Foolish Demo Design Considerations/Compromises

Thank you for the opportunity to complete this project. Below are a few thoughts on the design and implementation of the Foolish Demo Django project:

1. **URL Slug** - The Django slugify() method is used to convert article headlines to slugs for both URLs and as the key to link the articles in the database with those in the JSON data. This results in somewhat long URLs. The Django slugify() method is wrapped in another function to simplify updating the slugification method in the future if it is desirable to drop articles or other less necessary words to create a shorter slug. There is the potential that two separate articles may have the same name but that was considered unlikely. A more foolproof method would be to include the article date with the headline to create the slug, or simple hash the article to create an identifier that's almost guaranteed to be unique, though ugly.

2. **Django Models** - I was a little torn on whether or not to pull the quotes and content data into a database or leave it as JSON. The default answer was to import into a DB, but it seemed like there should be a reason for it. The deciding question for me was "would there be a clear benefit to the various fields and records in the data being in a model?" For me the answer is generally no. For the project criteria given, there is minimal benefit to being able to query most fields, there is no need to controlled edit access, and multiple users can easily view the data at the same time. Additionally, the data is provided as JSON and will likely be provided as JSON in the future. Further, there are a lot of fields in the content data, many of which will likely not benefit from the easy filtering tools provided by a model.

   However, it may be desirable in the future to add additional functionality that allows users to sort or filter articles by author or date or some other criteria, which would clearly benefit by representing the necessary data in a model. As a compromise, the articles are represented in the Articles model by their slug, with all other data remaining in the JSON file. All access to the JSON data is provided through the Articles model. This helps ensure that the views don't need to know whether data is in JSON or in the database and simplifies the effort required to add author, date, or other information to the model if desired.

   The load_articles() and slug_test() views do access the JSON data somewhat directly. As these functions were only used to assist development I felt comfortable breaking from the model-only access pattern.

   Stock quotes were left as pure JSON as it was anticipated they would need to be updated very frequently. This allows the file to be quickly swapped out, with the

assumption that it is not being read when the update is required, which may not be a valid assumption.

3. **Data import** - The articles were imported into the database using the load_articles() view. Using a view to update data would clearly need to be password protected in production and really isn't ideal. A better solution would be to use a Python script that can be automatically run whenever new data is available. The stub files load_articles.py and delete_old_articles.py were created in the articles directory but were not completed.

4. **Style** - I'm using style to refer to the design of the site as represented with CSS/HTML. Creating a beautiful style from scratch is one of my weak points and it should be obvious that the style was lifted pretty heavily from the example pages. I would have loved to spend more time on making things look pretty but chose to spend the time on the backend code instead.

5. **Comments** - Article comments were implemented using the "excontrib" comments library. The comment system does not allow anonymous comments as requested in the guidelines and redirects to an ugly thank you page. I would prefer to fix this but simply ran out of time.

6. **Tests** - The tests utilize the two JSON files for test data. Due to this, updating the JSON files in production would likely cause the tests to fail. Backups of the JSON files are stored in articles/test_data and used in the tests to prevent this issue.