Sprint Report

Contents

1 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 7 Team Member Names

1. Skylar Korn
2. Khalid Murtaza
3. Miguel Damien
4. Jacob Carlstrom
5.

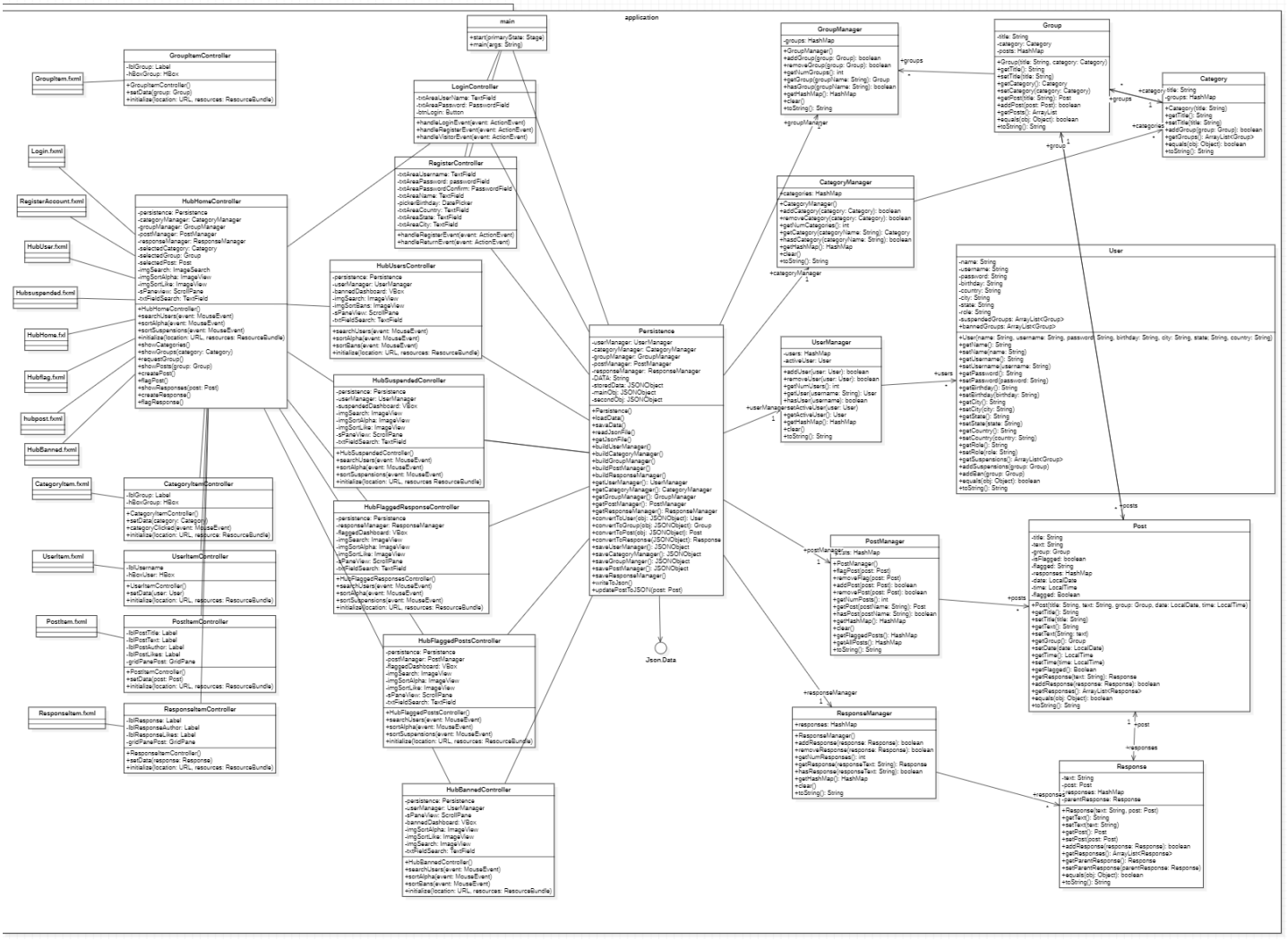
Video Link(s) (See Section 7):

