ООО «КВАНТУМ АРТ»

115184, Москва, Озерковский переулок, д. 12

тел. (495) 783-65-74

Модуль **GraphQL** для продукта QP8.CMS с поддержкой PostgreSQL

Руководство пользователя

Москва
2023

История изменений

Версия	Дата	Автор	Описание
1.1	30.08.2023	Молькова М. Е.	Первое техническое описание (доработки)
1.0	05.07.2023	Григорьева М. А.	Первичное техническое описание

Оглавление

1. Обозначения

Обозначение	Описание	Пример использования
Технические данные	Используется для выделения различных технических данных в тексте: URL, названия свойств и методов, имена файлов и т. п.	ГПИ Системы доступен по URL https://www.domainname.zone/
Код	Пример кода.	`public DataTable Data { get; set; }`
Переменная	Используется для указания переменного значения.	Формат URL: Базовый URI/Псевдоним объекта
Требует дополнения	TBD (to be determined). Указывает, что необходима доработка текста – проверка корректности утверждения, детализация, правка после внесения изменений в документ и т. п.	Система работает с одной БД.
Примечание:	Дополнительные данные справочного характера.	Используется при генерации классов LINQ to SQL.
Внимание:	Важные данные, которые требуется обязательно учитывать.	Опция поддерживается только ASP-сборкой в целях совместимости.

2. Термины и определения

2.1. Общие термины и определения

В таблице ниже приведено описание используемых терминов и определений.

Термин или определение	Описание
API	«Application Programming Interface» (интерфейс программирования приложений) – набор правил по использованию функциональных возможностей Системы, предоставляемый разработчикам для организации взаимодействия сторонних программных продуктов с Системой
QP8.CMS или **QP8.CMS с поддержкой PostgreSQL** (далее **«QP»**)	Программный продукт, обладающий широким спектром возможностей для разработки программной части Системы различной сложности
QP8.WidgetPlatform (также **«Виджетная платформа»**)	Расширяет возможности QP. Позволяет через бэкенд наполнять веб-страницы Системы самостоятельно разработанными модульными приложениями. Виджетная платформа и виджеты основаны на шаблоне архитектуры MVC (от англ. «Model-View-Controller», «Модель-Представление-Контроллер»)
URL	«Uniform Resource Locator» – система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса
БД	База данных
ГПИ	Графический пользовательский интерфейс
Роутинг	Маршрутизация, переходы по ссылкам
Alias	Псевдоним; имя, назначенное источнику данных в запросе
REST	«Representational State Transfer» – архитектурный стиль, определяющий правила взаимодействия между клиентом и сервером
HTTP	«HyperText Transfer Protocol» – протокол передачи гипертекста
Плагин	Независимо компилируемый программный модуль, динамически подключаемый к основной программе
Docker	Программная платформа для быстрой сборки, отладки и развертывания приложений с помощью контейнеров
Kubernetes	Инструмент оркестрации контейнеров с открытым исходным кодом, который позволяет беспрепятственно развертывать и масштабировать контейнерные приложения, управлять ими в различных производственных средах
nginx	Программное обеспечение с открытым исходным кодом, используемое в качестве почтового или обратного прокси-сервера
UI	«User Interface» – пользовательский интерфейс
Мультитенантность	Архитектура программного обеспечения, в которой один экземпляр приложения обслуживает много клиентов (тенантов)

2.2. Термины QP

В таблице ниже приведены термины QP.

Термин или определение	Описание
GraphQL	Отдельный сервис для доступа к данным QP при разработке новых сайтов/продуктов, подключаемый к QP как плагин
Бэкенд	Копия QP; обладает ГПИ для работы с содержимым БД Системы
Контент	Раздел сайта, отвечающий за содержание пользовательской таблицы БД Системы и ее настройки
Поле	Атрибут контента. С использованием полей формируется структура данных для контента.
Сайт	Набор данных в бэкенде. Допускается создание нескольких сайтов. Содержимое каждого сайта определяется созданным в нём контентом.
Код клиента (англ. «Customer code»)	Уникальный параметр (тенант), определяющий БД, с которой взаимодействует бэкенд QP. Выбирается пользователем при входе в Систему.
Плагин QP	Набор дополнительных настроек, расширяющий стандартные сущности QP, такие как поле, контент, сайт без изменения приложения

3. Общие сведения

3.1. Описание модуля

GraphQL — это язык запросов для API-интерфейсов, а также среда, в которой они выполняются. С помощью GraphQL можно получить данные из API и передать их в приложение (от сервера к клиенту). Обычно в проектах GraphQL API используется в качестве бэкенд-части.

В рамках QP GraphQL используется в качестве модуля, обеспечивающего доступ к данным QP при разработке новых сайтов/продуктов с помощью языка запросов GraphQL.

GraphQL разработан в 2012 году как альтернатива REST. GraphQL и REST – два разных подхода к разработке API для обмена данными через Интернет. REST позволяет клиентским приложениям обмениваться данными с сервером через команды HTTP. При этом под каждую сущность необходим отдельный запрос. При добавлении/исчезновении сущности требуется доработка API.

