Perfect 🚀 — let’s move your app from “laptop-only with tunnels” → public, permanent, secure deployment on Streamlit Community Cloud with proper login.

**🌐 Deploy Streamlit App with Login on Streamlit Cloud**

**🔧 Step 1. Prepare Your Project**

* Put your code in a folder with at least:
  1. app.py (your Streamlit code)
  2. requirements.txt (all dependencies, e.g.)
     + streamlit
     + streamlit-authenticator
     + pandas
     + numpy
  3. secrets.toml for credentials (not pushed to GitHub).
* Push the project to GitHub (public or private repo).

**🔧 Step 2. Deploy to Streamlit Cloud**

1. Go to [share.streamlit.io](https://share.streamlit.io).
2. Sign in with GitHub.
3. Click New App → select your repo + branch + file (app.py).
4. Deploy → you’ll get a permanent URL like:

https://your-repo-name.streamlit.app

**🔧 Step 3. Add Login / Authentication**

You have a few options:

**Option A. 🔒 Streamlit Authenticator (simple username/password)**

import streamlit as st

import streamlit\_authenticator as stauth

# Example credentials (move these into .streamlit/secrets.toml for security)

names = ["John Smith", "Rebecca Briggs"]

usernames = ["jsmith", "rbriggs"]

passwords = ["abc123", "def456"]

hashed\_passwords = stauth.Hasher(passwords).generate()

authenticator = stauth.Authenticate(names, usernames, hashed\_passwords,

"myapp", "abcdef", cookie\_expiry\_days=1)

name, authentication\_status, username = authenticator.login("Login", "main")

if authentication\_status:

st.write(f"Welcome \*{name}\*")

st.button("Logout", on\_click=authenticator.logout)

elif authentication\_status == False:

st.error("Username/password is incorrect")

else:

st.warning("Please enter your username and password")

👉 You can define users/passwords securely in secrets.toml like:

[credentials]

usernames = ["jsmith", "rbriggs"]

passwords = ["abc123", "def456"]

**Option B. 🔑 OAuth Login (Google/GitHub)**

If you’d rather let testers log in with Google or GitHub:

* Use Streamlit’s experimental OAuth components or libraries like [Auth0](https://auth0.com/).
* Example with Google → testers click “Sign in with Google” → no passwords to share/rotate.

**🔧 Step 4. Share Safely**

* You now have a permanent HTTPS link.
* Testers log in with credentials (or their Google/GitHub).
* You can revoke/change access anytime in secrets.toml.

✅ Result:

* No tunnels.
* No random rotating links.
* Built-in login.
* Works 24/7 from your iPhone (or anywhere).

👉 Do you want me to write you a ready-to-deploy GitHub template repo (with Streamlit + Authenticator + secrets.toml already set up) so you just copy your code in and deploy?

Awesome 🙌 Let’s build a mini working demo that shows:

1. Your app connects to a database (we’ll simulate it with dummy credentials in [database]).
2. Your testers log in with their own usernames/passwords from [credentials].
3. The two things stay separate but complementary.

**📝**

**secrets.toml**

[database]

user = "db\_admin"

password = "db\_secret\_pass"

host = "db.myapp.com"

[credentials]

usernames = ["jsmith", "rbriggs", "adjikeng"]

passwords = ["abc123", "def456", "cleanlist2025"]

**👩‍💻**

**app.py**

import streamlit as st

import streamlit\_authenticator as stauth

# 1️⃣ Database credentials (used internally by the app)

db\_user = st.secrets["database"]["user"]

db\_pass = st.secrets["database"]["password"]

db\_host = st.secrets["database"]["host"]

# (Simulating a database connection message)

st.sidebar.success(f"Connected to DB as '{db\_user}' on {db\_host}")

# 2️⃣ App login credentials (for human testers)

usernames = st.secrets["credentials"]["usernames"]

passwords = st.secrets["credentials"]["passwords"]

# Hash the passwords for secure comparison

hashed\_passwords = stauth.Hasher(passwords).generate()

# Build authenticator

authenticator = stauth.Authenticate(

names=usernames, # names can be same as usernames for demo

usernames=usernames,

passwords=hashed\_passwords,

cookie\_name="my\_app",

key="abcdef",

cookie\_expiry\_days=1

)

# Login widget

name, authentication\_status, username = authenticator.login("Login", "main")

if authentication\_status:

st.success(f"Welcome, {name}! 🎉")

st.write("You are now logged in and can use the app.")

elif authentication\_status == False:

st.error("Username/password is incorrect")

else:

st.warning("Please enter your username and password")

**🏃 How This Works**

1. When the app starts:
   * It reads [database] → silently connects to your “database.”
   * It does not expose this login to testers.
2. When a user visits:
   * They must log in with a username + password from [credentials].
   * Only then do they see the main app.

