



# National College of Ireland

BSHC

Software Development

2020/2021

Ben Carroll

x17501726

[x17501726@student.ncirl.ie](mailto:x17501726@student.ncirl.ie)

## Retro - Technical Report

## Contents

Table of Figures:	4
Executive Summary	6
1. Introduction	6
1.1. Background	6
1.2. Aims	7
1.3. Technology	7
1.4. Structure	8
2. System	8
2.1. Requirements	8
2.1.1. Functional Requirements	8
2.1.2. Data Requirements	17
2.1.3. User Requirements	17
2.1.4. Usability Requirements	17
2.2. Design & Architecture	18
2.3. Implementation	20
2.3.1. The Backend:	20
2.3.2. The Frontend:	28
2.4. Graphical User Interface (GUI)	34
2.4.1. Home Page:	34
2.4.2. Retrospective Page:	35
2.4.3. Archive Page:	36
2.4.4. Sentiment Analysis Page:	37
2.4.5. Search:	38
2.4.6. Modal Dialogs:	39
2.4.7. Voting System:	41
2.4.8. Retrospective Timestamps:	42
2.5. Testing	42
2.5.1. Backend	42
2.5.2. Frontend	45
2.6. Evaluation	46
3. Conclusions	47
4. Further Development or Research	48
5. References	49
6. Appendices	50
6.1. Project Plan	50

6.2.	Project Proposal .....	52
6.2.1.	Objectives .....	52
6.2.2.	Background .....	53
6.2.3.	Technical Approach .....	55
6.2.4.	Special Resources Required .....	56
6.2.5.	Project Plan .....	56
6.2.6.	Technical Details .....	59
6.2.7.	Evaluation .....	59
6.3.	Reflective Journals .....	60
6.3.1.	October Reflective Journal .....	60
6.3.2.	November Reflective Journal .....	60
6.3.3.	December Reflective Journal .....	61
6.3.4.	January Reflective Journal .....	61
6.3.5.	February Reflective Journal .....	62
6.3.6.	March Reflective Journal .....	62
6.3.7.	April Reflective Journal .....	62
6.4.	Informed Consent Form : .....	63
6.4.1.	Participant One: .....	63
6.5.	User Testing Results: .....	64
6.5.1.	Five Second Test: .....	64
6.5.2.	Trunk Test: .....	64
6.5.3.	Think Aloud Test: .....	64
6.5.4.	Tree Test: .....	65
6.5.5.	Click Test: .....	65
6.6.	NCI Ethics Application Form .....	66
6.7.	System Usability Scale Exit Survey: .....	72
6.8.	User Testing: User Interface .....	72

## Table of Figures:

Figure 1: Retro use case diagram .....	9
Figure 2: Angular frontend file structure .....	18
Figure 3: Java Spring backend file structure.....	19
Figure 4: Retro architecture diagram .....	20
Figure 5: LoadDatabase.java: 'Local' profile java file used for testing purposes .....	21
Figure 6: Backend: Retrospectives controller .....	22
Figure 7: Item class java file .....	23
Figure 8: Item type enumerator java file.....	23
Figure 9: Retrospectives JPA repository interface java file.....	24
Figure 10: RetrosNotFoundExpection - custom error handling method .....	24
Figure 11: Retrospectives service class java file.....	25
Figure 12: Stop Words.....	25
Figure 13: Afinn Dictionary - words and associated sentiment scores .....	26
Figure 14: Sentiment Analysis - Reading in dictionaries and retrieving retrospective item descriptions.....	26
Figure 15: Sentiment Analysis - words being assigned a score and sentiment analysis conducted.....	27
Figure 16: Exporting retrospective items and action items to a CSV file.....	27
Figure 17: Retrospectives component HTML file .....	28
Figure 18: Item form methods.....	29
Figure 19: Retrospectives Typescript file.....	30
Figure 20: Retrospectives services Typescript file.....	31
Figure 21: proxy.conf.json - allows backend and frontend to communicate.....	31
Figure 22: Archived retrospectives pipe method.....	32
Figure 23: Archived retrospectives pipe HTML filter .....	32
Figure 24: Angular pipe to filter items by type .....	32
Figure 25: Item votes Angular filter .....	33
Figure 26: Use of Angular pipes for item types and item votes in HTML.....	33
Figure 27: Application routing .....	33
Figure 28: GUI - Retro Home page .....	34
Figure 29: GUI - Individual retrospective page with items .....	35
Figure 30: GUI - Individual retrospective page with action items .....	35
Figure 31: GUI - Retrospective page - reviewed item .....	36
Figure 32: GUI - Archived retrospectives page.....	36
Figure 33: GUI - Sentiment Analysis Page – displaying a positive Sentiment Analysis.....	37
Figure 34: GUI - Sentiment Analysis Page – displaying a negative Sentiment Analysis.....	38
Figure 35: GUI - Searching for a retrospective .....	39
Figure 36: GUI - Searching for a Retrospective.....	39
Figure 37: GUI - Modal dialog to delete a retrospective.....	40
Figure 38: GUI - Modal dialogue to delete a retrospective item .....	40
Figure 39: GUI - Modal dialog to edit a retrospective item .....	41
Figure 40: GUI - Voting system .....	41
Figure 41: GUI - Voting System .....	42
Figure 42: Service tests - Unit tests for retrieving retrospectives.....	43
Figure 43: Code coverage present in each class and their respective methods.....	44
Figure 44: All tests passing for the application, service, and controller tests.....	44
Figure 45: YAML configuration file for 'local' profile .....	45

Figure 46: Gantt Chart timeline view .....	50
Figure 47: Gantt Chart items for backend .....	50
Figure 48: Gantt Chart items for frontend 1/2 .....	52
Figure 49: Gantt Chart items for frontend 2/2 .....	52
Figure 50: Project Plan - Gantt Chart timeline view .....	56
Figure 51: Project Plan - Gantt Chart items 1/3 .....	57
Figure 52: Project Plan - Gantt Chart items 2/3 .....	58
Figure 53: Project Plan - Gantt Chart items 3/3 .....	59
Figure 54: System Usability Scale Exit Survey .....	72
Figure 55: Usability testing Retro homepage .....	73
Figure 56: Usability testing individual retrospective page .....	73

## Executive Summary

A large number of modern Software Development teams have team members in different locations, and this factor has of course increased with many people working remotely due to the pandemic ([Taplin, 2021](#)). As most Software Development teams follow an Agile sprint methodology, remote working affects their ability to carry out retrospectives, which are typically done in person. An Agile sprint is a short, time-bound period where teams complete a defined amount of work and a retrospective is carried out after a sprint in order to discuss how it went with regards to individuals, interactions, processes, tools ([Scrum, 2021](#)). Similarly, it has become increasingly difficult for Software Development team managers to effectively manage and gauge team morale when working remotely.

This report details the background, requirements gathering, design and implementation of Retro, an implementation of a solution to this problem. Retro is a web application that allows Software Development teams to be able to facilitate their agile sprint retrospectives remotely and enables managers of Software Development teams to effectively manage remotely through a Sentiment Analysis (mining of text that finds and extracts information that helps a person understand the sentiment behind the data [[Gupta, 2018](#)]) that is performed on a retrospective, allowing managers to gauge team morale, with the data to back the analysis up.

Retro's implemented solution offers a lightweight, speedy web application that can be used by any Software Development team or enterprise to perform retrospectives. Its minimalist and user-friendly user interface places focus on the retrospectives and allows teams to address issues that have arisen from their retrospectives in real-time without compromising on functionality is also an advantage. Similarly, it allows Software Development team managers to effectively gauge team morale through its Sentiment Analysis functionality.

## 1. Introduction

### 1.1. Background

I undertook this project to provide a means for Software Development teams to be able to facilitate their agile sprint retrospectives remotely. The idea for this project occurred to me during my placement in third year. The team I was working in followed the Agile methodology but when the team were doing retrospectives at the end of a sprint, there was no dedicated application to facilitate them that provided the level of communication needed, especially when some team members were in different offices around the world.

I concluded that an application is needed to provide a more organised and effective way of carrying out a retrospective that would allow it to be both meaningful and useful to all members of the team and to management, especially at the moment with the majority of development teams working from home due to the pandemic.

Retrospectives are traditionally done in person where even if something is unclear during a retrospective, it can be clarified by popping down to someone's desk and asking again. However, this type of communication is no longer a possibility for a large number of development teams. I want to provide an application that could keep team communication clear in retrospectives, aid remote management of teams and give back some semblance of normalcy to teams that are working remotely.

## 1.2. Aims

Retro aims to facilitate sprint retrospectives remotely for Software Development teams within a responsive web application. Retro will allow development teams that are using an agile sprint methodology to run their retrospectives from anywhere.

Development teams will be able to add items to the retrospective that will tackle things that went well in the sprint, things that did not go well and any questions they have as a result of the sprint. They will also be able to add action items, issues that need to be solved in the following sprint or added to their backlog.

Retro seeks to allow teams to carry out their retrospectives in a clear, open manner, with excellent communication levels in development teams, as though they were in person, through an intuitive and user-friendly application that is designed to fit the needs of any team.

Retro aims to provide a sentiment analysis tool for team managers that analyses words commonly used in the retrospectives of their team in order to assess the morale levels within their team. This benefits managers if they are unable to gauge things like this by face-to-face interactions and meetings. This would aid managers in addressing productivity issues or maintaining morale with the benefit of data to back it up.

Retro also aims provide an application that can scale to fit the needs of any team or company without compromising on efficiency. This will be achieved by designing Retro using S.O.L.I.D. (Single responsibility, Open/closed, Liskov segregation, Interface Segregation, Dependency inversion) design principles throughout the project. Similarly, Retro will provide exceptional security standards within the application, giving users the peace of mind that their retrospectives are being properly safeguarded in the application.

## 1.3. Technology

Angular was used to develop the frontend of the application, both on mobile and desktop platforms. It was chosen as there is ample documentation and resources on it, and due to its component design pattern, it lends itself to scalability and efficiency needed for the project. Similarly, the Bootstrap framework was also implemented within the Angular project in order to improve the look and feel. This provides the user-friendly interface that achieves the clear communication Retro aims for.

The backend was implemented using Java coupled with the Spring framework. The Spring framework provides built-in libraries that was used to tackle the scalability and security of the application as well as lending itself to designing and building the API that is needed for Retro's core functionality. This provides both the performance and reliability needed for Retro to succeed.

The AFINN Dictionary ([\(AFINN, 2011\)](#), [\(Nielsen, 2011\)](#)) was used to carry out the Sentiment Analysis on retrospectives. It does so by assigning a score to each word used in a retrospective depending on if the level of positivity or negativity typically associated with the word. For example, the word 'masterpiece' has a score of 4, and the word 'accident' has a score of -2. There is also a list of 'stop words' that each word in a retrospective is compared to, and if the word is present amongst the list of stop words, it is not assigned a score. The stop words typically include pronouns such as 'I', 'we', or 'you'.

The score of the words is then added together and an overall score is produced. The team's manager can see their team's morale level in a visual form at based off of the overall score data, as well as seeing a recommendation message. The recommendation message is feedback from Retro based off the Sentiment Analysis data and depends on if the overall score is a positive or negative integer.

In order to design this project with efficiency and scalability in mind, the Java S.O.L.I.D. (Single responsibility, Open/closed, Liskov segregation, Interface Segregation, Dependency inversion) were employed. These principles are not a technology per se, but they were followed closely when implementing the above technology in the project to ensure the application can perform as intended and weather periods of high traffic effectively. This aids the scalability and security aspects of the project.

#### 1.4. Structure

This document delves into the functional, data, user and usability requirements of the system needed for Retro, the design and architecture of the application as well as how the frontend and backend were implemented, as visualised by screenshots of the user interface. Details of how unit tests were incorporated throughout the backend and usability tests throughout the frontend are also included. Similarly, the project's conclusion and plans for further development are documented.

The project plan, proposal, my reflective journals, and the results of the usability testing are also attached to the document's appendices.

## 2. System

### 2.1. Requirements

#### 2.1.1. Functional Requirements

##### 2.1.1.1. Use Case Diagram

The various use cases of Retro are detailed in *Figure 1*.



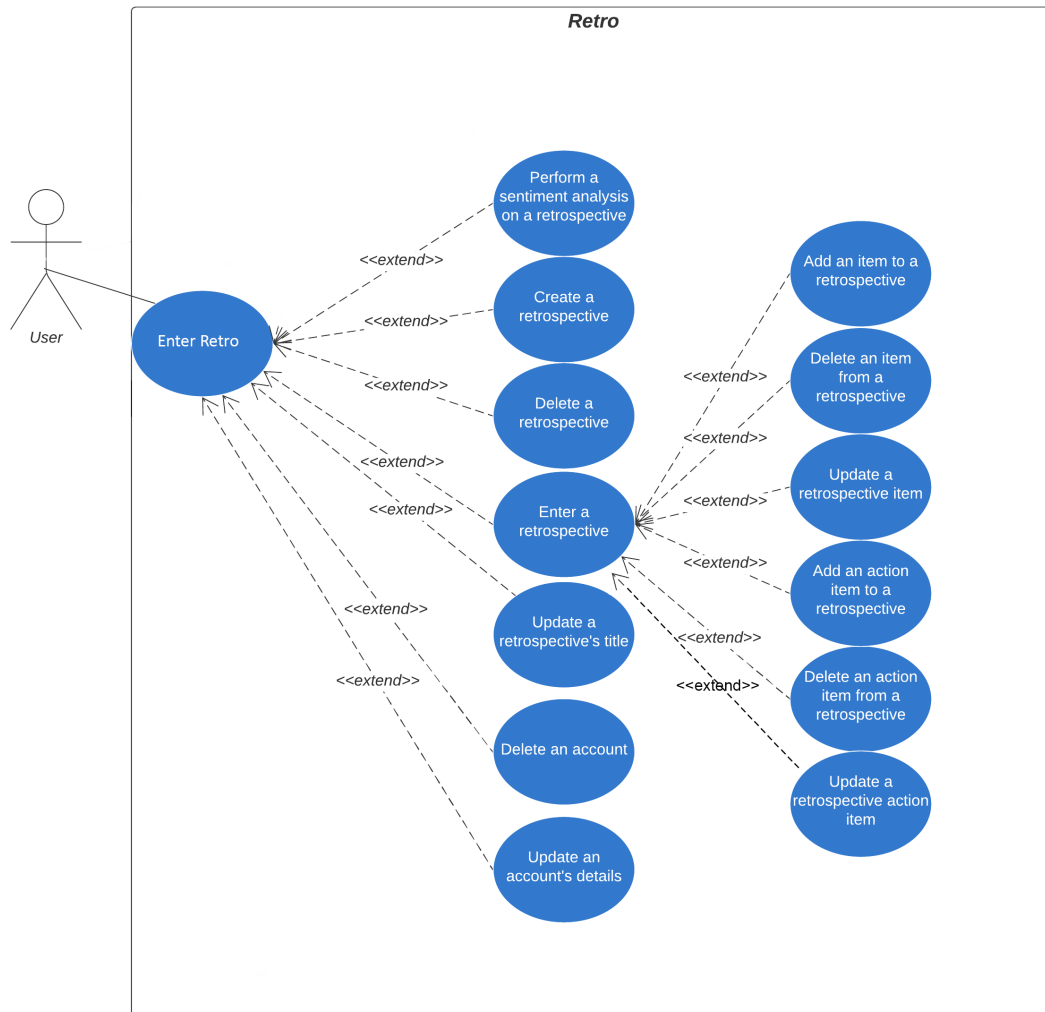


Figure 1: Retro use case diagram

2.1.1.2. Requirement 1: The User is able to create a retrospective.

2.1.1.2.1. Description and Priority

This use case is a priority for Retro as once the User enters the Retro web application, they must be able to create a retrospective and it saved to the database by the system .

2.1.1.2.2. Use Case

**Scope**

The scope of this use case is to let t he User create a retrospective on Retro.

**Description**

This use case describes the creation of a retrospective in the database by a User once they have entered the Retro web application.

**Flow Description**

**Precondition**

The system is in initialisation mode .

**Activation**

This use case starts when a User logs on to the Retro web application.

#### **Main flow**

1. The User enters a retrospective title in the text input area.
2. The User submits the retrospective title form to the system.
3. The system creates the retrospective with the specified title and stores it in the database.

#### **Termination**

The system becomes idle.

#### **Post condition**

The system goes into a wait state.

2.1.1.3. Requirement 2: User is able to retrieve and view a retrospective.

2.1.1.3.1. Description and Priority

This use case is a priority for Retro as once the User enters the Retro web application, they must be able to retrieve a retrospective from the database and view it .

2.1.1.3.2. Use Case

#### **Scope**

The scope of this use case is to let the User enter Retro and to retrieve and view a retrospective.

#### **Description**

This use case describes the retrieval of a retrospective from the database by a User once they have entered the Retro web application.

#### **Flow Description**

##### **Precondition**

The system is in initialisation mode and the retrospective has already been created .

##### **Activation**

This use case starts when a User prompts the system to enter a retrospective .

##### **Main flow**

1. The User clicks on a retrospective and the system identifies that the user wants to enter a retrospective .
2. The User enters the desired retrospective.

##### **Termination**

The system becomes idle.

##### **Post condition**

The system goes into a wait state.

#### 2.1.1.4. Requirement 3: User is able to delete a retrospective.

##### 2.1.1.4.1. Description and Priority

This use case is a priority for Retro as once the User creates a retrospective, they must be able to delete it.

##### 2.1.1.4.2. Use Case

#### **Scope**

The scope of this use case is to let the User delete a retrospective.

#### **Description**

This use case describes the deletion of a retrospective and have it removed from the database.

#### **Flow Description**

##### **Precondition**

The system is in initialisation mode and the retrospective has already been created .

##### **Activation**

This use case starts when a User enters Retro.

##### **Main flow**

1. The User prompts the system to delete a retrospective.
2. The User is prompted to confirm that they want to delete the retrospective.
3. The User clicks 'yes' to confirm deletion. (See A1)
4. The system identifies that the user wants to delete a retrospective and removes it from the database .

##### **Alternate flow**

A1: The User cancels deleting the retrospective.

1. The retrospective is not deleted from the database by the system.

#### **Termination**

The system becomes idle.

#### **Post condition**

The system goes into a wait state.

#### 2.1.1.5. Requirement 4: User is able to perform a Sentiment Analysis on a retrospective.

##### 2.1.1.5.1. Description and Priority

This use case is a top priority for Retro as once the User has carried out a retrospective with their team by adding items to it , they must be able to perform a Sentiment Analysis on it .

##### 2.1.1.5.2. Use Case

#### **Scope**

The scope of this use case is to let the User perform a Sentiment Analysis a retrospective.

### Description

This use case describes performing a Sentiment Analysis on a retrospective.

### Flow Description

#### Precondition

The system is in initialisation mode and the retrospective has already been created and carried out with the User's team by adding items to it.

#### Activation

This use case starts after a User carries out a retrospective with their team and prompts the system for the sentiment analysis for that retrospective.

#### Main flow

1. The User ensures that the retrospective has been completed and that their team members are no longer adding items to it.
2. The User prompts the system to produce a sentiment analysis on a particular retrospective.
3. The system produces the sentiment analysis on the specified retrospective and returns it to the user.

### Termination

The system becomes idle.

### Post condition

The system goes into a wait state.

#### 2.1.1.6. Requirement 5: User is able to add an item to a retrospective.

##### 2.1.1.6.1. Description and Priority

This use case is a priority for Retro as once the User enters a retrospective, they must be able to add an item to the retrospective.

##### 2.1.1.6.2. Use Case

### Scope

The scope of this use case is to let the User add an item to a retrospective.

### Description

This use case describes the addition of an item to a retrospective and have it saved to the database.

### Flow Description

#### Precondition

The system is in initialisation mode and the retrospective has already been created.

