bpm'online

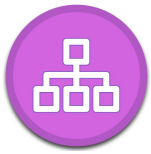
Документация по разработке bpm'online mobile

Будущее создавать просто

Содержание

1 Знакомство с платформой bpm'online mobile

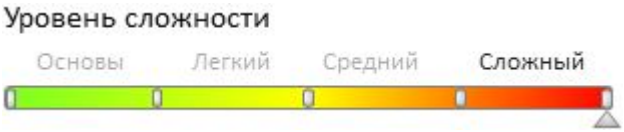
Содержание



Архитектура мобильного приложения (Section 1.1)

Архитектура, общая схема и режимы работы мобильного приложения bpm'online.

1.1 Архитектура мобильного приложения



Общие положения

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.

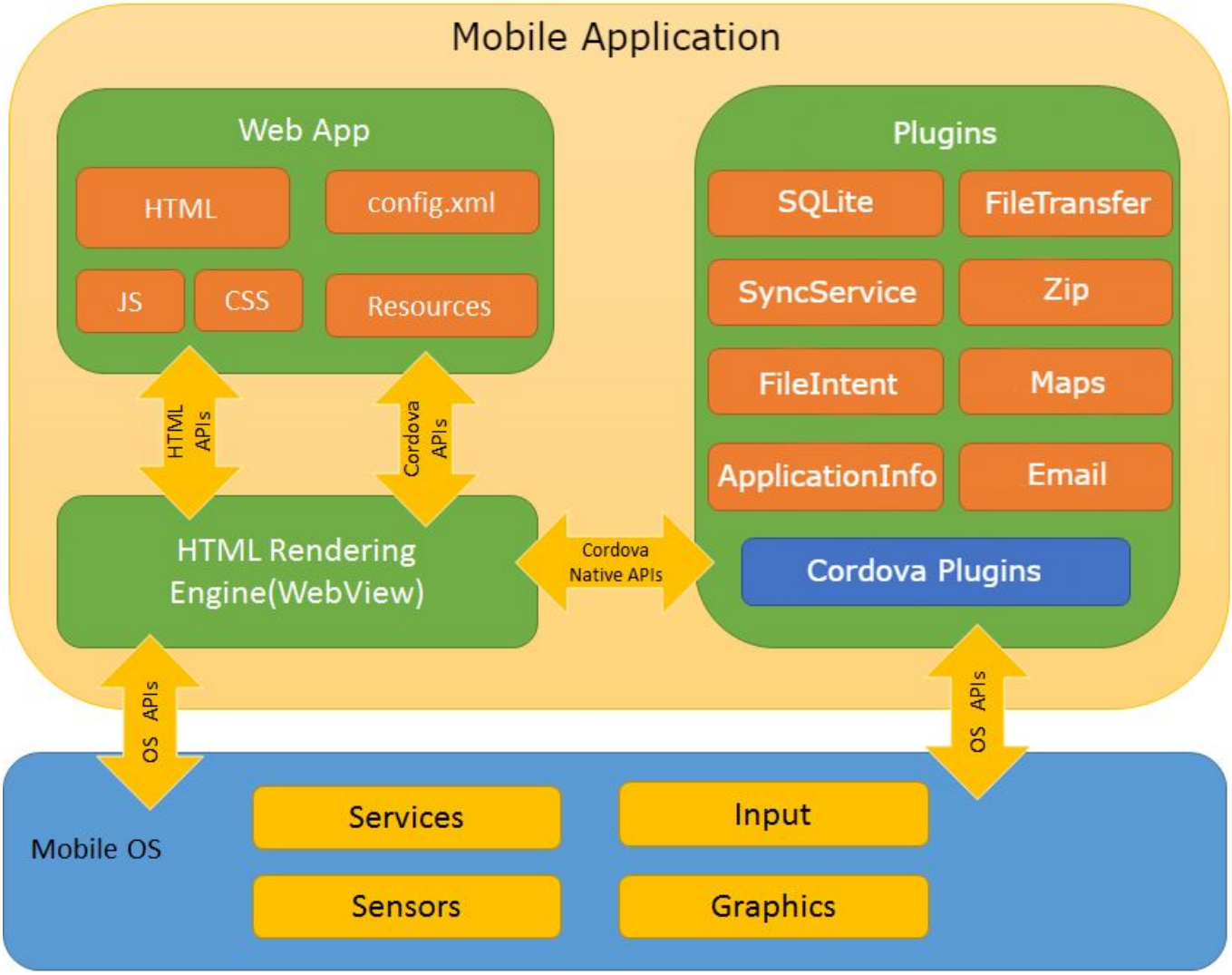
Гибридное приложение — мобильное приложение, "упакованное" в native-оболочку. Такое приложение, как и native, устанавливается из онлайн-магазина и имеет доступ к тем же возможностям мобильного устройства, но разрабатываетсяс помощью web-языков HTML5, CSS и JavaScript. В отличие от native-приложения является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

Общие сведения по установке и синхронизации мобильного приложения, а также специфика работы в онлайн и оффлайн режимах приведены в документации bpm'online.

Архитектура мобильного приложения bpm'online

В общем виде архитектура мобильного приложения bpm'online представлена на рис. 1.

Рис. 1. — Архитектура мобильного приложения



Для создания гибридных приложений, воспринимаемых мобильным устройством как native, мобильное приложение использует возможности фреймворка Cordova. Фреймворк Cordova предоставляет доступ к программному интерфейсу мобильного устройства (API) для взаимодействия с базой данных или оборудованием, например, камерой или картой памяти. Также Cordova предоставляет т. н. 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 можно найти здесь.

Ядро мобильного приложения предоставляет унифицированный интерфейс для взаимодействия всех остальных клиентских частей приложения. Используемые ядром Javascript-файлы условно можно разделить на две следующие категории:

1. Базовые:

- MVC-компоненты (представления страниц, контроллеры, модели);
- модули синхронизации (импорт\экспорт данных, импорт метаданных, импорт файлов и т. д.);
- клиентские классы веб-сервисов;
- классы, предоставляющие доступ к native-плагинам.

Базовые скрипты содержатся в сборке приложения, публикуемой в магазине приложений.

2. Конфигурационные:

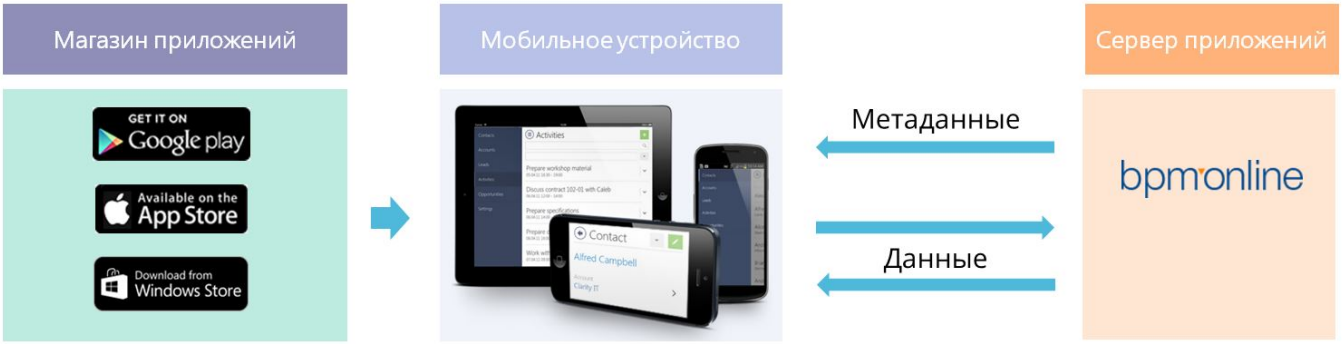
- манифест;
- схемы и настройки разделов.

Конфигурационные файлы приложение получает в ходе синхронизации с сервером bpm'online и сохраняет локально в файловой системе устройства.

Общая схема работы мобильного приложения bpm'online

Опубликованное в магазине мобильное приложение bpm'online 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 (рис. 2):

Рис. 2. — Общая схема работы мобильного приложения



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

Такая схема работы дает очевидое преимущество — мобильное приложение совместимо со всеми существующими продуктами bpm'online. 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 bpm'online.

Режимы работы мобильного приложения bpm'online

Мобильное приложение может работать в двух режимах:

- с подключением к основному приложению (online);
- без подключения к основному приложению (offline).

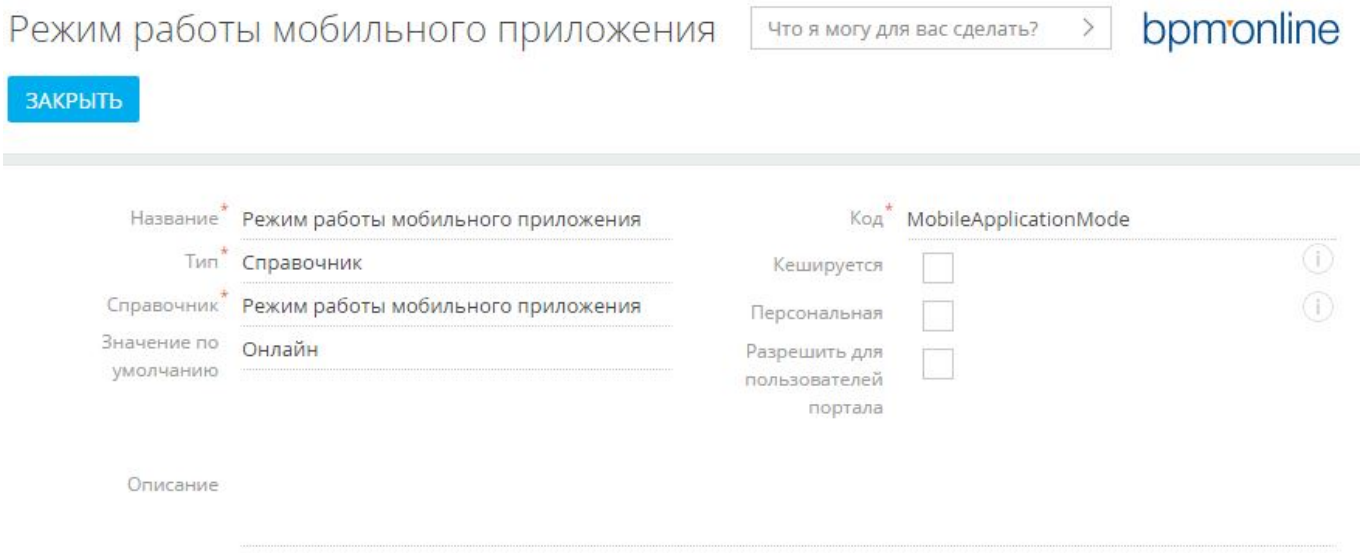
Разница между этими режимами отображена в таблице 1:

Табл. 1. — Сравнение режимов работы мобильного приложения

Online	Offline
Необходимо наличие соединения с интернетом.	Наличие соединения с интернетом необязательно. Необходимо только для первичного импорта и синхронизации.
Пользователь работает напрямую с сервером bpm'online.	Данные сохраняются локально на мобильном устройстве.
Синхронизацию необходимо выполнять только при конфигурационных изменениях (добавление или удаление колонки, изменение логики работы).	Синхронизацию необходимо выполнять регулярно для актуализации данных и для получения конфигурационных изменений.

За режим работы мобильного приложения отвечает системная настройка [Режим работы мобильного приложения] в bpm'online. Если нужно изменить режим работы одновременно для всех пользователей мобильного приложения, необходимо установить нужное значение этой настройки без установленного свойства [Персональная] (рис. 3). 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.

Рис. 3. — Системная настройка [Режим работы мобильного приложения]



Синхронизация мобильного приложения с bpm'online

В зависимости от режима работы приложения, синхронизация с сервером bpm’online 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 представлена на рис. 4:

Рис. 4. — Общая схема синхронизации (offline-режим)

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*.

