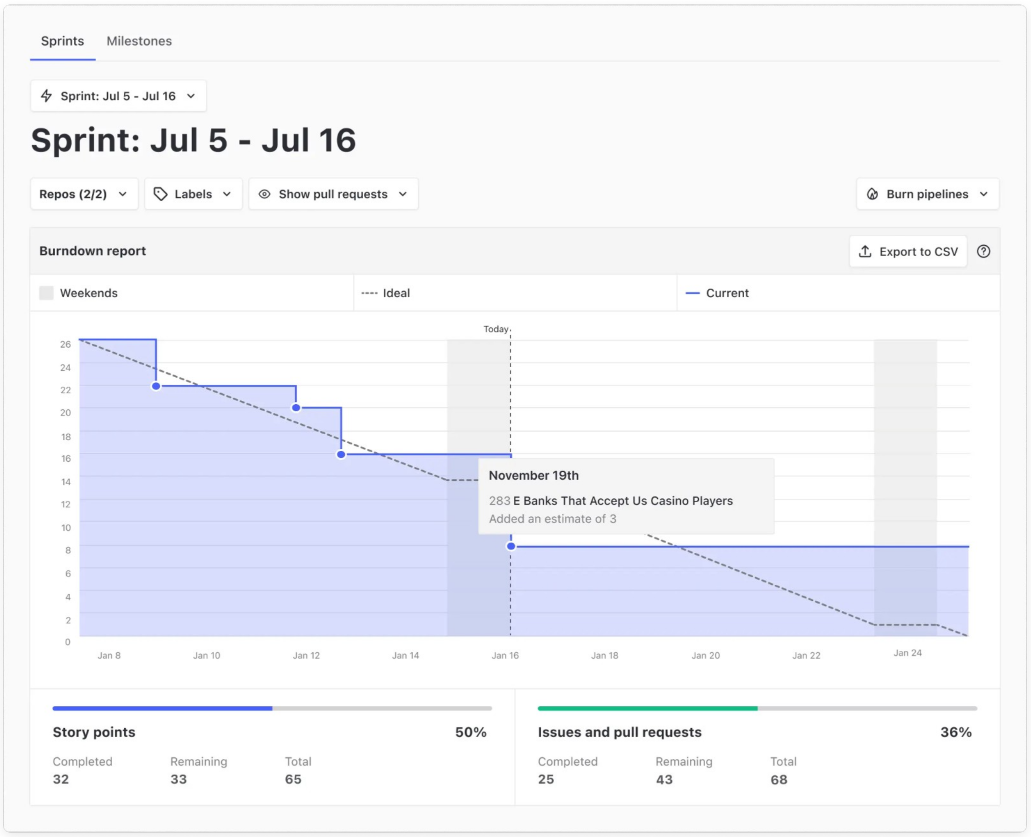
This sprint focused on delivering feature X and A. [Talk a bit more about what happened this sprint. What problems you faced, and what you had to do in response. What stories were pushed back. What stories were brought forward and a brief reason for all this].

Story ID		Planned USP	Status
ABC-1		8	**DONE**
ABC-2		3	**PUSHED TO SPRINT 2**
~~ABC-5~~		5	**REMOVED**
ABC-3*		1	**DONE**
Total		**12**	**9**

An * would be a good way to show stories added during the sprint. And a strikethrough would be good to denote a deleted story. Use anything you want, just make sure this information is clear.

Project velocity after 1 sprint: 9

[Burndown chart. The figure is self-explanatory. **A good plan execution should follow the Ideal line**]



Sprint: Jul 5 - Jul 16

Burndown report

November 19th

283 E Banks That Accept Us Casino Players

Added an estimate of 3

Story points 50%

Completed	Remaining	Total
32	33	65

Issues and pull requests 36%

Completed	Remaining	Total
25	43	68

Sprint 1 Retrospective (Obviously, more detailed than the example, I would like to see in practice how your retrospectives improve team communication and collaboration, processes and best practices)

Keep doing: Daily scrum.
Start doing: Standing up during scrum.

Stop doing: Playing League of Legends during group work sessions.

Do more of: Updating JIRA tasks.

Do less of: Micromanaging by PM.

