**Решение домашнего задания**

**RESTful API для системы управления библиотекой**

**Стажёр – Останина Ксения**

Оглавление

[Инструкция 2](#_Toc197800750)

[Обзор API 3](#_Toc197800751)

[Глоссарий 3](#_Toc197800752)

[Что из себя представляет система? 3](#_Toc197800753)

[Для кого предназначена система? 3](#_Toc197800754)

[Описание сервиса 4](#_Toc197800755)

[Назначение системы 4](#_Toc197800756)

[Основные функциональные модули 4](#_Toc197800757)

[Безопасность и доступ 4](#_Toc197800758)

[Интеграция и расширения 5](#_Toc197800759)

[Технические особенности 5](#_Toc197800760)

[Начало работы 5](#_Toc197800761)

[Требования аутентификации и авторизации API 6](#_Toc197800762)

[Коды статусов и ошибок 7](#_Toc197800763)

[Ограничения скорости 7](#_Toc197800764)

[Образцы кода 7](#_Toc197800765)

[Авторизация, обновление токена, смена пароля и получение API-ключа 7](#_Toc197800766)

[Обновление описания книги через PATCH-запрос 9](#_Toc197800767)

[Ресурс «Книги» 10](#_Toc197800768)

[1. Описание ресурса 10](#_Toc197800769)

[2. Конечные точки и методы 10](#_Toc197800770)

[3. Параметры 10](#_Toc197800771)

[4. Примеры запросов, ответов, JSON Schema 12](#_Toc197800772)

[Ресурс «Пользователи» 18](#_Toc197800773)

[1. Описание ресурса 18](#_Toc197800774)

[2. Конечные точки и методы 18](#_Toc197800775)

[3. Параметры 18](#_Toc197800776)

[4. Примеры запросов, ответов, JSON Schema 19](#_Toc197800777)

[Ресурс «Бронирование» 21](#_Toc197800778)

[1. Описание ресурса 21](#_Toc197800779)

[2. Конечные точки и методы 22](#_Toc197800780)

[3. Параметры 22](#_Toc197800781)

[4. Примеры запросов, ответов, JSON Schema 23](#_Toc197800782)

[Ресурс «Аутентификация» 26](#_Toc197800783)

[1. Описание ресурса 26](#_Toc197800784)

[2. Конечные точки и методы 27](#_Toc197800785)

[3. Параметры 27](#_Toc197800786)

[4. Примеры запросов, ответов, JSON Schema 28](#_Toc197800787)

[5. Особенности реализации 31](#_Toc197800788)

# Инструкция

Вам необходимо спроектировать и реализовать RESTful API для системы управления библиотекой. Система должна поддерживать основные функции, такие как управление книгами, пользователями и бронированиями.

library management system API (v 1.0)

## Обзор API

API предоставляет интерфейс для управления книгами, пользователями и бронированиями в библиотеке. Поддерживаются стандартные CRUD-операции, аутентификация через HMAC и ролевая модель доступа.

### Глоссарий

|  |  |
| --- | --- |
| **Термин** | **Описание** |
| *API Key* | Уникальный ключ, предоставляемый пользователю для аутентификации при доступе к API. Используется как заголовок X-API-Key и применяется для безопасного взаимодействия с API. |
| *Токен доступа* | Временный цифровой ключ (JWT), выдаваемый после успешной аутентификации. Дает право на выполнение запросов от имени пользователя. |
| *HMAC* | (Hash-based Message Authentication Code) Механизм, обеспечивающий целостность и подлинность данных запроса. В API используется в заголовке X-Signature, где значение — это HMAC-подпись тела запроса, сформированная с использованием секретного ключа. |
| *Пользователь* | Зарегистрированный участник системы, обладающий уникальными учетными данными (логин/пароль) и одной из ролей: Читатель, Библиотекарь, Администратор. |
| *Бронирование* | Запись о резервировании конкретной книги пользователем на определённый период. Может иметь статусы: active, returned, overdue. |
| *Читатель* | Пользователь с базовыми правами. Может создавать бронирования, просматривать свои бронирования и обновлять свои данные. |
| *Владелец*  *(в ресурсе «Бронирование»)* | Владелец бронирования, автор бронирования, Пользователь, создавший конкретное бронирование. Имеет право на доступ и изменение своего бронирования, включая отмену. |
| *Библиотекарь* | Роль с расширенными правами: управление книгами (добавление/редактирование), просмотр бронирований, подтверждение возврата книг. |
| *Администратор* | Пользователь с максимальными привилегиями. Имеет полный доступ ко всем операциям API, включая управление пользователями, ролями, бронированиями и параметрами системы. |
| *Гость* | Нерегистрированный пользователь или пользователь, не прошедший аутентификацию. Может получить доступ только к ограниченным публичным операциям, таким как POST /auth/login или POST /auth/password-reset. |

### **Связь ролей и прав**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Роль | Просмотр книг | Бронирование | Управление книгами | Управление пользователями |
| *Гость* | ✓ | ✗ | ✗ | ✗ |
| *Читатель* | ✓ | ✓ | ✗ | ✗ |
| *Библиотекарь* | ✓ | ✓ | ✓ | ✗ |
| *Администратор* | ✓ | ✓ | ✓ | ✓ |

### Что из себя представляет система?

Система позволяет:

* Управлять каталогом книг (добавление, редактирование, удаление);
* Регистрировать пользователей и управлять их правами;
* Бронировать книги и отслеживать их статус.

### Для кого предназначена система?

* Библиотекари – управление книгами и бронированиями.
* Читатели — для бронирования и поиска книг.
* Администраторы – управление пользователями и настройками системы.
* Интеграторы — для встраивания функциональности библиотеки в сторонние приложения.

## Описание сервиса

### Назначение системы

API предназначен для автоматизации работы библиотеки: управления книгами, пользователями и бронированиями. Он предоставляет стандартизированный интерфейс для взаимодействия с системой через HTTP-запросы, поддерживает ролевую модель доступа и обеспечивает безопасность за счёт HMAC-аутентификации.

