МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

Урок №1

Вводная лекция

на которой расскажут про курс, что такое программная инженерия, что такое интернет, чем интернет отличается от www, архитектуры современных веб-приложений, и расскажут про URL.

19 февраля 2025 года

Антон Кухтичев

Основные тезисы

- **Цель курса** — разработать современное и молодёжное веб-приложение;
- **Разработка** — индивидуальная, без команд;
  - Python 3.x и Django 5.x;
  - ~13 лекций и ~13 лабораторных работы;

- За каждую ЛР можно получить 10 баллов;
- За все ЛР можно получить 10 * ~13 = ~130 баллов.

Материалы будут лежать тут: https://github.com/toshunster/MAI-Backend

О лекторе

- Кухтичев Антон Алексеевич,
- Окончил МАИ в 2012 году,
- Работаю в Mail.Ru Group с 2011 года;
- Преподаю:
  - в МАИ с 2013 года принимаю лабораторные работы по курсу «Дискретный анализ»,
  - в 2018-2020 гг. в МФТИ вёл лекции по курсу «Backend разработка» (от ВК Образование),
  - в 2019-2020 гг. в МИФИ вёл лекции по курсу «Backend разработка» и «Углублённый Python» (от ВК Образование),
  - с 2020 года в МГУ веду лекции по курсу «Углублённый C++» (от ВК Образование),
  - с 2020 года в МАИ веду лекции по курсу «Программная инженерия» и «Информационный поиск».

Коротко о программе

1. Введение в веб, интенсив по Python;
2. Понятие HTTP, веб-сервера;
3. Django, Application Server;
4. Контейнеризация;
5. JSON, API, REST, RPC, gRPC;
6. Работа с СУБД;
7. Авторизация;
8. Работа с файлами;
9. Тестирование;
10. Real-time сообщения;
11. Микросервисная архитектура;
12. Поисковый движок, kafka, KV-хранилища (aerospike, redis) в веб-приложении;
13. Безопасность веб-приложений.

Шаблон курсового проекта

Веб-сервер (nginx)

KV-хранилище (Aerospike)

Сервер-приложения (django)

База данных (postgres)

Поиск (elasticsearch)

Облачное хранилище (S3)

Конвертер БД

Конвертер поиска

Конвертер S3

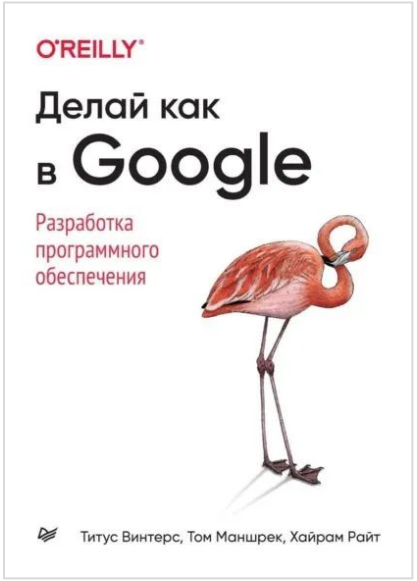
Брокер сообщений (kafka)

Обработчик$_1$

Обработчик$_2$

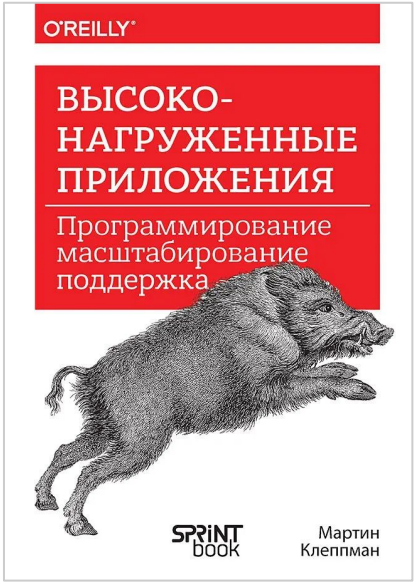
Обработчик$_n$

Полезная литература



Винтерс Титус,
Маншрек Том

Мартин
Роберт



Клеппман
Мартин

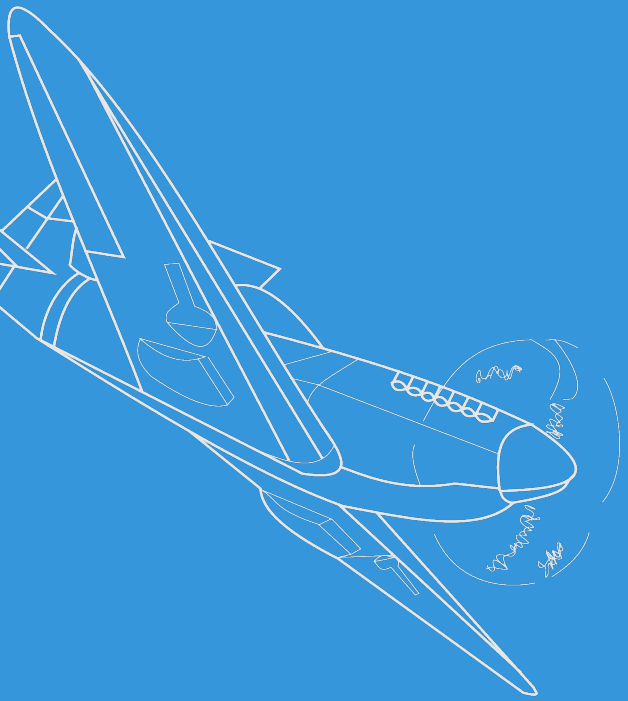
Дорожная карта разработчика

https://roadmap.sh/backend

Содержание занятия

1. Программная инженерия vs. программирование
2. Что такое интернет и www;
3. Архитектура современный веб-приложений;
4. Установка Ubuntu, Git, Python, Django;
5. URL;
6. Домашнее задание.


