ООО «КВАНТУМ АРТ»

115184, Москва, Озерковский переулок, д. 12

тел. (495) 783-65-74

Модуль React для QP8.WidgetPlatform

Руководство пользователя

Москва
2023

История изменений

Версия	Дата	Автор	Описание
1.1	16.10.2023	Молькова М.Е.	Добавлены разделы: • 4. Описание API виджетной платформы • 5. Руководство по установке модуля Angular для QP8WidgetPlatform (Linux)
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.	**Примечание:** используется при генерации классов LINQ to SQL.
Внимание:	Важные данные, которые требуется обязательно учитывать.	**Внимание:** опция поддерживается только ASP-сборкой в целях совместимости.

2. Определения и термины, используемые в документе

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
Apollo Client	Библиотека для управления состоянием React-приложений, позволяющая управлять как локальными, так и удаленными данными с помощью GraphQL
Module Federation	Плагин для Webpack, с помощью которого можно разбивать приложение на микрофронтенды, подключив их в хост приложение
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»)
React Context	Решение для упрощения обмена данными (состоянием) между компонентами приложений
React Router	Решение для переключения и маршрутизации страниц React
SSR	«Server Side Rendering» - рендеринг на сервере клиентской части или универсального приложения в HTML
WP.API	API виджетной платформы QP8.Widgets
URL	«Uniform Resource Locator» – система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса
БД	База данных
ГПИ	Графический пользовательский интерфейс
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

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
NPM	«Node Package Manager» – пакетный менеджер для JavaScript, работающий на Node.js
JSON	«JavaScript Object Notation» — текстовый формат хранения и передачи структурированных данных в формате «ключ:значение»
DNS	«Domain Name System» — система доменных имен
CORS	«Cross-Origin Resource Sharing» — механизм обеспечения безопасности и защиты пользователей, позволяющий определить список ресурсов, к которым страница может получить доступ

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.

3. Общие сведения

Модуль **React** для продукта **QP8.WidgetPlatform** (или QP8.WidgetPlatform.React) – это набор средств, позволяющих разработать сайт на технологии *React* с использованием возможностей виджетной платформы **QP8.WidgetPlatform**.

3.1. Состав модуля

Модуль состоит из:

- библиотек @quantumart/qp8-widget-platform-shell-core, @quantumart/qp8-widget-platform-shell, @quantumart/qp8-widget-platform-module, @quantumart/qp8-widget-platform-bridge (доступны в репозитории npm.js, включаются в состав сайтов, разрабатываемых на технологии React);
- сервиса API виджетной платформы;

Также в состав дистрибутива входит демо-сайт, показывающий возможности виджетной платформы на технологии React.

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

Внимание: Компонент WP.API может использоваться только либо для live, либо для stage (по умолчанию). Если необходимы оба режима, нужно установить ещё один экземпляр сервиса, поменяв название сервиса и порт.

Внимание: Компонент WP.API (сервис API виджетной платформы) входит также в состав модуля Angular. В случае одновременной установки обоих модулей можно использовать уже установленный компонент и закомментировать соответствующие разделы манифестов / не выполнять соответствующие шаги установки, либо установить ещё один экземпляр сервиса, поменяв название сервиса и порт.

3.2. Описание библиотек

Для работы сайта, содержащего структуру страниц виджетной платформы QP, реализовано четыре проекта:

- Shell;
- Shell-core;
- Bridge;
- Module.

Shell – обертка структуры сайта, которая загружает структуру сайта виджетной платформы QP с помощью Web API; создает структуру сайта на React Router, обеспечивает асинхронную загрузку параметров виджетов в случае необходимости, содержит в себе Apollo Client для взаимодействия с контентами QP.

Module – проект, содержащий в себе компоненты (визуальное представление) для отдельных типов страниц и виджетов в виджетной платформе на основе QP.

Shell-core - содержит в себе код бизнес-логики для проекта Shell; выделен отдельный npm-пакет для более простого обновления и доработки бизнес логики.

Bridge – проект предназначен для передачи данных от проекта Shell проекту Module через React Context.

Проекты Shell и Module работают по технологии Module Federation.

Примечание: дополнительные ссылки на документацию:
1) https://github.com/QuantumArt/qp8-widget-platform-shell
2) https://github.com/QuantumArt/qp8-widget-platform-shell-core
3) https://github.com/QuantumArt/qp8-widget-platform-bridge
4) https://github.com/QuantumArt/qp8-widget-platform-module/watchers

3.3. Логика работы

Логика работы модуля `React` для виджетной платформы `QP8.WidgetPlatform` состоит из отображения модулем React контента виджетной платформы `QP8.WidgetPlatform`.

Рассмотрим логику работы модуля `React` на примере создания страницы и полей в QP8.WidgetPlatform и ее отрисовки модулем `React`:

1. Создадим новый тип страницы в `QP8.WidgetPlatform`:
   1) перейдем в раздел `main_site/Container/Articels`, добавить новую статью;
   2) заполним форму создания статьи, ограничившись основными полями (Рисунок 1).
      - Title – заголовок страницы;
      - FrontModuleURL – URL, содержащий папки client и server, которые, в свою очередь, содержат входную точку модуля – remoteEntry.js;
      - страницы (укажем локальный адрес);
      - ModuleName – название модуля;
      - Identifier – уникальный идентификатор страницы или виджета;
   3) сохраним введенные данные, нажав на кнопку «Save»;
   4) опубликуем статью, установив статус «Published».

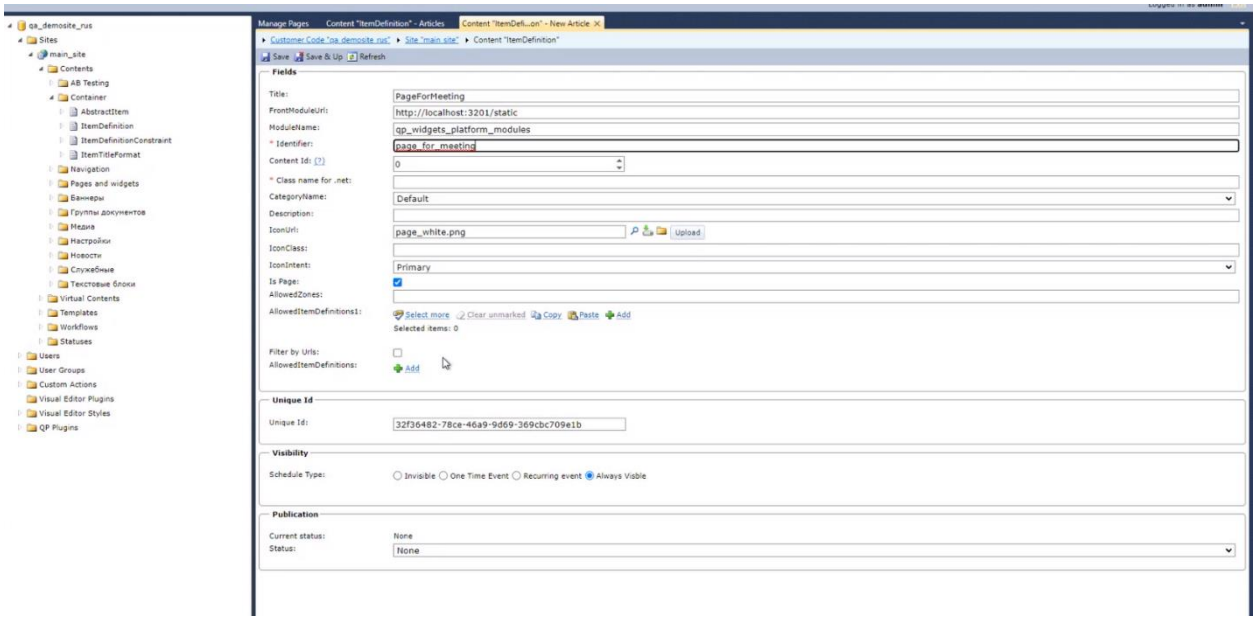

Рисунок 1 Форма создания статьи в QP8.WidgetPlatform

2. Создадим новый тип контента для созданной на первом шаге страницы:
   1) перейдем в раздел `main_site/Container/Pages and Widgets`, далее нажмем на «New Content» по правой кнопке;
   2) заполним форму создания поля, ограничившись основными полями (Рисунок 2);
   3) сохраним введенные данные, нажав на кнопку «Save».

