Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

Школа бакалавриата

ОТЧЕТ

По проекту
«Разработка сервиса для адаптации новых сотрудников “WelcomeTo”
(продолжение проекта)»

по дисциплине «Проектный практикум»

Заказчик: Астафьева Анна Викторовна

Куратор: Астафьева Анна Викторовна

Студенты команды: Кощеев Михаил Дмитриевич

Екатеринбург, 2025

Содержание

Введение

Welcome To — это веб-сервис для адаптации сотрудников, включающий гибкий редактор контента, геймификацию и искусственный интеллект для создания адаптационных траекторий. Проект направлен на повышение производительности и вовлеченности сотрудников с первых дней работы.

Наша главная цель - автоматизировать процесс онбординга сотрудников UDV Group, повысить эффективность процесса адаптации сотрудников, снизить нагрузку на HR и наставников, задействованных в адаптации новичков.

Инструмент позволит компаниям увеличить производительность и вовлеченность сотрудников с первых дней работы, что поможет сэкономить время и средства, которые могут впустую расходоваться при длительной адаптации в консервативном подходе.

Особенности продукта: гибкий редактор для добавления разнообразного контента: обучающих материалов, видеоуроков, документов и др.; вариативность геймификации — от превращения привычных задач в квесты, до кастомизированных достижений и специальных welcome-коинов; главная фича продукта — искусственный интеллект, который поможет HR-сотрудникам формировать углубленные адаптационные траектории по отдельно взятым должностям, ориентируясь на уровень знаний сотрудника и его будущие задачи.

Уровни результата (минимальный / базовый / оптимальный):

Минимальный: Реализовано асинхронное взаимодействие между микросервисами и выделен микро-сервис Наполнения траекторий контентом.

Базовый: Дополнительно выделены микро-сервисы АПИ Шлюза и Диспетчера Серверов.

Оптимальный: Полностью реализованы микро-сервисы Аккаунтинга и Статистики, завершена интеграция всех компонентов.

Основная часть

Проделанная работа по ПП в рамках весеннего семестра

Вся работа проделана одним человеком (мной - Кощеевым Михаилом Дмитриевичем), поэтому в дальнейшем я не буду касаться авторства той или иной части проекта

Был выделен новый микросервис, отвечающий за наполнение траекторий и любых других элементов сервиса, использующих данный редактор. Все взаимодейсвтие с редактором реализовано в ассинхронном виде - через кафку. API редактора включает в себя 4 ручки: получение списка элементов документа, перезапись нескольких элементов документа, удаление нескольких элементов документа, и копирование нескольких элементов документа. В данном случае под элементом подразумевается часть редактируемого документа - список, текст, или ссылка на изображение. В сервисе был настроен брокер кафки, созданы топики для общения с другими микросервисами, вот часть кода, отвечающая за инициализацию консюмера кафки:

```
func NewDuplexMessageQueue(producer *sarama.AsyncProducer, consumerGroup *sarama.ConsumerGroup, handler
*Handler) *DuplexMessageQueue {
  newDuplexMessageQueue := DuplexMessageQueue{
    AsyncProducer: producer,
    ConsumerGroup: consumerGroup,
  }

  newDuplexMessageQueue.Handler = &ConsumerGroupHandler{
    Ready: make(chan bool),
    TopicHandlers: map[string]func(*sarama.ConsumerMessage) (any, int, error){
      TopicEditorGetMultiple:    handler.getMultipleData,
      TopicEditorDeleteMultiple: handler.deleteMultipleData,
      TopicEditorPatchMultiple:  handler.patchMultipleData,
      TopicEditorCopyMultiple:   handler.copyMultipleData,
    },
  }

  return &newDuplexMessageQueue
}
```