#### Activation

This use case starts when a User enters a retrospective.

### **Main flow**

1. The system identifies that the user wants to add an item to a retrospective.
2. The User adds an item under the 'things that went well' column to a retrospective. (See A1, A2)
3. The item is saved to that retrospective by the system on the database.

### **Alternate flow**

A1: The User adds an item under the 'things that did not go so well' column to a retrospective.

1. The item is saved to that retrospective by the system on the database.

A2: The User adds an item under the 'questions' column to a retrospective.

1. The item is saved to that retrospective by the system on the database.

### **Termination**

The system becomes idle.

### **Post condition**

The system goes into a wait state.

#### **2.1.1.7. Requirement 6: User is able to delete an item from a retrospective.**

##### **2.1.1.7.1. Description and Priority**

This use case is a priority for Retro as once the User enters a retrospective, they must be able to delete an item from the retrospective.

##### **2.1.1.7.2. Use Case**

### **Scope**

The scope of this use case is to let the User delete an item from a retrospective.

### **Description**

This use case describes the deletion of an item from a retrospective and have it removed from the database.

### **Flow Description**

#### **Precondition**

The system is in initialisation mode and the retrospective and item have already been created.

#### **Activation**

This use case starts when a User enters a retrospective.

#### **Main flow**

1. The system identifies that the user wants to delete an item from a retrospective.
2. The User is prompted to confirm that they want to delete the item.
3. The User clicks 'yes' to confirm deletion. (See A1)

4. The item is deleted from that retrospective by the system on the database.

#### **Alternate flow**

A1: The User cancels deleting the item from the retrospective.

1. The item is not deleted from the retrospective by the system on the database.

#### **Termination**

The system becomes idle.

#### **Post condition**

The system goes into a wait state.

#### **2.1.1.8. Requirement 7: User is able to update an item on a retrospective.**

##### **2.1.1.8.1. Description and Priority**

This use case is a priority for Retro as once the User enters a retrospective, they must be able to update an item on the retrospective.

##### **2.1.1.8.2. Use Case**

#### **Scope**

The scope of this use case is to let the User update an item on a retrospective.

#### **Description**

This use case describes updating an item on a retrospective and have it updated on the database.

#### **Flow Description**

##### **Precondition**

The system is in initialisation mode and the retrospective and item have already been created.

##### **Activation**

This use case starts when a User enters a retrospective.

##### **Main flow**

1. The system identifies that the user wants to update an item on a retrospective.
2. The User is prompted with a modal dialog to update the text description of the item that they want to update.
3. The User enters the updated description and clicks 'yes' to confirm updating. (See A1)
4. The item is updated by the system on the database.

##### **Alternate flow**

A1: The User cancels updating the item on the retrospective.

1. The item is not updated on the retrospective by the system.

### **Termination**

The system becomes idle.

### **Post condition**

The system goes into a wait state.

#### **2.1.1.9. Requirement 8: User is able to add an action item to a retrospective.**

##### **2.1.1.9.1. Description and Priority**

This use case is a priority for Retro as once the User enters a retrospective , they must be able to add an action item to the retrospective .

##### **2.1.1.9.2. Use Case**

### **Scope**

The scope of this use case is to let the User add an action item to a retrospective.

### **Description**

This use case describes the addition of an action item to a retrospective and have it saved to the database .

### **Flow Description**

#### **Precondition**

The system is in initialisation mode and the retrospective has already been created .

#### **Activation**

This use case starts when a User enters a retrospective .

#### **Main flow**

1. The system identifies that the user wants to add an action item to a retrospective.
2. The User adds an action item to a retrospective.
3. The action item is saved to that retrospective by the system on the database.

### **Termination**

The system becomes idle.

### **Post condition**

The system goes into a wait state.

#### **2.1.1.10. Requirement 9: User is able to delete an action item from a retrospective.**

##### **2.1.1.10.1. Description and Priority**

This use case is a priority for Retro as once the User enters a retrospective , they must be able to delete an action item from the retrospective .

##### **2.1.1.10.2. Use Case**

### **Scope**

The scope of this use case is to let the User delete an action item from a retrospective.

## Description

This use case describes the deletion of an action item from a retrospective and have it removed from the database .

## Flow Description

### Precondition

The system is in initialisation mode and the retrospective and action item have already been created .

### Activation

This use case starts when a User enters a retrospective .

### Main flow

1. The system identifies that the user wants to delete an action item from a retrospective.
2. The User is prompted to confirm that they want to delete the action item .
3. The User clicks 'yes' to confirm deletion. (See A1)
4. The action item is deleted from that retr ospective by the system on the database.

### Alternate flow

A1: The User cancels deleting the action item from the retrospective.

1. The action item is not deleted from the retrospective by the system on the database.

## Termination

The system becomes idle.

## Post condition

The system goes into a wait state.

2.1.1.11. Requirement 10: User is able to update an action item on a retrospective.

2.1.1.11.1. Description and Priority

This use case is a priority for Retro as once the User enters a retrospective , they must be able to update an action item on the retrospective .

2.1.1.11.2. Use Case

## Scope

The scope of this use case is to let the User update an action item on a retrospective.

## Descript ion

This use case describes updating an action item on a retrospective and have it updated on the database .

## Flow Description

### Precondition



The system is in initialisation mode and the retrospective and action item have already been created.

### Activation

This use case starts when a User enters a retrospective .

### Main flow

1. The system identifies that the user wants to update an action item on a retrospective.
2. The User is prompted with a modal dialog to update the text description of the action item that they want to update.
3. The User enters the updated description and clicks 'yes' to confirm updating. (See A1)
4. The action item is updated by the system on the data base.

### Alternate flow

A1: The User cancels updating the action item on the retrospective.

1. The action item is not updated on the retrospective by the system.

### Termination

The system becomes idle.

### Post condition

The system goes into a wait state.

#### 2.1.2. Data Requirements

Retro store s any information relating to a retrospective that a user creates, such as items (things that went well, things that did not go so well, or questions ) and action items, into the H2 in -memory database. Users will be able to update, read, create, or delete this information once created.

#### 2.1.3. User Requirements

The User Requirements in Retro were largely taken care of by the browser they are accessing the web application from. Retro does not need a lot of user specific requirements as sufficient accessibility features are found within modern browsers and enable users to use Retro as they need to.

#### 2.1.4. Usability Requirements

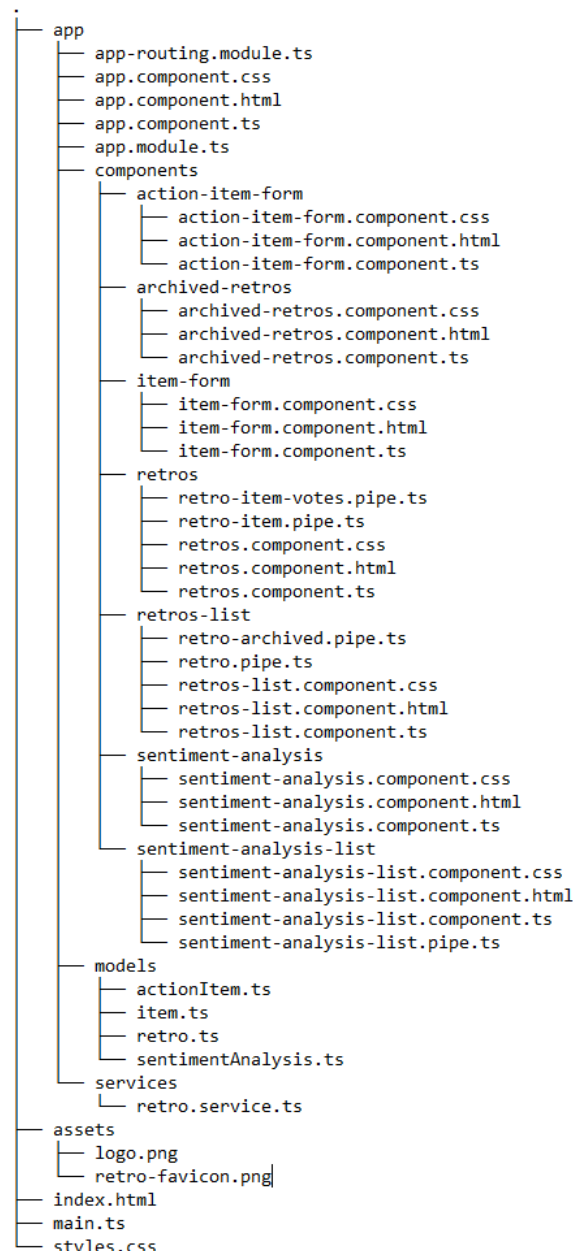
Usability requirements play an important role in Retro as one of the main aims of the project was to provide users with an application that is intuitive and user -friendly . As such Retro provide s a simple -to-use application that once a user has been shown how to use it once, they will not have trouble picking it ba ck up again.

Retro allow s users to complete retrospectives with their teams in a clear manner that allows them to regain an in -person level of communication and come away from a retrospective feeling like it was productive . This was achieved by a series o f [usability testing](#) and added functionality that can be seen in the Angular frontend.

## 2.2. Design & Architecture

Retro is designed on both the front and back ends with S.O.L.I.D. design principles in mind. As such, at every appropriate junction in the project, packages and directories are created to house files that have similarly follow the S.O.L.I.D. design pattern.

As shown in *Figure 2*, the frontend is separated out into directories ( components [each containing the projects HTML, CSS,and Typescript ], models, and services ) that each have their own function .



*Figure 2: Angular frontend file structure*

As shown in *Figure 3*, the backend is separated out into packages ( configs, controllers, models, repositories, and services ) that similarly each have their own function and responsibility.

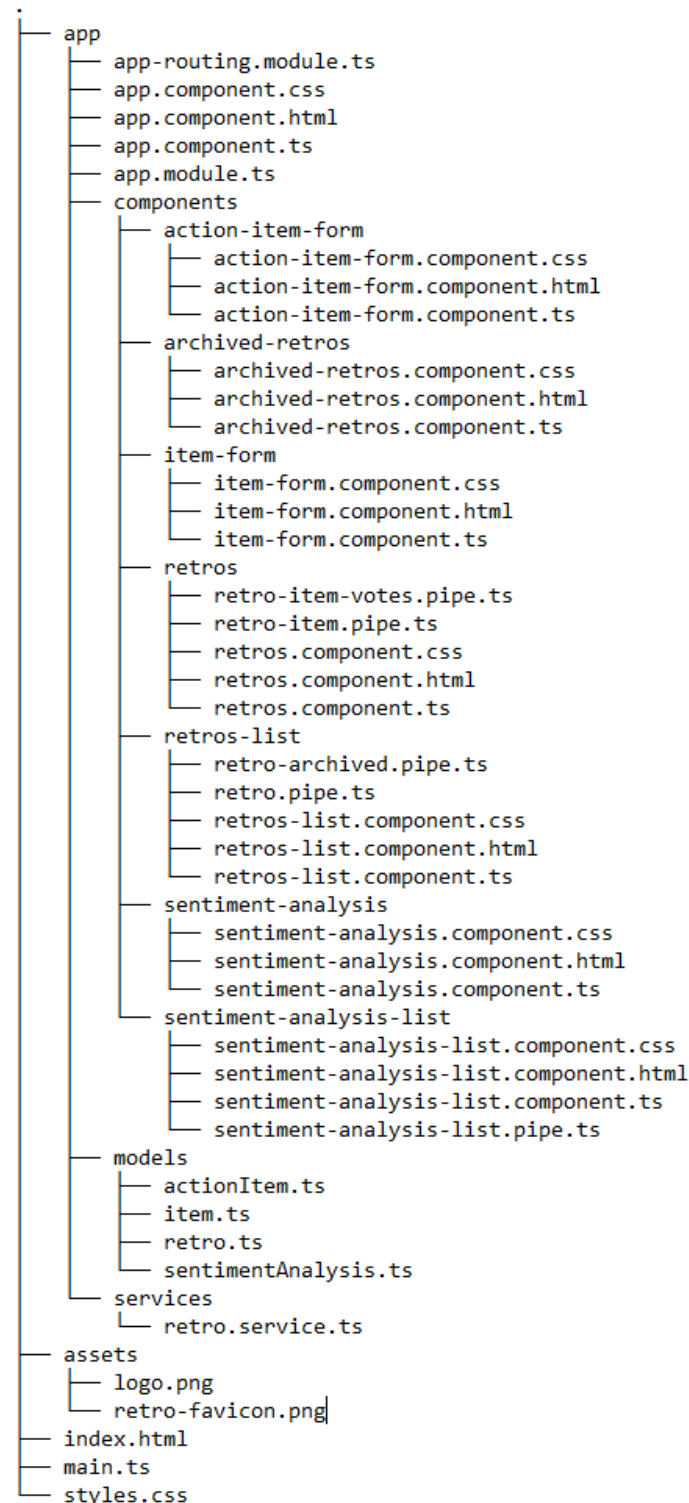


Figure 3: Java Spring backend file structure

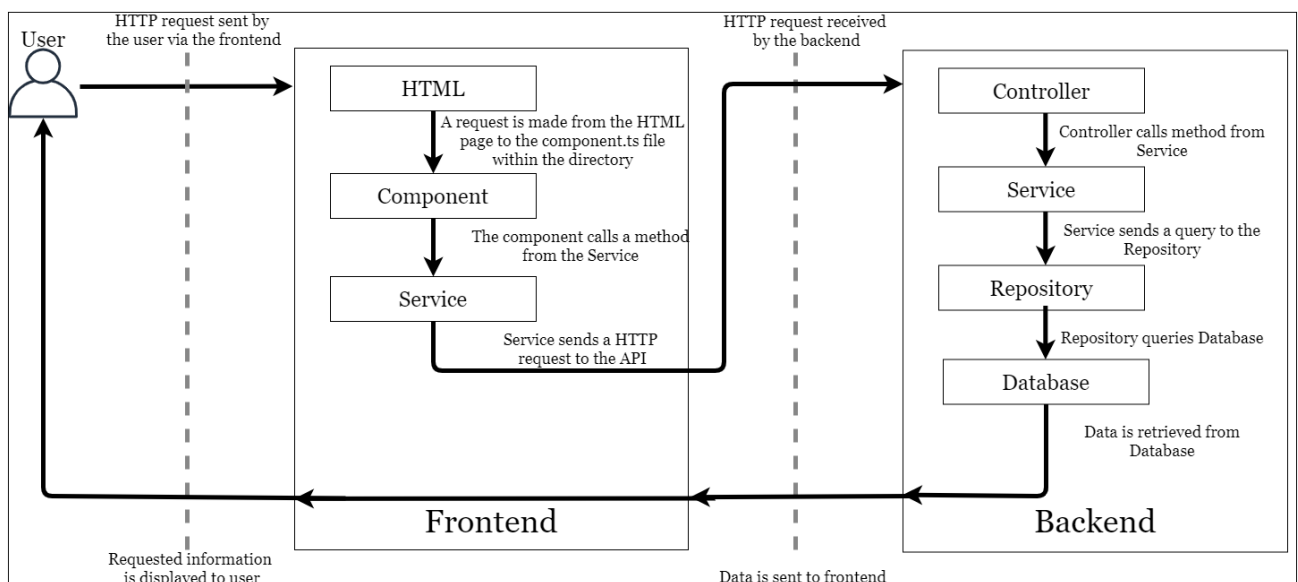
As seen in *Figure 4*, Retro is designed to allowing the user making a request through a particular endpoint to create, retrieve, update, or delete information pertaining to a retrospective. This request is in the form of a HTTP POST, GET, PUT, or DELETE request, respectively. HTTP methods make up a large amount of the backend for Retro. The request is sent via the frontend and received by the backend API. A method within the API's

controller is then called, depending on the endpoint the request was made from and the HTTP verb used, which then calls the associated method in the 'retros' service file, which in turn queries the appropriate JPA repository which retrieves the data from the database. This information is then returned to the frontend and displayed in an appropriate fashion to the user.

The creation and use of a RESTful API for the backend of Retro supports its aim of being a fast and scalable web application as instead of the costly (in terms of memory and time) storage of data within Retro's in-application memory, the API allows for abstraction and then the retrieval and serving of particular data through HTTP protocols on the frontend.

As seen in *Figure 4*, on the frontend, Retro's Angular service class method goes to specified endpoints for the backend API and creates, reads, updates, or deletes information in the database, as the user dictates from the frontend views.

Similarly, S.O.L.I.D. design principles are present throughout the system architecture as each component of the system only has a single responsibility, as detailed in *Figure 4*.



*Figure 4: Retro architecture diagram*

## 2.3. Implementation

### 2.3.1. The Backend:

The backend is separated out into packages each containing different classes that pertain to different operations for the API. The 'configs' package contains the sample database information of the project, as shown in *Figure 5*, and is what I used when testing and designing project to ensure Retro was functioning as intended by using the Spring 'local' profile created in the *LoadDatabase.java* file. Data relating to a retrospective is loaded into the H2 in-memory database as well as items and action items associated with the retrospective.

```

20 @Configuration
21 @Slf4j
22 //setting profile to be local so when the API is ran with this
23 //profile selected, it will load this data
24 @Profile("local")
25 public class LoadDatabase {
26
27     @Bean
28     public CommandLineRunner initDatabase(RetrosRepository retrosRepository, ItemsRepository itemsRepository, ActionItemsRepository actionItemsRepository) {
29         return args -> {
30             log.info("Beginning preload of data");
31             |
32             //creating data for first sample retrospective
33             Retro retro1 = retrosRepository.save(new Retro(name: "Software Development Team Retrospective", archiveRetroFlag: false));
34
35             //adding items
36             List<Item> itemsForRetrol = new ArrayList<>();
37             itemsForRetrol.add(new Item(description: "Excellent communication with team. Happy with how this sprint went", GOOD, itemFlag: false, itemVotes: 0, retro1));
38             itemsForRetrol.add(new Item(description: "Frustrated that I was blocked by database problem. Could not do my work", BAD, itemFlag: false, itemVotes: 2, retro1));
39             itemsForRetrol.add(new Item(description: "How can we solve the database issue?", QUESTION, itemFlag: false, itemVotes: -1, retro1));
40             itemsRepository.saveAll(itemsForRetrol);
41
42             //adding action items
43             List<ActionItem> actionItemsForRetrol = new ArrayList<>();
44             actionItemsForRetrol.add(new ActionItem(description: "Retro 1 - Action Item Number 1", retro1));
45             actionItemsForRetrol.add(new ActionItem(description: "Retro 1 - Action Item Number 2", retro1));
46             actionItemsForRetrol.add(new ActionItem(description: "Retro 1 - Action Item Number 3", retro1));
47             actionItemsRepository.saveAll(actionItemsForRetrol);

```

Figure 5: LoadDatabase.java : 'Local' profile java file used for testing purposes

The 'controllers' package contains the HTTP request methods , examples of which can be seen in *Figure 6*, that have been implemented for the API. These methods employ a HTTP verb and fetch or create data by calling another method from the 'retrosService' class which in turn requests or creates data from/in the datab ase via the JPA (Java Persistence API) repositories. The information is then served to the specified endpoint. These methods make up the core functionality of Retro as they serve the requested information to the user on the frontend once the correct API en dpoint has been reached by employ ing HTTP methods.

```

21 //using Spring annotations to take care of boilerplate code and assign this class as an API controller
22 @RestController
23 //setting base endpoint for API endpoints
24 @RequestMapping(value = "/api")
25 public class RetrosController {
26     ---
27     --- //creating instance of the retrosService
28     --- private final RetrosService retrosService;
29     ---
30     RetrosController(RetrosService retrosService) {
31         --- this.retrosService = retrosService;
32     }
33     ---
34     --- //GET method for displaying all retrospectives
35     --- @GetMapping("/retros")
36     --- ResponseEntity<List<Retro>> getAllRetros() {
37         --- return new ResponseEntity<>(retrosService.getAllRetros(), HttpStatus.OK);
38     }
39     ---
40     --- //POST method for creating a new retrospective
41     --- @PostMapping("/retros")
42     --- ResponseEntity<Retro> addNewRetro(@RequestBody Retro retro) {
43         --- return new ResponseEntity<>(retrosService.addRetro(retro), HttpStatus.CREATED);
44     }
45     ---
46     --- //GET method for displaying a retrospective by ID
47     --- //with try catch and exception if the retrospective(ID) is not found
48     --- @GetMapping("/retros/{id}")
49     --- ResponseEntity<Retro> getRetroById(@PathVariable Long id) {
50     ---     try {
51     ---         return new ResponseEntity<>(retrosService.getRetroById(id), HttpStatus.OK);
52     ---     } catch (Exception ex) {
53     ---         log.error("Retro not found " + id, ex);
54     ---         return new ResponseEntity<>(HttpStatus.NOT_FOUND);

```

Figure 6: Backend: Retrospectives controller

The 'models' package contains the structure of each model present in the project as well as how they relate to each other and should be stored in the H2 in-memory relational database. There are models present for 'retros', 'items' (as seen in Figure 7), 'sentiment analysis', and 'action items' as well as a 'base entity' super class that all models inherit from and an enumerator class, as seen in Figure 8, for the item type, good, question or bad.

