Alexander Hamlet

Project: SpeakEasy

Part 6

Implemented Features

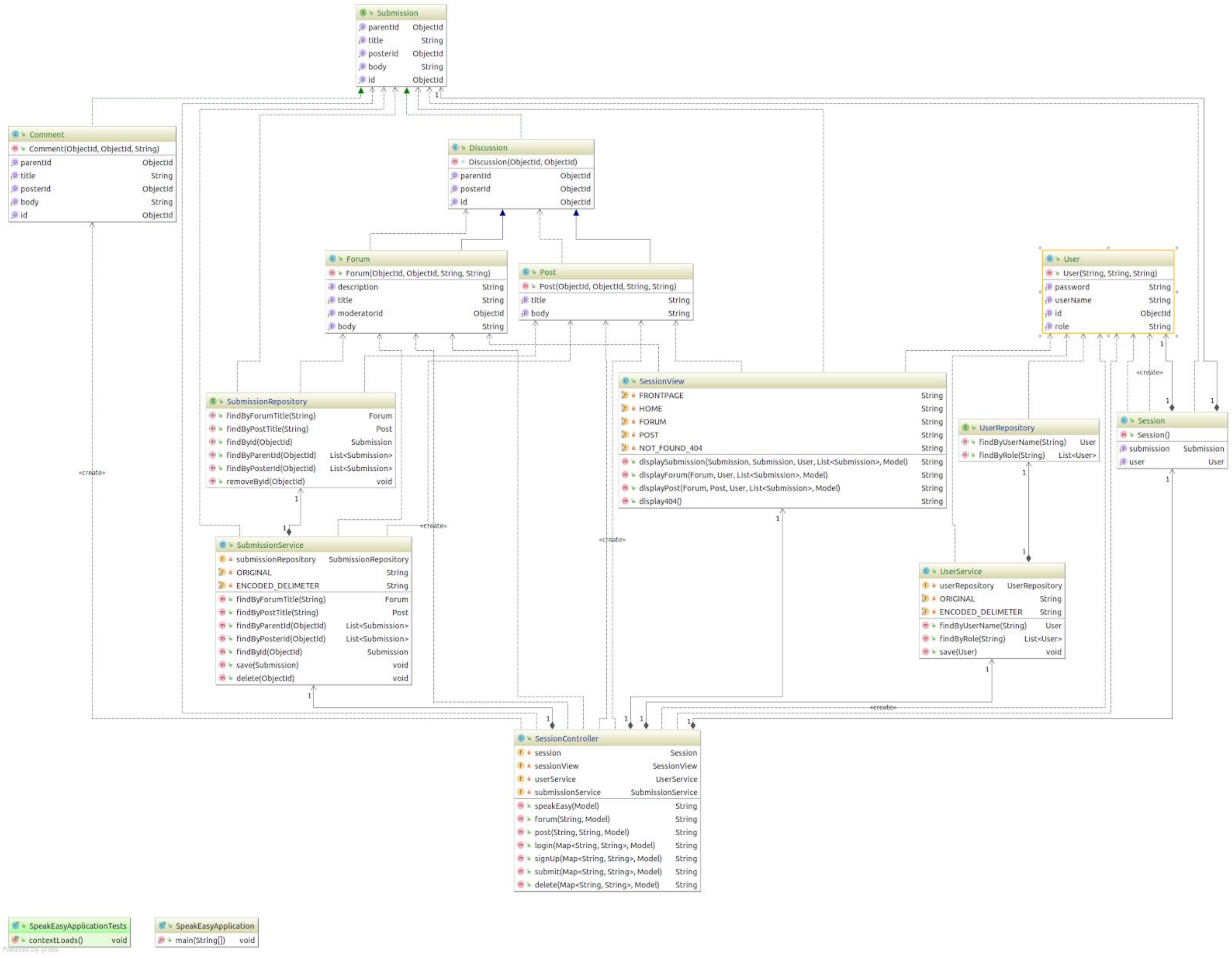
Sign Up	Sign up by creating an account on SpeakEasy. Users can create their account with admin or regular permissions.
Log In	Log in to an account by submitting username and password.
Create Forum	Users can create Forums with a title and a description, become the moderator of those Forums that they create, and delete posts inside those Forums.
Create Post	Users can create Posts in Forums of their choosing. Posts are discussions centered around the Forum topic.
Create Comment	Users can create Comments on Posts of their choosing. Comments are simple text responses to Posts. Comments cannot be deleted.
Delete Forum	Delete a Forum. User must be logged in as admin or be the moderator of that forum.
Delete Post	Delete a Post. User must be logged in as admin, the Forum moderator, or be the original poster to do this.
Navigation	Simple navigation through website via links. Created dynamically.

