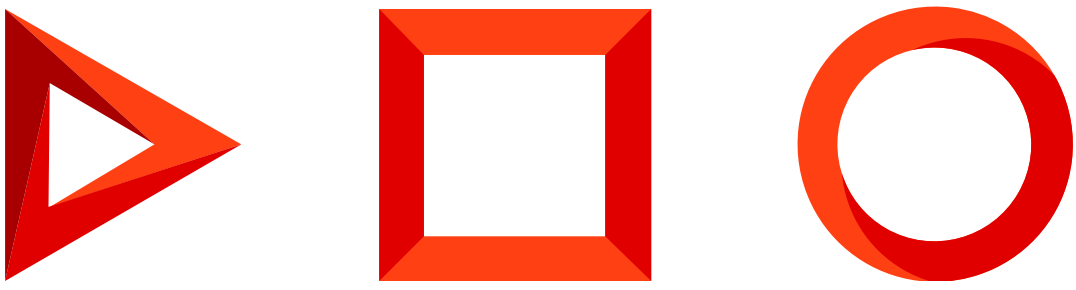
Creatio Academy

Разработка мобильного приложения

Общие принципы работы мобильного приложения

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

Общие принципы работы мобильного приложения

Легкий

Архитектура мобильного приложения

Реализация приложений для мобильных устройств

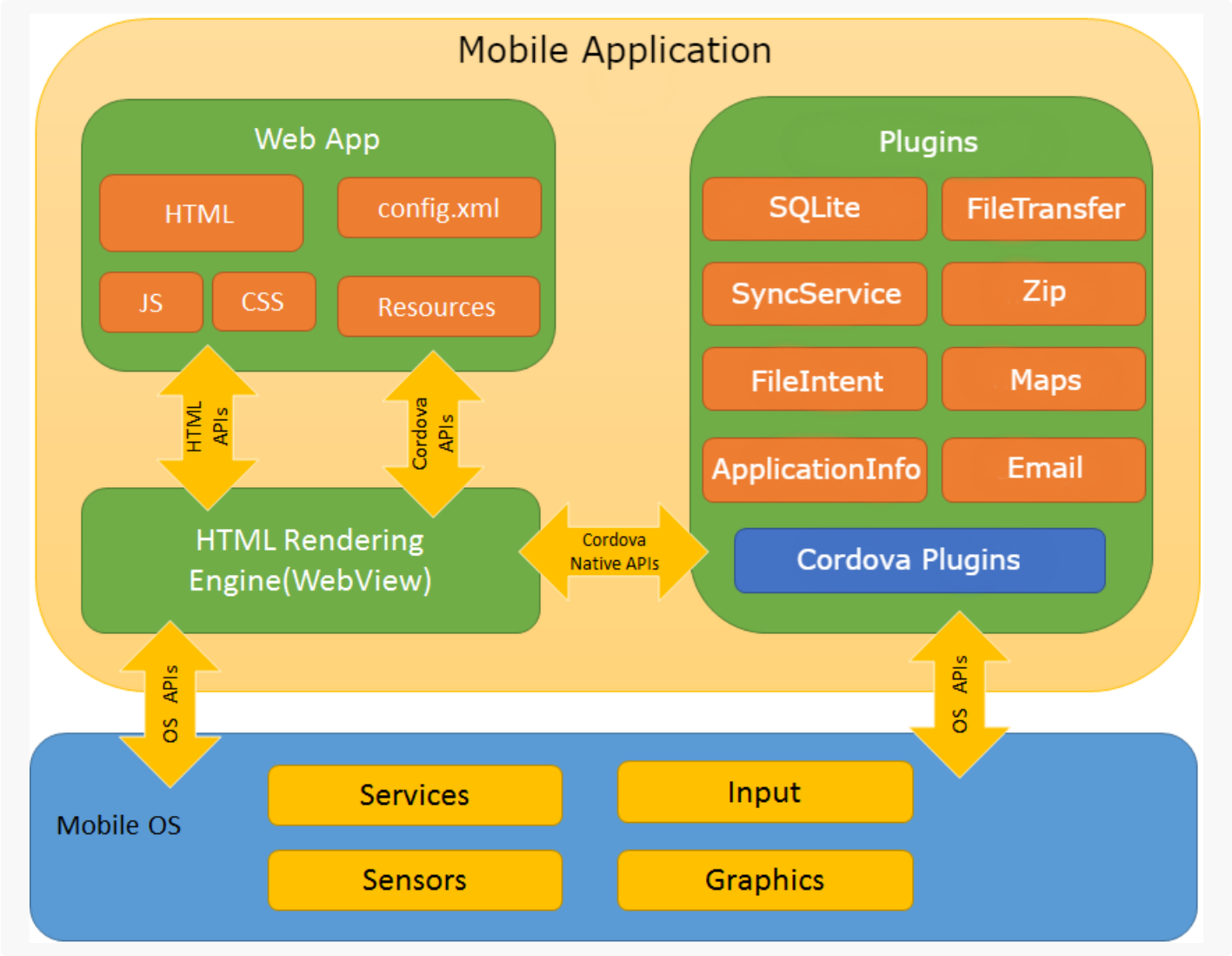
Существует три подхода технической реализации приложений для мобильных устройств:

Мобильное native-приложение — это специально разработанное приложение под конкретную мобильную платформу (iOS, Android, Windows Phone). Такое приложение разрабатывается на языке высокого уровня и компилируется в т. н. native-код ОС, обеспечивающий максимальную производительность. Главным недостатком мобильных приложений этого типа является низкая переносимость между мобильными платформами.

Мобильное web-приложение — специализированный web-сайт, адаптированный для просмотра и функционирования на мобильном устройстве. Такое приложение хоть и не зависит от платформы, однако требует постоянного подключения к сети, т. к. физически размещено не на мобильном устройстве, а на отдельном сервере.

Гибридное приложение — мобильное приложение, "упакованное" в native-оболочку. Такое приложение, как и native, устанавливается из онлайн-магазина и имеет доступ к тем же возможностям мобильного устройства, но разрабатываетсяс помощью web-языков HTML5, CSS и JavaScript. В отличие от native-приложения является легкопереносимым между различными платформами, однако несколько уступает в производительности. Мобильное приложение Creatio относится к этому типу приложений.

Схема архитектуры мобильного приложения:



Для создания гибридных приложений мобильное приложение Creatio использует возможности фреймворка Apache Cordova. Фреймворк Cordova обладает следующими преимуществами:

- Предоставляет доступ к программному интерфейсу мобильного устройства (API) для взаимодействия с базой данных или оборудованием (например, камерой или картой памяти).
- Предоставляет native-плагины для работы с API разных мобильных платформ (iOS, Android, Windows Phone и др.). Кроме того, разработка пользовательских плагинов позволяет добавлять функциональность и расширять API. Перечень доступных платформ и функциональность базовых native-плагинов Cordova содержится в документации Cordova.

Ядро мобильного приложения Creatio предоставляет унифицированный интерфейс для взаимодействия всех остальных клиентских частей приложения. Используемые ядром JavaScript-файлы условно можно разделить на базовые и конфигурационные категории.

Базовые скрипты содержатся в сборке приложения, публикуемой в магазине приложений, и включают в себя следующие элементы:

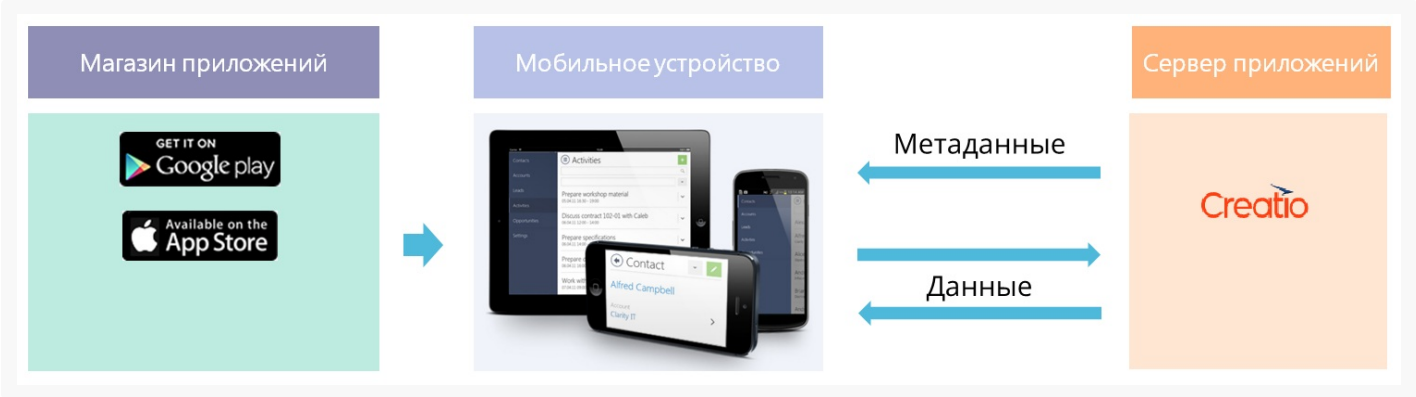
- MVC-компоненты (представления страниц, контроллеры, модели).
- Модули синхронизации (импорт и экспорт данных, импорт метаданных, импорт файлов и т. д.).