WELCOME
TO HE
INTERNET



Программная инженерия vs. программирование

Программная инженерия (1)

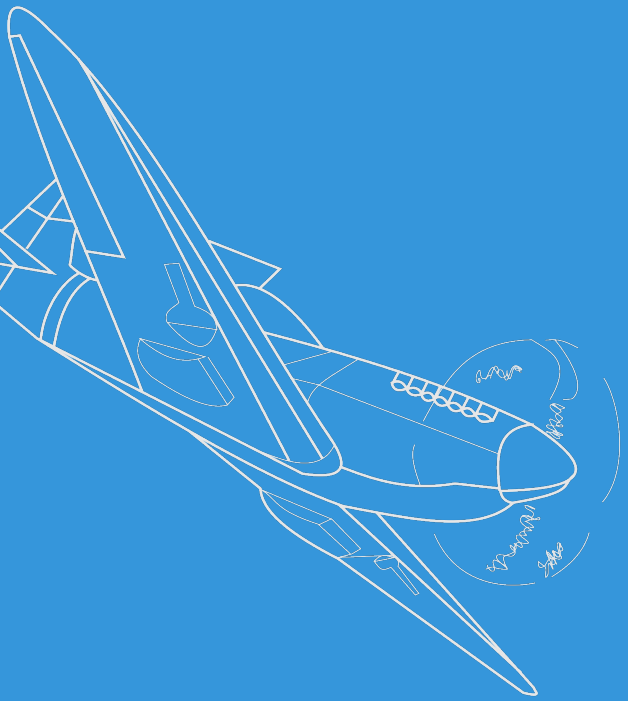
Программная инженерия — это область компьютерной науки, которая занимается построением программных систем, настолько больших и сложных, что для этого требуется участие команды разработчиков, порой даже нескольких команд.

Программирование — это дело, главным образом, индивидуальное, а программная инженерия — всегда коллективное.

Программная инженерия (2)

Есть три фундаментальных принципа, которые, следует учитывать при проектировании, разработке и написании кода:

- **Время и изменения**. Как код должен адаптироваться на протяжении срока действия.
- **Масштаб и рост**. Как организация должна адаптироваться по мере своего развития.
- **Компромиссы и издержки**. Как организация должна принимать решения, основываясь на показателях времени, изменений, масштаба и роста.



Интернет vs. WWW

Занимательный факт

«Правило: слово *интернет* пишется с маленькой буквы и склоняется по падежам»

— Артемий Лебедев. «Ководство», §55. Как писать слово интернет.

Интернет vs. WWW

Интернет — глобальная сеть передачи данных.

Протоколы:

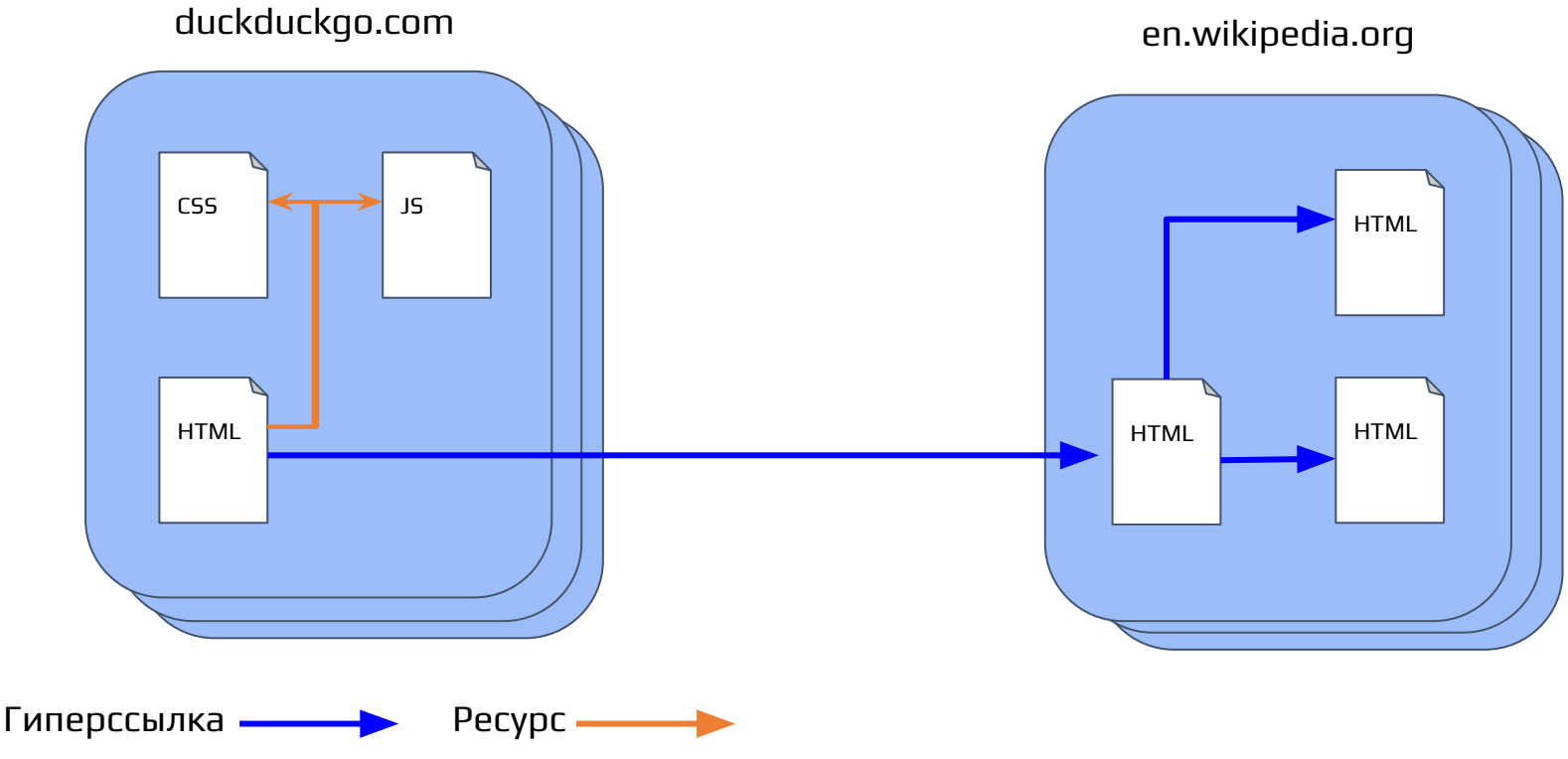
- HTTP, SSH, P2P — прикладные протоколы;
- DNS — доменная система имён;
- TCP — надежная последовательная передача данных;
- IP — глобальная адресация, передача в гетерогенной среде.