### Основные функциональные модули

1. Управление книгами

* Каталог книг: Хранение информации о книгах (название, автор, ISBN, год издания, статус доступности).
* Поиск и фильтрация: Возможность получать книги по критериям (например, доступные, по автору или жанру).
* Администрирование: Добавление, редактирование и удаление книг (доступно для библиотекарей и администраторов).

2. Управление пользователями

* Регистрация и аутентификация: Пользователи могут создавать аккаунты, получать API-ключи и входить в систему.
* Роли и доступ:
  + Читатели – могут просматривать книги и создавать бронирования.
  + Библиотекари – могут управлять книгами и бронированиями.
  + Администраторы – имеют полный доступ, включая управление пользователями.
* Профили: Пользователи могут просматривать и редактировать свои данные (например, email или пароль).

3. Бронирование книг

* Создание брони: Читатели могут резервировать книги, если они доступны.
* Управление бронями: Просмотр, продление или отмена бронирований (доступно для читателей, библиотекарей и администраторов).
* Контроль сроков: Система отслеживает сроки возврата книг и может автоматически менять их статус.

### Безопасность и доступ

* Аутентификация через HMAC: Каждый запрос подписывается с использованием секретного ключа.
* Ролевая модель: Доступ к эндпоинтам ограничен в зависимости от роли пользователя.
* HTTPS: Все данные передаются в зашифрованном виде.
* Ограничение запросов: Защита от DDoS-атак через rate limiting (100 запросов в минуту на пользователя).

### Интеграция и расширения

* Внешние системы: API может интегрироваться с CRM, платежными системами или электронными каталогами.
* Логирование: Все действия фиксируются для аудита и отладки.
* Webhooks: Возможность настройки уведомлений (например, о просроченных бронированиях).

### Технические особенности

* Формат данных: JSON
* Статусные коды: Четкое соответствие REST-стандартам (200, 400, 401, 403 и др.).
* Частичное обновление: Поддержка PATCH для точечного изменения данных.
* Документирование: OpenAPI (Swagger) для удобства разработчиков.

## Начало работы

**1. Войти в аккаунт**

Для работы с API необходимо зарегистрироваться и получить API-ключ.

**2. Получить API-ключ**

Отправьте POST-запрос на /auth/api-key с логином и паролем:

*Пример запроса:*

POST /auth/api-key HTTP/1.1

Host: api.library.com

Content-Type: application/json

{

"username": "user@example.com",

"password": "securepassword123"

}

*Пример ответа:*

{

"api\_key": "a1b2c3d4e5f6",

"secret\_key": "s3cr3tk3y!123"

}

Важно: secret\_key используется для HMAC и должен храниться в безопасности.

**3. Создание запроса**

*Каждый запрос должен содержать:*

* Заголовок Authorization с HMAC-подписью.
* Заголовок X-API-Key с вашим api\_key.

*Формат HMAC:*

Создайте строку для подписи:

METHOD + PATH + TIMESTAMP + BODY (если есть)

Подпишите её с помощью secret\_key и алгоритма SHA-256.

*Пример запроса (GET /books):*

GET /books HTTP/1.1

Host: api.library.com

X-API-Key: a1b2c3d4e5f6

X-Timestamp: 1620000000

Authorization: HMAC-SHA256 1a2b3c4d5e6f...

**4. Оценка ответа**

*Успешный ответ содержит данные в JSON:*

{

"data": [

{ "id": 1, "title": "Book 1", "author": "Author 1" },

{ "id": 2, "title": "Book 2", "author": "Author 2" }

]

}

*Ошибки возвращаются в формате:*

{

"error": "Invalid API key",

"code": 401

}

## Требования аутентификации и авторизации API

**1. Как получить API-ключ?**

* Отправьте POST-запрос на /auth/api-key с логином и паролем.
* В ответе вы получите api\_key и secret\_key.

**2. Как пройти аутентификацию запроса?**

Включите в заголовки:

* X-API-Key – ваш api\_key.
* X-Timestamp – текущее время в Unix-формате.
* Authorization – HMAC-подпись запроса.

Формирование подписи:

import hmac

import hashlib

method = "GET"

path = "/books"

timestamp = "1620000000"

body = "" # если есть тело запроса

secret\_key = "s3cr3tk3y!123"

message = f"{method}{path}{timestamp}{body}".encode()

signature = hmac.new(secret\_key.encode(), message, hashlib.sha256).hexdigest()

auth\_header = f"HMAC-SHA256 {signature}"

**3. Чувствительность информации аутентификации**

* secret\_key никогда не должен передаваться в запросах.
* Все запросы должны отправляться через HTTPS.

**4. Период действия токена доступа**

* API-ключ не имеет срока действия, но может быть отозван администратором.
* Рекомендуется периодически обновлять secret\_key через /auth/rotate-key.

## Коды статусов и ошибок

|  |  |
| --- | --- |
| **Код** | **Описание** |
| 200 | Успешный запрос |
| 201 | Ресурс создан |
| 204 | Запрос выполнен успешно, но ответ пуст |
| 400 | Некорректные входные данные или ошибка формата |
| 401 | Ошибка аутентификации |
| 403 | Доступ запрещен |
| 404 | Ресурс не найден |
| 409 | Попытка создать дубликат ресурса или конфликтующие изменения |
| 429 | Превышен лимит запросов |
| 500 | Ошибка сервера |

## Ограничения скорости

* 100 запросов в минуту для одного пользователя.
* 10 запросов в секунду на один эндпоинт.
* При превышении лимита – код 429 Too Many Requests.