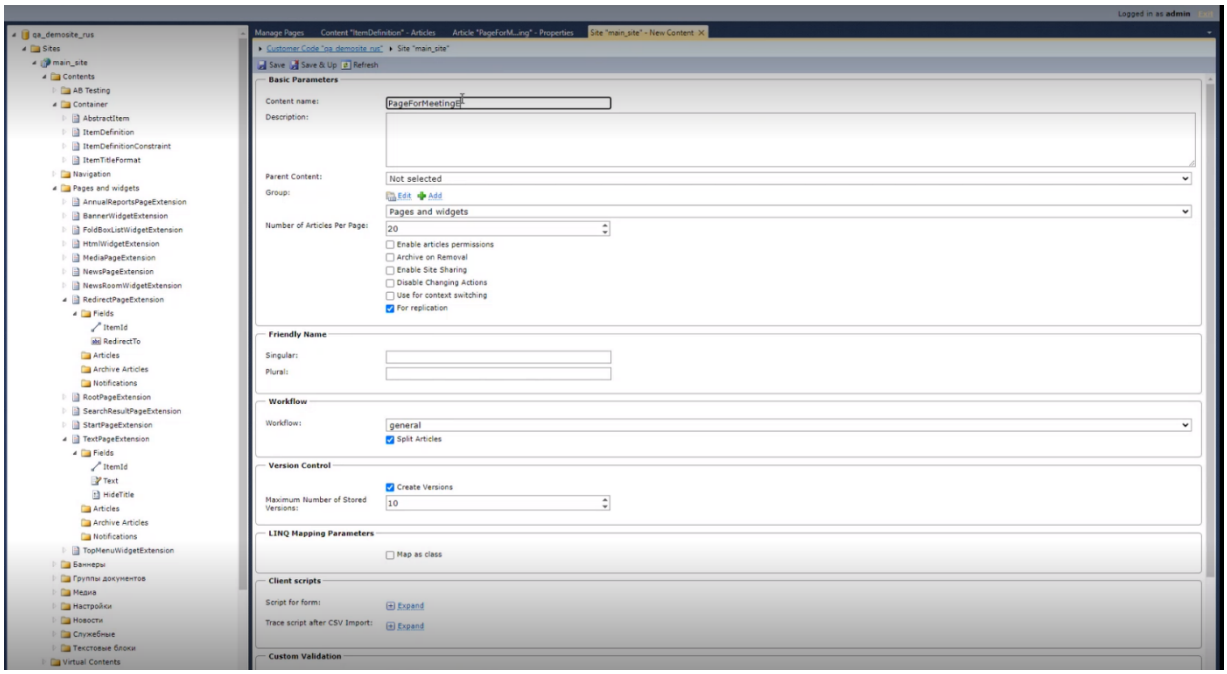

Рисунок 2 Создание нового поля для страницы

4) скопируем ID созданного контента и вставим его в форму созданной страницы в поле «Content ID».

3. Добавим новый тип страницы в структуру:

1) перейдем в раздел main_site;

2) далее перейдем в Template page и нажимаем «Add item» по правой кнопке мыши;

3) заполним форму (Рисунок 3):

- Tittle – название поля;
- Alias – тип поля;
- Content Type – выбираем из раскрывающегося списка, страницы которая была создана на первом шаге.

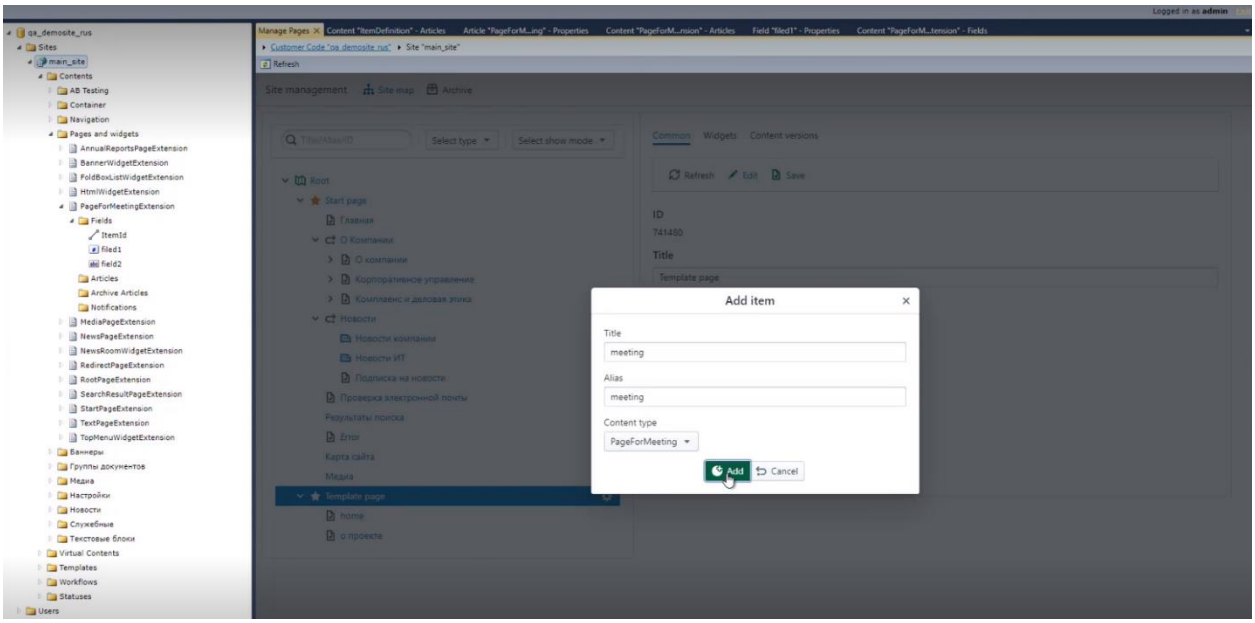

Рисунок 3 Добавление нового типа страницы в структуру сайта

4. Для того чтобы модуль `Shell` знал, как отображать новый тип страницы, необходимо указать данный тип в проекте Module:
    1) создаем в `Module` новую папку `page_for_meeting Module/src/components/pages` и добавляем новый файл `page_for_meeting.tsx`;
    2) в файле `module-federation.js Module/config` указываем ссылку (Рисунок 4);
    3) в файле `page_for_meeting.tsx` прописываем следующий код (Рисунок 5);

