МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математического и компьютерного моделирования

Микросервисная архитектура корпоративных порталов. Разработка и тестирование

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студента (ки) 3 курса 381 группы
направления 09.04.03 Прикладная информатика

механико-математического факультета

Планидиной Анастасии Александровны

Научный руководитель
профессор, д.э.н., профессор _____ Л.В. Кальянов

Зав.кафедрой
зав. каф., д.ф.-м.н., доцент _____ Ю.А. Блинков

Саратов 2024

Введение. В современном мире модернизация приложений часто означает переход на облачные приложения на основе микросервисов. Для их развертывания используют такие контейнерные технологии, как Docker и Kubernetes. Причина в том, что с микросервисной архитектурой можно облегчить масштабирование, ускорить разработку и сократить итеративный цикл разработки сервисов.

По мере того, как приложения становятся более распределенными и сложными, растет популярность микросервисов. Основной принцип микросервисной архитектуры — создание приложения путем разделения компонентов его бизнес-логики на небольшие сервисы, которые можно развертывать и использовать независимо друг от друга.

После создания новых микросервисов и интеграции их с основными сервисами системы, важно помнить про тестирование не только нового функционала, но и про регрессионное и интеграционное тестирование. Зачастую API запросы падают или возвращают некорректный ответ. При проведении регрессионного тестирования потенциально могут возникнуть дефекты, которые ранее не были обнаружены. При разработке микросервиса не всегда учитываются уже существующие интеграции системы, возможно появление дефектов при взаимодействии разных сервисов.

Целью данной работы является разработка чат-бота на корпоративном портале. В задачи данной работы входят: дать определение микросервисной архитектуры, проанализировать виды чат-ботов, выбрать вид чат-бота, который подойдет под нужды корпоративного портала, разработать чат-бота на языке программирования Python, протестировать чат-бота для корпоративного портала.

Магистерская работа состоит из введения, четырех разделов, заключения и списка использованных источников. Введение описывает актуальность темы магистерской работы, цель и задачи. В первом разделе представлена тема, в которой будет рассмотрена микросервисная архитектура. Также будут представлены различия между микросервисами и монолитом. Во втором разделе будет рассмотрена тема тестирования программного обеспечения - какие виды и фазы тестирования существуют. В третьем разделе будет рассмотрена тема разработки чат-бота: от бизнес требований до разработки чат-бота

на языке Python. В четвертом разделе будет рассмотрена тема тестирования чат-бота для корпоративного портала. Тестирование выделено в отдельный раздел, так как это является профессиональным интересом, данная тема будет рассмотрена глубже, чем разработка. В заключении рассматриваются итоги магистерской работы, делаются выводы о проведенной работе. В список использованных источников входят 25 наименований. В магистерской работе делаются ссылки на эти источники.

Определение микросервисной архитектуры. Что такое микросервисы? Микросервисы — это архитектурный и организационный подход к разработке программного обеспечения, при котором программное обеспечение состоит из небольших независимых сервисов, которые взаимодействуют через четко определенные API. Эти сервисы принадлежат небольшим автономным командам.

В микросервисной архитектуре приложение разбивается на ряд независимо развертываемых сервисов, которые взаимодействуют с помощью API-интерфейсов. Благодаря такому подходу каждый отдельный сервис можно развертывать и масштабировать независимо от других. В результате команды могут быстрее и чаще поставлять объемные и сложные приложения. В отличие от монолитного приложения, с микросервисной архитектурой команды могут быстрее внедрять новые возможности и вносить изменения, при этом им не приходится переписывать большие фрагменты существующего кода. Поэтому новый функционал на корпоративный портал было решено реализовать с помощью нового микросервиса, который будет общаться с другими микросервисами портала с помощью API.

Сравнение микросервисной и монолитной архитектур. Монолитная архитектура — это традиционная модель создания программного продукта в виде единого модуля, который работает автономно и независимо от других приложений. Микросервисная архитектура противоположна монолитной, поскольку в данном случае архитектура организована в виде ряда независимо развертываемых сервисов. В соответствии с рисунком 1 видно разницу между монолитной и микросервисной архитектурами.

