МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное
автономное образовательное
учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ РАБОТАМ

«Основы фронтенд-разработки»

Направление подготовки 09.03.02 «Информационные системы и технологии»
Направленность (профиль) «Прикладное программирование и интернет-технологии»
Квалификация выпускника: бакалавр

Ставрополь

2026

Введение

Цель изучения дисциплины «Основы фронтенд-разработки» формирование у обучающихся фундаментальных знаний и прикладных компетенций в области клиентской веб-разработки, необходимых для проектирования, верстки и программирования современных пользовательских интерфейсов с использованием стандартизированных веб-технологий, а также компетенций будущего бакалавра по направлению подготовки 09.03.02 Информационные системы и технологии.

Задачами обучения являются:

1. Изучить архитектурные принципы веб-документов и освоить семантическую разметку на основе актуальных стандартов HTML5.
2. Освоить механизмы визуального оформления интерфейсов, включая каскадные таблицы стилей, современные системы компоновки и принципы адаптивной вёрстки под различные устройства и разрешения экранов.
3. Сформировать устойчивые навыки программирования на JavaScript, включая работу с типами данных, управление потоком выполнения, функциональные конструкции и объектно-ориентированные подходы.
4. Научиться взаимодействовать с объектной моделью документа (DOM), реализовывать обработку пользовательских событий и динамическое обновление контента без перезагрузки страницы.
5. Освоить принципы асинхронного взаимодействия с серверами, работу с сетевыми интерфейсами, клиентскими хранилищами и базовыми механизмами управления состоянием интерфейса.
6. Познакомиться с инструментарием современной фронтенд-разработки: системами контроля версий, пакетными менеджерами, сборщиками модулей и платформами развертывания статических проектов.

1. Перечень планируемых результатов обучения по дисциплин (модулю), соотнесенных с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по

		дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-2 способен использовать современные инструментальные средства и технологии программирования при разработке прикладного программного обеспечения вычислительных средств и систем различного функционального назначения	ПК-2$_{ИД-2-1}$ Проводит классификацию и осуществляет выбор современных инструментальных средств разработки прикладного программного обеспечения вычислительных средств и систем различного функционального назначения	Результаты демонстрации знаний о грамотно сформулированной цели проекта и задач. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме
	ПК-2$_{ИД-2-2}$ Демонстрирует знания технологий программирования, разработки прикладного программного обеспечения вычислительных средств и систем различного функционального назначения	Результаты демонстрации знаний технологий разработки интерфейсов. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме
	ПК-2$_{ИД-2-3}$ Использует современные инструментальные средства разработки прикладного программного обеспечения вычислительных средств и систем различного функционального назначения	Результаты применения инструментов разработки интерфейсов. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме
	ПК-2$_{ИД-2-4}$ Применяет технологии программирования при разработке прикладного программного обеспечения вычислительных средств и систем различного функционального назначения	Результаты применения технологий программирования. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме
ПК-4 способен применять и осваивать новые средства автоматизированного проектирования, разработки, тестирования и сопровождения программного обеспечения	ПК–4ИД-4.1 Демонстрирует знания инструментальных средств для разработки приложений	Результаты демонстрации знаний инструментов проектирования.
	ПК–4ИД-4.2 Осуществляет выбор средств разработки, тестирования и сопровождения программного обеспечения	Результаты выбора средств разработки. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме

	ПК-4ИД-4.3 Применяет инструментальные средства, новые средства автоматизированного проектирования и разработки программного обеспечения,	Результаты разработки интерфейсов. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме
	ПК-4ИД-4.4 Осваивает новые средства автоматизированного проектирования и разработки программного обеспечения	Результаты освоения новых средств. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме
	ПК-$4_{ИД-4.5}$ Осуществляет сопровождение программного обеспечения	Результаты сопровождения решений. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме
	ПК-$4_{ИД-4.6}$ Разрабатывает и использует методики тестирования	Результаты тестирования интерфейсов. Представленный документ не содержит ошибок. Решены все частные задачи в полном объеме

Лабораторная работа № 1.
Введение во фронтенд-разработку. Архитектура веб-приложений и настройка среды разработки

Цель: сформировать понимание архитектуры веб-приложений, изучить роль фронтенд-разработки в программной системе, освоить базовые инструменты разработчика

Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

Фронтенд-разработка — это область программной инженерии, связанная с созданием пользовательского интерфейса веб-приложений.

Фронтенд отвечает за:

1. визуальное представление данных
2. взаимодействие пользователя с системой
3. обработку пользовательских действий

Фронтенд является частью более общей архитектуры веб-приложения, которая включает:

1. клиентскую часть (frontend)
2. серверную часть (backend)
3. базу данных

Архитектура веб-приложений

Современные веб-приложения строятся на основе **клиент–серверной архитектуры**.

Основные компоненты:

1. **Клиент** — браузер пользователя
2. **Сервер** — обрабатывает запросы
3. **Сеть (HTTP/HTTPS)** — канал взаимодействия

Принцип работы:

1. Пользователь вводит URL
2. Браузер отправляет HTTP-запрос

3. Сервер обрабатывает запрос
4. Возвращается HTML/CSS/JS
5. Браузер отображает страницу

3. Роль браузера

Браузер — это среда выполнения фронтенд-кода.

Он выполняет следующие функции:

1. парсинг HTML → построение DOM
2. применение CSS → формирование визуального представления
3. выполнение JavaScript
4. обработка событий пользователя

Важные подсистемы браузера:

1. Rendering Engine
2. JavaScript Engine
3. Networking

4. Технологии фронтенд-разработки

HTML (HyperText Markup Language)

Определяет структуру документа:

1. заголовки
2. абзацы
3. списки
4. формы

CSS (Cascading Style Sheets)

Отвечает за внешний вид:

1. цвета
2. шрифты
3. расположение элементов

JavaScript

Отвечает за поведение:

1. обработка событий
2. динамическое изменение DOM
3. работа с сервером

DevTools (инструменты разработчика)

Позволяют:

1. анализировать DOM
2. смотреть CSS
3. выполнять JS
4. отслеживать ошибки

Система контроля версий (Git)

Позволяет:

1. хранить историю изменений
2. работать в команде
3. управлять версиями проекта

6. Рабочее окружение разработчика

Минимальный набор:

1. редактор кода
2. браузер
3. файловая система
4. Рекомендуемый набор:
5. VS Code + расширения
6. Chrome DevTools
7. Git

Методика и порядок выполнения работы

Задание 1.

Установить:

Visual Studio Code

Установить расширения:

Live Server

Prettier

Открыть DevTools и изучить вкладки:

- Elements
- Console

- Network

Создание проекта

1. **Создать папку проекта**
2. **Создать структуру:**

```
project/
├── index.html
├── css/
│   └── style.css
└── js/
    └── script.js
```

Часть 4. Создание HTML-документа

Создать страницу, содержащую:

1. заголовок
2. 2–3 абзаца текста
3. список технологий
4. изображение
5. ссылку

Часть 5. Подключение CSS

1. Создать файл стилей
2. Подключить его к HTML
3. Реализовать:
    - изменение цвета
    - изменение шрифта
    - оформление заголовков

Часть 6. Подключение JavaScript

1. Создать файл script.js
2. Подключить его
3. Реализовать:
    - вывод сообщения в консоль
    - alert при загрузке страницы

Часть 7. Работа с DevTools

1. Найти элемент на странице через Elements
2. Изменить стиль вручную
3. В Console выполнить JavaScript-код
4. Найти сетевой запрос

Часть 8. Работа с Git (опционально, повышенный уровень)

1. Инициализировать репозиторий
2. Сделать первый коммит
3. Добавить README.md

Содержание отчета и его форма

Представить отчёт в формате .doc о предпроектном обследовании фирмы/организации (по индивидуальному варианту) для разработки информационной системы (подсистемы).

Вопросы для защиты работы

1. Что такое фронтенд-разработка?
2. Из каких частей состоит веб-приложение?
3. Как работает клиент–серверная архитектура?
4. Что делает браузер при загрузке страницы?
5. В чем отличие HTML, CSS и JavaScript?
6. Что такое DOM?
7. Для чего используются DevTools?
8. Что такое Git и зачем он нужен?

Лабораторная работа № 2.

HTML5: структура документа, семантика, формы, таблицы, мультимедиа

Цель: изучить структуру HTML-документа, освоить основные теги HTML5, научиться использовать семантическую разметку, получить навыки работы с формами, таблицами и мультимедиа

Формируемые компетенции: ПК-2, ПК-4

Основные теоретические сведения

Назначение языка HTML

HTML (HyperText Markup Language) — это язык разметки, используемый для создания структуры веб-страниц.

HTML:

описывает структуру документа
не отвечает за внешний вид (это CSS)
не отвечает за поведение (это JavaScript)

2. Структура HTML-документа

Каждый HTML-документ имеет базовую структуру:

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="UTF-8">
   <title>Название страницы</title>
</head>
<body>
   Содержимое страницы
</body>
</html>
```

Основные части:

<!DOCTYPE html> — объявление типа документа
<html> — корневой элемент
<head> — служебная информация
<body> — содержимое страницы

3. Основные теги HTML

Текстовые элементы:

<h1>–<h6> — заголовки
<p> — абзац
, — выделение

Списки:

 — маркированный список
 — нумерованный

 — элемент списка

Ссылки:

<a href=""> — гиперссылка

Изображения:

<img src="" alt="">

4. Семантические теги HTML5

HTML5 ввёл семантические элементы:

<header> — шапка
<nav> — навигация
<main> — основной контент
<section> — раздел
<article> — статья
<footer> — подвал

Значение семантики:

улучшает SEO
повышает доступность
упрощает поддержку кода

5. Таблицы

Таблицы используются для представления данных:

<table> — таблица
<tr> — строка
<td> — ячейка
<th> — заголовок

6. Формы

Формы используются для ввода данных пользователем.
Основные элементы:

<form>
<input>
<label>
<textarea>
<select>

Типы input:

text
email
password
checkbox
radio

7. Валидация форм

HTML5 поддерживает встроенную валидацию:

required
type="email"
minlength
pattern

8. Мультимедиа

HTML5 поддерживает:

<audio>
<video>

Позволяет встраивать медиа без сторонних плагинов

Методика и порядок выполнения работы
Создание базового HTML-документа

1. Создать файл index.html
2. Реализовать:
    - корректную структуру документа
    - заголовок страницы
    - мета-теги

Часть 2. Работа с текстом

1. Добавить:
   заголовки всех уровней
   абзацы
   выделение текста
2. Описать:
   что такое HTML
   роль HTML в веб-разработке

Часть 3. Списки и ссылки

1. Создать:
    - маркированный список технологий
    - нумерованный список шагов разработки
2. Добавить:
    - минимум 3 ссылки (внешние и внутренние)

Часть 4. Изображения и мультимедиа

1. Добавить изображение:
    - с атрибутом alt
2. Добавить:
    - видео или аудио

Часть 5. Таблицы

1. Создать таблицу:
    - 3–4 строки
    - заголовок таблицы
2. Пример содержания:
    - технологии
    - описание
    - уровень сложности

Часть 6. Семантическая верстка

1. Использовать:
    - header
    - nav
    - main
    - section
    - footer
2. Разделить страницу на логические блоки
3. **Часть 7. Формы**

1. Создать форму регистрации:
    - имя

- email
- пароль
- выбор пола (radio)
- чекбокс согласия

2. Добавить:
   - label
   - placeholder

Часть 8. Валидация

1. Добавить:
   - required
   - email validation
   - minlength

Содержание отчета и его форма

Представить отчёт в формате .doc

Вопросы для защиты работы

1. Что такое HTML?
2. Из каких частей состоит HTML-документ?
3. В чем отличие и <section>?
4. Что такое семантическая разметка?
5. Какие теги используются для таблиц?
6. Какие элементы используются в формах?
7. Что такое валидация?
8. Зачем нужен атрибут alt?

Лабораторная работа № 3.

CSS3: селекторы, каскад, блочная модель, Flexbox, Grid, анимации

Цель и содержание работы: изучить принципы применения CSS; освоить селекторы и каскадирование; понять блочную модель CSS; научиться позиционировать элементы; освоить Flexbox и Grid.

Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

CSS (Cascading Style Sheets) — язык описания внешнего вида HTML-документа.

CSS отвечает за:

оформление

расположение элементов

адаптивность

анимации

Разделение HTML и CSS обеспечивает:

удобство поддержки

повторное использование стилей

масштабируемость

2. Способы подключения CSS

Существует три способа:

Встроенный (inline)

<p style="color: red;">

Внутренний (internal)

<style>

p { color: red; }

</style>

Внешний (external) — предпочтительный

<link rel="stylesheet" href="style.css">

3. Селекторы CSS

Селекторы определяют, к каким элементам применяется стиль.

Основные виды:

по тегу: p

по классу: .class

по id: #id

вложенные: div p

групповые: h1, h2

4. Каскад и специфичность

CSS применяет стили по правилам:

- важность (important)
- специфичность
- порядок в коде

Специфичность:

id > class > тег

5. Блочная модель

Каждый элемент имеет структуру:

- content
- padding
- border
- margin

Это определяет:

размер элемента

расстояние между элементами

6. Позиционирование

Свойства:

- static
- relative

- absolute
- fixed

Позволяют управлять расположением элементов.

7. Flexbox

Используется для построения гибких макетов.

Основные свойства:

- display: flex
- justify-content
- align-items
- flex-direction

8. Grid Layout

Позволяет строить сложные сетки.

Основные свойства:

- display: grid
- grid-template-columns
- grid-template-rows
- gap

9. Псевдоклассы и псевдоэлементы

Примеры:

:hover

:focus

10. Анимации и переходы

Transitions:

- плавное изменение свойств
- Transform:
- rotate, scale
- Animation:
- keyframes

Методика выполнения лабораторной работы

Часть 1. Подключение CSS

Создать файл style.css

Подключить его к HTML

Часть 2. Селекторы

Применить:

1. селектор по тегу
2. по классу
3. по id

Создать:

минимум 3 класса

Часть 3. Оформление страницы

Настроить:

1. цвет текста
2. фон
3. шрифты
4. Добавить:
5. границы
6. отступы

Часть 4. Блочная модель

Изменить:

1. padding
2. margin
3. border

Проанализировать через DevTools

Часть 5. Позиционирование

Использовать:

1. relative
2. absolute

Разместить элементы на странице

Часть 6. Flexbox

Создать контейнер

Разместить 3–5 элементов

Настроить:

выравнивание

направление

Часть 7. Grid

Создать сетку

Разместить элементы

Использовать:

2–3 колонки

Часть 8. Псевдоклассы

Добавить:

hover эффект

активное состояние

Часть 9. Анимации

Реализовать:

- transition
- transform
- animation

Контрольные вопросы

1. Что такое CSS?
2. Какие есть способы подключения CSS?
3. Что такое специфичность?
4. Что входит в блочную модель?
5. В чем разница Flexbox и Grid?
6. Что делает :hover?
7. Что такое animation?

Лабораторная работа 4.

Основы CSS: селекторы, каскад, блочная модель

Цель: Изучить способы подключения CSS, освоить селекторы, механизм каскада и блочную модель элементов.

Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

CSS (Cascading Style Sheets) управляет визуальным представлением HTML. Стили подключаются через <link> (внешний), <style> (внутренний) или атрибут style (inline). Селекторы определяют, к каким элементам применяются правила: по тегу, классу (.class), ID (#id), комбинированные (, >, +, ~, []). Каскад определяет приоритет применения правил на основе специфичности (inline > ID > класс/атрибут > тег) и порядка объявления. Блочная модель включает content, padding, border, margin. Свойство box-sizing: border-box включает отступы и границы в ширину/высоту элемента, упрощая верстку.

1. Назначение и архитектура CSS

CSS (Cascading Style Sheets) — язык описания внешнего вида документа, написанного на языке разметки (HTML, XML, SVG). Его главная архитектурная цель — разделение ответственности: HTML отвечает за структуру и семантику, CSS — за визуальное представление, JavaScript — за интерактивность. Такой подход обеспечивает:

Поддерживаемость: изменение дизайна в одном файле влияет на весь проект.

Производительность: браузер кэширует внешние таблицы стилей, сокращая время загрузки.

Адаптивность и доступность: стили можно переопределять под разные устройства, режимы контрастности и вспомогательные технологии (скринридеры).

Методика и порядок выполнения лабораторной работы:

1. Создайте styles.css и подключите его к index.html через <link>.
2. Напишите стили для базовых элементов, используя селекторы по тегу, классу и ID.
3. Настройте margin, padding, border, background, color, font-family.
4. Примените box-sizing: border-box глобально (* { box-sizing: border-box; }).
5. Продемонстрируйте работу каскада: создайте конфликтующие правила и проверьте приоритеты через DevTools.
6. Сохраните скриншоты DevTools с рассчитанной специфичностью.

Контрольные вопросы

1. то такое CSS-каскад и как он влияет на применение стилей?
2. Как рассчитывается специфичность селектора?
3. В чем практическая польза box-sizing: border-box?
4. Чем отличаются margin и padding визуально и логически?
5. Как переопределить внешние стили библиотеки без !important?

Лабораторная работа № 5.

Современные методы компоновки: Flexbox и CSS Grid

Цель работы: Освоить принципы работы Flexbox и CSS Grid для создания гибких и адаптивных макетов.

Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

Эволюция подходов к созданию веб-макетов

На ранних этапах развития веба верстка опиралась на табличные структуры, плавающие элементы и ручное управление отступами. Эти методы требовали сложных обходных решений, часто ломались при изменении контента или размера экрана и затрудняли поддержку проектов. Появление специализированных модулей компоновки стало ответом на потребность в нативных, предсказуемых и декларативных инструментах управления пространством. Flexbox и CSS Grid были разработаны консорциумом W3C как взаимодополняющие стандарты, которые заменили хаки на строгую архитектуру. Их внедрение позволило разработчикам описывать макеты в терминах логики и отношений элементов, а не в терминах абсолютных координат и фиксированных размеров.

2. Flexbox: философия одномерного выравнивания

Flexbox (Flexible Box Layout) спроектирован для работы с контентом в одном направлении — по горизонтали или по вертикали. Его основная задача заключается в динамическом распределении свободного пространства между дочерними элементами и управлении их выравниванием относительно осей контейнера. При активации этого режима родительский элемент берёт на себя роль координатора: он анализирует доступное пространство, измеряет внутренние элементы и автоматически растягивает, сжимает или переносит их согласно заданным правилам.

Ключевые концепции Flexbox включают две перпендикулярные оси — главную и поперечную, направление которых можно менять в зависимости от задачи. Элементы внутри контейнера могут обладать разной степенью гибкости:

одни занимают фиксированное место, другие расширяются, заполняя свободное пространство, третьи сжимаются при нехватке места. Отдельно настраивается выравнивание по главной оси (распределение элементов вдоль строки или колонки), по поперечной оси (выравнивание по высоте или ширине) и поведение при переполнении (автоматический переход на следующую линию). Flexbox идеально подходит для компонентов интерфейса: навигационных меню, групп кнопок, карточек в ряд, центрирования содержимого и распределения элементов по краям контейнера.

3. CSS Grid: философия двумерного макетирования

CSS Grid Layout решает задачу создания сложных сеток, где элементы управляются одновременно по двум измерениям. В отличие от одномерного подхода, Grid работает на уровне структуры страницы или крупных секций, позволяя заранее задать каркас из строк и столбцов и размещать блоки в произвольных позициях этого каркаса. Контейнер делится на невидимую сетку, размеры линий которой могут быть фиксированными, адаптивными или рассчитанными автоматически на основе доступного пространства.

Элементы сетки могут занимать несколько ячеек, перекрываться, менять порядок отображения без изменения HTML-разметки и выравниваться внутри собственных ячеек. Важной особенностью является поддержка именованных областей, что позволяет описывать макет в декларативном виде, делая структуру интуитивно понятной и легко модифицируемой. Grid также предоставляет механизмы автоматического создания дополнительных строк или колонок при добавлении нового контента, что делает сетку устойчивой к изменениям. Данная технология незаменима при проектировании главных страниц, административных панелей, галерей, сложных форм и любых интерфейсов, требующих точного контроля над двумерным пространством.

4. Сравнительный анализ и стратегия комбинирования

Хотя обе технологии решают задачи компоновки, их архитектурные различия определяют области применения. Flexbox оперирует потоком контента в одном измерении, автоматически подстраиваясь под его размер и количество.

Grid оперирует заранее заданной структурой, в которую помещается контент, сохраняя заданную геометрию макета.

На практике эти методы не конкурируют, а образуют единый инструментарий. Современная стратегия предполагает использование Grid для глобального каркаса страницы (определение зон шапки, боковой панели, основной области и подвала) и Flexbox для внутренних компонентов (меню, списки, карточки, формы, группы кнопок). Такое разделение ответственности обеспечивает максимальную предсказуемость, упрощает рефакторинг, снижает количество вложенных контейнеров и соответствует принципам модульной вёрстки.

5. Механизмы адаптивности без жёстких контрольных точек

Современная компоновка минимизирует зависимость от множества фиксированных разрешений экрана. Flexbox позволяет элементам автоматически переноситься на новую строку при нехватке места, плавно перестраивая макет под доступную ширину. CSS Grid предоставляет функции автоматического заполнения и адаптивного расчёта размеров, благодаря которым сетка самостоятельно определяет оптимальное количество колонок или строк в зависимости от контейнера. Комбинация этих механизмов с относительными единицами измерения позволяет создавать «резиновые» интерфейсы, которые корректно работают на всём спектре устройств — от мобильных экранов до широкоформатных мониторов — без написания десятков условий для разных брейкпоинтов.

6. Управление отступами и предотвращение конфликтов вёрстки

Одной из распространённых проблем традиционной вёрстки было управление промежутками между элементами. Внешние отступы часто приводили к непредсказуемому пождению, особенно при использовании схлопывания или динамическом добавлении контента. Современные методы компоновки вводят единое свойство для задания промежутков между элементами внутри контейнера. Этот механизм применяется как во Flexbox, так и в Grid, создавая равномерные отступы без влияния на внешние границы блока. Использование встроенных промежутков исключает необходимость ручного

расчёта отрицательных отступов, упрощает адаптивность и делает макет визуально стабильным при изменении количества элементов.

7. Отладка, визуализация и инструменты разработчика

Понимание работы современных сеток напрямую связано с умением анализировать их в реальном времени. Браузерные панели разработчика предоставляют специализированные визуализаторы для Flexbox и Grid. При выборе элемента-контейнера браузер подсвечивает его границы, отображает оси, линии сетки, промежутки между элементами и размеры ячеек. Визуальные индикаторы помогают мгновенно оценить, как распределено пространство, почему элемент оказался не на своём месте или как изменилась структура при изменении ширины окна. Использование этих инструментов превращает процесс вёрстки из подбора значений в осознанное проектирование интерфейса с мгновенной визуальной обратной связью.