## Образцы кода

### Авторизация, обновление токена, смена пароля и получение API-ключа

**Java:**

import java.net.URI;

import java.net.http.\*;

import java.net.http.HttpResponse.BodyHandlers;

import java.util.Map;

import com.fasterxml.jackson.databind.ObjectMapper;

public class AuthClient {

private static final String BASE\_URL = "https://api.library.com";

private static final HttpClient client = HttpClient.newHttpClient();

private static final ObjectMapper mapper = new ObjectMapper();

public static String login(String username, String password) throws Exception {

String body = mapper.writeValueAsString(Map.of("username", username, "password", password));

HttpRequest request = HttpRequest.newBuilder()

.uri(new URI(BASE\_URL + "/auth/login"))

.header("Content-Type", "application/json")

.POST(HttpRequest.BodyPublishers.ofString(body))

.build();

HttpResponse<String> response = client.send(request, BodyHandlers.ofString());

return response.body();

}

public static String refresh(String refreshToken) throws Exception {

String body = mapper.writeValueAsString(Map.of("refresh\_token", refreshToken));

HttpRequest request = HttpRequest.newBuilder()

.uri(new URI(BASE\_URL + "/auth/refresh"))

.header("Authorization", "Bearer " + refreshToken)

.header("Content-Type", "application/json")

.POST(HttpRequest.BodyPublishers.ofString(body))

.build();

HttpResponse<String> response = client.send(request, BodyHandlers.ofString());

return response.body();

}

public static String changePassword(String accessToken, String currentPassword, String newPassword) throws Exception {

String body = mapper.writeValueAsString(Map.of(

"current\_password", currentPassword,

"new\_password", newPassword

));

HttpRequest request = HttpRequest.newBuilder()

.uri(new URI(BASE\_URL + "/auth/password"))

.header("Authorization", "Bearer " + accessToken)

.header("Content-Type", "application/json")

.method("PATCH", HttpRequest.BodyPublishers.ofString(body))

.build();

HttpResponse<String> response = client.send(request, BodyHandlers.ofString());

return response.body();

}

public static String getApiKey(String apiKey, int userId) throws Exception {

String body = mapper.writeValueAsString(Map.of("user\_id", userId));

HttpRequest request = HttpRequest.newBuilder()

.uri(new URI(BASE\_URL + "/auth/api-key"))

.header("X-API-Key", apiKey)

.header("Content-Type", "application/json")

.POST(HttpRequest.BodyPublishers.ofString(body))

.build();

HttpResponse<String> response = client.send(request, BodyHandlers.ofString());

return response.body();

}

}

**Python:**

import requests

BASE\_URL = "https://api.library.com"

# Вход в систему

def login(username, password):

response = requests.post(f"{BASE\_URL}/auth/login", json={

"username": username,

"password": password

})

return response.json()

# Обновление access-токена

def refresh\_token(refresh\_token):

headers = {"Authorization": f"Bearer {refresh\_token}"}

response = requests.post(f"{BASE\_URL}/auth/refresh", headers=headers, json={

"refresh\_token": refresh\_token

})

return response.json()

# Смена пароля

def change\_password(access\_token, current\_password, new\_password):

headers = {

"Authorization": f"Bearer {access\_token}",

"Content-Type": "application/json"

}

response = requests.patch(f"{BASE\_URL}/auth/password", headers=headers, json={

"current\_password": current\_password,

"new\_password": new\_password

})

return response.json()

# Получение API-ключа

def get\_api\_key(api\_key, user\_id):

headers = {

"Content-Type": "application/json",

"X-API-Key": api\_key

}

response = requests.post(f"{BASE\_URL}/auth/api-key", headers=headers, json={

"user\_id": user\_id

})

return response.json()

### Обновление описания книги через PATCH-запрос

**Java:**

import java.net.URI;

import java.net.http.HttpClient;

import java.net.http.HttpRequest;

import java.net.http.HttpResponse;

import java.net.http.HttpRequest.BodyPublishers;

public class UpdateBookDescription {

public static void main(String[] args) throws Exception {

String url = "https://api.library.example.com/books/153";

String json = """

{

"description": "Обновлённое описание книги с более точным изложением сюжета и контекста."

}

""";

HttpRequest request = HttpRequest.newBuilder()

.uri(URI.create(url))

.method("PATCH", BodyPublishers.ofString(json))

.header("Content-Type", "application/json")

.header("X-API-Key", "abc123key")

.header("X-Signature", "a17bc3...")

.header("X-Timestamp", "2025-05-08T12:10:00Z")

.build();

HttpClient client = HttpClient.newHttpClient();

HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

System.out.println(response.statusCode());

System.out.println(response.body());

}

}

**Python:**

import requests

import json

from datetime import datetime

url = "https://api.library.example.com/books/153"

headers = {

"Content-Type": "application/json",

"X-API-Key": "abc123key",

"X-Signature": "a17bc3...",

"X-Timestamp": datetime.utcnow().isoformat() + "Z"

}

payload = {

"description": "Обновлённое описание книги с более точным изложением сюжета и контекста."

}

response = requests.patch(url, headers=headers, data=json.dumps(payload))

print("Status code:", response.status\_code)

print("Response body:", response.json())

## Ресурс «Книги»

### 1. Описание ресурса

Ресурс **"Книги"** предоставляет доступ к каталогу книг библиотеки.

Поддерживает:

* Получение списка книг с фильтрацией и сортировкой
* Добавление, обновление и удаление книг
* Управление доступностью книг

### 2. Конечные точки и методы

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Конечная точка** | **Описание** | **Доступ** |
| POST | /books | Добавить новую книгу | Библиотекарь, Администратор |
| GET | /books | Получить список всех книг | Библиотекарь, Администратор |
| GET | /books/{id} | Получить информацию о книге по ID | Библиотекарь, Администратор |
| PUT | /books/{id} | Полностью обновить информацию о книге | Библиотекарь, Администратор |
| PATCH | /books /{id} | Обновить часть данных о книге | Библиотекарь, Администратор |
| DELETE | /books/{id} | Удалить книгу | Администратор |

### 3. Параметры

3.1. Параметры пути

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| id | integer | Да | Уникальный идентификатор книги | 25 |

