Change Log

|  |  |  |
| --- | --- | --- |
| Change | Date of Change | Author |
| Baseline | 8/28/22 | Pauline Wade |
|  |  |  |
|  |  |  |

Final Report

Product Name:

Customer Name

Team Member Names :   
<List Names>

CSCE 431 – Software Engineering  
<Semester>

Texas A&M University

Department of Computer Science and Engineering

Contents

[1 Project Scope 5](#_Toc112706462)

[2 Stakeholder Analysis 6](#_Toc112706463)

[2.1 Stakeholders 6](#_Toc112706464)

[3 Implementation Environment 7](#_Toc112706465)

[3.1 Hardware 7](#_Toc112706466)

[3.2 Software 7](#_Toc112706467)

[4 Scrum Process 8](#_Toc112706468)

[4.1 Release Planning 9](#_Toc112706469)

[4.2 Sprint Planning 9](#_Toc112706470)

[4.3 Scrum Meetings 9](#_Toc112706471)

[4.4 Sprint Review 9](#_Toc112706472)

[4.5 Sprint Retrospective 10](#_Toc112706473)

[5 Sprint 1: 11](#_Toc112706474)

[5.1 Overall Experience 11](#_Toc112706475)

[5.2 Stakeholder Management and Communication Plan 11](#_Toc112706476)

[5.3 Risk Mitigation, Monitoring, & Management Plan 12](#_Toc112706477)

[5.4 Release Planning 12](#_Toc112706478)

[5.5 Sprint Planning 12](#_Toc112706479)

[5.6 Scrum Meetings 12](#_Toc112706480)

[5.7 User Stories 13](#_Toc112706481)

[5.8 UX Models 13](#_Toc112706482)

[5.9 Data Design 14](#_Toc112706483)

[5.10 Version Control 14](#_Toc112706484)

[5.11 Quality 15](#_Toc112706485)

[5.11.1 Compliance 15](#_Toc112706486)

[5.11.2 Efficiency 15](#_Toc112706487)

[5.11.3 Correctness 15](#_Toc112706488)

[5.11.3.1 Test-Driven Development (TDD): 15](#_Toc112706489)

[5.11.3.2 Reviews: 15](#_Toc112706490)

[5.11.3.3 Definition of Done 16](#_Toc112706491)

[5.11.3.4 Test Coverage 17](#_Toc112706492)

[5.11.3.4.1 Code Coverage - Output of simplecov 18](#_Toc112706493)

[5.11.3.5 Test Results 18](#_Toc112706494)

[5.11.4 Maintainability 19](#_Toc112706495)

[5.11.4.1 Coding Standard 19](#_Toc112706496)

[5.11.4.2 Well Documented Code 20](#_Toc112706497)

[5.11.4.3 Linter Output 20](#_Toc112706498)

[5.11.5 Integrity 20](#_Toc112706499)

[5.11.5.1 Security Risk Analysis Table 20](#_Toc112706500)

[5.11.5.2 Authentication 20](#_Toc112706501)

[5.11.5.3 Integrity Test Coverage 20](#_Toc112706502)

[5.11.5.4 Integrity Test Results 21](#_Toc112706503)

[5.11.6 Usability 21](#_Toc112706504)

[5.11.6.1 Usability Requirements 21](#_Toc112706505)

[5.11.6.2 Usability Test Coverage 21](#_Toc112706506)

[5.11.6.3 Usability Test Results 21](#_Toc112706507)

[5.12 Monitoring and Control 22](#_Toc112706508)

[5.13 Deploying App 22](#_Toc112706509)

[5.14 Sprint Review 22](#_Toc112706510)

[5.15 Sprint Retrospective 22](#_Toc112706511)

**Recommended content for reflection:**

-Describe briefly what the activity/process/topic was about.

* How you felt as you were going through it.
* How does it help you personally & professionally?

-Describe what you did right (i.e., a WHOOP moment).

* How did you celebrate?

-Describe what you would do differently (i.e., how would you avoid that EPIC FAIL).

Remember that it’s best to FAIL – FAST or FAIL - NOW, instead of FAIL – SLOW or FAIL - LATER (e.g., in front of the customer)

* How will this help you avoid the same mistake next time?

