Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЙ С МИКРОСЕРВИСНОЙ АРХИТЕКТУРОЙ. ИСПОЛЬЗОВАНИЕ АЛГОРИТМА РАНЖИРОВАНИЯ EDGERANK

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Павкина Александра Андреевича

Научный руководитель
к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В платформах с высокой пользовательской активностью часто требуется отображать наиболее интересные публикации вначале новостной ленты. Это может потребоваться, когда количество ежедневных публикаций очень велико и просмотреть сразу все не представляется возможным. Существует несколько способов решения этой задачи. Один из них — это алгоритм EdgeRank.

Цель бакалаврской работы — проектирование веб-приложения с микросервисной архитектурой с использованием алгоритма ранжирования EdgeRank.

Поставленные задачи:

— Изучить поход микросервисного проектирования.

— Реализовать серверную часть на языке Java по принципу микросервисной архитектуры.

— Реализовать клиентскую часть с использованием JavaScript фреймворка Angular 6.

— Исследовать и реализовать алгоритм ранжирования новостной ленты EdgeRank.

— Настроить и организовать работу серверной части с:

NoSQL базой данных MongoDB для хранения информации о публикациях.

PostgreSQL базой данных для для хранения информации о пользователях.

— Настроить Docker для автоматизации развертывания.

— Настоить CI/CD для автоматизации сборки.

Материалы исследования включают официальные статьи, описывающие алгоритм EdgeRank. Также использовалась официальная документация по Spring Boot, Spring Cloud, Angular и MongoDB.

Работа состоит из трех глав. Глава «Сравнение типов архитектур приложений» содержит теоретические выкладки по основным архитектурным паттернам и их сравнения. Глава «Проектирование и разработка приложения» описывает практическую часть работы. В главе «Демонстрация работы приложения» представлены примеры работы приложение и результат работы алгоритма.

1 Микросервисная архитектура

В течение последних нескольких лет веб архитектура развивается с высокой скоростью. И как следствие появилось несколько подходов для организации архитектуры приложения. Одним из самых популярных является микросервисный стиль [1].

Микросервисная архитектура — это подход к созданию приложний при котором происходит декомпозиция функционала. Обычно применяется разделение функционала на небольшие или средние по размеру приложения. Во многом это сделано для того, чтобы сложность каждого приложения была очень низкой, а коммуникация происходила через набор API (HTTP или асинхронный обмен сообщениями). Каждое приложение выполняет несколько операций, которые ограничены областью единой функциональности. Например, управление клиентами.

Микросервисы во многом похожи на SOA (Service - Oriented Architectures). Основанные на микросервисах архитектуры — это те, которые имитируют SOA с очень маленькими сервисами, каждый из которых работает в собственном процессе и коммуницирует с остальными используя легковесные механизмы. Эти сервисы построены вокруг бизнес-потребностей и могут разрабатываться разными командами параллельно.

Микросервисы развертываются независимо с использованием полностью автоматизированной среды [2]. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе эти сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных. Сервисы коммуницируют посредством либо HTTP/REST запросов, либо через асинхронные протоколы, такие как AMQP. У каждого микросервиса должна быть своя собственная база данных чтобы быть независимой от других сервисов.

По сравнению с традиционными SOA, микросервисы приносят большую сложность на архитектурном уровне, так как есть еще больше, крошечных действующих лиц, но практическое преимущество в том, что они все изолированы, независимы и общаются только через простые интерфейсы (любой из вышеперечисленного вида API).

По сравнению с классическим монолитом у микросервисной архитектуры есть много преимуществ, таких как: относительно простое тестирование,

проще процесс развертывания приложения, возможность разделения на команды занимающихся каждый своим сервисом. Процесс разработки также значительно упрощается так как код, решающий конкретную бизнес задачу, становится изолированным друг от друга. Это также позволяет не повредить функционал, выходящий за пределы решаемой кодом задачи.

