inIT

Занятие 11

Введение в серверную разработку

План занятия

- Понятие сервера и клиента
- Коды ответов сервера
- Протокол HTTP
- MVC
- SOLID
- Git

Понятие сервера

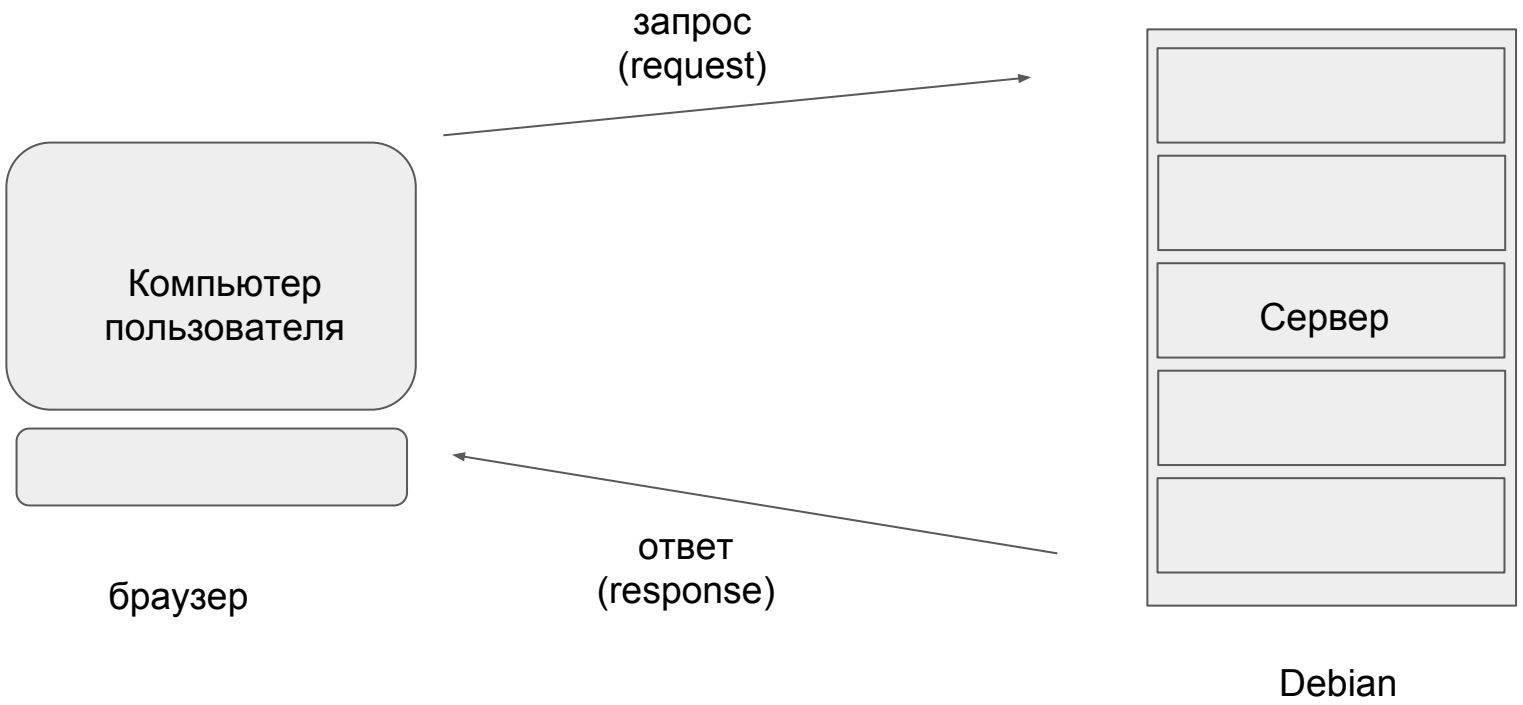
- У термина **сервер** есть два определения
- **Сервер** - это отдельное устройство, обычно выделенный и удалённый компьютер, на котором выполняются определённые программы в автоматическом режиме. Клиентские устройства подключаются к серверу удалённо и общаются через **протоколы передачи данных**
- **Сервер** - это специальное программное обеспечение, которое отвечает на запросы клиентского программного обеспечения и предоставляет данные или результат работы определённого функционала
- В основном своём большинстве на сервере работают UNIX-системы: Debian, Ubuntu Server, FreeBSD. Реже создаются сервера на Windows