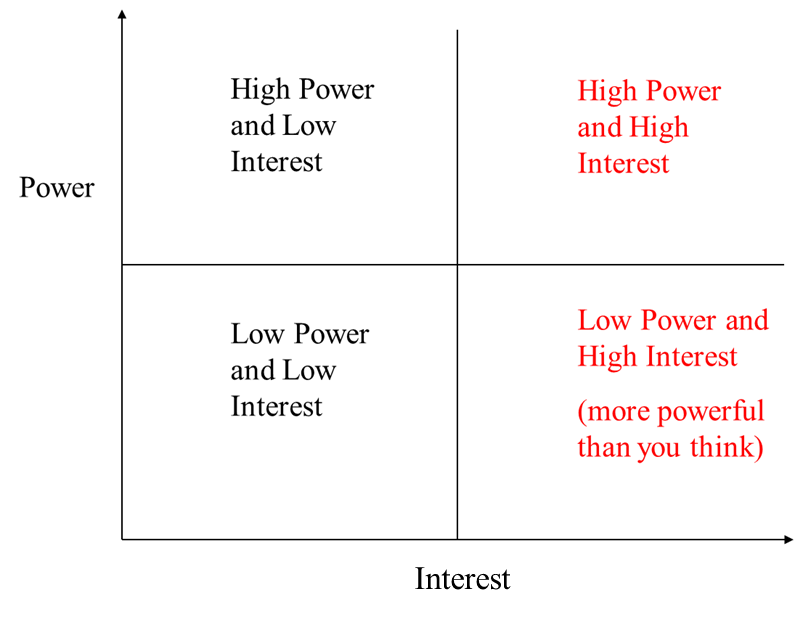
# Project Scope

<Overall reflection on the process of developing the scope using the recommended content for reflection listed below the table of contents.>

|  |  |
| --- | --- |
| **SECTION 1: PROJECT SUMMARY, DELIVERABLES, AND SCOPE EXCLUSION** | |
|  |  |
| **Project Summary** | |
| Organization Name: Aggie Pregnant & Parenting Student Organization (APPSO)  Primary Contact Name: Judith Tijerina  Officer position in the organization: Social Media Coordinator  Email: [juditijera17@tamu.edu](mailto:juditijera17@tamu.edu)  Contact Number: (956)-607-3116  Names of members of the project team: Thierry David, Macy Drew, Mualla Argin, Dillon Lee, Nathan Patterson  Summary of problem to be solved:  Track Texas A&M student parents’ participation in APPSO events, as well as their financial status and circumstances, in order to aid officer decision of quality scholarship candidates. | |
|  |  |
| *<Deliverables are tangible things that the project will produce, stated at a high level. They describe what is included in the scope and what the customer will get when the project is done. Provide a list the deliverables below.>* | |
| **Deliverables** | |
| **Deliverable Number** | **Description** |
| *[#]* | *[function or feature e.g., Keep information about each member, Track member participation, etc.]* |
| **1** | **Track member participation (attendance, etc.)** |
| **2** | **Record/update members’ scholarship eligibility and information** |
| **3** | **Authenticate both members and officers for access to profile data** |
| **4** | **Improve outreach to the community for the organization via the website and Parent Portal** |
| **5** | **Make website accessible on/off campus for TAMU students** |
| **6** | **Allow for saving a draft of the scholarship application and returning later** |
| **7** | **Display member information appropriately to members and officers** |
|  |  |
|  |  |
| *<It is important to also state exclusions, or what will not be included in the project. List the exclusions along with their reasons.>* | |
| **Scope Exclusion** | |
| **Exclusion** | **Reason for Assumption** |
| non-TAMU CS students | Customer does not award scholarships to non-College Station Texas A&M students |
| Budget | We were asked to focus on the scholarship and outreach aspects of the project |
|  |  |
|  |  |
|  |  |

# Stakeholder Analysis

First step was to identify key stakeholders using the stakeholder grid below, especially those with high power and high interest.



Who are your stakeholders? Individuals / organizations that are:

* Positively or negatively affected by the project
* Actively involved in the project

It is important to know their **interests**, **involvement**, and **impact** on project success / failure

[Overall reflection on this process.]

## Stakeholders

***Client***: NAME, representative of the [org name].

***Advising Faculty***: Professor Pauline Wade

***Teaching Team***: TA name

[Describe any special roles held by specific team members, if any. You can also state what everyone’s role was (e.g., contributed to the code either individually or through pair programing). ]

|  |  |  |
| --- | --- | --- |
| **Scrum Master** | **Product Owner** | **Team Members & Role if any** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Implementation Environment

The implementation environment of our application is described below.

[Overall reflection on the process of determining the environment and the technology chosen.]

## Hardware

[List here hardware used by the team.]

## Software

The following packages and tools were used in the project:

* Operating System x
* Ruby x.y.z
* Rails x.y.z
* Docker version x
* Bootstrap x.y.z (if applicable) – state why this was used
* jQuery-rails x.y.z (if applicable) – state why this was used
* rspec-rails – state why this was used
* Heroku
* PostgreSQL x.y.z
* Git Hub
* RuboCop or Code Climate – state why this was used
* Simplecov – state why this was used
* Brakeman – state why this was used
* React (if applicable) – state why this was used
* Jira – state why this was used e.g., We used Jira to distribute the tasks among the team members and keep track of the progress in the development.
* OTHER aspects of the environment and why it was used?