- Клиентские классы веб-сервисов.
- Классы, предоставляющие доступ к native-плагинам.

Приложение получает конфигурационные файлы в ходе синхронизации с сервером Creatio и сохраняет локально в файловой системе устройства. Конфигурационные файлы включают в себя манифест мобильного приложения Creatio, а также схемы и настройки разделов.

Схема работы мобильного приложения Creatio

Опубликованное в магазине мобильное приложение Creatio представляет собой набор модулей, необходимых для синхронизации с сервером — основным приложением. Именно в основном приложении хранятся все необходимые настройки мобильного приложения и данные. Условно функционирование мобильного приложения можно представить в виде следующей схемы:



После установки приложения на мобильное устройство пользователь, указав параметры соединения с сервером Creatio, получает метаданные (структура приложения, системные данные) и данные от сервера.

Такая схема работы дает очевидое преимущество — мобильное приложение совместимо со всеми существующими продуктами Creatio. Каждый продукт, каждый отдельно взятый сайт клиента может содержать собственный набор настроек мобильного приложения, свою логику работы, даже свой визуальный интерфейс. Все что нужно сделать мобильному пользователю — установить мобильное приложение и выполнить синхронизацию с сайтом Creatio.

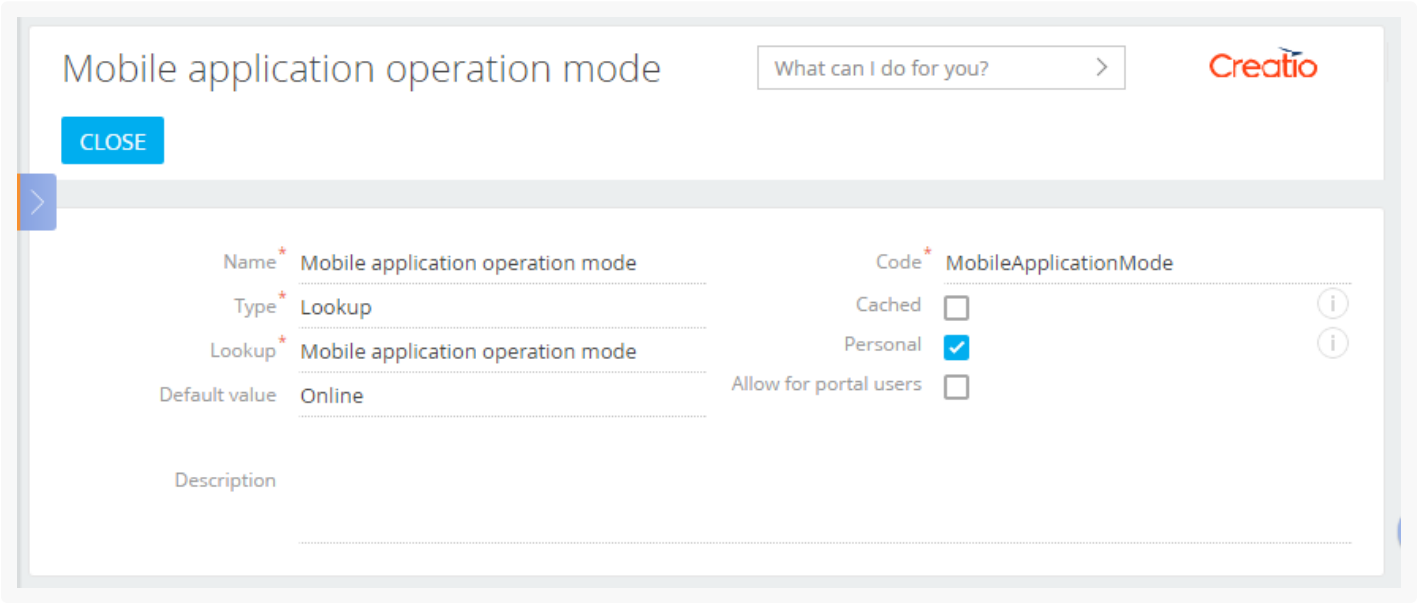
Режимы работы мобильного приложения Creatio