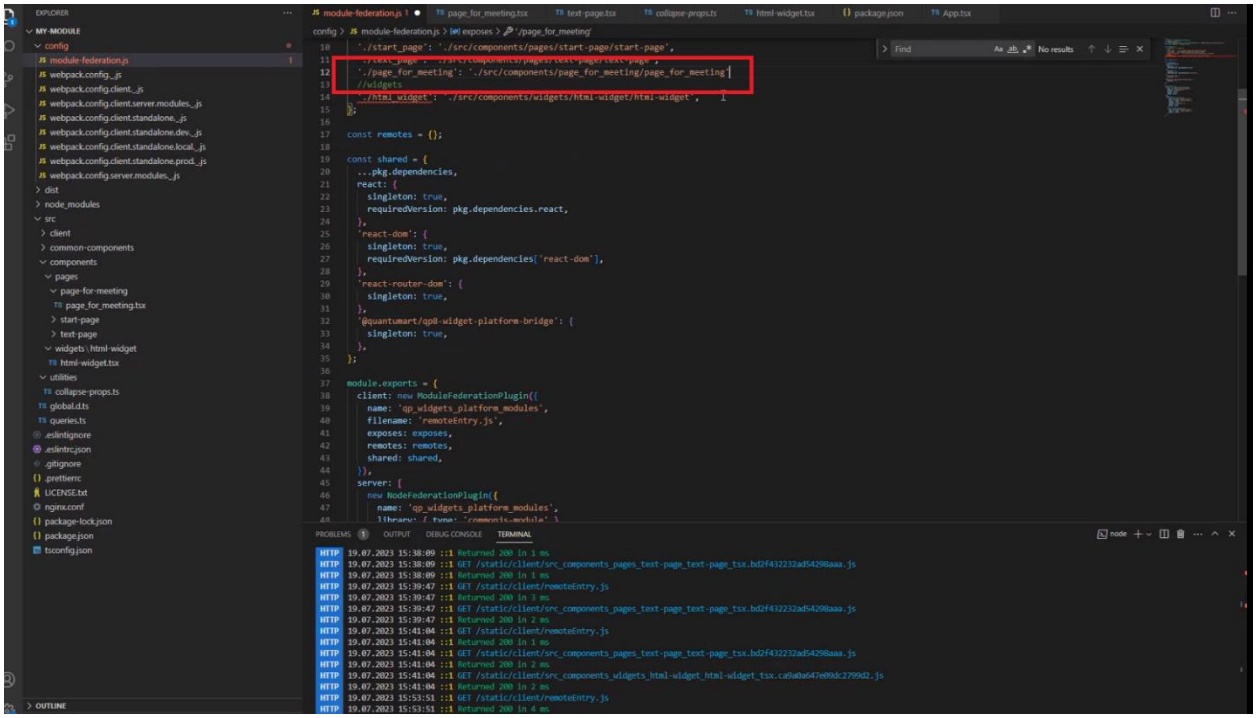

Рисунок 4 Указание нового типа страницы в Module

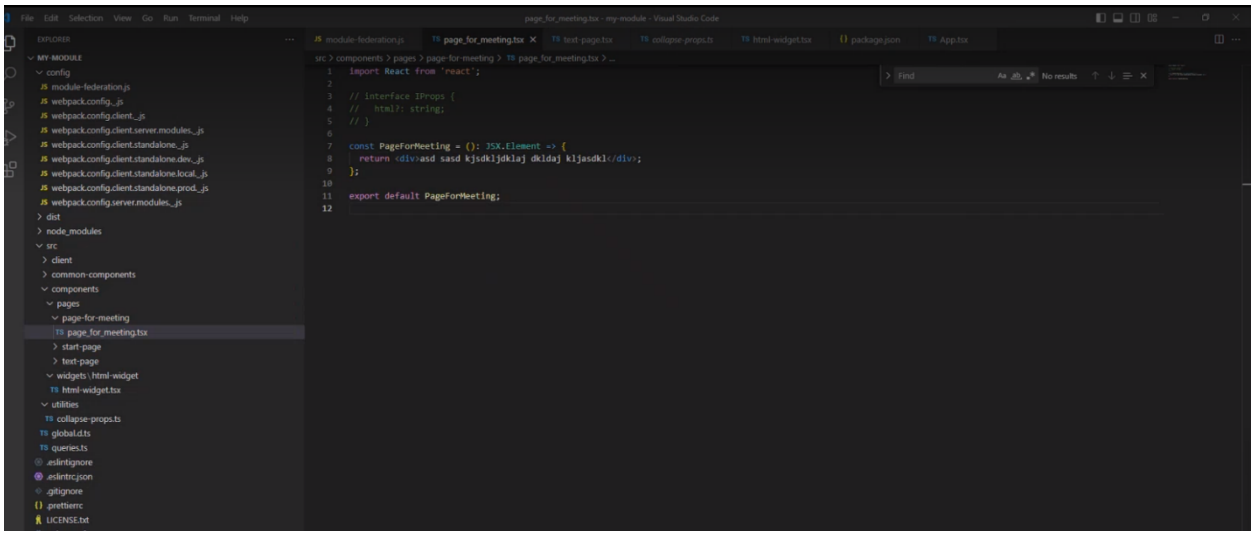

Рисунок 5 Указание нового типа страницы в Module

5. В случае если используется статическая загрузка модулей, в проект Shell необходимо также прописать ссылку на новый тип страницы. В файле `static-wpc-modules.ts Shell/src/app-settings-shell` прописываем ссылку на новый тип страницы (Рисунок 4).

   Также возможен динамический импорт с `QP8.WidgetPlatform`.

Для этого необходимо выполнить следующие настройки:

1) удалить информацию из файла `static-wpc-modules.ts Shell/src/app-settings-shell` по ссылке на новый тип проекта;
2) указать в файле `settings.json` проекта `Shell` динамические настройки и отключить `SSR`:

```
  "userDynamicModule": true
  "ssr": {
    active:false,
   ttl: 30
  }
```

3) в файле `module federation.js Module/config` выключаем загрузку модуля (Рисунок 6).

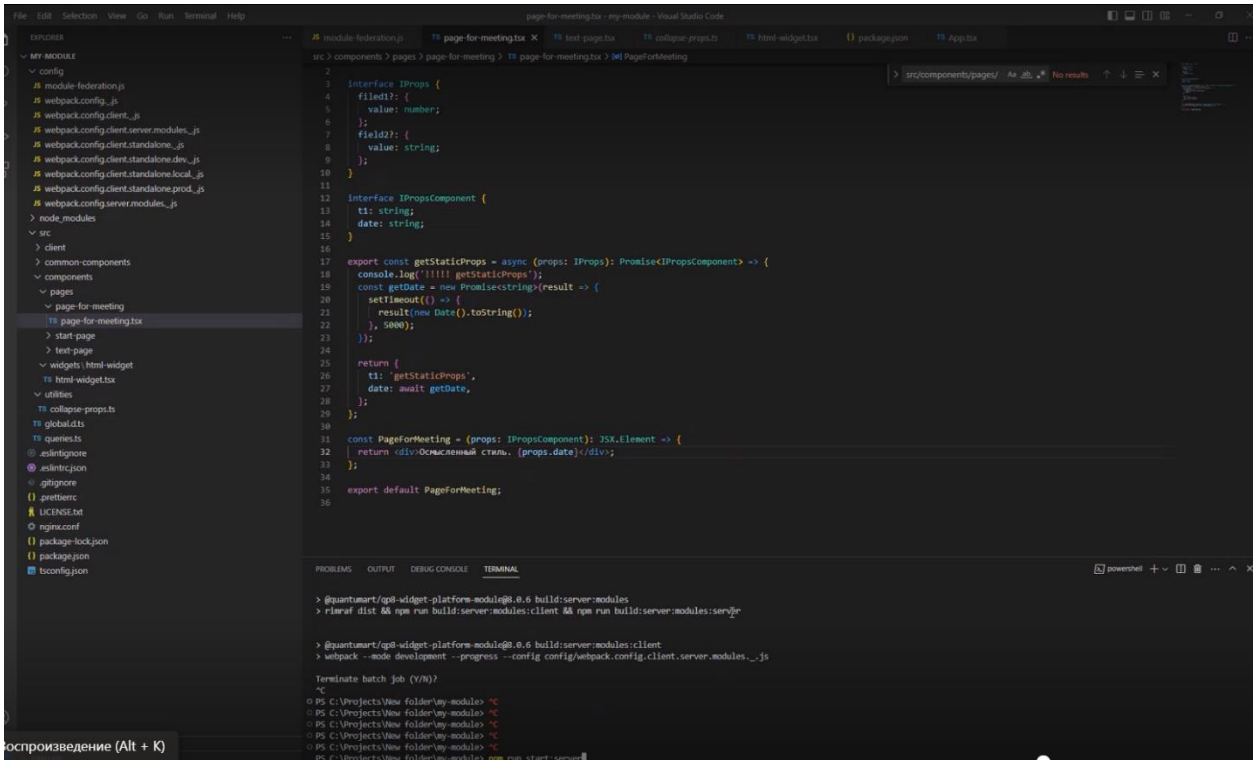

Рисунок 6 Выключение загрузки модуля

- Функция `getStaticProps` – функция для получения данных во время сборки.