**🔐 Key Takeaway**

* [database] credentials = app’s backstage pass.
* [credentials] list = front desk where guests check in.
* Both live in the same secrets.toml because it’s one safe place for all sensitive info.

⚡ Question for you:

Do you want me to extend this demo so the logged-in tester’s name controls what data they can see in the app (e.g. different dashboards for jsmith vs adjikeng)? That would show how secrets.toml can scale beyond just login.

Perfect — let’s tackle your question carefully, because this is a subtle but important point.

**1️⃣ authenticator.login() always returns 3 things**

No matter how many extra fields you add to your [credentials] in secrets.toml, the authenticator.login() function always returns exactly 3 variables:

name, authentication\_status, username = authenticator.login("Login", "main")

* name → the human-readable name
* authentication\_status → True/False/None
* username → the internal username

You cannot increase the number of returned variables by adding more arrays to TOML.

**2️⃣ How to use extra fields**

Even though login() only returns 3 things, you can look up additional info in TOML after the user is authenticated:

# Suppose you added extra fields in TOML

roles = st.secrets["credentials"]["roles"]

departments = st.secrets["credentials"]["departments"]

# Find the index of the logged-in user

if authentication\_status:

idx = st.secrets["credentials"]["usernames"].index(username)

user\_role = roles[idx]

user\_department = departments[idx]

st.write(f"Welcome {name}! Your role is {user\_role} and department is {user\_department}.")

✅ Key point:

* login() only handles authentication (username/password).
* Extra fields are looked up manually using the username after successful login.
* This keeps the Authenticator secure and focused on login, while giving you flexibility for extra data.

**3️⃣ Analogy**

Think of it like a hotel check-in desk:

| **Step** | **Analogy** |
| --- | --- |
| authenticator = stauth.Authenticate() | Hire the bouncer and give them the guest list |
| authenticator.login() | Bouncer checks the guest’s ID → returns: ✅ allowed or ❌ denied + name + guest ID |
| Extra fields in TOML | Hotel’s internal file with room number, VIP status, preferences |
| Lookup after login | Staff uses guest ID to fetch room number, privileges, etc. — doesn’t change what the bouncer returns |

**4️⃣ What this means for your app**

* You don’t try to assign extra variables to the left side of authenticator.login(). That would break the code.
* Instead, read your extra fields from TOML using the username as a key/index after authentication.
* This also scales well when your [credentials] list grows — you don’t need to modify the login function at all.

If you want, I can now build the full demo where:

1. Different users see custom dashboards based on role/department.
2. TOML stores usernames, passwords, roles, departments.
3. After login, the app looks up extra info and renders the page accordingly.

Do you want me to do that next?

Perfect! Let’s build a full demo that shows:

1. Users log in with username/password stored in secrets.toml.
2. After login, the app looks up extra info (role, department) from TOML.
3. Each user sees a custom dashboard depending on their role/department.

**1️⃣ secrets.toml**

[credentials]

usernames = ["jsmith", "rbriggs", "adjikeng"]

passwords = ["abc123", "def456", "cleanlist2025"]

names = ["John Smith", "Rebecca Briggs", "Albert Djikeng"]

roles = ["admin", "tester", "tester"]

departments = ["finance", "marketing", "operations"]

**2️⃣ app.py**

import streamlit as st

import streamlit\_authenticator as stauth

# 1️⃣ Authenticator setup

usernames = st.secrets["credentials"]["usernames"]

passwords = st.secrets["credentials"]["passwords"]

names = st.secrets["credentials"]["names"]

hashed\_passwords = stauth.Hasher(passwords).generate()

authenticator = stauth.Authenticate(

names=names,

usernames=usernames,

passwords=hashed\_passwords,

cookie\_name="my\_app\_cookie",

key="abcdef",

cookie\_expiry\_days=1

)

# 2️⃣ Login form

name, authentication\_status, username = authenticator.login("Please Sign In", "main")

# 3️⃣ Check authentication

if authentication\_status:

st.success(f"Welcome {name}!")