Unimplemented Features

Delete Comment	Deleting comments goes against my vision for the product. I want SpeakEasy to be open to discussion and allowing the deletion of comments hinders that. For instance, if a comment in the middle of a discussion was deleted, then there would be a gap in the conversation. Deleted comments would hinder the flow of discussion inhibiting people from having the full conversation.
Ban User	Banning users goes against my vision for the product. Not allowing certain users to talk in forums would inhibit possible conversations.

Strip Moderator of managing their forum	Stripping moderators from managing their forums goes against my vision for this site. Moderators created their forums for their own interest and allowing admins to strip them from that pursuit is counterintuitive. It would inhibit conversations.

<u>Full Class Diagram</u>

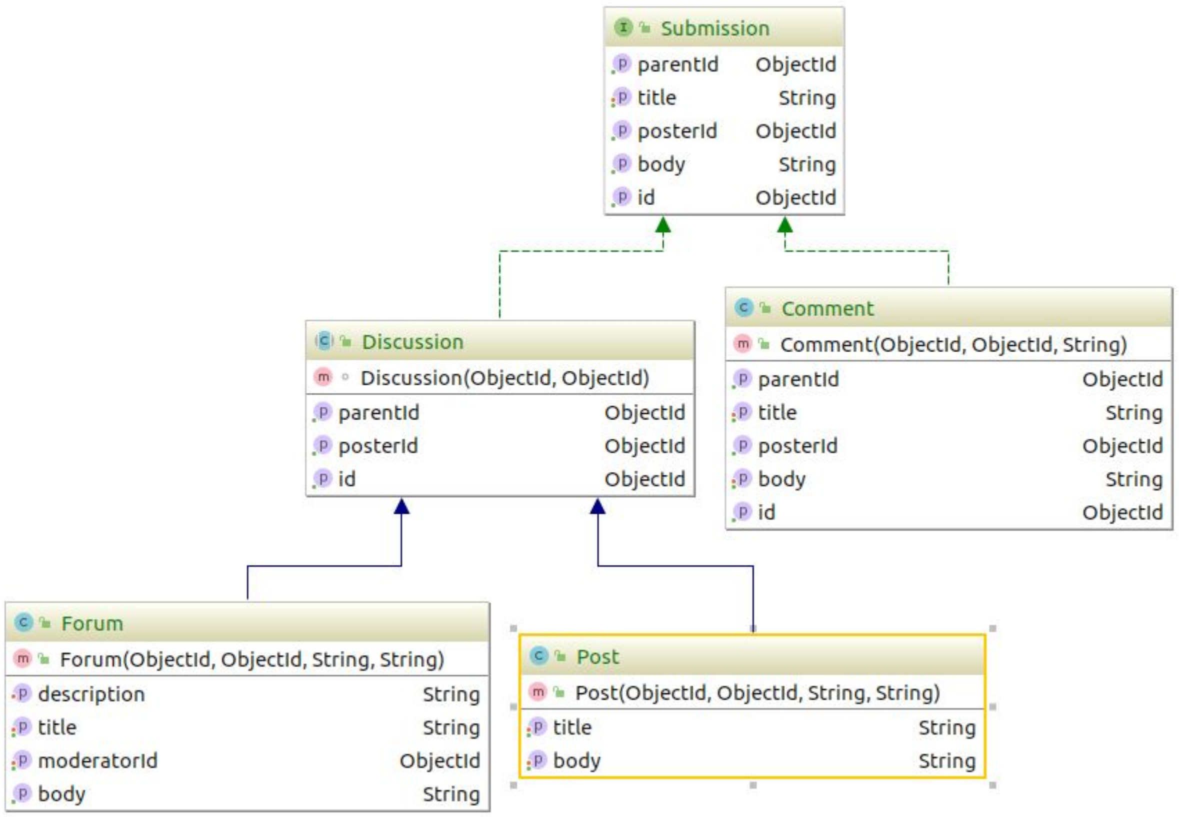

Submission
parentId ObjectId
title String
posterId ObjectId
body String
id ObjectId
Comment
Comment(ObjectId, ObjectId, String)
parentId ObjectId
title String
posterId ObjectId
body String
id ObjectId
Discussion
Discussion(ObjectId, ObjectId)
parentId ObjectId
posterId ObjectId
id ObjectId
Forum
Forum(ObjectId, ObjectId, String, String)
description String
title String
moderatorId ObjectId
body String
Post
Post(ObjectId, ObjectId, String, String)
title String
body String
User
User(String, String, String)
password String
userName String
id ObjectId
role String
SubmissionRepository
findByForumTitle(String) Forum
findByPostTitle(String) Post
findById(ObjectId) Submission
findByParentId(ObjectId) List<Submission>
findByPosterId(ObjectId) List<Submission>
removeById(ObjectId) void
SessionView
FRONTPAGE String
HOME String
FORUM String
POST String
NOT_FOUND_404 String
displaySubmission(Submission, Submission, User, List<Submission>, Model) String
displayForum(Forum, User, List<Submission>, Model) String
displayPost(Forum, Post, User, List<Submission>, Model) String
display404() String
UserRepository
findByUserName(String) User
