Documentation of User Management Subsystem

Note - Getter, Setter functions, and Constructors have their usual meanings and are not mentioned here.

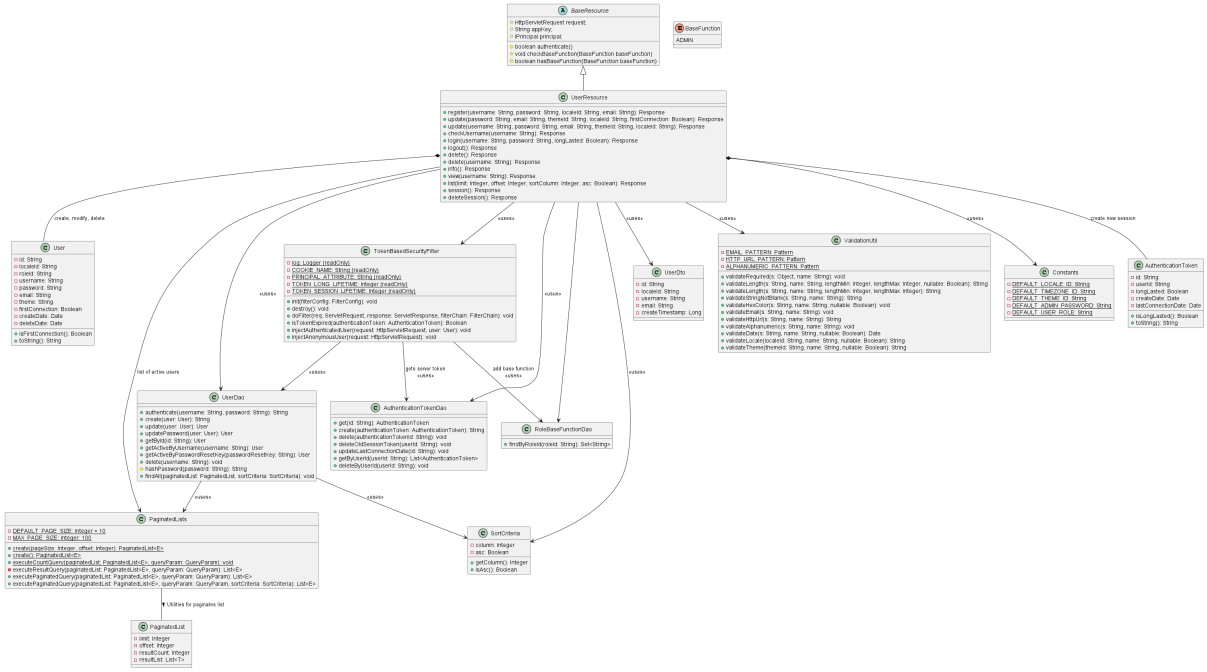

Figure 1: Class Diagram for User Management Sub System

The UserManagement system is used for addition, deletion and modification of users. It is managed by the UserResource controller which inherits BaseResource. It uses UserDao for database access. It uses UserDto for data transfer in the backend. It uses User class as the entity class, which is mapped to the database table.

UserResource

The **UserResource** class is designed to provide RESTful endpoints for managing user-related operations within the application. This includes creating new users, updating user information, authenticating users, and managing user sessions.

Methods

- **Response register(`String username, String password, String localeId, String email`)** [**PUT /user**]: Creates a new user account with the specified username, password, email, and locale. Validates the provided information before creating the user.
- **Response update(`String password, String email, String themeId, String localeId, Boolean firstConnection`)** [**POST /user**]: Updates the information of the currently authenticated user, including password, email, theme, and locale preferences. Allows marking the first connection wizard as acknowledged.
- **Response update(`String username, String password, String email, String themeId, String localeId`)** [**POST /user/{username}**]: Updates the specified user's information based on the provided username. This endpoint requires administrative privileges to access.
- **Response checkUsername(`String username`)** [**GET /user/check_username**]: Checks if the provided username is available for registration. Returns status indicating availability.
- **Response login(`String username, String password, boolean longLasted`)** [**POST /user/login**]: Authenticates the user and creates a session. Optionally creates a long-lasting session if requested.
- **Response logout()** [**POST /user/logout**]: Logs out the currently authenticated user and deletes the active session token.
- **Response delete()** [**DELETE /user**]: Deletes the currently authenticated user account. This action is irreversible.
- **Response delete(`String username`)** [**DELETE /user/{username}**]: Deletes the user account specified by the username. This endpoint requires administrative privileges.
- **Response info()** [**GET /user**]: Retrieves information about the currently authenticated user, including username, email, theme, and locale.
- **Response view(`String username`)** [**GET /user/{username}**]: Fetches information about a specific user identified by username. Requires administrative privileges to access.
- **Response list(`Integer limit, Integer offset, Integer sortColumn, Boolean asc`)** [**GET /user/list**]: Returns a paginated list of all active users, sorted according to the specified criteria. Accessible only to administrators.
- **Response session()** [**GET /user/session**]: Retrieves all active sessions for the currently authenticated user. Allows users to manage their own sessions.
- **Response deleteSession()** [**DELETE /user/session**]: Deletes all active sessions for the currently authenticated user except for the session used to make this request. Helps in managing session security.

