‘’’**Conduct** research to understand how **HTTP** applications preserve the state of an application across multiple request-response cycles, especially concerning user authentication and session management. ‘’’

#Write a paragraph describing your findings.Investigate and document the procedures for performing Django database migrations to a server-based relational database like MariaDB. Write a paragraph describing your findings.

1️⃣How HTTP Applications Preserve State Across Requests.

HTTP is “stateless”, meaning every request the browser makes is independent. For example, if you log in on a website and then click another page, the server doesn’t “remember” who you are unless you explicitly provide a way for it to. To handle this, web applications use “state-preserving techniques”.

1. **Cookies**

* A small piece of data stored in the client’s browser.
* When a user logs in, the server generates a **session ID** and sends it in a cookie.
* On each subsequent request, the browser sends this cookie back, so the server knows which user is making the request.
* Example: `sessionid=abcd1234`.

2. Server-Side Sessions

* Django stores session data on the server (like user ID, login time, preferences).
* The client only holds the **session ID**.
* This is more secure than storing sensitive information directly in cookies.

3. Token-Based Authentication (JWT)

* Instead of a session ID, the server gives the client a **JSON Web Token** containing user info.
* The client sends this token with each request (usually in HTTP headers).
* The server verifies the token’s signature and extracts the user info.
* Useful for **APIs and mobile apps**, where maintaining a server-side session may not be practical.

These methods allow the server to “remember” users and maintain a continuous experience across multiple page loads, even though HTTP itself is

****

2️⃣ **Django Migrations with MariaDB**

When working with Django, your **models** define the structure of your database tables. Whenever you create or change models, Django needs to **update the database schema**. That's where migrations come in.

Step-by-Step Process:

1. **Set Up *MariaDB* in *Django***
   In `settings.py

2. **Create *Migration* Files**

   * Django compares your current models with the last migration.
   * It generates migration files in your app's `migrations` folder.
   * These are Python scripts that describe how to modify the database (add tables, fields, constraints, etc.).

3. **Apply Migrations to the *Database***

   * Executes the migration scripts against the database.
   * Creates or updates tables and columns in MariaDB to match your models.

4. ***Deployment Workflow***

Local development: Make model changes → `makemigrations` → test locally.
   * **Version control:** Commit migration files → push code to server.
   * **Server deployment:** Pull code → run `python manage.py migrate` → production database updated safely.

Why it's safe and efficient:

* Each migration is versioned and reversible.
* Multiple developers can work on the same project without manually editing the database.
* Keeps production, staging, and local databases in sync with the code.

****