Приднестровский государственный университет им. Т.Г. Шевченко
Физико-технический институт

ОСНОВЫ ВЕБ-РАЗРАБОТКИ С ПРИМЕНЕНИЕМ ИИ-АССИСТЕНТОВ

Лабораторный практикум

Разработал:
ст. преподаватель
кафедры ИТ
Бричаг Д.В.

г. Тирасполь
2025 г.

Лабораторная работа №7

jQuery – «Что? Где? Когда?»

Что это такое, где его применяют, когда можно использовать.

Цель работы:

- познакомиться с jQuery, понять его сильные и слабые стороны,
- научиться читать и переписывать jQuery-код на чистый JavaScript.

Теоретическая справка

jQuery — это библиотека JavaScript, созданная в 2006 году Джоном Резигом. Ее девиз: "Пиши меньше, делай больше" (Write less, do more).

Чтобы понять ее взрывную популярность, нужно представить себе "дикий запад" веб-разработки 2000-х.

Проблема №1: Война браузеров

Тогда Internet Explorer, Firefox, Opera и Safari не просто конкурировали — они активно друг другу "вредили", реализуя одни и те же вещи по-разному.

События: Чтобы "повесить" клик на кнопку, в Internet Explorer нужно было писать element.attachEvent('onclick', ...) , а во всех остальных — element.addEventListener('click', ...). Вам приходилось писать if...else для каждого браузера.

AJAX: Объект XMLHttpRequest (для запросов к серверу без перезагрузки страницы) в IE создавался одним способом (new ActiveXObject()), а в других — другим (new XMLHttpRequest()).

Проблема №2: Ужасный DOM API

До появления querySelector (2013 г.), чтобы найти элемент, приходилось использовать громоздкие и неудобные методы:

document.getElementById('id') (только по ID)

document.getElementsByTagName('div') (возвращал "живую" коллекцию, а не удобный массив)

document.getElementsByClassName('name') (вообще не поддерживался в старых IE)

Найти "все ссылки с классом .nav-link внутри div с ID #menu" было настоящей головной болью.

Как jQuery все это "вылечил"

jQuery предоставил единый, простой и удобный "пульт управления" страницей.

1. "Швейцарский нож" — селектор $

Главная звезда jQuery — это функция $(). Она была как querySelector + querySelectorAll + еще 100 утилит в одной функции.

Вы просто писали селектор, как в CSS, и получали объект jQuery, который уже "знал", с какими элементами работать (будь то один или сто).

```
// Найти ВСЕ элементы с классом '.card'
const $allCards = $('.card');
```

2. Кросс-браузерность "под капотом"

jQuery брал всю "грязную" работу на себя.

```
// jQuery-код (просто и понятно)
$('.btn').click(() => console.log('Клик!'));
```

Что на самом деле делал jQuery:

```
«Ага, разработчик хочет клик. Сейчас я проверю браузер. Если это IE8, я использую attachEvent. Если это Firefox, я использую addEventListener. ...А разработчику я ничего об этом не скажу, пусть просто работает».
```

Цепочки вызовов (Method Chaining)

Это была одна из самых "залипательных" фишек. Вместо того чтобы писать код в столбик, можно было "нанизывать" методы один на другой:

```
// jQuery-магия
$('.modal-title')                 // 1. Найти элемент
  .text('Новый заголовок') // 2. Поменять ему текст
  .addClass('highlight')        // 3. Добавить CSS-класс
  .parent()                          // 4. Перейти к его родителю (modal-header)
  .slideDown(500);              // 5. Плавно "выехать" (анимация за 500 мс)
```

Это читалось почти как английское предложение и было невероятно мощно.

Простой AJAX

Вместо сложной настройки XMLHttpRequest, вы просто писали:

```
$.ajax({
  url: 'https://api.example.com/data',
  method: 'GET',
  success: (data) => {
    // Ура, данные пришли!
    console.log(data);
  },
  error: (err) => {
    // Ой, ошибка
    console.error(err);
  }
});
```

Почему нужно знать jQuery – работа с «легаси»

Если после всего вышесказанного вы зададитесь вопросом, зачем изучать устаревающую библиотеку, вы будете правы, но только от части. Да, на jQuery сегодня не начинают новые проекты. Но вы обязательно с ним столкнетесь.

