Основы web-технологий

Пишем тз на проект

Функционал

Авторизация

App level

#прочитали куки клиента -> идентифицировали

Создаваемый пользователем контент

#загрузка фото/оставить пост/…

Комментарии

#возможность user’а создать контент принадлежащий контенту другого user’а

Пользовательские действия «в одно нажатие»(лайк, дизлайк, …)(без отдельной формы)

Проверка прав пользователя

Готовим окружение

Верстка

Структура проекта

Frontend

#то что обрабатывается браузером

#быстрый веб-сервер умеющий только отдать статику и прокинуть запрос бекенду

Инфраструктура

Nginx

Gunicorn

#wsgi server

Backend

#exe то что не может front

Python

Django

DB

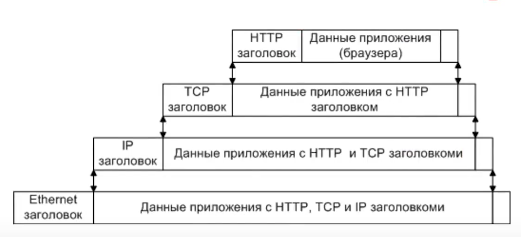
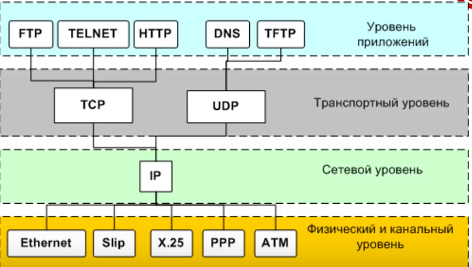
Mysql

Celery

#очередь задач

Redis

#сервер кеша

* СЕТЕВЫЕ ПРОТОКОЛЫ
* #появлялись исторически
* #соглашения о
  + Кодировании/декодировании сообщений
  + Назначение разных протоколов
  + Инкапсуляция протокола
  + 
  + Принципы работы
    - Dns
    - Ip
    - Tcp
  + HTTP
  + #∀ сообщение – str ⊃ данных в начале + заголовоковки с val + 2 переноса строки
    - Синтаксис
    - Возможности
  + HTTPS
  + IP
  + #протокол, соглашение о содержимом сообщения
  + ] мы заходим на сайт
  + #см инкапсуляция
    - В браузере переходим по ссылке
    - Браузер составляет http запрос и передает его oc
    - Ос по tcp протоколу перепаковывает
      * Берет http запрос (кусок текста в пару кило) добавляет нужные заголовки Tcp
        + Ip получателя
        + Порт получателя
        + Сообщение http становится телом Tcp
    - Ос|сетевуха заворачивает его в ip протокол
    - Сетевуха заворачивает его в ethernet протокол
    - Отправили по сети
    - Сервер получает это на сетевуху(которая занимается ethernet) etc в обратном порядке
    - В app попадет http запрос отправленный изначально
  + Разумеется мы не можем передавать большие куски данных одним пакетом(они бьются на куски)(т.к. потери)(таймауты)
    - ] мы грузим файл на AWS -> идет по локалке -> m8(москва) -> европа -> кабель в океане -> …
    - Потерянные куски отправляются повторно
* OSI
* #делит ∀ протоколы на 7 уровней
* TCP/IP
* #делит ∀ протоколы на 4 уровня(инкапсулируются друг в друга)
* 
  + Физический & канальный
  + #протоколы определяющие физические аспекты кодирования
  + Транспортный
  + #⊃ контроль доставки сообщений
* При разработке напр прошивки сетевой карты нужен только сетевой и физический уровни
* Ethernet
* #∀ комп сверяет его ли это пакет(по mac?)
* #неск подсетей
* TCP пакет/формат ip заголовка  
   

ОТЛИЧИЯ TCP ОТ IP

#разные уровни

#проверка доставки

Tcp

#контролирует доставку(при истечении таймаута переотправляет те пакеты о которых не поступило подтверждения)

#размер окна – контролирует скорость(число kb между ответами)

