Book Shop Report General

- **Student**: Alberto Campagnolo
- **Student code**: 897569
- **Group**: Diego Caspi, Marco Mihai Condrache, Alberto Campagnolo
- **Email**: 897569@stud.unive.it
- **Application Deadline**: 31/02/2025
- **Project Link**: https://github.com/marcocondrache/tw-project
- **Primary Skill Achieved**:
  - Good Understanding of Angular development environment
  - How to use a component library
  - Interact with a back-end using Angular services and component

1. Introduction

- **Project Description**:
  - This project🔗 implements a modern academic book auction platform using a sophisticated technology stack and following industry best practices. The system is built as a monorepo using Nx for efficient workspace management and better code organization.
- **Key Responsibilities**:
  - Support the creation of the front-end project structure
  - Develop front-end component with logic to correctly share data with the back-end
  - Correct use of the Taiga UI🔗 Angular component library concurrently with Tailwindcss

---

2. Project Architecture

The project has been built using **Nx** as a **monorepo** organization **tool**, providing several key advantages for development and maintenance:

- **Shared Code**: Common utilities and components can be easily reused across applications
- **Consistent Tooling**: Development experience remains uniform across the entire project
- **Atomic Changes**: Updates affecting multiple packages can be tested and deployed together
- **Simplified Dependencies**: Better management of internal dependencies between packages

2.1 Monorepo Structure

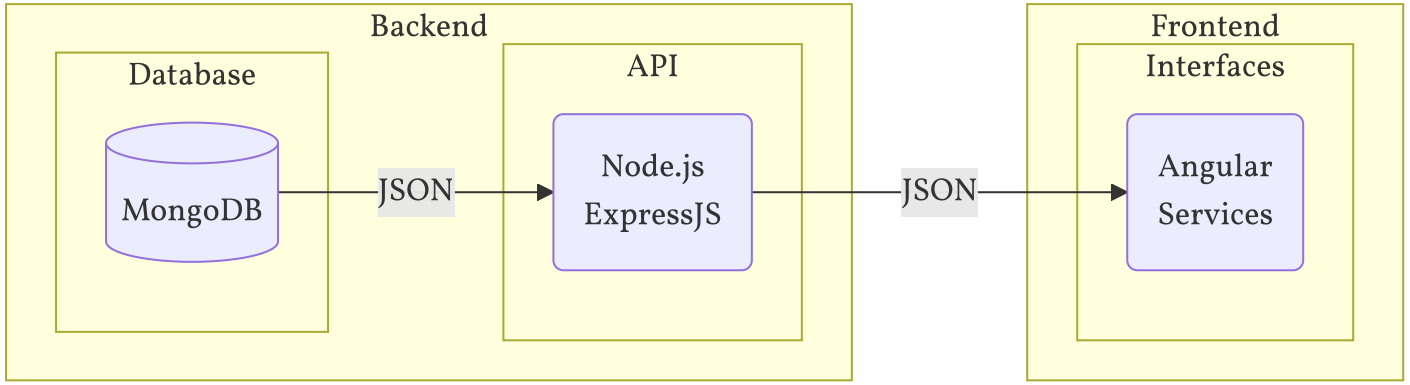
The project follow this pattern:

```
📁 apps/
├──📁 frontend/
├──📁 backend/
📁 libs/
├──📁 api/
├──📁 core/
├──📁 domain/
```

Key Feature:

- `📁 frontend/` : Angular 18 client application
- `📁 backend/` : Express JS server application
- `📁 api/:` Shared API interfaces and related types
- `📁 core/` : Common utilities and patterns
- `📁domain/` : Business logic and models

Application WorkFlow:



2.2 Libraries

The project follows a modular architecture using a library-based approach, all organized under the `libs/` directory, divided based on their purposes inside the project. All the modules are reachable inside the code using the `@shared/{folder name}` import.

Core Library

The Core library serves as the foundation for shared utilities and patterns across the entire application. It provides essential functionality that supports the entire system's operation like:

- 📁 `json-patch/`:
  - **JSON Patch Operations** : Standardized way to describe changes to JSON documents following the `RFC 6902`

```
[
    { "op": "replace", "path": "/book/title", "value": "New Title" },
    { "op": "remove", "path": "/book/oldField" }
]
```

- 📁`lhs/`:
  - **Left-hand Side Operations**: module use to enable flexibility inside the API filtering structures using a standardized query parameter syntax

```
// Example filters:
book.price[gt]=50
// Books with price greater than 50
book.university[eq]=Ca Foscari // Books from Ca' Foscari
auction.endDate[lt]=2024-04-01 // Auctions ending before April 1st
```

- 📁 `errors/`:
  - **Error Handling**: comprehensive approach to error management across the application. The following code represents the structure adopted

```
{
    "message": "string",
    "status": 401,
    "traceId": "5cbd8075-48e6-4801-bced-f8fd16f7b357"
}
```

- 📁 `pagination/`:
  - **Pagination**: pagination mechanism used the application the create an ideal approach for common use cases. The following code represents a p.o.c

```
{
    "list": [
    ...
    ],
    "metadata": {
        "totalItems": 100,
        "page": 4,
        "totalPages": 10
    }
}
```

Is good to mention even the `sort` module that provides a simple way to sort the results from a paginated request using a parameter called *sort*. This parameter is a string that contains the field to sort, prefixed by the direction of the sort represented by a `+` or `-`.

Domain Library

The Domain library serves as the cornerstone of our application's data architecture, encapsulating all core business **models** and **schemas** while ensuring type safety throughout the entire system.

The library is structured around two fundamental components with the goal of maintain data integrity across the application stack. The first one, housed in the `api/` directory, leverages the `zod` library to define our API schemas. These schemas act as a contract between frontend and backend services, providing **runtime type validation** while simultaneously **generating TypeScript types**.

> ⚠ This approach not only ensures type safety during development but also automatically generates OpenAPI documentation, keeping our API documentation perpetually synchronized with the actual implementation.

The following code provide an example of schema validation:

```
export const ApiAuctionCreationSchema = ApiAuctionSchema.omit({
    seller: true,
    winningBid: true,
})
    .refine(data => data.startingPrice < data.reservePrice, {
        message: "Starting price must be less than reserve price",
        path: ["startingPrice"],
    });
```

Meanwhile the `db/` directory contains our **database schemas** implemented using `typegoose`. This sophisticated **ODM** (Object Document Mapper) brings the
power of TypeScript to MongoDB operations, offering type safe database interactions.
These models handle the basic schema structure and incorporate sophisticated features such as **middleware** hooks for pre and post **database operations**, index definitions, and **relationship mappings** between collections. This architectural decision significantly **reduces** the **likelihood** of **runtime errors** while improving the developer experience.