Интернет vs. WWW

WWW — множество взаимосвязанных документов, располагающихся на машинах, подключённых к интернету.

WWW — набор протоколов, серверного и клиентского ПО, позволяющих получать доступ к документам.

Интернет vs. WWW



Клиент-серверная архитектура

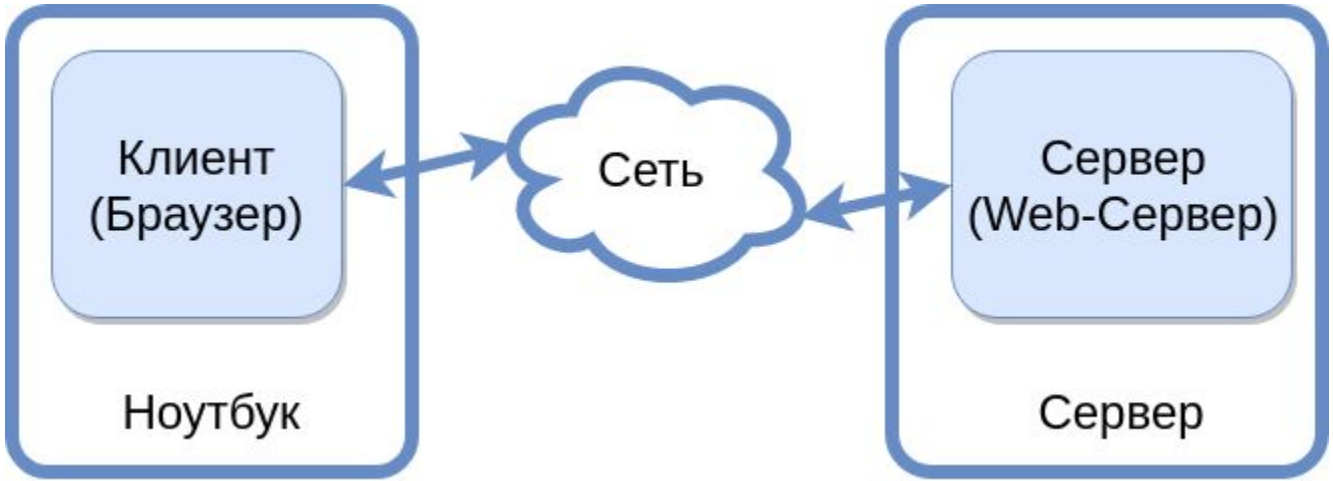
Веб-клиенты работают на компьютерах конечных пользователей. Задача веб-клиентов состоит в получении и отображении документов.

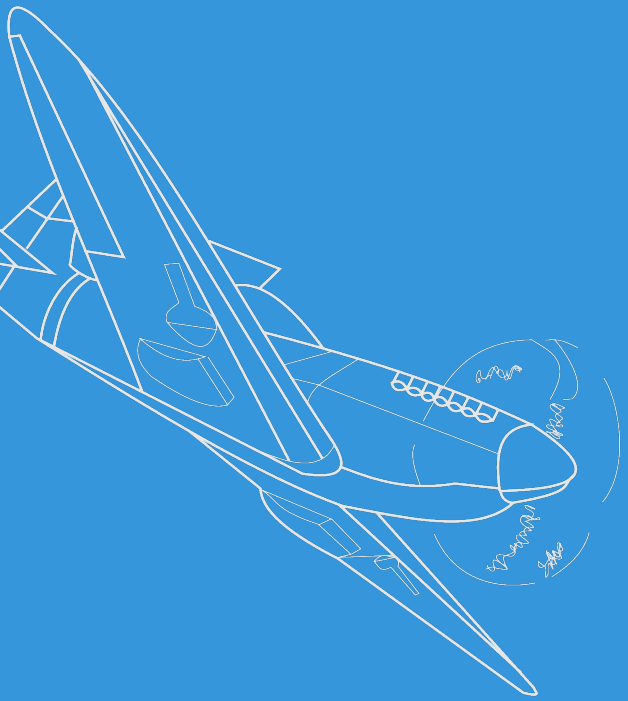
Веб-сервера работают (как правило) на серверах в датацентрах. Их задача заключается в хранении (или генерации) и отдачи документов.

Преимущества подхода

- Открытый протокол;
- Стандартный клиент;
- Прозрачный способ взаимодействия веб-приложений между собой;
- Распределённая и масштабируемая система.

Клиент-серверная архитектура





Веб-клиенты

Что такое веб-клиент. Разновидности веб-клиентов. Браузеры.

Разновидности web-клиентов

- Библиотеки в ЯП: `libcurl`, `urllib` и т.д.;
- Консольные утилиты: `wget`, `curl`, `telnet`;
- Роботы: поисковики, вредоносные скрипты;
- Браузеры:
  1. Полноценные: `firefox`, `chrome` и т.д.
  2. Встроенные: `web-view`, `webkit` и т.д.

Особенности библиотек веб-клиентов

- Предоставляют максимум опций для работы с HTTP;
- Осуществляют кодирование/декодирование данных;
- Перенаправления, куки - опционально;

Назначение: используются внутри других программ для простоты работы с HTTP.

Назначение консольных клиентов

- Автоматизация в shell-скриптах;
- Создание статической копии сайта;
- Отладка веб-приложений.