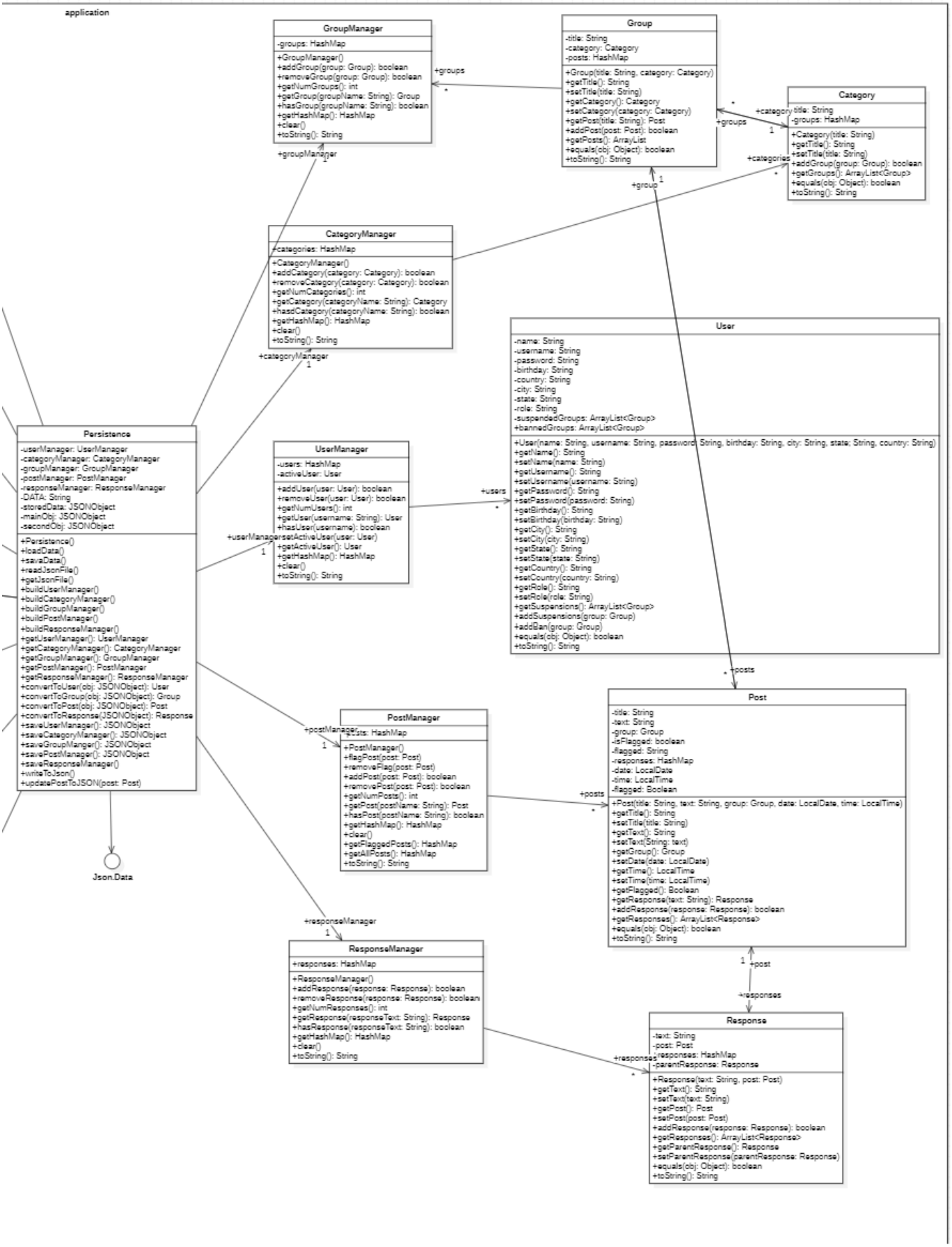
video 1 https://youtu.be/FZmK8CuExIY
video 2 https://youtu.be/MnLpwewy2Ro
video 3 https://youtu.be/VEt_srgETh4
video 4 https://youtu.be/LFqd4gfAp4c

