iOS Native Comments MVP Outline

OVERVIEW

We built an iOS coral talk client custom to our use case. This document is meant to be a general overview of how we did that.

SPECIFICATIONS

- Comments need to viewable in-line with content - below articles and videos.
- Comments will be styled similar to IGN's web implementation of the comment stream
- Replies should be indented according to the depth of that reply.
- Reply threads need to be collapsible (animated).
- Post/Reply/Edit/Like/Report will be the only supported actions.
- User will be notified if their comment was not posted because of a banned word.
- The comment section will not be updated live from other users on the same stream. (A user on web who clicks `LIKE` won't update the comment view on the native client until the stream is reloaded)
- Moderation will not be available.
- Authentication should be handled via JSON web tokens and the normal app sign-in behavior.
- Users can sort comments by time/likes/replies.
- Supported elements include comments, infobox, question box, featured comment, closed thread.

SOLUTION

In order to show comments inline, as well as supporting animating collapsing reply threads, we decided to build native comments to work intimately with UITableView.

This also allows for animating the changes.

We wanted to abstract the nitty gritty of the tree-like nature of comments and their replies from the UITableView and it's ViewController, so we created a CommentManager that would speak the UITableView's language.

We've used Apollo in the past on web and we're pleased so we decided to use their iOS client for graph calls to the Coral backend.

TECHNOLOGIES

1. Apollo (Graph client)
2. JSON Web Token
3. UITableView

STEPS

RETRIEVING COMMENTS FROM CORAL

Our first step was to retrieve a comment stream for an asset from the coral backend. The coral back end uses GraphQL, so we needed to set up a GraphQL client. Our choice was Apollo.

Since I was new at building GraphQL queries, I would often proxy our web implementation of Coral talk, and use the queries that were being used in that implementation as the basis for my own. This was true for retrieving a stream of comments for an asset.

This proved relatively simple.

CONSTRUCTING OUR COMMENT TREE

We knew we'd be viewing the comments in a linear way, but the need to be able to expand and collapse replies meant we need to store the underlying reply tree in some way.

When building our comment models from the coral response, we stored each reply in an array within its parent comment model. This constructs a tree-like structure.

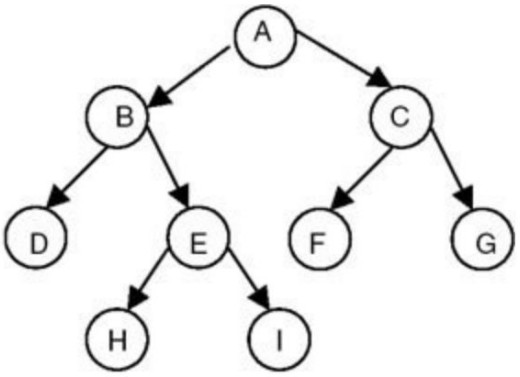
VIEWING OUR COMMENTS IN A TABLEVIEW

Since UITableView's represent data in linear way, we knew we'd need to abstract the tree-like structure of our comments from the UITableView in order to keep the code clean.

We found the best way to do this was to keep a proxy of the comment tree in a separate data structure, a simple array. We will refer to this array as the Linear Visible Tree.

To construct the Linear Visible Tree we would walk the comment tree using a Pre-Order traversal adding them to the Linear Visible Tree.

Consider the diagram below. Think of A as a root-level comment. B and C are replies to A, D and E are replies to B, and so on.



Inorder : DBHEIAFCG
Preorder : ABDEHICFG
Postorder : DHIEBFGCA

The Preorder result is exactly what you would expect seeing in a comment stream.

The UITableView could now use this Linear Visible Tree as a simple datasource.

We attached a depth to each comment, so each comment UITableViewCell could be indented to represent the reply depth of that comment.

This gave us a simple view of our comments in a UITableView, with a clear representation of what is a reply, and what comment it is in response to.

EXPANDING AND COLLAPSING REPLY THREADS

All that was needed for this was to add a property to each comment model `isExpanded=true`. And when construction the Linear Visible Tree, simply skip over and do not add the replies of comments that have `isExpanded == false` to the Linear Visible Tree.

Consider the diagram again - if B had `isExpanded==false` then the result should be `ABCFG`.

Notify the UITableView to remove the rows for `DEHI` and it's automatically animated for you.

EVERYTHING ELSE

We followed the same process for every other feature. Posting a comment? Proxy the web implementation, create similar GraphQL queries/mutations, update the comment tree and the resulting Linear Visible Tree, update the UITableView.