Мобильное приложение Creatio может работать в следующих режимах:

- **Гибридный режим**. Гибридный режим предназначен для работы с данными и автоматически включается при отсутствии стабильного соединения с сервером Creatio. Этот режим позволяет создавать новые записи и работать с расписанием. Также реализована возможность работы с недавними записями раздела (10 записей), с которыми взаимодействовал пользователь.
- **Online**. Для online-режима необходимо наличие интернет-соединения. При использовании этого режима пользователь работает напрямую с сервером Creatio, в качестве которого выступает основное приложение. Синхронизация конфигурационных изменений выполняется автоматически в режиме реального времени.
- **Offline**. Для offline-режима наличие интернет-соединения требуется только для первичного импорта и синхронизации. При использовании этого режима данные сохраняются локально на мобильном

устройстве. Для получения конфигурационных изменений и актуализации данных необходимо вручную выполнять синхронизацию с сервером Creatio.

За режим работы мобильного приложения отвечает системная настройка [*Режим работы мобильного приложения*] ([*Mobile application operation mode*]) в Creatio. Если нужно изменить режим работы одновременно для всех пользователей мобильного приложения, необходимо установить нужное значение этой настройки без установленного свойства [*Персональная*]. Если же необходимо для разных пользователей указать разные режимы, то нужно эти изменения делать непосредственно пользователям, устанавливая свойство [*Персональная*]. У этих пользователей должны быть права на изменение системных настроек.



Синхронизация мобильного приложения с Creatio

В зависимости от режима работы приложения, синхронизация с сервером Creatio выполняет разные задачи. В случае online-режима синхронизация нужна только для получения изменений в конфигурации. А в случае offline-режима синхронизация необходима как для получений обновлений, так и для передачи или получения измененных или новых данных. Общая схема синхронизации для offline-режима:

Сначала приложение выполняет аутентификацию. При этом, выполняя logout, на сервере уничтожается текущая активная сессия. Далее у сервера запрашиваются данные для формирования пакета разницы. Приложение анализирует эти данные и запрашивает измененные или новые конфигурационные схемы. После загрузки схем приложение получает системные данные, к которым относятся кешируемые справочники (так называемые “простые” справочники), системные настройки и т. д. Затем идет обмен данными с сервером.

Отличие синхронизации в online-режиме заключается в том, что у нее нет последних двух этапов — экспорта и импорта.

На заметку. В версии мобильного приложения 7.8.6 реализован еще один этап синхронизации — “Актуализация данных”. Если эта функциональность включена, то данный этап выполняется последним, после экспорта и импорта данных. Суть этапа в следующем: приложение сравнивает доступные на сервере данные с локальными и, в случае наличия разницы, загружает недостающие данные или удаляет неактуальные. Этот механизм предусматривает ситуацию, которая возможна в случае перераспределения прав доступа или удаления данных на сервере. Для его включения в манифесте в секции `SyncOptions,` в свойстве `ModelDataImportConfig` для нужного объекта-модели установить значение `true` для свойства `IsAdministratedByRights`.

Экспорт данных в пакетном режиме

По умолчанию мобильное приложение отправляет каждое произведенное пользователем изменение данных поочередно. Таким образом одно изменение производит как минимум один запрос на сервер. При достаточно большом количестве изменений выполнение таких запросов может занять существенное время.

Начиная с версии 7.9, стало возможным отправлять данные в пакетном режиме (batch mode), что позволяет значительно ускорить отправку данных на сервер.