Еще к преимуществам можно отнести изоляцию. Например, если в одном сервисе есть утечка памяти, тогда будет нарушена работа только этого сервиса. Другие будут продолжать обрабатывать запросы. Для сравнения, один неправильный компонент монолитной архитектуры может привести к поломке всего приложения.

Существует обратная сторона этого аспекта. В следствии того, что микросервисы работают все-таки не изолированно, а общаются друг с другом отказ одного сервиса может негативно сказаться на работу других. Поэтому у разработчика появляется необходимость проектирования приложения так, чтобы оно могло работать при отказе отдельных сервисов. Любое обращение к сервису может не сработать из-за его недоступности. Клиент должен реагировать на это настолько терпимо, насколько возможно. Это является недостатком микросервисов по сравнению с монолитом, так как это вносит дополнительную сложность в приложение. Simian Army от Netflix искусственно вызывает (симулирует) отказы сервисов и даже датацентров в течение рабочего дня для тестирования отказоустойчивости приложения и служб мониторинга.

Также к недостаткам можно отнести сложность тестирования межмодульного взаимодействия, необходимость разработки межсервисной связи, сложность реализации задач, которые охватывают несколько сервисов — это требует тщательной координации между командами.

Сложность также бывает с развертыванием микросервисов. Так как не всегда все сервисы пишутся на одном языке, то имеется операционная сложность развертывания и управления системой, состоящей из множества различных типов сервисов.

Тем не менее, не смотря на количество недостатков считается что микросервисы это новый виток эволюции в построении приложений. Зачастую, с проблемами приходится мириться чтобы решать те проблемы, которые невозможно решить, используя монолитную архитектуру.

2 Проектирование и разработка приложения

В ВКР разрабатывается веб-блог на микросервисной архитектуре с функцией умного ранжирования публикаций. Разбиение на микросервисы обусловлено тем, что у блога потенциально высокая нагрузка, а такая архитектура позволяет горизонтально масштабировать приложение на неограниченное количество запущенных сервисов.

2.1 Серверная часть

Серверная часть была написана на языке Java с использованием фреймворков Spring Boot и Spring Cloud.

Spring Cloud — это один из подмодулей Spring. Он был выбран так как содержит в себе реализации паттернов микросервисного подхода, некоторые из которых были описаны выше [3]. Spring Cloud предоставляет дополнительную архитектуру, реализующую эти паттерны, что значительно ускоряет процесс разработки продукта. Он содержит в себе набор современных решений от OSS Netflix. Сам по себе Spring Cloud, как называют это сами разработчики, — это Release Train, содержащий набор зависимостей/модулей, версии которых согласованы между собой и рассчитаны на конкретную версию Spring Boot.

Spring Boot использовался чтобы еще больше упростить и ускорить разработку, почти исключив необходимость писать какие либо конфигурации [4]. Также он содержит встроенный сервер приложений Tomcat что значительно упрощает развертывание приложения. Spring Boot уже содержит в себе самые частоиспользуемые зависимости в согласованных версиях и с заранее определенными конфигурациями.

Было принято решение декомпозировать приложение на функционал по управлению аккаунтами и на функционал по управлению публикациями. Каждый модуль представляет собой отдельное независимое приложение, которое может функционировать автономно.

Архитектуру приложения можно представить в виде диаграммы:

Каждый компонент реализует конкретные задачи:

— Account service — реализует логику и операции по настройке и предоставлению аккаунта.

