# Задание на лабораторные работы по курсу Web-Технологии. 2024.

# Лабораторная работа №1. Работа с утилитами ipconfig, ping, traceroute, netstat и т.д.

1. С помощью утилиты ipconfig (с /all) определите IP-адрес (IPv4) и маску подсети для своего компьютера. Определите количество адресов в сети, адрес сети и широковещательный адрес. Утилиту запустить на домашнем компьютере и в сети МАИ.
2. С помощью утилиты netstat (имеет параметры –a и –abno) проверить все возможности данной утилиты на кафедральном и домашнем компьютере. Вывести только строки с различными IP адресами. Выполнить проверку на домашнем компьютере netstat – abno с включенным браузером и без (отобразить отличия). Выполнить команду netstat –r на домашнем и кафедральном компьютере. На домашнем компьютере рассмотреть два различных подключения (к сети и через точку доступа телефона). По возможности рассмотреть подобные параметры на домашнем роутере.
3. С помощью утилиты nslookup определите ip-адреса (IPv4 и IPv6) для сайта Mai.ru, Yandex.ru и выбранного Вами host (web страница/ сайт). С помощью утилиты nslookup определите MX-адреса почтовых сервисов Mai.ru, mxs.mail.ru, Yandex.ru, mx.Yandex.ru
4. Выберите наиболее интересный для Вас host (web страницу/ сайт). Для данного сайта, с помощью утилиты tracert определите маршрут следования и время прохождения пакетов из различных сетей: домашней, института МАИ (кафедральный компьютер), общественной wi-fi сети.
5. Проверьте доступность 5 адресов хостов (IP и DNS) (любых, из полученных в пункте 4) из различных сетей: домашней, института МАИ (кафедральный компьютер), общественного wi-fi. Проверять адреса и DNS имена необходимо из тех, которые выдала команда tracert, т.е. из следующих друг за другом (не обязательно) при перемещении пакета по сети Интернет. Результаты занести в таблицу

|  |  |  |  |
| --- | --- | --- | --- |
| IP адрес | DNS | Доступность по IP | Доступность по DNS |
|  |  |  |  |

# Лабораторная работа №2. Сокеты

1. Разработать программу клиент, работающий на сокетах на выбранном Вами языке программирования. Данный клиент должен хранить лог файл соединений. В лог файле хранится время подключения и адрес сервера, время отправки сообщения и само сообщение, время получения сообщения и само сообщение. При подключении к серверу клиент через определенный промежуток времени (задается разработчиком клиента) отправляет сообщение серверу: ФИО и группа студента, а также ждет от сервера. Данные о сервере необходимо вводить через файл конфигурации. В отчете привести код клиента и один тестовый лог файл.
2. Разработать программу сервер, которая ожидает строку от клиента. Через промежуток времени (моделирование работы сервера), он отправляет в ответ эту же строку, только зеркально отраженную и добавляет к сообщению ФИО студента, написавшего сервер. (пример ответа: Чиволвап Йирю Вотит. Сервер написан Титовы Ю.П. М3О-3ХХБ-ХХ). Сервер должен эмулировать работу путем временной задержки. Отключение клиентов от сервера совершать через определенное время работы клиента. Сервер и клиент должны быть написаны на различных языках программирования. Сервер ведет лог-файл, в котором содержатся Время запуска сервера, Время подсоединения клиента, Время получения сообщения и само сообщение, Время отправки сообщения и само сообщение, Время отключения клиента от сервера. Лог-файл не очищать.
3. Доработать программу сервер для выполнения асинхронных запросов от клиентов. Четные варианты реализуют многопоточный асинхронный сервер, нечетные – многопроцессный. Номер варианта определяется номером по списку. При доработке сервера необходимо создать новый проект, т.е. в результате должно получиться две программы сервера.
4. Запустить 5 клиентов в сети кафедры и 1 сервер (5 студентов). В отчете указать настройки для подключения и лог файлы клиентов и серверов. Проводить тестирование одновременного подключения, т.е. второй, третий и т.д. пользователь подключается к серверу пока первый пользователь еще не получил ответ. Желательно провести тестирование как на однопоточном так и на асинхронном сервере, описать отличия.

# Лабораторная работа №3. Модель OSI. Стек протоколов. Wireshark.

1. Рассмотреть классическую многоуровневую модель OSI. Определить какие протоколы возможны на каждом уровне (привести по 2 примера на каждом уровне).
2. Запустить программу Wireshark. Настроить программу на активное интернет соединение. Рассмотреть различные TCP и UDP соединения. Привести в трехстороннее рукопожатие и трехстороннее завершение сеанса TCP (привести не только сам процесс, но и раскрыть флаги для каждого пакета). Привести пример UDP пакета.
3. Рассмотреть процесс установления TLS соединения. В отчете привести заголовки стека протоколов основных сообщений при создании TSL соединения. Раскрыть все параметры TLS соединения, отобразить все Chiper Suites, Extension, Certificate, Показать данные протокола DH.
4. С помощью браузера отправить запрос для получения HTML страницы по протоколу HTTP. Найти соответствующий запрос и ответ в программе Wireshark. В отчете полностью отобразить все содержимое стека протоколов на каждом уровне для получения HTML (не приводить полностью код HTML, а только до конца страницы и блок HTTP chunked response)
5. Проанализировать содержимое данных, отправленных в запросе и полученных в ответе. Расписать в отчете основные моменты соединения для каждого из протоколов.
6. Рассмотреть загрузку других ресурсов сайта (CSS/ PNG/ JPG и т.д.) как запрос на данный ресурс, так и ответ (привести только HTTP протокол).

