wrike

К микросервисам через reverse engineering и кодогенерацию

Вячеслав Тютюньков, BE Tech Lead at Wrike
LinkedIn

Wrike – Лидер на рынке Work Management

Our collaborative workflow management platform helps teams gain visibility, simplify planning, enable collaboration, and streamline their workflow.

Основан
в 2006

9 Офисов

20,000+
Компаний-
клиентов

1100+
Сотрудников

5 лет
в Fast 500

- **2M+** пользователей в **140+** странах на **9** языках
- **1B+** задач, папок и проектов было создано
- Датацентры в США, Европе и GCP

Leader in Collaborative Work Management Tools for the Enterprise

Forrester Research **2016, 2018**

Customers' Choice for Project Portfolio Management Worldwide

Gartner Peer Insights **2018**

Commercial Innovator of the Year

Microsoft Windows Developer Award **2018**

Customer Experience Award for Complete Software and Service Experience

Software Reviews **2018**

20,000+

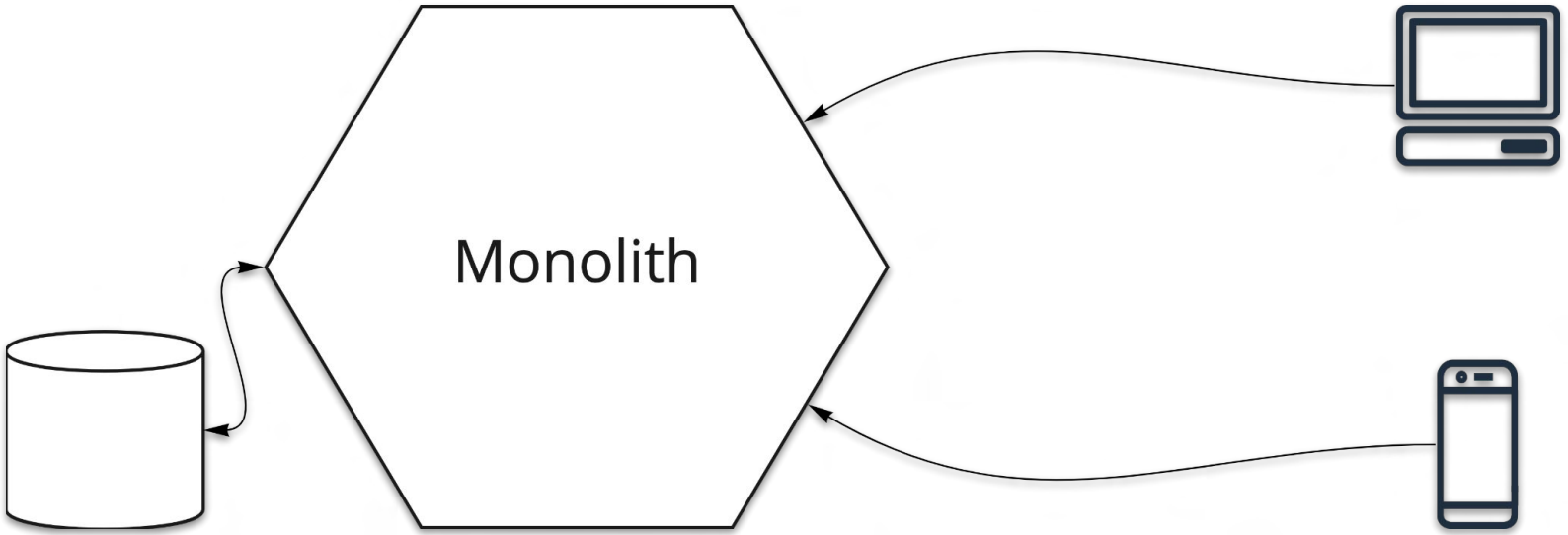
компаний выбирают Wrike

 К микросервисам через reverse engineering и кодогенерацию.

