МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теории функций и стохастического анализа

МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 38.03.05 — Бизнес-информатика

механико-математического факультета
Балакшиева Арсена Сережевича

Научный руководитель
доцент, к. э. н. _____ А. Р. Файзлиев

Заведующий кафедрой
д. ф.-м. н., доцент _____ С. П. Сидоров

Саратов 2022

Введение

Основными направлениями развития информационных технологий в 21-м веке являются повсеместный широкополосный доступ в интернет и рост мощности вычислительных систем. Эти два основополагающих фактора позволили совершить рывок в качестве и возможностях интернет-технологий, однако они же их и усложнили. Более того, ситуация усугубляется ускоренной глобализацией в использовании интернет сервисов. Отсюда и появление так называемых "монолитов приложение, доставляемое через единое развертывание и которое содержит внутри себя всю логику. Со временем такие монолиты становится довольно сложно поддерживать и добавлять в них новую функцию. Создание новой функции может означать прикосновение к 5 различным местам: 5 мест, в которых вам нужно написать тесты; 5 мест, которые могут иметь нежелательные побочные эффекты для существующих функций. Монолиты тяжело масштабировать и в них практически нет изоляции. Ошибки в одной части программы могут замедлить или разрушить всё приложение. Со всеми этими недостатками эффективно справляется микросервисная архитектура приложений.

В данной работе будут рассмотрены теоретические основы микросервисной архитектуры, технология для ее построения продемонстрировано программное обеспечение, в котором используются технологические основы микросервисной архитектуры.

1. Микросервисная архитектура. Микросервисная архитектура - это вариант сервис-ориентированной архитектуры. Сервис-ориентированная архитектура - модульный подход к разработке программного обеспечения, базирующийся на обеспечении удаленного по стандартизированным протоколам использования распределённых, слабо связанных легко заменяемых компонентов (сервисов) со стандартизированными интерфейсами. Однако существует сильная разница между двумя этими понятиями. Основное различие сводится к области применения. Сервис-ориентированная архитектура (SOA) имеет масштаб предприятия, а архитектура микросервисов — масш таб приложения. Многие из основных принципов каждого подхода становятся несовместимыми, если пренебрегать этой разницей. Если понять разницу в масштабах, можно быстро придти к тому, что потенциально они могут до-

полнять друг друга, а не конкурировать.

Традиционные процессы разработки программного обеспечения (водопад, agile и т. д.) обычно приводят к тому, что относительно большие группы работают над одним монолитным артефактом развертывания. Руководители проектов, разработчики и операционный персонал могут достичь разной степени успеха с этими моделями, выпуская приложения-кандидаты, которые могут быть проверены бизнесом, особенно по мере того, как они приобретают опыт использования определенного программного обеспечения и стека развертывания. Есть, однако, некоторые скрытые проблемы традиционных подходов. Архитектура микросервисов — в сочетании с технологиями облачного развертывания, управления API и технологиями интеграции — обеспечивает другой подход к разработке программного обеспечения. Вместо этого монолит разбирается на набор независимых сервисов, которые разрабатываются, развертываются и обслуживаются отдельно.

Модули в микросервисной архитектуре могут быть реализованы на основе практически любого современного языка программирования или инструмента, но существует набор ключевых инструментов, ставших для неё обязательными и определяющими.

Редкий случай: микросервисная архитектура практически так же популярна среди руководителей проектов, как и среди самих разработчиков. Причина в том, что микросервисы отлично вписываются в схемы, по которым многие менеджеры структурируют задачи и управляют командами разработчиков. Проще говоря, микросервисная архитектура лучше всего отражает привычные процессы управления.

Отдельные сервисы чрезвычайно просты и могут развёртываться независимо друг от друга, а это значит, что для того чтобы поменять какую-то строчку в коде, не требуется принятие решений на самом верхнем уровне. Тем самым внесение мелких изменений больше не отнимает лишнего времени. Небольшие команды, независимо работающие над отдельными сервисами, могут также объединяться в многофункциональные коллективы по Agile-методологии.

При разработке приложений по монолитной архитектуре обычно используется большая реляционная база данных, единая для всего приложе-

ния, даже если для каких-то отдельных процессов существуют более простые и удобные решения. Это делает всю архитектуру менее эффективной и громоздкой. В случае с микросервисами каждый из них может иметь собственный стек, модель и базу данных, оптимизированные для конкретного процесса.

Прежде всего, речь идёт о Docker — ПО для развертывания и управления приложениями на основе контейнеризации — модели вычислений, наиболее тесно ассоциируемой с микросервисами. Поскольку индивидуальные контейнеры для приложений не обладают всеми атрибутами полноценной операционной системы, они меньше и легче по объёму, чем обычные виртуальные машины. Благодаря этому они запускаются и отключаются быстрее, и тем самым идеально подходят для небольших и лёгких микросервисов.