3.2. Параметры строки запроса (для GET /books)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| available | boolean | Нет | Фильтр по доступности | true |
| author | string[] (массив) | Нет | Фильтрация книг по одному или нескольким авторам | author=Neil Gaiman&author=Terry Pratchett |
| genre | string[] (массив) | Нет | Фильтрация по жанрам (можно указать несколько) | genre=Fantasy&genre=Comedy |
| title | string | Нет | Поиск по названию книги (подстрока) | title=Good Omens |
| sort\_by | string | Нет | Поле сортировки (title, published\_date, author) | sort\_by=published\_date |
| order | string | Нет | Порядок сортировки: asc или desc | order=asc |
| limit | integer | Нет | Лимит книг (макс. 100) | 10 |
| offset | integer | Нет | Смещение (для пагинации) | 20 |

3.3. Параметры тела запроса (POST, PUT, PATCH)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность для POST и PUT** | **Обязательность (PATCH)** | **Описание** | **Пример** |
| isbn | string | Да | Нет | Уникальный ISBN | "978-5-699-12014-7" |
| available\_copies | integer | Да | Нет | Количество доступных копий | 5 |
| available | boolean | Нет | Нет | Доступность для бронирования | true |
| title | string | Да | Нет | Название книги | "Good Omens" |
| authors | string[] (массив) | Да | Нет | Список авторов | ["Neil Gaiman", "Terry Pratchett"] |
| genres | string[] (массив) | Да | Нет | Список жанров | ["Fantasy", "Comedy"] |
| published\_date | string (ISO 8601) | Нет | Нет | Дата публикации | "1990-05-01" |
| description | string | Нет | Нет | Краткое описание книги | "A comedic novel about the apocalypse..." |

3.4. Параметры заголовка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| X-API-Key | string | Да | Уникальный ключ для доступа к API | abc123key |
| X-Signature | string | Да | HMAC-подпись запроса | 5f2d9e7c... |
| X-Timestamp | string | Да | Метка времени запроса ISO 8601 | 2025-05-08T10:00:00Z |

### 4. Примеры запросов, ответов, JSON Schema

#### 4.1. GET /books

##### А. Успешный запрос

**Пример запроса:**

curl -X GET "https://api.library.example.com/books?author=Neil%20Gaiman&genre=Fantasy&limit=2" \

-H "X-API-Key: abc123key" \

-H "X-Signature: 39fae...6c7a" \

-H "X-Timestamp: 2025-05-08T12:00:00Z"

**Пример успешного ответа (HTTP 200 OK):**

{

"total": 12,

"limit": 2,

"offset": 0,

"results": [

{

"id": 101,

"isbn": "978-0-06-085398-3",

"title": "American Gods",

"authors": ["Neil Gaiman"],

"genres": ["Fantasy", "Drama"],

"published\_date": "2001-06-19",

"description": "A modern fantasy blending mythology and Americana.",

"available": true,

"available\_copies": 4

},

{

"id": 102,

"isbn": "978-0-06-123456-0",

"title": "Neverwhere",

"authors": ["Neil Gaiman"],

"genres": ["Urban Fantasy"],

"published\_date": "1996-09-16",

"description": "A journey through the mysterious world of London Below.",

"available": false,

"available\_copies": 0

}

]

}

**JSON Schema (успешный ответ):**

{

"type": "object",

"properties": {

"total": { "type": "integer" },

"limit": { "type": "integer" },

"offset": { "type": "integer" },

"results": {

"type": "array",

"items": {

"type": "object",

"properties": {

"id": { "type": "integer" },

"isbn": { "type": "string" },

"title": { "type": "string" },

"authors": {

"type": "array",

"items": { "type": "string" }

},

"genres": {

"type": "array",

"items": { "type": "string" }

},

"published\_date": { "type": "string", "format": "date" },

"description": { "type": "string" },

"available": { "type": "boolean" },

"available\_copies": { "type": "integer" }

},

"required": ["id", "isbn", "title", "authors", "genres"]

}

}

},

"required": ["total", "limit", "offset", "results"]

}

##### Б. Неуспешный запрос

***Б.1. Неуспешный запрос 1: Отсутствие ключа API***

**Пример запроса:**

curl -X GET https://api.library.example.com/books?limit=1

**Пример ответа (HTTP 401 Unauthorized):**

{

"error": {

"code": "missing\_api\_key",

"message": "Authentication required",

"details": [

{

"field": "X-API-Key",

"message": "Header is required"

}

]

}

}

***Б.2. Неуспешный запрос 2: Неверная подпись HMAC (403 Forbidden)***

**Пример запроса:**

curl -X GET "https://api.library.example.com/books?limit=1" \

-H "X-API-Key: abc123key" \

-H "X-Signature: invalidsignature" \

-H "X-Timestamp: 2025-05-08T12:01:00Z"

**Пример ответа:**

{

"error": {

"code": "invalid\_signature",

"message": "Request verification failed",

"details": [

{

"field": "X-Signature",

"message": "HMAC signature is invalid"

}

]

}

}

***Б.3. Неуспешный запрос 3: Недопустимое значение параметра (400 Bad Request)***

**Пример запроса:**

curl -X GET "https://api.library.example.com/books?limit=9999" \

-H "X-API-Key: abc123key" \

-H "X-Signature: validsignature" \

-H "X-Timestamp: 2025-05-08T12:03:00Z"

**Пример ответа (HTTP 400 Bad Request):**

{

"error": {

"code": "invalid\_parameter",

"message": "Request contains invalid parameters",

"details": [

{

"field": "limit",

"message": "Value must be between 1 and 100"

}

]

}

}

**JSON Schema — Ошибки (универсальная):**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"error": {

"type": "object",

"properties": {

"code": { "type": "string" },

"message": { "type": "string" },

"details": {

"type": "array",

"items": {

"type": "object",

"properties": {

"field": { "type": "string" },

"message": { "type": "string" }

}

}

}

},

"required": ["code", "message"]

}

},

"required": ["error"]

}

#### 4.2. POST /books

##### А. Успешный запрос

**Пример запроса:**

curl -X POST "https://api.library.example.com/books" \

-H "Content-Type: application/json" \

-H "X-API-Key: abc123key" \

-H "X-Signature: e83c17b..." \

-H "X-Timestamp: 2025-05-08T12:05:00Z" \

-d '{

"isbn": "978-5-17-118366-7",

"title": "Ночной дозор",

"authors": ["Сергей Лукьяненко"],

"genres": ["Фэнтези", "Городское фэнтези"],

"published\_date": "1998-06-10",

"available": true,

"available\_copies": 8,

"description": "Первый роман из серии о дозорах..."

