Login Process Flow

**1. User Enters Credentials:**

**Step:** The user enters their identifier (username, email, badge barcode, or RFID) and password into the login form.

**Package/Extension:**

**Django Forms**: Submits the form data to the backend via a ‘POST’ request. This sends the credentials to the Django authentication system, Django-Auth, which proceeds with validation steps. Validation is monitored by Django-Axes in Step 2 and Step 3 for failed login attempts.

**2. Django-Axes Tracks Failed Attempts:**

**Step:** After the form submission (Step 1), Django-Axes immediately monitors the number of failed login attempts associated with the user identifier (username, email, badge barcode, or RFID) or IP address.

**Package/Extension:**

**Django-Axes:** Before the credentials are validated, Django-Axes checks the number of failed attempts from the same user or IP address. If the number of failed attempts reaches 3, the user or IP is temporarily locked out based on the lockout policy.

Django-Axes determines whether Django-Auth should continue with credential validation or block further login attempts.

**Loop:**

If the user fails to log in 3 times, they are locked out, and no further validation is performed until the lockout expires.

If the user has **NOT** reached the limit, Django-Axes allows Django-Auth to proceed to Step 3.

**3. User Credentials Validation:**

**Step:** If Django-Axes has not triggered a lockout, the provided credentials (identifier and password) are validated against the ‘users\_auth.users’ table.

**Package/Extension:**

**Django-Auth:** The authentication backend (django.contrib.auth.backends) checks the credentials. It compares the submitted identifier and password against the stored data in the ‘users\_auth.users’ table.

If the credentials are valid, the user is authenticated, and the login process proceeds to Step 4.

If the credentials are invalid, Django-Axes increments the failed attempt counter and reverts to the login screen with an error message.

**Loop:**

If validation fails, Django-Axes increments the counter. After 3 failures, the user or IP is locked out.

**4. Multi-factor Authentication (OTP):**

**Step:** If the user’s credentials are valid, Django-Auth moves the process to Django-OTP. Django-OTP generates an OTP and sends to the user's registered device (recorded in the ‘users\_auth.mfa\_devices’ table).

**Package/Extension:**

**Django-OTP**: Handles the generation and sending of the OTP based on the MFA device registered for the user (e.g., SMS, email, or app-based OTP).

**5. OTP Validation:**

**Step:** The user enters the OTP received on their registered device. The OTP is validated using the algorithm (e.g., TOTP or HOTP).

**Package/Extension:**

**Django-OTP:** Validates the entered OTP using the time-based or counter-based OTP algorithm. If the OTP is correct, the user proceeds to the next step. If the OTP is incorrect, the user is prompted to try again or request a new OTP.

**Loop:**

The user can retry up to 3 times. Each failed OTP attempt is tracked by a **counter stored in Redis**. Redis manages the counter and tracks the time window for retry attempts.

After 3 failed OTP attempts within a set time period, the user is locked out from further attempts. Redis tracks the failed attempts and enforces the lockout. The lockout period expires after a certain amount of time, or the user can be manually unlocked by an administrator in the custom Django-Admin pages.

**6. Session Management:**

**Step:** Once the user successfully enters an OTP, their session is created and tracked using Redis. Redis handles the active session, while Django-user-sessions logs session history (start time, end time, activity, IP address, etc.).

**Package/Extension:**

**Redis:** Handles active session storage for speed and scalability.

**Django-user-sessions:** Logs session history in the sessions table, capturing details such as session duration, last activity, and session start/end times.

**7. Load User Permissions and Preferences:**

**Step:** After successfully logging in and a session being started and managed by Redis, Django ORM retrieves the user’s preferences (e.g., theme, language, layout) and permissions (roles, access levels) from the ‘users’ database, specifically from the ‘users.user\_preferences’ and ‘users.roles’ tables.

**Package/Extension:**

**Django ORM:** Fetches user-specific data from the ‘users’ database, loading the relevant information to configure the user’s environment (e.g., dashboard layout, UI theme, and any access-based permissions for the web app).

The preferences and permissions loaded here will dictate the user's experience (e.g., UI layout) and what they have access to within the application.

**8. Django Simple History Tracks User Changes:**

**Step:** Any changes made by the user, such as updates to any user profile, preferences, or roles, are logged using Django Simple History. This ensures that all modifications to user data are recorded for auditing and tracking purposes, whether performed by the user on their profile, preferences, or roles or a user with authorization to make changes to other users’ profile, preferences, or roles.