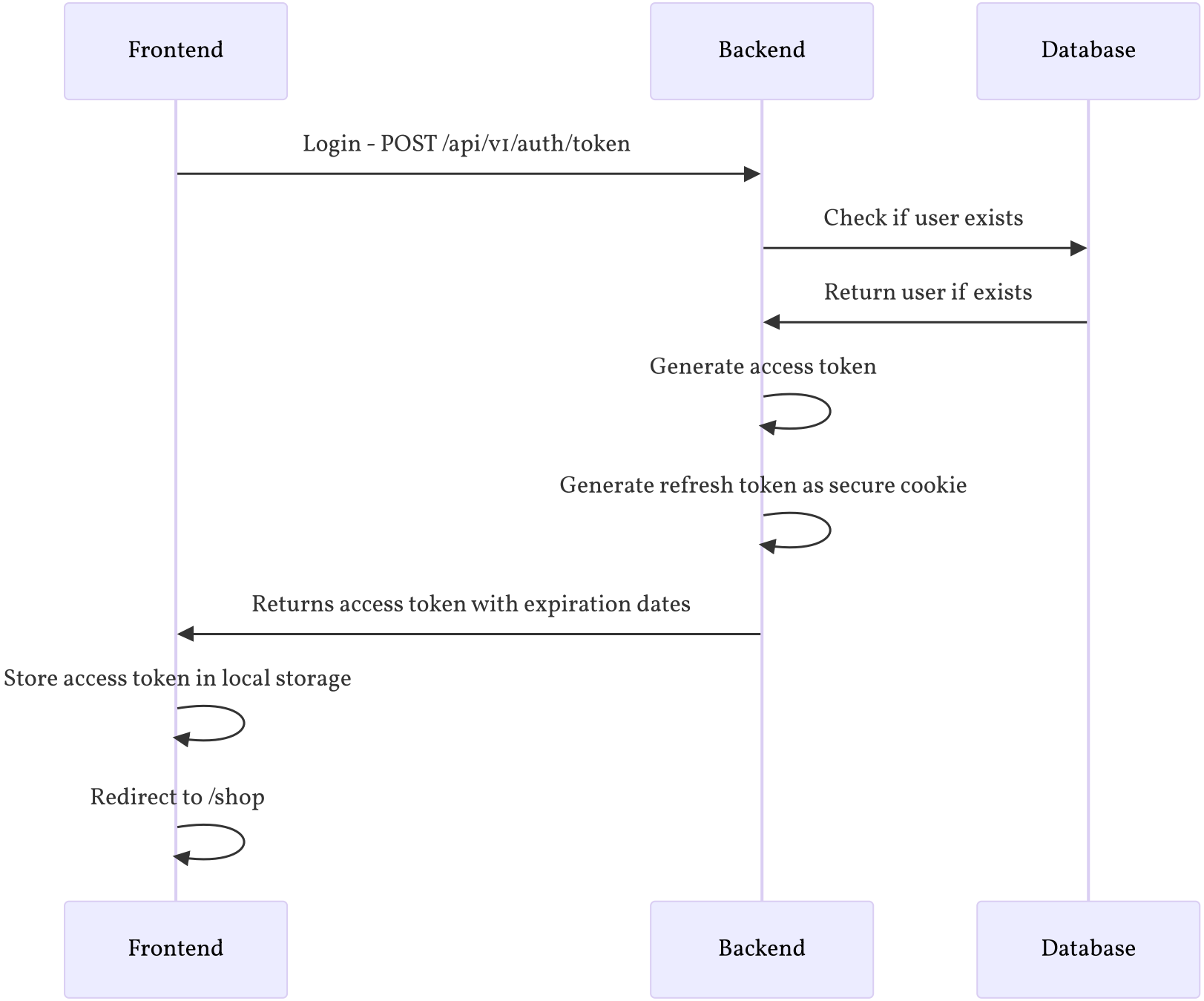
API Library

This library establishes a robust contract between the frontend and backend services, ensuring type safety and consistency across the entire application, providing simultaneously a comprehensive framework for defining and managing API interactions. The corner of this system is the `Endpoint` type, which serves to ensuring that the API endpoints contain necessary information for both runtime and development time type checking.
The following code provide an example of what an `Endpoint` object looks like:

```
export const getListingsEndpoint = endpoint({
    path: "/v1/listings",
    method: "get",
    lhs: ["book.university", "book.course", "auction.startingPrice"],
    paramsSchema: PaginationRequestSchema.innerType(),
    responseSchema: PaginatedResponseSchemaOf(ApiListingSchema),
    config: {
        authentication: false,
        tags: ["Listing"],
```

```
        summary: "Get all listings",
        description: "Get all listings with pagination",
    },
});
```

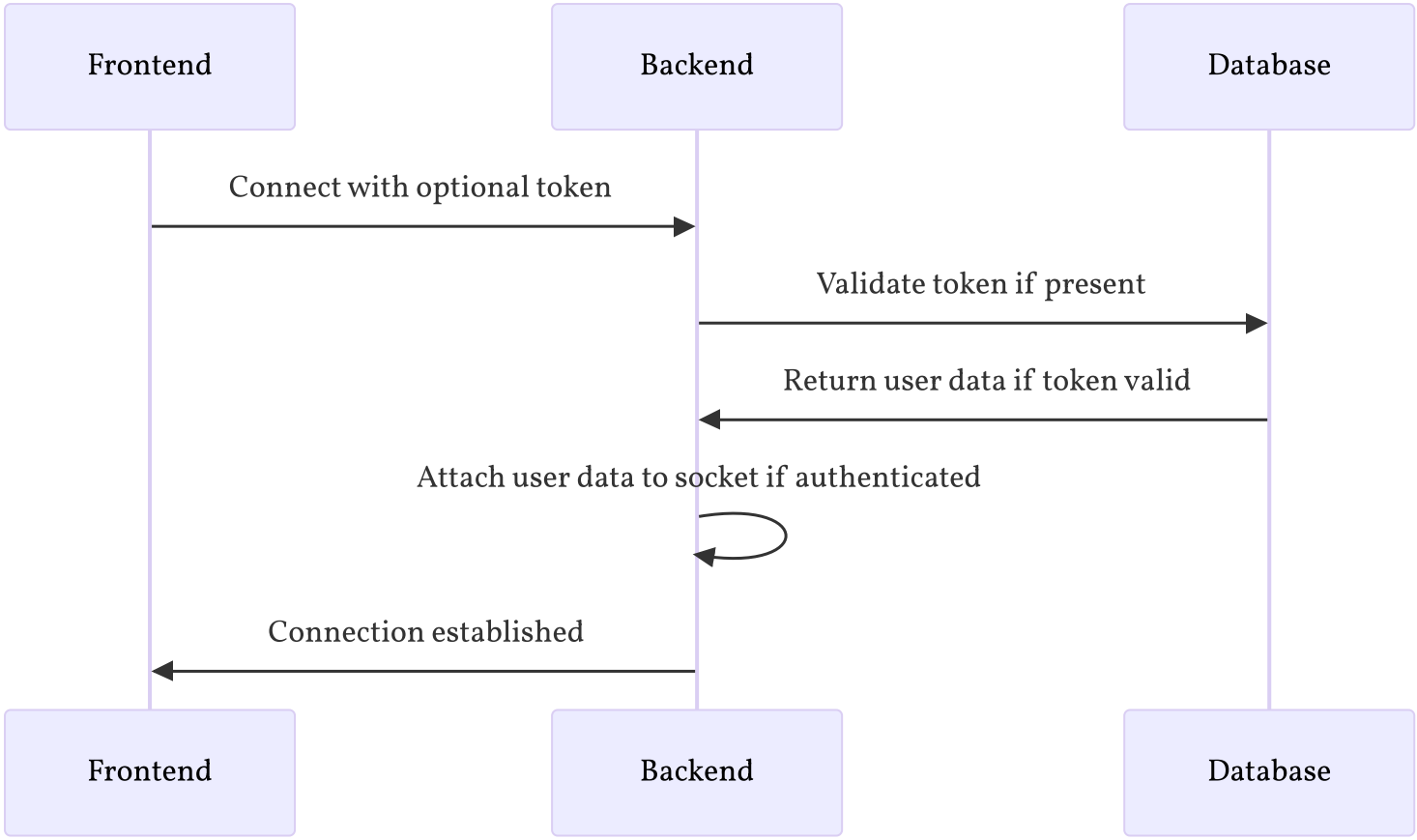
3. Deep Dive Into Backend Workflow


Frontend
Backend
Database
Login - POST /api/v1/auth/token
Check if user exists
Return user if exists
Generate access token
Generate refresh token as secure cookie
Returns access token with expiration dates
Store access token in local storage
Redirect to /shop
Frontend
Backend
Database

All generated **tokens** are **JSON Web Tokens**. They are signed with a **secret key** stored inside the environment variables.

