ML Advanced

REST-архитектура: Flask API

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

REST-архитектура: Flask API

Сизов Александр Александрович, PhD

Lead Data Scientist

15 years of software development and research experience;
8 years of experience in Machine Learning/Deep Learning projects development management.

https://www.linkedin.com/in/aleksandr-sizov-550593b8/

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в учебной группе

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

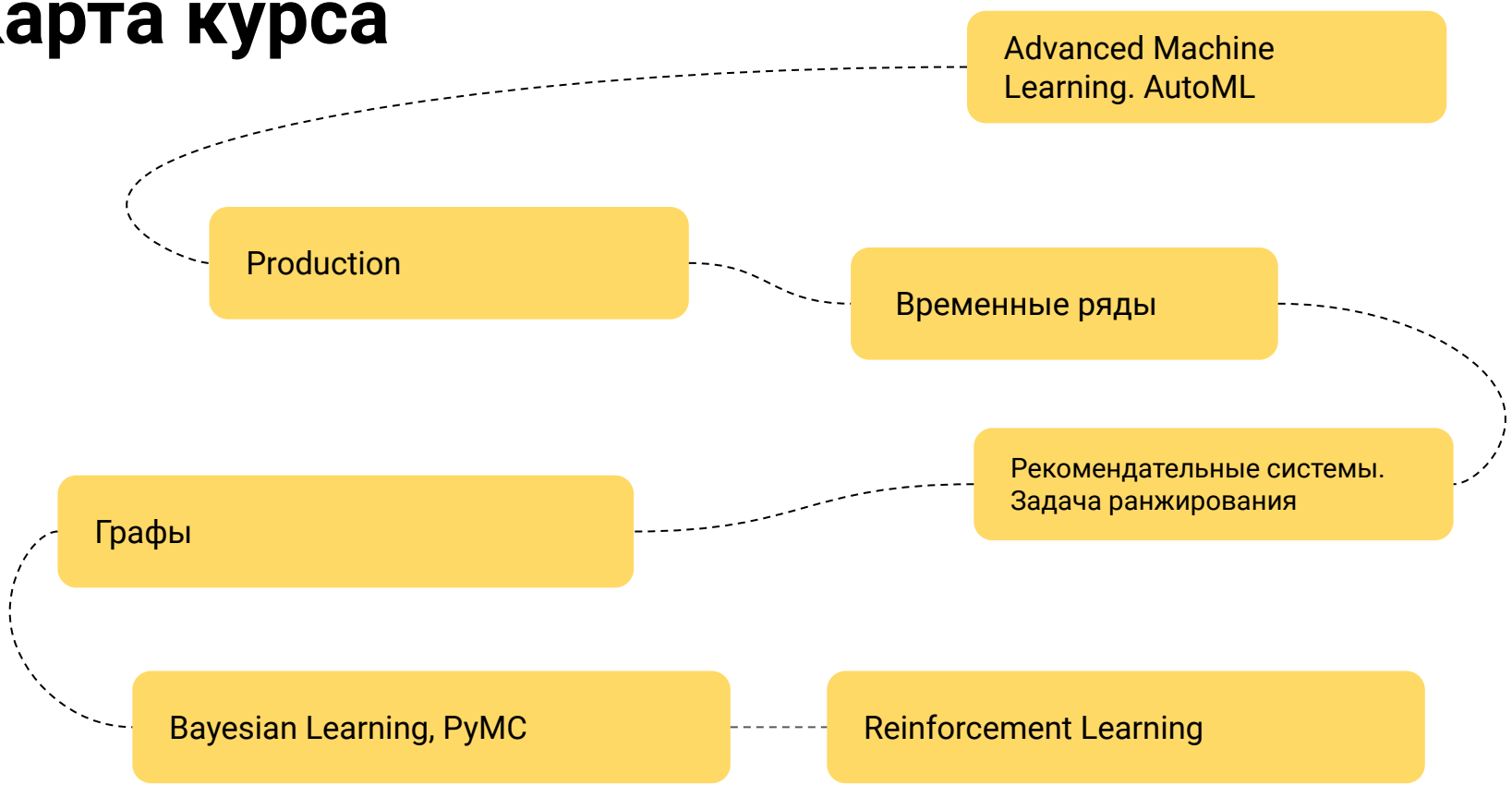
Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Карта курса


Advanced Machine Learning. AutoML
Production
Временные ряды
Рекомендательные системы. Задача ранжирования
Графы
Bayesian Learning, PyMC
Reinforcement Learning

