1. Coding Design for the QnA System

Components

1. **Data Ingestion and Preprocessing:**
   - **Load and preprocess the dataset** to ensure it is in a format suitable for querying. This includes handling missing values, normalizing text, and possibly embedding the text data.
2. **Retrieval-Augmented Generation (RAG) Model:**
   - **Retrieval:** Use embeddings and a search index to retrieve relevant documents.
   - **Generation:** Use a generative model to formulate answers based on the retrieved documents.
3. **Indexing:**
   - **Efficient Search:** Use vector databases like FAISS or Annoy for scalable and efficient similarity search.
   - **Document Embeddings:** Convert text into embeddings to facilitate quick retrieval.
4. **API Implementation:**
   - **Create an API** that allows querying the system and returns answers based on the dataset.
5. **Scalability:**
   - **Load Balancing:** Use load balancers to distribute incoming requests across multiple instances.
   - **Caching:** Cache frequently accessed data to reduce load times and improve response speed.
   - **Horizontal Scaling:** Deploy the application on a cloud platform with auto-scaling capabilities.

```
import pandas as pd

import os

# Load the dataset

data_path =
'/content/drive/MyDrive/path/to/your/projects_with_embeddings.csv'

 data = pd.read_csv(data_path)
```

```
# Preprocess the data

 data.fillna ("", inplace=True) data['text'] = data.apply(lambda row:
f"{row['project_name']} - {row['Unit type’]} - Price: {row['price']}",
axis=1)

# Convert to a list of documents

documents = data.to_dict(orient='records')

 texts = data['text'].tolist()
```

Creating an Index

Use a vector database like FAISS for indexing and retrieving documents.

```
import faiss

import numpy as np from sentence_transformers

 import SentenceTransformer # Initialize model

model = SentenceTransformer('all-MiniLM-L6-v2')

# Convert texts to embeddings

embeddings = model.encode(texts)

# Create FAISS index

dimension = embeddings.shape[1]

 index = faiss.IndexFlatL2(dimension)

 index.add(embeddings)
```

Building the API

Use a web framework like FastAPI to build and expose the API.

```
from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

import numpy as np
import faiss

app = FastAPI()

class Query(BaseModel):

 question: str

@app.post("/query")

 def query_system(query: Query): # Convert query to embedding

query_embedding = model.encode([query.question])

# Retrieve documents

distances, indices = index.search(query_embedding, k=5)

 retrieved_docs = [documents[idx] for idx in indices[0]]

answer = " ".join([doc['text'] for doc in retrieved_docs])

return {"answer": answer}
```

Testing and Latency Reporting

Functionality Testing:

1. **Test Endpoints:**
   - Ensure that API endpoints work correctly with various queries.
   - Use tools like Postman or `curl` to manually test.
2. **Automated Tests:**
   - Write test cases to validate different scenarios and edge cases.

Performance Testing:

1. **Simulate Load:**
   - Use tools like `Apache JMeter` or `locust` to simulate multiple concurrent requests.
2. **Measure Latency:**
   - Record response times for different loads and configurations.

Example

```
from locust import HttpUser, TaskSet, task, between

class UserBehavior(TaskSet):

@task

def query_api(self):

self.client.post("/query", json={"question": "What is the price of Project A?"})

class WebsiteUser(HttpUser):

tasks = [UserBehavior]

wait_time = between(1, 5)
```