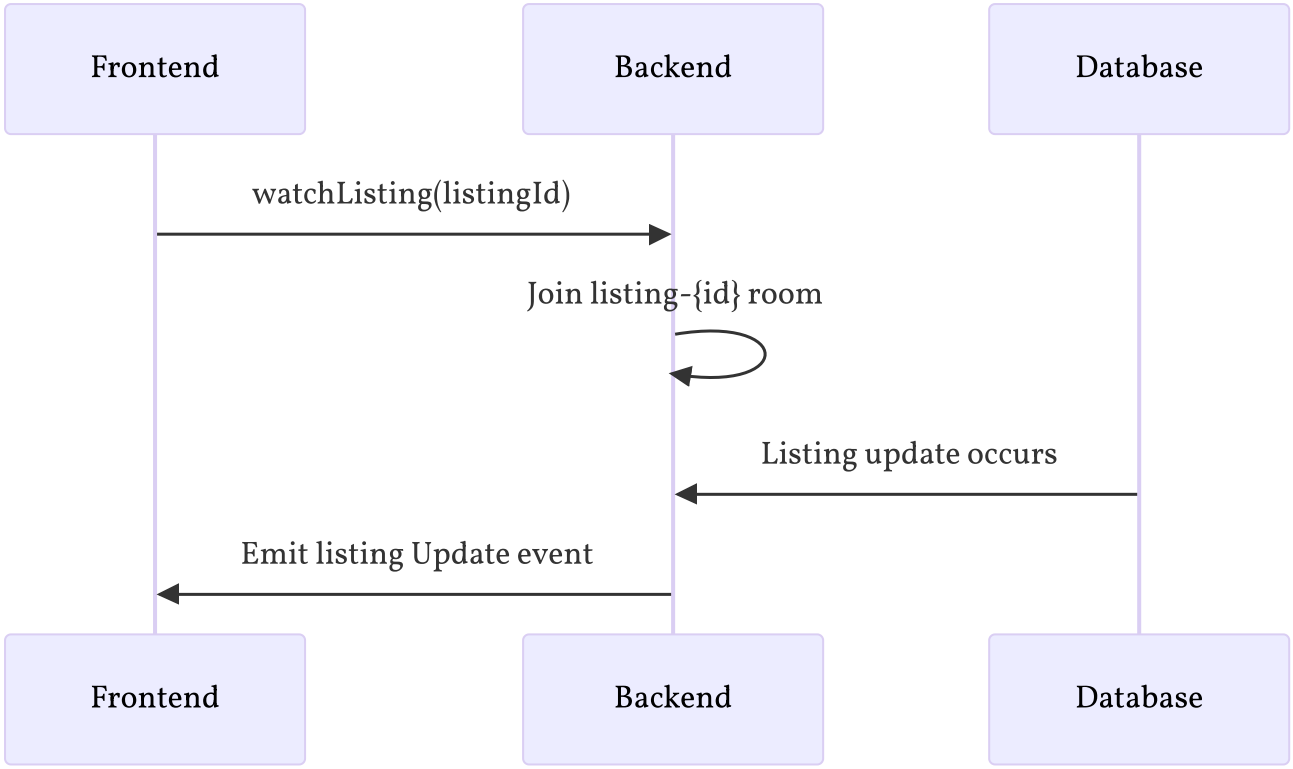
Each token contains the following information inside its payload:

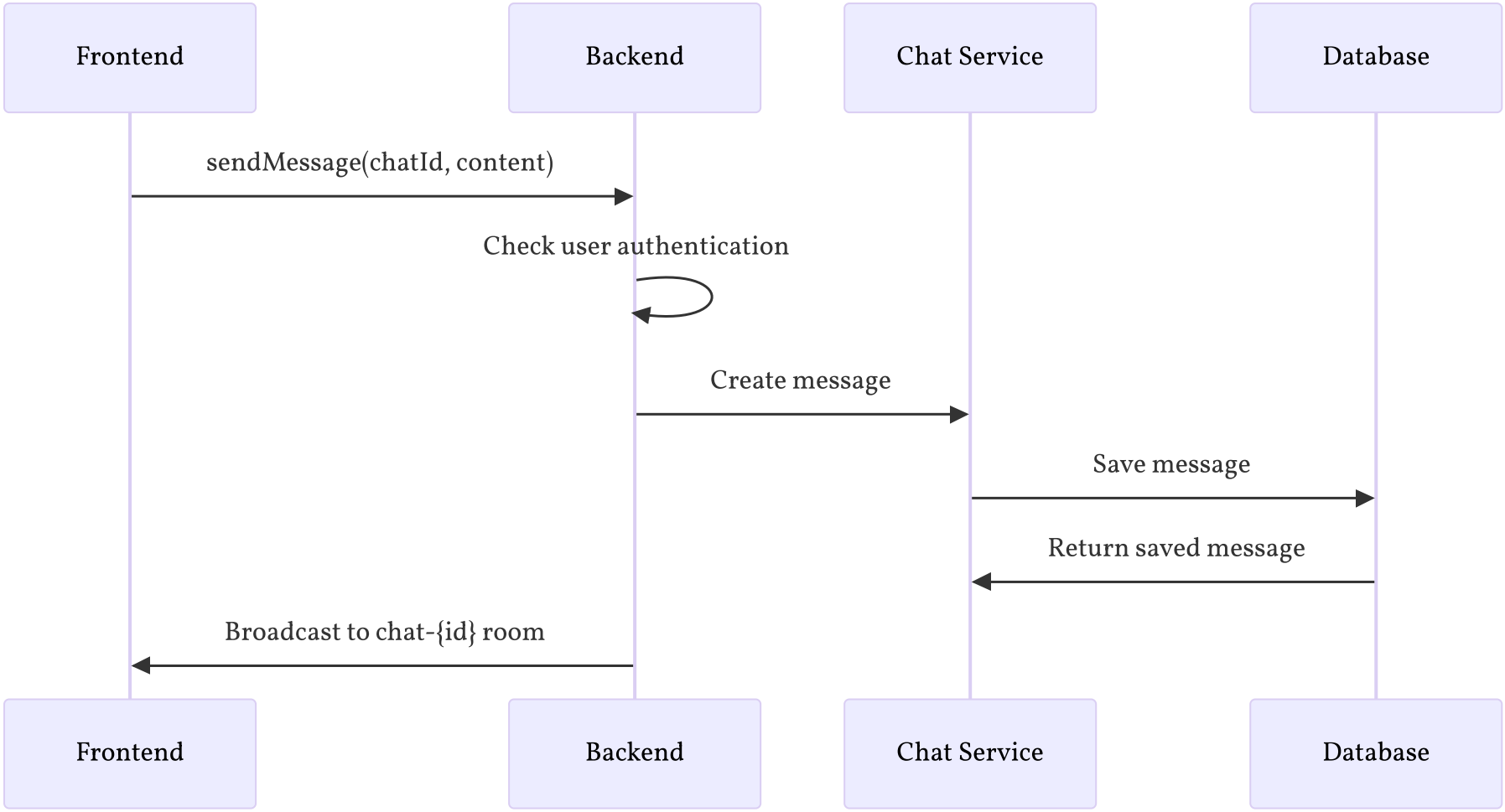
- `sub` : Public Id user's profile
- `username` : User's username
- `email` : User's username
- `type` : Set to *access* for access token, *refresh* for refresh token
- `scope` : Either *user* or *admin* indicating the user's role
- `exp` : Expiration timestamp in milliseconds



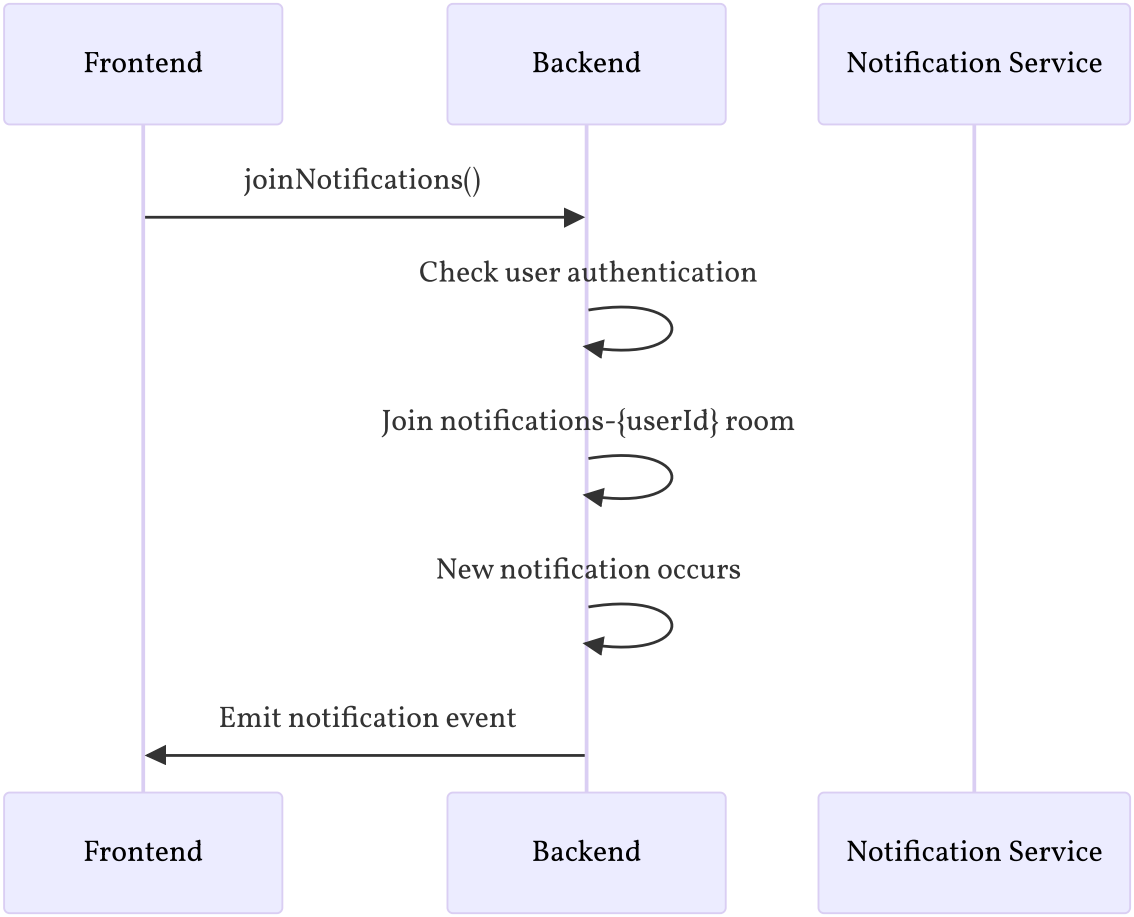
To implement the real-time experience between users, providing them with a real-time chat, we've implemented websocket connections through the `socket.io` library.
Above the diagram provide a good view of how the user is authenticated to use the websocket.

