

# Инструмент для поиска плагиата в исходном коде

Куратор	Дмитрий Иванов, 6304
Лидер	Корытов Павел, 6304
Разработчики	Артём Бутко, 8304 Дмитрий Перелыгин, 8303 Александр Алтухов, 8304 Александр Рыжиков, 8304

# Постановка задачи

✓	Реализация загрузки датасета StackOverflow в БД
✓	Реализация работы с репозиториями организаций на GitHub
✗✓	Подключение выгрузки репозитория к REST API
✓	Реализация алгоритма поиска плагиата
✓✗	Подключение фронтэнда к REST API
✗	Модульное тестирование

# Методы решения

- Язык программирования - Python
  - BeautifulSoup4 для парсинга датасета StackOverflow
  - Flask - веб-сервер
- PostgreSQL + fuzzystmatch для поиска похожего кода в БД
  - Конструктор запросов к SQL - PyPika
- Vue.js + Bootstrap - фронтэнд

# Работа с репозиториями

Реализована выгрузка репозиториев из организации (в том числе и приватных).

Пользователь предоставляет логин, токен и имя организации.

Реализована проверка корректности токена и его принадлежность пользователю.

# Результаты. Stackoverflow

Обработано 50 миллионов постов и разобрано на отдельные файлы:

ЯП	Количество строк
JavaScript	$2.2 \cdot 10^6$
Java	$1.4 \cdot 10^6$
C++	$4.7 \cdot 10^5$
Python	$1.9 \cdot 10^6$
C	$2.6 \cdot 10^5$

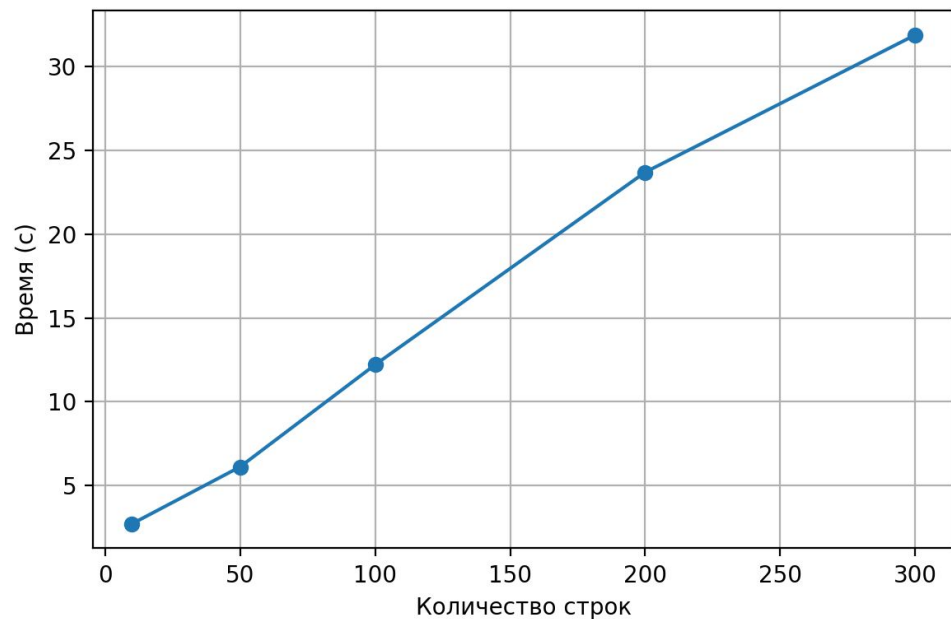
Время работы алгоритма заняло около 8-ми часов для датасета размером 80ГБ

