Threaded Comments
Technical Report

Joshua Sinclair Chong & Joseph Isaacs

10 December 2024

About

Threaded Comments is a commenting system made to be used on any social media platform. It features a flexible and extensible comment structure, dynamic user actions permissions & roles, a system for notifications, and is built such that it can be modified easily to reflect many kinds of existing comment systems. For example, adding a new reaction system would be easy with a few additions and no need to refactor.

Features

Comment Threads

The base structure of the comment threads; tree-like. Posts can have comments on them, and those comments can have more comments (aka replies). When a comment is posted it starts a new thread for all replies.

Notifying Commenters

When a user makes a new comment on a post or replies to a comment, they are added to its thread and will be notified of any updates on it (ex: new comments in the thread); this is called Subscribing.

Actions

Users have comment actions they can take based on their current role / permissions. User actions include replying with new comments, editing previous comments, or deleting comments.

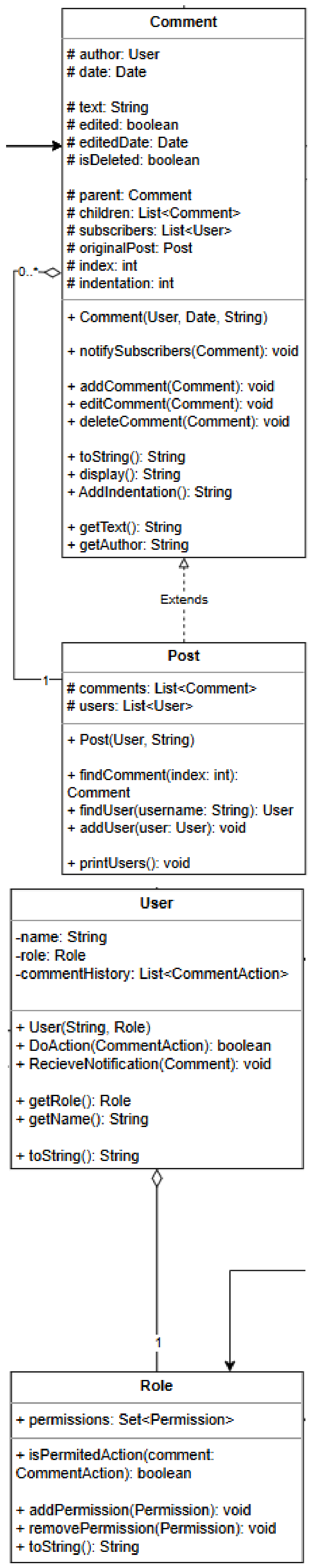
User Roles & Permissions

Users have different permissions based on their role in the comment tree. (Admin) can delete any comments because it's their post and they should be able to moderate it. (Commenter) People who can post new comments, edit, or delete their own comments. (Viewer) Only able to view. The Roles and their Permissions are dynamic as well.

Design Patterns

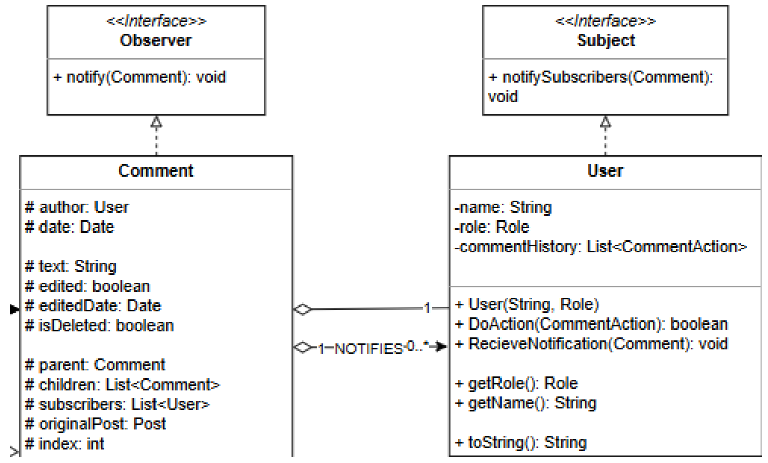
Comment Threads: Composite

We chose the Composite Pattern to help organize comments into tree-like structures. This organization is very intuitive because comments only care about what is attached to them or what they are attached to; so this makes sure the comments have only the information they need. A Post stores all Comments, which means that it's still easy for users to access the Comment they want.