The following sequence diagrams provide the flow to update listings, messaging through the chats, update the notifications.


Frontend
Backend
Database
watchListing(listingId)
Join listing-{id} room
Listing update occurs
Emit listing Update event
Frontend
Backend
Database


Frontend
Backend
Chat Service
Database
sendMessage(chatId, content)
Check user authentication
Create message
Save message
Return saved message
Broadcast to chat-{id} room
Frontend
Backend
Chat Service
Database

Notification flow



4. Deep Dive Into Frontend Application

The frontend application is built using **Angular 18** and follows a **feature first** architecture pattern, where the application is organized into distinct feature modules that encapsulate related functionality. The application is structured into two main sections:

- The public shop interface
- The administrative dashboard
  Each one with its own layout and routing configuration.

Component State Management

The **UI** is built using **Taiga UI v3** components, we choose v3 instead of v4 for better components stability and performance, enhanced with **Tailwind CSS** for styling, with the scope of create a better and more modern look.

The **application state management** is provided by the `rxjs` library, making easy manage both state and data flow. In fact, thanks to the library, all the state are managed as `Observable / Subscription` converted to Angular `Signals` to provide an **easy to use interface**.

Routing And Server communication

Inside the frontend root directory, all the paths of the application are managed by the `app.routes.ts` file, **which contains all the routes** organized into sections, each one with is **personal layout** and **routing configuration**.

```
{
    path: "admin",
    component: AdminLayoutComponent,
    canActivate: [AdminGuard],
    title: "Admin",
    children: [
        {
            path: "",
            redirectTo: "dashboard",
            pathMatch: "full"
        },
        { path: "dashboard", component: DashboardComponent, title: "Dashboard - Admin" },
        { path: "users", component: UsersComponent, title: "Users - Admin" },
        { path: "listings", component: ListingsComponent, title: "Listings - Admin" },
        { path: "create", component: AdminCreationComponent, title: "Create admin - Admin"
},

    ],

},
```

In order to provide an easy and adapted interface for the server communication, we've implemented the **ApiService** class, which is a **wrapper around the HttpClient** service.

This service takes care of all the necessary operations defined in the endpoints declared in the `@shared/api` library, providing an interface for the frontend components.

From a developer perspective, when integrating a new endpoint, it's just necessary to call the ApiService method with the endpoint we want to use and the body/params we want to send
to the server, the ApiService will take care of the rest.

The following code show an example of API service

```
manageUserBan(publicIds: string[], action: "ban" | "unban") {
    const body = { publicIds, action };
    return this.apiService.request(
        userBanEndpoint,
        { body }
    ).subscribe();
}
```

UI customization

In Taiga UI v3, color and text patterns cannot be modified using standard CSS directly, as they are predefined within the framework. This limitation led us to override the default variables to give the project more personality and make it more visually appealing. To achieve this, we defined custom values in a file called `styles.css`, allowing us to adapt the UI to better match our design vision.

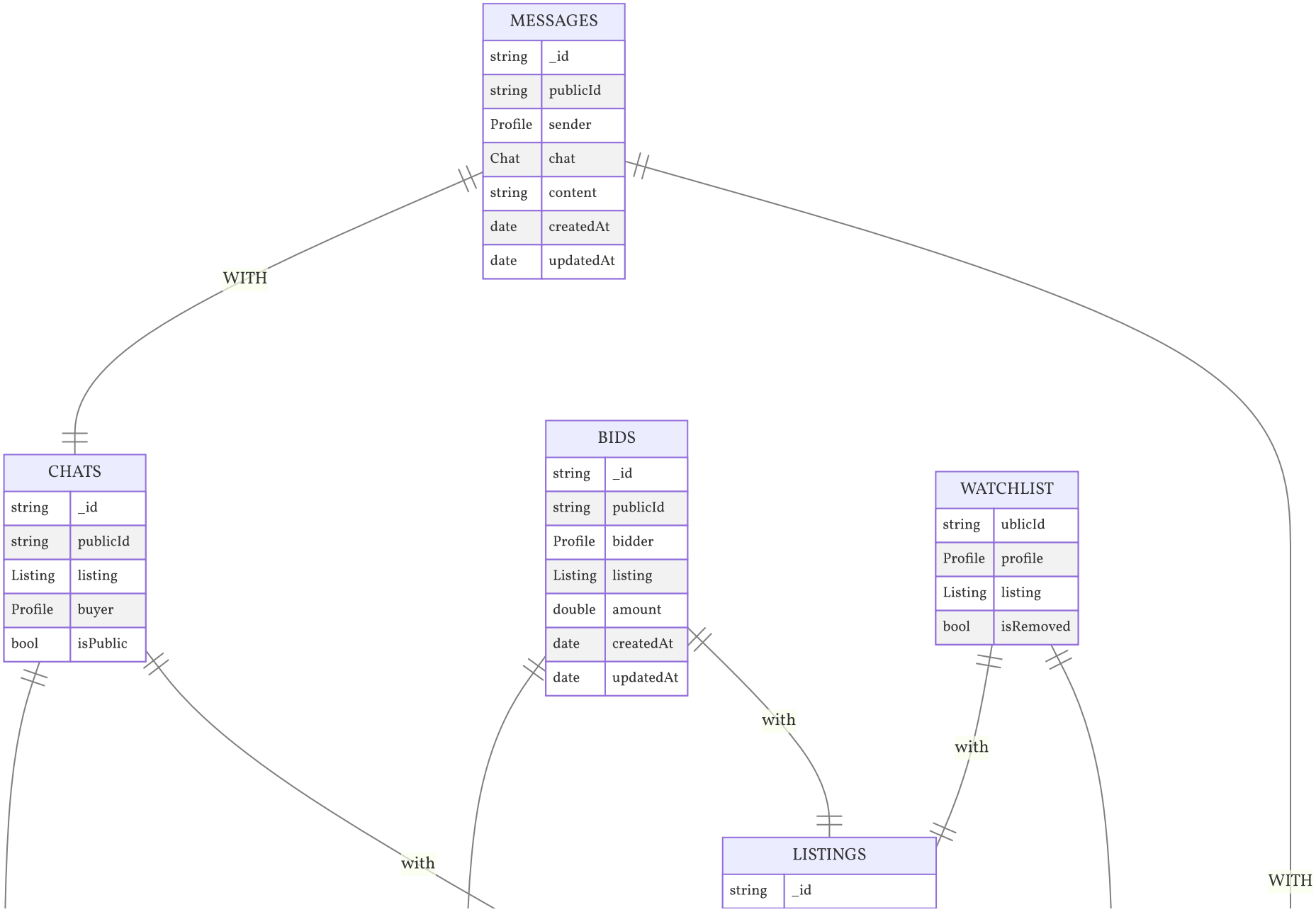
The following code demonstrates this concept in practice. By making these adjustments in a structured and maintainable way, we ensure that future modifications can be implemented efficiently and without disrupting the overall design consistency.

