Streamlit test

Date : 21st March, 2025

1) What are the key features of Streamlit that make it suitable for data science and machine learning applications?

Streamlit is a great tool for building data science and machine learning applications because it is simple, fast, and requires minimal coding. It allows you to turn Python scripts into interactive web apps with just a few lines of code. One of its best features is that it automatically updates the app whenever you change the code, making it very convenient for quick iterations. It also provides built in support for interactive widgets like sliders, buttons, and text inputs, so there's no need to worry about designing a user interface from scratch.

2) How does Streamlit handle state management, and what are some ways to persist data across interactions?

In Streamlit, each user interaction triggers a rerun of the entire script, which can pose challenges for maintaining state across interactions. To address this, Streamlit offers the `st.session_state` feature, allowing developers to store variables that persist across reruns during a user's session. This is particularly useful for retaining user inputs or selections. For scenarios requiring data persistence beyond individual sessions, integrating external databases or leveraging caching mechanisms can be effective solutions.

3) Compare Streamlit with Flask and Django. In what scenarios would you prefer Streamlit over these traditional web frameworks?

When comparing Streamlit with Flask and Django, Streamlit is much easier to use for building data apps because it requires less code and focuses on interactive visualizations. Flask and Django are general-purpose web frameworks, making them better for complex applications that require user authentication, advanced routing, or backend logic. Streamlit is tailored specifically for data-centric applications, enabling rapid development of interactive data apps with minimal code. This makes Streamlit particularly advantageous for data scientists and machine learning practitioners who need to create and share data visualizations, dashboards, or model interfaces without delving into the complexities of traditional web development.

Streamlit test

Date : 21st March, 2025

4) Describe the role of caching (@st.cache_data and @st.cache_resource) in Streamlit. How does it improve performance?

Caching in Streamlit helps improve performance by preventing expensive computations from running multiple times. @st.cache_data is used for caching functions that process or fetch data, while @st.cache_resource is used for caching resources like database connections or machine learning models. This reduces execution time and improves the app's responsiveness, especially when dealing with large datasets or complex computations.

5) How can you integrate a database with a Streamlit app? Provide an example using SQLite or PostgreSQL.

Streamlit can connect to databases like SQLite and PostgreSQL for storing and retrieving data. SQLite is lightweight and ideal for small projects, while PostgreSQL is better for large-scale applications.

```
import sqlite3
import streamlit as st

conn = sqlite3.connect('users.db')
cursor = conn.cursor()
cursor.execute('CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)')
conn.commit()

st.title("User Management")
name = st.text_input("Enter name:")
if st.button("Add User"):
    cursor.execute('INSERT INTO users (name) VALUES (?)', (name,))
    conn.commit()
    st.success(f'User {name} added!')

st.write("Users:", cursor.execute("SELECT * FROM users").fetchall())
```

This creates a simple database to store user names.

Streamlit test

Date : 21st March, 2025

6) Discuss how you can deploy a Streamlit application. Mention at least two deployment platforms.

Deploying a Streamlit application can be achieved through various platforms:

1. **Streamlit Community Cloud**: This is Streamlit's own deployment platform, offering a straightforward way to share apps. Developers can deploy directly from a GitHub repository, and the platform handles the necessary setup and hosting.
2. **Heroku**: A popular cloud platform that supports deploying Streamlit apps by configuring a Procfile and specifying dependencies in requirements.txt. Heroku manages the infrastructure, allowing for scalable and reliable app hosting.

7) What are some limitations of Streamlit, and how can you overcome them when building production-grade applications?

While Streamlit excels in rapid development and simplicity, it has certain limitations:

- **State Management**: Maintaining complex state across user interactions can be challenging due to Streamlit's rerun behavior. Utilizing `st.session_state` and designing apps with stateless principles can mitigate this issue.
- **Customization**: Streamlit's predefined components may not offer the level of customization required for intricate user interfaces. Integrating HTML, CSS, and JavaScript through components can enhance the app's appearance and functionality.
- **Scalability**: For high-traffic applications, Streamlit's performance may be a concern. Deploying behind a reverse proxy, implementing caching, and optimizing code can improve scalability.

Streamlit test

Date : 21st March, 2025

8) Explain the process of creating an interactive dashboard in Streamlit. What components would you use?

To build an interactive dashboard in Streamlit:

1. **Install & Setup**: Install with pip install streamlit and run using streamlit run app.py.
2. **Import Libraries**: Load streamlit, pandas, and visualization tools like matplotlib or plotly.
3. **Load Data**: Use pandas to read and preprocess data (pd.read_csv() or from a database).
4. **Design Layout**: Use st.sidebar for filters, st.title() for headers, and st.columns() for sections.
5. **Interactive Widgets**: Add sliders (st.slider()), buttons (st.button()), and dropdowns (st.selectbox()).
6. **Visualizations**: Use st.line_chart(), st.bar_chart(), or st.pyplot() for graphs.
7. **Tables & Metrics**: Show data with st.dataframe(), highlight insights using st.metric().
8. **Media Support**: Embed images (st.image()) or videos (st.video()).

This results in an intuitive, real-time, interactive dashboard for data analysis.

9) How would you implement user authentication in a Streamlit app? Provide possible solutions.

User authentication can be added using:

1. **Streamlit-Authenticator**: A package for login and password management.

```
import streamlit_authenticator as stauth
authenticator = stauth.Authenticate(names, usernames, passwords, 'cookie_name', 'signature_
name, status, username = authenticator.login('Login', 'main')
if status: st.write(f'Welcome {name}')
```

2. **OAuth2 Integration**: Use Google, GitHub, or Okta for authentication.
3. **Custom Authentication**: Manage user sessions using st.session_state.

For production, OAuth2 ensures better security.

Streamlit test

Date : 21st March, 2025

10) Describe a real-world use case where you have implemented or would implement a Streamlit application.

A real-world use case for Streamlit is building mini projects and major projects and it is a great too for the deployment of these projects. We could easily build applications using Streamlit. One real-world example of Streamlit is when I used it to make an application which has a title, a value slider which could be moved by the user, a button which could be clicked and it says button clicked when clicked, a display image with it's own image caption.

Streamlit is a fantastic choice for quickly building and deploying interactive data applications with minimal effort.