Streamlit技術仕様書

1. はじめに

2. Streamlitのアーキテクチャ

3. Streamlitのコンポーネント

4. Streamlit開発のベストプラクティス

5. Streamlitの使用ガイド

6. 結論

7. 参考文献

Streamlitは、Pythonのみを使用してインタラクティブなWebアプリケーションを迅速に構築するためのオープンソースフレームワークです。データサイエンティストや機械学習エンジニアが、データスクリプトを共有可能なWebアプリケーションに変換するプロセスを簡素化するために設計されました。このドキュメントでは、Streamlitのアーキテクチャ、主要コンポーネント、開発のベストプラクティス、およびアプリケーションの作成とデプロイに関する実用的なガイドを含む、Streamlitの技術的側面について詳しく説明します。

2.1 クライアント・サーバーアーキテクチャ

Streamlitアプリケーションは、クライアント・サーバー構造を持っています。アプリケーションのPythonバックエンドがサーバーであり、ブラウザを通じて表示されるフロントエンドがクライアントです[1]。ローカルでアプリケーションを開発する場合、コンピュータはサーバーとクライアントの両方を実行します。誰かがローカルネットワークまたはグローバルネットワークを介してアプリケーションを表示する場合、サーバーとクライアントは異なるマシンで実行されます。アプリケーションを共有またはデプロイする予定がある場合は、一般的な落とし穴を避けるために、このクライアント・サーバー構造を理解することが重要です。

`streamlit run your_app.py`コマンドを実行すると、コンピュータはPythonを使用してStreamlitサーバーを起動します。このサーバーはアプリケーションの頭脳であり、アプリケーションを表示するすべてのユーザーのために計算を実行します。ユーザーがローカルネットワークまたはインターネットを介してアプリケーションを表示するかどうかにかかわらず、Streamlitサーバーは、アプリケーションが`streamlit run`で初期化された1台のマシンで実行されます。Streamlitサーバーを実行しているマシンは、ホストとも呼ばれます。

誰かがブラウザを通じてアプリケーションを表示すると、そのデバイスはStreamlitクライアントになります。アプリケーションを実行または開発しているのと同じコンピュータからアプリケーションを表示する場合、サーバーとクライアントは偶然にも同じマシンで実行されます。ただし、ユーザーがローカルネットワークまたはインターネットを介してアプリケーションを表示する場合、クライアントはサーバーとは異なるマシンで実行されます。

Streamlitアプリケーションを構築する際には、以下の考慮事項に留意してください。

- Streamlitアプリケーションを実行またはホストするコンピュータは、すべてのユーザーのためにアプリケーションを実行するために必要な計算能力とストレージを提供する必要があり、同時接続ユーザーを処理するために適切にサイズ設定する必要があります。
- アプリケーションは、ユーザーのファイル、ディレクトリ、またはOSにアクセスできません。アプリケーションは、`st.file_uploader`のようなウィジェットを通じてユーザーがアプリケーションにアップロードした特定のファイルでのみ動作します。
- アプリケーションが周辺機器（カメラなど）と通信する場合、Streamlitコマンドまたはカスタムコンポーネントを使用して、ユーザーのブラウザを介してこれらのデ

バイスにアクセスし、クライアント（フロントエンド）とサーバー（バックエンド）間で正しく通信する必要があります。

- アプリケーションがPython以外のプログラムまたはプロセスを開いたり使用したりする場合、それらはサーバー上で実行されます。たとえば、ユーザーのためにブラウザを開くために`webbrowser`を使用したい場合がありますが、これはネットワーク経由でアプリケーションを表示する場合には期待どおりに機能しません。Streamlitサーバー上でブラウザが開き、ユーザーには見えません。
- デプロイでロードバランシングまたはレプリケーションを使用する場合、セッションアフィニティまたはスティッキネスがないと、一部のStreamlit機能は機能しません。