findByRole(String) List<User>
Session
Session()
submission Submission
user User
SubmissionService
submissionRepository SubmissionRepository
ORIGINAL String
ENCODED_DELIMETER String
findByForumTitle(String) Forum
findByPostTitle(String) Post
findByParentId(ObjectId) List<Submission>
findByPosterId(ObjectId) List<Submission>
findById(ObjectId) Submission
save(Submission) void
delete(ObjectId) void
UserService
userRepository UserRepository
ORIGINAL String
ENCODED_DELIMETER String
findByUserName(String) User
findByRole(String) List<User>
save(User) void
SessionController
session Session
sessionView SessionView
userService UserService
submissionService SubmissionService
speakEasy(Model) String
forum(String, Model) String
post(String, String, Model) String
login(Map<String, String>, Model) String
signUp(Map<String, String>, Model) String
submit(Map<String, String>, Model) String
delete(Map<String, String>, Model) String
SpeakEasyApplicationTests
contextLoads() void
SpeakEasyApplication
main(String[]) void
«create»
«create»
«create»
«create»
«create»
1
Powered by yFiles

Class Diagram Explanation

My class diagram looks very different from the diagram in part 2; however, it actually hasn't changed that much. I'm using three design patterns to make SpeakEasy operate. I'm using MVC, Composite, and DAO. The MVC pattern runs the website via the Session model, the SessionController, and the SessionView classes. The Composite pattern handles the structure of the submission objects (the forums, the posts, and the comments). The two Data Access Object patterns handle the user and submission data to and from their respective collections in the Mongo Database. In my original diagram, you'll notice I had Menu objects organized into a State Design pattern. During development, I realized this pattern was unnecessary as the SessionView handles the display of the menu based on the role of the current user. Having separate Menu objects to organize those displays would be redundant and have more complexity than is necessary.

