Санкт-Петербургский государственный политехнический университет
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА
по дисциплине «Конструирование программного обеспечения»
Система поиска

Выполнили студенты
гр. 5130904/20104

Ганиуллин Родион Ринатович
Овсянников Кирилл Александрович
Авдеева Наталья Алексеевна
Кумачев Антон Александрович

Руководитель:

Юркин Владимир Андреевич

Санкт-Петербург
2025

Содержание

Введение

Задача

Задача посталенная для курсовой работы – разработать проект и пройти большинство классических стадий создания ПО. Проект должен иметь базу данных, серверную часть, внешнюю зависимость и UI. Проект должен быть покрыт тестами.

Тема

В качестве темы мы выбрали выбрали систему поиска комиксов с довольно известного сайта https://xkcd.com по ключевым словам (полнотекстовый поиск)

Описание

xkcd.com имеет удобное API https://xkcd.com/{id}/info.0.json ,которое возвращает информацию о комиксе: название, описание и ссылку на изображение. Используя это API, мы и будем получать нужные данные

Основная часть

Требования

Функциональные требования

1. Поиск комиксов по ключевым словам
    - FR1.1: Система должна предоставлять возможность поиска комиксов, по ключевым словам, в названии, описании и других метаданных.
    - FR1.2: Результаты поиска должны возвращаться в порядке релевантности.
2. Веб-интерфейс для поиска
    - FR2.1: Система должна предоставлять веб-интерфейс с полем ввода для поискового запроса.
3. REST API для поиска
    - FR3.1: Система должна предоставлять REST API эндпоинт (GET /api/pics?search={query}) для поиска комиксов.
    - FR3.2: Ответ API должен быть в формате JSON и содержать список комиксов с их метаданными.
4. Обновление поискового индекса
    - FR4.1: Система должна предоставлять API эндпоинт (POST /api/update) для ручного обновления поискового индекса.
    - FR4.2: Система должна поддерживать автоматическое обновление индекса по расписанию (раз в сутки).
5. Авторизация для защищённых ручек
    - FR5.1: Эндпоинт обновления индекса (POST /api/update) должен быть доступен только аутентифицированным пользователям с правами администратора.
    - FR5.2: Аутентификация должна выполняться через токен (JWT/Bearer Token).
6. Функция любимых комиксов
    - FR6.1: Должна быть возможность добавить комиксы в любимые и удалить их
    - FR6.2: Должна быть возможность посмотреть свои любимые комиксы

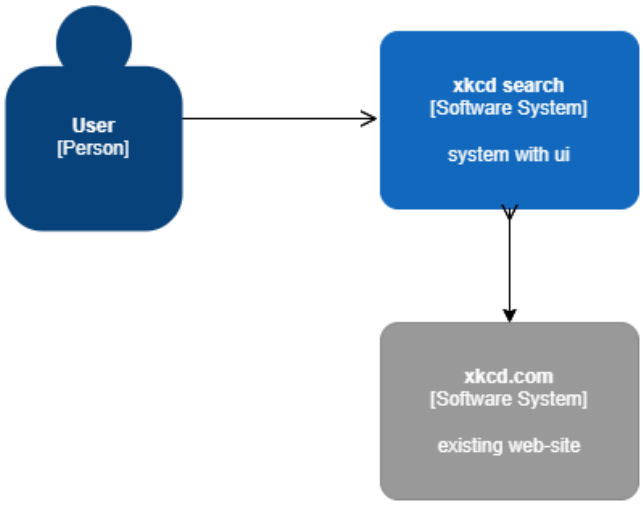
Нефункциональные требования

1. Производительность
    - NFR1.1: Поисковые запросы (веб и API) должны обрабатываться менее чем за 500 мс при средних нагрузках.
    - NFR1.2: Обновление индекса не должно занимать более 5 минут.
2. Масштабируемость
    - NFR2.1: Система должна выдерживать до 1000 RPS.
    - NFR2.2: Обновление индекса не должно блокировать поисковые запросы.
3. Надёжность