Пример отладки


```
# Простейший GET-запрос
curl -v 'https://python.org/'

# POST-запрос
curl -v -d POST -L -H 'User-agent: curl' 'https://python.org/'
```

- `-v`, `--verbose` — подробный вывод;
- `-d`, `--data` — POST-запрос;
- `-L`, `--location` — если redirect, то следуй по новому URL;
- `-H`, `--header` — установить заголовок.

Браузер

Браузер — это программа с графическим интерфейсом, которая позволяет отображать html-документы.

Основное назначение браузера — отображение HTML страниц.

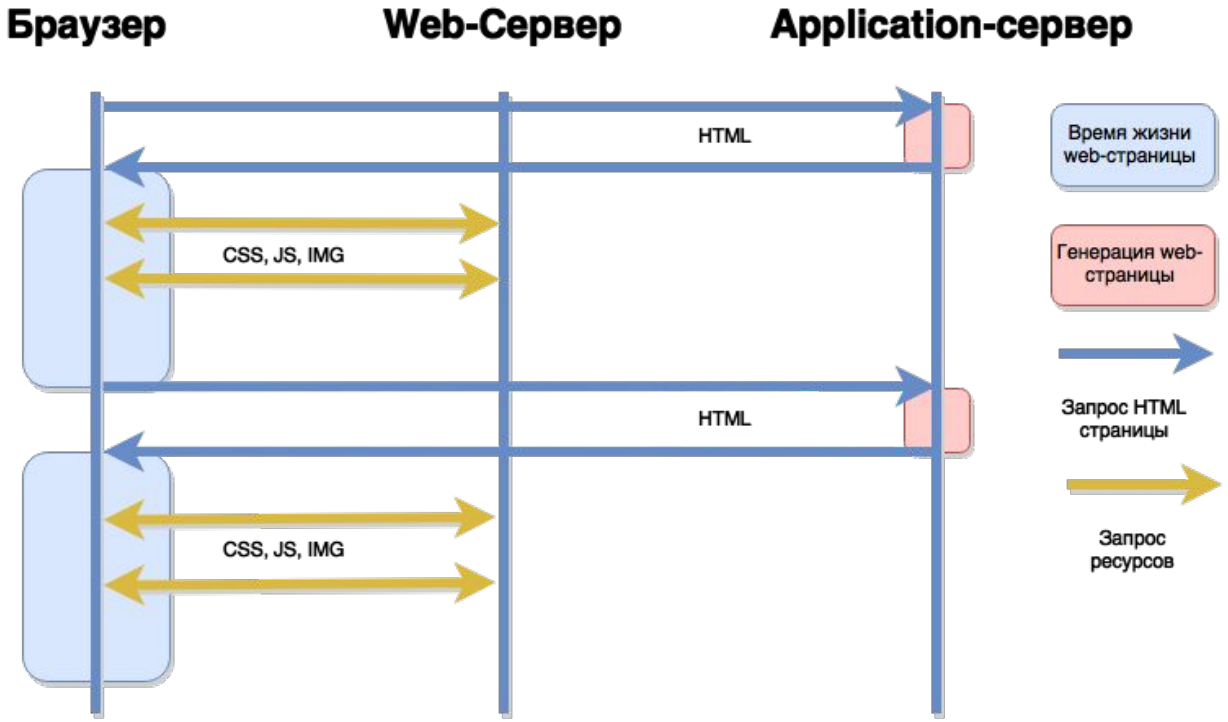
Однако, возможности современных браузеров огромны.
Существуют операционные системы и 3D-игры, работающие внутри браузеров!

www.evolutionoftheweb.com

Сценарий работы классического веб-приложения

- Пользователь вводит URL;
- Браузер загружает Web страницу — HTML документ;
- Браузер анализирует (parse) HTML и загружает доп. ресурсы;
- Браузер отображает (rendering) HTML страницу;
- Пользователь переходит по гиперссылке или отправляет форму;
- Цикл повторяется.

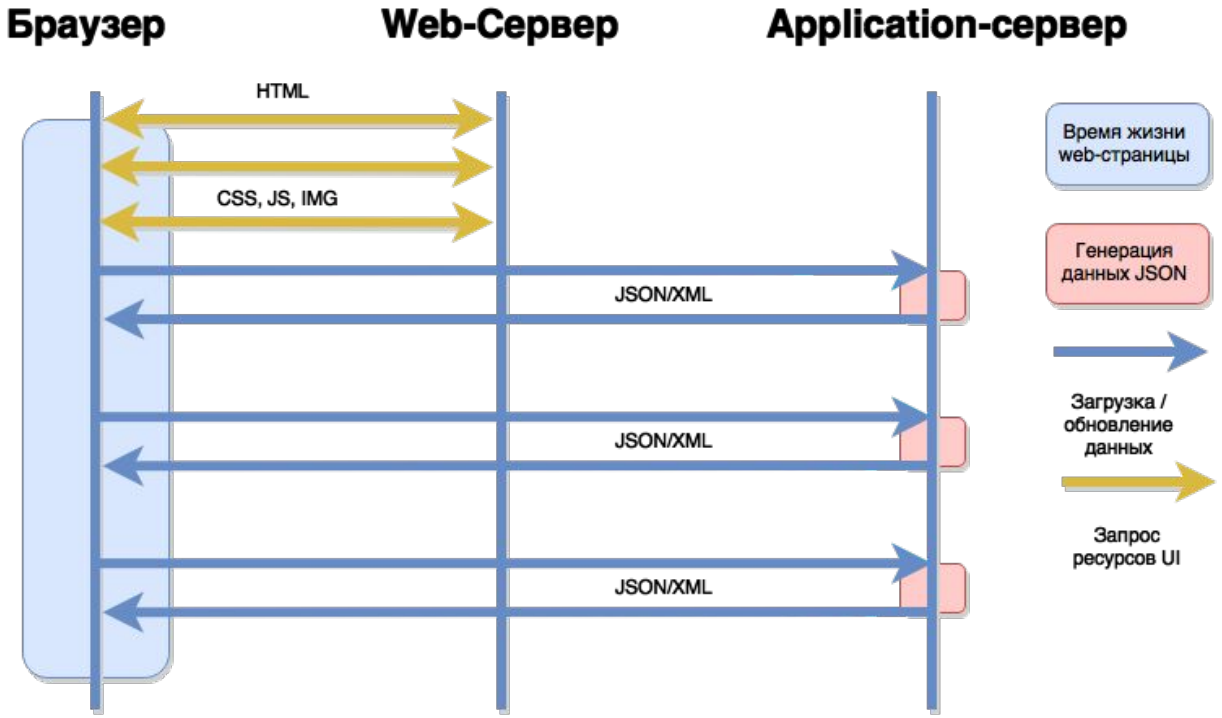
Сценарий работы классического веб-приложения



