

Разработка приложения Marketplace

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

Разработка приложения Marketplace	4
Формирование пакетов приложения	4
Рекомендации по проектированию UX продукта	4
Интеграция с Creatio	11
Привязка данных к пакету	11
Лицензирование приложений Marketplace	16
Перенос приложения на тестовый сайт	19
Пример разработки приложения для Marketplace	21
1. Реализовать веб-сервис	22
2. Создать пользовательский пакет	24
3. Создать замещающую схему	26
4. Реализовать функциональность	28
Пример разработки приложения с пользовательским разделом для Marketplace	31
1. Создать пользовательский пакет	32
2. Установить значения системных настроек Текущий пакет и Префикс названия объекта	34
3. Создать раздел с помощью мастера	35
4. Реализовать необходимую функциональность раздела	36
Класс LicHelper	40
Методы	40
Защита исходного кода приложения Marketplace от плагиата	41
Защита C#-кода от плагиата	41
Защита JavaScript-кода от плагиата	50

Разработка приложения Marketplace



Формирование пакетов приложения

Любое приложение Marketplace представляет собой определенный набор пакетов. С их помощью выполняются все конфигурационные изменения.

Пакет Creatio — это совокупность конфигурационных элементов, которые реализуют определенный блок функциональности.

Принцип разделения функциональности приложения по пакетам реализован с помощью [зависимостей](#).

Активная пользовательская [разработка новой функциональности](#) предполагает создание новых пакетов и использование системы контроля версий (SVN). Основным содержимым пакетов являются [схемы](#).

Рекомендации по проектированию UX продукта

Одним из основных факторов, которые существенно снижают вероятность приобретения продукта после тестового использования, является негативный опыт пользователя при установке, настройке и начале работы с продуктом. Часто бывает, что качественный и полезный продукт, требующий сложной и трудоемкой настройки, воспринимается потенциальными клиентами негативно.

Ниже приведены рекомендации для разработчиков продуктов на платформе Creatio, выполнение которых позволит избежать типовых проблем и ошибок, связанных с UX установки и первого использования продукта.




Организация навигации в системе

Удобная, логично организованная навигация поможет новому пользователю легко сориентироваться в том, как найти приложение, как перейти к необходимой функциональности и воспользоваться ею. Проектируя навигацию, следует учитывать описанные ниже нюансы.


1. Отображение раздела в рабочем месте

Если функциональность приложения включает в себя новый раздел Creatio, то его необходимо отобразить в рабочем месте системы. Рабочим местом называется набор разделов системы, к которому имеет доступ и использует в своей работе определенная группа пользователей. При добавлении нового раздела следует учитывать, что рабочие места в Creatio формируются по функциональным и организационным ролям пользователей.

Кейс: "Добавление нового раздела [Чаты]"

 Неверно	Создать отдельное рабочее место и включить в него новый раздел [Чаты]
 Неверно	Не включать раздел в рабочие места
 Верно	Включить раздел [Чаты] в рабочее место [Продажи] и/или [Маркетинг]

2. Переход к настройкам из дизайнера системы

Все разделы и блоки, используемые для настройки системы, должны быть собраны в дизайнере системы. Переход в дизайнер выполняется из любого места системы по кнопке  в правом верхнем углу приложения. Блоки настроек дизайнера системы и кейсы их использования приведены в таблице.

Блоки настроек дизайнера системы

Блок настроек	Использование
Импорт и интеграции	Из блока выполняется переход на страницу настройки любой интеграции, а также импорта записей. Может использоваться для коннекторов
Настройка системы	Из блока выполняется переход к странице, настройки поведения системы. Например, отсюда можно перейти в мастер, который управляет поведением дополнения или специальной логики в вертикальном решении.

Из дизайнера системы можно реализовать переход к любому интерфейсу, где выполняется настройка, например:



- к пользовательской странице настроек;
- к странице раздела, посвященного настройкам;
- к странице справочника (если настройка состоит в заполнении справочника).

3. Переход к настройкам из меню действий раздела

Часто user experience (восприятие продукта пользователем) формируется так: пользователь переходит к разделу с функциональностью приложения, начинает ее изучать, и только после этого понимает, что дальнейшая работа требует настроек.

Стандартное решение, которое поможет пользователю сразу сориентироваться и выполнить нужные настройки — добавление необходимого действия в меню кнопки [Действия] в разделе. Важно понимать, что раздел (реестр записей) и страница конкретной записи — это разные страницы. Для каждой из них набор действий может отличаться, поэтому кнопка [Действия] настраивается отдельно для реестра записей и для страницы записи. То есть, действие, добавленное в меню действий раздела, не продублируется в меню действий на странице записи.

Кейс: “Реализация перехода к пользовательской странице настройки авторизации во внешней системе X (логин/пароль), с которой работает реализованный в приложении раздел [Интеграция]

 Неверно	Реализовать отдельную страницу настроек, доступную только по прямому URL-адресу
 Верно	В меню кнопки [<i>Действия</i>] раздела [<i>Интеграция</i>] добавить пункт "Настройки подключения к внешней системе X"

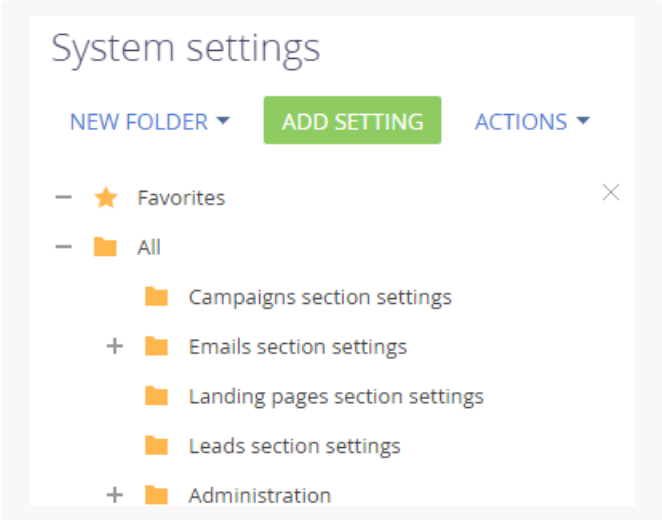
Настройки: простота и удобство нахождения

Многие приложения Marketplace требуют начальной настройки пользователем. Для этого в Creatio есть ряд пользовательских инструментов.

1. Настройки в разделе [*Системные настройки*]

Раздел [*Системные настройки*] используется для дополнительной настройки разделов системы. Например, здесь можно задать цвет для просроченных активностей, параметры отправки email-сообщений, параметры подключения к внешнему сервису и т. д.

Кейс: "Добавление новой системной настройки"

<p>Следует использовать лаконичные и понятные названия системных настроек. Название настройки должно отражать ее суть и указывать на функциональность, к которой она относится.</p>	<ul style="list-style-type: none">✗ Неверно (не лаконичное название): "Валюта, которая используется в системе по умолчанию".✓ Верно: "Базовая валюта".✗ Неверно (название не отражает суть) "Состояние заказа".✓ Верно "Состояние заказа по умолчанию".✗ Неверно (название не дает информации о связанной с настройкой функциональности) "Автоматический запуск процесса".✓ Верно "Автоматический запуск процесса управления инцидентами".
<p>Настройки необходимо группировать.</p>	<p>У пользователя очень мало шансов найти нужную настройку в общем списке, не зная ее точного названия. Группировка настроек — это простой и действенный способ упростить настройку приложения.</p> <div data-bbox="818 926 1476 1442"></div>
<p>Необходимо добавить описание</p>	<p>Описание поведения системы, которым управляет настройка, ответит на массу вопросов пользователя, которые у него обязательно возникнут.</p>

2. Настройки в разделе [Справочники]

Раздел [Справочники] используется для управления любыми данными (в т. ч. настройками и системными данными), которые можно представить в виде простого списка.

В таблице приведены примеры использования раздела [Справочники] для выполнения разных задач в системе. Рекомендуются использовать справочники для решения подобных кейсов.

Задача	Пример используемого справочника
Список данных для выбора пользователем	[<i>Состояния документа</i>] В справочнике содержится список всех состояний, в которые переводится документ в ходе рабочего процесса (например, "Актуальный", "Архивный", "В процессе подготовки").
Автозаполнение поля при интеграции	[<i>Каналы лида</i>] В справочнике содержится список всех типов ресурсов, по которым получен лид (например, "Социальные сети", "Поисковая реклама", "Email").
Настраиваемая бизнес-логика	[<i>Правила уведомления контакта по обращению</i>] Справочник содержит список правил, по которым происходит отправка уведомлений контакту о ходе работы над его обращением (например, "Отправляется сразу", "Отправляется с задержкой", "Не используется").
Системный список	[<i>Пользователи Webitel</i>]
Настройка отображения функциональности в разных разделах	[<i>Настройки диаграммы Гантта</i>]

3. Настройки в профиле пользователя

Если речь идет об индивидуальных настройках конкретного пользователя (например, индивидуальный логин/пароль для авторизации во внешней системе или настройки предпочтений по использованию определенной функциональности продукта), то целесообразно расположить их в профиле пользователя.

Ниже приведены примеры использования настроек в профиле пользователя:

1. Настройки подключения при интеграции:

- Id оператора телефонии / номер телефона, под которым пользователь работает с коннектором к телефонии.
- Настройки подключения пользователя к почтовому сервису.

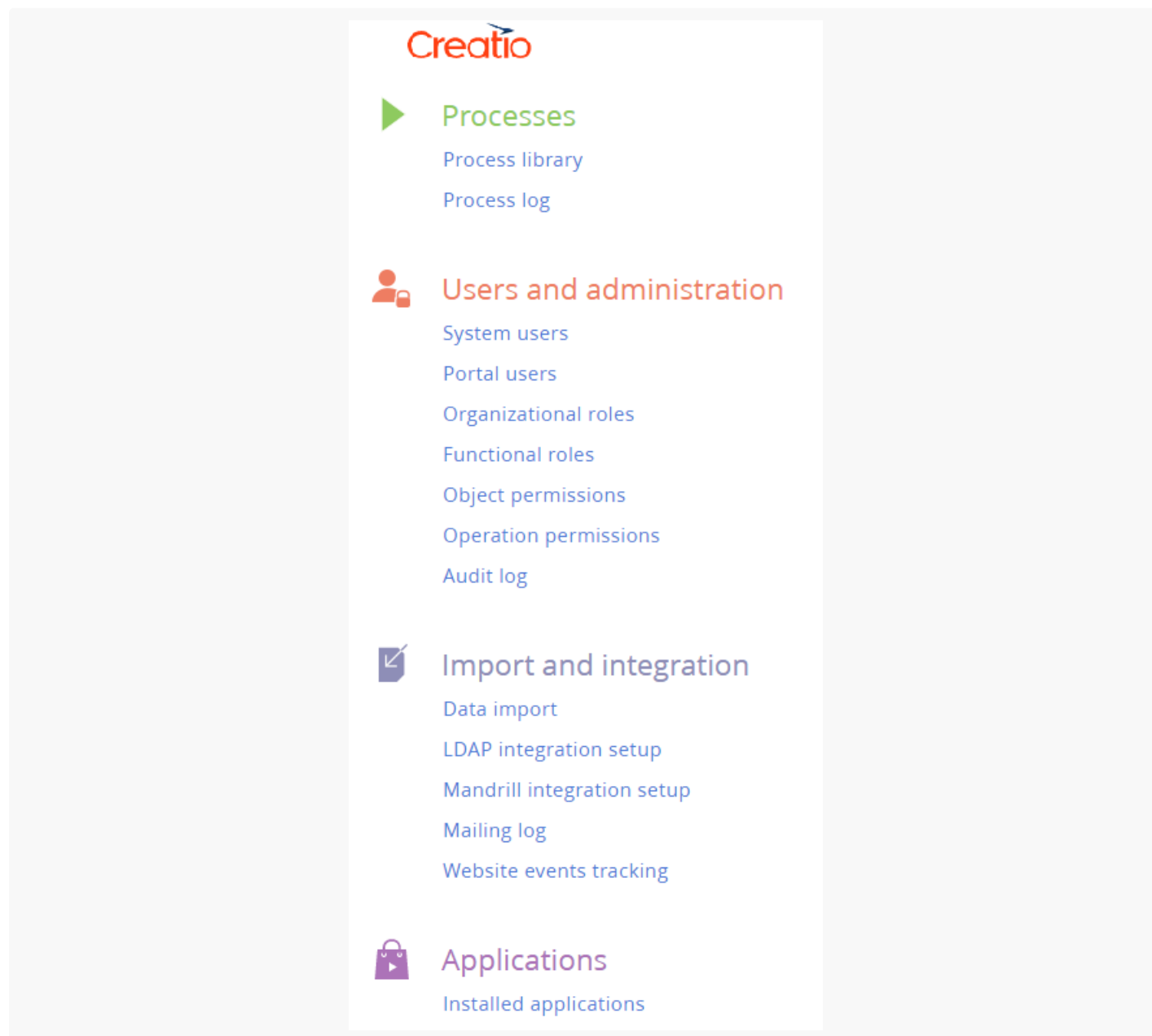
2. Настройки поведения системы для пользователя:

- Настройки уведомлений.
- Настройки языка по умолчанию, который используется данным пользователем в сканере визиток.

4. Отдельная страница настроек

Вынесение всех настроек на отдельную страницу — самый удобный для пользователя вариант

управления настройками приложения, хоть и более сложный в реализации.



Переход к странице настроек может выполняться из соответствующего раздела и/или из дизайнера системы.

Рекомендуется выносить все настройки приложения на отдельную страницу в любом из следующих кейсов:

- Если настройки необходимо выполнять в определенной последовательности.
- Если процесс настройки продукта требует и изменений в настройках и заполнения справочников.
- Если в ходе настройки, помимо заполнения настроек и справочников, нужно выполнять какие-то дополнительные действия.
- Если пользователю придется возвращаться к настройкам по ходу работы с приложением, а не выполнить их однократно после его установки.

Разработка с учетом базовой логики Creatio

Корректное заполнение полей при создании лидов

Создание лидов при интеграции Creatio с мессенджерами, чатами, посадочными страницами, социальными сетями — одна из типовых задач интеграции. При автоматическом создании лида важно сохранить корректную логику заполнения полей страницы лида, включая информацию о каналах привлечения потенциального клиента.

При разработке своего приложения следует позаботиться о том, чтобы пользователи Creatio соотносили название поля с его назначением и смогли пользоваться сквозной аналитикой по лидам. Пример названий полей лидов приведен в таблице.

Примеры названий полей лидов

Название	Использование
[Как зарегистрирован]	Способ регистрации лида: <ul style="list-style-type: none"> создан автоматически; добавлен вручную; входящий звонок/email; лендинг; обращение.
[Канал]	Тип ресурса, с которого получен лид, например: <ul style="list-style-type: none"> веб; социальные сети; реклама оффлайн; мероприятие; рекомендация/личный контакт.
[Источник]	Название конкретного ресурса, с которого получен лид, например, Twitter, Google и т. д.
[Сайт перехода]	Сайт, с которого пользователь перешел на посадочную страницу, в результате чего в системе был зарегистрирован лид. Поле недоступно для редактирования и заполняется автоматически при получении лида с посадочной страницы.

UI-минимум

Иконка раздела

Для каждого нового раздела должна быть добавлена иконка. Иконка раздела должна соответствовать таким требованиям:

- формат — *.png или *.svg.
- размер — не более 38 x 38px.
- стиль — плоское изображение белого цвета без мелких элементов/линий, фон без заливки.

Подобрать подходящие иконки можно в шаблонной [библиотеке иконок Marketplace](#). Также можно воспользоваться бесплатным сервисом для поиска и преобразования плоских иконок.

Интеграция с Creatio

Creatio предлагает широкие [возможности интеграции](#) с программными продуктами пользователей.

Для доступа в систему извне, сначала нужно произвести [аутентификацию](#). Начиная с версии 7.10 аутентификация выполняется с использованием механизма защиты от CSRF-атак.