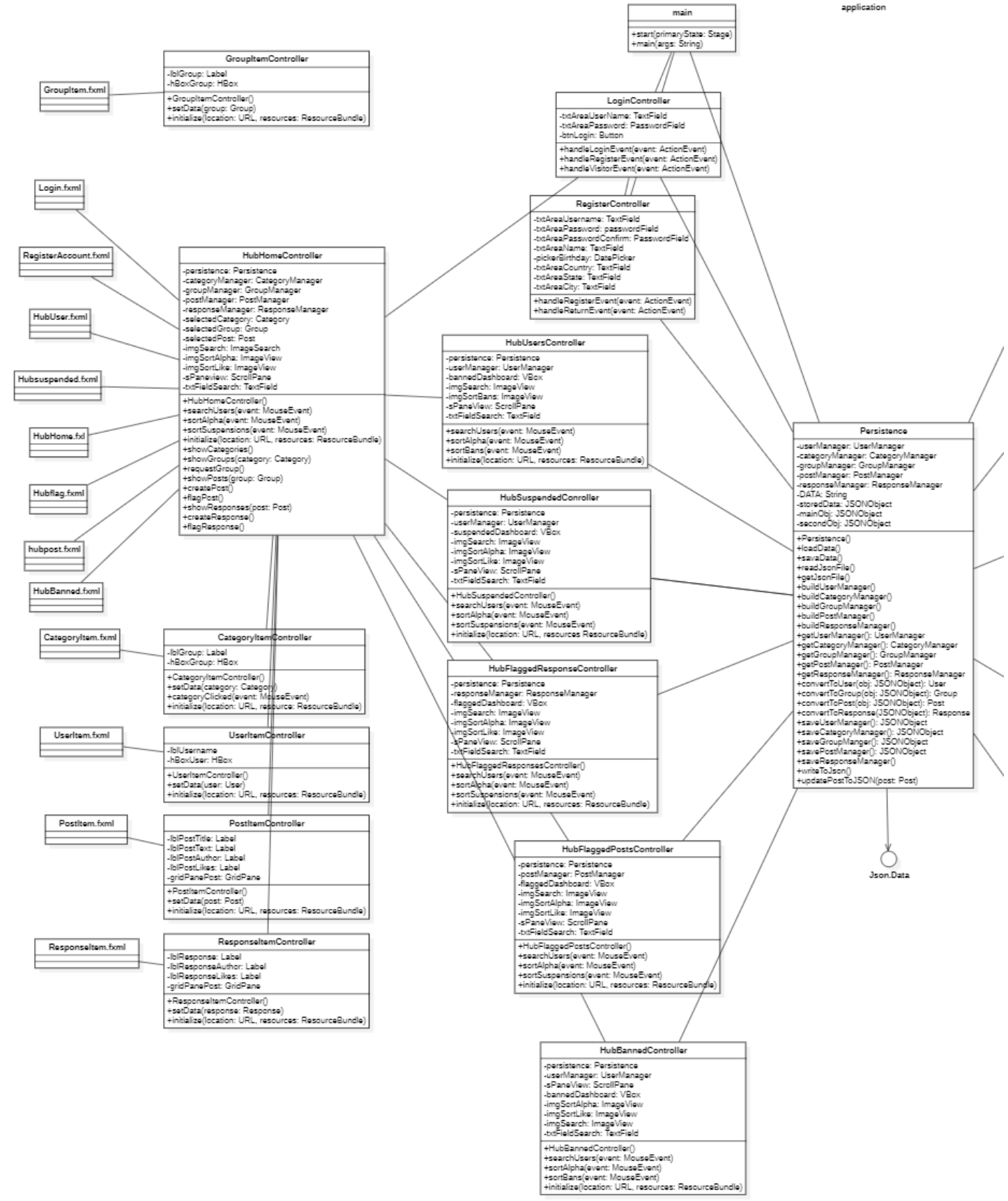
2 Class Diagram

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 <u>in this document</u> 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.