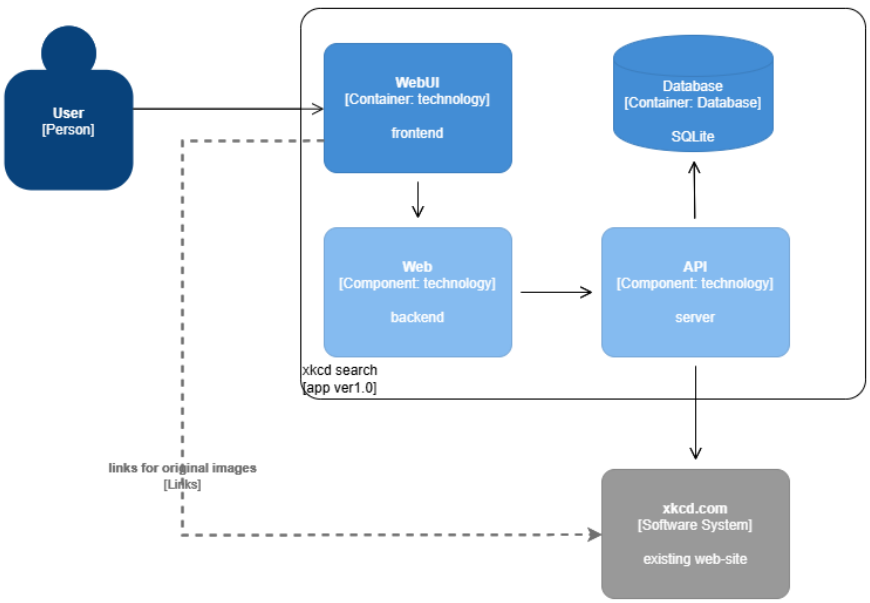
- NFR3.1: Поисковый индекс должен автоматически восстанавливаться в случае сбоя.

Архитектура

Системный уровень



Уровень контейнеров



Стэк

База данных

После того, как мы оценили объем хранимых данных (порядка 5 Мб) и нагрузку на системы были приняты следующие решения:

1. Хранить индекс в памяти приложения
    - Малый объем данных
    - Этот способ увеличивает пропускную способность и уменьшает время отклика
2. Для персистеного хранения комиксов использовать Sqlite3 (на случай падения и перестройки индекса)
    - Быстро востановиться из локального хранилища. Не надо идти в API

Сервисы

Написаны на языке Golang.

UI

Web-UI использует язак шаблонов Go для генерации HTML с добавкой JS для интерактивности

Нагрузка

Почти вся нагрузка на наш сервис Read, т.к. запись мы производим только при перестроении индекса по запросу или раз в 24 часа