The following gems were also used:

* [List Here]

# Scrum Process

Scrum is a process model that consists of releasing the product in increments during specific time frames referred to as sprints. [Fill in with more details on scrum.]

[Describe the different scrum ceremonies and the benefits. Reflect on the use of these ceremonies.]

See the following sections for a summary of the different scrum ceremonies.

Certain software development phases are repeated each sprint. Examples are:

* Requirements
* Design
* Design
* Development
* Test
* Deployment

In general, the team iterates across the different phases and do not have to be done in a specific order. For example, testing can (and should) happen simultaneously or before development.

Also, certain documents are prepared throughout the project, then monitored and revised when necessary, in each sprint. Below is a list of a few documents recommended in this class, organized by phases. Documents listed under “Umbrella Documents” may not belong to a specific phase but may (and should) be created / revised throughout the project timeline.

There is no universal set of phases or documents, as it differs for every organization.

Pre- Sprint 1 (Set-up)

* Coding Standard
* Definition of Done
* Scope

Umbrella Documents

* Stakeholder Management & Communication Plan
* Risk Plan
* Metrics Document

Requirements

* User Stories with Acceptance Criteria
* UX Models

Design

* Data Design
* Other Design Models (e.g., architectural, functional, behavioral, etc.)

Implementation

* ReadMe

Test

* Test Cases
* Test Results

Deployment

* Deployment Plan

Maintenance

* Support & Maintenance Plan

[Fill in the missing documents from the phases above. Include your overall reflection on the phases described above and how it is repeated in each sprint.]

## Release Planning

Release Planning involved capturing requirements as user stories which is a common language for all stakeholders, including non-technical customers, on the requirements of the application. Collectively these user stories make up the product backlog.

Jira was the primary tool used, and was helpful in doing the following:   
[List Jira features that the project used]

[Reflect on your experience on release planning.]

[Refer to a later section for the state of the product backlog in each sprint.}

## Sprint Planning

Sprint Planning involved determining team roles and assigning people to each user story.

[Describe what else happened during sprint planning].

Collectively these user stories make up the sprint backlog.

In the initial sprint (e.g., pre-sprint), one of the key activities was the set-up of our implementation environment, consisting of hardware, software, tools, which is described in an earlier section.

The project involved <number> sprints, each with its own sprint backlog.

Refer to a later section below for the state of the sprint backlog in each sprint.

[Reflect on your overall experience of releasing application increments over <number> sprints.]

## Scrum Meetings

[What is the purpose of these meetings? Overall reflection on this process, results, and benefits.]. The meeting should not exceed 15 minutes.

## Sprint Review

For each sprint review, we emailed our customer at least 5 days before our scheduled sprint review, to confirm their availability, with a maximum time allocation of 45 minutes.

Product Owners are required to attend. Everyone else in the team that are available are encouraged to attend (although extra credit was given if everyone in the team attended).

The meeting was done via our MS Teams channel and recorded.  Videos were on, faces visible.

Sprint Review agenda included (with recommended durations):

1. Welcome the Stakeholders – Product Owner welcomes the stakeholders to attend the review and introduces everyone (3 min)
2. Present Review Agenda – Product Owner presents the agenda for the Sprint Review (2 min)
3. Present Product Increments – Development Team presents the product demo that have been implemented in the Sprint (7 minutes)
4. Get Feedback – Product Owner asks the stakeholders for feedback regarding the product that have been delivered (13 minutes)
5. Present Product Backlog – Product Owner presents the top of the Product Backlog to stakeholder to get feedback for the upcoming sprint(s) and solicit feedback from the stakeholders related to the backlog (15 minutes)
6. Conclude meeting – give preliminary dates of future sprint reviews (5 minutes)

[Overall reflection on this process.]

## Sprint Retrospective

You will record this in your Teams channel (approx. 30-45 minutes) and the teaching team will watch it asynchronously (we may also choose to attend). All members need to be present (with videos on / face visible) and need to say something related to the items listed below.  The person in charge of the meeting is the Scrum Master.

The agenda, with recommended durations, included:

1. What went well in the Sprint? (10-15 minutes)
2. What went wrong in the Sprint & for each one, what should we do differently in the next sprint? (15-20 minutes)
3. What we learned in the Sprint? (5-10 minutes)
4. Action plan or next steps with assigned person for each one (10 minutes)

<Overall reflection on this process.>

# Sprint 1:

## Overall Experience

[Describe any notable details that will help the reader appreciate your experience in this sprint.]