application
GroupManager
-groups: HashMap
+GroupManager()
+addGroup(group: Group): boolean
+removeGroup(group: Group): boolean
+getNumGroups(): int
+getGroup(groupName: String): Group
+hasGroup(groupName: String): boolean
+getHashMap(): HashMap
+clear()
+toString(): String
Group
-title: String
-category: Category
-posts: HashMap
+Group(title: String, category: Category)
+getTitle(): String
+setTitle(title: String)
+getCategory(): Category
+setCategory(category: Category)
+getPost(title: String): Post
+addPost(post: Post): boolean
+getPosts(): ArrayList
+equals(obj: Object): boolean
+toString(): String
Category
-title: String
-groups: HashMap
+Category(title: String)
+getTitle(): String
+setTitle(title: String)
+addGroup(group: Group): boolean
+getGroups(): ArrayList<Group>
+equals(obj: Object): boolean
+toString(): String
+groups
+groups
+category
+categories
+groupManager
+group
CategoryManager
-categories: HashMap
+CategoryManager()
+addCategory(category: Category): boolean
+removeCategory(category: Category): boolean
+getNumCategories(): int
+getCategory(categoryName: String): Category
+hasCategory(categoryName: String): boolean
+getHashMap(): HashMap
+clear()
+toString(): String
+categoryManager
User
-name: String
-username: String
-password: String
-birthday: String
-country: String
-city: String
-state: String
-role: String
-suspendedGroups: ArrayList<Group>
+bannedGroups: ArrayList<Group>
+User(name: String, username: String, password: String, birthday: String, city: String, state: String, country: String)
+getName(): String
+setName(name: String)
+getUsername(): String
+setUsername(username: String)
+getPassword(): String
+setPassword(password: String)
+getBirthday(): String
+setBirthday(birthday: String)
+getCity(): String
+setCity(city: String)
+getState(): String
+setState(state: String)
+getCountry(): String
+setCountry(country: String)
+getRole(): String
+setRole(role: String)
+getSuspensions(): ArrayList<Group>
+addSuspensions(group: Group)
+addBan(group: Group)
+equals(obj: Object): boolean
+toString(): String
Persistence
-userManager: UserManager
-categoryManager: CategoryManager
-groupManager: GroupManager
-postManager: PostManager
-responseManager: ResponseManager
-DATA: String
-storedData: JSONObject
-mainObj: JSONObject
-secondObj: JSONObject
+Persistance()
+loadData()
+saveData()
+readJsonFile()
+getJsonFile()
+buildUserManager()
+buildCategoryManager()
+buildGroupManager()
+buildPostManager()
+buildResponseManager()
+getUserManager(): UserManager
+getCategoryManager(): CategoryManager
+getGroupManager(): GroupManager
+getPostManager(): PostManager
+getResponseManager(): ResponseManager
+convertToUser(obj: JSONObject): User
+convertToGroup(obj: JSONObject): Group
+convertToPost(obj: JSONObject): Post
+convertToResponse(JSONObject): Response
+saveUserManager(): JSONObject
+saveCategoryManager(): JSONObject
+saveGroupManger(): JSONObject
+savePostManager(): JSONObject
+saveResponseManager()
+writeToJson()
+updatePostToJSON(post: Post)
Json.Data
UserManager
-users: HashMap
-activeUser: User
+addUser(user: User): boolean
+removeUser(user: User): boolean
+getNumUsers(): int
+getUser(username: String): User
+hasUser(username): boolean
+setActiveUser(user: User)
+getActiveUser(): User
+getHashMap(): HashMap
+clear()
+toString(): String
+users
+userManager
PostManager
-posts: HashMap
+PostManager()
+flagPost(post: Post)
+removeFlag(post: Post)
+addPost(post: Post): boolean
+removePost(post: Post): boolean
+getNumPosts(): int
+getPost(postName: String): Post
+hasPost(postName: String): boolean
+getHashMap(): HashMap
+clear()
+getFlaggedPosts(): HashMap
+getAllPosts(): HashMap
+toString(): String
+postManager
+posts
+posts
Post
-title: String
-text: String
-group: Group
-isFlagged: boolean
-flagged: String
-responses: HashMap
-date: LocalDate
-time: LocalTime
-flagged: Boolean
+Post(title: String, text: String, group: Group, date: LocalDate, time: LocalTime)
+getTitle(): String
+setTitle(title: String)
+getText(): String
+setText(String: text)
+getGroup(): Group
+setDate(date: LocalDate)
+getTime(): LocalTime
+setTime(time: LocalTime)
+getFlagged(): Boolean
+getResponse(text: String): Response
+addResponse(response: Response): boolean
+getResponses(): ArrayList<Response>
+equals(obj: Object): boolean
+toString(): String
+responseManager
ResponseManager
+responses: HashMap
+ResponseManager()
+addResponse(response: Response): boolean
+removeResponse(response: Response): boolean
+getNumResponses(): int
+getResponse(responseText: String): Response
+hasResponse(responseText: String): boolean
+getHashMap(): HashMap
+clear()
+toString(): String
+post
+responses
+responses
Response
-text: String
-post: Post
-responses: HashMap
-parentResponse: Response
+Response(text: String, post: Post)
+getText(): String
+setText(text: String)
+getPost(): Post
+setPost(post: Post)
+addResponse(response: Response): boolean
+getResponses(): ArrayList<Response>
+getParentResponse(): Response
+setParentResponse(parentResponse: Response)
+equals(obj: Object): boolean
+toString(): String