С появлением множества сервисов на базе контейнеризации стали чрезвычайно востребованы средства автоматизации управления большими наборами контейнеров. Одной из самых популярных в мире технологий «оркестровки» контейнеров, то есть автоматического развёртывания, управления, масштабирования и сетевого подключения на сегодняшний день является Kubernetes.

Микросервисы часто взаимодействуют через API, особенно при первоначальном установлении связей. Несмотря на то, что клиенты и сервисы могут общаться напрямую, шлюзы API выступают полезным промежуточным элементом с ростом числа сервисов в приложении. Помимо функций реверсивного прокси и маршрутизатора, они обеспечивают дополнительный уровень безопасности для приложений.

Однако связь через API не является эффективным и практичным способом оперативного взаимодействия в реальном времени, поэтому наряду с ним в этих случаях применяются обмен сообщениями или потоками событий. С этой задачей лучше всего справляются брокеры сообщений и платформы потоков событий, такие как Apache Kafka.

Бессерверные вычисления — стратегия, доводящая до логического завершения некоторые ключевые особенности облачных и микросервисных технологий. При её реализации исполнительный блок представляет собой даже не просто маленький сервис, а лишь функцию, которая часто выражается

в нескольких строчках кода. Грань, отделяющая бессерверную функцию от микросервиса, довольно условна, поэтому функция обычно рассматривается как ещё более мелкий процесс, чем микросервис.

Методология DevOps (от англ. development и operations; набор практик, нацеленных на активное взаимодействие специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимную интеграцию их рабочих процессов друг в друга) часто рассматривается как одно из важнейших и непременных достоинств микросервисов, что неудивительно, поскольку речь идёт о частом развёртывании небольших сервисов. Более того, именно следование принципам DevOps делает микросервисы успешной архитектурой. В отличие от монолитной архитектуры, микросервисная — это сложная распределённая система с множеством независимых элементов, которая требует оперативного взаимодействия между разработчиками и пользователями, частого обновления и максимального уровня автоматизации. А это именно то, в чём заключается суть концепции DevOps.

2. Описание разработанного программного обеспечения.

Область применения разработанного продукта - работа с расписанием учебных занятий в университете СГУ. Программное обеспечение состоит из двух частей: мобильного приложения для вывода расписания в удобоваримом формате и серверной части, реализованной в микросервисной архитектуре. Бизнес-процесс выглядит следующим образом: сервер в фоновом режиме парсит сайт СГУ и сохраняет всё расписание для всех групп очного режима обучения, далее пользователь открывает мобильное приложение и, путем нескольких запросов и ответов от сервера, выбираются факультет и группа, необходимая пользователю, после чего расписание в формате JSON отправляется на мобильное приложение, где всё это парсится и выводится в читабельном формате на экране устройства.

2.1. Инструментарий.

— Языки программирования: Java и Kotlin.
— Среды разработки: IntelliJ IDEA и Android Studio.
— Фреймворк для разработки сервера: Spring (Spring Boot).
— Средства разработки под Android: Android SDK.

2.2. Описание серверной части. Серверная часть представляют со-

бой 4 микросервиса, каждый со своей задачей:

— Eureka-Server: eureka-сервер, необходимый для обнаружения микросервисами друг друга.

— MailService: Микросервис для отправки сообщений на электронную почту.

— ParserService: Сервис для парсинга сайта СГУ и записи данных в Базу Данных.

— RestAPI: API для взаимодействия с мобильным приложением.

Данный микросервисы описаны с помощью модулей Eureka-client (у Eureka-server соответственно серверная версия данного модуля). Микросервисы ParserService и RestAPI подключены к одной Базе Данных

По заданному времени либо по запросу ParserService начинает парсить сайт СГУ и записывает данные в Базу Данных. После чего они могут быть доступны через RestAPI.

2.3. Описание мобильной части. Мобильной приложение состоит из одного экрана и нескольких модулей. При запуске приложения происходит запрос на сервер для получения списка факультетов. Этот список выводится для пользователя и, соответственно, пользователь должен выбрать факультет. После выбора факультета отправляется запрос на сервер для получения списка групп. Пользователю снова нужно выбрать группу и нажать на кнопку "Получить". После нажатия отправляется запрос для получения расписания. Данные приходят в формате JSON.

3. Реализация программного обеспечения.3.1. Реализация серверной части. Реализация серверной части начинается с описания Eureka-server. Для этого необходимо указать порт, адрес и заполнить конфигурационные файлы. Далее происходит реализация остальных микросервисов: MailService, ParserService, RestAPI.

