

Platform V: микросервис за 15 минут

Базовые прикладные сервисы

Exported on 11/18/2021

Table of Contents

1 Вступление	4
2 Работа в SmartMarket Studio:	5
3 Приложение "Промоакция"	6
4 Знакомство с Platform V DataSpace	7
5 Подписание запросов (авторизация)	8
6 Архитектура приложения "Промоакция"	10
7 Function 1 Admin Frontend.....	11
7.1 Разработка	11
8 Знакомство с Platform V Functions.....	27
9 Публикация	28
10 Function 2 Client Frontend/ Function 3 Client Backend.....	33

Biryukov Victor

<https://sber-tech.com/>

Platform V¹ developer

email: vvbiryukov.sbt@sberbank.ru²

Telegram: @birvictor

¹ <https://platformv.sber.ru/#/>

² <http://sberbank.ru>

1 Вступление

1. Быстрая разработка приложения/микросервисный подход.
Основной посыл в том, чтобы по итогам прочтения статьи можно было начать быстро и удобно писать полноценные микросервисы (фронт+бэк+хранилище данных).
Рассматриваемый пример будет написан на TypeScript/JavaScript, но это требование не обязательное.
Код функций может быть также написан на Java или Python.
А обращение к DataSpace через GraphQL не накладывает в принципе каких-либо ограничений на клиента.
2. Статья разбита на две части:
 - a. В первой познакомимся с сервисом Platform V DataSpace, напомним frontend-приложение, используя DataSpace как сервис ([Backend-as-a-Service](https://en.wikipedia.org/wiki/Mobile_backend_as_a_service)³)
 - b. Во второй познакомимся с сервисом Platform V Functions, напомним backend-приложение как облачную функцию и разместим наше frontend-приложение также как функцию ([Function-as-a-Services](https://en.wikipedia.org/wiki/Function_as_a_service)⁴)

³ https://en.wikipedia.org/wiki/Mobile_backend_as_a_service

⁴ https://en.wikipedia.org/wiki/Function_as_a_service

2 Работа в SmartMarket Studio:

- Заходим на сайт: <https://developers.sber.ru/studio/login>
- Регистрируемся или входим по Сбер ID: можно через QR-код в приложении СберБанк Онлайн
- Создаем "Личное пространство"
- В созданном пространстве жмем "Создать проект", где выбираем "Platform V DataSpace"
- Вы попали в визуальный редактор конструирования модели данных.

Далее в разделе "Знакомство с DataSpace" в статье будет описано, как формировать модель данных предметной области вашего приложения, выпускать соответствующий сервис.

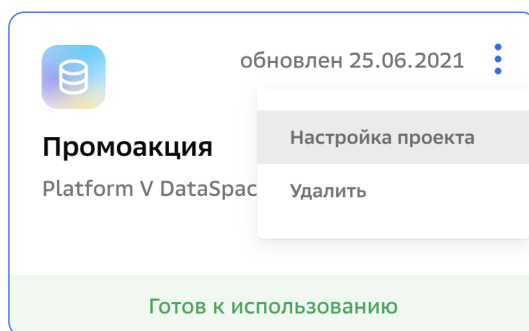
В принципе можете сразу же загрузить готовую модель данных для приложения "Промоакция": [model.xml](#)⁵ и нажать кнопку "Выпустить".

После выпуска сервиса по данному адресу https://smapi.pv-api.sbc.space/fn_d2527eab_2999_4a9a_99b1_4f90bf815b54

можно начать работы в приложении в роли администратора, где в качестве хранилища выступит ранее выпущенный Вами сервис DataSpace.

Нажав кнопку "Login page" необходимо указать данные авторизации: адрес/логин/пароль будут доступны в настройках вашего проекта DataSpace: адрес проекта, app_key, app_secret соответственно.

Проекты



По факту пройдя по ранее указанному [адресу](#)⁶ Вы уже воспользовались моим сервисом Platform V Functions, получив статистику web-приложения, работающего напрямую уже с Вашим DataSpace. Более детально поговорим об этом во второй части статьи в разделе "Знакомство с Functions".

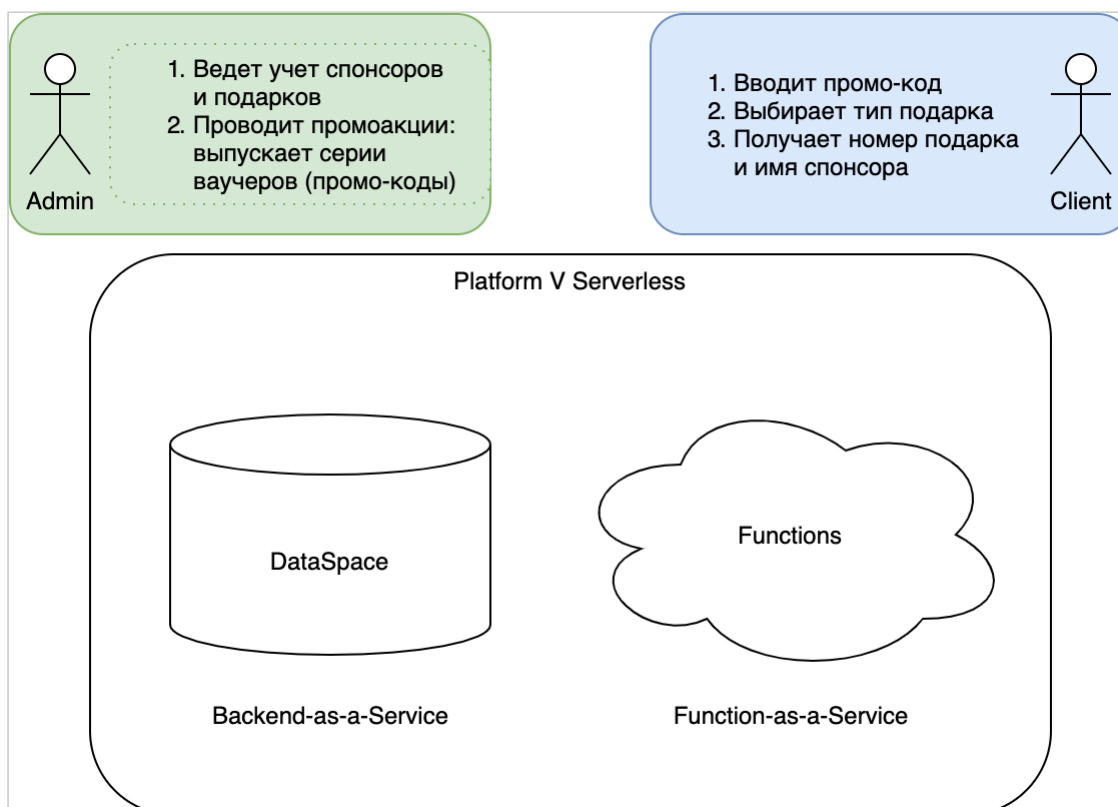
⁵ <https://github.com/VictorBiryukov/promo-action/blob/release/model.xml>

⁶ https://smapi.pv-api.sbc.space/fn_d2527eab_2999_4a9a_99b1_4f90bf815b54

3 Приложение "Промоакция"

Для наглядности примера использования платформенных сервисов будем разрабатывать небольшое приложение: "Промоакция"

Схема сервиса:



Два вида пользователей системы: Администратор и Клиент

Для разработки сервиса воспользуемся двумя сервисами Platform V (уже доступными разработчику в SmartMarket): DataSpace и Functions

4 Знакомство с Platform V DataSpace

Для работы с данными мы будем использовать платформенный сервис DataSpace.

Тут нужен промо-ролик про DataSpace!!!

