МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению лабораторных работ
по дисциплине
«Практикум по веб-программированию»
для студентов специальности 10.05.03 Информационная безопасность автоматизированных систем, специализация Анализ безопасности информационных систем

Ставрополь
2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель и задачи освоения дисциплины

Целью освоения дисциплины «Практикум по веб-программированию» является формирование у будущего специалиста по специальности 10.05.03 Информационная безопасность автоматизированных систем специализация «Анализ безопасности информационных систем» понимания основных методов, средств и технологий эффективного взаимодействия и реализации успешной командной работы.
Задачи дисциплины:

– изучение теоретических основ формирования и развития навыков командной работы;
– формирование умений управления групповыми проектами;
– овладение навыками эффективного социального взаимодействия, создания благоприятной и конструктивной атмосферы в команде средствами доступных информационных технологий.

Лабораторная работа 1. Основы HTML и структура веб-страницы

Теоретическая часть:

HTML (HyperText Markup Language) является базовым языком разметки, используемым для создания структуры веб-страниц. Он определяет содержание страницы и её логическую организацию при помощи тегов. Основные элементы включают заголовки, абзацы, списки, изображения, ссылки и таблицы. Семантические теги, такие как <header>, <nav>, <main>, <section>, <article> и <footer>, улучшают восприятие кода как человеком, так и поисковыми системами. Также важны атрибуты, которые добавляют дополнительную информацию к элементам. Строгая структура HTML-документа включает DOCTYPE, корневой элемент <html>, метаинформацию в <head> и содержимое в <body>.

Практические задания:

1. Создать базовую HTML-страницу с заголовками, списками и таблицей.
2. Добавить изображение и ссылку на внешний сайт.
3. Использовать семантическую структуру для разметки содержимого.

Лабораторная работа 2. Основы CSS. Стилизация веб-страницы

Теоретическая часть:

CSS (Cascading Style Sheets) позволяет описывать внешний вид элементов HTML-страницы. Стили могут быть встроенными, внутренними или внешними. Основными концепциями являются селекторы, свойства и значения. CSS управляет цветами, размерами, шрифтами, отступами, границами, расположением и анимациями. Принцип каскадности, специфичность селекторов и наследование играют ключевую роль в том, как применяются стили. Использование flexbox и grid облегчает создание адаптивных макетов. Также важна практика разделения структуры и

представления, поэтому предпочтительно использовать внешние таблицы стилей.

Практические задания:

1. Применить CSS к ранее созданной HTML-странице.
2. Использовать селекторы классов, идентификаторов и вложенные селекторы.
3. Оформить страницу с помощью flexbox.

Лабораторная работа 3. Адаптивная вёрстка. Медиа-запросы. Мобильные-first подход

Теоретическая часть: Адаптивная вёрстка (responsive design) позволяет веб-странице корректно отображаться на устройствах с различными размерами экранов. Основной техникой являются медиа-запросы, позволяющие применять стили в зависимости от ширины, высоты, плотности пикселей и ориентации устройства. Mobile-first — это стратегия, при которой стили сначала пишутся для мобильных устройств, а затем дополняются для планшетов и десктопов. Используются относительные единицы измерения (em, %, vw/vh) и гибкие макеты с flexbox или grid.

Практические задания:

1. Создать адаптивную страницу с тремя зонами (шапка, контент, подвал).
2. Использовать медиазапросы для изменения расположения элементов на мобильных устройствах.
3. Проверить корректность отображения в эмуляторах мобильных браузеров.

Лабораторная работа 4. Основы JavaScript. Первая интерактивность

Теоретическая часть:

JavaScript — это язык программирования, выполняющийся на стороне клиента и позволяющий добавлять интерактивность на веб-страницу. Основные конструкции включают переменные, операторы, условия, циклы и функции. Взаимодействие с DOM (Document Object Model) позволяет изменять содержимое страницы, обрабатывать события и управлять стилями в реальном времени. Современный JavaScript использует стандарты ES6+, включая стрелочные функции, шаблонные строки, деструктуризацию и модули.

