

Прототип поисковой В2В-платформы, связывающей поставщиков и потребителей

Платформа: ООО “Отус онлайн-образование”

Курс: Разработчик на Spring Framework

Период обучения: 30 мая – 30 декабря



Меня хорошо видно
& слышно?



Защита проекта

Тема: Разработка поисковой B2B-платформы, связывающей поставщиков и потребителей, на основе микросервисной архитектуры и RabbitMQ

О себе



- **ИП Николаев Александр Вадимович**
- МГТУ им Н.Э. Баумана. Специальность “Гидромашины, гидропривод и гидропневмоавтоматика”
- ООО “УНИВЕРСИТЕТ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА” профессиональная переподготовка. Специальность “Python разработчик”
- МГТУ им Н. Э. Баумана профессиональная переквалификация. Специальность “Инженер данных (Data engineer)”
- +79269079898
- mail@alexnika.ru

План защиты

Цель и задачи проекта

Описание проекта,
архитектурные схемы и
MVP

Логические блок-схемы
микросервисов

Live demo

Что получилось и выводы

Вопросы и рекомендации



Цель и задачи проекта

Цель проекта: создать несколько микросервисов с синхронной связью по RESTful API и асинхронной связью между ними на основе брокера сообщений RabbitMQ

Задачи:

1. Разработать микросервисную архитектуру и выделить MVP для проектной работы по курсу;
2. Реализовать микросервисы EUREKA-SERVER, CONFIG-SERVER, API-GATEWAY, AUTH-SERVICE, CLIENT-SERVICE;
3. Реализовать микросервисы ETLDATA-SERVICE и SEARCH-SERVICE;
4. Связать микросервисы между собой с помощью RESTful API и брокера сообщений RabbitMQ;
5. Убедится, что задуманная цель осуществлена и связь между микросервисами работает;
6. Наметить дальнейшие шаги для превращения проектной работы в полноценный PET-проект.