Цели вебинара

1 Познакомимся с REST API

2 Познакомимся с Flask

Смысл

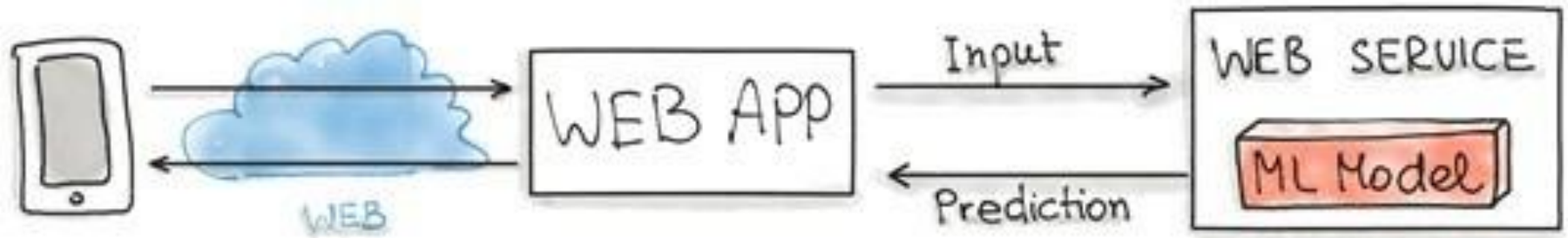
1. Сможем создавать свои REST службы
2. Сможем использовать Flask

Маршрут вебинара

- Мотивация
- REST API
- Flask
- Model-as-a-Service
- Рефлексия

Мотивация

Модель как услуга (Model-as-a-Service)

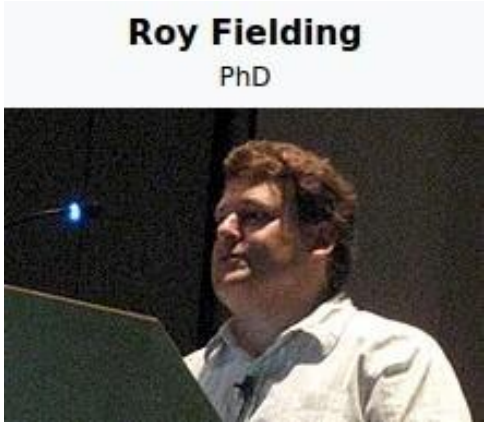


REST API

REST API

REST (**Re**presentational **S**tate **T**ransfer —

«передача состояния представления») — **архитектурный стиль** взаимодействия компонентов распределённого приложения

