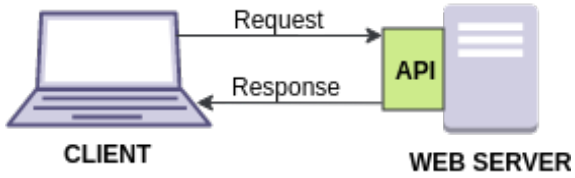
Web сервисы, фреймворки, API

Белов Виталий, весна 2021

API

Application Programming Interface

Это способ коммуникации програмных компонентов



- Внутренние API (собственных библиотек) - для коммуникации микросервисов внутри приложения/компании
- Внешние API (веб-сервисов) - позволяют получить доступ к сервису сторонним разработчикам через интернет, используя HTTP или другие протоколы

Как согласовать различные API? - *REST* и *RPC*

REST

- Как расшифровывается? - **RE**presentational **S**tate **T**ransfer
- Что это? - Набор правил, которые позволяют разного рода системам обмениваться данными и масштабировать приложение. Соответствие этим правилам - нестрогое
- Как? - Используется HTTP протокол - прикладной уровень обмена данными по сети

Если при проектировании правила REST соблюдены - такой протокол называется *RESTful*

RESTful API позволяет производить CRUD операции над всеми объектами, представленными в системе.

CRUD - аббревиатура, которая описывает четыре базовых действия

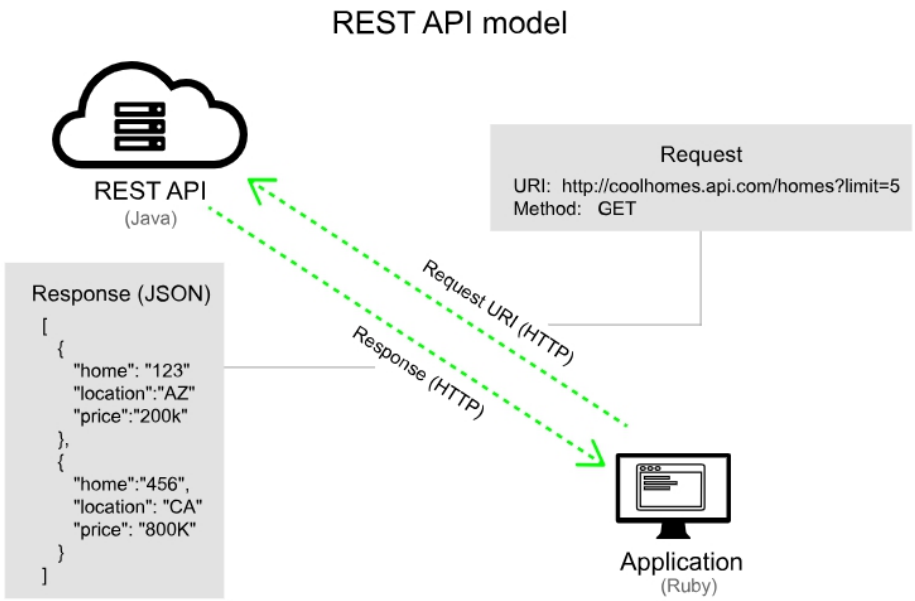
- C - create
- R - read
- U - update
- D - delete

Пример CRUD API:

Конечная точка (*endpoint*) - это ресурс, расположенный на веб-сервере по определенному пути.

Endpoint приложения может выглядеть так:

В REST API CRUD соответствуют *post*, *get*, *put*, *delete*.
Ответ (Response) возвращается, как правило, в формате JSON или XML(реже).



https://www.youtube.com/watch?v=LooL6_chvN4

- **get** - вернуть список объектов:

Request:

```
GET /api/train_samples
```

Response:

```
[
    {id:0, password: 'qwerty', times: 1601},
    {id:1, password: 'admin', times: 1920}
    ...
]
```

- **post** - добавить объект:

Request:

```
POST /api/train_samples/
```

Request object:

```
{password: '0000', times: 1000}
```

Response:

```
{id:9, password: '0000', times: 1000}
```

id - назначится сам

- **put** - обновить выбранную запись:

Request:

```
PUT /api/train_samples/1
```

Request object:

```
{id:1, password: 'admin', times: 2000}
```

Response:

```
{id:1, password: 'admin', times: 2000}
```

- **delete** - удалить выбранный объект:

Request:

```
DELETE /api/train_samples/1
```

📌 Еще пример описания API метода

Коды ответов

Код	Название	Описание
200	OK	Запрос выполнен успешно
201	Created	Возвращается при каждом создании ресурса в коллекции
204	No Content	Нет содержимого. Это ответ на успешный запрос, например, после DELETE)

- https://habr.com/ru/post/440900/ - статья про коды и REST
- https://developer.mozilla.org/ru/docs/Web/HTTP/Status - полная таблица кодов

Код	Название	Описание
400	Bad Request	Ошибка на стороне Клиента. Например, неправильный синтаксис запроса, неверные параметры запроса и т.д.
401	Unauthorized	Клиент пытается работать с закрытым ресурсом без предоставления данных авторизации
403	Forbidden	Сервер понял запрос, но отказывается его обрабатывать
404	Not found	Запрашивается несуществующий ресурс
405	Method Not Allowed	Клиент пытался использовать метод, который недопустим для ресурса. Например, указан метод DELETE, но такого метода у ресурса нет
500	Server error	Общий ответ об ошибкте на сервере, когда не подходит никакой другой код ошибки

Curl

- Что это? - Client URL, утилита командной строки
- Зачем? - позволяет выполнять запросы с различными параметрами и методами без перехода к веб-ресурсам в адресной строке браузера. Поддерживает протоколы HTTP, HTTPS, FTP, FTPS, SFTP и др.

Установка

- В MacOS, Ubuntu - доступен из командной строки
- В Windows - требуется установка. Инструкция. Можно также установить Git Bush.

Проверить установку в WIN можно из командной строки *cmd* ➔ *curl -V*
Если установлен, появится сообщение вида:

```
`curl 7.55.1 (Windows) libcurl/7.55.1 WinSSL`
```

Примеры запросов

- **Curl - GET запрос**

```
curl https://host.com
```

Метод GET - по умолчанию. Тот же результат получим, если вызовем так:

```
curl -X GET https://host.com
```

Чтобы получить ответ с заголовком:

```
curl https://host.com -i
```

Ответ будет содержать версию HTTP, код и статус ответа (например: HTTP/2 200 OK). Затем заголовки ответа, пустая строка и тело ответа.

- **Curl - POST запрос**

```
curl -X POST https://host.com
```

Используя передачу данных (URL-encoded):

```
curl -d "option=value_1&something=value_2"
     -X POST https://host.com/
```

Здесь -d или --data - флаг, обозначающий передачу данных

- **POST запрос, используя формат JSON**

```
curl -d '{"option": "val"}'
     -H "Accept:application/json"
     -X POST https://host.com/
```

Здесь -H или --header - флаг заголовока запроса на ресурс

Или можно передать json, как файл:

```
curl -d "@file.json"
     -X POST https://host.com/
```

Еще флаги:

- `–u user:pass` - если на сервере требуется аутентификация
- `curl –verbose`, отображает подробности
- –L, поддержка redirect (если ресурс перемещен)
- –0 - сохранить с тем же именем, -o data.json - со новым именем

Документация

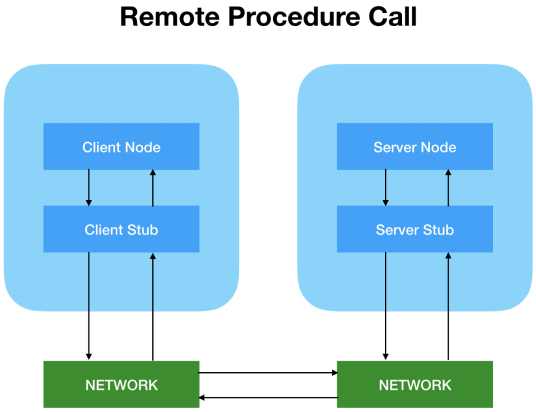
Недостатки/особенности REST API:

- Для каждого языка необходимость разработки своего API. (Можно использовать Swagger - рассмотрим далее)
- JSON для передачи данных - не бинарный формат. Медленнее передача данных, но удобнее просматривать данные
- Протокол HTTP 1.1 - не поддерживает передачу потоковых данных

Данные недостатки учтены в **gRPC(Google Remote Procedure Call)**

gRPC

Основан на **RPC** - вызове удаленного кода на других машинах.



