Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования

"Национальный исследовательский университет

"Высшая школа экономики"

Отделение программной инженерии
Кафедра Управления разработкой программного обеспечения

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

На тему “Технология создания кроссплатформенных приложений с динамическим формированием структуры и контента”

Студент группы 271мУРПО

_____

Пронин С. А.

Научный руководитель

_____

проф., к.т.н. Авдошин С. М.

Москва, 2014 г.

АННОТАЦИЯ

Данная работа посвящена разработке технологии создания кроссплатформенных приложений с динамическим формированием структуры и контента. Основные задачи включают в себя поиск и разработку оптимального паттерна проектирования мобильных приложений, на основе которого возможно построение архитектурного решения, позволяющего создание приложений с динамической структурой и контентом. Выбранное оригинальное решение является модифицированным паттерном Model-View-ViewModel, который включает структурные элементы паттерна Model-View-Controller.

В работе приведен анализ существующих подходов к решению проблемы создания кроссплатформенных приложений, и, в конечном счете, предложена оригинальная архитектура разработанной технологии, предусматривающей создание нативных мобильных кроссплатформенных приложений без необходимости написания программного кода.

Отдельное внимание уделяется описанию модели предметной области создаваемых кроссплатформенных приложений. Конечная реализация включает runtime-генерацию классов, на основе которых происходит конфигурация локального для клиентского приложения ORM хранилища данных с учетом возможности изменения описания модели.

Результатами являются: реализованный паттерн проектирования, разработанная технология создания кросс-платформенных приложений с динамическим формирование структуры и контента, разработанное веб-приложение - конструктор, позволяющий настраивать и изменять структуру и контент кросс-платформенных приложений, разработанное демонстрационное мобильное приложение, способное изменять структуру и содержание контента без внесения изменений в исходный код.

Объем работы: 73 страницы, 4 главы, 19 использованных источников.

Ключевые слова: MVC, MVVM, кроссплатформенные приложения, runtime-создание классов, технология создания приложений.

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ

В настоящее времен сложно отрицать взрывной рост количества пользователей мобильных устройств [17] (рис. 2). Рынку любой отрасли приходится в той или иной мере сконцентрировать свое внимание на мобильных устройствах, как с точки зрения маркетинга, так и с точки зрения непосредственного взаимодействия с клиентом. Потенциальные потребители в настоящий момент проводят огромное количество времени, используя мобильные устройства. Внутреннее распределение внимания можно разделить на две составляющих: использование приложений и просмотр мобильных веб-страниц.

