Askie Forum

Danh Nguyen
Onid:nguydanh
nguydanh@oregonstate.edu

Joshua Bell
Onid:belljos
belljos@oregonstate.edu

Cameron Kocher
Onid:kochecam
kochecam@oregonstate.edu

Nicholas Newell
Onid:newelln
newelln@oregonstate.edu

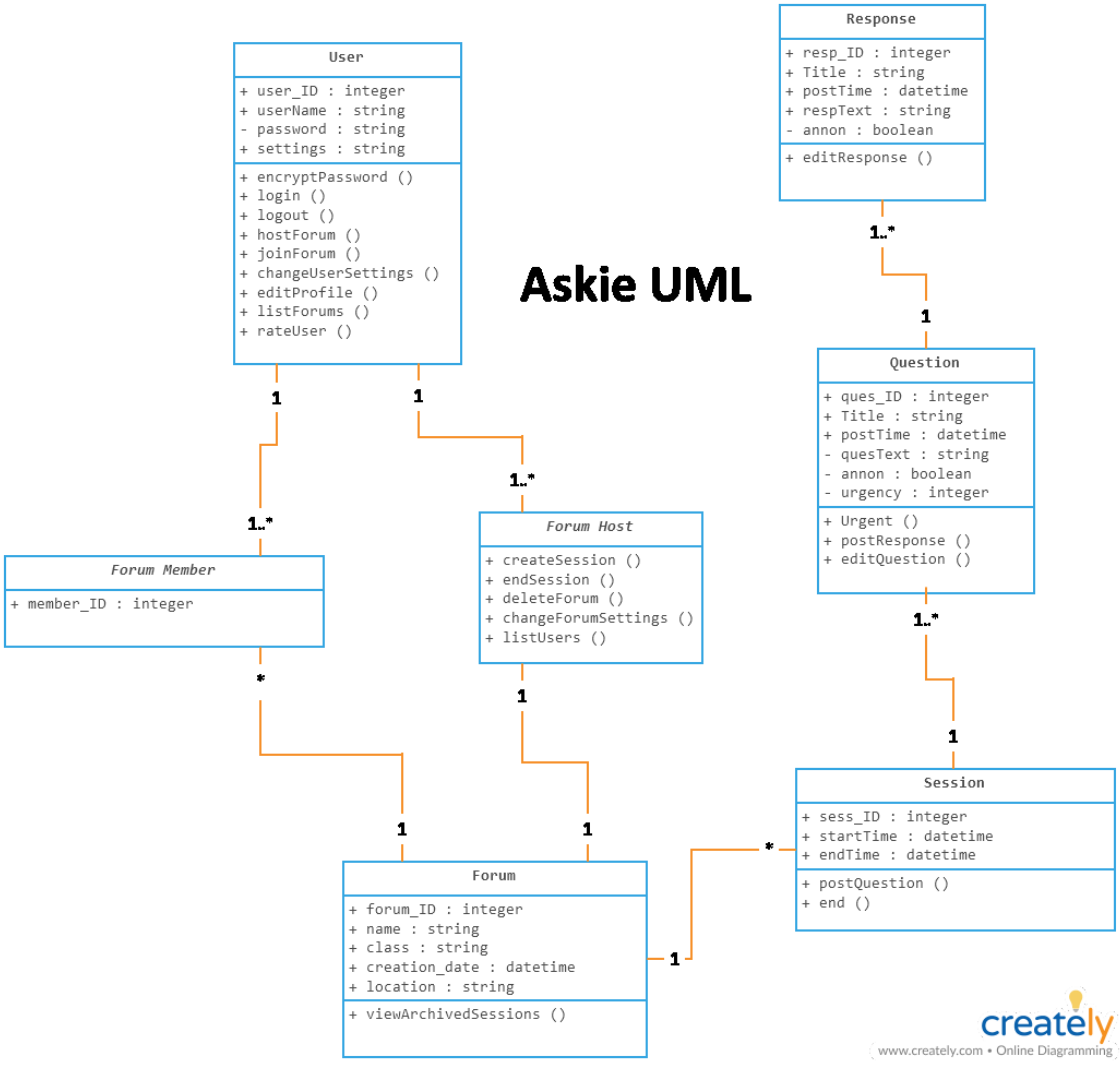
Andrew Morrill
Onid:morrilan
morrilan@oregonstate.edu

GitHub: https://github.com/hydra314/AskieForum

February 16, 2018

Contents

1 UML Class Diagram



The UML diagram specifies the separation kept between forum users and hosts, as they have different permissions.

2 Packages

Before we talk about how our team will package our entities, we will first discuss the types of entities we plan on implementing and what each of these entities does. Askie Forum has two main types of entities: the host and the user. A host is an Askie Forum user who is holding a forum for an event such as a class lecture or a presentation at a meeting. These host-created forums shall be part of a database, which is why constructing the database is one of our top priorities. If we take a look back at the user interface diagram for the host from the last assignment, we can see that the header bar contains a dashboard with features that are available only to the host. These features includes: a list of users in the forum; a list of urgent questions; data analytics; settings; and a system that allows the host to end the forum. In terms of functional requirements, a host should be notified of all urgent questions as well as the number of users currently in the forum. This will allow the host to view who's attending the event, giving the host a reliable way to

provide credit to users attending the forum (this system will almost always be used in a classroom setting). It's also necessary to implement a live question and answer feature, which will allow the host to see the questions that are currently being asked. On top of the live questions and answer section, there will be an archive that will store the questions that have been resolved. The archive will also store collections of previous archives that will allow the host to go back to previous lecture questions and answers. In terms of non-functional requirements, the purpose of having a host is so that a forum can be created with a subject, and so that the host (who should be the user with the most knowledge of the subject) can answer questions from the forum. A host should also be provided with features such as Data Analytics, which will help them identify who has answered the most questions within their forum and which users have the highest ratings within their forum. From this data, the user will be able to provide incentives to those who contribute the most to their forum.

