How HTTP Applications Preserve State (Authentication & Sessions)

HTTP is a stateless protocol, meaning each request is independent and carries no memory of previous interactions. To preserve state—especially for authentication—web applications rely on mechanisms like **cookies** and **sessions**. **Cookies** act as small pieces of data stored in the browser and automatically sent with each request, allowing the server to recognize the client across multiple interactions. **For authenticated users**, the server typically generates a **session ID**, stores session data on the server (or in a session store), and sends the session ID to the browser as a cookie. On each request, the browser returns this cookie, enabling the server to retrieve the user's session and maintain continuity, such as keeping a user logged in. Secure session management also involves attributes like `HttpOnly`, `Secure`, and `SameSite` to protect against hijacking and fixation attacks, and frameworks often rotate session IDs after login to prevent fixation vulnerabilities.

My Django sticky_notes app works the same way

Imagine you go to **Macys.com**, browse around, log in, add items to your cart, and come back later. Even though HTTP is *stateless*, Macy's still remembers:

- who you are
- what's in your cart
- whether you're logged in
- your browsing preferences

How? Through **cookies + server-side sessions**.

When you login, Macy's server:

- verifies your credentials
- updates the session data stored on the server
- marks your session as "authenticated"

Your browser keeps the same cookie: Now every request you make includes that cookie automatically.

When you click "Log Out," Macy's server:

- deletes or invalidates your session
- tells your browser to delete the cookie

Now if you refresh, Macy's no longer knows who you are.

Procedures for Migrating a Django Application to a Server-Based Database Like MariaDB

Migrating a Django project from its default SQLite database to a server-based relational database such as MariaDB involves configuring the database server, updating Django's settings, and applying migrations so the schema is created on the new backend. The process typically includes installing MariaDB, creating a database and user, installing the appropriate Python client library, updating Django's DATABASES configuration, verifying the connection, and running migrations to generate the schema on the MariaDB server. When moving from an existing database, developers may also export and import data to ensure continuity and integrity.

Step-by Step Procedures for Migrating Django to MariaDB

1. Install and configure MariaDB on the server. Ensure the service is running and create a database and user with appropriate privileges.

2. Install the MariaDB/MySQL client library in the Django virtual environment using pip install mysqlclient.

3. Update Django's DATABASES setting in settings.py to use the MySQL backend with the correct database name, user, password, host, and port.

4. Verify the connection using python manage.py check.

5. Apply Django migrations to MariaDB using python manage.py migrate.

6. (Optional) Import existing data using tools like mysqldump to align with the new schema.

7. Update deployment settings and restart the application so the new database configuration is active.