Широкий спектр возможностей Creatio доступен для использования внешними системами посредством программного интерфейса (API), предоставляемого сервисом работы с данными [DataService](#). Если внешняя система для обмена данными использует протокол [OData](#), то возможен вариант интеграции с Creatio посредством этого протокола. Также можно использовать интеграцию с помощью HTML-элемента `<iframe>`.

Простые интеграции можно выполнять с помощью механизма [Web-To-Object](#).

Привязка данных к пакету

При создании раздела с помощью мастера к пакету автоматически привязываются данные, необходимые для регистрации и корректной работы раздела. Привязанные данные можно добавлять, изменять и удалять. Чтобы после установки приложения раздел отображался в определенном рабочем месте с некоторым демо-наполнением, необходимо к пользовательскому пакету привязать дополнительные данные.

Привязать данные к пакету, содержащему разработанную функциональность, можно в разделе [*Конфигурация*] ([*Configuration*]).

<div> <div>+ Add ▾</div> <div>Data ▾</div> <div>Filters ▾</div> <div>Search</div> <div>⚙</div> </div>							
Name ▾	Title	Status	Type	Object	Modified on	Package	
SysModule_SectionManager_8a3879e5f91c49cf81a9d7dd5b3a47c4			Data	SysModule	4/23/2018, 1:12:06 PM	sdkBookExample	⋮
SysModuleInWorkplace_SectionInWorkplaceManager_a69d5bf b41024326adeca22d02f6dde6			Data	SysModuleInWorkplace	4/23/2018, 1:12:07 PM	sdkBookExample	⋮
SysModuleEntity_SysModuleEntityManager_8654e87e4fd34e0a91d903ad427373d9			Data	SysModuleEntity	4/23/2018, 1:12:04 PM	sdkBookExample	⋮
SysModuleEdit_SysModuleEditManager_e3a3124a5cd346919574d2d5f5f750c4			Data	SysModuleEdit	4/23/2018, 1:12:05 PM	sdkBookExample	⋮
SysImage_f8d3f0121ed2433a996610d4b00cf09a			Data	SysImage	4/23/2018, 1:12:07 PM	sdkBookExample	⋮
SysDetail_DetailManager_b78e48663abc45cb916779059502af6b			Data	SysDetail	4/23/2018, 1:35:29 PM	sdkBookExample	⋮

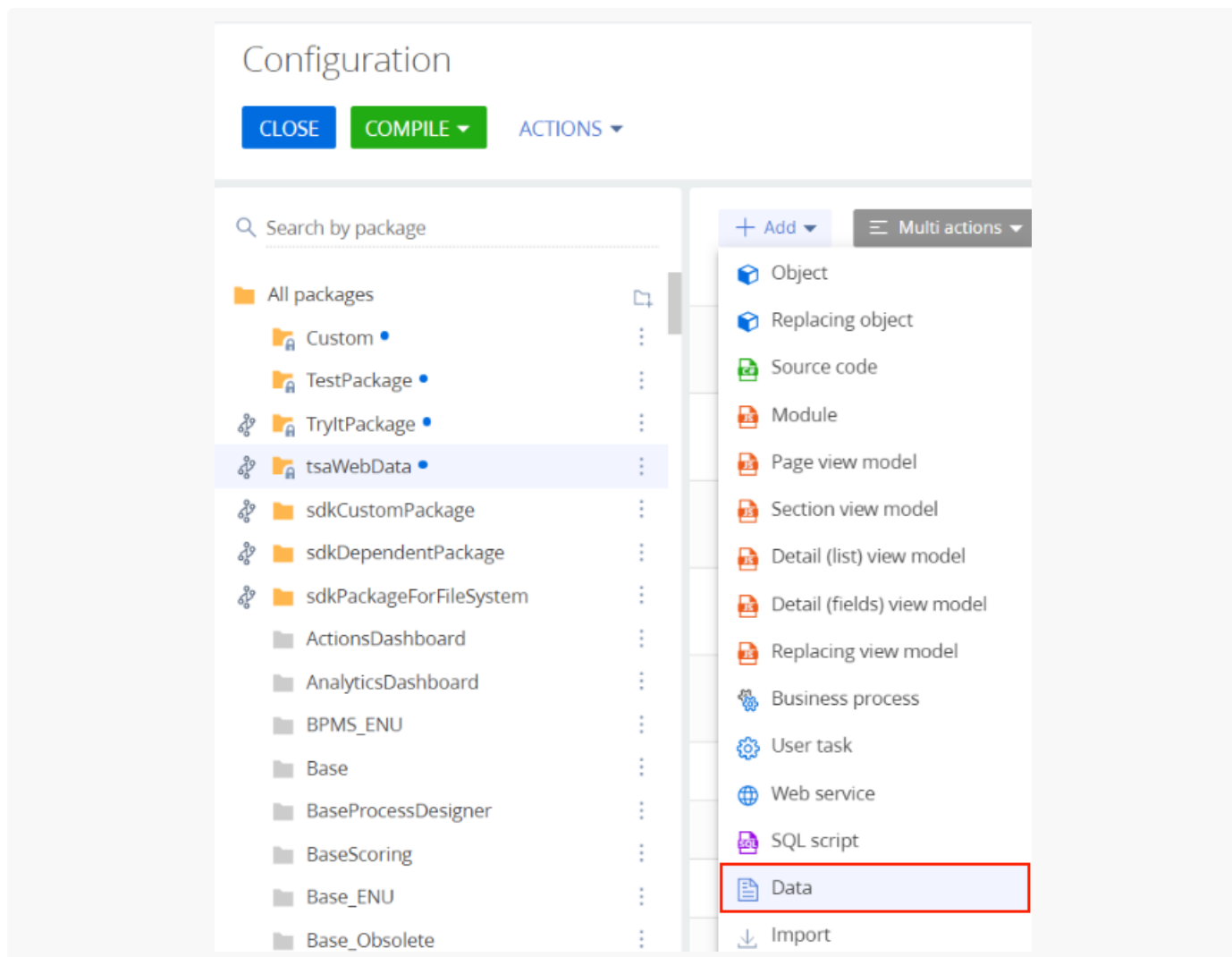
Приведенные ниже примеры привязки данных выполнены на основе раздела [*Web-данные*], создание которого описано в статье [Пример разработки приложения с пользовательским разделом для Marketplace](#).

Привязка раздела к рабочему месту

Для отображения раздела в рабочем месте [*Маркетинг*] нужно привязать данные схемы [*Раздел в рабочем месте*].

Для **привязки данных**:

1. Перейдите в [раздел \[Конфигурация \] \(\[Configuration \]\)](#) и выберите пользовательский пакет.
2. На панели инструментов рабочей области нажмите кнопку [*Добавить*] ([*Add*]) и выберите в списке вид конфигурационного элемента [*Данные*] ([*Data*]).



3. Заполните **свойства** страницы привязки данных:

- a. [*Название*] ([*Name*]) — "SysModuleInWorkplace".
- b. [*Объект*] ([*Object*]) — "Раздел в рабочем месте" ("SysModuleInWorkplace").
- c. [*Тип установки*] ([*Installation type*]) — "Установка" ("Installation").
- d. На вкладке [*Настройка колонок*] ([*Column setting*]) выберите **колонки**, по которым следует выполнить привязку:
 - [*Позиция*] ([*Position*]).
 - [*Раздел*] ([*Section*]).
 - [*Рабочее место*] ([*Workplace*]).

SAVE CANCEL Creatio

Name *
SysModuleInWorkplace

Object *
SysModuleInWorkplace

Installation type *
Installation

Package
TestPackage

Note
Installation type

COLUMNS SETTING BOUND DATA

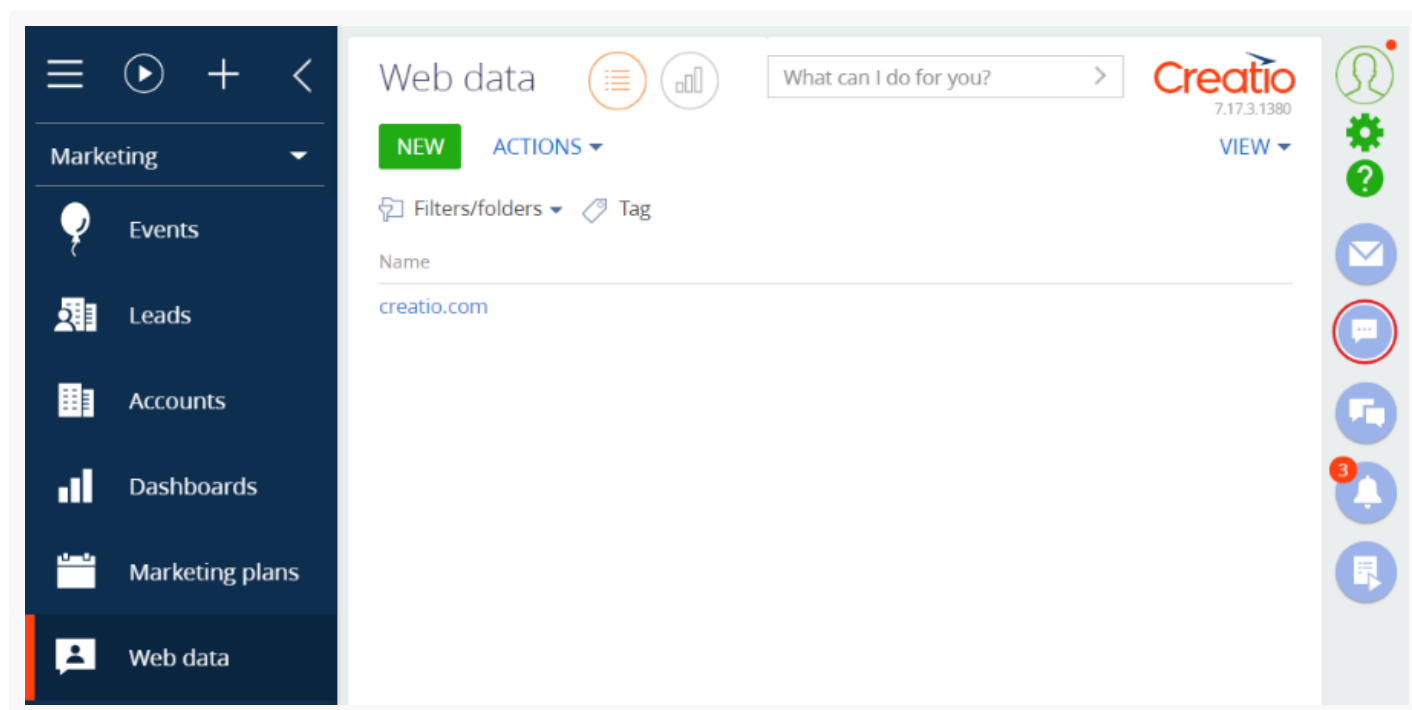
Caption	Name	Forced update	Key
<input type="checkbox"/> Caption	Id	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/> Position	Position	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Section	SysModule	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Workplace	SysWorkplace	<input type="checkbox"/>	<input type="checkbox"/>

+ Add

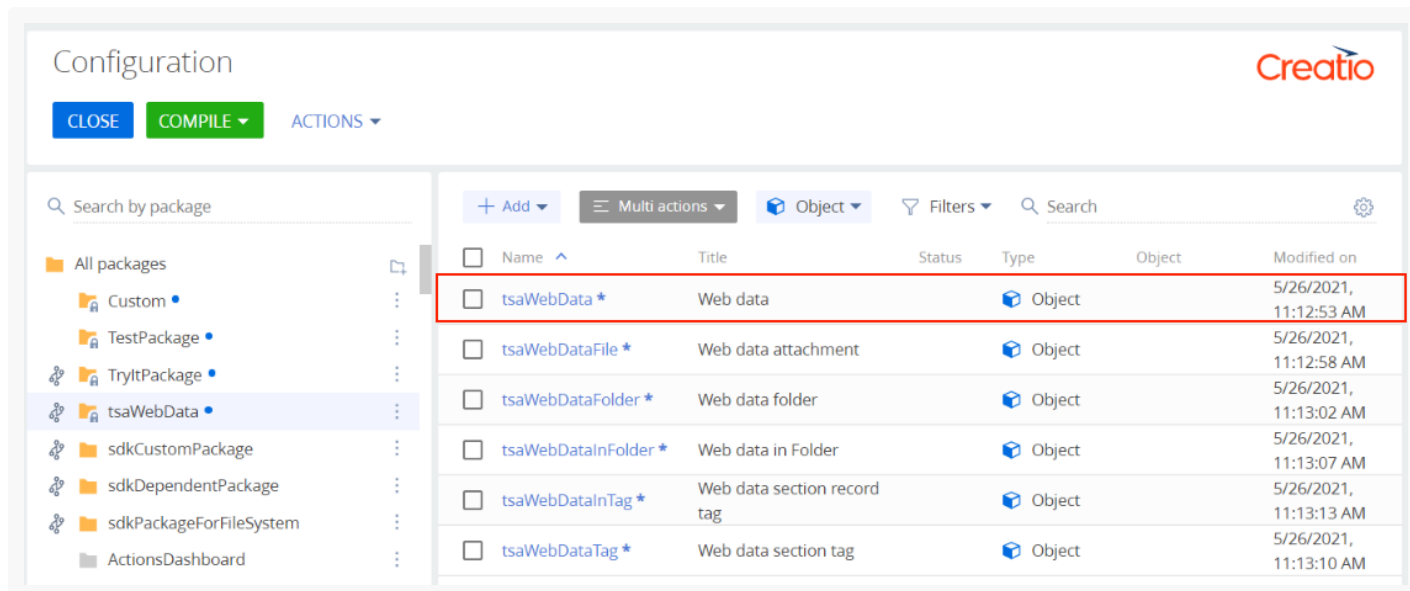
- h. На вкладке [Прикрепленные данные] ([Bound data]) выберите раздел [Web-данные].
- i. Сохраните данные.

Привязка данных наполнения раздела

Чтобы привязать данные раздела, необходимо сначала определить заголовок схемы основного объекта раздела.

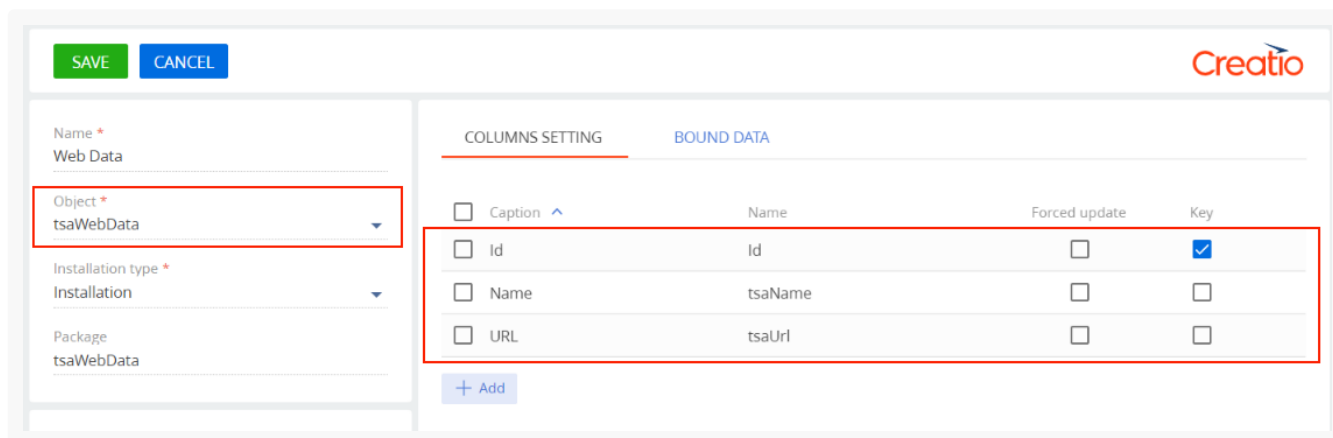


Эта схема создается автоматически мастером раздела и ее заголовок должен совпадать с заголовком раздела.