Отличия:

- Генерация кода стандартными средствами. Используется компилятор **Protoc**, который генерирует код для множетсва языков, включая python
- Бинарный формат данных **Protobuf**, использует сжатие -> быстрее передача данных
- Протокол HTTP 2 (2015 год) -> потоковая передача данных, бинарный формат, выше скорость и пр.

Что выбрать?:

- Если важна скорость - gRPC
- Если монолитное приложение с доступом извне или браузер - REST API
- Распределенная система на микросервисах - gRPC
- Потоковые данные (например, с датчиков) - gRPC

Быстрый старт и руководство gRPC для Python

Open API 

Общепринятым форматом для описания REST API на сегодняшний день является OpenAPI, который также известен как Swagger

Спецификация представляет из себя единый файл в формате JSON или YAML, состоящий из трёх разделов:

1. Заголовок, содержащий название, описание и версию API, а также дополнительную информацию;
2. Описание всех ресурсов, включая их идентификаторы, HTTP-методы, все входные параметры, а также коды и форматы тела ответов;
3. Определения объектов в формате JSON Schema, которые могут использоваться как во входных параметрах, так и в ответах.

Веб-фреймворки

Flask  Flask

Что это? - Веб-фреймворк для Python

Почему выбираем его по-умолчанию?

- Минималистичный фреймворк
- Быстрое прототипирование
- Низкоуровневый фреймворк, после освоения будет проще разобраться с Django

Также, лучшим решением будет выбрать Flask, если:

- Разрабатывается микросерсисная архитектура
- Реализуется REST API без фронтенда
- Требуется гибкая кастомизация

Minimal Flask App

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run()
```

Запустив приложение, получим сообщение:

```
~ python app.py
Running on http://127.0.0.1:5000/
```

Localhost - с IP адресом 127.0.01 ➔ внутренняя сеть компьютера

Параметры app.run()

1. Debug mode:

```
1 app.run(debug=True)
```

- *Сервер перезагружается сам при изменении кода*
- *Позволяет работать с отладчиком*
- *Не забыть отключить при развертывании сервиса*

2. Сделать сервер публично доступным

```
1 app.run(host='0.0.0.0')
```

По умолчанию - доступ local

Шаблоны

Шаблон — файл с HTML-кодом и элементами разметки, которые позволяют выводить динамический контент.

Функция render_template() вызывает механизм шаблонов Jinja2, который поставляется в комплекте с Flask.

```
from flask import render_template
```

Шаблоны хранятся в директории /templates

Пример шаблона (/templates/index.html):

```
<html>
    <body>
        <h1>Prediction: {{ pred }}</h1>
    </body>
</html>
```

Пример кода, которые преобразует шаблон в HTML страницу (рендеринг):

```
from flask import Flask, request, render_template


app = Flask(__name__)