#разгружает отсновые устройтва вроде роутера

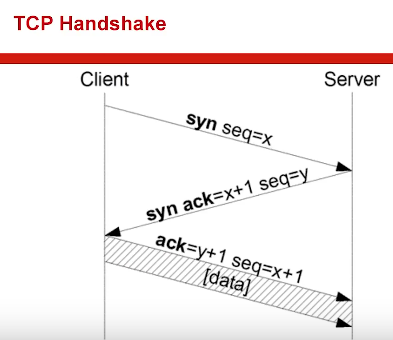
Ip протокол

#при разбиении мы не знаем дошли ли пакеты и в каком порядке

СОГЛАСОВАНИЕ В TCP

#тройное рукопожатие(схема первичного рукопожатия)

#вроде ~ connect()+accept()



#сначала договариваются о установлении сессии а затем передают данные

#изначальный x – рандомное число (не 0 тк неск сессий, и if бы пакет ∀ был нулевым – тогда можно было бы сделать оборудование ∀ отвечающее 1(и было бы не понятно получило оно пакет | пиздит)(страховка от ошибки)

#мы постоянно бомбим сервер пакетами а сервер их принимает и отвечает какой он принял последним(и это происходит в обе стороны с разными номерами пакетов)

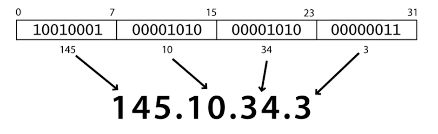
#x,y,... = номер пакета, размер окна, номер след пакета

#скажем изначальный номер пакета x, сервер дожен ответить с x+1 acknowledge number и дать свой sequence number

#контроль очередности доставки пакета(т.е. if ∀ пред пакеты доставлены – кидаем след пакет) (сервер отправляет только номер пакета для которого ∀ пред пакеты получены

Подсеть

ipV4



Следует читать в битовом представленин -> 31 уровень

#те адреса подсети варьируются в пределах 8бит(255)(1 роутер – обслуживающий подсеть)

Сообщение отправляется на уровень выше пока не найдется совпадения по подсети, затем снова отправляется вглубь

#на самом деле ∀ роутер ⊃ таблицу маршрутов, чтобы знать на какой порт отправить данные чтобы он дошел до нужного ip

NAT

#if на роутер извне придет запрос оно не будет знать куда его отправить(без предварительной настройки)(или может можно отправить ∀ чтобы каждый клиент разбирался сам нужно ли ему это msg)(снаружи не достучаться без белого ip)

#связывает свою подсеть с внешним миром, запоминая на какие адреса кто стучался что вернуть ответ на этого клиента

#в dnat это происходит с помощью разных портов (роутер подменяет порт отправителя в пакете и запоминает это и когда ответ возвращается с адресатом с портом скажем 3000 роутер может идентифицировать назначение и производит обратную подмену)

Обычно более 255 клиентов в одну подсеть не ставят

ПОРТ

2байта(~65000)

#идентифицирует назначение сообщения в пределах одного ip

80

#веб сервис

443

#ftp

1000+

#исп свободно

ТИПЫ IP-адресов

Публичные

#белый

#выдан ICANN

Частные

#192.168.0.0 – 192.168.255.255

#прячутся за nat

Localhost

#127.0.0.1

ICANN | IANA

#выдает ip

#некоммерческий фонд к которому общащаются ассоциации провайдеров за раздачей стека адресов которые провы делят между собой, отдают их более мелким провам, которые выдают их юзеров

Ipv6

#исп вроде 128 бит

#таблица маршрутизации больше

#при передаче ∀ маршрут должен поддерживать ipv6

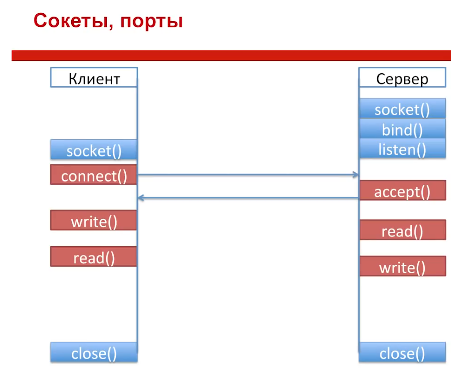
Ipv5

#была в виде черновика но не была реализована

УРОВЕНЬ ОС

#сокеты, порты

#сокеты беркли(стандарт)(видимо по названию вуза)



#bind() сервер резервирует сокет для себя (я хочу получать ∀ сообщения этого сокета)

