Controller

Package: com.backend.server.controller

Description

The DataTheaterAPIS class is a Spring RestController that handles HTTP requests related to movies in the backend server. It exposes various endpoints to perform CRUD operations on movie data through the MovieService.

Endpoints

GET /movies/getAllMovies

Description:

Retrieves a list of all movies available in the system.

HTTP Method: GET

Parameters: None

Produces: JSON (MediaType.APPLICATION\_JSON\_VALUE)

Returns:

(Type: List<Movies>) A list of all movies available in the system.

GET /movies/getByStatus

Description:

Retrieves a list of movies based on the provided status.

HTTP Method: GET

Parameters:

status (Type: String) - The status to filter the movies.

Returns:

(Type: List<Movies>) A list of movies that match the given status.

POST /movies/addAllMovies

Description:

Saves multiple movies in the system.

HTTP Method: POST

Parameters:

movies (Type: List<Movies>) - A list of Movies objects to be added to the system.

Returns: None

POST /movies/addMovie

Description:

Saves a single movie in the system.

HTTP Method: POST

Parameters:

movie (Type: Movies) - The Movie object to be added to the system.

Returns:

(Type: Movies) The saved Movie object with its generated ID.

Annotations

@RestController: Indicates that this class is a Spring RestController, allowing it to handle HTTP requests and automatically convert the response to JSON format.

@RequestMapping("/movies"): Maps all the endpoints in this controller to the base URL "/movies".

@CrossOrigin(origins = "http://localhost:3000"): Allows cross-origin requests from the specified origin (http://localhost:3000), enabling frontend applications running on that origin to access the backend server.

Autowired Dependencies

MovieService movieService: An instance of the MovieService class, which provides business logic to handle movie-related operations.

Usage

The DataTheaterAPIS class acts as the entry point for movie-related API requests in the backend server. It exposes endpoints to retrieve movies, add new movies, and retrieve movies based on their status.

To use this API, frontend applications can make HTTP requests to the corresponding endpoints, and the DataTheaterAPIS class will process those requests by calling the appropriate methods in the MovieService. The service will then interact with the database or perform necessary business logic to handle the requested movie operations.

Package: com.backend.server.controller

Description

The MovieApis class is a Spring RestController that handles HTTP requests related to movies in the backend server. It exposes various endpoints to perform CRUD operations on movie data through the MovieService.

Endpoints

GET /movies/getAllMovies

Description:

Retrieves a list of all movies available in the system.

HTTP Method: GET

Parameters: None

Produces: JSON (MediaType.APPLICATION\_JSON\_VALUE)

Returns:

(Type: List<Movies>) A list of all movies available in the system.

GET /movies/getByStatus

Description:

Retrieves a list of movies based on the provided status.

HTTP Method: GET

Parameters:

status (Type: String) - The status to filter the movies.

Returns:

(Type: List<Movies>) A list of movies that match the given status.

POST /movies/addAllMovies

Description:

Saves multiple movies in the system.

HTTP Method: POST

Parameters:

movies (Type: List<Movies>) - A list of Movies objects to be added to the system.

Returns: None

POST /movies/addMovie

Description:

Saves a single movie in the system.

HTTP Method: POST

Parameters:

movie (Type: Movies) - The Movie object to be added to the system.

Returns:

(Type: Movies) The saved Movie object with its generated ID.

POST /movies/updateMovie

Description:

Updates a movie in the system.

HTTP Method: POST

Parameters:

movie (Type: Movies) - The Movie object to be updated in the system.

Returns:

(Type: Movies) The updated Movie object.

Annotations

@RestController: Indicates that this class is a Spring RestController, allowing it to handle HTTP requests and automatically convert the response to JSON format.

@RequestMapping("/movies"): Maps all the endpoints in this controller to the base URL "/movies".

@CrossOrigin(origins = "http://localhost:3000"): Allows cross-origin requests from the specified origin (http://localhost:3000), enabling frontend applications running on that origin to access the backend server.

