Iterations	Evaluation criteria	Weight
Release 1 **Jan 13 - Feb 9** Sprint #1 Jan 13 - Jan 26 Sprint #2 Jan 27 - Feb 9	**-Repository setup and Continuous Integration** (You should setup a GitHub repository and GitHub actions that automate your test executions, test code coverage with CodeCov): **2** points **-User Stories Backlog** (You should create user stories for **all project requirements**. Before the beginning of each Sprint you will plan the user stories to be completed and estimate their user story points using planning poker): **3** points **-Agile Planning** (Each release consists of 2 Sprints and each Sprint is 2-weeks long. For each Sprint you will include the Sprint summary, burndown chart and retrospective. All actions in the retrospective should be justified based on the "lessons learned" from the previous sprints. You must use an Agile planning software, such as ZenHub, which provides a Board and Agile metrics, and you must provide access to the instructor): **2** points **-Software Architecture** (Domain Model, Component Diagram, ER Model if needed, analysis of external libraries that will be used with a justification about their pros and cons): **4** points **-UI Prototypes** (Mockups linked to each user story of the Sprint, all mockups will be annotated to explain the functionality of buttons/inputs/outputs, Persona descriptions, UI variations based on the personas, Show updates to the UI mockups after receiving feedback from the product owner): **5** points **-Unit/System Testing Report** (Unit test coverage report for the current release, Record automated System/UI tests as shown in https://github.com/RGPosadas/WayFinder/wiki/Acceptance-Tests-GIFs): **3** points **-Issue Tracker** (Create bug reports with replication steps, screenshots, links to the commits fixing the bug for traceability, links to commits updating tests to cover the reported bug. All issues related to bugs should be tagged with "bug" label, issues related to performance should be tagged with "performance" label): **2** points **-Pull Requests and Code reviewing** (All features will be contributed in the form of pull requests. Each pull request will have assigned code reviewers and discussions between the reviewers and authors about potential improvements related to code/design quality): **3** points **-Code Quality and Security Report** (Software metrics [size, duplication, documentation, complexity, coupling, cohesion] evolution, Report the technical debt, security vulnerabilities, and code convention violations for the current release. List the commits and pull requests in which technical debt, security vulnerabilities, and code convention violations have been addressed. The commits and pull requests should explicitly mention/discuss/motivate the type(s) of technical debt addressed). You can use SonarQube for automatically collecting metrics and technical debt: **3** points **-Presentation to the Instructor** (the content of the presentation will be announced before the release): **3** points	30%

Suggested Plan for Sprint #1

1. Setup your GitHub repository and yml scripts for GitHub workflows
2. Create your project Backlog in ZenHub (make a user story for each project requirement)
3. Define the **personas** representing potential users and describe some of the tasks they will perform on the system. You should consider 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.
4. Create UI mockups for the user stories you will implement in Sprint 2
5. Think about and document the design of your app (basic domain model, component diagram). Explore external libraries/frameworks that might be useful for your project and discuss their pros and cons.
6. Plan your Sprint 2 in ZenHub by adding some user stories from your backlog (add features than can be completed within 2 weeks)
7. Conduct and document your Sprint 1 Retrospective. Meet with your product owner and get feedback for your mockups and your backlog

Suggested Plan for Sprint #2

1. Implement the planned features
2. Write unit tests for your code
3. Submit your features in Pull Requests. Assign team members for code review. There should be active discussions and suggestions for improvements.
4. Create UI mockups for the user stories you will implement in Sprint 3
5. Investigate how you can make automated System/UI tests. In the meantime, some team members will do manual system testing for the implemented features and report well documented bug reports on GitHub as issues.
6. Install and run SonarQube for your project. Summarize the issues detected by SonarQube and discuss how and when you will address them.
7. Conduct and document your Sprint 2 Retrospective. Meet with your product owner for acceptance testing and getting feedback
8. Prepare your presentation for the Instructor and practice it with all team members at least once.
9. Submit your Release 1 report on Slack
10. Plan your Sprint 3 in ZenHub by adding user stories from your backlog (add features than can be completed within 2 weeks)