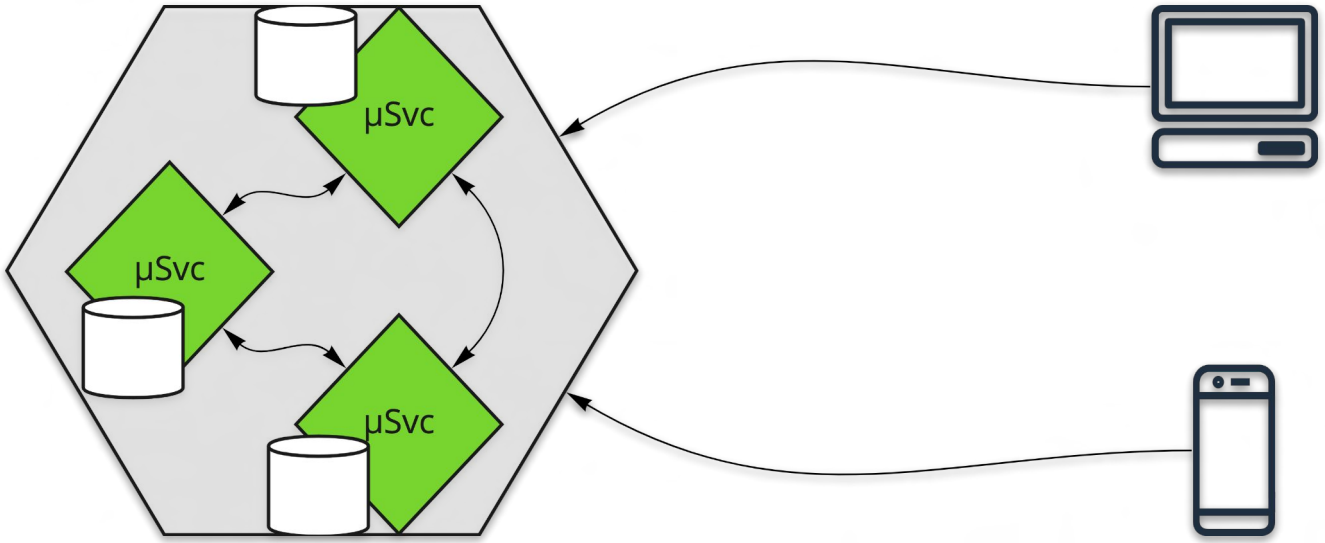
О чем сегодня поговорим

- Микросервисы
- Как построить BFF
- Как построение BFF приближает нас к микросервисам

