.

ООО «КВАНТУМ АРТ»

115184, Москва, Озерковский переулок, д. 12

тел. (495) 783-65-74

Программный продукт «QP8.Widgets»

Руководство разработчика

Назначение документа

Настоящий документ содержит руководство разработчика по программному продукту «QP8.Widgets». Цель документа – предоставить разработчику сведения, достаточные для разработки и сопровождения Информационных Систем, основанных на продукте.

Целевая аудитория

Документ предназначен для разработчиков, обладающих следующими компетенциями:

- знание технологий программной платформы .NET для разработки веб-приложений (ASP.NET Core);
- знание языков C#, SQL;
- знание технологий EF.Core;
- знание веб-технологий (HTTP, DNS).

История изменений

Версия	Дата	Автор	Описание

Содержание

1. Термины и определения

1.1. Общие термины и определения

Информационная Система (далее «Система») — автоматизированный программно-аппаратный комплекс, предназначенный для хранения, обработки и выдачи данных.

QP8.CMS (далее «QP») – программный продукт, обладающий широким спектром возможностей для разработки программной части Систем различной сложности.

Модульное приложение (также «Приложение», «Виджет») – обладающий ГПИ инструмент, содержащий набор функциональных возможностей для взаимодействия пользователей с какой-либо Системой (текущей или сторонней).

QP8.Widgets (также «Виджетная платформа») – продукт, расширяющий возможности QP. Позволяет через бекэнд 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»).

Development-окружение – среда, в которой осуществляется разработка и отладка Систем.

Stage-окружение – среда, максимально приближенная к production-окружению, в которой персоналом организации-разработчика осуществляется тестирование Систем.

Production-окружение – среда, используемая для размещения Систем, готовых к эксплуатации неограниченным кругом пользователей.

Графический пользовательский интерфейс (далее «ГПИ») – метод взаимодействия пользователя с Системой, при котором все ключевые способы управления Системой выполнены с использованием различных графических элементов.

Инструмент – часть Системы, обладающая определёнными функциональными возможностями.

Обработчик – программное средство, используемое на серверной части Системы для обработки запросов пользователей к веб-сайту Системы.

Active Directory (далее «AD») – служба каталогов для операционных систем Microsoft Windows Server. Базируется на протоколе LDAP.

Entity Framework (далее «EF») – технология для доступа к данным с использованием объектно-реляционного сопоставления (ORM, от англ. «Object-Relational Mapping»).

NuGet – средство для управления пакетами, используемое при разработке программных продуктов на платформе Microsoft.

1.2. Термины и определения для QP

DNS – доменное имя, используемое в Системе для работы с веб-сайтом.

Бекэнд – копия QP. Бекэнд обладает ГПИ для работы с содержимым БД Системы.

Виртуальный путь – URI до объекта.

Код клиента (Customer code) – уникальный параметр, определяющий БД Системы, с которой взаимодействует бекэнд QP.

Контент – раздел сайта.

Поле – атрибут контента. С использованием полей формируется структура данных для контента.

Пользовательское действие – дополнительная функциональная возможность для бекэнда, добавленная Разработчиком в Систему.

Реплейс – уникальное кодовое имя для статьи, с использованием которого можно вызвать содержимое этой статьи в других статьях текущего сайта.

Сайт – набор данных в бекэнде. Допускается создание нескольких сайтов. Содержимое каждого сайта определяется созданными в нём контентами.

Статья – элемент контента. Статья содержит данные, заданные в поля контента.

Физический путь – путь до объекта в файловой системе.

1.3. Определения для ролей пользователей в Системе

Пользователь – персона, осуществляющая взаимодействие с Системой.

Администратор – пользователь с правами на внесение любых изменений в Систему, которые можно выполнить с использованием бекэнда QP.

Контент-менеджер – пользователь с ограниченными правами на изменение содержимого Системы с использованием бекэнда QP.

Разработчик – пользователь с правами на внесение любых изменений в Систему (в том числе в содержимое скриптов, структуру БД).

1.4. Аббревиатуры

БД – База данных.

СУБД – система управления базами данных.

HTML (от англ. «HyperText Markup Language») – язык разметки документов.

JSON (от англ. «JavaScript Object Notation») – текстовый формат обмена данными.

XML (от англ. «eXtensible Markup Language») – расширяемый язык разметки документов.

API (от англ. «Application programming interface», интерфейс программирования приложений) – набор правил по использованию функциональных возможностей Системы, предоставляемый разработчикам для организации взаимодействия сторонних программных продуктов с Системой.

LINQ (от англ. «Language-Integrated Query») – компонент .NET Framework для работы с данными из БД, как с объектами. Запросы к СУБД формируются с использованием языков программирования .NET.

POCO (от англ. «Plain old CLR object») – подход к разработке программного обеспечения, предполагающий использование максимально простых классов для работы с объектами.

2. 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.

3. Общие сведения о продукте

3.1. Получение дистрибутива продукта

Дистрибутив содержит продукт «QP8.Widgets». Доступен по адресу http://downloads.quantumart.ru/Widgets. Скрипт инсталяции доступен только для ОС Microsoft Windows. Продукт QP8.Widgets состоит из:

- административного модуля управления структурой сайта QA.Engine.Administration,
- API для режима OnScreen QA.Engine.OnScreenAdmin,
- набора библиотек QA.DotNetCore.Engine.*, доступных для скачивания из nuget-репозитория,
- Live-версии сайта QA.Engine.DemoSite.Live,
- Stage-версии демо сайта QA.Engine.DemoSite.Stage.

3.2. Системные требования

3.2.1. Аппаратное обеспечение

	Минимальная конфигурация	**Рекомендуемая конфигурация**
Процессор	Intel Pentium IV 1.8 ГГц	Intel Xeon 2.4 ГГц x2
Память	2 ГБ	8 ГБ
Дисковое пространство	2 ГБ	100 ГБ и больше (в зависимости от применения)

3.2.2. Программное обеспечение

Операционная система

Окружение	**Описание**
Production	Требуется серверная ОС: • Microsoft Windows Server 2012 R2 или выше
Stage	Достаточно ОС для настольных ПК: • Microsoft Windows 8.1, • Microsoft Windows 10 ver. 1809 или выше Рекомендуется ОС из списка для production-окружения.
Development	

Поддерживаются 64-битные версии ОС.

Для рабочего места Разработчика достаточно любой из перечисленных ОС для настольных ПК.

СУБД

Полностью поддерживаемые СУБД:

- Microsoft SQL Server 2012 или выше
- PostgreSQL (или Postgres Pro) 11 или выше с установленным contrib

Веб-сервер

- IIS 8.5 и выше

Серверное ПО

- ASP.NET Core Runtime 3.1.8 (Hosting Bundle)
- QP8.CMS

Примечание: для установки на PostgreSQL / Postgres Pro необходимо использовать версию QP8 для ASP.NET Core.

ПО для Разработчика

- .NET Core SDK 3.1.401 и выше

- Поддерживаемые IDE
    - Microsoft Visual Studio 2019 (v.16.7)
    - Visual Studio Code
    - JetBrains Rider.

Программное обеспечение для работы с ГПИ продукта

Работа с ГПИ ведётся с использованием веб-браузера. Поддерживаемые веб-браузеры:

- Google Chrome (или веб-браузер на основе Chromium),
- Microsoft Internet Explorer (не ниже 11.0),
- Microsoft Edge,
- Mozilla Firefox.

Примечание: рекомендуется использовать актуальную версию веб-браузера.

4. Использование Виджетной платформы для ASP.NET Core

4.1. Архитектура платформы

В продукте используются понятия «Страница» и «Виджет».

Под страницей понимается объект, который требуется использовать для работы с данными на странице веб-сайта. Экземпляр страницы обладает URL.

Виджет предназначен для использования на различных страницах веб-сайта в качестве одной из её составляющих. Экземпляр виджета не обладает URL.

Примечание: в структуре данных продукта эти объекты имеют минимальные отличия.