#получение сокета – просто область mem для сетевого взаимодействия

#ос может создавать и слушать сокеты

СОКЕТ

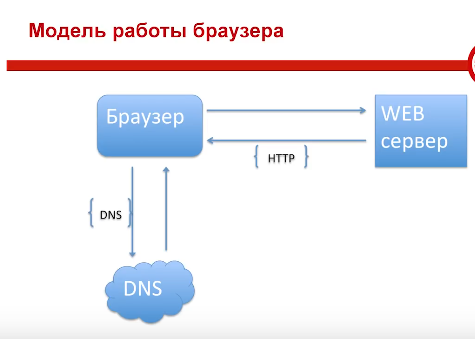
#IP + порт

Мы просим сокет у ос

Ос стучится в сетевуху проверяет может ли она обслужить запрос

#сокет в ос это просто буфер в ram куда она сваливает ∀ данные полученные по этому адресу и порту вовращая в приложение переменную с ссылокй на эту область mem

Файловый obj= файловый дескриптор



DNS

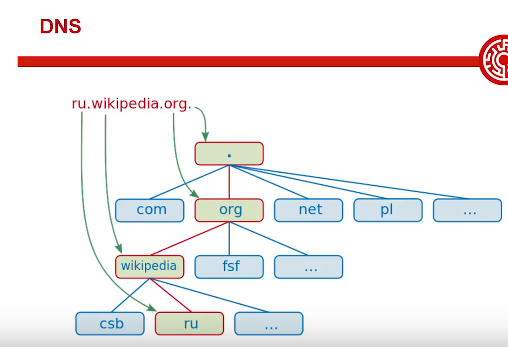
#~14 корневых

Dns протокол

#запрос ip по домену

#ограничивает max число вложенности поддоменов 255

ДЕРЕВО ДОМЕННЫХ ИМЕН



#дерево должно быть сбалансировано чтобы оно работало - ∀ пути корень-> листья примерно одинаковы(глубина левого и правого поддеревьев ~ друг другу) иначе производительность теряется(красно-черное дерево etc)(т.к. if одно ∀ левое поддерево вырождено – поиск превращается в поиск по связному списку что медленнее

ПЕРВЫЙ УРОВЕНЬ ОРГАНИЗАЦИИ

.