## Stakeholder Management and Communication Plan

A key part of the success of the project was the use of the stakeholder management and communication plan.

The development of this plan involved [description of the process].

[Reflect on how this plan was used during each sprint]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Stakeholder management and communication plan** | | | | | | | | | |
| **Stakeholder Name** | **Category** | **Levels of power and interest (according to grid)** | **Requirements / expectations** | **Strategies for gaining support / reducing obstacles** | **Information needed / Document Name** | **Document Format / Medium** | **Contact Person (if different than the stakeholder)** | **How Often / When Due** | **Status** |
|  | Customer |  |  |  |  |  |  |  |  |
|  | Instructor |  |  |  |  |  |  |  |  |
|  | TA |  |  |  |  |  |  |  |  |
|  | Project Team |  |  |  |  |  |  |  |  |

Example below:



## Risk Mitigation, Monitoring, & Management Plan

Risk Analysis was used to identify risks during every sprint, which are potential problems that may occur. For each risk, we estimated the probability of occurrence, and impact should the risk become reality. [Reflect on the benefit of using this method and how it contributes to project success.]

Each risk was ranked based on the risk exposure (probability multiplied by impact), after which a cutoff line was decided, with risks above the cutoff line considered important to mitigate, monitor, and manage. Once the risk table has been established, it was important for the team to constantly monitor the risks, according to the monitoring plan, to see which risks will likely become reality. To avoid the risk from becoming reality, the team developed and implemented the mitigation plan. Once it was apparent that the risk was going to happen, the team attempted to reduce the negative impact by following the management plan.

[Reflect on the use of this process, especially how the RMMM helped reduce negative impact to the project.]

See below the risk table with a plan for mitigating, monitoring and managing each one.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Prob in % | Impact | Risk Mitigation, Monitoring, and Management (RMMM) Plan | Status |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

See example below:



## Release Planning

[Place links to the MS Teams recording here. Do not use Zoom]

## Sprint Planning

[Place links to the MS Teams recording here. Do not use Zoom]

## Scrum Meetings

[Place links to the MS Teams recording here. Do not use Zoom]

## User Stories

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number** | **User Story with it's acceptance criteria** | | | | |
|  | **Persona (Who)** | **Requirement (What)** | **Value (Why)** | **Critical?** | |
|  |  |  |  | **Yes** | **No** |
| 1 |  |  |  |  |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  |  |  |  |  |
| 2 |  |  |  |  |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 3 |  |  |  |  |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| n |  |  |  |  |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Acceptance criteria or ‘conditions of satisfaction’ indicate when our team has been successful in implementing the user’s requirements, stated as user stories

Acceptance Criteria Goals:

* To clarify what the team should build before they start work
* To ensure everyone has a common understanding of the problem
* To help the team members know when the story is complete
* To help verify the story via automated tests

Should include:

* Negative scenarios of the functionality (rainy day)
* Functional and non-functional use cases. Example of a non-functional use case is performance requirements
* What system/feature intends to do
* End-to-end user flow
* Impact of a user story to other user stories (i.e., features)
* UX concerns

[Reflect on the use of acceptance criteria and how it contributes to the success of your project.]

## UX Models

To make sure that user stories are properly understood by the stakeholders (e.g., customers, development team, etc.), it is a common practice to supplement with UX models, such as lo-fi diagrams.