Для включения пакетного режима отправки данных необходимо в манифесте мобильного приложения в секции `SyncOptions` установить для свойства `UseBatchExport` значение `true`. В результате все пользовательские изменения будут сгруппированы в несколько пакетных запросов согласно типу операции, выполняемой пользователем. Возможные типы операций — вставка, обновление и удаление.

Жизненный цикл страниц в мобильном приложении

Во время навигации по мобильному приложению для каждой страницы выполняется ряд этапов — открытие, закрытие, выгрузка, возврат к странице и т. п. Время, прошедшее от загрузки страницы в память мобильного устройства и до ее окончательной выгрузки из памяти, принято называть жизненным циклом страницы.

Для каждого этапа жизненного цикла предусмотрены события страницы. Использование событий дает возможность расширять функциональность. К основным событиям относятся:

- инициализация представления;
- завершение инициализации класса;
- загрузка страницы;
- загрузка данных;
- закрытие страницы.

Понимание этапов выполнения жизненного цикла страницы позволяет качественно и максимально эффективно расширять логику страниц.

Этапы жизненного цикла

Важно. На экране телефона может отображаться только одна страница. На экране планшета — одна страница в портретной ориентации и две в ландшафтной. В связи с этим жизненный цикл страниц имеет отличия для телефона и планшета.

Открытие страницы

При первом открытии страницы выполняется загрузка скриптов, требуемых для ее работы. Далее инициализируется контроллер и создается представление.

События открытия страницы генерируются в следующей последовательности:

1. `initializeView` — инициализация представления.
2. `pageLoadComplete` — событие завершения загрузки страницы.

3. `launch` — инициирует загрузку данных.

Закрытие страницы

Во время закрытия страницы ее представление удаляется из объектной модели документа (Document object model, DOM), а контроллер удаляется из памяти устройства.

Закрытие страницы происходит в следующих случаях:

- Нажата кнопка [*Назад*]. В таком случае удаляется последняя страница.
- Выполнен переход в другой раздел. В таком случае удаляются все страницы, которые были открыты ранее.

Событие завершения закрытия страницы — `pageUnloadComplete`.

Выгрузка страницы

Выгрузка страницы происходит в случае, когда выполняется переход к другой странице в том же разделе. При этом текущая страница становится неактивной. Она может оставаться видимой на экране устройства. Например, если на планшете открыть страницу просмотра из реестра, то страница реестра останется видимой. В такой же ситуации на телефоне страница реестра не будет отображаться, но будет оставаться в памяти. Это и отличает выгрузку от закрытия страницы.

Событие выгрузки страницы — `pageUnloadComplete` (совпадает с событием закрытия страницы).

Возврат к странице

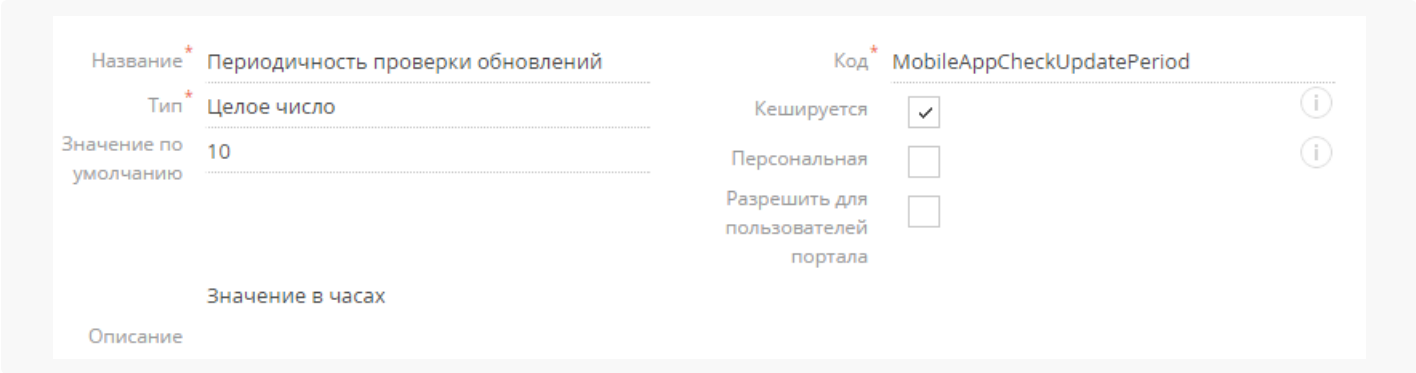
Возврат к выгруженной ранее странице происходит при нажатии на кнопку [*Назад*].