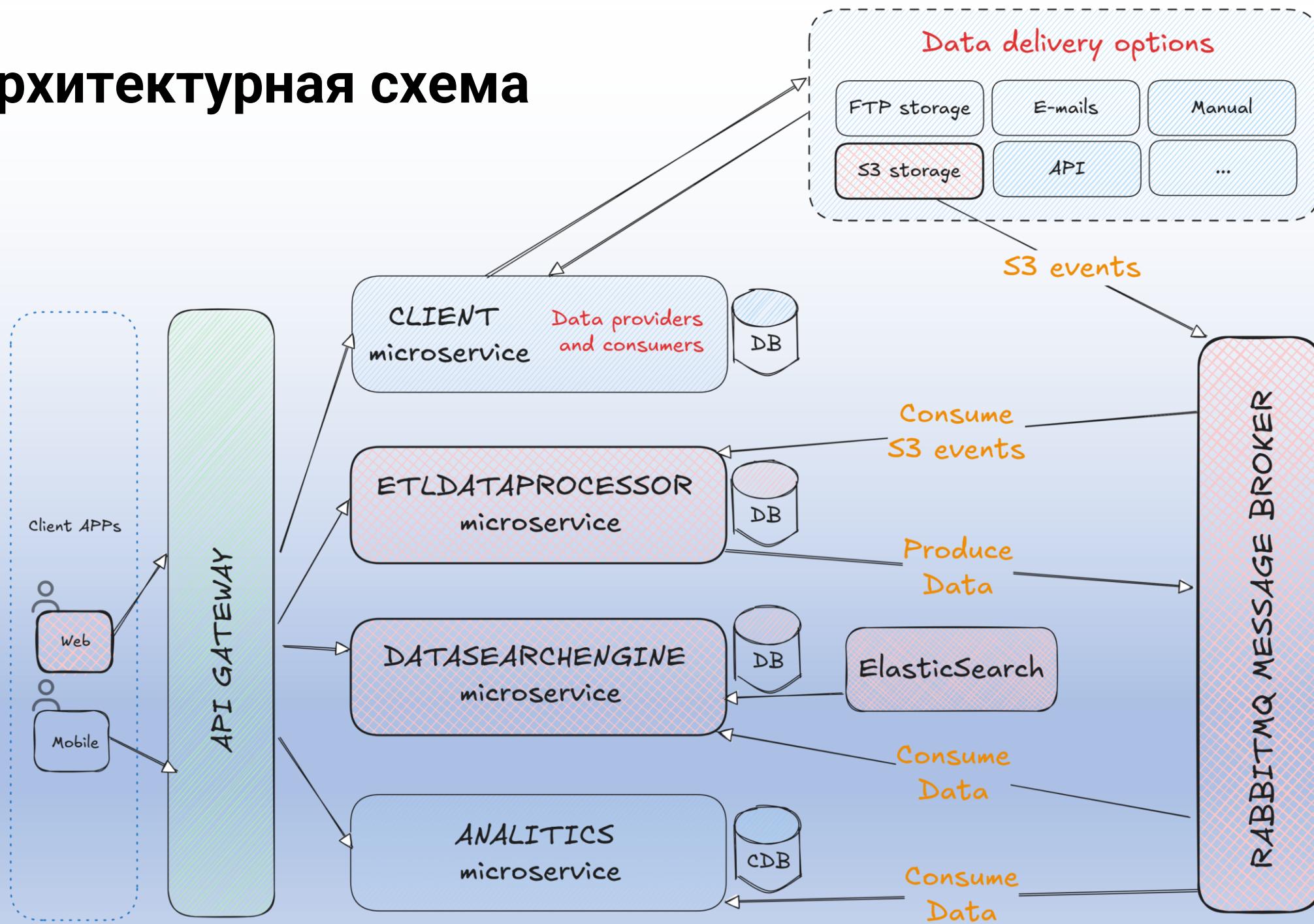
Краткое описание проекта

База:

1. Поставщики предоставляют цены и остатки (прайс-листы) на складах по товарам и/или услугам.
2. Информация может предоставляться в любом удобном поставщику виде: загрузка в S3 или FTP хранилище, E-mail или Telegram рассылка, собственный API и т.д.
3. Сервис PRICAT.RU предоставляет доступ к обработанной информации по ценам и остаткам в структурированном виде для любых клиентов, которые подключились к сервису.



