Creating and Deploying Streamlit Applications: Essential Tips

<https://medium.com/python-in-plain-english/creating-and-deploying-streamlit-applications-essential-tips-aef69039fc05>

<https://discuss.streamlit.io/t/streamlit-deployment-guide-wiki/5099>

<https://docs.streamlit.io/library/advanced-features/https-support>

Streamlit has become the tool of choice for many data scientists wanting to quickly create interactive and user-friendly web applications. This open-source platform allows you to transform Python scripts into elegant web applications, without requiring in-depth knowledge of web development.

스트림릿은 파이썬 코드를 웹 어플리케이션 코드로 변환하는 플랫폼이다.

However, to take full advantage of Streamlit and ensure the success of your application projects, it is essential to know best practices and understand the intricacies of the creation and deployment process. In this article, we will present a complete guide to creating and deploying Streamlit applications, focusing on key steps, best practices, and tips for making your applications powerful and user-friendly.

베스트 프랙티스

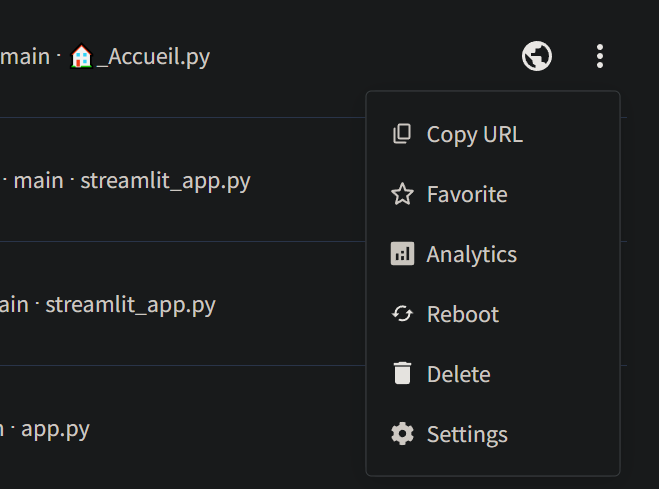
# Environment and secret key management

One of the essential first steps when developing a Streamlit application is environment management. Make sure your application works correctly with the specified dependencies and library versions.

To do this, follow these steps:

* Put only the packages needed for the application. This ensures that your application remains lightweight and avoids dependency conflicts.
* Use a requirements.txt file to specify the necessary dependencies. This makes it easier to create an isolated virtual environment with pip or conda.
* Don’t forget to “reboot” the application if certain changes are made to the requirements.txt file. This ensures that your application always uses the correct dependencies.

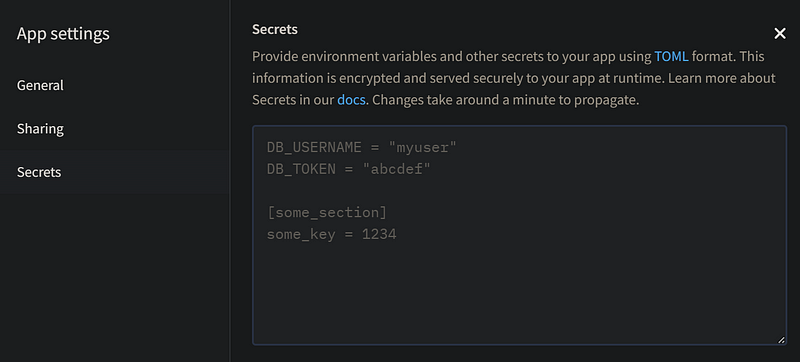
패키지 설치 후 반드시 재시작 하는 것을 추천한다.



## Secret Key Management

For security reasons, store secret keys and sensitive information securely, using environment variables. Here’s how you can do it:

민감한 정보는 환경 변수에 저장해라.