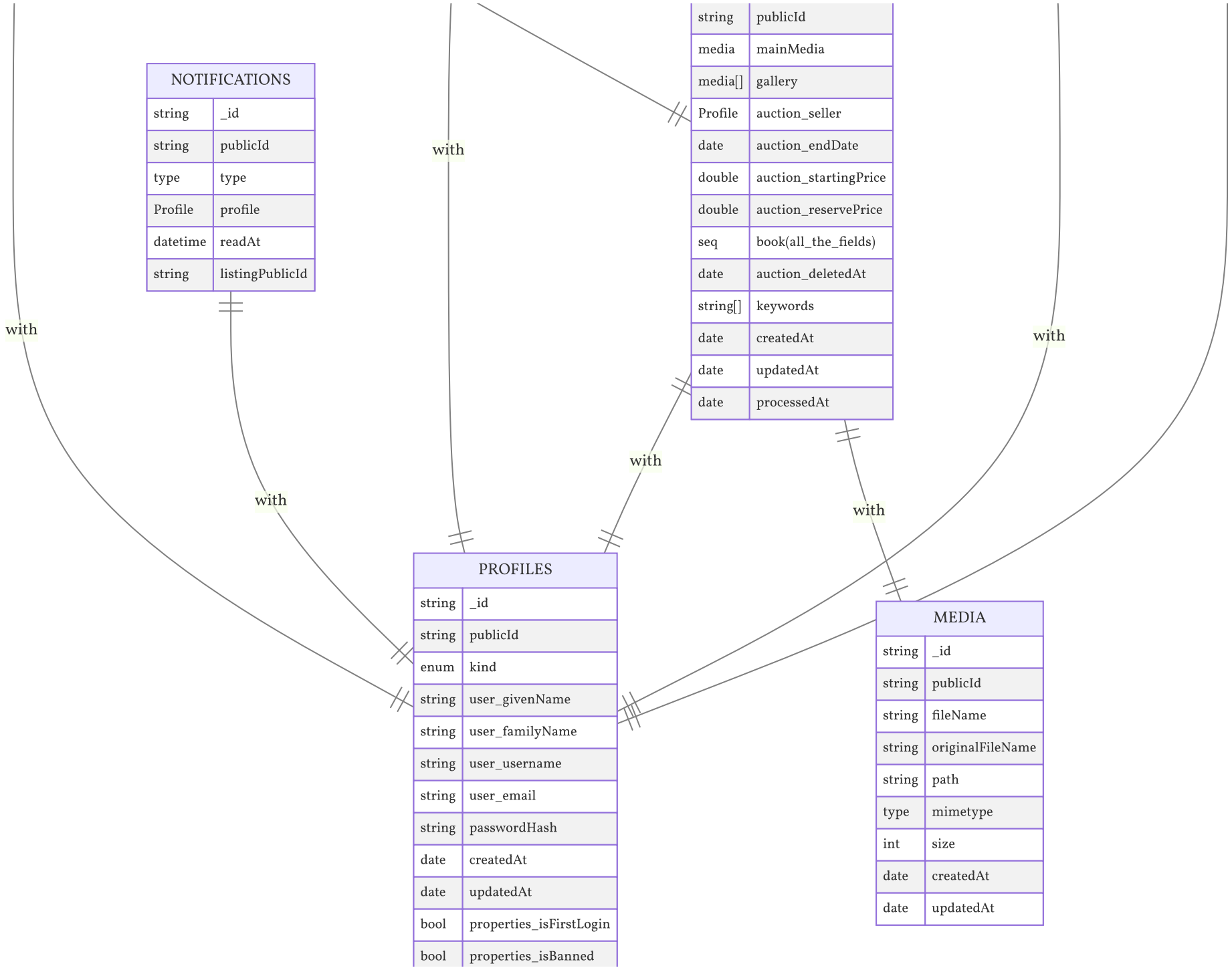
```
--tui-font-text: 'Geist', sans-serif !important;
--tui-font-heading: 'Geist', sans-serif !important;
/* Primary brand color (warm gold) */
--tui-primary: #E6BF48 !important;
--tui-primary-hover: #D4AF37 !important;
--tui-primary-active: #C4A136 !important;

/* Base colors - warm neutral background */
--tui-base-01: #FFFFFF !important;
--tui-base-02: #F8F7F4 !important;
--tui-base-03: #F0EDE6 !important;
--tui-base-04: #E5E1D8 !important;
--tui-base-05: #C7C2B7 !important;
--tui-base-06: #A19E96 !important;
--tui-base-07: #7C7972 !important;
--tui-base-08: #5C5952 !important;
--tui-base-09: #3E3C37 !important;
```

Database Schema

Data Schema


MESSAGES
string _id
string publicId
Profile sender
Chat chat
string content
date createdAt
date updatedAt
WITH
CHATS
string _id
string publicId
Listing listing
Profile buyer
bool isPublic
BIDS
string _id
string publicId
Profile bidder
Listing listing
double amount
date createdAt
date updatedAt
WATCHLIST
string ublicId
Profile profile
Listing listing
bool isRemoved
with
with
with
LISTINGS
string _id
WITH


NOTIFICATIONS
string _id
string publicId
type type
Profile profile
datetime readAt
string listingPublicId
string publicId
media mainMedia
media[] gallery
Profile auction_seller
date auction_endDate
double auction_startingPrice
double auction_reservePrice
seq book(all_the_fields)
date auction_deletedAt
string[] keywords
date createdAt
date updatedAt
date processedAt
PROFILES
string _id
string publicId
enum kind
string user_givenName
string user_familyName
string user_username
string user_email
string passwordHash
date createdAt
date updatedAt
bool properties_isFirstLogin
bool properties_isBanned
MEDIA
string _id
string publicId
string fileName
string originalFileName
string path
type mimetype
int size
date createdAt
date updatedAt
with
with
with
with
with
with