В отличии от REST, GraphQL позволяет **за один запрос** передать нужные данные сразу в приложение, даже если они находятся в нескольких источниках. Иными словами, в GraphQL всего одна точка взаимодействия (endpoint) с бэкендом. Неважно, что вы хотите запросить: клиент будет взаимодействовать с одной конечной точкой, одним URL. Благодаря этому технология извлечения данных GraphQL удобнее и практичнее, чем REST.

Таким образом, GraphQL позволяет динамически подстраиваться под изменения структуры данных без дополнительной разработки.

В состав модуля входит GraphQL API, который позволяет искать, фильтровать, сортировать, избирательно выводить данные. GraphQL API построен на двух основных блоках: **запросах** (queries) и **схеме** (schema).

Примечание: дополнительные ссылки на документацию:
- исходная документация по GraphQL;
- спецификация GraphQL;
- подробнее.

3.2. Состав модуля

Модуль GraphQL для QP8.CMS с поддержкой PostgreSQL состоит из:

- сервиса GraphQL API;
- настраиваемого плагина QP.

Внимание: Модуль GraphQL не является мультитенантным и устанавливается для конкретного *customer code*.

4. Установка

Ниже представлено пошаговое руководство по установке модуля **GraphQL** в системе Linux.

4.1. Системные требования

Для установки и использования модуля **GraphQL** требуется установка продукта **QP8.CMS с поддержкой PostgreSQL**.

4.2. Установка QP.GraphQL (с использованием Docker)

1. Скачать архив graphql-config.tar и распаковать его содержимое в `/etc/qp-graphql-config`.
2. Перейти в папку `/etc/qp-graphql/compose`.
3. В файле `.env` задать строку подключения к ранее развёрнутой базе демо-сайта в параметре `CONNECTION_STRING` (можно скопировать из конфигурационного файла QP).
4. В файле `docker-compose.yml` можно настроить ключ инстанса GraphQL в параметре `QpPluginSettings__InstanceKey` (по умолчанию он соответствует настройкам в БД демосайта)
5. Выполнить команду:

```
sudo docker-compose up –d
```

4.3. Установка QP.GraphQL (с использованием Kubernetes)

1. Скачать архив graphql-config.tar и распаковать его содержимое в `/etc/qp-graphql-config`.
2. Перейти в папку `/etc/qp-graphql-config/k8s`.
3. В файле `graphql.yml` можно настроить:
   - строку подключения к ранее развёрнутой базе демо-сайта в параметре `ConnectionStrings__QPConnectionString` (можно скопировать из конфигурационного файла QP);
   - ключ инстанса GraphQL в параметре `QpPluginSettings__InstanceKey` (по умолчанию он соответствует настройкам в БД демосайта).
4. Развернуть сервис командой:

```
kubectl apply –f graphql.yml
```

5. При необходимости настроить ингресс к `graphql-api` в файле `ing.yaml`, задав свой DNS вместо `graphql-demosite-rus-api.test` и выполнив команду:

```
kubectl apply -f ing.yaml
```

4.4. Установка QP.GraphQL (без использования Docker)

4.4.1. При сборке из исходников

1. Извлечь исходные файлы из репозитория на GitHub.
2. В папке проекта `QP.GraphQL.App` (где находится файл `QP.GraphQL.App.csproj`) выполнить команду на публикацию:

```
dotnet publish "QP.GraphQL.App.csproj" -c Release -o bin/release/publish/ -r linux-x64 --self-contained=false
```

3. В папке `/home/qp` создать подпапку `QP.GraphQL` и скопировать туда всё содержимое каталога `bin/release/publish/`, в который осуществлялась публикация. Следует убедиться, что у пользователя `QP` есть права на чтение и исполнение содержимого папки `QP.GraphQL`.

4.4.2. При использовании готовых бинарных файлов

1. Скачать архив graphql-config.tar и распаковать его содержимое в `/etc/qp-graphql-config.`

2. Скачать архив, содержащий бинарные файлы GraphQL API.

3. Распаковать полученный архив в домашний каталог пользователя, созданного в рамках установки QP8.CMS (по умолчанию - `/home/qp`). Должна появиться папка `QP.GraphQL`.

4.4.3. Общая часть

1. Перейти в папку `QP.GraphQL`.

2. В файле `appsettings.json` в секции `ConnectionStrings` изменить значение параметра `QpConnectionString` на строку подключения к ранее развёрнутой базе демо-сайта в параметре (можно скопировать из конфигурационного файла QP).

3. В файле `appsettings.json` в секции `QpPluginSettings` можно изменить ключ инстанса GraphQL (по умолчанию он соответствует настройкам в БД демосайта).

