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.

# 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.

| 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 |  | 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 shape is self-explanatory. You don’t need a full page for it.]

**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](https://en.wikipedia.org/wiki/Software_architecture" \l "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

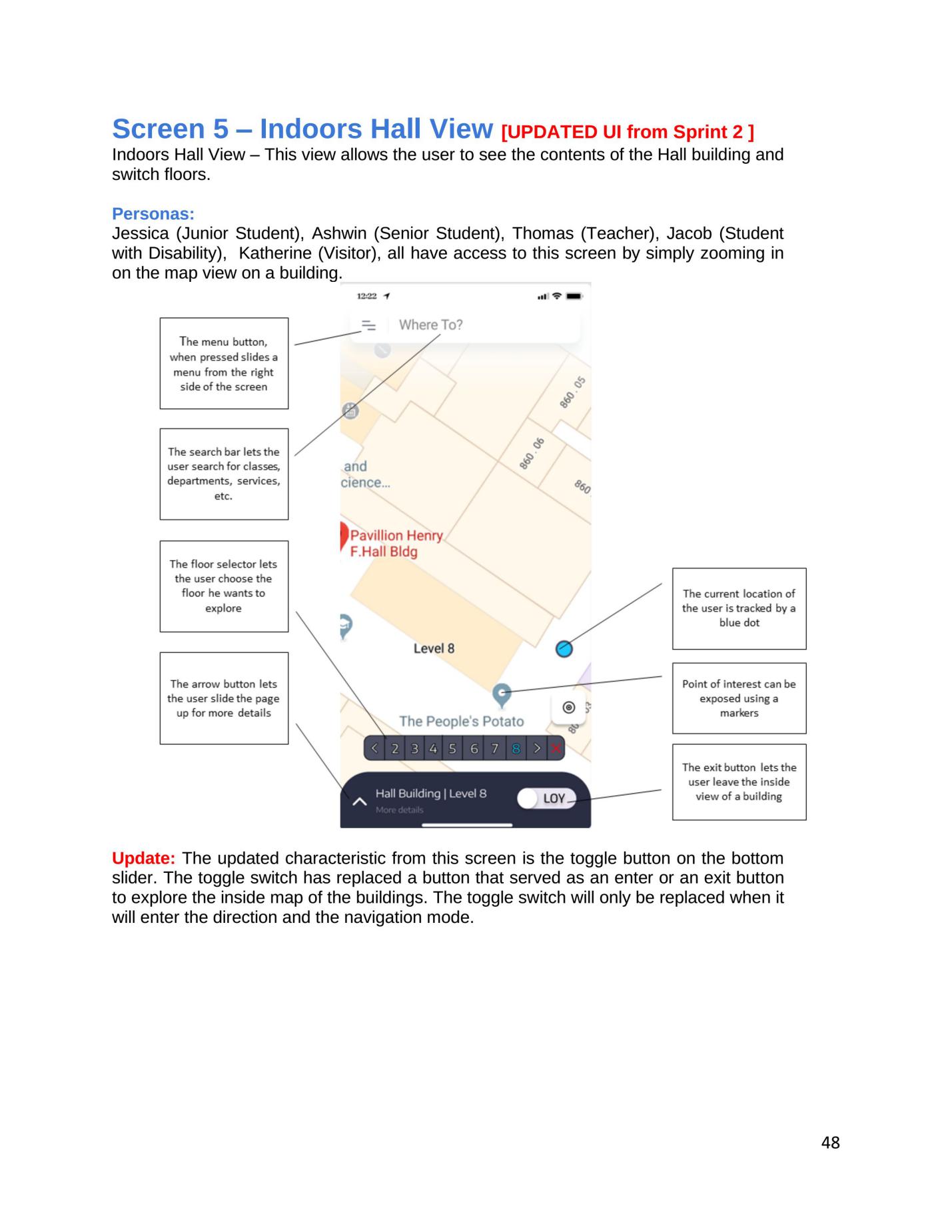
# 5.1 Personas

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 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.

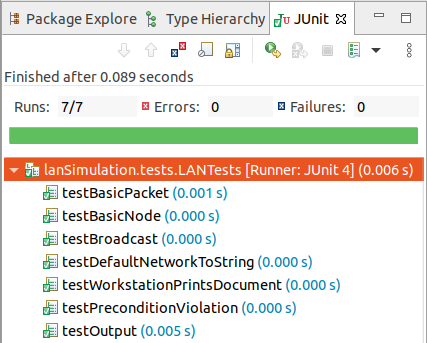
**Sample mockup from SOEN 390 project GuideMe**

****

# 6. Testing Plan and Report

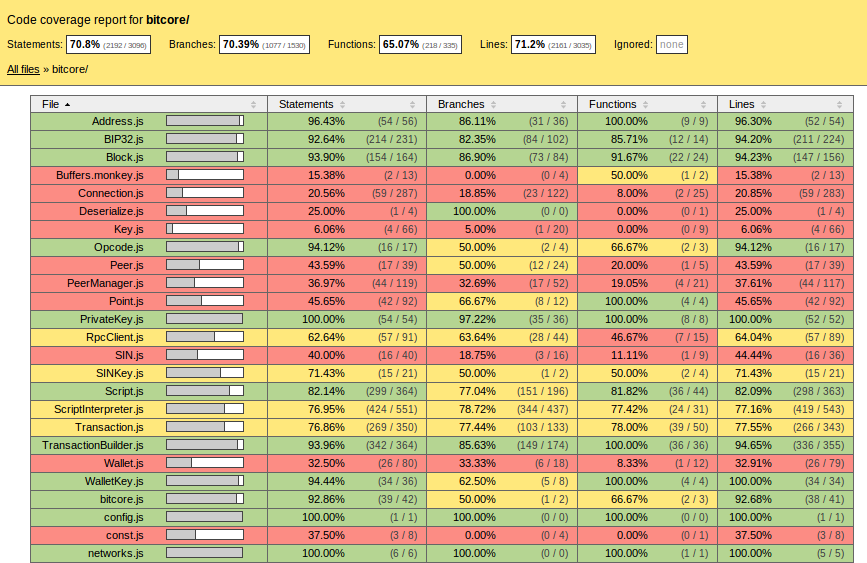
# 6.1 Unit Testing

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



# 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.



# 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](https://github.com/RGPosadas/WayFinder/wiki/Acceptance-Tests-GIFs) 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](https://techcrunch.com/wp-content/uploads/2020/06/Screenshot_Catch-Code-Issue_2-Annotations-LP.df0deb64bbfb02219d429db4d5bd3efd089e2a89.png) 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.

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](https://datascience.stackexchange.com/questions/14899/how-to-draw-deep-learning-network-architecture-diagrams) and discuss about the [deep learning design patterns](https://www.manning.com/books/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).