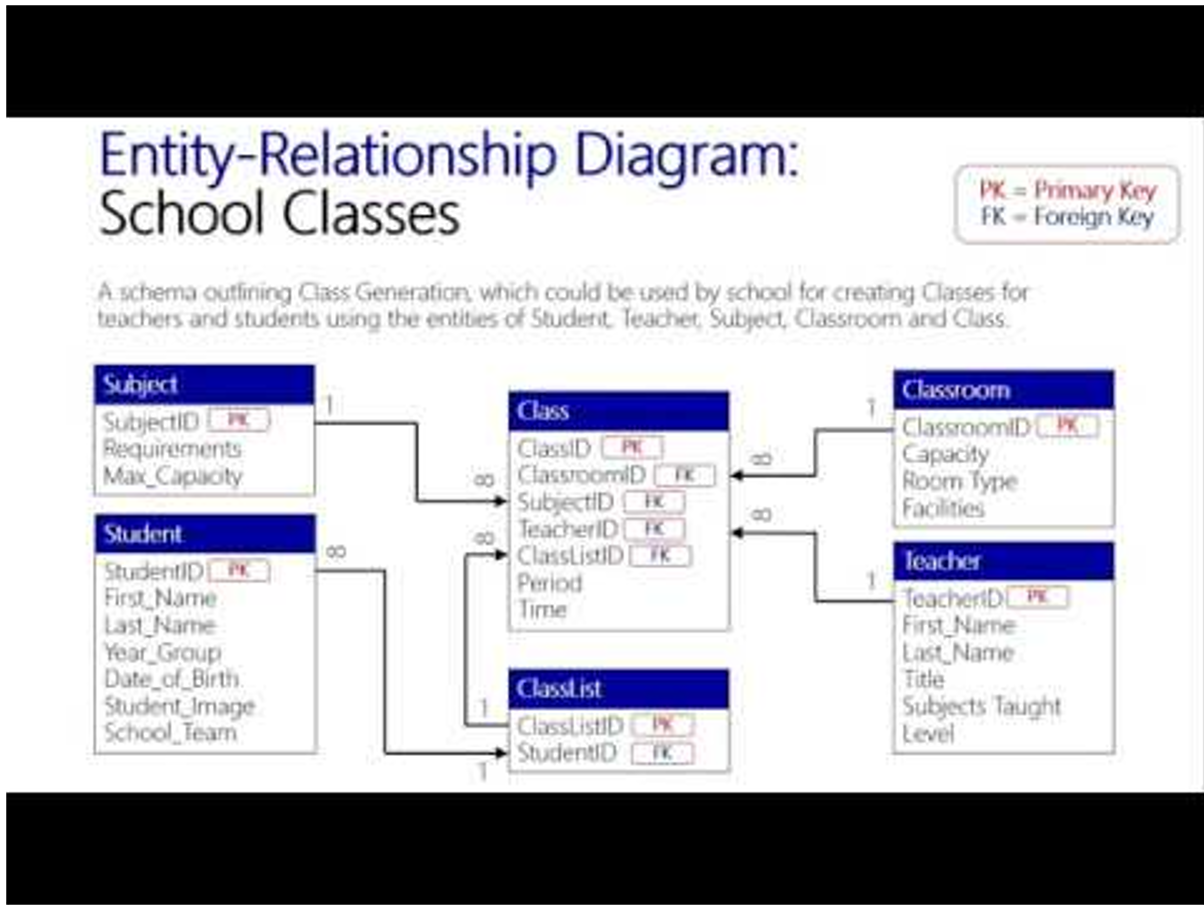
<Representative samples of UX models here and the complete set in the Annex.>

## Data Design

A relational database was used for the project using PostgreSQL DBMS for Heroku. To ensure completeness of the data model, and assess it for quality, an entity relationship diagram was developed.

[Reflect on any difficulties in determining the data needs and developing the data model, especially if your team had to work with another team to establish a consistent database design.]

See an example of the data design below:



[Overall reflection on the process of data design.]

## Version Control

In software engineering it is necessary, especially in big projects, to track and control the source code developed. In our project, we decided to use GitHub, which offers functionalities, such as integrated issue tracking, collaborative code review, team management, and highlighting of syntax. It allowed us to work on separate features of the application, track bugs, and manage coding tasks.

[Overall reflection on this process. Talk about branches you set up, how you merged each developer’s code contributions, avoided merge conflicts, etc.]

## Quality

Quality was assessed using the following quality metrics: Compliance, Efficiency, Correctness, Maintainability, Integrity, and Usability. Although there are many more metrics, these were the ones recommended for the class.

[Reflect on how paying attention to these metrics helped ensure quality of the product and software engineering process.]

### Compliance

[Reflect on how the team ensured compliance of requirements, including how you determined compliance (e.g., % acceptance criteria that passed). A key technique is to make sure each user story has its own acceptance criteria which is reviewed and approved by the customer. In addition, the team makes sure that the user story meets its acceptance criteria during testing / reviews.]

### Efficiency

[Reflect on how the team ensured efficiency through various strategies such as dividing work in manageable chunks among the team members in a certain way, including how you determined efficiency (e.g., Variance between expected schedule, as committed to, vs. actual schedule).]

### Correctness

[Reflect on how the team met this objective, including how you determined correctness (e.g., Defects per sprint that are high severity (no workarounds) where defects are errors discovered by the customer). Included below are some methods used by the team to ensure correctness.]

#### Test-Driven Development (TDD):

This technique involved writing unit/integration tests before coding, and involves the following:

* The developer writes an (initially failing) automated test case that defines a desired outcome;
* He/she produces the minimum amount of code to pass the test;
* Refactors the new code to according to best practices, or to make it more readable.

What we learned:

* These tests are quite tedious to write versus just going straight to coding, however we found the benefits included [fill in].
* [add more reflection here]

#### Reviews:

Code reviews (and other technical reviews such as pair programming) were a big part of ensuring quality and involved [describe your process e.g., GitHub pull requests].