Autowired Dependencies

MovieService movieService: An instance of the MovieService class, which provides business logic to handle movie-related operations.

Usage

The MovieApis class acts as the entry point for movie-related API requests in the backend server. It exposes endpoints to retrieve movies, add new movies, update existing movies, and retrieve movies based on their status.

To use this API, frontend applications can make HTTP requests to the corresponding endpoints, and the MovieApis class will process those requests by calling the appropriate methods in the MovieService. The service will then interact with the database or perform necessary business logic to handle the requested movie operations.

Package: com.backend.server.controller

Description

The TheaterApis class is a Spring RestController that handles HTTP requests related to movies in the context of theaters in the backend server. It exposes various endpoints to perform CRUD operations on movie data through the MovieService.

Endpoints

GET /movies/getAllMovies

Description:

Retrieves a list of all movies available in the system.

HTTP Method: GET

Parameters: None

Produces: JSON (MediaType.APPLICATION\_JSON\_VALUE)

Returns:

(Type: List<Movies>) A list of all movies available in the system.

GET /movies/getByStatus

Description:

Retrieves a list of movies based on the provided status.

HTTP Method: GET

Parameters:

status (Type: String) - The status to filter the movies.

Returns:

(Type: List<Movies>) A list of movies that match the given status.

POST /movies/addAllMovies

Description:

Saves multiple movies in the system.

HTTP Method: POST

Parameters:

movies (Type: List<Movies>) - A list of Movies objects to be added to the system.

Returns: None

POST /movies/addMovie

Description:

Saves a single movie in the system.

HTTP Method: POST

Parameters:

movie (Type: Movies) - The Movie object to be added to the system.

Returns:

(Type: Movies) The saved Movie object with its generated ID.

Annotations

@RestController: Indicates that this class is a Spring RestController, allowing it to handle HTTP requests and automatically convert the response to JSON format.

@RequestMapping("/movies"): Maps all the endpoints in this controller to the base URL "/movies".

@CrossOrigin(origins = "http://localhost:3000"): Allows cross-origin requests from the specified origin (http://localhost:3000), enabling frontend applications running on that origin to access the backend server.

Autowired Dependencies

MovieService movieService: An instance of the MovieService class, which provides business logic to handle movie-related operations.

Usage

The TheaterApis class acts as the entry point for movie-related API requests in the context of theaters in the backend server. It exposes endpoints to retrieve movies, add new movies, and retrieve movies based on their status.

To use this API, frontend applications can make HTTP requests to the corresponding endpoints, and the TheaterApis class will process those requests by calling the appropriate methods in the MovieService. The service will then interact with the database or perform necessary business logic to handle the requested movie operations.

Please ensure that the MovieService is correctly implemented and injected into the TheaterApis class to handle the movie-related operations in the context of theaters. Additionally, consider renaming the class to make it more semantically meaningful since the current naming (TheaterApis for a movie-related controller) might be confusing to other developers.

Package: com.backend.server.controller

Description

The TicketsApis class is a Spring RestController that handles HTTP requests related to tickets in the backend server. It exposes various endpoints to perform CRUD operations on ticket data through the ticketService.

Endpoints

GET /tickets/getAlltickets

Description:

Retrieves a list of all tickets available in the system for report generation.

HTTP Method: GET

Parameters: None

Produces: JSON (MediaType.APPLICATION\_JSON\_VALUE)

Returns:

(Type: List<tickets>) A list of all tickets available in the system for report generation.

GET /tickets/getByStatus

Description:

Retrieves a list of tickets based on the provided status.

HTTP Method: GET

Parameters:

status (Type: String) - The status to filter the tickets.

Returns:

(Type: List<tickets>) A list of tickets that match the given status.

POST /tickets/addAlltickets

Description:

Saves multiple tickets in the system.