Страница, на которой экспортируется асинхронная функция `getStaticProps`, предварительно рендерится с помощью возвращаемых этой функцией пропсов.

```
export async function getStaticProps(context) {
  return {
    props: {},
  };
}
```

Context — это объект со следующими свойствами:

- props — параметры виджета с QP.

- environment — дополнительно переданные параметры, такие как href (содержит текущий URL), graphQLClient (содержит клиент для GraphQL).

Пример работы функции - Рисунок 7.

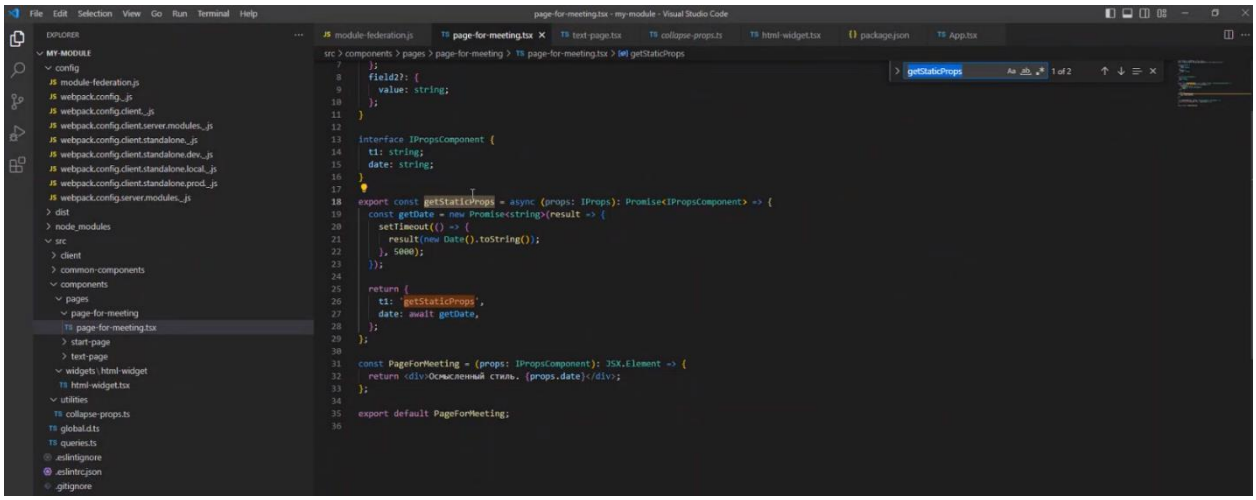

Рисунок 7 Пример работы функции getStaticProps

Связка Shell и Module позволяет получать данные с QP8.WidgetPlatform с использованием GraphQL клиента. В данный момент используется Apollo Client (Рисунок 8).

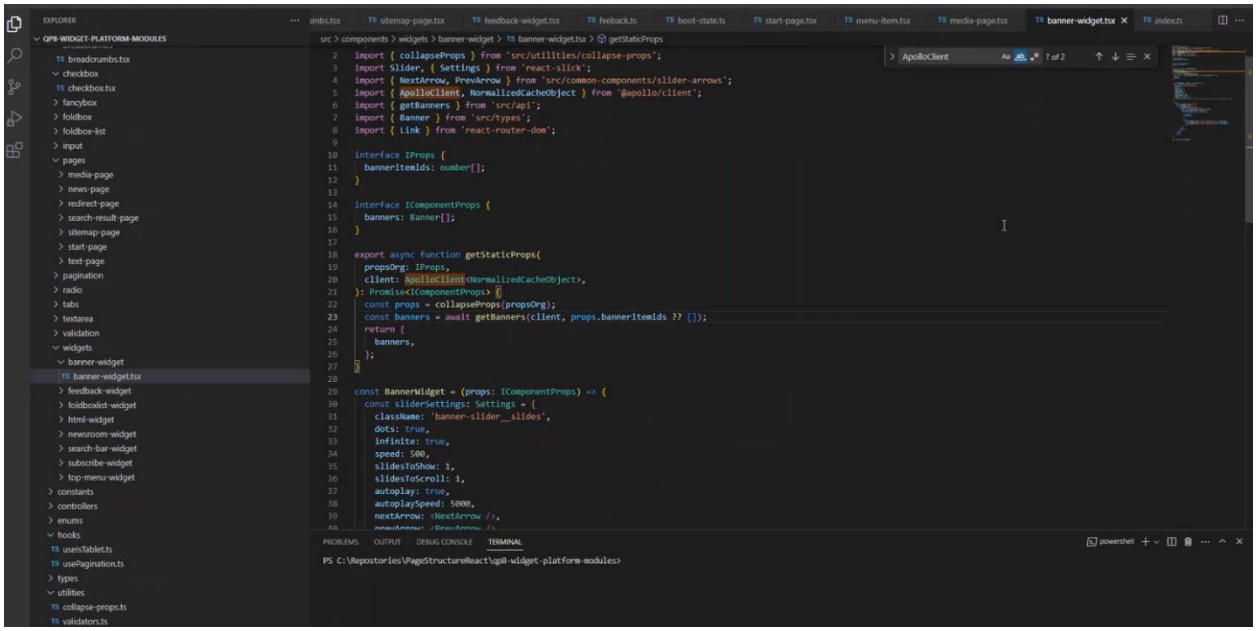

Рисунок 8 Пример работы ApolloClientma

6. Для созданного контента в предыдущем разделе создадим два поля:
   1) Перейдем в раздел main_site/Container/Pages and Widgets/PageForMeetingExtensions/Fields, далее нажимаем на «Add Fields» по правой кнопке мыши.
   2) Заполним форму создания поля, ограничившись основными полями:
      - Name – название поля;
      - FieldType – тип поля.
   3) Сохраним введенные данные, нажав на кнопку «Save».

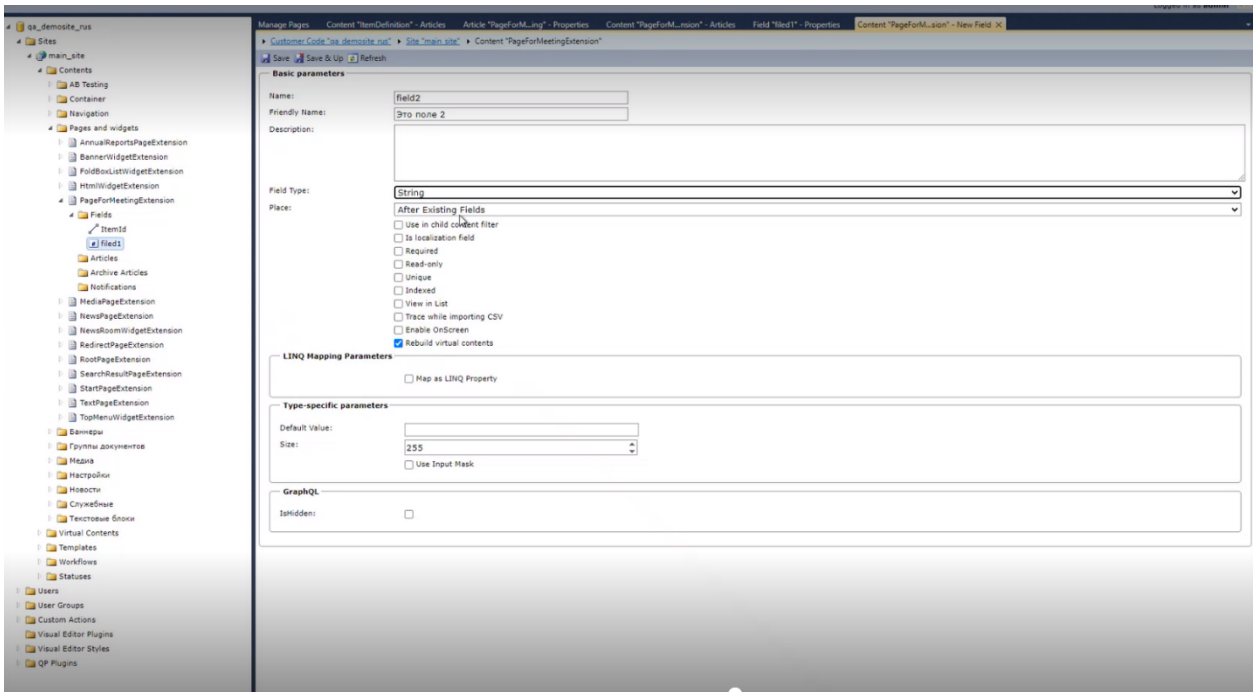

Рисунок 9 Создание полей для контента

7. Для отображения созданных полей в QP8.WidgetPlatform необходимо добавить в файл page_for_meeting.tsx Module/src/components/pages следующий код (Рисунок 10).