- Археология кода (Legacy): Более 70% всех сайтов в мире до сих пор так или иначе используют jQuery. Это не «старые» сайты, это успешные сайты, которые работают и приносят деньги. Даже «пустая» главная страница поискового сервиса Google подключена к библиотеке jQuery. Рано или поздно вас попросят «добавить фичу вот на этом 10-летнем проекте».
- CMS и темы: Весь мир WordPress, Joomla, Drupal, OpenCart построен на jQuery. Любая купленная «тема» (шаблон) для WordPress будет напичкана плагинами, которые работают на jQuery.
- Плагины: Тысячи готовых решений (слайдеры, карусели, модальные окна, календари, галереи) написаны как jQuery-плагины. Часто бывает проще взять готовый, чем писать свой.

Ключевой навык сегодня — не писать на jQuery, а уметь мигрировать с него. То есть, взять старый код и переписать его на чистый (Vanilla) JavaScript, чтобы «выпилить» библиотеку и ускорить сайт.

Почему не стоит писать на jQuery сегодня

Эпоха «дикого запада» закончилась. Браузеры (Chrome, Firefox, Safari, Edge) договорились и стали следовать веб-стандартам.

Современный JavaScript (ES6+) — это и есть «новый jQuery». Все, что делал jQuery, теперь можно сделать так же просто (или даже проще) "родными" средствами браузера.

Давайте посмотрим на «переводчик» с jQuery на чистый JS (его еще называют Vanilla JS).

Задача	jQuery	Чистый (Vanilla) JS
Выбор одного элемента	const header = $('#header');	const header = document.querySelector('#header');
Выбор всех элементов	const items = $('.item');	const items = document.querySelectorAll('.item');

Обработка клика	$('.btn').click(myFunc);	document.querySelector('.btn').addEventListener('click', myFunc);
Работа с классами	$(el).addClass('active');$(el).removeClass('active');$(el).toggleClass('active');	el.classList.add('active');el.classList.remove('active');el.classList.toggle('active');
Скрыть/Показать	$(el).hide();$(el).show();	el.style.display = 'none';el.style.display = 'block';
Изменить текст	$(el).text('Привет');	el.textContent = 'Привет';
Изменить HTML	$(el).html('Жирный');	el.innerHTML = 'Жирный';
AJAX-запрос (GET)	$.get(url, callback);	fetch(url).then(res => res.json()).then(data => ...);
Работа с атрибутом	$(el).attr('href', '#');	el.setAttribute('href', '#');
Добавить в DOM	$(parent).append(child);	parent.append(child);

Живой пример:

```
// jQuery
$('.card').hide();
```

Что тут происходит? jQuery находит ВСЕ элементы с классом .card и внутренним циклом проходит по ним, выставляя им style.display = 'none'. Это называется неявной итерацией (implicit iteration) — очень удобно.

JavaScript

```
// чистый JS
document.querySelectorAll('.card')
  .forEach(el => el.style.display = 'none');
```

Что тут происходит? querySelectorAll возвращает NodeList (похож на массив). Он не умеет сам применять стили ко всем. Мы должны явно (explicitly) пройти по нему циклом forEach и применить стиль к каждому элементу el.

Итог:

- **Лишний вес**: jQuery – это десятки килобайт JS-кода, который пользователю нужно скачать, а браузеру – проанализировать и выполнить, прежде чем ваш собственный код начнет работать.
- **Декларативный подход**: Современная разработка ушла в сторону фреймворков (React, Vue, Svelte). Они используют декларативный подход («что я хочу видеть на экране»), который принципиально отличается от императивного подхода jQuery («найди этот элемент, потом измени его, потом передвинь»).

Практическая часть

Сформируем простое техническое задание. Необходимо:

Реализовать интерактивные скрипты при помощи библиотеки jQuery и на примерах сравнить работу нативного JavaScript и с использованием библиотеки. Для этого реализовать:

- навигацию и плавную прокрутку;
- переключатели контента (вкладки);
- AJAX-запросы;
- галерею с эффектами.

Подготовка проекта

- Как и в предыдущих работах, создайте папку lab7.
- Тут тоже без изменений, добавьте файлы:
  - index.html
  - style.css
  - script.js – здесь пока пусто
- Подключите jQuery:

```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
<script src="script.js"></script>
```

Проверка: Добавьте в script.js базовую проверку, что jQuery загрузился. Это — канонический способ дождаться загрузки DOM в jQuery:

JavaScript

```
$(document).ready(function() {
  // Это выполнится, когда страница будет готова
  console.log('jQuery готов к работе!');
  // Можете даже добавить что-то видимое:
  $('body').css('background-color', '#f9f9f9');
});
```

Откройте консоль в браузере и убедитесь, что видите сообщение.