2 С чего начать разработку

Содержание

Отладка мобильного приложения (Section 2.1)

Во время разработки пользовательских решений для мобильного приложения bpm'online необходимо многократно выполнять проверку правильности работы новой функциональности — отладку приложения. Как это сделать с помощью инструментов разработки современных браузеров читайте в этой статье.

2.1 Отладка мобильного приложения

Общие сведения

Во время разработки пользовательских решений для мобильного приложения bpm'online необходимо многократно выполнять проверку правильности работы новой функциональности — *отладку приложения*.

Поскольку мобильное приложение bpm'online является **приложением гибридного типа (Section 1.1)** — мобильным web-приложением, "упакованным" в native-оболочку, то существует возможность его отладки средствами инструментов разработчика браузера Google Chrome в режиме мобильного устройства. Подробнее об отладке клиентского кода приложения средствами инструментов разработчиков современных браузеров можно узнать в статье "Отладка клиентского кода" документации по разработке.

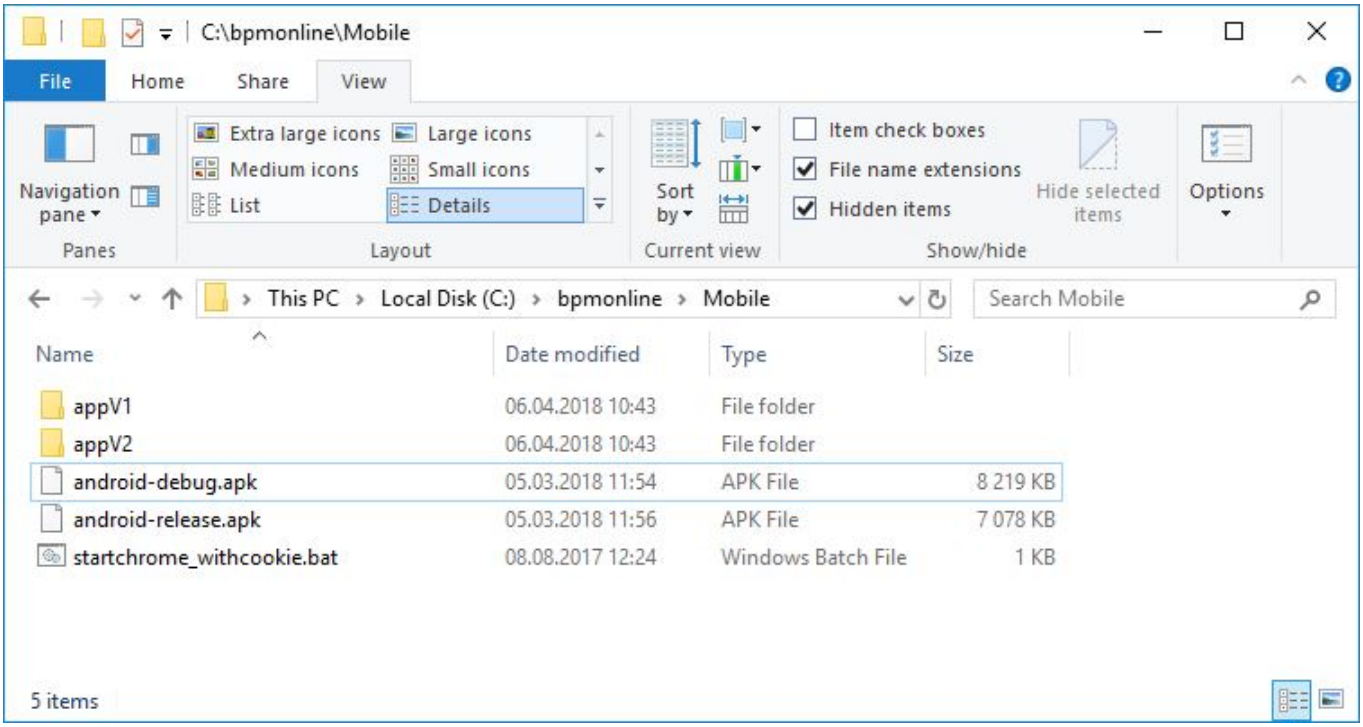
Последовательность запуска мобильного приложения в режиме отладки:

1. Получить необходимые для отладки мобильного приложения файлы.
2. Запустить пакетный файл *startchrome_withcookie.bat*.
3. В браузере Google Chrome перейти в режим отладки мобильного устройства.
4. Внести необходимые настройки и синхронизировать мобильное приложение bpm'online.

Получение необходимых файлов

Для получения необходимых для отладки мобильного приложения файлов нужно обратиться в службу поддержки. Служба поддержки предоставит архив, содержащий нужные файлы. Архив необходимо распаковать в произвольный каталог, например, *C:\bpmonline\Mobile* (рис. 1).

Рис. 1. — Содержимое распакованного архива



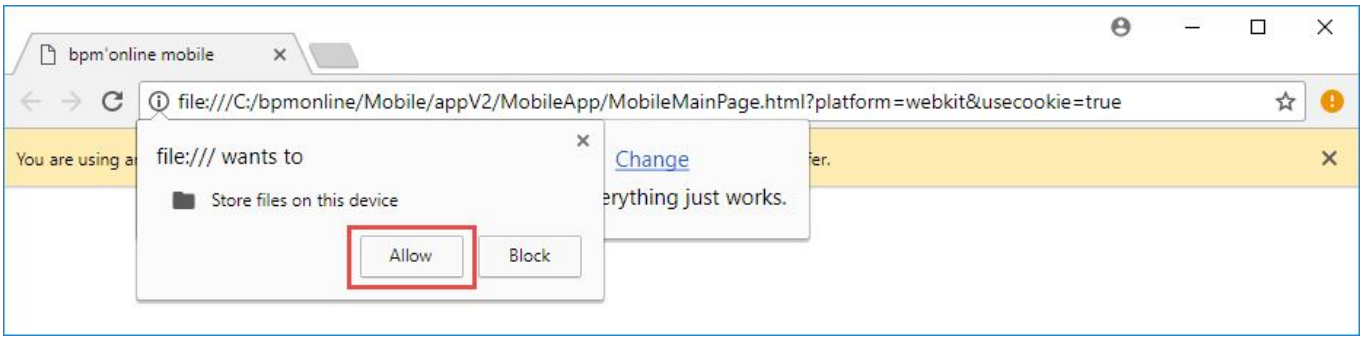
Запуск пакетного файла startchrome_withcookie.bat

> ВАЖНО
>
> Прежде чем запустить пакетный файл *startchrome_withcookie.bat* на выполнение, необходимо закрыть все экземпляры браузера Google Chrome.

Пакетный файл *startchrome_withcookie.bat* размещен в корневом каталоге распакованного архива. После выполнения команд пакетного файла запустится браузер Google Chrome.

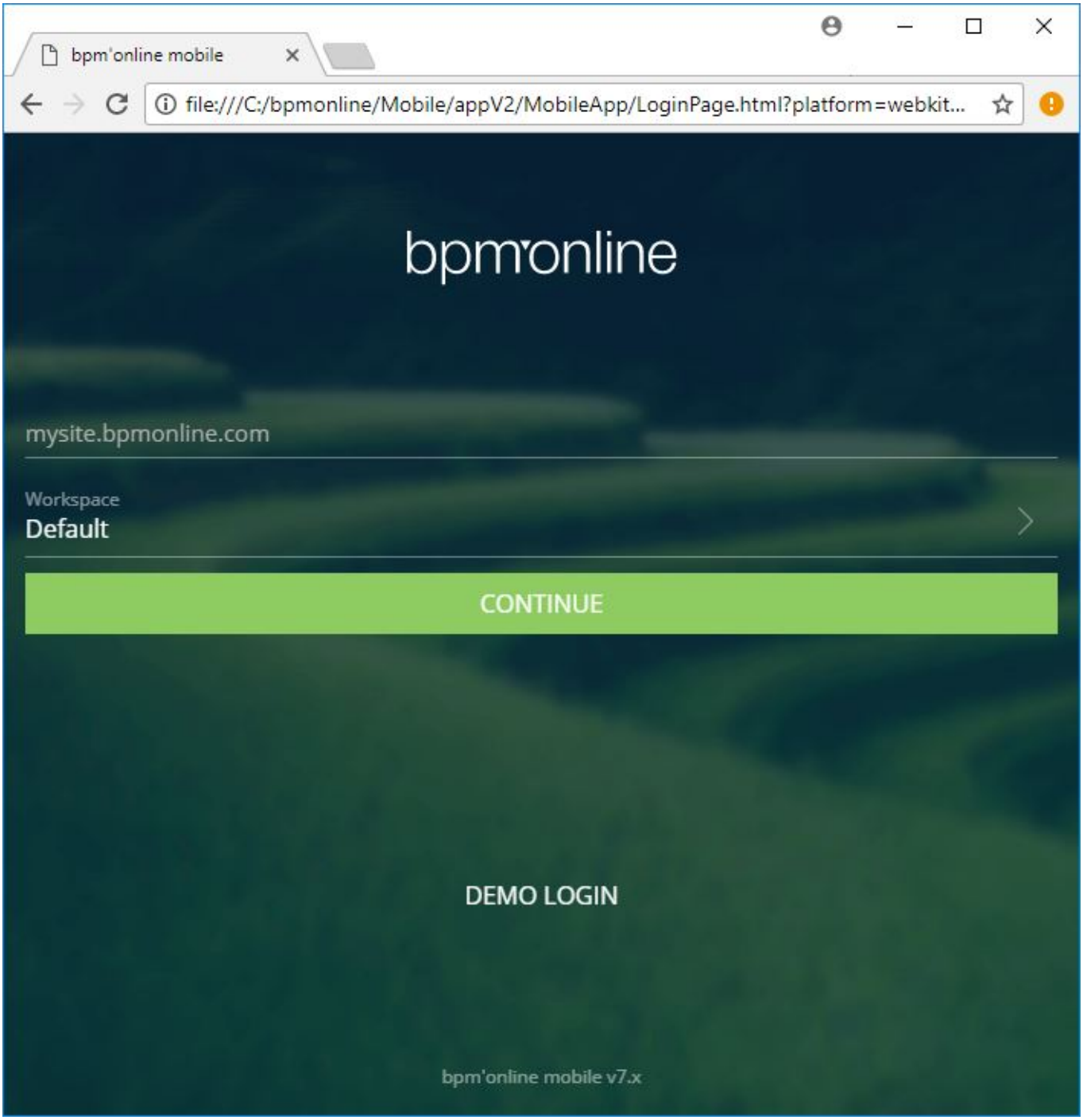
При первом запуске браузера с помощью пакетного файла отобразится информационное окно, предупреждающее о сохранении файлов в файловую систему (рис. 2). Нужно разрешить сохранение файлов. Также можно закрыть предупреждение о неподдерживаемом флаге *--disable-web-security* (рис. 2).

Рис. 2. — Информационное окно с предупреждением



После выполнения команд пакетного файла *startchrome_withcookie.bat* запустится браузер Google Chrome с открытой страницей настройки мобильного приложения bpm'online (рис. 3).

Рис. 3. — Страница настройки мобильного приложения



Переход в режим отладки мобильного устройства

Чтобы вызвать инструментарий разработчика в браузере Google Chrome, необходимо нажать функциональную клавишу *F12* клавиатуры или комбинацию клавиш *Ctrl* + *Shift* + *I*. Отладку локальной версии мобильного приложения можно выполнять средствами браузера. Подробнее об отладке клиентского кода приложения средствами инструментов разработчиков современных браузеров можно узнать в статье "Отладка клиентского кода" документации по разработке.

> ⚠ К СВЕДЕНИЮ
>
> После перехода в режим отладки мобильного устройства необходимо обновить страницу отображения, нажав клавишу F5.