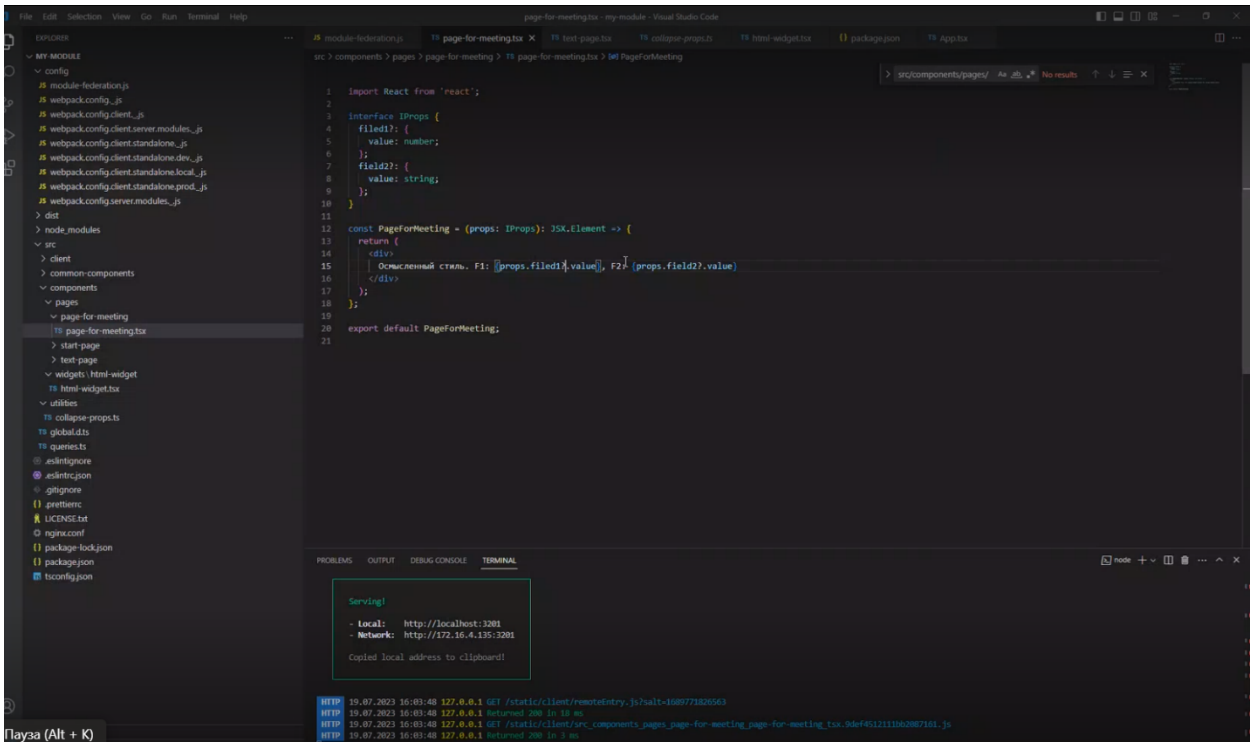

Рисунок 10 Настройка отображения полей в QP8

3.4. Использование модуля

Предварительно должны быть установлены QP8.WidgetPlatform и Web API.

1. В консоли выполнить следующие команды установки проектов Shell и Module:
   - npx @quantumart/qp8-widget-platform-shell my-shell;
   - npx @quantumart/qp8-widget-platform-module my-module.

2. Запустить проект Module и перейти в package.json module/config/package.json. В терминале ввести следующую команду: npm run start.

   Для сборки проекта под SSR и front здесь же выполнить команду: npm run start:server.

   Проект Module не предназначен работать в отдельности (по адресу localhost: 3201 не будет данных для отображения), поэтому следует перейти к запуску проекта Shell.

3. Запустить проект Shell и перейти в settings.json: shell/stc/appsettings/settings.json. В терминале ввести следующую команду: npm run start:server.

   Открыть в браузере localhost:3200. В браузере отображается страница home в QP8.

4. Описание API виджетной платформы

Сервис Web API позволяет получать информацию о структуре страниц и виджетов с помощью REST-методов. Web API позволяет получить те же данные, которые доступны в интерфейсе административной панели управления структурой сайта.

4.1. Предварительная загрузка сайта

Описание: метод предназначен для предварительной загрузки кэша страниц и виджетов сайта.

Формат запроса:

```
GET /Site/Warmup
```

4.2. Получение структуры страниц сайта

Описание: метод предназначен для получения структуры страниц сайта.

Формат запроса:

```
GET /Site/structure
```

Входные данные:

Название	Тип данных	Описание	Дополнительная информация
dnsName	string	Доменное имя сайта	Required
t	object	Словарь значений таргетирования	
fields	array[string]	Поля деталей к выдаче	Если пусто, то детали выдаваться не будут
deep	integer($int32)	Глубина структуры	0 - это корневой элемент
fillDefinitionDetails	boolean	Заполнять дополнительные поля из Definition	По умолчанию: false

Ответ:

В ответе получаем:

- код состояния:
  - 200 (Success),
  - 400 (Bad Request),
  - 404 (Not Found);
- тип данных (Media type):
  - application/json;
- значение в json – пример указан ниже:


```
{
  "id": 741114,
  "alias": "start_page",
  "nodeType": "start_page",
  "children": [
    {
      "id": 741164,
      "alias": "home",
      "nodeType": "text_page"
```

```
    },
    {
      "id": 15,
      "alias": "news_and_events",
      "nodeType": "redirect_page",
      "children": [
        {
          "id": 6,
          "alias": "corporate_releases",
          "nodeType": "news_page"
        },
        {
          "id": 40,
          "alias": "technology_news",
          "nodeType": "news_page"
        },
        {
          "id": 442,
          "alias": "subscribe",
          "nodeType": "text_page"
        }
      ]
    },
    {
      "id": 455,
      "alias": "media",
      "nodeType": "media_page"
    },
    {
      "id": 17,
      "alias": "CheckEmailInformation",
      "nodeType": "text_page"
    },
    {
      "id": 275,
      "alias": "Error",
      "nodeType": "text_page"
    },
    {
      "id": 431,
      "alias": "sitemap",
      "nodeType": "sitemap_page"
    },
    {
      "id": 44,
      "alias": "searchresultpage",
      "nodeType": "search_result_page"
    }
  ]
}
```


Ниже представлено описание ключей типовых ответов формата JSON:

Ключ	Описание	Соответствующий контент	Поле контента
id	уникальный идентификатор страницы или виджета	AbstractItem	ID
alias	алиас	AbstractItem	Name / Имя
nodeType	тип виджета/страницы	ItemDefinition	Name / Имя
children	дочерние страницы		

4.3. Получение плоского списка страниц и виджетов по заданным параметрам таргетирования

Описание: метод предназначен для получения массива объектов, соответствующих заданным фильтрам.