application
main
+start(primaryStage: Stage)
+main(args: String)
GroupItemController
-lblGroup: Label
-hBoxGroup: HBox
+GroupItemController()
+setData(group: Group)
+initialize(location: URL, resources: ResourceBundle)
GroupItem.fxml
LoginController
-txtAreaUserName: TextField
-txtAreaPassword: PasswordField
-btnLogin: Button
+handleLoginEvent(event: ActionEvent)
+handleRegisterEvent(event: ActionEvent)
+handleVisitorEvent(event: ActionEvent)
Login.fxml
RegisterController
-txtAreaUsername: TextField
-txtAreaPassword: passwordField
-txtAreaPasswordConfirm: PasswordField
-txtAreaName: TextField
-pickerBirthday: DatePicker
-txtAreaCountry: TextField
-txtAreaState: TextField
-txtAreaCity: TextField
+handleRegisterEvent(event: ActionEvent)
+handleReturnEvent(event: ActionEvent)
RegisterAccount.fxml
HubHomeController
-persistence: Persistence
-categoryManager: CategoryManager
-groupManager: GroupManager
-postManager: PostManager
-responseManager: ResponseManager
-selectedCategory: Category
-selectedGroup: Group
-selectedPost: Post
-imgSearch: ImageSearch
-imgSortAlpha: ImageView
-imgSortLike: ImageView
-sPaneview: ScrollPane
-txtFieldSearch: TextField
+HubHomeController()
+searchUsers(event: MouseEvent)
+sortAlpha(event: MouseEvent)
+sortSuspensions(event: MouseEvent)
+initialize(location: URL, resources: ResourceBundle)
+showCategories()
+showGroups(category: Category)
+requestGroup()
+showPosts(group: Group)
+createPost()
+flagPost()
+showResponses(post: Post)
+createResponse()
+flagResponse()
HubUser.fxml
Hubsuspended.fxml
HubHome.fxl
Hubflag.fxml
hubpost.fxml
HubBanned.fxml
HubUsersController
-persistence: Persistence
-userManager: UserManager
-bannedDashboard: VBox
-imgSearch: ImageView
-imgSortBans: ImageView
-sPaneView: ScrollPane
-txtFieldSearch: TextField
+searchUsers(event: MouseEvent)
+sortAlpha(event: MouseEvent)
+sortBans(event: MouseEvent)
+initialize(location: URL, resources: ResourceBundle)
HubSuspendedConroller
-persistence: Persistence
-userManager: UserManager
-suspendedDashboard: VBox
-imgSearch: ImageView
-imgSortAlpha: ImageView
-imgSortLike: ImageView
-sPaneView: ScrollPane
-txtFieldSearch: TextField
+HubSuspendedController()
+searchUsers(event: MouseEvent)
+sortAlpha(event: MouseEvent)
+sortSuspensions(event: MouseEvent)
+initialize(location: URL, resources ResourceBundle)
HubFlaggedResponseController
-persistence: Persistence
-responseManager: ResponseManager
-flaggedDashboard: VBox
-imgSearch: ImageView
-imgSortAlpha: ImageView
-imgSortLike: ImageView
-sPaneView: ScrollPane
-txtFieldSearch: TextField
+HubFlaggedResponsesController()
+searchUsers(event: MouseEvent)
+sortAlpha(event: MouseEvent)