# Лабораторная работа №4. Протокол HTTP. Протокол telnet. Web-сервер.

1. Запустите сеанс telnet (запускается в командной строке командой telnet). Разрешите режим отображения вводимых с клавиатуры символов с помощью команды set localecho. В соответствии с протоколом HTTP необходимо установить соединение с веб-сервером. Для этого с помощью команды open устанавливается соединение, например: open www.yandex.ru 80.
2. С **использованием сокета** напишите запрос к этому же сайту. Получите ответ от сервера (в отчете ответ приводить не полностью).
3. Изучите полученный ответ сервера. Рассмотрите, как именно приходит код документа, как получаются CSS стили. Как получить доступ к дополнительным файлам (CSS и JS). Привести пример такого дополнительного HTTP запроса.
4. Отобразите в отчете coockies. Как устанавливаются сервером и как передаются от клиента к серверу.
5. Попробуйте сформировать HTTP запрос для получения картинки или другого файла, с расширением отличным от текстового. Изучите ответ от сервера. Приведите запрос и ответ в отчете.
6. **На сокетах** необходимо написать web-сервер осуществляющий вывод страницы с указанием ФИО студента и всех простых чисел от 0 до 100. Четные варианты реализуют многопоточный асинхронный сервер, нечетные – многопроцессный. Номер варианта определяется номером по списку.

# Лабораторная работа №5. FTP сервер.

1. С помощью telnet провести подключение к FTP серверу и вывести его файловую структуру глубиной не более 5 папок. Процесс подключения и файловую структуру привести в отчете. Скачать с сервера файл с расширение JPG. В отчете вывести все команды, которые требовались для скачивания.
2. С помощью программы, написанной на сокетах скопировать структуру папок (с начального уровня и на 2 уровня ниже) с сервера в определенную папку на машине клиента.
3. Провести анализ передаваемых команд по созданию FTP соединения (начального подключения, для передачи файла) и примеры передачи данных с помощью программы Wireshark.
4. Используя специальное программное обеспечение (например filezilla) скопировать файл с компьютера на FTP сервер.

# Лабораторная работа №6. Почтовый сервер.

1. Используя почтовые протоколы отправить письмо на почтовый ящик одногрупника. В почтовом сообщении написать: тема сообщения – название лабораторной работы, в тексте сообщения написать ФИО и группу отправителя, ФИО и группу получателя и время отправки сообщения. Установить флажок «Важное» для данного письма.
2. Используя протокол получить письмо зайти на свой ящик. Получить информацию (и привести в отчете) о количестве писем на ящике. Привести в отчете заголовки некоторых писем (не более 5 шт.). Получить письмо, присланное одногрупником из почтового ящика. Привести его в отчете.
3. Получить письмо с вложенным файлом, получить вложенный файл .
4. Провести анализ передаваемых команд и данных с помощью программы Wireshark.

# Лабораторная работа №7. HTML, CSS.

1. Выберите одну понравившуюся Вам страницу сайта. Откройте ее в браузере, например Google Chrome, и найдите средства отображения ее кода.
2. Убедитесь в наличии обязательных тегов для HTML. (привести скриншот или записать обязательные теги). Метатеги должны быть представлены со всеми их атрибутами.
3. В блоке Head найдите заголовок страницы. Убедитесь, что он соответствует показанному в браузере (Привести код заголовка и скриншот заголовка страницы в браузере). Попробуйте изменить заголовок и посмотреть полученные изменения. (Привести код заголовка и скриншот заголовка страницы в браузере). После обновите страницу.
4. В блоке Head найдите метатеги. Выпишите (если есть) метатеги описание и ключевые слова. Проверьте, находит ли поисковик данную страницу по ее описанию и ключевым словам (В результате привести скриншоты поисковых запросов). Рассмотрите другие метатеги, установленные на странице (в отчет привести не более 5 метатегов). Метатеги приводятся в виде кода. Для каждого метатега необходимо дать его описание. При наличии на сайте метатегов og: привести 3 из них. Дать описание этим метатегам. Аналогично для метатегов Twitter:
5. В блоке Body рассмотрите разметку страницы тегами div. В случае наличия новостной, или любой другой ленты, рассмотрите, как именно она размечается. В отчете опишите теги div с глубиной вложенности до 5. На одном скриншоте выделите рамочками выбранные 5 тегов div. Под скриншотом опишите код этих div со всеми их атрибутами (вложенные в div другие теги не приводите). Посмотреть, как изменятся эти теги при отображении страницы на другом устройстве, например на мобильном телефоне: Скриншот с рамочками и код.
6. Для пяти элементов (желательно один из них div) определить их class или id. Выписать их код в отчете. Для каждого параметра в таблице стилей найти стиль, который описывает отображение данного элемента. Результаты отобразить в виде кода из CSS файла. Изменить значения стиля (показать значения до и после). Изменились ли другие объекты на странице (показать скриншот страницы до и после). Поменять класс одного любого тега на другой из файла CSS, и посмотреть изменение стиля. (класс до и после, скриншот до и после) Определить момент подключения стиля к блоку.
7. Выбрать любой объект HTML (например кнопку). В зависимости от остатка деления номера по списку на 11 сделать
   1. При наведении курсора элемент должен увеличиваться
   2. При наведении курсора элемент должен уменьшаться
   3. При наведении курсора элемент должен поворачиваться на 90 градусов по часовой стрелке
   4. При наведении курсора элемент должен поворачиваться на 90 градусов против часовой стрелки
   5. При наведении курсора элемент должен вращаться по часовой стрелке
   6. При наведении курсора элемент должен вращаться против часовой стрелки
   7. При наведении курсора элемент должен вращаться по окружности
   8. При наведении курсора элемент должен плавно перемещаться вправо и влево
   9. При наведении курсора элемент должен мигать (переключаться между видимым и невидимым состояниями)
   10. При наведении курсора элемент должен изменять прозрачность элемента от полностью непрозрачного до полностью прозрачного и обратно
   11. При наведении курсора элемент должен создавать эффект "пульсации" элемента (то есть увеличение и уменьшение размера постепенно)
8. Найти и отобразить в отчете ссылки на картинки, видео и аудиофайлы (не более 5). В отчете привести только код ссылки в теге со всеми его атрибутами. Отметить ссылки, которые позволяют перейти на другую страницу (не более 5). В отчете привести только код ссылки в теге со всеми его атрибутами.