— Post-service — реализует логику и операции по настройке и предостав-

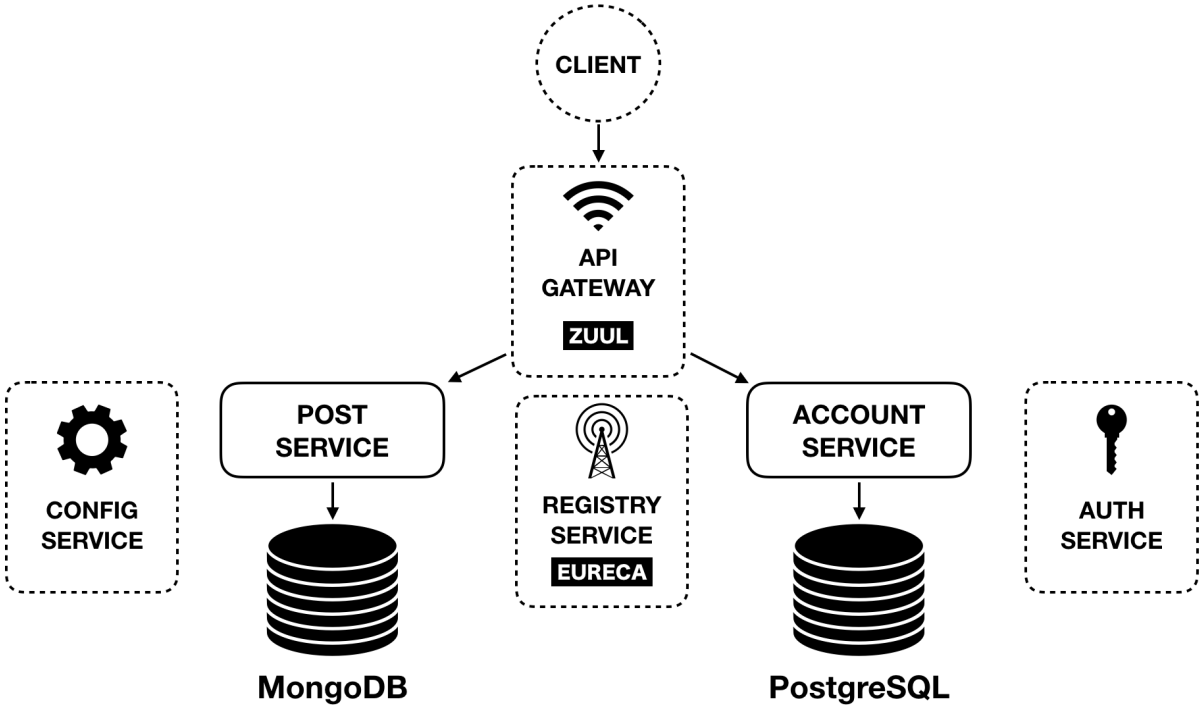

Рисунок 1 – Диаграмма компонентов

лению публикаций. Опционально позволяет ранжировать публикации, а также обрезать контент поста для отправки на ленту.

— API Gateway — предоставляет единую точку входа в приложение. Ее используют для приема внешних запросов и маршрутизациив нужные сервисы внутренней инфраструктуры, отдачи статического кон-тента, аутентификации, стресс тестирования, канареечного развертывания,миграции сервисов, динамического управления трафиком.

— Registry service — предназначен для регистрации поднятых приложений в системе для последующего обнаружения другими сервисами. Это реализация так называемого Client-side Discovery паттерна, что означает клиент должен запросить адреса доступных поднятых сервисов и осуществлять балансировку между ними самостоятельно. Для реализации этого сервиса использовался компонент Spring Cloud — Netflix Eureka.

— Config service — это горизонтально масштабируемое хранилище конфигураций для распределенной системы.

2.2 Клиентская часть

При разработке клиентской части приложения использовались самые современные технологии, позволяющие быстро и качественно создать интер-

фейсы для конечного пользователя продукта. В качестве основополагающего элемента выступает UI фреймворк, созданный и поддерживаемый компанией Google — Angular [5]. Основными критериями при подборе технологии, задающей структуру приложения, были скорость работы и взаимодействия с объектной моделью документа (DOM) браузера, производительность, легковесность, наличие инструментов для комфортной разработки и, конечно же, наличие крупной компании, способной не только поддерживать, но и развивать данный фреймворк.