Понятие клиента

- Понятие клиента, как и сервера, также разделяется на два направления
- **Клиент** - это компьютер пользователя, подключённый к локальной или глобальной сети и выполняющий запрос на сервер для получения данных и/или их передачи. Например, это ваш ноутбук, ПК, смартфон
- **Клиент** - это программа, через которую пользователь общается с сервером. Обычно это браузер, но может быть и любая другая программа (игры, приложения погоды, курсов валют, мессенджеры)

По-сути, сервер - это то, что ожидает запросы клиентов, поэтому возможна ситуация, когда одна и та же программа или одно и то же устройство в один момент времени или в разное время, являются как клиентом, так и сервером. Например, мы можем установить локальный сервер на свой компьютер и обращаться к нему со своего же браузера.

Клиент-серверное взаимодействие

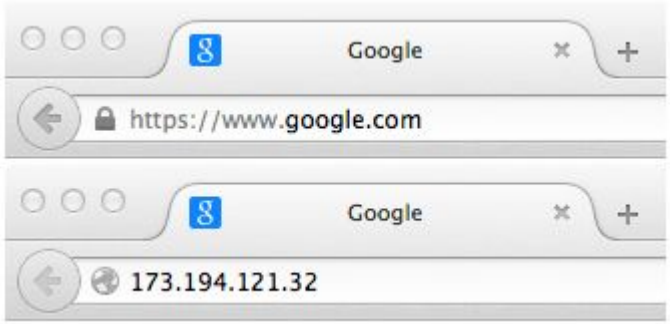


Запрос в адресной строке

inIT

При наборе адреса в адресной строке браузера происходит следующее:

- Браузер обращается к DNS серверу. Этот сервер сопоставляет строковое имя сайта с IP-адресом сервера, на котором этот сайт хранится и может предоставить его содержимое клиенту. DNS сервер сообщает браузеру реальное имя сайта (в виде набора цифр)



- Браузер отправляет HTTP-запрос серверу с просьбой предоставить копию сайта.
- Сервер отправляет код статуса. Если он равен 200, то отправляются файлы сайта в браузер (пакетные данные)
- Браузер из отдельных пакетов формирует отображение полноценной страницы

Коды ответов сервера

inIT

Это 3-значные числа, отправляемые сервером в ответе на запрос пользователя. Делятся на несколько классов

- 1xx - информационные. Информируют, что запрос принят и будет обработан. Пример - 100
- 2xx - коды успеха. Принятый запрос успешно обработан. Пример - 200 "ОК"
- 3xx - перенаправление (редирект). Необходимо сделать дополнительные действия со стороны клиента. Например выбрать один из форматов видео или обратиться по другому адресу, так как запрашиваемый ресурс перемещён. Браузер, обычно, всё это делает самостоятельно
- 4xx - ошибки пользователя. 404 - "не найдено", 400 - "bad request" (неверный запрос)
- 5xx - ошибки сервера

Протоколы передачи данных

inIT

Протоколы передачи данных - это особые правила, соглашения, устанавливающие, как общается программное обеспечения одного устройства с другим. Образно можно назвать это языком общения в сети.