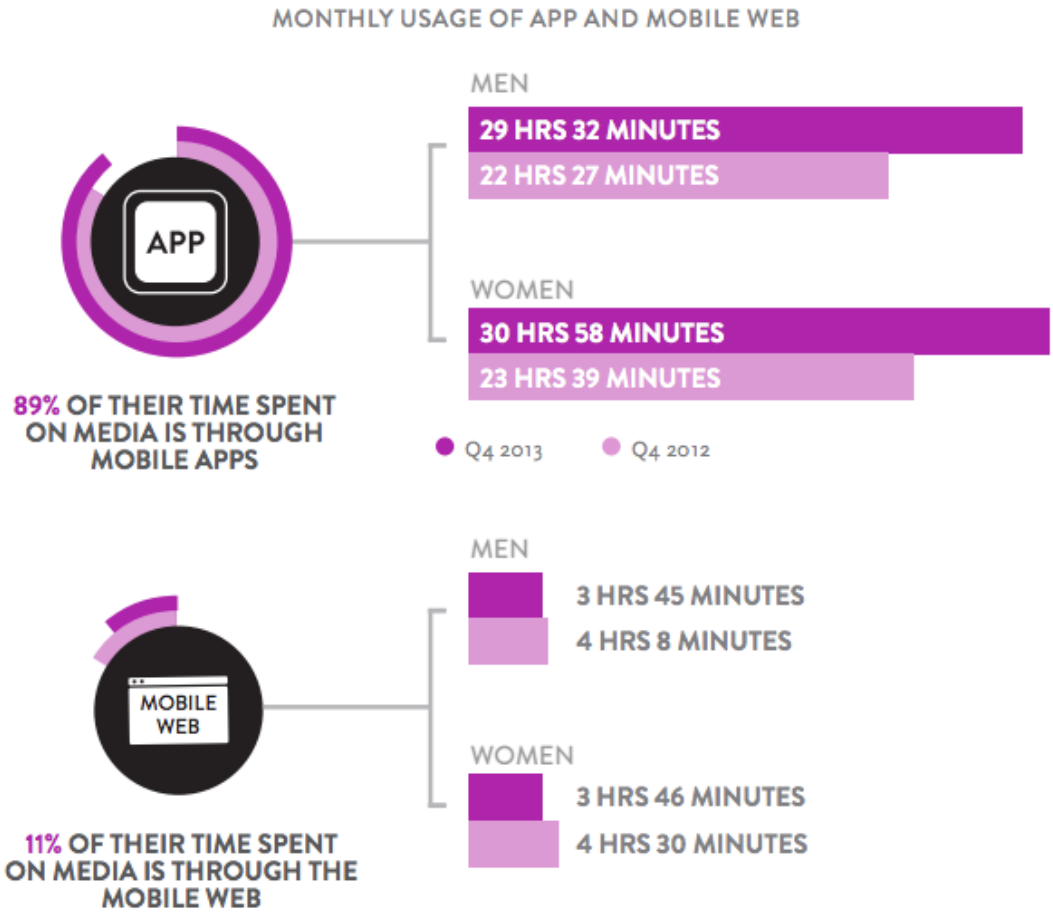


Рисунок 1. Распределение времени пользователя при использовании мобильного устройства

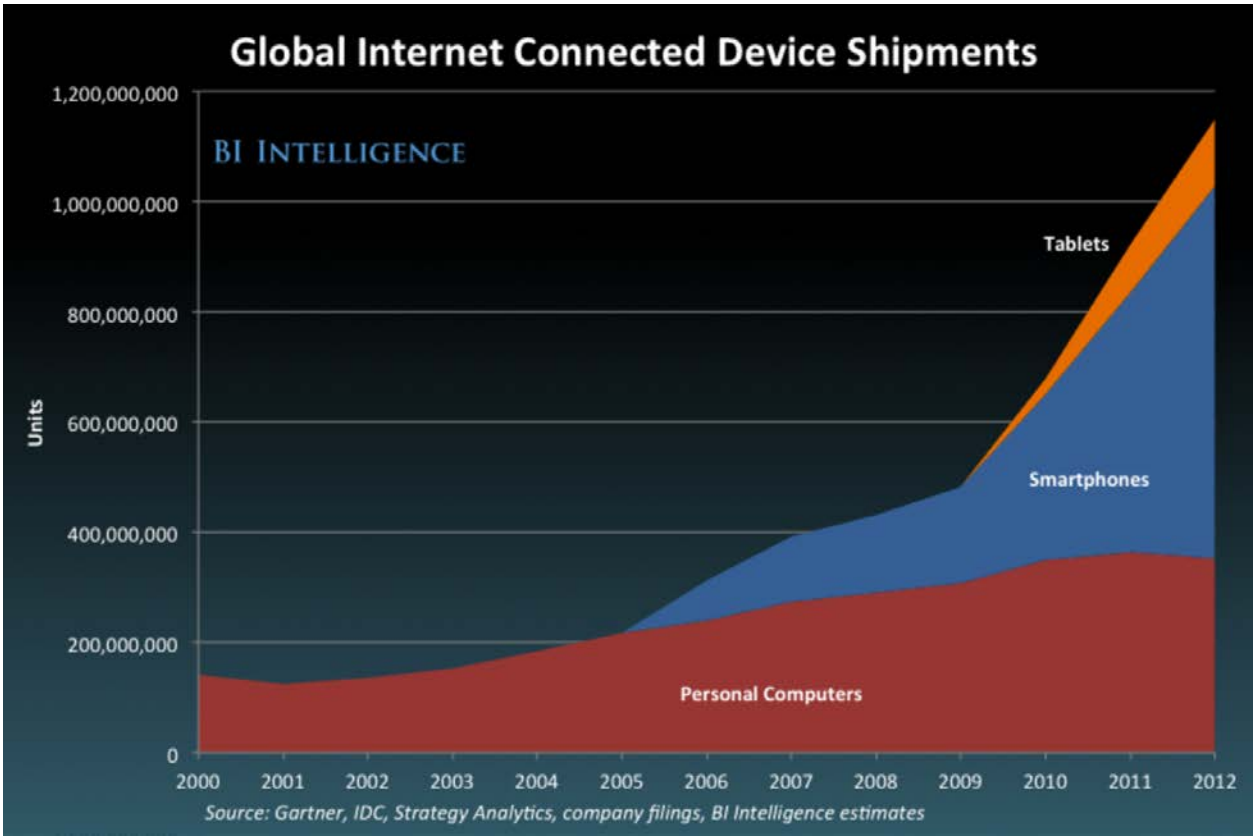


Рисунок 2. Поставки персональных устройств доступа к сети Интернет на глобальном рынке