#some code

@app.route('/')
def index():
    return render_template('index.html', pred=model.prediction)
```

Flask API

Flask-RESTX - это расширение для Flask, которое добавляет поддержку для быстрой разработки REST API.

Документация:
https://flask-restx.readthedocs.io/en/latest/index.html

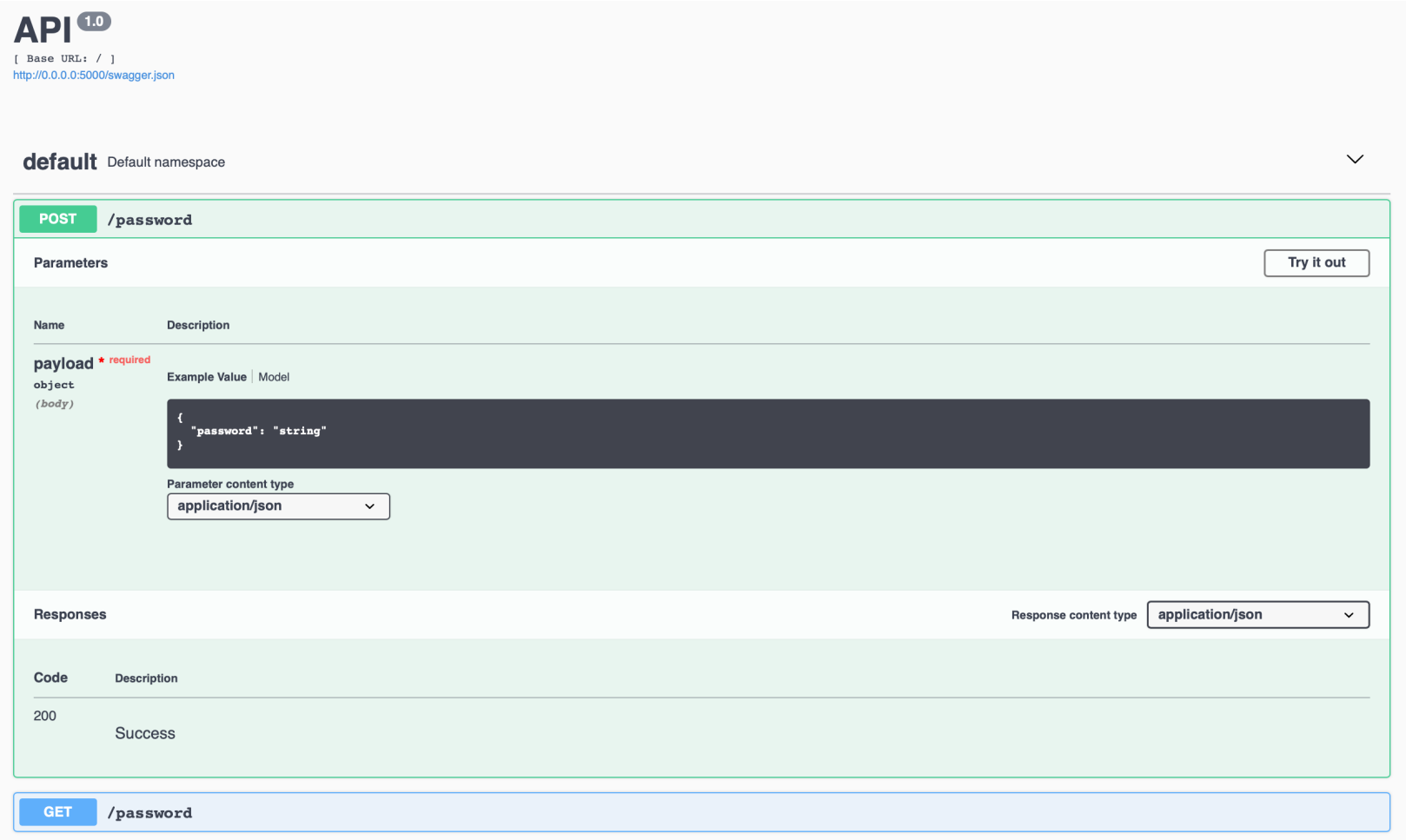
Аналоги: *flask-restplus*, *flask-restful*

Простой пример приложения, реализующий API на Flask:

```
from flask import Flask
from flask_restx import Api, Resource, fields

app = Flask(__name__)
api = Api(app)

passwords = []
a_password = api.model('Resource', {'password': fields.String})

@api.route('/password')
class Prediction(Resource):
    def get(self):
        return passwords

    @api.expect(a_password)
    def post(self):
        passwords.append(api.payload)
        return {'Result': 'pass added'}, 201
```

Flask-RESTX предоставляет набор инструментов для генерации документации с использованием Swagger.

Документация **Swagger API** создается автоматически и доступна по корневому URL API:



Deploy

Пошаговое руководство как развернуть Flask приложение на Heroku

На 4-й лекции более подробно будет разобран вопрос развертывания.
Также будет полезно закрепить базовые представления о Git.

Django

- Что это? - еще один популярный фреймворк на python для разработки веб приложения или API.

Особенности:

- Встроенная Django Admin
- Встроенная защита от наиболее распространенных уязвимостей и атак, в частности: SQL-инъекции,CSRF,XSS, кликджекинг, и т.д.
- Поддержка ORM

Django - хороший выбор для быстрой разработки масштабируемого приложения. Но не лучший выбор для микросервисов, простого API-приложения без фронтенда и баз данных.

FastAPI

Преимущества:

- встроенная документация API
- асинхронность
- валидация (pydantic)
- быстродействие

Установка:

```
pip install uvicorn fastapi pydantic
```

Интерактивная документация:

http://127.0.0.1:8000/docs

Чуть подробнее о FastAPI на хабре

Streamlit

- Что это? Opensource Python фреймворк для быстрой разработки дашборда в проектах с машинным обучением, не требующий знания frontend (HTML, CSS и JavaScript)

Установка

```
pip install streamlit
```

Особенности

1. Виджеты
    - checkboxes
    - selectBox
    - slider
    - multiSelect (tags)
2. Визуализация
    - matplotlib
    - component for rendering Folium maps.

)

What to do

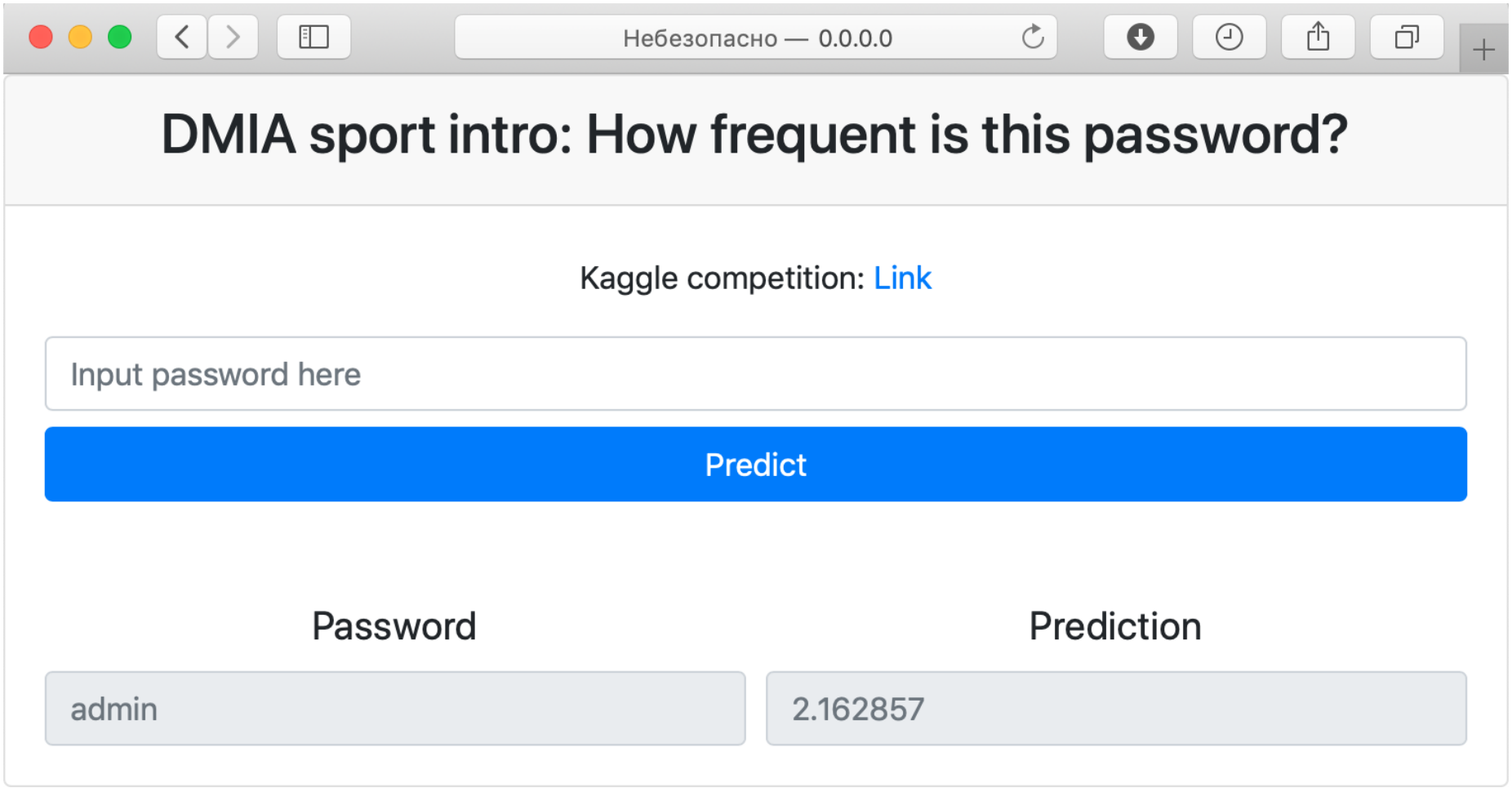
Choose the app mode

Show instructions

To continue select "Run the app".

UI

- Что это? User Interface, пользовательский интерфейс
- Зачем? - помочь пользователю, организовав комфортное и, по возможности, интуитивно понятное взаимодействие с сайтом. Включает перечень оформленных графических элементов (кнопок, чекбоксов, селекторов и т.д.)


Небезопасно — 0.0.0.0
DMIA sport intro: How frequent is this password?
Kaggle competition: Link
Input password here
Predict
Password
Prediction
admin
2.162857

- **HTML** - язык гипертекстовой разметки, определяет содержание и структуру веб-контента
  Справочник HTML
- **CSS** - язык иерархических правил, используемый для представления внешнего вида страницы
  Справочник CSS

Bootstrap

Фреймворк, позволяющий быстро создавать адаптивный сайт. Включает набор инструментов для создания сайтов и веб-приложений. Содержит HTML и CSS-шаблоны оформления для типографики, веб-форм, кнопок и прочих компонентов веб-интерфейса

https://getbootstrap.com/docs/5.0/getting-started/introduction/

Описание на русском

Базовый шаблон, подключающий bootstrap:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=n
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/

    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
  </body>
</html>
```