2.2 WebSocketとセッション管理

ほとんどのStreamlitアプリケーション開発者はWebSocketと直接やり取りする必要はありませんが、高度なデプロイ、カスタムコンポーネント、または大規模な接続管理には、その役割を理解することが重要です。

StreamlitのサーバーはTornado Webフレームワーク上に構築されており、WebSocketを使用してクライアントとサーバー間の永続的な双方向通信チャネルを維持します。この永続的な接続により、サーバーはリアルタイムの更新をクライアントにプッシュし、各ユーザーのセッションコンテキストを維持できます。各ブラウザタブまたはウィンドウは独自のWebSocket接続を作成し、アプリケーション内で個別のセッションを生成します。

大規模なデプロイや本番環境でのデプロイでは、ロードバランシングとサーバーレプリケーションが一般的です。ただし、Streamlitがセッションとローカル（サーバー）ファイルを処理する方法は、これらの環境で特別な考慮が必要です。クライアントがHTTP経由でメディア（画像や音声ファイルなど）を要求する場合、その要求にはセッションコンテキストが添付されません。複数のサーバーレプリカまたはロードバランサーがあるデプロイでは、メディアファイルのHTTP要求が、ユーザーのWebSocket接続およびセッション情報を処理しているサーバーとは異なるサーバーにルーティングされる可能性があります。メディアファイルがすべてのレプリカで利用できない場合、`MediaFileStorageError: Bad filename`のようなエラーが発生する可能性があります。ユーザーがファイルをアップロードできるコマンドも影響を受け、HTTPステータスコード400を発生させる可能性があります。

一般的に、この制限を解決または軽減するには、次のいずれかを実行できます。

- プロキシでセッションアフィニティ（スティッキネスとも呼ばれる）を有効にします。これにより、ユーザーのセッションからのすべての要求が同じサーバーインスタンスによって処理されるようになります。
- メディアをBase64エンコードされたデータURIに変換してからStreamlitコマンドに渡します。これにより、HTTP要求を介してアクセスされるStreamlitのメディアストレージを使用する代わりに、WebSocketを介してメディアデータが渡されます。
- 動的に生成されたファイルをサーバーレプリカの外部の安定した場所（S3ストレージなど）に保存し、外部URLをStreamlitコマンドに渡します。これにより、Streamlitのメディアストレージが回避されます。

セッションアフィニティを有効にすることは、メディアファイルとアップロードされたファイルの両方の制限を解決する一般的なソリューションです。構成の詳細については、デプロイプラットフォームのドキュメントを参照してください。

Base64エンコードされたデータURIまたは外部ファイルストレージを使用すると、メディアファイルの制限を簡単に解決できますが、ファイルアップロードの制限を解決するための完全なソリューションではありません。詳細については、GitHub issue #4173を参照してください。

[1] Understanding Streamlit's client-server architecture. Streamlit Docs. https://docs.streamlit.io/develop/concepts/architecture/architecture [2] GitHub issue #4173. https://github.com/streamlit/streamlit/issues/4173

3. Streamlitのコンポーネント

Streamlitのコンポーネントは、Streamlitで可能なことを拡張するサードパーティモジュールです。これらはコミュニティによって構築され、コミュニティのために提供されています[3]。コンポーネントは、Streamlitアプリケーションの構成要素であり、ユーザーインターフェースを提供し、ユーザーがアプリケーションと対話できるようにします[4]。

Streamlitの組み込みウィジェット（`st.slider`, `st.button`, `st.text_input`など）に加えて、開発者はカスタムコンポーネントを作成して、Streamlitの機能を拡張できます。カスタムコンポーネントは、静的コンポーネント（一度レンダリングされ、Pythonによって制御される）または双方向コンポーネント（Streamlit Pythonスクリプトからデータを受信し、データを送信できる）のいずれかです[5, 6]。