Название	Описание
Контент «ItemDefinition»	Данные о типах элементов (страницы веб-сайта, виджеты). Контент содержит базовые поля, общие для всех типов виджетов и страниц веб-сайта.
Группа контентов «Pages and widgets»	Контенты-расширения для страниц и виджетов. Контент-расширение содержит поля, уникальные для типа страницы или виджета (отсутствующие в контенте «ItemDefinition»).
Контент «AbstractItem»	Данные обо всех экземплярах страниц и виджетов, созданных в Системе

Структура контентов

Поля контента «ItemDefinition»

Название	Тип	Описание
Title	String	Название страницы или виджета
Name	String	Идентификатор страницы или виджета
PreferredContentId	Numeric	Идентификатор контента-расширения
FriendlyDescription	String	Уникальное описание страницы или виджета. **Примечание:** в качестве значения допускаются только латиница и цифры.
CategoryName	String	Название категории виджета. Используется для группировки виджетов при выводе их в ГПИ виджетной платформы.
Description	String	Описание страницы или виджета
IconUrl	Image	Файл с пиктограммой для виджета. **Примечание:** используется в ГПИ виджетной платформы.
IsPage	Boolean	Указатель, что статья содержит данные о странице, а не виджете

Поля контента «AbstractItem»

Название	Тип	Описание
Title	String	Имя элемента
Name	String	URI для страницы. **Примечание:** используется только для страниц. **Примечание:** применяется в URL страницы.
Parent	One-to-Many Relation	Родительский элемент
IsVisible	Boolean	Флаг доступности для отображения
IsPage	Boolean	Флаг, является ли данный элемент страницей
Regions	Many-to-Many Relation	Регионы, для которых отображается данный элемент
ZoneName	String	Имя зоны, в которой находится виджет. **Примечание:** используется только для виджетов.
Description	String	Описание

Discriminator	One-to-Many Relation	Тип страницы или виджета
VersionOf	One-to-Many Relation	Оригинальная страница (страница, для которой создается версия)
Culture	One-to-Many Relation	Языковая культура
ExtensionId	Numeric	Идентификатор контента присоединённой статьи

4.2. Список библиотек и общее описание

Название	Версия	Описание
QA.DotNetCore.Caching	1.5.*	Интерфейс и реализация провайдера кэширования для .NET Core.
QA.DotNetCore.Engine.Abstractions	1.5.*	Интерфейсы и абстрактные классы для виджетной
QA.DotNetCore.Engine.AbTesting	1.5.*	Возможности A/B тестирования на виджетной платформе
QA.DotNetCore.Engine.AbTesting.Configuration	1.5.*	Простое конфигурирование A/B тестирования на виджетной платформе
QA.DotNetCore.Engine.CacheTags	1.5.*	Инфраструктура поддежрки кэш-тегов в виджетной платформе
QA.DotNetCore.Engine.OnScreen	1.5.*	Интеграция с админкой OnScreen.
QA.DotNetCore.Engine.Persistent.Dapper	1.5.*	Доступ к данным структуры сайта через Dapper
QA.DotNetCore.Engine.Persistent.Interfaces	1.5.*	Интерфейсы доступа к данным структуры сайта
QA.DotNetCore.Engine.QpData.	1.5.*	Реализация структуры сайта на основе QP
QA.DotNetCore.Engine.QpData.Configuration	1.5.*	Простое конфигурирование структуры сайта на основе QP
QA.DotNetCore.Engine.Reflection	1.5.*	Реализация ITypeFinder для .NET Core
QA.DotNetCore.Engine.Routing	1.5.*	Роутинг для виджетной платформы
QA.DotNetCore.Engine.Targeting	1.5.*	Таргетирование для виджетной платформы
QA.DotNetCore.Engine.Widgets	1.5.*	Серверный рендеринг для виджетов
QA.DotNetCore.Engine.Xml	1.5.*	Реализация структуры сайта на основе XML
QA.DotNetCore.Engine.Xml.Configuration	1.5.*	Простое конфигурирование структуры сайта на основе XML

4.3. Подключение структуры сайта из QP

Классический кейс использования виджетной платформы - использовать структуру сайта, заведенную в QP, для функционирования сайта, а именно:

- создание/управление страницами,
- роутинг,
- виджеты (с серверным рендерингом).

Структура контентов

Необходимо, чтобы в QP была создана база и в ней существовала определенная структура контентов.

Внимание: Платформе не важно, как называется тот или иной контент (т.е. поле *Content Name* из свойств контента не проверяется) – важно только ***NetName*** (поле ***Name (singular)*** из LINQ Mapping Parameters).

Аналогично с полями контентов: инвариантами являются LINQ ***Property Name***, а не ***Name***. Это сложившийся подход: в старой версии платформы для доступа к данным использовался linq2sql-слой доступа к этим данным, который опирался на ***NetName***.

Ниже приведено описание необходимой структуры в QP (приведены ***NetName*** контентов и необходимых полей):

- Контент **QPAbstractItem**. Его необходимые поля:
  - ***Name,***
  - ***Title,***
  - ***Parent,***
  - ***IsVisible,***
  - ***ZoneName,***
  - ***IndexOrder,***
  - ***ExtensionId,***
  - ***VersionOf,***
  - ***Discriminator.***

  Разработчики могут заводить и другие поля, если проект того требует, специальных ограничений нет.

- Контент **QPDiscriminator** (обычное название, как правило, **ItemDefinition**). Его необходимые поля:
  - ***Name,***
  - ***IsPage,***
  - ***TypeName,***
  - ***Title,***
  - ***Description,***
  - ***IconUrl,***
  - ***IconClass,***
  - ***IconIntent,***
  - ***PreferredContentId.***

- Контенты-расширения для **QPAbstractItem** для типов страниц и виджетов.

  Netname и обычные названия для этих контентов не регламентированы. На практике принято оставлять у этих контентов пустой ***netname*** и называть их как:

  `[название типа страницы/виджета]Extension`, например, `TextPageExtension.`

Единственное правило для этих контентов: они должны быть расширениями контента **QPAbstractItem**. Это достигается добавлением поля ***ItemId***, являющееся O2M-связью на **QPAbstractItem** с включенным признаком *Aggregated*. (подробнее о контентах-расширениях см. в документации QP).

Примечание: Назначение контентов структуры сайта.

QPAbstractItem – это контент, где заведены все страницы и виджеты, из которых состоит структура сайта. Структура имеет древовидный характер (за счёт поля ***Parent***).

QPDiscriminator – это контент с описанием типов страниц и виджетов.

Контенты-расширения для **AbstractItem** хранят в себе дополнительные поля, специфичные только для конкретных типов страниц или виджетов (например: если виджет новостей должен иметь свойство «Количество новостей для отображения», то это свойство должно быть заведено в контенте-расширении для этого виджета).

Правила контентного наполнения структуры сайта

Структура сайта – это дерево страниц и виджетов. Страница или виджет определяется свойством *IsPage* у **QPDiscriminator**.

Структура страниц задаёт роутинг на сайте:

- стартовая страница имеет адрес `/`,
- её дочерняя страница 1 уровня `/page1` (где page1 – это значение её поля ***Name***),
- её дочерняя страница 2 уровня `/page1/page2` и т.д.

Если виджет является дочерним элементом по отношению к странице, то он размещен на ней.

У виджета должно быть обязательно заполнено поле ***ZoneName***, которое означает место размещения на странице.

Какие именно зоны поддерживает страница определяет разработчиком - размещая в cshtml-шаблонах конструкции вида `@await Component.WidgetZone(Html, "Meta")`

Если у виджета не будет указана зона, или указанная зона не будет поддерживаться страницей, то он будет проигнорирован.

Примечание: зоны могут быть *обычными*, *глобальными* или *рекурсивными*.

Тип зоны определяется согласно конвенции наименования:

- если она начинается на префикс **Site** или **Global**, то она *глобальная*,
- если на **Recursive**, то *рекурсивная*,
- иначе *обычная*.

Если необходимо разместить один и тот же виджет на большом количестве страниц, то можно привязать его только раз к одной странице в такой зоне.

Виджеты в глобальных зонах отличаются от рекурсивных тем, что обязательно должны быть привязаны к стартовой странице (рекурсивные могут быть привязаны к любой), и распространяются не только на страницы структуры сайта, но и на весь сайт, включая «обычные» страницы, не обслуживаемые контроллерами структуры сайта.

Есть возможность ограничить распространение рекурсивного/глобального виджета по сайту: нужно добавить в **QPAbstractItem** поля ***AllowedUrlPatterns*** и ***DeniedUrlPatterns***, в которых можно задать белые или черные списки паттернов адресов страниц для отображения виджета.

Виджет может быть размещен не только на странице, но и в виджете, если тип родительского виджета поддерживает нужную зону.

Внимание: Страница не может быть размещена в виджете (т.е. родительским элементом для страницы не может быть виджет): это контентная ошибка, такие страницы не будут работать.

Структура сайта всегда должна иметь *корневую страницу* (т.е. страницу с определённым типом *«корневая страница»*). Она не несёт особой функциональности, и ей не соответствует никакой реальный URL-адрес страницы. Она нужна как маркер для виджетной платформы: её дискриминатор должен быть равен ***root_page*** (может быть переопределено при конфигурации свойством **RootPageDiscriminator**).

Второй уровень структуры сайта, после корневой страницы — это уровень *стартовых страниц*. Может быть больше одной стартовой страницы.

Например, если необходимо, чтобы одна база обслуживала несколько сайтов, допустим `somesite.ru` и `lk.somesite.ru`. У этих сайтов свои независимые структуры, но они могут использовать в них какие-то общие типы страниц и виджетов.

Для каждого такого сайта нужно заводить отдельную стартовую страницу. Для типа *«стартовая страница»* в таком случае нужно предусмотреть некое поле, исходя из которого можно ассоциировать запросы, идущие к `somesite.ru` и `lk.somesite.ru`, к своим стартовым страницам.