Здесь нам потребуется вёрстка страницы. Вы можете взять готовую вёрстку лендинга без скриптов или сгенерировать новую, как в предыдущих работах.

Ниже будет показан пример вёрстки, которая будет оживляться JavaScript, вы можете создать свои блоки, секции и страницы.

Задачи: jQuery против Vanilla JS

Для каждой задачи ниже мы пройдем 4 этапа:

1. Создание HTML/CSS (базовая разметка).
2. Решение на jQuery (с помощью ИИ).
3. Решение на Vanilla JS (с помощью ИИ).
4. Анализ (сравнение синтаксиса).

Задача А: Манипуляция DOM и события (вкладки)

Классическая задача: переключение контента без перезагрузки страницы.

1. HTML: Добавьте в index.html:

```
<div class="tabs-container">
    <div class="tabs">
        <button class="tab-btn active" data-tab="tab-1">О нас</button>
        <button class="tab-btn" data-tab="tab-2">Отзывы</button>
        <button class="tab-btn" data-tab="tab-3">Контакты</button>
    </div>
    <div class="tab-content active" id="tab-1">
        <p>Содержимое "О нас".</p>
    </div>
    <div class="tab-content" id="tab-2">
        <p>Содержимое "Отзывы".</p>
    </div>
    <div class="tab-content" id="tab-3">
        <p>Содержимое "Контакты".</p>
    </div>
</div>
```

2. Библиотека jQuery и промпт для ИИ для создания скрипта:

```
"Напиши код на jQuery для табов. По клику на кнопку с классом .tab-btn нужно:
Удалить класс active у всех кнопок .tab-btn и у всех .tab-content.
Добавить класс active нажатой кнопке.
Получить значение data-tab у нажатой кнопки (например, "tab-1").
Найти элемент по ID (который получили в п.3) и добавить ему класс active."
```

3. Обычный JS или Vanilla JS и промпт для ИИ:

```
"Перепиши этот jQuery-код на чистый JavaScript (Vanilla JS), используя querySelectorAll, addEventListener и classList."
```

4. Анализ полученного результата**:**

- Попросите ИИ объяснить оба фрагмента кода.
- Сравните селекторы: $('.tab-btn') vs document.querySelectorAll('.tab-btn').
- Сравните обработчик клика: $(...).click(function() { ... }) vs element.addEventListener('click', () => { ... }).
- Сравните работу с классами: $(...).addClass('...') / removeClass('...') vs element.classList.add('...') / remove('...').
- Сравните получение data- атрибута: $(this).data('tab') vs element.dataset.tab.

Задача B: Анимация: плавная прокрутка к якорю

jQuery был королем анимации. Его метод .animate() был спасением.

1. HTML: Добавьте меню и несколько секций с id:

```
<nav>
    <a href="#section-1">Секция 1</a>
    <a href="#section-2">Секция 2</a>
</nav>
<div id="section-1" style="height: 100vh; background: #ecec;">Секция 1</div>
<div id="section-2" style="height: 100vh; background: #dada;">Секция 2</div>
```

2. Промпт для jQuery:

```
"Напиши код на jQuery, который перехватывает клик по ссылке a[href^='#'], отменяет стандартное поведение и делает плавную анимацию прокрутки к элементу, на который указывает href." (Ожидаемый результат будет использовать $('html, body').animate({ scrollTop: ... }, 500);)
```

3. Промпт для Vanilla:

```
"Как сделать то же самое (плавную прокрутку к якорю) на чистом JS? Используй e.pre-
ventDefault() и scrollIntoView."
```

4. Анализ кода для библиотеки и простого JS:

- Сравните лаконичность jQuery .animate() с современным методом element.scrollIntoView({ behavior: 'smooth' }).
- Обратите внимени, что эта задача – идеальный пример того, как браузеры «впитали» функциональность jQuery. То, что требовало кастомной анимации (.animate()), теперь является одной нативной строкой в JS.

Задача C: AJAX запросы и загрузка данных

Получение данных с сервера без перезагрузки страницы (AJAX) было сложным. jQuery сделал это тривиальным.

