Package: Authenticate (via r-universe)

September 14, 2024

Title R Shiny Authentication Module

Version 1.0.0.0000

Description R Shiny Authentication Module which includes View, Controller and Data Layer.

License file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports shiny

Suggests testthat (>= 3.0.0), uuid, Validate, Environment, Query, Storage, digest

Remotes FlippieCoetser/Validate, FlippieCoetser/Environment, FlippieCoetser/Query, FlippieCoetser/Storage

Config/testthat/edition 3

Depends R (>= 2.10)

Repository https://flippiecoetser.r-universe.dev

RemoteUrl https://github.com/FlippieCoetser/Authenticate

RemoteRef HEAD

RemoteSha 151d4cd393ec313cc6fd738e0c79fb325a1c3d24

Contents

Controller 2
Encryption.Processor 3
Encryption.Service 3
Orchestrator 4
User.Broker 4
User.Model 5
User.Service 5

User.Validation.Exceptions 6
User.Validator 7
Users 7
View 8

Index **9**

`Controller` *Authentication Controller for Shiny Application Plugin*

Description

This function manages user authentication within a Shiny application. It sets up a module server for handling user interactions related to login, registration, and session management. It includes mechanisms to validate user inputs, manage user session states, and dynamically update UI elements based on the authentication state.

Usage

```
Controller(
  id,
  storage,
  user = shiny::reactiveValues(),
  title = "Authenticate",
  debug = FALSE
)
```

Arguments

`id`	A unique identifier for the Shiny module.
`storage`	A storage backend for managing user data.
`user`	An optional reactive values with cached user details.
`title`	A character string representing the title of the authentication dialog.
`debug`	A logical value indicating whether to print debug messages.

Details

The function creates a series of event observers and reactive expressions that:

- Handle guest and registered user logins.
- Manage user registration.
- Authenticate user credentials.
- Provide functionality for user logout and cancellation of operations.
- Dynamically control the visibility of UI elements such as username display and login/logout buttons based on the user's authentication status.

Each user interaction is validated through a set of predefined validation functions, and the UI is updated accordingly to reflect the current state. Errors in user input are handled gracefully, providing modal dialogs to inform users of specific issues.

Value

A Shiny module server function.

Encryption.Processor *Encryption Processor Constructor*

Description

Creates a processor that utilizes an encryption service to handle password hashing operations within a model object. This processor relies on an external encryption service, passed as a parameter, to hash passwords using the service's specific cryptographic methods.

Usage

```
Encryption.Processor(service)
```

Arguments

`service`	An encryption service object that provides cryptographic functions, such as password hashing. This service should include a `Hash.Password` function that accepts a password and salt, returning a hashed password.

Value

A list of functions that perform operations on model data, specifically:

`Set.Hash` Updates a model object with a hashed password. The model must have a `hash` field and a `salt` field. The password is hashed using the provided salt and the hash function from the encryption service.

- `model`: The model object containing the user data.
- `password`: The plain text password to be hashed.

Returns the modified model object with the hashed password.

Encryption.Service *Encryption Service Constructor*

Description

Creates an encryption service that provides cryptographic functionalities, specifically for hashing passwords. This service uses SHA-512 hashing algorithm to combine a password with a salt and generate a hashed output.

Usage

```
Encryption.Service()
```

Value

A list containing cryptographic functions, including:

`Hash.Password` Combines a password with a salt and computes a SHA-512 hash.

- `password`: The plain text password to be hashed.
- `salt`: A string used to salt the password to enhance security.

Returns a hashed string using SHA-512 algorithm.

`Orchestrator` *Authentication Orchestration Service*

Description

Orchestration Service used to:

- Register User
- Check Username
- Authenticate User

Usage

```
Orchestrator(storage)
```

Arguments

`storage`	The storage provider to use by orchestration service

`User.Broker` *User.Broker Component*

Description

This component provides operations for managing user data in the storage.

Usage

```
User.Broker(storage)
```

Arguments

`storage`	The storage object used for data persistence.