Архитектурная схема



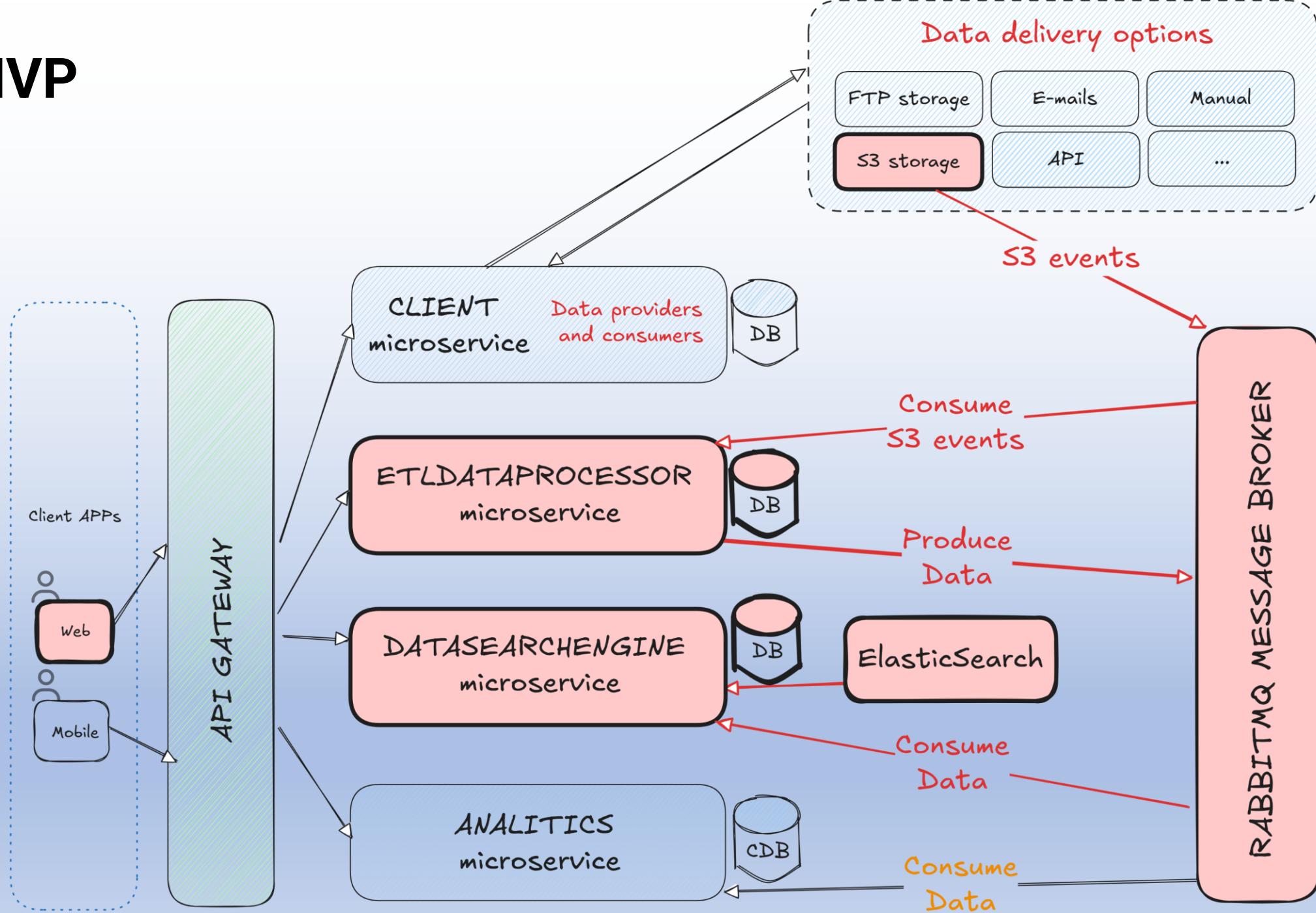
Выделение MVP

Минимально работающий продукт отображает концепцию:

1. Компании – поставщики данных заранее определены;
2. Прайс-листы заданы в едином облегченном JSON формате и размещаются в S3 хранилище через CLIENT-SERVICE с помощью S3Minio API;
3. Формат сообщений для микросервисов упрощен, чтобы можно было показать суть работы PRICAT;
4. Полнотекстовый поиск реализован не по всем полям;
5. Health check, observability, logging реализованы на минимальном уровне;
6. Отказоустойчивость реализовывалась в микросервисах CLIENT-SERVICE и SEARCH-SERVICE.



MVP



Используемые технологии

1. Java 25 + Java Spring Boot 3.5.7
2. PostgreSQL 18
3. RabbitMQ 4.1.3
4. MINIO 2025-04
5. Elasticsearch 8.18.6
6. Docker 4.48.0