# Алгоритм

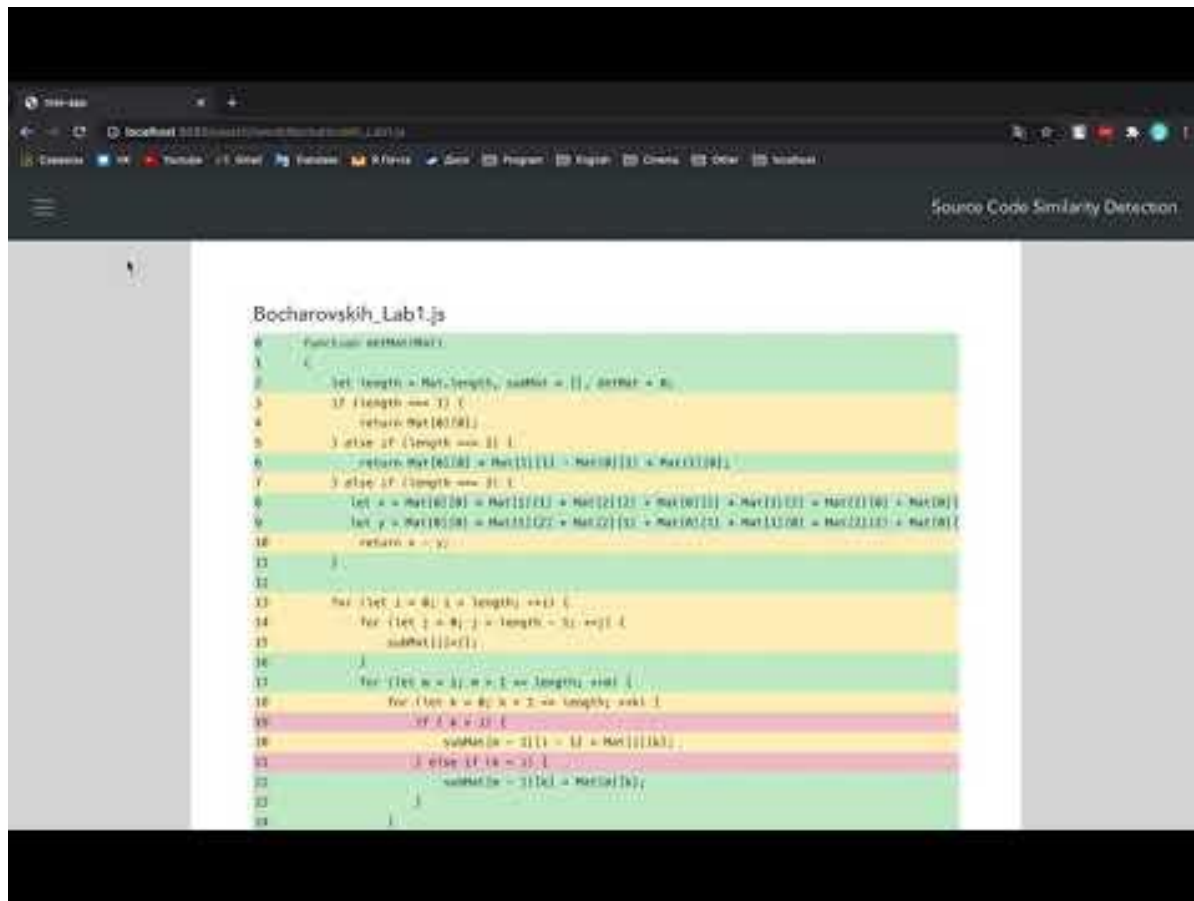
Был реализован алгоритм, анализирующий строки кода, преобразованные с помощью функции `metaphone`, предоставляемой модулем `fuzzystmatch`. Между закодированными строками, хранящимися в базе данных, производится поиск расстояния Левенштейна. На основе схожести строк делается вывод о степени плагиата.

# Зависимость времени работы от количества строк

В базе находится 120000 строк на языке анализируемого файла.



# Демонстрация



```
Bocharovskih_Lab1.js
1  function getMat(Mat)
2  {
3      let length = Mat.length, subMat = [], getMat = 0;
4      if (length === 1) {
5          return Mat[0][0];
6      } else if (length === 2) {
7          return Mat[0][0] + Mat[1][1] + Mat[0][1] + Mat[1][0];
8      } else if (length === 3) {
9          let x = Mat[0][0] + Mat[1][1] + Mat[2][2] + Mat[0][1] + Mat[1][2] + Mat[2][0] + Mat[0][2] + Mat[1][0] + Mat[2][1];
10         let y = Mat[1][0] + Mat[1][2] + Mat[2][1] + Mat[0][1] + Mat[1][0] + Mat[2][2] + Mat[0][2] + Mat[1][1];
11         return x + y;
12     }
13 }
14
15 for (let i = 0; i < length; ++i) {
16     for (let j = 0; j < length - 1; ++j) {
17         subMat[i][j] = 0;
18     }
19     for (let m = 1; m < length; ++m) {
20         for (let k = 0; k < 2 - m; ++k) {
21             if (i + k < 2) {
22                 subMat[i + k][j] = Mat[i][j + k];
23             } else if (i + k < 3) {
24                 subMat[i + k][j] = Mat[i][j];
25             }
26         }
27     }
28 }
```



# Инструкция по запуску

1. Установить virtualenv  
`pip install virtualenv`  
Альтернативный вариант (советую его) - [Miniconda3](#).
  - Установить Miniconda
  - `conda create --name mse_plagiarism_search`
  - `conda activate mse_plagiarism_search`
  - `conda install python`
  - Опустить шаги 3.2, 3.3 далее
2. Склонировать репозиторий  
`git clone git@github.com:moevm/mse_plagiarism_search.git`  
`cd mse_plagiarism_search`
3. Настройка бэкэнда. В корне репозитория:
  - `cd backend`
  - `python -m virtualenv venv`
  - `source venv/bin/activate`
  - Установить зависимости Python  
`pip install -r requirements.txt`  
В случае проблем с `psycogp2`, ставить как `pip install psycogp2-binary`
4. Настройка фронтэнда. В корне репозитория:
  - `cd frontend`
  - Установить зависимости Node.js  
`npm install`

В корне репозитория:

1. Запуск БД (консоль 1). В корне репозитория:
  - i. `docker-compose up`  
Будет развернута база PostgreSQL на порте. 5432 и pgAdmin на порте 81.
2. Запуск бэкэнда (консоль 2):
  - i. `cd backend`
  - ii. `source venv/bin/activate` (или `conda activate mse_plagiarism_search`)
  - iii. `export FLASK_APP=app.py`
  - iv. `python -m flask run` сервер запустится на порту 5000. При первом запуске в базе будут созданы таблички и будет установлено расширение `fuzzystmrmatch`
3. Запуск фронтэнда (консоль 3):
  - i. `cd frontend`
  - ii. `npm run serve`  
Клиент запустится на порту 8080.

# Планы на следующую итерацию

- Подключение выгрузки репозитория к REST API
- Завершение работы над фронтэндом
  - Работа с репозиториями
  - Улучшение формы результатов проверки
  - Настройки приложения
- Загрузка нескольких файлов на проверку
- Оптимизация алгоритма
- Модульное тестирование
- Развертывание в Docker