Java API Documentation

12 Classes | 0 Interfaces

Classes

LoginController

The LoginController class handles user authentication and registration, provides username retrieval, and allows creating tasks.

Methods:

- **LoginController**(void): public — The provided code snippet refers to a class named `LoginController` within a Java application, likely part of a web-based or Spring Boot framework. However, no actual code content is included for analysis.
- **login**(String, String): boolean — The `LoginController.login` method is a backend endpoint in a web application that handles user authentication requests. It receives login credentials (typically username and password), validates them against stored data (e.g., via a service or repository), and returns a boolean value indicating success or failure of the login attempt.
- **register**(String, String, String): boolean — The method `LoginController.register` is likely responsible for handling user registration logic within a web application. As a method in the `LoginController` class, it typically processes incoming requests to register new users by validating input data (such as username, email, password), checking for duplicates, and persisting the user information to a database or storage system.
- **getUsername**(void): String
- **createTask**(String): void
- **launchMainView**(void): void
- **launchAddTask**(void): void

TasksController

The TasksController manages task data by loading it from a database and providing access to tasks based on filters, while ensuring singleton pattern through private constructor and instance retrieval.

Methods:

- **TasksController**(void): private — The provided snippet references `TasksController.TasksController`, but no actual Java code is included for analysis. Based on the context:
- **getInstance**(void): TasksController — The expression `TasksController.getInstance()` retrieves a singleton instance of the `TasksController` class.
- **setMainView**(MainView): void — The method `TasksController.setMainView` is a member function within the `TasksController` class, responsible for setting or updating the main view associated with the controller. Although the provided code snippet is incomplete, based on naming and context:
- **loadTasksFromDatabase**(void): void
- **getTasks**(boolean, boolean): List
- **reloadTasksInMainView**(void): void
- **deleteTask**(int): void
- **CreateTask**(String, String, String): void
- **setTaskAsFinished**(int): void
- **login**(String, String): boolean
- *(+1 more methods)*

Launch

The Launch class handles application initialization, detects whether it is the first launch, and manages splash screen display.

Methods:

- **main**(String[]): void — The provided information is insufficient to provide a technical description of the `Launch.main` method.
- **isFirstLaunch**(void): boolean — The `Launch.isFirstLaunch` method is a static utility method in the `Launch` class that determines whether the application is being launched for the first time. It typically checks a persistent storage mechanism (such as a preference file, database, or configuration flag) to track if the app has been previously run. If no prior launch record exists, it returns `true`, indicating this is the first launch; otherwise, it returns `false`.
- **setSplashShown**(void): void — The method `Launch.setSplashShown` is a static method in the `Launch` class that likely controls whether a splash screen is displayed during application startup.

AppState

AppState manages application state, including detecting first launches and tracking splash screen display.

Methods:

- **main**(String[]): void — The method `AppState.main` is the entry point of a Java application, typically defined within the `AppState` class. As a static method, it serves as the starting point when the program is executed via the `java` command. Its primary purpose is to initialize the application's state and perform essential setup tasks—such as creating instances, configuring system components, or launching core functionality—before any other operations begin.
- **isFirstLaunch**(void): boolean — The expression `AppState.isFirstLaunch` is a static boolean property (or method) in the `AppState` class that indicates whether the application has been launched for the first time.
- **setSplashShown**(void): void — The method `AppState.setSplashShown` is a static method in the `AppState` class that sets a boolean flag indicating whether the splash screen has been displayed. Its primary purpose is to track the state of splash screen visibility during application startup.

Task

A Task class represents a task with properties such as ID, user ID, name, type, and due date.

Methods:

- **getId**(void): int — The method `getById` in the `Task` class is not valid based on standard Java naming conventions and semantics. A method named `getById` typically belongs to a repository or service layer that retrieves an entity (like a Task) by a unique identifier (e.g., ID). However, given the context that "the method returns int," this implies a misalignment.
- **getUserId**(void): int — The method `getUserId()` in the `Task` class is a public accessor method that retrieves an integer value representing the user ID associated with the task. Its primary purpose is to provide read-only access to the user identifier stored within the Task object, enabling other components of the system to identify or associate the task with a specific user. The method does not modify any state; it simply returns the current `userId` field (of type `int`). It assumes that the `userId` is a persistent attribute of the task and is typically used in contexts such as authentication, authorization, logging, or auditing. No parameters are required, and no side effects occur—making it a straightforward getter method.
- **getName**(void): String — The method `Task.getName()` is a public instance method within the `Task` class that returns a `String` value representing the name of the task. Its purpose is to provide access to the task's identifier or descriptive label, typically used for display, logging, or identification in user interfaces, configuration systems, or workflow management. The method assumes that the task object has a named property (e.g., stored as a private field) and exposes it via getter semantics, following standard Java encapsulation practices. No parameters are required, and no side effects occur—its behavior is strictly to retrieve and return the name string.
- **getTaskType**(void): String
- **getDueDate**(void): String
- **getRecurrencePattern**(void): String