}'

**Пример успешного ответа (201 Created):**

{

"id": 153,

"message": "Book successfully created"

}

**JSON Schema (успешный ответ):**

{

"type": "object",

"properties": {

"id": { "type": "integer" },

"message": { "type": "string" }

},

"required": ["id", "message"]

}

##### Б. Неуспешный запрос

***Б.1 Пример ошибки (400 Bad Request: пропущено обязательное поле)***

{

"error": {

"code": "missing\_required\_field",

"message": "Validation failed",

"details": [

{

"field": "title",

"message": "This field is required"

}

]

}

}

***Б.2 Пример ошибки (409 Conflict: дублирование ISBN)***

{

"error": {

"code": "duplicate\_isbn",

"message": "Resource conflict detected",

"details": [

{

"field": "isbn",

"message": "A book with this ISBN already exists"

}

]

}

}

**JSON Schema — Ошибки (универсальная):**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"error": {

"type": "object",

"properties": {

"code": { "type": "string" },

"message": { "type": "string" },

"details": {

"type": "array",

"items": {

"type": "object",

"properties": {

"field": { "type": "string" },

"message": { "type": "string" }

}

}

}

},

"required": ["code", "message"]

}

},

"required": ["error"]

}

#### 4.2. PATCH /books/{id}

##### А. Успешный запрос

***А.1 Закончились экземпляры книги***

**Пример запроса:**

curl -X PATCH "https://api.library.example.com/books/153" \

-H "Content-Type: application/json" \

-H "X-API-Key: abc123key" \

-H "X-Signature: a17bc3..." \

-H "X-Timestamp: 2025-05-08T12:10:00Z" \

-d '{

"available": false,

"available\_copies": 0

}'

**Пример успешного ответа (200 OK):**

{

"message": "Book successfully updated"

}

***А.2 Обновление описания книги***

**Пример запроса:**

curl -X PATCH "https://api.library.example.com/books/153" \

-H "Content-Type: application/json" \

-H "X-API-Key: abc123key" \

-H "X-Signature: a17bc3..." \

-H "X-Timestamp: 2025-05-08T12:10:00Z" \

-d '{

"description": "Обновлённое описание книги с более точным изложением сюжета и контекста."

}'

**Пример успешного ответа (200 OK):**

{

"message": "Book successfully updated"

}

**JSON Schema (успешный ответ):**

{

"type": "object",

"properties": {

"message": { "type": "string" }

},

"required": ["message"]

}

##### Б. Неуспешный запрос

***Б1. Пример ошибки (404 Not Found)***

{

"error": {

"code": "not\_found",

"message": "Requested resource not found",

"details": [

{

"field": "id",

"message": "Book with id 153 not found"

}

]

}

}

***Б.2 Пример ошибки (400 Bad Request: неверное значение)***

{

"error": {

"code": "invalid\_value",

"message": "Validation error in request data",

"details": [

{

"field": "available\_copies",

"message": "Must be a non-negative integer"

}

]

}

}

***Б.3 Пример ошибки (400 Bad Request: неверный тип данных)***

{

"error": {

"code": "invalid\_type",

"message": "Type validation failed",

"details": [

{

"field": "description",

"message": "Must be a string"

}

]

}

}

**JSON Schema — Ошибки (универсальная):**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"error": {

"type": "object",

"properties": {

"code": { "type": "string" },

"message": { "type": "string" },

"details": {

"type": "array",

"items": {

"type": "object",

"properties": {

"field": { "type": "string" },

"message": { "type": "string" }

}

}

}

},

"required": ["code", "message"]

}

},

"required": ["error"]

}

## Ресурс «Пользователи»

### 1. Описание ресурса

Ресурс "Пользователи" предоставляет интерфейс для управления учетными записями в системе библиотеки.

Включает:

* Регистрацию и аутентификацию пользователей
* Управление ролями (читатель, библиотекарь, администратор)
* Просмотр и редактирование профилей

### 2. Конечные точки и методы

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Конечная точка** | **Описание** | **Доступ** |
| POST | /users | Создать нового пользователя | Открытый доступ |
| GET | /users | Получить список всех пользователей | Администратор |
| GET | /users/{id} | Получить информацию о конкретном пользователе | Пользователь, Библиотекарь, Администратор |
| PUT | /users/{id} | Полностью обновить данные пользователя | Пользователь, Администратор |
| PATCH | /users/{id} | Частично обновить данные пользователя | Пользователь, Администратор |
| DELETE | /users/{id} | Удалить пользователя | Администратор |

### 3. Параметры

3.1. Параметры пути

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| id | integer | Да | Уникальный идентификатор пользователя | 1 |

3.2. Параметры строки запроса (для GET /users)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| role | string | Нет | Фильтр по роли (reader, librarian, admin) | role=admin |
| is\_active | boolean | Нет | Фильтр по активности аккаунта | active= true |
| limit | integer | Нет | Кол-во возвращаемых результатов | 20 |
| offset | integer | Нет | Смещение для пагинации | 0 |

3.3. Параметры тела запроса (POST/PUT/PATCH)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность (POST, PUT)** | **Обязательность (PATCH)** | **Описание** | **Пример** |
| username | string | Да | Нет | Уникальное имя пользователя | "batman1989" |
| email | string | Да | Нет | Электронная почта пользователя | "bruce@wayne.com" |
| full\_name | string | Нет | Нет | Полное имя пользователя | "Bruce Wayne" |
| active | boolean | Нет | Нет | Флаг активности аккаунта | true |
| role | string (enum) | Нет | Нет | Роль пользователя (user/admin) | "admin" |
| password | string | Да (для POST) | Нет | Пароль | "securePassword123" |