Comment
# author: User
# date: Date
# text: String
# edited: boolean
# editedDate: Date
# isDeleted: boolean
# parent: Comment
# children: List<Comment>
# subscribers: List<User>
# originalPost: Post
# index: int
# indentation: int
+ Comment(User, Date, String)
+ notifySubscribers(Comment): void
+ addComment(Comment): void
+ editComment(Comment): void
+ deleteComment(Comment): void
+ toString(): String
+ display(): String
+ AddIndentation(): String
+ getText(): String
+ getAuthor: String
0..*
Extends
1
Post
# comments: List<Comment>
# users: List<User>
+ Post(User, String)
+ findComment(index: int): Comment
+ findUser(username: String): User
+ addUser(user: User): void
+ printUsers(): void
User
-name: String
-role: Role
-commentHistory: List<CommentAction>
+ User(String, Role)
+ DoAction(CommentAction): boolean
+ RecieveNotification(Comment): void
+ getRole(): Role
+ getName(): String
+ toString(): String
1
Role
+ permissions: Set<Permission>
+ isPermitedAction(comment: CommentAction): boolean
+ addPermission(Permission): void
+ removePermission(Permission): void
+ toString(): String

Notifying Commenters: Observer

The main functionality of the Observer pattern is to notify the subscribers and parent of a comment that is replied to

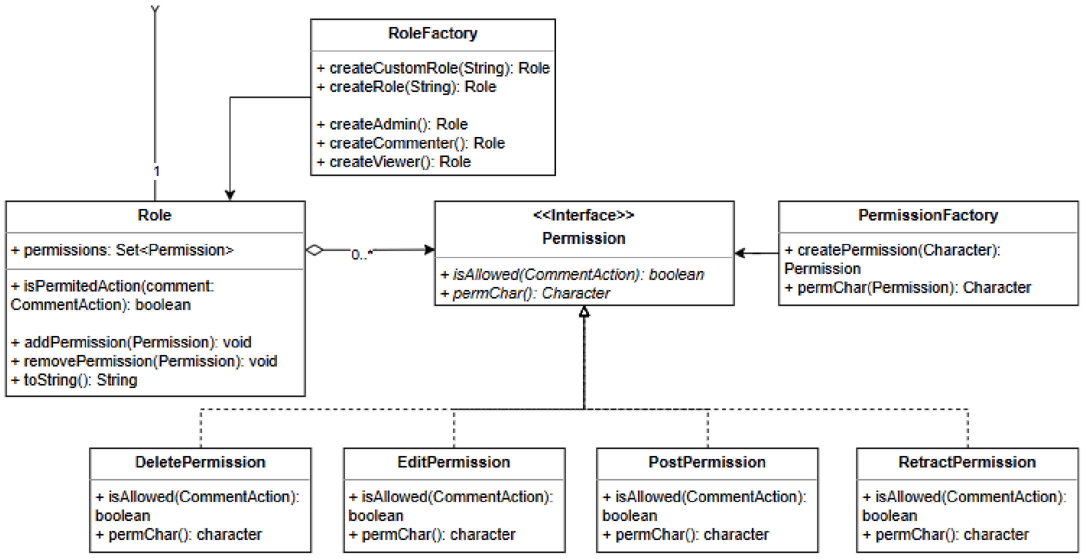


User Roles & Permissions: Strategy

We used the Strategy pattern because it greatly decouples the user and permissions. Dealing with permissions is offloaded to the Role class, meaning the User class can have simpler functions and whether they can actually run is on the Role class. It also means roles can be changed easily at runtime.

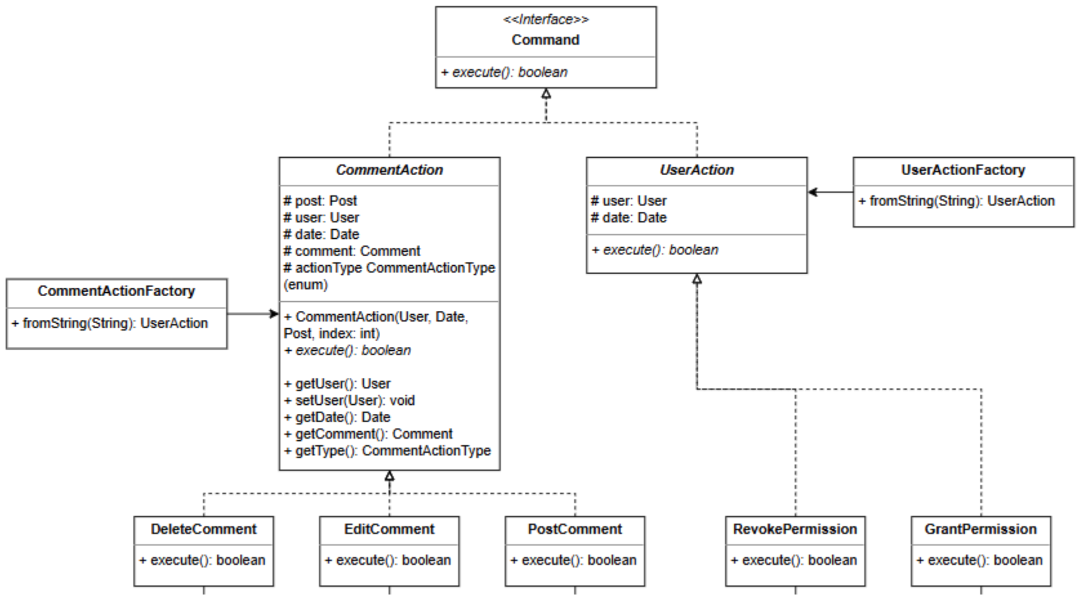
Roles, Actions, & Actions on Users: Factory