+sortSuspensions(event: MouseEvent)
+initialize(location: URL, resources: ResourceBundle)
HubFlaggedPostsController
-persistence: Persistence
-postManager: PostManager
-flaggedDashboard: VBox
-imgSearch: ImageView
-imgSortAlpha: ImageView
-imgSortLike: ImageView
-sPaneView: ScrollPane
-txtFieldSearch: TextField
+HubFlaggedPostsController()
+searchUsers(event: MouseEvent)
+sortAlpha(event: MouseEvent)
+sortSuspensions(event: MouseEvent)
+initialize(location: URL, resources: ResourceBundle)
HubBannedController
-persistence: Persistence
-userManager: UserManager
-sPaneView: ScrollPane
-bannedDashboard: VBox
-imgSortAlpha: ImageView
-imgSortLike: ImageView
-imgSearch: ImageView
-txtFieldSearch: TextField
+HubBannedController()
+searchUsers(event: MouseEvent)
+sortAlpha(event: MouseEvent)
+sortBans(event: MouseEvent)
+initialize(location: URL, resources: ResourceBundle)
Persistence
-userManager: UserManager
-categoryManager: CategoryManager
-groupManager: GroupManager
-postManager: PostManager
-responseManager: ResponseManager
-DATA: String
-storedData: JSONObject
-mainObj: JSONObject
-secondObj: JSONObject
+Persistence()
+loadData()
+saveData()
+readJsonFile()
+getJsonFile()
+buildUserManager()
+buildCategoryManager()
+buildGroupManager()
+buildPostManager()
+buildResponseManager()
+getUserManager(): UserManager
+getCategoryManager(): CategoryManager
+getGroupManager(): GroupManager
+getPostManager(): PostManager
+getResponseManager(): ResponseManager
+convertToUser(obj: JSONObject): User
+convertToGroup(obj: JSONObject): Group
+convertToPost(obj: JSONObject): Post
+convertToResponse(JSONObject): Response
+saveUserManager(): JSONObject
+saveCategoryManager(): JSONObject
+saveGroupManger(): JSONObject
+savePostManager(): JSONObject
+saveResponseManager()
+writeToJson()
+updatePostToJSON(post: Post)
Json.Data
CategoryItem.fxml
CategoryItemController
-lblGroup: Label
-hBoxGroup: HBox
+CategoryItemController()
+setData(category: Category)
+categoryClicked(event: MouseEvent)
+initialize(location: URL, resource: ResourceBundle)
UserItem.fxml
UserItemController
-lblUsername
-hBoxUser: HBox
+UserItemController()
+setData(user: User)
+initialize(location: URL, resources: ResourceBundle)
PostItem.fxml
PostItemController
-lblPostTitle: Label
-lblPostText: Label
-lblPostAuthor: Label
-lblPostLikes: Label
-gridPanePost: GridPane
+PostItemController()
+setData(post: Post)
+initialize(location: URL, resources: ResourceBundle)
ResponseItem.fxml
ResponseItemController
-lblResponse: Label
-lblResponseAuthor: Label
-lblResponseLikes: Label
-gridPanePost: GridPane
+ResponseItemController()
+setData(response: Response)
+initialize(location: URL, resources: ResourceBundle)

