АО «АЛМАТИНСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»
https://atu.edu.kz/

Факультет «Информационные технологии»
Кафедра «Компьютерная инженерия»

Название курса: «Разработка клиентских Web-приложений»



Зиятбекова Гулзат Зиятбеккызы – PhD, ассоциированный профессор
g.ziyatbekova@atu.edu.kz

ЛЕКЦИЯ №15 · CLIENT-SIDE WEB DEVELOPMENT

асинхронный JavaScript

Как связать JavaScript с данными, сетью и клиентским хранением.

01 · ЗАЧЕМ

Почему браузеру нужна асинхронность



Серверные запросы

Требуют времени: ответ приходит не сразу.



Загрузка файлов

Операция продолжается, пока интерфейс живёт.



Внешние API

Асинхронность не блокирует интерфейс.

Цель лекции Связать JavaScript с данными, сетью и клиентским хранением.

02 · ДВЕ МОДЕЛИ

Синхронный и асинхронный код

СИНХРОННЫЙ

Шаг за шагом

Операции выполняются последовательно и задерживают следующий шаг.

Долгая операция удерживает выполнение.

АСИНХРОННЫЙ

Запустить и продолжить

Длительная операция запускается, а программа продолжает реагировать.

Результат обрабатывается после завершения операции.

Главная разница · не когда запускается операция, а блокирует ли она следующий шаг.

03 · CALLBACK

Callback — функция обратного вызова

01

Передать

Callback передаётся другой функции.

02

Запустить

Основная операция выполняется.

03

Вызвать

Callback вызывается после события или завершения операции.

ПОДХОД УДОБЕН

Для простых сценариев.

ПРОБЛЕМА

Цепочки callback могут становиться сложными для чтения.

Следующий шаг: Promise и async/await делают такую логику понятнее.

04 · PROMISE

Promise описывает результат в будущем

Promise

Результат асинхронной операции

1. **Pending**

   Операция выполняется.

2. **Fulfilled**

   Операция завершилась успешно.

3. **Rejected**

   Операция завершилась ошибкой.

Один Promise описывает путь от ожидания к успешному результату или ошибке.

05 · ASYNC / AWAIT

Async/Await делает код последовательным

```
async function loadData() {
  try {
    const response = await getData()
    return response
  } catch (error) {
    console.log(error)
  }
}
```

Код выглядит последовательным, хотя ожидает Promise.

1. **async**

   Помечает функцию как асинхронную.

2. **await**

   Ожидает результат Promise внутри async-функции.

3. **try/catch**

   Выносит обработку ошибок в ясную структуру.

Fetch API связывает приложение с сервером

1	2	3
fetch Выполнить HTTP-запрос.	response.json() Преобразовать ответ.	Обработка Работать с данными приложения.

Fetch — браузерный API для выполнения HTTP-запросов из JavaScript.

GET получает данные · **POST** отправляет новые данные.

HTTP-запросы: получить и отправить

GET

Получить список товаров

```
fetch("/api/products")
```

Запрос данных через fetch.

POST

Отправить новые данные

```
fetch("/api/products", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify(product)
})
```

GET читает ресурс; POST передаёт тело запроса и описывает его формат.

JSON — общий язык клиента и сервера

ОБЪЕКТ ТОВАРА

```
{
  "name": "Кофе",
  "price": 450,
  "available": true
}
```

Формат хранения и передачи структурированных данных.

1. **response.json()**

   Преобразует ответ сервера в данные JavaScript.

2. **JSON.stringify()**

   Подготавливает объект к отправке.

09 · ХРАНЕНИЕ И ПОВТОРЕНИЕ

LocalStorage и SessionStorage

localStorage

После закрытия браузера

Сохраняет небольшие данные даже после закрытия браузера.

Базовые операции: setItem, getItem, removeItem.

sessionStorage

В течение текущей сессии

Хранит данные в течение текущей сессии браузера.

Примеры применения: имя пользователя, корзина или настройки интерфейса.

Контрольные вопросы

- Какое состояние Promise означает ошибку?
- Что делает await?
- Какой метод отправляет новые данные?
- Чем отличаются localStorage и sessionStorage?