BaseResource

The **BaseResource** class serves as the abstract base class for REST resources in the application, providing common functionalities such as authentication checks and permission validations.

Attributes

- `protected HttpServletRequest request`: Injects the HTTP request to allow access to request information within resource classes.
- `protected String appKey`: A query parameter that might be used for application-level authentication or API key validation.
- `protected IPrincipal principal`: Represents the principal of the authenticated user, providing an interface to user identity and roles.

Methods

- **boolean authenticate()**: Checks if the user is authenticated and not anonymous. It retrieves the user principal from the request attributes, set by a security filter, and checks if the user is not marked as anonymous.
- **void checkBaseFunction(**`BaseFunction baseFunction`**)**: Validates if the authenticated user has a specific base function (role or permission). Throws `JSONException` if the user does not have the required base function.
- **boolean hasBaseFunction(**`BaseFunction baseFunction`**)**: Checks if the authenticated user has a specific base function. Returns a boolean indicating if the user has the specified base function. Throws `JSONException`.

User

The **User** class represents a user entity in the system, containing information such as user ID, username, password, email, and creation date.

Attributes

- `String id`: User ID.
- `String localeId`: Locale ID.
- `String roleId`: Role ID.
- `String username`: User's username.
- `String password`: User's password.
- `String email`: Email address.
- `String theme`: Theme.
- `boolean firstConnection`: True if the user hasn't dismissed the first connection screen.
- `Date createDate`: Creation date.
- `Date deleteDate`: Deletion date (optional).

Methods

- **`boolean isFirstConnection()`**: Returns true if the user hasn't dismissed the first connection screen.

UserDao

The **UserDao** class is responsible for managing user-related database operations, such as authentication, creation, updating, deletion, and retrieval of user information.

Methods

- **String authenticate(**`String username`**,** `String password`**)**: Authenticates a user given the username and password. Returns the ID of the authenticated user or null if authentication fails.
- **String create(**`User user`**)**: Creates a new user in the database. Throws an exception if the username already exists. Returns the ID of the newly created user.
- **User update(**`User user`**)**: Updates the information of an existing user in the database.
- **User updatePassword(**`User user`**)**: Updates the password of an existing user in the database.
- **User getById(**`String id`**)**: Retrieves a user from the database based on the user ID.
- **User getActiveByUsername(**`String username`**)**: Retrieves an active user from the database based on the username.
- **User getActiveByPasswordResetKey(**`String passwordResetKey`**)**: Retrieves an active user from the database based on the password reset token.
- **void delete(**`String username`**)**: Deletes a user from the database based on the username.
- **void findAll(**`PaginatedList<UserDto> paginatedList`**,** `SortCriteria sortCriteria`**)**: Retrieves a paginated list of all users from the database, including their ID, username, email, creation timestamp, and locale ID.

UserDao

The **UserDao** class is responsible for managing user-related database operations, such as authentication, creation, updating, deletion, and retrieval of user information.

Methods

- **String authenticate**(`String username`, `String password`): Authenticates a user given the username and password. Returns the ID of the authenticated user or null if authentication fails.
- **String create**(`User user`): Creates a new user in the database. Throws an exception if the username already exists. Returns the ID of the newly created user.
- **User update**(`User user`): Updates the information of an existing user in the database.
- **User updatePassword**(`User user`): Updates the password of an existing user in the database.
- **User getById**(`String id`): Retrieves a user from the database based on the user ID.
- **User getActiveByUsername**(`String username`): Retrieves an active user from the database based on the username.
- **User getActiveByPasswordResetKey**(`String passwordResetKey`): Retrieves an active user from the database based on the password reset token.
- **void delete**(`String username`): Deletes a user from the database based on the username.
- **void findAll**(`PaginatedList<UserDto> paginatedList`, `SortCriteria sortCriteria`): Retrieves a paginated list of all users from the database, including their ID, username, email, creation timestamp, and locale ID.

