|  |  |  |
| --- | --- | --- |
| RAZDA Co. | | |
| **Filename: [ui\_loader.py]** | | |
| **Summary***:*  This file manages the loading of user interface (UI) elements based on the role of the user. It leverages caching, session data, and permission management to ensure users only see UI components relevant to their roles (admin, sub-admin, market admin, or customer). The UILoader class dynamically loads, caches, and retrieves UI elements, aiming for efficiency and scalability, especially in high-traffic environments. | | |
| ***Processes*** | | |
| * **Initialize UILoader** | **Instantiates the UILoader with a user\_id, determines the user's role, and prepares to load the appropriate UI elements.** | |
| * **Retrieve User Role** | **Uses session data to fetch the user’s role, querying the database via the Permissions class if not already stored. This role is cached for faster access.** | |
| * **Load UI Elements** | **Retrieves UI elements for the user based on their role. If the UI for a specific role is already in cache, it retrieves it from there to save loading time.** | |
| * **Fetch UI Elements Based on Role** | **Dynamically loads UI components for specific roles (admin, sub-admin, market admin, customer). Each role has access to different sections of the dashboard, modularized for easy access control.** | |
| * **Admin UI Loading** | **Loads admin-specific UI elements, providing access to settings, user management, analytics, and alerts.** | |
| * **Sub-admin UI Loading** | **Loads UI components specific to sub-admins, including access to customer support and content moderation tools.** | |
| * **Market Admin UI Loading** | **Loads UI elements for market admins, with a focus on order management, inventory, and sales reporting.** | |
| * **Customer UI Loading** | **Loads UI elements for customers, such as profile, cart, wishlist, and order history pages.** | |
| * **Clear UI Cache** | **Clears the cached UI for all users, typically triggered after role updates or UI changes, ensuring that users see the most up-to-date interface.** | |
| **Files it Gets Information From:** | | **Files it Sends too:** |
| * **Permissions Management** (permissions\_management.py): Retrieves the user's role from the Permissions class if not found in the session data. | * **Session**: Stores the user role in session data, reducing future lookups and improving loading speed. | |
| * **Session Data**: Verifies the logged-in user's session to determine user\_id and role. | * **Log File** (ui\_loader.log): Logs actions, role detections, and UI load times for performance monitoring and debugging. | |
|  | * **Cache**: Caches the loaded UI elements for faster retrieval on subsequent loads. | |
| **Expected input into file:** | | **Expected output from file:** |
| - User ID: Passed during UILoader initialization to identify the user and load their specific UI components.  - User Role: Retrieved from session or determined via permissions management to define the relevant UI elements. | | - UI Elements: Returns a dictionary of UI paths specific to the user’s role, directing them to accessible sections of the dashboard.  - Log Entries: Logs role detection, UI loading status, cache retrieval, and load times to ui\_loader.log for performance and monitoring.  - Flash Messages: Provides feedback to the user for unauthorized access attempts or missing session data. |
| **Things that need to be taking place:** | | |
| |  | | --- | |  | | **● Session and Role Verification: Ensures each user’s role is correctly identified from session or database, using cached results where possible to optimize loading.** | |  |  | | --- | | **● Caching: Implements caching for UI elements based on user role, reducing load times and server resource usage in high-traffic situations. TTLCache is used to control cache expiration, refreshing UI components every 5 minutes.** |  |  | | --- | | **● Logging: Records all major actions in ui\_loader.log, including cache hits, role determination, UI load times, and cache clearing events, aiding in tracking and diagnosing UI performance issues.** |  |  | | --- | | **● Error Handling and User Feedback: In cases of unauthorized access or session issues, appropriate flash messages are shown, guiding the user back to the login page or home.** |  |  | | --- | | **● Modularized UI Loading: Divides the UI into modular sections, allowing each role (admin, sub-admin, market admin, and customer) to access only their relevant dashboard components.** |  |  | | --- | | **● Asynchronous Execution: Utilizes a ThreadPoolExecutor for potential asynchronous UI loading, preparing the file for future expansion to handle multiple UI load requests simultaneously in high-traffic scenarios.** | | | |
| Edit log (update each time you make changes to doc or file). | | |
| **Contribution to the Entire Project:**  The **ui\_loader.py** file is integral to creating a dynamic, efficient, and role-based user experience across the platform. By leveraging caching and modularized loading, it minimizes load times and ensures that users only see the relevant parts of the interface, enhancing security and usability. Its well-structured approach to role-based access control allows for streamlined navigation and an intuitive user interface that adapts dynamically to each user’s permissions. | | |
| - Oliver Smith (Razda Admin) Nov 8, 2024: Set up UI loader with caching and asynchronous loading. Integrated role-based access control for dynamic UI rendering and modularized UI loading by user role. Configured logging for load times, cache events, and role-based access. | | |