3.4. Параметры заголовка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| X-API-Key | string | Да | Уникальный ключ для доступа к API | abc123key |
| X-Signature | string | Да | HMAC-подпись запроса | 5f2d9e7c... |
| X-Timestamp | string | Да | Метка времени запроса ISO 8601 | 2025-05-08T10:00:00Z |

### 4. Примеры запросов, ответов, JSON Schema

##### А. Успешный запрос

***А.1 Успешный POST /users (создание пользователя)***

**Пример запроса:**

curl -X POST "https://api.library.example.com/users" \

-H "Content-Type: application/json" \

-H "X-API-Key: abc123key" \

-H "X-Signature: 5f2d9e7c..." \

-H "X-Timestamp: 2025-05-08T10:00:00Z" \

-d '{

"username": "batman1989",

"email": "bruce@wayne.com",

"full\_name": "Bruce Wayne",

"password": "securePassword123",

"role": "admin",

"active": true

}'

**Пример ответа (201 Created):**

{

"id": 1,

"username": "batman1989",

"email": "bruce@wayne.com",

"full\_name": "Bruce Wayne",

"role": "admin",

"active": true,

"created\_at": "2025-05-08T10:01:00Z"

}

**JSON Schema:**

{

"type": "object",

"properties": {

"id": { "type": "integer" },

"username": { "type": "string" },

"email": { "type": "string", "format": "email" },

"full\_name": { "type": "string" },

"role": { "type": "string", "enum": ["reader", "librarian", "admin"] },

"active": { "type": "boolean" },

"created\_at": { "type": "string", "format": "date-time" }

},

"required": ["id", "username", "email", "role", "active", "created\_at"]

}

***А.2 Успешный PATCH /users/1 (обновление активности)***

**Пример запроса:**

curl -X PATCH "https://api.library.example.com/users/1" \

-H "Content-Type: application/json" \

-H "X-API-Key: abc123key" \

-H "X-Signature: 5f2d9e7c..." \

-H "X-Timestamp: 2025-05-08T10:10:00Z" \

-d '{

"active": false

}'

**Пример ответа (200 OK):**

{

"message": "User successfully updated"

}

**JSON Schema:**

{

"type": "object",

"properties": {

"message": {

"type": "string",

"description": "Сообщение об успешной операции"

}

},

"required": ["message"]

}

##### Б. Неуспешный запрос

***Б.1 Ошибка 401 Unauthorized (неверный API-ключ)***

{

"error": {

"code": "unauthorized",

"message": "Authentication failed",

"details": [

{

"field": "X-API-Key",

"message": "Invalid or missing API key"

}

]

}

}

***Б.2 Ошибка 403 Forbidden (попытка изменить чужие данные)***

{

"error": {

"code": "forbidden",

"message": "Permission denied",

"details": [

{

"field": "user\_id",

"message": "You do not have permission to modify this user"

}

]

}

}

***Б.3 Ошибка 404 Not Found (не найден пользователь)***

{

"error": {

"code": "not\_found",

"message": "Resource not found",

"details": [

{

"field": "id",

"message": "User with id=99 does not exist"

}

]

}

}

**JSON Schema — Ошибки (универсальная):**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"error": {

"type": "object",

"properties": {

"code": { "type": "string" },

"message": { "type": "string" },

"details": {

"type": "array",

"items": {

"type": "object",

"properties": {

"field": { "type": "string" },

"message": { "type": "string" }

}

}

}

},

"required": ["code", "message"]

}

},

"required": ["error"]

}

## Ресурс «Бронирование»

### 1. Описание ресурса

Ресурс "Бронирования" позволяет управлять резервированием книг в библиотеке.

Включает:

* Создание новых бронирований
* Просмотр активных бронирований
* Обновление статуса бронирования
* Отмену бронирований

### 2. Конечные точки и методы

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Конечная точка** | **Описание** | **Доступ** |
| POST | /reservations | Создать новое бронирование | Читатель |
| GET | /reservations | Получить список всех бронирований | Владелец, Библиотекарь, Администратор |
| GET | /reservations/{id} | Получить информацию о бронировании | Владелец, Библиотекарь, Администратор |
| PUT | /reservations/{id} | Полное обновление бронирования | Владелец, Библиотекарь, Администратор |
| PATCH | /reservations/{id} | Частичное обновление бронирования | Владелец, Библиотекарь, Администратор |
| DELETE | /reservations/{id} | Отменить бронирование | Владелец, Библиотекарь, Администратор |

### 3. Параметры

3.1. Параметры пути

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| id | integer | Да | Уникальный идентификатор бронирования | 1 |

3.2. Параметры строки запроса (GET /reservations)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| user\_id | integer | Нет | Фильтрация по пользователю | user\_id=101 |
| book\_id | integer | Нет | Фильтрация по книге | book\_id=2048 |
| status | string | Нет | Статус бронирования (active, returned, overdue) | status=active |
| limit | integer | Нет | Кол-во возвращаемых результатов (макс. 50) | limit=10 |
| offset | integer | Нет | Смещение для пагинации | offset=0 |

3.3. Параметры тела запроса (POST/PUT/PATCH)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность (POST, PUT)** | **Обязательность (PATCH)** | **Описание** | **Пример** |
| user\_id | integer | Да | Нет | ID пользователя | 1 |
| book\_id | integer | Да | Нет | ID бронируемой книги | 24 |
| start\_date | string  (ISO 8601) | Да | Нет | Дата начала бронирования | "2025-05-10" |
| end\_date | string  (ISO 8601) | Да | Нет | Дата окончания бронирования | "2025-05-24" |
| status | string  (enum) | Нет | Нет | Статус бронирования (active, returned, overdue) | "active" |

3.4. Параметры заголовка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| X-API-Key | string | Да | Уникальный ключ для доступа к API | abc123key |
| X-Signature | string | Да | HMAC-подпись запроса | 5f2d9e7c... |
| X-Timestamp | string | Да | Метка времени запроса ISO 8601 | 2025-05-08T10:00:00Z |