# Лабораторная работа №8. JS.

1. Проверьте возможности консоли в браузере. Выведите в конце сайта Ваше ФИО, номер группы и все четные числа от Вашего номера в списке группы до 100. Весь код отображать в виде текстовой информации, а результаты делать с помощью скриншотов.
2. С помощью функций [prompt(question, [default])](https://developer.mozilla.org/ru/docs/Web/API/Window/prompt), [confirm(question)](https://developer.mozilla.org/ru/docs/Web/API/Window/confirm), [alert(message)](https://developer.mozilla.org/ru/docs/Web/API/Window/alert) (использовать все 3 функции), составьте диалог с пользователем для определения его желания зайти на сайт.
3. С помощью перемещения по DOM дереву выведите на страницу сайта все заголовки метатегов (элементы) на один уровень ниже для тега Head.
4. С помощью перемещения по DOM дереву выведите на страницу сайта самую длинную (в глубину) цепочку тегов (только заголовки метатегов).
5. С помощью поиска объектов в DOM дереве измените стиль, подобно пункту 6 в лабораторной работе №2. Приведите скриншот до и после изменения
6. Выберите текстовую информацию на сайте. Создайте кнопку, по нажатию на которую текст будет пропадать.
7. При прокрутке сайта вниз добавьте на страницу время и величину прокрутки.
8. Создайте несколько функций – промисов. Задача функции вывести запрос ввода почты через диалог при прокрутки страницы вниз. Используйте метод Promise.all() для выполнения нескольких промисов параллельно и выведите результаты их выполнения в консоль. Используйте метод Promise.race() для выполнения нескольких промисов параллельно, но верните только результат первого завершившегося промиса. Создайте асинхронную функцию, которая будет использовать async/await для управления выполнением промисов и выведите результат в консоль.
9. Создайте функцию, которая будет обрабатывать изменения размеров окна браузера и пересчитывать позиции элементов на этапе onload
10. Если на странице есть ShadowDOM – опишите его и попробуйте обратиться к одному из его объектов. Если нет, то приведите скриншот поиска.

## Лабораторная работа №9. Критический путь рендеринга

1. Выберите одну понравившуюся Вам страницу сайта. Откройте ее в браузере, например Google Chrome, в режиме «Инкогнито» и найдите средства отображения ее пути рендеринга. Разберитесь со стадиями рендеринга страницы.
2. С помощью вкладки Perfomance рассмотрите критический путь рендеринга. Рассмотрите как последовательно загружаются элементы страницы. Что занимает самое большое время при **отрисовке** страницы. Сколько времени заняло работа со стилями CSS. Сколько всего времени **отрисовывалась** страница. Привести скриншоты.
3. Протестируйте страницу с помощью webpagetest.org или другим подобным сервисом. Рассмотрите вмененные интервалы основных событий отрисовки. Сравните моменты времени с моментами из вкладки Perfomance. Укажите, к каким сторонним ресурсам (сайтам) обращается страница сайта.
4. Рассмотрите ресурсы, запрошенных и загруженных до зеленой линии «Start Render». Рассмотрите методы, необходимые для уменьшения блокировки рендеринга. Найдите элемент JS. Предложите их перенос. Найдите элемент CSS, затрудняющий рендеринг. Предложите перенос этого элемента. Проанализируйте данные, ищите CSS- и JS-файлы, которые запрашиваются перед зелёной линией «Start Render», но не являются критичными для загрузки контента на первом экране. После определения потенциального блокирующего рендеринг ресурса, протестируйте его удаление, чтобы увидеть, не повлияет ли это на контент на отображение первого экрана страницы. Если такой файл не найден, то приведите пример для любого файла, меняющего отображение. (Привести имя файла и скриншоты до и после блокировки, Блокировку проводить на панели «Network» в пункте «Block Request URL»).
5. На вкладке «Request blocking»нажмите на значок «+». Введите любой паттерн для блокировки обнаруженных вами ресурсов, используя \* в качестве подстановочного знака. (не блокируйте все ресурсы, а только некоторые из них, например видеофайлы) Нажмите «Add» и обновите страницу. Опишите ожидаемый результат и приведите скриншот страницы.
6. Найти атрибуты async и defer и описать их назначение для данной страницы. По отображению критического пути доказать правильность выполнения браузером этих атрибутов. Если атрибутов нет, привести скриншот результатов поиска.
7. Проверьте наличие и использование HTTP/2 протокола на выбранной странице сайта. Опишите преимущества HTTP/2 перед HTTP/1.1 и его влияние на скорость загрузки контента.
8. Исследуйте вкладку "Coverage" в инструментах разработчика браузера для выбранной страницы сайта. Оцените, какие CSS и JS файлы не используются на странице, и предложите оптимизировать загрузку только необходимых ресурсов.
9. Изучите использование кэширования ресурсов на выбранной странице сайта. Предложите стратегии кэширования для улучшения производительности загрузки страницы.
10. Исследуйте возможности использования lazy loading для изображений на странице. Опишите, как это может помочь ускорить первоначальную загрузку страницы

