# SkyWindGroup\_TechnicalTest API Documentation

## Introduction

Welcome to the comprehensive documentation for the Online Lottery Management API, your definitive guide to integrating and harnessing the full potential of our API for seamless online lottery management. This resource is tailored to serve developers and administrators alike, equipping you with the insights needed to effortlessly incorporate our API into your operations.

### About the API

The Online Lottery Management API empowers you to take control of various dimensions within online lottery operations. Whether you're in the business of creating, organizing, or overseeing lotteries, our API provides a versatile toolkit to facilitate your endeavors.

### Key features

* **Scalability**: In the dynamic landscape of online lotteries, accommodating growing numbers of participants and transactions is paramount. Our API's architecture is finely tuned to ensure smooth operations as your lottery ecosystem expands.
* **Resilience**: The heart of online operations lies in maintaining consistent service. Our API boasts resilient architecture, ensuring uninterrupted functionality and responsiveness even in the face of surging demand.
* **Caching for Performance**: Performance is the cornerstone of user experience. By integrating intelligent caching mechanisms, our API optimizes response times and reduces backend load, elevating the overall quality of engagement.
* **Structured Design**: Navigating complexity is simplified with our API's meticulously designed structure. With intuitive endpoint designations and well-defined parameters, seamless integration and operation become second nature.

### Getting started

This documentation serves as your compass through the intricacies of authentication, available endpoints, request and response formats, error handling, and industry best practices. Whether you're a developer enriching your application with lottery management capabilities or an administrator orchestrating the intricate dance of lottery activities, this documentation provides a compass to navigate your journey.

## Endpoints

### UserController

The *UserController* class defines several endpoints to manage user-related operations. Here's a breakdown of each endpoint along with a description of its functionality:

1. **GET /api/user:**

* **Description:** Retrieves a list of all users.
* **Functionality:** Retrieves and returns a collection of UserDTO objects representing all users stored in the system.
* **Full URL:** <http://localhost:5203/api/user>

1. **GET /api/user/{userId}:**

* **Description:** Retrieves a specific user by their ID(userId)
* **Functionality**: Retrieves the user with the provided userId and returns a UserDTO object if found. If no user matches the ID, returns a "User not found!" message.
* **Full URL:** http://localhost:5203/api/user/{userId}

1. **GET /api/user/winners:**

* **Description:** Retrieves a list of winners for the last draw.
* **Functionality:** Retrieves and returns a collection of UserDTO objects representing the winners of the last draw. If no winners are found, returns a 404 response.
* **Full URL:** <http://localhost:5203/api/user/winners>

1. **GET /api/user/winners/{date}:**

* **Description:** Retrieves a list of winners by a specified date.
* **Functionality**: Retrieves and returns a collection of UserDTO objects representing the winners of the draw that took place on the specified date. The date parameter is used to filter the winners. If no winners are found for the given date, returns an empty list.
* **Full URL**: [http://localhost:5203/api/user/winners/{date}](http://localhost:5203/api/user/winners/%7bdate%7d)

1. **POST /api/user:**

* **Description:** Adds a new user.
* **Functionality:** Validates the incoming UserDTO\_POST object using the UserValidator.ValidateUserModel method. If validation succeeds and the user does not already exist, adds the user to the repository. If the user already exists, returns a "A user with the identification number/email/phonenumber already exists!" message. If validation fails, returns a "User properties format incorrect!" message.
* **Full URL:** http://localhost:5203/api/user