Сценарий работы современного веб-приложения

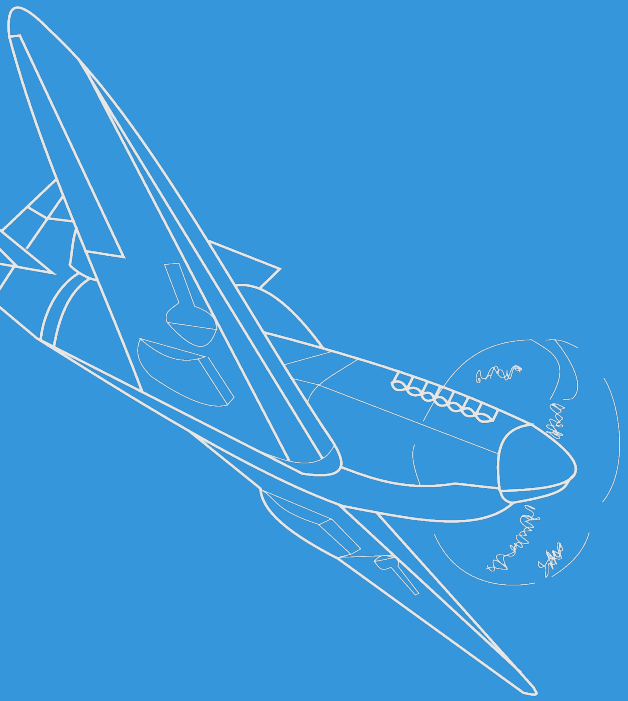
- Браузер загружает Web страницу, ресурсы и отображает её;
- JavaScript загружает данные с помощью AJAX запросов;
- JavaScript обеспечивает полноценный UI на странице;
- Пользователь взаимодействует с UI, что приводит к вызову JavaScript обработчиков;
- JavaScript обновляет данные на сервере или загружает новые данные, используя AJAX.

Сценарий работы современного веб-приложения



Особенности современных веб-приложений

- UI находится на 1 или нескольких страницах (single page);
- UI полностью статичен: HTML, CSS, JS — статические файлы;
- Логика UI полностью работает на стороне клиента;
- Используется шаблонизация в JavaScript;
- Application сервер возвращает чистые данные (JSON или XML, а не HTML).

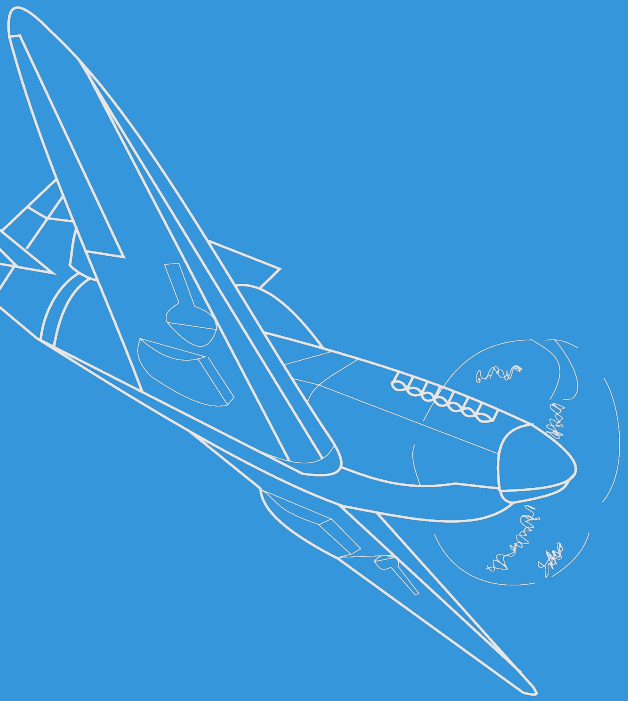


Документы

Документы

Документы могут быть:

- Статические
    - Это файлы на дисках сервера;
    - Как правило, обладают постоянным адресом.
- Динамические
    - Создаются на каждый запрос;
    - Содержимое зависит от времени и пользователя;
    - Адрес может быть постоянным или меняться.



URI, URL, URN

Определения

- **URI** — Uniform Resource Identifier (унифицированный идентификатор ресурса);
- **URL** — Uniform Resource Locator (унифицированный локатор/указатель ресурса);
- **URN** — Uniform Resource Name (унифицированное имя ресурса).

URI является либо URL, либо URN, либо одновременно обоими.

URN — uniform resource name

`urn:<NID>:<NSS>`

- `<NID>` — идентификатор пространства имён;
- `<NSS>` — строка из определённого пространства имён.

Пример:

- `urn:isbn:5170224575` — URN книги, идентифицируемой номером ISBN;

URL — uniform resource locator

`<схема>://[[<логин>[:<пароль>]@]<хост>[:<порт>]][/<URL - путь>][?<параметры>][#<якорь>]`

http://server.org:8080/path/doc.html?a=1&b=2#part1

- `http` — протокол;
- `server.org` — DNS имя сервера (может указываться ip-адрес машины);
- `8080` — TCP порт;
- `/path/doc.html` — путь к файлу;
- `a=1&b=2` — параметры запроса;
- `part1` — якорь, положение на странице.