3.1 主要なコンポーネントカテゴリ

Streamlitコンポーネントは、さまざまなカテゴリに分類できます。以下に主要なカテゴリといくつかの例を示します。

カテゴリ	説明	例
LLM	大規模言語モデルとの統合を可能にするコンポーネント。	`streamlit-chat`
ウィジェット	ユーザー入力を受け付け、アプリケーションの状態を制御するためのインタラクティブな要素。	`streamlit-option-menu`
チャート	データの視覚化を生成するためのコンポーネント。	`streamlit-echarts`, `streamlit-folium`
オーディオ	音声関連の機能を提供するコンポーネント。	`streamlit-webrtc`, `streamlit-audiorec`
テキスト	テキスト処理と表示を強化するコンポーネント。	`st-annotated-text`
その他	上記のカテゴリに分類されない、さまざまな機能を提供するコンポーネント。	`pygwalker`, `streamlit-aggrid`, `streamlit-elements`, `streamlit-extras`, `streamlit-drawable-canvas`, `streamlit-agraph`, `st-pages`, `streamlit-pandas-profiling`, `streamlit-pydantic`, `streamlit-shadcn-ui`

3.2 カスタムコンポーネントの作成

カスタムコンポーネントを開発する最初のステップは、静的コンポーネントを作成するか、双方向コンポーネントを作成するかを決定することです[5]。Streamlitコンポーネントは、Streamlit Pythonスクリプトからデータを受信し、データを送信できます[6]。これにより、開発者はStreamlitの組み込み機能では不可能な独自のインタラクションと視覚化を作成できます。

カスタムコンポーネントの作成プロセスには、通常、以下の手順が含まれます。

1. **フロントエンドの構築**: React、Vue、または純粋なJavaScriptなどのWebテクノロジーを使用して、コンポーネントのUIとロジックを構築します。
2. **Pythonラッパーの作成**: Streamlitアプリケーションからフロントエンドコンポーネントと通信するためのPythonラッパーを作成します。
3. **データの送受信**: `st.component.v1.declare_component`などのStreamlit APIを使用して、PythonとJavaScriptの間でデータを渡します。

カスタムコンポーネントは、Streamlitアプリケーションの機能を大幅に拡張し、特定のユースケースに合わせてカスタマイズされたソリューションを可能にします。

[3] Components. Streamlit. https://streamlit.io/components [4] Everything You Need to Know about Streamlit Components. Kanaries. https://docs.kanaries.net/topics/Streamlit/streamlit-components [5] Intro to custom components. Streamlit Docs. https://docs.streamlit.io/develop/concepts/custom-components/intro [6] Create a Component. Streamlit Docs. https://docs.streamlit.io/develop/concepts/custom-components/create

4. Streamlit開発のベストプラクティス

大規模なStreamlitアプリケーションを開発する際には、保守性とスケーラビリティを確保するために、特定のベストプラクティスに従うことが重要です。これらのプラクティスは、コードの構造化、セッション状態の管理、および一般的な開発課題への対処に役立ちます。

4.1 コードのモジュール化

大規模なStreamlitアプリケーションでは、コードをモジュール化することが不可欠です。これにより、可読性と保守性が向上します。典型的なプロジェクト構造は次のようになります[7]。

```
project/
│
├── app_core/ # バックエンドロジックとSQLAlchemy関連のコード
│ ├── database.py # データベース接続とモデル
│ ├── queries.py # SQLクエリとヘルパー
│ └── utils.py # ユーティリティ関数
│
├── pages/ # フロントエンド関連のコード
│ ├── home.py # アプリのホームページ
│ ├── settings.py # 設定ページ
│ └── reports.py # レポート生成ページ
│
└── main.py # Streamlitアプリを実行するためのメインエントリポイント
```

この構造により、各コンポーネントが明確に分離され、アプリケーションのさまざまな部分を独立して開発およびテストできます。

4.2 セッション状態の慎重な処理

Streamlitでセッション状態を操作する際には、以下のベストプラクティスに従ってください[7]。

- **一意の変数名を使用する**: アプリケーションの複数の部分がセッション状態と対話する場合に、競合を避けるために一意の変数名を使用します。同じ変数名が異なるコンテキストで表示される可能性がある場合は、インデックスまたは識別子を追加することを検討してください。
- **セッション状態に変数が存在するかどうかを確認する**: 使用する前に、セッション状態に変数が存在するかどうかを確認します。例えば:

  `python if 'my_variable' not in st.session_state: st.session_state['my_variable'] = []`

- **ウィジェットの状態を保存する**: チェックボックス、スライダー、テキスト入力などのウィジェットの状態をセッション状態を使用して保存し、アプリの再実行間で値を保持します。

セッション状態を適切に管理することは、インタラクティブで応答性の高いStreamlitアプリケーションを構築するために不可欠です。

4.3 繰り返しタスクのヘルパー関数の作成

大規模なアプリケーションでは、繰り返しタスクをヘルパー関数に抽象化する必要があります。例えば、アクションボタン付きのデータ行を表示することは、再利用可能な関数に

抽象化できます[7]。

```
def display_row_with_buttons(data):
for idx, row in data.iterrows():
col1, col2 = st.columns([1, 3])
col1.write(row['column_name'])
col2.button(f"Action {idx}", key=f"button_{idx}")
```

このアプローチにより、コードがDRY（Don't Repeat Yourself）に保たれ、可読性が向上します。

4.4 動的コンテンツのループの使用

動的コンテンツ（データ行や複雑なフォームなど）を表示する必要がある場合、ループを使用するとロジックを簡素化できます。例えば、チェックボックス付きのグループ化されたデータを表示する場合[7]。

```
group_names = ['Group 1', 'Group 2', 'Group 3']
selected_groups = []
for group in group_names:
if st.checkbox(f'Select {group}'):
selected_groups.append(group)
```

論理条件（AND/OR）によるグループ化を処理する場合、競合を避けるためにセッション状態を慎重に管理します。これには、選択を単一のセッション状態変数に結合し、ユーザーの選択に基づいてロジックを実装することが含まれる場合があります。

4.5 SQLAlchemyとバックエンド統合

StreamlitをSQLAlchemyと統合してデータベース操作を行う場合[7]。

- `app_core`モジュールでデータベースモデルとCRUD（作成、読み取り、更新、削除）操作を定義します。
- SQLAlchemyクエリを使用してデータベースと対話し、結果をフロントエンドに渡します。
- 一般的なクエリやデータ変換を処理するためのユーティリティ関数を作成し、必要に応じてページで使用することを検討してください。

SQLAlchemy統合の例：

```
from app_core.database import session, MyModel

def get_data_from_db():
return session.query(MyModel).all()

def insert_data_to_db(data):
new_data = MyModel(name=data)
session.add(new_data)
session.commit()
```

4.6 エラー処理とデバッグ

Streamlitには組み込みのエラー処理メカニズムがありません。特にデータベース接続、セッション状態、またはユーザー入力を使用する場合は、エラーを適切に処理することが重要です。必要に応じて、常に `try-except` ブロックを使用してください[7]。

[7] Best Practices for Streamlit Development: Structuring Code and Managing Session State. Medium. https://medium.com/@jashuamrita360/best-practices-for-streamlit-development-structuring-code-and-managing-session-state-0bdcfb91a745

5. Streamlitの使用ガイド

Streamlitは、Pythonスクリプトをインタラクティブなwebアプリケーションに変換するプロセスを簡素化するように設計されています。このセクションでは、Streamlitアプリケーションの作成と実行に関する基本的な手順と重要な概念について説明します。

5.1 アプリケーションの作成と実行

Streamlitアプリケーションを作成する最初のステップは、Pythonスクリプトを作成することです。例えば、`uber_pickups.py` という名前のファイルを作成し、必要なライブラリをインポートし、アプリケーションのタイトルを追加します[8]。

```
import streamlit as st
import pandas as pd
import numpy as np

st.title('Uber pickups in NYC')
```