# 4️⃣ Lookup extra info

idx = usernames.index(username)

role = st.secrets["credentials"]["roles"][idx]

department = st.secrets["credentials"]["departments"][idx]

st.write(f"Your role: {role}")

st.write(f"Your department: {department}")

# 5️⃣ Conditional dashboard content

if role == "admin":

st.subheader("Admin Dashboard")

st.write("You can manage users and view company-wide analytics.")

elif role == "tester":

st.subheader("Tester Dashboard")

st.write("You can run tests and submit feedback here.")

st.button("Logout", on\_click=authenticator.logout)

elif authentication\_status == False:

st.error("Username/password is incorrect")

else:

st.warning("Please enter your username and password")

**3️⃣ How it works**

1. Login form → shows “Please Sign In” in main app area
2. User enters credentials → login() checks authentication
3. After successful login → app looks up extra info (role/department) using the username index
4. Conditional rendering → each user sees different dashboards based on role
5. Logout → clears the session cookie

**✅ Key Points**

* You don’t add extra return variables to login(). All extra info is fetched after authentication.
* The TOML file stores all sensitive info in one place.
* The structure scales — just add new users + roles + departments in TOML.
* Works locally or on Streamlit Cloud; no tunnel needed for permanent access.

If you want, I can also show how to extend this demo so your app reacts differently based on department AND role — for example, showing different tables or charts for “finance” vs “marketing.”

Do you want me to do that next?

Perfect! Let’s build a full demo that shows:

1. Users log in with username/password stored in secrets.toml.
2. After login, the app looks up extra info (role, department) from TOML.
3. Each user sees a custom dashboard depending on their role/department.

**1️⃣ secrets.toml**

[credentials]

usernames = ["jsmith", "rbriggs", "adjikeng"]

passwords = ["abc123", "def456", "cleanlist2025"]

names = ["John Smith", "Rebecca Briggs", "Albert Djikeng"]

roles = ["admin", "tester", "tester"]

departments = ["finance", "marketing", "operations"]

**2️⃣ app.py**

import streamlit as st

import streamlit\_authenticator as stauth

# 1️⃣ Authenticator setup

usernames = st.secrets["credentials"]["usernames"]

passwords = st.secrets["credentials"]["passwords"]

names = st.secrets["credentials"]["names"]

hashed\_passwords = stauth.Hasher(passwords).generate()

authenticator = stauth.Authenticate(

names=names,

usernames=usernames,

passwords=hashed\_passwords,

cookie\_name="my\_app\_cookie",

key="abcdef",

cookie\_expiry\_days=1

)

# 2️⃣ Login form

name, authentication\_status, username = authenticator.login("Please Sign In", "main")

# 3️⃣ Check authentication

if authentication\_status:

st.success(f"Welcome {name}!")

# 4️⃣ Lookup extra info

idx = usernames.index(username)

role = st.secrets["credentials"]["roles"][idx]

department = st.secrets["credentials"]["departments"][idx]

st.write(f"Your role: {role}")

st.write(f"Your department: {department}")

# 5️⃣ Conditional dashboard content

if role == "admin":

st.subheader("Admin Dashboard")

st.write("You can manage users and view company-wide analytics.")

elif role == "tester":

st.subheader("Tester Dashboard")

st.write("You can run tests and submit feedback here.")

st.button("Logout", on\_click=authenticator.logout)

elif authentication\_status == False:

st.error("Username/password is incorrect")

else:

st.warning("Please enter your username and password")

**3️⃣ How it works**

1. Login form → shows “Please Sign In” in main app area
2. User enters credentials → login() checks authentication
3. After successful login → app looks up extra info (role/department) using the username index
4. Conditional rendering → each user sees different dashboards based on role
5. Logout → clears the session cookie

**✅ Key Points**

* You don’t add extra return variables to login(). All extra info is fetched after authentication.
* The TOML file stores all sensitive info in one place.
* The structure scales — just add new users + roles + departments in TOML.
* Works locally or on Streamlit Cloud; no tunnel needed for permanent access.