Ввод настроек мобильного приложения и синхронизация

При первом входе в мобильное приложение на странице настроек необходимо ввести http-адрес приложения bpm'online, для которого необходимо выполнить отладку, и нажать на кнопку [Далее] ([Continue]) (рис. 4). Затем необходимо ввести имя пользователя и пароль (рис. 5).

Рис. 4. — Страница настроек локальной версии мобильного приложения

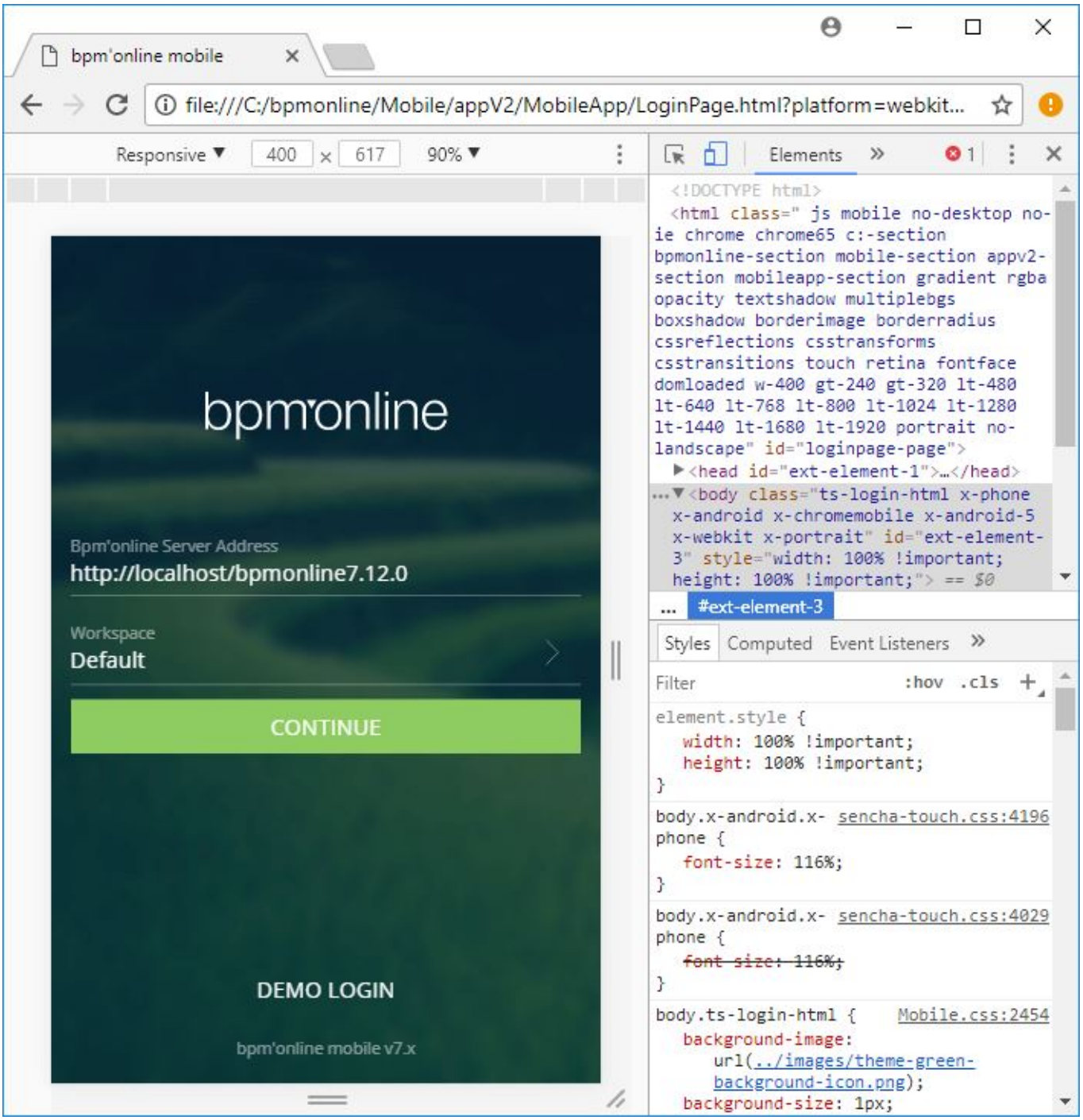
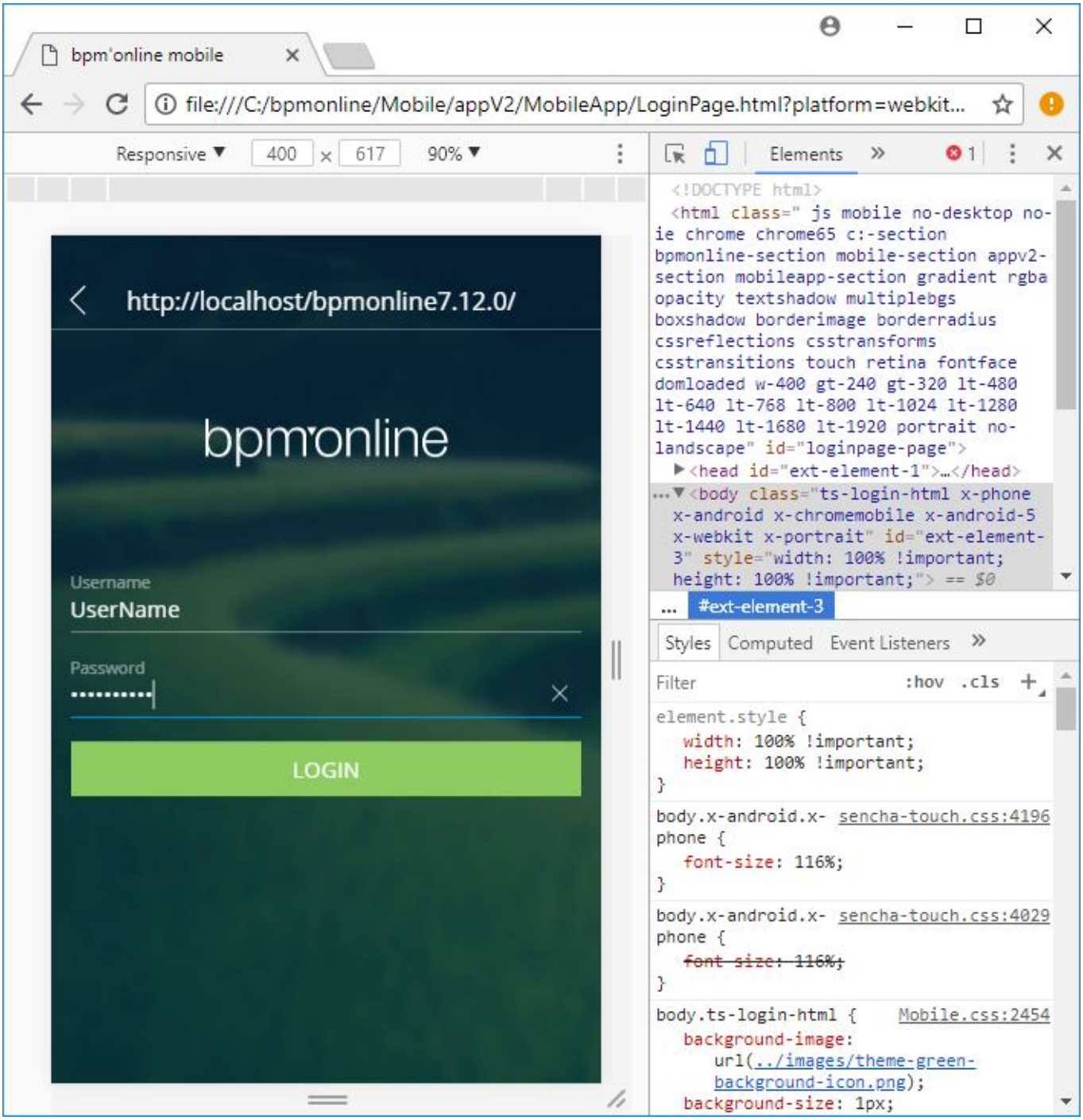


Рис. 5. — Страница авторизации



После выполнения настроек и входа в приложение локальная версия мобильного приложения будет вести себя аналогично приложению, только что установленному в мобильное устройство. При этом native-функции мобильного устройства, например, работа с камерой, загрузка файлов и др., поддерживаться не будут. О том, как работать с мобильным приложением bpm'online, можно ознакомиться в документации bpm'online mobile.

3 Описание платформы

Содержание

Манифест мобильного приложения (Section 3.1)

Манифест мобильного приложения описывает структуру всего мобильного приложения — его объекты и связи между ними.

- **Манифест. Свойства интерфейса приложения (Section 3.1.1)**
- **Манифест. Свойства данных и бизнес-логики (Section 3.1.2)**
- **Манифест. Свойства синхронизации приложений (Section 3.1.3)**
- **Экспорт данных в пакетном режиме (Section 3.1.4)**

Жизненный цикл страниц в мобильном приложении (Section 3.2)

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.

Фоновое обновление конфигурации в мобильном приложении (Section 3.3)

В мобильном приложении bpm'online реализован механизм синхронизации структуры приложения, который может работать в автоматическом фоновом режиме.

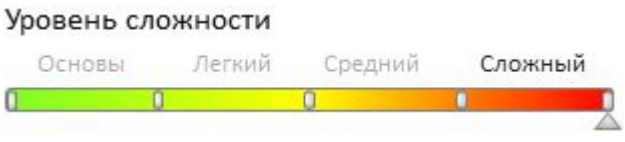
Получение настроек и данных раздела [Итоги] (Section 3.4)

Функциональность получения настроек и данных по дашбордам реализована в сервисе *AnalyticsService* и в утилитном классе *AnalyticsServiceUtils* пакета *Platform*.

Автоматическое разрешение конфликтов при синхронизации (Section 3.5)

В ходе синхронизации мобильного приложения, работающего в offline-режиме, могут возникать ситуации, когда переданные в bpm'online данные не могут быть сохранены по ряду причин.

3.1 Манифест мобильного приложения



Общие положения

Манифест мобильного приложения описывает структуру всего мобильного приложения — его объекты и связи между ними. Базовая версия мобильного приложения bpm'online описывается манифестом, который содержится в схеме *MobileApplicationManifestDefaultWorkspace* пакета *Mobile*.

В процессе доработки мобильного приложения пользователями создаются новые разделы и страницы. Все они должны быть зарегистрированы в манифесте для того, чтобы приложение могло с ними работать. Так как у сторонних разработчиков нет возможности вносить изменения в манифест базового приложения, то

при регистрации пользовательских разделов и страниц при помощи мастера мобильных приложений система автоматически создает пользовательский манифест, в котором в заданном формате описаны все взаимосвязи созданных объектов. Название схемы манифеста формируется по маске MobileApplicationManifest[Название рабочего места]. Так, например, для рабочего места [Полевые продажи], система сформирует название схемы манифеста MobileApplicationManifestFieldForceWorkspace, а для рабочего места [Основное рабочее место] — название MobileApplicationManifestDefaultWorkspace.

Структура манифеста мобильного приложения

Манифест мобильного приложения — это конфигурационный объект, с помощью свойств которого описывается структура мобильного приложения. Перечень и назначение свойств конфигурационного объекта манифеста приведены в таблице 1.