Согласно последним отчетам аналитиков [1] внимание пользователей в большей мере сосредоточено именно на приложениях, нежели чем на просмотре мобильных веб-страниц: 89% против 11% соответственно (рис. 1). Таким образом, одной из основных стратегий современных компаний (как малого, так и среднего бизнеса) становится развитие направления, связанного с присутствием своих приложений на мобильных платформах. Отчеты маркетинговых аналитиков от 2013 года показывают высокий интерес пользователей в отношении мобильных приложений, связанных с брендом той или иной компании (рис. 3).

В настоящее время на рынке существует множество агенств, занимающихся заказной разработкой мобильных приложений для нужд современного бизнеса. Многие из этих приложений в конечном счете преследуют одну и ту же цель - продвижение бренда, повышение узнаваемости компании, развитие отношений с клиентами, продвижение с помощью мобильных социальных инструментов и сервисов. Такого рода

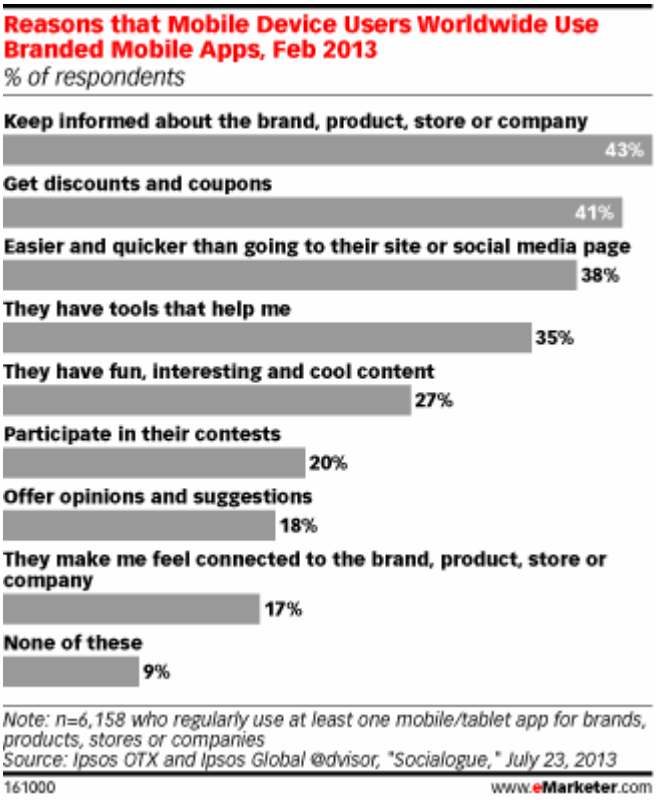


Рисунок 3. Заинтересованность пользователя в мобильных приложениях компаний [2]

заказные приложения могут позволить себе только крупные компании, тогда как компании малого и среднего размера, а так же государственные и образовательные учреждения, не могут позволить себе такого рода решения, так как цена их разработки очень высока. Решение этой проблемы и лежит в основе данной работы.

В рамках курсовой работы первого курса магистратуры был создан комплекс приложений для продвижения образовательных программ в социальных медиа, в составе которого были разработаны клиентское и серверное приложения, позволяющие изменять содержание и структуру контента в рамках заранее заданных шаблонов без изменения исходного кода клиентской стороны. Примененные подходы и методы подвели автора настоящей работы к пониманию требуемого уровня абстракции технологии, при которой была бы возможна разработка мобильных приложений без необходимости написания программного кода с использованием произвольных шаблонов.

1.1. Цели работы

Целью работы является создание технологии, которая бы позволила создавать кросс-платформенные мобильные приложения с динамическим формированием структуры и содержания контента. Уникальной составляющей технологической платформы является возможность создания приложений, не покидая веб-браузер, то есть без необходимости привлечения программистов и разработчиков.