For User Roles, Actions, & Actions on Users, we used the Factory Pattern in order to quickly and dynamically make different kinds.

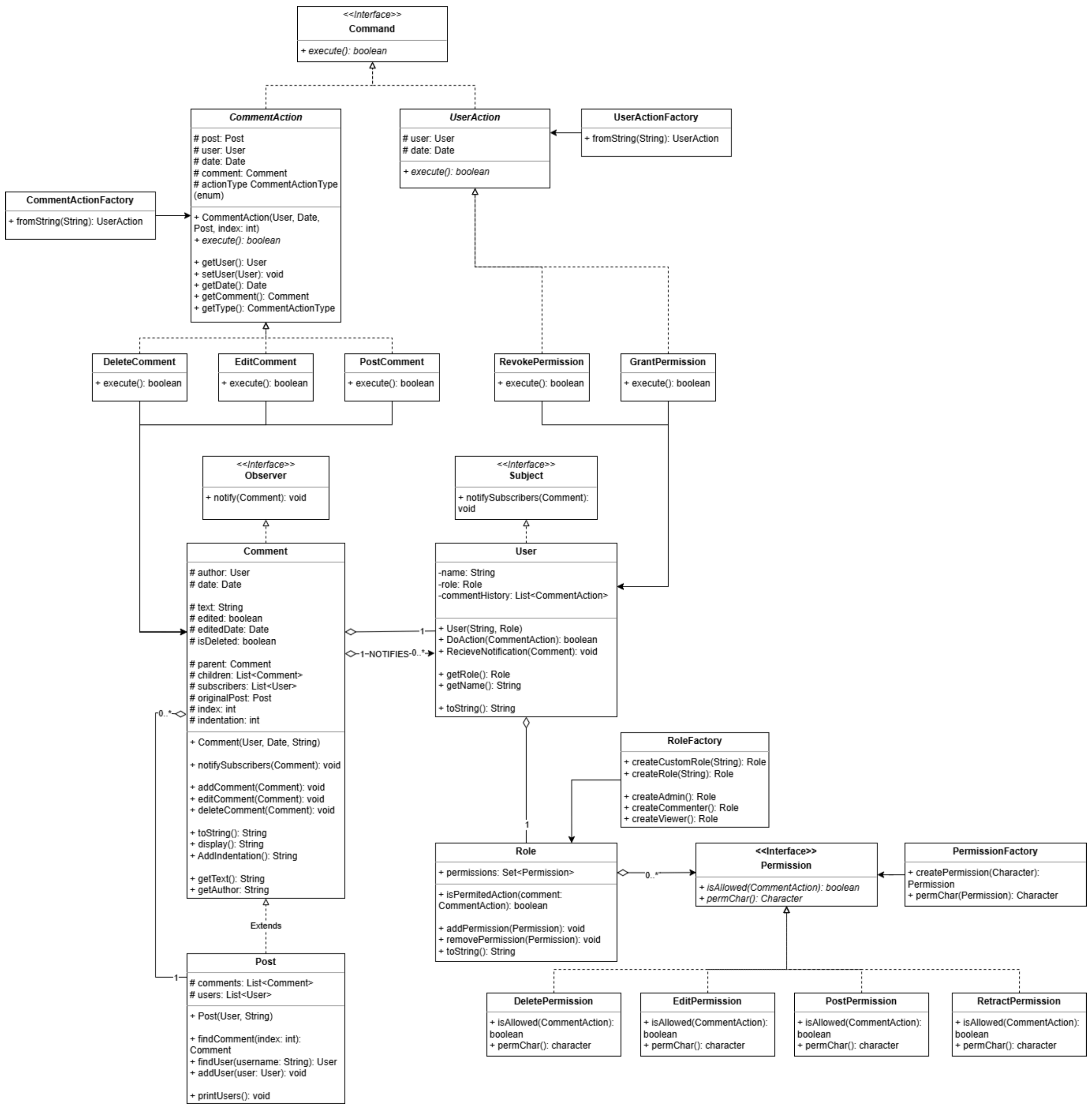

RoleFactory
+ createCustomRole(String): Role
+ createRole(String): Role
+ createAdmin(): Role
+ createCommenter(): Role
+ createViewer(): Role
Role
+ permissions: Set<Permission>
+ isPermitedAction(comment: CommentAction): boolean
+ addPermission(Permission): void
+ removePermission(Permission): void
+ toString(): String
1
0..*
<<Interface>>
Permission
+ isAllowed(CommentAction): boolean
+ permChar(): Character
PermissionFactory
+ createPermission(Character): Permission
+ permChar(Permission): Character
DeletePermission
+ isAllowed(CommentAction): boolean
+ permChar(): character
EditPermission
+ isAllowed(CommentAction): boolean
+ permChar(): character
PostPermission
+ isAllowed(CommentAction): boolean
+ permChar(): character
RetractPermission
+ isAllowed(CommentAction): boolean
+ permChar(): character

User Actions & Actions on Users: Command

We used the Command pattern for all the User Actions for flexibility, to decouple the Users from Actions, and in order to have a history. Adding a new Action does not require touching the User class. Same thing for Actions on Users, which handle dynamically changing Permissions on a User's Role.



Full UML



<<Interface>>
Command
+ execute(): boolean
CommentAction
# post: Post
# user: User
# date: Date
# comment: Comment
# actionType CommentActionType (enum)
+ CommentAction(User, Date, Post, index: int)
+ execute(): boolean
+ getUser(): User
+ setUser(User): void
+ getDate(): Date
+ getComment(): Comment
+ getType(): CommentActionType
UserAction
# user: User
# date: Date
+ execute(): boolean
UserActionFactory
+ fromString(String): UserAction
CommentActionFactory
+ fromString(String): UserAction
DeleteComment
+ execute(): boolean
EditComment
+ execute(): boolean
PostComment
+ execute(): boolean