На практике обычно используется поле **DnsBindings**, в котором перечисляются шаблоны URL-хостов, которые должны относится к этой стартовой странице.

Как использовать

Инструкция для разработчика, как подключить функциональность структуры сайта от виджетной платформы:

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.QpData.Configuration**.
2. Сконфигурировать **IServiceCollection,** в `Startup.cs`, вызвав метод `services.AddSiteStructureEngine (options => …);` описание опций для этой конфигурации описано в отдельном пункте. См. Опции конфигурации
3. В `Startup.cs` использовать middleware, добавляющую структуру сайта в ***pipeline*** запроса `app.UseSiteStructure();`
4. Зарегистрировать `route/endpoint`, который будет обрабатывать запросы к страницам структуры сайта.
   Поддерживается 2 варианта:
   - В ASP.NET Core до 3 версии можно активировать с помощью `services.AddMvc(o => o.EnableEndpointRouting = false),` добавить `route:` в методе `app.UseMvc` зарегистрировать `routes.MapContentRoute("default", "{controller}/{action=Index}/{id?}");`
   - в ASP.NET Core 3 можно активировать с помощью `EndpointRouting`, который используется по умолчанию, и в методе `app.UseEndpoints` вызвать `endpoints.MapSiteStructureControllerRoute();`
5. Для всех типов страниц и виджетов создать классы моделей.

   Желательно заводить их все вместе в одной сборке для удобства регистрации типов в `services.AddSiteStructureEngine (options.TypeFinder.RegisterFromAssemblyContaining).`

   Свой класс нужно завести также и для корневой и стартовой страницы.

Классы страниц должны наследоваться от **AbstractPage**, классы виджетов от **AbstractWidget**. Стартовая страница должна также реализовывать интерфейс **IStartPage**.
В этих классах можно создавать поля, которые соответствуют полям, заведенным в соответствующих контентах-расширениях в QP.

Это следует делать с помощью метода `GetDetail`.

Например, `public string Text { get { return GetDetail("Text", String.Empty); } }`.

Если поле является many-to-many, то вместо **GetDetail** надо использовать **GetRelationIds**, что даст список ID объектов, соответствующих этому полю.

6. Для всех типов страниц создать контроллеры.

   Правила заведения контроллеров зависят от конвенции того, как классам страниц соответствуют контроллеры.

   Конвенция по умолчанию – это конвенция **Name**, она предполагает, что контроллер должен называться также как класс страницы (`TextPage -> TextPageController`).
   Конвенция **Attribute** предполагает, что контроллер должен быть помечен атрибутом `SiteStructureControllerAttribute`, в котором должен быть указан тип страницы. Конвенцию можно настраивать при конфигурировании с помощью опции **ControllerMapperConvention**.

   Для возможности удобно получать **AddSiteStructureEngine** из контроллера текущую и стартовую страницы, его следует унаследовать от **ContentControllerBase<T>**, где **T** – это класс типа страницы.

7. Для всех типов виджетов создать viewcomponent.

   Правила заведения viewcomponent зависят от конвенции того, как классам виджетов соответствуют viewcomponent.
   Конвенция по умолчанию – это конвенция **Name**, она предполагает, что viewcomponent должен называться также как класс виджета (`TextWidget -> TextWidgetViewComponent`).

   Конвенция **Attribute** предполагает, что viewcomponent должен быть помечен атрибутом `SiteStructureComponentAttribute`, в котором должен быть указан тип виджета.

   Конвенцию можно настраивать при конфигурировании **AddSiteStructureEngine** с помощью опции **ControllerMapperConvention**. Для возможности удобно получать из viewcomponent текущий виджет, его следует унаследовать от, **WidgetComponentBase<T>**где **T** – это класс типа виджета.

8. Создать нужные view для методов контроллеров и для viewcomponent.
9. Для подключения таргетирования см. «Подключение таргетирования к сайту»
10. Для подключения кэштегов см. «Подключение функциональности кэштегов»
11. Для подключения onscreen см. «Подключение и использование режима Onscreen»
12. Для поиска элементов по структуре сайта, нужно использовать **IItemFinder**, он уже добавлен в IoC-контейнер. В него нужно передать предикат для поиска и элемент, в чьём поддереве нужно выполнять поиск.

Опции конфигурации

При регистрации сервисов для структуры сайта в `Startup.cs` вызывается метод `services.AddSiteStructureEngine (options => …)` с параметром **Action<SiteStructureEngineOptions>**. Таким образом, можно задать и переопределить опции, которые есть в объекте **SiteStructureEngineOptions**.

Большую часть из них явно задавать не нужно, они уже имеют некоторые значения по умолчанию:

options.UseQpSettings

Обязательная	Настройки для взаимодействия с QP (строка подключения, **SiteId, live/stage** итд), в котором хранится структура сайта.

options.TypeFinder.RegisterFromAssemblyContaining

Обязательная	Регистрация сборки, которая содержит все классы страниц и виджетов, созданных разработчиком. Если они не будут зарегистрированы, то ни один узел дерева структуры сайта не будет инстанцирован. Пример: `options.TypeFinder.RegisterFromAssemblyContaining<RootPage, IAbstractItem>()` (т.е. необходимо зарегистрировать все типы, реализующие **IAbstractItem**, находящиеся в той же сборке, где находится класс **RootPage**.)

options.UrlHeadPatterns

Необязательная	Значение по умолчанию - `new List<HeadUrlMatchingPattern> { new HeadUrlMatchingPattern { Pattern = "/"} }` (т.е. пустой паттерн, ни в хосте, ни в первых сегментах пути нет никаких таргетирующих токенов).

Примечание: Пояснение к понятиям «голова»/«хвост» URL

URL любой страницы (без querystring и fragment) можно представить как `{схема}://{хост}/{путь}.`

URLстраницы сайта, работающего на виджетной платформе можно представить немного по-другому, в виде: `{схема}://{голова}/{путь от стартовой страницы}/{хвост}.`

«Голова» - это хост сайта + опционально первые несколько сегментов пути; в этих сегментах и хосте могут быть заключены значения таргетирования (например, регион, культура итд).

«Хвост» - это последние сегменты пути; в этих сегментах могут быть **routevalues** (например, **action**, **id** итд).

Примеры:

`https://msk.somesite.ru/en-us/welcome/general-information/.`

`//msk.somesite.ru/en-us` – это голова, причём токены moskva и en-us могут быть использованы как значения таргетирования.

Путь от стартовой страницы - `welcome/general-information`. Хвост пустой.

`https://msk.somesite.ru/customers/products/mobile/tariffs/details/ne-odin-doma/`
`//msk.somesite.ru` – это голова.

Путь от стартовой страницы - `customers/products/mobile/tariffs.`

Хвост - `details/ne-odin-doma.`

options.RegisterUrlHeadTokenPossibleValues

Необязательная	по умолчанию ни одного I**HeadTokenPossibleValuesProvider** не зарегистрировано. Используется в связке с **options.UrlHeadPatterns**. Если в **UrlHeadPatterns** заданы токены, то можно через **RegisterUrlHeadTokenPossibleValues** зарегистрировать один или несколько классов-реализаций **IHeadTokenPossibleValuesProvider**. К примеру, если `UrlHeadPatterns = new List<HeadUrlMatchingPattern> { new HeadUrlMatchingPattern { Pattern = "/{region}"} },` т.е. первым сегментом пути может идти токен со значением текущего региона. Когда платформа обрабатывает запрос `http://somesite.com/segment1/segment2/segment3`, для корректной работы ей нужно понимать: `segment1` – это допустимое значение для токена **region** или нет? Если допустимое, то «головой» URL будет считаться `//somesite.com/segment1`, иначе – просто `//somesite.com`.

options.DefaultUrlTailPattern

Необязательная	Имеет смысл ТОЛЬКО для **Endpoint routing** (дефолтный режим роутинга в ASP.NET Core с 3 версии). Значение по умолчанию - `new TailUrlMatchingPattern {Pattern = "{action=Index}/{id?}" }` Шаблон по умолчанию для всех контроллеров.

options.UrlTailPatternsByControllers

Необязательная	Имеет смысл ТОЛЬКО для **Endpoint routing** (дефолтный режим роутинга в ASP.NET Core с 3 версии). Значение по умолчанию – пустое, т.е. по умолчанию у всех контроллеров только один шаблон «хвоста» - тот, что задан в **options.DefaultUrlTailPattern**. Шаблоны «хвоста» URL, которые будут использоваться только для определенных контроллеров структуры сайта (т.е., по сути, для определенных типов страниц). Некоторые типы страниц могут иметь свои специфичные потребности, например страница - новостей, где «хвост» может иметь вид: `/{year}/{month}/{day}/{article_alias}`.