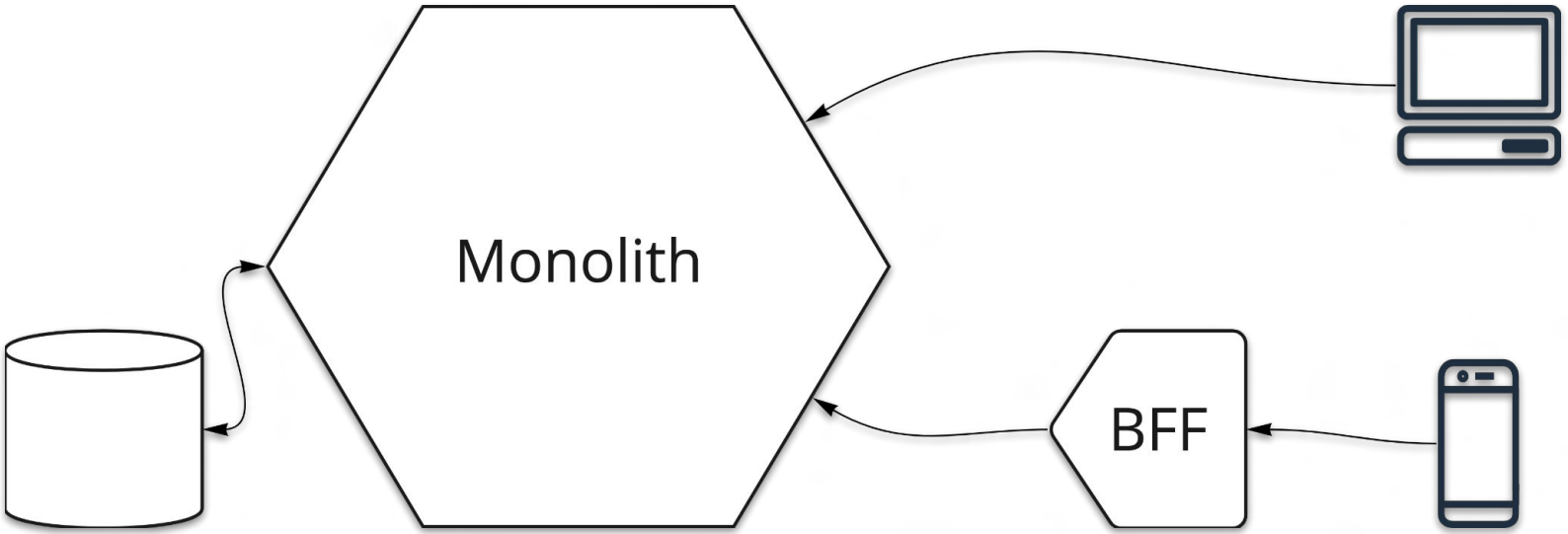
Как выглядит система сейчас



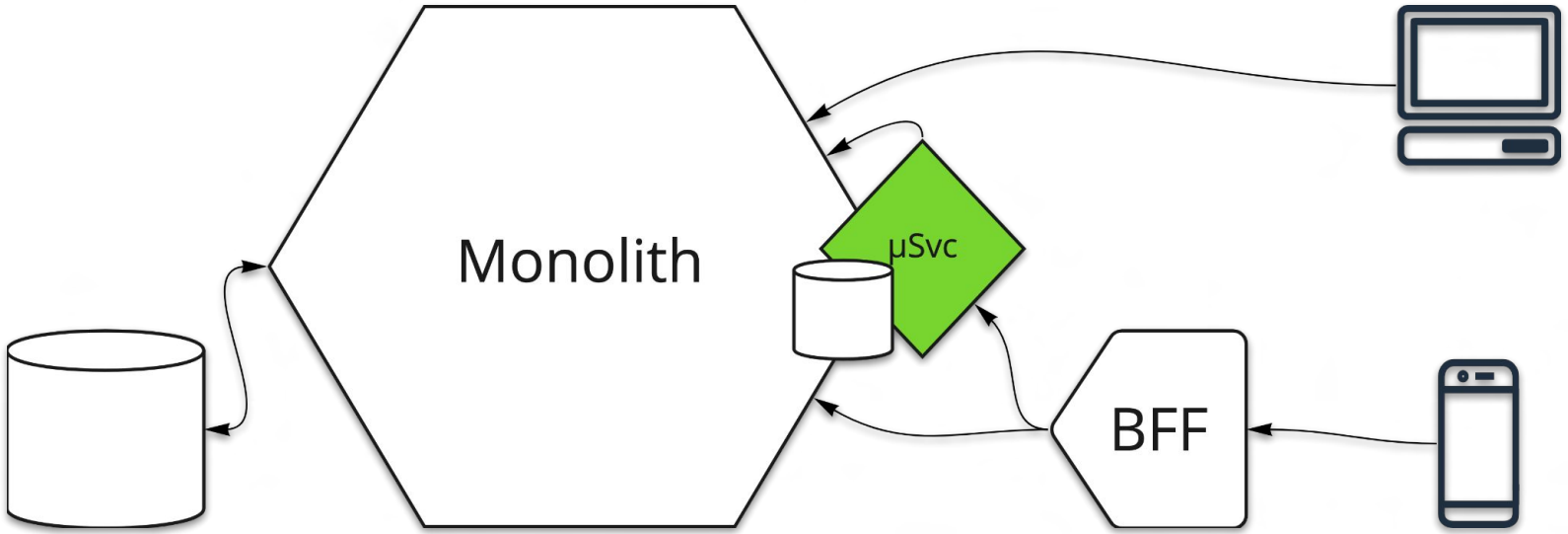
К чему хотим прийти



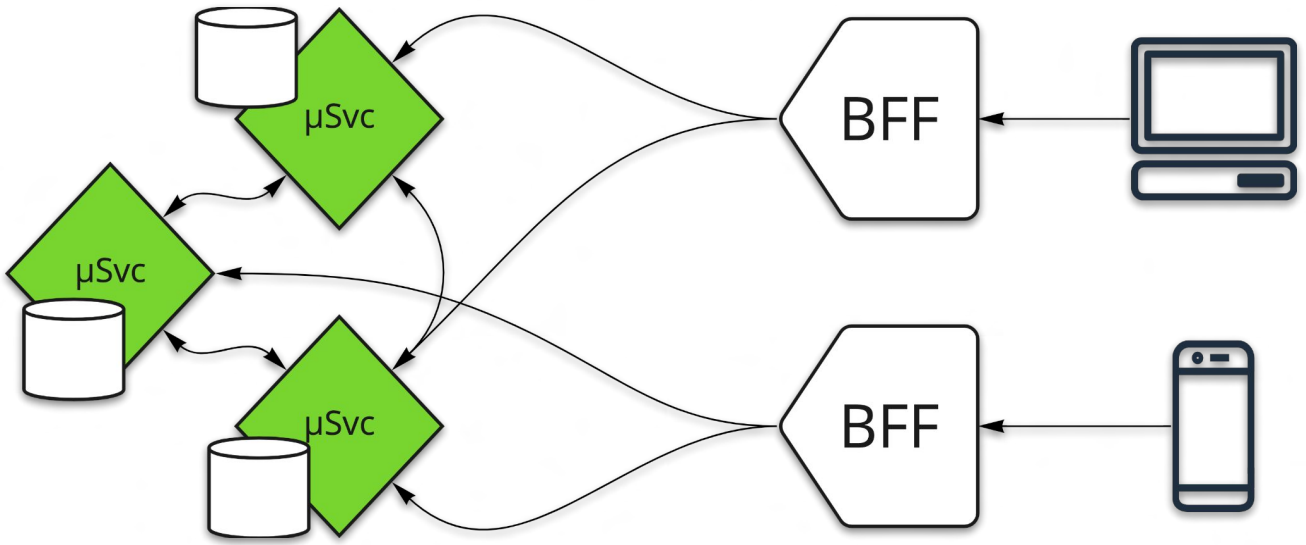
С чего начать



Что делаем потом



И как в итоге все должно получиться



О чем на самом деле пойдет речь

- Как подготовиться к работе
- Реверс инжениринг
- Кодогенерация
- Сложности решения и оптимизация
- Что в итоге получилось

wrike

Подготовка

О чем нужно договориться

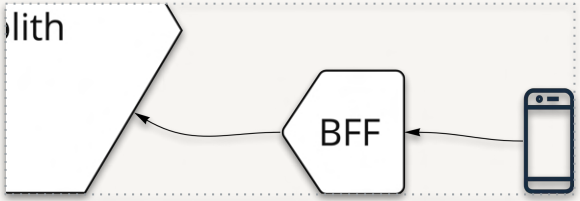
- Протокол взаимодействия
  - REST + JSON
- Библиотеки
  - Retrofit2 + Jackson
- Описание схемы
  - OpenAPI
- Организация процесса
  - schema first
  - spring boot/mvc

Выбираем «жертву»

Почему BFF для андроид приложения

- Отсутствие собственного «домена»
- Узкоспециализированное API
- Отдельный релизный цикл
- Особые требования обратной совместимости