Release 2 **Feb 10 - Mar 9** Sprint #3 Feb 10 - Feb 23 Sprint #4 Feb 24 - Mar 9	**-Agile Planning**: **2** points **-Software Architecture** (Discuss about the source code implementation of your design patterns and how you extended these patterns over time as new requirements were implemented): **2** points **-UI Prototypes**: **3** points **-Unit/System Testing Report**: **3** points **-Usability/Performance testing** (Develop code in your app or use tools for collecting user session recordings. Analyze the collected data to find ways to improve the UI experience): **5** points **-Issue Tracker**: **2** points **-Refactoring activity** (List the commits and pull requests in which refactoring took place. The commits and pull requests should explicitly mention/discuss/motivate the type(s) of refactoring. The complete list of refactoring types is available at https://refactoring.com/catalog/: **2** points **-Code reviewing**: **4** points **-Code Quality and Security Report**: **3** points **-Presentation to the Instructor**: **4** points	30%

Suggested plan for Sprint #3

1. Implement the planned features
2. Write unit tests for your code
3. Submit your features in Pull Requests. Assign team members for code review. There should be active discussions and suggestions for improvements. These suggestions could also include refactorings. Document systematically your refactoring efforts by collecting commits and PRs with specific types of refactorings
4. Create UI mockups for the user stories you will implement in Sprint 4
5. At this point you should have automated System/UI tests for all implemented features so far. You will do manual system testing for the ongoing features and report well documented bug reports on GitHub as issues.
6. Investigate potential design patterns that could be useful for your project. Justify their need in your project.
7. Investigate how you will conduct usability testing and create the required infrastructure (setup and try usability testing tools).
8. Plan your Sprint 4 in ZenHub by adding some user stories from your backlog (add features than can be completed within 2 weeks. Note that in the next Sprint you will conduct usability testing. Make a realistic plan)
9. Conduct and document your Sprint 3 Retrospective. Meet with your product owner for acceptance testing and getting feedback

Suggested plan for Sprint #4

1. Implement the planned features following the same practices (PRs, code reviews, unit tests)
2. Introduce design patterns in your project. Document commits where your design pattern instances were introduced and evolved (adding new abstract methods, adding new subtypes)
3. Address technical debt and security warnings from SonarQube. Update your SonarQube (code quality) report
4. Create UI mockups for the user stories you will implement in Release 3 (Sprints 5 and 6).
5. Add automated System/UI tests for the newly implemented features.
6. Recruit users and conduct your first usability testing. Collect and analyze your usability test results. Make well documented issues for UI improvements and bugs found during testing.
7. Conduct and document your Sprint 4 Retrospective. Meet with your product owner for acceptance testing and getting feedback
8. Prepare your presentation for the Instructor and practice it with all team members at least once.
9. Submit your Release 2 report on Slack
10. Plan your Sprint 5 in ZenHub by adding some user stories from your backlog (add features than can be completed within 2 weeks)

Release 3 **Mar 10 - Apr 6** Sprint #5 Mar 10 - Mar 23 Sprint #6 Mar 24 - Apr 6	**-Agile Planning**: **2** points **-Unit/System Testing Report** (At this point all user stories should be automated): **4** points **-Usability/Performance testing** (Implement the improvement opportunities you found in the previous release and rerun your usability/performance tests to measure the improvements quantitatively): **5** points **-Issue Tracker** (At this point all open bugs, performance issues should be addressed): **3** points **-Refactoring activity**: **4** points **-Code reviewing**: **4** points **-Code Quality and Security Report** (At this point all technical debt, security vulnerabilities, code convention violations, and code smells should be addressed. If not addressed, you should explain the reasons): **4** points **-Presentation to the Instructor**: **4** points	30%

Suggested plan for Sprint #5

1. Address all UI issues discovered in usability testing
2. Implement the remaining planned features
3. Write unit tests for the new features
4. Submit your features in Pull Requests. Assign team members for code review. There should be active discussions and suggestions for improvements. These suggestions could also include refactorings. Document systematically your refactoring efforts by collecting commits and PRs with specific types of refactorings
5. Add automated System/UI tests for the newly implemented features.
6. Address technical debt and security warnings from SonarQube. Update your SonarQube (code quality) report

7. Plan your Sprint 6 in ZenHub by adding any remaining user stories from your backlog. 8. Conduct and document your Sprint 5 Retrospective. Meet with your product owner for acceptance testing and getting feedback		
Suggested plan for Sprint #6 1. Conduct a second round of usability testing. Compare the current usability test results with the previous ones to show the improvement in user satisfaction or other usability metrics. 2. Add automated System/UI tests for all implemented features. 3. Address all remaining technical debt and security warnings from SonarQube. Update your SonarQube (code quality) report 4. Conduct and document your Sprint 6 Retrospective. Meet with your product owner for any remaining acceptance testing 5. Prepare your presentation for the Instructor and practice it with all team members at least once. 6. Submit your Release 3 report on Slack		
Acceptance tests	Percentage of the number of user stories signed-off by the product owner through acceptance tests over the total number of user stories in the backlog. **For each user story there should be an acceptance test in the issue tracker of the project. The test is considered as accepted if it is signed-off with the account of the product owner by making an accept comment on the corresponding issue.**	10%

What are the evaluation criteria?