8. Практические рекомендации и распространённые ошибки

Эффективное использование современных методов компоновки требует соблюдения нескольких архитектурных принципов. Во-первых, важно чётко разделять ответственность между контейнером и элементами: свойства макета применяются к родителю, а параметры поведения и выравнивания — к потомкам. Во-вторых, следует избегать избыточной вложенности сеток, которая усложняет отладку и снижает производительность рендеринга. В-третьих, необходимо учитывать особенности переполнения контента: гибкие элементы могут сжиматься до минимальных значений, поэтому важно задавать разумные ограничения или разрешать перенос содержимого. Наконец, современная практика рекомендует проектировать макет с учётом изменения количества элементов: сетка должна оставаться работоспособной при добавлении, удалении или изменении длины контента без необходимости ручного пересчёта размеров.

Методика и порядок выполнения лабораторной работы:

Создайте layout.html и layout.css.

Реализуйте навигационную панель с display: flex, выравниванием элементов по краям и адаптивным переносом.

Создайте галерею карточек с display: grid, используя repeat(auto-fit, minmax(250px, 1fr)) и gap.

Объедините Flexbox и Grid в одном макете (например, шапка + основная сетка + футер).

Протестируйте поведение при изменении ширины окна через DevTools.

Приложите скриншоты макета в трех контрольных разрешениях.

Контрольные вопросы:

1. В каких случаях предпочтительнее использовать Flexbox, а в каких — Grid?
2. Что делает свойство flex-wrap и как оно влияет на макет?
3. Как создать полностью отзывчивую сетку без медиа-запросов?
4. Для чего используется grid-template-areas?
5. Как выровнять элемент идеально по центру с помощью Flexbox?

Лабораторная работа № 6

Основы JavaScript: переменные, типы данных, функции и управление потоком

Цель : Изучить базовый синтаксис JavaScript, работу с переменными, типами данных, функциями и условными конструкциями.

Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

1. Архитектурная роль и особенности языка JavaScript

JavaScript — мультипарадигменный язык программирования, изначально разработанный для добавления интерактивности веб-страницам. В современной экосистеме он выполняется не только в браузерах, но и на серверных платформах, в мобильных и десктопных приложениях, а также в микроконтроллерах. Ключевая архитектурная особенность языка заключается в однопоточности с асинхронной моделью обработки событий: основной поток выполнения не блокируется длительными операциями, что обеспечивает плавную работу пользовательского интерфейса. JavaScript поддерживает императивный, объектно-ориентированный и функциональный стили, позволяя разработчику выбирать наиболее подходящий подход для каждой задачи. Динамическая природа языка обеспечивает высокую гибкость, но требует осознанного управления состоянием и структурой данных для предотвращения трудноуловимых ошибок.

2. Система объявления переменных и управление областью видимости

Переменная в JavaScript представляет собой именованную ссылку на значение, хранящееся в памяти. Исторически язык использовал единый механизм объявления с функциональной областью видимости и механизмом всплытия, что часто приводило к непредсказуемому поведению и конфликтам имён в крупных проектах. Современные стандарты ввели два новых механизма, ставших отраслевой нормой. Первый предназначен для создания изменяемых значений с строгой блочной областью видимости: переменная существует

только внутри ограничивающих скобок и недоступна за их пределами, что исключает случайное переопределение. Второй механизм создаёт неизменяемые ссылки: после первоначального присваивания значение не может быть заменено, что гарантирует предсказуемость состояния и защищает данные от нежелательных изменений. Использование современных механизсов объявления является обязательной практикой, так как оно упрощает чтение кода, предотвращает типичные ошибки управления памятью и соответствует принципам безопасной разработки.

3. Динамическая типизация и классификация типов данных

JavaScript использует динамическую типизацию, при которой тип переменной не указывается явно при создании, а определяется автоматически в момент присваивания значения и может изменяться в процессе выполнения программы. Все типы данных делятся на две фундаментальные категории: примитивные и объектные. Примитивные типы хранят непосредственное значение, передаются по значению при копировании или передаче в функции и являются неизменяемыми на уровне содержимого. К ним относятся текстовые последовательности, числовые значения, логические состояния, специальные маркеры отсутствия данных и уникальные идентификаторы. Объектные типы представляют собой структурированные коллекции данных и методов, передаются по ссылке и обладают изменяемым внутренним состоянием. Понимание различий между этими категориями критически важно для корректного сравнения значений, управления памятью, предсказуемой передачи данных между модулями и избежания скрытых ошибок, связанных с изменением общих структур.

4. Механизмы управления потоком выполнения

Управление потоком определяет последовательность, в которой интерпретатор обрабатывает инструкции программы. Ветвление позволяет выбирать один из нескольких путей выполнения в зависимости от логических условий, что является основой для обработки пользовательского ввода, валидации данных и построения адаптивной логики интерфейсов. Циклические конструкции обеспечивают многократное повторение блока инструкций до достижения заданного условия или завершения перебора коллекции данных.

Современные подходы рекомендуют использовать конструкции, ориентированные на прямой перебор элементов, а не на ручное управление счётчиками, так как они делают код более читаемым, снижают риск бесконечных циклов и естественным образом работают с любыми перечислимыми структурами. Грамотное комбинирование условий и циклов позволяет описывать сложную бизнес-логику, сохраняя при этом структурную простоту, лёгкость тестирования и удобство сопровождения.

5. Функции как фундаментальные строительные блоки

Функции в JavaScript являются объектами первого класса, что означает возможность их передачи в качестве аргументов, возврата из других функций, присваивания переменным и сохранения в коллекциях. Они служат основными модулями для инкапсуляции логики, повторного использования кода и организации архитектуры приложения. Существует несколько способов объявления функций, каждый из которых обладает собственными особенностями поведения в отношении момента доступности, привязки контекста и возврата результатов. Традиционные объявления подходят для создания именованных блоков логики, которые можно вызывать до их фактического расположения в тексте программы. Функциональные выражения позволяют создавать анонимные блоки, привязанные к переменным, что удобно для передачи в качестве параметров или немедленного выполнения. Современный сокращённый синтаксис обеспечивает компактную запись, автоматическую привязку к месту определения и упрощённый возврат результатов в однострочных вариантах. Выбор конкретного способа зависит от задачи, требований к читаемости, необходимости явного указания имени в стеке вызовов и архитектурных ограничений проекта.