Our Team 7 project was divided into three core sections: frontend, backend, and a crucial persistence layer, managed by different team members. Skylar handled the frontend, creating an intuitive interface using Scene Builder and FXML files. Damian and I managed the controllers, ensuring smooth data transfer between the frontend and backend. Skylar's contribution of a persistence class was vital for converting data into a format understood by both sides, ensuring seamless communication.

Jacob was pivotal in the backend, designing data storage and taking charge of quality assurance (QA) testing. His dual role ensured the backend's reliability and security, along with its effective functionality.

We followed the Model-View-Controller (MVC) architecture, creating clear divisions between frontend, backend, and controllers. Skylar's frontend seamlessly integrated with controller functions, facilitating effective communication with Jacob's backend.

Jacob's involvement in backend development and QA testing significantly enhanced the system's reliability. His thorough testing validated the system's robustness, contributing greatly to the project's stability.

The collaborative effort among team members in these core sections ensured a cohesive integration of frontend design, backend functionality, and data persistence. This integrated approach followed industry best practices, resulting in a functional and efficient system.

In summary, the coordinated efforts across frontend, backend, and QA testing by each team member led to a well-structured project that met and exceeded our expectations.

3 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.

<<<Example files and narrative go here>>>

The way in which we stored all data was using a JSON file. This included: user information, categories, groups, posts, and responses. Any methods that have anything to do with touching the JSON file were done in the persistence class.


```
{
  "users": [
    {
      "name": "John Doe",
      "username": "johndoe123",
      "password": "password123",
      "birthday": "1990-05-15",
      "country": "USA",
      "city": "New York",
      "state": "NY",
      "role": "user"
    },
```

```
  // Other user objects...
 ],
 "categories": [
  "Category 1",
  "Category 2",
  "Category 3"
  // Other category titles...
 ],
 "groups": [
  {
   "title": "Group 1",
   "parentCategory": "Category 1"
  },
  // Other group objects...
 ],
 "posts": [
  {
   "title": "Post 1",
   "text": "This is the content of Post 1.",
   "parentGroup": "Group 1",
   "isFlagged": false
  },
  // Other post objects...
 ],
 "responses": [
  {
   "text": "Response to Post 1",
   "parentPost": "Post 1"
  },
  // Other response objects...
 ],
 "responseResponses": [
```

```
  {
    "text": "Response to a Response",
    "parentResponse": "Response to Post 1"
  },
  // Other response response objects...
 ]
}
```

To store each of the objects reliably and neatly, all objects are stored under respective JSON arrays. For example, a user would have all their information including: username, password, birthday, etc., would be stored in an array called “users”. This is done for any objects within the software.

The persistence class has two main functions which run a multitude of smaller functions: one to load all the data from the JSON file to allow for the managers to get the data they need, and one to save all data from the managers and add all the data to the JSON file.

By keeping all the data in a single JSON file, it makes for an easier time with the persistence to load and save data as well as keeping track of the data.

4 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	2	US 21	2	US 28		US 40	
US 2	2	US 11	5	US 22		US 29		US 41	
US 3	1	US 12	2	US 23		US 30		US 42	
US 4		US 13	1	US 24		US 31			
US 5		US 14		US 25		US 32			
US 6	1	US 15	2	US 26		US 33			
US 7		US 16		US 27		US 34			
US 8		US 17				US 35			
US 9		US 18				US 36			
		US 19				US 37			
		US 20				US 38			
						US 39			
Total	6		12		2		0		0

Grand Total: 20

5 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.

a. What worked well for us?

Answer
Multiple things worked well. We were good at establishing meeting times to discuss and communicate plans for the project. We were good at establishing the MVC model and assigning people to different parts of the model. We were able to utilize the JavaFX Scene Builder efficiently to create a working GUI to allow for more focus on backend code.

b. What did not work well for us?

Answer
We had a lot of issues with version control. There were too many instances where pulling each other's code to our local systems resulted in our local systems completely breaking. There were a lot of times where classpaths would be pulled. Merge conflicts were a huge issue since they usually were massive and complicated. We also had some moments where we were working on issues that were already fixed on a different branch.

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

Answer
In hindsight, committing more often with less changes per commit would have fixed a lot of issues. Having less changes would reduce the scale of merge conflicts, and it would reduce the probability and quantity of code issues when pulling new commits. Utilizing more GitHub feature could have prevented many of the issues we had. For example, using .gitIgnore to prevent pulling classpaths and having a better understand of merge conflicts to quickly resolve them.

6 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.

7 Video Demo

Deliverable

When your project is complete, create a video that demo's your User Stories and provide the link in Section 1. 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:
    5. (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.
    6. (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.
    7. (1 minute) GitHub:
        a. Display your Project Board for 10 seconds.
        b. Display the Issues for 5 seconds, then display the "Closed" issues
    8. For each User Story:
        a. State the number of the User Story, and then state the User Story itself.
        b. 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.