RevokePermission
+ execute(): boolean
GrantPermission
+ execute(): boolean
<<Interface>>
Observer
+ notify(Comment): void
<<Interface>>
Subject
+ notifySubscribers(Comment): void
Comment
# author: User
# date: Date
# text: String
# edited: boolean
# editedDate: Date
# isDeleted: boolean
# parent: Comment
# children: List<Comment>
# subscribers: List<User>
# originalPost: Post
# index: int
# indentation: int
+ Comment(User, Date, String)
+ notifySubscribers(Comment): void
+ addComment(Comment): void
+ editComment(Comment): void
+ deleteComment(Comment): void
+ toString(): String
+ display(): String
+ AddIndentation(): String
+ getText(): String
+ getAuthor: String
User
-name: String
-role: Role
-commentHistory: List<CommentAction>
+ User(String, Role)
+ DoAction(CommentAction): boolean
+ RecieveNotification(Comment): void
+ getRole(): Role
+ getName(): String
+ toString(): String
1
1~NOTIFIES~0..*
0..*
1
Extends
Post
# comments: List<Comment>
# users: List<User>
+ Post(User, String)
+ findComment(index: int): Comment
+ findUser(username: String): User
+ addUser(user: User): void
+ printUsers(): void
RoleFactory
+ createCustomRole(String): Role
+ createRole(String): Role
+ createAdmin(): Role
+ createCommenter(): Role
+ createViewer(): Role
1
Role
+ permissions: Set<Permission>
+ isPermitedAction(comment: CommentAction): boolean
+ addPermission(Permission): void
+ removePermission(Permission): void
+ toString(): String
0..*
<<Interface>>
Permission
+ isAllowed(CommentAction): boolean
+ permChar(): Character
PermissionFactory
+ createPermission(Character): Permission
+ permChar(Permission): Character
DeletePermission
+ isAllowed(CommentAction): boolean
+ permChar(): character
EditPermission
+ isAllowed(CommentAction): boolean
+ permChar(): character
PostPermission
+ isAllowed(CommentAction): boolean
+ permChar(): character
RetractPermission
+ isAllowed(CommentAction): boolean
+ permChar(): character