options.ItemDefinitionConvention

Необязательная	Значение по умолчанию – ***ItemDefinitionConvention.Name*** Конвенция того, как **ItemDefinition** сопоставляется с .Net классами страниц и виджетов, которые заводит разработчик. **Примечание: ItemDefinition** – это контент с netname **QPDiscriminator** - в нем хранятся все типы страниц и виджетов. **ItemDefinitionConvention.Name** - предполагает совпадение поля **TypeName** у **ItemDefinition** и имени класса .Net **ItemDefinitionConvention.Attribute** – **[пока не реализовано]** предполагает наличие атрибута у класса .Net, в котором задаётся дискриминатор **ItemDefinition**.

options.ControllerMapperConvention

Необязательная	Значение по умолчанию – **ControllerMapperConvention.Name** Конвенция того, как .Net классам страниц соответствуют контроллеры. **ControllerMapperConvention.Name** - предполагает, что контроллер должен называться также как класс страницы (`TextPage -> TextPageController`) **ControllerMapperConvention.Attribute** - предполагает, что контроллер должен быть помечен атрибутом **SiteStructureControllerAttribute**, в котором должен быть указан тип страницы.

options.ComponentMapperConvention

Необязательная	Значение по умолчанию – **ComponentMapperConvention.Name** Конвенция того, как .Net классам виджетов соответствуют viewcomponent. **ComponentMapperConvention.Name** - предполагает, что компонент должен называться также как тип виджета (`TextWidget -> TextWidgetViewComponent`) **ComponentMapperConvention.Attribute** - предполагает, что компонент должен быть помечен атрибутом **SiteStructureComponentAttribute**, в котором должен быть указан тип виджета.

options.LoadAbstractItemFieldsToDetailsCollection

Необязательная	Значение по умолчанию – ***true***. Нужно ли загружать ли в нетипизированную коллекцию всех свойств страниц и виджетов поля из **AbstractItem**. Нетипизированная коллекция всех свойств страниц и виджетов – это коллекция к которой имеют доступ методы **GetDetail** и **GetRelationIds**. Если эта настройка ***false***, то она состоит только из полей контента-расширения, если ***true*** – то из полей и **QPAbstractItem**, и контента-расширения.

options.QpConnectionString

Не нужно использовать	если строка подключения уже задана через **options.UseQpSettings**

options.QpDatabaseType

Не нужно использовать	если уже задано через **options.UseQpSettings**. Возможные значения: ***postgres***/***mssql***. Значение по умолчанию – ***mssql***.

options.SiteId

Не нужно использовать	если уже задано через **options.UseQpSettings**. ***Id*** сайта в QP.

options.IsStage

Не нужно использовать	если уже задано через **options.UseQpSettings**. Режим работы сайта – ***Stage/Live***.

options.RootPageDiscriminator

Необязательная	Значение по умолчанию - ***root_page***. Дискриминатор корневого элемента структуры сайта (поле ***Name*** в контенте **QPDiscriminator** для записи, соответствующей корневой странице).

options.UploadUrlPlaceholder

Необязательная	Значение по умолчанию - `<%=upload_url%>`. Плейсхолдер, который заменяет собой путь до библиотеки сайта в QP.

options.SiteStructureCachePeriod

Необязательная	Значение по умолчанию - `new TimeSpan(0, 20, 0).` Длительность кэширования уже построенной структуры сайта.

options.QpSchemeCachePeriod

Необязательная	Значение по умолчанию - `new TimeSpan(0, 0, 30)`. Длительность кэширования схемы QP (таблица SITE, CONTENT_ATTRIBUTE и т.п.).

options.ItemDefinitionCachePeriod

Необязательная	Значение по умолчанию - `new TimeSpan(0, 20, 0)`. Длительность кэширования **ItemDefinition**.

4.4. Использование структуры сайта из QP как сервис

Иногда возникают задачи, когда необходимо получить структуру сайта, и при этом полноценный сайт с серверным рендерингом по ней строить не нужно. Пример такой задачи – это построение API для сайтов, работающих на *react*.

Структура контентов и правила наполнения базы QP такие же как для кейса «Подключение структуры сайта из QP».

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.QpData.Configuration**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод `services.AddSiteStructure(options => …)`; См. Опции конфигурации.
3. Для работы со структурой сайта заинжектить **IAbstractItemStorageProvider** в нужный класс, использовать у него метод `Get()`.

Опции конфигурации

Все опции для настройки по этому кейсу подробно описаны в кейсе «Подключение структуры сайта из QP». Список этих опций:

- **UseQpSettings**
- **QpConnectionString**
- **QpDatabaseType**
- **SiteId**
- **IsStage**
- **RootPageDiscriminator**
- **UploadUrlPlaceholder**
- **LoadAbstractItemFieldsToDetailsCollection**
- **SiteStructureCachePeriod**
- **QpSchemeCachePeriod**

4.5. Подключение таргетирования к сайту

На практике таргетирование используется совместно с кейсом «Подключение структуры сайта из QP». Некоторые концепции таргетирования (такие как, провайдеры таргетирования) не зависит от структуры сайта, и могут использоваться без неё.

Общая информация

Одна и та же страница сайта может выглядеть по-разному (или даже существовать, или нет) для разных пользователей сайта. Примеры:

1) У авторизованных пользователей на страницах сайта могут появляться новые элементы, появляться целые новые разделы на сайте.
2) У разных типов авторизованных пользователей (например, пользователи b2c и b2b) могут быть свои разделы.
3) Пользователи разных регионов могут видеть разные страницы по-разному (например, какие-то виджеты могут показываться только для пользователей определенных регионов; продукты в разных регионах могут отличаться).
4) Мультиязычность: пользователи разных культур (допустим на сайте можно выбирать язык) могут видеть разные версии сайта – как на разных языках, так и структурно разные (т.е., например, английская версия сайта может быть очень урезанной и с малым числом страниц)

Пользователи сайта делятся на аудитории по некоторым признакам, которые называются ***признаками таргетирования***. Признаки таргетирования могут влиять по-разному на серверный процесс обработки http-запроса от пользователя: например, признак «Выбранный язык» может влиять на выбор локализации файлов ресурсов resx, а признак «Регион пользователя» на набор продуктов, доступных в этом регионе.

Ниже описано как виджетная платформа работает с признаками таргетирования и как они влияют (при должной настройке) на её работу.

Реализуем специальные классы: ***провайдеры и фильтры таргетирования***, а также использовать **ITargetingContext** – ***контекст таргетирования***.

Провайдеры таргетирования

Их назначение – предоставлять словарь `([ключ таргетирования], [текущее значение таргетирования])`.
Должны реализовывать интерфейс **ITargetingProvider** с единственным методом `IDictionary<string, object> GetValues()`.
Для каждого сайта можно создавать более одного провайдера. Единственное ограничение, чтобы они возвращали словари с разными ключами, потому что middleware **UseTargeting** вызовет метод `GetValues` у каждого из них и объединит в единый словарь.

Контекст таргетирования

В любой класс, которому нужно знать текущие значения таргетирования, нужно инжектить **ITargetingContext** – контекст таргетирования. С помощью его метода `object GetTargetingValue(string key);` можно получать значение, вычисленное в соответствующем провайдере таргетирования.

Фильтры таргетирования

Если сайт построен на базе структуры сайта (кейс «Подключение структуры сайта из QP»), и необходимо, чтобы таргетирование воздействовало на него (различные версии сайта для различных значений таргетирования), следует использовать фильтры таргетирования.

Такой фильтр — это класс, реализующий интерфейс **ITargetingFilter** или наследующийся от **BaseTargetingFilter (namespace QA.DotNetCore.Engine.Abstractions.Targeting)**.

Основные методы этого интерфейса, которые требуется реализовать:

- `bool Match(IAbstractItem item),`
  где **IAbstractItem** – это элемент структуры сайта, т.е. страница или виджет.
  Смысл метода: «удовлетворяет ли элемент структуры сайта item текущим значениям признаков таргетирования?».

- `IEnumerable<T> Pipe<T>(IEnumerable<T> items) where T : IAbstractItem;`
  Смысл метода: отфильтровать множество смежных элементов уровня.
  Его стандартная реализация из **BaseTargetingFilter** базируется на методе `Match` (Pipe возвращает только те элементы, которые проходят проверку Match). Её можно переопределять, в случае если на проекте используется более сложная логика таргетирования, когда для того, чтобы отфильтровать элемент, требуется знание об его смежных элементах.

Несколько фильтров можно объединять в композицию фильтров (с помощью класса **UnitedFilter** или операндов **+/&**).

С помощью фильтра (и его метода **Pipe**) можно отфильтровать всю структуру сайта, как страницы, так и виджеты. Как правило для этого все страницы (и виджеты) должны иметь в QP некоторое специальное поле (поля), участвующие в логике таргетирования: например, many-to-many поле **«Регионы»** или **«Культура»**.