The type assigned to an item is defined by which column the user creates the item from on the user interface – things that went well (good), questions about the sprint (question), or things that did not go so well (bad), respectively. This was important as only items of the type that have been posted to one of the mentioned columns are shown in that column on the frontend, i.e., items that have been created and HTTP posted from the 'things that went well' column are assigned the item type 'good' and are only shown in the 'things that went well' column as decided by an Angular pipe filter.

```

13 //adding data annotation which removes the need for getter/setter, toString etc boilerplate code
14 @Data
15 @EqualsAndHashCode(callSuper = true)
16 //no args constructor removes the need to add a constructor with no arguments
17 //and similar for all args constructor
18 @NoArgsConstructor
19 @AllArgsConstructor
20 @Entity
21 //mapping items to custom SQL table using JPA
22 @Table(name = "items")
23 //class inherits characteristics from BaseEntity
24 public class Item extends BaseEntity {
25     ---
26     --- //setting columns in items table
27     --- @Column(name = "description", nullable = false)
28     --- private String description;
29     ---
30     --- @Column(name = "type", nullable = false)
31     --- private ItemType type;
32     ---
33     --- @Column(name = "item_flag", nullable = false)
34     --- private boolean itemFlag;
35     ---
36     --- @Column(name = "item_votes", nullable = false)
37     --- private int itemVotes;
38     ---
39     --- //setting many to one relationship type
40     --- //joining table to retro table by retroId column
41     --- @ManyToOne(fetch = FetchType.LAZY, optional = false)
42     --- @JoinColumn(name = "retro_id", nullable = false)
43     --- @OnDelete(action = OnDeleteAction.CASCADE)
44     --- @JsonIgnore
45     --- private Retro retro;
46 }

```

Figure 7: Item class java file

```

1 package com.retrospective.models;
2
3 public enum ItemType {
4     --- GOOD,
5     --- BAD,
6     --- QUESTION
7 }

```

Figure 8: Item type enumerator java file

The 'repositories' package contains JPA repositories for action items, items, sentiment analyses and retrospectives. An example of this can be seen in [Figure 9](#), for the retrospectives JPA repository. The JPA repositories allow the persistence of the action items, items, and retros objects. They make it possible to map the objects to the H2 database and allow the data relating to them to be queried by the 'retrosService' and returned to the user via the user interface.

```

RetrosRepository.java
1 package com.retrospective.repositories;
2
3 import ...
7
8 public interface RetrosRepository extends JpaRepository<Retro, Long> {
9     Optional<Retro> findById(Long id);
10 }

```

Figure 9: Retrospectives JPA repository interface java file

The 'exceptions' package contains the custom exception handling classes created for each model present in the backend. Each exception handler is employed throughout the service class methods as a security measure and exception handling event that is unique to each model. In Figure 10, the 'RetrosNotFoundException' class is shown along with the message printed to the console in the event that a retrospective cannot be found in the database. This allows for simplified tracking -down and fixing of errors that may arise in the application as the model, repository, or method is causing the issue at hand known from the outputted message.

```

public class RetrosNotFoundException extends Exception {
    public RetrosNotFoundException() {
        super("Could not find retrospective.");
    }
}

```

Figure 10: RetrosNotFoundException - custom error handling method

The 'services' package contains the methods, as shown in Figure 11, that query each of the respective repositories. The methods contained within this class are called by the 'retrosController' class to create, read, update, and delete data in/from the database via the repositories.



```

13  @Service
14  public class RetrosService {
15      private final RetrosRepository retrosRepository;
16      private final ItemsRepository itemsRepository;
17      private final ActionItemsRepository actionItemsRepository;
18
19      RetrosService(RetrosRepository retrosRepository, ItemsRepository itemsRepository, ActionItemsRepository actionItemsRepository) {
20          this.retrosRepository = retrosRepository;
21          this.itemsRepository = itemsRepository;
22          this.actionItemsRepository = actionItemsRepository;
23      }
24
25      public List<Retro> getAllRetros() {
26          return retrosRepository.findAll();
27      }
28
29      public Retro addRetro(Retro retro) {
30          return retrosRepository.save(retro);
31      }
32
33      public Retro getRetroById(Long id) throws Exception {
34          return retrosRepository.findById(id)
35              .orElseThrow(Exception::new);
36      }
37
38      public List<Item> getRetroItemsById(Long id) {
39          return itemsRepository.findAllByRetroId(id);
40      }
41
42      public List<ActionItem> getRetroActionItemsById(Long id) {
43          return actionItemsRepository.findAllByRetroId(id);
44      }

```

Figure 11: Retrospectives service class java file

In particular, in order to perform a Sentiment Analysis, the 'getRetroItemsByIdForSentimentAnalysis' method in the service class is called. This method works by creating an Array List of stop words, which are words that do not have an association with positive or negative sentiment, by reading in a text file, as seen in Figure 12, filled with stop words through a Buffered Reader.

1	i
2	me
3	my
4	myself
5	we
6	our
7	ours
8	ourselves
9	you
10	your

Figure 12: Stop Words.

Similarly, a HashMap is created for a series of words and their associated positive/negative score from the Afinn Dictionary ([AFINN, 2011](#)), examples of which can be seen in Figure 13. The words and scores are then read in by a Buffered Reader and added to the HashMap.

1	abandon	-2
2	abandoned	-2
3	abandons	-2
4	abducted	-2
5	abduction	-2
6	abductions	-2
7	abhor	-3
8	abhorred	-3
9	abhorrent	-3
10	abhors	-3

Figure 13: AFINN Dictionary - words and associated sentiment scores

Once the files are read in, and the words are added to the respective ArrayList and HashMap, the method retrieves the words used by Software Development team members in a retrospective by retrieving them from the Items JPA Repository by using the retrospective ID, as seen in Figure 14.

The method then converts the retrieved list of words to a String, trims it to remove excess whitespace, and removes any non-alphabetic characters. Each word present is then added to an Array by splitting the String by the whitespace between each word.

```
// method to perform a sentiment analysis on a retrospective's items by retro id
public SentimentAnalysis getRetroItemsByIdForSentimentAnalysis(Long id) throws IOException {
    Retro retro = new Retro();
    SentimentAnalysis sentimentAnalysis = new SentimentAnalysis();
    retro.setId(id);

    float retrospectiveItemScore = 0;
    String wordLine;

    ArrayList<String> stopWords = new ArrayList<>();
    BufferedReader stopWordsBuffReader = new BufferedReader(new FileReader("src/main/resources/Dictionaries/stopWords.txt"));

    while ((wordLine = stopWordsBuffReader.readLine()) != null) {
        stopWords.add(wordLine);
    }
    stopWordsBuffReader.close();

    Map<String, String> AFINNDictionary = new HashMap<>();
    BufferedReader AFINNBuffReader = new BufferedReader(new FileReader("src/main/resources/Dictionaries/AFINN-en-165.txt"));
    while ((wordLine = AFINNBuffReader.readLine()) != null) {
        String[] parts = wordLine.split(regex: "\\t");
        AFINNDictionary.put(parts[0], parts[1]);
    }
    AFINNBuffReader.close();

    String retroItemWords = (itemsRepository.findAllByRetroId(id).toString().replaceAll(regex: "[^a-zA-Z]", replacement: " ").trim());
    String[] words = retroItemWords.split(regex: " ");
}
```

Figure 14: Sentiment Analysis - Reading in dictionaries and retrieving retrospective item descriptions.

As seen in Figure 15, the method then uses a for loop to compare each word against the stop words and if the word is present in the list of stop words, it is not assigned a score. If a word is not present amongst the stop words, it is assigned a score based on the associated positive/negative sentiment behind it. An integer then keeps track of the overall score of the sentiment behind a retrospective's items, and the final score is saved to the Sentiment Analysis (through an instance of the Sentiment Analysis class) associated with a retrospective's items, along with the retrospective and its ID, and the array of words analysed.

```

for (String word : words) {
    if (stopWords.contains(word.toLowerCase())) {
        // skip word
        System.out.println("Word skipped: " + word);
    } else {
        if (AfinnDictionary.get(word) != null) {
            String wordScore = AfinnDictionary.get(word.toLowerCase());
            retrospectiveItemScore = retrospectiveItemScore + Integer.parseInt(wordScore);
            sentimentAnalysis.setScore(retrospectiveItemScore);
        }
    }
}
sentimentAnalysis.setRetro(retro);
sentimentAnalysis.setId(retro.getId());
sentimentAnalysis.setRetroItems(retroItemWords);

return sentimentAnalysis;

```

Figure 15: Sentiment Analysis - words being assigned a score and sentiment analysis conducted.

In regard to the service methods that allows retrospective items and action items to be exported to a CSV file, the use of the Super CSV Java library ([Graversen, 2015](#)) was employed, which was added as a Maven dependency in the POM file.

This method works by first setting the response type of the method to be a CSV file and creating the file, then assigning the file name which includes information about the retrospective such as the retrospective title, ID number and the date and time it was downloaded at.

After this the retrospective items are retrieved and added to a list. The item IDs, dates and times created at, descriptions, item types and item votes received are then written to the CSV file under each of the respective headings shown in [Figure 16](#).

```

public List<Item> getRetroItemsAndExportToCSV(Long id, HttpServletResponse response) throws IOException, RetrosNotFoundExcepion {
    response.setContentType("text/csv");
    DateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss");
    String currentDateTime = dateFormatter.format(new Date());

    String headerKey = "Content-Disposition";
    String headerValue = "attachment; filename=retrospective_" +
        this.getRetroById(id).getName() +
        "_items_ID-" + id + "-" + currentDateTime + ".csv";

    response.setHeader(headerKey, headerValue);

    List<Item> listItems = this.getRetroItemsById(id);

    ICsvBeanWriter csvWriter = new CsvBeanWriter(response.getWriter(), CsvPreference.STANDARD_PREFERENCE);
    String[] csvHeader = {"Item ID", "Created", "Description", "Type", "Votes"};
    String[] nameMapping = {"id", "created", "description", "type", "itemVotes"};

    csvWriter.writeHeader(csvHeader);

    for (Item item : listItems) {
        csvWriter.write(item, nameMapping);
    }

    csvWriter.close();

    return itemsRepository.findAllByRetroId(id);
}

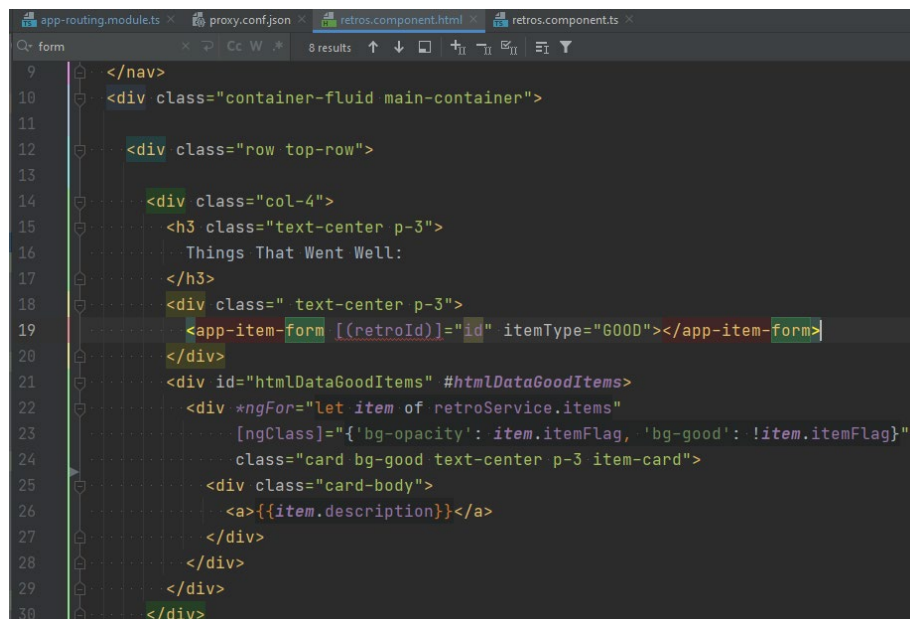
```

Figure 16: Exporting retrospective items and action items to a CSV file.

### 2.3.2. The Frontend:

The frontend is similarly separated out into directories. The frontend contains directories for components, models, and services. The components directory is made up of the action item form, item form, retros, retros list, archived retros, sentiment analysis and sentiment analysis list components.

The action item and item forms are both used in the retros component in the HTML file, as seen in *Figure 17*, when a user enters text in the input field for an action item or item to be added to a retrospective. Instead of the code for the forms being directly written into the retros component, they have been separated out into their own form components in accordance with S.O.L.I.D. design principles, to ensure they have a single responsibility.



```
9 </nav>
10 <div class="container-fluid main-container">
11
12 <div class="row top-row">
13
14 <div class="col-4">
15 <h3 class="text-center p-3">
16   Things That Went Well:
17 </h3>
18 <div class="text-center p-3">
19   <app-item-form [(retroId)]="id" itemType="GOOD"></app-item-form>
20 </div>
21 <div id="htmlDataGoodItems" #htmlDataGoodItems>
22 <div *ngFor="let item of retroService.items"
23   [ngClass]="{'bg-opacity': item.itemFlag, 'bg-good': !item.itemFlag}"
24   class="card bg-good text-center p-3 item-card">
25 <div class="card-body">
26 <a>{{item.description}}</a>
27 </div>
28 </div>
29 </div>
30 </div>
```

*Figure 17: Retrospectives component HTML file*

As can be seen in *Figure 18*, validation is carried out on the text to ensure it is not an empty String. Once the form is validated and submitted, the item or action item is added to the retrospective's items /action items.

```

ngOnInit(): void {
  this.itemForm = new FormGroup( controls: {
    // performing validation on entries to the item form
    description: new FormControl(this.description, validatorOrOpts: [
      Validators.required
    ])
  });
}

onSubmit() {
  const item: Item = {
    description: this.description,
    type: this.itemType
  };
  // on submission call the add item to retro method, add the item to the specified retro
  // tslint:disable-next-line:no-shadowed-variable
  this.retroService.addItemToRetro(item, this.retroId).subscribe( next: (item: Item) => {
    this.retroService.items.unshift(item);
    console.log(item);
  });
}
}

```

Figure 18: Item form methods

The retros , retros list , archived retros, sentiment analysis and sentiment analysis list components each respectively call methods from ' *retrosService.ts*' to retrieve, create, update, or delete data from/to the backend . This works by subscrib ing to methods present in the ' *retrosService.ts*' file in order to become aware of any changes that are made, in which case the methods are called, and the information is fetched again with the changes made , as seen in *Figure 19* for the retros component methods .

```

sortItems(itemsArray: Item[]): Item[] {
  // sort items by if item is done or not: method needs work
  return itemsArray.sort( compareFn: (a :Item , b :Item ) => {
    return (a.itemFlag === b.itemFlag) ? 0 : a.itemFlag ? 1 : -1;
  });
}

getRetroActionItems() {
  // get retros action items
  console.log('fetching retro action items');
  this.retroService.getRetroActionItemsById(this.id).subscribe( next: response => {
    this.actionItems = response.body;
  });
}

updateItemDescription(item: Item) {
  console.log(item.id);
  console.log(item.description);
  this.retroService.updateRetroItemById(item);
}

deleteRetroItemById(itemId) {
  this.retroService.deleteRetroItemById(itemId).subscribe( next: (response :void) => {
    this.retroService.items = this.retroService.items.filter((item :Item) => {
      return item.id !== itemId;
    });
  });
}

```

Figure 19: Retrospectives Typescript file

The 'services' directory contains all of the methods that make HTTP requests to the backend API and gets the response, as seen in *Figure 20*. The endpoint addresses have been added to this file as strings to allow for simple modification if the endpoints change over time. For example, the 'this.urlPrefix' seen in the *addRetro* method in *Figure 20* relates to the urlPrefix string which has a value of '/api/retros/'.

```

// method to get all retrospectives from API
getRetros(): Observable<HttpResponse<Retro[]>> {
  return this.http.get<Retro[]>(
    this.urlPrefix, {options: {observe: 'response'}});
}

// method to add a retrospective to the database via the API
addRetro(retro: Retro): Observable<Retro> {
  return this.http.post<Retro>(this.urlPrefix, retro, this.httpOptions);
}

// method to get a retrospective from the API by ID
getRetroById(id): Observable<HttpResponse<Retro>> {
  return this.http.get<Retro>(url: this.urlPrefix + id, {options: {observe: 'response'}});
}

// method to update a retros contents by ID
updateRetroById(updateRetro: Retro): void {
  this.http.put<Retro>(url: this.urlPrefix + updateRetro.id, updateRetro, this.httpOptions).subscribe(next response => {
    this.retros = this.retros.filter((retro: Retro) => {
      return retro.id !== updateRetro.id;
    });
    this.retros.push(updateRetro);
    console.log(response);
  }, error error => {
    console.error('Error updating item: ', error);
  });
}

```

Figure 20: Retrospectives services Typescript file

The 'proxy.conf.json' file, as seen in *Figure 21*, configures the proxy that allows the front and back end to communicate as it tells the methods within the service file the base URI to direct HTTP requests to. It also disguises the base endpoint address of the API behind '/api/\*\*/' for security purposes.

```

{
  "/api/**": {
    "target": "http://localhost:8080",
    "secure": false
  }
}

```

Figure 21: proxy.conf.json - allows backend and frontend to communicate.

The Angular pipes ([Angular, 2021](#)) present in the frontend allow for the transformation of data that is received from the service methods. In Retro, pipes are used to filter the data that is being returned from the backend API.

When returning a list of retrospectives that are in progress, a pipe was used to filter out retrospectives that have been marked as archived. This was done by checking the Boolean flag 'archiveRetroFlag' to see if it had been raised using the pipe seen in *Figure 22*. If it had not been raised (i.e., not had a value of 1/true), the retrospective would be displayed amongst the list of in progress retrospectives, as can be seen in *Figure 23*.

```

export class RetroArchivedPipe {
  transform(retros: Retro[], retroArchiveFilter: boolean) {
    if (!retroArchiveFilter) {
      return retros.filter(retro => {
        return retro.archiveRetroFlag === false;
      });
    } else {
      return retros.filter(retro => {
        return retro.archiveRetroFlag === true;
      });
    }
  }
}

```

Figure 22: Archived retrospectives pipe method

```

<tr *ngFor="let retro of retros | retroFilter: retroNameFilter | retroArchiveFilter: retroArchiveFilter: false">
  <a [routerLink]="['/retros', retro.id]" style="...">
    <td class="retro-details"><strong>{{retro.name}}</strong></td>
  </a>

```

Figure 23: Archived retrospectives pipe HTML filter

Similarly, a pipe is used when searching for a retrospective on the homepage of Retro , the archived retrospectives, and the sentiment analysis page by filtering by retrospective title .