What we learned:

* Code reviews allowed us to discover defects related to [fill in] and verify that we followed the different standards set forth in the project, including [fill in with the list of standards e.g., coding standards, definition of done, etc.].
* [add more reflection here]

#### Definition of Done

The "Definition of Done" is a guide to determining completion of a user story and its tasks.

[Reflect on the importance of using this artifact in ensuring quality.]

Below is the project’s “Definition of Done.”

[Note: below is one that is recommended for each project, which the project team can add to, but not delete from.]

| Sprint Definition of Done Criteria | Objective | Verified (Y/N) | Notes: |
| --- | --- | --- | --- |
| Data Design complete | Maintainability |  |  |
| All changes merged to Main | Efficiency |  |  |
| Any configuration or build changes documented | Efficiency |  |  |
| Sprint Review conducted | Compliance |  |  |
| Client Feedback given during Sprint Reviews. During the last sprint, feedback can be gathered from the Acceptance Test Results and/or customer feedback survey | Compliance |  |  |
| User stories delivered in sprints represented value to the customer (i.e., most valuable delivered in earlier sprints) | Compliance |  |  |
| Working product delivered (not just a prototype) | Compliance |  |  |
| Sprint deliverable reviewed by the product owner | Compliance |  |  |
| Sprint Scrum meetings conducted on a daily basis | Efficiency |  |  |
| Sprint documentation generated (meeting the criteria in the project rubric), including key scrum artifacts (e.g., sprint backlog, etc.) | Maintainability |  |  |
| Peer feedback submitted | Efficiency |  |  |
| Sprint Retrospective Conducted | Maintainability |  |  |
| Plans (e.g., stakeholder management, risk plan, etc.) were discussed at the sprint retrospective and improvement identified (when applicable) | Efficiency |  |  |
| Key performance indicators (especially those related to each objective whether it be group or individual performance) were measured and discussed at the sprint retrospective | Efficiency |  |  |
| Assessed risks to make sure that high impact and high probability risks are mitigated, monitored, and managed. | Compliance |  |  |
| Second to the last sprint: Deployment, support, and maintenance plans submitted and approved | Maintainability |  |  |
| Final Sprint only: All pending issues / user stories resolved | Compliance |  |  |
| Final Sprint only: Project turnover items prepared | Compliance |  |  |

| User Story Definition of Done Criteria | Objective | Verified (Y/N) | Notes: |
| --- | --- | --- | --- |
| Code review performed | Correctness |  |  |
| 100% code coverage using automated test tools | Correctness |  |  |
| 100% test coverage (i.e., tested against all test cases) | Correctness |  |  |
| Meets conditions of satisfaction for the user story (i.e., acceptance criteria), as communicated by the customer during sprint reviews (did we build the "right" product?). During the last sprint, satisfaction can be measured from the user acceptance test form and/or the customer feedback survey | Compliance |  |  |
| App reviewed by the product owner on a continuous basis | Compliance |  |  |
| End-user documentation is ready (should be delivered incrementally during each sprint, with the complete version delivered during the last sprint) | Compliance |  |  |
| Code refactored (as agreed upon by the team) | Maintainability |  |  |
| No fatal ‘code smells’ in code analysis output (e.g., Rubocop) | Maintainability |  |  |
| Meets Coding Standard | Maintainability |  |  |
| All Unit Tests passed | Correctness |  |  |
| All Integration tests passed (including system test when all components integrated) | Correctness |  |  |
| Tests on all supported devices & browsers passed | Correctness |  |  |
| All Integrity testing passed | Integrity |  |  |
| Usability tests passed (should be based on acceptance criteria of user) | Usability |  |  |
| All Performance testing passed to ensure minimum response time met | Correctness |  |  |
| All Regression Tests passed | Correctness |  |  |
| All Validation Tests passed (Did we build the "right" product?) using real data for testing; especially critical/mandatory user stories | Compliance |  |  |
| Any configuration or build changes documented | Maintainability |  |  |
| Deployed live to customer | Efficiency |  |  |

#### Test Coverage

Evaluating test coverage is one of the methods that indicate how well the code was tested.

To ensure that the most common scenarios were tested, both sunny and rainy day, we documented our test cases. Representative test cases are shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Example below:



Guidelines for identifying edge cases:

|  |  |
| --- | --- |
| **Types of Valid Input** | **Edge cases that are candidate inputs to be tested** |
| Range (e.g., 0 to 100). Example a range of valid area codes | •1 valid (e.g., 0 to 100) |
|  | •2 invalid equivalence classes (e.g., < 0 or > 100 ; null) |
| Specific value (e.g., Y,N) | •1 valid (e.g., Y, N) |
|  | •2 invalid equivalence classes (e.g., not Y, not N, null) |
| Member of a set (e.g., set of majors in the college) | •1 valid (e.g,. CPSC) |
|  | •1 invalid equivalence class (e.g., BIMS) |
| Boolean | •1 valid (e.g., true) |
|  | •1 invalid equivalence class (e.g., false) |

[Talk about the data you used in the different test phases (e.g., unit, integration, validation, etc.)]