## Лабораторная работа №10. Чат на сокетах

1. Разработать программу сервер, которая хранит список всех подключении клиентов (например, в динамическом массиве). Хранится сокет подключения, полученный в результате функции accept. Для каждого клиента сервер запрашивает кодовое имя (nickname). При получении сообщения от клиента (с соответствующим nickname) происходит передача этого сообщения всем остальным клиентам через сокетное соединение. Сервер сохраняет сообщение пользователя в файл. При подключении клиента передаются (от сервера к клиенту) последние 10 сообщений из данного файла.
2. Разработать программу клиент, которая работает в диалоге с пользователем и позволяет ему вводить данные. При подключении к серверу (ip адрес и порт сервера вводится пользователем) сервер запрашивает nickname, а пользователь вводит его в программе-клиенте. Программа клиент позволяет вводить сообщения пользователем от имени nickname и выводит сообщения от сервера на экран. Последние 5 сообщений сохраняются в текстовом файле (эмулирую кэширование). При запуске программы выводится содержимое данного файла.
3. Развернуть данное приложение в сети с одним сервером и 3-мя запущенными программами-клиентами. В отчете привести файлы клиента и сервера, а также скриншот работы программы.

## Лабораторная работа №11. P2P соединение на сокетах на примере обмена файлами

1. Разработать программу сервер, которая хранит список подключенных к серверу клиентов (например, в динамическом массиве). При этом, на сервере хранится информация об открытом (в состоянии listen) сокете на клиентах: ip-адрес и порт. Сервер ждет от клиента сообщение «информация о группе». На это сообщение сервер высылает список клиентов. При получении сообщения от клиента, о желании подключения к группе, сервер добавляет присланный в сообщении порт клиента (и ip-адрес клиента) в список клиентов, подключенных к серверу. При получении сообщения об отключения клиента, сервер удаляет запись об сокете клиента из списка. Формат сообщений на усмотрение студента.
2. Разработать программу клиент, которая подключается к серверу. Ip-адрес и порт сервера вводится из текстового файла конфигурации. Клиент отправляет серверу сообщение «информация о группе». После получения списка клиентов, клиент открывает сокет на прослушивание listen. Клиент отправляет серверу сообщении, о желании подключения к группе. В данном сообщении клиент сообщает серверу номер порта, открытого в состоянии listen. Клинет может закрыть сокет, с помощью которого общается с сервером. Клиент создает новый сокет, который присоединяется к открытым сокетам клиентов, полученных в сообщении от сервера. Аналогично к listen сокету клиента могут подключаться другие клиенты. Обработку подключенных клиентов проводить в асинхронном режиме работы. При завершении работы необходимо отправить серверу сообщение об отключении клиента. Если TCP-соединение с сервером разорвано, то необходимо заново подключиться к серверу. Также при завершении работы клиента необходимо закрыть listen сокет и всех соединений с другими клиентами.
3. Каждый клиент определяет папку для синхронизации, содержащую текстовые файлы. Клиенты, после установки соединения, обмениваются содержимым своих папок. При обмене содержимым папок пересылаются названия файлов. Если у получателя нет каких-нибудь файлов в папке, то он отправляет запрос на передачу содержимого файлов. Содержимое файлов передается через то же сокетное соединение. Для удобства можно сначала передавать данные от клиента, к которому подключаются, а поле передачи всех недостающих файлов, передавать файлы подключившегося клиента. После синхронизации всех файлов необходимо по остальным соединениям также отправить набор файлов. После работы всех программ у всех клиентов в папках будет одинаковы набор файлов. Все подключения и передаваемые файлы логируются (без содержимого). Для передачи содержимого файла логируется только наличие соединения и передаваемых данных.
4. Провести тестирование данного программного обеспечения для связи 4-х клиентов. При этом необходимо добавить 3 файла в разные клиенты и привести в отчете логии всех клиентов и сервера.
5. Модифицировать сервер с возможностью создавать группы клиентов. При присоединении клиента к серверу, сервер присылает список возможных файлов. При выборе клиентом определенного файла сервер присылает всех клиентов, которые присоединились к данному файлу. Клиент отправляет серверу сообщении, о желании подключения к файлу. В данном сообщении клиент сообщает серверу номер порта, открытого в состоянии listen. Клиенты делят большой текстовый файл на части и договариваются о том, кто будет передавать части текстового файла. У принимающей стороны хранятся части файла и он сам выбирает какую часть запросить у других клиентов. При этом логируется передача файлов.
6. Провести тестирование данного программного обеспечения для связи 4-х клиентов и передачи 3-х файлов.