5. Screenshot of the application

Books shop

Search for books, courses, or universities

Sign in

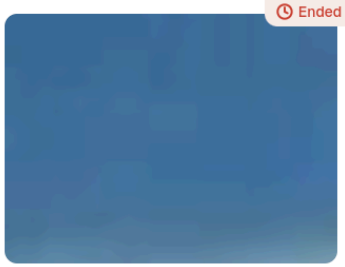
Find Your Next Academic Book

Buy and sell university textbooks with other students

Start Selling

Recently Added

Ended



The Time Machine

Don DeLillo

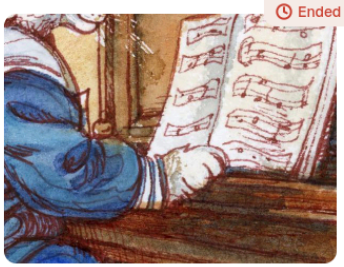
Kilback LLC

Physics

Starting at 65,85 €

Ends Jan 31, 2025

Ended



The Naked and the Dead

James Ellroy

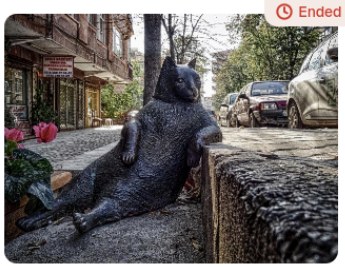
Botsford - Klocko

Dance

Starting at 69,59 €

Ends Jan 19, 2025

Ended



The Kite Runner

Ford Madox Ford

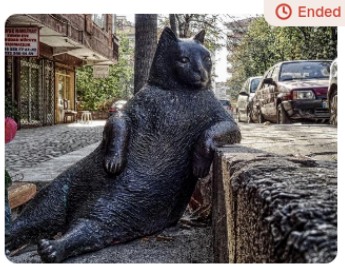
Kreiger, Zboncak and Durgan

Literature

Starting at 58,59 €

Ends Jan 19, 2025

Ended



The Prime of Miss Jean Brodie

Philip Pullman

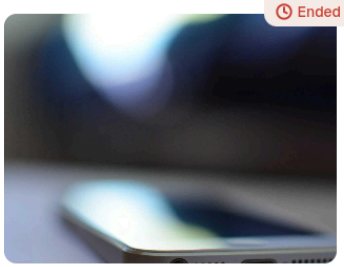
Keeling, Grimes and Sipes

Psychology

Starting at 18,90 €

Ends Jan 29, 2025

Ended



The Call of the Wild

Marcel Proust

Bergstrom, O'Reilly and Lang

Literature

Starting at 24,65 €

Ends Jan 20, 2025

Books Shop

+ Create admin

A Admin User

Welcome back, Admin User

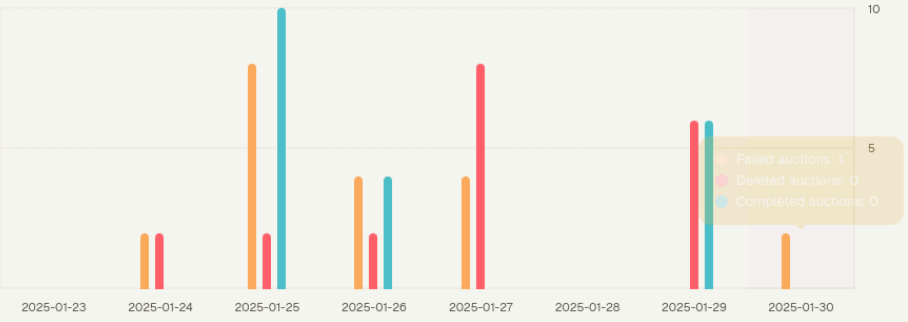
Here's an overview of your platform's performance

Active Auctions
10

Active Users
10

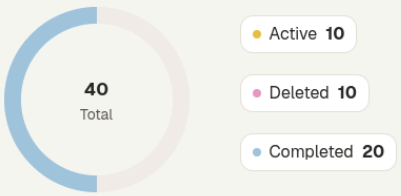
Daily Auctions Overview

Track auction activity over the last 7 days



Active Auctions Overview

Distribution of current auction statuses



Books Shop

+ Create admin

A Admin User

Users Management

View and manage student accounts

Refresh

Search by username or email

11 users

User	Email	Status	Actions
K k4mp47	campagnoloalberto5@gmail.com	Active	Ban
K Kasey.Gleichner	Kasey.Gleichner@hotmail.com	Active	Ban
D Davon.Ebert85	Davon.Ebert@gmail.com	Active	Ban
J Jordon_Koepp	Jordon.Koepp46@hotmail.com	Active	Ban
C Carmelo_Lowe	Carmelo_Lowe76@yahoo.com	Active	Ban
M Mireille.Wilkinson80	Mireille.Wilkinson90@yahoo.com	Active	Ban
E Ernestine.Klein1	Ernestine.Klein@gmail.com	Active	Ban
B Brice_Cremin	Brice_Cremin1@yahoo.com	Active	Ban
J Jewel_Fritsch52	Jewel_Fritsch@yahoo.com	Active	Ban
U Ubaldo_Carroll	Ubaldo_Carroll@hotmail.com	Active	Ban

Books Shop

+ Create admin

A Admin User

Listings Management

View and manage book listings

Refresh

Search listings

30 listings

Book Details	Course Info	Auction Details	Actions
The Grapes of Wrath Mary Shelley ISBN: 978-1-126-10166-6	New Hampshire University Sociology	€59.25 Expired Ended Jan 25, 2025, 4:51:16 AM	Edit Delete
Portnoy's Complaint J.K. Rowling ISBN: 978-1-7098-1283-5	Texas University Physics	€53.75 Expired Ended Jan 26, 2025, 1:38:17 AM	Edit Delete
The Art of War Margaret Atwood ISBN: 978-0-7340-2534-0	Nebraska University Literature	€79.85 Ends Feb 2, 2025, 12:21:26 PM	Edit Delete
It Vladimir Nabokov ISBN: 978-0-16-493934-7	Virginia University Computer Science	€38.09 Expired Ended Jan 29, 2025, 9:45:30 PM	Edit Delete
Nostromo Samuel Richardson ISBN: 978-1-033-28202-1	Illinois University Engineering	€35.45 Expired Ended Jan 25, 2025, 12:42:08 PM	Edit Delete
One Flew Over the Cuckoo's Nest George Orwell ISBN: 978-1-285-25370-1	New Hampshire University Nursing	€43.15 Ends Feb 5, 2025, 11:06:00 AM	Edit Delete
The Catcher in the Rye			

Books shop

Search for books, courses, or universities

Filters

A

Courses: Choose...

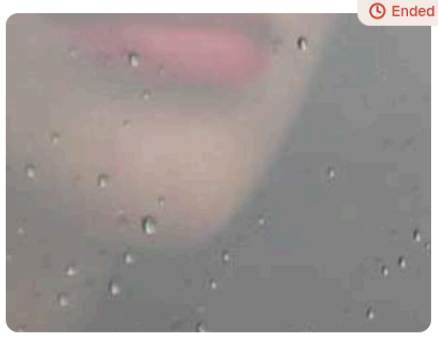
Geographical Area: Choose...

University: Choose...

Starting Price: Starting Price 14,60 79,85

Sort by: Choose...

Ended



The Grapes of Wrath

Mary Shelley

New Hampshire University

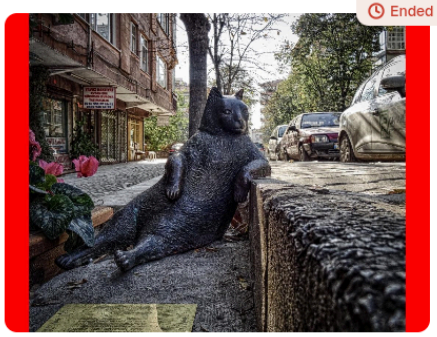
Sociology

Starting at 59,25 €

Ends Jan 25, 2025

View Details

Ended



Portnoy's Complaint

J.K. Rowling

Texas University

Physics

Starting at 53,75 €

Ends Jan 26, 2025

View Details

The Art of War

Margaret Atwood

Nebraska University

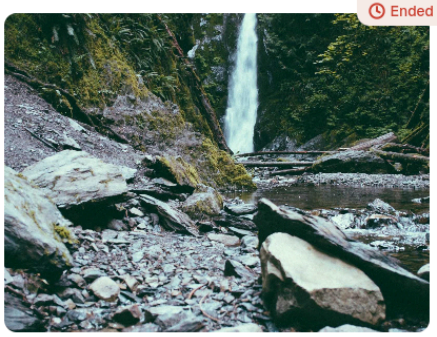
Literature

Starting at 79,85 €

Ends Feb 2, 2025

View Details

Ended



It

Vladimir Nabokov

Virginia University

Computer Science

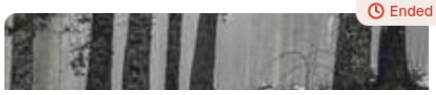
Starting at 38,09 €

Ends Jan 29, 2025

View Details

Ended

Ended



Ended

Books shop

Search for books, courses, or universities

A

Account Settings

My Listings

Account Settings

Manage your personal information

Edit Profile

Personal Information

Given Name
Alberto

Family Name
Campagnolo

Account Information

Username
k4mp47

Email Address
campagnoloalberto5@gmail.com

Books shop

Search for books, courses, or universities

A

Create a new listing

Fill in the details below to create your book listing

Book Details

Book title

Author name

Course name

University

ISBN code

Geographic area

Book condition
Good

Choose the condition that best describes your book's state

Edition

Specify the book's edition and year if available

Language
Italian

Select the language in which the book is written

Number of pages

Enter the total number of pages in the book

Book Condition Guide

Like New: Appears unread, no visible wear
Very Good: Minimal wear, no markings in text
Good: Some wear but remains intact
Fair: Worn but readable, may have markings
Poor: Significant wear, all pages present

Book Images

6. API Documentation

Bookshop API

Overview

A comprehensive RESTful API for an academic book auction platform.
 This API enables students to buy and sell textbooks through an auction-based system, with features including real-time bidding, secure authentication, messaging between users, and moderation capabilities.

 Key Features:
 - Student and moderator authentication
 - Book auction management
 - Real-time bidding system
 - Public and private messaging
 - Advanced search and filtering
 - Auction monitoring and statistics

 For detailed authentication requirements and rate limits, please refer to the individual endpoint documentation.

Version

1.0.0

POST /v1/users

Create a new user.

Create a new user with the given information.

Request Body:

Content: application/json | UserCreation

All of:

```
User
{
    givenName: string;  // The given name of the user.
    familyName: string;  // The family name of the user.
    username: string;  // The username of the user.
    email: string;  // The email of the user.
}
```

and

```
{
    password: string;  // The password of the user.
}
```

Response 201:

Success.