6. Область видимости, контекст выполнения и механизмы изоляции

Поведение переменных и функций напрямую зависит от области видимости, которая определяет, где и как долго данные остаются доступными. Блочная область видимости ограничивает доступ к переменным пределами логических блоков, создавая изолированные пространства, что предотвращает конфликты имён и утечку временного состояния во внешнюю среду. Контекст выполнения определяет, к какому объекту привязано текущее выполнение

функции, и динамически меняется в зависимости от способа вызова. Замыкания возникают, когда функция запоминает окружение, в котором была создана, и продолжает иметь доступ к переменным этого окружения даже после завершения внешней функции. Это мощный механизм для создания приватных данных, фабричных генераторов и поддержания состояния без использования глобальных переменных. Понимание этих механизмов позволяет проектировать устойчивые архитектурные решения и избегать типичных ошибок, связанных с потерей данных или непредсказуемым изменением состояния.

7. Современные стандарты и рекомендации по написанию кода

Разработка на JavaScript регулируется единой спецификацией, которая регулярно обновляется, добавляя новые возможности, улучшая безопасность и устраняя исторические неоднозначности. Современная практика строго рекомендует активировать строгий режим выполнения для выявления потенциальных ошибок на этапе написания кода, использовать неизменяемые ссылки там, где изменение данных не требуется, и избегать неявного преобразования типов при сравнении значений. Важным аспектом является соблюдение единообразия в именовании, структуре файлов, разделении логики и обработке исключений, что обеспечивается стандартизированными руководствами по стилю и автоматизированными проверками. Чистый, предсказуемый и хорошо документированный код снижает затраты на поддержку, упрощает командную работу, повышает надёжность приложений и соответствует требованиям промышленной разработки.

8. Инструменты отладки и анализа выполнения

Понимание работы переменных, типов данных и потоков выполнения невозможно без навыков практической отладки. Встроенные инструменты браузера позволяют пошагово выполнять код, отслеживать значения переменных в реальном времени, анализировать стек вызовов и выявлять точные места возникновения ошибок. Консоль предоставляет механизмы вывода диагностической информации, группировки сообщений и профилирования производительности. Визуализация области видимости в отладчике помогает мгновенно оценить, какие данные доступны в текущий момент, почему значение не изменилось или как функция взаимодействует с

внешним окружением. Регулярное использование этих инструментов формирует инженерное мышление, позволяя разработчику не угадывать поведение программы, а точно диагностировать причины некорректной работы и строить устойчивые алгоритмы.

Методика и порядок выполнения лабораторной работы:

1. Создайте script.js и подключите к HTML через <script defer src="script.js">.
2. Объявите переменные let и const, выполните арифметические и строковые операции.
3. Напишите функцию расчета площади фигуры с проверкой входных данных.
4. Реализуйте циклический перебор массива чисел с фильтрацией четных/нечетных.
5. Выведите результаты в консоль и отобразите в DOM-элементе.
6. Прокомментируйте код, объяснив выбор конструкций.

Контрольные вопросы:

1. Чем let отличается от var и const?
2. Какие примитивные типы данных существуют в JavaScript?
3. Что такое всплытие переменных (hoisting)?
4. Как работают стрелочные функции и в чем их ограничения?
5. Зачем использовать defer при подключении скриптов?

Лабораторная работа № 7

Работа с DOM: манипуляции элементами и обработка событий

Цель: Освоить методы доступа к DOM-элементам, динамическое изменение содержимого и обработку пользовательских событий.
Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

1. Архитектурная роль JavaScript в веб-среде

JavaScript является основным языком сценариев для браузерной платформы, обеспечивающим интерактивность, динамику и бизнес-логику пользовательских интерфейсов. В отличие от языков разметки и стилей, которые описывают структуру и визуальное представление документа, JavaScript отвечает за поведение элементов в реальном времени, обработку пользовательских действий и взаимодействие с внешними ресурсами. Он выполняется в изолированной песочнице браузера, поддерживает динамическую типизацию, событийно-ориентированную модель и однопоточную архитектуру с асинхронным выполнением задач. Понимание фундаментальных принципов языка необходимо для создания отзывчивых, безопасных и поддерживаемых веб-приложений.

2. Переменные и механизмы управления областью видимости

Переменная представляет собой именованную ссылку на значение, хранящееся в памяти. Современные стандарты языка предоставляют два основных механизма объявления с блочной областью видимости: один предназначен для изменяемых данных, другой — для неизменяемых ссылок, значение которых не может быть переназначено после инициализации. Исторический механизм, ограничивающий переменные областью видимости функции, сохранился исключительно для обратной совместимости с унаследованными проектами. Блочная область видимости гарантирует, что данные существуют только внутри ограничивающих структур, что предотвращает случайные конфликты имён, исключает нежелательное переопределение состояния и делает поток выполнения предсказуемым.

Осознанный выбор механизма объявления напрямую влияет на безопасность данных, читаемость архитектуры и устойчивость к ошибкам.

3. Динамическая типизация и классификация данных

Язык не требует явного указания типа данных при создании переменной. Тип определяется автоматически на основе присвоенного значения и может изменяться в процессе выполнения программы. Все данные делятся на две фундаментальные категории. Примитивные типы хранят непосредственное значение, передаются по значению при копировании или передаче в функции и являются неизменяемыми на уровне содержимого. К ним относятся текстовые последовательности, числовые значения, логические состояния, маркеры отсутствия данных и уникальные идентификаторы. Объектные типы представляют собой структурированные коллекции данных и методов, передаются по ссылке и позволяют динамически изменять своё внутреннее состояние. Понимание различий между передачей по значению и по ссылке критически важно для корректного копирования данных, предотвращения непреднамеренных изменений общих структур и построения устойчивой логики.

4. Управление потоком выполнения программы

Линейное выполнение инструкций редко соответствует реальной логике приложений. Для реализации ветвлений используются условные конструкции, позволяющие выбирать путь выполнения на основе проверки логических выражений. Они обеспечивают обработку различных сценариев, валидацию пользовательского ввода и реакцию на изменения состояния интерфейса. Для повторяющихся операций применяются циклические конструкции. Современные подходы отдают предпочтение конструкциям, ориентированным на прямой перебор элементов коллекций, поскольку они снижают риск ошибок индексации, делают код более декларативным и естественным образом работают с различными структурами данных. Грамотное сочетание условий и циклов позволяет описывать сложные алгоритмы, сохраняя структурную ясность, лёгкость тестирования и удобство сопровождения.