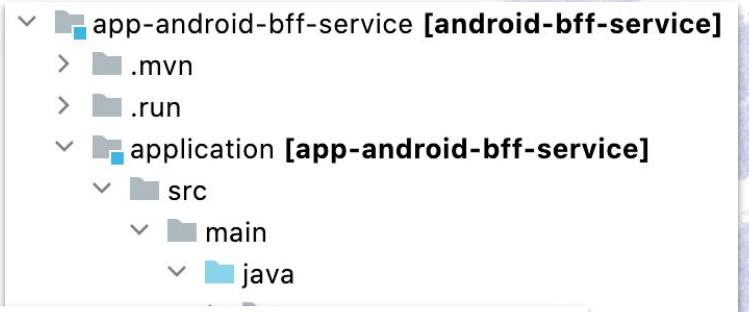
Засучиваем рукава и начинаем «копать»



Шаг первый

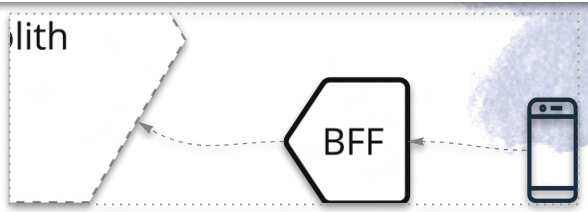
Нужен сервис — BFF

- Создали сервис
- Настроили
- Запустили
- Работает




```
management.server.port=8081
management.endpoints.web.base-path=/api/v1
management.endpoint.health.probes.enabled=true
management.endpoints.web.exposure.include=info,h2
```

```
main] o.a.c.c.StandardService               : Starting service [Tomcat]
main] o.a.c.c.StandardEngine                : Starting Servlet engine: [Apache Tomcat/9.0.48]
main] o.a.c.c.C.[.[.[/]                     : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in
main] o.s.b.a.e.w.EndpointLinksResolver     : Exposing 4 endpoint(s) beneath base path '/api/v1'
main] o.a.c.h.Http11NioProtocol             : Starting ProtocolHandler ["http-nio-8081"]
main] o.s.b.w.e.t.TomcatWebServer           : Tomcat started on port(s): 8081 (http) with context path
main] c.w.c.a.AndroidBffServiceApplication  : Started AndroidBffServiceApplication in 6.213 seconds (J
```



Шаг второй

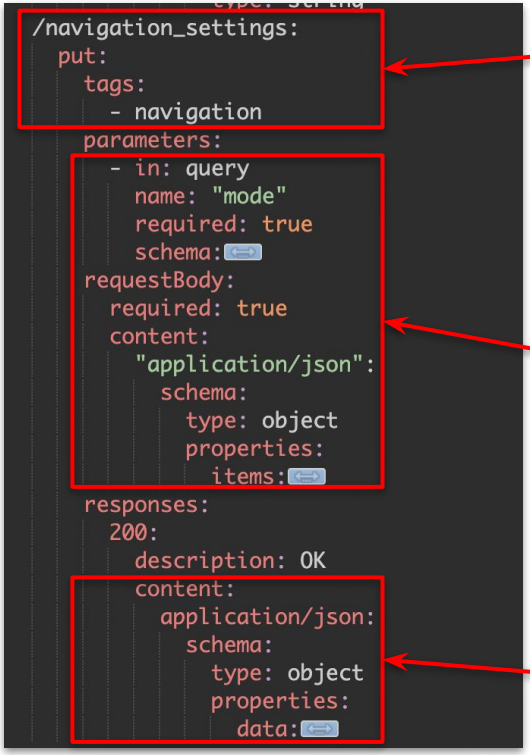
Нужно где-то брать данные

- Использовать кастомный «клиент»
- Использовать готовую схему
- Создать схему

Шаг второй

Как создать схему

- Описать схему вручную
- Использовать готовое решение
- Автоматизировать получение схемы