4. Architecture

4.1 Overview of the architecture

Describe the architectural style you adopted for your software and justify your decision.
You can find a list of most commonly used architectural styles at
https://en.wikipedia.org/wiki/Software_architecture#Architectural_styles_and_patterns

4.2 Domain Model

4.3 Component Diagram

4.4 ER Diagram

Applicable only if your software uses a database to persist data.

4.5 Class Diagram

A basic principle for a good quality design is to include abstraction wherever needed and design your classes to depend on abstractions. Following this principle will make your design easier to extend and will reduce the overall system coupling.

You can split your class diagram into smaller sub-diagrams based on the structure of your packages/components.

For all diagrams you can provide external links to online diagram drawing tools.
Note: If your design does not follow the object-oriented paradigm, you can use other kind of diagrams for this section **after consulting with the course instructor**.

4.6 Design Patterns

Present the design patterns implemented in your design (Gang of Four patterns, concurrency patterns, security patterns, framework-specific patterns). Please note that design patterns are applicable regardless of the programming paradigm you follow. Justify the use of the design patterns and explain what extension mechanisms they provide that you will take advantage of as your software evolves.
https://en.wikipedia.org/wiki/Software_design_pattern
https://en.wikipedia.org/wiki/Security_pattern

4.7 External libraries

List all external libraries that you investigated for your software (assuming there are multiple alternative libraries for the same functionality, list all those you researched). Justify the selection of the libraries your current software implementation depends on (familiarity, more releases/updates, more secure, easier to use API).

5. User Interface Design

5.1 Personas

In this section, you must clearly define the **personas** representing potential users and describe some of the tasks they will perform on the system.

5.2 Equity

Research about how to make your software accessible to more people. People with disabilities (color-blind, blind, deaf, people with hand injuries), illiteral people, or marginalized people.

5.3 Mockups

You must have a **UI mockup** for each feature (user story) you intend to complete in each sprint. Mockups should be linked to each user story they are related, all mockups should be annotated to explain the functionality of buttons/inputs/outputs, UI variations based on the personas should be discussed and shown, you should list the updates to the UI mockups after receiving feedback from the product owner.

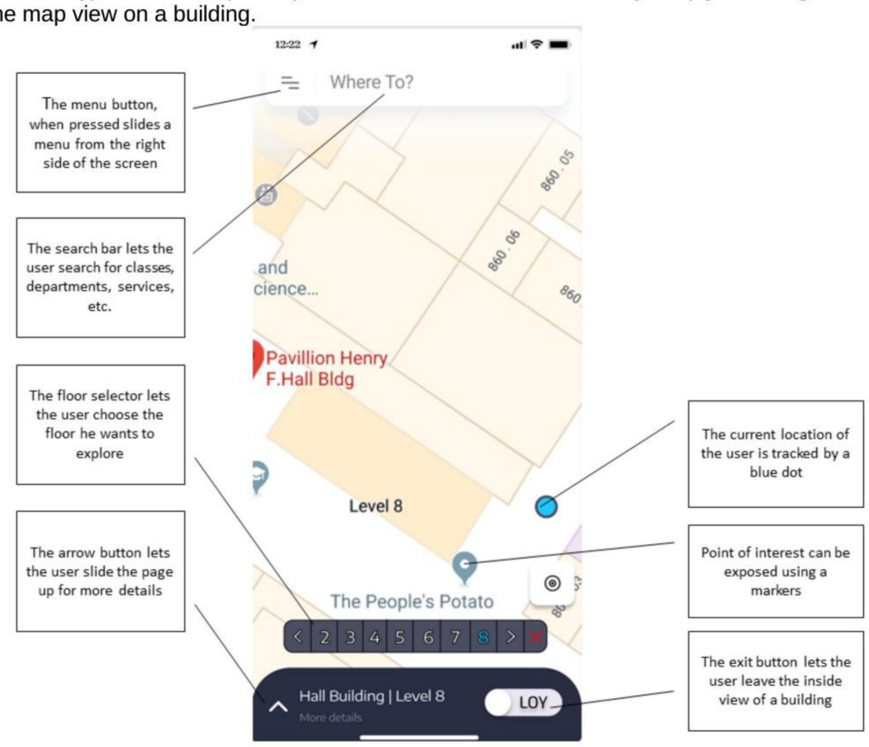
Sample mockup from SOEN 390 project GuideMe

Screenshot 5 – Indoors Hall View [UPDATED UI from Sprint 2]

Indoors Hall View – This view allows the user to see the contents of the Hall building and switch floors.