Монолиты полезно использовать на начальных этапах проектов, чтобы облегчить развертывание и не тратить много ресурсов на управление кодом.

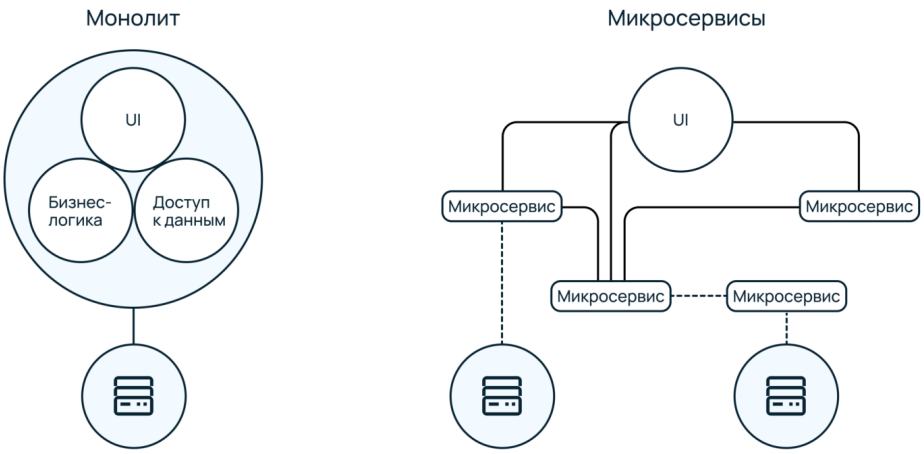

Рисунок 1 — Отличия между монолитом и микросервисами

Когда монолитное приложение становится большим и сложным, возникают трудности с его масштабированием и непрерывным развертыванием, а обновление становится неудобным.

Монолитное приложение создается как единое и неделимое целое, тогда как в микросервисной архитектуре его разбивают на множество независимых модулей, каждый из которых вносит свой вклад в общее дело. Приложение создают как набор независимо развертываемых сервисов, которые являются децентрализованными и разрабатываются независимо друг от друга.

Тестирование программного обеспечения. Тестирование — это процесс, проводимый в основном после разработки. Тестирование программного обеспечения — обширное понятие, которое включает планирование, проектирование и, собственно, выполнение тестов.

К задачам обеспечения качества относятся: проверка технических характеристик и требований к программного обеспечения; оценка рисков; планирование задач для улучшения качества продукции; подготовка документации, тестового окружения и данных; тестирование; анализ результатов тестирования, а также составление отчетов и других документов.

Фазы тестирования. Основными фазами тестирования являются:

- Анализ продукта;
- Работа с требованиями;

- Разработка стратегии тестирования и планирование процедур контроля качества;

- Создание тестовой документации;

- Тестирование прототипа;

- Основное тестирование;

- Стабилизация;

- Эксплуатация.

Все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на следующие группы:

- Функциональные;

- Нефункциональные;

- Связанные с изменениями.

Инструменты для тестирования программного обеспечения. Для того, чтобы тестирование было грамотно проведено, необходимо содержать в порядке тестовую документацию. В компании Terra используется Allure TestOps - инструмент для управления тестированием, в нем могут быть собраны как ручные тесты, так и автоматизированные. Тесты в Allure TestOps могут быть отображены как в папках, так и списком тестов.

Разработка чат-бота для корпоративного портала. Для создания чат-бота на базе Django, который будет отвечать на часто задаваемые вопросы из базы данных, потребуется следующее:

Создать проект Django. Начать нужно с установки Django:

```
pip install django
```

Потом создать новый проект:

```
django-admin startproject faq_bot
cd faq_bot
```

Далее нужно создать новое приложение внутри проекта:

```
python manage.py startapp bot
```

Следующий этап это - настройка модели для FAQ: В файле models.py приложения bot создать модель для хранения вопросов и ответов:

```
from django.db import models

class FAQ(models.Model):
    question = models.CharField(max_length=255)
    answer = models.TextField()

    def __str__(self):
        return self.question
```

Затем применить миграции:

```
python manage.py makemigrations
python manage.py migrate
```

Создать админку для FAQ.