Для **привязки наполнения** раздела:

1. Перейдите в раздел [Конфигурация] ([Configuration]) и выберите пользовательский пакет.
2. На панели инструментов рабочей области нажмите кнопку [Добавить] ([Add]) и выберите в списке вид конфигурационного элемента [Данные] ([Data]).
3. Заполните **свойства** страницы привязки данных:
 - a. [Название] ([Name]) — "Web Data".
 - b. [Объект] ([Object]) — "tsaWebData".
 - c. [Тип установки] ([Installation type]) — "Установка" ("Installation").
 - d. На вкладке [Настройка колонок] ([Column setting]) выберите **колонок**, по которым следует выполнить привязку:
 - [Название] ([Name]).
 - [URL].



- g. На вкладке [Прикрепленные данные] ([Bound data]) выберите все данные раздела.
- h. Сохраните данные.

Лицензирование приложений Marketplace

Платные приложения Marketplace должны быть лицензированы для контроля их использования.

Основные типы лицензий:

- **Именные лицензии** — предоставляют доступ к приложению для конкретных пользователей. Эти лицензии привязываются к учетным записям и не могут быть использованы другими пользователями. Администратор системы может в любой момент [перераспределить именные лицензии](#).
- **Серверные лицензии** — не предусматривают необходимости лицензирования отдельных пользователей. Все пользователи Creatio, обладающие соответствующими правами, будут иметь доступ к функциональности, которая лицензируется.

Независимо от типа лицензии в нее могут входить:

- **Объекты лицензирования** — названия пользовательских объектов, добавленных в приложение Marketplace. В этот список могут входить любые пользовательские объекты, например, объект раздела, детали или справочника.
- **Операции лицензирования** — названия операций, добавленных в логику приложения для проверки наличия лицензии на использование определенной функциональности. Например, в один из базовых разделов системы было добавлено дополнительное действие, с которым нужно связать операцию лицензирования. При вызове этого действия в программной логике приложения сначала должна выполняться проверка наличия лицензии, а затем, в зависимости от результата, работа функциональности должна быть либо продолжена, либо прервана.

Количество оплаченных и предоставленных пользователям лицензий разработчик приложения Marketplace может узнать, обратившись в службу поддержки Marketplace (marketplace@terrasoft.ua).

Способы лицензирования

Возможные способы лицензирования зависят от типа лицензии и предмета лицензирования.

Возможные способы лицензирования

Тип лицензии	Предмет лицензирования	Бизнес-кейсы
Именная	Объекты	Приложение Marketplace представляет собой новый раздел в Creatio. Разработчик хочет получить плату за каждого пользователя, работающего с разделом.
Именная	Операции	<p>Приложение Marketplace представляет собой коннектор к внешнему сервису. Разработчик хочет получить плату за каждого пользователя, которому предоставляется доступ к сервису.</p> <p>Например, коннектор к системе телефонии, доступ к использованию которого оплачивается для отдельных пользователей.</p>
Серверная	Объекты	Приложение Marketplace представляет собой новый раздел в Creatio. Разработчик хочет получить фиксированную плату, независимо от того, сколько пользователей будет работать с разделом.
Серверная	Операции	<p>Приложение Marketplace представляет собой коннектор к внешнему сервису. Разработчик хочет получить фиксированную плату за коннектор, независимо от того, скольким пользователям будет предоставляться доступ к сервису.</p> <p>Например, коннектор к Web-чату, в ходе работы которого регистрируются лиды/обращения в Creatio. Клиент оплачивает лицензию независимо от количества пользователей.</p>
Именная	Объекты и/или операции	<p>Приложение Marketplace представляет собой раздел, в котором созданы записи, использующие коннекторы к разным внешним сервисам. Разработчик хочет получить плату за каждого пользователя, которому предоставляется доступ к разделу. Также разработчик хочет получить плату за каждый коннектор к сервису.</p> <p>Например, в рамках вертикального решения доработан раздел [Заявки]. В нем реализована интеграция с внешней системой, в которой регистрируются заявки. Клиент оплачивает доступ к функциональности раздела для каждого пользователя.</p>

Важно. Запрещается одновременное использование именных и серверных лицензий, лицензирующих одинаковые объекты и операции.

Срок действия лицензии

В лицензии указывается срок ее действия. Платформа Creatio следит контролирует сроки действия лицензий. Если срок действия истек, работа лицензируемого объекта или операции будет заблокирована.

В зависимости от срока действия, лицензии могут быть двух типов:

- **Годовая** — предназначена для приложений Marketplace, распространяемых с годовой подпиской. Срок действия лицензии — один год. После завершения срока действия лицензии клиент должен снова оплатить годовую подписку и получить новую лицензию.
- **Бессрочная** — предназначена для приложений Marketplace, которые необходимо оплатить один раз. Срок действия лицензии не ограничен.

Процесс выпуска лицензии для нового продукта

1. Во время создания приложения Marketplace разработчик должен решить:

- Что будет входить в лицензию — объекты лицензирования или операции (возможны оба варианта). Также он должен сформировать список лицензируемых объектов и/или операций.
- [Определить тип лицензии](#) — именная или серверная.
- Определить срок действия лицензии — лицензия годовая или бессрочная.

На заметку. Если в лицензию будут входить операции лицензирования, то в программном коде приложения необходимо реализовать проверку наличия лицензии. Для объектов лицензирования логика проверки уже реализована в базовой версии Creatio.

Важно. Имена лицензируемых объектов должны содержать префикс, указанный при [настройке профиля разработчика](#). Префикс разработчика — это уникальный префикс, который устанавливается в названиях пользовательских схем, пакетов, а также в названиях пользовательских колонок в объектах-наследниках системных объектов и позволяет определить функциональность, созданную разработчиком. Префикс должен содержать цифры или буквы латинского алфавита. Длина префикса — от трех до пятнадцати символов. После регистрации префикс не может быть изменен.

2. После успешной проверки приложения сотрудниками Marketplace разработчик должен предоставить список лицензируемых объектов и операций, а также информацию о типе и сроке действия лицензии команде поддержки Marketplace (marketplace@terrasoft.ua).
3. На основании предоставленного списка служба поддержки Marketplace генерирует лицензии для этого приложения.
4. В дальнейшем, после заказа и оплаты приложения Marketplace, пользователю будут установлены необходимые лицензии службой поддержки Marketplace.

Реализация проверки лицензий в продукте

Проверка объектов лицензирования

Если в лицензию приложения входят только объекты лицензирования, то никаких доработок в приложении Marketplace не требуется. Проверка наличия лицензии для объектов лицензирования выполняется базовыми средствами Creatio.

Проверка операций лицензирования

Если в лицензию приложения входит операция лицензирования, то в нужных местах исходного кода приложения необходимо предусмотреть проверку наличия лицензии на выполнение этой операции.

Важно. Проверку наличия лицензии на операцию можно выполнить только на серверной стороне приложения. Если приложение Marketplace работает только с клиентской частью Creatio, то необходимо:

- создать пользовательский конфигурационный сервис, в котором реализовать проверку наличия лицензии;
- добавить в приложение программную логику обращения к созданному сервису.

Важно. Проверка наличия лицензии на операцию непосредственно в пользовательских схемах является небезопасной, поскольку исходный код пользовательских схем конфигурационных пакетов доступен для чтения и замещения. Поэтому рекомендуется использовать объекты лицензирования.

Демонстрационный режим работы приложения

Если в приложении Creatio нет загруженных лицензий ни для одного из продуктов, то по умолчанию включается демонстрационный режим работы приложения. В демонстрационном режиме пользователю доступна вся функциональность Creatio, но в каждую таблицу базы данных можно добавить не более 1000 записей.

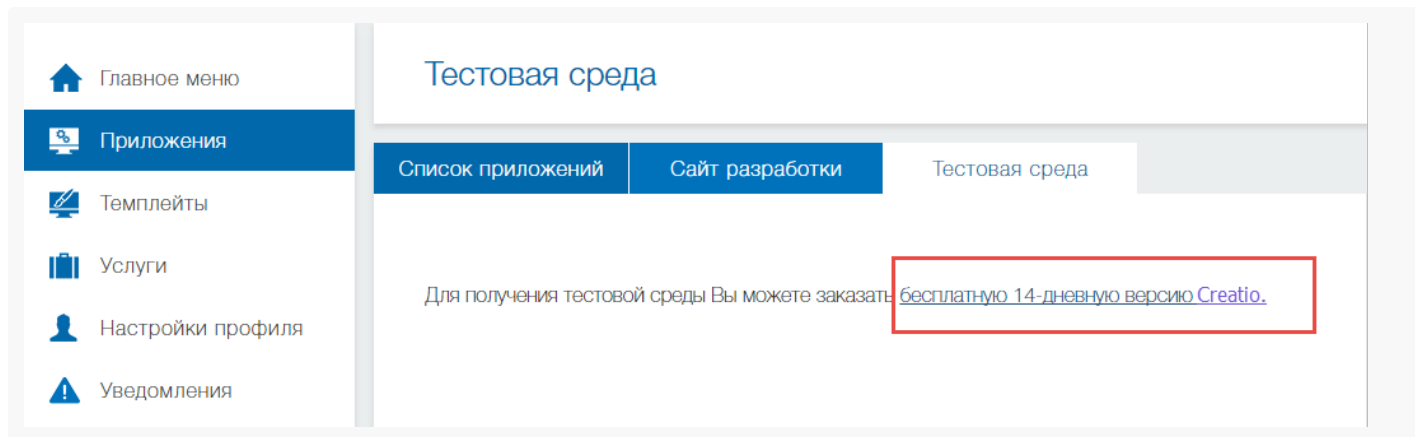
Перенос приложения на тестовый сайт

Разработка сложной функциональности требует правильной [организации процесса разработки](#).

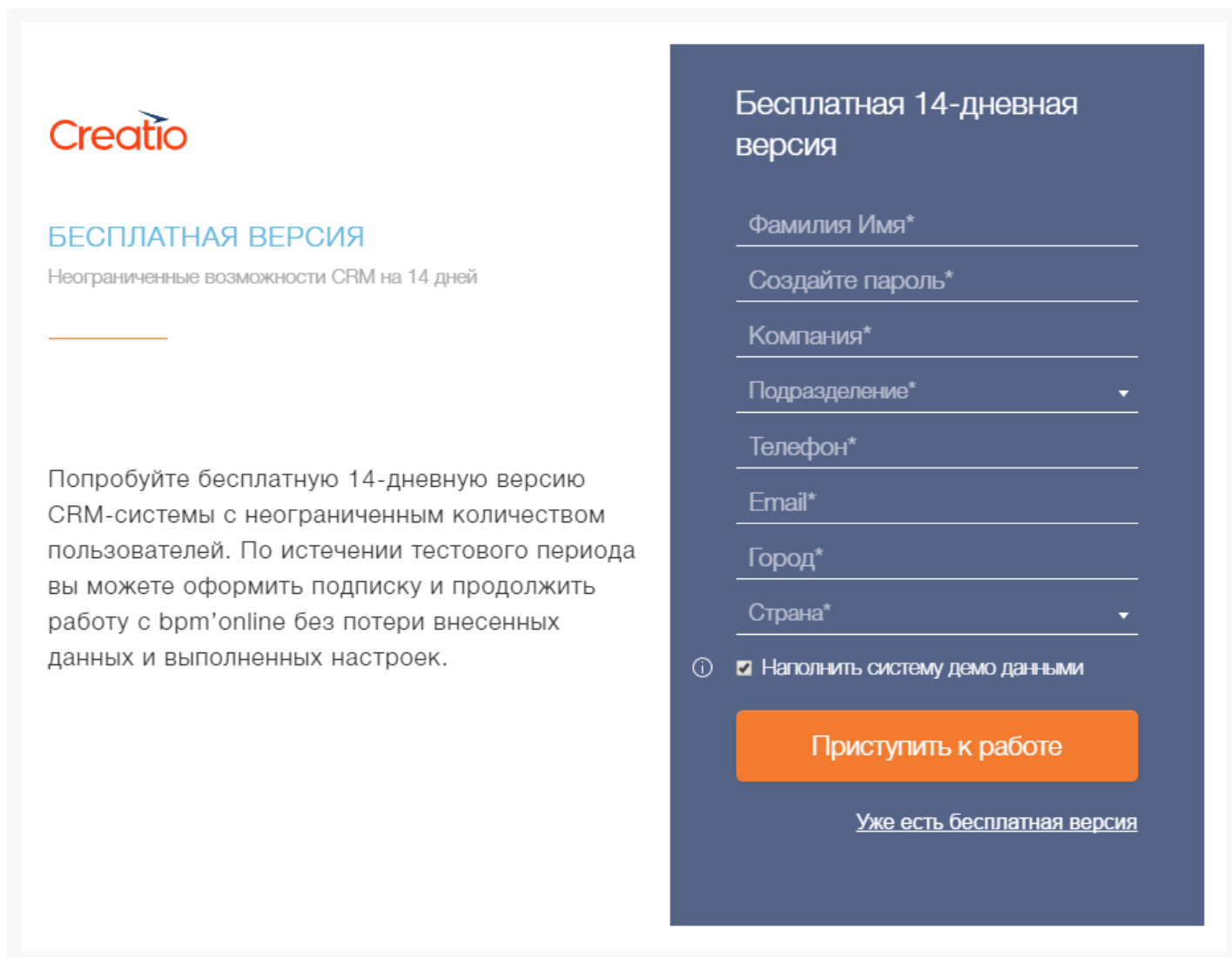
Для тестирования разработанной функциональности рекомендуется использовать бесплатную 14-дневную версию Creatio.

Регистрация тестового сайта

Заказать бесплатную 14-дневную версию Creatio для использования в качестве тестовой среды можно непосредственно из Личного кабинета. Для этого необходимо в разделе [*Приложения*] перейти на вкладку [*Тестовая среда*] и воспользоваться ссылкой на страницу заказа пробной версии Creatio.

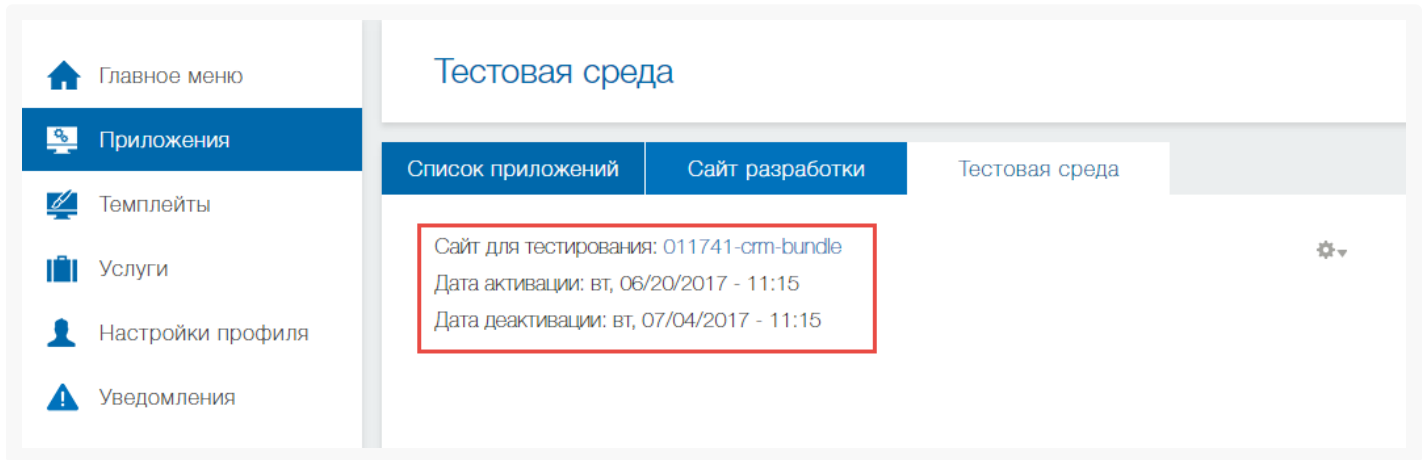


После перехода по ссылке на странице заказа необходимо заполнить форму и нажать на кнопку [*Приступить к работе*].



