**Sprint Report**

Contents

[1 Document Ownership 1](#_Toc146121850)

[2 Class Diagram 2](#_Toc146121851)

[3 Data Persistence 2](#_Toc146121852)

[4 System Tests 2](#_Toc146121853)

[5 Retrospective 2](#_Toc146121854)

[6 Mid Sprint Demo 3](#_Toc146121855)

[7 Video Demo 3](#_Toc146121856)

# Document Ownership

This document is contained in your GitHub repository in a folder named *docs*. At the end of the Sprint, Sections 2 and 3 will contain information you supply.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Team | 2 | | Team Member Names |  |
|  | |  |  | | --- | --- | | 1. | Himanshu Patel | | 2. | Grayson Wall | | |  |  | | --- | --- | | 3. | Jeremy Scott | | 4. | Ryan Scandrol | | 5. | John James | |

Video Link(s) (See [Section 7](#_Video_Demo)):

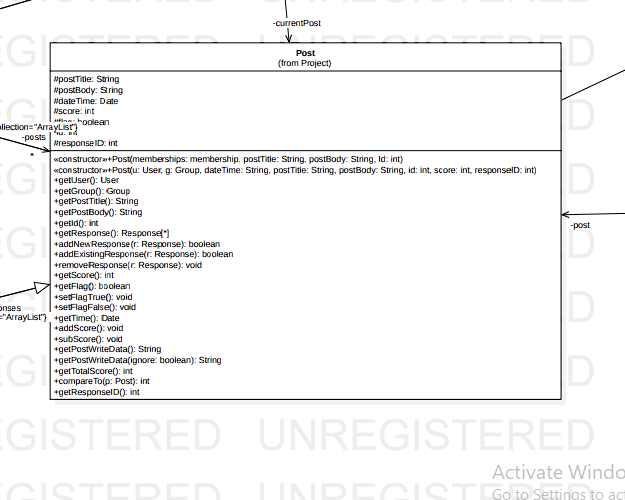
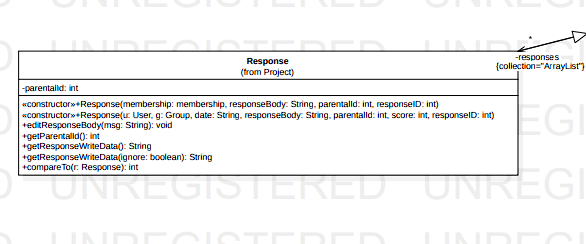
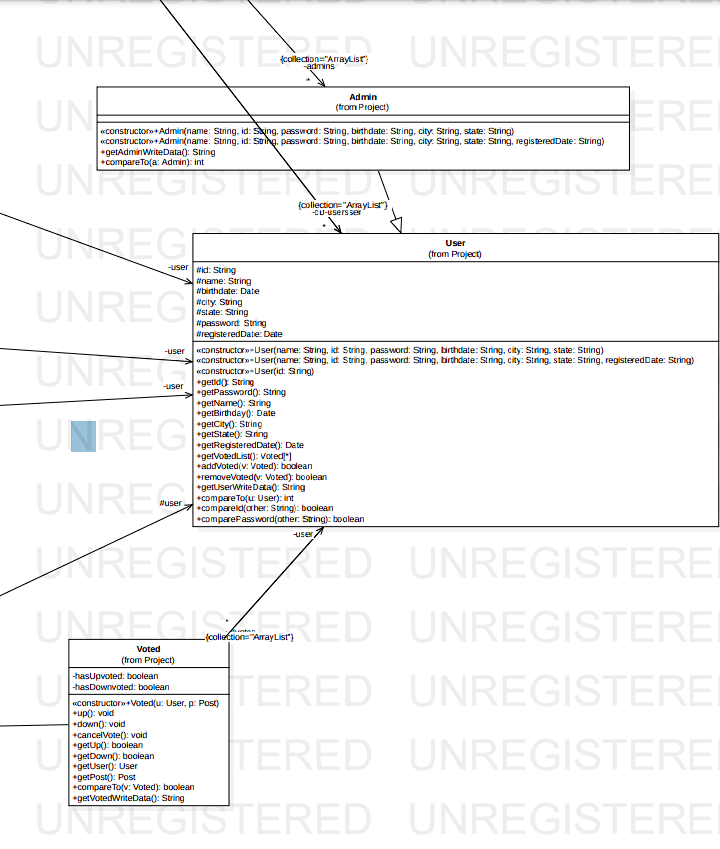
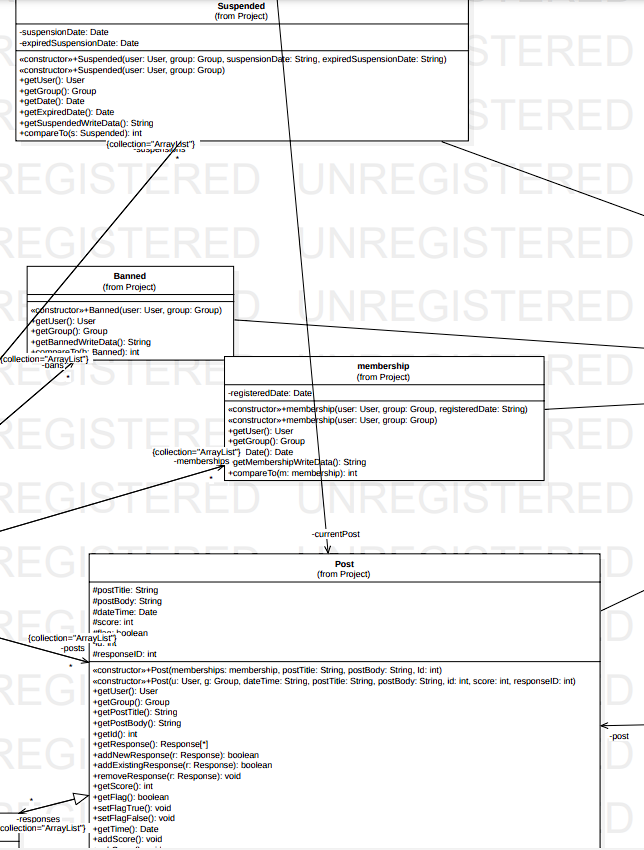
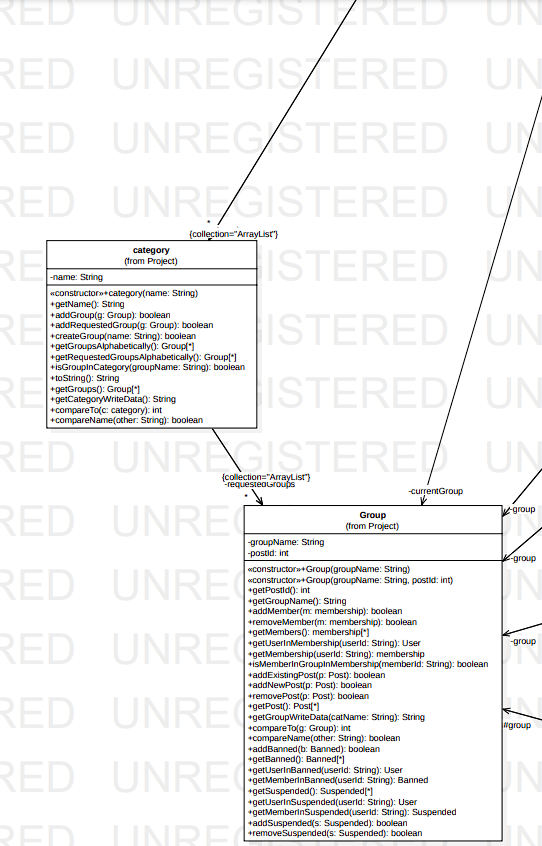
|  |
| --- |
| <https://www.youtube.com/watch?v=5bzKSXJydCw> <https://youtu.be/l66cVQLqMDQ> |

# Class Diagram

|  |
| --- |
| **Deliverable**  At the conclusion of this sprint, you should make at least one class diagram using StarUML. You can make multiple diagrams at different levels of granularity, or just break it up. **The diagram should be included in this document along with several paragraphs to explain your design.** The diagram(s) must be readable. Also, include the diagrams saved as image files (jpg or png) in your *docs* folder on GitHub. |







