Perfect, let’s go step by step so you have a crystal-clear understanding before we extend the demo.

**1️⃣ Project Folder for Streamlit Cloud Deployment**

Yes, at a minimum, you need three files in your project folder:

| **File** | **Purpose** |
| --- | --- |
| app.py | Your Streamlit code (UI + logic) |
| requirements.txt | List of all Python packages your app needs |
| secrets.toml | Secure storage for passwords, API keys, DB credentials, etc. |

**But what goes into**

**requirements.txt**

**?**

This is how Python/Streamlit Cloud knows what libraries to install. You don’t include the actual library files, just the names and optional versions. For your demo:

streamlit

streamlit-authenticator

pandas

numpy

* streamlit → the framework running the app
* streamlit-authenticator → handles username/password login easily
* pandas → for data manipulation (tables, CSV, etc.)
* numpy → for numerical operations

Think of requirements.txt as a shopping list. You don’t pack the groceries, you just tell someone else what to buy.

✅ Optional extras:

* .streamlit/config.toml → Streamlit app configs (theme, server port, etc.)
* Any .csv or data files your app uses

But strictly speaking, only app.py + requirements.txt + secrets.toml is required.

**2️⃣ Diving Deeper: stauth.Authenticate**

authenticator = stauth.Authenticate(

names=usernames,

usernames=usernames,

passwords=hashed\_passwords,

cookie\_name="my\_app",

key="abcdef",

cookie\_expiry\_days=1

)

**What it does:**

1. names → Human-readable names displayed on the UI (welcome messages).
2. usernames → Internal IDs used to match credentials.
3. passwords → Hashed passwords that are compared securely.
4. cookie\_name → Name of the browser cookie that remembers who is logged in.
5. key → Secret key for encrypting cookies.
6. cookie\_expiry\_days → How long the login session lasts.

Basically: it creates a login manager object. You give it the list of usernames/passwords, and it handles authentication, sessions, cookies, and logout.

**3️⃣ Diving Deeper: authenticator.login**

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

**Parameters:**

1. "Login" → The title of the login widget on the app page.
2. "main" → The location in your Streamlit page where the login widget appears.
   * "main" → the main app area
   * "sidebar" → displays the widget in the sidebar

**Returns:**

| **Variable** | **Meaning** |
| --- | --- |
| name | The human-readable name of the logged-in user |
| authentication\_status | True if credentials are correct, False if wrong, None if not entered yet |
| username | The internal username used to match credentials |

So this one line:

* Displays a login form in the app
* Checks credentials against the Authenticate object
* Returns info about the current user/session

**✅ TL;DR**

* Project folder: minimum 3 files; requirements.txt lists library names, not code
* stauth.Authenticate → login manager object (handles sessions, cookies)
* authenticator.login → displays the form, validates credentials, tells you who logged in

If you want, I can now extend our demo so that different testers see different dashboards or content based on their username, showing exactly how the credentials list scales.

Do you want me to do that next?

Perfect — let’s go through your three questions carefully, one by one.

**1️⃣ Is TOML an acronym?**

* No, it’s not a strict acronym.
* TOML stands for “Tom’s Obvious, Minimal Language.”
* It was designed to be human-readable and easy to write, focusing on simplicity and clarity over complex syntax.

Think of it as a clean, friendly alternative to JSON or YAML for configuration files.

**2️⃣ Title of the login widget**

When you write:

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

* The first argument "Login" is exactly what appears as the title of the login widget on your Streamlit app page.
  + Example: You’ll see a small box in the main app area (or sidebar) that says “Login” at the top.
* The second argument "main" just tells Streamlit where to place the widget.

So yes, the title is explicitly set — it’s not implied. You can change it:

authenticator.login("Please Sign In", "sidebar")

→ The widget will show “Please Sign In” in the sidebar instead of "Login" in the main area.

**3️⃣ LocalTunnel / Ngrok + laptop sleep**

Here’s the key:

* LocalTunnel / Ngrok only expose a service that is running on your laptop.
* If your laptop goes to sleep, the local Streamlit server stops.
* Consequently, the public URL (like https://random-id.ngrok-free.app) will stop working or give an error.

**Why your**

**https://loca.lt/mytunnelpassword**

**link works:**

* That link is a preconfigured tunnel with LocalTunnel’s own temporary password protection.
* It may appear to work, but your laptop must still be awake to serve the app.
* If your laptop sleeps or disconnects, anyone trying to access the link will get errors.

