Framework

I chose to write the code for the API server in Python.
My background has primarily been in Java. I thought this would be a good opportunity to demonstrate my ability to work in something new.

Along the lines of working with something new. Was the choice of framework for implementing the API. The two main choices were between Flask and Django Restframework.

I chose to go with Django Restframework. Not having worked with either framework I understood there would be a bit of a learning curve with either.
While Flask is supposed to be a bit more lightweight and easier to learn. I chose to go with Django Restframework. It appeared to be a bit more enterprise ready and had good integrations for authentication and databases with its ORM. It felt like a good choice for quickly putting together a robust API.

Being newer to Python and really new to Django and DRF. I did go back and forth a bit on how the project should be structured.

Testing

For simplicity chose to stick with using the Django Test Runner for running the unit and integration tests.

Created unit tests for the model, views and serializers

Created integration test for the users and posts API.

Covered a good number of test cases, but definitely did not hit all test cases.

Deployment

Included are instructions for running the API server locally within the development environment.

Also included are instructions for running the API server locally in Docker using Docker Compose. This will launch the PostgreSQL database to serve as the storage backend for the API. The Docker Compose script manages the connection between the API server and the database, and handles running migrations, including pre-populating the server with test data.

I had worked on being able to deploy to a local Kubernetes Cluster.   But when I went back to retest the deployment, I had some network issue being able to connect local cluster without using port forwarding so I decided to not include that. I was spending too much time trying to figure out what was wrong with the NodePort configuration

The yaml files for the setup are included with the code in a /k8s/ directory off the root.

Documentation

My original plan was to use Swagger and OpenAPI to auto generate the documentation. While I was able to get it mostly working there were some formatting issue, that made me thing it was not worth the extra time, so I backed it out of the code. This is partly due to getting up to speed with Python, Django and Django Restframework.

I included a README-api.md in the root directory of the code that has documentation on the API.

Possible improvements

While working through the API, testing and documentation. There were a few things that crossed my mind that would have been nice adds if there were a bit more time. Things that would be more necessary in production.

The would include

- Pagination. Add query parameters page and page_size. Return metadata about results like total_count, in addition to the list of objects.
- Add an endpoint that would return all the posts for a given user.
- Authentication. Potentially adding roles that control which users can create, edit and view posts. As well as roles that would control being able to create and edit user information of someone other than the current user.
- Potentially adding an expandUser query parameter to the posts endpoint to embed the User details of the user, and not just the user_id.   The query parameter would give this some control over when to expand the user.   Some performance considerations if this were to be implemented on an endpoint returning a large number of users.
- Updating posts. Do we want better control on who can change the user_id of a given post. This is something that could be tied back in with authentication.