Individual Design Patterns

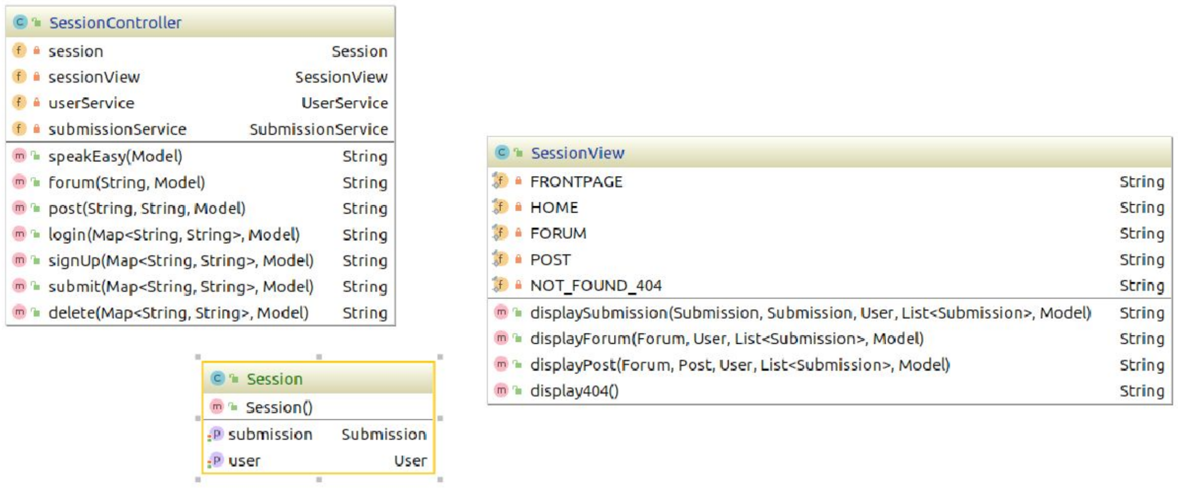
Composite



My design works by having each Submission in the tree keep track of its parent Submission. This was done, versus parents keeping track of children, to utilize features from MongoDB. Only Discussions (Forums and Posts) can have other Submissions call them parents. Comments cannot be the parent to any Submission as they are the leaf nodes in this design.

I implemented this design pattern in order to keep the structure of the website easily navigable and dynamic in size. It works since comments are the children of posts, posts are the children of forums, and forums are the children of the main home page called the “Frontpage”. When users visit the site their greeted with the Frontpage. SpeakEasy then populates the page with a list of forums (the Frontpage's children) that users can visit and contribute to. When they visit a forum, SpeakEasy populates the page with that forum's posts (the forum's children) that users can discuss. If they choose to discuss or read on, they click on the post title, then SpeakEasy populates that post's page with its comments (a.k.a the post's children). This also allows for simple deletion of objects as if the parent of an object is deleted, its children get deleted as well.