Необходимо разработать инструмент, позволяющий создавать и конфигурировать структуру приложения без вмешательства со стороны разработчика клиентского приложения. Таким образом, любая компания или образовательное учреждение сможет создать свое мобильное приложение с использованием без привлечения сторонних компаний и каких-либо затрат.

Разработка технологической платформы включает в себя следующие задачи.

1. Проектирование архитектуры, допускающей создание и настройку динамической структуры и содержания контента.
2. Разработка web-инструментов проектирования и генерации динамического отображения контента.
3. Разработка API платформы, которое бы позволило сторонним программистам разрабатывать свои модули для использования в создаваемых приложениях.
4. Разработка внешнего API, которое будет позволять мобильным приложениям взаимодействовать с платформой: получение структуры и шаблонов отображения контента, обработка событий и действий клиентского приложения.
5. Разработка демонстрационного мобильного клиента на платформе iOS,

которое будет реагировать на изменения в структуре, контенте и отображении без необходимости пересборки.

По сути, можно говорить о технологической платформе, как о **конструкторе**, позволяющем удаленно модифицировать имеющееся у пользователя приложение без внесения изменений в исходных код и без повторной компиляции и сборки.

2. АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

С точки зрения анализа подобных решений можно выделить две категории существующих на рынке продуктов: платформы для генерации готовых мобильных приложений и платформы для создания приложений, используя веб-технологии с последующей генерацией мобильного клиента для разных платформ.

Все эти продукты так или иначе пытаются решить проблему кросс-платформенной разработки мобильных приложений. Согласно современным тенденциям можно выделить три основных типа создаваемых мобильных приложений [12]:

1. Нативные приложения (native), то есть специфические для какой-либо конкретной платформы: iOS, Android, Windows Phone и т.д.. Разработку необходимо вести с помощью специальных инструментов на разных для каждой отдельной взятой платформы языках программирования. Созданное приложение обычно не является универсальным, то есть необходима разработка отдельных приложений для разных мобильных платформ. В целом, нативные приложения обладают полноценным набором функций, отзывчивостью интерфейса, а также отличаются производительностью. Можно выделить основные отличительные способности:

    - multi-touch жесты;
    - быстрый графический API - эта особенность является критической при отображении больших объемов данных и взаимодействии с ними;
    - доступ к встроенным компонентам устройства и системы: камера, адресная книга, геолокация, а так защищенное локальное хранилище (мобильные приложения в рамках мобильной

операционной системы и ее технологической платформы находятся в, так называемой, “песочнице” (“sandbox”), что обычно исключает доступ к их локальным данным извне);

- простота в использовании для конечного пользователя: нативное решение - это решение, к которому пользователь мобильного устройства уже успел привыкнуть;
- документация: на просторах сети Интернет и на полках книжных магазинов можно найти массу информации о разработке приложений для таких мобильных платформ, как, например, Android и iOS, а также материалы для изучения обычно поставляются компаниями разработчиками мобильных операционных систем и активно поддерживаются сообществами разработчиков.

Нативные приложения обычно разрабатываются с использованием интегрированных сред разработки, которые предоставляют инструменты для создания и отладки мобильных приложений, а также зачастую интегрированы с системами управления проектами (TeamCity, Jira и т.п.) и системами контроля версий (Git, SVN, Mercurial). Необходимость в использовании подобных зачастую сложных и перегруженных инструментов объясняется сложностью разработки для разных платформ.

Большим преимуществом нативного приложения является его представление для пользователя: каждая мобильная платформа предоставляет свой “магазин приложений”, где, с одной стороны, разработчики размещают свои приложения, а потребители, с другой стороны, устанавливают их себе на устройства без дополнительных трудовых затрат.

2. HTML5 приложения, то есть использующие стандартные веб-

технологии: обычно это HTML5, JavaScript и CSS. Такой подход подразумевает разработку кросс-платформенных приложений, которые будут работать на разных с точки зрения программной платформы устройствах. Не смотря на кажущиеся преимущества такой разработки, зачастую открытым остается вопрос управления сессиями, offline-хранение данных, работа без соединения, а также доступ к нативной функциональности устройства.