Then in the script you can retrieve the key with the following code:

# Using environment variables for secret keys

import os

secret\_key = os.getenv("SECRET\_KEY")

# Structuring the Application

To maintain a well-organized Streamlit application, consider the following points

## Organization in Multiple Scripts

* Divide your application into multiple scripts or modules for better code management.
* Project Tree: Organize your project using a logical folder tree. Here is a suggestion for a typical project structure:

# Project Structure: Organize your project using a logical folder structure.

# Here is a suggested typical project structure:

my\_streamlit\_app/

├── app.py # Main file, choose a meaningful name for multipage

├── requirements.txt # Dependency specification file

├── data/ # Folder for data files

│ ├── data.csv

│ └── ...

├── pages/ # Folder for different pages of the application

│ ├── 1\_app\_name.py # The name displays as-is (without "\_" and number)

│ ├── 2\_app\_name.py # Ensure to give a meaningful and concise name

│ └── ...

├── utils/ # Folder for reusable utility modules

│ ├── data\_loader.py

│ ├── plotting.py

│ └── ...

├── assets/ # Folder for static files (images, CSS, etc.)

└── README.md # Project documentation

* Modularity: Divide your code into separate modules or files to organize your application. For example, place data loading functions, data processing functions, and visualization functions in separate modules for better organization.

기능별로 별도의 파일로 만들어서 관리해라.

* Documentation: Comment your code appropriately and provide clear documentation to explain the purpose of each module, function, and variable. Use docstrings to document functions and classes.

적절한 문서화 doc string 활용

* Unit Tests: Write unit tests for your critical functions. Use libraries such asunittest orpytest to automate testing.

단위 테스트

* Configuration Management: If your application requires specific configurations (for example, database connection settings), use external configuration files (for example, a JSON or YAML file) to store these settings. Do not include this sensitive information directly in the source code.

외부 환경설정 파일을 활용해라.

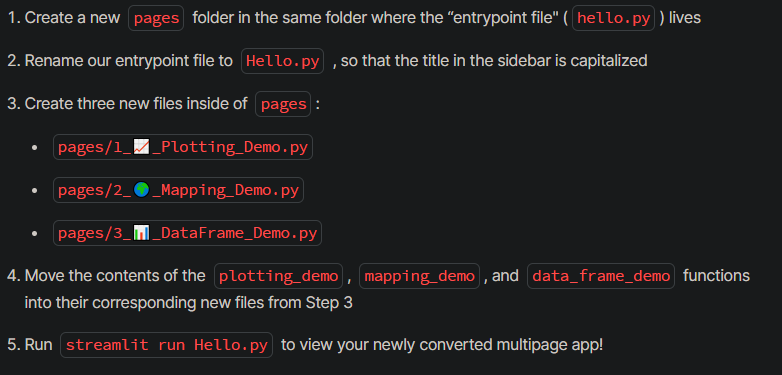
* Example Run: Include a file README.md describing how to run the application, install dependencies, and provide real-world examples of usage.
* Naming Convention: Follow a consistent naming convention for your files, variables and functions. This will make your code more readable and understandable.

코드 가독성

## Multipage

If you have multiple pages, use Streamlit’s “Multipage” feature for better organization. Simply put several applications in a “pages” subfolder, they will be automatically recognized.

pages 라는 하위 폴더를 생성하면 자동으로 인식한다.



## Multiple applications in a single repo

Here is a simple example of organizing files for a Streamlit project with multiple applications:

Projet\_Streamlit/

│

├── App1/

│ ├── app.py

│ ├── requirements.txt

│ └── (other files if necessary)

│

├── App2/

│ ├── app.py

│ ├── requirements.txt

│ └── (other files if necessary)

│

├── main.py

├── requirements.txt

└── (other files if necessary)

In this structure:

* Each application has its own folder (App1, App2, etc.).
* Each application folder contains a main file app.py which contains the Streamlit code specific to this application.
* A file requirements.txt in each application folder can contain dependencies specific to that application.
* The main filemain.py in the main directory can be used to select and launch different applications.

You may also have other files or folders depending on the specific needs of your project. This modular structure makes it easier to manage individual applications and makes the project extensible to add new features.