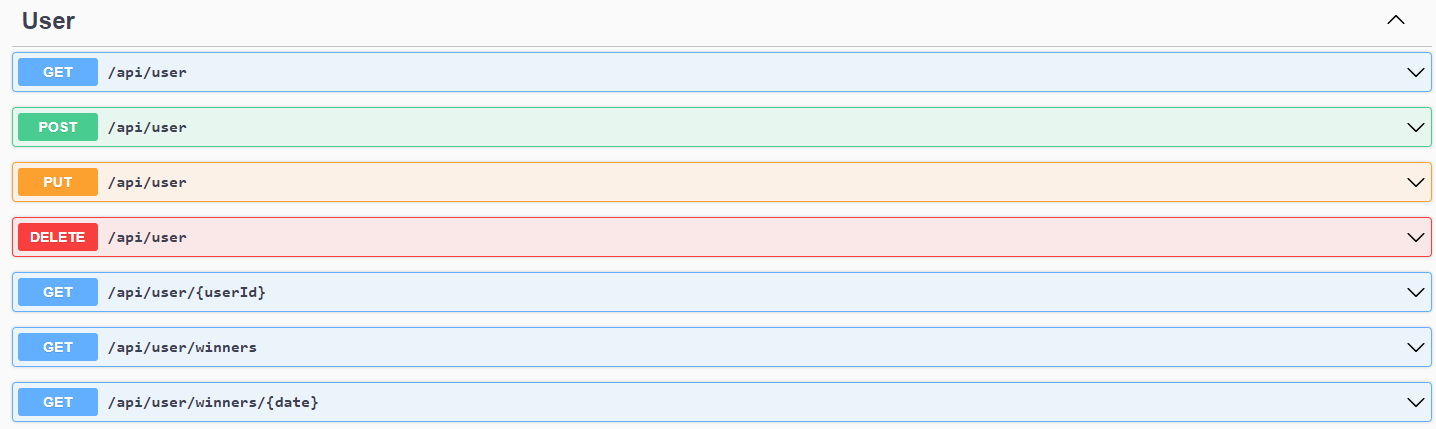
1. **PUT /api/user:**

* **Description:** Updates an existing user.
* **Functionality:** Validates the incoming UserDTO\_PUT object using the UserValidator.ValidateUserModel method. If validation succeeds, updates the user in the repository. If validation fails, returns a "User properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/user>

1. **DELETE /api/user:**

* **Description:** Deletes a user by their ID.
* **Functionality:** Retrieves the user with the provided userId. If the user exists, deletes the user from the repository and returns a "User deleted!" message. If no user matches the ID, returns a "User not found!" message.
* **Full URL:** <http://localhost:5203/api/user>

Each endpoint handles specific user-related tasks, such as retrieving user information, adding new users, updating existing users, and deleting users. The endpoints also incorporate caching mechanisms for improved performance and response times.



### TicketController

1. **GET /api/ticket:**

* **Description:** Retrieves a list of all tickets.
* **Functionality:** Retrieves and returns a collection of TicketDTO objects representing all tickets in the system.
* **Full URL:** <http://localhost:5203/api/ticket>

1. **GET /api/ticket/user/{userId}**

* **Description:** Retrieves tickets for a specific user.
* **Functionality:** Retrieves and returns a collection of TicketDTO objects associated with the provided userId.
* **Full URL:** [http://localhost:5203/api/ticket/user/{userId}](http://localhost:5203/api/ticket/user/%7buserId%7d)

1. **GET /api/ticket/played-tickets:**

* **Description:** Retrieves tickets based on their played status.
* **Functionality:** Retrieves and returns a collection of TicketDTO objects filtered by the specified played status.
* **Full URL:** <http://localhost:5203/api/ticket/played-tickets>

1. **GET /api/ticket/{ticketId}:**

* **Description:** Retrieves a specific ticket by its ID.
* **Functionality:** Retrieves and returns a TicketDTO object representing the ticket with the provided ticketId. If no ticket matches the ID, returns a "Ticket not found!" message.
* **Full URL:** [http://localhost:5203/api/ticket/{ticketId}](http://localhost:5203/api/ticket/%7bticketId%7d)

1. **POST /api/ticket:**

* **Description:** Adds a new ticket.
* **Functionality:** Validates the incoming TicketDTO\_POST object using the TicketValidator.ValidateTicketModel method. If validation succeeds, adds the ticket to the repository. If validation fails, returns a "Ticket format properties incorrect!" message.
* **Full URL:** <http://localhost:5203/api/ticket>