1. Проектирование модели предметной области приложения в визуальном редакторе
 - a. Обучающее видео



Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

<https://www.youtube.com/watch?v=fBosd4qJxy8>

- b. Документация: <https://developers.sber.ru/docs/ru/platform-v/dataspace/overview>
 - c. Получившаяся модель в формате xml



PromoAction.xml

2. После выпуска сервиса DataSpace можно работать с данными через GraphQL API, в т.ч. через конструктор в визуальном редакторе
 - a. Обучающее видео:



Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

<https://www.youtube.com/watch?v=nDaJVmRJu10>

- b. Документация: <https://developers.sber.ru/docs/ru/platform-v/dataspace/graphql/overview>

5 Подписание запросов (авторизация)

Итак, ранее мы с вами уже создали модель предметной области и выпустили соответствующий сервис DataSpace, предоставляющий GraphQL API для работы с данными.

Но для вызова извне необходимо отправлять http-запросы на соответствующий сервис, предварительно подписывая их при помощи ключа.

Можно делать такие запросы, воспользовавшись соответствующим [JavaScript SDK](#)⁷.

Также SDK предоставляет возможность ручного формирования запросов через визуальную форму:

Apigateway Signature Test

Key: 6436a38d9217493f92af49a083432cc8

Secret: [REDACTED]

Method: POST

Url: https://smapi.pv-api.sbc.space/ds-6977680424981364738/graphql

Headers:

- content-type: application/json

Body:

```
{
  "operationName": "q",
  "variables": {},
  "query": "query q{searchGiftVendor(elems{id.name}})"
}
```

Buttons: Add, Debug, Send request

curl -X POST "https://smapi.pv-api.sbc.space/ds-6977680424981364738/graphql" -H "content-type: application/json" -H "X-Sdk-Date: 20211011T142850Z" -H "host: smapi.pv-api.sbc.space" -H "Authorization: SDK-HMAC-SHA256 Access=6436a38d9217493f92af49a083432cc8, SignedHeader"

Данные для подписи запроса доступны в настройках соответствующего проекта в SmartMarket:

Проекты

обновлен 25.06.2021

Промоакция

Platform V DataSpace

Настройка проекта

Удалить

Готов к использованию

⁷ <https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/ApiGateway-javascript-sdk.zip>

В настройках доступны ak/sk + адрес сервиса для вызова:

App_key и app_secret

Используйте для обращения к инструментам Platform V через бэкэнд

Ваш app_key

6436a38d9217493f92af49a083432cc8



Показать app_secret

Адрес проекта

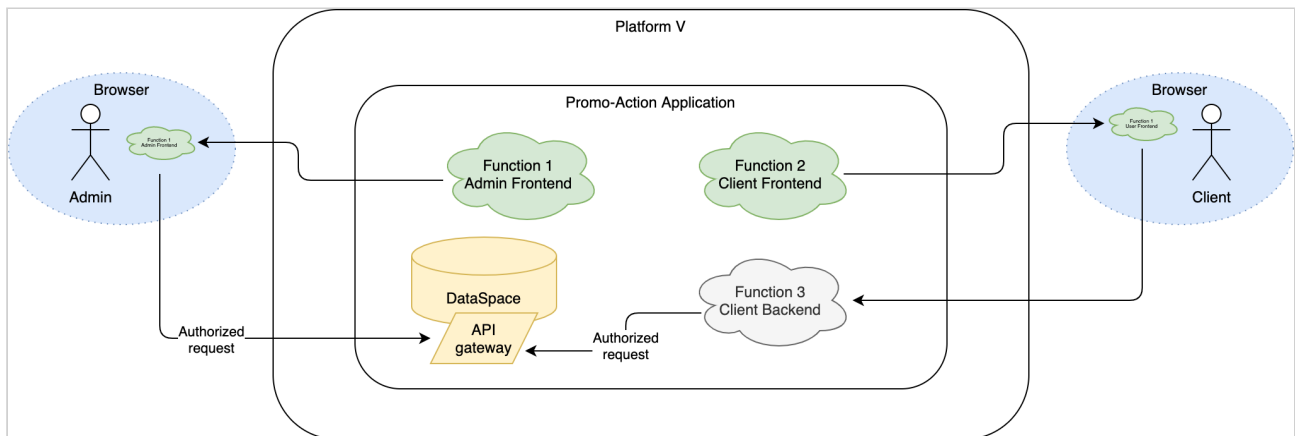
https://smapi.pv-api.sbc.space/ds-6977680...



6 Архитектура приложения "Промоакция"

У разных типов пользователей разные каналы для работы:

- "Администратор" (Admin) работает с DataSpace через авторизованные запросы на API gateway, зная адрес сервиса, appKey(логин) и appSecret(пароль), далее ak/sk.
 - Необходимая для работы статика приезжает посредством функции Function 1 Admin Frontend
- "Клиент" (Client) через общедоступный сервис вводит промокод и выбирает подарок. Воспользоваться промокодом можно только один раз.
 - Необходимая для работы статика приезжает посредством функции Function 2 Client Frontend
 - В функции Function 3 Client Backend реализуем серверную логику обработки запросов от "Клиентов"



7 Function 1 Admin Frontend

Давайте теперь перейдем к написанию frontend-приложения, когда в качестве backend'а используется DataSpace.

Вооружимся следующим технологическим стеком:

1. [TypeScript](#)⁸ - собственно, язык программирования
2. [GraphQL](#)⁹ - язык запросов к серверной части (DataSpace)
3. [GraphQL Code Generator \(TypeScript\)](#)¹⁰ - очень полезная утилита для преобразования GraphQL-запросов в конструкции на TypeScript
4. [ReactJS](#)¹¹ - ReactJS
5. [Apollo Client v3](#)¹² - JS-библиотека для работы через GraphQL: кэширование, react-хуки и другие "печеньки"
6. [Ant Design](#)¹³ - готовые экранные компоненты
7. Собирать будем при помощи [Webpack 5](#)¹⁴.

i Сразу оговорюсь, что не надо пугаться этого расширенного списка. Приложение будет достаточно простое и основная идея тут в том, чтобы структура и используемые компоненты были понятны любому человеку, имеющему базовые технические знания в области программирования. Важно отметить, что далеко не обязательно это должен быть матерый frontend-разработчик, коим также не является Ваш покорный слуга. Также я постарался по максимуму снабдить описание ссылками на конкретные документации. Уверен, это будет полезно тем, кто захочет более детально погрузиться в суть используемых технологий.

Исходники приложения доступны по ссылке на GitHub: <https://github.com/VictorBiryukov/promo-action.git>

7.1 Разработка

Начнем с режима разработки: [webpack.dev.config.js](#)¹⁵

Из интересного в конфигурации сборки только настройки прокси у сервера разработки:

⁸ <https://www.typescriptlang.org/>

⁹ <https://graphql.org/>

¹⁰ <https://www.graphql-code-generator.com/docs/plugins/typescript>

¹¹ <https://reactjs.org/>

¹² <https://www.apollographql.com/docs/react/>

¹³ <https://ant.design/>

¹⁴ <https://webpack.js.org/blog/2020-10-10-webpack-5-release/>

¹⁵ <https://github.com/VictorBiryukov/promo-action/blob/release/webpack.dev.config.js>

```

...
devServer: {
  hot: true,
  port: 3000,
  before: (app) => {
    app.use(createProxyMiddleware("/graphql",
      {
        target: process.env.DS_ENDPOINT,
        changeOrigin: true,
        secure: false,
        pathRewrite: { '/graphql': '' }
      }
    ));
  },
  watchOptions: {
    poll: true,
    ignored: "/node_modules/"
  }
},
...

```

Таким образом мы обходим проверку запрета на CORS¹⁶ со стороны сервиса при работе через браузер в режиме локальной разработки.