If you have multi-page applications in each application subfolder with sub-subfolders named “pages”, here is how you might organize your file structure:

Projet\_Streamlit/

│

├── App1/

│ ├── app.py

│ ├── requirements.txt

│ └── pages/

│ ├── page1.py

│ ├── page2.py

│ └── (other pages if necessary)

│

├── App2/

│ ├── app.py

│ ├── requirements.txt

│ └── pages/

│ ├── page1.py

│ ├── page2.py

│ └── (other pages if necessary)

│

├── main.py

├── requirements.txt

└── (other files if necessary)

In this structure:

* Each application (App1, App2, etc.) has its own folder.
* Each application folder contains a main file app.py which can be the entry point for this application.
* The folder pages in each application subfolder contains individual files for each page of that application (page1.py, page2.py, etc.).

When you deploy your application to share.streamlit.io, simply specify the full path to the main deployment script, including the subfolder, followed by the Python script name. For example, to deploy the App1 application, you would specify the path “App1/app.py”.

# Data and File Management

## Choice of data format

Use the Parquet format or a database (.db) to efficiently store and manage large amounts of data.

## Use of Related Links

Use relative links to reference files in your application.

Thus, the application can run locally as on a server without the code changing

상대 경로 사용

## Caching

Streamlit runs your script from start to finish with every user interaction or code change. This execution model makes development much easier. However, it presents two main challenges:

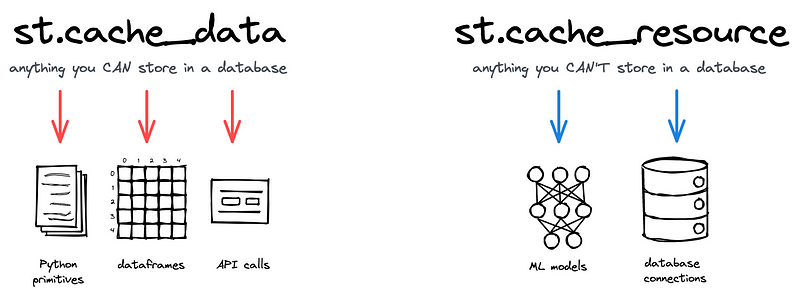
* Functions that take a long time to execute are re-executed each time, which slows down your application.
* Objects are recreated each time, making it difficult to persist between reruns or sessions.

But do not worry! Streamlit allows you to solve both of these problems with its built-in caching mechanism. Caching stores the results of slow function calls, so they only need to be executed once. This makes your application much faster and makes it easier to persist objects between reruns.

@st.cache\_data

def long\_running\_function(param1, param2):

return …



## Session Management

Use st.session\_state to manage user sessions and store data that needs to persist between different web pages.

# Example of using st.session\_state to store session data

if "user\_data" not in st.session\_state:

st.session\_state.user\_data = {}

st.session\_state.user\_data["username"] = "JohnDoe"

# Layout and Design

## Using sidebar

# Object notation

st.sidebar.[element\_name]

# "with" notation

with st.sidebar:

st.[element\_name]

A concrete example

import streamlit as st

# Using object notation

add\_selectbox = st.sidebar.selectbox(

"How would you like to be contacted?",

("Email", "Home phone", "Mobile phone")

)

# Using "with" notation

with st.sidebar:

add\_radio = st.radio(

"Choose a shipping method",

("Standard (5-15 days)", "Express (2-5 days)")

)

## Using Layout, Page Wide and Columns



* Customize your app layout with st.layout, st.page

# Example of using st.layout to customize layout

with st.layout("wide"):

# Votre contenu ici

* The use of st.columns

import streamlit as st

col1, col2, col3 = st.columns(3)

with col1:

st.header("A cat")

st.image("https://static.streamlit.io/examples/cat.jpg")

with col2:

st.header("A dog")

st.image("https://static.streamlit.io/examples/dog.jpg")

with col3:

st.header("An owl")

st.image("https://static.streamlit.io/examples/owl.jpg")

# Conclusion

By following these tips and organizing your Streamlit application in a structured way, you can create powerful and user-friendly interactive web applications. Streamlit offers great flexibility for application development, and by paying attention to these aspects, you can maximize the impact of your application on users.

Remember to document and test your application regularly to ensure its quality and usability. Happy Streamlit application development!