Рассмотрим пример, как может быть реализован простой фильтр по региону. Сначала сделаем специальный интерфейс-маркер для тех элементов структуры сайта, у которых есть many-to-many поле **«Регионы»** и реализуем во всех типах страниц и виджетов, в которых это нужно.

```
public interface IDemoAbstractItem : IAbstractItem
{
    IEnumerable<int> Regions { get; }
}
```

Код фильтра может выглядеть так

```
public class RegionTargetingFilter : BaseTargetingFilter
{
        public RegionTargetingFilter(ITargetingContext ctx)
        {
            Ctx = ctx;
        }

        public ITargetingContext Ctx { get; }

        public override bool Match(IAbstractItem item)
        {
            if (item is IDemoAbstractItem)
            {
                var siteItem = item as IDemoAbstractItem;

                var ctxRegion = Ctx.GetTargetingValue(TargetingKeys.Region);
                if (ctxRegion == null || !(ctxRegion is RegionDto))
                {
                    return false;//если из контекста запроса не смогли определить
регион
                }

                var itemRegionIds = siteItem.Regions;
                if (itemRegionIds == null || !itemRegionIds.Any())
                {
                    return true;//если у текущего элемента не указано ни одного
региона, то значит этот элемент предназначается для всех
                }

                return itemRegionIds.Contains((ctxRegion as RegionDto).Id);
            }
            return false;
        }
    }
```

Регистрация

Инструкция для разработчика, как подключить к сайту функциональность таргетирования от виджетной платформы.

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.Targeting**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод **services.AddTargeting()**
3. Написать и зарегистрировать в IoC-контейнере с нужным **lifetime** все нужные провайдеры и фильтры таргетирования (уже реализованный провайдер **UrlTokenTargetingProvider** регистрировать в IoC не нужно, он добавляется туда при вызове `services.AddTargeting())`
4. Использовать middleware, вычисляющую для запроса значения таргетирования **app. UseTargeting();** при вызове этого метода нужно указать все провайдеры таргетирования, которые платформа должна использовать.

   Например:

   ```
   providers => { providers.Register<UrlTokenTargetingProvider>(); }
   ```

5. Подключить фильтры структуры сайта (если они нужны), вызвав метод `app.UseSiteStructureFilters`, при вызове этого метода нужно указать все фильтры таргетирования, которые платформа должна использовать.

Уточнения для кейса, когда в URL сайта есть токены таргетирования

Для случая, когда на сайте используется структура сайта, и в «голове» URL есть токены таргетирования (см. настройку options.UrlHeadPatterns) уже существует готовый провайдер таргетирования **UrlTokenTargetingProvider** – он извлекает значения таргетирования из URL, согласно шаблонам **options.UrlHeadPatterns**.

В этом случае полезно также уметь генерировать для каждой страницы из структуры сайта URL с учётом текущих значений таргетирования (например, при генерации главного меню). Если просто вызвать метод `GetUrl()` у страницы из структуры сайта, то вернется URL без учёта таргетирования. Если же передать инстанс **ITargetingUrlTransformator** в `GetUrl()`, то вернется URL с учётом текущих значений таргетирования и шаблонов **options.UrlHeadPatterns**. **ITargetingUrlTransformator** уже добавлен в IoC-контейнер, его просто нужно заинжектить в нужное место.

Контентные и структурные версии

Сайт при таргетировании позволяет иметь несколько смежных страниц с одним алиасом, но разными значениями полей таргетирования на одном уровне.

Пример:

```
/ (id=1)
---> /tariffs (id=2 regions=moscow)
---> /tariffs (id=3 regions=spb)
---> /tariffs (id=4 regions=novosib)
```

Такие страницы еще называют версиями, в зависимости от значения таргетирования пользователю покажется только какая-то одна из них.

Можно выделить два вида версий: контентные и структурные. Концептуально они различаются тем, делят ли они между собой поддерево страниц и виджетов или же имеют каждый своё собственное поддерево.

Пример контентных версий (поддерево наследуется):

```
/ (id=1)
---> /tariffs (id=2 regions=moscow)
---> ---> /special (id=5)
---> ---> [crosssale widget] (id=6)
---> /tariffs (id=3 regions=spb versionOf=2)
---> /tariffs (id=4 regions=novosib versionOf=2)
```

В этом случае, даже несмотря на то, что у страниц 3 и 4 нет своих подстраниц, они наследуют подстраницу special и crosssale widget, и у пользователя в регионах spb и novosib они будут.

Пример структурных версий (поддерево не наследуется)

```
/ (id=1)
---> /tariffs (id=2 regions=moscow)
---> ---> /special (id=5)
---> ---> [crosssale widget] (id=6)
---> /tariffs (id=3 regions=spb)
---> ---> /special (id=7)
---> ---> [other widget] (id=8)
---> /tariffs (id=4 regions=novosib)
---> ---> /special-novosib (id=9)
```

В этом случае поддерево у всех версий своё, редактируется независимо, и может иметь различную структуру.

Какой из вариантов версий использовать – зависит от требований и ситуации.

Примечание: В админке структуры сайта (Manage Pages) контентные и структурные версии показываются по разному: контентные показываются в отдельной вкладке:

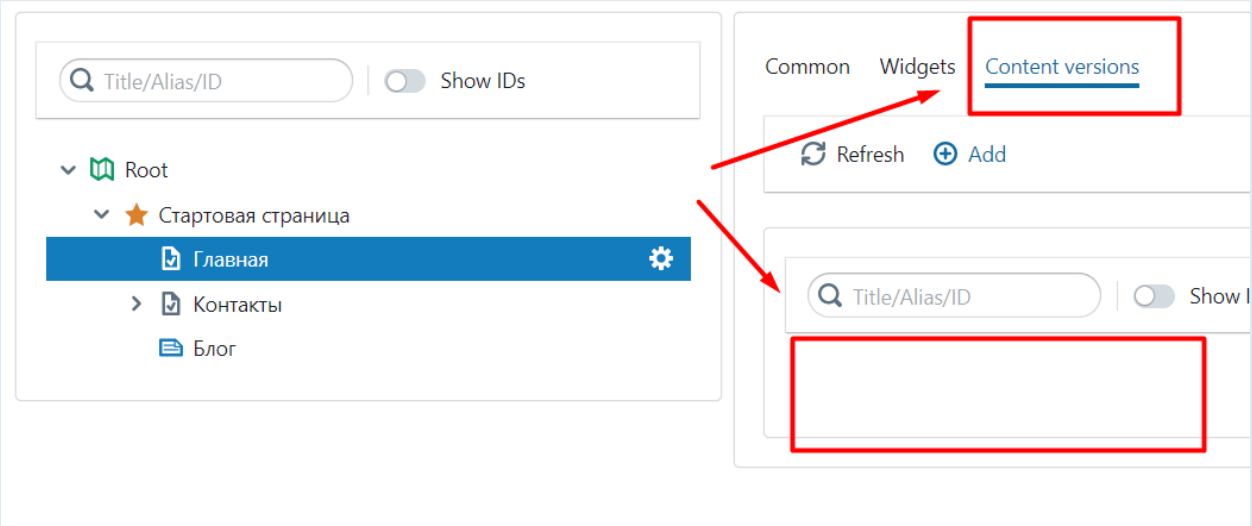

Рисунок 3.2.2-1. Manage pages – Content versions.

Структурные версии отображаются как обычные страницы, просто с тем же алиасом.

4.6. Подключение движка AB-тестирования к сайту

Кейс использования виджетной платформы, никак не связанный с виджетами: использовать структуру, заведенную в QP, для возможности проведения AB-тестирования на сайте.

Этот кейс может использоваться независимо от кейса «Подключение структуры сайта из QP».

Структура контентов

Для этого кейса необходимо, чтобы существовала база QP и в ней существовала определенная структура контентов.

Примечание: существует xml-скрипт, который можно «проиграть» в QP и который автоматически создаст всю необходимую структуру в базе.

Примечание: по аналогии с «Подключение структуры сайта из QP» платформе всё равно как называется тот или иной контент или поле – важно только так называемое ***NetName***. Более подробно см. в описании кейса «Подключение структуры сайта из QP».

Ниже приведено описание необходимой структуры в QP (приведены ***NetName*** контентов и необходимых полей)

1) Контент **AbTest**.
   В нем хранятся сами тесты и их базовые настройки. Тест – это некоторый эксперимент, который может состоять из одной или нескольких частей (так называемых *контейнеров*), каждая из которых по-своему меняет сайт. Также у теста должно быть хотя бы 2 варианта (иначе он не имеет смысла), может быть и больше – подробнее см. поле **Percentage**.