## Лабораторная работа №12. Терминал на сокетах.

1. Разработать код, который представляет удаленный терминал. Данная программа при обнаружении ключевых слов “ls” и “cat” запускает соответствующие процессы. В случае ОС Windows можно использовать функцию createprocess ().При получении сообщения на остановку программы, посылает сигнал и останавливает запущенную программу.
2. Разработать программу сервер, которая открывает сокет в состоянии listen и программу клиент, которая подключается к этому сокету.
3. Каждая программ содержит терминал, предоставляющий доступ к консольному интерфейсу пользователя. В консольном интерфейсе вводятся команды. Командой setterminal можно выбрать к какому терминалу подключиться. Если произошло подключение к удаленному терминалу (например, на сервере подключение к терминалу клиента), то все команды терминала будут передаваться через сокетное соединение пункту назначения. Для переключения в каком терминале будут выполняться команды можно использовать специальные идентификаторы, например ip-адреса. Терминалы запускают код из пункта №1 и позволяют по сети, удаленно, запускать и завершать работу программного обеспечения. Обе программы-терминала сохраняют лог-файл, в котором содержатся команды, полученные, переданные и выполненные терминалом.
4. Запустить программы на двух компьютерах в сети. Запустить программы на основном и удаленном компьютере. По результатам программа должна запустить приложение, например браузер. В отчете привести лог-файлы.
5. Модифицировать программу, добавив возможность в терминале открыть терминальное соединение с другим устройством. Благодаря этой возможности первый терминал может установить удаленное соединение с клиентом №2 и уже в терминале клиента №2 открыть сокет для установления соединения с клиентом №3.
6. Запустить программы на 4-х компьютерах в сети и продемонстрировать последовательный запуск приложений на 3-х компьютерах с одного, основного. В отчете привести лог-файлы.

## Лабораторная работа №13. Web-сервер на сокетах

1. Разработать программу сервер **на сокетах**, которая принимает на вход запрос браузера. Web-сервер анализирует стартовую строку запроса, определяет какой файл необходимо выдать в ответ на запрос. Файл определяется по URL-пути. Ответом будет текстовая информация, HTTP-ответ, который содержит заголовки ответа (Server, Date, Content-Type, Connection, Content-Encoding) и содержимое файла.
2. Программа должна выдавать ответы 200, 404 и 5хх. Для одного URL-пути необходимо выдавать ошибку 302 или 301.
3. Запустить программу сервер и отобразить в браузере файлы: Текстовый, HTML, JSON, JPG и MP4. Результаты ответов привести в отчете.
4. Проанализировать запрос и ответ JSON файла с помощью сниффера Wireshark. Результаты анализа привести в отчете.
5. Модифицировать программу сервер. При первом обращении к web-серверу предложить браузеру сохранить cookie, в которых содержится id пользователя. При каждом подключении без передачи cookie увеличивать id. При получении cookie позволить перейти по URL-пути на «секретный» JSON файл.
6. Привести в отчете отправленные от сервера и полученные от клиента cookie с помощью Wireshark.
7. Для браузера Google Chrome не выдавать файлы с расширением MP4. Привести пример открывающегося файла из другого браузера.
8. Код итоговой версии программы привести в отчете.

## Лабораторная работа №14. Шифрованный web-сервер. SSH

1. Для выполнения лабораторной работы необходимо создать (если отсутствует) аккаунт на GitHub.com. Настоятельно рекомендую разобраться и применять данный сервис.
2. Подключиться к GitHub через протокол SSH при помощи ключа развертывания. Процесс настройки и подключения отобразить в отчете.
3. Разработать программу HTTPS-сервер. Программа открывает шифрованный сокет на порту 443 в состоянии listen. По запросу на данный сокет web-сервер отправляет HTTPS ответ (HTTP ответ + шифрование SSL/TLS). Ответ содержит заголовки ответа (Server, Date, Content-Type, Connection, Content-Encoding) и HTML страницу, выводящую Текущее время, ФИО студента и сообщение пользователям от автора. Для работы SSL/TLS необходим самоподписанный сертификат. Все подключения логируются на стороне сервера.
4. Подключить к серверу 3 клиента. В отчете привести пример подключения, логии сервера и код сервера. В отчете привести полученные клиентом сертификаты.
5. Получить бесплатное DNS имя. Подключить сертификат из сертификационного центра, например Let's Encrypt. В отчете описать процесс сертификации и с помощью Wireshark привести, полученные клиентом, сертификаты.