Content: application/json | {

```
    user: Profile;
    token: AuthToken;
}
```

Response 400:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/users

List users.

List all users, must be an admin
Available filters: user.givenName, user.familyName, user.username, user.email.

Request Parameters:

```
page: integer;
pageSize: integer;
user.givenName?: LhsApiQueryOption;
user.familyName?: LhsApiQueryOption;
user.username?: LhsApiQueryOption;
user.email?: LhsApiQueryOption;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<Profile>;  // The list of items.
```

```
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/users/me

Get current user information.

Get the information of the current user.

Response 200:

Success.

Content: application/json | Profile

```
{
    publicId: PublicId;
    kind: string;  // The kind of the profile.
    user: User;
    properties: ProfileProperties;
    createdAt: string;  // The creation date.
    updatedAt: string;  // The last update date.
}
```

Response 400:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

PATCH /v1/users/me

Patch current user information.

Patch the information of the current user
Available patch fields: user/givenName, user/familyName, user/username, user/email, user/password.

Request Body:

Content: application/json | JsonPatchList

```
{
    list: Array<JsonPatch>;
}
```

Response 204:

Success.

Response 400:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

Content: application/json | {

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

Content: application/json | {

```
    message: string;  // The error message.
```

```
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/users/status

Ban users.

Ban a list of users, must be an admin.

Request Body:

```
Content: application/json | {
    action: string;
    publicIds: Array<PublicId>;
}
```

Response 204:

Success.

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

Content: application/json | {
```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/auth/token

Create a new token.

Create a new token for a user.

Request Body:

Content: application/json | LoginType

```
{
    username: string;  // The username of the user.
    password: string;  // The password of the user.
    scope: string;  // The scope of the login.
}
```

Response 201:

Success.

Content: application/json | AuthToken

All of:

BaseToken
```
{
    token: string;  // The token.
    expiresAt: number;  // The expiration date of the token.
}
```

and

```
{
    type: string;  // The type of the token.
}
```

Response 400:

Error.

Content: application/json | {
```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

Content: application/json | {
```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

DELETE /v1/auth/token

Delete a token.

Delete a token for a user.

Response 204:

Success.

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/auth/token/refresh

Refresh a token.

Refresh a token for a user.

Response 201:

Success.

Content: application/json | AuthToken

All of:

```
BaseToken
{
    token: string;  // The token.
    expiresAt: number;  // The expiration date of the token.
}
```

and

```
{
    type: string;  // The type of the token.
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/listings

Create a listing.

Create a listing with main and gallery images.

Request Body:

```
Content: multipart/form-data | {
    body: ListingCreation;
    main: string;  // The file.
    gallery: Array<string>;  // The files.
}
```

Response 201:

Success.

```
Content: application/json | Listing

{
    publicId: PublicId;
    mainMedia: Media;
    gallery: Array<undefined>;  // The gallery of the listing.
    book: Book;
```

```
    auction: Auction;
    createdAt: string;  // The creation date.
    updatedAt: string;  // The last update date.
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings

Get all listings.

Get all listings with pagination
Available filters: book.university, book.course, auction.startingPrice, publicId.

Request Parameters:

```
page: integer;
pageSize: integer;
search?: string;
sort?: undefined;
book.university?: LhsApiQueryOption;
book.course?: LhsApiQueryOption;
auction.startingPrice?: LhsApiQueryOption;
publicId?: LhsApiQueryOption;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<Listing>;  // The list of items.
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings/filter-metadata

Get listings filter metadata.

Get listings filter metadata.

Response 200:

Success.

Content: application/json | ListingFilterMetadata

```
{
    universities: Array<string>;  // The universities of the listings.
    courses: Array<string>;  // The courses of the listings.
    geographicalAreas: Array<string>;  // The geographical areas of the listings.
    startingPrice: object;  // The starting price range of the listings.
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
```

```
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings/keywords

Get listing keywords.

Get paginated listing keywords by search query.

Request Parameters:

```
search: string;
```

Response 200:

Success.

```
Content: application/json | Array<string>
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
```

```
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings/history/won

Get listing history of a user won.

Get listing history of a user won, must be authenticated as a user.

Request Parameters:

```
page: integer;
pageSize: integer;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<Listing>;  // The list of items.
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
```

```
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings/history/participated

Get listing history of a user participated in.

Get listing history of a user participated in, must be authenticated as a user.

Request Parameters:

```
page: integer;
pageSize: integer;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<Listing>;  // The list of items.
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings/owned

Get listings owned by a user.

Get listings owned by a user, must be authenticated as a user.

Request Parameters:

```
page: integer;
pageSize: integer;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<Listing>;  // The list of items.
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings/{publicId}

Get a listing.

Get a listing by its public ID.

Request Parameters:

```
publicId: string;
```

Response 200:

Success.

```
Content: application/json | Listing

{
    publicId: PublicId;
    mainMedia: Media;
    gallery: Array<undefined>;  // The gallery of the listing.
    book: Book;
    auction: Auction;
    createdAt: string;  // The creation date.
    updatedAt: string;  // The last update date.
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

DELETE /v1/listings/{publicId}

Delete a listing.

Delete a listing by its public ID.

Request Parameters:

```
publicId: string;
```

Response 204:

Success.

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

PATCH /v1/listings/{publicId}

Patch a listing.

Patch a listing by its public ID
Available patch fields: book/title, book/author, book/geographicalArea, book/isbn, book/university, book/course, auction/startingPrice, auction/endDate.

Request Parameters:

```
publicId: string;
```

Request Body:

Content: application/json | JsonPatchList

```
{
    list: Array<JsonPatch>;
}
```

Response 204:

Success.

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings/{listingId}/chats

Get all chats for a listing.

Get all chats for a listing, the authentication is required for private chats.

Request Parameters:

```
listingId: string;
page: integer;
pageSize: integer;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<Chat>;  // The list of items.
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/listings/{listingId}/chats

Create a chat.

Create a chat.

Request Parameters:

```
listingId: string;
```

Request Body:

```
Content: application/json | MessageCreation
```

```
{
    content: string;  // The content of the message.
}
```

Response 201:

Success.

```
Content: application/json | Chat
```

```
{
    publicId: PublicId;
    isPublic: boolean;  // Whether the chat is public.
    buyer?: object;  // The buyer of the chat.
    lastMessage?: Message;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
```

```
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/chats/{chatId}/messages

Get all messages for a chat.

Get all messages for a chat, the authentication is required for private chats.

Request Parameters:

```
chatId: string;
page: integer;
pageSize: integer;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<undefined>;  // The list of items.
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
```

```
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/watchlist

Retrieve the watchlist for the current user.

Retrieve the watchlist for the current user.

Request Parameters:

```
page: integer;
pageSize: integer;
listingGuids?: undefined;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<Listing>;  // The list of items.
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
```

```
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/watchlist

Add a listing to watchlist.

Add a listing to watchlist.

Request Body:

```
Content: application/json | {
    listings: Array<string>;
}
```

Response 201:

Success.

Content: application/json |

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

DELETE /v1/watchlist/{listingId}

Remove a listing from watchlist.

Remove a listing from watchlist.

Request Parameters:

```
listingId: string;
```

Response 204:

Success.

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/notifications

Get all notifications.

Get all notifications for the current user.

Request Parameters:

```
page: integer;
pageSize: integer;
```

Response 200:

Success.

```
Content: application/json | {
    list: Array<Notification>;  // The list of items.
    metadata: object;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/notifications/{publicId}/read

Mark a notification as read.

Mark a notification as read.

Request Parameters:

```
publicId: string;
```

Response 204:

Success.

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/listings/{listingId}/bids

Create a bid for a listing.

Create a bid for a listing.

Request Parameters:

listingId: string;

Request Body:

Content: application/json | BidCreation

```
{
    amount: number;  // The amount of the bid.
}
```

Response 201:

Success.

Content: application/json

All of:

```
    Bid
    {
        publicId: PublicId;
        bidder: object;  // The bidder of the bid.
        amount: number;  // The amount of the bid.
    }
```

and

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
```

```
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/listings/{listingId}/bids

Get bids for a listing.

Get bids for a listing.

Request Parameters:

```
listingId: string;
```

Response 200:

Success.