Bottom line:

* Using tunnels is fine for testing, but the app will not stay live if your machine is asleep or offline.
* For a permanent, always-accessible app, you need a cloud deployment (Streamlit Cloud, Render, or a VPS).

⚡ Quick recap:

1. TOML = Tom’s Obvious, Minimal Language
2. The login widget title is set by the first argument of authenticator.login()
3. Tunnels like LocalTunnel / Ngrok stop working if your laptop sleeps — the server must be running for the public link to function

If you want, I can now extend your demo so that different testers see different dashboards, which works both locally and will be fully compatible when deployed to Streamlit Cloud — solving all these sleep/tunnel issues.

Do you want me to do that next?

Ah, perfect — this is an important point, and it’s easy to get tangled. Let’s untangle it step by step with a clear analogy.

**1️⃣ What’s happening conceptually**

**Step 1:**

**authenticator = stauth.Authenticate(...)**

* This creates a login manager object called authenticator.
* Think of it like building a bouncer at the door of a club:
  + You tell the bouncer: “Here’s the list of allowed people (usernames + passwords) and how to recognize them (names).”
* At this point, nothing has been “checked” yet — you’re just hiring the bouncer and giving them instructions.

**Step 2:**

**name, authentication\_status, username = authenticator.login(...)**

* This calls a method on the bouncer you just created.
* When you call authenticator.login():
  1. The login form is displayed (in Streamlit, either main area or sidebar).
  2. The bouncer checks whatever the user enters against the credentials you gave it.
  3. The bouncer returns three things:
     + name → the friendly name of the person who logged in
     + authentication\_status → whether they passed (True), failed (False), or haven’t entered yet (None)
     + username → the internal username used to identify them

✅ So the variables on the left-hand side are assigned by the login() method, not fed into it.

**2️⃣ Analogy**

Think of it like a theme park ticketing system:

| **Step** | **Code** | **Analogy** |
| --- | --- | --- |
| authenticator = stauth.Authenticate(...) | Create a login manager | You hire a bouncer and give them the guest list |
| name, authentication\_status, username = authenticator.login(...) | Call the login method | Guests arrive, the bouncer checks the tickets, and reports:  - name = guest’s full name  - authentication\_status = True/False  - username = guest ID |

* You don’t provide name, authentication\_status, username when calling login() — they’re outputs, like a report from the bouncer.
* The bouncer already knows the rules (from Authenticate) and uses them to check every guest.

**3️⃣ Flow in the app**

1. App starts → creates authenticator with the allowed usernames/passwords
2. User sees login form → enters credentials
3. authenticator.login() checks the credentials
4. Returns values → assigned to name, authentication\_status, username variables
5. App reacts → e.g., if authentication\_status: → show main content

✅ Key takeaway:

* stauth.Authenticate = the bouncer with rules
* authenticator.login() = the act of checking credentials at the door
* Variables on the left = results of the check, not input to it

If you want, I can now extend your Streamlit demo so that different users see different dashboards based on their username, showing the Authenticate → login → conditional display flow in action.

Do you want me to do that next?

Exactly — you’re on the right track, and here’s a more precise breakdown:

**1️⃣ TOML File: Store the extra info**

* Yes, if you want more info per user (e.g., department, role, favorite color, permissions level), you need to add new arrays or fields in your [credentials] section.
* Example:

[credentials]

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

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

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

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

* Each array must align by index:
  + jsmith → password abc123 → role admin → department finance
  + rbriggs → password def456 → role tester → department marketing

**2️⃣ Python app: Add input fields**

* Streamlit doesn’t automatically create new fields beyond username/password.
* To make extra fields visible on the login screen:

import streamlit as st

# Custom input example

role\_input = st.selectbox("Select your role:", ["admin", "tester"])

department\_input = st.text\_input("Enter your department:")

* Then you can validate or use that info after the user logs in.

**3️⃣ Notes on Integration**

* Streamlit Authenticator only manages username/password login.
* Any extra fields beyond login have to be created manually in app.py.
* You can still read these extra fields from the TOML to validate or pre-fill values.

✅ Summary

1. Extra user info → add arrays/fields in [credentials] of secrets.toml
2. Extra input fields → explicitly create them in app.py (Streamlit widgets)
3. Keep array indices aligned so data matches the correct user

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

* Each logged-in user sees a custom dashboard based on their username
* Additional info from TOML (like role or department) controls what content is visible

Do you want me to do that next?