Был написан middeware для обработки всех запросов сервису, логирования ошибок, и отправку ответа:

```
func (d *DuplexMessageQueue) logAndResponseRequest(fn func(c *sarama.ConsumerMessage) (any, int, error)) func(c
*sarama.ConsumerMessage) {
```

```
  return func(c *sarama.ConsumerMessage) {
    originService, err := retrieveServiceOriginFromConsumerMessage(c)
    if err != nil {
      slog.Error("failed to get service origin header", "topic", c.Topic, "key", string(c.Key), "error", err.Error())
      return
    }

    responseBody, statusCode, err := fn(c)
    if err != nil {
      slog.Error("internal handler error", "topic", c.Topic, "key", string(c.Key), "error", err.Error())
      err = d.responseError(c, err.Error(), responseServicesTopics[originService], statusCode)
      if err != nil {
        slog.Error("failed to send error response to kafka", "topic", c.Topic, "key", string(c.Key), "error", err.Error())
        return
      }
      return
    }

    producerMessage, err := createProducerMessageFromConsumerMessage(responseServicesTopics[originService],
responseBody, c, http.StatusOK)
    if err != nil {
      slog.Error("failed to create kafka message", "topic", c.Topic, "key", string(c.Key), "error", err.Error())
      err = d.responseError(c, err.Error(), responseServicesTopics[originService], http.StatusInternalServerError)
      if err != nil {
        slog.Error("failed to send error response to kafka", "topic", c.Topic, "key", string(c.Key), "error", err.Error())
        return
      }
      return
    }

    err = d.produceMessageWithoutResponse(producerMessage)
    if err != nil {
      slog.Error("failed to send response to kafka", "topic", c.Topic, "key", string(c.Key), "error", err.Error())
      return
    }
    slog.Info("message sent", "topic", c.Topic, "key", string(c.Key))
  }
}
```

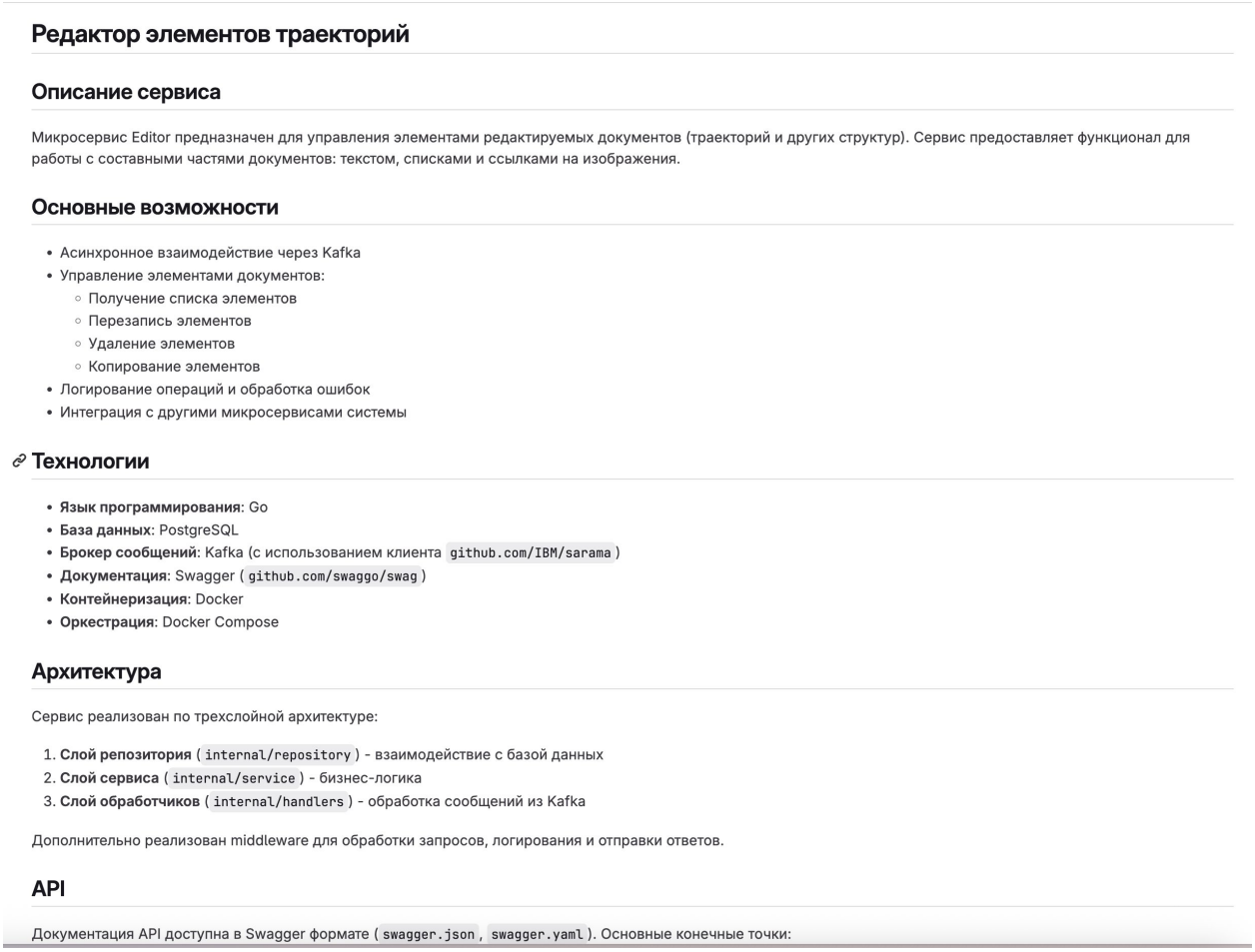
Сервис создан в соответствии с трехслойной архитектурой - отделены методы взаимодействия с БД, сервисные интерфейсы и бизнес логика, и часть сервиса, отвечающая за прием и отправку сообщений.

