Questions:

Views: http://127.0.0.1:8000/admin

http://127.0.0.1:8000/hello/

**Data Generation:** For generating significant amount of data to simulate the intended scale of the application we shall use Factory Boy. It will generate all our dummy data. It's a library that's built for automated testing, but it also works well for this use-case. Factory Boy can easily be configured to generate random but realistic data like names, emails and paragraphs by internally using the [Faker](https://faker.readthedocs.io/en/master/) library.

**Test.py:** It is created in the app directory but file is not prepared for time constrain.

**Requirement.txt:** It is provided in project directory.

**Strategy for Scalable Architecture and Deployment:**

**Database engine:** Postgresql is the preferred database for Django applications due to its open-source nature; and it's also ideal for complex queries

**efficient querying:-**

* **Use select\_related and prefetch\_related to optimize database queries and reduce the number of database hits when fetching related objects.**
* **Avoid using SELECT \* in queries; only fetch the columns you need.**
* **Use the defer() and only() methods to limit the fields fetched from the database.**
* **Utilize database-level optimizations like database views, stored procedures, and functions where appropriate.**
* **Implement caching mechanisms to store frequently accessed data in memory. Django provides a caching framework that supports various backends like Memcached and Redis.**
* **Use bulk insert and update operations to reduce the number of database queries. The bulk\_create() and update() methods in Django can be more efficient when dealing with large amounts of data.**
* **Database Denormalization**
* **Database Sharding**
* **Database Maintenance:**

**Containerization: Through Docker**

Create a requirements.txt

Build the Docker Image

Run the Docker Container

Create a Docker File

**Orchestration: Through Kubernetes.**