AI-Powered Kolam Generator — Q&A Document for Presentation

What is the project about?

The project is a Streamlit web app that generates traditional Indian Kolam designs using Google's Gemini generative AI model. Users input design parameters, and the AI produces drawing instructions which are then visualized step-by-step.

Which technologies are used?

- **Streamlit**: For building the interactive web user interface.
- **Google Gemini (Generative AI)**: Used as the core AI model to generate Kolam drawing instructions in JSON.
- **Matplotlib**: For incremental drawing and visualization of the Kolam.
- **python-dotenv**: For secure storage and loading of environment variables (API keys).
- **Google Generative AI Python SDK**: To call the Gemini API.

How does the Kolam generation work?

Users choose parameters like festival type, motif, symmetry, complexity, and grid size. These feed into a natural language prompt sent to the Gemini model. Gemini responds with JSON specifying coordinates of dots, curves, motif positions, and symmetry type. The app parses this and sequentially draws:

- Dots as points
- Curves connecting the dots
- Motifs as symbols at specific positions

Each step updates the plot visually on the same canvas.

How is the Gemini model integrated and configured?

- The `google.generativeai` Python SDK is initialized with an API key securely loaded from `.env`.
- The model selected is `"gemini-2.5-flash"`.
- Requests specify the prompt with Kolam parameters and request JSON output.
- Responses are parsed and handled gracefully to display errors if present.

What is the role of incremental rendering?

The app uses time delays and a Streamlit placeholder to create an animated effect, drawing dots first, then curves, then motifs. This makes the Kolam construction engaging and visually informative rather than showing the final result instantly.

How is error handling implemented?

- If the JSON response parsing fails, an error message with the raw response is shown.
- The app stops further execution on such failure to prevent crashes.
- This ensures that user feedback is immediate when the AI output is malformed.

How can this project be deployed?

- Install the required dependencies.
- Store the Google API key in a `.env` file.
- Run the app locally with `streamlit run app.py`.
- For cloud deployment, ensure environment variables are configured securely on the host.

What are some potential improvements?

- Add caching of generated Kolam instructions to avoid repeated API calls.
- Implement user accounts for saving favorite Kolam designs.
- Support for exporting Kolam to SVG or image files.
- Enhance prompt engineering for more elaborate and context-aware designs.
- UI improvements for better responsiveness and mobile support.

What challenges were encountered?

- Parsing JSON safely from AI-generated output due to possible formatting variability.
- Balancing user interface responsiveness with incremental plotting and delays.
- Managing API key security and environment configurations.
- Designing flexible prompts that cover multiple Kolam style parameters accurately.

What are the key takeaways?

- Integrating LLMs with graphical visualization creates engaging creative applications.
- Proper prompt design and error handling are critical for AI-driven generation.
- Streamlit is highly effective for quick prototyping of interactive ML-powered apps.
- The project bridges traditional art forms with modern AI and web tech.