HTTP Method: POST

Parameters:

tickets (Type: List<tickets>) - A list of tickets to be added to the system.

Returns: None

POST /tickets/addticket

Description:

Saves a single ticket in the system.

HTTP Method: POST

Parameters:

ticket (Type: tickets) - The ticket object to be added to the system.

Returns:

(Type: tickets) The saved ticket object with its generated ID.

POST /tickets/cancelTicket

Description:

Cancels a ticket in the system.

HTTP Method: POST

Parameters:

ticket (Type: tickets) - The ticket object to be canceled in the system.

Returns:

(Type: tickets) The ticket object after cancellation.

Annotations

@RestController: Indicates that this class is a Spring RestController, allowing it to handle HTTP requests and automatically convert the response to JSON format.

@RequestMapping("/tickets"): Maps all the endpoints in this controller to the base URL "/tickets".

@CrossOrigin(origins = "http://localhost:3000"): Allows cross-origin requests from the specified origin (http://localhost:3000), enabling frontend applications running on that origin to access the backend server.

Autowired Dependencies

ticketService ticketService: An instance of the ticketService class, which provides business logic to handle ticket-related operations.

Usage

The TicketsApis class acts as the entry point for ticket-related API requests in the backend server. It exposes endpoints to retrieve tickets, add new tickets, cancel tickets, and retrieve tickets based on their status.

To use this API, frontend applications can make HTTP requests to the corresponding endpoints, and the TicketsApis class will process those requests by calling the appropriate methods in the ticketService. The service will then interact with the database or perform necessary business logic to handle the requested ticket operations.

Please ensure that the ticketService is correctly implemented and injected into the TicketsApis class to handle the ticket-related operations. Additionally, consider renaming the class to make it more semantically meaningful since the current naming (TicketsApis for a ticket-related controller) might be confusing to other developers.

Package: com.backend.server.Model

Description

The User class represents a user entity in the backend server. It is annotated with Lombok annotations (@Data, @AllArgsConstructor, @NoArgsConstructor, @Builder) to generate boilerplate code for getters, setters, constructors, and builder methods. The class is also annotated with @Document to specify its mapping to the "user" collection in MongoDB.

Fields

email (Type: String)

Description: The email address of the user.

firstName (Type: String)

Description: The first name of the user.

lastName (Type: String)

Description: The last name of the user.

mobileNumber (Type: String)

Description: The mobile number of the user. This field is annotated with @Id to indicate it as the identifier field for MongoDB.

password (Type: String)

Description: The password of the user.

Address (Type: String)

Description: The address of the user.

role (Type: String)

Description: The role of the user. It represents the user's access level or permissions.

Annotations

@Data: A Lombok annotation that generates getter and setter methods for all fields, as well as toString, equals, and hashCode methods.

@AllArgsConstructor: A Lombok annotation that generates a constructor with all arguments.

@NoArgsConstructor: A Lombok annotation that generates a default constructor with no arguments.

@Builder: A Lombok annotation that generates a builder pattern for creating instances of the class.

@Document(collection = "user"): Indicates that instances of this class are stored in the MongoDB collection named "user".

Usage

The User class is used to represent user data in the backend server. It can be used as a data model to store and retrieve user information in the database, such as user registration details, login information, and user profiles.

The @Data annotation generates the necessary getter and setter methods for all fields, making it easy to access and modify user data. The @AllArgsConstructor and @NoArgsConstructor annotations provide constructors to create instances of the User class with or without arguments, respectively. The @Builder annotation allows for the creation of instances using the builder pattern, which can be useful for constructing complex objects with many optional fields.

When saving instances of the User class to the MongoDB database, ensure that the necessary Spring Data MongoDB configurations are in place, including the MongoDB connection settings and proper mapping of the User class to the "user" collection. Also, consider adding additional validation and security mechanisms, such as password hashing and validation rules, to ensure the security and integrity of user data.