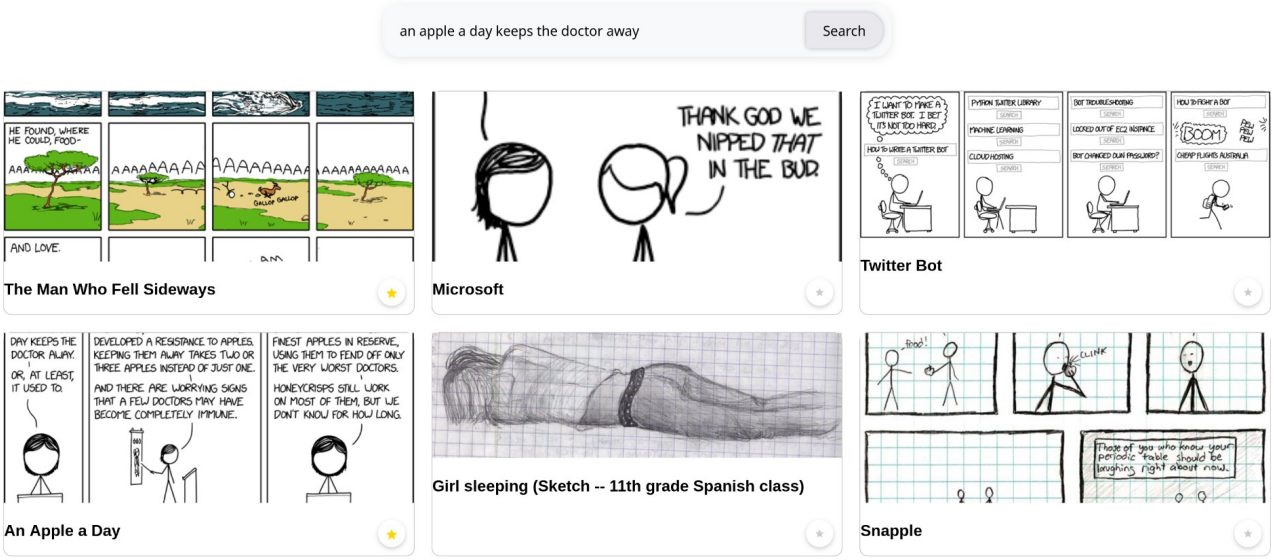
Данные, которые нам надо хранить ~ 5 Мб

Тестирование

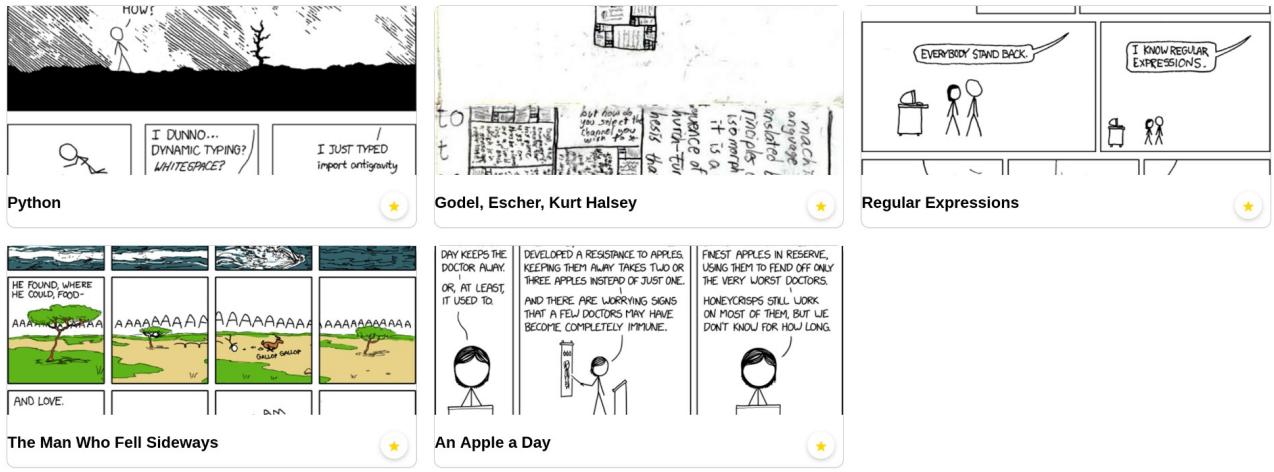
Система покрыта юнит, интеграционными, нагрузочными и end-to-end тестами