mail.ru

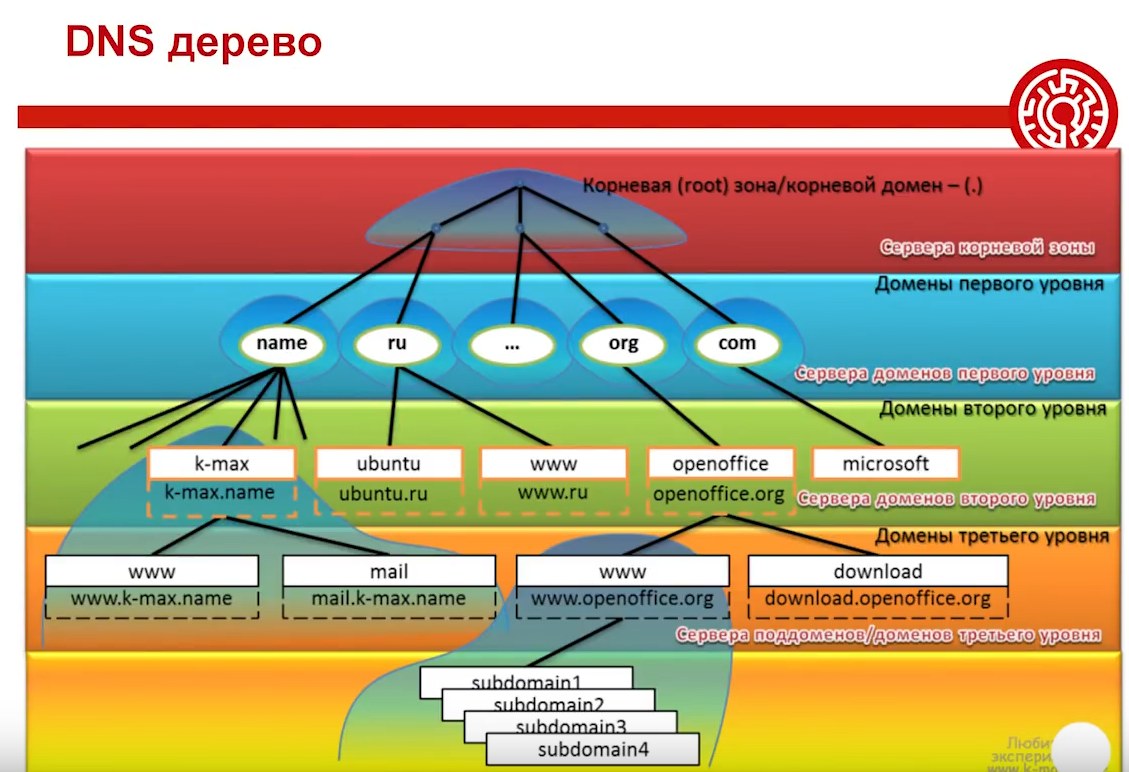
#Сокращенная запись

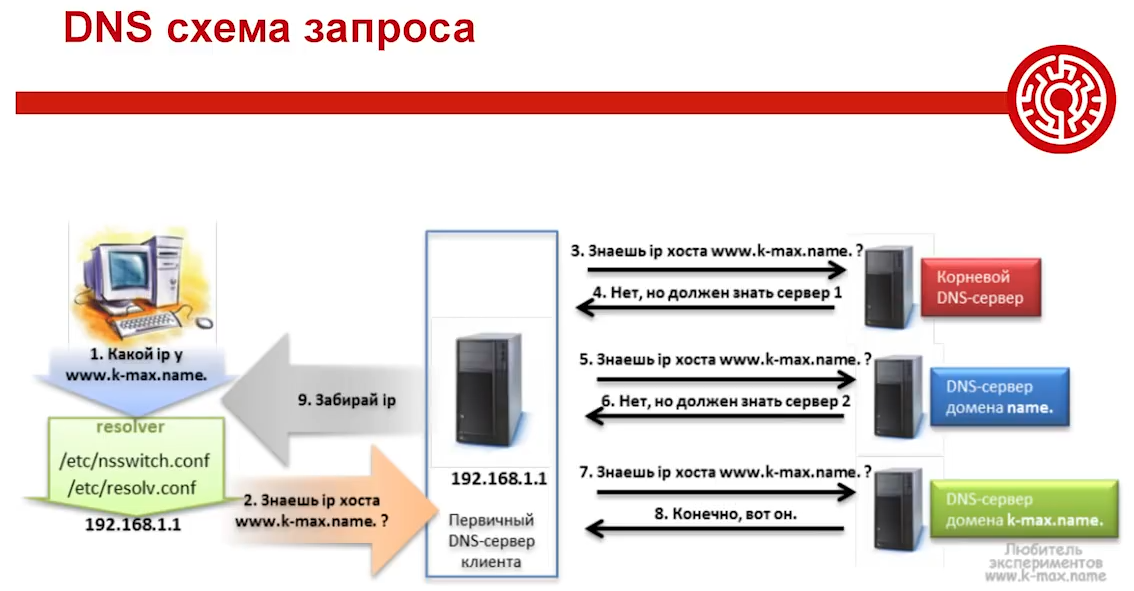
Mail.ru**.**

**#**точка в конце – правильная запись

Домен нельзя купить(только его обслуживание – на одном из коренвых dns появляется запись домен-ip) а остальные dns постепенно подтягивают их ~ в течение суток(по цепочке предоставления услуг(~ выдаче ip))(разделегирование это наоборот)

Добавление новой зоны вроде ru требует обновления записей ∀ коренвых dns(требует согласования с контролирующей организацией)





Dns запросы мб

#можно настраивать закие запросы будут рекурсивны, а какие нет(по идее на стороне сервера)(высоконагруженные серверы обычно не исп рекурсию)

Рекурсивные

Сервер сам пойдет искать нужный днс и ∀ след будет искать вглубь другого

#при нахождении вернет результат и сохранит в своем кеше(протокол реализован с целью минимизации запросов между серверами dns

#кеш устаревает ∀ пару часов

Не рекурсивыне

#изображены на картинке

**ОБРАТНЫЙ DNS ЗАПРОС**

#какие домены обслуживаются этим ip, а не какой у домена ip