Для корректной работы прокси-сервера необходимо в файле .env указать адрес вашего сервиса DataSpace:

```
DS_ENDPOINT=[Enter your dataspace graphql endpoint here]
```

Перейдем непосредственно к написанию приложения.

В файле `src/index.tsx`¹⁷ ничего необычного: подключаем React, отрисовываем корневой компонент `App`¹⁸.

Прежде чем начать работать с DataSpace, надо научить наш провайдер GraphQL-запросов правильно их подписывать.

Для этого даем возможность пользователю ввести адрес DataSpace + ak/sk, сохраняем данные в `localStorage`:

¹⁶ https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

¹⁷ <https://github.com/VictorBiryukov/promo-action/blob/release/src/index.tsx>

¹⁸ <https://github.com/VictorBiryukov/promo-action/blob/release/src/components/App.tsx>

```

<Form>
  <Form.Item>
    <Input placeholder="Service address"
      value = {appAddress}
      onChange={e => setAppAddress(e.target.value)}
    />
  </Form.Item>
  <Form.Item>
    <Input placeholder="Service key"
      value = {appKey}
      onChange={e => setAppKey(e.target.value)}
    />
  </Form.Item>
  <Form.Item>
    <Input.Password placeholder="Service secret"
      value = {appSecret}
      onChange={e => setAppSecret(e.target.value)}
    />
  </Form.Item>
</Form>

```

Заполнив параметры, передаем их далее в [AppProvider](#)¹⁹, где уже при помощи ранее представленного [JavaScript SDK](#)²⁰ определяем правило подписания, после чего инициализируем [ApolloClient](#)²¹:

```

const authFetch = (uri: any, options: any) => {

  let sig = new signer.Signer()
  sig.Key = appKey
  sig.Secret = appSecret
  let request = new signer.HttpRequest(options.method, appAddress, options.headers, options.body)
  sig.Sign(request)

  return fetch(uri, options);
};

console.log(process.env.NODE_ENV);

return new ApolloClient({
  cache: cache,
  link: new HttpLink({
    uri: process.env.NODE_ENV === 'production' ? appAddress : 'graphql',
    fetch: authFetch,
  })
})

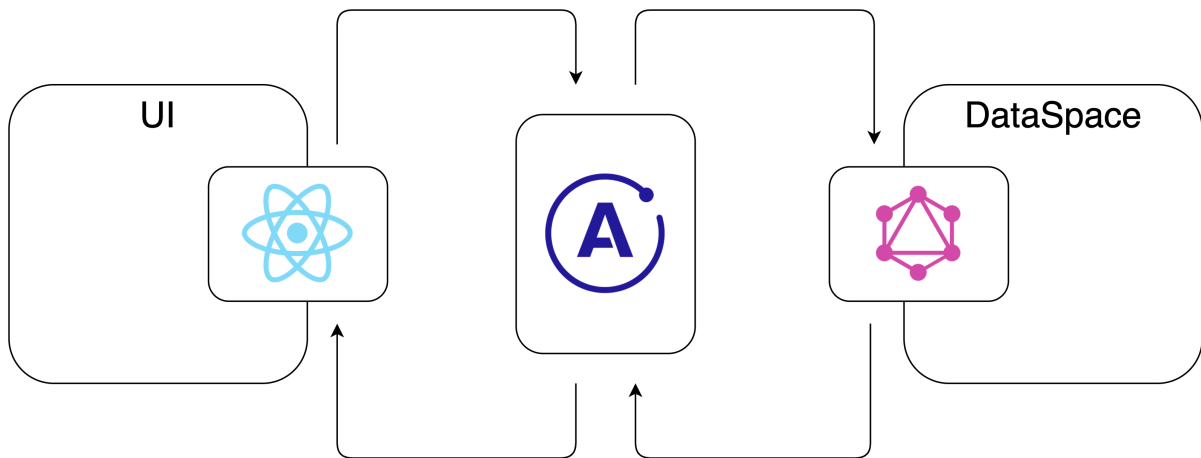
```

¹⁹ <https://github.com/VictorBiryukov/promo-action/blob/release/src/components/AppProvider.tsx>

²⁰ <https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/ApiGateway-javascript-sdk.zip>

²¹ <https://www.apollographql.com/docs/react/>

Apollo Client призван значительным образом упростить работу с сервисом DataSpace через GraphQL API:



С одной стороны он замечательным образом интегрируется с React через хуки ([hooks²²](https://reactjs.org/docs/hooks-intro.html)), с другой - обеспечивает бесшовную интеграцию с GraphQL-сервером DataSpace, элегантно решая вопросы кэширования и нормализации данных на клиенте.

Вернемся к нашему приложению "Промоакция".

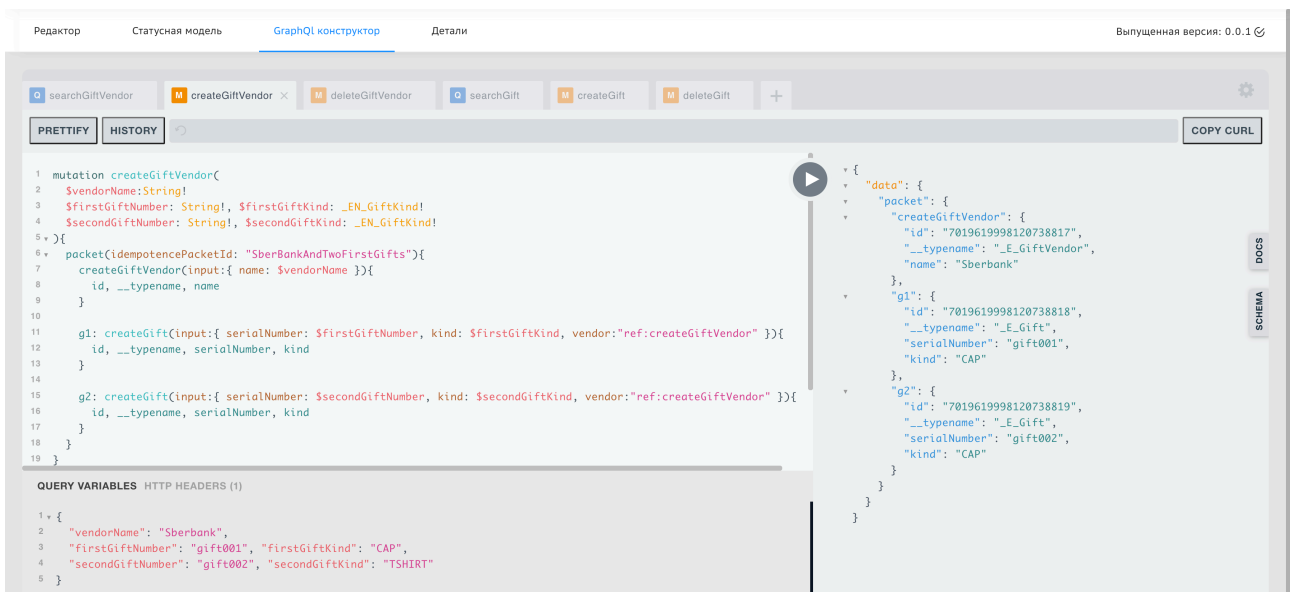
В UI-консоли администратора нам необходимо обеспечить следующие возможности:

- запрос списка компаний-спонсоров
- создание/удаление компании-спонсора
- запрос списка подарков компании-спонсора
- создание/удаление подарка

Для написания соответствующих запросов воспользуемся GraphQL-конструктором внутри визуального редактора.