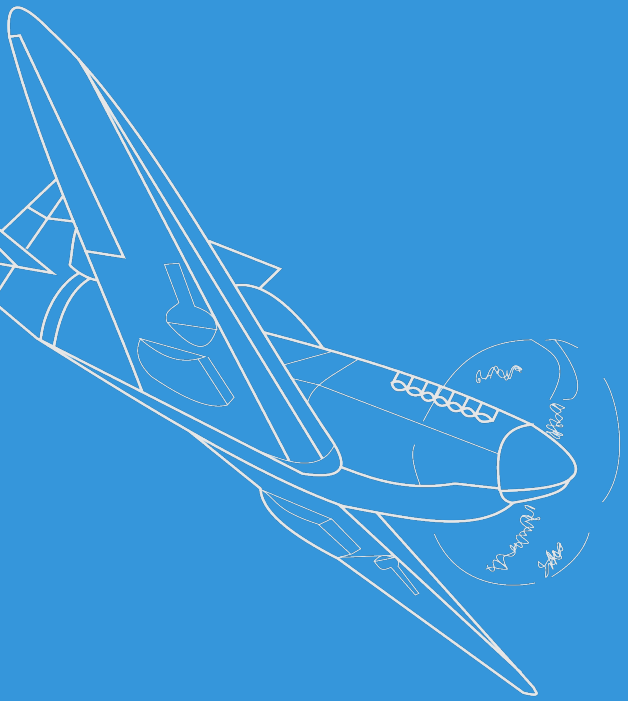
Абсолютные и относительные URL

- `http://server.org/1.html` — абсолютный;
- `//server.org/1.html` — абсолютный (schemeless);
- `/another/page.html?a=1` — относительный (в пределах домена);
- `pictures/cat.png` — относительный (от URL текущего документа);
- `?a=1&b=2` — относительный (от URL текущего документа);
- `#part2` — относительный (в пределах текущего документа);

Правила разрешения URL

`https://site.com/path/page.html` — основной документ

- `http://wikipedia.org` = `http://wikipedia.org`
- `//cdn.org/jquery.js` = `https://cdn.org/jquery.js`
- `/admin/index.html` = `https://site.com/admin/index.html`
- `another.html` = `https://site.com/path/another.html`
- `?full=1` = `https://site.com/path/page.html?full=1`
- `#chapter2` = `https://site.com/path/page.html#chaprer2`



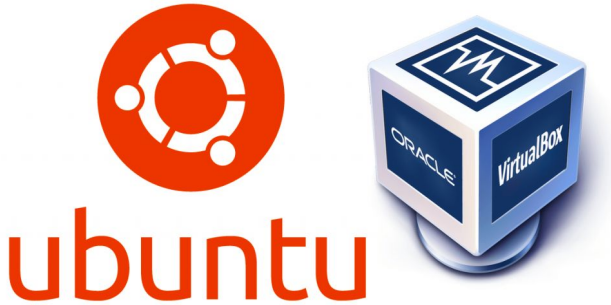
Установка рабочей среды

Как установить Ubuntu, Python, работать с git, устанавливать библиотеки.

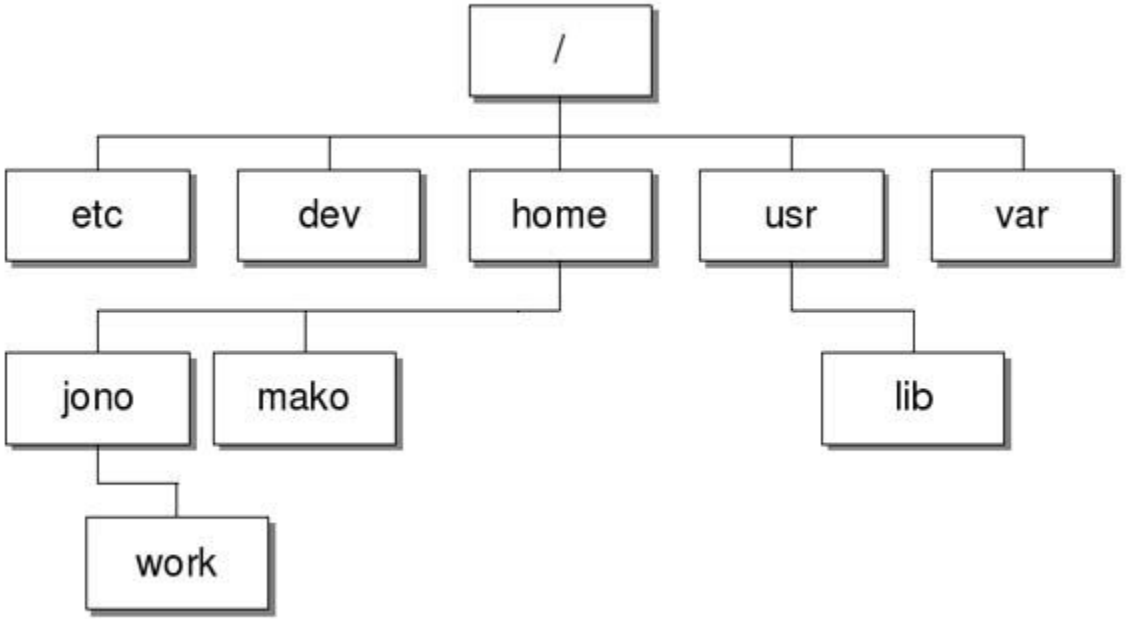
Устанавливаем Ubuntu

- Скачиваем VirtualBox
  https://www.virtualbox.org/wiki/Downloads

- Скачиваем образ Ubuntu
  https://www.ubuntu.com/download/desktop
  ...
- Profit!



Структура директорий Linux



Установка программ в Linux


```
1. sudo apt install git    # установить известную программу
2. sudo apt search Django  # поискать среди доступных
3. sudo apt remove git     # удалить программу
4. sudo apt purge nginx    # удалить программу и все её файлы
```

apt — пакетный менеджер Ubuntu (yum, pacman, emerge...)

sudo — временное повышение привилегий до root

В репозиториях ОС обычно не самые свежие программы.