Формат запроса:

```
GET /Site/details
```

Входные данные:

Название	Тип данных	Описание	Дополнительная информация
dnsName	string	Доменное имя сайта	Required
t	object	Коллекция параметров таргетирования	Ключом является параметр таргетирования, а значением – конкретное значение таргетирования
fields	array[string]	Поля деталей к выдаче	Если пусто, то будут выведены все детали

Ответ:

В ответе получаем:

- код состояния:
  - 200 (Success),
  - 400 (Bad Request),
  - 404 (Not Found);
- тип данных (Media type):
  - application/json;
- значение в json – пример указан ниже:


```
[
  {
    "id": 741114,
    "details": {
      "bindings": {
        "value":
"*\nlocalhost:5001\nlocalhost:54832\nmscdev02:7714\nlocalhost:5502\nlocalhost:3000\nd
emositerus.dev.qsupport.ru"
      },
      "mode": {
        "value": "Redirect"
      },
      "defaulthost": {
        "value": "https://demositerus.dev.qsupport.ru"
      },
      "title": {
        "value": "Start page 2"
      },
      "isvisible": {
        "value": true
      },
      "ispage": {
        "value": true
      },
```

```
      "isinsitemap": {
        "value": true
      },
      "titleformat": {
        "value": 741017
      }
    }
  },
  {
    "id": 741164,
    "details": {
      "hidetitle": {
        "value": true
      },
      "title": {
        "value": "Главная"
      },
      "isvisible": {
        "value": false
      },
      "ispage": {
        "value": true
      },
      "isinsitemap": {
        "value": true
      },
      "titleformat": {
        "value": 741017
      }
    }
  },
  {
    "id": 161,
    "details": {
      "text": {
        "value": "<!--<div class=\"page__banner\" style=\"background-image:
url('/static/images/committe-banner.jpg')\"></div>-->\n<section
class=\"page__section\">\n<div class=\"wrapper\">\n<div class=\"page__block\">\n<h1
class=\"h1 center\">Политика по соблюдению антикоррупционного</h1>\n<div
class=\"page__text center\">\n<p>Для обеспечения соответствия деятельности компании
требованиям применимого антикоррупционного законодательства, выявления, оценки,
анализа и минимизации коррупционных рисков в Компании выстраивается система
комплаенс. Утверждена Политика по соблюдению антикоррупционного законодательства и
Политика по управлению конфликтом интересов, в соответствии с которыми должен
действовать каждый сотрудник.</p></div></div></div></section>"
      },
      "hidetitle": {
        "value": false
      },
      "title": {
        "value": "Антикоррупционная политика"
      },
      "isvisible": {
        "value": true
      },
      "ispage": {
        "value": true
      },
      "isinsitemap": {
        "value": true
```

```
      },
      "titleformat": {
        "value": 741017
      }
    }
  }
]
```

Ниже представлено описание основных ключей типовых ответов формата JSON:

Ключ	Описание	Соответствующий контент	Поле контента
id	уникальный идентификатор страницы или виджета	AbstractItem	ID
details	данный ключ содержит большое количество полей, в т.ч. из контентов-расширений (Extensions); ниже описаны лишь основные поля		
details.bindings.value	стартовые DNS-адреса	AbstractItem	Bindings
details.mode.value	режим функционирования стартовой страницы (контентная страница / перенаправление)	AbstractItem	Mode
details.keywords.value	ключевые слова; для SEO	AbstractItem	Keywords / MetaKeywords
details.metadescription.value	описание метаданных; для SEO	AbstractItem	MetaDescription
details.tags.value	тэги; для SEO	AbstractItem	Tags / Meta tags
details.title.value	название страницы или виджета	AbstractItem	Title / Заголовок
details.ispage.value	указывает, что статья содержит данные о странице, а не виджете	AbstractItem	IsPage
details.titleformat.value	формат названия страницы или виджета	AbstractItem	TitleFormat / Шаблон заголовка
details.isinsitemap.value	указывает, что статья содержится в структуре сайта	AbstractItem	IsInSiteMap / Показывать в структуре сайта
details.isvisible.value	видимость элемента	AbstractItem	IsVisible / Отображать в навигации

4.4. Получение детальной информации по странице или виджету

Описание: метод предназначен для получения детальной информации по странице или виджету.

Формат запроса:

```
GET /Site/node/{nodeId}
```

Входные данные:

Название	Тип данных	Описание	Дополнительная информация
nodeId	integer($int32)	ID страницы или виджета	Required

Ответ:

В ответе получаем:

- код состояния:
  - 200 (Success),
  - 400 (Bad Request),
  - 404 (Not Found);
- тип данных (Media type):
  - application/json;
- значение в json – пример указан ниже:


```
{
    "id": 431,
    "alias": "sitemap",
    "nodeType": "sitemap_page",
    "details": {
      "title": {
        "value": "Карта сайта"
      },
      "isvisible": {
        "value": true
      },
      "ispage": {
        "value": true
      },
      "isinsitemap": {
        "value": true
      },
      "titleformat": {
        "value": 741017
      }
    }
  }
```


Ниже представлено описание основных ключей типовых ответов формата JSON:

Ключ	Описание	Соответствующий контент	Поле контента
id	уникальный идентификатор страницы или виджета	AbstractItem	ID
alias	алиас	AbstractItem	
nodeType	тип виджета/страницы	ItemDefinition	Name / Имя
details	данный ключ содержит большое количество полей, в т.ч. из контентов-расширений (Extensions); ниже описаны лишь основные поля		
details.title.value	название страницы или виджета	AbstractItem	Title / Заголовок
details.ispage.value	указывает, что статья содержит данные о странице, а не виджете	AbstractItem	IsPage

details.titleformat.value	формат названия страницы или виджета	AbstractItem	TitleFormat / Шаблон заголовка
details.isinsitemap.value	указывает, что статья содержится в структуре сайта	AbstractItem	IsInSiteMap / Показывать в структуре сайта
details.isvisible.value	видимость элемента	AbstractItem	IsVisible / Отображать в навигации

4.5. Получение дочерних виджетов для страницы или виджета, сгруппированных по зонам

Описание: метод предназначен для получения виджетов, сгруппированных по зонам.

Формат запроса:

```
GET /Site/widgets/{abstractItemId}
```

Входные данные:

Название	Тип данных	Описание	Дополнительная информация
abstractItemId	integer($int32)	ID страницы или виджета	Required
t	object	Словарь значений таргетирования	
zones	array[string]	Список виджетных зон	Если не передавать, то поиск виджетов не будет производиться для рекурсивных и глобальных зон
fillDefinitionDetails	boolean	Заполнять дополнительные поля из Definition	По умолчанию: false

Ответ:

В ответе получаем:

- код состояния:
  - 200 (Success),
  - 400 (Bad Request),
  - 404 (Not Found);
- тип данных (Media type):
  - application/json;
- значение в json – пример указан ниже:


```
{
  "SiteFooterZone": [
    {
      "zone": "SiteFooterZone",
      "sortOrder": 100,
      "id": 741119,
      "alias": "footer",
      "nodeType": "html_widget",
      "details": {
        "html": {
```

```
          "value": "<footer class=\"footer\">\n<div class=\"footer__row\">\n<div
class=\"wrapper\">\n<div class=\"footer__group\">\n<div class=\"footer__col\">\n<div
class=\"footer__box\">\n<p><a class=\"footer__menu-link\"
href=\"/about/about_company\">О компании</a></p>\n<p><a class=\"footer__menu-link\"
href=\"/about/compliance_and_business_ethics\">Комплаенс</a></p></div></div>\n<div
class=\"footer__col\">\n<div class=\"footer__box\">\n<p><a class=\"footer__menu-
link\" href=\"/news_and_events/\">Новости </a></p>\n<p><a class=\"footer__menu-link\"
href=\"/sitemap/\">Карта сайта </a></p></div></div></div></div></div>\n<div
class=\"footer__row\">\n<div class=\"wrapper\">\n<div class=\"footer__group
footer__group--sb\">\n<div class=\"footer__col\">\n<div class=\"footer__box\">\n<div
class=\"footer__copyright\">\n<p><a href=\"https://quantumart.ru/\"
target=\"_blank\">Компания</a></p>\n<p>&copy; Квантум Арт</p></div></div></div>\n<div
class=\"footer__col\">\n<div class=\"footer__box\">\n<div
class=\"footer__contacts\">\n<p>Контакты:</p>\n<div class=\"footer__contacts-
group\"><a href=\"tel:78123094927\">+7 (812) 309-49-27</a><a
href=\"mailto:info@quantumart.ru\">info@quantumart.ru</a></div></div></div></div></di
v></div></div></footer>"
        },
        "title": {
          "value": "footer"
        },
        "isvisible": {
          "value": true
        },
        "ispage": {
          "value": false
        },
        "tags": {
          "value": "1"
        },
        "isinsitemap": {
          "value": true
        },
        "titleformat": {
          "value": 741017
        }
      }
    }
  ],
  "SiteHeaderZone": [
    {
      "zone": "SiteHeaderZone",
      "sortOrder": 100,
      "id": 741167,
      "alias": "top_menu",
      "nodeType": "top_menu_widget",
      "details": {
        "title": {
          "value": "top_menu"
        },
        "isvisible": {
          "value": true
        },
        "ispage": {
          "value": false
        },
        "isinsitemap": {
          "value": true
        },
        "titleformat": {
```

```
          "value": 741017
        }
      }
    }
  ],
  "SiteSearchHeaderZone": [
    {
      "zone": "SiteSearchHeaderZone",
      "sortOrder": 100,
      "id": 46,
      "alias": "search_bar",
      "nodeType": "search_bar_widget",
      "details": {
        "title": {
          "value": "Строка поиска"
        },
        "isvisible": {
          "value": true
        },
        "ispage": {
          "value": false
        },
        "isinsitemap": {
          "value": true
        },
        "titleformat": {
          "value": 741017
        }
      }
    }
  ]
}
```


Ниже представлено описание основных ключей типовых ответов формата JSON:

Ключ	Описание	Соответствующий контент	Поле контента
id	уникальный идентификатор страницы или виджета	AbstractItem	ID
alias	алиас	AbstractItem	
nodeType	тип виджета/страницы	ItemDefinition	Name / Имя
zone	виджетная зона	AbstractItem	ZoneName / Название зоны размещения
sortOrder	порядковый номер	AbstractItem	IndexOrder / Порядок
allowedUrlPatterns	поле принимает абсолютные и относительные ссылки; виджеты отображаются по указанным ссылкам, если на странице существует такая же зона, что указана в текущей статье	AbstractItem	AllowedUrlPatterns / Показывать на страницах

deniedUrlPatterns	поле принимает абсолютные и относительные ссылки; виджеты по указанным ссылкам не отображаются	AbstractItem	DeniedUrlPatterns / Скрывать на страницах
details	данный ключ содержит большое количество полей, в т.ч. из контентов-расширений (Extensions); ниже описаны лишь основные поля		
details.isvisible.value	видимость элемента	AbstractItem	IsVisible / Отображать в навигации
details.title.value	название страницы или виджета	AbstractItem	Title / Заголовок
details.ispage.value	указывает, что статья содержит данные о странице, а не виджете	AbstractItem	IsPage
details.titleformat.value	формат названия страницы или виджета	AbstractItem	TitleFormat / Шаблон заголовка
details.isinsitemap.value	указывает, что статья содержится в структуре сайта	AbstractItem	IsInSiteMap / Показывать в структуре сайта
details.categoryids.value	id элементов данной категории	ItemDefinition	CategoryName

4.6. Конфигурация

Ниже представлен конфигурационный файл WP.API и его описание.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information",
      "QA.*": "Debug"
    }
  },
  "QpSettings": {
    "ConnectionString": "Server=localhost;Database=demosite;User
Id=postgres;Password=pass",
    "DatabaseType": "Postgres",
    "IsStage": true,
    "SiteId": 52
  },
  "AllowedHosts": "*",
  "Cors": {
    "AllowedOrigins": ["*"]
  },
  "TargetingFilterSettings": {
    "UseRegionFilter": false
  }
}
```

Название	Тип	Описание
Logging	Объект	Раздел настроек логирования
LogLevel	Объект	Уровни логирования для различных модулей
Default	Строка	Минимальный уровень логирования по умолчанию
Microsoft	Строка	Минимальный уровень логирования для модулей Microsoft
Microsoft.Hosting.Lifetime	Строка	Уровень логирования для конкретной библиотеки
QA.*	Строка	Уровень логирования для библиотек с префиксом QA
QpSettings	Объект	Задает настройки QP
ConnectionString	Строка	Строка подключения к БД
DatabaseType	Строка	Тип БД – postgres или sqlserver. Производит выбор строки подключения: DatabaseQPPostgre или DatabaseQP
IsStage	Булевый	Режим работы сайта
SiteId	Число	Значение Id сайта
AllowedHosts	Строка	Хосты, по которым разрешён доступ к приложению
Cors	Объект	Настройки CORS
AllowedOrigins	Строка	Разрешённые источники
TargetingFilterSettings	Объект	Задает настройки фильтров таргетирования
UseRegionFilter	Булевый	Возможность использования фильтрации по регионам

5. Руководство по установке модуля Angular для QP8.WidgetPlatform (Linux)

5.1. Системные требования

Модуль **React** требует для своей работы установленный продукт **QP8.WidgetPlatform**.

Внимание: Для установки демо-сайта необходим предварительная установка модуля **QP.GraphQL.**

Внимание: При установке модуля **React** без использования Docker требуется предварительная установка Node.js 18 (или выше).

5.2. Установка QP8.WidgetPlatform.React (с использованием Docker)

1. Скачать архив widget-react-config.tar и распаковать его содержимое в `/etc/widget-react-config`.
2. Перейти в папку `/etc/widget-react-config`.
3. В файле `settings.json` задать:
   - внешний URL приложения WP.API в параметре `widgets.apiUrl` вместо `http://localhost:6200`, если этот этот внешний URL известен на текущий момент, и если планируется, что пользователь будет открывать приложение через браузер с другого компьютера;
   - внешний URL приложения QP.GraphQL в параметре `widgets.graphql.apiUrl` вместо `http://localhost:6300`, если этот этот внешний URL известен на текущий момент, и если планируется, что пользователь будет открывать приложение через браузер с другого компьютера.

Внимание: URL приложения QP.GraphQL должен заканчиваться на /graphql или /graphql/stage

4. Перейти в папку `/etc/widget-react-config/compose`.
5. В файле `.env` задать:
   - строку подключения к ранее развёрнутой базе демосайта в параметре `CONNECTION_STRING` (можно скопировать из конфигурационного файла QP);
   - внешний URL сервиса `react-module` в параметре `MODULE_CLIENT_URL` вместо `http://localhost:6500`, если этот этот внешний URL известен на текущий момент, и если планируется, что пользователь будет открывать приложение через браузер с другого компьютера;
6. Выполнить команду:

```
sudo docker-compose up -d
```

5.3. Установка QP8.WidgetPlatform.React (с использованием Kubernetes)

1. Скачать архив widget-react-config.tar и распаковать его содержимое в `/etc/widget-react-config`.
2. Перейти в папку `/etc/widget-react-config/k8s`.
3. В файле `widget.yml` можно настроить:
   - строку подключения к ранее развёрнутой базе демо-сайта в параметре `QpSettings__ConnectionString` (можно скопировать из конфигурационного файла QP);
   - внешний DNS приложения WP.API в параметре `widgets.apiUrl` внутри `settings.json`, настраиваемого в configmap, вместо `wp-demosite-rus-api.test`;
   - внешний DNS приложения QP.GraphQL в параметре `widgets.graphql.apiUrl` внутри `settings.json`, настраиваемого в configmap, вместо `graphql-demosite-rus-api.test`.

Внимание: URL приложения QP.GraphQL должен заканчиваться на /graphql или /graphql/stage

4. Развернуть сервисы командой:

```
kubectl apply –f widget.yml
```

5. При необходимости можно настроить ингрессы к сервисам `wp-demosite-rus-api`, `wp-demosite-rus-react-module` и `wp-demosite-rus-react-shell` в файле `ing.yaml`, задав свои DNS вместо `wp-demosite-rus-api.test`, `wp-demosite-rus-react-module.test` и `wp-demosite-rus-react-shell.test` соответственно и выполнив команду:

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

5.4. Установка QP8.WidgetPlatform.React (без использования Docker)

При установке из скомпилированного кода:

1. Скачать архив, содержащий бинарные файлы API виджетной платформы (WP.API) и демо-сайта React (WP.DemositeRus.React).
2. Распаковать полученный архив в домашний каталог пользователя, созданного в рамках установки QP8.CMS (по умолчанию - `/home/qp`). Должна получиться следующая структура из двух каталогов:

   - `WP.API;`
   - `WP.DemositeRus.React.Shell.`
   - `WP.DemositeRus.React.Module.`

5.4.1. Установка WP.API

При сборке из исходников:

1. Вытянуть исходники из репозитория на github.
2. В папке проекта `QA.WidgetPlatform.Api` (где должен быть файл `QA.WidgetPlatform.Api.csproj`) выполнить команду на публикацию:

```
dotnet publish "QA.WidgetPlatform.Api.csproj" -c Release -o
bin/release/publish/ -r linux-x64 --self-contained=false
```

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

При всех вариантах установки:

1. Перейти в папку `WP.API`. В файле `appsettings.json` в секции `QpSettings` изменить значение параметра `ConnectionString` на строку подключения к ранее развёрнутой базе демо-сайта в параметре (можно скопировать из конфигурационного файла QP).
2. В файле `NLog.config` найти `internalLogFile` и `<variable name="logDirectory" value=` и прописать там путь `/var/log/wp-api/`.

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

3. В файле `NLog.config` в секции `rules` во всех логгерах, в которых в параметре `writeTo` задано значение `console`, заменить его на `fileStructured`.
4. Создать на сервере папку `/var/log/wp-api/` и выдать пользователю QP права на владение папкой.
5. Создать файл `wp-api.service` в папке `/usr/lib/systemd/system/` и заполнить его следующим содержимым:

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

```
[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/WP.API/
ExecStart=/bin/dotnet QA.WidgetPlatform.Api.dll --urls http://*:6200

[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/`.

6. Прописать сервис в автозапуск, выполнив команду:

```
systemctl enable wp-api.service
```

7. Запустить сервис, выполнив команду:

```
systemctl start wp-api.service
```

5.4.2. Установка WP.DemoSiteRus.React.Module

При сборке из исходников:

1. Вытянуть исходники из репозитория на github.
2. В папке `demosite-rus-module` выполнить установку npm-пакетов командой `npm ci`.
3. Собрать модуль в подпапку `dist`, выполнив команду:

```
npm run build:server:prod
```

4. В папке `/home/qp` создать подпапку `WP.DemoSiteRus.React.Module` и скопировать туда всё содержимое каталога `dist`, в который осуществлялась публикация. Следует проверить, что у пользователя QP есть права на чтение и исполнение содержимого папки `WP.DemoSiteRus.React.Module`.

При всех вариантах установки:

1. Установить компонент `http-server` командой:

```
npm install --global http-server
```

2. Создать файл `wp-demosite-rus-react-module.service` в папке `/usr/lib/systemd/system/` и заполнить его следующим содержимым:

```
[Unit]
Description=QP DemositeRus.React.Module
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
```

```
RestartSec=5
User=qp
Environment=NODE_ENV=production
WorkingDirectory=/home/qp/WP.DemositeRus.React.Module/
ExecStart=http-server –d false –p 6500

[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/`.

3. Прописать сервис в автозапуск, выполнив команду:

```
systemctl enable wp-demosite-rus-react-module.service
```

4. Запустить сервис, выполнив команду:

```
systemctl start wp-demosite-rus-react-module.service
```

5.4.3. Установка WP.DemoSiteRus.React.Shell

1. Вытянуть исходники из репозитория на github.
2. В папке `demosite-rus-shell` выполнить установку npm-пакетов командой `npm ci`.
3. Собрать модуль в подпапку `dist`, выполнив команду:

```
npm run build:server:prod
```

4. В папке `/home/qp` создать подпапку `WP.DemoSiteRus.React.Shell` и скопировать туда всё содержимое каталога `dist`, в который осуществлялась публикация. Следует проверить, что у пользователя *QP* есть права на чтение и исполнение содержимого папки `WP.DemoSiteRus.React.Shell`.

При всех вариантах установки:

1. Создать файл `wp-demosite-rus-react-shell.service` в папке `/usr/lib/systemd/system/` и заполнить его следующим содержимым:

```
[Unit]
Description=QP DemositeRus.React.Shell
After=qp-graphql.service wp-api.service wp-demosite-rus-react-shell
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
Environment=NODE_ENV=production
WorkingDirectory=/home/qp/WP.DemositeRus.React.Shell/
ExecStart=/bin/node server/main.js

[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/`.

2. В файле `settings.json` задать:
   - внешний URL приложения WP.API в параметре `widgets.apiUrl` вместо `http://localhost:6200`, если этот этот внешний URL известен на текущий момент, и если планируется, что пользователь будет открывать приложение через браузер с другого компьютера;
   - внешний URL приложения QP.GraphQL в параметре `widgets.graphql.apiUrl` вместо `http://localhost:6300`, если этот этот внешний URL известен на текущий момент, и если планируется, что пользователь будет открывать приложение через браузер с другого компьютера.

Внимание: URL приложения QP.GraphQL должен заканчиваться на /graphql или /graphql/stage

3. Скопировать файл `settings.json` в папку `WP.DemoSiteRus.React.Shell/server`
4. Если внешний URL приложения WP.DemositeRus.React.Shell известен на текущий момент, и если планируется, что пользователь будет открывать приложение через браузер с другого компьютера, то можно выполнить команду в папке `WP.DemoSiteRus.React.Shell` (предварительно заменив `demosite-rus-react-shell.test` на реальный внешний URL):

```
sed -i -e 's/http:\/\/localhost:6500/http:\/\/demosite-rus-react-
shell.test/g' /static/client/*.*
```

5. Прописать сервис в автозапуск, выполнив команду:

```
systemctl enable wp-demosite-rus-react-shell.service
```

6. Запустить сервис, выполнив команду:

```
systemctl start wp-demosite-rus-react-shell.service
```

5.5. Настройка nginx

Для корректной работы WP.API и QP.GraphQL необходимо решить вопрос безопасности, связанный с CORS-ограничениями. Для их обхода следует формировать запросы на тот же домен, на котором располагается WP.DemoSiteRus.React. Осуществить это можно с помощью сервера nginx.

5.5.1. 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/widget-react-config/nginx/wp-nginx.conf`;
4. Задать свои DNS вместо `wp-demosite-rus-api.test`, `graphql-demosite-rus-api.test`, `wp-demosite-rus-react-shell.test` и `demosite-rus-react-module.test`;
5. Выполнить команду:

```
docker-compose up –d
```

5.5.2. 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/widget-react-config/nginx/wp-nginx.conf`;
3. Задать свои DNS вместо `wp-demosite-rus-api.test`, `graphql-demosite-rus-api.test`, `wp-demosite-rus-react-shell.test` и `demosite-rus-react-module.test`;
4. Проверить корректность конфигурации командой:

```
nginx -t
```

5. Если всё корректно, то прочитать обновленную конфигурацию nginx командой:

```
nginx -s reload
```