## Лабораторная работа №15. HTTP чат. Web-сокеты.

1. Разработать программу HTTP-сервер, которая принимает запрос GET от браузера и отправляет в ответ HTML страницу с полем чата, строкой ввода сообщения и кнопкой отправить. В этом HTML ответе передаются предыдущие 100 сообщений, которые отображаются в браузере пользователя. На стороне пользователя вводится информация в браузер, и, по нажатию кнопки, отправляется информация (POST запрос) о введенном сообщении. Это сообщение сохраняется на стороне сервера. Ответом на данный запрос будет новая HTML страница.
2. Провести тестирование данного программного обеспечения на 3-х клиентах и одном сервере. Программный код и результаты тестирования привести в отчете.
3. Доработать HTTP-сервер для общения при помощи web-сокетов. На стороне сервера необходимо предложить соединение через web-сокеты. Отправление данных от клиентов будет аналогичным, но при получении данных соединение не будет разорвано. Серверу необходимо сохранять все открытые соединения web-сокетов. При GET запросе от клиента, сервер отправляет HTML-страницу чата, JS файл для отображения сообщений и JSON файл, содержащий сообщения. По результатам получения POST запроса от клиента, web-сервер дополняет JSON файл (максимум 100 сообщений) и рассылает его всем клиентам.
4. Провести тестирование данного программного обеспечения на 3-х клиентах и одном сервере. Программный код и результаты тестирования привести в отчете.

## Лабораторная работа №16. Установка Web-сервера NGINX (начало)

1. Установить NGINX на компьютер, флеш-накопитель или виртуальную облачную машину. Рекомендуется использование ОС Linux (например Ubuntu), но возможна и установка ограниченной версии под ОС Windows. В отчете привести процесс установки (действия).
2. Взять web-страницу, содержащую текст HTML, CSS, JS и картинки в различных форматах. Запустить отображение этой страницы на стандартном порту 80 с применением web-сервера NGINX.
3. Добавить еще 3 страницы с переходами. Обновить файл конфигурации и перезапустить web-сервер. В файле конфигурации постараться использовать различные директивы. Обязательные дериктивы: worker\_process; worker\_connections; multi\_accept; charset; open\_file\_cache; Настройки буфферов; Конфигурация timeout; add\_header; expires; gzip; limit\_conn\_zone; limit\_req\_zone. Для новых 3-х страниц (только для них) добавить новый файл логирования ошибок. Для одной из страниц отключить логии (для примера). Для одной из страниц настроить редирект с кодом ответа 301 или 302. В отчете привести основные файлы логов и новый файл логов для 3-х страниц.
4. Настроить NGINX на запуск исполняемого файла на backend стороне. В отчете описать процесс настройки и запуска.
5. Обеспечить потоковую передачу видео на сайт с помощью модуля mp4. Добавить для доступа к видео требование ввода логина и пароля (на стороне сервера).
6. В отчете привести файл конфигурации.

## Лабораторная работа №17. Web-сервера, обратный прокси-сервер и балансировщик нагрузки NGINX (продолжение)

1. Установить NGINX на компьютер, флеш-накопитель или виртуальную облачную машину. Рекомендуется использование ОС Linux (например Ubuntu), но возможна и установка ограниченной версии под ОС Windows. В отчете привести процесс установки (действия).
2. Настроить сервер NGINX для передачи данных по протоколу HTTP/2.0. Отобразить в браузере передачу данных по HTTP/2.0. Необходимо создать ключ с помощью openssl. Выполнить подключение с самоподписанным сертификатом. При тестировании необходимо передавать web-страницу, содержащую HTML, CSS и JS файлы.
3. Получить бесплатное DNS имя. Получить сертификат от Let's Encrypt. Установить сертификаты с помощью certbot. В отчете описать процесс сертификации и с помощью Wireshark привести, полученные клиентом, сертификаты.
4. Настроить NGINX для проксирования ngnix.org. Настроить NGINX как обратный прокси-сервер для простейшего сервера на Node.js или PHP. Использовать заголовки add\_header и proxy\_set\_header. В отчете привести пример и файл конфигурации.
5. Настроить NGINX как балансировщик нагрузки для 3-х серверов, запущенных на различных портах с помощью upstream. Запустить 3 простых сервера Node.js или PHP. Показать работу балансировщика по методу RoundRobin. Показать работу балансировщика при отключении и подключении серверов.
6. Настроить NGINX как балансировщик нагрузки с Sticky Session. Настроить NGINX как балансировщик нагрузки по методу равномерной загрузки.