Для создания наиболее комфортных условий использования приложения, было принято решение разрабатывать его как SPA. Это означает, что при переходе между различными частями приложения, не тербуется перезапрашивать новую страницу с сервера заново, html-разметка будет генерироваться практически моментально на клиенте, и тут же отрисовываться браузером.

Все современные UI фреймворки продвигают компонентный подход и Angular не исключение. Каждый элемент страницы представляется собой изолированный компонент, имеющий минимум знаний об окружающих его сущностях и всё необходимое для собственной работы. Каждый компонент, в свою очередь может содержать более мелкие компоненты, следующие всё тем же правилам. Типичным примером можно считать компонент post-preview (превью публикации) имеющий следующую структуру:

— Файл с расширением .ts, содержащий в себе объявление переменных, приходящих извне и используемых в html-шаблоне, а также логику, срабатывающую при нажатии на кнопку просмотра поста
— scss файл, содержащий CSS стили для компонента, использующий возможности препроцессора SASS, такие как вложенные стили, миксины и переменные;
— html файл, возможности которого расширены встроенным в Angular шаблонизатором, позволяющим использовать различные интерполяции и директивы, производящие те или иные операции над данными в данном шаблоне.

Для хранения данных и манипуляции с ними на стороне клиента использовался мощный стейт-контейнер ngrx/state, выросший на основе Flux архитектуры. Наличие данной технологии позволяет создать на стороне кли-

ента глобальный контейнер для состояния приложения, доступный из разных его частей.

Для реализации взаимодействия клиент-сервер, была подключена библиотека ngrx/effects позволяющая в удобно производить асинхронные операции по получению и обработке данных о пользователях и публикациях.

2.3 Алгоритм ранжирования EdgeRank

EdgeRank — это алгоритм ранжирования Facebook, который определяет, какие публикации появляются в ленте новостей каждого пользователя [6].

Каждое действие, которое принимают их подписчики, является потенциальной новостью. Facebook называет эти действия «Края» (далее Edge). Это означает, что каждый раз, когда друг отправляет обновление статуса, комментирует другое обновление статуса, публикует фотографию, подписывается на страницу какой-то группы или присоединяется к событию, которое генерирует Edge, то информация об этом Edge может отображаться в личной новостной ленте.

$$Score = \sum u_i w_i d_i,$$

где

$$u_i = Affinity, w_i = Weight, d_i = Timedecay$$

2.3.1 Affinity

Affinity Score означает, что насколько близок конкретный пользователь к текущему, то есть к Edge. Например, пользователь дружит с братом на Facebook. Кроме того, он часто пишет на его стене, и у них есть пятьдесят общих друзей. Получается очень высокая оценка совместимости таких пользователей. Из этого следует что они оба скорее всего захотят увидеть новости друг друга или другие обновления в своей ленте.

В практической части работы считалось 3 метрики:

$$Affinity = \log_{10}(nLUP) * log_{10}(nMF) * \log_{10}(nMI)$$

$nLUP$ — колличество публикаций понравившихся у автора публикации.

nMF — колличество общих подписок

nMI — колличество общих и интересов

Опираясь на потребности предметной области, для вычисления значений оценки было решено использовать функцию логарифма по основанию 10 из-за ее темпов роста.

2.3.2 Weight

Вес может определяться рядом различных факторов в зависимости от типа предметной области и модели данных. Ключевая идея состоит в том, чтобы определить какая публикации имеет наибольшую ценность для пользователя. Расчет веса происходит из расчета что разные действия — такие как комментирование публикации или «like» — имеют разный вес.