Основные утилиты в Linux

`cd` — поменять директорию (change directory);

`ls` — вывести список файлов;

`mkdir` — создать папку;

`pwd` — абсолютный путь до текущей директории;

`rm` — удалить файл;

`cp` — копировать файл;

`mv` — переместить;

`tree` — содержимое текущей директории в виде дерева;

`man` — документация.

Работа с Git


```
1. git init # создать новый репозиторий
   # или
2. git clone git@github.com:nuf/quack.git # склонировать
3. git status # посмотреть статус файлов
4. git add some_file some_dir # добавить файлы в индекс
5. git commit # сформировать новый коммит
6. git push # отправить изменения в github
7. git pull # получить последние изменения
```

Работа с Git. Как назвать репозиторий?

Шаблон имени:

`YYYY-MAI-Backend-N-LAST_NAME`

- `YYYY` — текущий год
- `N` — первая буква имени
- `LAST_NAME` — фамилия

Пример: `2025-MAI-Backend-A-Kukhtichev`

Содержимое .gitignore


```
1.  venv/
2.  *.pyc
3.  *.swo
4.  *.swp
```

Установка библиотек Python (1)

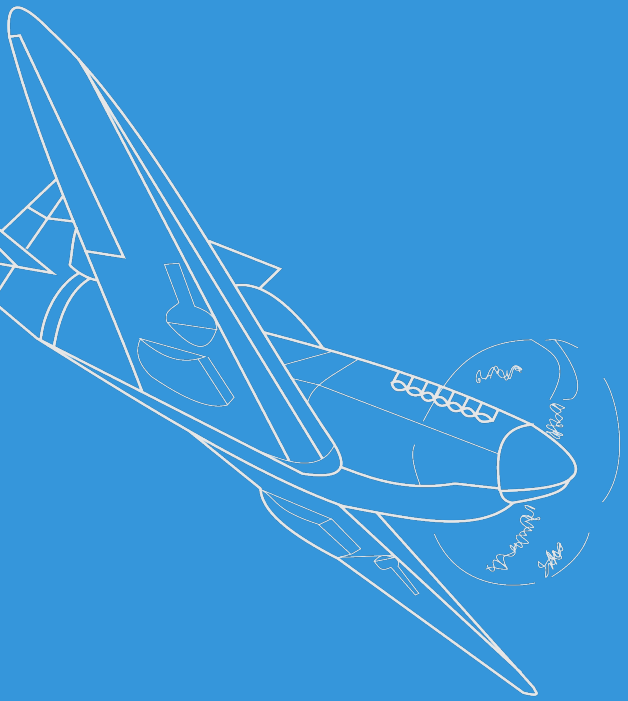

```
1. # Устанавливаем pip3
2. sudo apt install python3-pip
3. # А затем уже библиотеки Python
4. sudo pip3 install Django==5 # в систему
5. pip3 install --user py.test # или только для себя
6. # Или ...
```

Использование виртуального окружения


```
1.  # Переходим в директорию проекта
2.  cd /home/nuf/quack
3.  # Возможно потребуется сделать ещё вот это вот
4.  sudo apt-get install python3-venv
5.  # Создаем виртуальное окружение в директории venv
6.  python3 -m venv venv
7.  # "Активируем" его
8.  source ./venv/bin/activate
```

Установка библиотек Python (2)


```
# Устанавливаем необходимые библиотеки в venv
pip install Django==5 pytest psycopg2
# "Запоминаем" версии установленных библиотек
pip freeze > requirements.txt
```



Интенсив по Python

Краткий экскурс, синтаксис, типы, структура программы.

Python

Интерпретируемый язык с динамической типизацией и автоматической сборкой мусора.

Python - название спецификации языка

Реализации:

- CPython
- IronPython (DotNet)
- PyPy

https://github.com/python/cpython

https://www.python.org/doc/

Гвидо Ван Россум
(создатель языка Python)

Начинаем программировать >>>


```
python
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017,20:11:06)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
on darwin
Type "help", "copyright", "credits" or
"license" for more information.
>>> 2 * 3
6
>>> help(print)
>>> exit()
# Запуск скрипта
python file_name.py
```

Переменные


```
num = 1 # операция присваивания переменной
        # num значения 1
# допустимые символы: буквы, цифры, _
# НО: начинается с буквы или _

# допустимо:
num = 1
_num = 1
__num = 1
num35num35 = 1
first_num = 1

# неверно:
1num = 1
```

Базовые типы и конструкции

Целые числа (int)

```
num = 42
num = 42_000_000 # начиная с python 3.6
>>> print(type(num))
<class 'int'>
```

Вещественные числа (float)

```
float_num = 3.14
float_num = 3.14e2 # 3.14 умножить на 10 в степени 2
```

Конвертация типов

```
float_num = float(num)
```

Базовые типы и конструкции

Комплексные числа (complex)

```
num = 14 + 1j # num.real, num.imag - для доступа
              # до реальной и мнимой частей
```

Основные операции с числами

```
+ - * / ** % //
>>> 6 * 6.0
36.0
>>> 36 / 6
6.0 # результат деления всегда вещественный
>>> 8 / 0 # ZeroDivisionError
# % - остаток от деления
# ** - возведение в степень
# // - целочисленное деление
```

Базовые типы и конструкции

Строки

```
some_str = 'Valid'
some_str2 = "Valid too"
some_str3 = """Valid as well"""
some_raw_str = r'Valid raw str'
```

Операции со строками

```
'Hello' + ' world'
'hello ' * 3
hello[1:4] # 'ell'
hello'[::-1] # 'olleh'
len('hello') # 5
```