В примере ниже мы одним запросом создаем компанию-спонсора(GiftVendor) и два ее первых подарка(Gift):

²² <https://reactjs.org/docs/hooks-intro.html>



Комментарий к рисунку: В верхней левой части представлен сам запрос, в левом нижнем углу - передаваемые в запрос параметры, справа - результат его выполнения

Давайте уделим внимание двум моментам:

- Указывая ключ **"SberBankAndTwoFirstGifts"** в параметре **idempotencePacketId** в мутации **packet**, мы делаем этот запрос **идемпотентным**²³. При повторном выполнении данного запроса (если первый был успешным) DataSpace вернет результат выполнения, но не будет повторять саму операцию создания (изменения состояния БД). Этот функционал DataSpace призван значительно упростить жизнь разработчику клиентской части, когда необходимо повысить надежность взаимодействия с сервисом: можно не беспокоиться о лишних данных, делая повтор команды в случае, к примеру, получения ошибки таймаута выполнения при первоначальном вызове.
- Также обратите внимание на лексему **"ref:createGiftVendor"**, передаваемую в поле **vendor** создаваемых подарков: таким образом обеспечена связь между подарками и компанией-спонсором, создаваемой на первом шаге выполнения пакета.

Детальное описание формата GraphQL-запросов DataSpace доступно в документации: [documentation/graphql²⁴.md²⁵](#)

Для нашего же приложения понадобится набор совсем простых GraphQL-запросов:

[searchGiftVendor](#) [createGiftVendor](#) [deleteGiftVendor](#) [searchGift](#) [createGift](#) [deleteGift](#)

²³<https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BC%D0%BF%D0%BE%D1%82%D0%B5%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C>

²⁴ <https://github.com/VictorBiryukov/promo-action/blob/release/documentation/graphql.md>

²⁵ <http://expression.md>

```
query searchGiftVendor{
  searchGiftVendor{
    elems{
      id
      __typename
      name
    }
  }
}
```

searchGiftVendor [createGiftVendor](#) deleteGiftVendor searchGift createGift deleteGift

```
mutation createGiftVendor($name:String!){
  packet{
    createGiftVendor(input:{
      name: $name
    }){
      id
      __typename
      name
    }
  }
}
```

searchGiftVendor createGiftVendor [deleteGiftVendor](#) searchGift createGift deleteGift

```
mutation deleteGiftVendor($id: ID!){
  packet{
    deleteGiftVendor(id: $id)
  }
}
```

searchGiftVendor createGiftVendor deleteGiftVendor [searchGift](#) createGift deleteGift

```

query searchGift($cond: String){
  searchGift(cond: $cond){
    elems{
      id
      __typename
      serialNumber
      kind
    }
  }
}

```

searchGiftVendor createGiftVendor deleteGiftVendor searchGift createGift deleteGift

```

mutation createGift($vendorId: ID!, $serialNumber:String!, $kind: _EN_GiftKind){
  packet{
    createGift(input:{
      vendor: $vendorId
      serialNumber: $serialNumber
      kind: $kind
    }){
      id
      __typename
      serialNumber
      kind
    }
  }
}

```

searchGiftVendor createGiftVendor deleteGiftVendor searchGift createGift deleteGift

```

mutation deleteGift($id: ID!){
  packet{
    deleteGift(id: $id)
  }
}

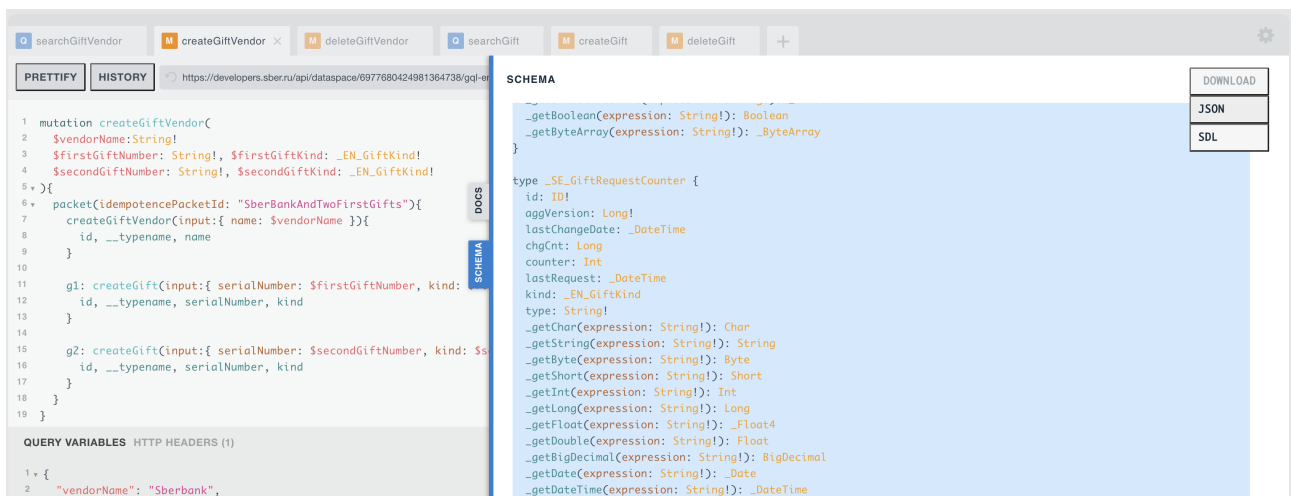
```

Зафиксируем данные запросы в файле [src/graphql/requests.graphql](#)²⁶.

Нам также понадобится GraphQL-схема API DataSpace, которая доступна в конструкторе визуального редактора.

Справа в закладке SCHEMA выбираем DOWNLOAD → SDL:

²⁶ <https://github.com/VictorBiryukov/promo-action/blob/release/src/graphql/requests.graphql>



Выгруженный файл поместим в корневую папку проекта: [schema.graphql](#)²⁷

Теперь давайте еще упростим жизнь разработчику приложения за счет генерации Typescript-конструкций на основе запросов и схемы, зафиксированных ранее.

В файле [package.json](#)²⁸ у нас подключены необходимые JS-библиотеки для этапа разработки и прописана команда генерации в разделе `scripts`:

```
...
"devDependencies": {
  ...
  "@graphql-codegen/cli": "^1.9.0",
  "@graphql-codegen/typescript": "1.22.3",
  "@graphql-codegen/typescript-operations": "1.18.2",
  "@graphql-codegen/typescript-react-apollo": "2.2.7",
  ...
},
...
"scripts": {
  ...
  "codegen": "graphql-codegen --config codegen.yml",
  ...
},
...
```

Конфигурируем правила генерации в [codegen.yml](#)²⁹:

²⁷ <https://github.com/VictorBiryukov/promo-action/blob/release/schema.graphql>

²⁸ <https://github.com/VictorBiryukov/promo-action/blob/release/package.json>

²⁹ <https://github.com/VictorBiryukov/promo-action/blob/release/codegen.yml>

```

overwrite: true # флаг перезаписи файла генерируемого кода
schema: 'schema.graphql' # файл graphql-схемы
documents: 'src/graphql/**/*.graphql' # маска файлов с graphql-запросами
generates:
  ./src/__generate/graphql-frontend.ts: # результирующий файл генерации
  plugins:
    - typescript # генерация типов
    - typescript-operations # генерация операций
    - typescript-react-apollo # генерация React Apollo компонентов

```

Все готово для генерации. Запускаем из консоли соответствующую команду **"npm run codegen"**:

```
[sbt-biryukov-vv@cab-wsm-0041856 promo-action % npm run codegen
```

```

> promo-action@0.1.0 codegen
> graphql-codegen --config codegen.yml

```

```

✓ Parse configuration
✓ Generate outputs

```

```
sbt-biryukov-vv@cab-wsm-0041856 promo-action % █
```

Генерация прошла успешно, без ошибок. Давайте взглянем на результаты: [src/__generate/graphql-frontend.ts](#)³⁰

Остановимся более детально на некоторых конструкциях в этом файле.