   Структура этого контента:
   a) **Title**.
      Наименование теста.
   b) **Percentage**.
      Вероятности выпадения того или иного варианта теста. Должно быть задано в специальном формате: положительные целые числа через запятую или точку с запятой.
      Количество чисел в этом поле определяет количество вариантов теста.
      Сумма чисел не должна быть равной 100 или любому другому числу, вероятности вариантов вычисляются исходя из пропорций указанных чисел.
      Например: «1,1» значит, что в тесте два равновероятных варианта (по 50% каждый), «1,1» означает то же что и «50,50», или скажем, «1234,1234». «1,2,3,4» значит, что в тесте 4 варианта: вариант 0 – 10%, вариант 1 – 20%, вариант 2 – 30%, вариант 3 – 40%.
   c) **StartDate**.
      Дата начала теста. С помощью него можно регулировать время, когда тест появится на сайте.
   d) **EndDate**.
      Дата окончания теста. С помощью него можно регулировать время, когда тест пропадёт с сайта.
   e) **Comment**.
      Комментарий, незначимое для функциональности поле.
   f) **Enabled**.
      Признак того, включен ли тест.

2) Контент **AbTestBaseContainer**.
   Это базовый контент для *контейнеров* тестов (т.е. составных частей тестов).
   Контейнер – это одно воздействие теста на сайт. Оно может заключаться или в добавлении js-скрипта на сайт, или в редиректе на какую-то страницу сайта. И js-скрипт, и адрес для редиректа задаются отдельно для каждого варианта теста.

Поэтому контейнер может быть 2 типов: *контейнер скриптов* или *контейнер клиентских редиректов*, их общие свойства находятся в **AbTestBaseContainer**, а различающиеся – в контентах расширениях **AbTestScriptContainer** и **AbTestClientRedirectContainer**.

Структура этого контента:

a) **Description**.
   Описание воздействия контейнера на сайт. Например, «Изменение цвета такой-то кнопки».
b) **AllowedUrlPatterns**.
   Ограничение для контейнера, на каких страницах сайта он должен быть. Шаблоны адресов страниц, размещенные на разных строках, возможно использование wildcard-символа * в конце шаблона.
c) **DeniedUrlPatterns**.
   Исключения из значений, заданных в **AllowedUrlPatterns**.
d) **Domain**.
   Домен сайта, на котором должен работать контейнер. Если не указывать – домен не будет проверяться.
e) **Precondition**.
   Условие срабатывания контейнера (таргетирование).
   Значение этого поля –Javascript-выражение. При его написании можно использовать переменную *ctx*. В ней есть все значения таргетирования, вычисленные на сервере и имеющиеся в контексте таргетирования (см. Контекст таргетирования). С помощью этого можно ограничивать выполнение контейнера теста только некоторой аудиторией сайта. Допустим, на сайте есть таргетирование userType, вычисляемое некоторым провайдером таргетирования (см. Провайдеры таргетирования). Тогда можно ограничить контейнер только b2c-пользователями, задав **Precondition**, например, ctx.userType == 'b2c'.
f) **Arguments**.
   Объект javascript, который может быть использован в Precondition как ctx.arguments.
g) **Type**.
   Тип контейнера(скрипты или редирект).

3) **AbTestScriptContainer** - контент-расширение для **AbTestBaseContainer** и контент **AbTestScript**. **AbTestScriptContainer** имеет только одно поле ScriptsInContainer, являющее many-to-many ссылкой на **AbTestScript**. Каждый привязанный к контейнеру скрипт – это вариант скрипта в определенной версии теста.

   Структура **AbTestScript**

   a) **Description**.
      Описание скрипта.
   b) **VersionNumber**.
      Zero-based номер версии скрипта в контейнере. Числовое значение 0,1,2..
   c) **ScriptText**.
   d) Текст скрипта.

4) **AbTestClientRedirectContainer** - контент-расширение для **AbTestBaseContainer** и контент **AbTestClientRedirect**. **AbTestClientRedirectContainer** имеет только одно поле ClientRedirectsInContainer, являющее many-to-many ссылкой на **AbTestClientRedirect**. Каждый привязанный к контейнеру **AbTestClientRedirect** – это вариант редиректа в определенной версии теста.

   Структура **AbTestClientRedirect**

   a) **VersionNumber**.
      Zero-based номер версии в контейнере. Числовое значение 0,1,2..

b) **RedirectUrl**.
URL редиректа

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.AbTesting.Configuration**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод `services.AddAbTestServices(options => …)`; описание options для этой конфигурации описано в отдельном пункте. См. Опции конфигурации
3. Подключить к главному шаблону сайта (как правило `Views/Shared/_Layout.cshtml`) специальный viewcomponent, в котором есть js-скрипт, который получает AB-тесты с сервера по специальному endpoint.
`@await Component.InvokeAsync("AbTest")`. Рекомендуется размещать его в нижней части <head>.

Правила работы AB-тестов

За счёт **AbTestViewComponent**, на все страницы сайта, на которых возможно AB-тестирование, размещается скрипт, который обращается на сервер по специальному endpoint, чтобы получить все скрипты, которые нужно выполнить на этой странице.

Сервер при обработке этого запроса в первую очередь определяет все контейнеры всех включенных тестов, которые подходят текущей странице.

Примечание: Тест считается включенным, когда у него `Enabled = true`, и текущая дата находится между **StartDate** и **EndDate**, если они заданы.

Примечание: Контейнер считается подходящим текущей странице, если URL этой страницы удовлетворяет полям **AllowedUrlPatterns, DeniedUrlPatterns, Domain.**

Далее сервер делает выбор варианта для каждого теста, задействованного на странице.

Количество вариантов теста определяется его полем **Percentage**. Вариант теста – zero-based значение, т.е. может принимать значения 0,1..N-1, где N – кол-во вариантов.

Выбор варианта для теста делается следующим образом:

1) Проверяем специальный query-параметр `test-{id теста}`. Он имеет самый большой приоритет. Ожидается целое неотрицательное значение. Например, test-12345=0.
2) Если выбор еще не сделан, смотрим передан ли вариант для этого теста в cookies. Имя cookie `abt-{id теста}`.
3) Если выбор еще не сделан, то выбираем один из вариантов с помощью класса **Random** и вероятности выпадения того или иного варианта теста, заданного в **Percentage**.

Примечание: обратите внимание, что выбор делается для теста, а не для контейнера, тем самым гарантируется, что для разных контейнеров, объединенных в один тест, будет сделан одинаковый выбор версии.

После этого сервер собирает ответ: каждый контейнер в нём представлен определенным js-скриптом. Для контейнера скриптов он имеет вид:

```
(function(ctx, window){{
    if(ctx && !({precondition})) return;
    window.abTestingContext['{AbTestChoiceResolver.CookieNamePrefix + test.Test.Id}'].targetedCids.push({container.Id});
    {script.ScriptText}
}})(ctx, window);
```

Для контейнера редиректов:

```
(function(ctx, window){{
    if(ctx && !({precondition})) return;
    window.location = '{redirect.RedirectUrl}';
}})(ctx, window);
```

В этом скрипте есть проверки, заданные через поле **Precondition**, а также сам скрипт/редирект выбранной версии (т.е. тот, чей **VersionNumber** равен выбранному варианту этого теста).

Дополнительно, в этот ответ помещается скрипт сохранения выбора варианта по каждому тесту в cookie. Это нужно для консистентности пользовательского опыта: чтобы при навигации по сайту у него не менялись постоянно варианты тестов, а оставались выбранные изначально при первом заходе на сайт.

Браузер, получив этот ответ, запускает все скрипты оттуда.

Опции конфигурации

При регистрации сервисов для работы AB-тестов в `Startup.cs` вызывается метод `services.AddAbTestServices(options => …)` с параметром **Action<AbTestOptions>**.

Таким образом здесь можно задать и переопределить опции, которые есть в объекте **AbTestOptions**.

Часть из них явно задавать не нужно, они уже имеют некоторые значения по умолчанию:

options.UseQpSettings

Обязательная	Настройки для взаимодействия с QP (**SiteId, live/stage, QpConnectionString, QpDatabaseType**), в которой хранятся тесты/контейнеры/скрипты.

options.SiteId

Не нужно использовать	если уже задано через **options.UseQpSettings**. ***Id*** сайта в QP.

options.IsStage

Не нужно использовать	если уже задано через **options.UseQpSettings**. Режим работы сайта – ***Stage/Live***.

options.QpConnectionString

Не нужно использовать	если строка подключения уже задана через **options.UseQpSettings.**

options.QpDatabaseType

Не нужно использовать	если уже задано через **options.UseQpSettings**. Возможные значения ***postgres*** и ***mssql***. Значение по умолчанию – ***mssql***.

options.TestsCachePeriod

Необязательная	Значение по умолчанию – `new TimeSpan(0, 1, 0).` Длительность кэширования описаний тестов.

options.ContainersCachePeriod

Необязательная	Значение по умолчанию – `new TimeSpan(0, 1, 0).` Длительность кэширования контейнеров тестов.