Для вычисления весовой компоненты использовались следующие метрики:

$$Weight = \log_{10}(likes + 1) * \log_{10}(comments + 1) * UI * TR$$

где

UI — колличество комментариев пользователя + отметка «мне нравится»

TR — метрика релевантности тэгов публикации к интересам пользователя [7]

2.3.3 Time decay

Метрика time decay применяется к приложениям, когда требуется учитывать «свежесть» публикации. Важно повысить релевантность нового контента, а устаревший контент «опустить» в ленте новостей. Оценка своевременности гарантирует, что новые статьи имеют шанс появиться в верхней части ленты, хотя в них может не быть очень интересного контента.

В практической части работы временной компонент определяет кусочная функция, состоящая из:

$$d_i(time) = \begin{cases} \sqrt{time} + 0.3, & time \in [0, 1] \\ 1, & time \in [2, 4] \\ \frac{1}{time^2}, & time \in [5, \infty] \end{cases}$$

3 Демонстрация работы приложения

Для демонстрации работы приложения необходимо зайти в веб-интерфейс. Для неавторизованного пользователя будет доступна только вкладка с общей новостной лентой. Далее для входа в систему необходимо ввести логин и пароль. После этого станет доступна влкадка «My posts» и переключатель режимов умной ленты.

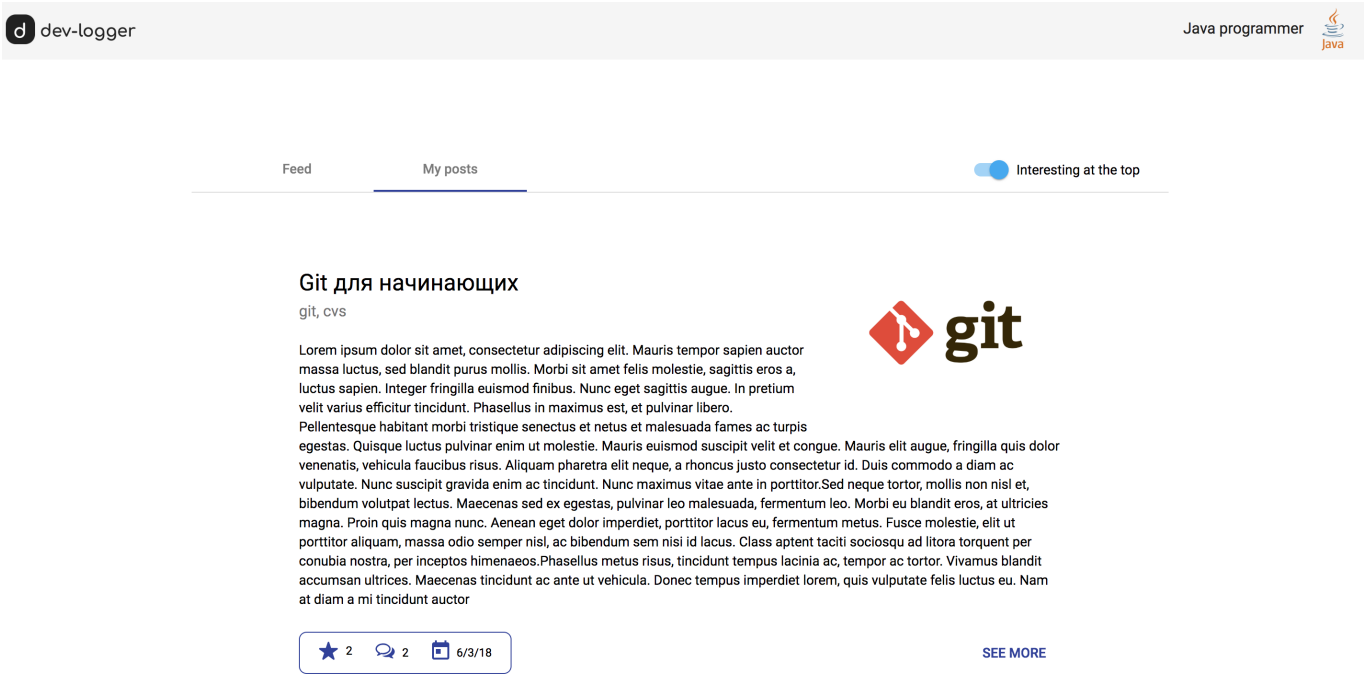

Рисунок 2 – Interesting at the top

Разрешение экрана не позволяет увидеть весь результат ранжирования. Для того чтобы убедиться что алгоритм работает корректно требуется проанализировать логи сервера.

```
========================= EDGE RANK INFO =========================
EdgeRank 0.000057564 FOR POST django_orm WITH AFFINITY=0.006194507, WIGHT=0.030975753, DECAY=0.300000000
EdgeRank 0.000047807 FOR POST python_gui WITH AFFINITY=0.006194507, WIGHT=0.378161105, DECAY=0.020408163
EdgeRank 0.000022622 FOR POST multithreading WITH AFFINITY=0.001479337, WIGHT=2.997221868, DECAY=0.005102041
EdgeRank 0.002660340 FOR POST spring_in_action WITH AFFINITY=0.001479337, WIGHT=5.994443736, DECAY=0.300000000
EdgeRank 0.000167828 FOR POST jvm WITH AFFINITY=0.001479337, WIGHT=0.378161105, DECAY=0.300000000
EdgeRank 0.000095951 FOR POST python3 WITH AFFINITY=0.001479337, WIGHT=0.216203856, DECAY=0.300000000