Чтобы легко добавлять и управлять вопросами и ответами, надо зарегистрировать модель в админке. В файле admin.py:

```
from django.contrib import admin
from .models import FAQ
admin.site.register(FAQ)
```

Теперь добавить суперпользователя для доступа к панели администрирования:

```
python manage.py createsuperuser
```

Запустить сервер и добавить вопросы/ответы через админку:

```
python manage.py runserver
```

Перейти по адресу http://localhost:8000/admin/ и добавить записи.

Создать обработку запросов.

Теперь нужно создать логику для получения запросов от пользователей и поиска ответа в базе данных. В файле views.py:

```
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from .models import FAQ
import json
```

```
@csrf_exempt
def chatbot_response(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        user_question = data.get('question', '').lower()

        # Найти соответствующий ответ в базе данных
        faq = FAQ.objects.filter
        (question__icontains=user_question).first()

        if faq:
            return JsonResponse({'answer': faq.answer})
        else:
            return JsonResponse
            ({'answer': 'Извините, я не знаю ответа на этот вопрос.'})
```

Настроить маршруты.

Теперь нужно добавить маршрут для чат-бота. В файле urls.py приложения bot:

```
from django.urls import path
from . import views

urlpatterns = [
    path('chatbot/', views.chatbot_response, name='chatbot_response'),
]
```

И включить эти маршруты в основной файл urls.py проекта:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('bot.urls')),
]
```

Теперь можно отправить POST-запросы к чат-боту:

```
curl -X POST http://localhost:8000/chatbot/ -d
'{"question": "Какой у вас график работы?"}'
-H "Content-Type: application/json"
```

Ответ будет JSON с ответом на вопрос из базы данных.

Тестирование чат-бота для корпоративного портала.

Самый первый и один из важнейших тестов на портале - это проверка авторизации. Если не работает авторизация, значит нет доступа к порталу, а это ведет к тому, что портал не доступен, проводить дальнейшее тестирование нет необходимости.

Далее будет проверена интеграция с разделом постов posts на портале. Посты публикуют сотрудники, эти посты модерируются администратором портала. Посты обычно публикуются на определенные корпоративные темы.

Автотест для проверки интеграции posts выглядит следующим образом:

```
def open_posts_page:
    driver = appium_driver
    with allure.step('Открыть страницу постов'):
        click(driver, locators.posts()["posts"])
    check_element_visibility(driver, locators.posts()
    ["open_posts"], "Кнопка 'Посты'")
    check_element_visibility(driver, locator.button()
    ["push_open_posts"], "Нажать кнопку 'Посты'")
    check_element_visibility(driver, locator.opened_posts()
    ["check_opened_posts"], "Отобразилась надпись 'Лента постов'")
```

Далее будет проверена интеграция с разделом вакансий vacancies на портале.

```
api/v1/vacancies/actions/open/
```

Автотест для проверки интеграции vacancies выглядит следующим образом:

```
def open_vacancies_page:
    driver = appium_driver
    with allure.step('Открыть страницу с вакансиями'):
        click(driver, locators.vacancies()["posts"])
    check_element_visibility(driver, locators.vacancies()
    ["open_vacancies"], "Кнопка 'Вакансии'")
    check_element_visibility(driver, locator.button()
```

```
    ["push_open_vacancies"], "Нажать кнопку 'Вакансии'")
    check_element_visibility(driver, locator.opened_vacancies()
    ["check_opened_vacancies"], "Отобразилась надпись 'Вакансии'")
```

Далее нужно проверить, что страница с новостями пролистывается корректно:

```
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from allure import step
import allure
import time

# Заглушка для данных пользователя
user3 = {
    "email": "user3@example.com",
    "password": "password123"
}

class LoginPage:
    def __init__(self, driver):
        self.driver = driver

    def login(self, email, password):
        with step("Авторизоваться"):
            self.driver.find_element
            (By.NAME, "username").send_keys(email)
            self.driver.find_element
            (By.NAME, "password").send_keys(password)
            self.driver.find_element
            (By.NAME, "submit").click()