Personas:

Jessica (Junior Student), Ashwin (Senior Student), Thomas (Teacher), Jacob (Student with Disability), Katherine (Visitor), all have access to this screen by simply zooming in on the map view on a building.



The menu button, when pressed slides a menu from the right side of the screen

The search bar lets the user search for classes, departments, services, etc.

The floor selector lets the user choose the floor he wants to explore

The arrow button lets the user slide the page up for more details

The current location of the user is tracked by a blue dot

Point of interest can be exposed using a markers

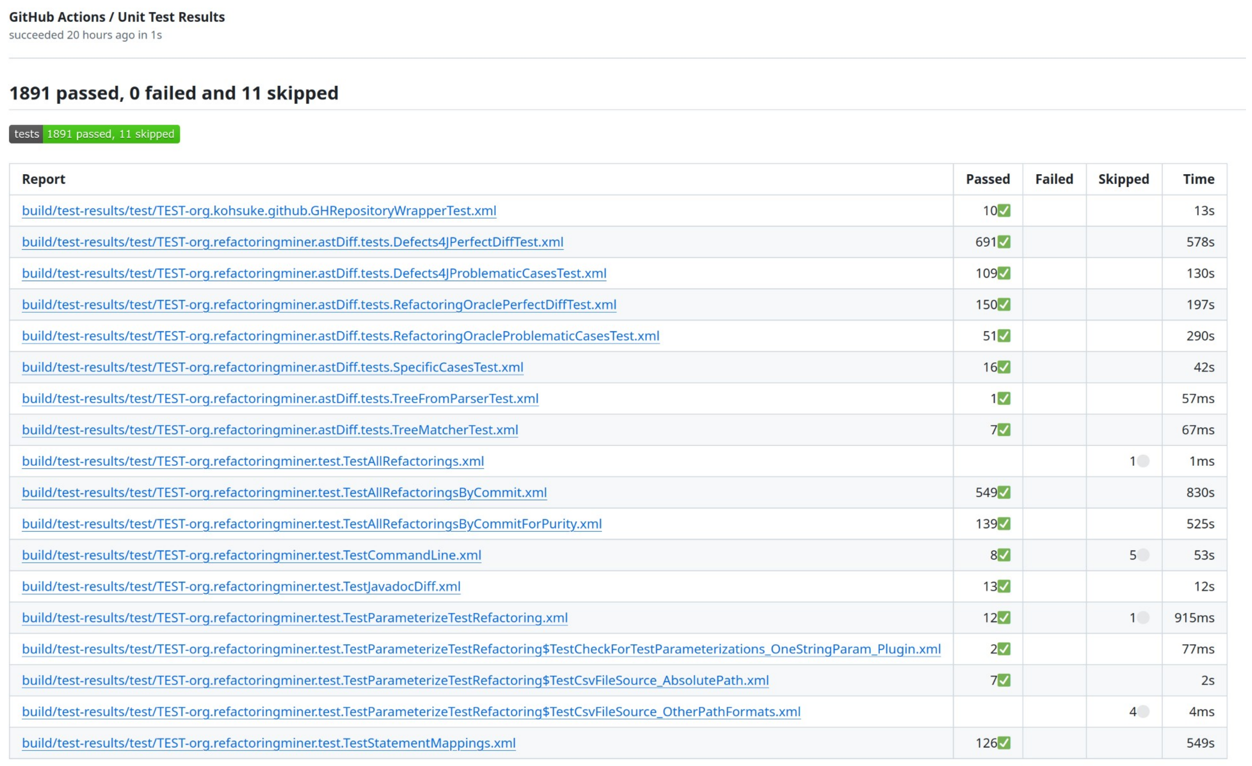
The exit button lets the user leave the inside view of a building

Update: The updated characteristic from this screen is the toggle button on the bottom slider. The toggle switch has replaced a button that served as an enter or an exit button to explore the inside map of the buildings. The toggle switch will only be replaced when it will enter the direction and the navigation mode.