4. В файле `NLog.config` найти `internalLogFile` и `<variable name="logDirectory" value=` и прописать там путь `/var/log/qp-graphql/.`

Внимание: для `internalLogFile` должен сохранится конечный файл, т. е. правильно строка будет выглядеть так: `/var/log/qp-graphql/internal-nlog.txt.`

5. Создать на сервере директорию `/var/log/qp-graphql/` и выдать QP-пользователю права на владение директорией.

6. Создать файл `qp-graphql.service` в папке `/usr/lib/systemd/system/` и заполнить его следующим содержимым:

```
[Unit]
Description=QP GraphQL API
After=qp.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/QP.GraphQL/
ExecStart=/bin/dotnet QP.GraphQL.App.dll --urls http://*:6300
```

```
[Install]
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка `/usr/lib/systemd/system/`, то все изменения следует вносить в папку `/lib/systemd/system/`.

7. Зарегистрировать сервис в автозапуск, выполнив команду:

```
systemctl enable qp-graphql.service
```

8. Запустить сервис, выполнив команду:

```
systemctl start qp-graphql.service
```

Примечание: в дальнейшем URL, под которым запущен сервис (http://localhost:6300 / http://<имя машины>:6300 / http://graphql-demosite-rus-api.test или свой DNS), следует использовать как **<URL сервиса>** при формировании относительных ссылок.

4.5. Настройка nginx

4.5.1. Nginx с использованием docker

Если nginx при установке QP8.CMS был запущен в docker, то следует:

1) выполнить команду:

```
docker-compose down
```

2) открыть на редактирование файл nginx.conf, предположительно расположенный по пути `/etc/qpconfig/nginx/nginx.conf`;
3) в конфигурационный файл добавить секции из файла `/etc/qp-graphql-config/nginx/graphql-nginx.conf`;
4) задать свой DNS вместо `graphql-demosite-rus-api.test`;
5) выполнить команду:

```
docker-compose up –d
```

4.5.2. Nginx без использования docker

Если nginx при установке QP8.CMS был запущен не в docker, то следует:

1) открыть на редактирование файл nginx.conf, предположительно расположенный по пути `/etc/nginx/nginx.conf`;
2) в конфигурационный файл добавить секции из файла `/home/qp/graphql-config/nginx/graphql-nginx.conf`;
3) задать свой DNS вместо `graphql-demosite-rus-api.test;`
4) проверить корректность конфигурации командой:

```
nginx -t
```

5) если всё корректно, то перечитать конфигурацию nginx командой:

```
nginx -s reload
```

5. Настройка плагина QP

Настройка параметров QP может осуществляться на нескольких уровнях.

5.1. Создание плагина QP

В БД демо-сайта, поставляемой вместе с продуктом QP8.WidgetPlatform, уже имеется созданный плагин (Рисунок 1).

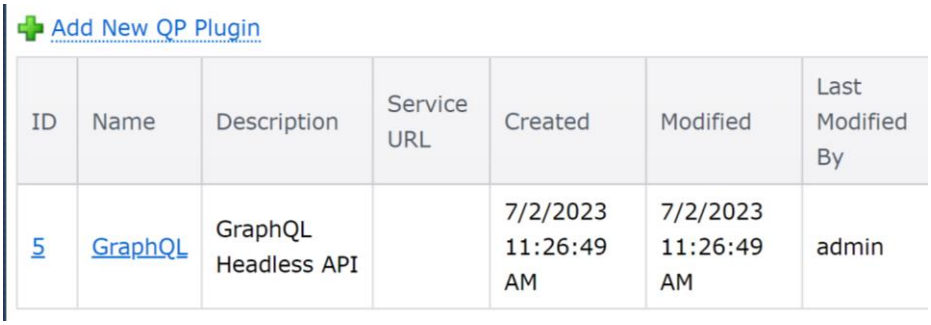

Рисунок 1 - Плагин демо-сайта

При необходимости создания нового плагина в контекстном меню следует выбрать «New QP Plugin» (Рисунок 2).

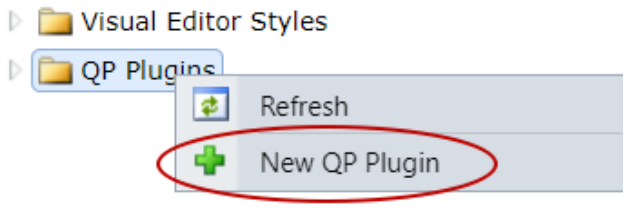

Рисунок 2 - Создание нового плагина

Основные параметры настройки следует выбирать в окне «Basic parameters». Система предлагает два способа его создания: через URL-код (<URL сервиса>/api/qpcontract) или через контракт (Рисунок 3).

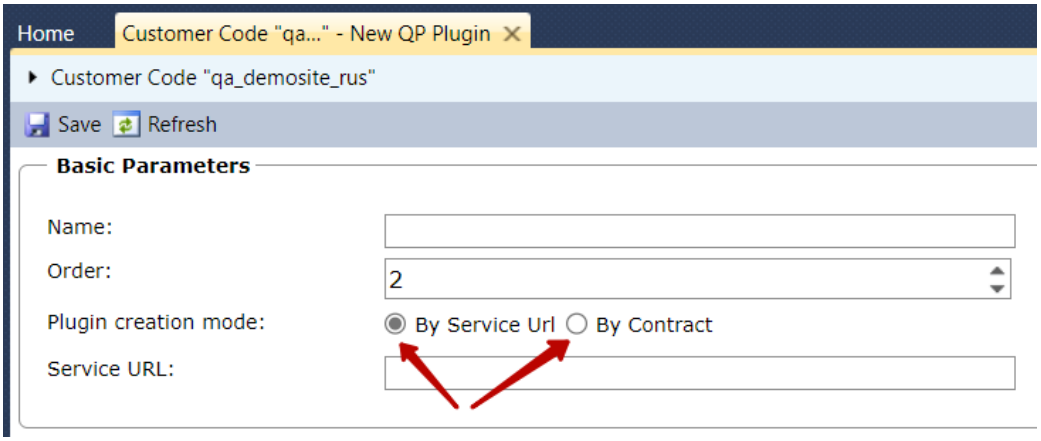

Рисунок 3 - Параметры создания плагина

Ниже представлен код контракта из демо-сайта:

```
{
    "code": "graphql",
    "description": "GraphQL Headless API",
    "version": "0.5.1",
    "instanceKey": "B3D1B88B-57F2-4933-BB79-1BF47B27F25A",
    "allowMultipleInstances": true,
    "fields": [
        {
            "id": 0,
            "name": "ApiKey",
            "description": "Ключ доступа для GraphQL API",
            "valueType": "String",
            "relationType": "Site",
            "sortOrder": 0
        },
        {
            "id": 0,
            "name": "MaxDepth",
            "description": "Максимальная глубина запроса",
            "valueType": "Numeric",
            "relationType": "Site",
            "sortOrder": 5
        },
        {
            "id": 0,
            "name": "MaxComplexity",
            "description": "Максимальный коэффициент сложности документа",
            "valueType": "Numeric",
            "relationType": "Site",
            "sortOrder": 10
        },
        {
            "id": 0,
            "name": "FieldImpact",
            "description": "Максимальное число объектов, возвращаемых каждым
полем",
            "valueType": "Numeric",
            "relationType": "Site",
            "sortOrder": 15
        },
        {
            "id": 0,
            "name": "MaxRecursionCount",
            "description": "Максимальное количество итераций для обхода узлов
дерева",
            "valueType": "Numeric",
            "relationType": "Site",
```