Событие возврата к странице — `pageLoadComplete`.

Важно. В приложении может использоваться только один экземпляр страницы. Поэтому, если последовательно открыть две одинаковые страницы, то при возврате к первой из них повторно выполняется обработчик события `launch`. Это следует учитывать при разработке.

Фоновое обновление конфигурации в мобильном приложении

В мобильном приложении Creatio реализован механизм синхронизации структуры приложения, который может работать в автоматическом фоновом режиме. Для управления этим процессом необходимо использовать системную настройку [*Периодичность проверки обновлений*] ([*Update checks frequency*]).



Эта настройка указывает по истечении какого времени (в часах) мобильное приложение может запросить изменения конфигурации у Creatio. Если настройке установить значение 0, то приложение будет всегда загружать обновления конфигурации.

Условия работы

Приложение запускает синхронизацию структуры в фоновом режиме только при соблюдении следующих условий:

- на мобильном устройстве используется платформа iOS или Android;
- синхронизация ранее не была запущена;
- с момента последней синхронизации структуры прошло больше времени, чем указано в системной настройке [*Периодичность проверки обновлений*] ([*Update checks frequency*]);
- осуществляется запуск приложения, или приложение активируется (т.е. если оно было ранее свернуто или в него переходят из другого приложения).

Если в ходе обновления структуры изменения были получены, то для применения полученных изменений приложение автоматически перезапустится когда пользователь свернет его или перейдет в другое приложение.

Особенности работы на разных платформах

1. На платформе Android фоновый режим реализован через параллельно запущенный сервис. Такой подход гарантирует, что запущенная синхронизация гарантировано завершится, даже если вручную выгрузить приложение из памяти устройства.
2. На платформе iOS для запуска синхронизации в фоновом режиме используется второй `webView`, в то время как само приложение работает в основном `webView`. Это гарантирует нормальную работу пользователя в приложении при одновременно запущенной синхронизации структуры.

   В отличие от реализации на платформе Android это не гарантирует завершения синхронизации на 100%, поскольку синхронизация может быть прервана при выгрузке приложения вручную либо если это сделает платформа iOS.

В приложении под iOS (начиная с версии 7.17.2) вместо `UIWebView` используется `WKWebView`, что привело к использованию фреймворка Cordova версии 6.1.1, а также минимально поддерживаемой версия iOS 11.

`WKWebView` имеет следующие особенности:

- Не допускается использовать абсолютные пути (для ресурсов, скриптов, iframe и т.п.).
- Не допускается использовать кросс-доменные ссылки (для ресурсов, скриптов, iframe и т.п.).
- Данные `localStorage` не сохранятся при переходе на `WKWebView`.
- Не рекомендуется использование iframe.

Изменения в iOS WebView в Apache Cordova подробнее описаны в документации Cordova.

Если необходимо вставить на страницу ссылку на локальный файл, то необходимо конвертировать путь к нему через метод `Terrasoft.util.toUrlScheme`.

Вставка на страницу ссылки на локальный файл

```
this.element.setStyle('background-image', 'url("' + Terrasoft.util.toUrlScheme(value) +'")');
```

Пример ссылок представлен ниже.

Недопустимая ссылка

```
file:///var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C271D9871/Documents/BP
```

Ссылка после конвертации

```
app://localhost/_app_file_/var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C27
```

Использование плагина `inappbrowser` имеет следующие особенности:

- Все пути должны быть относительными и находиться внутри корневой папки сайта.

```
<img src="images/2.jpg">
```

- Не допускается использовать кросс-доменные ссылки (для ресурсов, скриптов, iframe и т.п.).

```
<img src="https://www.worldometers.info/img/worldometers-logo.gif">
<img src="file:///var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C271D9871
```

- При открытии сайта необходимо указывать абсолютный путь.

```
cordova.InAppBrowser.open("file:///var/mobile/Containers/Data/Application/DE..9871/Documents/
```

Открытие сайта через плагин `inappbrowser` имеет особенности, аналогичные особенностям `WKWebView`.