[Overall reflection on this process.

##### Code Coverage - Output of simplecov

A tool we used to determine code coverage was Simplecov which [describe the purpose of the tool].

See below Simplecov output for our application.

[Insert index.html of SimpleCov.]

[Overall reflection on this process.]

#### Test Results

Prior to releasing to the customer, we tested the app thoroughly, to ensure that bugs do not escape to the customer.

[Describe your test results, including tests that passed and failed. Include a representative sample of results of testing.]

[Describe any bugs that escaped and were discovered by the customer (i.e., defects) and how these can be avoided.]

[Put Heroku app link used for validation testing, prior to release to the customer. Should be a different app link than the one provided to the customer. Include credentials (username & password) used for testing, if needed. This will be used by the teaching team to test your app.]

### Maintainability

One of the approaches we used to fulfill this requirement of producing readable code (especially for any teams that will be enhancing our code base) is the use of a code style checker (e.g., Rubocop) to detect code smells such as excessive nesting of conditional and looping constructs, methods with too many parameters, layout, etc.

[Reflect on how the team ensured maintainability, including how you determined maintainability (e.g., Mean time to deploy a bug fix).]

#### Coding Standard

We also reviewed our code against guidelines of clean code as specified in our coding standard, which is included below. [Your project team can add more, however not recommended to delete any.]

|  |
| --- |
| **Ruby on Rails coding standards** |
| Go easy on comments. If the code is obvious, don’t comment. Remove old, commented code |
| Use two-space indentation |
| Use each instead of for. Use unless instead of !if. However, if you need to involve an else to your conditional, never use unless-else. Use until instead of while ! (negated condition). |
| Use meaningful variable names. |
| Use snake\_case for methods and variables. Use CamelCase for classes and modules. (Keep acronyms like HTTP, RFC, XML uppercase.). Always name your methods based on their behavior, not implementation. |
| The names of predicate methods (methods that return a boolean value) should end in a question mark. Avoid prefixing predicate methods with the auxiliary verbs such as "is," "does," or "can." e.g., person.tall? |
| Instance variables are defined using the single "at" sign (@) followed by a name. It is suggested that a lowercase letter should be used after the @. |
| Global variable starts with a dollar ($) sign followed by other characters. |
| Constants should be all upper case with words separated by underscores ('\_'). |
| Table names have all lowercase letters and underscores between words; all table names must be plural noun, e.g. invoice\_items, orders, etc. |
| The model is named using the class naming convention of unbroken MixedCase and is always the singular of the table name, e.g. if the table name might be orders, the model name would be Order. |
| Controller class names are pluralized, such that OrdersController would be the controller class for the orders table. |
| The primary key of a table is assumed include the word "id" e.g., order\_id |
| The foreign key is named with the singular version of the target table name with id appended to it, e.g, order\_id in the ITEMS table that links to the order\_id in the ORDERS table. |
| Tables used to join two tables in a many to many relationship is named using the table names they link, with the table names in alphabetical order, for example ITEMS\_ORDERS |
| Skinny Controllers, Fat models: best practice is to keep non-response related logic out of the controllers. Examples of code you don’t want in a controller are any business logic or persistence/model changing logic. |
| Views should have very little ruby in them and certainly shouldn’t interact with the data repository (e.g., databases). |
| Ternaries (?:) are good if they fit on one line (remember the short lines rule). |
| Use def with parentheses when there are parameters. Omit the parentheses when the method doesn't accept any parameters |
| Convention over Configuration - try to use the Rails defaults when you can |
| Do not repeat yourself (DRY). Do whatever it takes to make sure that you don’t repeat yourself, avoiding duplication as much as you can. For example, use abstract classes, modules |
| Smart use of Enums |
| Use db:schema:load when creating the application database on a new system. Use db:migrate in all other cases when you need to apply the newly added migrations. |
| Nested Resources/Routes: If you have a resource which belongs to another resource, then it’s a good idea to define the routes of the child resource nested within the routes of parent resource. |

[Overall reflection on the use of a coding standard.]

#### Well Documented Code

[Overall reflection on this process of ensuring well-documented code and the benefit.]

#### Linter Output

See below the output from the linter (e.g., RuboCop).

[Insert]

[Overall reflection on this process of linting and the benefit.

### Integrity

[Reflect on how the team met this objective, including how you determined that you ensured integrity (e.g., # of security vulnerabilities resolved).]