Pipes are also used when displaying the list of retrospective items. There is a pipe present to filter and restrict items by type (good, bad, questions) to the 'things that went well', 'things that did not go so well' or 'questions' columns present on the UI , as seen in *Figure 24*. If an item is not of the specified type, it will not be shown in the column. This allows for only the relevant items to be displayed in each column on the UI. An example of the pipe being used is shown in *Figure 26*.

```

export class RetroItemPipe implements PipeTransform {
  transform(items: any[], itemFilter: string): any {
    if (!items || !itemFilter) {
      return items;
    }
    return items.filter(itemType => itemType.type.indexOf(itemFilter) !== -1);
  }
}

```

Figure 24: Angular pipe to filter items by type

Similarly, there is a pipe for sorting items by votes in descending order , as shown in *Figure 25*. This allows for the most upvoted items to be displayed at the top of the column. An example of the pipe being used is shown in *Figure 26*.



```

export class RetroItemVotesPipe implements PipeTransform {
  transform(items: Item[], args?: any): Item[] {
    return items.sort( compareFn: (a: Item, b: Item) => {
      return b.itemVotes - a.itemVotes;
    });
  }
}

```

Figure 25: Item votes Angular filter

```

<div *ngFor="let item of retroService.items | itemFilter: itemFilter: 'GOOD' | sortByVotes"
  [ngClass]="{'bg-opacity': item.itemFlag, 'bg-good': !item.itemFlag}"
  class="card bg-good text-center p-3 item-card">

```

Figure 26: Use of Angular pipes for item types and item votes in HTML.

In regard to the routing of the application, the routes were set up as shown in Figure 27 to be linked to a specific component, allowing the application to know which HTML, CSS and Typescript files to load when the route endpoints are reached. A Hash was also used in the configuration of the routing to safeguard the API endpoints.

```

// routing for frontend endpoints
const routes: Routes =
  [{path: 'retros', component: RetrosListComponent},
  {
    path: '',
    redirectTo: '/retros',
    pathMatch: 'full'
  },
  {path: 'retros/:id', component: RetrosComponent},
  {path: 'archived-retros', component: ArchivedRetrosComponent},
  {path: 'retros/:id/sentiment-analysis', component: SentimentAnalysisComponent},
  {path: 'sentiment-analysis', component: SentimentAnalysisListComponent}
];

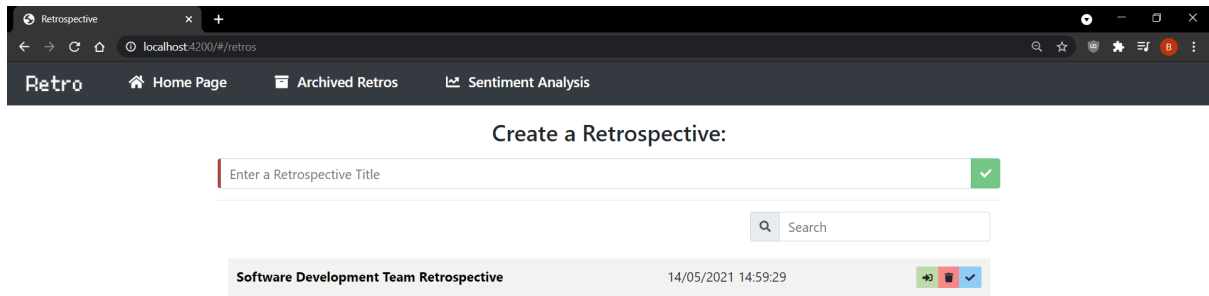
const routerConfig = {
  useHash: true
};

```

Figure 27: Application routing

## 2.4. Graphical User Interface (GUI)

### 2.4.1. Home Page:



*Figure 28: GUI - Retro Home page*

*Figure 28* represents the home page of Retro's GUI. On this page, users can create a new retrospective and have it saved to the backend database of retrospectives. Users can also see a list of existing retrospectives, including details about when a retrospective was created. Similarly, users can also search for a specific retrospective using the Angular pipe search bar, to filter the list of retrospectives to a specific one.

On this page users can also delete a retrospective by clicking the trash bin button in red, archive a retrospective after they are finished with it by clicking the tick button in blue, or enter a retrospective to see its items and action items by clicking the arrow button in green.

Users can also visit any of the other pages present on Retro from this page by clicking the options present on the navbar, which is present throughout the application.

## 2.4.2. Retrospective Page:

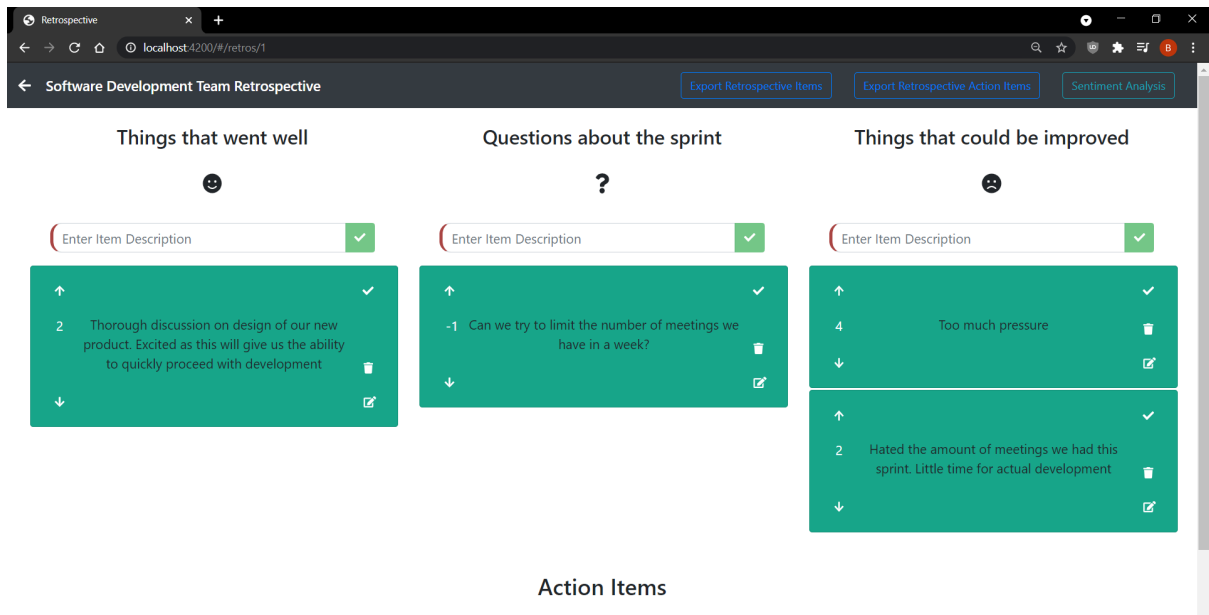


Figure 29: GUI - Individual retrospective page with items

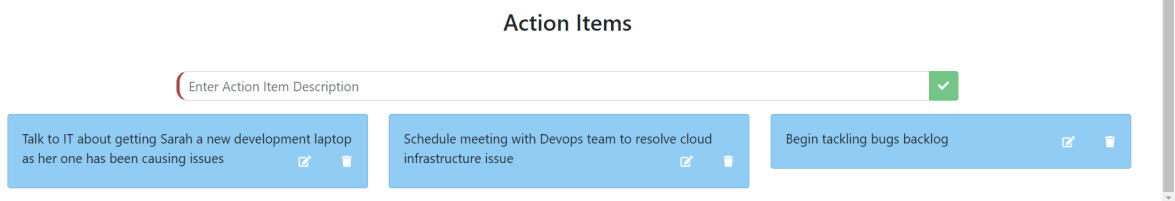


Figure 30: GUI - Individual retrospective page with action items

When a user clicks the arrow button in green or the retrospective title, as seen in [Figure 28](#), they enter a retrospective's details page. The GUI of the retrospective pages can be seen in [Figure 29](#) and [Figure 30](#). On this page, users will see the retrospectives items and action items. From this page users can also export the retrospectives items and action items to CSV files for further review by clicking the buttons on the navbar to carry out these actions. Similarly, users can also be brought to the Sentiment Analysis page for retrospectives by clicking the respective button on the navbar to do so.

The retrospectives action items are displayed on the retrospective , and users can also create action items, edit their descriptions, and delete them .

Details about things that went well, questions about the sprint, and things that did not go so well are displayed. Users can also create items in any of the columns and have them stored to the backend database . Users can also upvote or downvote the retrospective items and they will be sorted according to their score, delete an item, edit an items description, and mark an item as reviewed. Marking an item as reviewed will change the CSS colour properties of a retrospective item's Bootstrap card, clearly showing the user it has been reviewed and discussed, as seen in [Figure 31](#).

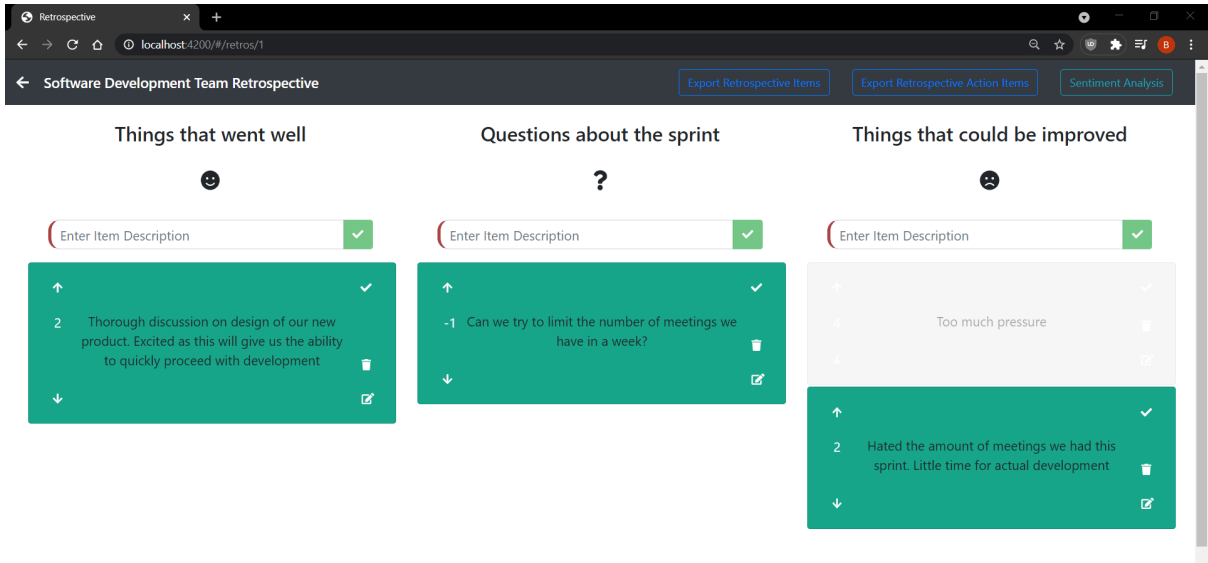


Figure 31: GUI - Retrospective page- reviewed item

### 2.4.3. Archive Page:

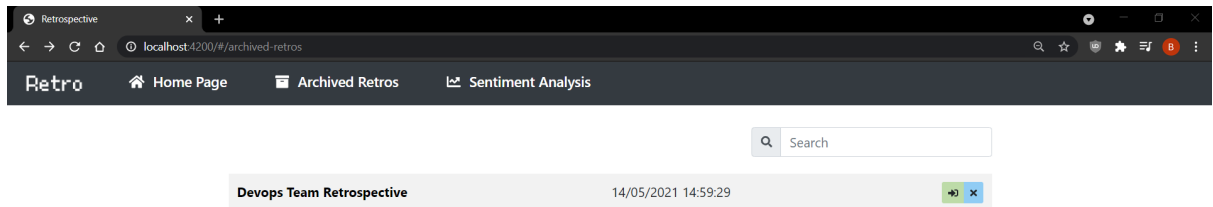
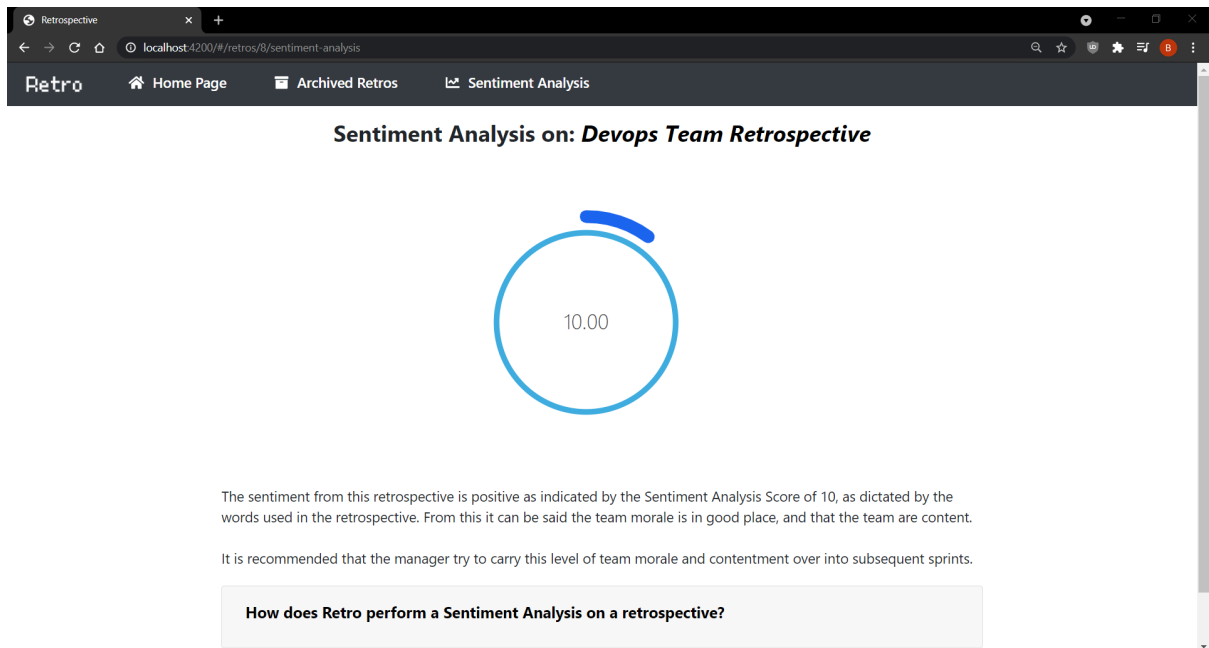


Figure 32: GUI - Archived retrospectives page

Figure 32 shows the Archived Retrospectives page. This page lists all of the retrospectives that have been marked as archived by clicking the blue tick button as seen on Figure 28. These are retrospectives that the team in questions are finished with and have been reviewed by them. As can be seen in Figure 32, the buttons beside the shown archived retrospective are to enter it to view its contents by clicking the green button, and to unarchive (bring it back into the list of ongoing retrospectives) it by clicking the blue 'x' button.

#### 2.4.4. Sentiment Analysis Page:



*Figure 33: GUI - Sentiment Analysis Page – displaying a positive Sentiment Analysis.*

As seen in *Figure 33*, the Sentiment Analysis that has been carried out for a retrospective is displayed. On this page, the title of the retrospective is shown along with a circular progress bar. This progress bar measures the level of positive sentiment present in a retrospective, ranging from one to one hundred. If a retrospective receives an overall positive sentiment score from the analysis based on the words used, that score is displayed using the progress bar. This allows Software Development team managers to see the data from a Sentiment Analysis on their team's retrospective in a visual format.

Similarly, a recommendation message that changes depending on if the overall sentiment of the retrospective in question is positive or negative is shown. If the sentiment is positive, the message seen in *Figure 33* will be shown. However, if overall sentiment for a retrospective is negative, the circular progress bar will not be shown the recommendation message shown in *Figure 34* will be shown.

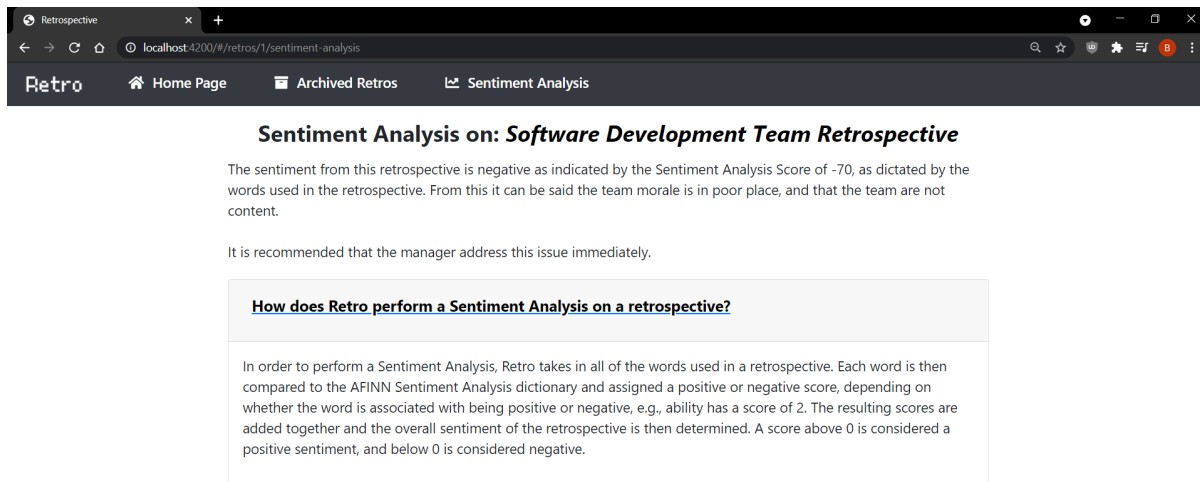


Figure 34: GUI - Sentiment Analysis Page – displaying a negative Sentiment Analysis.

An explanation for how Retro performs a Sentiment Analysis on retrospectives is shown at the bottom of each Sentiment Analysis inside a Bootstrap Accordion card that can be collapsed or expanded by the user when they wish to read it.

#### 2.4.5. Search:

The GUI contains a search bar that allows users to enter a retrospective title and they can filter the list of retrospectives that are in progress in order to find the one they are looking for, as seen in Figure 35 and Figure 36. This functionality is also present on the Archived retrospectives page.

This functionality was created using a pipe in Angular which works as a filter. When the user enters a retrospective title into the search bar, the pipe immediately starts to filter the list of retrospectives present and if any of the listed retrospectives do not contain the characters present in what the user has searched for, the retrospective is removed from the displayed list, and only the corresponding retrospective are returned.

## Create a Retrospective:

The screenshot shows a form titled "Create a Retrospective:". At the top is a text input field with the placeholder "Enter a Retrospective Title" and a green checkmark icon on the right. Below this is a search bar with a magnifying glass icon and the text "Search". Underneath the search bar is a table listing five retrospectives. Each row contains the retrospective name, its creation timestamp, and three action icons: a green arrow pointing right, a red trash can, and a blue checkmark.

Retrospective Title	Timestamp	Actions
SRE Team Retrospective	06/05/2021 11:21:58	→ 🗑️ ✓
Data science team retrospective	06/05/2021 11:21:42	→ 🗑️ ✓
Devops Team Retrospective	06/05/2021 11:21:31	→ 🗑️ ✓
Software Development Team Retrospective	06/05/2021 11:21:20	→ 🗑️ ✓
First Retrospective	06/05/2021 11:20:40	→ 🗑️ ✓

Figure 35: GUI - Searching for a retrospective

## Create a Retrospective:

This screenshot is similar to Figure 35, but the search bar now contains the word "software". The table below the search bar has been filtered to show only one entry: "Software Development Team Retrospective" with the timestamp "06/05/2021 11:21:20" and the same three action icons.