В первую очередь теперь у нас есть Typescript-типы, определяющие ранее заведенные в DataSpace сущности.

В т.ч. ряд служебных полей:

- `aggVersion`: версия агрегата, которую можно использовать для формирования транзакции между получением и сохранением данных в БД (оптимистическая блокировка).
- `lastChangeDate`: дата/время последнего изменения экземпляра сущности
- `type`: тип сущности (может быть полезен в случае использования наследования в модели данных)
- `aggregateRoot`: ID корня агрегата
- и др.

Например, тип для сущности Gift:

³⁰ https://github.com/VictorBiryukov/promo-action/blob/release/src/__generate/graphql-frontend.ts

```
export type Gift = {
  id: Scalars['ID'];
  aggVersion: Scalars['Long'];
  lastChangeDate?: Maybe<Scalars['_DateTime']>;
  chgCnt?: Maybe<Scalars['Long']>;
  kind?: Maybe<_En_GiftKind>;
  serialNumber: Scalars['String'];
  type: Scalars['String'];
  vendor: GiftVendor;
  aggregateRoot?: Maybe<GiftVendor>;
  ...
};
```

Также отражено определенное нами ранее перечисление GiftKind:

```
export enum _En_GiftKind {
  Cap = 'CAP',
  Tshirt = 'TSHIRT',
  Mug = 'MUG'
}
```

В дальнейшем мы воспользуемся данными перечислением при написании формы создания подарков.

В конце файла видим сгенерированный ряд хуков, соответствующих нашим graphql-запросам ([src/graphql/requests.graphql](https://github.com/VictorBiryukov/promo-action/blob/release/src/graphql/requests.graphql)³¹).

Например, запросы searchGift, createGift, deleteGift представляют следующие функции

```
...
export function useSearchGiftQuery(baseOptions?: Apollo.QueryHookOptions<SearchGiftQuery,
SearchGiftQueryVariables>) {
  const options = {...defaultOptions, ...baseOptions}
  return Apollo.useQuery<SearchGiftQuery, SearchGiftQueryVariables>(SearchGiftDocument, options);
}
...
export function useCreateGiftMutation(baseOptions?: Apollo.MutationHookOptions<CreateGiftMutation,
CreateGiftMutationVariables>) {
  const options = {...defaultOptions, ...baseOptions}
  return Apollo.useMutation<CreateGiftMutation, CreateGiftMutationVariables>(CreateGiftDocument,
options);
}
...
export function useDeleteGiftMutation(baseOptions?: Apollo.MutationHookOptions<DeleteGiftMutation,
DeleteGiftMutationVariables>) {
  const options = {...defaultOptions, ...baseOptions}
  return Apollo.useMutation<DeleteGiftMutation, DeleteGiftMutationVariables>(DeleteGiftDocument,
options);
}
...
```

³¹ <https://github.com/VictorBiryukov/promo-action/blob/release/src/graphql/requests.graphql>

Это функции-обертки над React-хуками [Apollo.useQuery](https://www.apollographql.com/docs/react/data/queries/)³² и [Apollo.useMutation](https://www.apollographql.com/docs/react/data/mutations/)³³. Данные конструкции призваны типизировать нашу бесшовную интеграцию между серверной и клиентской частью приложения.

Давайте более детально взглянем на useCreateGiftMutation:

- CreateGiftMutationVariables определяют сигнатуру входящих параметров

```
...
export type CreateGiftMutationVariables = Exact<{
  vendorId: Scalars['ID'];
  serialNumber: Scalars['String'];
  kind?: Maybe<_En_GiftKind>;
}>;
...
```

- CreateGiftMutation определяет сигнатуру возвращаемого результата

```
...
export type CreateGiftMutation = (
  { __typename?: '_Mutation' }
  & { packet?: Maybe(
    { __typename?: '_Packet' }
    & { createGift?: Maybe(
      { __typename: '_E_Gift' }
      & Pick<_E_Gift, 'id' | 'serialNumber' | 'kind'>
    )> }
  )> }
);
...
```

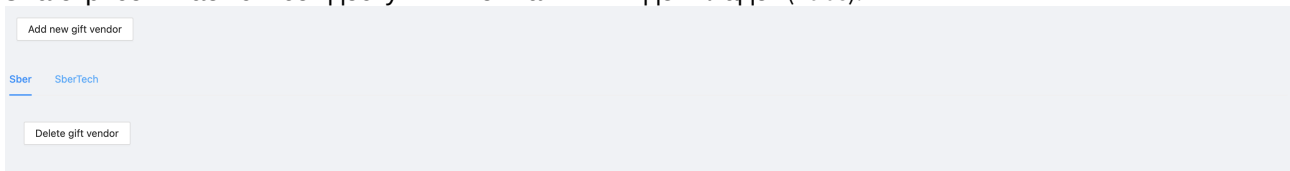
Итак, на данном этапе мы:

- научились подписывать наши http-запросы к серверной части
- определили набор GraphQL-запросов, которые нам понадобятся для работы
- осуществили генерацию необходимых Typescript-конструкций, призванных упростить жизнь разработчика при написании кода

Перейдем непосредственно к прикладным формам.

Форма отображения/добавления/удаления компаний-спонсоров реализована в соответствующем компоненте: <src/components/GiftVendorList.tsx>³⁴.

Она отрисовывает список доступных компаний в виде вкладок (Tabs):



32 <https://www.apollographql.com/docs/react/data/queries/>

33 <https://www.apollographql.com/docs/react/data/mutations/>

34 <https://github.com/VictorBiryukov/promo-action/blob/release/src/components/GiftVendorList.tsx>

Кнопка "Add new gift vendor" позволяет заводить новые компании-спонсоры:

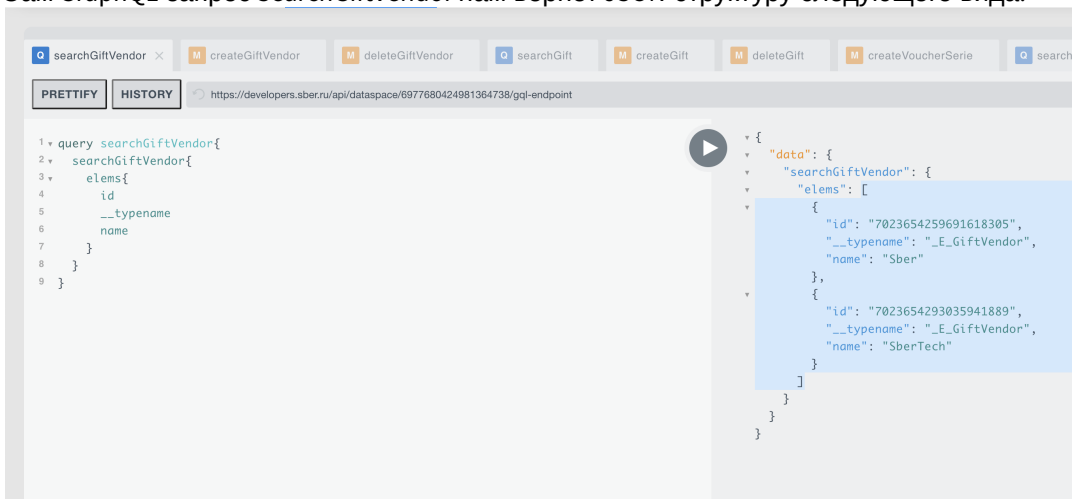
Кнопка "Delete gift vendor" удаляет компанию

На что хочется обратить внимание в коде:

- Для получения списка компаний-спонсоров используется хук `useSearchGiftVendorQuery`, который был сгенерирован на основе запроса `SearchGiftVendor`, зафиксированного в файле [src/graphql/requests.graphql](#)³⁵. Из результата выполнения хука деструктурируем параметры `data`, `loading`, `error`. Чуть ниже по коду обрабатываем соответствующим образом значения `loading`, `error`

```
...
const { data, loading, error } = useSearchGiftVendorQuery()
const giftVendorList = data?.searchGiftVendor?.elems
...
if (loading) return (<Spin tip="Loading..." />);
if (error) return <p>`Error! ${error.message}`</p>;
```

Сам GraphQL-запрос `SearchGiftVendor` нам вернет JSON-структуру следующего вида:



Т.к. в дальнейшем для отрисовки вкладок с компаниями нам нужен только сам массив компаний: определим для этого отдельную константу `giftVendorList`, ссылающуюся на массив `elems` возвращаемой

³⁵ <https://github.com/VictorBiryukov/promo-action/blob/release/src/graphql/requests.graphql>

запросом конструкции.