Тестовое приложение Creatio сразу же готово к использованию, а ссылка на него с датами активации и деактивации доступна на вкладке [*Тестовая среда*].

На заметку. Подтверждение о готовности к использованию тестового приложения придет на адрес электронной почты, указанный при регистрации.



Перенос решения в тестовую среду

Прежде чем публиковать разработанное приложение в Creatio Marketplace, необходимо удостовериться в его работоспособности на тестовом сайте. Для этого необходимо:

1. [Экспортировать пакеты](#) с разрабатываемой функциональностью из среды разработки.
2. Из экспортированных *.gz-архивов пакетов сформировать *.zip-архив. Если выгружен только один пакет, то *.zip-архив можно не создавать.
3. [Импортировать](#) *.zip-архив или единичный *.gz-архив пакета приложения Marketplace в тестовое приложение Creatio.

После установки приложения в тестовую среду необходимо проверить работоспособность разработанной функциональности. В случае обнаружения ошибок в работе, их необходимо устранить в среде разработки и повторить процедуру установки приложения.

Пример разработки приложения для Marketplace



Разработка функциональности пользовательского решения является одним из шагов общей последовательности разработки приложений для Marketplace.

Приложение для Creatio Marketplace является проектным решением, расширяющим возможности базовых продуктов Creatio и имеющее дополнительную бизнес-ценность. Поэтому разработка приложения для Marketplace практически ничем не отличается от [разработки обычного проектного решения](#).

Поскольку приложение для Creatio Marketplace является проектным решением, предоставляемым сторонним пользователям, то необходим механизм установки этого решения в пользовательские приложения. В Creatio этот механизм реализован с помощью пакетов. [Пакет Creatio](#) — это

совокупность конфигурационных элементов, которые реализуют определенный блок функциональности.

Количество и состав пакетов, реализующих приложение для Creatio Marketplace зависит от сложности функциональности, заложенной в приложение. Например, для вызова стороннего сервиса с существующей страницы записи достаточно заместить одну или несколько схем, сгруппировав их в одном пользовательском пакете. А для создания нового раздела Creatio, реализующего сложную функциональность, необходимо разработать уже несколько десятков новых схем и модулей.

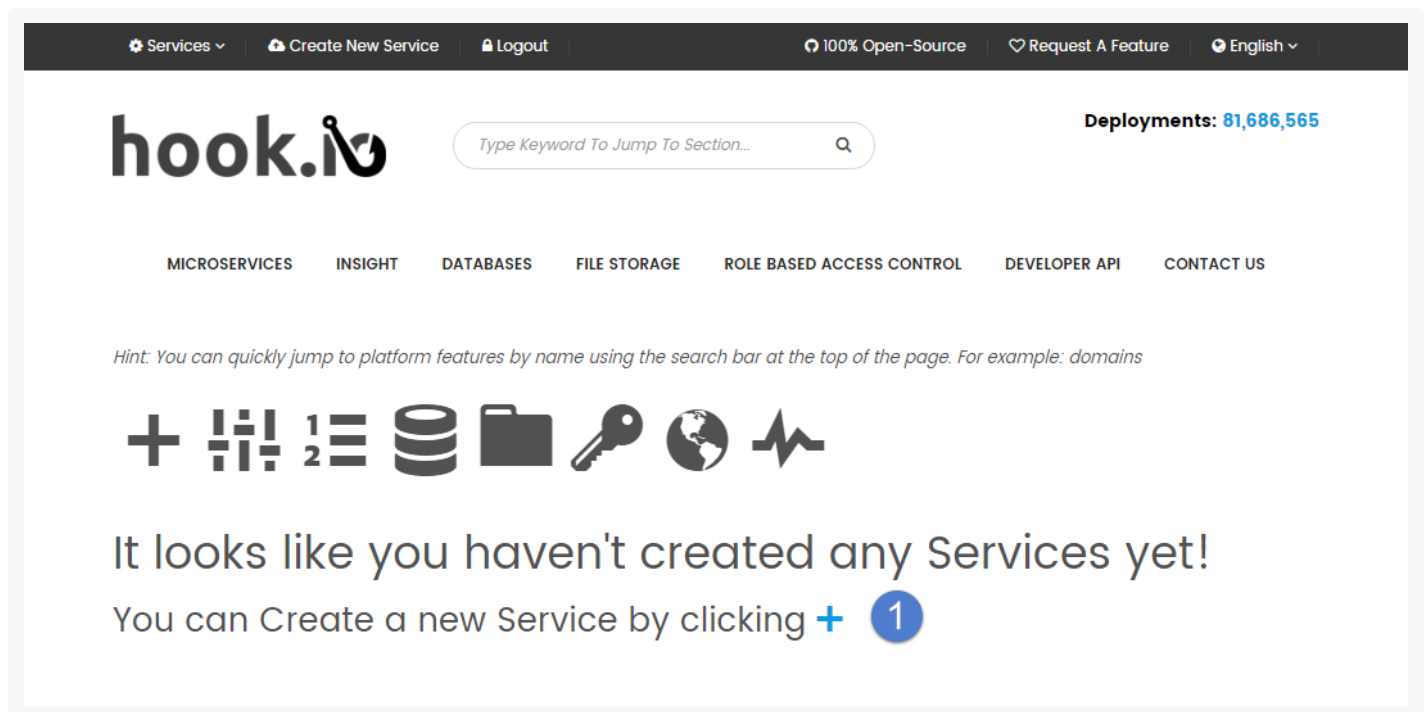
Пример. Создать проектное решение, реализующее на странице главного меню приложения Creatio для продукта sales enterprise вызов стороннего Web-сервиса, предоставляющего строку приветствия. Вызов Web-сервиса реализовать при помощи HTTP-запроса методом `POST`, передав в качестве параметра имя текущего пользователя. Результат отобразить под секцией ссылок на страницы социальных сетей.

1. Реализовать веб-сервис

В зависимости от назначения, веб-сервис может быть реализован разными способами, с помощью различных языков программирования и на разных платформах.

Исходя из условий кейса, необходим простейший, так называемый, [микросервис](https://hook.io). Для его реализации можно использовать одну из площадок, предоставляющую возможность создания микросервисов, например, open-source проект <https://hook.io>.

После регистрации пользователя площадки и входа в систему отобразится страница управления микросервисами, на которой можно создать новый Web-сервис (1).



После создания нового сервиса необходимо на странице свойств задать имя конечной точки доступа к сервису, выбрать язык программирования, на котором будет написан исходный код сервиса (1), и

добавить исходный код сервиса (2).

Endpoint

Name

Will be part of the url to access the Service. Cannot contain non-url safe characters.

Source Code

Language 1

javascript

Select where the service's source code will come from.

☒ hook.io Code Editor

☐ Github Gist

☐ Github Repo

3

Test Code Edit Code Save Code

```

1 // A simple hello world microservice
2 // Click "Save Code" to deploy this code
3 // Service will respond to HTTP requests with a string
4 module['exports'] = function helloWorld (hook) {
5   var result = "Welcome to creatiohello," + hook.params.name + "!";
6   hook.res.end(result);
7 };

```

2

После сохранения исходного кода (3) сервис становится сразу же доступен для использования. Адрес конечной точки <https://hook.io/academy-creatio-com/creatiohello>.

Исходный код сервиса представляет собой функцию, которая на основе параметра `name` HTTP-запроса формирует строку приветствия и отправляет ее в качестве ответа клиентскому приложению. Полностью исходный код микросервиса представлен ниже.

Код микросервиса



```

module["exports"] = function helloWorld (hook) {
  // Формирование строки ответа.
  var result = "Добро пожаловать в микросервис creatiohello, " + hook.params.name + "!";
  // Отправка ответа клиенту.
  hook.res.end(result);
};

```

На заметку. Приведенный исходный код микросервиса работает как с методом `POST`, так и с методом `GET` принимаемого HTTP-запроса. Например, если в адресной строке браузера ввести <https://hook.io/academy-creatio-com/creatiohello?name=User>, то сервис отработает корректно.

2. Создать пользовательский пакет

1. Перейдите в дизайнер системы по кнопке .
2. В блоке [*Конфигурирование разработчиком*] ([*Admin area*]) перейдите по ссылке [*Управление конфигурацией*] ([*Advanced settings*]).
3. В области работы с пакетами нажмите кнопку .
4. Заполните **свойства пакета**:
 - [*Название*] ([*Name*]) — название пакета (обязательное свойство). Не может совпадать с названием существующих пакетов.
 - [*Описание*] ([*Description*]) — описание пакета, например, расширенная информация о функциональности, которая будет реализована в пакете.
 - [*Хранилище системы контроля версий*] ([*Version control system repository*]) — название хранилища системы контроля версий, в котором будут фиксироваться изменения пакета (обязательное свойство). Хранилища, которые находятся в перечне хранилищ конфигурации, но не помечены как активные, не попадут в выпадающий список доступных хранилищ.

Важно. Поле Хранилище системы контроля версий (Version control system repository) заполняется при создании нового пакета и в дальнейшем недоступно для редактирования.

- [*Версия*] ([*Version*]) — версия пакета (обязательное свойство). Версия пакета может содержать цифры, символы латинского алфавита и знаки "." и "_". Добавляемое значение должно начинаться с цифры или буквы. Все элементы пакета имеют ту же версию, что и сам пакет. Указываемая версия пакета не обязательно должна совпадать с фактической версией приложения.

Package

Name*
sdkTestPackage

Description

Version control system repository
SDKPackages

Version*
7.18.0

CANCEL

CREATE AND ADD DEPENDENCIES

SAVE

5. Определите зависимости пакета

Чтобы текущий пакет наследовал функциональность приложения, необходимо определить **зависимости пакета**.

Чтобы **добавить зависимости** пакета:

- В карточке пакета нажмите кнопку [Создать и добавить зависимости] ([Create and add dependencies]).
- На вкладке [Зависимости] ([Dependencies]) в детали [Зависит от пакетов] ([Depends on packages]) установите необходимые зависимости. Чтобы текущий пакет наследовал всю **функциональность приложения**, в качестве родительского пакета необходимо выбрать пакет, который в иерархии находится следующим после пакета [Custom].

3. Создать замещающую схему

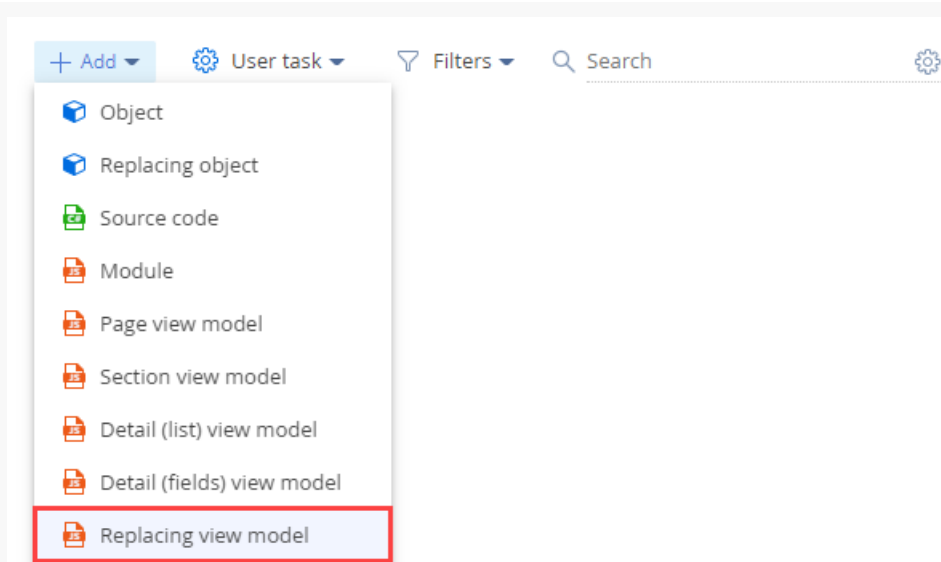
Главное меню приложения Creatio формируется в нескольких схемах модели представления, образующих иерархию наследования:

- **Базовая схема главной страницы** (`BaseIntroPageSchema` , пакет `UIv2`) — родительская схема модели представления, в которой формируются основные контейнеры главной страницы. Добавляет в модель представления разделение на левую и правую части, панели отображения видео и панель ссылок на страницы социальных сетей.
- **Схема главного меню для базового продукта** (`SimpleIntro` , пакет `UIv2`) — наследуется от `BaseIntroPageSchema` . Добавляет на главную страницу ссылки блоков [*Базис*], [*Аналитика*] и [*Настройка*].
- **Схема главного меню для продукта Enterprise** (`EnterpriseIntro` , пакет `SalesEnterprise`) — наследуется от `SimpleIntro` . Добавляет в левую часть меню дополнительные ссылки секции [*Базис*], а также ссылки секции [*Продажи*]. Дополнительно переопределяет ссылку на видеоролик.

Для внесения пользовательских изменений в главное меню продукта Sales Enterprise необходимо [заместить схему](#) `EnterpriseIntro` и реализовать в ней дополнительную функциональность.

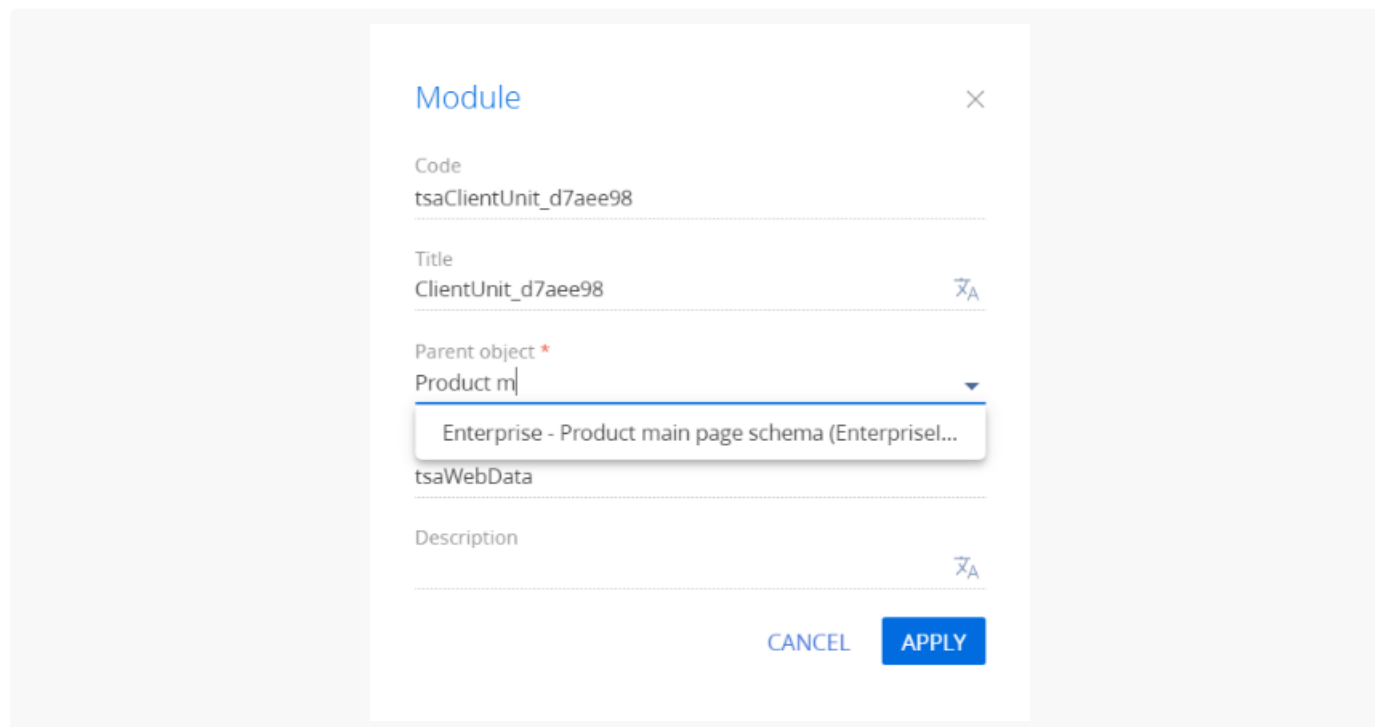
Чтобы создать **замещающую схему**:

1. [Перейдите в раздел \[Конфигурация \]](#) (`Configuration`) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Замещающая модель представления*] ([`Add`] —> [`Replacing view model`]).