If you want, I can also show how to extend this demo so your app reacts differently based on department AND role — for example, showing different tables or charts for “finance” vs “marketing.”

Do you want me to do that next?

Perfect — let’s upgrade the demo so each user’s grocery list is persisted in Firestore, meaning it stays even if they log out or close the app. I’ll also integrate it with your current Streamlit + Authenticator setup.

**1️⃣ Firestore Setup**

1. Go to [Firebase Console](https://console.firebase.google.com/) → Create a project.
2. Navigate to Firestore Database → Create a database (start in test mode for now).
3. Create a service account:
   * Settings → Project Settings → Service Accounts → Generate New Private Key → Download JSON.
4. Save the JSON file somewhere safe (e.g., firebase\_key.json).

**2️⃣ Install Dependencies**

Add to your requirements.txt:

streamlit

streamlit-authenticator

pandas

numpy

firebase-admin

Then install locally:

pip install streamlit-authenticator firebase-admin pandas numpy

**3️⃣ Initialize Firestore in app.py**

import streamlit as st

import streamlit\_authenticator as stauth

import firebase\_admin

from firebase\_admin import credentials, firestore

# 1️⃣ Initialize Firebase

if not firebase\_admin.\_apps:

cred = credentials.Certificate("firebase\_key.json")

firebase\_admin.initialize\_app(cred)

db = firestore.client()

**4️⃣ Save & Load User Grocery List**

**Function to load list:**

def load\_grocery\_list(username):

doc\_ref = db.collection("grocery\_lists").document(username)

doc = doc\_ref.get()

if doc.exists:

return doc.to\_dict().get("items", [])

return []

**Function to save list:**

def save\_grocery\_list(username, items):

doc\_ref = db.collection("grocery\_lists").document(username)

doc\_ref.set({"items": items})

**5️⃣ Full App Integration (Login + Autocomplete + Firestore)**

# Authenticator setup (same as before)

usernames = st.secrets["credentials"]["usernames"]

passwords = st.secrets["credentials"]["passwords"]

names = st.secrets["credentials"]["names"]

hashed\_passwords = stauth.Hasher(passwords).generate()

authenticator = stauth.Authenticate(

names=names,

usernames=usernames,

passwords=hashed\_passwords,

cookie\_name="my\_app\_cookie",

key="abcdef",

cookie\_expiry\_days=1

)

# Login

name, authentication\_status, username = authenticator.login("Please Sign In", "main")

if authentication\_status:

st.success(f"Welcome {name}!")

# Load user grocery list from Firestore

user\_list = load\_grocery\_list(username)

# Autocomplete items

GROCERY\_ITEMS = [

"Apples", "Bananas", "Bread", "Butter", "Carrots", "Cheese",

"Eggs", "Milk", "Orange Juice", "Spinach", "Tomatoes", "Yogurt"

]

item\_to\_add = st.text\_input("Add an item to your grocery list:", placeholder="Start typing...")

suggestions = [item for item in GROCERY\_ITEMS if item.lower().startswith(item\_to\_add.lower())] if item\_to\_add else []

if suggestions:

st.write("Suggestions:", ", ".join(suggestions))

if st.button("Add Item"):

if item\_to\_add:

user\_list.append(item\_to\_add)

save\_grocery\_list(username, user\_list) # Save to Firestore

st.success(f"Added '{item\_to\_add}' to your list!")

else:

st.warning("Enter an item first.")

# Display grocery list

st.subheader("Your Grocery List")

if user\_list:

for idx, item in enumerate(user\_list, 1):

st.write(f"{idx}. {item}")

else:

st.write("No items yet.")

st.button("Logout", on\_click=authenticator.logout)

elif authentication\_status == False:

st.error("Username/password is incorrect")

else:

st.warning("Please enter your username and password")

**6️⃣ How it Works**

1. User logs in → Authenticator validates credentials.
2. Firestore loads their grocery list (if any).
3. Autocomplete suggests items as they type.
4. Add item → updates Firestore immediately.
5. List persists across sessions and devices.