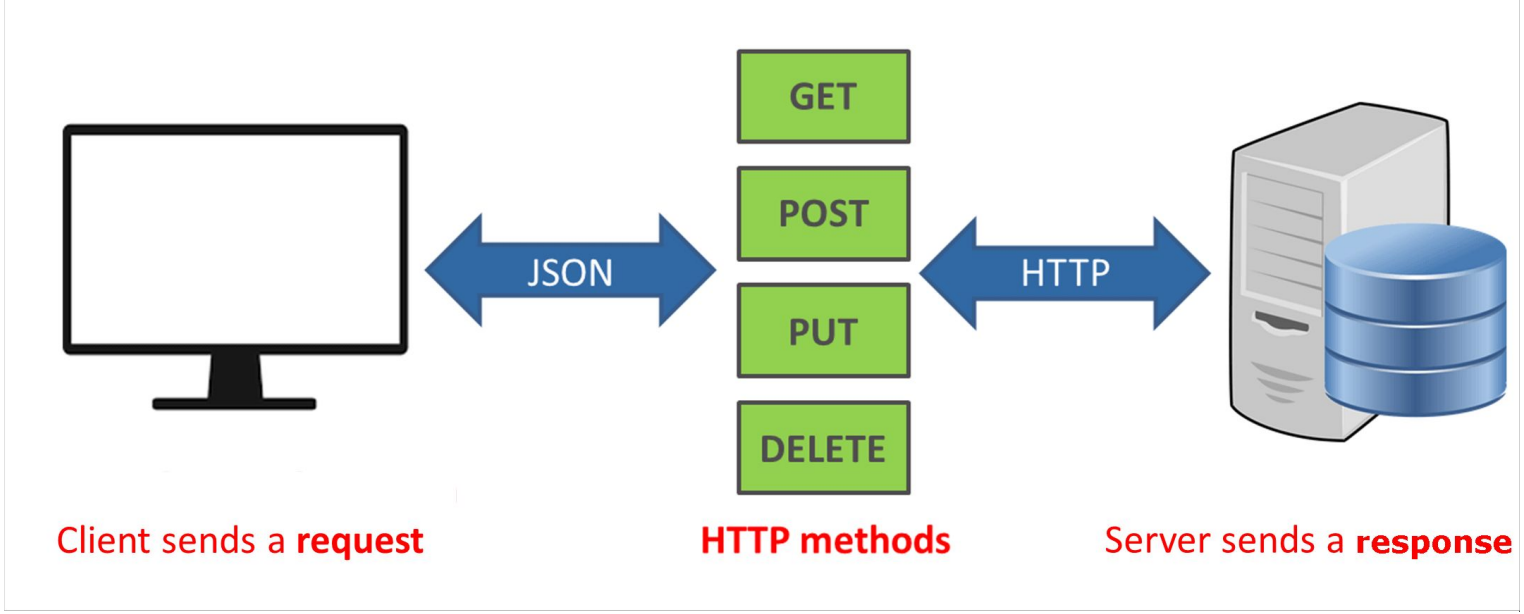


https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

RESTful — службы, не нарушающие

ограничений REST

REST API


GET
POST
PUT
DELETE
JSON
HTTP
Client sends a **request**
HTTP methods
Server sends a **response**

Ресурс

- Всё сущее является ресурсом
- Каждый ресурс должен иметь уникальный URL
  - */api/posts/12345* - ресурс
  - */api/posts/* - коллекция ресурсов

Семантика HTTP методов

- **GET** – получить состояние ресурса
- **POST** – создать ресурс
- **PUT** – обновить ресурс
- **DELETE** – удалить ресурс

Семантика HTTP методов

Request method	Target	Description	HTTP response status code
GET	Individual resource URL	Obtain the resource.	200
GET	Resource collection URL	Obtain the collection of resources (or one page from it if the server implements pagination).	200
POST	Resource collection URL	Create a new resource and add it to the collection. The server chooses the URL of the new resource and returns it in a `Location` header in the response.	201
PUT	Individual resource URL	Modify an existing resource. Alternatively, this method can also be used to create a new resource when the client can choose the resource URL.	200 or 204
DELETE	Individual resource URL	Delete a resource.	200 or 204
DELETE	Resource collection URL	Delete all resources in the collection.	200 or 204

Требования к архитектуре REST

1) Модель клиент – сервер
2) Отсутствие состояния
3) Кэширование
4) Единообразие интерфейса
   1) Идентификация ресурсов (URI)
   2) Манипуляция ресурсами через представления
   3) «Самоописываемые» сообщения
   4) HATEOAS (**H**ypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate)
5) Слои
6) Код по требованию (опционально)

Гипермедиа как средство изменения состояния

REST-клиент обращается к фиксированному URL, сервер возвращает ресурсы для последующих действий.

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
Accept: application/xml
...
```

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
    <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
    <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
    <link rel="close" href="https://bank.example.com/accounts/12345/close" />
</account>
```

Гипермедиа как средство изменения состояния

REST-клиент обращается к фиксированному URL, сервер возвращает ресурсы для последующих действий.

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
Accept: application/xml
...
```

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">-25.00</balance>
    <link rel="deposit" href="https://bank.example.com/account/12345/deposit" />
</account>
```

Преимущества

- Надёжность
- Производительность
- Масштабируемость
- Прозрачность системы взаимодействия
- Простота интерфейсов
- Портативность компонентов
- Легкость внесения изменений
- Способность эволюционировать

OpenAPI Specification (Swagger)

Формат описания API:

- Конечные точки (/users) и операции над ними (GET /users, POST /users)
- Параметры операций, ввод и вывод
- Методы аутентификации
- Контактная информация, лицензия, условия, использования и т.п.

Описывается на JSON или YAML

Фреймворки

- Flask (flask.palletsprojects.com)
- FastAPI (fastapi.tiangolo.com)
- Django (www.djangoproject.com)

Клиенты

- cURL (curl.se)
- Insomnia (insomnia.rest)
- Postman (www.postman.com)

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Flask

Flask

- Микрофреймворк
- Модульный дизайн
- Зависимости:
  - Werkzeug — маршрутизация, WSGI
  - Jinja 2 — шаблонизатор

Пример

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

@app.route("/")
def hello():
    return "Hello World!"