Автоматическое разрешение конфликтов при синхронизации

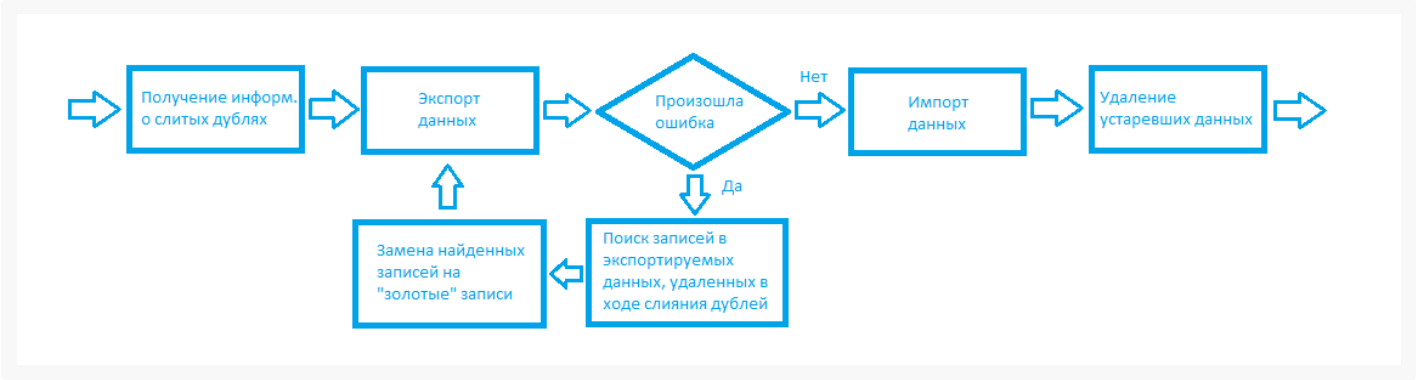
В ходе синхронизации мобильного приложения, работающего в offline-режиме, могут возникать ситуации, когда переданные в Creatio данные не могут быть сохранены по ряду причин. К таким причинам могут относиться следующие:

- Запись была слита в Creatio с другой дублирующейся записью, поэтому ее не существует.
- Запись была удалена из Creatio.

Каждая из приведенных выше ситуаций обрабатывается мобильным приложением автоматически.

Слияние дублей

Алгоритм разрешения конфликта, возникающего по причине использования данных, которые в Creatio были удалены в ходе процедуры слияния дублей:



Как видно на схеме, в ходе синхронизации приложение сначала забирает на сервере информацию о том, по каким записям с момента последней синхронизации производилось слияние дублей. А именно какие записи были удалены и какие записи их заменили. Если в ходе экспорта не было никаких ошибок, то далее выполняется импорт. Если же произошла ошибка, связанная с исключением внешнего ключа (`Foreign Key Exception`), или ошибка, связанная с тем, что на сервере не была найдена какая-то из записей (`Item Not Found Exception`), то выполняется процедура разрешения этого конфликта со следующими этапами:

- В экспортируемых данных ищутся колонки, содержащие “старую” запись.
- В найденных колонках “старая” запись заменяется новой, в которой данные объединялись.

После этого запись повторно отправляется в Creatio. Как только заканчивается импорт и появляется информация о слитых дублях, локально производится удаление “старых” записей.

Запись не найдена

В случае, когда сервер возвращает ошибку, свидетельствующую о том, что измененная пользователем запись в Creatio не найдена, приложение выполняет следующие действия:

1. Проверяет наличие записи в списке записей, удаленных в ходе слияния дублей.

2. Если в списке удаленных записи нет, то приложение удаляет ее локально.
3. Удаляет информацию по этой записи из лога синхронизации.

Таким образом, приложение считает этот конфликт разрешенным и продолжает экспорт данных.

Класс BaseConfigurationPage JS

Сложный

Классы контроллеров страниц наследуются от класса `Terrasoft.controller.BaseConfigurationPage`, который предоставляет методы обработки событий жизненного цикла.

Методы

`initializeView(view)`

Вызывается после того как было создано (но еще не было отрисовано) представление страницы в DOM. На этом этапе можно подписываться на события классов представления, выполнять дополнительные манипуляции с DOM.