RabbitMQ

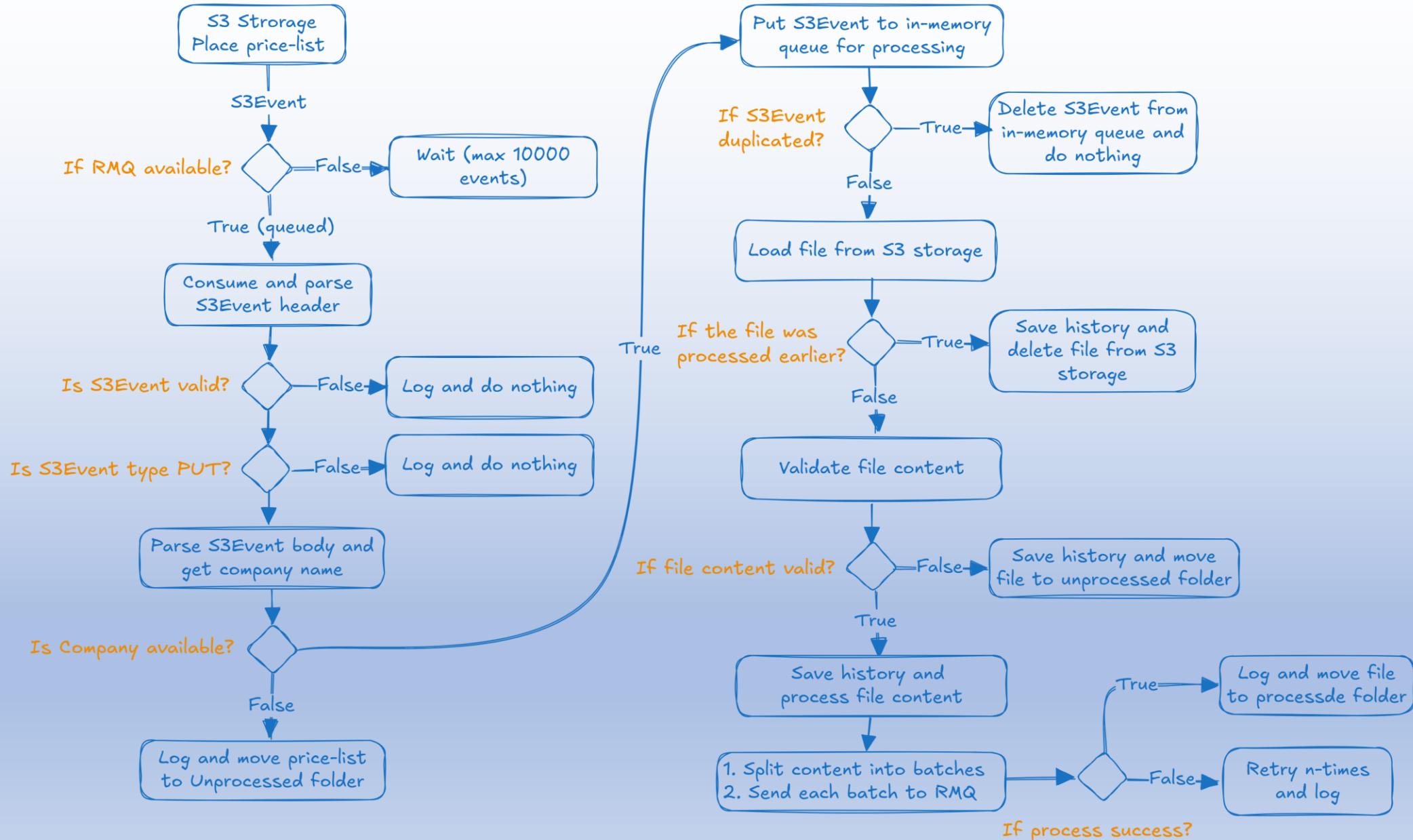


spring
boot



MINIO

Логическая схема ETLDATA-SERVICE (упрощенная)



Логическая схема SEARCH-SERVICE (упрощенная)