Value

A list of operations for managing user data.

`User.Model` *User Business Entity*

Description

User Business Entity is a data.frame with the following attributes:

- id
- username
- hash
- salt

Both id and salt are UUIDs and auto-generated on instantiation.

Usage

```
User.Model(username)
```

Arguments

`username`	The username of the user

`User.Service` *User Service Constructor*

Description

Constructs a service layer for user management that integrates validation with database operations. This function provides an interface to add, retrieve, update, and delete user records, ensuring that all operations are preceded by appropriate validations.

Usage

```
User.Service(broker)
```

Arguments

`broker`	An object that handles the data storage operations, typically a database broker with methods like `Insert`, `Select`, `Update`, and `Delete`. This broker must support the operations expected by each service function.

Value

A list of functions that manage user data, ensuring validation and interaction with the data storage layer:

`Add` Validates and adds a user to the database.

`Retrieve` Retrieves all users from the database.

`RetrieveById` Retrieves a user by their unique ID after validating the ID format.

`Update` Validates and updates a user record in the database.

`Delete` Deletes a user record from the database after validating the ID format.

`User.Validation.Exceptions`

User Validation Exceptions Constructor

Description

Constructs a list of exception handling functions specifically for user validation. This function provides a standardized way to handle and throw exceptions related to user data integrity. Each exception function is designed to stop execution with a specific error message if invoked.

Usage

```
User.Validation.Exceptions()
```

Details

The available exceptions are:

- `User.NULL`: Triggered when a user object is expected but not provided.
- `Attribute.NULL`: Triggered when a required attribute of the user object is missing.

Value

A list of functions, each corresponding to a specific type of exception related to user validation. Functions are invoked with parameters controlling whether the exception should be thrown and provide custom error messages.

User.Validator *User Validator Component*

Description

This function creates a suite of validators for checking the integrity of user data. It encapsulates various checks to ensure that user objects meet expected criteria, such as non-null values for essential attributes and specific format validations. Each validator function is designed to throw a specific exception from `User.Validation.Exceptions` if the validation fails.

Usage

```
User.Validator()
```

Value

A list of validator functions, which includes:

`User` Validates a complete user object by sequentially applying all individual attribute checks.

`Exists` Checks if the user object is not NULL.

`HasId`, `HasUsername`, `HasHash`, `HasSalt` Ensures that each respective attribute is not NULL.

`Id` Validates that the user ID is in UUID format.

Users *Users Dataset*

Description

This dataset contains user information for a sample application. Each record represents a user with a unique identifier, username, hashed password, and associated salt for additional security during the hashing process. This data can be used for authentication system examples, security demonstrations, or testing user management functionalities.

Usage

```
Users
```

Format

A data frame with 3 rows and 4 columns:

id Unique identifier for the user, stored as a UUID string.

username Email address used as the username.

hash SHA-512 hashed password, as a hex string, for user authentication.

salt UUID string used as a salt for the password hash.

Source

Generated synthetic data.

`View` *Authentication View for Shiny Application*

Description

This function creates a user interface for managing authentication displays within a Shiny application. It provides reactive UI elements that show the current user's username and offer login or logout actions depending on the user's authentication state.

Usage

```
View(id)
```

Arguments

`id`	A unique identifier for the Shiny module which scopes the UI elements.

Details

The view consists of:

- A username display that only appears if the user is logged in.
- A login action link that is visible when the user is not logged in.
- A logout action link that appears when the user is logged in.

The visibility of these elements is controlled by Shiny's server-side logic which outputs reactivity conditions. These conditions are set based on the user's authentication status, ensuring that UI elements reflect the current state appropriately.

Value

A Shiny UI component that includes conditional panels for user authentication management.

Index

∗ **datasets**
    Users, 7

Controller, 2

Encryption.Processor, 3
Encryption.Service, 3

Orchestrator, 4

User.Broker, 4
User.Model, 5
User.Service, 5
User.Validation.Exceptions, 6
User.Validator, 7
Users, 7

View, 8