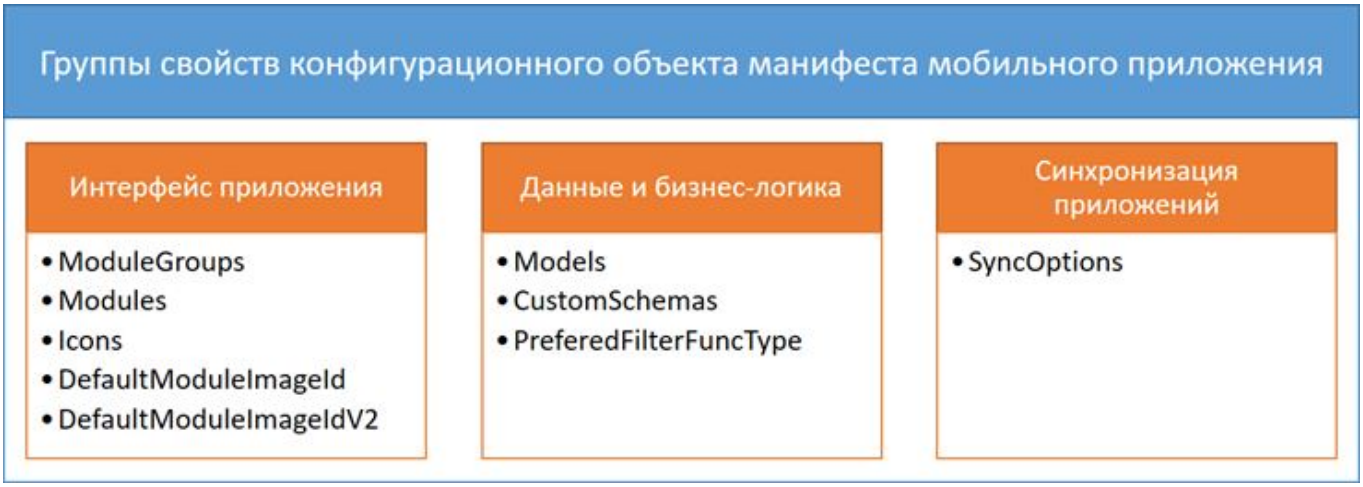
Табл. 1. Свойства конфигурационного объекта манифеста.

Свойство	Назначение
ModuleGroups	Содержит верхнеуровневую настройку групп главного меню.
Modules	Описывает свойства модулей мобильного приложения.
SyncOptions	Описывает параметры для настройки синхронизации данных.
Models	Содержит конфигурацию импортируемых моделей приложения.
PreferedFilterFuncType	Определяет операцию, которая будет использоваться при поиске и фильтрации данных.
CustomSchemas	Подключает к мобильному приложению дополнительные схемы.
Icons	Позволяет добавить в приложение пользовательские изображения.
DefaultModuleImageId	Устанавливает изображение по умолчанию для пользовательского интерфейса V1.
DefaultModuleImageIdV2	Устанавливает изображение по умолчанию для пользовательского интерфейса V2.

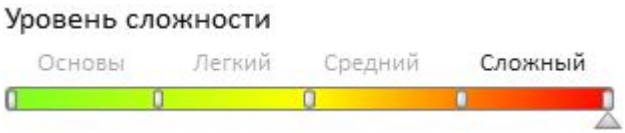
Все свойства конфигурационного объекта манифеста условно можно разделить на три группы (рис. 1):

- *Свойства интерфейса приложения* — содержит свойства, с помощью которых формируется интерфейс мобильного приложения. При помощи свойств этой группы происходит формирование разделов приложения, главного меню, настраиваются пользовательские изображения. Подробнее о свойствах, входящих в эту группу можно узнать в статье "**Манифест. Свойства интерфейса приложения (Section 3.1.1)** .
- *Свойства данных и бизнес-логики* — содержит свойства, в которых описываются импортируемые данные а также пользовательская бизнес-логика обработки этих данных в мобильном приложении. Подробнее о свойствах, входящих в эту группу можно узнать в статье "**Манифест. Свойства данных и бизнес-логики (Section 3.1.2)**
- *Свойства синхронизации приложений* — содержит единственное свойство настройки синхронизации данных с основным приложением. Подробнее об этом свойстве можно узнать в статье **Манифест. Свойства синхронизации приложений (Section 3.1.3)** ".

Рис. 1. — Группы свойств конфигурационного объекта манифеста



3.1.1 Манифест. Свойства интерфейса приложения



Общие положения

Условная группа свойств конфигурационного объекта манифеста. Содержит свойства, с помощью которых формируется интерфейс мобильного приложения. При помощи свойств этой группы происходит формирование разделов приложения, главного меню, настраиваются пользовательские изображения. Подробнее о манифесте мобильного приложения и всех его свойствах можно узнать в статье "**Манифест мобильного приложения (Section 3.1)**".

Свойство ModuleGroups

Группы модулей приложения. Описывает верхнеуровневую настройку групп главного меню мобильного приложения. Для каждой группы меню задается списком именованных конфигурационных объектов с единственным возможным свойством Position (см. табл. 1).

Табл. 1. Свойство конфигурационного объекта для настройки группы меню.

Свойство	Значение
Position	Позиция группы в главном меню. Начинается с 0.

Пример

Настройка меню мобильного приложения, состоящего из двух групп — основной группы и группы [Продажи].

```
// Групы модулей мобильного приложения.
"ModuleGroups": {
    // Настройка группы основного меню.
    "main": {
        // Позиция группы в главном меню.
        "Position": 0
    },
    // Настройка группы меню [Продажи].
    "sales": {
        // Позиция группы в главном меню.
        "Position": 1
    }
```

```
}
```

Свойство Modules

Модули мобильного приложения. Модуль представляет собой раздел приложения. Каждый модуль в свойстве [Modules] конфигурационного объекта манифеста описывается конфигурационным объектом со свойствами, приведенными в таблице 2. Имя конфигурационного объекта раздела должно совпадать с названием модели, которая предоставляет данные раздела.

Табл. 2. Свойства конфигурационного объекта раздела.

Свойство	Значение
Group	Группа меню приложения, в которой размещается раздел. Задается строкой с названием соответствующего раздела меню из свойства *ModuleGroups* конфигурационного объекта манифеста.
Model	Название модели, которая предоставляет данные раздела. Задается строкой с названием одной из моделей, объявленных в свойстве *Models* конфигурационного объекта манифеста.
Position	Позиция раздела в группе главного меню. Задается числовым значением, начиная с 0.
Title	Заголовок раздела. Строка с названием локализованного значения заголовка раздела. Локализованное значение заголовка раздела должно быть добавлено в блок *[LocalizableStrings]* схемы манифеста.
Icon	Свойство, предназначенное для подключения пользовательского изображение к разделу в меню пользовательского интерфейса версии 1.
IconV2	Свойство, предназначенное для подключения пользовательского изображение к разделу в меню пользовательского интерфейса версии 2.
Hidden	Признак, отображается ли данный раздел в меню (*true* — скрыт, *false* — отображается). Необязательное свойство. По умолчанию — *false*.

Пример

Настроить разделы приложения следующим образом:

1. Разделы основного меню: [Контакты], [Контрагенты].
2. Стартовая страница приложения: раздел [Контакты].

В блоке *[LocalizableStrings]* схемы манифеста должны быть созданы строки содержащие заголовки разделов:

- ContactSectionTitle со значением "Контакты".
- AccountSectionTitle со значением "Контрагенты".

```
// Модули мобильного приложения.
"Modules": {
    // Раздел "Контакт".
    "Contact": {
        // Группа меню приложения, в которой размещается раздел.
        "Group": "main",
        // Название модели, которая предоставляет данные раздела.
        "Model": "Contact",
        // Позиция раздела в группе меню.
        "Position": 0,
        // Заголовок раздела.
        "Title": "ContactSectionTitle",
        // Подключение пользовательского изображения к разделу.
        "Icon": {
            // Уникальный идентификатор изображения.
            "ImageId": "4c1944db-e686-4a45-8262-df0c7d080658"
```

```
        },
        // Подключение пользовательского изображения к разделу.
        "IconV2": {
            // Уникальный идентификатор изображения.
            "ImageId": "9672301c-e937-4f01-9b0a-0d17e7a2855c"
        },
        // Признак отображения в меню.
        "Hidden": false
    },
    // Раздел "Контрагент".
    "Account": {
        // Группа меню приложения, в которой размещается раздел.
        "Group": "main",
        // Название модели, которая предоставляет данные раздела.
        "Model": "Account",
        // Позиция раздела в группе меню.
        "Position": 1,
        // Заголовок раздела.
        "Title": "AccountSectionTitle",
        // Подключение пользовательского изображения к разделу.
        "Icon": {
            // Уникальный идентификатор изображения.
            "ImageId": "c046aa1a-d618-4a65-a226-d53968d9cb3d"
        },
        // Подключение пользовательского изображения к разделу.
        "IconV2": {
            // Уникальный идентификатор изображения.
            "ImageId": "876320ef-c6ac-44ff-9415-953de17225e0"
        },
        // Признак отображения в меню.
        "Hidden": false
    }
}
```

Свойство Icons

Свойство предназначено для подключения к мобильному приложению пользовательских изображений.

Задается массивом конфигурационных объектов, каждый из которых имеет свойства, приведенные в таблице 3.

Табл. 3. Свойства конфигурационного объекта для подключения пользовательского изображения.

Свойство	Значение
ImageListId	Идентификатор списка изображений.
ImageId	Идентификатор подключаемого изображения из списка *ImageListId*.

Пример

```
// Подключение пользовательских изображений.
"Icons": [
    {
        // Идентификатор списка изображений.
        "ImageListId": "69c7829d-37c2-449b-a24b-bcd7bf38a8be",
        // Идентификатор подключаемого изображения.
        "ImageId": "4c1944db-e686-4a45-8262-df0c7d080658"
    }
]
```

Свойства DefaultModuleImageId и DefaultModuleImageIdV2

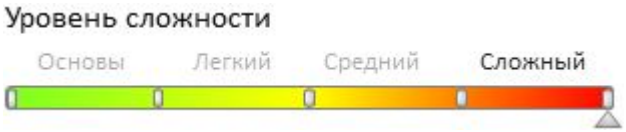
Свойства предназначены для установки уникальных идентификаторов изображений по умолчанию для

вновь создаваемых разделов или для разделов, у которых не указаны идентификаторы изображений в свойствах *Icon* или *IconV2* свойства *Modules* конфигурационного объекта манифеста.

Пример

```
//Идентификатор изображения по умолчанию для пользовательского интерфейса V1.
"DefaultModuleImageId": "423d3be8-de6b-4f15-a81b-ed454b6d03e3",
//Идентификатор изображения по умолчанию для пользовательского интерфейса V2.
"DefaultModuleImageIdV2": "1c92d522-965f-43e0-97ab-2a7b101c03d4"
```

3.1.2 Манифест. Свойства данных и бизнес-логики



Общие положения

Условная группа свойств конфигурационного объекта манифеста. Содержит свойства, в которых описываются импортируемые данные, а также пользовательская бизнес-логика обработки этих данных в мобильном приложении. Подробнее о манифесте мобильного приложения и всех его свойствах можно узнать в статье "**Манифест мобильного приложения (Section 3.1)**".

Свойство Models

Содержит импортируемые модели приложения. Каждая модель в свойстве описывается конфигурационным объектом с соответствующим именем. Свойства конфигурационного объекта модели представлены в табл. 1.

Табл. 1. Свойства конфигурационного объекта модели.

Свойство	Значение
Grid	Название схемы страницы реестра модели. Страница будет сгенерирована автоматически с именем Mobile[Название_модели][Тип_страницы]Page. Не обязателен для заполнения.
Preview	Название схемы страницы просмотра элемента модели. Страница будет сгенерирована автоматически с именем Mobile[Название_модели] [Тип_страницы]Page. Не обязателен для заполнения.
Edit	Название схемы страницы редактирования элемента модели. Страница будет сгенерирована автоматически с именем Mobile[Название_модели] [Тип_страницы]Page. Не обязателен для заполнения.
RequiredModels	Названия моделей, от которых зависит данная модель. Необязательное свойство. Здесь перечисляются все модели, колонки которых добавляются в текущую модель, а также колонки, на которые у текущей модели есть внешние ключи.
ModelExtensions	Расширения модели. Необязательное свойство. Представляет собой массив названий схем, в которых реализуются дополнительные настройки для модели (например, добавление в модель бизнес-правил, событий, значений по умолчанию для полей и т.д.).
PagesExtensions	Расширения страниц модели. Необязательное свойство. Представляет собой массив названий схем, в которых реализуются дополнительные настройки для различных типов страниц модели (добавление деталей, установка заголовков и т.д.).