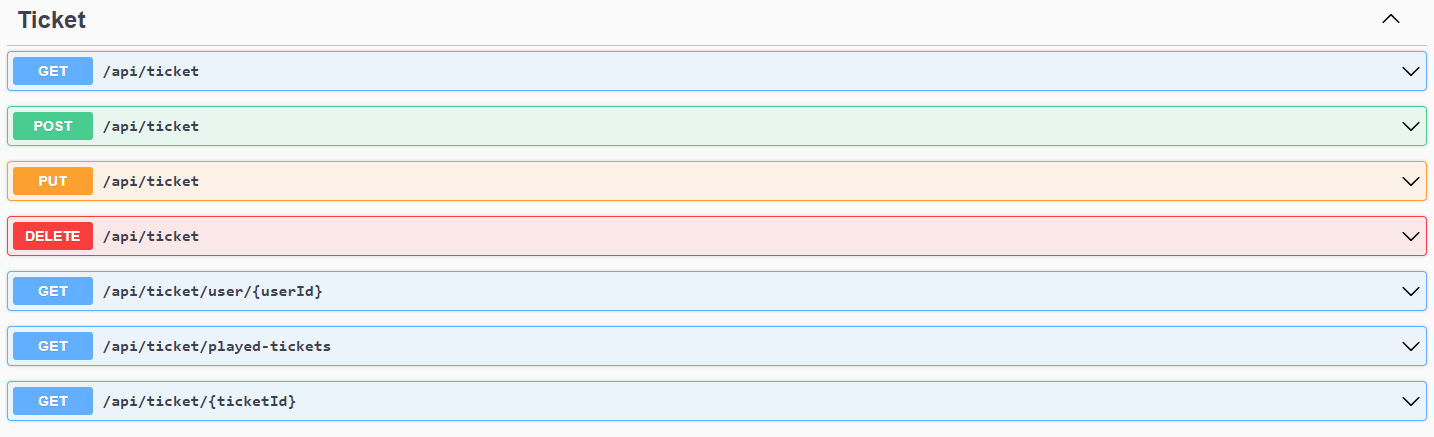
1. **PUT /api/ticket:**

* **Description:** Updates an existing ticket.
* **Functionality:** Validates the incoming TicketDTO\_PUT object using the TicketValidator.ValidateTicketModel method. If validation succeeds, updates the ticket in the repository. If validation fails, returns a "Ticket properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/ticket>

1. **DELETE /api/ticket:**

* **Description:** Deletes a ticket by its ID.
* **Functionality:** Retrieves the ticket with the provided ticketId. If the ticket exists, deletes the ticket from the repository and returns a "Ticket deleted!" message. If no ticket matches the ID, returns a "Ticket not found!" message.
* **Full URL:** <http://localhost:5203/api/ticket>

Each endpoint in the TicketController handles specific ticket-related tasks, such as retrieving tickets, adding new tickets, updating tickets, and deleting tickets. The full URLs incorporate the base URL http://localhost:5203/api/[controller] along with the endpoint-specific route segments.



### PrizeController

1. **GET /api/prize**:

* **Description**: Retrieves a list of all prizes.
* **Functionality:** Retrieves and returns a collection of PrizeDTO objects representing all available prizes.
* **Full URL:** <http://localhost:5203/api/prize>

1. **GET /api/prize/lottery-type/{lotteryTypeId}:**

* **Description:** Retrieves prizes based on the associated lottery type.
* **Functionality:** Retrieves and returns a collection of PrizeDTO objects associated with the provided lotteryTypeId.
* **Full URL:** [http://localhost:5203/api/prize/lottery-type/{lotteryTypeId}](http://localhost:5203/api/prize/lottery-type/%7blotteryTypeId%7d)

1. **GET /api/prize/{prizeId}:**

* **Description:** Retrieves a specific prize by its ID.
* **Functionality:** Retrieves and returns a PrizeDTO object representing the prize with the provided prizeId. If no prize matches the ID, returns a "Prize not found!" message.
* **Full URL:** [http://localhost:5203/api/prize/{prizeId}](http://localhost:5203/api/prize/%7bprizeId%7d)

1. **POST /api/prize:**

* **Description:** Adds a new prize.
* **Functionality:** Validates the incoming PrizeDTO\_POST object using the PrizeValidator.ValidatePrizeModel method. If validation succeeds, adds the prize to the repository. If validation fails, returns a "Prize properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/prize>