**Package/Extension:**

**Django Simple History:** Tracks and logs changes to the ‘users’ table, capturing a historical record of any modifications made to user data, preferences, or roles. This includes the original and updated values, the user who made the change, and the timestamp of the change.

9. Redirect to User’s Home Page:

**Step:** After all previous steps are completed, the user is redirected to the ‘base.html’ template. This serves as the foundational layout for the app, with navigation links to various sections populated based on the user’s roles and permissions. The user’s preferences (e.g., theme, layout) are applied to customize the appearance and functionality of the page.

**Package/Extension:**

**Django:** Handles the redirection to ‘base.html’ and ensures that the navigation links and content are dynamically populated based on the user’s permission/role status and preferences.

**Django ORM:** Loads user-specific data (roles, permissions, preferences) from the database to determine which navigation links are displayed and how the page is customized.

Logout Process Flow

**1. User Initiates Logout**

**Step:** The user clicks the ‘Logout’ button from the navigation bar.

**Package/Extension:**

**Django:** Handles the logout request by directing the user to a logout view.

**Process:** The logout request is sent to Django-Auth, where it processes the user's session termination.

**2. Session Termination (Handled by Redis)**

**Step:** Django-Auth terminates the active user session stored in Redis. This step ensures that the session is invalidated, and the user’s data is cleared.

**Package/Extension:**

**Redis:** The session data stored in Redis is cleared, preventing the user from accessing any protected parts of the app after logout.

**Django-user-sessions:** Logs the session end time and updates the session history to mark the session as terminated.

**3. Clear Django Authentication**

**Step:** The user’s authentication cookies and tokens are cleared by Django to fully log out the user from the system.

**Package/Extension:**

**Django:** Clears the authentication cookies and any stored tokens in the browser, making sure the user cannot reauthenticate without logging in again.

**4. Log Logout Event with Django Simple History**

**Step:** The user’s logout is logged by Django Simple History to keep track of all login and logout events in the system.

**Package/Extension:**

**Django Simple History:** Records the logout event, capturing details such as the time of logout, the user ID, and any relevant session information.

**5. Redirect to Logout Confirmation Page**

**Step:** After the session is terminated, the user is redirected to a logout confirmation page (e.g., logout.html) that confirms they have successfully logged out.

**Package/Extension:**

**Django:** Redirects the user to the logout confirmation page. This page confirms the user’s logout and offers options, such as returning to the login page or accessing public parts of the site.

**6. Ensure User Cannot Access Previous Pages**

**Step:** Prevent the user from accessing any previously cached pages by clearing the browser cache or using cache-control headers.

**Package/Extension:**

**Django:** Implements cache-control headers or other methods to ensure that once logged out, the user cannot access cached pages by hitting the browser’s back button.

Session Logout Flow

**1. Session Timeout Warning:**

**Step:** Before the session expires, the system displays a warning modal (‘session\_timeout\_warning.html’) to inform the user that their session is about to expire.

**Package/Extension:**

**Redis:** Tracks session expiration and session data.

**Django-user-sessions:** Monitors session activity and determines when the session is nearing expiration.

**Custom Modal:** A session timeout warning modal is triggered via JavaScript to notify the user, allowing them to extend the session if needed.

**2. User Attempts to Extend Session:**

**Step:** The user can choose to extend their session by interacting with the timeout warning modal, which will refresh the session and prevent automatic logout.

**Package/Extension:**

**Django-user-sessions:** Updates the session data and resets the expiration timer if the user opts to extend their session.

**Redis:** Updates the active session data, extending the session.

**3. Session Expiration:**

**Step:** If the user does not respond to the session timeout warning or the session naturally expires, Redis and Django-user-sessions terminate the active session.

**Package/Extension:**

**Redis:** Clears the session from memory.

**Django-user-sessions:** Logs the session end time and updates the session history, marking the session as terminated.

**4. Force Logout:**

**Step:** Once the session expires, the user is automatically logged out, and Django-Auth terminates the user’s authentication state.

**Package/Extension:**

**Django-Auth:** Handles clearing the authentication cookies and session data, logging the user out completely.

**Redis:** Ensures session data is fully removed from the cache.

**5. Redirect to Login Page:**

**Step:** After being logged out, the user is redirected to the login page (login.html) to re-authenticate if they want to continue using the system.

**Package/Extension:**

**Django:** Redirects the user to the login page once the session has expired.

**6. Record Session Expiration Event with Django Simple History:**

**Step:** The session expiration event is logged by Django Simple History for audit and tracking purposes.

**Package/Extension:**

**Django Simple History:** Records the session termination event, including the time of expiration and the user ID.