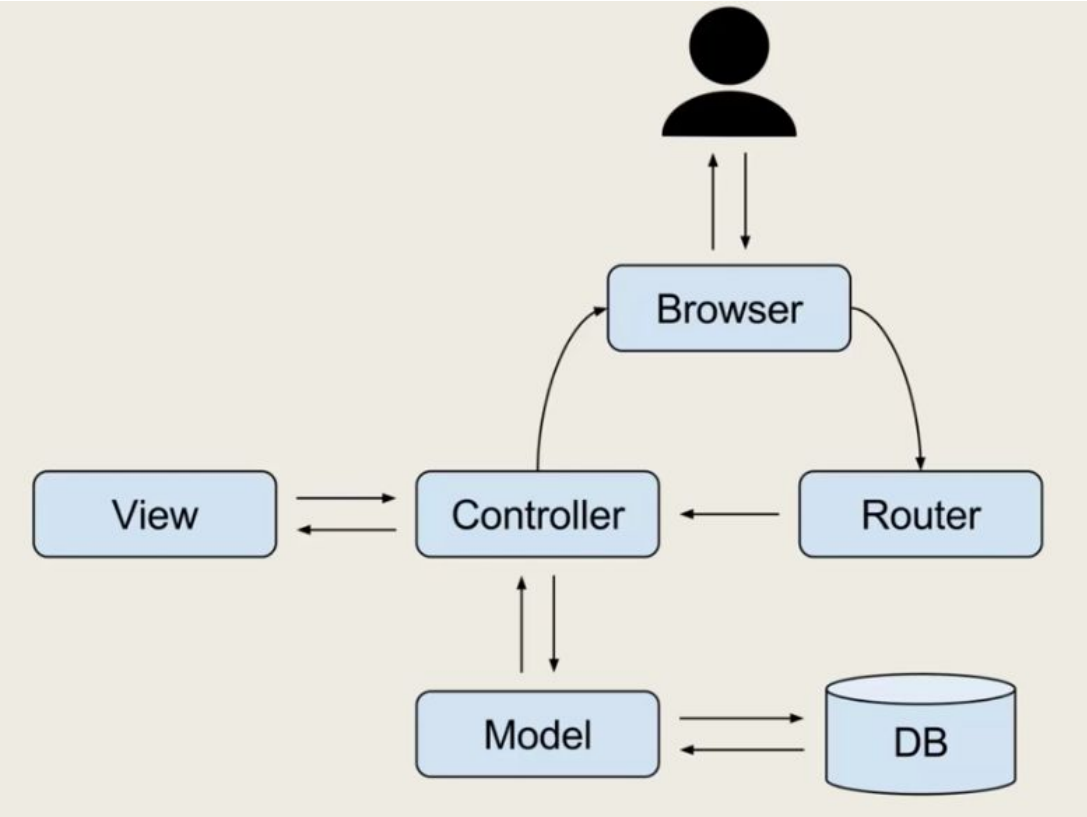
- TCP/IP - это стек технологий и протоколов различных уровней
- UDP - транспортный уровень. Обычно обеспечивает передачу данных в системах реального времени (DNS, связь), но не гарантирует порядок передачи пакетов, доставку и отсутствие дублирования
- FTP - прикладной уровень. Используется для передачи файлов с удалённого компьютера на пользовательский и наоборот
- HTTP - прикладной уровень. Применяется для получения данных с веб-сайтов

HTTP

inIT

- Самый популярный протокол для обмена данными в сети Интернет
- При использовании протокола соблюдается определённая структура передаваемых сообщений. Сообщения содержат 3 части:
  - Стартовая строка. В случае запроса здесь метод, путь к запрашиваемому документу и версия протокола. А в случае ответа версия, код состояния (код ответа сервера) и пояснение к коду
  - Заголовки. По типу ключ-значение, разделённые двоиточием
  - Тело сообщения (если есть).
- Протокол описывает **методы**, указывающие на операцию, производимую над запрашиваемым ресурсом. Методов около десятка, но основные из них, позволяющие выполнять так называемый CRUD (create, read, update, delete - распространённые манипуляции с данными), это:
  - GET - чтение
  - POST - запись
  - PUT - изменение
  - DELETE - удаление

MVC



Архитектурный шаблон, подход, проектирования программы, структурирования кода, управления и обмена данными между частями программы и с конечными пользователями.

Не является строгим предписанием, от чего интерпретируется и внедряется с отличиями

MVC

inIT

Подход предполагает разделение программы на 3 основных компонента:

- **Модель** (Model). Содержит бизнес-логику приложения, работает с базой данных
- **Представление**, вид (View). Отображает результат конечному пользователю, принимает входящие данные
- **Контроллер** (Controller). Обрабатывает запросы, обращается к модели за данными и отправляет их для отображения в представлении. Связующее звено модели и вида

При правильном проектировании модель ничего не знает о представлении, а оно не знает о модели. Взаимодействием занимается контроллер. Это делает модель универсальной, а представление перемещаемым в системы с другими моделями.

SOLID

inIT

Является набором принципов разработки в ООП, соблюдение которых необходимо для гибкости, лёгкости понимания и поддержки ПО

- **S - Принцип единой ответственности (single responsibility)**. У класса (сущности) должна быть одна причина для изменений. То есть класс должен выполнять один вид работы. Распространённый случай, когда мы в методах класса делаем какое-либо вычисление и в нём же выводим на экран (например в console.log). Если потом нам понадобится этот результат записывать в массив, базу данных, или вывести через alert(), то придётся переделывать класс или дополнять его логику, что не есть правильно. Более логично лишь возвращать результат в простом виде (массив, объект или просто значение), а вызвавший код сам определит, что делать дальше

SOLID

inIT

O - Принцип открытости/закрытости (open/close). Программные сущности (классы, модули, функции) должны быть открыты для расширения, но закрыты для модификации. Проблема в том, чтобы в новом проекте использовать повторно определённые участки предыдущего кода либо добавлять функционал к существующему. При нарушении принципа нам приходится изменять и старый код, а значит заново тестировать всё, опасаясь, что новые изменения нарушили работу исходников.