### 4. Примеры запросов, ответов, JSON Schema

##### А. Успешный запрос

***А.1 Создание бронирования (POST /reservations)***

**Пример запроса:**

curl -X POST "https://api.library.com/reservations" \

-H "Content-Type: application/json" \

-H "X-API-Key: abc123key" \

-H "X-Signature: 5f2d9e7c..." \

-H "X-Timestamp: 2025-05-08T10:00:00Z" \

-d '{

"user\_id": 101,

"book\_id": 24,

"start\_date": "2025-05-10",

"end\_date": "2025-05-24"

}'

**Пример ответа (201 Created):**

{

"id": 1,

"user\_id": 101,

"book\_id": 24,

"status": "active",

"start\_date": "2025-05-10",

"end\_date": "2025-05-24",

"created\_at": "2025-05-08T10:00:00Z"

}

**JSON Schema:**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"id": { "type": "integer" },

"user\_id": { "type": "integer" },

"book\_id": { "type": "integer" },

"status": {

"type": "string",

"enum": ["active", "returned", "overdue"]

},

"start\_date": { "type": "string", "format": "date" },

"end\_date": { "type": "string", "format": "date" },

"created\_at": { "type": "string", "format": "date-time" }

},

"required": ["id", "user\_id", "book\_id", "status", "start\_date", "end\_date"]

}

***А.2 Получение списка бронирований (GET /reservations)***

**Пример запроса:**

curl -X GET "https://api.library.com/reservations?user\_id=101&status=active&limit=1" \

-H "X-API-Key: abc123key" \

-H "X-Signature: 5f2d9e7c..." \

-H "X-Timestamp: 2025-05-08T10:00:00Z"

**Пример ответа (200 OK):**

{

"data": [

{

"id": 1,

"user\_id": 101,

"book\_id": 24,

"book\_title": "The Great Gatsby",

"status": "active",

"start\_date": "2025-05-10",

"end\_date": "2025-05-24"

}

],

"meta": {

"total": 1,

"limit": 1,

"offset": 0

}

}

**JSON Schema:**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"data": {

"type": "array",

"items": {

"type": "object",

"properties": {

"id": { "type": "integer" },

"user\_id": { "type": "integer" },

"book\_id": { "type": "integer" },

"book\_title": { "type": "string" },

"status": {

"type": "string",

"enum": ["active", "returned", "overdue"]

},

"start\_date": { "type": "string", "format": "date" },

"end\_date": { "type": "string", "format": "date" }

},

"required": ["id", "user\_id", "book\_id", "status"]

}

},

"meta": {

"type": "object",

"properties": {

"total": { "type": "integer" },

"limit": { "type": "integer" },

"offset": { "type": "integer" }

},

"required": ["total", "limit", "offset"]

}

},

"required": ["data", "meta"]

}

##### Б. Неуспешный запрос

***Б.1 Ошибка валидации (400 Bad Request)***

**Запрос (неверные даты):**

curl -X POST "https://api.library.com/reservations" \

-H "Content-Type: application/json" \

-H "X-API-Key: abc123key" \

-H "X-Signature: 5f2d9e7c..." \

-H "X-Timestamp: 2025-05-08T10:00:00Z" \

-d '{

"user\_id": 101,

"book\_id": 24,

"start\_date": "2025-06-01",

"end\_date": "2025-05-01"

}'

**Ответ:**

{

"error": {

"code": "validation\_failed",

"message": "Request data validation failed",

"details": [

{

"field": "end\_date",

"message": "Must be after start\_date"

}

]

}

}

***Б.2 Конфликт (409 Conflict)***

**Запрос (книга уже забронирована):**

curl -X POST "https://api.library.com/reservations" \

-H "Content-Type: application/json" \

-H "X-API-Key: abc123key" \

-H "X-Signature: 5f2d9e7c..." \

-H "X-Timestamp: 2025-05-08T10:00:00Z" \

-d '{

"user\_id": 101,

"book\_id": 24,

"start\_date": "2025-05-10",

"end\_date": "2025-05-24"

}'

**Ответ:**

{

"error": {

"code": "reservation\_conflict",

"message": "Resource conflict detected",

"details": [

{

"field": "book\_id",

"message": "Book is already reserved for this period"

}

]

}

}

***Б.3 Доступ запрещен (403 Forbidden)***

**Запрос (попытка обновить чужое бронирование):**

curl -X PATCH "https://api.library.com/reservations/1" \

-H "Content-Type: application/json" \

-H "X-API-Key: user-key" \

-H "X-Signature: invalid-signature" \

-H "X-Timestamp: 2025-05-08T10:00:00Z" \

-d '{

"status": "returned"

}'

**Ответ:**

{

"error": {

"code": "forbidden\_access",

"message": "Permission denied",

"details": [

{

"field": "reservation\_id",

"message": "You don't have permission to modify this reservation"

}

]

}

}

**JSON Schema — Ошибки (универсальная):**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"error": {

"type": "object",

"properties": {

"code": { "type": "string" },

"message": { "type": "string" },

"details": {

"type": "array",

"items": {

"type": "object",

"properties": {

"field": { "type": "string" },

"message": { "type": "string" }

}

}

}

},

"required": ["code", "message"]

}

},

"required": ["error"]

}

## Ресурс «Аутентификация»

### 1. Описание ресурса

Ресурс "Аутентификация" предоставляет интерфейс для:

* Получения API-ключей и токенов доступа
* Обновления учетных данных
* Выхода из системы
* Восстановления доступа

### 2. Конечные точки и методы

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Конечная точка** | **Описание** | **Доступ** |
| POST | /auth/login | Вход в систему (получение токена) | Все пользователи и гости |
| POST | /auth/refresh | Обновление токена | Пользователь |
| POST | /auth/logout | Выход из системы | Пользователь |
| POST | /auth/password-reset | Запрос сброса пароля | Пользователь |
| POST | /auth/api-key | Получение API-ключа | Пользователь |
| POST | /users/{id} | Удалить пользователя | Администратор |
| PATCH | /auth/password | Смена пароля | Пользователь |