3. В дизайнере схем выберите родительский объект.

Чтобы модуль замещал раздел или страницу, в обязательном свойстве [*Родительский объект*] ([*Parent object*]) схемы укажите заголовок той базовой схемы модели представления, которую необходимо заместить. Для этого в поле [*Родительский объект*] ([*Parent object*]) свойств замещающей схемы необходимо начать вводить заголовок "Схема главного меню для продукта Enterprise" ("Enterprise - Product main page schema (EnterpriseIntro)") и выбрать нужное значение из выпадающего списка.



После подтверждения выбранного родительского объекта остальные свойства будут заполнены автоматически.

Для применения заданных свойств нажмите [Применить] ([Apply]).

4. Реализовать функциональность

Для реализации пользовательской функциональности в [созданную схему](#) необходимо добавить исходный код.

Чтобы отобразить результат вызова пользовательского микросервиса на странице главного меню, в схему модели представления в массив модификаций `diff` необходимо добавить конфигурационные объекты для контейнера и содержащейся в контейнере надписи.

Для реализации необходимой пользовательской логики необходимо разработать метод вызова микросервиса `SetHelloAttribute()`, в котором будет использован JavaScript-объект `XMLHttpRequest`. Этот метод необходимо вызвать при загрузке страницы. Для этого необходимо переопределить метод родительской схемы `Init()`, в котором вызвать `SetHelloAttribute()`. Результат вызова запроса к микросервису можно получить в Callback-функции, присвоенной свойству `onreadystatechange` объекта `XMLHttpRequest`. Однако вследствие асинхронности выполнения запроса, результат необходимо сохранить в атрибуте `HelloAttribute` схемы. Сам же атрибут необходимо связать со свойством `caption` надписи, отображающей запрос.

Исходный код схемы модели представления главного меню представлен ниже.

EnterpriseIntro

```
define("EnterpriseIntro", [], function() {
    return {
        attributes: {
            // Атрибут, содержащий сообщение от пользовательского сервиса.
            "HelloAttribute": {
```

```

        // Тип данных атрибута.
        "dataValueType": Terrasoft.DataValueType.TEXT,
        // Тип атрибута – виртуальная колонка.
        "type": Terrasoft.ViewModelColumnType.VIRTUAL_COLUMN,
        // Значение по умолчанию.
        "value": ""
    },
    methods: {
        // Метод инициализации схемы.
        init: function() {
            // Вызов базовой функциональности.
            this.callParent(arguments);
            // Вызов пользовательского сервиса.
            this.SetHelloAttribute();
        },
        // Метод вызова пользовательского сервиса.
        SetHelloAttribute: function() {
            // Создание HTTP-запроса.
            var xhr = new XMLHttpRequest();
            // URL вызова пользовательского сервиса.
            var url = "https://hook.io/academy-creatio-com/creatiohello";
            // Определение имени текущего пользователя.
            var currUser = Terrasoft.SysValue.CURRENT_USER_CONTACT.displayValue;
            // Формирование параметров HTTP-запроса.
            var params = "name=" + currUser;
            xhr.open("POST", url, true);
            xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
            // Сохранение контекста.
            var self = this;
            // Callback-функция, вызываемая при получении ответа.
            xhr.onreadystatechange = function() {
                // Если получен ответ с необходимым статусом.
                if (xhr.readyState === 4 && xhr.status === 200) {
                    // Установка значения атрибута.
                    self.set("HelloAttribute", xhr.responseText);
                }
                else {
                    self.set("HelloAttribute", "Микросервис creatiohello недоступен!");
                }
            };
            // Отправка HTTP-запроса.
            xhr.send(params);
        }
    },
    diff: [
        // Пользовательский контейнер, в котором будет содержаться приветствие.
        {
            // Операция добавления.

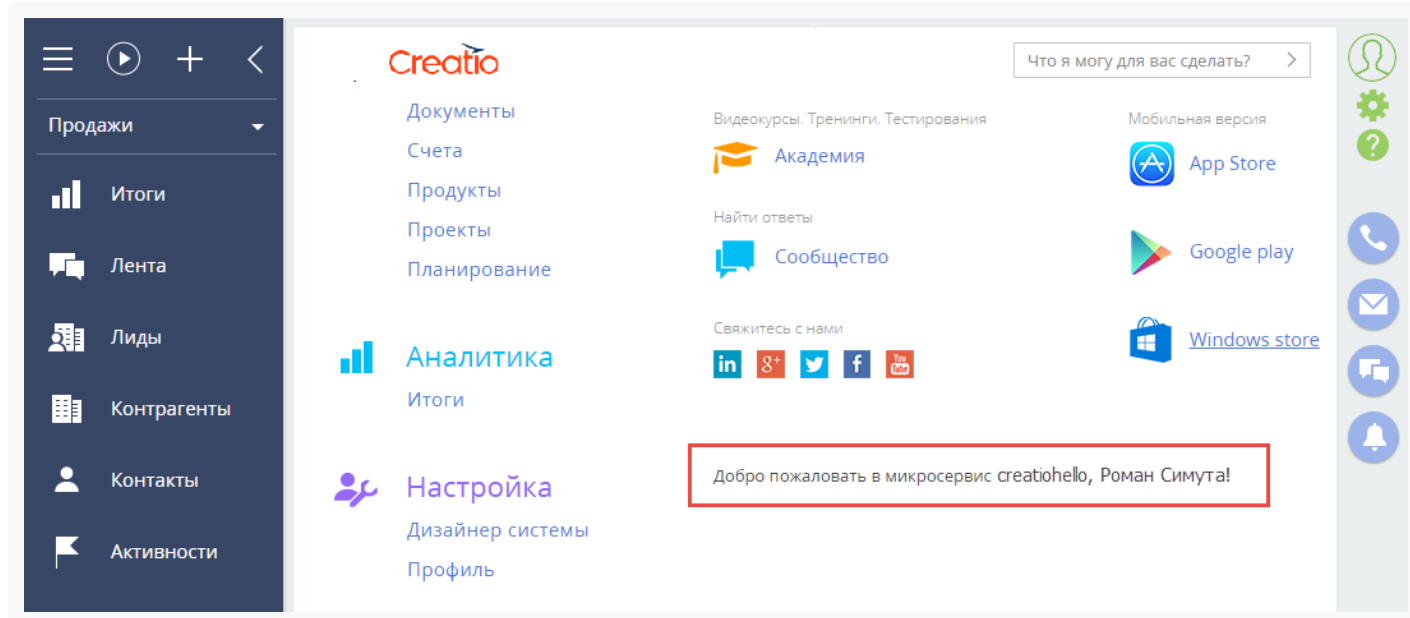
```

```

        "operation": "insert",
        // Имя элемента.
        "name": "HelloContainer",
        // Имя родительского элемента-контейнера.
        "parentName": "RightContainer",
        // Имя свойства родительского элемента для вложенных элементов.
        "propertyName": "items",
        // Конфигурационный объект свойств элемента.
        "values": {
            // Тип элемента – контейнер.
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            // Массив вложенных элементов.
            "items": []
        }
    },
    // Надпись для отображения приветствия.
    {
        "operation": "insert",
        "name": "HelloLabel",
        "parentName": "HelloContainer",
        "propertyName": "items",
        "values": {
            // Тип элемента – надпись.
            "itemType": this.Terrasoft.ViewItemType.LABEL,
            // Ссылка на атрибут с текстом надписи.
            "caption": { "bindTo": "HelloAttribute" }
        }
    }
]
};
});

```

После сохранения схемы необходимо обновить в браузере страницу главного меню приложения. На ней отобразится результат запроса к микросервису `creatiohello`.



Пример разработки приложения с пользовательским разделом для Marketplace

 Средний

Приложение для Creatio Marketplace, в котором реализован пользовательский раздел, практически ничем не отличается от разработки обычного проектного решения создания пользовательского раздела.

Для создания пользовательского раздела необходимо использовать [мастер разделов](#). Все схемы, создаваемые мастером разделов, сохраняются в пакет, который указан в системной настройке [Текущий пакет] (`CurrentPackageId`). По умолчанию текущим пакетом является пакет [Custom].

Однако, если изменения необходимо переносить в другую среду, то вести разработку в [пакете \[Custom \]](#) нельзя. Это связано с тем, что сам пакет [Custom] является системным пакетом, следовательно, его нельзя экспортировать ни с помощью утилиты WorkspaceConsole, ни с помощью SVN.

На заметку. Из пакета [Custom] можно экспортировать только схемы с помощью [механизма экспорта и импорта схем](#). Однако привязанные к пакету данные и SQL-сценарии с помощью этого механизма перенести в другой пакет нельзя.

Поэтому всю разработку новой функциональности следует вести в пользовательском пакете.



Последовательность создания приложения с пользовательским разделом:

1. Создать пользовательский пакет.
2. Установить значения системных настроек [Текущий пакет] и [Префикс названия объекта] для разработки в пользовательском пакете. Значение системной настройки [Префикс названия объекта] должно соответствовать префиксу, указанному в Кабинете разработчика.

3. С помощью мастера разделов создать раздел.
4. Реализовать необходимую функциональность раздела.
5. Привязать к пользовательскому пакету все необходимые данные, относящиеся к разделу.

Пример. Создать проектное решение, реализующее в рабочем месте [*Маркетинг*] новый раздел [*Web-данные*]. На странице записи раздела нужно реализовать отображение статистики по заданному названию сайта (его URL). Статистику отображать в html-элементе `<iframe>` с помощью стороннего Web-сайта <https://www.similarweb.com>.

1. Создать пользовательский пакет

1. Перейдите в дизайнер системы по кнопке .
2. В блоке [*Конфигурирование разработчиком*] ([*Admin area*]) перейдите по ссылке [*Управление конфигурацией*] ([*Advanced settings*]).
3. В области работы с пакетами нажмите кнопку .
4. Заполните **свойства пакета**:
 - [*Название*] ([*Name*]) — "tsaWebData".
 - [*Описание*] ([*Description*]) — "Раздел веб-статистика" ("Web data section").
 - [*Хранилище системы контроля версий*] ([*Version control system repository*]) — название хранилища системы контроля версий, в котором будут фиксироваться изменения пакета (обязательное свойство). Хранилища, которые находятся в перечне хранилищ конфигурации, но не помечены как активные, не попадут в выпадающий список доступных хранилищ.

Важно. Поле Хранилище системы контроля версий (Version control system repository) заполняется при создании нового пакета и в дальнейшем недоступно для редактирования.

- [*Версия*] ([*Version*]) — "1.0.0".

Package

Name *
tsaWebData

Description
Web data section

Version control system repository
SDKPackages

Version *
1.0.0

CANCEL CREATE AND ADD DEPENDENCIES SAVE

5. Определите зависимости пакета

Чтобы текущий пакет наследовал функциональность приложения, необходимо определить **зависимости пакета**.

Чтобы **добавить зависимости** пакета:

- В карточке пакета нажмите кнопку [Создать и добавить зависимости] ([Create and add dependencies]).
- На вкладке [Зависимости] ([Dependencies]) в детали [Зависит от пакетов] ([Depends on packages]) установите необходимые зависимости. Поскольку по условию примера необходимо добавить раздел в рабочее место [Маркетинг], то необходимо добавить зависимость от пакета [MarketingSoftkey] ([MarketingSoftKeyEnu]).

DEPENDENCIES SYSTEM INFORMATION

Depends on Packages ⓘ

Search by package

Name ^

MarketingSoftkeyEnu

+ Add

Dependent Packages ⓘ

Search by package

Name ^

Custom

2. Установить значения системных настроек [Текущий пакет] и [Префикс названия объекта]

Для того чтобы мастер разделов сохранил связанные с разделом схемы в пользовательский пакет, следует указать этот пакет в качестве текущего. Для этого необходимо в дизайнера системы перейти в раздел [Системные настройки], открыть системную настройку [Текущий пакет] и в поле [Значение по умолчанию] выбрать из справочника пакетов необходимый пользовательский пакет.

The screenshot shows the 'Текущий пакет' (Current Package) system settings form. The form has a title bar with 'Текущий пакет' and a 'Creatio' logo. Below the title bar are buttons for 'СОХРАНИТЬ' (Save) and 'ОТМЕНА' (Cancel). The form contains several fields:

- Название *** (Name): Текущий пакет
- Тип *** (Type): Справочник
- Справочник *** (Reference): Пакеты (представление)
- Значение по умолчанию** (Default Value): tsaWebData (highlighted with a red box)
- Код *** (Code): CurrentPackageId
- Кешуется** (Cached): ☐
- Персональная** (Personal): ☐
- Разрешить для пользователей портала** (Allow for portal users): ☐
- Описание** (Description): (empty field)

Затем системную настройку необходимо сохранить.

Также необходимо установить указанное в профиле разработчика значение для системной настройки [Префикс названия объекта].

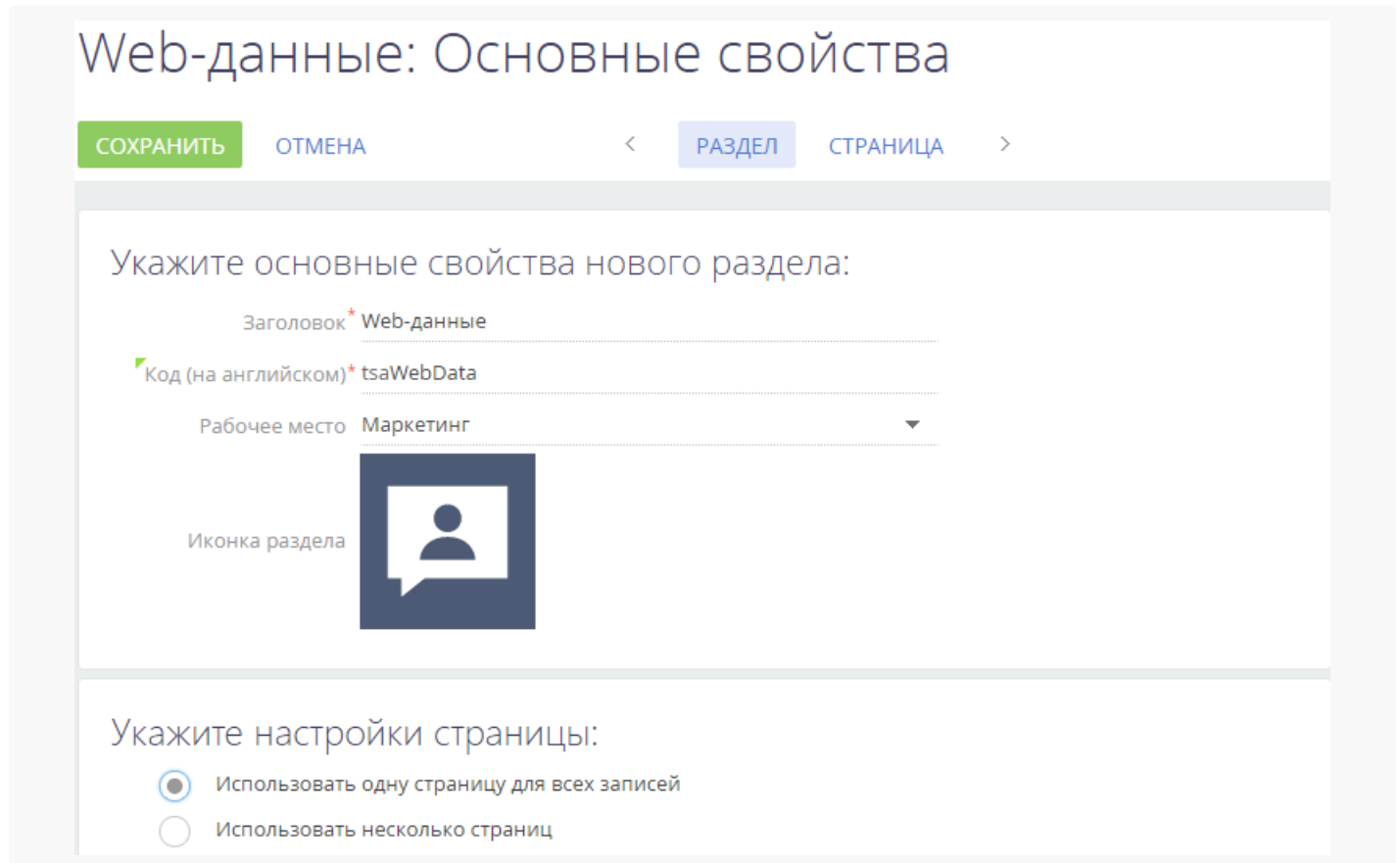
The screenshot shows the 'Префикс названия объекта' (Object Name Prefix) system settings form. The form has a title bar with 'Префикс названия объекта' and a 'Creatio' logo. Below the title bar are buttons for 'СОХРАНИТЬ' (Save) and 'ОТМЕНА' (Cancel). The form contains several fields:

- Название *** (Name): Префикс названия объекта
- Тип *** (Type): Строка (50 символов)
- Значение по умолчанию** (Default Value): tsa (highlighted with a red box)
- Код *** (Code): SchemaNamePrefix
- Кешуется** (Cached): ☒
- Персональная** (Personal): ☐
- Разрешить для пользователей портала** (Allow for portal users): ☐
- Описание** (Description): (empty field)

Этот префикс будет автоматически добавлен мастером к названиям всех схем, которые он создаст.