5. Функции как модули логики

Функции являются основными строительными блоками для организации кода. Они инкапсулируют набор инструкций, принимают входные параметры и возвращают результат вычислений. В языке функции обладают статусом объектов первого класса: их можно передавать в качестве аргументов, возвращать из других функций, сохранять в переменных и динамически создавать во время выполнения. Существует несколько синтаксических способов объявления, каждый из которых определяет момент доступности, привязку контекста и особенности работы с окружением. Традиционные объявления обеспечивают подъём определения, выражения позволяют создавать гибкие привязки, а современный лаконичный синтаксис автоматически наследует контекст окружающего кода и упрощает запись коротких преобразований. Выбор способа зависит от архитектурных требований, сценария использования и необходимости явного указания имени в стеке вызовов.

6. Замыкания и управление состоянием

Важным следствием механизма функций является способность сохранять доступ к переменным внешнего окружения даже после завершения выполнения родительского блока. Это явление позволяет создавать приватные данные, фабричные генераторы и поддерживать состояние без обращения к глобальной области видимости. Правильное использование этого механизма обеспечивает изоляцию логики, предотвращает загрязнение глобального пространства имён и способствует созданию переиспользуемых, самодостаточных компонентов.

7. Современные стандарты и рекомендации по написанию кода

Разработка регулируется единой спецификацией, которая регулярно расширяется новыми возможностями, улучшающими безопасность и выразительность. Современная практика предписывает использовать строгий режим выполнения для выявления скрытых ошибок на этапе написания, избегать неявного преобразования типов при сравнении значений, применять неизменяемые ссылки там, где изменение данных не требуется, и соблюдать единообразие в структуре, именовании и разделении ответственности. Эти

принципы снижают когнитивную нагрузку при чтении кода, упрощают командную разработку, ускоряют рефакторинг и повышают надёжность конечного продукта.

8. Инструменты анализа и отладки

Понимание работы переменных, типов данных и потока выполнения невозможно без практической отладки. Встроенные среды разработчика позволяют пошагово выполнять инструкции, отслеживать значения переменных в реальном времени, анализировать стек вызовов и выявлять точные места возникновения ошибок. Визуализация области видимости помогает мгновенно оценить доступность данных, причины некорректного поведения и влияние замыканий на текущий контекст. Регулярное использование этих инструментов формирует инженерное мышление, позволяя переходить от интуитивного исправления ошибок к системной диагностике, прогнозированию поведения программы и проектированию устойчивых алгоритмов.

Методика и порядок выполнения лабораторной работы:

1. Создайте script.js и подключите к HTML через <script defer src="script.js">.
2. Объявите переменные let и const, выполните арифметические и строковые операции.
3. Напишите функцию расчета площади фигуры с проверкой входных данных.
4. Реализуйте циклический перебор массива чисел с фильтрацией четных/нечетных.
5. Выведите результаты в консоль и отобразите в DOM-элементе.
6. Прокомментируйте код, объяснив выбор конструкций.

Контрольные вопросы:

1. Чем let отличается от var и const?

2. Какие примитивные типы данных существуют в JavaScript?
3. Что такое всплытие переменных (hoisting)?
4. Как работают стрелочные функции и в чем их ограничения?
5. Зачем использовать defer при подключении скриптов?

Лабораторная работа № 8

Асинхронный JavaScript: Fetch API, Promises и async/await

Цель: изучить возможности программы ArgoUML и ее установку. Изучить асинхронные операции, работу с сетью через Fetch API, обработку успешных и ошибочных ответов.

Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

1. Архитектурные основы асинхронности в JavaScript

JavaScript выполняется в однопоточной среде, что означает, что интерпретатор в каждый момент времени обрабатывает только одну инструкцию. Однако современные веб-приложения постоянно взаимодействуют с внешними ресурсами, обрабатывают таймеры, реагируют на пользовательский ввод и загружают данные из сети. Чтобы эти длительные операции не блокировали пользовательский интерфейс и не приводили к зависанию страницы, язык использует событийно-ориентированную, неблокирующую архитектуру. Долговременные задачи делегируются внутренним механизмам браузера или серверной среде, а их результаты возвращаются в основной поток только после завершения. Координация этого процесса осуществляется через цикл обработки событий, который последовательно извлекает задачи из очереди и передаёт их на выполнение. Такой подход позволяет приложению оставаться отзывчивым даже при выполнении ресурсоёмких или сетевых операций.

2. Механизм Promises: состояния, цепочки и обработка результатов

Для структурирования асинхронных операций в язык был внедрён механизм обещаний. Объект-обещание представляет собой контейнер для результата, который может стать доступным в будущем или никогда не будет получен. Он проходит через три строго определённых состояния: ожидание, успешное завершение и отклонение. После перехода в одно из финальных состояний значение обещания становится неизменяемым, что

гарантирует предсказуемость поведения. Ключевое преимущество механизма заключается в возможности выстраивать последовательные цепочки обработки, где каждая следующая операция запускается только после завершения предыдущей. Ошибки автоматически распространяются по цепочке до ближайшего обработчика, что избавляет разработчика от необходимости проверять статус на каждом этапе и устраняет проблему глубокой вложенности обратных вызовов.

3. Синтаксис async/await: упрощение асинхронного кода

Несмотря на эффективность обещаний, их цепочечная запись может становиться трудной для восприятия при усложнении бизнес-логики. Современный синтаксис предоставляет декларативный способ работы с асинхронными операциями, делая код визуально похожим на последовательные синхронные инструкции. Функция, помеченная специальным маркером, всегда возвращает обещание, а оператор ожидания приостанавливает её выполнение до получения результата, не блокируя при этом основной поток выполнения приложения. Это позволяет использовать стандартные конструкции ветвления и циклов для управления асинхронным потоком, упрощает чтение алгоритмов и снижает когнитивную нагрузку при разработке. Ошибки обрабатываются через привычные блоки перехвата исключений, что унифицирует подход к синхронной и асинхронной логике и делает код более устойчивым к изменениям.

4. Fetch API: современный стандарт сетевых запросов

Для взаимодействия с серверами и внешними сервисами в браузере реализуется программный интерфейс запросов. Он пришёл на смену устаревшим механизмам и стал единым стандартом для выполнения HTTP-запросов на основе обещаний. Интерфейс оперирует моделями запроса и ответа, позволяя гибко настраивать метод обращения, передавать заголовки, управлять аутентификационными данными и обрабатывать различные форматы полезной нагрузки. Поддерживается работа с текстовыми данными, структурированными объектами, бинарными файлами и потоковой передачей информации. Архитектура