Retrospective Title	Timestamp	Actions
Software Development Team Retrospective	06/05/2021 11:21:20	→ 🗑️ ✓

Figure 36: GUI - Searching for a Retrospective

### 2.4.6. Modal Dialogs:

The GUI also contains a modal dialogs for when it is necessary to restrict users to an action before they can continue with regular use of the applications functionality. This was implemented with the use of Angular powered Bootstrap, and it is present in the application when a user attempts to delete a retrospective, as seen in *Figure 37*, delete a retrospective item or action item, as seen in *Figure 38*, and when attempting to edit a retrospective item or action item, as seen in *Figure 39*.

When the user clicks the respective button to delete or edit, a modal dialog opens, and the user is shown the content of the modal. The user then confirms their choice to delete or edit and the respective method to delete or edit is called from within the modal method.

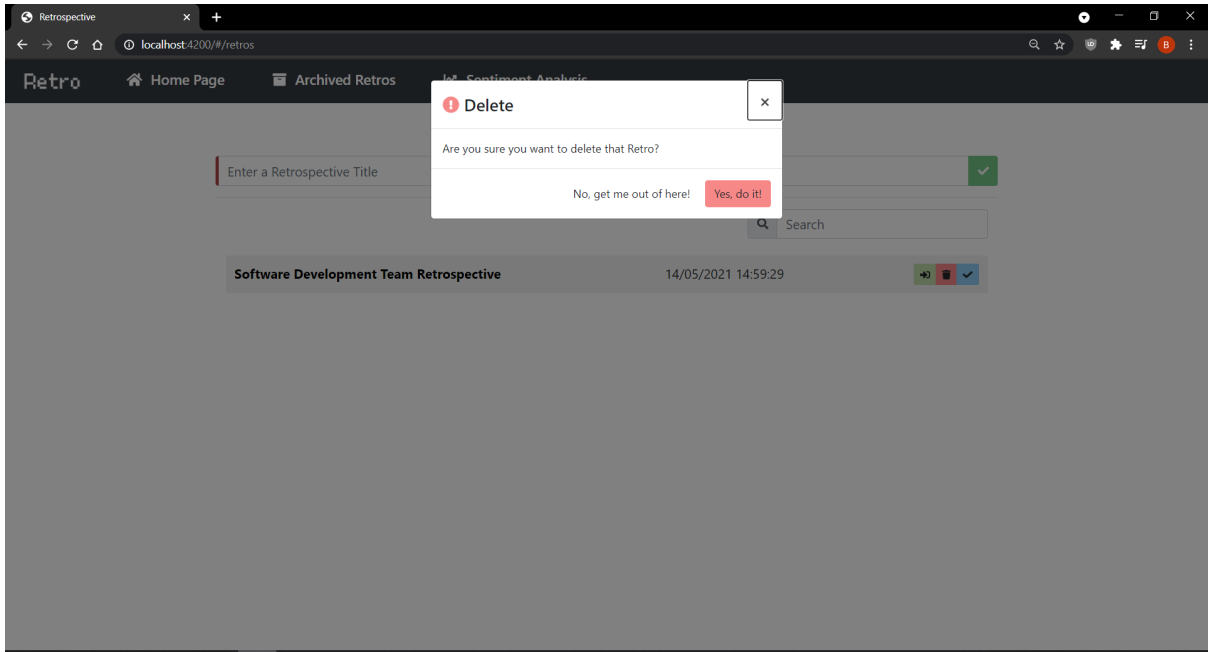


Figure 37: GUI - Modal dialog to delete a retrospective.

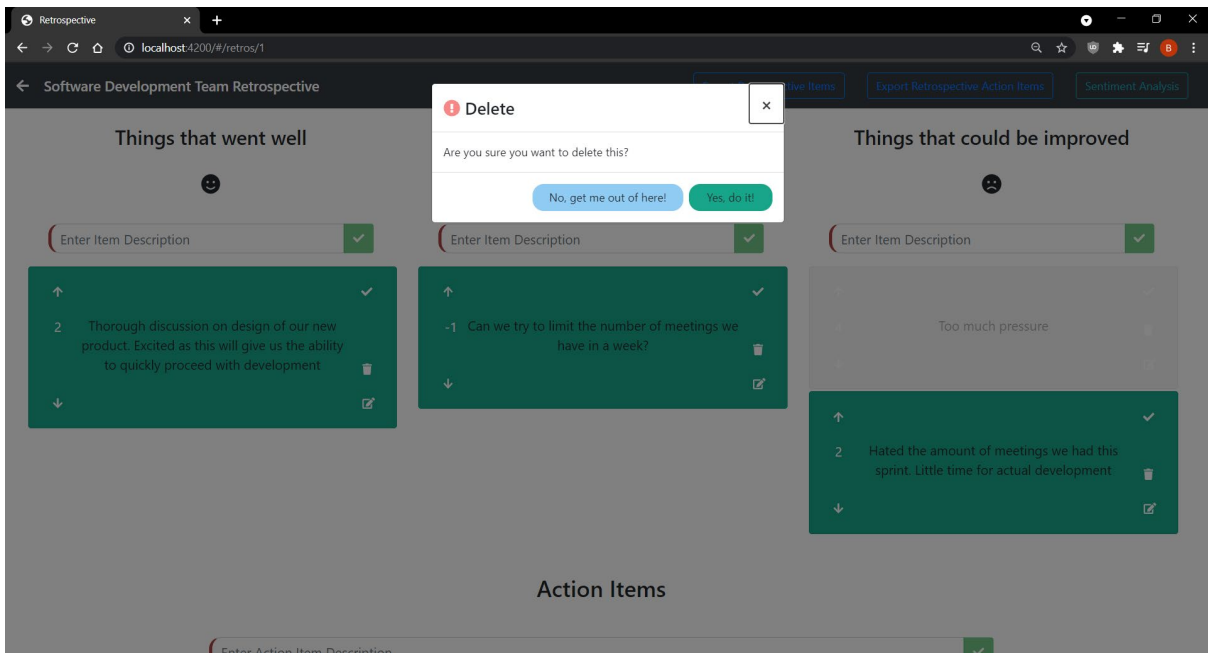


Figure 38: GUI - Modal dialogue to delete a retrospective item.



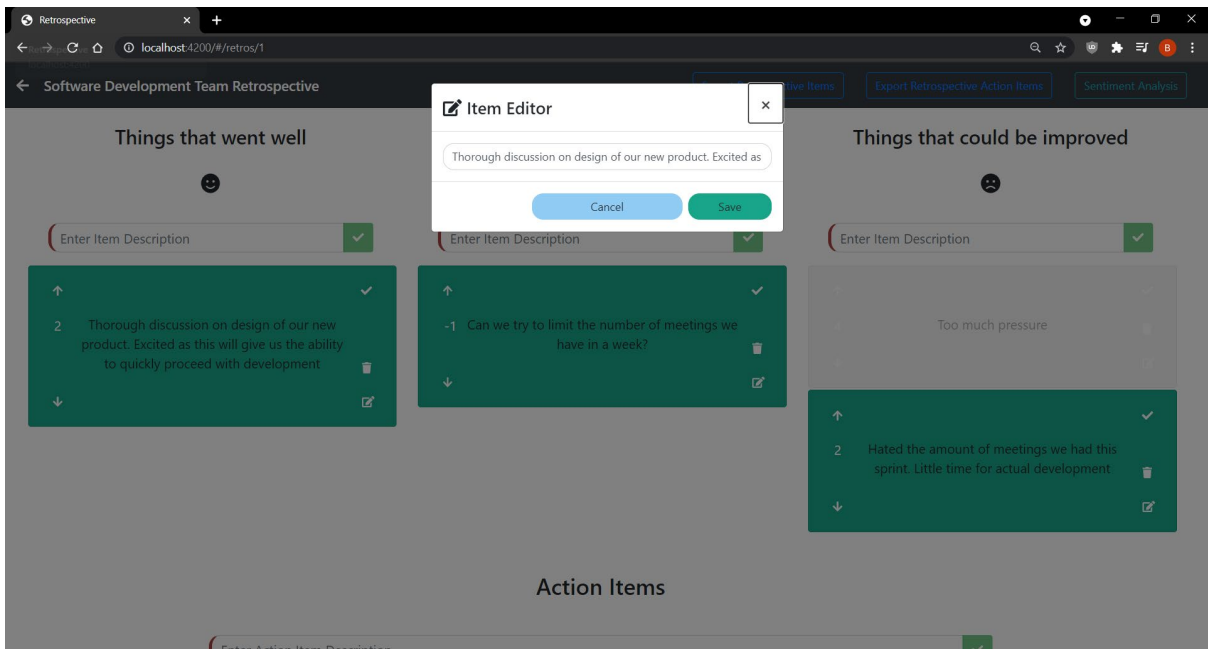


Figure 39: GUI - Modal dialog to edit a retrospective item.

#### 2.4.7. Voting System:

Similarly, the GUI also contains a voting system for retrospective items. This allows users to up vote and agree with an item a colleague has added, or downvote and disagree with a point they have made. The items with the most upvotes in a row ('Things that went well', 'Questions about the sprint', and 'Things that could be improved') will be sorted to the top of that row. The voting system works by using an Angular pipe that takes the list of items in a row and sorts each time by vote in descending order. For example, in Figure 40, an item with the description 'Too much pressure' is at the bottom of the 'Things that could be improved' column. After receiving the most upvotes in that column, it is then moved to the top of the column's items by the pipe, as seen in Figure 41.

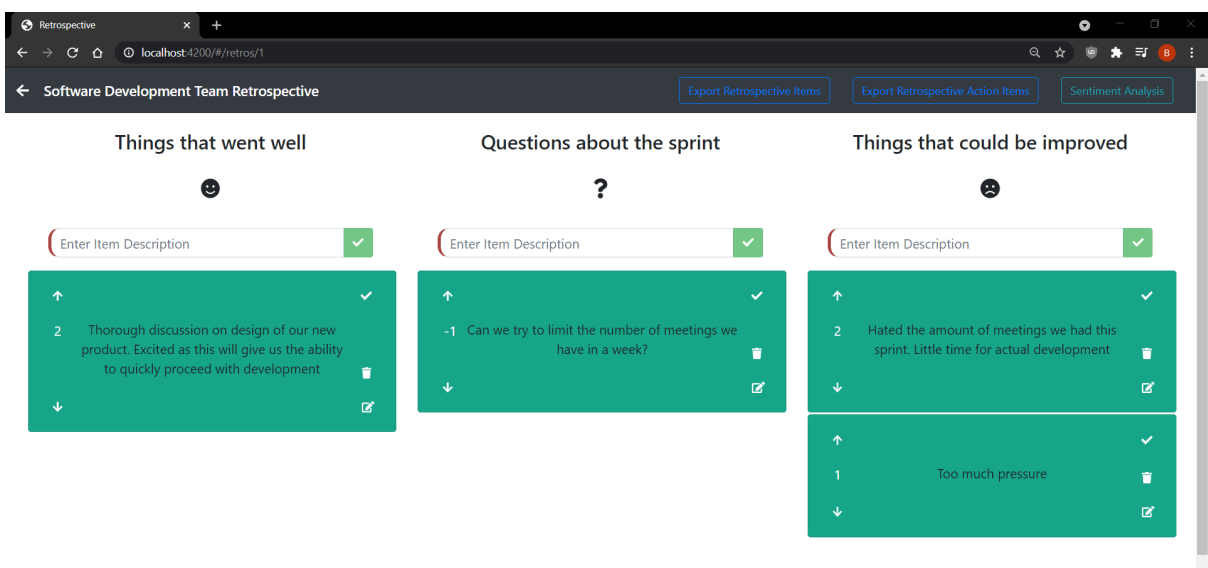


Figure 40: GUI - Voting system

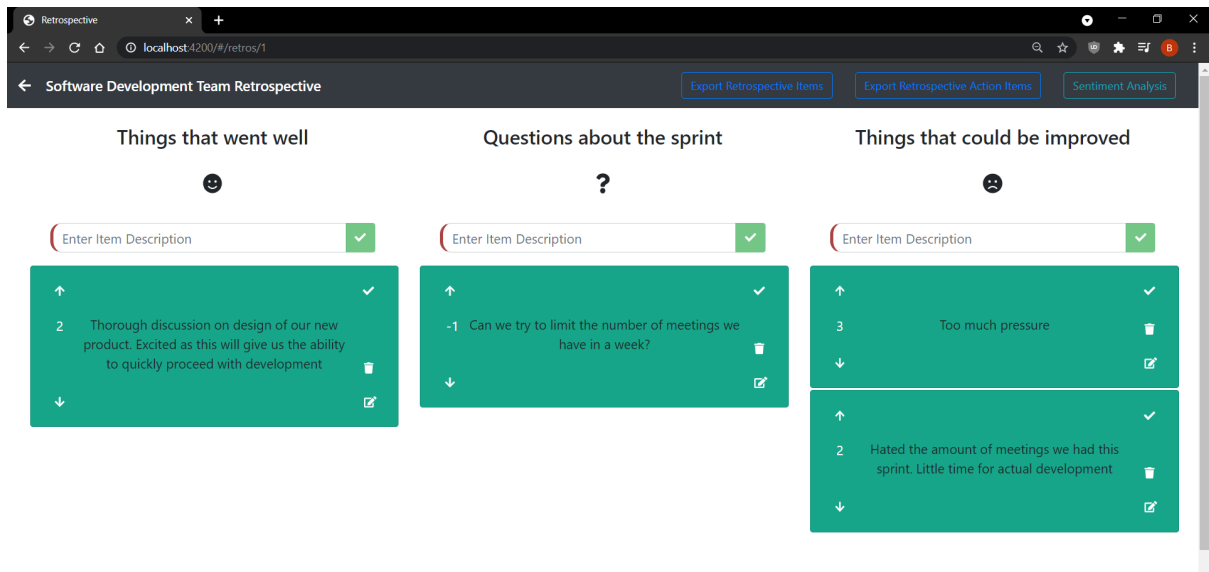


Figure 41: GUI - Voting System

#### 2.4.8. Retrospective Timestamps:

The GUI contains a timestamp present on the homepage of Retro for each retrospective. This timestamp indicates when each retrospective was created. This allows users to easily find a retrospective based on when it was created but it is also how the retrospectives are sorted by default.

When the user is not searching for a retrospective, they are sorted by time of creation in ascending order, as seen in [Figure 35](#). This works by using a sort method on the array of retrospectives present on the home page to sort and display them by date.

## 2.5. Testing

### 2.5.1. Backend

A series of unit tests were written for the backend system of Retro using JUnit5 ([JUnit, 2021](#)). When carrying out testing for the backend, a high percentage of code coverage was needed in order to ensure that each part of functionality within Retro's backend was working as intended. To do this, unit tests were written for the application configuration, the service methods, and the controller methods.

Similarly, when each unit test was being written, the opposite of each test was also created. For example, as seen in [Figure 42](#), a test was written for one of the service (RetrosService.java) methods to retrieve all retrospectives when the retrospectives exist, by creating dummy retrospectives within the test, and a test was also written that tries to retrieve all retrospectives when no retrospectives exist. In doing this, both test cases are being covered in the method and ensuring any exceptions thrown are handled within the service method (by adding handling for that exception to the method), and that the end user is seeing an appropriate error message and not information that compromises the system. Covering both cases also ensures that the test coverage for the service and controller methods is an accurate representation of the coverage present within the system.

```

38 // test to get all retros from the retros repository when they exist there
39 // passes by list sizes matching by using assertions
40 @Test
41 void getAllRetrosWhenRetrosExist() {
42     // given
43     List<Retro> expectedRetrosList = Arrays.asList(
44         new Retro( name: "Retro one", archiveRetroFlag: false),
45         new Retro( name: "Retro two", archiveRetroFlag: true),
46         new Retro( name: "Retro three", archiveRetroFlag: false)
47     );
48     when(retrosRepository.findAll()).thenReturn(expectedRetrosList);
49
50     // when
51     List<Retro> actualRetrosList = retroService.getAllRetros();
52
53     // then
54     Assertions.assertEquals(expectedRetrosList.size(), actualRetrosList.size());
55 }
56
57 // test to get all retros when no retros exist in retros repo
58 // returns both as empty if successful, done using assertions
59 @Test
60 void getAllRetrosWhenNoRetrosExist() {
61     // given
62     List<Retro> expectedRetrosList = Arrays.asList();
63     when(retrosRepository.findAll()).thenReturn(expectedRetrosList);
64
65     // when
66     List<Retro> actualRetrosList = retroService.getAllRetros();
67
68     //then
69     Assertions.assertEquals(expectedRetrosList.size(), actualRetrosList.size());
70 }

```

Figure 42: Service tests- Unit tests for retrieving retrospectives.

As a result of writing unit tests for the application context, service methods and the controller methods, code coverage of 75% was achieved in all classes and 66% coverage for each method within each class, as evidenced by Figure 43 and Figure 44. Details of what each test does can be seen in Figure 44 from the test names.

Coverage: All in retrospective-api x

75% classes, 40% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
aj			
antlr			
ch			
Class50			
com	75% (9/12)	66% (42/63)	40% (57/1...
images			
java			
javassist			
javax			
jdk			
lombok			
META-INF			
net			
netscape			
org			

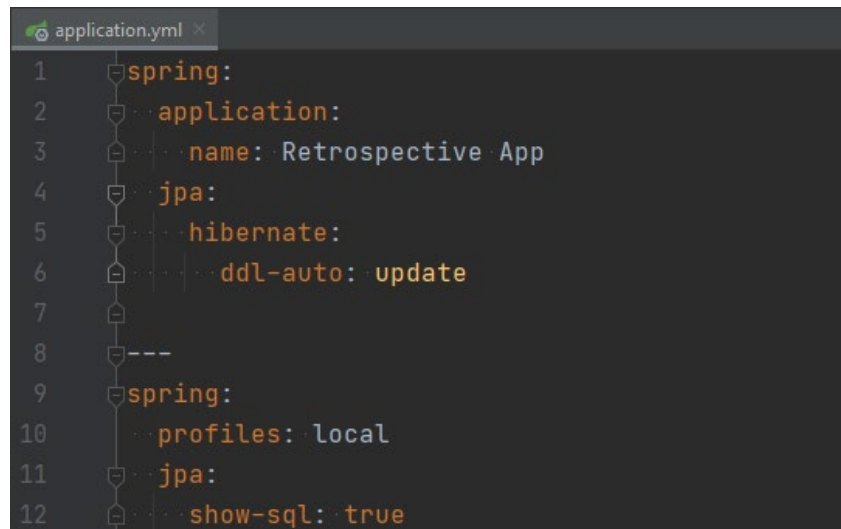
Figure 43: Code coverage present in each class and their respective methods.

Run: All in retrospective-api x

<default package>	899 ms
RestServiceApplicationTests	245 ms
contextLoads()	245 ms
RetrosControllerTest	532 ms
postNewItemToExistingRetro()	337 ms
postNewRetroWhenRetroGiven()	55 ms
getRetroByIdWhenIdExists()	26 ms
getAllRetrosWhenRetrosExist()	24 ms
getActionItemsByRetroIdWhenActionItemsExist()	27 ms
postNewActionItemToExistingRetro()	32 ms
getItemByRetroIdWhenItemsExist()	31 ms
RetrosServiceTest	122 ms
getRetroItemsByIdWhenItemsDoNotExist()	17 ms
postNewRetroWhenRetroGiven()	12 ms
postItemByIdWhenRetroExist()	14 ms
getAllRetrosWhenRetrosExist()	7 ms
getAllRetrosWhenNoRetrosExist()	9 ms
postActionItemByIdWhenRetroExists()	8 ms
getRetroByIdWhenRetroDoesNotExist()	10 ms
getRetroItemsByIdWhenItemsExist()	10 ms
postActionItemByIdWhenRetroDoesNotExist()	6 ms
getRetroByIdWhenRetroExists()	8 ms
getRetroActionItemsByIdWhenActionItemsDoNot	7 ms
getRetroActionItemsByIdWhenActionItemsExist()	6 ms
postItemByIdWhenRetroDoesNotExist()	8 ms

Figure 44: All tests passing for the application, service, and controller tests.