3. Создать раздел с помощью мастера

Для [нового раздела](#) в мастере необходимо указать значения для следующих свойств:



- [*Заголовок*] — "Web-данные". Заголовок отображается в главном меню приложения и на странице раздела.
- [*Код*] — "tsaWebData". Это название схемы объекта раздела.
- [*Рабочее место*] — "Маркетинг". Рабочее место, в котором будет отображен раздел.

Исходя из условий примера, на странице записи раздела необходимо создать поле для ввода названия URL сайта, для которого будет отображена статистика. Для этого можно использовать колонку [*Название*], создаваемую мастером по умолчанию (1). Также нужно добавить вспомогательное поле (2), в котором будет храниться создаваемая программно ссылка на страницу статистики Web-сайта <https://www.similarweb.com> для введенного URL. Эта ссылка будет присваиваться атрибуту `src` элемента `<iframe>`, размещенного на странице. Для размещения элемента `<iframe>` необходимо добавить новую вкладку [*Web-данные*] (3) на панель вкладок.

После нажатия на кнопку [*Сохранить*] мастер создаст все необходимые схемы в пакете и сохранит результаты в базу данных.

На заметку. Для завершения работы мастеру разделов может потребоваться несколько минут. После сохранения всех результатов работы мастера, в системе отобразится соответствующее сообщение.

Для отображения нового раздела необходимо перезагрузить страницу браузера с очисткой кэша.

4. Реализовать необходимую функциональность раздела

Созданный при помощи мастера раздел уже имеет всю необходимую функциональность по добавлению, удалению и сортировке записей. Также в схеме страницы записи уже унаследована вся функциональность, необходимая для сохранения и загрузки основных данных записи.

Для формирования ссылки на страницу статистики, отображаемой в поле URL, необходимо добавить кнопку. Для добавления кнопки используется конфигурационный объект, добавляемый в массив `diff` модели представления страницы. Заголовок кнопки связан с локализуемой строкой `AddUrlButtonCaption`. Событие `click` нажатия кнопки нужно связать с методом `addUrl()`, в котором формируется ссылка на страницу статистики и устанавливается новое значение для колонки `tsaURL`, связанной с полем [*URL*] страницы записи. Также эта ссылка присваивается атрибуту `src` элемента `<iframe>`.

Элемент `<iframe>` добавляется на созданную мастером вкладку также с помощью конфигурационного объекта, добавляемого в массив `diff`. В свойстве `html` конфигурационного объекта формируется html-элемент `<iframe>` с заданием нужных стилей. При этом атрибут `src` не указывается, поскольку он формируется программно в методе `addUrl()`. Для того чтобы в элементе `<iframe>` данные восстанавливались после переключения на другую вкладку, необходимо связать событие `afterrender` перерисовки контейнера, содержащего элемент `<iframe>`, с методом `addUrl()`.

После открытия сохраненной записи, событие `afterrender` отрисовки контейнера с элементом `<iframe>` наступает, как правило, раньше полной загрузки данных. Поэтому метод `addUrl()` следует

также вызвать в методе-обработчике события загрузки данных `onEntityInitialized()`.

Полностью исходный код схемы модели представления страницы записи представлен ниже.

tsaWebData1Page

```
define("tsaWebData1Page", [], function() {
    return {
        entitySchemaName: "tsaWebData",
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        diff: /**SCHEMA_DIFF*/[
            // Название Web-страницы.
            {
                "operation": "insert",
                "name": "tsaName",
                "values": {
                    "layout": {
                        "colSpan": 24,
                        "rowSpan": 1,
                        "column": 0,
                        "row": 0,
                        "layoutName": "ProfileContainer"
                    },
                    "bindTo": "tsaName"
                },
                "parentName": "ProfileContainer",
                "propertyName": "items",
                "index": 0
            },
            // Ссылка на статистику Web-страницы.
            {
                "operation": "insert",
                "name": "tsaURL",
                "values": {
                    "layout": {
                        "colSpan": 24,
                        "rowSpan": 1,
                        "column": 0,
                        "row": 0,
                        "layoutName": "Header"
                    },
                    "labelConfig": {},
                    "enabled": true,
                    "readonly": true,
                    "bindTo": "tsaURL"
                },
                "parentName": "Header",
                "propertyName": "items",
                "index": 0
            }
        ]
    };
});
```

```

},
// Вкладка на панели вкладок.
{
    "operation": "insert",
    "name": "TabData",
    "values": {
        "caption": "Web-данные",
        "items": []
    },
    "parentName": "Tabs",
    "propertyName": "tabs",
    "index": 0
},
// Кнопка добавления URL.
{
    "operation": "insert",
    "parentName": "ProfileContainer",
    "propertyName": "items",
    "name": "AddUrlButton",
    "values": {
        "layout": {
            "colSpan": 24,
            "rowSpan": 1,
            "column": 0,
            "row": 1
        },
        "itemType": Terrasoft.ViewItemType.BUTTON,
        "caption": {"bindTo": "Resources.Strings.AddUrlButtonCaption"},
        "click": {"bindTo": "addUrl"},
        "style": Terrasoft.controls.ButtonEnums.style.BLUE
    }
},
// Контейнер с внедренным Html-элементом iframe.
{
    "operation": "insert",
    "name": "IFrameStat",
    "parentName": "TabData",
    "propertyName": "items",
    "values": {
        "id": "testiframe",
        "itemType": Terrasoft.ViewItemType.CONTAINER,
        "selectors": {"wrapEl": "#stat-iframe"},
        "layout": { "colSpan": 24, "rowSpan": 1, "column": 0, "row": 0 },
        "html": "<iframe id='stat-iframe' class='stat-iframe' width='100%' height='500px'>
            <div style = 'border: 1px solid silver;'></div></iframe>",
        "afterrender": {
            "bindTo": "addUrl"
        }
    }
}

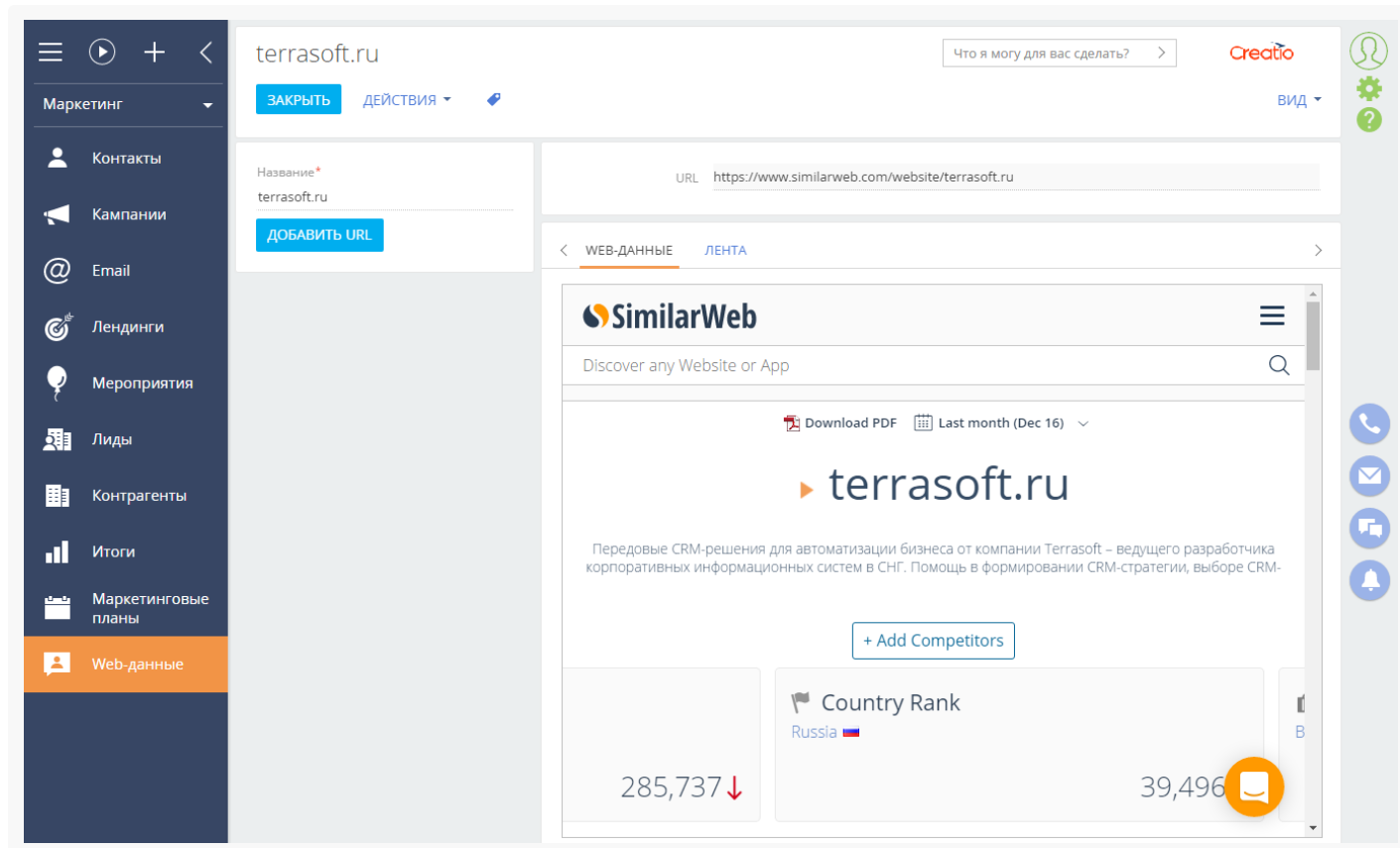
```

```

    }
  }
  /**SCHEMA_DIFF*/,
  methods: {
    // Обработчик события полной загрузки данных.
    onEntityInitialized: function() {
      // Вызов родительской реализации метода.
      this.callParent(arguments);
      // Вызов метода добавления URL к Html-элементу iframe
      this.addUrl();
    },
    // Метод добавления URL к Html-элементу iframe.
    addUrl: function() {
      // Получение компонента по его id.
      var iframe = Ext.get("stat-iframe");
      if (!iframe) {
        window.console.error("Не найдена вкладка с элементом iframe");
        return;
      }
      // Получение значения колонки [tsaName].
      var siteName = this.get("tsaName");
      if (!siteName) {
        window.console.error("Не указано имя сайта");
        return;
      }
      // Формирование ссылки на страницу статистики.
      var url = "https://www.similarweb.com/website/" + siteName;
      // Установка значения колонки [tsaName].
      this.set("tsaURL", url);
      // Присвоение ссылки на страницу статистики Html-элементу iframe.
      iframe.dom.src = url;
    }
  },
  rules: {}
};
});

```

После сохранения схемы, на странице записи раздела [*Web-статистика*] появится кнопка добавления URL и элемент `<iframe>`, в котором будет отображаться статистика по введенному Web-ресурсу.



Класс LicHelper C#

 Средний

Пространство имен `Terrasoft.Core`.

Если в лицензию приложения входит операция лицензирования, то в нужных местах исходного кода приложения необходимо предусмотреть проверку наличия лицензии на выполнение этой операции.

Для этого необходимо использовать один из методов класса `LicHelper` библиотеки `Terrasoft.Core`.

Методы

`GetHasOperationLicense(string sysPackageOperationCode) bool`

Возвращает `true`, если лицензия найдена, и `false`, если лицензия отсутствует.

Параметры

<code>sysPackageOperationCode</code>	<code>string</code>	Содержит название лицензируемой операции, переданное в службу поддержки Marketplace.
--------------------------------------	---------------------	--------------------------------------------------------------------------------------

Пример использования метода


```
UserConnection.LicHelper.GetHasOperationLicense("MyMarketplaceApplication.Use");
```

```
CheckHasOperationLicense(string sysPackageOperationCode) bool
```

Возвращает `true`, если лицензия найдена. Если лицензия отсутствует — генерирует исключение `LicException`.

Параметры

sysPackageOperationCode	string	Содержит название лицензируемой операции, переданное в службу поддержки Marketplace.
-------------------------	--------	--------------------------------------------------------------------------------------

Пример использования метода

```
UserConnection.LicHelper.CheckHasOperationLicense("MyMarketplaceApplication.Use");
```

Защита исходного кода приложения Marketplace от плагиата



Открытый исходный код приложения Marketplace содержится в [пакете](#), который заблокирован для изменения. При этом код доступен к просмотру пользователям с соответствующим уровнем доступа и не защищен от плагиата. Реализация защиты от плагиата выполняется отдельно для back-end и для front-end кода.

Защита C#-кода от плагиата

Использование [пакета-проекта](#) позволяет защитить C#-код приложения Marketplace от плагиата.

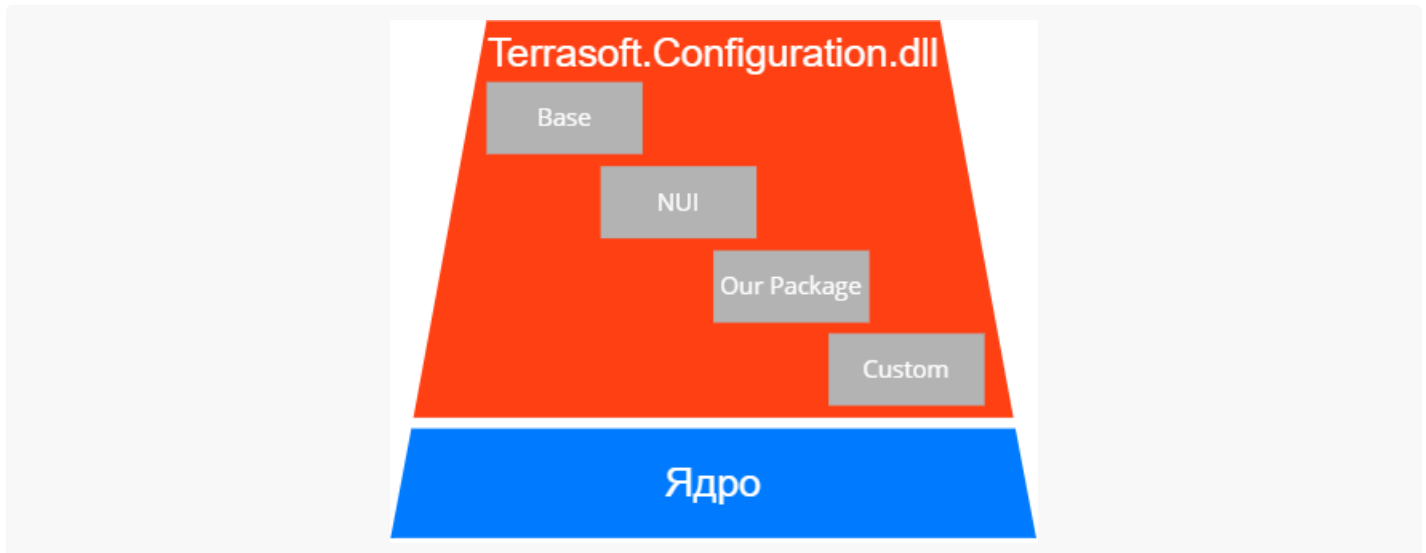
Важно. Настройку защиты исходного кода приложения Marketplace от плагиата разрешено выполнять только для исходного C#-кода собственной разработки.