- **isCompleted**(void): boolean
- **getCreatedAt**(void): LocalDateTime
- **isBacklog**(void): boolean
- **toString**(void): String
- *(+1 more methods)*

User

A User class represents a user with a username and password, allowing retrieval and modification of these credentials.

Methods:

- **User**(String, String): public — The provided snippet "User.User" and context indicating "Class: User, Method returns public" appear to be incomplete or malformed. A proper Java class named `User` would typically define a structure for representing user data (e.g., name, email, ID), but the expression "User.User" is not valid Java syntax—it does not correspond to a method, constructor, or meaningful construct.
- **getUsername**(void): String — The method `getUsername()` is a getter method in the `User` class that retrieves and returns the username associated with an instance of the `User` object. It follows Java's encapsulation principles by providing public access to the private or protected `username` field without exposing its internal implementation. The return type is `String`, indicating that it produces a textual identifier representing the user's name. This method is typically used for data retrieval in applications such as authentication, profile display, or user identification across systems. It assumes that the username has already been set during object creation or initialization and does not modify any state of the object.
- **getPassword**(void): String — The method `getPassword` in the `User` class is a getter method that retrieves the password value associated with a user instance. It returns a `String`, representing the stored password (typically encrypted or hashed for security).
- **setUsername**(String): void
- **setPassword**(String): void

DatabaseService

A DatabaseService class provides methods to retrieve credentials, establish a database connection, check if a database exists, and execute queries returning result counts.

Methods:

- **getUsername**(void): String — The `getUsername` method in the `DatabaseService` class is designed to retrieve a user's username from a database. It typically executes a query (e.g., using JDBC or an ORM like Hibernate) against a database table (such as `users`) to fetch the username field based on provided criteria—commonly a user ID, email, or other unique identifier.
- **getPassword**(void): String — The `getPassword` method in the `DatabaseService` class is designed to retrieve a password, typically from a secure storage or configuration source, for use in database authentication.
- **connect**(void): Connection — The `DatabaseService.connect()` method is a static utility method within the `DatabaseService` class that establishes a database connection. Its primary purpose is to initialize and return a `Connection` object representing an active session to a backend database (e.g., MySQL, PostgreSQL, etc.), enabling subsequent operations such as querying or updating data.
- **databaseExists**(void): boolean
- **executeQuery**(String): int
- **updateTask**(Task): int
- **login**(String, String): boolean
- **register**(String, String, String): boolean
- **getTasksByQuery**(String): List
- **extractTaskFromResultSet**(ResultSet): Task
- *(+10 more methods)*

UIStyleManager

The UIStyleManager class handles theme application, font loading, rounded frame styling, component resizing events, and high-quality image drawing for graphical user interface elements.

Methods:

- **applyDarkTheme**(void): void — The method `UIStyleManager.applyDarkTheme()` is a utility function within the `UIStyleManager` class that applies a dark-themed visual style to a user interface (UI).
- **loadFont**(String, float): Font — The `UIStyleManager.loadFont` method is a utility function within the `UIStyleManager` class that retrieves or loads a font resource for use in a user interface. Its primary purpose is to ensure consistent and styled text rendering across UI components by managing font assets dynamically.
- **applyRoundedFrame**(JFrame, int, int): void — The method `UIStyleManager.applyRoundedFrame` is responsible for applying a rounded frame style to UI components within a graphical user interface.
- **componentResized**(java.awt.event.ComponentEvent): void
- **qualityImageDraw**(Graphics, Component, Image): void

AddTaskView

AddTaskView is a Java class that provides a graphical interface for adding tasks, featuring event handling and a main method for execution.

extends `JFrame`

Methods:

- **AddTaskView**(JPanel): public — The class `AddTaskView` appears to be a UI component (likely part of a mobile or desktop application) responsible for providing a view interface to add new tasks. As a public class with a method returning public, it likely exposes functionality allowing users to interact with the task addition process—such as entering task details, submitting data, and possibly receiving confirmation or feedback.
- **AbstractAction**(void): new — The entity `AddTaskView.AbstractAction` appears to be an abstract class within the `AddTaskView` component of a

Java application. Based on naming conventions and context (e.g., "AbstractAction" in a view related to task management), it likely defines a base or shared behavior for actions associated with adding or managing tasks—such as submitting, canceling, or validating a new task.