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

```
(venv) $ export FLASK_APP=hello.py
(venv) $ flask run
 * Serving Flask app "hello"
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Регистрация функций

Аргументы:

- URL
- Имя
- Функция

```
def index():
    return '<h1>Hello World!</h1>'

app.add_url_rule('/', 'index', index)
```

Динамические маршруты

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

@app.route('/')
def index():
    return '<h1>Hello World!</h1>'

@app.route('/user/<name>')
def user(name):
    return '<h1>Hello, {}!</h1>'.format(name)
```

Контексты приложения и запроса

- Контекст приложения
  - `current_app` — экземплер активного приложения
  - `g` — временное хранилище
- Контекст запроса
  - `request` — содержимое HTTP-запроса
  - `session` — сеанс пользователя

Request

- form
- args
- values
- cookies
- headers
- files
- get_data()
- get_json()
- endpoint
- method
- scheme
- is_secure()
- host
- path
- query_string
- full_path
- url
- base_url
- remote_addr
- environ

Обработчики событий жизненного цикла

- `before_first_request` — перед обработкой первого запроса
- `before_request` — перед обработкой каждого запроса
- `after_request` — после обработки каждого запроса, если нет необработанных исключений
- `teardown_request` — после обработки каждого запроса, если есть необработанные исключения

Ответ

```
@app.route('/')
def index():
    return '<h1>Bad Request</h1>', 400
```

```
from flask import make_response

@app.route('/')
def index():
    response = make_response('<h1>This document carries a cookie!</h1>')
    response.set_cookie('answer', '42')
    return response
```

Response

- status_code
- headers
- set_cookies()
- delete_cookies()
- content_length
- content_type
- set_data()
- get_data()

Переадресация (redirect)

```
from flask import redirect

@app.route('/')
def index():
    return redirect('http://www.example.com')
```

Обработка ошибок (abort)

```
from flask import abort

@app.route('/user/<id>')
def get_user(id):
    user = load_user(id)
    if not user:
        abort(404)
    return '<h1>Hello, {}</h1>'.format(user.name)
```

Структура больших приложений (пример)

```
|-flasky
  |-app/
    |-templates/
    |-static/
    |-main/
      |-__init__.py
      |-errors.py
      |-forms.py
      |-views.py
    |-__init__.py
    |-email.py
    |-models.py
  |-migrations/
  |-tests/
    |-__init__.py
    |-test*.py
  |-venv/
  |-requirements.txt
  |-config.py
  |-flasky.py
```

Вопросы?

Ставим “+”,
если вопросы есть

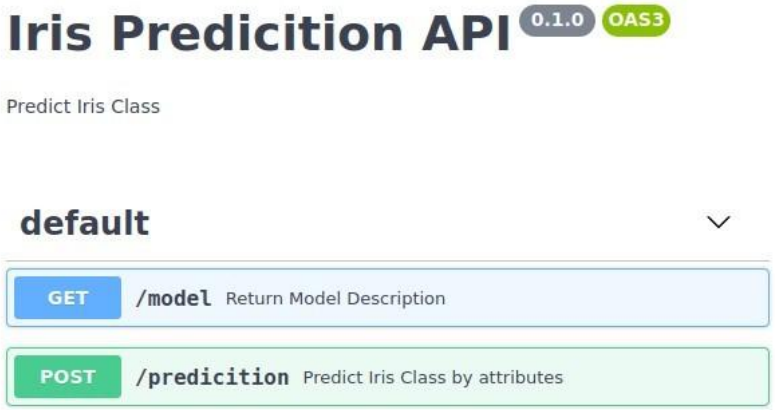
Ставим “–”,
если вопросов нет

Model-as-a-Service

OpenAPI

```yaml
openapi: 3.0.0
info:
  title: Iris Predicition API
  description: Predict Iris Class
  version: 0.1.0

paths:
  /model:
    get:
      summary: Return Model Description
      responses:
        '200':
          description: JSON with Model Description
          content:
            application/json:
              schema:
                type: object
                properties:
                  name:
                    type: string
                    example: Decision Tree Classifier
                  accuracy:
                    type: number
                    example: 0.8
  /predicition:
    post:
      summary: Predict Iris Class by attributes
      responses:
        '200':
          description: Prediction
          content:
            application/json:
              schema:
                type: object
                properties:
                  class:
                    type: string
                    example: setosa
```



Flask приложение

```python
from flask import Flask, jsonify, request
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

app = Flask(__name__)


def train_model():
    iris_df = datasets.load_iris()
    data = iris_df.data
    target = iris_df.target
    target_names = iris_df.target_names
    train_data, test_data, train_target, test_target = train_test_split(data, target, test_size=0.3)
    dt = DecisionTreeClassifier().fit(train_data, train_target)
    acc = accuracy_score(test_target, dt.predict(test_data))
    return dt, acc, target_names


model, accuracy, names = train_model()
```