## Лабораторная работа №18. Установка Web-сервера Apache

1. Установить Apache на компьютер, флеш-накопитель или виртуальную облачную машину. Рекомендуется использование ОС Linux (например Ubuntu), но возможна и установка ограниченной версии под ОС Windows. В отчете привести процесс установки (действия).
2. Взять web-страницу, содержащую текст HTML, CSS, JS и картинки в различных форматах. Запустить отображение этой страницы на стандартном порту 80 с применением web-сервера Apache.
3. Добавить еще 3 страницы с переходами. Для одной из страниц настроить редирект с кодом ответа 301 или 302.
4. Создать виртуальные хосты на web-сервере Apache для различных доменов или поддоменов. Настроить каждый виртуальный хост для отображения разных сайтов или приложений.
5. Использовать модули Apache для улучшения производительности и безопасности сервера. Например, настроить модуль mod\_security для защиты от атак или модуль mod\_cache для кэширования ресурсов.
6. Настроить доступ к файлам и директориям на web-сервере Apache с помощью файла .htaccess. Реализовать различные правила доступа, например, запрет на доступ к определенным файлам или директориям.
7. Создать скрипты CGI (Common Gateway Interface) на языке Perl или Python и настроить их выполнение на web-сервере Apache. Продемонстрировать работу скриптов через браузер.
8. Настроить логирование доступа и ошибок на web-сервере Apache. Проанализировать логи для выявления информации о запросах, статусах ответов и возможных проблемах.
9. Реализовать аутентификацию пользователей на web-сервере Apache с использованием базы данных или файлов htpasswd. Обеспечить защиту доступа к определенным страницам или директориям с помощью пароля.
10. Настроить сжатие ресурсов (CSS, JS, HTML) на web-сервере Apache для уменьшения объема передаваемых данных и ускорения загрузки страниц. Сравнить скорость загрузки до и после применения сжатия.
11. Использовать модуль mod\_rewrite для настройки URL-адресов на web-сервере Apache. Создать правила перенаправления URL и преобразования ссылок для улучшения SEO и удобства пользователей.
12. Настроить кэширование ресурсов (изображения, CSS, JS) на web-сервере Apache с помощью модуля mod\_expires или других методов. Оценить влияние кэширования на скорость загрузки страниц и снижение нагрузки на сервер

## Лабораторная работа №19. Web-сервер на Node.js

1. Разработать программу HTTP-сервер с помощью Node.js. Для создания сервера подключить модуль http. Поднять сервер на порту 8080 в состоянии listen. Для сервера необходимо создать 4 HTML страницы (4 файла, необходимо использовать модули fs и path) с различным наполнением: Текст, Поле ввода, Таблица, Картинка. Картинка будет использоваться для отображения страницы с кодами 404 и 5xx. Оставшиеся 3 страницы будут выдаватьсяс кодом 200 при различных URL-путях. На каждой странице должен быть переход на следующую через HTML теги.
2. В отчете привести пример работы web-сервера и отображения всех страниц. (можно отдельные фрагменты)
3. Добавить файл стилей CSS.
4. В отчете привести пример работы web-сервера и отображения всех страниц. (можно отдельные фрагменты) и код всех файлов сервера. Также привести запросы и ответы из вкладки Network в браузере.
5. Реализовать обработку POST запросов на сервере Node.js для отправки данных с формы на одну из HTML страниц. Сохранить полученные данные в файл или базу данных и вывести сообщение об успешной отправке данных на экран.
6. Реализовать использование шаблонизатора (например, EJS или Handlebars) для упрощения создания HTML страниц на сервере Node.js. Создать шаблоны для всех страниц и использовать их при генерации контента для клиента.
7. Настроить логирование доступа и ошибок на сервере Node.js с использованием модуля morgan. Проанализировать логи для выявления информации о запросах, статусах ответов и возможных проблемах.
8. Реализовать аутентификацию пользователей на сервере Node.js с использованием Passport.js и стратегии локальной аутентификации. Обеспечить защиту доступа к определенным страницам или действиям с помощью аутентификации по паролю.
9. Использовать модуль Express.js для упрощения создания маршрутов на сервере Node.js. Создать REST API для работы с данными (например, CRUD операции) и протестировать его работу с помощью Postman или другого инструмента.
10. Реализовать кэширование ресурсов (изображения, CSS, JS) на сервере Node.js с помощью модуля node-cache или других методов. Оценить влияние кэширования на скорость загрузки страниц и снижение нагрузки на сервер.

## Лабораторная работа №20. Чат на web-сокетах.

1. Создать простой чат на web-сокетах. Реализовать возможность отправки и приема сообщений между пользователями.
2. Добавить функционал аутентификации пользователей в чате. Пользователь должен вводить свое имя перед присоединением к чату.
3. Реализовать возможность отправки приватных сообщений между пользователями. Сообщения должны быть видны только адресату.
4. Добавить функционал отправки файлов через чат. Реализовать возможность загрузки и отображения изображений, документов и других файлов.
5. Создать возможность создания комнат (чатов) в чате. Пользователи могут выбирать комнату для общения и видеть только сообщения из этой комнаты.
6. Добавить функционал отображения онлайн пользователей в чате. Пользователи должны видеть список участников чата и их статус (онлайн/офлайн)

## Лабораторная работа №21. Сервис с Rest API.

1. Создать простой REST API сервис для управления списком задач (To-Do list). Реализовать методы для создания, чтения, обновления и удаления задач.
2. Добавить функционал аутентификации пользователей в REST API сервисе. Пользователь должен получить токен доступа для выполнения операций с задачами.
3. Реализовать возможность фильтрации и сортировки задач в REST API сервисе. Добавить параметры запроса для выборки задач по определенным критериям.
4. Создать метод для экспорта данных в различные форматы (например, JSON, CSV) в REST API сервисе. Пользователь может получить данные в нужном формате для дальнейшей обработки.
5. Добавить возможность комментирования задач в REST API сервисе. Пользователь может оставлять комментарии к задачам и видеть комментарии других пользователей.