6. Testing Plan and Report

6.1 Unit Testing

Discuss how you will be conducting unit testing (the tools) and what are the relevant units to test. If a component or method cannot be tested with unit tests, explain why not and how else you can test it. At the end of each Sprint include a report with the number of unit tests


GitHub Actions / Unit Test Results
succeeded 20 hours ago in 1s

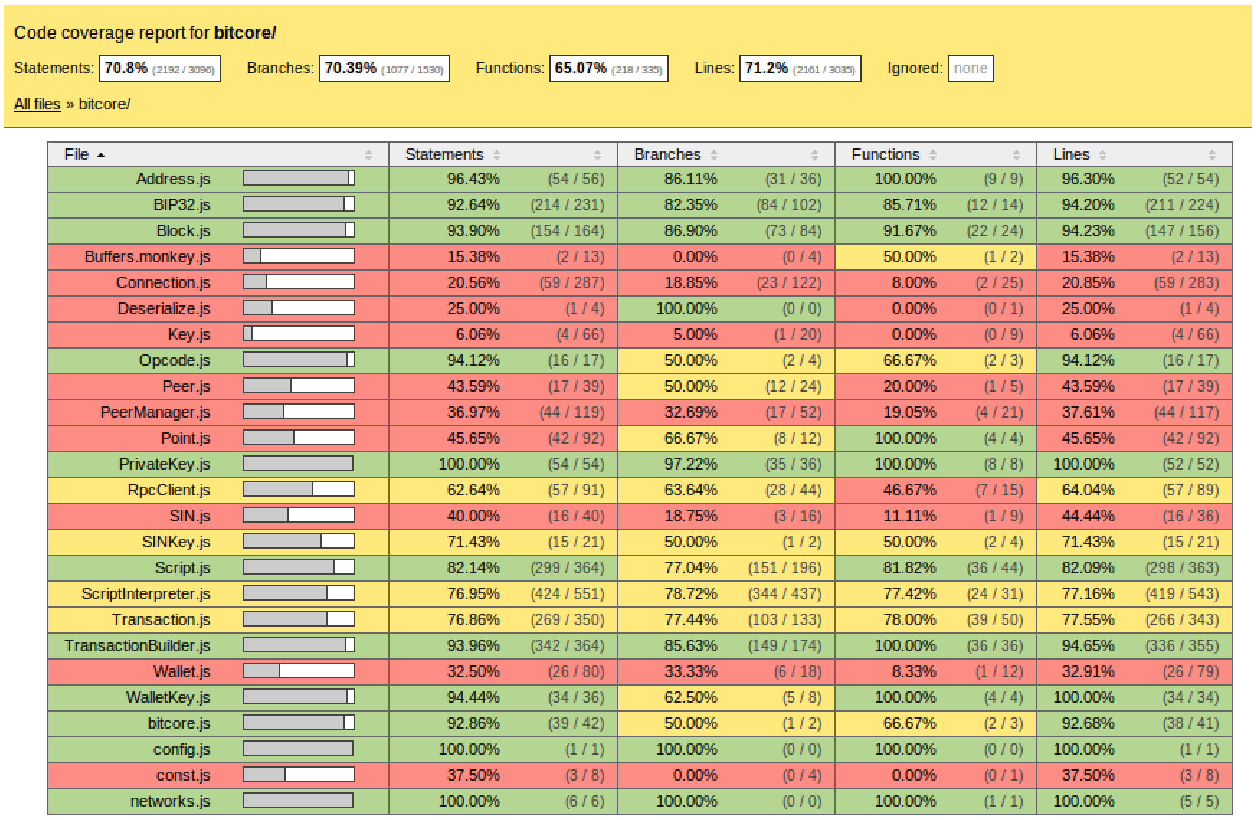
1891 passed, 0 failed and 11 skipped

tests 1891 passed, 11 skipped

Report	Passed	Failed	Skipped	Time
build/test-results/test/TEST-org.kohsuke.github.GHRepositoryWrapperTest.xml	10 ✓			13s
build/test-results/test/TEST-org.refactoringminer.astDiff.tests.Defects4JPerfectDiffTest.xml	691 ✓			578s
build/test-results/test/TEST-org.refactoringminer.astDiff.tests.Defects4JProblematicCasesTest.xml	109 ✓			130s
build/test-results/test/TEST-org.refactoringminer.astDiff.tests.RefactoringOraclePerfectDiffTest.xml	150 ✓			197s
build/test-results/test/TEST-org.refactoringminer.astDiff.tests.RefactoringOracleProblematicCasesTest.xml	51 ✓			290s
build/test-results/test/TEST-org.refactoringminer.astDiff.tests.SpecificCasesTest.xml	16 ✓			42s
build/test-results/test/TEST-org.refactoringminer.astDiff.tests.TreeFromParserTest.xml	1 ✓			57ms
build/test-results/test/TEST-org.refactoringminer.astDiff.tests.TreeMatcherTest.xml	7 ✓			67ms
build/test-results/test/TEST-org.refactoringminer.test.TestAllRefactorings.xml			1 ●	1ms