4.7. Подключение функциональности кэштегов

Общая информация

Для разработки высокопроизводительного сайта необходимо кэширования. В частности, оно используется и в библиотеках виджетной платформы, например, при построении структуры сайта из QP или получении АВ-тестов. Также кэширование должен использовать и разработчик сайта, например, для сохранения часто запрашиваемых и относительно редко меняющихся данных из базы.

Инвалидация кэша – процесс, посредством которого удаляются записи в кэше. Стандартная стратегия инвалидации – это инвалидация по таймауту, когда разработчик указывает сколько должна «жить» та или иная запись в кэше до момента её обновления.

Есть также дополнительная стратегия: *инвалидация по кэштегам*. Она предполагает, что разработчик при добавлении записи в кэш помечает её одним или несколькими тегами. *Тег* – это некоторая строка. Помимо того, должен существовать определенный отслеживающий процесс, который может сбрасывать все кэши, привязанные к определенному тегу, при наступлении каких-то условий. При грамотном использовании такого подхода можно:

- Существенно увеличить таймаут для кэширования (если знать, что данные не меняются, нет смысла делать запросы в базу для их обновления);
- Улучшить согласованность сохраненных в кэше данных (данные, привязанные к одному тегу, как правило связаны и сбрасываются одновременно);
- Уменьшить время между обновлением данных и появлением их на сайте.

Кэштрекеры

Виджетная платформа для инвалидации кэша предполагает использование специальных классов, *кэштрекеров*.

Кэштрекер – это класс, реализующий интерфейс **ICacheTagTracker**. Он используется для отслеживания изменений по тегам, его единственный метод:

```
IEnumerable<CacheTagModification> TrackChanges()
```

Этот метод возвращает множество всех тегов, за которые ответственен кэштрекер, с привязкой к времени их последнего обновления.

Виджетная платформа может использовать множество трекеров. В определенное время, когда наступает время проверки тегов, платформа запускает метод **TrackChanges** у всех зарегистрированных кэштрекеров, отслеживает те теги, которые обновились с момента предыдущей проверки тегов, и запускает метод **InvalidateByTags** у **ICacheProvider**.

Кэштеги и QP

В проектах, использующих QP, использование тегов обычно базируется на таблице **CONTENT_MODIFICATION**. В ней хранится информация о времени последнего изменения данных по каждому контенту, причём по ***live*** и ***stage*** отдельно (например, если изменится какая-то статья в статусе **None**, то **CONTENT_MODIFICATION** для этого контента обновится только для ***stage***, для ***live*** останется прежним). Это даёт возможность использовать контенты QP с суффиксами ***live/stage*** как кэштеги: можно отслеживать изменения в **CONTENT_MODIFICATION** и инвалидировать кэш по соответствующим тегам.

В виджетной платформе уже реализован кэштрекер, работающий с таблицей **CONTENT_MODIFICATION** – ***QpContentCacheTracker***, разработчику самостоятельно нужно реализовывать кэштрекеры только для собственных систем кэштегов.

Теги у ***QpContentCacheTracker*** имеют вид `{contentName}_{siteId}_{(isStage ? "Stage" : "Live")}`

При использовании ***QpContentCacheTracker*** имеет смысл использовать механизм кодогенерации для создания **enum** с тегами в указанном выше формате для всех существующих контентов.

Пример tt шаблона:

CacheTags.tt

Пример результата работы этого шаблона:

CacheTags.cs

Пример работы с кэштегами (класс **CacheTagUtilities** создаётся tt шаблоном):

```
CacheProvider.GetOrAdd($"NewsService.GetAllByOrganization_{organizationId}",
                CacheTagUtilities.Merge(CacheTags.NewsItem, CacheTags.NewsTag,
CacheTags.NewsTheme),
                TimeSpan.FromSeconds(30),
                () => GetAllByOrganization(organizationId));
```

Внимание: сам файл кодогенерации не является частью библиотек виджетной платформы.

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.CacheTags**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод `services.AddCacheTagServices(options => …);` описание опций для этой конфигурации описано в отдельном пункте. См. Опции конфигурации
3. Подключить `app.UseCacheTagsInvalidation(trackers => …).` При вызове этого метода нужно указать все кэштрекеры, которые платформа должна использовать.
4. Например, `trackers => { trackers.Register<QpContentCacheTracker>(); }`
5. [опционально] Добавить в проект **tt** шаблон для удобного использования кэштегов QP, зарегистрировать автосгенерированный класс `CacheTagUtilities в IoC: services.AddScoped<CacheTagUtilities>();`

Опции конфигурации

При регистрации сервисов для структуры сайта в `Startup.cs` вызывается метод `services.AddCacheTagServices(options => …)` с параметром **Action<CacheTagsRegistrationConfigurator>**. Таким образом мы можем здесь задать и переопределить опции, которые есть в объекте **CacheTagsRegistrationConfigurator**.

options.InvalidateByTimer(TimeSpan interval)

Значение по умолчанию отсутствует.	Процесс инвалидации кэштегов будет запускаться по таймеру в фоновом процессе с периодичностью равной ***interval***. Если используется эта настройка, то **options.UseTimer** и **options.TimerInterval** не нужно использовать.

options.UseTimer

Значение по умолчанию отсутствует.	Значение по умолчанию - ***false*** Признак того, что процесс инвалидации кэштегов будет запускаться по таймеру.

options.TimerInterval

Значение по умолчанию отсутствует.	Значение по умолчанию - `TimeSpan.FromSeconds(30)` Периодичность запуска процесса инвалидации кэштегов по таймеру. Не имеет смысла при `options.UseTimer = false`.

options.InvalidateByMiddleware(string excludeRequestPathRegex)

Значение по умолчанию отсутствует.	Процесс инвалидации кэштегов будет запускаться в middleware (её регистрировали с помощью **app.UseCacheTagsInvalidation**), т.е. при каждом запросе, но при условии, что **Request.Path** не матчится **excludeRequestPathRegex**. С помощью **excludeRequestPathRegex** можно настраивать, для каких запросов нужно, и для каких нет, запускать инвалидацию. Например, стоит отсекать запросы к статическим ресурсам, типа js, css итд. Если используется эта настройка, то **options.UseMiddleware** и **options.ExcludeRequestPathRegex** не нужно использовать.

options.UseMiddleware

Значение по умолчанию - ***false***	Признак того, что процесс инвалидации кэштегов будет запускаться в middleware (её регистрировали с помощью **app.UseCacheTagsInvalidation**).

options.ExcludeRequestPathRegex

Значение по умолчанию отсутствует.	**Regex** шаблон для **Request.Path**, определяющий для каких запросов НЕ нужно запускать процесс инвалидации. Не имеет смысла при `options.UseMiddleware = false`.

Пример использования

Можно настроить по-разному процесс инвалидации для режимов сайта ***Stage*** и ***Live***. Как правило, желательно настроить ***stage***-сайт так, чтобы он отображал максимально новую информацию, производительность для него не так важна, и он используется гораздо меньшим числом пользователей, чем ***Live***.

Поэтому довольно распространенный подход состоит в том, чтобы для ***stage***-режима была настроена инвалидация по middleware, а для ***live*** – по таймеру.

```
services.AddCacheTagServices(options =>
            {
                if (isStage)
                {
                    options.InvalidateByMiddleware(@"^.*\/(__webpack.*|.+\.[a-zA-Z0-
9]+)$");//отсекаем левые запросы для статики (для каждого сайта может настраиваться
индивидуально)
                }
                else
                {
                    options.InvalidateByTimer(TimeSpan.FromSeconds(30));
                }
            });
```

4.8. Подключение и использование режима onscreen

Если сайт использует хотя бы один из кейсов «Подключение структуры сайта из QP» или «Подключение движка АВ-тестирования к сайту», то к нему также можно подключить режим onscreen.

Onscreen – это особый режим редактирования сайта, когда пользователь может смотреть на stage-версию сайта и оттуда же управлять некоторыми его аспектами: виджетами (если подключена структура сайта из QP) и/или АВ-тестами (если подключен движок АВ-тестирования). Более подробно о функциональности этого режима см. соответствующую документацию (*её пока еще нет*).

Требования

Для возможности подключения Onscreen необходимо, чтобы был развернут специальный сервис **OnscreenAdmin**. Это консолидированный сервис, т.е. который подключается не к одной базе, а к целому семейству баз, прописанных в конфигурации customer-кодов QP. Нужно чтобы был развернут **OnscreenAdmin**, который подключен к той же QP, что и сайт.

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.OnScreen**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод **services.AddOnScreenIntegration**(mvcBuilder, options => …);
   mvcBuilder – это результат вызова `services.AddMvc()`; описание options для этой конфигурации описано в отдельном пункте. См. Опции конфгурации
3. Подключить middleware **app.UseOnScreenMode(customerCode)**, проверяющую возможность работы в режиме onscreen, добавляющая OnScreenContext в pipeline запроса.
4. Подключить к главному шаблону сайта (как правило `Views/Shared/_Layout.cshtml`) специальный viewcomponent, инициализирующий взаимодействие с OnscreenAdmin. `@await Component.InvokeAsync("OnScreen")`.