```
            "sortOrder": 20
        },
        {
            "id": 0,
            "name": "IsExposed",
            "description": "Доступность контента в GraphQL API",
            "valueType": "Bool",
            "relationType": "Content",
            "sortOrder": 0
        },
        {
            "id": 0,
            "name": "AliasSingular",
            "description": "Алиас в ед. числе в схеме GraphQL",
            "valueType": "String",
            "relationType": "Content",
            "sortOrder": 5
        },
        {
            "id": 0,
            "name": "AliasPlural",
            "description": "Алиас в множ. числе в схеме GraphQL",
            "valueType": "String",
            "relationType": "Content",
            "sortOrder": 10
        },
        {
            "id": 0,
            "name": "IsHidden",
            "description": "Недоступность поля в GraphQL API",
            "valueType": "Bool",
            "relationType": "ContentAttribute",
            "sortOrder": 0
        }
    ]
}
```

Примечание: подробнее о создании плагина в QP – см. раздел 4.17 (Плагины QP) Руководства разработчика.

5.2. Настройки сайта

На уровне сайта предлагаются следующие параметры настройки:

Таблица 1 - Описание параметров настройки сайта

Название	Описание
ApiKey	Ключ доступа для GraphQL. В случае, если ApiKey не задан, он не передается.

MaxDepth	Максимальная глубина (вложенность) запроса.
MaxComplexity	Максимальный коэффициент сложности документа.
FieldImpact	Максимальное число объектов, возвращаемых каждым полем.
MaxRecursionCount	Максимальное число рекурсий для обхода узлов дерева.

Данные параметры можно настроить, перейдя во вкладку «Properties» через контекстное меню (Рисунок 4, Рисунок 5). Подробнее о данных настройках – по ссылке.

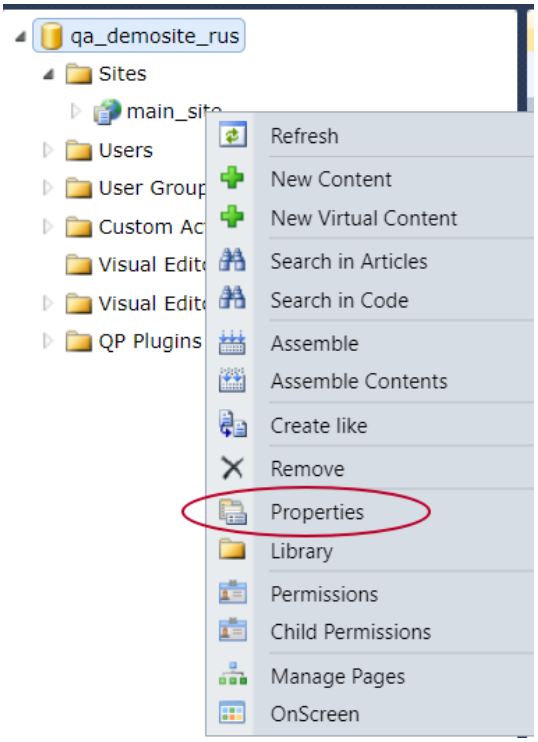

Рисунок 4 - Открытие свойств сайта через контекстное меню

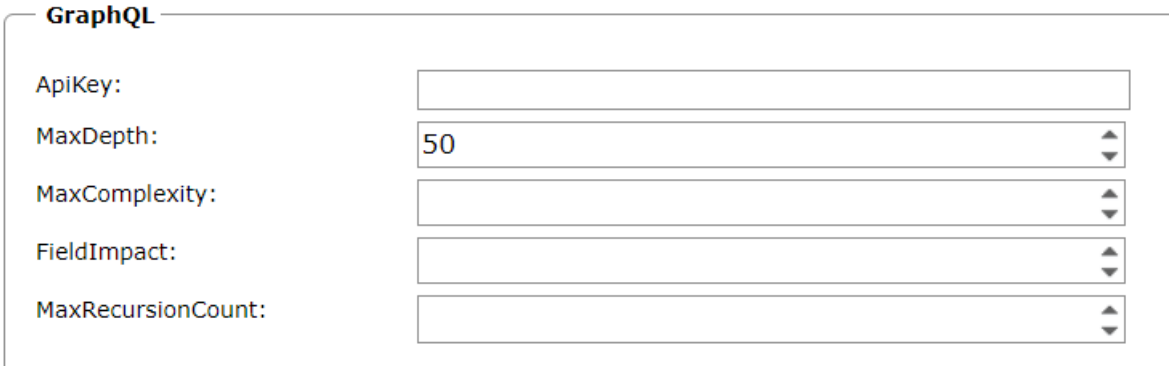

Рисунок 5 - Установка параметров сайта

5.3. Настройки контента

По умолчанию весь контент в QP не виден. Для отображения/редактирования данных необходимо в свойствах контента для нужных плагинов GraphQL поставить галочки, разрешающие отображение данных (Рисунок 6). В том же окне необходимо ввести имена сущности в единственном и множественном числе. При этом важно выбирать уникальные имена для каждого типа контента, включая формы единственного (AliasSingular) и множественного (AliasPlural) чисел:

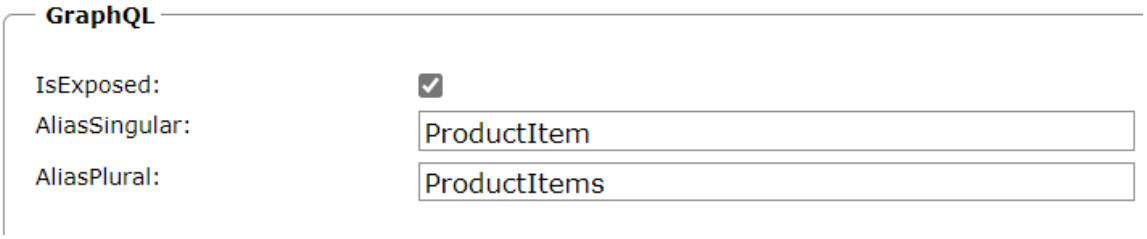

Рисунок 6 - Выбор контента для отображения

Рисунок 7 - Установка параметров сайта

5.4. Настройки поля

Если контент разрешен для отображения, то по умолчанию все поля выводятся. Для ограничения отображения полей необходимо для нужных плагинов GraphQL указать поля, которые запрещены для просмотра (Рисунок 8):

Рисунок 8 - Скрытие контента

6. Обновление схемы данных

При старте хоста API из QP загружается схема данных, которая нужна для валидации запросов и трансляции запросов в запросы к БД. Поскольку с течением времени данные, определяющие схему в QP, могут меняться (и новый контент добавляться), они должны стать доступны из API. Для контроля используется адрес API `<URL сервиса>/api/Schema/context`. Он возвращает данные в следующем виде:

```
{
    "loadDate": "2023-08-09T09:01:13.5858024+00:00",
    "state": "Active",
    "settings": {
        "schemaAutoReload": true,
        "schemaReloadInterval": "00:02:00"
    }
}
```

где:

loadDate - время последнего успешного обновления схемы;
state - статус текущего обновления: None / Loading / Active / Reloading;
settings.schemaAutoReload - автоматическое обновление схемы;
settings.schemaReloadInterval - интервал обновления схемы.

В приведенном случае после заведения/настройки контента им можно будет пользоваться в API через две минуты.

Также возможно принудительное обновление схемы по запросу POST: `<URL сервиса>/api/Schema/reload`

Кроме того, пользователь может посмотреть схему данных, доступную на текущий момент в API с помощью GraphQL API UI: `<URL сервиса>/ui/voyager`. Там отображаются все сущности со связями (Рисунок 9):

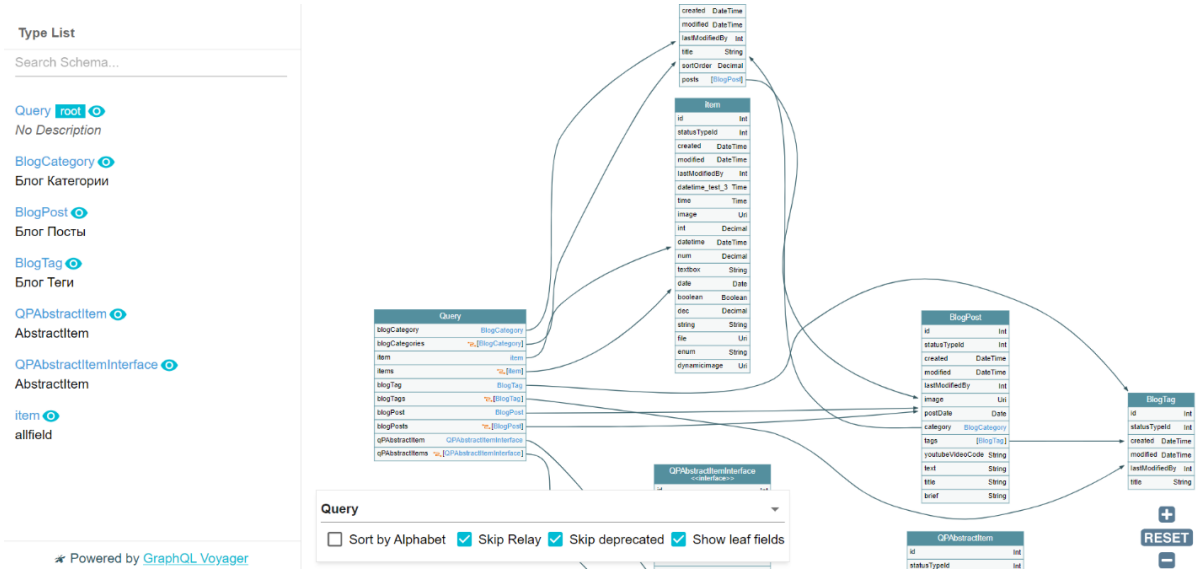

Рисунок 9 - Просмотр схемы связей

7. Запросы

UI для написания запросов к API выглядит следующим образом (Рисунок 10):

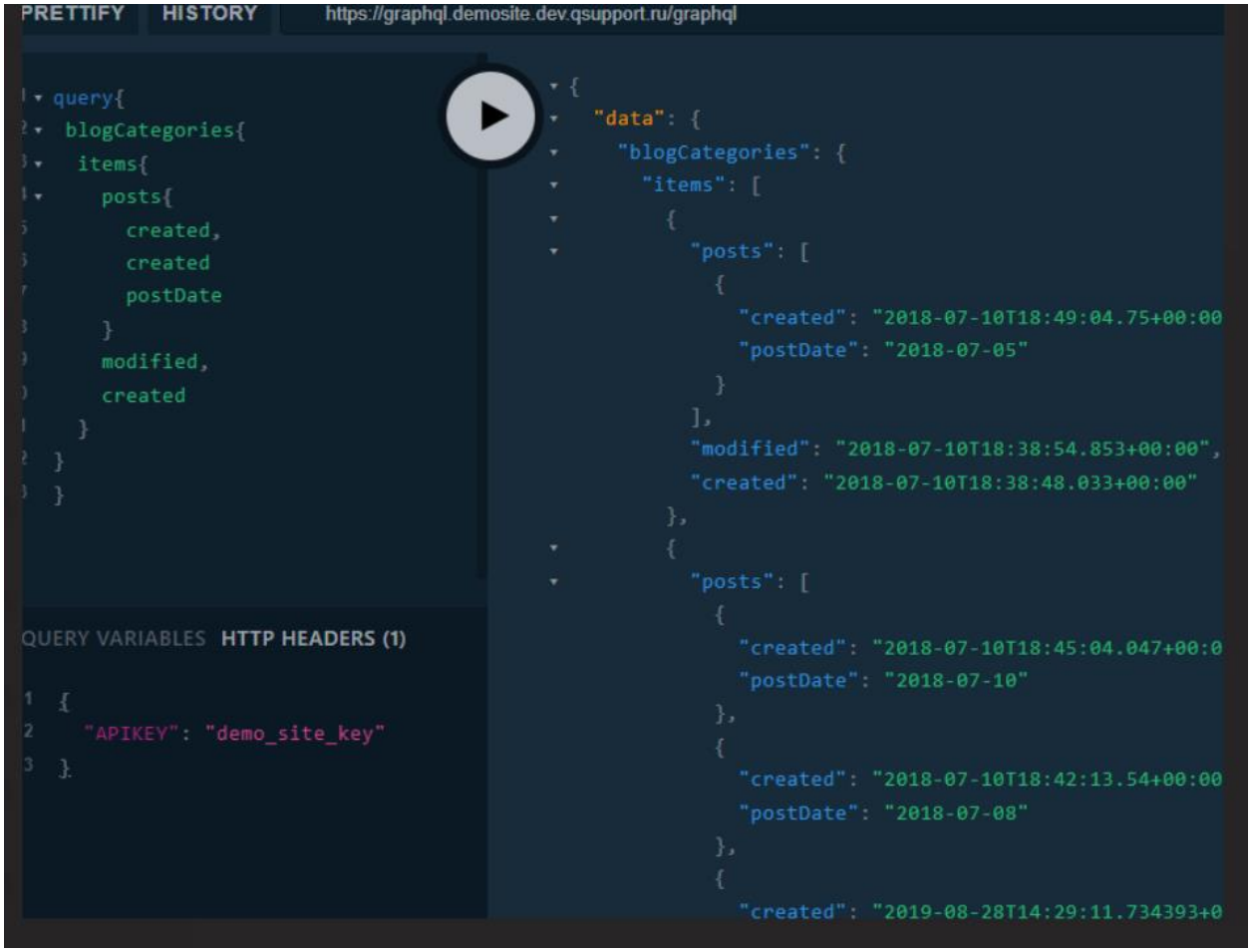

Рисунок 10 - UI для написания запросов

Одна из наиболее частых причин, почему фронт-приложение не получает данные – некорректное написание запроса к API.

Для решения данной проблемы можно воспользоваться GraphQL API UI: `<URL сервиса>/ui/v`. Он помогает написать запрос с помощью intellisense, осуществляет его проверку, а также показывает возвращаемые данные или ошибки. Если запрос написан некорректно, UI об этом предупредит, укажет причину ошибки, и предложит подсказки для пользователя (Рисунок 11):

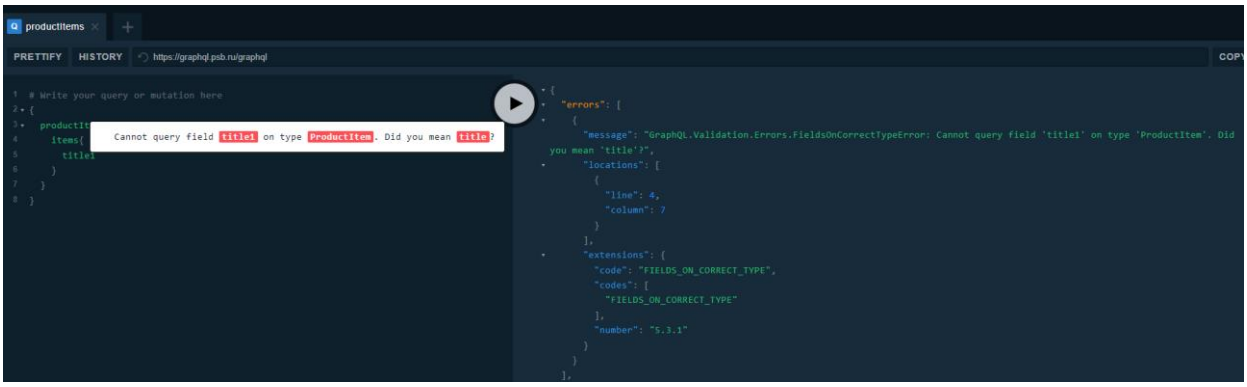

Рисунок 11 - UI при ошибке в написании запроса

Ниже перечислены примеры типовых запросов.

Пример 1. Вывод статей для каждой категории:

```
query {
blogCategories{
    items{
      id
      title
      sortOrder
      posts{
        postDate
        title,
      }
    }
  }
}
```

Пример 2. Запрос с параметрами и фильтрацией:

```
query blogPostsQuery($title: String, $date: Date, $order:
[PossibleOrderForBlogPost]){
  blogPosts(
    filter: {
      titleLike : $title
      postDateEq: $date
    }
    order: $order){
    items{
      id
      title
      postDate
      category{
        title
        sortOrder
      }
    }
```

```
  }
}
```

Пример 3. Запрос с курсорной навигацией относительно id-записи:

```
query abstractItemsQuery($first: Int, $after: String)
 qPAbstractItems(first: $first, after: $after){
    items{
      id
      title
    }
    pageInfo{
      hasNextPage
      hasPreviousPage
      startCursor
      endCursor
    }
    totalCount
  }
}
```

Пример 4. Запрос с постраничной навигацией (пейджингом):

```
query abstractItemsPagingQuery($first: Int, $skip: Int){
qPAbstractItems(first: $first, skip: $skip){
    items{
      id
      title
    }
    totalCount
  }
}
```

8. Диагностика и коррекция ошибок

Неправильная настройка контента может привести к ошибке загрузки схемы в API. Для диагностики такого сценария следует:

1. Убедиться, что необходимый контент отсутствуют в схеме с помощью GraphQL API UI: `<URL сервиса>/ui/playground` или `<URL сервиса>/ui/voyager`.
2. Если контент там уже есть, перейти к разделу «Отладка запросов» в GraphQL API.
3. Если контента еще нет, открыть адрес `<URL сервиса>/api/Schema/context` и посмотреть дату и статус обновления схемы.
4. Если последний раз схема обновлялась менее двух минут назад, следует немного подождать, когда схема обновится сама. В качестве альтернативы ожиданию, можно сделать запрос POST: `<URL сервиса>/api/Schema/reload`
5. Если схема обновлялась очень давно и запрос `<URL сервиса>/api/Schema/reload` выдает ошибку 500, следует скорректировать настройку контента в QP.
6. После внесения исправлений схема автоматически обновится через две минуты, а недостающий контент появятся в AP.

Существует несколько вариантов поиска ошибок контента. Ниже представлено их описание.

8.1. Проверка лог-файлов

В лог-файлах напрямую указывается, если некоторые алиасы контента повторяются (Рисунок 12).

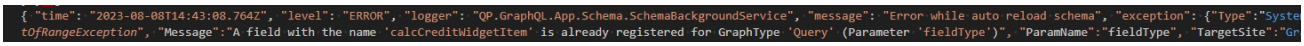

Рисунок 12 - UI для диагностики ошибок через лог-файл

Так, в указанном примере контент с алиасом calcCreditWidgetItem используется более одного раза. Скорее всего, пользователем было указано идентичное название для единственной и множественной форм (см. п. 5.3. Настройки контента). В таком случае следует найти контент по названию и исправить его. Если же разные типы контента названы одинаково, то найти нужный контент будет проблематично. Тем не менее, проверка лог-файлов позволяет диагностировать любые типы ошибок.

8.2. Просмотр списка контента

Следует открыть в QP список контента и отсортировать по убыванию поля Modified (дата изменения). Верхние записи, у которых дата обновления позже даты последней загрузки схемы, таким образом, будут отображать проблемный контент, который нужно проверить и исправить (Рисунок 13).

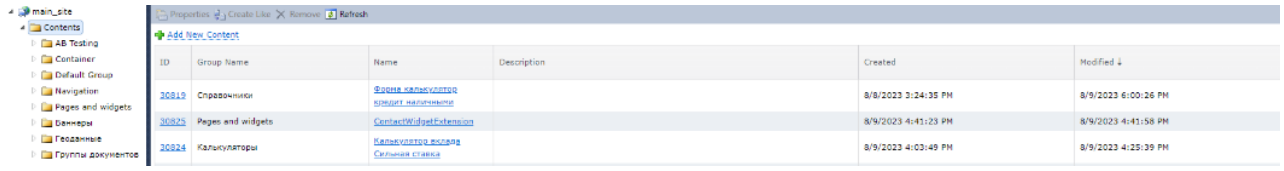

Рисунок 13 - UI для диагностики ошибок через список контента

9. Конфигурация веб-сервиса

Настройки веб-сервиса задаются в конфигурационном файле appsetings.json.

Пример конфигурационного файла:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "ConnectionStrings": {
    "QPConnectionString": "connection to QP database",
    "Type": "Postgres"
  },
  "QpArticlesAccessor": {
    "CalculatePagingData": true
  },

  "GraphQLRouting": {
    "BasePath": "/graphql",
    "States": {
      "Live": {
        "Allow": true,
        "Default": true
      },
      "Stage": {
        "Allow": true
      }
    }
  },

  "SchemaAutoReload": true,
  "SchemaReloadInterval": "00:02:00",
  "EnableMetrics": true,
  "ExposeExceptions": true,

  "QpPluginSettings": {
    "ContractVersion": "0.5.1",
    "InstanceKey": "B3D1B88B-57F2-4933-BB79-1BF47B27F25A"
  }
```

Описание параметров конфигурационного файла:

Таблица 4 - Описание секций

Название	Описание
ConnectionStrings	Параметры подключения к БД
QPConnectionString	Подключение к базе QP
Type	Тип БД: Postgres или SqlServer
GraphQLRouting	Настройки URL- роутинга
BasePath	Базовый URL приложения
States	Настройка доступных состояний приложений. По умолчанию настроены два состояния: live и stage. Состояние live выбрано в качестве дефолтного (не требует дополнительных параметров URL).
SchemaAutoReload	Включение режима автоматической перезагрузки схемы
SchemaReloadInterval	Интервал перезагрузки схемы (если SchemaAutoReload = true)
QpPluginSettings	Настройки для плагина GraphQL в QP, если плагин настроен на получение контракта через endpoint текущего приложения
ContractVersion	Версия контракта
InstanceKey	Уникальный ключ приложения. Может быть полезным при одновременном использовании нескольких экземпляров приложения в одной базе QP