スクリプトを作成したら、コマンドラインから以下のコマンドでStreamlitアプリケーションを実行できます。

```
streamlit run uber_pickups.py
```

アプリケーションは自動的にブラウザの新しいタブで開きます。スクリプトに変更を加えて保存するたびに、StreamlitのUIはアプリケーションを再実行して変更を表示するかどうかを尋ねます。これにより、迅速なインタラクティブなループで作業できます。

5.2 データの読み込みとキャッシュ

Streamlitアプリケーションでは、データの読み込みは一般的なタスクです。大規模なデータセットを扱う場合、アプリケーションのパフォーマンスを最適化するためにデータをキャッシュすることが重要です。Streamlitは、`@st.cache_data`デコレータを使用してデータのキャッシュをサポートしています[8]。

```
 DATE_COLUMN = 'date/time'
DATA_URL = ('https://s3-us-west-2.amazonaws.com/'
            'streamlit-demo-data/uber-raw-data-sep14.csv.gz')

@st.cache_data
def load_data(nrows):
    data = pd.read_csv(DATA_URL, nrows=nrows)
    lowercase = lambda x: str(x).lower()
    data.rename(lowercase, axis='columns', inplace=True)
    data[DATE_COLUMN] = pd.to_datetime(data[DATE_COLUMN])
    return data

data_load_state = st.text('Loading data...')
data = load_data(10000)
data_load_state.text("Done! (using st.cache_data)")
```

`@st.cache_data`デコレータは、関数の入力パラメータとコードが変更されていない限り、関数の結果をキャッシュします。これにより、アプリケーションの再実行時にデータの再読み込みと再処理が不要になり、パフォーマンスが大幅に向上します。

5.3 データの表示と視覚化

Streamlitは、さまざまな方法でデータを表示および視覚化するための豊富な機能を提供します。`st.write()`関数は、データフレーム、チャート、文字列など、ほとんどすべてのものをレンダリングできます[8]。

```
 st.subheader('Raw data')
st.write(data)

st.subheader('Number of pickups by hour')
hist_values = np.histogram(
    data[DATE_COLUMN].dt.hour, bins=24, range=(0,24))[0]
st.bar_chart(hist_values)
```

より具体的な表示には、`st.dataframe()`や`st.bar_chart()`などの専用コマンドを使用できます。また、`st.map()`を使用して地理空間データをプロットすることもできます。

5.4 インタラクティブなウィジェット

Streamlitのウィジェットは、アプリケーションをより動的にするための優れた方法です。データフィルタリング、アクションのトリガーなどに使用できます。例えば、`st.slider()`を使用してデータを時間でフィルタリングできます[8]。

```
 hour_to_filter = st.slider('hour', 0, 23, 17)
filtered_data = data[data[DATE_COLUMN].dt.hour == hour_to_filter]

st.subheader(f'Map of all pickups at {hour_to_filter}:00')
st.map(filtered_data)
```

これにより、ユーザーはスライダーを操作して表示されるデータをリアルタイムで変更でき、アプリケーションのインタラクティブ性が向上します。

[8] Create an app. Streamlit Docs. https://docs.streamlit.io/get-started/tutorials/create-an-app

6. 結論

Streamlitは、Pythonのシンプルさと Webアプリケーション開発の強力な機能を組み合わせることで、データアプリケーションの構築方法に革命をもたらしました。その直感的なAPI、コンポーネントのエコシステム、および効率的なクライアント・サーバーアーキテクチャにより、データサイエンティストや開発者は、最小限の労力でインタラクティブで視覚的に魅力的なアプリケーションを迅速に作成できます。本技術仕様書で概説されているベストプラクティスと使用ガイドラインに従うことで、ユーザーはStreamlitアプリケーションのパフォーマンス、保守性、およびスケーラビリティを最大化できます。Streamlitは、プロトタイピング、データ探索、または本番環境でのデプロイのいずれにおいても、データ駆動型ソリューションを現実のものにするための貴重なツールとして機能します。