Логика некоторых методов была сделана асинхронной для оптимальной работы.

Была написана документация API (пусть ей и нельзя пользоваться в полной мере, т.к. она предназначена для http запросов, но можно на нее ориентироваться при подключении новых частей сервиса к библиотеке).

Была настроена контейнеризация, сборка сервиса в registry, и дальнейшее его развертывание вместе со всем сервисом.

Был написан README файл с описанием сервиса



Редактор элементов траекторий

Описание сервиса

Микросервис Editor предназначен для управления элементами редактируемых документов (траекторий и других структур). Сервис предоставляет функционал для работы с составными частями документов: текстом, списками и ссылками на изображения.

Основные возможности

- Асинхронное взаимодействие через Kafka
- Управление элементами документов:
  - Получение списка элементов
  - Перезапись элементов
  - Удаление элементов
  - Копирование элементов
- Логирование операций и обработка ошибок
- Интеграция с другими микросервисами системы

Технологии

- **Язык программирования**: Go
- **База данных**: PostgreSQL
- **Брокер сообщений**: Kafka (с использованием клиента `github.com/IBM/sarama`)
- **Документация**: Swagger (`github.com/swaggo/swag`)
- **Контейнеризация**: Docker
- **Оркестрация**: Docker Compose

Архитектура

Сервис реализован по трехслойной архитектуре:

1. **Слой репозитория** (`internal/repository`) - взаимодействие с базой данных
2. **Слой сервиса** (`internal/service`) - бизнес-логика
3. **Слой обработчиков** (`internal/handlers`) - обработка сообщений из Kafka

Дополнительно реализован middleware для обработки запросов, логирования и отправки ответов.

API

Документация API доступна в Swagger формате (`swagger.json`, `swagger.yaml`). Основные конечные точки:

Помимо нового сервиса, были переработаны существующие - монолит (изначальный сервис, в котором остаются части еще не вынесенные в отдельные репозитории), и гайдер (сервис, хранящий информацию о шаблонах траекторий и индивидуальных образовательных траекториях).

В частности, все межсервисное взаимодействие, за исключением вебсокетов, было перенесено с синхронного взаимодействия по http на асинхронное через kafka. В монолите эти изменения коснулись 53 эндпоинтов, и 57 в гайдере. Также была переделана логика взаимодействия нескольких эндпоинтов - были устранены лишние циклические запросы.

Была настроена kafka с использованием zookeeper, а также подключена kafka ui для визуализации взаимодействия.

ЗАКЛЮЧЕНИЕ

В заключении можно сказать, что в рамках работы над проектом удалось выполнить работу, заявленную на первую контрольную точку. Изначальный план подразумевал крайне активную работу в течение всего семестра - обстоятельства сложились так, что суммарно удалось уделить проекту меньше времени (но все еще крайне много - не меньше, чем уделяют другие люди в рамках работы над их проектами)

Остальная же часть работы, такая как выделение микро-сервисы АПИ Шлюза и Диспетчера Серверов, потребовала бы существенных временных вложений команды, на изучение новых для нее технологий.

Предложения по развитию включают в себя в первую очередь доработку сервиса по плану 2 и 3 контрольных точек, и, в дальнейшем, анализ рынка и конкурентов, для актуализации техник, применяемых для привлечения, удержания пользователей, а также улучшения пользовательского опыта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Высоконагруженные приложения. : книга о проектировании распределенных систем / М. Клеппман : o'reilly, 2024. – 640с.

2. Apache Kafka: обзор : статья об использовании и сферах применения kafka : Издательский дом «Питер»

Приложение А

Планируемая схема межсервисного взаимодействия