Пример

Добавить в манифест конфигурацию следующих моделей:

1. Контакт — указать названия схем страниц реестра, просмотра и редактирования, требуемые модели, модули расширения модели и страниц модели.

2. Адрес контакта — указать только модуль расширения модели.

Свойство *Models* конфигурационного объекта манифеста должно выглядеть следующим образом:

```
// Импортируемые модели.
"Models": {
    // Модель "Контакт"
    "Contact": {
        // Схема страницы реестра.
        "Grid": "MobileContactGridPage",
        // Схема страницы просмотра.
        "Preview": "MobileContactPreviewPage",
        // Схема страницы редактирования.
        "Edit": "MobileContactEditPage",
        // Названия моделей, от которых зависит модель "Контакт".
        "RequiredModels": [
            "Account", "Contact", "ContactCommunication", "CommunicationType",
"Department",
            "ContactAddress", "AddressType", "Country", "Region", "City",
"ContactAnniversary",
            "AnniversaryType", "Activity", "SysImage", "FileType",
"ActivityPriority",
            "ActivityType", "ActivityCategory", "ActivityStatus"
        ],
        // Расширения модели.
        "ModelExtensions": [
            "MobileContactModelConfig"
        ],
        // Расширения страниц модели.
        "PagesExtensions": [
            "MobileContactRecordPageSettingsDefaultWorkplace",
            "MobileContactGridPageSettingsDefaultWorkplace",
            "MobileContactActionsSettingsDefaultWorkplace",
            "MobileContactModuleConfig"
        ]
    },
    // Модель "Адреса контактов".
    "ContactAddress": {
        // Страницы реестра, просмотра и редактирования сгенерированы автоматически.
        // Расширения модели.
        "ModelExtensions": [
            "MobileContactAddressModelConfig"
        ]
    }
}
```

Свойство PreferedFilterFuncType

Свойство предназначено для явного определения операции, которая будет использоваться при поиске и фильтрации данных в реестре (в разделах, деталях, справочниках). Значение для свойства задается перечислением *Terrasoft.FilterFunctions*. Перечень функций фильтрации приведен в таблице 2.

Табл. 2. Функции фильтрации (*Terrasoft.FilterFunctions*)

Функция	Значение
SubStringOf	Определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки *property*.
ToUpper	Приводит значения колонки, заданной в *property*, к верхнему регистру.
EndsWith	Проверяет, оканчивается ли значение колонки *property* значением, переданным в качестве аргумента.
StartsWith	Проверяет, начинается ли значение колонки *property* значением, переданным в

	качестве аргумента.
Year	Возвращает год по значению колонки *property*.
Month	Возвращает месяц по значению колонки *property*.
Day	Возвращает день по значению колонки *property*.
In	Проверяет вхождение значения колонки *property* в диапазон значений, переданных в качестве аргумента функции.
NotIn	Проверяет невхождение значения колонки *property* в диапазон значений, переданных в качестве аргумента функции.
Like	Определяет, совпадает ли значение колонки *property* с заданным шаблоном.

Если данное свойство явно не инициализировано в манифесте, то по умолчанию для поиска и фильтрации данных используется функция *Terrasoft.FilterFunctions.StartWith*, так как это обеспечивает использование соответствующих индексов в таблицах базы данных SQLite.

Пример

Для поиска данных использовать функцию поиска подстроки.

Свойство PreferedFilterFuncType конфигурационного объекта манифеста должно выглядеть следующим образом:

```
// Для поиска данных используется функция поиска подстроки.
"PreferedFilterFuncType": "Terrasoft.FilterFunctions.SubStringOf"
```

> ⚠ *ВАЖНО*
>
> Если в секции *PreferedFilterFuncType* в качестве функции фильтрации данных задается функция, отличная от *Terrasoft.FilterFunctions.StartWith*, то при поиске в таблицах БД индексы использоваться *не будут*.

Свойство CustomSchemas

Свойство предназначено для подключения к мобильному приложению дополнительных схем (пользовательских схем с исходным кодом, написанным на JavaScript), расширяющих возможности приложения. Это могут быть, например, дополнительные классы, реализованные разработчиком в рамках проекта, либо утилитные классы, реализующие служебную функциональность, упрощающую работу разработчика, и т.д.

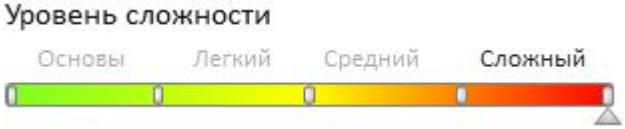
Значение свойства задается массивом с именами подключаемых пользовательских схем.

Пример

Подключить дополнительные пользовательские схемы регистрации действий и утилит.

```
// Подключение дополнительных пользовательских схем.
"CustomSchemas": [
    // Пользовательская схема регистрации действий.
    "MobileActionCheckIn",
    // Пользовательская схема утилит.
    "CustomMobileUtilities"
]
```

3.1.3 Манифест. Свойства синхронизации приложений



Общие положения

Условная группа свойств конфигурационного объекта манифеста. Содержит единственное свойство, с помощью которого выполняются настройки синхронизации данных с основным приложением. Подробнее о манифесте мобильного приложения и всех его свойствах можно узнать в статье "**Манифест мобильного приложения (Section 3.1)**".

Свойство SyncOptions

Описывает параметры для настройки синхронизации данных. Содержит конфигурационный объект со свойствами, представленными в таблице 1.

Табл. 1. Свойства конфигурационного объекта для настроек синхронизации.

Свойство	Значение
ImportPageSize	Количество страниц, импортируемых в одном потоке.
PagesInImportTransaction	Количество потоков импорта.
SysSettingsImportConfig	Массив импортируемых ситемных настроек.
SysLookupsImportConfig	Массив импортируемых ситемных справочников.
ModelDataImportConfig	Массив моделей, для которых будут загружаться данные при синхронизации.

В массиве моделей *ModelDataImportConfig* для каждой модели можно указать дополнительные параметры синхронизации, список загружаемых колонок, а также условия фильтрации загружаемых данных модели. Если при синхронизации должна загружаться полная модель, в массиве просто указывается объект с именем модели. Если к модели должны применяться дополнительные условия при синхронизации, в массив *ModelDataImportConfig* добавляется конфигурационный объект со свойствами, приведенными в таблице 2.

Табл. 2. Свойства конфигурационного объекта для настройки синхронизации модели.

Свойство	Значение
Name	Название модели (см. свойство *Models* конфигурационного объекта манифеста).
SyncColumns	Массив колонок модели, для которых импортируются данные. Помимо явно перечисленных колонок, при синхронизации в обязательном порядке будут импортироваться системные колонки (CreatedOn, CreatedBy, ModifiedOn, ModifiedBy) и колонка, первичная для отображения.
SyncFilter	Фильтр, накладываемый на модель при импорте модели.

Фильтр *SyncFilter*, накладываемый на модель при импорте модели представляет собой конфигурационный объект со свойствами, представленными в таблице 3.

Табл. 3. Свойства конфигурационного объекта фильтра модели.

Свойство	Значение
type	Тип фильтра. Задается значением перечисления *Terrasoft.FilterTypes*. Необязательное свойство. По умолчанию *Terrasoft.FilterTypes.Simple*. Типы фильтров (*Terrasoft.FilterTypes*): *Simple* — Фильтр с одним условием. *Group* — Групповой фильтр с несколькими условиями.
logicalOperation	Логическая операция объединения коллекции фильтров (для фильтров с типом *Terrasoft.FilterTypes.Group*). Задается значением перечисления *Terrasoft.FilterLogicalOperations*. Значение по умолчанию - *Terrasoft.FilterLogicalOperations.And*. Виды логических операций (*Terrasoft.FilterLogicalOperations*): *Or* — Логическая операция ИЛИ.

	And	Логическая операция И.
subfilters	Коллекция фильтров, применяемых к модели. Обязательное свойство для типа фильтра *Terrasoft.FilterTypes.Group*. Фильтры между собой объединяются логической операцией, указанной в свойстве *logicalOperation*. Каждый фильтр представляет собой конфигурационый объект фильтра.	
property	Название колонки модели, по которой выполняется фильтрация. Обязательное свойство для типа фильтра *Terrasoft.FilterTypes.Simple*.	
valueIsMacrosType	Признак, определяющий, является ли значение для фильтрации макросом. Необязательное свойство. Может принимать значения: *true*, если для фильтрации используется макрос, иначе — *false*.	
value	Значение для фильтрации колонки, указанной в свойстве *property*. Обязательное свойство для типа фильтра *Terrasoft.FilterTypes.Simple*. Может задаваться непосредственно значением для фильтрации (в том числе, может быть *null*) либо макросом (для этого свойство *valueIsMacrosType* должно иметь значение *true*). Макросы, которые можно использовать в качестве значения свойства, содержатся в перечислении *Terrasoft.ValueMacros*.	
	Макросы значения (*Terrasoft.ValueMacros*):	
	CurrentUserContactId	Идентификатор текущего пользователя.
	CurrentDate	Текущая дата.
	CurrentDateTime	Текущие дата и время.
	CurrentDateEnd	Полная дата окончания текущей даты.
	CurrentUserContactName	Имя текущего контакта.
	CurrentUserContact	Идентификатор и имя текущего контакта.
	SysSettings	Значение системной настройки. Имя системной настройки передается в свойстве *macrosParams*.
	CurrentTime	Текущее время.
	CurrentUserAccount	Идентификатор и имя контрагента текущего пользователя.
	GenerateUId	Сгенерированный идентификатор.
macrosParams	Значения, которые передаются в макрос в качестве параметра. Необязательное свойство. В настоящее время используется только для макроса *Terrasoft.ValueMacros.SysSettings*.	
isNot	Определяет, применяется к фильтру оператор отрицания. Необязательное свойство. Принимает значенние *true*, если к фильтру применяется оператор отрицания, иначе — *false*.	
funcType	Тип функции, которая применяется к колонке модели, заданой в свойстве *property*. Необязательное свойство. Может принимать значения перечисления *Terrasoft.FilterFunctions*. Значения аргументов для функций фильтрации задаются в свойстве *funcArgs*. Значение, с которым сравнивается результат функции, задается свойством value.	
	Функции фильтрации (*Terrasoft.FilterFunctions*):	
	SubStringOf	Определяет, является ли строка, переданная в качестве аргумента, подстрокой колонки *property*.
	ToUpper	Приводит значения колонки, заданной в *property,* к верхнему регистру.
	EndsWith	Проверяет, оканчивается ли значение колонки *property* значением, переданным в качестве аргумента.

	StartsWith	Проверяет, начинается ли значение колонки *property* значением, переданным в качестве аргумента.
	Year	Возвращает год по значению колонки *property*.
	Month	Возвращает месяц по значению колонки *property*.
	Day	Возвращает день по значению колонки *property*.
	In	Проверяет вхождение значения колонки *property* в диапазон значений, переданных в качестве аргумента функции.
	NotIn	Проверяет невхождение значения колонки *property* в диапазон значений, переданных в качестве аргумента функции.
	Like	Определяет, совпадает ли значение колонки *property* с заданным шаблоном.

funcArgs	Массив значений аргументов для функции фильтрации, заданной в свойстве *funcType*. Порядок значений в массиве *funcArgs* должен соответствовать порядку параметров функции *funcType*.
name	Имя фильтра или группы фильтров. Необязательное свойство.
modelName	Название модели, для которой выполняется фильтрация. Необязательное свойство. Указывается, если фильтрация выполняется по колонкам связанной модели.
assocProperty	Колонка связанной модели, по которой осуществляется связь с основной моделью. В качестве колонки для связи у основной модели выступает первичная колонка.
operation	Тип операции фильтрации. Необязательный параметр. Может принимать значения из перечисления *Terrasoft.FilterOperation*. По умолчанию имеет значение *Terrasoft.FilterOperation.General*.