1. HTML: Добавьте кнопку и место для цитаты.

```
<button id="loadQuote">Загрузить цитату</button>
<blockquote id="quote">
    <p>...</p>
    <cite>...</cite>
</blockquote>
```

2. Промпт для jQuery:

```
«Напиши jQuery-код. По клику на кнопку #loadQuote он должен:
Отправить GET-запрос на https://api.quotable.io/random.
Когда придет ответ (data), вставить data.content в blockquote p.
Вставить data.author в blockquote cite.»
```

Обратите внимание, что наш ожидаемый результат: $.get(...) или $.ajax(...))

3. Промт для классического JavaScript:

```
"Перепиши этот AJAX-запрос с $.get на чистый JavaScript, используя fetch и .then()."
```

4. Анализ:

   - Сравните синтаксис $.get(url, callback) с цепочкой fetch(url).then(res => res.json()).then(data => { ... }).
   - Важный момент тут заключается в том, что fetch — это тоже нативный стандарт, который пришел на смену XMLHttpRequest, который jQuery и «прятал». Хотя fetch требует .then() (или async/await), он является более мощным и стандартным подходом.

Задача D: экосистема jQuery UI

jQuery был силен не только ядром, но и огромной экосистемой плагинов. **jQuery UI** — официальная библиотека компонентов, которая значительно расширяла возможности jQuery привнеся в него огромное количество готовых решений для пользовательского интерфейса (user interface – UI).

1. Подключение**:** Добавьте в <head> стили и скрипт jQuery UI (после jQuery).

```
<link rel="stylesheet" href="https://code.jquery.com/ui/1.13.2/themes/base/jquery-
ui.css">
<script src="https://code.jquery.com/ui/1.13.2/jquery-ui.js"></script>
```

2. HTML**:** Добавьте простой div и поле input.

```
<div id="draggable" class="ui-widget-content" style="width: 150px; padding: 10px;">
  <p>Перетащи меня</p>
</div>
<input type="text" id="datepicker">
```

3. Задание:

   - Попросите ИИ:

```
«Как сделать элемент #draggable перетаскиваемым с помощью jQuery UI?»
```

- Попросите ИИ:

```
«Как превратить инпут #datepicker в календарь (datepicker) с помощью jQuery UI?»
```

- Анализ**:** Посмотрите, как легко ($('#draggable').draggable();) добавляется сложный функционал.
- Рефлексия**:** Попросите ИИ:

```
«Напиши код для перетаскиваемого (drag-and-drop) элемента на чистом JS».
```

Сравните объем кода и сложность. Это покажет, что для сложного UI нативные решения (HTML5 Drag and Drop API) могут быть многословнее, чем готовые библиотеки.

Проверка работы скриптов

− Открывая страницу в браузере, убедитесь в отсутствии ошибок в консоли в панели разработчика.

− Проверьте:

- Переключение вкладок меняет активный блок с текстом.
- Клики по навигации плавно прокручивают страницу.
- Устанавливается текст цитаты из удалённого API.
- Работает перетаскивание элементов средствами плагина библиотеки.

− В случае появления ошибок в процессе манипуляций с документом, уточните у нейросети причину и как исправить.

Работа с git и GitHub

1. Инициализируйте репозиторий:

```
git init
git add .
git commit -m "Lab7: jQuery past and future"
```

2. Создайте новый репозиторий на GitHub.

3. Свяжите локальный проект с GitHub:

```
git remote add origin https://github.com/<ваш_логин<lab7_repo>.git
git branch -M main
git push -u origin main
```

Отчёт (в README.md)

В README.md добавить:

- Заголовок: *Лабораторная работа №7. jQuery – сравнительный анализ*
- Скриншоты работы интерактивных элементов.

- Ответы на вопросы:
  - Какие jQuery-функции использовались?
  - Что показалось быстрее/проще — jQuery или чистый JS?
  - Что из этого вы бы оставили, если писали современный сайт?

Результаты работы

В итоге у вас должно быть:

1. Страница с интерактивными элементами.
2. Рабочая секция с вкладками, переключающими активные блоки.
3. Плавная прокрутка страницы.
4. Блок с динамичной цитатой.
5. Секция с перетягиваемыми элементами.
6. Репозиторий на GitHub и отчёт в README.md.

Критерии оценки

- Работают основные jQuery-компоненты (2 балла).
- Реализован AJAX-запрос (2 балла).
- Реализована обработка динамичных цитат (2 балла).
- Реализован блок перетягивания элементов (2 балла).
- Код выложен на GitHub и оформлен отчёт (2 балла).