✅ Next Enhancements You Could Do

* Add item deletion and editing.
* Integrate CleanScore analysis on items before saving.
* Make autocomplete dynamic using a database of thousands of grocery items.
* Restrict access to only their own document for security.

If you want, I can refactor this code into a fully production-ready Streamlit app with:

* Authenticator login
* Firestore persistence
* Autocomplete
* Optional CleanScore integration

This version would be ready to deploy on Streamlit Cloud.

Do you want me to do that next?

Perfect — let’s design a production-ready Streamlit app with all the pieces integrated and ready for deployment on Streamlit Cloud. This version will include:

1. Secure login with streamlit-authenticator
2. Firestore-backed grocery list storage per user
3. Autocomplete for grocery items
4. Optional CleanScore analysis placeholder
5. Ready-to-deploy folder structure and requirements

**1️⃣ Project Folder Structure**

cleanlist\_app/

│

├─ app.py # Main Streamlit app

├─ requirements.txt # Dependencies

├─ secrets.toml # User credentials & roles

├─ firebase\_key.json # Firestore service account (keep secret)

**2️⃣ requirements.txt**

streamlit

streamlit-authenticator

pandas

numpy

firebase-admin

**3️⃣ secrets.toml**

[credentials]

usernames = ["jsmith", "rbriggs", "adjikeng"]

passwords = ["abc123", "def456", "cleanlist2025"]

names = ["John Smith", "Rebecca Briggs", "Albert Djikeng"]

roles = ["admin", "tester", "tester"]

departments = ["finance", "marketing", "operations"]

Tip: Never commit firebase\_key.json to GitHub. Keep it locally or set Firestore credentials using Streamlit Cloud secrets.

**4️⃣ app.py — Full Production-Ready Code**

import streamlit as st

import streamlit\_authenticator as stauth

import firebase\_admin

from firebase\_admin import credentials, firestore

import pandas as pd

import numpy as np

# -----------------------

# 1️⃣ Initialize Firestore

# -----------------------

if not firebase\_admin.\_apps:

cred = credentials.Certificate("firebase\_key.json")

firebase\_admin.initialize\_app(cred)

db = firestore.client()

# -----------------------

# 2️⃣ Authenticator Setup

# -----------------------

usernames = st.secrets["credentials"]["usernames"]

passwords = st.secrets["credentials"]["passwords"]

names = st.secrets["credentials"]["names"]

hashed\_passwords = stauth.Hasher(passwords).generate()

authenticator = stauth.Authenticate(

names=names,

usernames=usernames,

passwords=hashed\_passwords,

cookie\_name="cleanlist\_cookie",

key="abcdef",

cookie\_expiry\_days=1

)

# -----------------------

# 3️⃣ Login Form

# -----------------------

name, authentication\_status, username = authenticator.login("Please Sign In", "main")

if authentication\_status:

st.success(f"Welcome {name}!")

# -----------------------

# 4️⃣ Load user grocery list

# -----------------------

def load\_grocery\_list(username):

doc\_ref = db.collection("grocery\_lists").document(username)

doc = doc\_ref.get()

if doc.exists:

return doc.to\_dict().get("items", [])

return []

def save\_grocery\_list(username, items):

doc\_ref = db.collection("grocery\_lists").document(username)

doc\_ref.set({"items": items})

user\_list = load\_grocery\_list(username)

# -----------------------

# 5️⃣ Autocomplete Items

# -----------------------

GROCERY\_ITEMS = [

"Apples", "Bananas", "Bread", "Butter", "Carrots", "Cheese",

"Eggs", "Milk", "Orange Juice", "Spinach", "Tomatoes", "Yogurt"

]

item\_to\_add = st.text\_input("Add an item to your grocery list:", placeholder="Start typing...")

suggestions = [item for item in GROCERY\_ITEMS if item.lower().startswith(item\_to\_add.lower())] if item\_to\_add else []

if suggestions:

st.write("Suggestions:", ", ".join(suggestions))

# -----------------------

# 6️⃣ Add Item Button

# -----------------------

if st.button("Add Item"):

if item\_to\_add:

user\_list.append(item\_to\_add)

save\_grocery\_list(username, user\_list) # Save to Firestore

st.success(f"Added '{item\_to\_add}' to your list!")

else:

st.warning("Enter an item first.")