build/test-results/test/TEST-org.refactoringminer.test.TestAllRefactoringsByCommit.xml	549 ✓			830s
build/test-results/test/TEST-org.refactoringminer.test.TestAllRefactoringsByCommitForPurity.xml	139 ✓			525s
build/test-results/test/TEST-org.refactoringminer.test.TestCommandLine.xml	8 ✓		5 ●	53s
build/test-results/test/TEST-org.refactoringminer.test.TestJavadocDiff.xml	13 ✓			12s
build/test-results/test/TEST-org.refactoringminer.test.TestParameterizeTestRefactoring.xml	12 ✓		1 ●	915ms
build/test-results/test/TEST-org.refactoringminer.test.TestParameterizeTestRefactoring$TestCheckForTestParameterizations_OneStringParam_Plugin.xml	2 ✓			77ms
build/test-results/test/TEST-org.refactoringminer.test.TestParameterizeTestRefactoring$TestCsvFileSource_AbsolutePath.xml	7 ✓			2s
build/test-results/test/TEST-org.refactoringminer.test.TestParameterizeTestRefactoring$TestCsvFileSource_OtherPathFormats.xml			4 ●	4ms
build/test-results/test/TEST-org.refactoringminer.test.TestStatementMappings.xml	126 ✓			549s

6.2 Test code coverage

Add a screenshot from the tool you use for computing test code coverage. The report should include statement/line coverage, branch coverage, function/method coverage.


Code coverage report for **bitcore/**

Statements: **70.8%** (2192 / 3096) Branches: **70.39%** (1077 / 1530) Functions: **65.07%** (218 / 335) Lines: **71.2%** (2161 / 3035) Ignored: none