Способы защиты C#-кода приложения Marketplace от плагиата:

- Для **нового приложения Marketplace** выполните разработку в пакете-проекте.
- Для **существующего приложения Marketplace** выполните конвертацию пакета в пакет-проект.

Разработка приложения Marketplace, как и других приложений, выполняется на уровне конфигурации, который содержит предустановленные пакеты приложения. В процессе публикации исходный C#-код

пакетов компилируется в библиотеку `Terrasoft.Configuration.dll` и может взаимодействовать с ядром. Описание уровней кастомизации приложения Creatio содержится в статье [Разработка приложений на платформе Creatio](#). Детализированная схема уровней кастомизации приложения Creatio представлена на рисунке ниже.

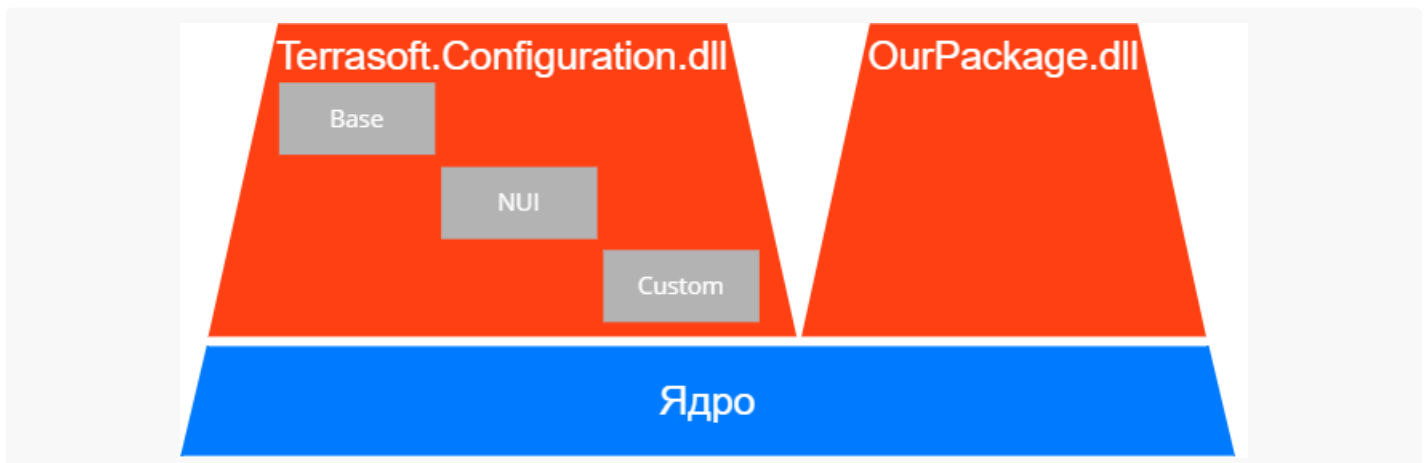


- `Base`, `NUI` — базовые пакеты приложения Creatio.
- `OurPackage` — пакет-проект с приложением Marketplace.
- `Custom` — специальный пакет приложения Creatio.

Возможности, которые предоставляет использование пакета-проекта с приложением Marketplace:

- Исключение C#-кода пользовательского приложения Marketplace из библиотеки `Terrasoft.Configuration.dll`.
- Выполнение установки приложения Marketplace, как отдельной *.dll-библиотеки.

Схема уровней кастомизации приложения Creatio при наличии пакета-проекта, который содержит приложение Marketplace, представлена на рисунке ниже.



Разработать приложение Marketplace в пакете-проекте

Разработку нового приложения Marketplace рекомендуется выполнять в пакете-проекте.

Чтобы **разработать приложение Marketplace в пакете-проекте**:

1. Настройте Creatio для работы в файловой системе.
2. Создайте пользовательский пакет.
3. Разработайте пользовательскую функциональность.
4. Выполните сборку пакета-проекта.

1. Настроить Creatio для работы в файловой системе

Для настройки Creatio для работы в файловой системе воспользуйтесь инструкцией, которая приведена в статье [Внешние IDE](#).

2. Создать пользовательский пакет

Инструменты, который позволяют создать пользовательский пакет:

- Creatio IDE. Создание пользовательского пакета с использованием Creatio IDE описано в статье [Создать пользовательский пакет](#).
- Утилита [Creatio command-line interface utility \(clio\)](#).

Чтобы **создать пользовательский пакет с использованием утилиты clio**:

1. Выполните установку clio (при необходимости).

Команда для установки clio

```
dotnet tool install clio -g
```

Установка clio подробно описана в официальной [документации утилиты на GitHub](#).

2. Перейдите в каталог `pkg` приложения.

Команда для перехода в каталог Pkg

```
cd C:\inetpub\wwwroot\creatio\Terrasoft.WebApp\Terrasoft.Configuration\Pkg;
```

3. Создайте новый пакет.

Команда для создания нового пакета

```
clio init OurPackage;
```

4. Установите зависимости пакета. Для этого отредактируйте файл `descriptor.json`.

Ниже приведен пример установки зависимостей (свойство `DependsOn`) пакета `OurPackage` от пакета `ProductCore` и добавления описания (свойство `Descriptor`) пакета.

Пример установки зависимостей и добавления описания пакета

```
{
  "Descriptor": {
    "Uid": "45cc06b2-6448-4d9e-9f51-bee31a6dbc25",
    "PackageVersion": "7.8.0",
    "Name": "OurPackage",
    "ModifiedOnUtc": "/Date(1633420586000)/",
    "Maintainer": "Customer",
    "Description": "Payment calculator",
    "DependsOn": [
      {
        "Uid": "2fabaf6c-0f92-4530-aef8-40345c021da2",
        "PackageVersion": "7.8.0",
        "Name": "ProductCore"
      }
    ]
  }
}
```

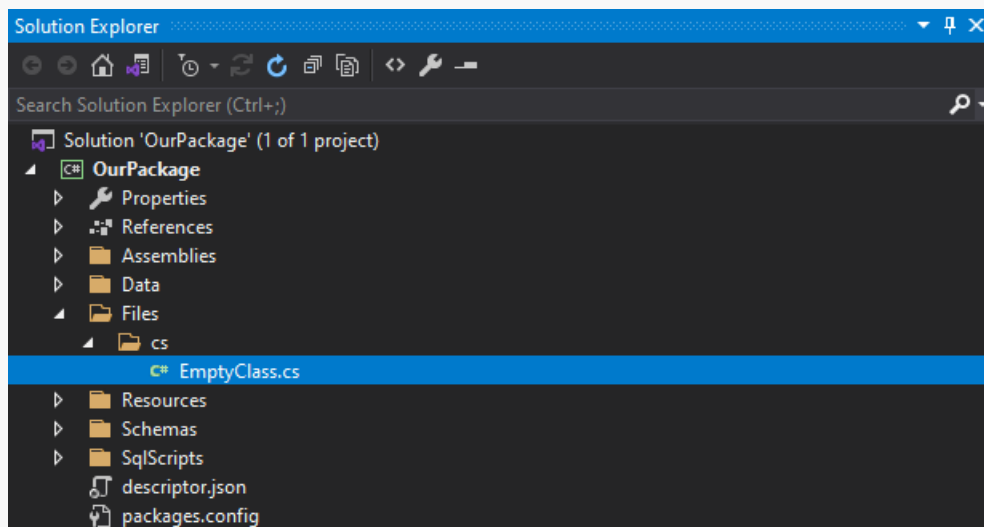
В результате создан пользовательский пакет `OurPackage`, который зависит от пакета `ProductCore`.

3. Разработать пользовательскую функциональность

Для разработки пользовательской функциональности можно использовать любую [внешнюю IDE](#). В нашем примере разработка выполняется в Microsoft Visual Studio Code.

Чтобы **разработать пользовательскую функциональность**:

1. Откройте проект `OurPackage.sln`.

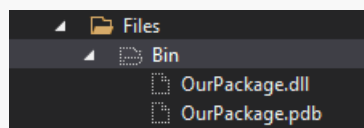



2. Установите NuGet-пакет CreatioSDK из репозитория, который доступен на официальном [сайте nuget](#). Выберите версию CreatioSDK, которая подходит для ваших целей.
3. Реализуйте пользовательскую функциональность в каталоге `Files\cs` приложения.

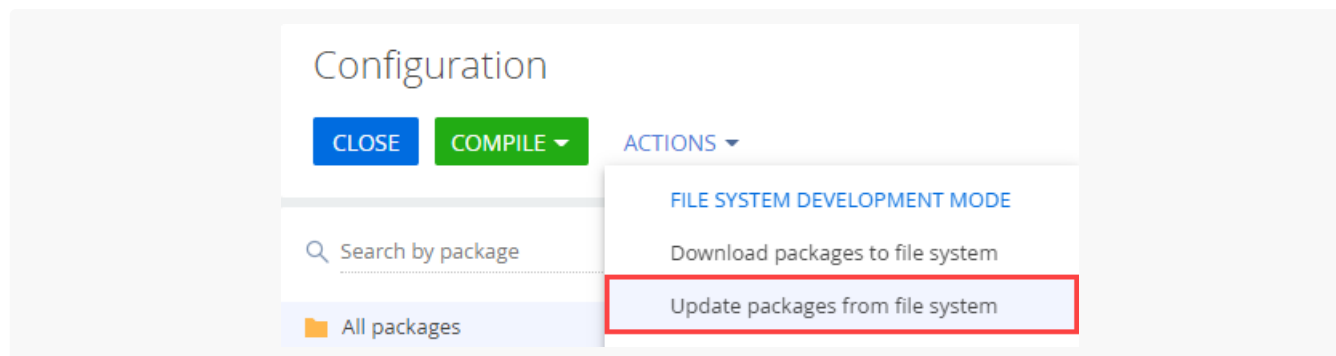
При разработке C#-кода можно создать приложение с помощью IDE. Для этого в Visual Studio нажмите комбинацию `Ctrl+Shift+B`.

4. Выполните сборку приложения.

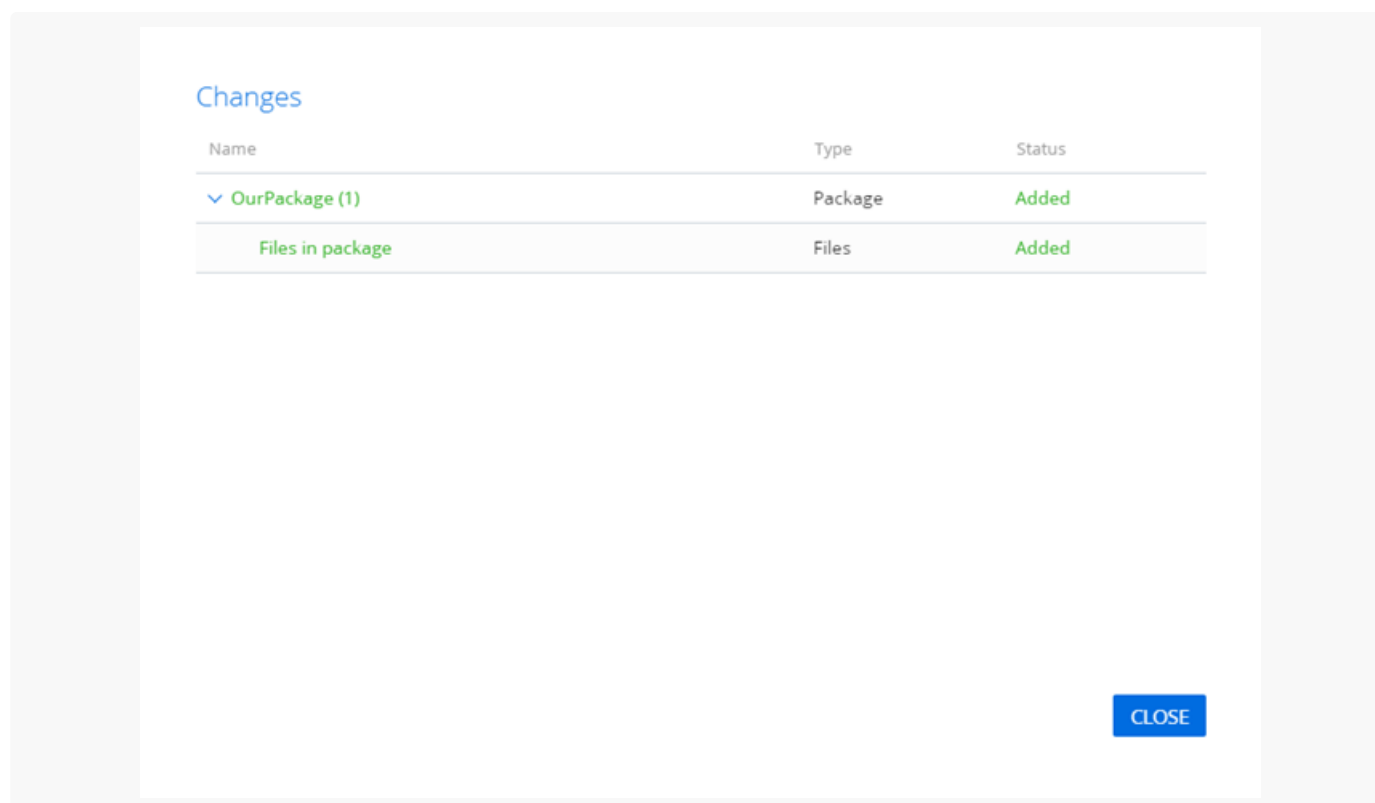
Если сборка приложения выполнится успешно, то *.dll, *.pdb и другие вспомогательные файлы помещаются в каталог `Files\Bin` приложения.



5. Загрузите пакет `OurPackage` из каталога `[Путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration\Pkg` в базу данных.
 - a. Перейдите в дизайнер системы по кнопке .
 - b. В блоке [Конфигурирование разработчиком] ([Admin area]) перейдите по ссылке [Управление конфигурацией] ([Advanced settings]).
 - c. В группе [Разработка в файловой системе] ([File system development mode]) выпадающего списка [Действия] ([Actions]) панели инструментов нажмите [Обновить пакеты из файловой системы] ([Update packages from file system]).



В результате пакет `OurPackage` загружен в Creatio IDE.



6. Перезапустите приложение.

Команда для перезапуска приложения

```
clio restart
```

4. Выполните сборку пакета-проекта

Назначение выполнения сборки пакета-проекта — подготовка приложения Marketplace к публикации на онлайн-площадке Creatio Marketplace.

Важно. Если вы не хотите, чтобы исходный C#-код собственной разработки был включен в пакет-проект, то обязательно удалите его перед выполнением экспорта пакета.

Чтобы **выполнить сборку пакета-проекта**:

1. Удалите исходный C#-код собственной разработки из пакета-проекта (при необходимости).
Выполнить, если вы не хотите, чтобы исходный C#-код был включен в пакет-проект.
2. Для автоматизации процесса выполнения сборки пакета-проекта создайте файл `PackagePublish.target`.
3. В файл `PackagePublish.target` добавьте код.