UserDto

The **UserDto** class represents a Data Transfer Object (DTO) containing user information, including user ID, locale ID, username, email address, and creation timestamp.

Attributes

- `String id`: User ID.
- `String localeId`: Locale ID.
- `String username`: Username.
- `String email`: Email address.
- `Long createTimestamp`: Creation timestamp of this user.

ValidationUtil

The **ValidationUtil** class provides utility methods for validating parameters, such as strings, emails, URLs, and dates. It also includes methods for validating locales and themes.

Methods

- **void validateRequired**(`Object s`, `String name`): Checks that the argument is not null.
- **String validateLength**(`String s`, `String name`, `Integer lengthMin`, `Integer lengthMax`, `boolean nullable`): Validates a string length.
- **String validateLength**(`String s`, `String name`, `Integer lengthMin`, `Integer lengthMax`): Validates a string length. The string mustn't be empty.
- **String validateStringNotBlank**(`String s`, `String name`): Checks if the string is not null and is not only whitespaces.
- **void validateHexColor**(`String s`, `String name`, `boolean nullable`): Checks if the string is a hexadecimal color.
- **void validateEmail**(`String s`, `String name`): Checks if the string is an email.
- **String validateHttpUrl**(`String s`, `String name`): Validates that the provided string matches an URL with HTTP or HTTPS scheme.
- **void validateAlphanumeric**(`String s`, `String name`): Checks if the string uses only alphanumerical or underscore characters.
- **Date validateDate**(`String s`, `String name`, `boolean nullable`): Validates and parses a date.
- **String validateLocale**(`String localeId`, `String name`, `boolean nullable`): Validates a locale.
- **String validateTheme**(`String themeId`, `String name`, `boolean nullable`): Validates a theme.

AuthenticationToken

The **AuthenticationToken** class represents an authentication token entity in the application.

Attributes

- `String id`: Token.
- `String userId`: User ID.
- `boolean longLasted`: Indicates whether to remember the user next time (long-lasting session).
- `Date creationDate`: Token creation date.
- `Date lastConnectionDate`: Last connection date using this token.

AuthenticationTokenDao

The **AuthenticationTokenDao** class provides data access methods for managing authentication tokens in the application.

Methods

- **AuthenticationToken get(`String id`)**: Retrieves an authentication token by its ID.
- **String create(`AuthenticationToken authenticationToken`)**: Creates a new authentication token and returns its ID.
- **void delete(`String authenticationTokenId`)**: Deletes the authentication token with the specified ID.
- **void deleteOldSessionToken(`String userId`)**: Deletes old short-lived tokens associated with the specified user ID.
- **void updateLastConnectionDate(`String id`)**: Updates the last connection date for the authentication token with the specified ID.
- **List<AuthenticationToken> getByUserId(`String userId`)**: Returns all authentication tokens associated with a specific user ID.
- **void deleteByUserId(`String userId, String id`)**: Deletes all authentication tokens associated with a specific user ID, except for the token with the specified ID.

TokenBasedSecurityFilter

The **TokenBasedSecurityFilter** class provides authentication functionality based on authentication tokens stored in cookies.

Constants

- `COOKIE_NAME`: Name of the cookie used to store the authentication token.
- `PRINCIPAL_ATTRIBUTE`: Name of the attribute containing the principal.
- `TOKEN_LONG_LIFETIME`: Lifetime of the authentication token in seconds since login.
- `TOKEN_SESSION_LIFETIME`: Lifetime of the authentication token in seconds since the last connection.

Methods

- **void init(`FilterConfig filterConfig`)**: Initializes the filter.
- **void destroy()**: Destroys the filter.
- **void doFilter(`ServletRequest req, ServletResponse response, FilterChain filterChain`)**: Performs the authentication process for incoming requests.
- **boolean isTokenExpired(`AuthenticationToken authenticationToken`)**: Checks if the given authentication token is expired.
- **void injectAuthenticatedUser(`HttpServletRequest request, User user`)**: Injects an authenticated user into the request attributes.
- **void injectAnonymousUser(`HttpServletRequest request`)**: Injects an anonymous user into the request attributes.

RoleBaseFunctionDao

The **RoleBaseFunctionDao** class provides functionality to find the set of base functions associated with a role.

Methods

- **Set<String> findByRoleId(`String roleId`)**: Finds the set of base functions associated with the given role ID.

