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** - Django models are used to represent the articles, quotes, and article tags. Not all data is in the JSON files is represented within the models, however, it would be simple to update the models (and load_XXX() functions) to handle additional article/quote data as required.

   I previously made only minimal use of models for this project, instead leaving the majority of the data in the JSON files. This was done based on the argument that the quotes and articles would need to be updated regularly and it would be simpler to keep the base data in the JSON file. Unfortunately, if the site were to gain any traffic, it might be difficult to replace the JSON files as they might be locked by the file system. Furthermore, any expansion of the site requiring sorting or filtering the articles and quotes by any criteria would be greatly assisted by having the data stored in a model.

3. **Data import** - The articles and quotes are imported into the database using the load_articles() and load_quotes() views. 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, load_quotes.py, and delete_old_articles.py were created in the articles directory but are not complete.

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.