Семинар

1. Пробуем работу с **curl** на примере сайта https://reqres.in/:

- **get** запрос

```
curl https://reqres.in/api/users/2
curl https://reqres.in/api/users/2 -i
curl https://reqres.in/api/users/2 -I
```

отправляем вывод в файл:

```
curl -o test_curl.txt https://reqres.in/api/users/2
```

- **post** запрос

```
curl -X POST  https://reqres.in/api/users -d "name=morpheus&job=datascientist" -i
```

используя **json:**

```
curl -d '{"name": "Ivan", "job: "data scientist"}' -H "Accept:application/json" -X POST http
```

- **delete** запрос

```
curl -X DELETE https://reqres.in/api/users/1 -i
```

получаем код 204 и пустое тело

- Изучаем файл app.py

2. **Flask**
- И шаблон index.html

3. **Flask API**

- Ветка с игрушечным примером, как добавить API в проект:

https://gitlab.com/production-ml/password_app/-/blob/test_api/app.py

- Запустить приложение и посмотреть работу **Swagger**

Добавить запись с помощью curl:

```
curl -X POST "http://0.0.0.0:5000/password" -H "accept: application/json" -H "Content-Type:
```

Посмотреть консоль, где запущено приложение. Должно быть вида:

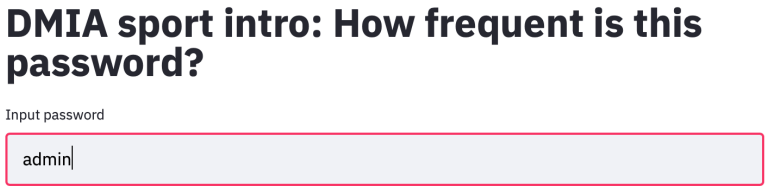
```
127.0.0.1 - - [16/Mar/2021 16:57:43] "GET /swagger.json HTTP/1.1" 200 -
127.0.0.1 - - [16/Mar/2021 16:59:31] "POST /password HTTP/1.1" 201 -
127.0.0.1 - - [16/Mar/2021 17:00:31] "POST /password HTTP/1.1" 400 -
```

4. **Streamlit**
- Пример сервиса в отдельной ветке:

https://gitlab.com/production-ml/password_app/-/blob/streamlit_try/app.py

- Поднимаем сервис локально:

```
streamlit run app.py
```


DMIA sport intro: How frequent is this password?
Input password
admin
Prediction: 2.246 times

Home work