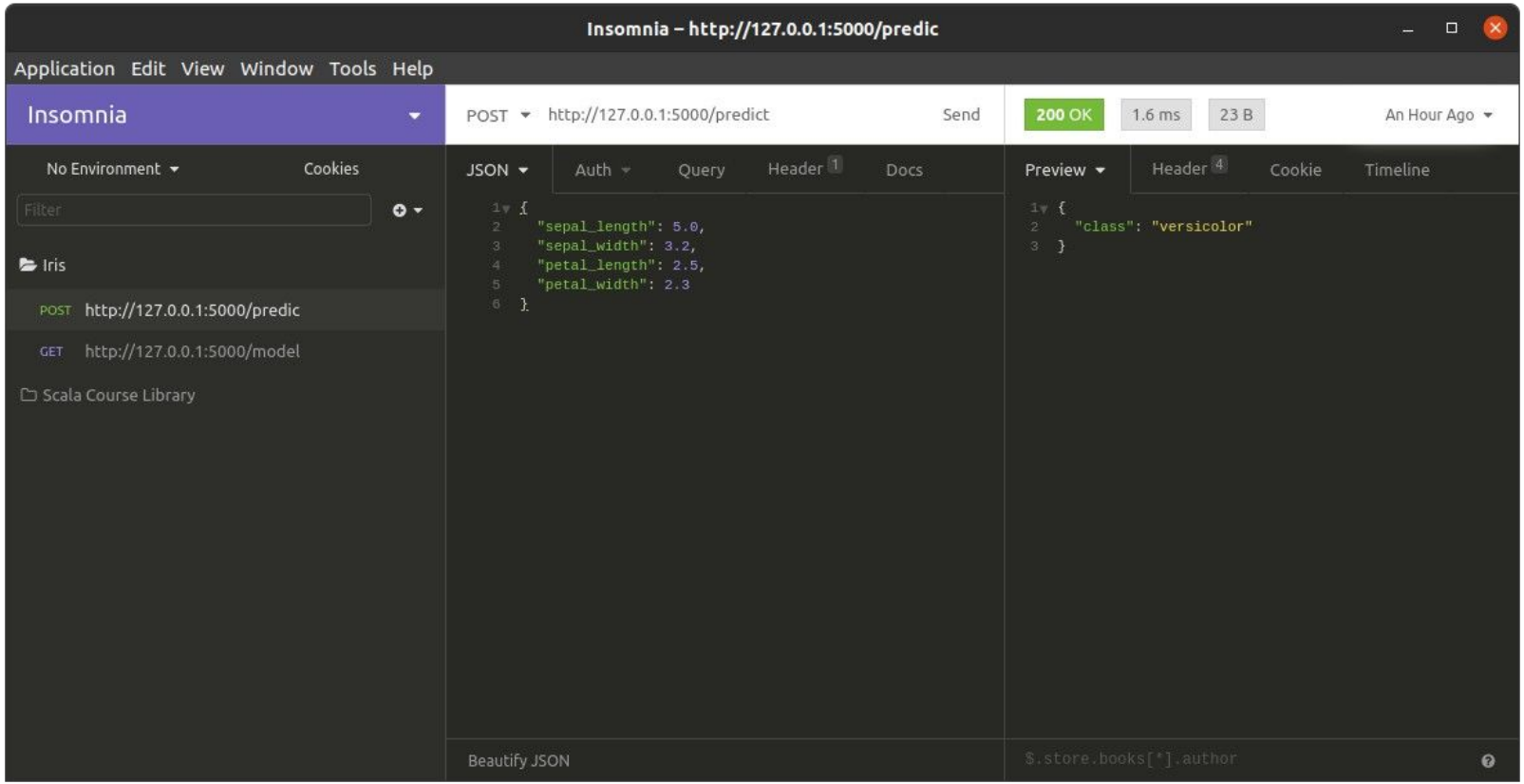
Flask приложение (продолжение)

```python
@app.route('/predict', methods=['POST'])
def predict():
    posted_data = request.get_json()
    sepal_length = posted_data['sepal_length']
    sepal_width = posted_data['sepal_width']
    petal_length = posted_data['petal_length']
    petal_width = posted_data['petal_width']
    prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])[0]
    return jsonify({'class': names[prediction]})


@app.route('/model')
def get_model():
    return jsonify({'name': 'Decision Tree Classifier',
                    'accuracy': accuracy})


if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Работа с приложением



Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

Рефлексия

С какими впечатлениями уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Сизов Александр Александрович, PhD

Lead Data Scientist

15 years of software development and research experience;
8 years of experience in Machine Learning/Deep Learning projects development management.

https://www.linkedin.com/in/aleksandr-sizov-550593b8/