## 1. Общая характеристика протокола

Взаимодействие между клиентской и серверной частями системы «Домашняя библиотека» осуществляется с использованием **протокола HTTP версии 1.1** поверх защищённого соединения **HTTPS**.

HTTP-протокол используется в соответствии с архитектурным стилем **REST (Representational State Transfer)**, что предполагает:

- отсутствие хранения состояния сессии на стороне сервера;
- использование стандартных HTTP-методов для выполнения операций;
- представление ресурсов в виде уникальных URL;
- передачу данных в формате JSON.

Клиентом системы является веб-браузер пользователя, сервером — сервер приложений, реализующий REST API.

---

## 2. Модель взаимодействия клиент–сервер

Клиентская часть системы инициирует HTTP-запросы к серверу приложения, передавая параметры запроса и данные в теле сообщения.
Сервер обрабатывает запрос, выполняет соответствующую бизнес-логику и возвращает HTTP-ответ с результатом выполнения операции.

Сервер не хранит состояние пользовательской сессии, вся информация о пользователе передаётся в каждом запросе в виде токена аутентификации.

---

## 3. Используемые HTTP-методы

В системе используются следующие HTTP-методы:

### GET

Применяется для получения данных без изменения состояния сервера.
Используется для:

- получения списка библиотек;
- получения списка книг;
- просмотра закладок и цитат.

### POST

Применяется для создания новых ресурсов.
Используется для:

- создания библиотек;
- добавления книг;
- отправки приглашений;

- добавления закладок и цитат.

**PUT**

Применяется для обновления существующих ресурсов.
Используется для:

- обновления прогресса чтения пользователя.

**DELETE**

Применяется для удаления ресурсов.
Используется для:

- удаления закладок;
- удаления цитат;
- отзыва доступа пользователя.

---

**4. Формат передачи данных**

Все данные между клиентом и сервером передаются в формате **JSON** с использованием кодировки **UTF-8**.

**Заголовки HTTP-запроса**

Обязательные заголовки:

```
Content-Type: application/json
Accept: application/json
```

При аутентифицированных запросах также используется заголовок:

```
Authorization: Bearer <JWT-токен>
```

---

**5. Аутентификация и авторизация**

**Аутентификация**

Аутентификация пользователей осуществляется с использованием **JWT (JSON Web Token)**.

После успешного входа в систему сервер возвращает токен, который клиент обязан передавать в каждом последующем запросе.

**Авторизация**

Авторизация выполняется на серверной стороне и заключается в проверке:

- наличия токена;

- корректности токена;
- прав пользователя на доступ к запрашиваемому ресурсу.

Права доступа определяются на основе данных таблицы доступа пользователей к библиотекам.

---

### 6. Структура HTTP-запроса

Типовой HTTP-запрос состоит из:

1. строки запроса (метод и URL);
2. заголовков;
3. тела запроса (для методов POST и PUT).

**Пример HTTP-запроса**

```
POST /api/libraries HTTP/1.1
Host: example.com
Authorization: Bearer eyJhbGciOiJIUzI1...
Content-Type: application/json

{
  "name": "Домашняя библиотека",
  "description": "Книги по программированию",
  "is_public": false
}
```

---

### 7. Структура HTTP-ответа

HTTP-ответ сервера содержит:

- код состояния;
- заголовки ответа;
- тело ответа с результатом операции.

**Пример HTTP-ответа**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": 5,
  "name": "Домашняя библиотека",
  "description": "Книги по программированию",
  "is_public": false
}
```

---

### 8. Коды состояния HTTP

В системе используются следующие коды состояния:

| Код | Описание |
|---|---|
| 200 OK | Запрос успешно выполнен |
| 201 Created | Ресурс успешно создан |
| 400 Bad Request | Ошибка в данных запроса |
| 401 Unauthorized | Пользователь не аутентифицирован |
| 403 Forbidden | Недостаточно прав доступа |
| 404 Not Found | Запрашиваемый ресурс не найден |
| 500 Internal Server Error | Внутренняя ошибка сервера |

---

**9. Обработка ошибок**

При возникновении ошибок сервер возвращает структурированный JSON-ответ, содержащий описание ошибки.

**Пример ответа с ошибкой**

```
{
  "error": "Access denied",
  "message": "User has no access to this library"
}
```

Это позволяет клиенту корректно обрабатывать ошибки и отображать пользователю соответствующие сообщения.

---

**10. Идемпотентность и безопасность**

- Методы **GET**, **PUT** и **DELETE** являются идемпотентными.
- Метод **POST** используется для операций создания ресурсов.
- Все соединения выполняются по HTTPS, что обеспечивает конфиденциальность передаваемых данных.
- Сервер проверяет права доступа для каждого запроса.