- **actionPerformed**(ActionEvent): void — The method `AddTaskView.actionPerformed` is an event handler in the `AddTaskView` class that responds to user interactions (such as button clicks) within a graphical user interface. It typically receives an `ActionEvent` object as a parameter, which identifies the source of the action (e.g., a "Submit" or "Add" button). The method is responsible for processing this event—such as validating input data, creating a new task object, and updating the underlying model or view to reflect the added task. Its primary purpose is to translate user interface actions into logical operations within the application workflow. Since the method returns `void`, it performs side effects (like UI updates or data persistence) without returning a value. This method is central to making the add-task functionality interactive and responsive in a GUI-based Java application, likely using Swing or JavaFX event-driven architecture.
- **AddTaskView**(void): public — The class `AddTaskView` appears to be a UI component (likely part of a mobile or desktop application) responsible for providing a view interface to add new tasks. As a public class with a method returning public, it likely exposes functionality allowing users to interact with the task addition process—such as entering task details, submitting data, and possibly receiving confirmation or feedback.
- **main**(String[]): void

LoginView

The LoginView class is a GUI component that handles login functionality, including user interaction through action events and initialization via a main method.

Methods:

- **main**(void): void — The provided code snippet references `LoginView.main`, but no actual Java code is included for analysis. As a result, a technical description cannot be generated with sufficient detail.

- **AbstractAction**(void): new — The identifier `LoginView.AbstractAction` appears to refer to an abstract action class within a Java application's UI framework (likely using Swing or a similar component-based design), where `LoginView` is a view component responsible for handling user login functionality.
- **actionPerformed**(ActionEvent): void — The method `LoginView.actionPerformed` is an event handler in a Java Swing GUI application that responds to user actions, such as clicking a "Login" button. As part of the LoginView class (likely extending or implementing a view component), this method is typically triggered by action events from UI components like buttons.

MainView

The MainView class initializes and sets up a graphical user interface by configuring its style, layout, and main window through a series of dedicated methods.

extends `JFrame`

Methods:

- **MainView**(LoginController): public — The provided snippet "MainView.MainView" refers to a class named `MainView` in Java, likely indicating the start of a view or user interface component in a larger application (such as a desktop app or mobile UI framework). However, the code itself is incomplete — it only names the class and does not include actual implementation or method definitions.
- **initialize**(void): void — The method `MainView.initialize()` is a static initializer method in the `MainView` class that is responsible for setting up or configuring the initial state of the main view component. As a void-returning method with no explicit parameters, it typically performs one-time setup tasks such as initializing UI components, loading configuration data, establishing dependencies, or setting up event listeners necessary for the view to function properly upon first use.
- **setupUIStyle**(void): void — The method `MainView.setupUIStyle` is responsible for configuring the user interface (UI) appearance of the `MainView` class. It typically applies styling attributes such as font sizes,

colors, layout properties, or theme settings to ensure a consistent and visually coherent UI. Since it is named `setupUIStyle`, it likely initializes or modifies visual components (e.g., buttons, labels, containers) using predefined style rules—potentially leveraging Java Swing or JavaFX component styling mechanisms.

- **createMainWindow**(void): void
- **setupLayout**(void): void
- **createMessagePanel**(void): JPanel
- **setCurrentViewLabel**(String): void
- **createTaskPanel**(void): JPanel
- **createTaskScrollPane**(JPanel): JScrollPane
- **getTaskPanel**(void): JPanel
- *(+11 more methods)*

SplashScreen

A SplashScreen class that initializes and displays a splash screen with a specified duration.

extends `JWindow`

Methods:

- **SplashScreen**(void): public — The provided code snippet `SplashScreen.SplashScreen` refers to a class named `SplashScreen` that appears to be part of a Java application's UI initialization process. However, the given information is insufficient to fully analyze the actual implementation or behavior of the class.
- **showSplash**(int): void — The method `SplashScreen.showSplash()` is a static method belonging to the `SplashScreen` class. Its primary purpose is to display a splash screen—typically a brief graphical interface shown at application startup—to indicate that the program is initializing or loading.