@pytest.fixture(scope="function")
def driver():
    driver = webdriver.Chrome()
    driver.set_window_size(1920, 1080)
    driver.get("http://localhost:8000")
```

```
    yield driver
    driver.quit()

@allure.feature("Новости")
@allure.story("Пагинация")
def test_news_pagination(driver):
    login_page = LoginPage(driver)

    # Авторизация
    login_page.login(user3["email"], user3["password"])

    # Ожидание загрузки страницы после авторизации
    time.sleep(2)

    # Переход на страницу новостей
    with step("Перейти на страницу новостей"):
        all_news_button = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable(
            (By.XPATH, "//button[contains(text(), 'Все новости')]"))
        )
        all_news_button.click()

    # Проверка загрузки страницы новостей
    with step("Проверить успешную загрузку страницы новостей"):
        WebDriverWait(driver, 10).until(
            EC.presence_of_all_elements_located(
            (By.CSS_SELECTOR, "[data-qa='listItem:news']"))
        )

    # Получение начального количества новостей
    with step("Получить изначальное количество новостей"):
        initial_news_items = driver.find_elements
        (By.CSS_SELECTOR, "[data-qa='listItem:news']")
        initial_count = len(initial_news_items)

    # Клик по кнопке "Показать еще"
    with step('Кликнуть кнопку "Показать еще"'):
        load_more_button = WebDriverWait
        (driver, 10).until(
            EC.element_to_be_clickable(
```

```
            (By.CSS_SELECTOR, "[data-qa='loadMoreButton']"))
        )
        load_more_button.click()

    # Проверка загрузки новой страницы новостей
    time.sleep(2)  # Задержка для загрузки новых новостей

    # Проверка увеличения количества новостей
    with step("Проверить, что количество новостей увеличилось"):
        updated_news_items = driver.find_elements
        (By.CSS_SELECTOR, "[data-qa='listItem:news']")
        updated_count = len(updated_news_items)
        assert updated_count > initial_count,
        "Количество новостей не увеличилось после нажатия кнопки
        'Показать еще'"
```

Объяснение кода:

1. LoginPage: Класс для авторизации на портале.

2. Фикстура driver: Настраивает WebDriver с заданным размером экрана и открывает страницу.

3. Тест-кейс:

• Выполняет авторизацию и переходит на страницу всех новостей.

• Запоминает начальное количество новостей.

• Нажимает на кнопку «Показать еще» и проверяет, что количество новостей увеличилось.

После успешной проверки интеграций на портале, необходимо провести регрессионное тестирование.

В регрессионное тестирование будет включено сквозное тестирование портала. Отдельно не выделяется сквозное тестирование, так как регрессия подразумевает тестирование всего функционала портала.

Регрессионное тестирование будет выполнено вручную. Тесты с применением ручных настроек быстрее выполнить вручную, чем создать автотесты.

Заключение. Микросервисная архитектура представляет собой эффективный и гибкий подход к разработке программного обеспечения. Она

позволяет компании разделять сложные приложения на более мелкие и легко управляемые сервисы, что повышает масштабируемость и улучшает общую производительность системы.

С использованием микросервисной архитектуры компании могут достичь большей гибкости и легкости в разработке, отладке и масштабировании приложений. Однако такой подход вносит сложности в управление различными сервисами и взаимодействие между ними. Это требует долгосрочного планирования и адекватной организации коммуникации между командами разработчиков.

В данной работе был разработан и успешно протестирован чат-бот на корпоративном портале. На практике получилось создать чат-бот как отдельный микросервис и с помощью API интегрировать на портал. Также были решены следующие задачи:

- Описан функционал корпоративного портала с микросервисной архитектурой;
- Проанализированы виды чат-ботов;
- Выбран вид чат-бота, который подойдет под нужды корпоративного портала;
- Разработан чат-бота на языке программирования Python;
- Протестирован чат-бота для корпоративного портала.

Были приобретены прикладные навыки, на всех этапах разработки было проведено тестирование функционала.