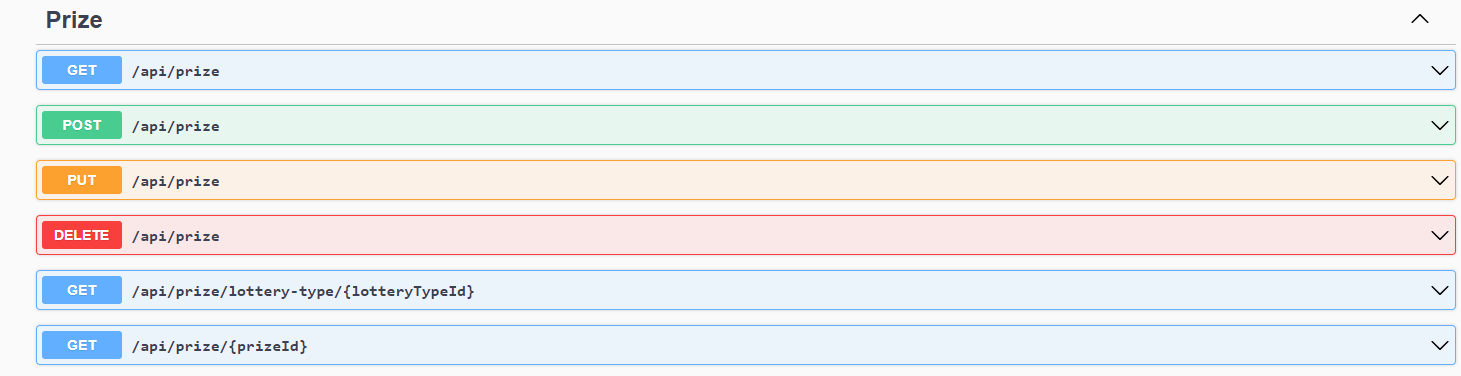
1. **PUT /api/prize:**

* **Description:** Updates an existing prize.
* **Functionality:** Validates the incoming PrizeDTO\_PUT object using the PrizeValidator.ValidatePrizeModel method. If validation succeeds, updates the prize in the repository. If validation fails, returns a "Prize properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/prize>

1. **DELETE /api/prize**

* **Description:** Deletes a prize by its ID.
* **Functionality:** Retrieves the prize with the provided prizeId. If the prize exists, deletes the prize from the repository and returns a "Prize deleted!" message. If no prize matches the ID, returns a "Prize not found!" message.
* **Full URL:** <http://localhost:5203/api/prize>

Each endpoint in the PrizeController handles specific prize-related tasks, such as retrieving prizes, adding new prizes, updating prizes, and deleting prizes. The full URLs incorporate the base URL http://localhost:5203/api/[controller] along with the endpoint-specific route segments.



### LotteryTypeController

1. **GET /api/lotterytype:**

* **Description:** Retrieves a list of all lottery types.
* **Functionality:** Retrieves and returns a collection of LotteryTypeDTO objects representing all available lottery types.
* **Full URL**: <http://localhost:5203/api/lotterytype>

1. **GET /api/lotterytype/{lotteryTypeId}:**

* **Description:** Retrieves a specific lottery type by its ID.
* **Functionality:** Retrieves and returns a LotteryTypeDTO object representing the lottery type with the provided lotteryTypeId. If no lottery type matches the ID, returns a "Lottery type not found!" message.
* **Full URL:** [http://localhost:5203/api/lotterytype/{lotteryTypeId}](http://localhost:5203/api/lotterytype/%7blotteryTypeId%7d)

1. **POST /api/lotterytype:**

* **Description:** Adds a new lottery type.
* **Functionality:** Validates the incoming LotteryTypeDTO\_POST object using the LotteryTypeValidator.ValidateLotteryTypeModel method. If validation succeeds, adds the lottery type to the repository. If validation fails, returns a "Lottery type properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/lotterytype>

