**1. MIME**

**Аббревиатура MIME** означает "Multipurpose Internet Mail Extensions" - Многоцелевые расширения интернет-почты. Это стандарт, который используется для определения типа контента, передаваемого по интернету. MIME позволяет отправлять различные типы данных (текст, изображения, видео и т. д.) через протоколы, которые изначально предназначались для передачи текстовых данных.

**Организации, поддерживающие MIME:** MIME поддерживается IETF (разрабатывает и публикует разные стандарты), IANA (управляет реестром типов, отвечает за назначение и регистрацию типов), W3C (работают над стандартами, связанными с передачей данных в интернете) и определяется в документах RFC (Request for Comments).

**Компоненты запросов и ответов, использующие MIME:**

* **HTTP-заголовок Content-Type**: Указывает тип содержимого, отправляемого в запросе или ответе.
* **HTTP-заголовок Accept**: Указывает типы контента, которые клиент может обрабатывать.
* **Файлы и данные в теле HTTP-запросов и ответов**: MIME-тип помогает интерпретировать содержимое.

**2. Теги HTML, приводящие к HTTP-запросам**

* **<img>**: Загружает изображение по указанному URL.
* **<link>**: Загружает стили CSS.
* **<script>**: Загружает JavaScript-файлы.
* **<audio>** и **<video>**: Загружают медиафайлы.
* **<iframe>**: Загружает другой HTML-документ.
* **<source>**: Используется внутри <audio> и <video> для указания источников медиа.

**3. Способы выполнения HTTP-запросов из JS-сценария**

* **XMLHttpRequest**: Старый метод для выполнения асинхронных запросов.
* **fetch API**: Современный метод, который возвращает промис (промежуточный результат асинхронной операции) и поддерживает удобный синтаксис.
* **Axios**: Библиотека для выполнения HTTP-запросов, которая работает как на клиенте, так и на сервере. Поддерживает промисы и имеет более удобный интерфейс.
* **jQuery.ajax()**: Метод из библиотеки jQuery для выполнения AJAX-запросов.

**4. Параметризованный модуль**

**Параметризованный модуль** — это модуль, который принимает параметры (аргументы) при инициализации или вызове. Это позволяет создавать более гибкие и переиспользуемые компоненты.

Например, в контексте JavaScript, параметризованный модуль может быть функцией или классом, который принимает настройки или конфигурацию:

javascript

function createModule(config) {

return {

doSomething: function() {

console.log(config.option);

}

};

}

const myModule = createModule({ option: 'value' });

myModule.doSomething(); // Вывод: value

Этот подход позволяет изменять поведение модуля в зависимости от переданных параметров.