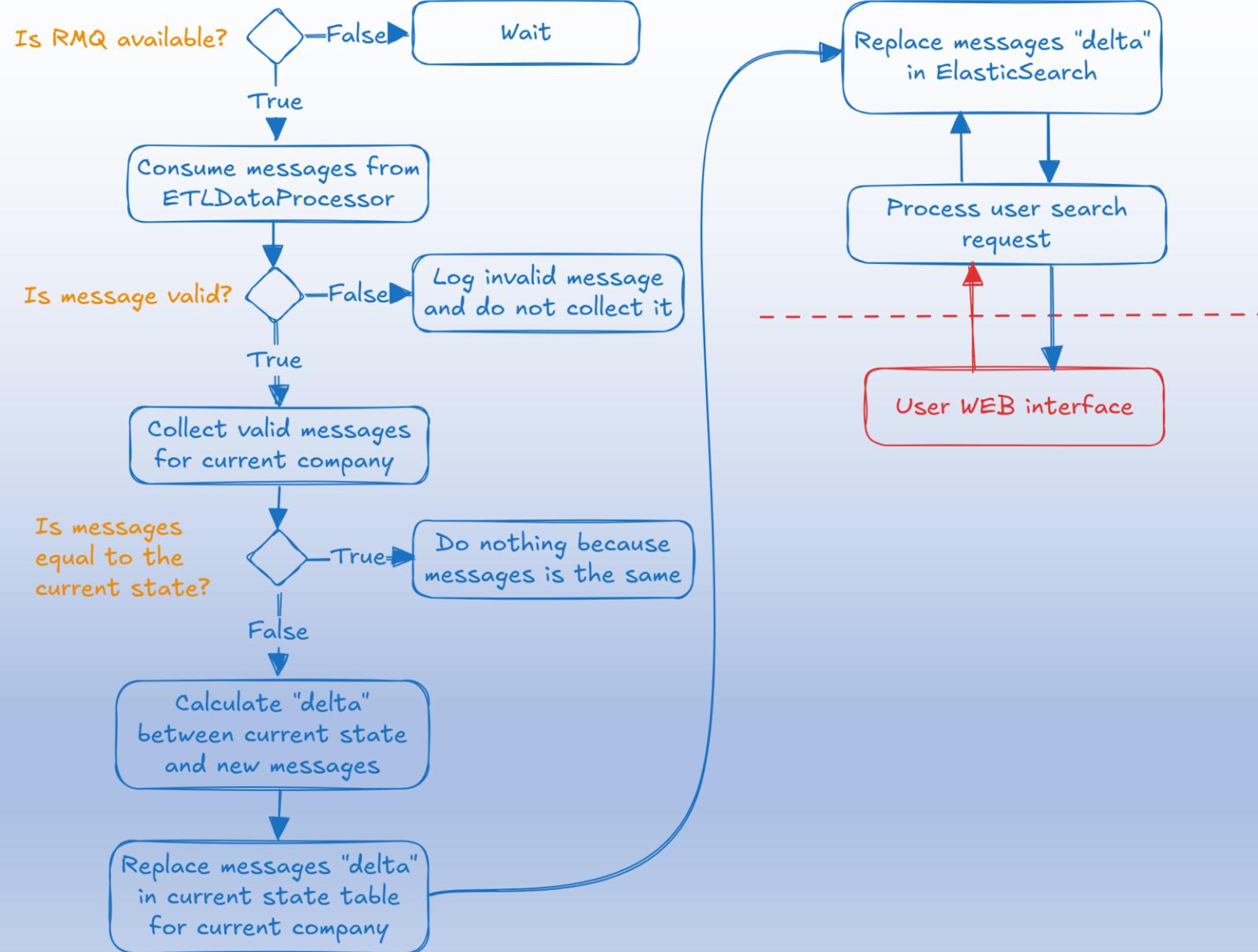
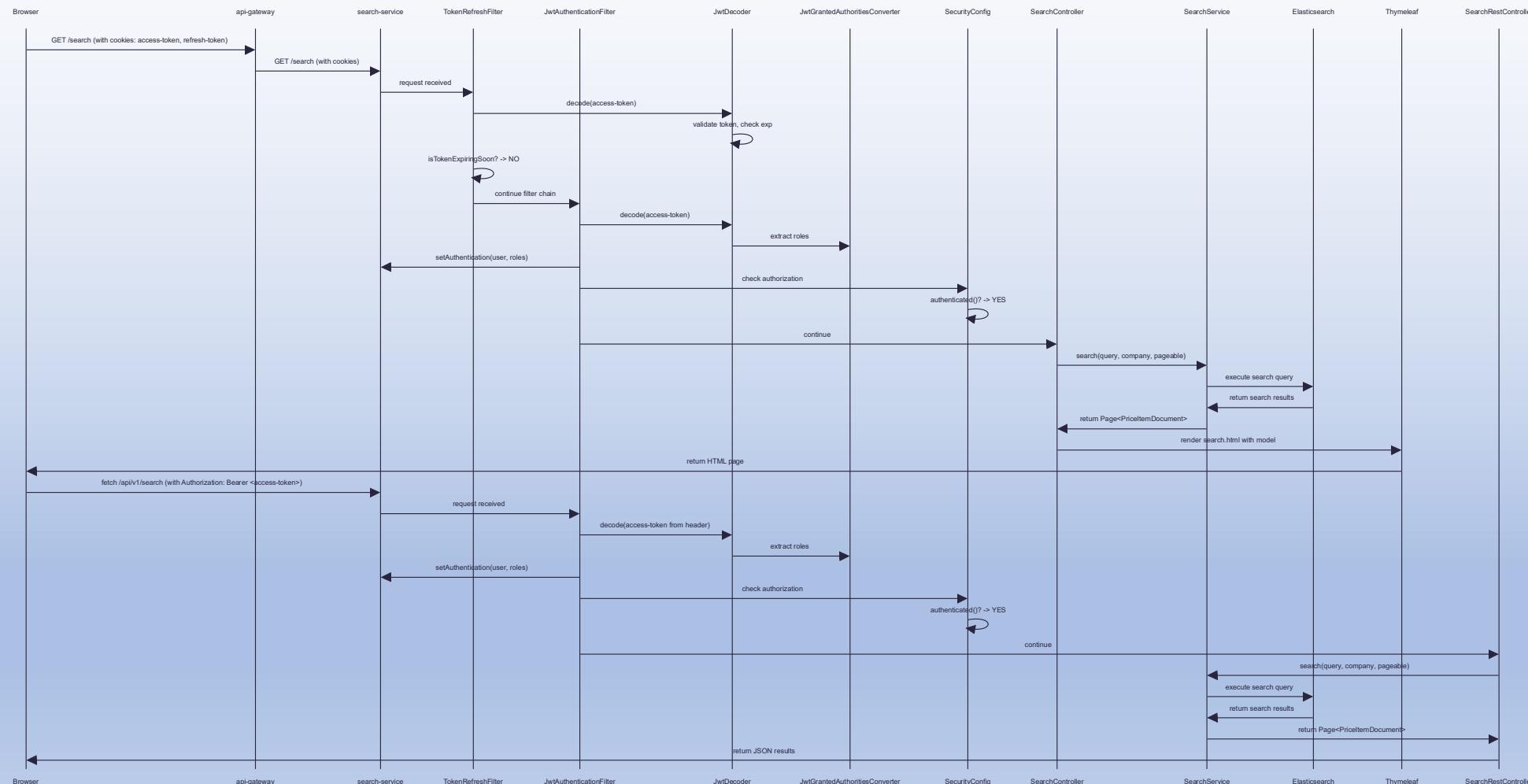


Диаграмма последовательности безопасности CLIENT-SERVICE

Сценарий - пользователь заходит на <http://localhost:8080/search>, access-token валиден, refresh-token валиден, доступ разрешён.