All files » bitcore/

File	Statements		Branches		Functions		Lines	
Address.js	96.43%	(54 / 56)	86.11%	(31 / 36)	100.00%	(9 / 9)	96.30%	(52 / 54)
BIP32.js	92.64%	(214 / 231)	82.35%	(84 / 102)	85.71%	(12 / 14)	94.20%	(211 / 224)
Block.js	93.90%	(154 / 164)	86.90%	(73 / 84)	91.67%	(22 / 24)	94.23%	(147 / 156)
Buffers.monkey.js	15.38%	(2 / 13)	0.00%	(0 / 4)	50.00%	(1 / 2)	15.38%	(2 / 13)
Connection.js	20.56%	(59 / 287)	18.85%	(23 / 122)	8.00%	(2 / 25)	20.85%	(59 / 283)
Deserialize.js	25.00%	(1 / 4)	100.00%	(0 / 0)	0.00%	(0 / 1)	25.00%	(1 / 4)
Key.js	6.06%	(4 / 66)	5.00%	(1 / 20)	0.00%	(0 / 9)	6.06%	(4 / 66)
Opcode.js	94.12%	(16 / 17)	50.00%	(2 / 4)	66.67%	(2 / 3)	94.12%	(16 / 17)
Peer.js	43.59%	(17 / 39)	50.00%	(12 / 24)	20.00%	(1 / 5)	43.59%	(17 / 39)
PeerManager.js	36.97%	(44 / 119)	32.69%	(17 / 52)	19.05%	(4 / 21)	37.61%	(44 / 117)
Point.js	45.65%	(42 / 92)	66.67%	(8 / 12)	100.00%	(4 / 4)	45.65%	(42 / 92)
PrivateKey.js	100.00%	(54 / 54)	97.22%	(35 / 36)	100.00%	(8 / 8)	100.00%	(52 / 52)
RpcClient.js	62.64%	(57 / 91)	63.64%	(28 / 44)	46.67%	(7 / 15)	64.04%	(57 / 89)
SIN.js	40.00%	(16 / 40)	18.75%	(3 / 16)	11.11%	(1 / 9)	44.44%	(16 / 36)
SINKey.js	71.43%	(15 / 21)	50.00%	(1 / 2)	50.00%	(2 / 4)	71.43%	(15 / 21)
Script.js	82.14%	(299 / 364)	77.04%	(151 / 196)	81.82%	(36 / 44)	82.09%	(298 / 363)
ScriptInterpreter.js	76.95%	(424 / 551)	78.72%	(344 / 437)	77.42%	(24 / 31)	77.16%	(419 / 543)
Transaction.js	76.86%	(269 / 350)	77.44%	(103 / 133)	78.00%	(39 / 50)	77.55%	(266 / 343)
TransactionBuilder.js	93.96%	(342 / 364)	85.63%	(149 / 174)	100.00%	(36 / 36)	94.65%	(336 / 355)
Wallet.js	32.50%	(26 / 80)	33.33%	(6 / 18)	8.33%	(1 / 12)	32.91%	(26 / 79)
WalletKey.js	94.44%	(34 / 36)	62.50%	(5 / 8)	100.00%	(4 / 4)	100.00%	(34 / 34)
bitcore.js	92.86%	(39 / 42)	50.00%	(1 / 2)	66.67%	(2 / 3)	92.68%	(38 / 41)
config.js	100.00%	(1 / 1)	100.00%	(0 / 0)	100.00%	(0 / 0)	100.00%	(1 / 1)
const.js	37.50%	(3 / 8)	0.00%	(0 / 4)	0.00%	(0 / 1)	37.50%	(3 / 8)
networks.js	100.00%	(6 / 6)	100.00%	(0 / 0)	100.00%	(1 / 1)	100.00%	(5 / 5)

6.3 Acceptance Testing

For each acceptance test you will make an issue and the issue will be labeled as “acceptance test”. The acceptance test will clearly describe the acceptance criteria. When you demo a feature to your product owner, you will go through the acceptance criteria step by step. If the product owner is satisfied with the feature, he/she can sign-off by adding a comment to the issue with his/her account. If the product owner is not satisfied with the feature, he/she should add comments explaining his/her concerns and suggesting changes.

For each release, you should provide a link with the acceptance tests signed-off or commented by the product owner for further improvements.
In GitHub you can simply add the following suffix to your project url
issues?q=**label**%3Aacceptance-test+**is**%3Aclosed+**milestone**%3A"Sprint+3"

where **label** indicates the label you use for acceptance tests (typically acceptance-test)
is indicates the status of the acceptance test (open or closed)
milestone indicates the Sprint (typically acceptance tests should be done in the last sprint of each release)

6.4 System Tests

Discuss what automation tools you will use to develop your system tests. Here you will be testing individual features by interacting with the system itself. Develop a set of steps that will test the full functionality of a feature (user story).
For each system test, you will record your web/mobile/desktop application or service, as shown in this link You will add the recording in your repository, and provide a link

6.5 Usability~~/Performance~~ testing

6.5.1 Metrics

List the metrics that you will use along with a description/definition.

6.5.2 Tools

Discuss the research you did on tools, and justify the selection of the tool(s) you will use.

6.5.3 User group characteristics/Load characteristics

For usability testing, present the characteristics (gender, age, technical expertise, etc.) of the test user group. If the test users are not the actual end-users, show that the characteristics of the test users are similar to those of the intended end-users.
~~For performance testing, explain your testing load characteristics, and how these meet the requirements set by the product owner.~~

6.5.4 Test results

Depending on the tool(s) you selected, there are different kinds of generated reports.
In Release #3 you will add the reports after your UI~~/Performance~~ improvements and compare with the previous ones.