Операции фильтрации (*Terrasoft.FilterOperation*):

General	Стандартная фильтрация.
Any	Фильтрация с применением фильтра *exists*.

compareType	Тип операции сравнения в фильтре. Необязательный параметр. Принимает значения из перечисления *Terrasoft.ComparisonType*. По умолчанию — *Terrasoft.ComparisonType.Equal*.

Операции сравнения (*Terrasoft.ComparisonType*):

Equal	Равно.
LessOrEqual	Меньше или равно.
NotEqual	Не равно.
Greater	Больше.
GreaterOrEqual	Больше или равно.
Less	Меньше.

Пример

При синхронизации в мобильное приложение должны загружаться данные для таких моделей:

1. Активность. Загружаются все колонки. Выполняется фильтрация модели - загружаются только те активности, у которых участником является текущий пользователь.
2. Тип активности — загружается полная модель.

Свойство *SyncOptions* конфигурационного объекта манифеста должно выглядеть следующим образом:

```
// Настройки синхронизации.
```

```
"SyncOptions": {
    // Количество страниц, импортируемых в одном потоке.
    "ImportPageSize": 100,
    // Количество потоков импорта.
    "PagesInImportTransaction": 5,
    // Массив импортируемых ситемных настроек.
    "SysSettingsImportConfig": [
        "SchedulerDisplayTimingStart", "PrimaryCulture", "PrimaryCurrency",
"MobileApplicationMode", "CollectMobileAppUsageStatistics",
"CanCollectMobileUsageStatistics", "MobileAppUsageStatisticsEmail",
"MobileAppUsageStatisticsStorePeriod", "MobileSectionsWithSearchOnly",
"MobileShowMenuOnApplicationStart", "MobileAppCheckUpdatePeriod",
"ShowMobileLocalNotifications", "UseMobileUIV2"
    ],
    // Массив импортируемых ситемных справочников.
    "SysLookupsImportConfig": [
        "ActivityCategory", "ActivityPriority", "ActivityResult",
"ActivityResultCategory", "ActivityStatus", "ActivityType", "AddressType",
"AnniversaryType", "InformationSource", "MobileApplicationMode",
"OppContactInfluence", "OppContactLoyality", "OppContactRole", "OpportunityStage",
"SupplyPaymentDelay", "SupplyPaymentState", "SupplyPaymentType"],
    // Массив моделей, для которых будут загружаться данные при синхронизации.
    "ModelDataImportConfig": [
        // Конфигурирование модели Activity.
        {
            "Name": "Activity",
            // Фильтр, накладываемый на модель при импорте.
            "SyncFilter": {
                // Название колонки модели, по которой выполняется фильтрация.
                "property": "Participant",
                // Название модели, для которой выполняется фильтрация.
                "modelName": "ActivityParticipant",
                // Колонка связанной модели, по которой осуществляется связь с
основной моделью.
                "assocProperty": "Activity",
                // Тип операции фильтрации.
                "operation": "Terrasoft.FilterOperations.Any",
                // Для фильтрации используется макрос.
                "valueIsMacros": true,
                // Значение для фильтрации колонки — идентификатор и имя текущего
контакта.
                "value": "Terrasoft.ValueMacros.CurrentUserContact"
            },
            // Массив колонок модели, для которых импортируются данные.
            "SyncColumns": [
                "Title", "StartDate", "DueDate", "Status", "Result",
"DetailedResult", "ActivityCategory", "Priority", "Owner", "Account", "Contact",
"ShowInScheduler", "Author", "Type"
            ]
        },
        // Модель ActivityType загружается полностью.
        {
            "Name": "ActivityType",
            "SyncColumns": []
        }
    ]
}
```

Свойство SyncOptions.ModelDataImportConfig.QueryFilter

Доступно в bpm'online, начиная с версии 7.12.1, и в мобильном приложении bpm'online, начиная с версии

7.12.3.

Свойство синхронизации *QueryFilter* позволяет настроить фильтрацию данных указанной модели при импорте с помощью службы DataService. Ранее для фильтрации данных использовалось свойство *SyncFilter*, а импорт выполнялся с помощью OData (EntityDataService).

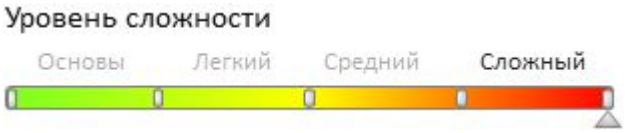
> ВАЖНО
>
> Импорт данных с помощью службы DataService доступен только для платформ Android и iOS. Для платформы Windows используется OData (EntityDataService).

Формат фильтра *QueryFilter* представляет собой набор параметров в виде JSON-объекта, передаваемых в запросе к службе DataService. Описание параметров DataService можно найти в статье "DataService. Фильтрация данных" документации по разработке bpm'online.

Пример exists-фильтра приведен ниже:

```
{
  "SyncOptions": {
     "ModelDataImportConfig": [
        {
           "Name": "ActivityParticipant",
           "QueryFilter": {
              "logicalOperation": 0,
              "filterType": 6,
              "rootSchemaName": "ActivityParticipant",
              "items": {
                 "ActivityFilter": {
                    "filterType": 5,
                    "leftExpression": {
                       "expressionType": 0,
                       "columnPath": "Activity.[ActivityParticipant:Activity].Id"
                    },
                    "subFilters": {
                       "logicalOperation": 0,
                       "filterType": 6,
                       "rootSchemaName": "ActivityParticipant",
                       "items": {
                          "ParticipantFilter":{
                             "filterType": 1,
                             "comparisonType": 3,
                             "leftExpression": {
                                "expressionType": 0,
                                "columnPath": "Participant"
                             },
                             "rightExpression": {
                                "expressionType": 1,
                                "functionType": 1,
                                "macrosType": 2
                             }
                          }
                       }
                    }
                 }
              }
           }
        }
     ]
  }
}
```

3.1.4 Экспорт данных в пакетном режиме

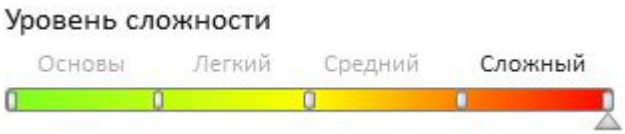


По умолчанию мобильное приложение отправляет каждое произведенное пользователем изменение данных поочередно. Таким образом одно изменение производит как минимум один запрос на сервер. При достаточно большом количестве изменений выполнение таких запросов может занять существенное время.

Начиная с версии bpm'online 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.

3.2 Жизненный цикл страниц в мобильном приложении



Общие сведения

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.

Обработчики событий жизненного цикла

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

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.

Навигация страниц

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

forward(openingPageConfig)

Метод открывает страницу с учетом свойств конфигурационного объекта-параметра *openingPageConfig*. Основные свойства этого объекта представлены в таблице 1.

Табл. 1. — Свойства объекта *openingPageConfig*

Свойство	Описание
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.

Навигация с использованием маршрутов

Маршрутизация

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()*.

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:
"MobileActivityGridPage"}]);
```

back()

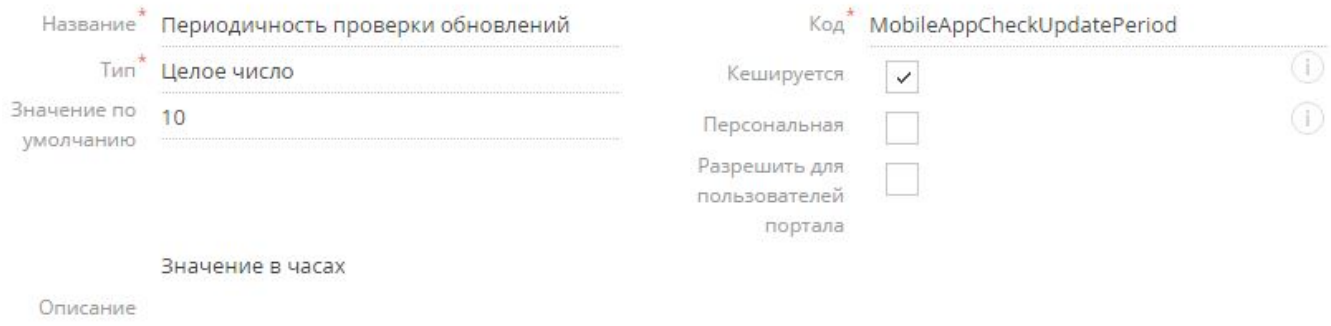
Выполняет закрытие текущего маршрута и восстановление предыдущего.

3.3 Фоновое обновление конфигурации в мобильном приложении

Общая информация

В мобильном приложении bpm'online реализован механизм синхронизации структуры приложения, который может работать в автоматическом фоновом режиме. Для управления этим процессом необходимо использовать системную настройку [Периодичность проверки обновлений] (рис. 1).

Рис. 1. — Системная настройка [Периодичность проверки обновлений]



Эта настройка указывает по истечении какого времени (в часах) мобильное приложение может запросить изменения конфигурации у bpm'online. 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];
- 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.

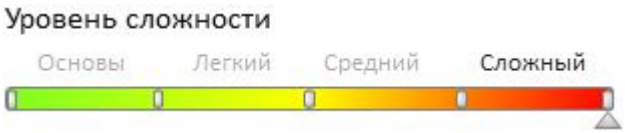
Особенности работы на разных платформах

1. На платформе Android фоновый режим реализован через параллельно запущенный сервис. Такой подход гарантирует, что запущенная синхронизация гарантированно 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.
3. На платформе Windows 10 приложение при старте проверяет (не в фоновом режиме) наличие на сервере обновлений.

   В случае наличия обновлений отобразится страница с соответствующей инфомрацией.

3.4 Получение настроек и данных раздела [Итоги]



Общая информация

Функциональность получения настроек и данных по дашбордам реализована в сервисе *AnalyticsService* и в утилитном классе *AnalyticsServiceUtils*, пакет *Platform*.

AnalyticsService

Класс, реализующий сервис *AnalyticsService*, содержит следующие публичные методы:

- *public Stream GetDashboardViewConfig(Guid id)* — возвращает настройки представления и виджетов на вкладке итогов по идентификатору страницы итогов.
- *public Stream GetDashboardData(Guid id, int timeZoneOffset)* — возвращает данные по всем виджетам на вкладке итогов по идентификатору страницы итогов.
- *public Stream GetDashboardItemData(Guid dashboardId, string itemName, int timeZoneOffset)* — возвращает данные по определенному виджету по идентификатору страницы итогов и имени виджета.

Здесь *timeZoneOffset* — смещение (в минутах) часового пояса относительно UTC. Данные по итогам будут получены с использованием этого часового пояса.

Примеры запросов к сервису AnalyticsService

HEADERS

```
Accept:application/json
```

Метод GetDashboardViewConfig()

URL

```
POST /0/rest/AnalyticsService/GetDashboardViewConfig
```

Содержимое запроса

```
{
    "id": "a71d5c04-dff7-4892-90e5-9e7cc2246915"
}
```

Содержимое ответа

```
{
  "items": [
    {
      "layout": {
        "column": 0,
        "row": 0,
        "colSpan": 12,
        "rowSpan": 5
      },
      "name": "Chart4",
      "itemType": 4,
      "widgetType": "Chart"
    }
  ]
}
```

Метод GetDashboardData()

URL

```
POST /0/rest/AnalyticsService/GetDashboardData
```

Содержимое запроса

```
{
    "id": "a71d5c04-dff7-4892-90e5-9e7cc2246915",
    "timeZoneOffset": 120
}
```

Содержимое ответа