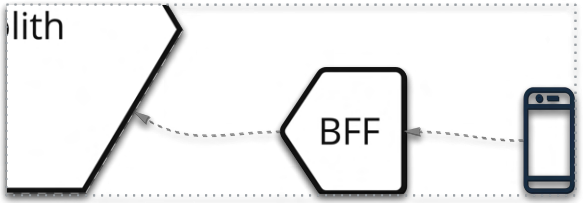



```
@HandlerMetaInfo(
        tags = "navigation",
        path = "api/navigation_settings",
        method = HttpMethod.PUT,
        securitySchemas = {}
)
public class PutNavigationSettings implements SchemaHandler<Input, Output> {

    protected Input parseRequest(final HttpServletRequest request) {...}

    protected Output processRequest(final Input input) {...}

    static class Input {
        private final int mode;
        private final List<NavigationItem> items;

        Input(final int mode, final List<NavigationItem> items) {
            this.mode = mode;
            this.items = items;
        }

        public int getMode() { return mode; }

        public List<NavigationItem> getItems() { return items; }
    }

    static class Output {
        private final List<NavigationItem> items;

        Output(final List<NavigationItem> items) { this.items = items; }

        public List<NavigationItem> getItems() { return items; }
    }
}
```

Шаг второй

Генерируем схему

- Пишем библиотеку
- Собираем список эндпоинтов
- Получаем схему
- Публикуем



Шаг третий

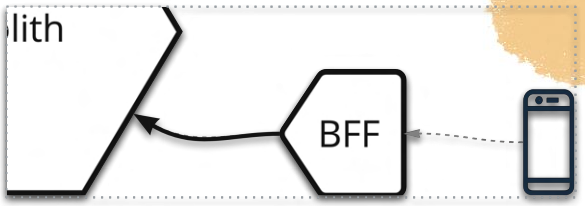
Из схемы в код

- Процессинг схемы
  - openapi-generator
- Модели + маппинг
  - Jackson
- Клиент для REST
  - Retrofit2

```java
@JsonPropertyOrder({
        JSON_PROPERTY_MODE,
        JSON_PROPERTY_ITEMS
})
public class PutNavigationSettingsDto {
    public static final String JSON_PROPERTY_MODE = "mode";
    public static final String JSON_PROPERTY_ITEMS = "items";

    private final int mode;
    private final List<NavigationItem> items;

    @JsonCreator
    private PutNavigationSettingsDto(@JsonProperty(JSON_PROPERTY_MODE) int mode,

        this.mode = mode;
        this.items = items;
    }

    @JsonGetter(JSON_PROPE
    public int getMode() {

    @JsonGetter(JSON_PROPE
    public List<Navigation
```

```java
@JsonPropertyOrder({
        JSON_PROPERTY_ITEMS
})
public class PutNavigationSettingsResponseDto {
    public static final String JSON_PROPERTY_ITEMS = "items";

    private final List<NavigationItem> items;

    @JsonCreator
    private PutNavigationSettingsResponseDto(@JsonProperty(JSON_PROPERTY_ITEMS) List<NavigationItem> items) {...}

    @JsonGetter(JSON_PROPERTY_ITEMS)
```

```java
public interface NavigationSettingsServiceGateway {

  @PUT("navigation_settings")
  RetrofitCall<PutNavigationSettingsResponseDto> putNavigationSettings(@Header("x-w-auth") String authToken,
                                                                       @Header("x-w-account") IdOfAccount accountId,
                                                                       @Body PutNavigationSettingsDto input) throws IOException;
```