User stories backlog: The user stories should follow the format
As a [user description], I want to [functionality], so that [goal/benefit].
Also, the acceptance criteria should be crystal clear, so that there are no misunderstandings between the product owner and the team during the acceptance testing.
The Backlog should be complete and cover all project requirements by Sprint #1.

Agile planning: A plan is considered **successful** when all planned user stories are complete and accepted by the end of the Sprint. A plan is considered **ideal** if it follows the “ideal” line in the burndown chart, which can be achieved by completing tasks throughout the Sprint, and avoid completing all tasks close to the end of the Sprint.
Moreover, the retrospectives should be of high quality, and result in action items that help to improve the productivity, resource management, communication, collaboration of your team.
A retrospective is considered **successful** if it results in action items that can improve the team practices in a measurable way.
It is highly recommended to use online tools for Agile retrospectives
https://echometerapp.com/en/retrospective-tools-online/

Software Architecture: The design diagrams should have no mistakes, or incorrectly used notations. Moreover, they should be complete and cover all requirements of the project. You should also investigate alternative external libraries/frameworks that could be utilized in your project, and select the most appropriate ones based on their pros and cons. You should do a thorough investigation about design patterns that could be useful for your project and justify their need with reasonable and pragmatic arguments. Design patterns are **universal best design practices** that can be applied in every programming paradigm (object-oriented, functional).

UI prototyping: You must have a UI mockup for each 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. The documentation template includes an example of how you are expected to document the mockups.
Your personas and UI design should take into consideration people with disabilities (color-blind, blind, deaf, people with hand injuries, people in wheelchair), illiteral people, or marginalized people. Such consideration will make your app accessible and usable by more people.

You should use specialized tools for creating mockups (e.g., moqups)
https://zapier.com/blog/best-wireframe-tools/
Moqups is highly recommended as it can model the execution flow between mockup screens.

Unit/System Testing: Unit test code coverage should be maintained at a relatively high percentage. You should automate system tests for all your user stories, using tools, such as appium, cypress, selendroid, espresso
https://www.headspin.io/blog/top-automated-mobile-testing-tools
Ideally, your system tests should execute along with your unit tests in your CI pipeline.

Usability testing: You should investigate different usability testing methods and tools.
https://maze.co/guides/usability-testing/methods/
You should create usability test scenarios and recruit users to run these scenarios. You can collect both quantitative and qualitative data, which can be used to compare the usability of different versions of your app.

Issue tracking: Bug reports should be documented in detail with clear replication steps, screenshots (if applicable). The commits fixing the bug or adding a test for the bug fix should be linked to the issue using #Issue_number. GitHub labels should be used to distinguish between functionality-related bugs and performance bugs.
You should use bug templates to make sure that bug reports are consistently documented.
https://docs.github.com/en/communities/using-templates-to-encourage-useful-issues-and-pull-requests/configuring-issue-templates-for-your-repository
Templates that you can reuse in your project
https://github.com/devspace/awesome-github-templates

Refactoring activity: Apply a variety of refactoring types including low-level (Extract Variable, Rename Variable), medium-level (Extract Method), and high-level (Extract Class, Extract Component). Document the commits were you applied refactorings. Promote refactoring activities in Pull Request discussions.

Code reviewing: Discuss about code quality issues in your reviews. Recommend changes on specific lines of the code using the GitHub code reviewing tool
https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/reviewing-proposed-changes-in-a-pull-request
Make sure that all team members participate in code reviews throughout the project (not in every single PR) to share knowledge and learn best practices from each other.

Code Quality and Security: Utilize tools, such as SonarQube to keep track of the technical debt and security issues in your project. You should try to address most of your technical debt and security issues. Your should provide justification for unaddressed technical debt (e.g., false positives, or trivial issues).

Presentation: Your presentations will be evaluated with the following criteria
Punctuality
content coverage as requested
preparation for delivering the presentation
coordination among multiple presenters
presentation skills
addressing comments/questions