`pageLoadComplete(isLaunch)`

Предоставляет возможность расширения логики, которая выполняется как при загрузке страницы, так и при возврате. Значение параметра `isLaunch` равное `true` указывает на то, что страница загружается первый раз.

`launch()`

Вызывается только при открытии страницы. Метод инициирует начало загрузки данных. Если требуется загрузка дополнительных данных, то правильно будет делать это в методе `launch()`.

`pageUnloadComplete()`

Предоставляет возможность расширения логики, которая выполняется как при закрытии страницы, так и при ее выгрузке.

Класс PageNavigator JS

Сложный

Управлением жизненным циклом страниц занимается класс `Terrasoft.PageNavigator`. Класс предоставляет возможности открытия и закрытия страниц, обновления неактуальных данных, а также хранения истории открытых страниц.

Методы

`forward(openingPageConfig)`

Метод открывает страницу с учетом свойств конфигурационного объекта-параметра `openingPageConfig`. Основные свойства этого объекта представлены в таблице.

Свойства объекта `openingPageConfig`

`controllerName`	Имя класса контроллера.
`viewXType`	Тип представления по xtype.
`type`	Тип страницы из перечисления `Terrasoft.core.enums.PageType`.
`modelName`	Имя модели страницы.
`pageSchemaName`	Название схемы страницы в конфигурации.
`isStartPage`	Признак, указывающий на то, что страница должна быть первой. Если до этого уже были открыты страницы, то они будут закрыты.
`isStartRecord`	Признак, указывающий на то, что страница карточки просмотра/редактирования должна быть первой после реестра. Если есть другие открытые страницы после реестра, они закрываются.
`recordId`	Идентификатор записи открываемой страницы.
`detailConfig`	Настройки стандартной детали.

`backward()`

Метод закрывает страницу.

`markPreviousPagesAsDirty(operationConfig)`

Метод отмечает все предыдущие страницы как неактуальные. После возврата к предыдущим страницам для каждой из них вызовется метод `refreshDirtyData()`, который выполняет повторную загрузку данных или актуализирует данные на основании объекта `operationConfig`.

`refreshPreviousPages(operationConfig, currentPageHistoryItem)`

Метод выполняет для всех предыдущих страниц повторную загрузку данных или актуализирует данные на основании `operationConfig`. Если установлено значение для параметра

`currentPageHistoryItem`, метод выполняет те же действия для предшествующих страниц.

`refreshAllPages(operationConfig, excludedPageHistoryItems)`

Метод выполняет для всех страниц повторную загрузку данных или актуализирует данные на основании `operationConfig`. Если установлен параметр `excludedPageHistoryItems`, метод исключает из актуализации указанные страницы.

Класс Router JS

Сложный

Маршрутизация используется для управления визуальными компонентами: страницами, пикерами и др. Маршрут имеет три состояния:

1. `Load` — выполняет открытие текущего маршрута.
2. `Unload` — выполняет закрытие текущего маршрута при возврате.
3. `Reload` — выполнят восстановление предыдущего маршрута при возврате.

Для маршрутизации используется класс `Terrasoft.Router` и его основные методы `add()`, `route()`, `back()`.

Методы

`add(name, config)`

Добавляет маршрут.

Параметры

`name`	Уникальное имя маршрута. В случае повторного добавления, последний переопределит предыдущий.
`config`	Описывает имена функций обработчиков состояний маршрута. В свойстве `handlers` устанавливаются обработчики состояний маршрута.

Пример использования

```
Terrasoft.Router.add("record", {
  handlers: {
     load: "loadPage",
     reload: "reloadPage",
     unload: "unloadLastPage"
  }
```

```
});
```

`route(name, scope, args, config)`

Выполняет открытие маршрута.

Параметры

`name`	Имя маршрута.
`scope`	Контекст функции обработчиков состояний.
`args`	Параметры функций обработчиков состояний.
`config`	Дополнительные параметры маршрута.

Пример использования

```
var mainPageController = Terrasoft.util.getMainController();
Terrasoft.Router.route("record", mainPageController, [{pageSchemaName: "MobileActivityGridPag
```

`back()`

Выполняет закрытие текущего маршрута и восстановление предыдущего.