Базовые типы и конструкции

Форматирование строк

```
>>>'%s do not like this method' % ('I')
'I do not like this method'
>>>'{} method is much {}'.format('This', 'better')
'This method is much better'
>>>'{name} likes {what} most of all'.format(
  name='Anton',
  what='this method'
)
#f-строки >= python 3.6
attribute = 'new'
>>> f'This method is {attribute} in Python 3.6'
```

Конструкции управления потоком

If … elif … else

```
welcome_str = 'Hello, System, this is Nick'
if 'Nick' in welcome_str:
    print('This is Nick')
elif 'Mary' in welcome_srt:
    print('This is Mary')
else:
    print('Unknown person detected!')

# аналог тернарного оператора
person_name = None # объект типа NoneType
test_text = person_name if person_name else 'Name'
```

Конструкции управления потоком

while

```
i = 0
while i < 100:
    i += 1
print(i) # 100
```

range

```
for i in range(3):
    print(i) # 0 1 2

pass # определяет пустой блок
break # выход из цикла досрочно
continue # перейти к следующей итерации цикла
```

Коллекции

Списки

```
empty_list = []
empty_list = list()
none_list = [None] * 10
user_list = [['Anton', 5.5], ['Gennady', 6.9]]
len(user_list)
user_list[0]
user_list[0:]
new_user_list = user_list[:] # скопирует список в новый
user_list.append(['Алёна', 3.2])
user_list.extend([['Valentin', 5.7]])
del user_list[2]
```

Коллекции

Кортежи - неизменяемые списки

```
empty_tuple = ()
empty_tuply = tuple()
immutables = ('int', 'str', 'tuple')
```

Кортежи хэшируемы и могут использоваться в качестве ключей словаря

Коллекции

Словари

```
empty_dict = {}
empty_dict = dict()
teachers = {
    'Math': 'Kate Johnson',
    'English': 'David Lewis'
}
teachers['Math'] # Kate Johnson
teachers.get('Math','Unknown')
teachers['Chemistry'] = 'Julia White'
teachers.update({'Chemistry': 'Nolan Black'})
del teachers['Math']
'Chemistry' in teachers # True
for i in teachers:
    print(i) # итерируемся по ключам словаря
```

Функции

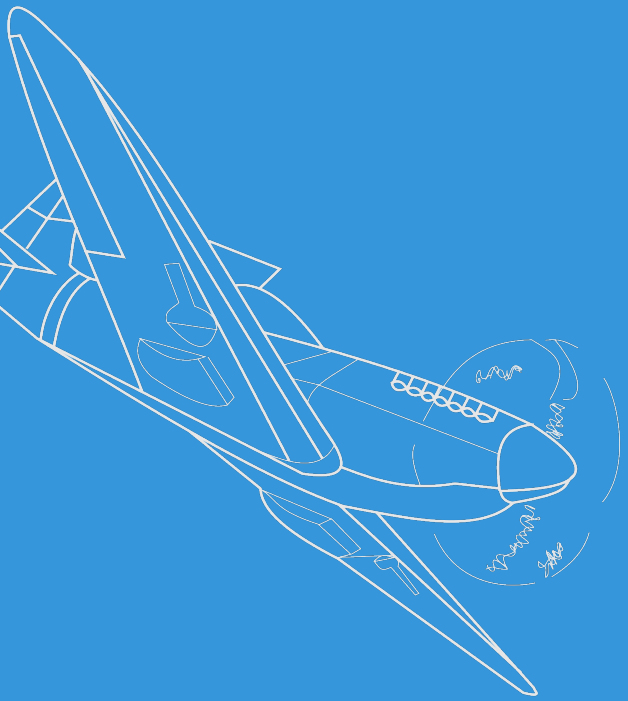
```
def multiply(a, b):
    '''Multiply a by b'''
     return a * b
multiply.__doc__ - docstring для функции
```

Классы


```
isinstance(3, int) # проверка на принадлежность классу

# объявляем пустой класс
class Human(object):
    pass

print(dir(Human)) # методы

class Planet(object):
  count = 0 # экзепляр класса
  def __init__(self, name):
      self.name = name # атрибут экземпляра
  def __str__(self):
     return self.name

planet = Planet('Earth') # создаем экземпляр класса
```

Побудительный пример


```
>>> import requests
>>>
>>> data = {
...     'name1' : 'value1',
...     'name2': 'value2',
... }
>>> headers = {
...     'cache-control': "no-cache",
... }
>>> url = ‘https://python.org/’
>>> response = requests.get(url, headers=headers, data=data)
>>> print(response.text[:10]) # первые 10 символов ответа.
```



Домашняя работа

Домашнее задание №1

1. Завести репозиторий на github, установить Python (>=3.6) — 2 балла;
2. Создать виртуальное окружение для Python и установить Django (4.0) — 1 балл;
3. Описать зависимости в requirements.txt — 1 балл;
4. Создать правильный .gitignore файл и оформить изменения в виде отдельных осмысленных коммитов — 1 балл;
5. Написать реализацию LRU-cache на языке Python — 5 баллов.

Домашнее задание №1


```
# cache.py

class LRUCache:
    def __init__(self, capacity: int=10) -> None:
        pass

    def get(self, key: str) -> str:
        pass

    def set(self, key: str, value: str) -> None:
        pass

    def rem(self, key: str) -> None:
        pass
```

Домашнее задание №1


```python
from cache import LRUCache

cache = LRUCache(100)
cache.set('Jesse', 'Pinkman')
cache.set('Walter', 'White')
cache.set('Jesse', 'James')
cache.get('Jesse') # вернёт 'James'
cache.rem('Walter')
cache.get('Walter') # вернёт ''
```

Рекомендуемая литература

Инженерия программного обеспечения | Иан Соммервилл
DNS и BIND | Ли Крикет, Альбитц Пол
Виртуальное окружение
Github
Virtual Box
Тут берём LTS Ubuntu (например 20.04)

Для саморазвития (опционально)
Чтобы не набирать двумя пальчиками

Спасибо за внимание!

Антон Кухтичев

a.kukhtichev@mail.ru

@toshunster