EdgeRank 0.000036319 FOR POST best_python_ide WITH AFFINITY=0.001479337, WIGHT=0.081835673, DECAY=0.300000000
EdgeRank 0.000266001 FOR POST mongodb_for_beginners WITH AFFINITY=0.001479337, WIGHT=0.599371170, DECAY=0.300000
EdgeRank 0.003993084 FOR POST git_for_beginners WITH AFFINITY=0.001479337, WIGHT=8.997466496, DECAY=0.300000000
==================================================================
```

Рисунок 3 – Оценка EdgeRank

ЗАКЛЮЧЕНИЕ

В ходе дипломной работы был изучен подход микросервисного проектирования веб-приложений и реализован веб-блог с функцией умного ранжирования публикаций. В частности, был изучен и реализован алгоритм ражирования новосной ленты от компании Facebook EdgeRank. Были приобретены навыки разработки UI части с помощью JavaScript фреймворка Angular, а также навыки построения микросервисов с помощью Spring Boot и Spring Cloud.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 On monoliths, service-oriented architectures and microservices [Электронный ресурс]. — URL: https://odino.org/on-monoliths-service-oriented-architectures-and-microservices/ (Дата обращения 27.05.2018). Загл. с экр. Яз. англ.

2 Pattern: Microservice Architecture [Электронный ресурс]. — URL: http://microservices.io/patterns/microservices.html (Дата обращения 27.05.2018). Загл. с экр. Яз. англ.

3 Spring Cloud Documentation [Электронный ресурс]. — URL: https://projects.spring.io/spring-cloud/spring-cloud.html (Дата обращения 31.05.2018). Загл. с экр. Яз. англ.

4 Spring Boot Reference Guide [Электронный ресурс]. — URL: https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/ (Дата обращения 31.05.2018). Загл. с экр. Яз. англ.

5 Angular Documentation [Электронный ресурс]. — URL: https://angular.io/docs (Дата обращения 31.05.2018). Загл. с экр. Яз. англ.

6 EdgeRank [Электронный ресурс]. — URL: http://edgerank.net/ (Дата обращения 30.05.2018). Загл. с экр. Яз. англ.

7 Building Your Own Instagram Discovery Engine: A Step-By-Step Tutorial [Электронный ресурс]. — URL: https://getstream.io/blog/instagram-discovery-engine-tutorial/ (Дата обращения 28.05.2018). Загл. с экр. Яз. англ.