## Лабораторная работа №22. Сервис на базе SOAP.

1. Создать простой SOAP сервис для конвертации валют. Реализовать методы для получения курса обмена валют, конвертации суммы из одной валюты в другую.
2. Добавить функционал аутентификации пользователей в SOAP сервисе. Пользователь должен предоставить учетные данные для доступа к методам сервиса.
3. Создать метод для работы с базой данных через SOAP сервис. Пользователь может добавлять, обновлять, удалять записи в базе данных с помощью соответствующих методов.
4. Добавить функцию шифрования и дешифрования данных в SOAP сервисе. Пользователь может передавать зашифрованные данные через методы сервиса.
5. Реализовать функцию асинхронного выполнения задач в SOAP сервисе. Пользователь может отправлять запрос на выполнение задачи и получать результат в отдельном ответе.
6. Разработать функцию логирования действий пользователей в SOAP сервисе. Хранить информацию о запросах, ответах и других событиях для последующего анализа.
7. Создать метод для интеграции с внешними API через SOAP сервис. Пользователь может получать данные из сторонних сервисов и обрабатывать их в своем приложении.

## Лабораторная работа №23. Калькулятор (микросервисы) при помощи RPC.

1. Создать микросервис для сложения двух чисел. Реализовать метод для передачи двух чисел и получения результата сложения.
2. Добавить функционал в микросервис для вычитания чисел. Реализовать методы для передачи двух чисел и получения результата вычитания.
3. Разработать микросервис для умножения чисел. Реализовать методы для передачи двух чисел и получения результата умножения.
4. Создать микросервис для деления чисел. Реализовать методы для передачи двух чисел и получения результата деления.
5. Добавить возможность работы с дробными числами в микросервисах калькулятора. Реализовать методы для работы с дробными числами при выполнении арифметических операций.
6. Разработать функцию калькулятора для вычисления квадратного корня числа. Реализовать методы для передачи числа и получения результата вычисления квадратного корня.
7. Создать микросервис для вычисления процентов от числа. Реализовать методы для передачи числа и процентов и получения результата вычисления процентов.
8. Добавить функцию округления чисел в микросервисах калькулятора. Реализовать методы для передачи числа и количества знаков после запятой для округления.
9. Разработать микросервис для вычисления степени числа. Реализовать методы для передачи числа и степени и получения результата возведения в степень.
10. Создать метод для работы с последовательностью операций в микросервисе калькулятора. Реализовать возможность передачи списка операций (сложение, вычитание, умножение, деление) и получение результата последовательного выполнения операций.

## Лабораторная работа №24. Развертывание приложения при помощи Docker.

1. Создать Docker образ веб-приложения. Образ должен содержать все необходимые зависимости и файлы приложения.
2. Настроить Dockerfile для сборки образа веб-приложения, включая установку зависимостей, копирование файлов и запуск приложения.
3. Запустить контейнер с созданным Docker образом веб-приложения и проверить его работоспособность через браузер.
4. Развернуть базу данных (например, MySQL) в Docker контейнере и настроить веб-приложение на подключение к этой базе данных.
5. Создать Docker-compose файл для запуска веб-приложения и базы данных в отдельных контейнерах, а также настройки их взаимодействия.
6. Добавить возможность масштабирования веб-приложения путем запуска нескольких инстансов контейнера с помощью Docker-compose.
7. Настроить автоматичесное обновление Docker образа веб-приложения при обновлении кода из репозитория (например, GitHub).
8. Использовать Docker Swarm для развертывания кластера из нескольких узлов и размещения веб-приложения на этом кластере.
9. Настроить мониторинг и логирование контейнеров с помощью инструментов, таких как Prometheus и Grafana.
10. Запустить в Docker приложении сервер из задания №2, подключиться к нему 3-мя клиентами.

## Лабораторная работа №25. Изучение работы с хранилищем данных в браузере, создание React-приложения с использованием Redux Toolkit.

1. Добавить несколько объектов в localStorage с помощью js. Обновить страницу, затем открыть новую вкладку или браузер. В обоих случаях убедиться, что значения сохранились, нажав f12 и открыв Приложение → Локальное хранилище. С помощью цикла вывести все объекты, оставшиеся после обновления.
2. Аналогично пункту 1 добавить несколько объектов в sessionStorage с помощью js. Обновить страницу и проверить, сохранилось ли введённое сообщение, открыв вкладку Приложение → Хранилище сеанса. Повторить то же самое, перезапустив браузер.
3. Добавить в cookie один объект, указав 1 минуту в качестве времени жизни. Проверить, что данные успешно добавились, затем подождать указанное время и проверить содержимое cookie снова.
4. Через CRA (create-react-app) или любым другим способом установить React.
5. Создать контекст с помощью функции createContext, например, сохранить там никнейм пользователя. Создать компонент, который будет выводить эти данные на страницу и обернуть его в Provider. Вывести полученное значение, используя хук useContext. Сделать то же самое, переместив компонент, выводящий никнейм, на отдельную страницу.
6. Добавить к проекту redux-toolkit.
7. Сконфигурировать store (хранилище) и подключить его к приложению через тег Provider. В качестве наполнителя для хранилища создать слой, в котором с помощью reducer’a будет реализовано дополнение массива “Пользователи” данными, полученными из формы. Отобразить нового пользователя на той же странице.
8. Добавить еще один слой, который будет отправлять запрос на https://jsonplaceholder.typicode.com/users, обрабатывать полученные значения и выводить их на страницу. При написании слоя использовать createAsyncThunk.