Что показывает диаграмма?

1. access-token валиден → токены не обновляются.
2. TokenRefreshFilter пропускает запрос.
3. JwtAuthenticationFilter проверяет токен → доступ разрешён.
4. SearchController и SearchRestController обрабатывают запросы.
5. Данные возвращаются пользователю.



Связь микросервисов по RabbitMQ

Микросервис	Тип	Exchange	Queue	Retry	DLQ
S3Minio	Producer	Direct	Classic	no	no
ETL DATA-SERVICE	Consumer			no	no
ETL DATA-SERVICE	Producer	Topic	Quorum	yes	no
SEARCH-SERVICE	Consumer			no	yes



Гарантии доставки и риски

Компонент	Гарантия доставки	Риски
ETL DATA-SERVICE. S3EventConsumer	At-Least-Ones	Дубли, блокировка
ETL DATA-SERVICE. RabbitMQBatchService	At-Least-Ones	Потери при падении
SEARCH-SERVICE. DataItemConsumer	At-Least-Ones	Нет обработки DLQ



Live demo

```
package ru.pricat.etldataprocessor;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.boot.context.properties.ConfigurationPropertiesScan;  
import ru.pricat.etldataprocessor.config.EnvLoader;  
  
@SpringBootApplication  
@ConfigurationPropertiesScan  
public class EtlDataProcessorApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(EtlDataProcessorApplication.class, args);  
    }  
}
```

Project

- ETLDataProcessor D:\OTUS\RabbitMQ\Diploma-work\PRICAT.RU\ETLDataProcessor
- .idea
- .mvn
- logs
- src
 - cat
 - etldataprocessor
 - config
 - consumer
 - converter
 - exception
 - indicators
 - model
 - repository
 - service
 - shell
 - EtlDataProcessorApplication
 - resources
 - test
 - target
 - .env
 - .gitattributes
 - .gitignore
 - docker-compose.yml
 - Dockerfile
 - ETLDataProcessor.log
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
- External Libraries



Что получилось?

1. Получилось создать 7 микросервиса с асинхронной передачей данным между ними через брокер сообщений RabbitMQ и Rest API для других взаимодействий.
2. Получилось настроить Eureka server, попробовать Config server, API gateway.
3. В RabbitMQ получилось использовать разные типы обменников и очередей, выбирая в зависимости от решаемой задачи.
4. Получилось настроить гарантию доставки с текущим уровнем надежности по моей оценке 6/10:
 - ✓ Retry механизм работает, дубликаты сообщений обрабатываются;
 - ✓ Quorum очередь устойчива к сбоям нод;
 - ✓ Есть DLQ для основных данных;
 - ✗ Нет end-to-end подтверждений;
 - ✗ Нет защиты от потерь при падении приложения;
 - ✗ Нет мониторинга failed сообщений.
5. Получилось точно понять, что нужно доработать и улучшить в разрабатываемых микросервисах.

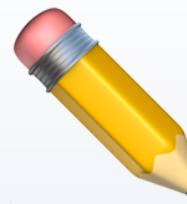


Что улучшить в работе с RabbitMQ?

1. Для ETLDATA-SERVICE как потребителя S3Events настроить DLQ.
2. Для ETLDATA-SERVICE как поставщика PricelItemMessage настроить publisher confirms и returns.
3. Реализовать outbox pattern для критичных сообщений.
4. Настроить мониторинг и алертинг по DLQ.
5. Для SEARCH-SERVICE решить, что делать дальше с сообщениями попавшими в DLQ. Сейчас происходит просто логирование.
6. Для выпуска JWT токенов перейти от единого ключа подписи на публичный и приватный ключи издателя (issuer).
7. Для RESTful API запросов, охватывающих 2 микросервиса (например удаление пользователя в auth-service и его профиля в client-service) реализовать паттерн Saga или компенсационные транзакции
8. Улучшить UI собрав все воедино.



Ваши вопросы и рекомендации



Спасибо за внимание!