# -----------------------

# 7️⃣ Display Grocery List

# -----------------------

st.subheader("Your Grocery List")

if user\_list:

for idx, item in enumerate(user\_list, 1):

st.write(f"{idx}. {item}")

else:

st.write("No items yet.")

# -----------------------

# 8️⃣ Optional CleanScore Placeholder

# -----------------------

if user\_list:

st.subheader("CleanScore Analysis (Demo)")

clean\_scores = [np.random.randint(50, 100) for \_ in user\_list] # Replace with real CleanScore logic

df = pd.DataFrame({"Item": user\_list, "CleanScore": clean\_scores})

st.dataframe(df)

# -----------------------

# 9️⃣ Logout Button

# -----------------------

st.button("Logout", on\_click=authenticator.logout)

elif authentication\_status == False:

st.error("Username/password is incorrect")

else:

st.warning("Please enter your username and password")

**5️⃣ How It Works**

1. User login:
   * Authenticator handles username/password securely.
2. Firestore persistence:
   * Each user’s list is stored under their username document.
   * List persists across sessions and devices.
3. Autocomplete:
   * Suggests items from GROCERY\_ITEMS as the user types.
   * Easily replaceable with a real item database.
4. Conditional CleanScore table:
   * Currently a demo with random scores.
   * Can be replaced with your ingredient parsing logic.
5. Logout:
   * Clears cookie/session.

✅ Next Steps for Deployment

1. Push your folder to GitHub (do not include firebase\_key.json).
2. Use Streamlit Cloud secrets to store Firebase credentials securely.
3. Deploy via [share.streamlit.io](https://share.streamlit.io) → select repo + branch + app.py.
4. Your app now has:
   * Persistent grocery lists per user
   * Autocomplete
   * Login with secure credentials
   * Optional CleanScore analysis

If you want, I can also show exactly how to move your Firebase credentials into Streamlit Cloud secrets, so you never have to store firebase\_key.json locally, making the app fully production-ready.

Do you want me to do that next?

Perfect — let’s make your app production-ready by moving Firebase credentials into Streamlit Cloud secrets so you never have to store firebase\_key.json locally.

**1️⃣ Export Firebase Key as JSON string**

1. Open your Firebase service account JSON (firebase\_key.json).
2. Copy the entire contents as a single string.
   * Example snippet:

{

"type": "service\_account",

"project\_id": "myproject-123",

"private\_key\_id": "ABC123...",

"private\_key": "-----BEGIN PRIVATE KEY-----\nMIIEvgIBADANB...snip...\n-----END PRIVATE KEY-----\n",

"client\_email": "firebase-adminsdk-xxx@myproject-123.iam.gserviceaccount.com",

...

}

**2️⃣ Add it as a Streamlit Cloud secret**

1. Go to [Streamlit Cloud](https://share.streamlit.io/) → Your app → Secrets
2. Add a new secret:

FIREBASE\_KEY = <entire JSON string>

* Make sure the string is all on one line (or use \n in the private key as shown).
* This will be encrypted and safe — no JSON file needed locally.

**3️⃣ Update app.py to use the secret**

import streamlit as st

import firebase\_admin

from firebase\_admin import credentials, firestore

import json

# -----------------------

# 1️⃣ Initialize Firestore from Streamlit secret

# -----------------------

if not firebase\_admin.\_apps:

# Load JSON string from secrets

firebase\_json\_str = st.secrets["FIREBASE\_KEY"]

cred\_dict = json.loads(firebase\_json\_str)

cred = credentials.Certificate(cred\_dict)

firebase\_admin.initialize\_app(cred)

db = firestore.client()

✅ That’s it! Now your Firestore connection uses Streamlit Cloud secrets, and firebase\_key.json is no longer needed.

**4️⃣ Key Notes**

* Secrets in Streamlit Cloud are encrypted and never committed to GitHub.
* The rest of your app (app.py, secrets.toml, requirements.txt) remains unchanged.
* Deployment is fully secure and works across devices.

If you want, I can write a fully final version of your app that:

* Uses Streamlit Cloud secrets for Firebase
* Maintains login + Firestore persistence + autocomplete + CleanScore
* Is ready to just push to GitHub and deploy, no local JSON needed

Do you want me to do that next?