PaginatedList<T>

The **PaginatedList<T>** class represents a paginated list of items.

Attributes

- `int limit`: The size of a page.
- `int offset`: The offset of the page (in number of records).
- `int resultCount`: The total number of records.
- `List<T> resultList`: The list of records of the current page.

PaginatedLists

The **PaginatedLists** class provides utilities for working with paginated lists.

Attributes

- `int DEFAULT_PAGE_SIZE`: The default size of a page.
- `int MAX_PAGE_SIZE`: The maximum size of a page.

Methods

- **create(**`Integer pageSize, Integer offset`**)**: Constructs a paginated list with the specified page size and offset. If `pageSize` is `null`, the default page size is used. If `offset` is `null`, the offset is set to 0.
- **create():** Constructs a paginated list with default parameters.
- **executeCountQuery(**`PaginatedList<E> paginatedList, QueryParam queryParam`**)**: Executes a native `count(*)` request to count the number of results and sets the result count in the `paginatedList` object.
- **executePaginatedQuery(**`PaginatedList<E> paginatedList, QueryParam queryParam`**)**: Executes a paginated request with two native queries (one to count the number of results and one to return the page) and returns the list of results.
- **executePaginatedQuery(**`PaginatedList<E> paginatedList, QueryParam queryParam, SortCriteria sortCriteria`**)**: Executes a paginated request with two native queries (one to count the number of results and one to return the page) and returns the list of results sorted according to the `sortCriteria`.

SortCriteria

The **SortCriteria** class represents the sort criteria of a query.

Attributes

- `int column`: Index of the column to sort (starting from 0).
- `boolean asc`: Indicates whether to sort in increasing order (`true`) or decreasing order (`false`).

Methods

- **getColumn(): int**: Returns the index of the column to sort.
- **isAsc(): boolean**: Returns `true` if sorting is in increasing order, `false` if sorting is in decreasing order.

Relationships

1. **UserResource *– User**

   This relationship indicates that the UserResource class is responsible for performing CRUD (Create, Read, Update, Delete) operations on User entities. It can create, modify, and delete user records.

2. **UserResource ”1” *– ”1” UserDao**

   UserResource has a one-to-one composition relationship with UserDao. This means that each instance of UserResource is associated with one instance of UserDao to get information of the User entity from the database.

3. **UserDao ”1” o– ”0..*” User**

   UserDao has an aggregation relationship with User, suggesting that UserDao manages interactions with multiple instances of User entities.

4. **UserResource –> UserDto**

   UserResource utilizes UserDto for transferring user-related data.

5. **UserResource –> Constants**

   UserResource uses Constants for accessing constant values or configurations related to user management.

6. **UserResource –> ValidationUtil**

   UserResource utilizes ValidationUtil for validating input data related to user operations.

7. **UserResource *– AuthenticationToken**

   This relationship indicates that UserResource is responsible for creating new authentication sessions using AuthenticationToken.

8. **UserResource ”1” *– ”1” AuthenticationTokenDao**

   UserResource and AuthenticationTokenDao have a composition relationship as AuthenticationTokenDao is a data access object.

9. **UserResource –> TokenBasedSecurityFilter**

   UserResource uses TokenBasedSecurityFilter for handling security-related tasks, such as authentication and authorization.

10. **UserResource ”1” *– ”1” RoleBaseFunctionDao**

    UserResource and RoleBaseFunctionDao have a composition relationship

11. **TokenBasedSecurityFilter – AuthenticationTokenDao**

    TokenBasedSecurityFilter interacts with AuthenticationTokenDao to retrieve server tokens for authentication purposes.

12. **TokenBasedSecurityFilter – RoleBaseFunctionDao**

    TokenBasedSecurityFilter communicates with RoleBaseFunctionDao to add base functions for authorization purposes.

13. –> UserDao

    TokenBasedSecurityFilter utilizes UserDao for accessing user information during security checks.

14. **UserResource –> PaginatedLists**

    UserResource utilizes PaginatedLists for managing lists of active users.

15. **PaginatedLists – PaginatedList**

    PaginatedLists contains utilities for paginating lists of data, and PaginatedList represents an instance of a paginated list.

16. **UserResource –> SortCriteria**

    UserResource uses SortCriteria for specifying sorting criteria when retrieving user data.

17. **UserDao –> PaginatedLists**

    UserDao utilizes PaginatedLists for managing paginated lists of user data.

18. **UserDao –> SortCriteria**

    UserDao uses SortCriteria for specifying sorting criteria when querying user data.