Вся прелесть такого подхода заключается в том, что мы работаем с данными через типизированную структуру, которая основывается на модели предметной области приложения, описанной ранее в DataSpace.

Более подробно про работу с мутациями в Apollo можно почитать здесь: <https://www.apollographql.com/docs/react/data/queries/>

Далее при помощи функции `getTabs`, принимающей на вход параметром список полученных компаний, отрисовываем вкладки.

```
...
    <Form style={{ margin: "10px" }}>
      <Form.Item>
        <Tabs>
          {getTabs(giftVendorList)}
        </Tabs>
      </Form.Item>
    </Form>
...
```

Давайте теперь разберемся с добавлением/удалением компаний:

```
...
const [createGiftVendorMutation] = useCreateGiftVendorMutation()
const [deleteGiftVendorMutation] = useDeleteGiftVendorMutation()
...
```

Вытаскиваем из соответствующих хуков функции - мутации добавления/удаления.

Более подробно про работу с мутациями в Apollo можно почитать здесь: <https://www.apollographql.com/docs/react/data/mutations/>

В модальной форме на кнопку "Ок" вешаем соответствующий обработчик, где делаем вызов мутации, передавая два параметра:

```

...
<Modal visible={showcreateForm}
  onCancel={() => setShowcreateForm(false)}
  onOk={() => {
    setShowcreateForm(false)
    createGiftVendorMutation({
      variables: {
        name: vendorName!
      },
      update: (store, result) => {
        store.writeQuery({
          query: SearchGiftVendorDocument,
          data: {
            searchGiftVendor: {
              elems: [...giftVendorList!,
result.data?.packet?.createGiftVendor]
            }
          }
        })
      })
    })
  }}
>
...

```

Первый - variables - это набор параметров, заполняемых пользователем на форме. В нашем случае имя компании-спонсора.

Второй - update - параметр update позволяет передать функцию, которая в нашем случае обновит кэш Apollo-клиента(store) для запроса SearchGiftVendor, добавив туда результат (result) GraphQL-запроса createGiftVendor.

Такой подход позволяет обновлять списочные формы (в нашем случае список вкладок) не делая дополнительного запроса на сервер.

Более детально про этот механизм можно почитать тут: <https://www.apollographql.com/docs/react/caching/cache-interaction/>

С удалением ситуация аналогичная: только здесь обработчик не добавляет элемент, а фильтрует массив, исключая ранее удаленный элемент


```

...
    <Button style={{ margin: "20px" }}
      key={elem.id ?? ""}
      onClick={e => {
        deleteGiftVendorMutation({
          variables: {
            id: elem.id
          },
          update: (store) => {
            store.writeQuery({
              query: SearchGiftVendorDocument,
              data: {
                searchGiftVendor: {
                  elems: giftVendorList!.filter(x => x.id !== elem.id)
                }
              }
            })
          }
        })
      }}
    >Delete gift vendor</Button>cond: "it.vendor.$id == '" + vendorId + "'"
...

```

Функционал для заведения подарков в рамках конкретной компании-спонсора аналогичен заведению самих компаний: [src/components/GiftList.tsx](https://github.com/VictorBiryukov/promo-action/blob/release/src/components/GiftList.tsx)³⁶.

Вместо компонента `Tabs`³⁷ используется компонент `Table`³⁸ в самом простом его варианте. Остановим наше внимание на двух моментах:

- в хук `useSearchGiftQuery` передается через переменную `cond` соответствующего GraphQL-запроса условие фильтрации: `"it.vendor.$id == '" + vendorId + "'"`. Т.е. запрашиваются подарки только конкретной компании-спонсора, на вкладке которой мы сейчас находимся. При формировании условий используется нативно понятный язык регулярных выражений, оперирующий структурой модели. В то же время доступен гибкий и мощный инструментальный составления разного рода условий выборки данных. Например:

³⁶ <https://github.com/VictorBiryukov/promo-action/blob/release/src/components/GiftList.tsx>

³⁷ <https://ant.design/components/tabs/>

³⁸ <http://ant.design/table>

```

query searchGiftVendor{
  searchGiftVendor(cond: "it.name $like 'Sber%' && it.gifts.$exists"){
    elems{
      name
      lastChangeDate
      gifts(cond:"it.lastChangeDate < root.lastChangeDate.$addDays(1) || it.serialNumber.
$substr(1,1) == '1'"){
        elems{
          serialNumber
        }
      }
    }
  }
}

```

Запрос всех спонсоров, начинающихся с лексемы 'Sber' и имеющих хотя бы один подарок. У таких компаний нам будут интересны только подарки, которые создавались/менялись в течение суток после создания компании-родителя

Детальное описание всех возможностей данного синтаксиса фильтрации: [documentation/expressions.md](#)³⁹.

- отдельно обратите внимание, что есть возможности сортировки и постраничной вычитки запросов. Детали в документации: [documentation/graphql.md](#)⁴¹.

Итак, мы написали приложение для фиксации компаний-спонсоров и выпускаемых ими подарков. Если попробуете аналогичным образом добавить формы для фиксации серий (промоакций) и выдаваемых в рамках акций ваучеров (промокодов), то увидите, что это не займет много времени. Необходимые сущности в модели и сервис для работы с ними у вас уже есть.

Нужно лишь зафиксировать новые запросы по аналогии с запросами к спонсорам и их подаркам: [src/graphql/requests.graphql](#)⁴³.

А также отразить новые формы по аналогии с ранее рассмотренными компонентами: [src/components/GiftVendorList.tsx](#)⁴⁴ и [src/components/GiftList.tsx](#)⁴⁵.

Вторая часть статьи:

- знакомство с Functions
- использование сервисов Platform V при построении логики голосовых помощников

В первой части мы с Вами познакомились с возможностями сервиса Platform V DataSpace: создали хранилище данных и сервис работы с этими данными через GraphQL-запросы, реализовали frontend-приложение для работы в роли администратора системы. Теперь нам необходимо опубликовать данное frontend-приложение, а также разработать и опубликовать ту часть сервиса (backend-приложение), которая будет предоставлять возможность резервировать подарки в роли "Клиента". Для решения этих задач нам понадобится другой сервис - Platform V Functions.

³⁹ <https://github.com/VictorBiryukov/promo-action/blob/release/documentation/expressions.md>

⁴⁰ <http://expression.md>

⁴¹ <https://github.com/VictorBiryukov/promo-action/blob/release/documentation/graphql.md>

⁴² <http://expression.md>

⁴³ <https://github.com/VictorBiryukov/promo-action/blob/release/src/graphql/requests.graphql>


⁴⁴ <https://github.com/VictorBiryukov/promo-action/blob/release/src/components/GiftVendorList.tsx>

⁴⁵ <https://github.com/VictorBiryukov/promo-action/blob/release/src/components/GiftList.tsx>

8 Знакомство с Platform V Functions

Платформенный сервис Functions позволяет реализовывать приложения в парадигме [serverless](#)⁴⁶, реализовав архитектурный шаблон [Function-as-a-Services](#)⁴⁷.