```
Content: application/json | Array<undefined>
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

Content: application/json | {
    message: string; // The error message.
    status: number; // The error status.
    traceId: string; // The trace ID.
}

Response 500:

Error.

Content: application/json | {
    message: string; // The error message.
    status: number; // The error status.
    traceId: string; // The trace ID.
}

GET /v1/statistics/auctions/now

Get the current auction statistics.

Get the current auction statistics, user must be an admin.

Response 200:

Success.

Content: application/json | CurrentAuctionStatistics

{
    auctions: object; // The statistics of the auctions.
}

Response 400:

Error.

Content: application/json | {
    message: string; // The error message.
    status: number; // The error status.
    traceId: string; // The trace ID.
}

Response 401:

Error.

Content: application/json | {
    message: string; // The error message.
    status: number; // The error status.
    traceId: string; // The trace ID.
}

Response 403:

Error.

Content: application/json | {
    message: string; // The error message.
    status: number; // The error status.
    traceId: string; // The trace ID.
}

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/statistics/auctions/daily

Get the daily auction statistics.

Get the daily auction statistics, user must be an admin.

Request Parameters:

```
fromDate: string;
toDate: string;
```

Response 200:

Success.

```
Content: application/json | DailyAuctionStatistics
{
    daily: Array<object>;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

GET /v1/statistics/active

Get the active statistics.

Get the active statistics, user must be an admin.

Response 200:

Success.

```
Content: application/json | {
    activeAuctions: number;  // The number of active auctions.
    activeUsers: number;  // The number of active users.
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

POST /v1/admins

Create a new admin user.

Create a new admin user with the given information, must be authenticated as an admin.

Request Body:

Content: application/json | UserCreation

All of:

```
User
{
    givenName: string;  // The given name of the user.
    familyName: string;  // The family name of the user.
    username: string;  // The username of the user.
    email: string;  // The email of the user.
}
```

and

```
{
    password: string;  // The password of the user.
}
```

Response 201:

Success.

```
Content: application/json | {
    user: Profile;
}
```

Response 400:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 401:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 403:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 404:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Response 500:

Error.

```
Content: application/json | {
    message: string;  // The error message.
    status: number;  // The error status.
    traceId: string;  // The trace ID.
}
```

Schemas

PublicId

string

User

```
{
    givenName: string;  // The given name of the user.
    familyName: string;  // The family name of the user.
    username: string;  // The username of the user.
    email: string;  // The email of the user.
}
```

ProfileProperties

```
{
    isFirstLogin: boolean;  // Whether the profile is the first login.
    isBanned: boolean;  // Whether the profile is banned.
}
```

Profile

```
{
    publicId: PublicId;
    kind: string;  // The kind of the profile.
    user: User;
    properties: ProfileProperties;
    createdAt: string;  // The creation date.
    updatedAt: string;  // The last update date.
}
```

BaseToken

```
{
    token: string;  // The token.
    expiresAt: number;  // The expiration date of the token.
}
```

AuthToken

All of:

```
    BaseToken
    {
        token: string;  // The token.
        expiresAt: number;  // The expiration date of the token.
    }
```

and

```
{
    type: string;  // The type of the token.
}
```

UserCreation

All of:

```
User
{
    givenName: string;  // The given name of the user.
    familyName: string;  // The family name of the user.
    username: string;  // The username of the user.
    email: string;  // The email of the user.
}
```

and

```
{
    password: string;  // The password of the user.
}
```

JsonPatchOp

string
Values: add, remove, replace

JsonPatchPath

string

JsonPatchValue

Any of:

```
string
```

or

```
number
```

or

```
boolean
```

JsonPatch

```
{
    op: JsonPatchOp;
    path: JsonPatchPath;
    value: JsonPatchValue;
}
```

JsonPatchList

```
{
    list: Array<JsonPatch>;
}
```

LhsApiQueryOption

```
{
    eq: string;  // The field must be equal to the value.
    lt: string;  // The field must be less than the value.
    le: string;  // The field must be less than or equal to the value.
    gt: string;  // The field must be greater than the value.
    ge: string;  // The field must be greater than or equal to the value.
    ne: string;  // The field must not be equal to the value.
    in: string;  // The field must be in the list of values.
    nin: string;  // The field must not be in the list of values.
    m: string;  // The field must match the regex.
    mi: string;  // The field must match the regex, ignoring case.
}
```

LoginType

```
{
    username: string;  // The username of the user.
    password: string;  // The password of the user.
    scope: string;  // The scope of the login.
}
```

Media

```
{
    publicId: PublicId;
    fileName: string;  // The file name of the media.
    originalFileName: string;  // The original file name of the media.
    path: string;  // The path of the media.
    mimetype: string;  // The mimetype of the media.
    size: number;  // The size of the media.
    createdAt: string;  // The creation date.
    updatedAt: string;  // The last update date.
}
```

Book

```
{
    title: string;  // The title of the book.
    author: string;  // The author of the book.
    course: string;  // The course of the book.
    isbn: string;  // The ISBN of the book.
    university: string;  // The university of the book.
    geographicalArea: string;  // The geographical area of the book.
    condition?: string;  // The condition of the book.
    edition?: string;  // The edition of the book.
    language?: string;  // The language of the book.
    pages?: number;  // The number of pages of the book.
```

```
}
```

Bid

```
{
    publicId: PublicId;
    bidder: object;  // The bidder of the bid.
    amount: number;  // The amount of the bid.
}
```

Auction

```
{
    seller: object;  // The seller of the auction.
    endDate: string;  // The end date of the auction.
    startingPrice: number;  // The starting price of the auction.
    reservePrice: number;  // The reserve price of the auction.
    winningBid?: Bid;
}
```

Listing

```
{
    publicId: PublicId;
    mainMedia: Media;
    gallery: Array<undefined>;  // The gallery of the listing.
    book: Book;
    auction: Auction;
    createdAt: string;  // The creation date.
    updatedAt: string;  // The last update date.
}
```

AuctionCreation

```
{
    endDate: string;  // The end date of the auction.
    startingPrice: number;  // The starting price of the auction.
    reservePrice: number;  // The reserve price of the auction.
}
```

ListingCreation

```
{
    book: undefined;
    auction: AuctionCreation;
}
```

ListingFilterMetadata

```
{
```

```
    universities: Array<string>;  // The universities of the listings.
    courses: Array<string>;  // The courses of the listings.
    geographicalAreas: Array<string>;  // The geographical areas of the listings.
    startingPrice: object;  // The starting price range of the listings.
}
```

Message

```
{
    publicId: PublicId;
    sender: object;  // The sender of the message.
    content: string;  // The content of the message.
    chatId: string;  // The chat ID.
    createdAt: string;  // The creation date of the message.
}
```

Chat

```
{
    publicId: PublicId;
    isPublic: boolean;  // Whether the chat is public.
    buyer?: object;  // The buyer of the chat.
    lastMessage?: Message;
}
```

MessageCreation

```
{
    content: string;  // The content of the message.
}
```

Notification

```
{
    publicId: PublicId;
    type: string;  // The type of the notification.
    listingPublicId: undefined;
    readAt?: string;  // The date the notification was read.
    createdAt: string;  // The date the notification was created.
}
```

BidCreation

```
{
    amount: number;  // The amount of the bid.
}
```

CurrentAuctionStatistics

```
{
```

```
    auctions: object;  // The statistics of the auctions.
}
```

DailyAuctionStatistics

```
{
    daily: Array<object>;
}
```

7. Deployment

The project uses `nx` and `docker`
to manage how to **build** and **deployment** of the application.
Each part of the application has its own **Dockerfile**, providing a specialize build configuration, in particular the frontend application is built using the **nginx image**, while the backend application
is built using the **node:lts-alpine image**.
To build the app image through `nx` the following command will be used to build the applications and after that the docker image:

```
nx run-many --target=docker-build --all
```

This command will build the applications using **esbuild** for both the **frontend** and the
backend, then it will build the **docker images** including the built artifacts.
In order **to improve** the **security** of the containers, a **multi stage build** is used.

8. Conclusions

I want to thank my two teammates for this course project, as working with them has given me the opportunity to witness the creation of a finished product, discover new tools for managing repositories across different projects, increase my knowledge about containers, learn new Git functionalities, understand models and concepts useful for maintaining and making code reusable, and expand my horizons toward some of the most widely used frameworks in today's tech industry.