```
{
    "items": [
        {
            "name": "Indicator1",
            "caption": "Среднее время выполнения активности",
            "widgetType": "Indicator",
            "style": "widget-green",
            "data": 2
        }
    ]
}
```

Метод GetDashboardItemData()

URL

```
POST /0/rest/AnalyticsService/GetDashboardItemData
```

Содержимое запроса

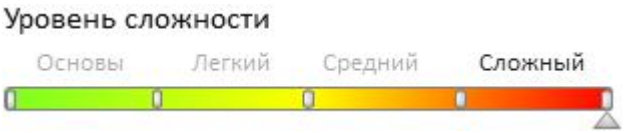
```
{
    "dashboardId": "a71d5c04-dff7-4892-90e5-9e7cc2246915",
    "itemName": "Chart4",
    "timeZoneOffset": 120
}
```

Содержимое ответа

```
{
  "name": "Chart4",
  "caption": "Invoice payment dynamics",
  "widgetType": "Chart",
  "chartConfig": {
    "xAxisDefaultCaption": null,
    "yAxisDefaultCaption": null,
    "seriesConfig": [
      {
        "type": "column",
        "style": "widget-green",
        "xAxis": {
          "caption": null,
          "dateTimeFormat": "Month;Year"
        },
        "yAxis": {
```

```
            "caption": "Actually paid",
            "dataValueType": 6
          },
          "schemaName": "Invoice",
          "schemaCaption": "Invoice",
          "useEmptyValue": null
        }
      ],
      "orderDirection": "asc"
    },
    "style": "widget-green",
    "data": []
}
```

3.5 Автоматическое разрешение конфликтов при синхронизации



Общая информация

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

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

Каждая из приведенных выше ситуаций обрабатывается мобильным приложением автоматически.

Слияние дублей

Алгоритм разрешения конфликта, возникающего по причине использования данных, которые в bpm'online были удалены в ходе процедуры слияния дублей, представлен схематически на рис. 1:

Рис. 1. — Схема разрешения конфликта, возникшего в результате слияния дублей на сервере

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 bpm'online. 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 bpm'online не найдена, приложение выполняет следующие действия:

1. Проверяет наличие записи в списке заисей, удаленных в ходе слияния дублей (см. "Слияние 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.

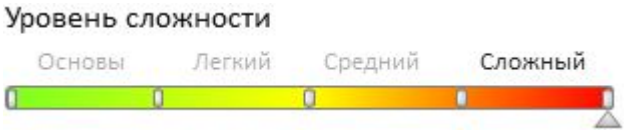
4 Mobile SDK

Содержание

SDK реестра (Section 4.1)

Классы, методы и свойства реестра мобильного приложения bpm'online.

4.1 SDK реестра



> ВАЖНО
>
> Эта статья актуальна для мобильного приложения версии 7.11.1 и выше.

Общие сведения

SDK реестра — это инструмент, позволяющий настраивать внешний вид реестра, сортировку, логику поиска и т. д. Он реализован в классе *Terrasoft.sdk.GridPage*.

Методы Terrasoft.sdk.GridPage

setPrimaryColumn()

Устанавливает первичную колонку для отображения. Настраивает отображение заголовка записи реестра.

Сигнатура метода

setPrimaryColumn(modelName, column)

Параметры

modelName — название модели.

column — название колонки.

Пример вызова

```
Terrasoft.sdk.GridPage.setPrimaryColumn("Case", "Subject");
```

setSubtitleColumns()

Устанавливает колонки, которые отображаются под заголовком. Настраивает отображение подзаголовка в виде списка колонок с разделителем.

Сигнатура метода

setSubtitleColumns(modelName, columns)

Параметры

modelName — название модели.

columns — массив колонок или конфигурационных объектов колонок.

Пример вызова

Вариант 1

```
Terrasoft.sdk.GridPage.setSubtitleColumns("Case", ["RegisteredOn","Number"]);
```

Вариант 2

```
Terrasoft.sdk.GridPage.setSubtitleColumns("Case", ["RegisteredOn", { name: "Number",
convertFunction: function(values) { return values.Number; } }]);
```

setGroupColumns()

Устанавливает группу с колонками, которые отображаются вертикально. Настраивает отображение группы колонок.

Сигнатура метода

setGroupColumns(modelName, columns)

Параметры

modelName — название модели.

columns — массив колонок или конфигурационных объектов колонок.

Пример вызова

Вариант 1

```
Terrasoft.sdk.GridPage.setGroupColumns("Case", ["Symptoms"])
```

Вариант 2

```
Terrasoft.sdk.GridPage.setGroupColumns("Case", [
{
name: "Symptoms",
isMultiline: true,//Отображать как многострочное поле
label: "CaseGridSymptomsColumnLabel",//Имя локализованной строки
convertFunction: function(values) {
return values.Symptoms;
}
}]);
```

setImageColumn()

Устанавливает колонку изображения.

setOrderByColumns()

Устанавливает сортировку реестра.

setSearchColumn()

Устанавливает колонку поиска.

setSearchColumns()

Устанавливает колонки поиска.

setSearchPlaceholder()

Устанавливает текст подсказки в поле поиска.

setTitle()

Устанавливает заголовок страницы реестра.

Пример

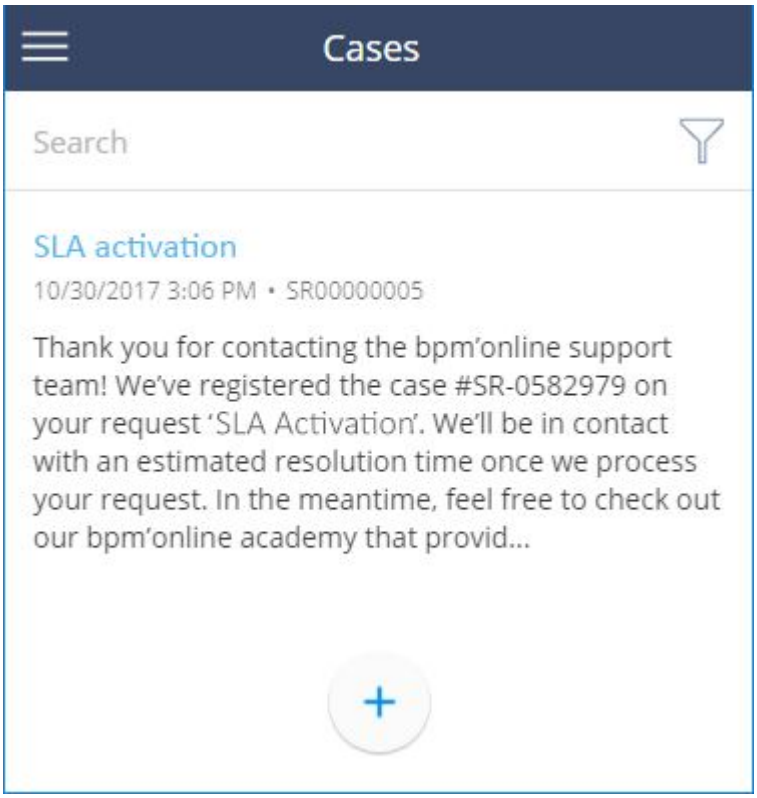
Необходимо настроить реестр раздела [Обращения] ([Cases]) таким образом, чтобы отображался заголовок с темой обращения, подзаголовок с датой регистрации и номером, а также описание обращения в виде многострочного поля.

Для настройки реестра необходимо использовать приведенный ниже исходный код:

```
// Настройка первичной колонки с темой обращения.
Terrasoft.sdk.GridPage.setPrimaryColumn("Case", "Subject");
// Установка подзаголовка с датой регистрации и номером обращения.
Terrasoft.sdk.GridPage.setSubtitleColumns("Case", ["RegisteredOn","Number"]);
// Добавление многострочного поля с описанием.
Terrasoft.sdk.GridPage.setGroupColumns("Case", [
{
name: "Symptoms",
isMultiline: true
}]);
```

В результате реестр будет отображаться так, как показано на рис. 1.

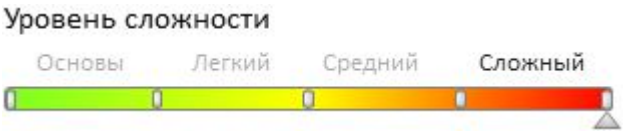
Рис. 1. — Настроенный реестр обращений



5 Разработка bpm'online mobile на примерах

Содержание

5.1 Как добавить стандартную деталь с колонками



Общие сведения

Для добавления детали в раздел мобильного приложения bpm'online необходимо использовать мастер мобильного приложения. Как настроить деталь с помощью мастера мобильного приложения описывается в статье "Настройка детали раздела" документации bpm'online mobile.

Однако, если объект детали не является объектом какого-либо раздела мобильного приложения bpm'online, то на странице детали вместо значений будет отображаться идентификатор связанной записи раздела. Для отображения значений необходимо выполнить конфигурирование схемы страницы детали.

Описание примера

На страницу редактирования записи раздела [Контакты] мобильного приложения добавить деталь [Карьера]. В качестве основной отображать колонку [Должность].

Исходный код

Пакет с реализацией примера можно скачать по ссылке.

Алгоритм реализации кейса

1. Добавить деталь [Карьера] с помощью мастера мобильного приложения

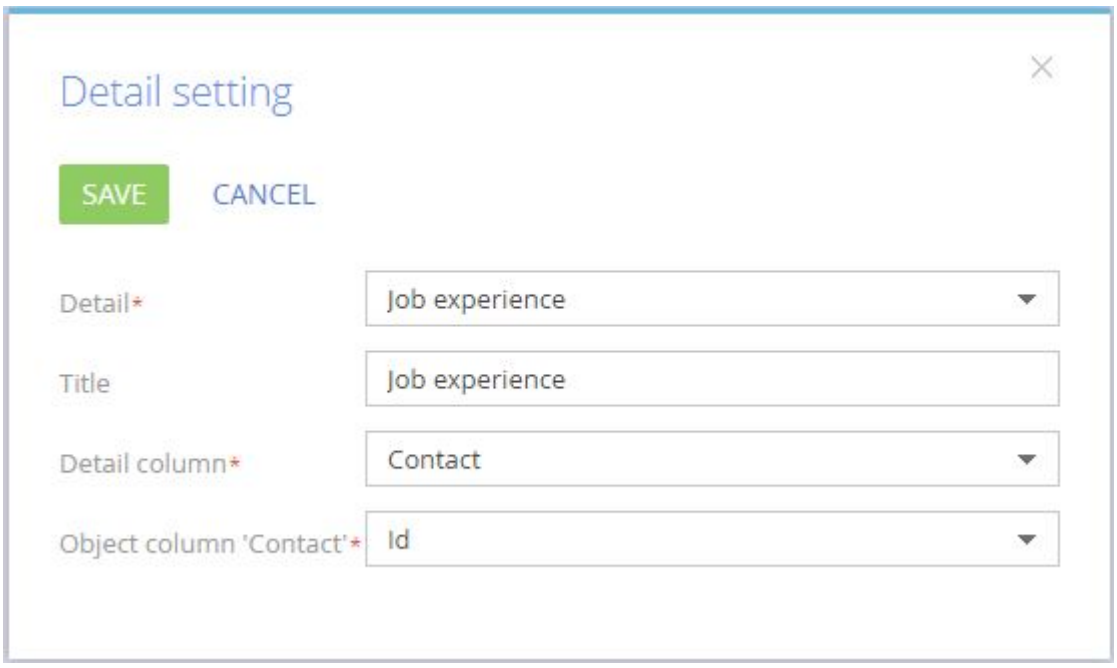
Чтобы добавить деталь на страницу редактирования записи, используйте мастер мобильного приложения. Для этого:

1.1. Откройте нужное рабочее место, например, [Основное рабочее место] ([Main workplace]) и нажмите кнопку [Настроить разделы] ([Set up sections]).

1.2. Выберите раздел [Контакты] и нажмите кнопку [Настроить детали] ([Details setup]).

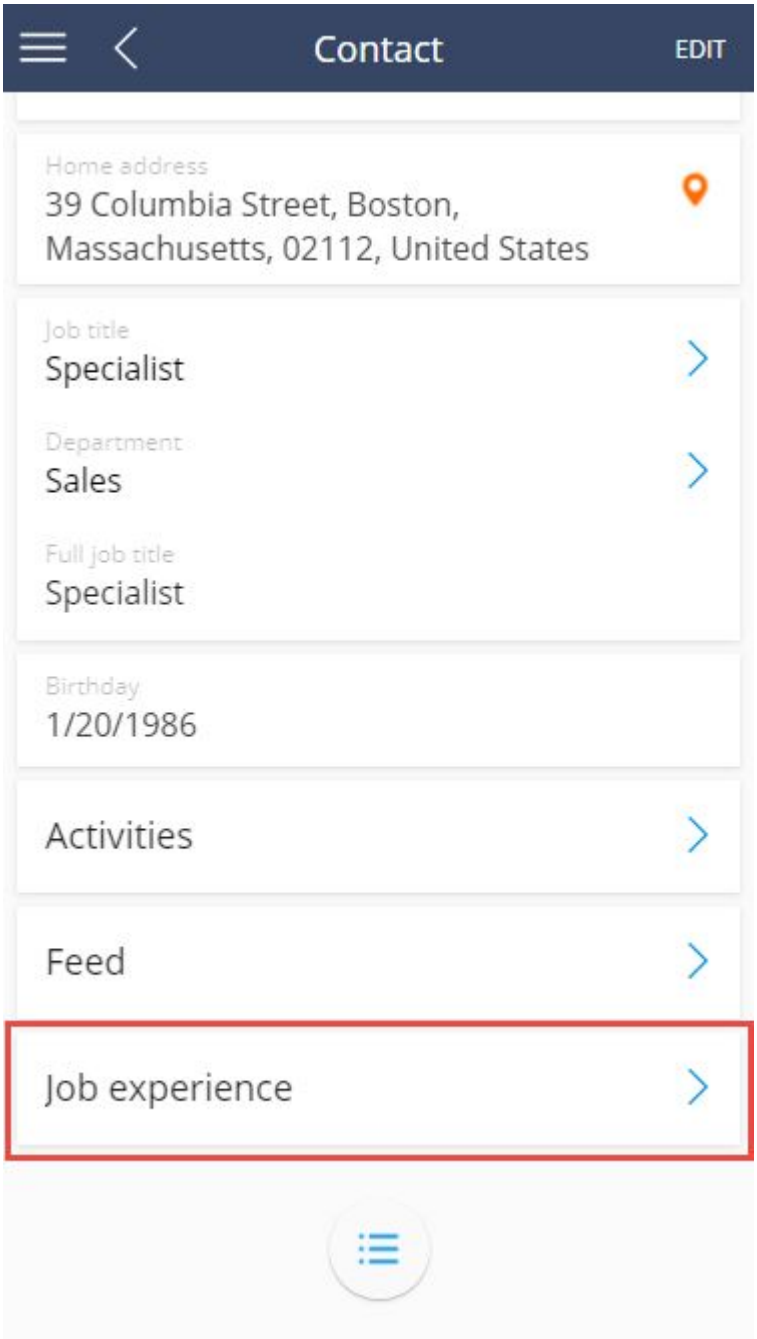
1.3. Настройте деталь [Карьера] ([Job experience]) (рис. 1).

Рис. 1. — Настройка детали [Карьера] ([Job experience])



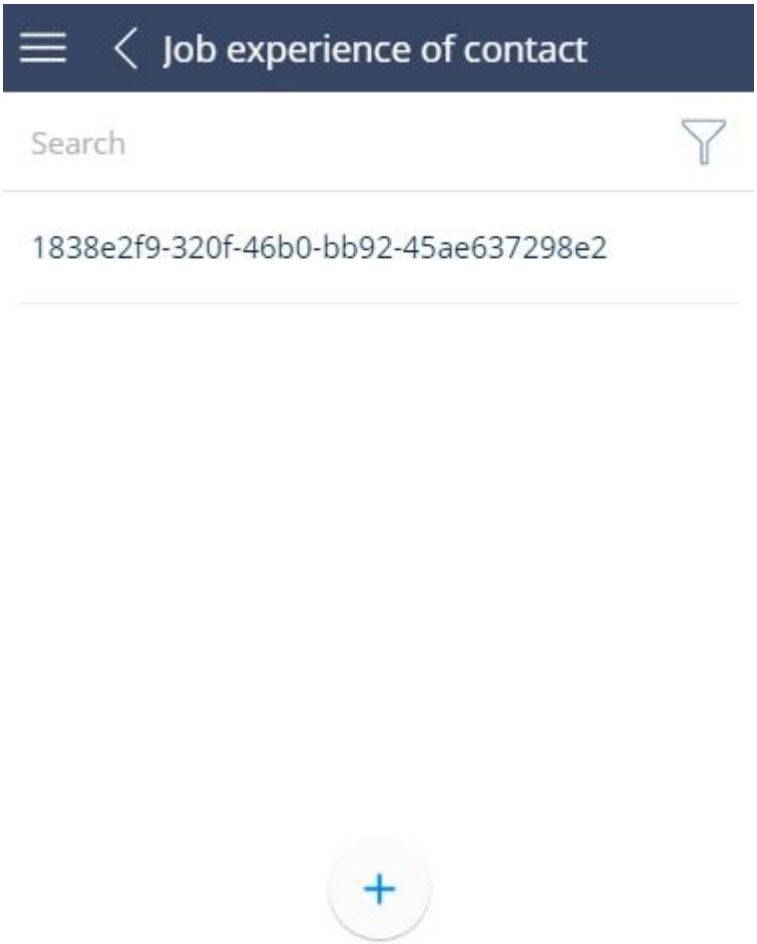
После сохранения результатов настройки детали, раздела и рабочего места в мобильном приложении отобразится деталь [Карьера] ([Job experience]) (рис. 2).

Рис. 2. — Деталь [Карьера] ([Job experience]) на странице записи раздела [Контакты] ([Contacts])



Однако, поскольку объект детали [Карьера] не является объектом раздела мобильного приложения bpm'online, то на странице детали отображается значение основной колонки [Контакт](идентификатор связанной записи контакта).

Рис. 3. — Отображение идентификатора связанной записи контакта

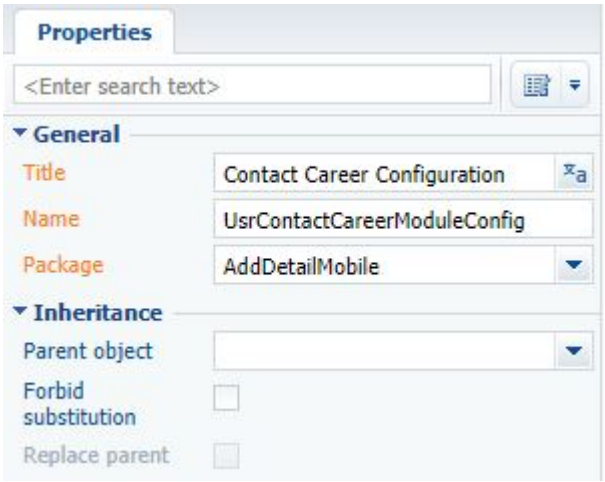


2. Создать схему модуля, в которой выполнить конфигурирование реестра детали

В разделе [Конфигурация] приложения bpm'online в пользовательском пакете создайте клиентский модуль со следующими свойствами (рис. 4):

- [Заголовок] ([Title]) — "Настройки карьеры контакта" ("Contact Career Configuration").
- [Название] ([Name]) — "UsrContactCareerModuleConfig".

Рис. 4. — Свойства схемы модуля



Добавьте в схему модуля следующий исходный код:

```
// Установка колонки [Должность] в качестве первичной.
```

```
Terrasoft.sdk.GridPage.setPrimaryColumn("ContactCareer", "JobTitle");
// Добавление колонки [Должность] в коллекцию первичных колонок.
Terrasoft.sdk.RecordPage.addColumn("ContactCareer", {
        name: "JobTitle",
        position: 1
    }, "primaryColumnSet");
// Удаление предыдущей первичной колонки [Контакт] из коллекции первичных колонок.
Terrasoft.sdk.RecordPage.removeColumn("ContactCareer", "Contact",
"primaryColumnSet");
```

Здесь:

- "ContactCareer" — название таблицы, которая соответствует детали (как правило оно совпадает с названием объекта детали).
- "Job Title" — название колонки, которую требуется отобразить на странице.

3. Подключить схему модуля в манифесте мобильного приложения

Для применения настроек реестра детали, выполненный в модуле *UsrContactCareerModuleConfig*, выполните следующие шаги:

3.1. Откройте в дизайнере клиентского модуля схему манифеста мобильного приложения *MobileApplicationManifestDefaultWorkplace*. Эта схема создается в пользовательском пакете мастером мобильного приложения (см. "**Как добавить пользовательский раздел в мобильное приложение (on-line documentation)**").

3.2. Добавьте модуль *UsrContactCareerModuleConfig* в секцию *PagesExtensions* модели *ContactCareer*:

```
{
    "SyncOptions": {
        ...
    },
    "Modules": {},
    "Models": {
        "ContactCareer": {
            "RequiredModels": [
                ...
            ],
            "ModelExtensions": [],
            "PagesExtensions": [
                ...
                "UsrContactCareerModuleConfig"
            ]
        },
        ...
    }
}
```

3.3. Сохраните схему манифеста мобильного приложения

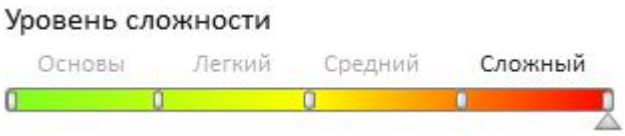
В результате выполнения примера на странице детали [Карьера] ([Job experience]) будут отображаться записи по колонке [Должность] (рис. 5).

Рис. 5. — Результат выполнения примера

ВАЖНО

Чтобы в мобильном приложении отображались сконфигурированные колонки, необходимо выполнить очистку кэша мобильного приложения. В некоторых случаях предварительно следует выполнить компиляцию приложения bpm'online.

5.2 Модификаторы доступа страницы в мобильном приложении



В мобильном приложении версий 7.11.0 и выше добавилась полноценная возможность настройки модификаторов доступа раздела или стандартной детали. Например, можно запретить в разделе изменение, добавление и удаление записей для всех пользователей.

Для установки доступа в режим чтения необходимо в схему, название которой содержит значение "ModuleConfig", добавить следующий исходный код :

```
Terrasoft.sdk.Module.setChangeModes("UsrClaim", [Terrasoft.ChangeModes.Read]);
```

Или для стандартной детали:

```
Terrasoft.sdk.Details.setChangeModes("UsrClaim", "StandardDetailName",
[Terrasoft.ChangeModes.Read]);
```

В результате на странице реестра будет удалена кнопка добавления, а на странице просмотра – кнопка

редактирования. Также на странице просмотра будут отсутствовать кнопки действий [Добавить], [Удалить], [Добавить запись] для работы с записями раздела на встроенной детали др.

Модификаторы доступа можно комбинировать. Например, если нужно добавлять и изменять записи, но удалять их нельзя, то необходимо добавить следующий исходный код:

```
Terrasoft.sdk.Module.setChangeModes("UsrClaim", [Terrasoft.ChangeModes.Create,
Terrasoft.ChangeModes.Update]);
```

Все модификаторы доступа приведены в перечислении *Terrasoft.ChangeModes*.