1. Обучающее видео:

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:
<https://www.youtube.com/watch?v=nqIpMpA7BkI>

2. Документация: <https://developers.sber.ru/docs/ru/platform-v/functions/overview>
3. Рассмотренная в примере функция доступна здесь: promoaction-backend (на архитектурной диаграмме этот компонент отражен как Function 3 Client Backend)

⁴⁶ https://en.wikipedia.org/wiki/Serverless_computing

⁴⁷ https://en.wikipedia.org/wiki/Function_as_a_service

9 Публикация

Давайте вернемся к нашему ранее разработанному примеру Function 1 Frontend. Воспользуемся Platform V Functions для публикации разработанного frontend-приложения.

Здесь нам поможет файл конфигурации для промышленной сборки: [webpack.config.js](#)⁴⁸
Запустим сборку соответствующей командой:

```
npm run bundle
```

Результат сборки - директория static в папке [fn-admin-frontend](#)⁴⁹. Теперь нам достаточно только запаковать содержимое папки [fn-admin-frontend](#)⁵⁰ в zip-архив и загрузить в заранее созданную функцию в SmartMarket.

После создания функции нажимаем "Импортировать zip", выбираем подготовленный архив, нажимаем "Загрузить".

После того, как функция загружена, необходимо ее опубликовать нажатием соответствующей кнопки в правом нижнем углу экрана.

После публикации на вкладке "Детали" Вам доступен адрес этой функции в поле "Синхронный endpoint":

Редактор	Тестирование	Логи функции	Логи сборки	Детали	Мониторинг	DataSpace	Состояние функции: READY
Название	Function_9db03b47_1262_49e8_8006_e544a5b493bc						
Состояние	READY						
Дата создания	15.10.2021, 13:57:15 GMT+3						
Дата обновления	09.11.2021, 10:04:23 GMT+3						
Триггер	HTTP						
Реплики	1						
Доступные реплики	1						
Переменные среды	DATASPACE_URL: https://smapi.pv-api.sbc.space/ds-6977680424981364738/graphql						
Синхронный endpoint	https://smapi.pv-api.sbc.space/fn_d2527eab_2999_4a9a_99b1_4f90bf815b54						
Асинхронный endpoint							

По-умолчанию данный адрес также как и у DataSpace защищен парой ak/sk.

Но согласно архитектурной схеме нам нет необходимости такой проверки для этого компонента, т.к. это статика, которая должна быть доступна через браузер любому потребителю.

Для отключения данной проверки на функции необходимо составить запрос на снятие проверки ak/sk тут же в чате SmartMarket Studio, указав адрес вашей функции.

Пример такого запроса:

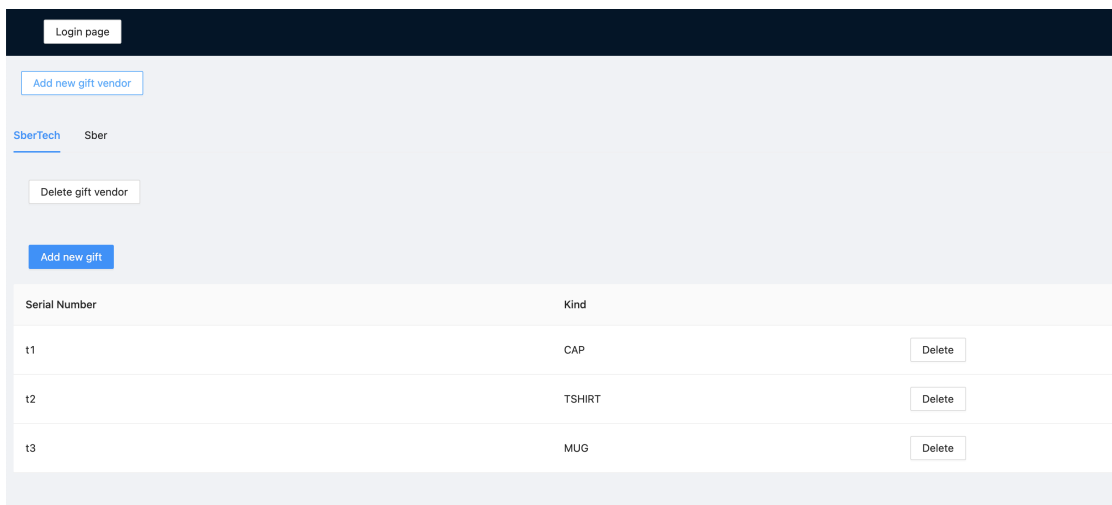
"Добрый день! Просьба отключить проверки ak/sk для функции: "

После снятия ограничения по соответствующему адресу теперь вам доступно приложение для ведения компаний-спонсоров(GiftVendor) и их подарков(Gift):

⁴⁸ <https://github.com/VictorBiryukov/promo-action/blob/release/webpack.config.js>

⁴⁹ <https://github.com/VictorBiryukov/promo-action/tree/release/fn-admin-frontend>

⁵⁰ <https://github.com/VictorBiryukov/promo-action/tree/release/fn-admin-frontend>



Аналогичным образом можно доработать текущий сервис или создать новый для заведения серий ваучеров (VoucherSerie) и самих ваучеров с промокодами (Voucher). Попробуйте сделать это сами. Пока же можно просто добавить несколько серий и ваучеров непосредственно из "GraphQL конструктора" DataSpace.

Ниже пример создания двух серий и ваучеров в рамках них следующим GraphQL-запросом:

```

mutation createVoucherSeries{
  p1: packet(idempotencePacketId:"SerieOne"){
    createVoucherSerie(input:{
      code:"SerieOne"
    }){
      id
    }

    c001: createVoucher(input:{
      code:"PromoS1G1"
      serie:"ref:createVoucherSerie"
    }){
      id
    }

    c002: createVoucher(input:{
      code:"PromoS1G2"
      serie:"ref:createVoucherSerie"
    }){
      id
    }

    c003: createVoucher(input:{
      code:"PromoS1G3"
      serie:"ref:createVoucherSerie"
    }){
      id
    }

    c004: createVoucher(input:{
      code:"PromoS1G4"
      serie:"ref:createVoucherSerie"
    }){
      id
    }

    c005: createVoucher(input:{
      code:"PromoS1G5"
      serie:"ref:createVoucherSerie"
    }){
      id
    }

    c006: createVoucher(input:{
      code:"PromoS1G6"
      serie:"ref:createVoucherSerie"
    }){
      id
    }

    c007: createVoucher(input:{
      code:"PromoS1G7"
      serie:"ref:createVoucherSerie"
    }){

```

```

    id
  }

  c008: createVoucher(input:{
    code:"PromoS1G8"
    serie:"ref:createVoucherSerie"
  }){
    id
  }
}

p2: packet(idempotencePacketId:"SerieTwo"){
  createVoucherSerie(input:{
    code:"SerieTwo"
  }){
    id
  }

  c001: createVoucher(input:{
    code:"PromoS2G1"
    serie:"ref:createVoucherSerie"
  }){
    id
  }

  c002: createVoucher(input:{
    code:"PromoS2G2"
    serie:"ref:createVoucherSerie"
  }){
    id
  }

  c003: createVoucher(input:{
    code:"PromoS2G3"
    serie:"ref:createVoucherSerie"
  }){
    id
  }

  c004: createVoucher(input:{
    code:"PromoS2G4"
    serie:"ref:createVoucherSerie"
  }){
    id
  }

  c005: createVoucher(input:{
    code:"PromoS2G5"
    serie:"ref:createVoucherSerie"
  }){
    id
  }

  c006: createVoucher(input:{
    code:"PromoS2G6"