Файл `PackagePublish.target`

```
<?xml version="1.0" encoding="utf-8" ?>
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <DestinationFolder>C:\PkgRelease\$(AssemblyName)</DestinationFolder>
  </PropertyGroup>

  <ItemGroup>
    <PkgAssemblies Include="Assemblies\**"/>
  </ItemGroup>
  <ItemGroup>
    <PkgData Include="Data\**"/>
  </ItemGroup>
  <ItemGroup>
    <PkgFiles Include="Files\Bin\**"/>
  </ItemGroup>
  <ItemGroup>
    <PkgProperties Include="Properties\**"/>
  </ItemGroup>
  <ItemGroup>
    <PkgResources Include="Resources\**"/>
  </ItemGroup>
  <ItemGroup>
    <PkgSchemas Include="Schemas\**"/>
  </ItemGroup>
  <ItemGroup>
    <PkgSqlScripts Include="SqlScripts\**"/>
  </ItemGroup>
  <ItemGroup>
    <PkgDescriptor Include="descriptor.json"/>
  </ItemGroup>

  <Target Name="CopyFiles">
    <Copy
      SourceFiles="@ (PkgAssemblies)"
      DestinationFiles="@ (PkgAssemblies->'$(DestinationFolder)\Assemblies\%(RecursiveDi
    />
```

```

    <Copy
      SourceFiles="@ (PkgData)"
      DestinationFiles="@ (PkgData->'$(DestinationFolder)\Data\%(RecursiveDir)\%(Filename)
    />
    <Copy
      SourceFiles="@ (PkgFiles)"
      DestinationFiles="@ (PkgFiles->'$(DestinationFolder)\Files\Bin\%(RecursiveDir)\%(Fi
    />
    <Copy
      SourceFiles="@ (PkgProperties)"
      DestinationFiles="@ (PkgProperties->'$(DestinationFolder)\Properties\%(RecursiveDi
    />
    <Copy
      SourceFiles="@ (PkgResources)"
      DestinationFiles="@ (PkgResources->'$(DestinationFolder)\Resources\%(RecursiveDir)
    />
    <Copy
      SourceFiles="@ (PkgSchemas)"
      DestinationFiles="@ (PkgSchemas->'$(DestinationFolder)\Schemas\%(RecursiveDir)\%(Fi
    />
    <Copy
      SourceFiles="@ (PkgSqlScripts)"
      DestinationFiles="@ (PkgSqlScripts->'$(DestinationFolder)\SqlScripts\%(RecursiveDi
    />
    <Copy
      SourceFiles="@ (PkgDescriptor)"
      DestinationFiles="@ (PkgDescriptor->'$(DestinationFolder)\%(RecursiveDir)\%(Filenam
    />
  </Target>

  <Target Name="CreateRelease" AfterTargets="CopyFiles">
    <Exec Command="clio generate-pkg-zip  $(DestinationFolder) -d C:\PkgRelease\$(Assembl
  </Target>
</Project>

```

4. В файл `OurPackage.csproj` добавьте строку.

Файл `OurPackage.csproj`

```
<Import Project="PackagePublish.target" />
```

5. Откройте командную строку и выполните команду.

```
msbuild /t:CreateRelease
```

В результате пакет-проект будет выгружен в каталог `C:\PkgRelease`, который содержит вложенный

каталог `OurPackage` и *.gz-архив `OurPackage.gz`. Этот архив содержит подготовленное к публикации на онлайн-площадке Creatio Marketplace приложение Marketplace.

Конвертировать пакет с приложением Marketplace в пакет-проект

Подготовка ранее разработанного приложения Marketplace к конвертации в пакет-проект может потребовать значительных модификаций.

Конвертация разработанного приложения Marketplace в пакет-проект выполняется с помощью команды `clio convert`, которая описана в официальной документации утилиты [на GitHub](#).

Невозможно корректно выполнить конвертацию некоторых файлов и схем, например, элемента [*Действие процесса*] ([*User task*]) бизнес-процесса. Независимо от значения признака [*Partial*], элемент [*Действие процесса*] ([*User task*]) является частичным классом и должен находиться в библиотеке `Terrasoft.Configuration.dll`. При выполнении конвертации код элемента [*Действие процесса*] ([*User task*]) по умолчанию сохраняется в каталог `AutoGenerated` пакета-проекта, а не в библиотеку `Terrasoft.Configuration.dll`.

Чтобы **конвертировать разработанное приложение Marketplace в пакет-проект**, выполните одну из команд:

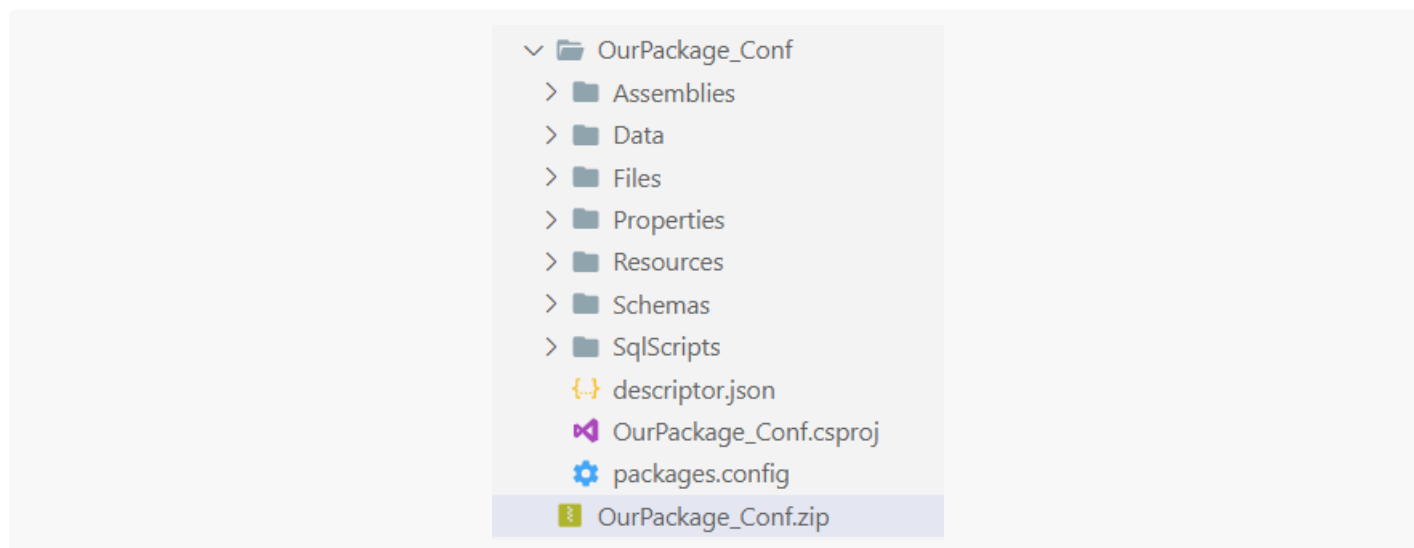
- `clio convert .\OurPackage_Conf\ -c false`.
- `clio convert .\OurPackage_Conf\`.

Результат выполнения команд идентичен, поскольку для ключа `-c` (ConvertSourceCode) значение `false` установлено по умолчанию.

Содержимое C#-проекта после выполнения конвертации:

- Пакет-проект с конвертированным приложением Marketplace. Структура пакета-проекта описана в статье [Пакет-проект](#).
- *.zip-архив с неконвертированным приложением Marketplace.

Структура C#-проекта после выполнения конвертации представлена на рисунке ниже.



В результате разработанное приложение Marketplace конвертировано в пакет-проект, который можно устанавливать в приложение Creatio.

На заметку. Формирование пакета-проекта является обязательным звеном CI/CD-конвейера. В репозитории рекомендуется хранить незащищенный от плагиата исходный C#-код приложения Marketplace.

Защита JavaScript-кода от плагиата

Способы защиты JavaScript-кода приложения Marketplace от плагиата:

- Минификация.
- Обфускация.

Важно. Запрещено изменять структуру схемы клиентского модуля, поскольку дизайнеры приложения настроены на работу с определенной структурой схемы.

Лучшей практикой защиты JavaScript-кода от плагиата является использование миксинов для реализации логики, которую необходимо защитить. Не рекомендуется обфусцировать схемы клиентских модулей, которые используются мастерами (разделов, страниц, деталей) приложения.

Существует большое количество решений с открытым исходным кодом для обфускации JavaScript-кода. В нашем примере мы используем обфускатор JavaScript Obfuscator. Официальная документация обфускатора доступна [на GitHub](#).

Чтобы **защитить JavaScript-код от плагиата с использованием JavaScript Obfuscator**:

1. Выполните установку JavaScript Obfuscator.

Команда для установки JavaScript Obfuscator

```
npm install javascript-obfuscator -g
```

Установка JavaScript Obfuscator подробно описана в официальной [документации обфускатора на GitHub](#).

2. Подготовьте JavaScript-код к обфускации.
 - a. Создайте миксин. В нашем примере это `MRKT_DemoMixin`. Миксины подробно описаны в статье [Клиентская схема](#).
 - b. В миксине реализуйте JavaScript-код, который планируется обфусцировать.

`MRKT_DemoMixin`

```
define("MRKT_DemoMixin", [], function() {  
    Ext.define("Terrasoft.configuration.mixins.MRKT_DemoMixin", {
```

```

    alternateClassName: "Terrasoft.MRKT_DemoMixin",

    secretMethod: function(){
        console.log("MRKT_DemoMixin");
    },
  });
});

```

3. Укажите МИКСИН `MRKT_DemoMixin` в СВОЙСТВЕ `mixins` КЛИЕНТСКОЙ СХЕМЫ (например, `ContactPageV2`).

ContactPageV2

```

define("ContactPageV2", ["MRKT_DemoMixin"],
function() {
    return {
        entitySchemaName: "Contact",
        mixins: {
            "MRKT_DemoMixin": "Terrasoft.MRKT_DemoMixin"
        },
        attributes: {},
        modules: /**SCHEMA_MODULES*/{}/**SCHEMA_MODULES*/,
        details: /**SCHEMA_DETAILS*/{}/**SCHEMA_DETAILS*/,
        businessRules: /**SCHEMA_BUSINESS_RULES*/{}/**SCHEMA_BUSINESS_RULES*/,
        methods: {
            onEntityInitialized: function() {
                this.callParent(arguments);

                /* Consume MRKT_DemoMixin. */
                this.secretMethod();
            },
        },
        dataModels: /**SCHEMA_DATA_MODELS*/{}/**SCHEMA_DATA_MODELS*/,
        diff: /**SCHEMA_DIFF*/[]/**SCHEMA_DIFF*/
    };
});

```

4. Скопируйте JavaScript-код, который планируется обфусцировать. Возможно, будет необходимость редактировать код в будущем.
5. Выполните обфускацию JavaScript-кода.

Команда для обфускации JavaScript-кода

```
javascript-obfuscator MRKT_DemoMixin.js --output MRKT_DemoMixin.obfuscated.js
```

Выполнение обфускации в JavaScript Obfuscator подробно описана в официальной [документации](#)

[обфускатора на GitHub.](#)

В результате получим обфусцированный файл. Пример обфусцированного файла приведен ниже.

Пример обфусцированного файла

```
function a0_0x2c69() {
    var _0x272852 = ['Terrasoft.MRKT_DemoMixin', 'Ok\x20-\x20MRKT_DemoMixin', '636527SjITzq', '4
    a0_0x2c69 = function() {
        return _0x272852;
    };
    return a0_0x2c69();
}
var a0_0x31e3ab = a0_0x3f9a;

function a0_0x3f9a(_0x104961, _0x4f9883) {
    var _0x2c694d = a0_0x2c69();
    return a0_0x3f9a = function(_0x3f9a09, _0x1d247d) {
        _0x3f9a09 = _0x3f9a09 - 0xa7;
        var _0x481962 = _0x2c694d[_0x3f9a09];
        return _0x481962;
    }, a0_0x3f9a(_0x104961, _0x4f9883);
}(function(_0x479332, _0x464f89) {
    var _0x279ddd = a0_0x3f9a,
        _0x3c692e = _0x479332();
    while (!![]) {
        try {
            var _0xae3ae0 = -parseInt(_0x279ddd(0xac)) / 0x1 + parseInt(_0x279ddd(0xa7)) / 0x2 +
            if (_0xae3ae0 === _0x464f89) break;
            else _0x3c692e['push'](_0x3c692e['shift']());
        } catch (_0x44cb38) {
            _0x3c692e['push'](_0x3c692e['shift']());
        }
    }
})(a0_0x2c69, 0xc4309), define(a0_0x31e3ab(0xa8), [], function() {
    var _0x3dbe12 = a0_0x31e3ab;
    Ext['define']('Terrasoft.configuration.mixins.MRKT_DemoMixin', {
        'alternateClassName': _0x3dbe12(0xaa),
        'secretMethod': function() {
            var _0x3b8302 = _0x3dbe12;
            return _0x3b8302(0xab);
        }
    });
}));
```

Чтобы **посмотреть отображение JavaScript-кода на странице браузера:**

1. Очистите кэш браузера.

2. Обновите страницу.
3. Запустите [инструменты отладки](#).

Пример отображения JavaScript-кода на странице браузера приведен на рисунке ниже.

```

1  define('MRKT_DemoMixinResources', ['terrasoft'], function(Terrasoft) {
2    var localizableStrings = {};
3    var localizableImages = {};
4    return {
5      localizableStrings: localizableStrings,
6      localizableImages: localizableImages
7    };
8  });
9  Terrasoft.configuration.Structures["MRKT_DemoMixin"] = {
10   innerHierarchyStack: ["MRKT_DemoMixin"]
11 };
12 function a0_0x2c69() {
13   var _0x272852 = ['Terrasoft.MRKT_DemoMixin', 'Ok\\x20\\x20MRKT_DemoMixin', '6365275jITz
14   a0_0x2c69 = function() {
15     return _0x272852;
16   }
17   ;
18   return a0_0x2c69();
19 }
20 var a0_0x31e3ab = a0_0x3f9a;
21 function a0_0x3f9a(_0x104961, _0x4f9883) {
22   var _0x2c694d = a0_0x2c69();
23   return a0_0x3f9a = function(_0x3f9a09, _0x1d247d) {
24     _0x3f9a09 = _0x3f9a09 - 0xa7;
25     var _0x481962 = _0x2c694d[_0x3f9a09];
26     return _0x481962;
27   }
28   ,
29   a0_0x3f9a(_0x104961, _0x4f9883);
30 }
31 (function(_0x479332, _0x464f89) {
32   var _0x279ddd = a0_0x3f9a
33   , _0x3c692e = _0x479332();
34   while (!![]) {
35     try {
36       var _0xae3ae0 = -parseInt(_0x279ddd(0xac)) / 0x1 + parseInt(_0x279ddd(0xa7)) /
37       if (_0xae3ae0 === _0x464f89)
38         break;
39       else
40         _0x3c692e['push'](_0x3c692e['shift']());
41     } catch (_0x44cb38) {
42       _0x3c692e['push'](_0x3c692e['shift']());
43     }
44   }
45 }(a0_0x2c69, 0xc4309),
46 define(a0_0x31e3ab(0xa8), [], function() {
47   var _0x3dbe12 = a0_0x31e3ab;
48   Ext['define']('Terrasoft.configuration.mixins.MRKT_DemoMixin', {
49     'alternateClassName': _0x3dbe12(0xaa),
50     'secretMethod': function() {
51       var _0x3b8302 = _0x3dbe12;
52       return _0x3b8302(0xab);
53     }
54   });
55 });
56

```

На заметку. Обфусцирование JavaScript-кода является обязательным звеном CI/CD-конвейера. В репозитории рекомендуется хранить незащищенный от плагиата исходный JavaScript-код приложения Marketplace.