Практические задания:

1. Написать скрипт для изменения текста по нажатию кнопки.
2. Реализовать интерактивную форму с валидацией.
3. Использовать цикл для создания динамического списка на странице.

Лабораторная работа 5. Введение в DOM. Обработка событий

Теоретическая часть:

DOM представляет собой структуру HTML-документа в виде дерева, где каждый элемент является узлом, к которому можно получить доступ и изменить его с помощью JavaScript. Обработка событий позволяет реагировать на действия пользователя: щелчки мыши, ввод с клавиатуры, фокус и прокрутку. Важны понятия всплытия событий (event bubbling) и делегирования событий (event delegation), позволяющие управлять обработчиками более эффективно. Методы addEventListener, querySelector, createElement, appendChild, removeChild и setAttribute являются базовыми для манипуляций с DOM.

Практические задания:

1. Реализовать добавление нового элемента в список при нажатии кнопки.
2. Сделать цвет фона изменяемым при наведении курсора.
3. Использовать делегирование событий на списке.

Лабораторная работа 6. Формы и работа с данными пользователя

Теоретическая часть:

Формы являются основным способом получения данных от пользователя. HTML-формы включают различные поля ввода: текстовые, числовые, радиокнопки, чекбоксы, выпадающие списки. JavaScript позволяет получать данные из формы, проводить валидацию и отправлять данные на сервер. Валидация может быть как встроенной (required, pattern), так и пользовательской. Работа с формами также включает предотвращение стандартного поведения (preventDefault) и асинхронную обработку отправки.

Практические задания:

1. Создать форму регистрации с валидацией обязательных полей.
2. Вывести данные формы на страницу без перезагрузки.
3. Проверять корректность email и пароля в реальном времени.

Лабораторная работа 7. Асинхронные запросы: Fetch API и работа с JSON

Теоретическая часть:

Fetch API позволяет отправлять HTTP-запросы к серверу, получать и обрабатывать ответы, не перезагружая страницу. Это основа AJAX (асинхронного взаимодействия). Наиболее часто данные передаются в формате JSON. Асинхронные функции и конструкции async/await упрощают работу с промисами. Работа с сервером требует обработки ошибок, статусов ответов, заголовков и тела ответа.

Практические задания:

1. Получить данные из открытого API и отобразить их на странице.
2. Реализовать отправку данных формы через POST-запрос.
3. Обработать ошибки ответа сервера.

Лабораторная работа 8. Хранение данных в браузере: LocalStorage и SessionStorage

Теоретическая часть:

Web Storage API предоставляет возможность хранения данных в браузере пользователя. localStorage сохраняет данные без срока действия, а sessionStorage — до закрытия вкладки. Данные хранятся в формате ключ-значение и не передаются на сервер автоматически. Эта технология позволяет сохранять состояние пользовательского интерфейса, корзины товаров, тему оформления и прочее. Также необходимо учитывать ограничения по объему и безопасность хранения.

Практические задания:

1. Сохранить данные формы в localStorage и восстановить их при загрузке.
2. Сделать переключатель темы (светлая/тёмная) с сохранением выбора.
3. Хранить счётчик посещений страницы.

Лабораторная работа 9. Основы Git и деплой статического сайта

Теоретическая часть:

Для публикации веб-приложения необходимо использовать систему контроля версий и хостинг. Git позволяет отслеживать изменения, работать с ветками и совместно разрабатывать проект. GitHub Pages предоставляет возможность бесплатного размещения статического сайта прямо из репозитория. Важно понимать этапы инициализации проекта, добавления файлов, фиксации изменений, отправки на сервер и настройки публикации. Также можно использовать Netlify или Vercel для CI-деплоя.

Практические задания:

1. Инициализировать Git-репозиторий и отправить проект на GitHub.
2. Настроить публикацию сайта через GitHub Pages.
3. Автоматизировать деплой с помощью Netlify.