UI

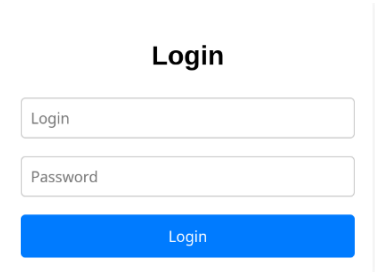
Поиск



Любимые комиксы



Страница авторизации



Код стемера

```
package words

import (
	"bufio"
	"io"
	"strings"
	"unicode"

	"github.com/kljensen/snowball"
)

type Stemmer struct {
	stopWords map[string]struct{}
}

// NewStemmer is a function to construct new Stemmer.
// If stopWords == nil uses default dictionary https://www.ranks.nl/stopwords
func NewStemmer(stopWords map[string]struct{}) *Stemmer {
	if stopWords != nil {
		return &Stemmer{stopWords: stopWords}
	}

	return &Stemmer{stopWords: map[string]struct{}{
		"a": {}, "about": {}, "above": {}, "after": {}, "again": {}, "against": {}, "all": {},
		"am": {}, "an": {}, "and": {}, "any": {}, "are": {}, "aren't": {}, "as": {}, "at": {},
		"be": {}, "because": {}, "been": {}, "before": {}, "being": {}, "below": {},
"between": {},
		"both": {}, "but": {}, "by": {}, "can't": {}, "cannot": {}, "could": {}, "couldn't": {},
		"did": {}, "didn't": {}, "do": {}, "does": {}, "doesn't": {}, "doing": {}, "don't": {},
```

```
                "down": {}, "during": {}, "each": {}, "few": {}, "for": {}, "from": {}, "further": {},
                "had": {}, "hadn't": {}, "has": {}, "hasn't": {}, "have": {}, "haven't": {}, "having":
{},
                "he": {}, "he'd": {}, "he'll": {}, "he's": {}, "her": {}, "here": {}, "here's": {}, "hers":
{},
                "herself": {}, "him": {}, "himself": {}, "his": {}, "how": {}, "how's": {}, "i": {},
"i'd": {},
                "i'll": {}, "i'm": {}, "i've": {}, "if": {}, "in": {}, "into": {}, "is": {}, "isn't": {},
                "it": {}, "it's": {}, "its": {}, "itself": {}, "let's": {}, "me": {}, "more": {}, "most": {},
                "mustn't": {}, "my": {}, "myself": {}, "no": {}, "nor": {}, "not": {}, "of": {}, "off":
{},
                "on": {}, "once": {}, "only": {}, "or": {}, "other": {}, "ought": {}, "our": {}, "ours":
{},
                "ourselves": {}, "out": {}, "over": {}, "own": {}, "same": {}, "shan't": {}, "she": {},
                "she'd": {}, "she'll": {}, "she's": {}, "should": {}, "shouldn't": {}, "so": {}, "some":
{},
                "such": {}, "than": {}, "that": {}, "that's": {}, "the": {}, "their": {}, "theirs": {},
                "them": {}, "themselves": {}, "then": {}, "there": {}, "there's": {}, "these": {},
"they": {},
                "they'd": {}, "they'll": {}, "they're": {}, "they've": {}, "this": {}, "those": {},
                "through": {}, "to": {}, "too": {}, "under": {}, "until": {}, "up": {}, "very": {},
"was": {},
                "wasn't": {}, "we": {}, "we'd": {}, "we'll": {}, "we're": {}, "we've": {}, "were": {},
                "weren't": {}, "what": {}, "what's": {}, "when": {}, "when's": {}, "where": {},
"where's": {},
                "which": {}, "while": {}, "who": {}, "who's": {}, "whom": {}, "why": {}, "why's":
{},
                "with": {}, "won't": {}, "would": {}, "wouldn't": {}, "you": {}, "you'd": {}, "you'll":
{},
                "you're": {}, "you've": {}, "your": {}, "yours": {}, "yourself": {}, "yourselves": {},
                "alt": {}, "text": {}, "title": {},
        }}
}
```

```
func ParseStopWords(reader io.Reader) map[string]struct{} {
	stopWords := make(map[string]struct{})
	scanner := bufio.NewScanner(reader)
	scanner.Split(bufio.ScanWords)

	for scanner.Scan() {
		word := scanner.Text()
		if word != "" {
			stopWords[word] = struct{}{}
		}
	}

	return stopWords
}

func ParsePhrase(phrase string) []string {
	words := strings.FieldsFunc(phrase, func(r rune) bool {
		return !(unicode.IsNumber(r) || unicode.IsLetter(r))
	})
	for i := 0; i < len(words); i++ {
		words[i] = strings.ToLower(words[i])
	}

	return words
}

// isStopWord is method, that check if word is not significant
func (s *Stemmer) isStopWord(word string) bool {
```

```
	_, ok := s.stopWords[word]
	return ok
}

func (s *Stemmer) Stem(words []string) map[string]int {
	// using map to avoid duplicates
	stemmed := make(map[string]int)

	for _, word := range words {
		if len(word) < 4 || s.isStopWord(word) {
			continue
		} else {
			word, _ = snowball.Stem(word, "english", false)
			if len(word) < 3 {
				continue
			}

			stemmed[word] += 1
		}
	}

	return stemmed
}
```

Запуск

Для запуска необходимо

1. Скачать репозиторий (git pull --depth 1 https://github.com/Coding-Seal/xkcd-search.git)
2. Собрать и запустить (docker compose up -d)
3. Теперь по адресу http://localhost:8080 вы сможете найти приложение

Результат

В результате мы создали функционирующую систему поиска, пройдя через основные этапы разработки программного обеспечения: сбор требований, проектирование, кодирование и тестирование.

Все этапы были выполнены успешно. Система удовлетворяет всем поставленным требованиям и пригодна для использования по основным пользовательским сценариям.

Список литературы

- The English (Porter2) stemming algorithm // snowballstem URL: https://snowballstem.org/algorithms/english/stemmer.html (дата обращения: 21.05.2025).
- Sqlite // Sqlite URL: https://sqlite.org/ (дата обращения: 21.05.2025).
- Go // Go programming language URL: https://go.dev/ (дата обращения: 21.05.2025).
- xkcd // xkcd.com URL: https://xkcd.com/ (дата обращения: 21.05.2025).
- Основы Natural Language Processing для текста // Хабр URL: https://habr.com/ru/companies/Voximplant/articles/446738/ (дата обращения: 21.05.2025).