To aid the testing of the backend, a 'local' Spring framework profile was implemented, as seen in *Figure 45*, that allows for testing the backend and the application in general to make sure the features implemented to date are working correctly. This is useful for the frontend too, as it shows that the back and front ends are communicating correctly.



```
1  spring:
2    application:
3      name: Retrospective App
4    jpa:
5      hibernate:
6        ddl-auto: update
7
8  ---
9  spring:
10   profiles: local
11   jpa:
12     show-sql: true
```

*Figure 45: YAML configuration file for 'local' profile*

## 2.5.2. Frontend

### 2.5.2.1. Usability Testing:

A series of usability tests were carried out with Retro's target end users in order to find out how simple it is to use Retro to perform Agile sprint retrospectives in an effective and efficient manner that provides satisfaction. To do this, five usability tests were carried out: a five second test, trunk test, think aloud test, tree test, and a click test. A System Usability Scale survey was also filled out by [participant one](#) and included as a debrief.

An informed consent (see [Informed Consent Form](#)) was signed by the participant following research integrity guidelines and best practices. A complete version of the results is available in the [User Testing Results appendix](#). The five tests carried out were a five second test, trunk test, think aloud test, tree test, and a click test. An Ethics Application Form signed by my project supervisor can also be seen in the [NCI Ethics Application Form appendix](#).

#### 2.5.2.1.1. Five Second Test:

The five second test was fairly successful, the participant recalled that it was the home page shown, and that they could create and search for retrospectives on the page. As a result, it would be useful to put more material about Retro on the homepage so the user will remember what the application is and what it does.

#### 2.5.2.1.2. Trunk Test:

The trunk test showed that the retrospective page has a strong message hierarchy and defined structure as they were able to recall the page name, key parts of functionality and give reasoning as to why that functionality was essential. As the retrospective page is the core functionality of Retro, this result was admirable.

#### 2.5.2.1.3. Think Aloud Test:

The think aloud test proved very useful. The participant was able to clearly carry out the tasks given and provide solid reasoning as to why they took each step. This proves the flow and hierarchical structure of Retro has good usability design.

#### 2.5.2.1.4. Tree Test:

The tree test has reinforced the belief that the architecture of Retro is clear and simple to use. The user was presented with three tasks to complete, was easily able to navigate through Retro's structure to find the correct location.

#### 2.5.2.1.5. Click Test:

The click test questions were a mix of behavioural and comprehension style questions in an attempt to get the participant to explain the reasoning of what they were clicking on and why. This insight into the users thinking is valuable when considering making changes to the website usability before a final design is committed to.

#### 2.5.2.1.6. Debrief:

After carrying out user testing, an exit survey (see [System Usability Scale Exit Survey](#)) was applied. The result was high from the participant, with the SUS (System Usability Scale) score being 80. As the SUS average score is 68 ([Fernandez, 2021](#)), Retro has scored well in terms of its usability.

There is still some work that could be done to add to the usability of Retro from looking at the SUS survey results, but it is largely positive and shows that Retro has a strong level of usability present in its design.

## 2.6. Evaluation

When re-examining the [objectives](#) outlined in the proposal of Retro and the [aims](#) stated in the introduction of this technical report, I believe it is clear that I have met both the aims and objectives of the project.

When I was setting out to develop Retro at the beginning of this project, I stated that the aims and objectives of Retro must be able to allow Software Development teams to perform their retrospectives remotely, by being able to add items and action items, the core part of a retrospective to the application for their team to view and interact with simultaneously. Retro easily allows for this once deployed to a space or system that suits each team or company. I also stated that Retro must allow for clear communication with an intuitive user interface, which I believe I have achieved, as evidenced by the [user interface](#) of Retro. Retro's UI is clean and clear, allowing teams to easily pick up and start a retrospective while having the focus be on the retrospective itself through its minimalist design.

Similarly, Retro allows for Software Development team managers to effectively manage their teams remotely, while being able to monitor team morale levels. Retro achieves this by displaying a Sentiment Analysis for retrospectives. This includes a visual sentiment score based on the results of the analysis, as well as a recommendation message on how to keep this trend going for a team if the sentiment is positive, or how to improve the team morale levels if the sentiment is negative.

As I also outlined in the aims and objectives of Retro, it was important to have Retro be designed with S.O.L.I.D. design principles in order to allow for optimal performance and simplified debugging should the need arise. Retro was designed from the beginning with these principles in mind and I believe I have demonstrated that throughout this report.

It was also stated that security must be a top priority when designing Retro. I believe this was achieved by the built-in security features of the Spring framework, which was used when implementing Retro's backend API, as well as the testing carried out for Retro. A result of 75% coverage of all classes and 66% coverage for all methods present in the backend by the creation of [unit tests](#) for Retro's backend was achieved. Similarly, carrying out [usability testing](#) also helped to ensure users did not access or discover any components of the application they should not have access to.

Finally, the ability to scale the application was an important factor of the application. Through using the S.O.L.I.D. design principles throughout the application as well as the H2 in-memory database, I believe this was achieved. Any Software Development team or company using Retro can deploy this system on a cloud-based system or their intranet and Retro can perform as needed there. The H2 in-memory database will connect to relational databases if the team or company wish to expand or remotely store the data present within Retro.

### 3. Conclusions

In conclusion, I believe Retro is a strong project, that possesses many strengths and advantages from the way in which it was implemented. Retro's strengths include full CRUD functionality in the backend API that was designed and built from the ground up, use of robust and well-documented languages and frameworks to develop Retro, which makes it easier to continue development of new features as well as supporting existing ones. Retro also contains a responsive user interface that is designed for use on various displays which is an advantage for remote retrospectives. Retro can also be deployed and set up to suit any environment with ease, allowing teams or companies to get started using Retro quickly.

Retro also has the advantages of being a lightweight, fast application that can be used by any Software Development team or enterprise to perform retrospectives. Its minimalist and user-friendly user interface places focus on the retrospectives and allows teams to address issues that have arisen from their retrospectives in real-time without compromising on functionality is also an advantage, as well as allowing Software Development team managers to effectively gauge team morale through its Sentiment Analysis functionality.

I believe Retro's item voting system, marking items as reviewed, and search functionality present throughout the application are all advantages to using the application.

However, Retro in its current form, is not without its disadvantages and limitations. The functionality and performance of the application could always be improved, allowing Retro to become more efficient without sacrificing performance. Similarly, there are also many additions that could be made to the functionality and user interface of Retro. These might not be considered disadvantages or limitations as such, but rather things that could be added to Retro with added time and further research into how best to implement them to enhance Retro.

Designing and developing Retro has been a challenging but very rewarding experience for me, and I believe it has made me a stronger software developer. I have learned a great deal about the

process of designing a projects architecture and system from the ground up, creating and adapting to a development timeline for a project without compromising on the initially outlined functionality, how to dive into and research new technology including new languages and frameworks and producing some tangible from that research and how to be persistent and keep trying different ways of solving an issue until the desired result is achieved. I will carry the lessons I have learned from this project forward and use them throughout my career.

#### 4. Further Development or Research

With additional time and resources, there are a number of things I would like to do next with Retro to expand its capabilities. Firstly, I would like to add more ways to analyse the data present in retrospectives for Software Development team managers. The forms that this may take is something I would have to look into, but I believe there is a huge amount of potential to spot trends and draw various conclusions from the data that is created in Retro.

On the same note, I would like to improve the algorithm used to perform a Sentiment Analysis in Retro as I believe fine tuning or exploring other means to carry this out could be a valuable addition to Retro. Similarly, a recommendation message that is more specific to each retrospective would be beneficial to team managers.

I would like to also add in functionality to allow teams to export their action items to their team sprint board, such as the Rally ([Broadcom, 2021](#)) or Jira ([Atlassian, 2021](#)) sprint boards, so they do not have to manually add them to it after a retrospective. This would undoubtedly improve the User Experience (UX) ([Norman and Nielsen, 2021](#)) on Retro.

I would also like to include an authentication system within Retro. This would allow users to sign up and log in to Retro using their credentials. Specifically, I would like to implement this using a Lightweight Directory Access Protocol (LDAP) repository ([Oracle, 2021](#)) to store and access profile data for users. Using an LDAP repository, Retro would be able to easily integrate and handle retrospectives in large companies. The reason I would choose an LDAP repository is many large modern companies make use of LDAP repositories to handle authentication across their intranet and internal systems already. Integrating this with Retro would be a onetime set up and would allow companies and teams to quickly include all relevant team members in a retrospective by simply typing their names, or email addresses into the implementation LDAP repository search bar. This would be a huge benefit to Retro and an important selling point to large companies.

On the same note, I would like to include functionality to restrict the view of retrospectives to only the members of that team. This would be relatively easy to implement with the use of an LDAP repository.

I would also like to make use of a cloud based relational database for Retro and abstract Retros database system away from being an internal component of the application. This would improve overall performance, allow for improved use of the S.O.L.I.D. design principles and come with the dedicated security features that cloud database providers offer.

Finally, I would also like to further enhance the user interface of Retro in order to improve the overall UX of application as an excellent user experience is at the heart of what Retro aims to provide to users.



## 5. References

- Taplin, S., 2021. Council Post: Managing Remote Software Development Teams In 2021. [online] Forbes. Available at: <<https://www.forbes.com/sites/forbestechcouncil/2021/03/23/managing-remote-software-development-teams-in-2021/?sh=a018576e3f1e>> [Accessed 11 May 2021]. ([Back to Executive Summary](#))
- Scrum, 2021. What is a Sprint Retrospective? [online] Scrum.org. Available at: <<https://www.scrum.org/resources/what-is-a-sprint-retrospective>> [Accessed 26 April 2021]. ([Back to Executive Summary](#))
- Gupta, S., 2018. Sentiment Analysis: Concept, Analysis and Applications. [online] Medium. Available at: <<https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17>> [Accessed 11 May 2021]. ([Back to Executive Summary](#))
- Nielsen, F., 2011. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. CEUR Workshop Proceedings: 93 -98, [online] Volume 718. Available at: <<http://www2.imm.dtu.dk/pubdb/edoc/imm6006.pdf>> [Accessed 11 May 2021]. ([Back to Technology](#))
- AFINN, F., 2011. AFINN. [online] Www2.imm.dtu.dk. Available at: <<http://www2.imm.dtu.dk/pubdb/pubs/6010-full.html>> [Accessed 11 May 2021]. ([Back to Technology](#)) ([Back to Backend Implementation](#))
- Graversen, K., 2015. Super CSV – Welcome. [online] Super-csv.github.io. Available at: <<http://super-csv.github.io/super-csv>> [Accessed 13 May 2021]. ([Back to Backend Implementation](#))
- Angular, 2021. Angular. [online] Angular.io. Available at: <<https://angular.io/guide/pipes>> [Accessed 13 May 2021]. ([Back to frontend](#))
- JUnit, 2021. JUnit 5. [online] Junit.org. Available at: <<https://junit.org/junit5/>> [Accessed 5 May 2021]. ([Back to Backend Testing](#))
- Fernandez, C., 2021. Heuristic Evaluation Lecture Presentation, National College of Ireland. ([Back to Usability Testing](#))
- Broadcom, 2021. Rally Software. [online] Broadcom.com. Available at: <<https://www.broadcom.com/products/software/agile-development/rally-software>> [Accessed 14 May 2021]. ([Back to Further Development](#))
- Atlassian, 2021. Learn how to use sprints in Jira Software | Atlassian. [online] Atlassian. Available at: <<https://www.atlassian.com/agile/tutorials/sprints>> [Accessed 14 May 2021]. ([Back to Further Development](#))
- Norman, D. and Nielsen, J., 2021. The Definition of User Experience (UX). [online] Nielsen Norman Group. Available at: <<https://www.nngroup.com/articles/definition-user-experience/>> [Accessed 14 May 2021]. ([Back to Further Development](#))
- Oracle, 2021. LDAP Repositories. [online] Docs.oracle.com. Available at: <[https://docs.oracle.com/cd/E26180\\_01/Platform.94/RepositoryGuide/html/s18011daprepositories01.html](https://docs.oracle.com/cd/E26180_01/Platform.94/RepositoryGuide/html/s18011daprepositories01.html)> [Accessed 14 May 2021]. ([Back to Further Development](#))
- Retrium, 2020. Retrospectives Made Easy For Scrum & Agile Teams | Retrium. [online] Retrium.com. Available at: <<https://www.retrium.com/>> [Accessed 20 December 2020]. ([Back to similar projects](#))
- TeamRetro, 2020. Team Retro - Online Retrospective And Team Health Check Tool. [online] TeamRetro. Available at: <<https://www.teamretro.com/>> [Accessed 20 December 2020]. ([Back to similar projects](#))

## 6. Appendices

### 6.1. Project Plan

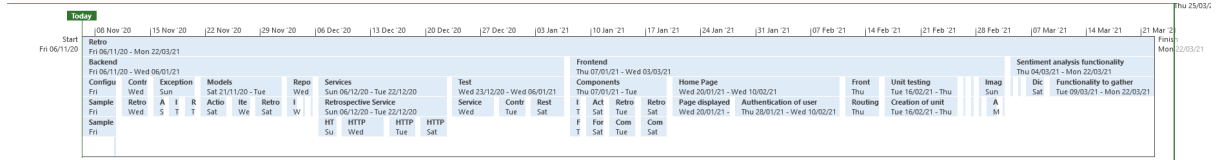


Figure 46: Gantt Chart timeline view

▲ Retro	98 days	Fri 06/11/20	Mon 22/03/21
▲ Backend	45 days	Fri 06/11/20	Wed 06/01/21
▲ Configuration	3 days	Fri 06/11/20	Tue 10/11/20
▲ Sample data	3 days	Fri 06/11/20	Tue 10/11/20
Sample data of retrospectives, their items and action items	3 days	Fri 06/11/20	Tue 10/11/20
Local profile for sample data use and display	1 day	Tue 10/11/20	Tue 10/11/20
▲ Controller	4 days	Wed 11/11/20	Sat 14/11/20
Retrospective Control	4 days	Wed 11/11/20	Sat 14/11/20
▲ Exceptions	5 days	Sun 15/11/20	Fri 20/11/20
Action Item Exception	2 days	Sun 15/11/20	Mon 16/11/20
Item Exceptions	2 days	Tue 17/11/20	Wed 18/11/20
Retrospective Exceptions	2 days	Thu 19/11/20	Fri 20/11/20
▲ Models	7 days	Sat 21/11/20	Tue 01/12/20
Action item	3 days	Sat 21/11/20	Tue 24/11/20
Item	3 days	Wed 25/11/20	Fri 27/11/20
Retrospective	3 days	Sat 28/11/20	Tue 01/12/20
▲ Repositories	3 days	Wed 02/12/20	Sat 05/12/20
ItemRepository	2 days	Wed 02/12/20	Thu 03/12/20
Action Item Repository	1 day	Fri 04/12/20	Fri 04/12/20
Retrospective Repository	1 day	Sat 05/12/20	Sat 05/12/20
▲ Services	12 days	Sun 06/12/20	Tue 22/12/20
▲ Retrospective Service	12 days	Sun 06/12/20	Tue 22/12/20
HTTP GET Method	3 days	Sun 06/12/20	Tue 08/12/20

Figure 47: Gantt Chart items for backend

HTTP POST Method	4 days	Wed 09/12/20	Mon 14/12/20
HTTP DELETE Method	4 days	Tue 15/12/20	Fri 18/12/20
HTTP UPDATE Method	3 days	Sat 19/12/20	Tue 22/12/20
<b>▸ Test</b>	<b>11 days</b>	<b>Wed 23/12/20</b>	<b>Wed 06/01/21</b>
Service Tests	4 days	Wed 23/12/20	Mon 28/12/20
Controller Tests	4 days	Tue 29/12/20	Fri 01/01/21
Rest Service Application Tests	4 days	Sat 02/01/21	Wed 06/01/21
<b>▸ Frontend</b>	<b>40 days</b>	<b>Thu 07/01/21</b>	<b>Wed 03/03/21</b>
<b>▸ Components</b>	<b>9 days</b>	<b>Thu 07/01/21</b>	<b>Tue 19/01/21</b>
<b>▸ Item Form Component</b>	<b>2 days</b>	<b>Thu 07/01/21</b>	<b>Fri 08/01/21</b>
Form template for entering items	2 days	Thu 07/01/21	Fri 08/01/21
<b>▸ Action Item Form Component</b>	<b>1 day</b>	<b>Sat 09/01/21</b>	<b>Mon 11/01/21</b>
Form template for entering action	2 days	Sat 09/01/21	Mon 11/01/21
<b>▸ Retrospectives Component</b>	<b>4 days</b>	<b>Tue 12/01/21</b>	<b>Fri 15/01/21</b>
Component for displaying retrospective data	4 days	Tue 12/01/21	Fri 15/01/21
<b>▸ Retrospectives List Component</b>	<b>2 days</b>	<b>Sat 16/01/21</b>	<b>Tue 19/01/21</b>
Component for displaying a list of all available retrospectives	3 days	Sat 16/01/21	Tue 19/01/21
<b>▸ Home Page</b>	<b>16 days</b>	<b>Wed 20/01/21</b>	<b>Wed 10/02/21</b>

Page displayed upon application launch	6 days	Wed 20/01/21	Wed 27/01/21
Authentication of user	10 days	Thu 28/01/21	Wed 10/02/21
▣ Front end routing	3 days	Thu 11/02/21	Mon 15/02/21
Routing between pages / how components interact	3 days	Thu 11/02/21	Mon 15/02/21
▣ Unit testing	8 days	Tue 16/02/21	Thu 25/02/21
Creation of unit tests within each component to make sure they function as	8 days	Tue 16/02/21	Thu 25/02/21
▣ Bootstrap Implementation	1 day	Fri 26/02/21	Fri 26/02/21
Adding bootstrap framework to project to improve both the UI and UX	1 day	Fri 26/02/21	Fri 26/02/21
▣ UI Colour Scheme	1 day	Sat 27/02/21	Sat 27/02/21
Selection of appropriate colours for the design of the	1 day	Sat 27/02/21	Sat 27/02/21
▣ Image Assets	3 days	Sun 28/02/21	Wed 03/03/21
Favicon	1 day	Sun 28/02/21	Sun 28/02/21
Application Logo	2 days	Mon 01/03/21	Tue 02/03/21
Functionality image assets e.g. back button, tick icon, etc	1 day	Wed 03/03/21	Wed 03/03/21
▣ Sentiment analysis functionality	13 days	Thu 04/03/21	Mon 22/03/21

Figure 48: Gantt Chart items for frontend 1/2

▣ Sentiment analysis functionality	13 days	Thu 04/03/21	Mon 22/03/21
Investigate framework to do this	1 day	Thu 04/03/21	Thu 04/03/21
Graphing package	1 day	Fri 05/03/21	Fri 05/03/21
Dictionary of 'good' and 'bad' words 2	2 days	Sat 06/03/21	Mon 08/03/21
Functionality to gather words from retrospectives and plot them based on dictionary against productivity	10 days	Tue 09/03/21	Mon 22/03/21

Figure 49: Gantt Chart items for frontend 2/2

## 6.2. Project Proposal

### 6.2.1. Objectives

This project aims to facilitate sprint retrospectives remotely for Software Development teams within a responsive web application that can be used on desktop and mobile devices. Retro will allow development teams that are using an agile sprint methodology to run their retrospectives from any location. Development teams will be able to add items to the retrospective that will tackle things that went well in the sprint, things that did not go well and any questions they have as a result of the sprint. They will also be

able to add action items, issues that need to be solved in the following sprint or added to a backlog of bugs to fix.