6.5.5 Analysis and actions

Analyze the reports and explain the actions you will take for UI~~/Performance~~ improvements.
In Release #3, include pull requests implementing the actions for UI~~/Performance~~ improvements.

7. Defect Tracking Report

For each release you will include 3 links with the bug reports for each Sprint.
In GitHub you can simply add the following suffix to your project url
issues?q=**label**%3Abug+**is**%3Aclosed+**milestone**%3A"Sprint+3"

where **label** indicates the label you use for bug-related issues (typically bug)
is indicates the status of the bug (open or closed)
milestone indicates the Sprint

8. Code review report

For each release, include the links to your **top-5 pull requests** with the most extensive and in-depth code reviews. Such pull requests should include discussions about code quality issues (potential bugs, incorrect API usage, complex code, long functions, poor variable names, bad practices), by making comments directly on specific source code snippets and giving suggestions on how to improve them.
In this link you can see an example of a code review generated by Amazon CodeGuru, which discusses a potential concurrency bug, due to the wrong use of the ConcurrentHashMap API.
For each pull request give a **brief** overview about the issues discussed during the code review and how they have been resolved. Make sure that all team members are involved in code reviews.

9. Refactoring report

For each Sprint, list the commits containing refactoring information. The commit links should point exactly to the lines where the refactoring took place, as shown in the Table below. You should also provide a brief explanation about the reason(s) you performed each refactoring.
The complete list of refactoring types is available at https://refactoring.com/catalog/
To keep track of your refactoring activity more effectively, you are advised to **tag the commits** including refactoring with [Refactoring] prefix in the commit message.

Refactoring type	**Commit**
Extract Method	https://github.com/spring-projects/spring-boot/commit/becce#diff-181e25db0419c2507d3e3ff6a24969388d5ab9e2c43aa6aaf27753418a801eb0R172
Extract Variable	https://github.com/spring-projects/spring-boot/commit/becce#diff-f6cc88b09865ca5605bc9ee8cce9b5cba3f839602157876c779dadf7499bc17fR137

10. Quality Measurements

At the end of each Sprint record the value for each of the following metrics:
size, duplication, documentation, complexity, coupling, cohesion, technical debt, security vulnerabilities, and code convention violations.

You should take actions in each Sprint to improve these metrics. List the commits and pull requests in which technical debt, security vulnerabilities, and code convention violations have been addressed. Link to your Refactoring report actions taken to improve complexity, duplication, coupling, cohesion, readability and other design/code quality metrics.

Finally, create a chart showing the evolution of each metric over the Sprints.

11. Professional ethics and accountability

Discuss if your software is related with ethical concerns and how you address those. For example, software that could have some negative impacts to people or society, such as health-related, environment-related, or financial-related software.

Appendix

If you are developing a **machine learning** or **deep learning based solution**, you should adjust your report as follows:

1. In the **architecture section**, the experiment design should be discussed. In particular, you will explain your process for data collection, data validation, data/noise cleaning. You will provide your machine/deep learning network architecture diagram and discuss about the deep learning design patterns you applied. You will list the machine learning algorithms that will be used. If these algorithms are available in libraries, I expect a discussion of alternative libraries and a justification about why specific libraries were selected.

2. For the implementation, you will commit in your repository the code you developed for processing the data, the code you developed for training and testing the machine learning model, and your training and testing datasets.

3. In the testing section, you will present all your experiment results. For example, you might have different configuration parameters for the machine learning algorithms or different training/testing datasets that produce different results. To measure the performance of any machine learning algorithm, you must have labelled data points (both positive and negative cases) that will be used to train and test the model. Typically, the performance of a model is measured with metrics, such as precision, recall, F-measure.

4. In the defect tracking, code review, and refactoring sections, you will include the bug reporting/fixing, code reviewing, and refactoring activities you performed on the machine learning code you implemented. Even if your code is in the form of R or Python scripts, there is still a lot of space to improve its quality.

5. In the quality measurements section, you will include the metrics reported by Linters and static analysis tools for your machine learning code. To the best of my knowledge SonarQube supports Python.

6. For the User interface design section, if you will have a user interface you can still create mockups. If your solution is intended to be used as an API or service, in this section you will discuss about your API design instead (explain how it meets the user needs, provide snippets to show your API usage scenarios).