```

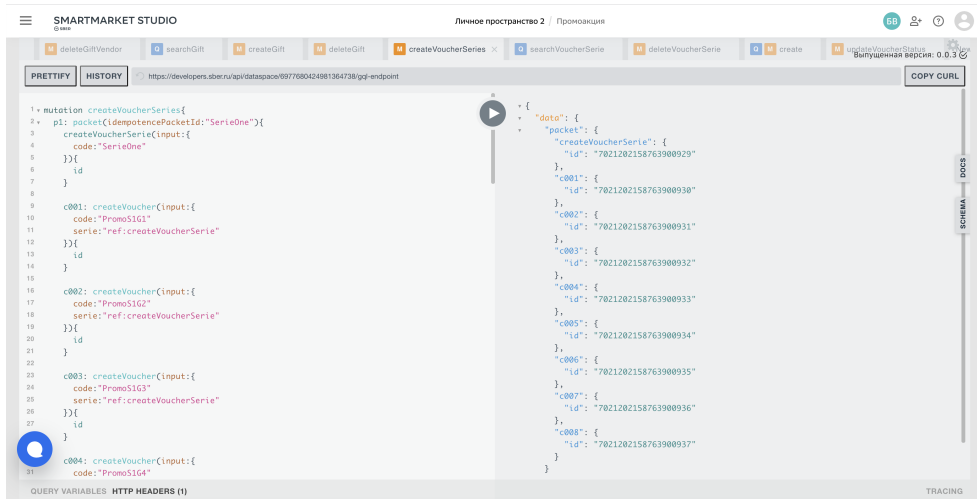
```

    serie:"ref:createVoucherSerie"
  }}{
    id
  }

c007: createVoucher(input:{
  code:"PromoS2G7"
  serie:"ref:createVoucherSerie"
  }}{
    id
  }

c008: createVoucher(input:{
  code:"PromoS2G8"
  serie:"ref:createVoucherSerie"
  }}{
    id
  }
}
}

```



10 Function 2 Client Frontend/ Function 3 Client Backend

Итак, у нас теперь уже есть подарки/ваучеры.

Осталась только та часть приложения, которая будет обслуживать "Клиентов".

Реализуем данный функционал в более традиционной варианте:

- функция Function 2 Client Frontend будет реализовывать frontend-часть
- функция Function 3 Client Backend - backend-часть, работая напрямую с DataSpace как с хранилищем данных

Начнем с серверной части - функции Function 3 Client Backend.

При создании функции в SmartMarket укажем название, выберем язык "JavaScript", и обязательно выберем наш DataSpace, ранее созданный в этом пространстве.

The screenshot shows the 'SMARTMARKET STUDIO' interface. At the top, it says 'Проект Platform V Functions'. Below this, there are three steps to create a project:

- 1 Название проекта**: A text input field containing 'promo-action-client-back'.
- 2 Язык программирования**: Three buttons labeled 'JavaScript', 'Python', and 'Java'. 'JavaScript' is selected.
- 3 Platform V DataSpace**: A dropdown menu with the option 'Промоакция' selected. Below it, a smaller dropdown shows 'Платформа V DataSpace'.

At the bottom left, there is a blue circular button with a white plus sign. At the bottom center, there is a blue button labeled 'Создать проект'.

Здесь представлен сам код функции: [promo-action-client-back](#)⁵¹

В отличие от первого примера Function 1 Admin Frontend, данная функция написана на чистом JavaScript. Код очень простой.

В файле [promo-action-client-back/main/apolloClient.js](#)⁵² также используется [Apollo Client](#)⁵³: формируются необходимые GraphQL-запросы, подписываемые при помощи ak/sk.

Обратите внимание, что адрес DataSpace + ak/sk мы получаем из соответствующих инфраструктурных переменных: DATASPACE_URL, APP_KEY, APP_SECRET. Система знает эти данные за счет того, что ранее при создании функций мы выбрали необходимый нам DataSpace. Данная функция обрабатывает http-запрос с параметрами code (код ваучера, введенный "Клиентом") и kind (тип подарка, выбранный "Клиентом"). См. тут: [promo-action-client-back/main/handler.js](#)⁵⁴.

По аналогии с первой функцией запаковываем содержимое папки [promo-action-client-back](#)⁵⁵ в архив, импортируем код в функцию и публикуем. После публикации нам доступен адрес функции в поле "синхронный endpoint" на вкладке Детали.

Т.к. данная функциональность будет вызываться из frontend-приложения нашими "Клиентами", необходимо снять проверку подписи через ak/sk. Для этого также через чат SmartMarket формируем

⁵¹ <https://github.com/VictorBiryukov/promo-action/tree/release/promo-action-client-back>

⁵² <https://github.com/VictorBiryukov/promo-action/blob/release/promo-action-client-back/main/apolloClient.js>

⁵³ <https://www.apollographql.com/docs/react/>

⁵⁴ <https://github.com/VictorBiryukov/promo-action/blob/release/promo-action-client-back/main/handler.js>

⁵⁵ <https://github.com/VictorBiryukov/promo-action/tree/release/promo-action-client-back>

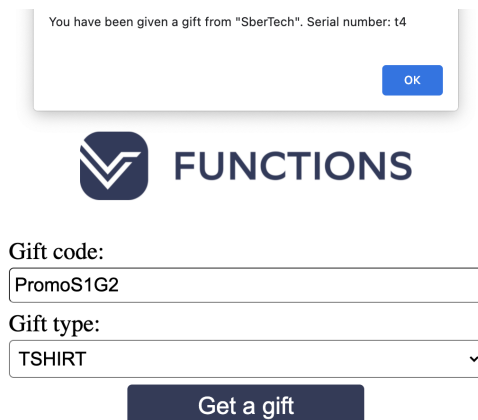
запрос на снятие проверки.

Серверная часть (backend) "Клиентского" приложения у нас готова. Перейдем к frontend'у: функция Function 2 Client Frontend.

Код функции доступен тут: [promo-action-client-front](https://github.com/VictorBiryukov/promo-action/tree/release/promo-action-client-front)⁵⁶

Нам необходимо внести незначительные изменения: в строке шесть файла [promo-action-client-front/static/js/main.js](https://github.com/VictorBiryukov/promo-action/blob/release/promo-action-client-front/static/js/main.js)⁵⁷ указать адрес (синхронный endpoint) backend-функции, созданной нами шагом ранее. Далее также сохраняем содержимое папки в архив, импортируем, публикуем, снимаем проверку ak/sk для соответствующего endpoint'a.

Поздравляю! Ваш микросервис "Промоакция" готов! Теперь вы можете поделиться адресом функции Function 2 Client Frontend со своими "Клиентами". В браузере им будет доступна форма получения подарка:



"Клиент" должен ввести свой промокод, выбрать тип подарка. Нажать кнопку "Get a gift". При наличии подарка выбранного типа будет получено информационное сообщение с указанием компании-спонсора и серийным номер подарка.

На этом все! Надеюсь, Вы все-таки добрались с нами до конца статьи. И еще больше надеюсь, что вам было интересно.

В следующих статьях мы планируем поговорить с Вами о более сложных концепциях промышленных приложений: разделения на микросервисы, авторизации доступа к данным DataSpace, возможностях использования функционала Platform V для написания голосовых помощников в SmartMarket Studio и т.п.

Спасибо за внимание!

⁵⁶ <https://github.com/VictorBiryukov/promo-action/tree/release/promo-action-client-front>

⁵⁷ <https://github.com/VictorBiryukov/promo-action/blob/release/promo-action-client-front/static/js/main.js>