интерфейса построена на принципах модульности и расширяемости, что позволяет интегрировать его с современными инструментами мониторинга, кэширования и обработки ошибок без дополнительных библиотек.

5. Обработка успешных и ошибочных сценариев

При работе с сетью важно различать два уровня отказов: ошибки уровня сети и ошибки уровня протокола. Первый тип возникает при отсутствии соединения, таймаутах или блокировке запроса системой безопасности. В этом случае операция немедленно завершается с отклонением. Второй тип соответствует ситуациям, когда запрос успешно дошёл до сервера, но тот вернул статус, указывающий на проблему с обработкой, авторизацией или доступом к ресурсу. В таких случаях интерфейс не считает операцию ошибочной автоматически, поэтому разработчик должен явно анализировать коды состояний и принимать решение о дальнейших действиях. Грамотная обработка включает отображение индикаторов загрузки, информирование пользователя о причинах отказа, повторные попытки при временных неполадках и безопасное завершение сценария при неустранимых ошибках.

6. Ограничения безопасности и механизмы кросс-доменного взаимодействия

Браузеры реализуют строгую политику одинакового источника, которая запрещает скриптам обращаться к ресурсам на других доменах без явного разрешения. Это фундаментальный механизм защиты пользовательских данных от несанкционированного доступа. Для легального взаимодействия между разными источниками применяется система заголовков, позволяющая серверу указывать, какие домены, методы и заголовки разрешены для кросс-доменных запросов. При определённых условиях браузер автоматически выполняет предварительный запрос для проверки прав, прежде чем отправить основной запрос. Понимание этих ограничений необходимо для корректной настройки взаимодействия клиентских и серверных частей,

выбора методов аутентификации и проектирования безопасных архитектур.

7. Современные практики и рекомендации

Разработка сетевой логики требует соблюдения ряда архитектурных принципов. Рекомендуется использовать современный синтаксис ожидания вместо ручного управления цепочками, так как это повышает читаемость и упрощает сопровождение. Все запросы должны сопровождаться состояниями загрузки и явной обработкой как сетевых, так и протокольных ошибок. Для длительных или фоновых операций целесообразно предусматривать возможность отмены, чтобы освободить ресурсы при закрытии страницы или смене контекста. Важно избегать передачи избыточных данных, использовать кэширование там, где это допустимо, и не допускать блокировки пользовательского интерфейса тяжёлыми операциями парсинга. Следование этим принципам обеспечивает стабильность, безопасность и высокую производительность приложений.

8. Инструменты отладки и мониторинга сетевых операций

Анализ работы асинхронных механизмов и сетевых запросов невозможен без использования встроенных панелей разработчика. Специализированные вкладки позволяют отслеживать все исходящие и входящие запросы, просматривать заголовки, тела ответов, время выполнения каждого этапа и статусы соединений. Визуальные диаграммы загрузки помогают выявить узкие места, дублирующие обращения или избыточные данные. Эмуляция медленных соединений и режимов офлайн-работы позволяет проверить устойчивость интерфейса к нестабильной сети. Регулярное применение этих инструментов формирует навыки системной диагностики, позволяя переходить от эмпирического исправления ошибок к инженерному проектированию надёжных коммуникационных слоёв.

Методика и порядок выполнения лабораторной работы:

1. Создайте кнопку «Загрузить данные» и контейнер для вывода.

2. Напишите async функцию с fetch('https://jsonplaceholder.typicode.com/users').
3. Преобразуйте ответ в JSON, отрендерите список пользователей в DOM.
4. Реализуйте обработку ошибок сети и HTTP-статусов через if (!response.ok).
5. Добавьте индикатор загрузки (spinner) и блокировку кнопки во время запроса.
6. Протестируйте с неверным URL, убедитесь в корректном отображении ошибки.

Контрольные вопросы:

1. Как работает Event Loop в контексте асинхронных операций?
2. Чем Promise удобнее callback-функций?
3. Как преобразовать async/await в цепочку .then()?
4. Что такое CORS и почему браузер блокирует некоторые запросы?
5. Как отличить ошибку сети от ошибки HTTP-статуса в Fetch?

Лабораторная работа № 9

Знакомство с работой Git

Цель работы: ознакомиться с основами работы с **Git**

Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

Разработка – это почти всегда командная игра. Умение работать в команде – одно из требований к разработчикам.

Чтобы эффективно работать в команде, мало знать синтаксис языка, ключевые библиотеки и уметь обращаться с базами данных. Необходимо уметь работать в удобной для команды системе контроля версий.

О системах контроля версий их преимуществах и недостатках можно почитать:

https://ru.hexlet.io/courses/git_base/lessons/vcs_intro/theory_unit https://devpractice.ru/git-for-beginners-part-1-what-is-vcs/ https://biz30.timedoctor.com/ru/система-контроля-версий/

и пр.

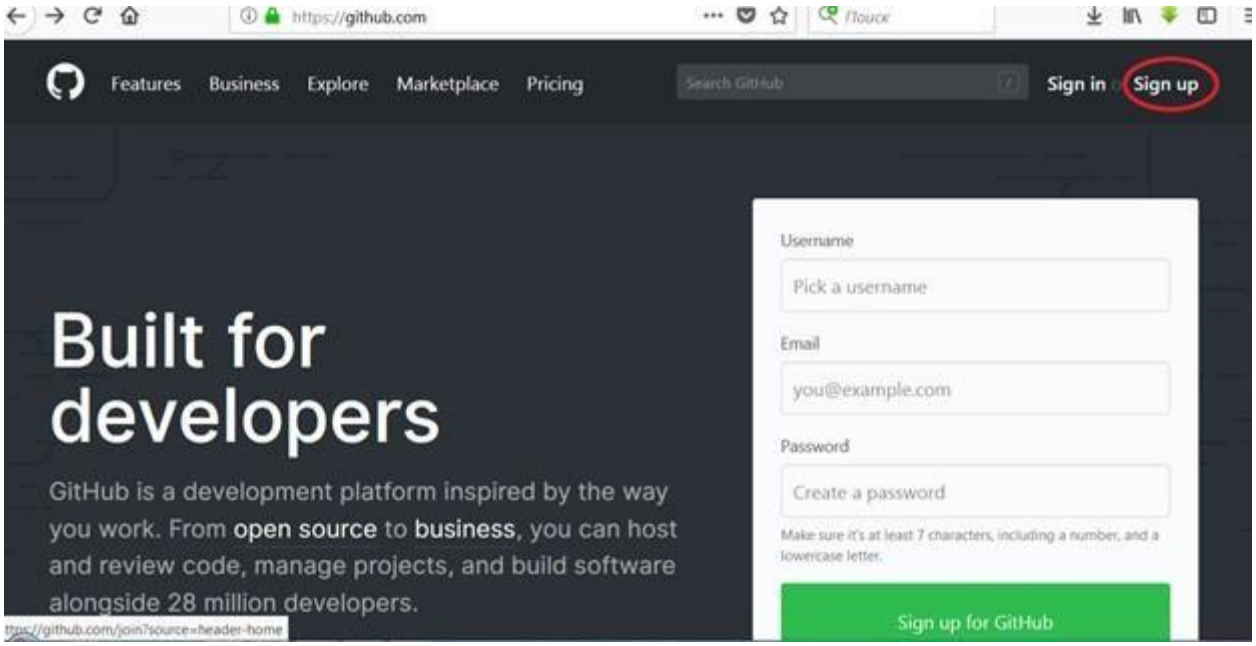
Основные шаги.

Шаг 1. Выбираем git-хостинг

Git-хостинг на разных условиях предлагают десятки компаний. Самые известные из них: Github, Sourceforge, Google Code, GitLab, Codebase. Выбери удобный интерфейс и регистрируйся на понравившемся хостинге. В этой статье мы рассмотрим работу с git-хостингом на примере Github’a.

Шаг 2. Регистрация

Процедура регистрации на Гитхабе простая и интуитивно понятная даже для тех, чей уровень английского далёк от Upper Intermediate.



Логин, пароль, почта –> подтверждение, и связь с мировым сообществом программистов налажена.

Шаг 3. Создание репозитория

Можно создать любое количество репозиториев, у каждого из которых будет issue tracking, wiki, условия для проведения code review и многое другое.

Политика разработчиков Github предполагает бесплатное использование хостинга для всех open-source проектов.

Чтобы создать новый репозиторий нажмём кнопку + в верхней части экрана и выберем New repository

Создание репозитория на Гитхабе

Многие разработчики рано или поздно сталкиваются с необходимостью создания приватного репозитория, код из которого доступен только их команде. Для этих случаев на Github’е есть определённый тарифный

план.

Но пока острой необходимости в создании приватного репозитория у нас нет, создадим обычный.

Жмём волшебную кнопку Create внизу экрана, и репозиторий готов.

Шаг 4. Работа с репозиторием

Работа с репозиторием может вестись из командной строки, напрямую из среды разработки или из графического интерфейса (git — клиент приложения).

Работа с графическим интерфейсом позволяет лучше понимать процессы, происходящие в локальном и удалённом репозитории. Поэтому я рекомендую начать работу с git с использованием графического интерфейса.

Шаг 5. Выбираем Гит-клиент

Потом, когда суть процессов изменения и обновления (восстановления) информации в репозитории станет для Вас очевидна, можно работать и через командную строку. В этом принципе работы есть немало своих преимуществ. Например, все новые опции Гитхаба реализуются сначала для использования в командной строке, и только потом адаптируются под графические интерфейсы.

Но вернёмся к git-клиентам.

Самыми популярными гит- клиентами на данный момент являются:

SmartGit