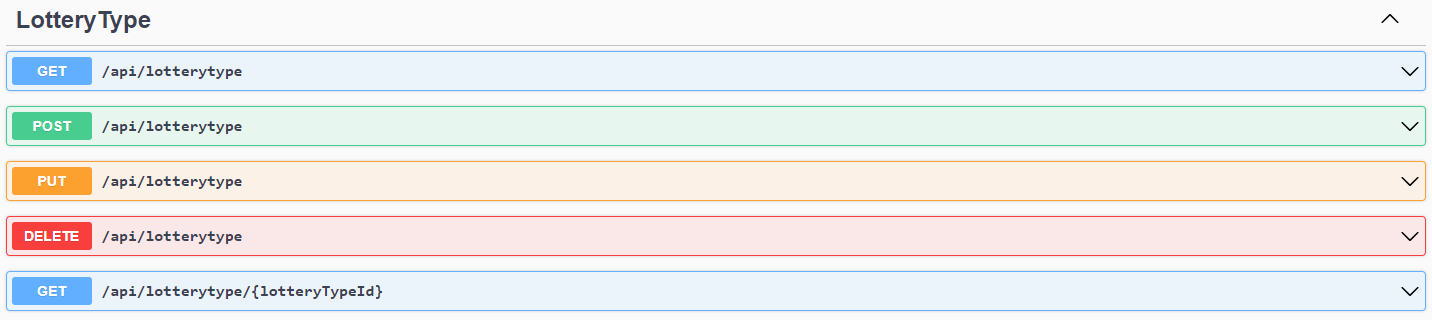
1. **PUT /api/lotterytype:**

* **Description:** Updates an existing lottery type.
* **Functionality:** Validates the incoming LotteryTypeDTO\_PUT object using the LotteryTypeValidator.ValidateLotteryTypeModel method. If validation succeeds, updates the lottery type in the repository. If validation fails, returns a "Lottery type properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/lotterytype1>

1. **DELETE /api/lotterytype:**

* **Description:** Deletes a lottery type by its ID.
* **Functionality:** Retrieves the lottery type with the provided lotteryTypeId. If the lottery type exists, deletes it from the repository and returns a "Lottery type deleted!" message. If no lottery type matches the ID, returns a "Lottery type not found!" message.
* **Full URL:** http://localhost:5203/api/lotterytype

Each endpoint in the LotteryTypeController handles specific lottery type-related tasks, such as retrieving lottery types, adding new lottery types, updating lottery types, and deleting lottery types. The full URLs incorporate the base URL http://localhost:5203/api/[controller] along with the endpoint-specific route segments.



### DrawHistoryController

1. **GET /api/drawhistory:**

* **Description:** Retrieves a list of all draw history entries.
* **Functionality:** Retrieves and returns a collection of DrawHistoryDTO objects representing the history of all draws.
* **Full URL:** <http://localhost:5203/api/drawhistory>

1. **GET /api/drawhistory/lottery-type/{lotteryTypeId}:**

* **Description:** Retrieves draw history entries based on the associated lottery type.
* **Functionality:** Retrieves and returns a collection of DrawHistoryDTO objects associated with the provided lotteryTypeId.
* **Full URL:** [http://localhost:5203/api/drawhistory/lottery-type/{lotteryTypeId}](http://localhost:5203/api/drawhistory/lottery-type/%7blotteryTypeId%7d)

1. **GET /api/drawhistory/date-interval:**

* **Description:** Retrieves draw history entries based on a date interval.
* **Functionality:** Validates the incoming date interval using the DrawHistoryValidator.ValidateDateInterval method. If validation succeeds, retrieves and returns a collection of DrawHistoryDTO objects from within the specified date range. If validation fails, returns a "Date interval format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/drawhistory/date-interval>

1. **GET /api/drawhistory/{drawHistoryId}:**

* **Description:** Retrieves a specific draw history entry by its ID.
* **Functionality:** Retrieves and returns a DrawHistoryDTO object representing the draw history entry with the provided drawHistoryId. If no draw history entry matches the ID, returns a "Draw history not found!" message.
* **Full URL:** [http://localhost:5203/api/drawhistory/{drawHistoryId}](http://localhost:5203/api/drawhistory/%7bdrawHistoryId%7d)

1. **POST /api/drawhistory:**

* **Description:** Adds a new draw history entry.
* **Functionality:** Validates the incoming DrawHistoryDTO\_POST object using the DrawHistoryValidator.ValidateDrawHistoryModel method. If validation succeeds, adds the draw history entry to the repository. If validation fails, returns a "Draw history properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/drawhistory>