For the class system design, we planned to take full advantage of the MVC, Model View Controller, Design. In the example of our system, the SystemManager class is the Controller. It controls nearly the entirety of the software. The functions there utilize many of the functions set up in the other classes to perform large actions by summing together each one’s parts. View is represented, of course, by the GUI. The Model is represented by the rest of the classes shown in the diagram. The Manager holds a one-to-many relationship with the Category, User, and Admin classes. Categories hold a one-to-many relationship to Groups, with an ArrayList of the Groups they hold. Groups hold a one-to-many relationship with Banned, Suspended, membership, and Post. Admin is-a User. User has a one to many relationship with Voted, and has a one-to-one relationship with Suspended, Banned, membership. Voted has a one-to-one relationship with Post. Post has a one-to-many relationship with Response, and finally, SystemManager has a one-to-one relationship with Post in the form of a selected “currentPost.” Suspended, Banned, membership work as a “marker” which denotes a Group and User and helps to track whether a user is a member, suspended, or banned from the group. The SystemManager utilizes and manages the connections between its one-to-many relationships to allow the system to fully work together.

# Data Persistence

|  |
| --- |
| **Deliverable**  At the conclusion of this sprint, you should write a brief narrative explaining the format of data file(s) used for data persistence. You should show a brief example of each file. |

For our system, we decided to use text files for simplicity and accessibility as not all of our group members were familiar with binary files and Json. We separated each type of object used in the system into different text files, using the file names to determine which objects were currently being read. Because of this, we had to ensure that the objects were read in a specific order: 1. Admins, Users, and Categories (the order for these three doesn’t matter), 2. Groups, 3. Memberships, Banned, Suspensions (the order for these three doesn’t matter) 4. Posts, 5. Responses, 6. Voted. Each individual object’s data was separated into individual sets, with each piece of data placed on its own line in the file. Each data set starts with data starter, @START, to tell the system that a data set will be placed in the following lines. The starter is followed immediately by an object identifier, such as @ADMIN, which tells the system what type of object is about to be read in order for the system to know what type of data is expected. Next is all the individual pieces of data that the system needs for each object. The data set is followed by an ender, @END, to tell it where the end of the current data set is so it can properly move on to the next data set. Once all the data has been read, the system will check to make sure all the required data has been read before add any objects in Examples: for each objects data set:

@START

@ADMIN

@NAME=Jeremy

@BIRTHDATE=04/27/1992

@CITY=New York City

@STATE=New York

@USERNAME=unth!nk@b1e

@PASSWORD=password#3

@REGISTERED\_DATE=11/11/2011

@END

@START

@USER

@NAME=Dan

@BIRTHDATE=02/21/2000

@CITY=Chicken

@STATE=Alaska

@USERNAME=theWiz

@PASSWORD=WartH@g77

@REGISTERED\_DATE=09/16/2016

@END

@START

@CATEGORY

@NAME=Sports

@END

@START

@GROUP

@NAME=Football

@CATEGORY=Sports

@POSTID=27

@END

@START

@MEMBERSHIP

@USER=jackster3

@GROUP=Football

@REGISTEREDDATE=10/12/2013

@END

@START

@SUSPENDED

@USER=susTest

@GROUP=Memes

@SUSPENDEDDATE=12/03/2023 6:39 PM

@EXPIREDDATE=12/09/2023 1:37 AM

@END

@START

@BANNED

@USER=susTest

@GROUP=Photography

@END

@START

@POST

@USERNAME=jackster3

@GNAME=Football

@DATETIME=10/10/2008 4:27 AM

@TITLE=P1

@BODYSTART

P1 Body

@BODYEND

@PSTID=1

@FLAG <- optional, only included if post was flagged

@SCORE=718

@RESPONSEID=2

@END

@START

@RESPONSE

@USERNAME=jackster3

@GNAME=Football

@DATETIME=10/10/2009 4:27 PM

@BODYSTART

R1 Body

@BODYEND

@PARENTALID=1

@FLAG <- optional, only included if response was flagged

@SCORE=23

@RESPONSEID=0

@END

@START

@VOTED

@USER=test

@GROUP=Football

@POSTID=2

@RESPONSEID=0 <- only included if the vote belongs to a response, not a post

@DOWNVOTE

@END

# System Tests

|  |
| --- |
| **Deliverable**  At the conclusion of this sprint, supply the number of System Tests for each US in the shaded cells below. |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Num |  | Num |  | Num |  | Num |  | Num |
| US 1 | 2 | US 10 | 4 | US 21 | 3 | US 28 | 3 | US 40 | 2 |
| US 2 | 3 | US 11 | 1 | US 22 | 1 | US 29 | 1 | US 41 | 1 |
| US 3 | 1 | US 12 | 1 | US 23 | 1 | US 30 | 2 | US 42 | 1 |
| US 4 | 2 | US 13 | 1 | US 24 | 1 | US 31 | 2 |  |  |
| US 5 | 1 | US 14 | 4 | US 25 | 1 | US 32 | 1 |  |  |
| US 6 | 1 | US 15 | 1 | US 26 | 2 | US 33 | 1 |  |  |
| US 7 | 1 | US 16 | 1 | US 27 | 1 | US 34 | 2 |  |  |
| US 8 | 1 | US 17 | 1 |  |  | US 35 | 1 |  |  |
| US 9 | 1 | US 18 | 1 |  |  | US 36 | 1 |  |  |
|  |  | US 19 | 1 |  |  | US 37 | 1 |  |  |
|  |  | US 20 | 1 |  |  | US 38 | 1 |  |  |
|  |  |  |  |  |  | US 39 | 1 |  |  |
| **Total** | 13 |  | 17 |  | 10 |  | 17 |  | 4 |

Grand Total: 61

# Retrospective

|  |
| --- |
| **Deliverable**  Towards the end of the sprint, read this short page about what a software retrospective is and why it is important and then answer the questions below. **The answers should be included in this document**.  <https://searchsoftwarequality.techtarget.com/definition/Agile-retrospective> |

Meet as a group and discuss the following questions and provide a group written response below. Write as much as is appropriate.

1. What worked well for us?

|  |
| --- |
| **Answer**  Coding in our own times and meeting up together to talk about the code we wrote and the next objectives before the next meeting. Setting clear and concise goals for ourselves without much chance for obscurity or confusion. |

1. What did not work well for us?

|  |
| --- |
| **Answer**  Meeting up together and writing code, we just got too distracted and were unorganized. Also just communication through discord, wasn’t as successful sense we didn’t know if someone has read it or not. |

1. What actions can we take to improve our process going forward?

|  |
| --- |
| **Answer**  Using the project board more, and meeting up more often. Sometimes classes got in the way of us meeting up and hindered our project. Also paying more attention to the project board, in the case of it was seen but not used as much. |

# Mid Sprint Demo

|  |
| --- |
| **Deliverable**  Your group will meet with me at the allotted time to show me your progress. You have 15 minutes, but it might not take that long.   * Agenda:  1. (3-5 minutes) Explain your design. You should use class diagrams sized so that the relevant portions fill as much of the screen as possible. As part of this, explain how your design implements MVC. 2. (1-2 minutes) Demo your code illustrating several User Stories 3. (2-3 minutes) Open GitHub and I will ask questions. 4. (2-3 minutes) Open your User Stories spreadsheet and I will ask questions. |

# Video Demo

|  |
| --- |
| **Deliverable**  When your project is complete, create a video that demo’s your User Stories and provide the link in [Section 1](#_Document_Ownership). Requirements:   * You can make a single video, or, if needed, 2 or 3. * The total length should be whatever is needed to accomplish the agenda below. * Preferably, post your video(s) on YouTube. * Agenda:  1. (3-5 minutes) Explain your design. You should use class diagrams sized so that the relevant portions fill as much of the screen as possible. As part of this, explain how your design implements MVC. 2. (1-2 minutes) Choose one User Story and step through the code as if it were being executed. You’ll start by showing the code where the appropriate event handler responds to the user. Next, to whatever it calls, etc, explaining as you go. 3. (1 minute) GitHub: 4. Display your Project Board for 10 seconds. 5. Display the Issues for 5 seconds, then display the “Closed” issues 6. For each User Story: 7. State the number of the User Story, and then state the User Story itself. 8. Demonstrate it with your software. If you have multiple system tests, you can demo all of them, or just the main (success) one. You can decide on the basis of time that you have. |