После реализации Eureka-микросервиса можно приступать к реализации остальных микросервисов. При каждом создании нового микросервиса необходимо установить аннотацию "@EnableEurekaClient" а конфигурацинном файле "application.properties"указать URL адресс Eureka через свойства "eureka.client.service-url.defaultZone"и название инстанса (экземпляра класса, модуля, приложения и так далее) через свойство "eureka.instance.instance-id".

Также необходимо подключить нужные зависимости в файле сборки.

В микросервисе MailService необходимо описать два класса: один класс для конфигурация почтового агрегатора, второй является контроллером, который принимает адрес почты и само сообщение. Для этого из микросервиса RestAPI отправляется запрос на этот контроллер посредством самописного Rest-клиента.

Микросервис ParserService состоит из 3 логических частей: Парсер, описанный с помощью библиотеки Jsoup, Классы связи с Базой Данных, реализованных на технологии Hibernate и контроллер. Контроллер в данном случае необходим для ручного обновления записей в Базе Данных. В данном контроллере содержатся методы refreshFaculties() и refreshGroups(). Страница элемент сайта с факультетами представляет из себя HTML-тег <ul>, который используется для отображения списков значений. Для парсинга факультетов в данном микросервисе есть функция parseFaculties(). Парсинг списка групп не проходит отдельно от парсинга расписания и для этого используется одна общая функция с подфункциями. Список групп на сайте находится в теге <fieldset> с классами do "form_education form-wrapper". Внутри этого элемента есть элементы в классом "course wrapper". Далее нужно перебрать все элементы с классом "fieldset-wrapper". Внутри этих элементов содержатся теги <a> , представляющие собой гиперссылки, выглядящие как номер группы. Переходя по этой ссылке можно попасть на страницу с расписанием, которое парсится следующим образом: получить элемент с id "schedule". Данный элемент является таблицей <tb>, у каждой ячейки таблицы есть свой id, которые надо перебрать, и, если под таким id имеется элемент ячейки, то он парсится. После парсинга всех занятий данные сохраняются в виде двухмерного массива, после чего преобразуется в данные в формате JSON и записываются в бд, вместе с названием группы и факультета. В данном классе также подключён класс для логгирования, чтобы отображать в консоли все важные события.

RestAPI, как микросервис, состоит уже из двух частей: Классы связи с Базой Данных и контроллер. Контроллер, в данном случае, необходим для реализации логики по получению с сервера информации о факультетах, группах и расписании и имеет, соответственно следующие методы:

getAllFaculties(), getAllGroupsOnFacultyWithUrl(), getGroup().

3.2. Реализация мобильной части. Особенностью разработки под мобильную платформу Android является необходимость описания вертски мобильных экранов с помощью языка разметки XML и описание логики на языке Java или Kotlin. Разработка мобильного приложения начинается с описания визуальной части. В данном приложении будет один экран, на котором уместятся все интерфейсы управления, а также будет выводиться контент. Для расположения визуальных элементов в Android SDK используются так называемые Layout-ы, также называемые ViewGroup - контейнеры для базовых View (элементов UI). Существует несколько видов Layout-ов, но в данном случае будут применяться только два: LinearLayout и RelativeLayout. В первом случае все элементы располагаются в линию (горизонтальную либо вертикальную) один за другим. Во втором случае указывается один элемент, а остальные по цепочке располагаются относительно корневого или уже указанного.

В данном приложении Имеется один корневой LinearLayout, в котором располагаются все остальные элементы:

1. LinearLayout с интерфесами управления.
2. RelativeLayout с кнопками выбора дня недели.
3. LinearLayout с списком дополнительной информации.
4. RecyclerView со списком занятий.

Для получения данных с сервера использование стандартных ресурсов языка является достаточно проблематичным. Поэтому использование сторонних библиотек представляется логичным решением. В данно случае используется библиотека Retrofit2, основанная на библиотеке OkHttp.

Основной модуль приложение условно можно разделать на три части: предзагрузка приложения, отправка запроса на сервер и получение ответа с сервера, работа с полученными с сервера данными. Предзагрузка представляет собой подготовку необходимых классов (таких как, например, адаптер для RecyclerView и HTTP-клиент), настройку View-элементов (настройки цвета, кликабельности) и так далее. Отправка запроса и получение ответа с сервера происходит посредством уже указанного HTTP-клиента. Сначала получается информацию о факультетах, потом о списке групп на факультете и в конце о

занятиях группы. Все это происходит в многоуровневых запросах. После получения все данные передаются в метод setData(), который отправляет часть данных адаптеру, а другую устанавливает напрямую в интерфейс.

ЗАКЛЮЧЕНИЕ

В данной работе была проведена разработка программного обеспечения, основанного на микросервисной архитектура и мобильного приложения. В ходе разработки были освоены следующие технологии: Spring Boot, Spring Cloud, Android SDK и так далее. Изучены основы разработки программного обеспечения, состоящего из разных платформ и способы взаимодействия между ними.