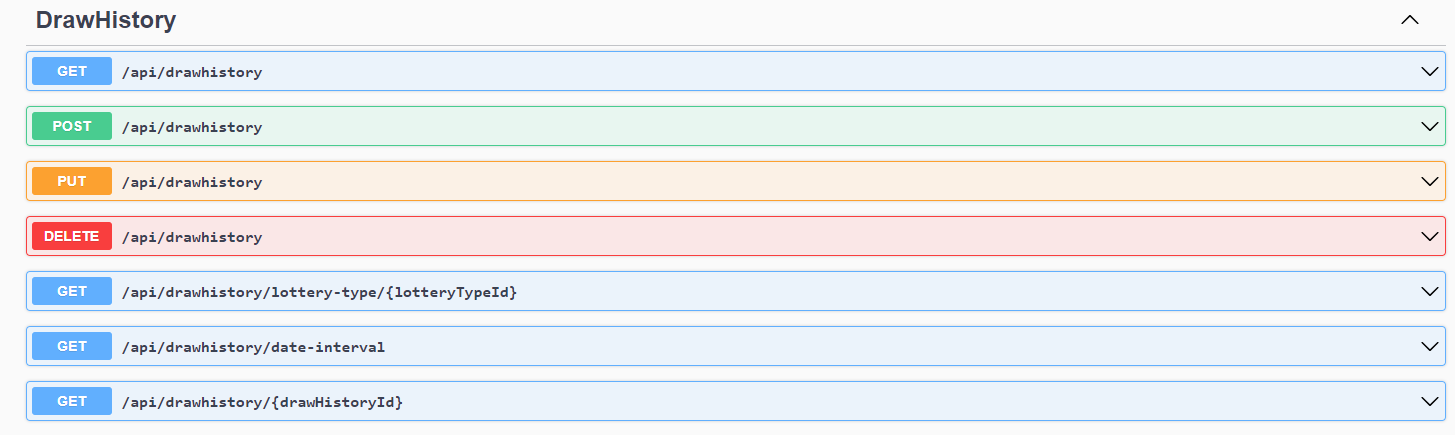
1. **PUT /api/drawhistory:**

* **Description:** Updates an existing draw history entry.
* **Functionality:** Validates the incoming DrawHistoryDTO\_PUT object using the DrawHistoryValidator.ValidateDrawHistoryModel method. If validation succeeds, updates the draw history entry in the repository. If validation fails, returns a "Draw history properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/drawhistory>

1. **DELETE /api/drawhistory:**

* **Description:** Deletes a draw history entry by its ID.
* **Functionality:** Retrieves the draw history entry with the provided drawHistoryId. If the draw history entry exists, deletes it from the repository and returns a "Draw history deleted!" message. If no draw history entry matches the ID, returns a "Draw history not found!" message.
* **Full URL:** <http://localhost:5203/api/drawhistory>

Each endpoint in the DrawHistoryController handles specific draw history-related tasks, such as retrieving draw history entries, adding new draw history entries, updating draw history entries, and deleting draw history entries. The full URLs incorporate the base URL http://localhost:5203/api/[controller] along with the endpoint-specific route segments.



### CurrencyController

1. **GET /api/currency:**

* **Description:** Retrieves a list of all currencies.
* **Functionality:** Retrieves and returns a collection of CurrencyDTO objects representing all available currencies.
* **Full URL:** <http://localhost:5203/api/currency>

1. **GET /api/currency/{currencyId}:**

* **Description:** Retrieves a specific currency by its ID.
* **Functionality:** Retrieves and returns a CurrencyDTO object representing the currency with the provided currencyId. If no currency matches the ID, returns a "Currency not found!" message.
* **Full URL:** [http://localhost:5203/api/currency/{currencyId}](http://localhost:5203/api/currency/%7bcurrencyId%7d)

1. **POST /api/currency:**

* **Description:** Adds a new currency.
* **Functionality:** Validates the incoming CurrencyDTO\_POST object using the CurrencyValidator.ValidateCurrencyModel method. If validation succeeds, adds the currency to the repository. If validation fails, returns a "Currency properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/currency>

1. **PUT /api/currency:**

* **Description:** Updates an existing currency.
* **Functionality:** Validates the incoming CurrencyDTO\_PUT object using the CurrencyValidator.ValidateCurrencyModel method. If validation succeeds, updates the currency in the repository. If validation fails, returns a "Currency properties format incorrect!" message.
* **Full URL:** <http://localhost:5203/api/currency>

1. **DELETE /api/currency:**

* **Description:** Deletes a currency by its ID.
* **Functionality:** Retrieves the currency with the provided currencyId. If the currency exists, deletes it from the repository and returns a "Currency deleted!" message. If no currency matches the ID, returns a "Currency not found!" message.
* **Full URL:** http://localhost:5203/api/currency

Each endpoint in the CurrencyController handles specific currency-related tasks, such as retrieving currencies, adding new currencies, updating currencies, and deleting currencies. The full URLs incorporate the base URL http://localhost:5203/api/[controller] along with the endpoint-specific route segments.