MVC

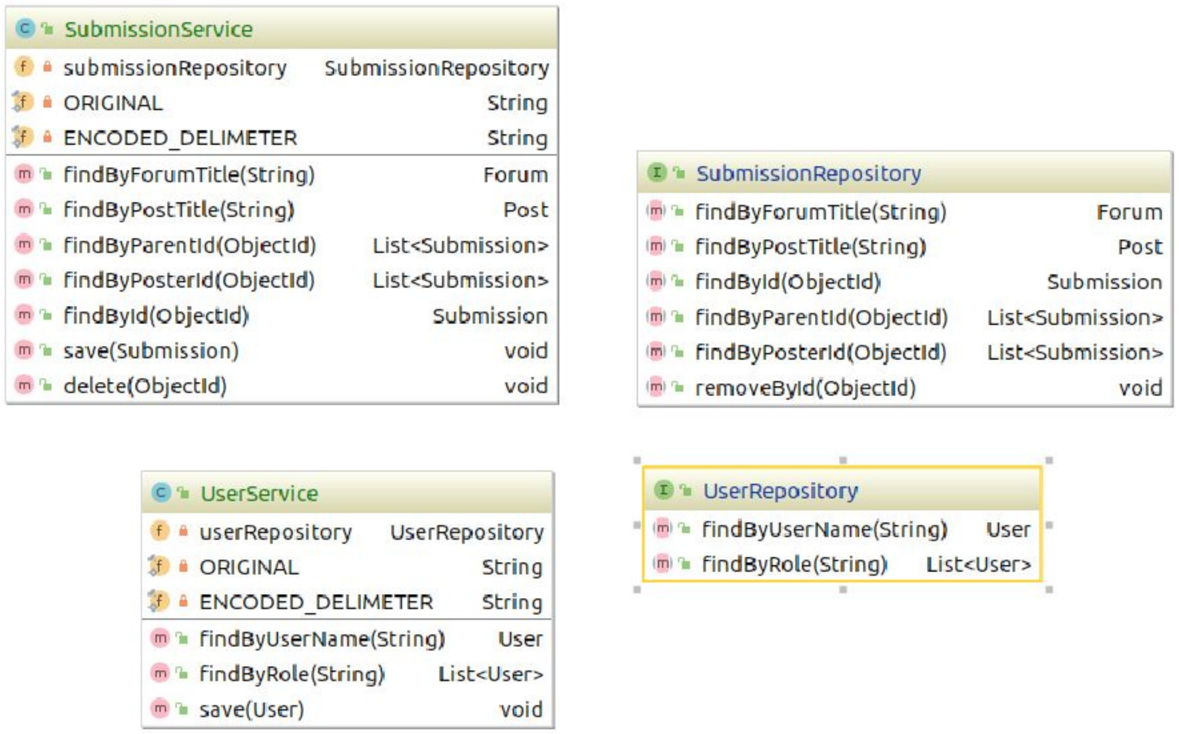


SessionController
session Session
sessionView SessionView
userService UserService
submissionService SubmissionService
speakEasy(Model) String
forum(String, Model) String
post(String, String, Model) String
login(Map<String, String>, Model) String
signUp(Map<String, String>, Model) String
submit(Map<String, String>, Model) String
delete(Map<String, String>, Model) String
SessionView
FRONTPAGE String
HOME String
FORUM String
POST String
NOT_FOUND_404 String
displaySubmission(Submission, Submission, User, List<Submission>, Model) String
displayForum(Forum, User, List<Submission>, Model) String
displayPost(Forum, Post, User, List<Submission>, Model) String
display404() String
Session
Session()
submission Submission
user User

I'm using this pattern to control the whole functionality of the site. The Session (the model) holds the current user data and the current submission data that the user is viewing. The SessionView handles the display and dispatch of the website templates and also ensures pieces appear where they're supposed to. The SessionController handles the routing and logic and is the main controller for the SpringBoot framework that I am using.

I chose this pattern as a way to differentiate out logic from data and displays, and allows me to keep the central functions of the website (submit, login, signup, and delete) in one place for easy maintainability.

Data Access Object


SubmissionService
submissionRepository SubmissionRepository
ORIGINAL String
ENCODED_DELIMETER String
findByForumTitle(String) Forum
findByPostTitle(String) Post
findByParentId(ObjectId) List<Submission>
findByPosterId(ObjectId) List<Submission>
findById(ObjectId) Submission
save(Submission) void
delete(ObjectId) void
SubmissionRepository
findByForumTitle(String) Forum
findByPostTitle(String) Post
findById(ObjectId) Submission
findByParentId(ObjectId) List<Submission>
findByPosterId(ObjectId) List<Submission>
removeById(ObjectId) void
UserService
userRepository UserRepository
ORIGINAL String
ENCODED_DELIMETER String
findByUserName(String) User
findByRole(String) List<User>
save(User) void
UserRepository
findByUserName(String) User
findByRole(String) List<User>

The two DAO’s, one for user objects and one for submission objects, allow for the separation of responsibilities of accessing the different collections in the Mongo Database. The Session and the SessionController don’t have to handle these functionalities, because these objects do. The SubmissionRepository and UserRepository objects directly deal with the respective collections. They hold the CRUD operations and implement the MongoRepository interface to do so. This means they also have access to Mongo’s “findBy” convention allowing Lists of objects to be pulled out at once. This is why the composite design’s tree structure relies on nodes keeping track of their parent’s id. The SubmissionRepository can pull out every child with a specified parent’s id all at once. The SubmissionService and UserService utilize the repository objects and are used by the SessionController to access the database. They also hold a small amount of logic to correctly format submissions and users going in and out of the database.

What have I learned about the process of analysis and design?

I have learned that the process of analysis and design is necessary and essential for the creation and maintenance of software systems both big and small.

All systems have essential decisions that must be made in order to run smoothly. Even small system have multiple moving parts that create, store, and perform logic. These parts must be designed to interact in specific predictable ways to not only ensure that the quality of the system is maintained but also so developers know how the system is working from the ground up. In large systems, there would be no maintainability without class diagrams. Large projects need people to know how they work in order to be refactored, debugged, and updated. With software engineers coming and going, using common design patterns and class diagrams is essential for the survival of the system.

After creating, designing, and implementing a small system I’m glad there are common designs and steps that simplify this process. It would be very difficult if they did not exist.