Как активируется этот режим

Чтобы активировать этот режим недостаточно открыть сайт в браузере. Этот режим доступен только пользователям QP, у которых есть доступ к редактированию этого сайта, для таких пользователей должен существовать с прописанным URL -адресом ***stage***-версии сайта.

При открытии **Custom Action** админка QP открывает фрейм с этим URL **Custom Action**, добавляя некоторые свои параметры в *query*-строку. По этим ***query***-параметрам производится авторизация пользователя для приложения **OnscreenAdmin**, именно из-под этого пользователя будут производиться все изменения в базе, которые можно сделать в этом режиме.

После первичной авторизации на сайте будет сохранена специальная cookie (её название определяется опцией **AuthCookieName**), которая позволит какое-то время открывать сайт в режиме Onscreen не во фрейме QP, а напрямую в браузере.

Опции конфигурации

При регистрации сервисов для интеграции Onscreen в `Startup.cs` вызывается метод `services.AddOnScreenIntegration(options => …)` с параметром **Action<OnScreenConfigurationOptions>.** Таким образом, здесь можно задать и переопределить опции, которые есть в объекте **OnScreenConfigurationOptions**.

Большую часть из них явно задавать не нужно, они уже имеют некоторые значения по умолчанию:

options.UseQpSettings

Обязательная	настройки для взаимодействия с QP (***SiteId, live/stage, CustomerCode***) с которой работает сайт.

options.AdminSiteBaseUrl

Обязательная	URL компонента **OnscreenAdmin**, который работает с тем же QP, что и сайт.

options.SiteId

Не нужно использовать	если уже задано через **options.UseQpSettings**. Id сайта в QP.

options.IsStage

Не нужно использовать	если уже задано через **options.UseQpSettings**. Режим работы сайта – ***Stage/Live***.

options.CustomerCode

Не нужно использовать	если уже задано через **options.UseQpSettings**. Customer-код сайта в QP.

options.AvailableFeatures

Обязательная	Значение по умолчанию - ***OnScreenFeatures.None*.** Фичи режима onscreen, которые нужно включить. Задаётся как набор флагов. Доступные значения флагов: • **Widgets**. фича OnScreen для работы с виджетами и зонами, имеет смысл только при использовании кейса «подключение структуры сайта из QP». • **AbTests**. фича OnScreen для работы с АВ-тестами, имеет смысл только при использовании кейса «подключение движка АВ-тестирования к сайту». Как правило, имеет смысл делать режим Onscreen активным только для ***stage***-версии сайта, как минимум потому, что у ***live***-версии могут быть более жесткие стратегии кэширования. В таких случаях можно задать эту опцию, например, так: `options.AvailableFeatures = qpSettings.IsStage ? OnScreenFeatures.Widgets | OnScreenFeatures.AbTests : OnScreenFeatures.None;`

options.AuthCookieName

Необязательная	Значение по умолчанию - ***"qa_onscreen_token"***. Имя cookie, в которой хранится информация об аутентификации onscreen.

options.AuthCookieLifetime

Необязательная	Значение по умолчанию - `TimeSpan.FromMinutes(60)`. Время жизни cookie, в которой хранится информация об аутентификации onscreen.

options.BackendSidQueryKey

Необязательная	значение по умолчанию - ***"backend_sid"***. Имя query-параметра с **backend_sid**, который создаёт QP для **custom action**, открытых через фрейм.

options.ApiApplicationNameInQp

Необязательная	значение по умолчанию - ***"onscreen-api"***. Под каким именем админка onscreen авторизуется в QP (должно совпадать с соответствующей настройкой у **OnscreenAdmin**).

options.OverrideAbTestStageModeCookieName

Необязательная	значение по умолчанию - ***"qa_onscreen_abtests_stage"***. Имя cookie, в которой хранится переопределенная для режима AB-тестов настройка **isStage**.

options.PageIdQueryParamName

Необязательная	значение по умолчанию - ***"pageId"***. Имя query-параметра с текущим **Id** страницы структуры сайта.

options.SkipWidgetTypes

Необязательная	значение по умолчанию - ***null***. Типы виджетов, которые надо игнорировать в режиме onscreen (не подсвечивать, обрамлять рамками итд). Массив из дискриминаторов (это поле ***Name*** у **QPDiscriminator**).

4.9. Подключение структуры сайта из XML

Альтернативный возможный кейс использования структуры сайта виджетной платформы: когда источник структуры сайта не QP, а XML-файл.

В реальных проектах этот кейс еще не был опробован. Причин для создания возможности получения страниц и виджетов не из QP, а из другого источника, в данном случае – XML-файл, несколько:

- Использование QP накладывает довольно большое ограничение на проект, теоретически может быть полезно использовать XML в качестве источника данных вместо базы, для каких-то подходящих кейсов.
- Продемонстрировать некий proof of concept, что концепция структуры сайта не завязана на использование QP, могут существовать и другие источники данных.

Структура XML

Пример того, как может выглядеть структура

```
<XmlRootPage>
	<XmlStartPage Alias="main">
		<XmlTextPage Alias="page1" Title="page1" Text="page1 content" />
		<XmlTextPage Alias="page2" Title="page2" Text="page2 content">
			<XmlTextPart	Alias="widget1"	Title="widget1"	Zone="Above"
Text="widget1 content"/>
		</XmlTextPage>
		<XmlTextPage Alias="page44" Title="page4" Text="page4 content" />
	</XmlStartPage>
</XmlRootPage>
```

Все правила, упоминающиеся в правилах контентного наполнения структуры сайта (см. кейс «Подключение структуры сайта из QP») актуальны и для наполнения XML.

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.Xml.Configuration**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод `services.AddSiteStructureEngineViaXml(options => …)`; описание опций для этой конфигурации описано в отдельном пункте. См. Опции конфигурации
3. В `Startup.cs` использовать middleware, добавляющую структуру сайта в pipeline запроса **app.UseSiteStructure**();
4. Зарегистрировать route, который будет обрабатывать запросы к страницам структуры сайта. Сейчас поддерживается только старый вариант роутинга: до 3 версии ASP.NET Core, в 3 версии его можно активировать с помощью

   `services.AddMvc(o => o.EnableEndpointRouting = false)`.

   В методе **app.UseMvc** зарегистрировать `routes.MapContentRoute("default", "{controller}/{action=Index}/{id?}");`
5. Для всех типов страниц и виджетов создать классы моделей. Желательно заводить их все вместе в одной сборке для удобства регистрации типов в `services.AddSiteStructureEngineViaXml` (речь о `options.TypeFinder.RegisterFromAssemblyContaining`).

   Свой класс нужно завести также и для корневой и стартовой страницы. Классы страниц должны наследоваться от **XmlAbstractPage**, классы виджетов от. Называться классы должны также как соответствующие xml-ноды. Стартовая страница должна также

реализовывать интерфейс IStartPage **XmlAbstractWidget**. В этих классах можно создавать поля, которые соответствуют атрибутам, заведенным в XML для такого типа страницы. Это следует делать с помощью метода **GetDetail**. Например, `public string Text { get { return GetDetail("Text", String.Empty); } }.`

6. Остальные действия, такие как создание контроллеров, viewcomponent и т.д. описаны в инструкции по использованию кейса «Подключение структуры сайта из QP»

Опции конфигурации

При регистрации сервисов для структуры сайта в `Startup.cs` вызывается метод `services.AddSiteStructureEngineViaXml(options => …)` с параметром **Action<XmlSiteStructureEngineOptions>**. Таким образом мы можем здесь задать и переопределить опции, которые есть в объекте **XmlSiteStructureEngineOptions**.

Большую часть из них явно задавать не нужно, они уже имеют некоторые значения по умолчанию:

options.TypeFinder.RegisterFromAssemblyContaining

Обязательная	Регистрация сборки, которая содержит все классы страниц и виджетов, созданных разработчиком. Если они не будут зарегистрированы, то ни один узел дерева структуры сайта не будет инстанцирован. Пример: `options.TypeFinder.RegisterFromAssemblyContaining<RootPage, IAbstractItem>()` – читай как зарегистрируй все типы, реализующие **IAbstractItem**, находящиеся в той же сборке, где находится класс **RootPage**.

options.Settings.FilePath

Обязательная	Путь до xml-файла со структурой сайта.

options.ControllerMapperConvention

Необязательная	Значение по умолчанию – ***ControllerMapperConvention.Name*** Конвенция того, как .NET классам страниц соответствуют контроллеры. Описание опции такое же как у аналогичной опции в кейсе «Подключение структуры сайта из QP».

options.ComponentMapperConvention

Необязательная	Значение по умолчанию – ***ComponentMapperConvention.Name*** Конвенция того, как .NET классам виджетов соответствуют viewcomponent. Описание опции такое же как у аналогичной опции в кейсе «Подключение структуры сайта из QP».