These are the key aspects of retrospectives and will allow teams to analyse and address how their sprint went, any issues the team might have and how to solve them in a subsequent sprint.

Retrospectives are traditionally done in person where teams can talk through their sprints and resolve any issues face to face. However, as the ever-changing IT industry grows, teams are not always able to meet face to face due to geographical constraints, personal issues, or in the interest of social distancing. This has a significant impact on communication levels and standards within development teams. Retro seeks to solve that problem and allow teams to carry out their retrospectives in a clear, open manner, with excellent communication levels in development teams, as though they were in person, through an intuitive and user-friendly application that is designed to fit the needs of any team.

The unique feature of Retro is an analysis tool for team managers that performs a sentiment analysis on the words commonly used in the retrospectives of teams they manage in order to assess the general mood or morale levels within their teams versus productivity levels. This could be of huge benefit to managers as they are missing face-to-face interactions and meetings to gauge such things. This would aid managers in addressing productivity issues or maintaining morale with the benefit of tangible data.

For Retro to succeed and be used in the real world, it needs to be able to scale to fit the demands of any company, large or small, while retaining the efficiency of the application at any size. Taking this into consideration, designing Retro to be able to meet those demands is essential and will be a top priority during the development of this project and will be the innovative aspect of it. This element is essential for this application to succeed. It must be able to scale to large corporations or small start-ups, this will begin with a strong emphasis on designing a strong architecture for the project and using S.O.L.I.D. (Single responsibility, Open/closed, Liskov segregation, Interface Segregation, Dependency inversion) design principles throughout development.

Similarly, when designing RETRO, the security of the application will be paramount. Development teams must be able to rest assured that their remote retrospectives, including issues they raise during them, will only be privy to those within their team and their manager. Retro will be designed with a heavy emphasis on authentication and authorisation to support this aspect as security is of huge importance to both the companies and development teams potentially using this product.

## 6.2.2. Background

### 6.2.2.1. Origin of the Idea

The origin of this idea began when I was doing my placement in third year. The company I was working in during my internship followed an agile methodology. However, I noticed that when it came time to do our sprint retrospectives there was no dedicated application to facilitate them in an organised way that allowed for clear communication. The effects of this were especially felt where team members were in other offices around the world and when everyone moved to working from home at the start of the pandemic.

Even existing retrospective applications provide an inadequate level of support for extensive communication during retrospectives as they rely on the option of teams bringing up issues the next time they were in the office. However, this is no longer an option in most development teams as the majority are working from home.

Of course, breakdown of communication is bound to happen during these unforeseen times, but I began to think that there had to be a better way to organise and facilitate these important meetings where communication levels would be the same as when we were in the office.

An application is needed that could keep communication clear in retrospectives, aid remote management of teams and give teams some semblance of normalcy during times of extended working from home, and when team members are based in other locations.

I believe that especially at the moment, a project like this is very topical due to the current pandemic and it could hugely benefit development teams as they are now working from home full time, performing their retrospectives remotely and noticing the drawbacks of their current retrospective application or lack thereof, as it likely was not developed with full time remote working in mind. Retro will be able to overcome this problem and meet the need for an application to facilitate retrospectives remotely.

#### 6.2.2.2. Similar Projects

##### [Retrium \(2020\)](#) :

Retrium is a retrospective program. It offers to enable agile teams to have effective conversations, discover new insights and generate action plans. Retrium promises engaging retrospectives that fuel continuous improvement. It boasts features with functionality such as:

- **Stop, Start, Continue:** this functionality asks ‘what activities should be started on? What activities should be stopped and what activities need to be continued?’.
- **What went well:** this functionality asks what went well and what did not during a sprint.
- **Mad, Sad, Glad:** functionality to check the emotions of a team.
- **Action Plan:** functionality that allows the integration of a plan for executing a retrospectives action items.
- **Integration with Jira:** integration action items directly into Jira workflow in conjunction with Retrium.
- **Retrospective history:** functionality to see a timeline past conversations, notes and action items that were created as a result of a retrospective.

##### [Team Retro \(2020\)](#) :

Team Retro is a retrospective application that offers functionality for both face-to-face and remote meetings. It offers a fully customisable, cross device experience, for development teams. Its offered functionality includes:

- **Easy invitation:** functionality to invite team members via email, through Slack or SSO.
- **Brainstorming:** functionality to add what things went well in a sprint or not so well. Addition of items can be anonymous, or with the users’ name.

- Group related ideas: functionality to combine related added items together, such as items to start or stop.
- Voting on items: functionality to anonymously vote on items to ensure team members can decide where the team's focus should be.
- Integration with existing workflow: functionality to add action items to existing workflow such as Jira or Trello.

### 6.2.3. Technical Approach

For the front-end, I believe Angular could be an appropriate option as there are many resources to research for development using Angular, and it would be able to fulfil the scope of requirements of the project. End-to-end and unit testing in Angular is also a strong aspect of Angular and once I carry out further investigation, this will help testing the performance of the application. To aid the look and feel of the project, implementing the Bootstrap framework alongside Angular will be a great asset to the project.

Similarly, I will need to select a suitable back-end language for the project. I believe Java will be the best option for this as I have experience developing using this language. There is also a vast amount of resources such as various frameworks for unit testing which is essential to this project's objectives and for debugging.

To aid the development of the backend of the project, I believe the Spring framework would be a great asset to the project as it removes a lot of boilerplate code and there are libraries within it that deal with the security and scalability of a project. This project will also likely require developing an API in order to function as intended. The Spring framework's libraries could also aid this aspect. I have briefly looked into this framework, and although it will require more research, I believe it will fulfil the needs of the project.

For the sentiment analysis, I believe the Tidytext library will be appropriate. This library contains three general purpose dictionaries that assign a score for positive or negative sentiment on words as well as possible emotions such as joy, anger, fear, etc that are associated with a word. This will be implemented in Retro using the words posted by team members in a retrospective in order to provide managers with a tangible sense of the team morale level and the emotions team members are experiencing during a sprint retrospective.

When considering the objectives of this application such as efficiency, security and scalability are three of the most important aspects. As a result, the architecture of the project will require detailed design and careful planning. To achieve this, I will research S.O.L.I.D. design principles and implement them to the project. These principles will encourage more maintainable, flexible, and understandable code than can achieve the objectives I have set out.

A large part of this project will centre around enabling better communication levels within development teams during retrospectives. Many existing retrospective applications were not built to facilitate the level of remote retrospectives that are taking place at the moment due to the majority of development teams working from home, and perhaps relied on issues being cleared up during the next in-person office meeting, and as a result there is an inadequate level of communication functionality present in these applications. Similarly, other applications were not built to handle the number of retrospectives taking place remotely currently taking place in companies and certainly

not with the scale of larger companies' teams in mind. Retro will be designed with a high level of communication and scaling in mind to solve this issue.

Functionality will include individuals in development teams being able to add what went well in a sprint, what did not go so well and any questions they might have. Users will also be able to add action items to retrospectives, these are the tasks that arise because of the retrospectives that need to be completed in a subsequent sprint. Users will also be able to export action items to their agile development task platform, such as Jira or Rally, as a task to be completed.

Items will also have a voting function, for other users that agree with a point raised by a team member to support or not. Similarly, when teams are going through their retrospective discussion, items will be able to be marked as reviewed and greyed out.

#### 6.2.4. Special Resources Required

For this project, I will require resources for designing the project with S.O.L.I.D. design principles. As this will be a consistent objective throughout the project, having something to refer back to easily when designing the project will be a great asset. I believe a book will be sufficient for this.

Materials dealing with scaling and security for the application the application could also be useful. However, there are many detailed online resources that will be sufficient for this such as the Baeldung website.

#### 6.2.5. Project Plan

Gantt chart project timeline and tasks.

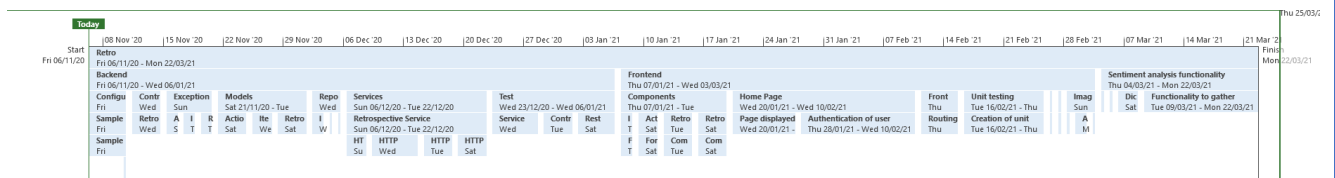


Figure 50: Project Plan - Gantt Chart timeline view



▸ Retro	98 days	Fri 06/11/20	Mon 22/03/21
▸ Backend	45 days	Fri 06/11/20	Wed 06/01/21
▸ Configuration	3 days	Fri 06/11/20	Tue 10/11/20
▸ Sample data	3 days	Fri 06/11/20	Tue 10/11/20
Sample data of retrospectives, their items and action items	3 days	Fri 06/11/20	Tue 10/11/20
Local profile for sample data use and display	1 day	Tue 10/11/20	Tue 10/11/20
▸ Controller	4 days	Wed 11/11/20	Sat 14/11/20
Retrospective Control	4 days	Wed 11/11/20	Sat 14/11/20
▸ Exceptions	5 days	Sun 15/11/20	Fri 20/11/20
Action Item Exception	2 days	Sun 15/11/20	Mon 16/11/20
Item Exceptions	2 days	Tue 17/11/20	Wed 18/11/20
Retrospective Exceptions	2 days	Thu 19/11/20	Fri 20/11/20
▸ Models	7 days	Sat 21/11/20	Tue 01/12/20
Action item	3 days	Sat 21/11/20	Tue 24/11/20
Item	3 days	Wed 25/11/20	Fri 27/11/20
Retrospective	3 days	Sat 28/11/20	Tue 01/12/20
▸ Repositories	3 days	Wed 02/12/20	Sat 05/12/20
ItemRepository	2 days	Wed 02/12/20	Thu 03/12/20
Action Item Repository	1 day	Fri 04/12/20	Fri 04/12/20
Retrospective Repository	1 day	Sat 05/12/20	Sat 05/12/20
▸ Services	12 days	Sun 06/12/20	Tue 22/12/20
▸ Retrospective Service	12 days	Sun 06/12/20	Tue 22/12/20
HTTP GET Method	3 days	Sun 06/12/20	Tue 08/12/20

Figure 51: Project Plan - Gantt Chart items 1/ 3

HTTP POST Method	4 days	Wed 09/12/20	Mon 14/12/20
HTTP DELETE Method	4 days	Tue 15/12/20	Fri 18/12/20
HTTP UPDATE Method	3 days	Sat 19/12/20	Tue 22/12/20
▸ Test	11 days	Wed 23/12/20	Wed 06/01/21
Service Tests	4 days	Wed 23/12/20	Mon 28/12/20
Controller Tests	4 days	Tue 29/12/20	Fri 01/01/21
Rest Service Application Tests	4 days	Sat 02/01/21	Wed 06/01/21
▸ Frontend	40 days	Thu 07/01/21	Wed 03/03/21
▸ Components	9 days	Thu 07/01/21	Tue 19/01/21
▸ Item Form Component	2 days	Thu 07/01/21	Fri 08/01/21
Form template for entering items	2 days	Thu 07/01/21	Fri 08/01/21
▸ Action Item Form Component	1 day	Sat 09/01/21	Mon 11/01/21
Form template for entering action	2 days	Sat 09/01/21	Mon 11/01/21
▸ Retrospectives Component	4 days	Tue 12/01/21	Fri 15/01/21
Component for displaying retrospective data	4 days	Tue 12/01/21	Fri 15/01/21
▸ Retrospectives List Component	2 days	Sat 16/01/21	Tue 19/01/21
Component for displaying a list of all available retrospectives	3 days	Sat 16/01/21	Tue 19/01/21
▸ Home Page	16 days	Wed 20/01/21	Wed 10/02/21

Figure 52: Project Plan - Gantt Chart items 2/3

Page displayed upon application launch	6 days	Wed 20/01/21	Wed 27/01/21
Authentication of user	10 days	Thu 28/01/21	Wed 10/02/21
▣ Front end routing	3 days	Thu 11/02/21	Mon 15/02/21
Routing between pages / how components interact	3 days	Thu 11/02/21	Mon 15/02/21
▣ Unit testing	8 days	Tue 16/02/21	Thu 25/02/21
Creation of unit tests within each component to make sure they function as	8 days	Tue 16/02/21	Thu 25/02/21
▣ Bootstrap Implementation	1 day	Fri 26/02/21	Fri 26/02/21
Adding bootstrap framework to project to improve both the UI and UX	1 day	Fri 26/02/21	Fri 26/02/21
▣ UI Colour Scheme	1 day	Sat 27/02/21	Sat 27/02/21
Selection of appropriate colours for the design of the	1 day	Sat 27/02/21	Sat 27/02/21
▣ Image Assets	3 days	Sun 28/02/21	Wed 03/03/21
Favicon	1 day	Sun 28/02/21	Sun 28/02/21
Application Logo	2 days	Mon 01/03/21	Tue 02/03/21
Functionality image assets e.g. back button, tick icon, etc	1 day	Wed 03/03/21	Wed 03/03/21
▣ Sentiment analysis functionality	13 days	Thu 04/03/21	Mon 22/03/21

Figure 53: Project Plan - Gantt Chart items 3/3

#### 6.2.6. Technical Details

The project's backend will be written in Java. To aid this, the Spring Framework will be added too. Utilising this framework will allow the use of its security libraries, removal of boilerplate code, and aid development of the API and scaling the application.

The frontend of the project will be written in Typescript using Angular. This will involve using the Angular CLI to create components as needed. For the user interface of the front end, implementation of the Bootstrap framework for CSS as well as using HTML will be necessary.

The sentiment analysis will be implemented using the Tidytext library. This will involve gathering words from retrospectives and analysing them in order to provide management with a sense of team morale levels versus productivity.

#### 6.2.7. Evaluation

For the evaluation of this project, I plan to incorporate JUnit tests throughout the back end to ensure there is a high level of coverage for the code in each class. This coverage will likely be in excess of 80%, with complete coverage of all methods.

I also plan on setting a 'local' Spring profile for the backend. This will involve creating a Java class and filling it with sample data in order to test that the API is outputting the right data where needed. This will be useful when demonstrating progress of the project on both the front-end and back-end, allowing for continuous testing and checking that each part of the development is performing as it should, and how it should.

Similarly, I plan to do component testing for the front -end Angular code to ensure everything is functioning as intended. I will also incorporate error pages if users navigate to the wrong endpoints. The front end will also have an emphasis on usability testing.

## 6.3. Reflective Journals

### 6.3.1. October Reflective Journal

This was an exciting and busy month as we had to come up with our idea for the final year project and make a video pitch of the idea. Initially, I was overwhelmed by the prospect of coming up with an idea for a project that carries so much weight for my degree grade and one that will be impressive enough to talk about when going on interviews.

As this year is worth 80% of my degree, I must admit that the pressure is mounting to do well and have a strong final year project as well as performing well in other assessments.

However, once I sat down to think of an idea, the pieces started coming together. I came up with an idea of making an application to facilitate sprint retrospectives remotely, and I am excited to get started on developing it. Although I do not know how I am going to do every part of the project, I have a high-level view of what I want it to do, the scope of the project and some technologies I am interested in looking into incorporating into my project.

I am eager to get started on my project proposal and get the ball rolling. I am trying not to think too far ahead and keep focused on deadlines closer to me so as not become overwhelmed with the workload.

There is a lot to plan and a lot of work to be done while also keeping on top of my other modules, but hopefully I can manage to commit enough time to everything and achieve the grades I want.

### 6.3.2. November Reflective Journal

This month was particularly full of assignments from other modules, with a huge workload attached. The result of this is that I have not gotten an awful lot of work done for my final year project. I had intended on having made a good start to it, but I am finding it hard to make time for the project when there are so many other deadlines closer that are from other modules.

However, although I have not been able to get much tangible work done for the final year project. I have been able to make considerable progress on the planning of how to get my project started and the steps I will take to get it to a decent amount of the project done before the mid-point submission. Even though this is not where I would like to be progress-wise, it has given me time to flesh out the idea for my project a bit better.

I am hoping that I will be able to get a lot more done in December and particularly over the Christmas break. This will give me the much-needed time to dedicate to the final year project. However, I realise this semester's workload is intense and with only having four modules next semester, I am hopeful that I will be able to dedicate the time it deserves next semester.

### 6.3.3. December Reflective Journal

This month I was again very busy with the workload from other modules, but I managed to take a bit of time out each week to work on my final year project. I feel I have made a decent amount of progress with the code considering the point I am at before the final deadline.

I have made efforts to ensure the backend of the project is designed as I would like it and employed the S.O.L.I.D. Java design principles to achieve this. It is nowhere near finished but I am happy with the state of it for the midpoint presentation as some of the end-product functionality is present. The front end is also functioning and features the information from the backend and some of the functionality I plan to have for it to communicate properly with the backend, however, it is not aesthetically looking how I want it for the final product.

I was getting stressed before this month that I was not getting enough done as I was spending the time, I had just researching how to go about my final year project and thinking of how to design it rather than coding yet. However, I feel that going about the project in this way has actually benefitted it as when I did finally get to work on coding this month, I knew exactly how I wanted to implement the architecture of the project and why. It was aided by the fact that I implemented it with the structure of the Java S.O.L.I.D. design principles.

I am learning not to try and tackle the whole project immediately but to really try to take it bit by bit and build upon the stable foundation I have created up to now. That is not to say I am taking it easy but there is time before the final deadline in May and I will have ample opportunity to work on it until then. I am getting there, and although there is a lot to do, I am happy with the progress I have made so far.

I am eager to take the project as it stands and build upon it until I get it to where I want it to be, as well as it delivering all of the functionality it needs.

### 6.3.4. January Reflective Journal

This month was far more productive than I thought it would be. I thought I would be working less on my project due to wanting to take a break after an intense first semester, but I was feeling very motivated towards Retro and getting some work done.

I added in a lot of new features. This was a great period of time to work on my project as I was off from college meaning I had no other assignments or work to get done. I was able to focus on my final year project.

I added in unit testing throughout the backend with a high level of code-coverage. I thought it best to add in testing now when I have the chance as I know once it gets close to the final deadline, testing will be at the bottom of my priorities list. I also added in full CRUD functionality to my back end and front-end, UI changes to the colour scheme and look and feel, archiving functionality for retrospectives, pipes for filtering retros, and pipes for item functionality like being finished with an item (it is greyed out once marked as reviewed), and filtering by votes.

I am happy with the progress I made over January as I did not think I would get much done at all, but I surprised myself with the passion and dedication I have towards this project. I really want it to be a well-made project that I will be proud of.

I still have a lot of work to do and challenges to overcome. However, I am looking forward to getting stuck into it over the next couple of months and adding more functionality and polishing the overall application into something worthwhile.

#### 6.3.5. February Reflective Journal

This month was a lot slower in terms of productivity than I would have liked. I have made some progress on my final year project, which is great, but I would have liked to get more done. I have instead been getting to grips and focusing on my other modules that I have this semester.

I am doing so in the hopes that if I lay decent groundwork on these other modules early on in the semester, I can then focus on my final year project towards the end of the semester as I will have a decent chunk of the work done already for the other modules.

I am excited and feeling very motivated to continue working away on my project for the final few weeks of the semester. Its surreal to be coming to the end of my undergraduate degree. I plan to continue working hard on my project to ensure I can produce something worth being proud of at the end of it.