Для выполнения принципа активно применяется наследование - мы создаём новый класс, наследующий старый класс и уже в новом классе делаем дополнения, модификации, ничего не меняя в базовом классе.

SOLID

inIT

L - Принцип подстановки Барбары Лисков (the Liskov substitution principle). Если у нас есть базовый класс (тип) от которого наследуется производный, то функции, использующие базовый класс, должны иметь возможность использовать и производный даже не осознавая, что это не базовый класс. То есть класс-наследник должен предоставлять как минимум тот же функционал, что и его родительский класс.

SOLID

inIT

I - Принцип разделения интерфейса (the interface segregation principle). Вместо написания обобщённых компонентов, лучше писать конкретные реализации. Обобщённые реализации часто содержат функционал, который нам может не понадобится. Представьте, что у вас один огромный чемодан с вещами на все случаи жизни, а вы хотите выехать отдохнуть куда-то на пару дней и вам нужен минимум. Наверное, лучше купить сумку среднего размера или рюкзак и наполнять его только необходимым, чем брать с собой везде чемодан.

В случае программирования хороший пример - всплывающие окна. Можно сделать их универсальными, но тогда надо каждый раз думать, что нам нужна или не нужна та или иная кнопка, нужно или нет затенение экрана и так далее. Лучше разбить на компоненты и применять их по потребности

SOLID

inIT

D - Инверсия зависимостей (dependency inversion). 1) детали должны зависеть от абстракций, а не наоборот 2) модули верхних уровней не должны зависеть от модулей нижних - и те, и те должны зависеть от абстракций. То есть зависимости не должны быть инвертированы.

Для JavaScript, как динамического языка и лишённого абстрактных классов и интерфейсов, 1-й пункт не актуален, но 2-й важен.

Примеры использования SOLID для JS ограничены в виду особенностей языка, но хорошее пояснение можно посмотреть здесь https://www.youtube.com/watch?v=wi3wPzReKZQ

Version control (системы контроля версий)

inIT

При создании кода часто бывает необходимость попробовать внести какие-то изменения и, если они не подойдут, то вернуться к предыдущему коду. А может и к коду, который был год назад. Можно, конечно, комментировать участки кода, который хочется вернуть потом, но тогда наши файлы увеличатся в разы. К тому же, не всегда легко увидеть изменения. В больших проектах сложность растёт на несколько порядков.

А что делать, если над одним и тем же кодом работает множество программистов? Откуда брать внесённые изменения? Может просить коллег архивировать все файлы, а потом выкладывать на гугл.диск, а мы архив скачаем и заменим все файлы на файлы коллеги? Звучит ужасно. К счастью, есть решения - **системы контроля версий**.

Git

inIT

Наиболее популярной и удобной VCS является **Git**. Она используется на таких известных веб-хостингах для совместной разработки, как **github**, **bitbucket**, **gitlab**.

Основные преимущества:

- распределённость системы (все участники содержат весь код проекта локально, что ускоряет сохранение и чтение)
- сохранение разницы между старым и новым кодом, а не копии в полном виде
- удобная система ветвления, слияния, истории коммитов

Для установки используйте официальный сайт https://git-scm.com/ , где можно загрузить соответствующую версию программы

Git

inIT

Основные команды в терминале, для минимальной работы:

- git help (или просто git) - всегда подскажет доступные команды
- git init - создание нового **репозитория**. Папка, в которой запущена команда, обретает контроль версий (всех изменений)
- git status - проверка, есть ли файлы, содержащие изменения, которые не сохранены с помощью коммитов. **Коммит** - это фиксация изменения
- git add - если в нашем репозитории есть изменённые с прошлого коммита файлы, то команда позволяет их указать по имени, чтобы их индексировать и подготовить к последующему коммиту (git add . - для индексирования всех файлов, имеющих изменения)
- git commit -m 'message' - выполнить коммит и добавить сообщение с комментарием
- git pull - получить изменения с удалённого репозитория
- git push - отправить в удалённый репозиторий закоммиченный код