### 3. Параметры

3.1. Параметры тела запроса для POST /auth/login

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| username | string | Да | Логин пользователя | "user123" |
| password | string | Да | Пароль | "securePassword!" |

3.2. Параметры тела запроса для POST /auth/refresh

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| refresh\_token | string | Да | Токен обновления | "eyJhbGciOi..." |

3.3. Параметры тела запроса для POST /auth/api-key

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| user\_id | integer | Да | ID пользователя | 101 |

3.4. Параметры тела запроса для PATCH /auth/password

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| current\_password | string | Да | Текущий пароль | "qwerty123" |
| new\_password | string | Да | Новый пароль | "newSecure!456" |

3.5. Параметры тела запроса для POST /auth/password-reset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| email | string | Да | Email пользователя | example@mail.com |

3.6. Параметры заголовка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Тип данных** | **Обязательность** | **Описание** | **Пример** |
| X-API-Key | string | Да (для /auth/api-key) | Уникальный ключ для доступа к API | abc123key |
| Authorization | string | Да (для /auth/refresh, /auth/logout) | Токен доступа (Bearer) | Bearer eyJhbGciOi... |

### 4. Примеры запросов, ответов, JSON Schema

##### А. Успешный запрос

***А.1 Вход в систему (POST /auth/login)***

**Пример запроса:**

curl -X POST "https://api.library.com/auth/login" \

-H "Content-Type: application/json" \

-d '{

"username": "spiderman",

"password": "withGreatPower123"

}'

**Пример ответа (200 OK):**

{

"access\_token": "eyJhbGciOi...",

"refresh\_token": "eyJhbGciOi...",

"expires\_in": 3600

}

**JSON Schema:**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"access\_token": { "type": "string" },

"refresh\_token": { "type": "string" },

"expires\_in": { "type": "integer" }

},

"required": ["access\_token", "refresh\_token"]

}

***А.2 Обновление токена (POST /auth/refresh)***

**Пример запроса:**

curl -X POST "https://api.library.com/auth/refresh" \

-H "Authorization: Bearer eyJhbGciOi..." \

-H "Content-Type: application/json" \

-d '{

"refresh\_token": "eyJhbGciOi..."

}'

**Пример ответа (200 OK):**

{

"access\_token": "eyJhbGciOi...NEW",

"expires\_in": 3600

}

**JSON Schema:**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"access\_token": {

"type": "string",

"description": "Новый токен доступа"

},

"expires\_in": {

"type": "integer",

"description": "Время жизни токена в секундах",

"minimum": 1

}

},

"required": ["access\_token", "expires\_in"],

"additionalProperties": false

}

***А.3. Получение API-ключа (POST /auth/api-key)***

**Пример запроса:**

curl -X POST "https://api.library.com/auth/api-key" \

-H "Authorization: Bearer eyJhbGciOi..." \

-H "Content-Type: application/json" \

-d '{

"user\_id": 101

}'

**Успешный ответ (201 Created):**

{

"api\_key": "a1b2c3d4e5",

"secret\_key": "s3cr3tk3y!123",

"expires\_at": "2025-12-31T23:59:59Z"

}

**JSON Schema:**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"api\_key": {

"type": "string",

"description": "Публичный ключ API"

},

"secret\_key": {

"type": "string",

"description": "Секретный ключ для подписи запросов"

},

"expires\_at": {

"type": "string",

"format": "date-time",

"description": "Срок действия ключей"

}

},

"required": ["api\_key", "secret\_key", "expires\_at"],

"additionalProperties": false

}

##### Б. Неуспешный запрос

***2.1. Неверные учетные данные (401 Unauthorized)***

**Запрос:**

curl -X POST "https://api.library.com/auth/login" \

-H "Content-Type: application/json" \

-d '{

"username": "wronguser",

"password": "invalid"

}'

**Ответ:**

{

"error": {

"code": "invalid\_credentials",

"message": "Authentication failed",

"details": [

{

"field": "password",

"message": "Incorrect username or password"

}

]

}

}

***2.2. Неверный токен обновления (403 Forbidden)***

**Запрос:**

curl -X POST "https://api.library.com/auth/refresh" \

-H "Content-Type: application/json" \

-d '{

"refresh\_token": "invalid\_token"

}'

**Ответ:**

{

"error": {

"code": "invalid\_refresh\_token",

"message": "Refresh token is invalid or expired",

"details": []

}

}

***2.3. Слабый пароль (400 Bad Request)***

**Запрос:**

curl -X PATCH "https://api.library.com/auth/password" \

-H "Authorization: Bearer eyJhbGciOi..." \

-H "Content-Type: application/json" \

-d '{

"current\_password": "old123",

"new\_password": "123"

}'

**Ответ:**

{

"error": {

"code": "weak\_password",

"message": "Password does not meet requirements",

"details": [

{

"field": "new\_password",

"message": "Must be at least 8 characters"

}

]

}

}

**JSON Schema — Ошибки (универсальная):**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"type": "object",

"properties": {

"error": {

"type": "object",

"properties": {

"code": { "type": "string" },

"message": { "type": "string" },

"details": {

"type": "array",

"items": {

"type": "object",

"properties": {

"field": { "type": "string" },

"message": { "type": "string" }

}

}

}

},

"required": ["code", "message"]

}

},

"required": ["error"]

}

### 5. Особенности реализации

#### **Безопасность:**

Все запросы требуют HTTPS

Пароли передаются только при входе (хэшируются на сервере)

API-ключи выдаются с ограниченным сроком действия

#### **Жизненный цикл токенов:**

Access Token: 1 час

Refresh Token: 30 дней

API Key: 6 месяцев (можно обновить)

#### **Рекомендации:**

Храните secret\_key в безопасном месте

Не передавайте refresh\_token в клиентский код

Используйте механизм автоматического обновления токенов