#### 6.3.6. March Reflective Journal

This month was my slowest month in terms of progress due to the workload from other modules. However, it is now coming to the end of assignments due for other modules so I will have more time to focus solely on my final year project. This has been difficult as because the final year project deadline approaches, its stressful that I cannot dedicate the time I would like to it.

I have been working on a few small things for my final year project and researching how I will implement certain functionality. This will allow me to go full steam ahead in carrying out the implementation of this functionality when I get the chance. There are a number of features and bits of functionality that I still need to look into how exactly I will implement but I am confident that I will be able to achieve a strong project by the deadline.

I am coming to the final weeks before my project is due, and I am feeling motivated and eager to get the majority of features finished. This is an exciting time and I have been really enjoying working on my project over the last few months.

#### 6.3.7. April Reflective Journal

When I was starting out on my final year project, I was overwhelmed. I had no solid idea of what I wanted to do my project on, how I could complete a strong enough project when it was worth so much of my degree, or where to start. Now that it is coming to the end of the project, I feel like I have learned a huge amount. From a technical point of view, I feel like a stronger developer than I was at the start of fourth year. Even from a planning and design point of view for projects I feel like my skills have grown, as well as from a programming point of view.

Similarly, I have gained a better understanding of organisation and the development timeline for a project and the work that goes into keeping to deadlines to achieve a project's goals. I also feel like my soft skills have improved greatly over the last few months, from contacting my supervisor and lecturers to ask questions and pick their brains about how to go about tackling parts of my project.

Along the way I have had moments of panic where I was thinking “How am I ever going to do this?” or “I don’t know where to even begin” but eventually I started to remind myself of all the previous parts of the project where I was not sure where to start and how I had successfully gotten past them. I started to think “I don’t know how to do this... yet” and got to work on tackling the task in front of me, which is an important lesson I will try to remember.

The entire experience of doing a final year project, while seemingly insurmountable at first, has been invaluable in teaching me many lessons, like those stated above, that I will take into my career with me.

Although the process has been tough, I have really enjoyed creating this project from beginning to end and it has been really rewarding in terms of learning. I am very proud of the work I have put into my project and incredibly excited to get to work on finishing it up over the next few weeks and submitting the final implementation.

#### 6.4. Informed Consent Form :

##### 6.4.1. Participant One:

##### User Testing Informed Consent Form

**Study administrator is: Ben Carroll**

**Participant is: Grace McKeown**

**Participant number: 1**

This is a study about Retro, a web application that allows Software Development teams to be able to facilitate their agile sprint (short, time-bound period where teams complete a defined amount of work) retrospectives remotely intended for people are software developers. Our goal is to make the web app user interface appealing, intuitive and user friendly. Your participation will help us achieve this goal.

In this session you will be working with working prototype. We will ask you to perform tasks a typical user might do. A member of the design team will sit in the same room, quietly observing and taking notes. A facilitator will sit near you and help you if you are stuck or have questions.

All information collected in the session belongs to [the college / the company] and will be used for internal purposes. We will videotape and audiotape the session. We may publish our results from this and other sessions in our reports, but all such reports will be confidential and will not include your name.

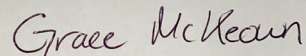
This is a test of the software. We are not testing you. We want to find out what aspects are confusing, so we can make it better. You may take breaks as needed and stop your participation in the study at any time.

##### **Statement of Informed Consent**

I have read the description of the study and of my rights as a participant. I voluntarily agree to participate in the study.

**Print Name: Grace McKeown**

Signature:



Date: 23/04/2021

[Back to Usability Testing.](#)

## 6.5. User Testing Results:

### 6.5.1. Five Second Test:

The user was presented with the home page of Retro for the five second test as seen in [Figure 55](#). A recording for the user testing with [participant one](#) can be found here: [Usability testing recording](#). The participant was asked the following:

**Question:** *What do you remember from the home page?*

The search bar and that this was the home page of the whole website.

**Question:** *What seemed important?*

The search bar seemed important and green button to create a retrospective too.

### 6.5.2. Trunk Test:

A recording for the user testing with [participant one](#) can be found here: [Usability testing recording](#). The participant was asked the following:

**Question 1:** *What website is this?*

**Answer:** It is the retrospective page.

**Question 2:** *What is the page name?*

**Answer:** The page name is the 'first retrospective'

**Question 3:** *What are the major section and important parts of the page?*

**Answer:** The sections for good things, bad things, questions, and action items.

**Question 4:** *How did you determine the important parts of the page?*

**Answer:** The headings and icons say what each important part is.

**Question 5:** *What functionality is present on the page? How can you interact with the page?*

**Answer:** You can type into each section given, icons for deleting and editing, and you vote using the arrows.

**Question 6:** *Where are you on the website? What page?*

**Answer:** The first retrospective page because it says it up the top of the page.

### 6.5.3. Think Aloud Test:

A recording for the user testing with [participant one](#) can be found here: [Usability testing recording](#). For the think aloud test, the user was asked to:

1. Create a retrospective.
2. Enter it.



3. Create an item of each type.
4. Return home.
5. Archive the retrospective.

The user completed each part of the test with ease. The participant quickly went through each part of the test. The only minor issue is they marked each item type as reviewed, i.e., greyed it out when they created it.

#### 6.5.4. Tree Test:

When conducting the tree testing, [participant one](#) was presented with the main groups and subgroups of topics and details on Retro in a list view and asked to find various locations on the website. The test was carried out as moderated tree testing.

**Task 1:** *Find the page for archived retrospectives.*

The participant was able to find the archived page with ease they immediately clicked the archive button on the navbar.

**Task 2:** *Find a specific the managers area page.*

The participant was able to find the requested locations with ease, using the navbar to redirect them to the managers area.

**Task 3:** *Find the action items of a retrospective that had been archived.*

The participant was able to complete the task but first searched for the archived retrospectives where the in-progress retrospectives are. It took them time to understand why the archived retrospectives were not grouped in with the ongoing ones.

#### 6.5.5. Click Test:

When carrying out the click test with [participant one](#), four questions were asked.

**Question 1:** *Where would you expect to click to find the information about archived retrospectives?*

The participant was able to click the 'archived' button quickly upon being shown the user interface of the home page shown in [Figure 55](#).

**Question 2:** *Where would you click to navigate to back to the home page from a retrospective page?*

The participant was able to find the back button shown on the retrospective user interface shown in [Figure 56](#), but it took them some time to do so. This points to the button perhaps being too small and not obvious enough, especially when displayed next to the retrospective title.

**Question 3:** *Where would you click to progress to submit a retrospective item?*

This question was quickly answered by the participant. The participant made it clear that the green coloured button shown beside the text field for each item type shown in [Figure 55](#) made it simple to understand that that was how they should post an item.

**Question 4:** *What button would you click to archive a retrospective?*

The fourth question proved to be the most difficult. The participant was shown the home page of Retro shown in [Figure 55](#) but clicked into the retrospective page shown on [Figure 56](#) to try and find the archive button within. After that, the user clicked the archive button on the navbar and tried to find the button there. Eventually, the participant returned to the home page and reasoned that the blue button with a check was the archive button as it would mark the retrospective as reviewed. This led to some discussion with the participant on what the button icon should be changed to and an icon similar to the one beside the archive heading on the navbar would be suitable.

[Back to Usability Testing.](#)

## 6.6. NCI Ethics Application Form

### National College of Ireland

#### Human Participants Ethical Review Application Form

All parts of the below form must be completed. However in certain cases where sections are not relevant to the proposed study, clearly mark NA in the box provided.

#### Part A: Title of Project and Contact Information

##### Name

Ben Carroll

##### Student Number (if applicable)

X17501726

##### Email

X17501726@student.ncirl.ie

##### Status:

- Undergraduate   
Postgraduate   
Staff

##### Supervisor (if applicable)

Lisa Murphy

##### Title of Research Project

Retro

##### Category into which the proposed research falls (see guidelines)

Research Category A

Research Category B

Research Category C

##### Have you read the NCI Ethical Guidelines for Research with Human Participants?

- Yes   
No

Please indicate any other ethical guidelines or codes of conduct you have consulted

Has this research been submitted to any other research ethics committee?

Yes

No

If yes please provide details, and the outcomes of this process, if applicable:

Is this research supported by any form of research funding?

Yes

No

If yes please provide details, and indicate whether any restrictions exist on the freedom of the researcher to publish the results:

## Part B: Research Proposal

Briefly outline the following information (not more than 200 words in any section).

**Proposed starting date and duration of project**

September 2020 – May 2021

**The rationale for the project**

Retro is a web application that allows Software Development teams to be able to facilitate their agile sprint retrospectives remotely and enables managers of Software Development teams to effectively manage remotely through a Sentiment Analysis (mining of text that finds and extracts information that helps a person understand the sentiment behind the data) that is performed on a retrospective, allowing managers to gauge team morale, with the data to back the analysis up. It is therefore important to discover the level of usability of the project's user interface.

**The research aims and objectives**

To research and gauge the level of usability present within the frontend of Retro.

**The research design**

To conduct a series of usability tests including a five second test, trunk test, think aloud test, tree test, and a click test. A System Usability Scale survey will also be filled out by the participant and included as a debrief.

### The research sample and sample size

Please indicate the sample size and your justification of this sample size. Describe the age range of participants, and whether they belong to medical groups (those currently receiving medical treatment, those not in remission from previous medical treatment, those recruited because of a previous medical condition, healthy controls recruited for a medical study) or clinical groups (those undergoing non-medical treatment such as counselling, psychoanalysis, in treatment centres, rehabilitation centres, or similar, or those with a DSM disorder diagnosis).

One participant should be sufficient as this will indicate problem areas and strengths present on the user interface. Age range of the participant will be 18-50 and will not belong to medical groups.

If the study involves a MEDICAL or CLINICAL group, the following details are required:

- a) Do you have approval from a hospital/medical/specialist ethics committee?  
If YES, please append the letter of approval. Also required is a letter from a clinically responsible authority at the host institution, supporting the study, detailing the support mechanisms in place for individuals who may become distressed as a result of participating in the study, and the potential risk to participants.  
If NO, please detail why this approval cannot or has not been sought.
- b) Does the study impact on participant's medical condition, wellbeing, or health?  
If YES, please append a letter of approval from a specialist ethics committee.  
If NO, please give a detailed explanation about why you do not expect there to be an impact on medical condition, wellbeing, or health.

The nature of any proposed pilot study. Pilot studies are usually required if a) a new intervention is being used, b) a new questionnaire, scale or item is being used, or c) established interventions or questionnaires, scales or items are being used on a new population. If no such study is planned, explain why it is not necessary.

No pilot studies are being used.

The methods of data analysis. Give details here of the analytic process (e.g. the statistical procedures planned if quantitative, and the approach taken if qualitative. It is not sufficient to name the software to be used).

A five second test, trunk test, think aloud test, tree test, and a click test will be used. These tests all include a question and answer based approach that can be analysed to improve the user interface of Retro.

### Study Procedure

Please give as detailed an account as possible of a participant's likely experience in engaging with the study, from point of first learning about the study, to study completion. State how long project participation is likely to take, and whether participants will be offered breaks. Please attach all questionnaires, interview schedules, scales, surveys, and demographic questions, etc. in the Appendix.

During the testing, the user will just be asked a series of questions about what they see on the user interface and what they think about it. Other tests will ask the participant to describe their actions on the user interface while performing them.

## Part C: Ethical Risk

Please identify any ethical issues or risks of harm or distress which may arise during the proposed research, and how you will address this risk. Here you need to consider the potential for physical risk, social risk (i.e. loss of social status, privacy, or reputation), outside of that expected in everyday life, and whether the participant is likely to feel distress as a result of taking part in the study. Debriefing sheets must be included in the appendix if required. These should detail the participant's right to withdraw from the study, the statutory limits upon confidentiality, and the obligations of the researcher in relation to Freedom of Information legislation. Debriefing sheets should also include details of helplines and avenues for receiving support in the event that participants become distressed as a result of their involvement in this study.

N/a

Do the participants belong to any of the following vulnerable groups?  
(Please tick all those involved).

- Children;
- The older old (85+)
- People with an intellectual or learning disability
- Individuals or groups receiving help through the voluntary sector
- Those in a subordinate position to the researchers such as employees
- Other groups who might not understand the research and consent process
- Other vulnerable groups

How will the research participants in this study be selected, approached and recruited?  
From where will participants be recruited? If recruiting via an institution or organisation other than NCI please attach a letter of agreement from the host institution agreeing to host the study and circulate recruitment advertisements/email etc.

A computer science student or software developer will be chosen as the participant as they are the target end user and will provide the most beneficial feedback from testing.

What inclusion or exclusion criteria will be used?

Participant must have knowledge of Agile methodology.

How will participants be informed of the nature of the study and participation?

By the Informed Consent form that will be signed by the participant.

Does the study involve deception or the withholding of information? If so, provide justification for this decision.

No.

What procedures will be used to document the participants' consent to participate?

An Informed Consent form.

Can study participants withdraw at any time without penalty? If so, how will this be communicated to participants?

Yes, this will be communicated through the Informed Consent form.

**If vulnerable groups are participating, what special arrangements will be made to deal with issues of informed consent/assent?**

*Please include copies of any information letters , debriefing sheets, and consent forms with the application.*

#### Part D: Confidentiality and Data Protection

**Please indicate the form in which the data will be collected.**

Identified       Potentially Identifiable       De-Identified

**What arrangements are in place to ensure that the identity of participants is protected?**

Identifiable information about the participant will be destroyed after 3 months as per GDPR guidelines.

**Will any information about illegal behaviours be collected as part of the research process? If so, detail your consideration of how this information will be treated.**

No

**Please indicate any recording devices being used to collect data (e.g. audio/video).**

Audio and video will be captured.

**Please describe the procedures for securing specific permission for the use of these recording devices in advance.**

The Informed Consent form.

**Please indicate the form in which the data will be stored.**

Identified       Potentially Identifiable       De-Identified

**Who will have responsibility for the data generated by the research?**

Ben Carroll (project researcher)

Is there a possibility that the data will be archived for secondary data analysis? If so, has this been included in the informed consent process? Also include information on how and where the data will be stored for secondary analytic purposes.

No

If not to be stored for secondary data analysis, will the data be stored for 5 years and then destroyed, in accordance with NCI policy?

Yes       No

**Dissemination and Reporting**

Please describe how the participants will be informed of dissemination and reporting (e.g. submission for examination, reporting, publications, presentations)?

The participant will be asked to provide contact information for such a case.

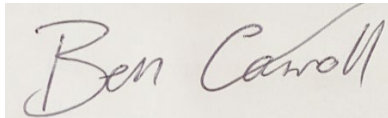
If any dissemination entails the use of audio, video and/or photographic records (including direct quotes), please describe how participants will be informed of this in advance.

The participant will be asked to provide contact information for such a case.

### Part E: Signed Declaration

I confirm that I have read the NCI Ethical Guidelines for Research with Human Participants, and agree to abide by them in conducting this research. I also confirm that the information provided on this form is correct (Electronic signature is acceptable).

Signature of Applicant \_\_\_\_\_



Date 13/05/21

Signature of Supervisor (where appropriate):

Date 

Any other information the committee should be aware of?

[Back to Usability Testing.](#)

## 6.7. System Usability Scale Exit Survey:

PARTICIPANT NAME: **Grace McKeown**

DATE: **2/5/21**

### System Usability Scale

For each of the following statements, please mark one box that best describes your reactions to Retro today.

	Strongly disagree				Strongly agree
1. I think that I would like to use Retro frequently.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. I found Retro unnecessarily complex.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. I thought Retro was easy to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
4. I think that I would need the support of a technical person to be able to use Retro.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. I found the various functions in Retro were well integrated.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6. I thought there was too much inconsistency in Retro.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7. I would imagine that most people would learn to use Retro very quickly.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
8. I found Retro very cumbersome (awkward) to use.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9. I felt very confident using Retro.	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10. I needed to learn a lot of things before I could get going with Retro.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Comments (optional):

Figure 54: System Usability Scale Exit Survey

[Back to Usability Testing.](#)

## 6.8. User Testing: User Interface

As the usability testing was carried out before the final UI of Retro was completed, screenshots of the UI as it appeared during the usability testing are found below.



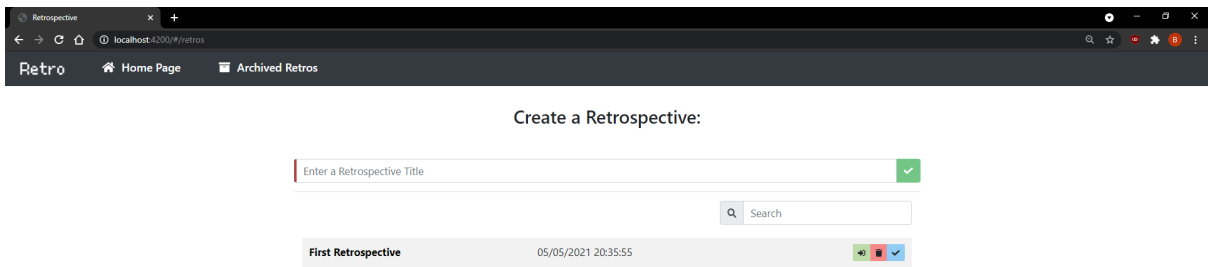


Figure 55: Usability testing Retro homepage

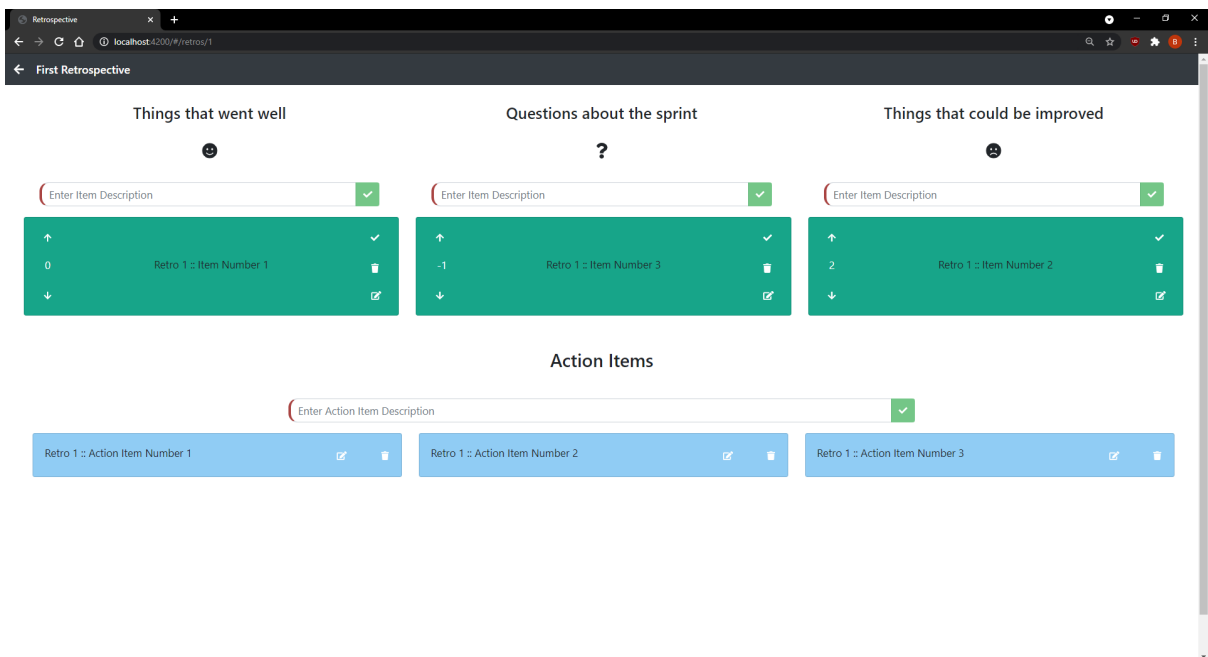


Figure 56: Usability testing individual retrospective page.

[Back to User Testing Results.](#)