Шаг четвертый

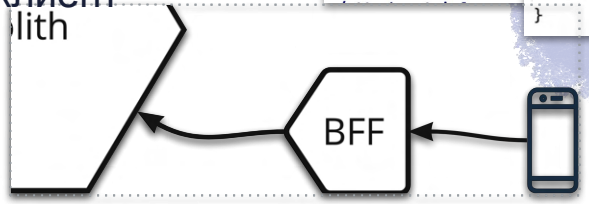
Проксируем

- Описываем схему BFF
- Генерируем интерфейсы
  - spring-mvc
- Генерируем модели
  - Jackson
- Проксируем запросы
  - используем Retrofit2 клиент

```
@Validated
public interface Navigatio

    @RequestMapping(value
            produces = {"a
            consumes = {"a
            method = Reque
    default ResponseEntity

        return new Respons
    }
}
```

```
@Override
public ResponseEntity<WrikeResponseDto> navigationSettingsPut(final
    final Response<PutNavigationSettingsResponseDto> response;
    try {
        final WrikeToken wrikeToken = principal.getWrikeToken();
        final String authToken = wrikeToken.getBearerToken();
        response = navigationSettingsApi.putNavigationSettings(
                authToken,
                wrikeToken.getRequestAccountId().get(),
                PutNavigationSettingsDto.builder()
                        ...
        ).execute();
    } catch (IOException e) {
        log.error("", e);
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERRO
    }

    if (response == null) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERRO
    }
    if (!response.isSuccessful()) {
        return ResponseEntity.status(response.code()).build();
    }

    final PutNavigationSettingsResponseDto data = response.body();
    return ResponseEntity.ok(

    );
}
```

```
required: true
content:
  "application/j
    schema:
      $ref: '#/c
responses:
  200:
    description: C
    content:
      "applicatior
        schema:
          $ref: '#
```



Анализируем, что получилось

Что имеем

BFF на проде, работает стабильно

- Специфика эндпоинтов монолита
  - Нужно передавать токен авторизации
  - Нужно передавать дополнительные заголовки
- Бойлерплейт обработки
  - Нужно проверять статус ответа
  - Нужно явно обращаться за респонс данными

```
@Override
public ResponseEntity<WrikeResponseDto> navigationSettingsPut(final
    final Response<PutNavigationSettingsResponseDto> response;
    try {
        final WrikeToken wrikeToken = principal.getWrikeToken();
        final String authToken = wrikeToken.getBearerToken();
        response = navigationSettingsApi.putNavigationSettings(
                authToken,
                wrikeToken.getRequestAccountId().get(),
                PutNavigationSettingsDto.builder()
                        ...
        ).execute();
    } catch (IOException e) {
        log.error("", e);
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERRO
    }

    if (response == null) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERRO
    }
    if (!response.isSuccessful()) {
        return ResponseEntity.status(response.code()).build();
    }

    final PutNavigationSettingsResponseDto data = response.body();
    return ResponseEntity.ok(

    );
}
```

Что хотим изменить

- Избавиться от бойлерплейта
- Не передавать/не заполнять «общие» параметры
- Абстрагироваться от протокола и маппинга данных
  - Работать с микросервисом как с обычным бином
  - Не завязывать интерфейсы на конкретные имплементации REST (Retrofit2)
  - Убрать маппинг (Jackson) из описаниях моделей
- Оставить возможность работать на «низком уровне»

«Тюним» кодогенерацию

Разбиваем кодогенерацию на два этапа (слоя)

- Первый — интерфейс микросервиса
  - API сервиса
  - Модель данных
- Второй — имплементация «транспорта»
  - Retrofit2 клиент
    - дополнительные адаптеры
  - Jackson mixin-ы для моделей
  - Дефолтная имплементация интерфейса через вызовы Retrofit2 клиента
  - Дефолтная конфигурация Jackson, Retrofit2, etc

Что получилось в итоге

- Убрали бойлерплейт обработки http/rest
- Вынесли отдельно модель и интерфейс сервиса
- Код обработчика в BFF выглядит чистенько
- Используем текущий подход для других микросервисов
    - На первом этапе использовали только первый «слой»
    - Вместо транспорта подставляем локальные бины

Выводы

Чему мы научились

- BFF на проде
- Второй сервис на подходе

- Выбор подхода — важный шаг при перестроении системы
- Реверс инжениринг — хороший способ описать текущую систему
- Кодогенерация упрощает жизнь и избавляет от бойлерплейта

wrike

К микросервисам через reverse engineering и когодегенерацию

habr

wrike

Through reverse engineering and code generation to microservices

medium.com

Больше о нас!

Читайте о нас на Wrike TechClub

wrike