The other entity that Askie Forum will focus on is the Askie user. Askie users are presented a similar interface to Askie hosts, but will not have access to the administrative tools that are provided to the hosts. The main difference between users and hosts lies in their functional requirements, which will transfer over to the features that are given to each type of user while they are using Askie Forum's services. Some of these different features will be presented under the user's settings, where the host will get to decide on things such as how many red flags each user will have. These red flags - given by the host - represent the number of irrelevant questions a user may ask. The host will also be able to set a passcode that other users will need in order to join the host's forum. In contrast, non-host users' settings will only be able to set things like how urgent the user's questions are. From these similarities and differences, we will able to set up the packaging of our code that will best suit the site's functionality, as well as any potential future upgrades or maintenance work that we might perform.

2.1 Coupling

Since coupling refers to the relationship between a code unit and another code unit, let's discuss the similar features that both entities described above contain. Some of the similar features both entities contain are the Live Q&A board, the Q&A Archive, and a collection of past archives. These features are the most correlated mainly because of the purpose of the application. These features will appear perfectly similar to both entities. Features that may not be as similar include settings and data analytics. Because the settings provide both users and hosts with the ability to modify fonts, colors, and interface styling, they can be implemented similarly; however, the main functions that each will provide to the two type of users will be different, so they'll require additional code segments.

2.2 Cohesive

Functionally, most features of each entity are highly cohesive, as changing one feature will strongly affect the rest of the system. A good example of this is the Q&A Archive. Changing or deleting the archive would prevent users from viewing questions and answers from their peers, which would result in many questions being rehashed and vital information being lost. Without an archive, it's also very hard

for students or users who are clients of the forum to see the activities that they may have missed out on due to them not being able to attend the session. Changing the Q&A Archive features may also affect the collection of archives, which may lead to the inability for Askie users to look back on previous session and obtain information they may have missed out on.

3 Design Patterns

Due to the nature of this application (that of an online forum), there are several design patterns that will be put into use. User creation, post creation, post notifications, and post viewing are complex tasks that will each require specific patterns. In no specific order, they are as follows:

3.1 Observer

The Observer pattern will likely be used in the monitoring of the forums. This pattern will also be tied in to the notification process. The Observer pattern will be looking for new questions being posted by actors who have enrolled in the event. On detecting a new post, the pattern will perform one or more actions, depending on the question posted. At the very least, the posting of the question will trigger the main page to update, and allowing other actors to view the post. Depending on the operation mode, the pattern may trigger a notification to the Instructor that a user has posted a reply. This will happen in either the Q & A use case or the Participation Required use case.

3.2 Visitor

The Visitor pattern will be used to enumerate the questions whenever an actor selects either a forum or a question to view posts. Whenever a user clicks on an active event, this pattern will be used to list the posted questions. Similarly, the Visitor pattern will be used to list any responses others have posted to these questions.

3.3 Factory

The Factory pattern is used to build predefined entities within an application by means of a template system. User creation, post creation, and forum creation are all entities that follow a specific pattern, and are stored as such. Whenever an instructor finishes setting up an event, entries such as event name, event passcode, instructor name, and possibly others will be saved in a similar format every time. The same is true of either question or answer posts, as well as user account information. User account creation will benefit from this pattern in particular, as all relevant information must be readily accessible at any time.

4 Interfaces and Contract

THIS SECTION HAS BEEN VOIDED AS OF 2/8 BY PROFESSOR ALI.

5 Exceptions & Handling

One of the most common forms of an exception that the system is likely to face is an error communicating with the server due to an interruption in the Internet connection of a user's computer. In order to mitigate the problems caused by this event, the application must be able to wait for the connection to be re-established rather than crashing as soon as the connection fails. Therefore, the handler would have to pause the process while it checks for connectivity and then once the connection is established, it will release the process to go back to standard operation. The application will also need to continuously save data to the server while there is a connection so that if the Internet connection fails or the user's computer malfunctions then the data will not be lost. Another exception that is likely to occur is the possibility of trying to post to a forum that has been deleted while the user was composing their post. If the application tries to write to a file that no longer exists then it will crash so their must be handlers implemented to check if the file exists before attempting to write to it. If the file does not exist, then the handler should return an error message for the user indicating that the forum was deleted by another user while they were editing their post.

6 Meeting Report

Our primary focus this week was preparing for the presentation. Due to midterms, assignments for other classes, and other concerns, our group was unable to meet in person; however, we conducted an online meeting through our shared Discord server. During this meeting, we decided that the presentation slides and scripts should be worked on by the entire group as a whole. Each member contributed additional points to the slides pertaining to the topics they've been covering throughout the project's development, however. Apart from the presentation, we also made some other minor advances during this past week. In particular, the construction of the UML class diagram will make the coding process far easier, and the research done into potential exceptions and solutions should speed up the debugging process.

For this assignment, each section was worked on by individual group members. Andrew Morrill created the UML diagram, while Danh Nguyen wrote the section on Packaging. Nicholas Newell authored the section on Design Patterns, and Joshua Bell took care of the Exceptions and Handling segment. Cameron Kocher wrote the Meeting Report and was responsible for revising and editing the document and presentation.