#### Security Risk Analysis Table

The security-risk analysis table is shown below which includes common vulnerabilities in our application. Strategies to address each vulnerability is in the RMMM column.

{Reflect on how you performed security risk analysis.]

|  |  |  |  |
| --- | --- | --- | --- |
| Security Risk | Prob in % | Impact | Risk Mitigation, Monitoring, and Management (RMMM) Plan |
| Example: SQL injection |  |  |  |
| Weak Authentication |  |  |  |
|  |  |  |  |

#### Authentication

[Describe approach and implementation (e.g., Google OAuth)].

#### Integrity Test Coverage

Evaluating integrity test coverage is one of the methods that indicate how secure your system is from attacks.

To ensure that the most common scenarios were tested, both sunny and rainy day, we documented our test cases. Some representative test cases are shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
|  |  |  |  |  |
|  |  |  |  |  |

[Talk about the data you used in the different test phases (e.g., unit, integration, etc.)]

[Overall reflection on this process of ensuring there was sufficient test coverage to avoid any security breaches]

#### Integrity Test Results

We ensured that our RMMM plan worked to address common security vulnerabilities by [fill in with approach such as attempting to enter the system without the proper credentials, etc.]

See below representative output of any integrity related tests (e.g., actual RSpec tests, use of tools such as Brakeman).

[Describe your test results, including tests that passed and failed.]

[Describe any vulnerabilities discovered by the customer (e.g., security breaches) and how these can be avoided in the future.]

### Usability

[Reflect on how the team met this objective, including how you determined that you ensured usability (e.g., amount of time required to be proficient in the use of the system).]

#### Usability Requirements

The team first gathered user requirements related to "usability" and reflected them as both user stories and ‘usability’ acceptance criteria within particular user stories. Some examples below:

[insert representative sample]

#### Usability Test Coverage

Determining sufficient test coverage is one of the methods for ensuring usability of the system.

To ensure that the most common scenarios were tested, both sunny and rainy day, we documented our test cases. Some representative test cases are shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
|  |  |  |  |  |

<Talk about the data you used in the different test phases (e.g., unit, integration, etc.)>

<Overall reflection on this process.>

#### Usability Test Results

We ensured that our system was usable by [fill in with how you tested for usability.]

[Describe your test results, including tests that passed and failed. [Describe any defects discovered by the customer (i.e., defects) and how these can be avoided]

See below representative output of any usability related tests.

## Monitoring and Control

[Explain overall what the team monitored to make sure that they were doing the right actions to produce the right product.]

[Mention any reports /graphs you used from Jira and other sources and include samples here.]

[Reflect on the importance of Monitoring and Control in your project, for example, evaluation of actual performance against expected results (e.g., efficiency, compliance, correctness, etc.). In addition, reflect on the importance of reviewing original plans such as the stakeholder management & communication plan, and risk plan, and modifying when necessary to reflect reality.]

Some sample key performance indicators for each of the different objectives are listed below:

* Efficiency: variance between expected and actual release date, individual performance (e.g., % of issues / member, resolved and unresolved issues at the end of each sprint that may need to be deferred to a future sprint)
* Compliance - # of non-compliance to customer requirements mentioned during customer meetings, # of requirements change requests from the customer which may mean that original requirements were misinterpreted, % acceptance criteria met, % of user stories completed that were requested by the customer
* Correctness - test coverage, code coverage using Simplecov, # of code reviews, # of defects (errors discovered by customer), # of high severity errors discovered during later test phases such as last stages of integration test or system test (when all components integrated), # of errors deferred to a future sprint, etc.
* Integrity - # of security tests performed, # of defects related to security breaches, etc.
* Usability - # of negative feedback related to usability during sprint reviews, % compliance with acceptance criteria related to usability, % user stories done which are related to usability, etc.
* Maintainability - length of time for the team to fix a bug (developed by someone else), effort to release per sprint, statistics from code style checker (e.g., Rubocop), etc.

## Deploying App

[Reflect on the process of deploying the app to the customer]

[Put the Heroku link given to customer and any default credentials.]

## Sprint Review

[Reflect on this process and how it helped the team.]

[Place links to the MS Teams recording here. Do not use Zoom]

[Evidence uploaded in MS Teams that issues are documented and highly likely to be addressed (e.g., screenshot of corresponding issue in Jira, or action plan using action plan template)

\*NOTE\* Please mention where to find this artifact.]

## Sprint Retrospective

[Reflect on this process and how it helped the team.]

[Place links to the MS Teams recording here. Do not use Zoom]

[Evidence uploaded in MS Teams that issues are documented and highly likely to be addressed (e.g., screenshot of corresponding issue in Jira, action plan using action plan template)

\*NOTE\* Please mention where to find this artifact.]