Удобное приложение гармонично сочетает все необходимые функции и доступную интуитивно понятную систему управления. SmartGit – один из самых удобных графических интерфейсов для профессиональной разработки. Некоммерческая разработка и разработка open-sourse проектов не требуют платной лицензии.

GitHub Desktop

«Родной» графический интерфейс Гитхаба. GitHub Desktop работает под Windows и Mac и практически полностью копирует функционал основного сайта. Работает под той же учётной записью. Правда, не всегда оперативно справляется с большими программами. Зато отлично подходит для начала работы с git.

GitKraken

Поддерживает GitHub, Bitbucket и Gitlab. Кракен очень любят программисты – фрилансеры, которым периодически приходится менять команды, а значит, и условия командной разработки. Возможность работы с разными git-хостингами через привычное приложение со знакомым интерфейсом в таких случаях играет важную роль.

SourceTree

SourceTree позволяет работать с Bitbucket и GitHub. В приложении довольно простой интерфейс, подходящий, как для опытных программистов, так и для новичков.

Шаг 6. Работа со SmartGit

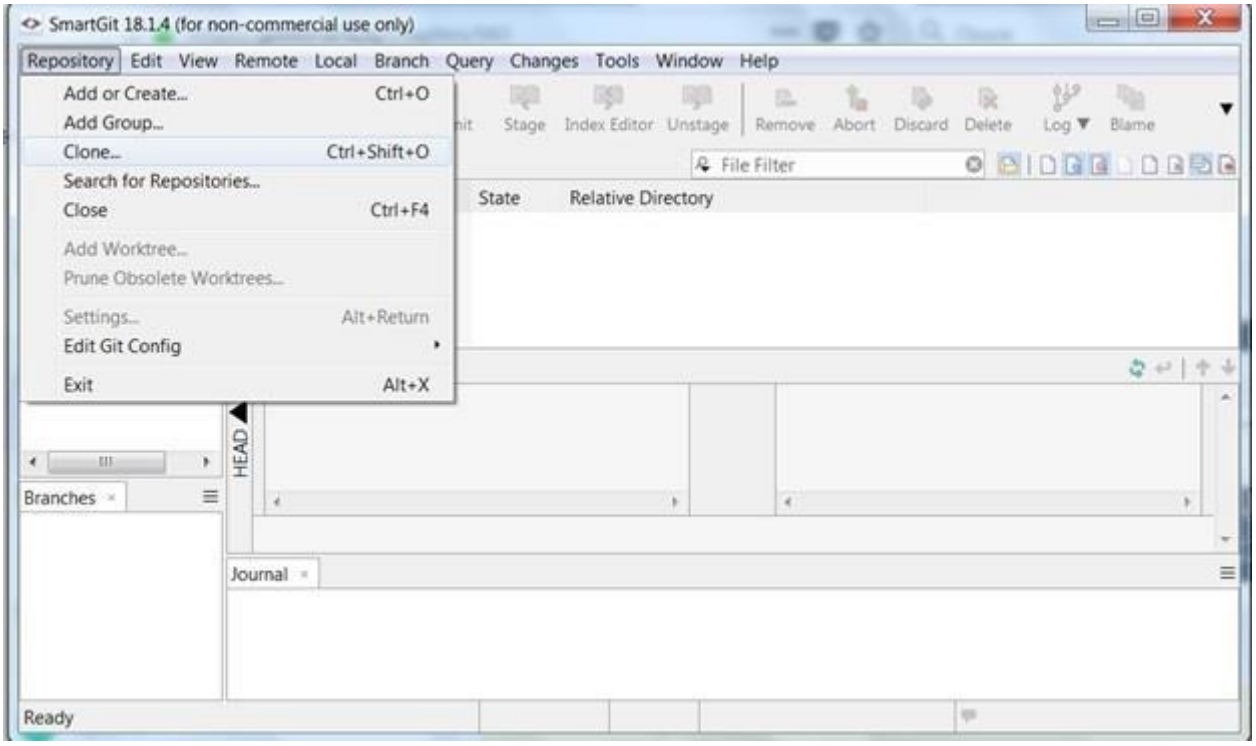
Скачать SmartGit можно, например,
https://www.syntevo.com/smartgit/download/
Устанавливаем и запускаем.

Основные операции для работы с git

Clone

Первое, чему стоит научиться – это снимать копию проекта из удалённого репозитория в локальный.

Делается это довольно просто:



Clone

Затем копируем ссылку репозитория, созданного на Гитхабе (шаг 2)

Ссылка на репозиторий

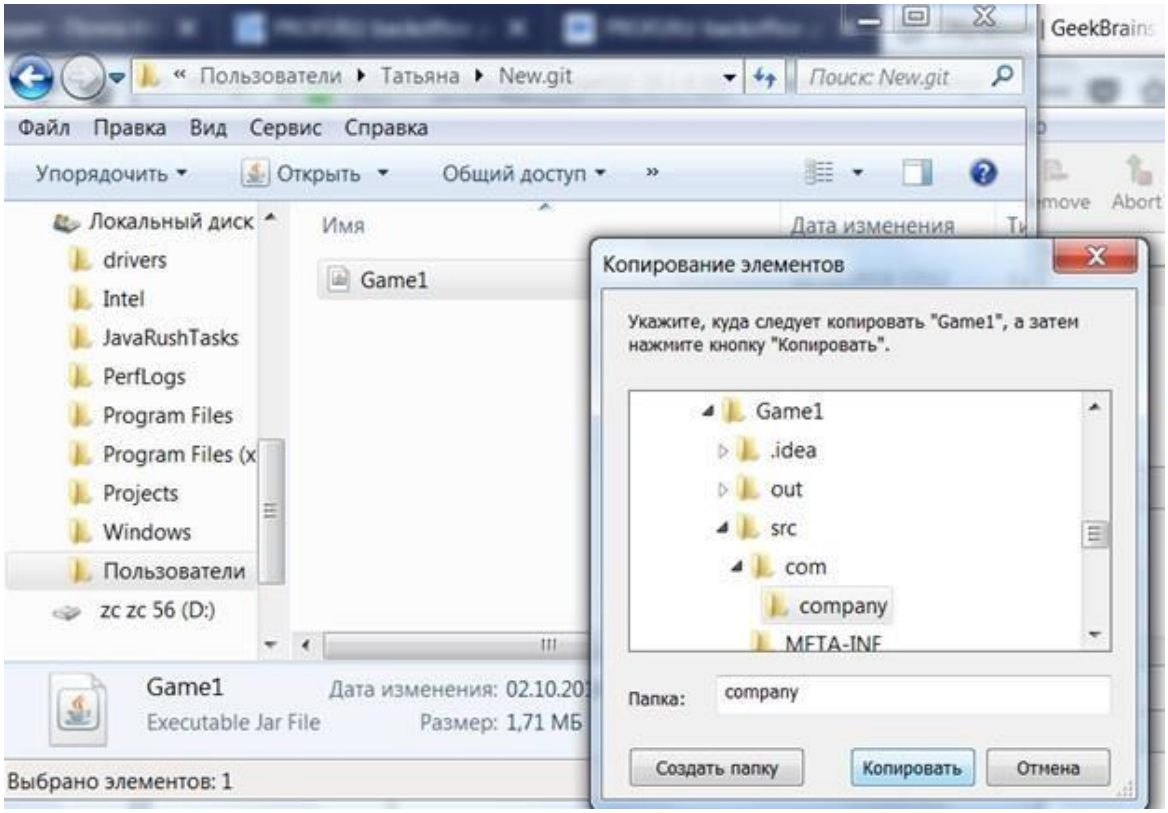
Вставляем адрес удалённого репозитория в нужную ячейку в открывшемся

окне, выбираем расположение нового локального репозитория у нас на компьютере, и получаем готовый локальный репозиторий.

К слову, аналогичным образом можно клонировать чужой открытый репозиторий и поближе познакомиться с чужим кодом.

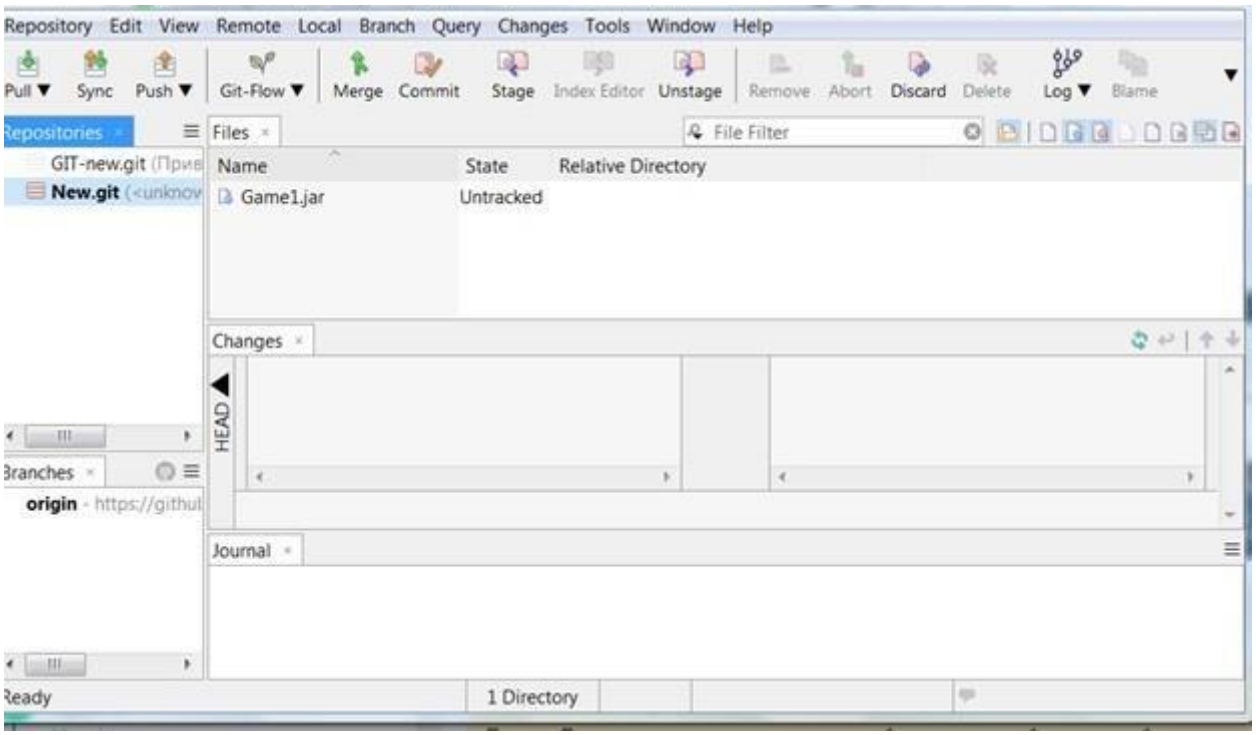
Commit

Репозиторий готов – пора приступать к работе. Написанный код мы помещаем в локальный репозиторий — папку .git (путь к которой мы указали в операции clone).



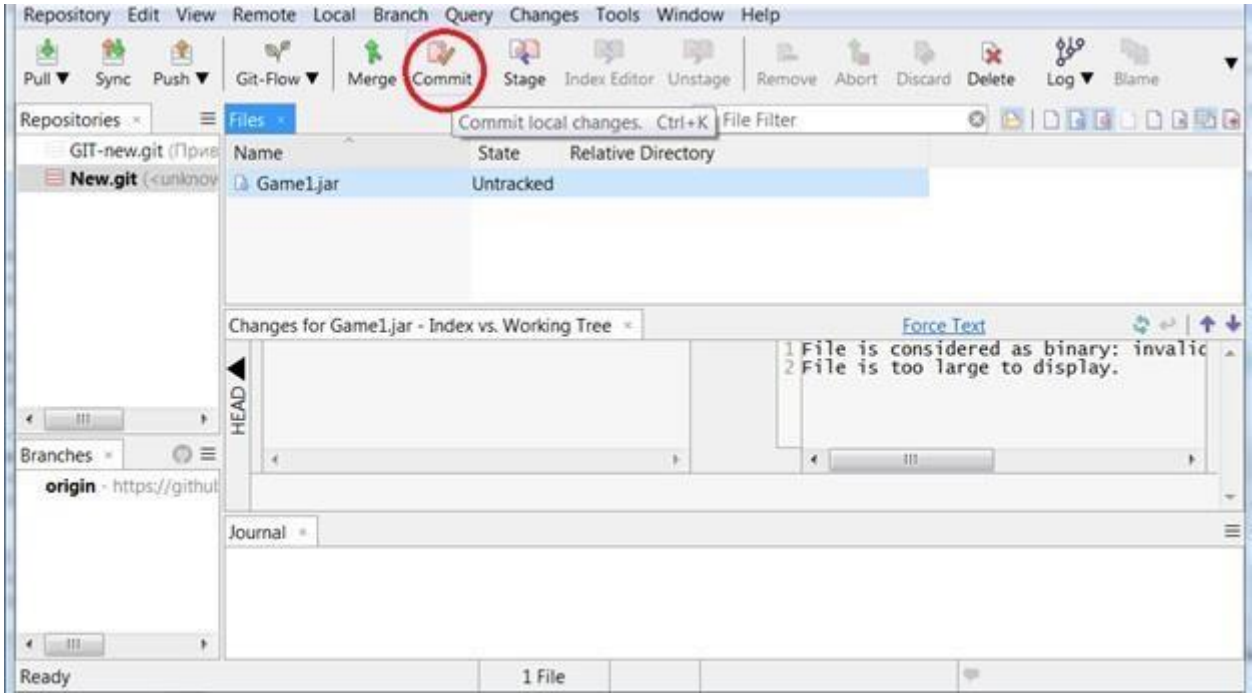
Добавление файла в локальный репозиторий.

Если всё прошло успешно, в окошке SmartGit’a появится скопированный файл.



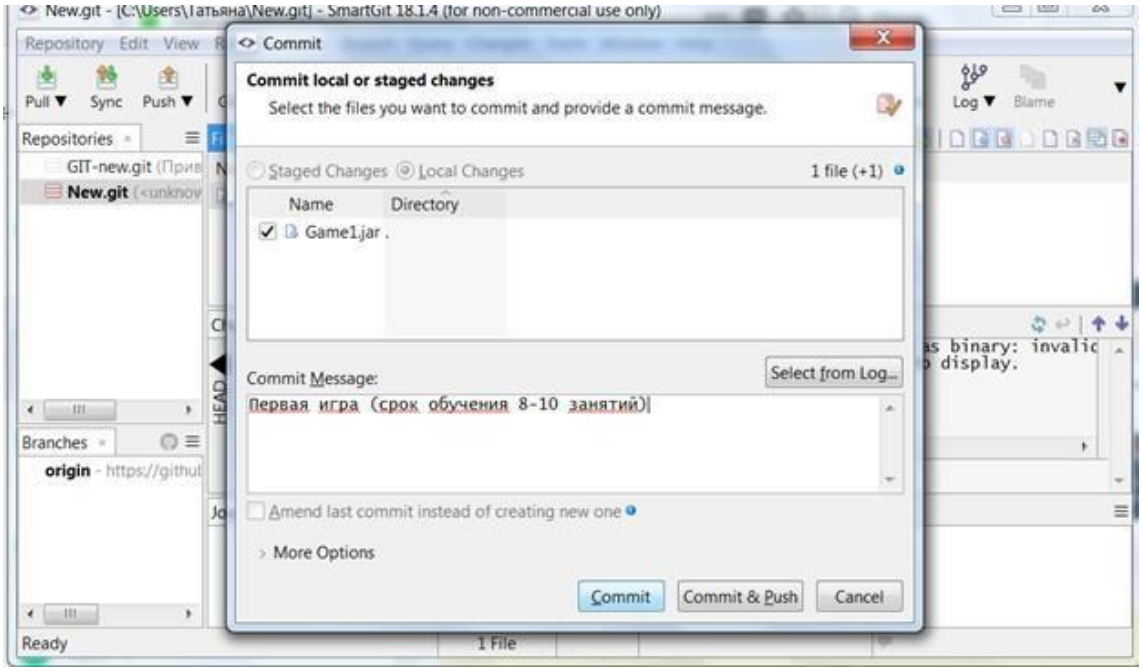
Новый файл в SmartGit.

Для того чтобы зафиксировать изменения в локальном репозитории, нажимаем кнопку Commit.



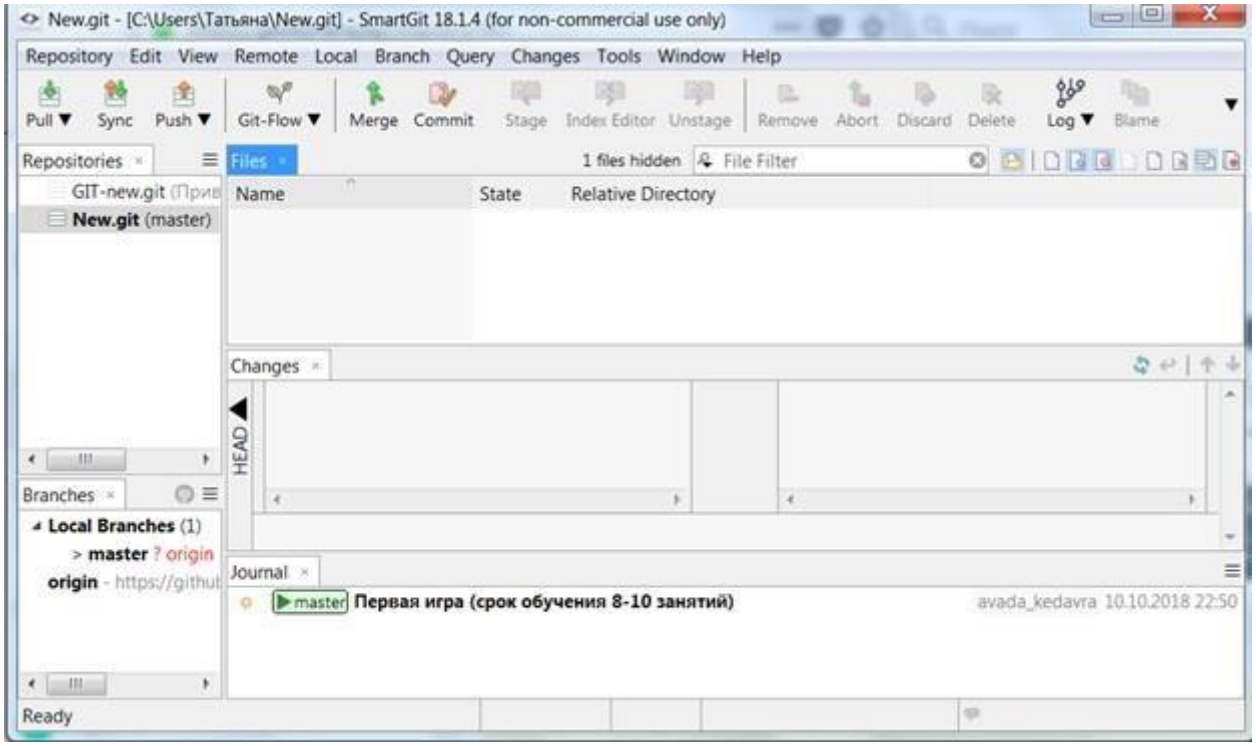
Commit

В открывшемся окне пишем пояснительный комментарий к сохраняемому файлу и снова нажимаем кнопку Commit



Пояснения к Commit’у

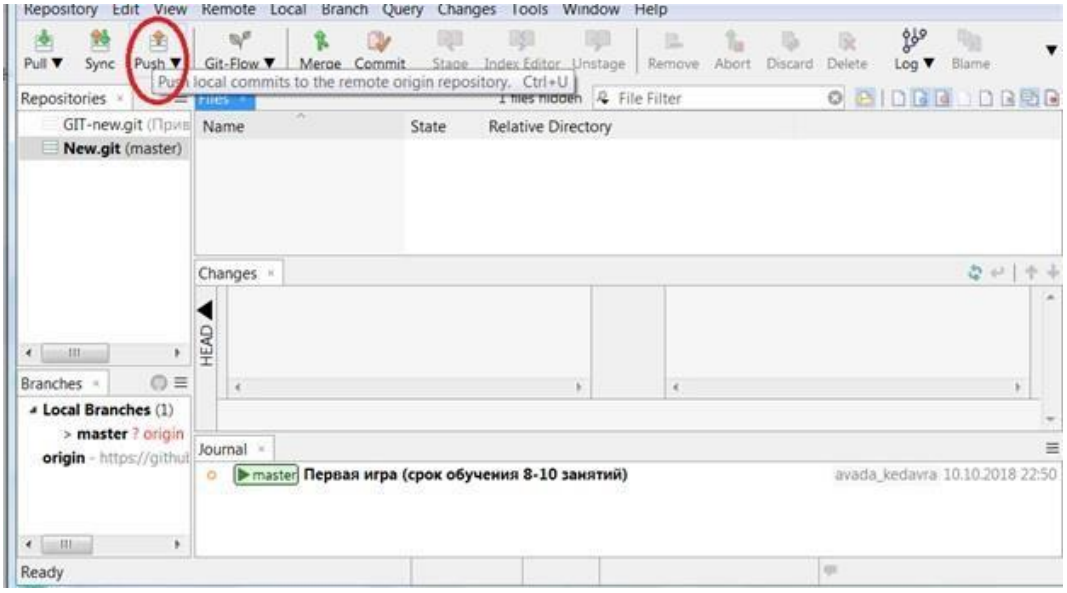
Файл сохранён, а изменения внесены в журнал.



Файл отправлен в локальный репозиторий

Push

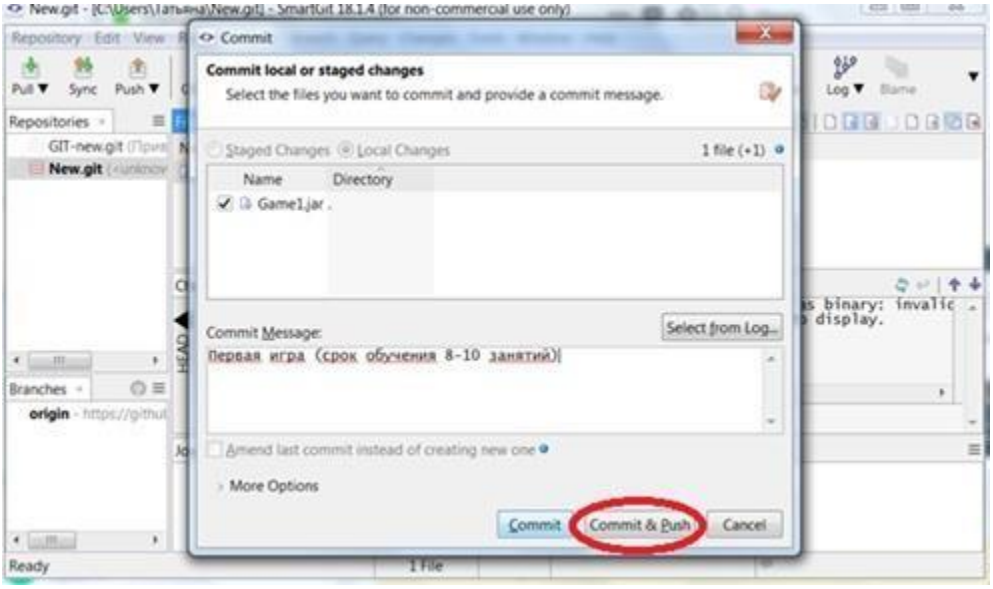
Теперь заглянем на Github.com в наш удалённый репозиторий. Там до сих пор нет ни одного файла. Нужно срочно менять ситуацию. Чтобы перенести изменения, внесённые в локальный репозиторий, в удалённый репозиторий, необходимо нажать кнопку Push.



Push

К слову, отправить изменения в удалённый репозиторий, нам предлагают

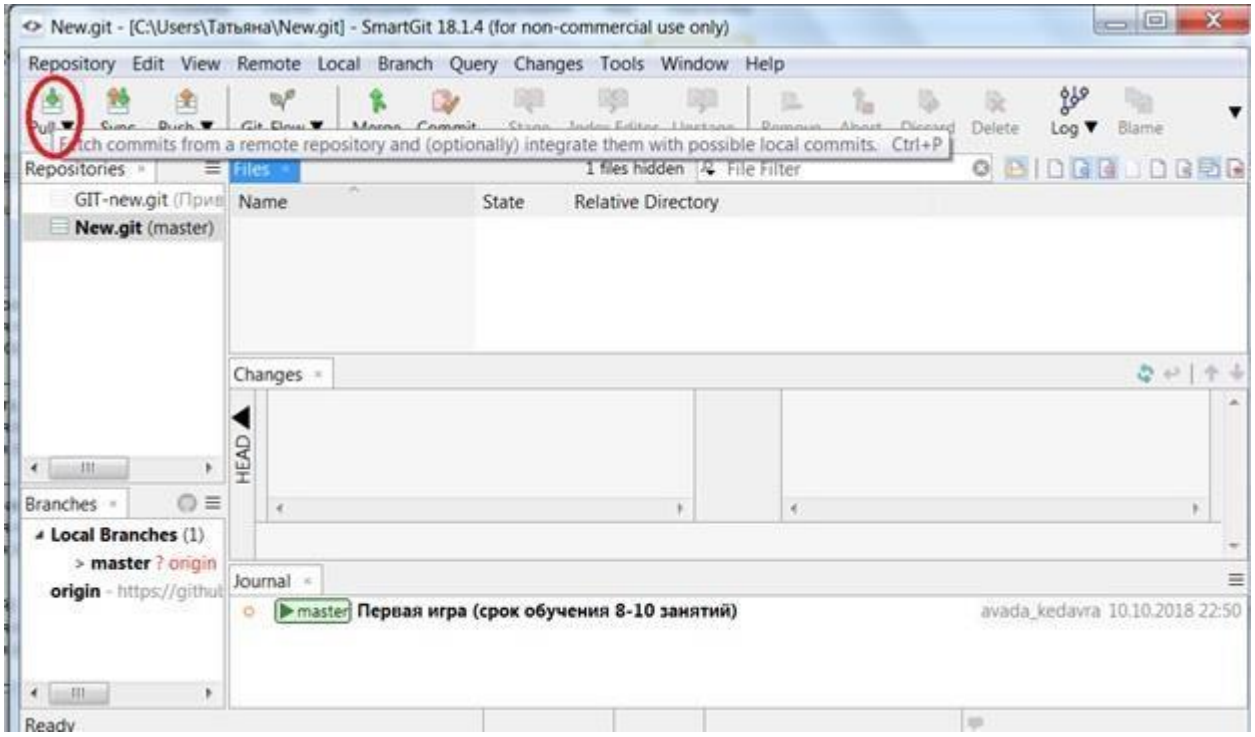
ещё в точке Commit’a



Commit & Push

Pull

Возникает резонный вопрос: как получат изменения остальные участники разработки, если они клонировали проект в самом начале? Для этого существует команда Pull, передающая в локальный репозиторий все изменения, происходящие в удалённом.

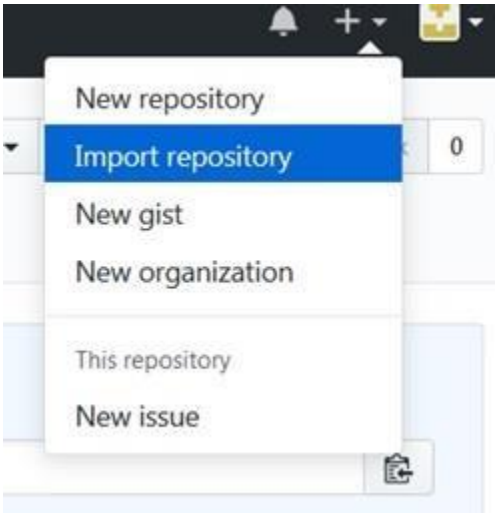


Pull

К слову, для командной разработки на Гитхабе есть ещё несколько важных опций.

Перенос информации из сторонних репозиториев на Гитхаб

Когда нужно собрать разрозненные кусочки кода в один проект, используйте кнопку Import repository и работайте с файлами в удобном репозитории Гитхаба.



Импортировать репозиторий

Кнопка New gist на этой панели предназначена для мгновенного обмена информацией.

А кнопка New organization открывает массу возможностей для командной разработки.

Задание

1. Изучить теоретическую часть лабораторной работы
2. Пройти курс: https://gb.ru/courses/1117
3. Изучить теоретический материал курса https://gb.ru/courses/1117 .
4. Написать отчет о прослушанном курсе с описанием выполненных заданий

Лабораторная работа № 10

Клиентское хранилище данных и основы управления состоянием

Цель: Изучить механизмы локального хранения данных в браузере, освоить работу с хранилищами сессии, локального хранилища и cookies, понять принципы сохранения и восстановления состояния пользовательского интерфейса.

Формируемые компетенции: ПК-2, ПК-4

Теоретическое обоснование

В современной веб-разработке разделение между кратковременным интерфейсным состоянием и долгосрочными данными пользователя является архитектурной необходимостью. Браузер предоставляет несколько встроенных механизмов клиентского хранения, каждый из которых решает определённый класс задач. Хранилище сессии сохраняет данные исключительно в пределах текущей вкладки браузера и автоматически очищается при её закрытии, что делает его оптимальным для временных форм, черновиков или навигационных фильтров. Локальное хранилище сохраняет информацию бессрочно до явного удаления пользователем или программной очистки, применяется для настроек интерфейса, кэширования редко меняющихся конфигураций и реализации офлайн-режимов. Механизм cookies исторически предназначен для передачи данных между клиентом и сервером, поддерживает автоматическую отправку с каждым запросом, позволяет задавать срок жизни, область видимости по домену и пути, а также флаги безопасности. Все клиентские хранилища работают синхронно, имеют ограничения по объёму хранимых данных и подвержены тем же правилам безопасности изоляции по источнику, что и сетевые запросы. Понимание разницы между механизмами, их жизненным циклом, ограничениями и сценариями применения позволяет проектировать устойчивые интерфейсы,

минимизировать лишние обращения к серверу и обеспечивать предсказуемое поведение приложения при перезагрузке страницы или разрыве соединения.

Методика и порядок выполнения лабораторной работы:

1. Создайте HTML-страницу с формой настроек интерфейса (выбор цветовой темы, языка, чекбокс сохранения истории поиска).
2. Реализуйте скрипт, который при изменении параметров сохраняет их в соответствующее хранилище: настройки темы в локальное хранилище, временные черновики ввода в хранилище сессии, идентификатор сессии в cookies.
3. Напишите логику восстановления состояния при загрузке страницы: чтение сохранённых значений, применение темы, заполнение полей формы.
4. Добавьте кнопки очистки: отдельное хранилище, все данные, конкретный cookie с параметром срока жизни.
5. Используйте инструменты разработчика для мониторинга записей, проверки ограничений квоты, просмотра флагов безопасности и эмуляции очистки данных.
6. Зафиксируйте результаты тестирования, скриншоты вкладок хранилищ и выводы об эффективности каждого механизма в отчёте.

Контрольные вопросы:

1. В чём принципиальное различие между жизненным циклом хранилища сессии и локального хранилища?
2. Какие ограничения по объёму и безопасности характерны для клиентских хранилищ?
3. Почему cookies автоматически отправляются с каждым HTTP-запросом, а механизмы localStorage и sessionStorage — нет?
4. Как корректно обрабатывать ситуацию превышения квоты хранилища или блокировки доступа в режиме инкогнито?
5. Какие данные допустимо хранить на клиенте, а какие необходимо передавать только через защищённые серверные сессии?