Мобильное HTML5 приложение - это, обычно, веб-страница (или набор связанных веб-страниц) со специально спроектированной для отображения на мобильных устройствах разметкой. Современные HTML5 приложения не зависят от конкретной платформы и могут исполняться любым современным мобильным веб-браузером. Порог входа для разработки HTML5 приложений обычно ниже, однако

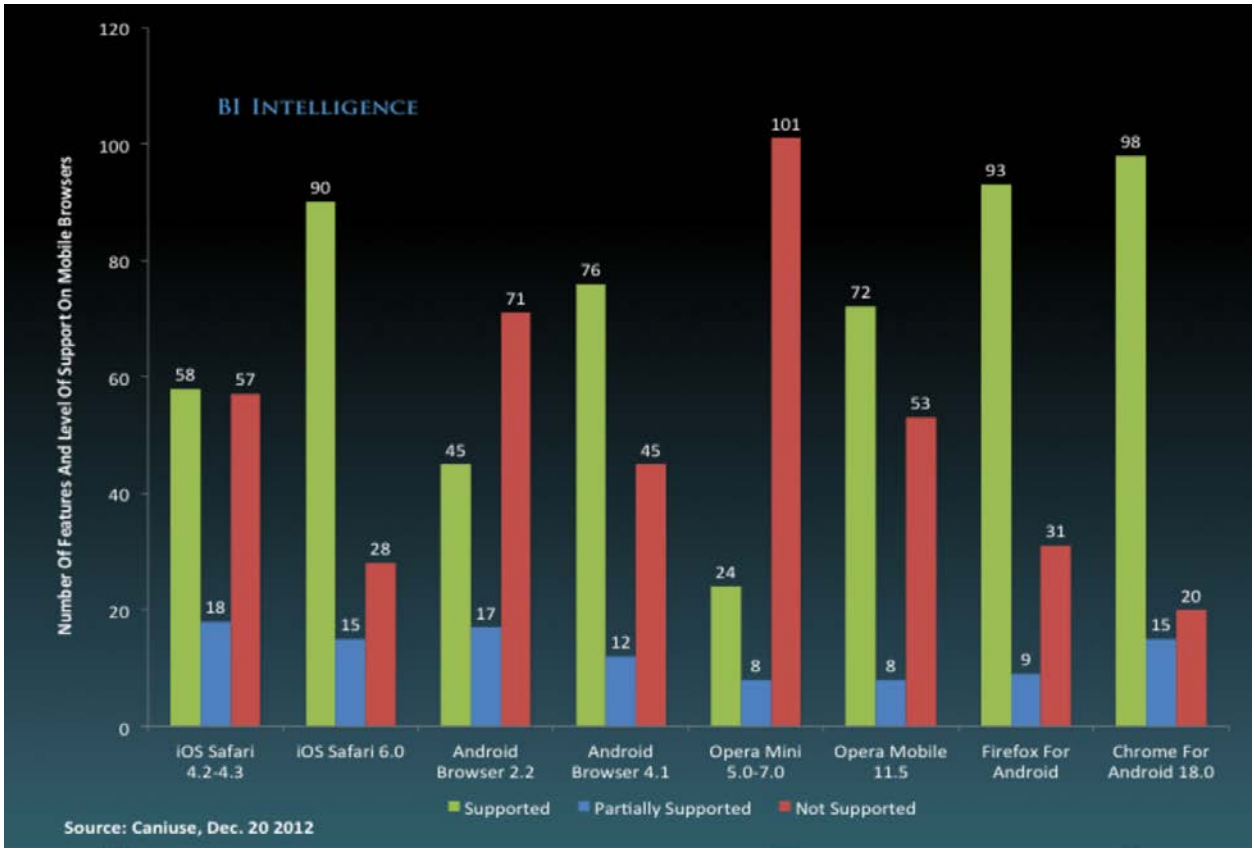


Рисунок 4. Количество поддерживаемых функций стандарта HTML5 в зависимости от веб-браузера

специфика верстки веб-страниц приводит к определенным трудностям: программные платформы по-разному реализуют некоторые функции отображения веб-страниц (рис. 4).

Основным преимуществом мобильных HTML5 приложений является возможность обновления контента и структуры приложения без необходимости выпуска отдельного обновления (сборки). Современная разработка HTML5 приложений упрощается благодаря таким библиотекам, как iScroll, JQuery Mobile, Sencha Mobile и др., которые предоставляют UI компоненты, по производительности не уступающие их нативным аналогам. Главным же недостатками HTML5 приложений являются:

- отсутствие защищенного локального хранилища данных;
- базовые UI компоненты;

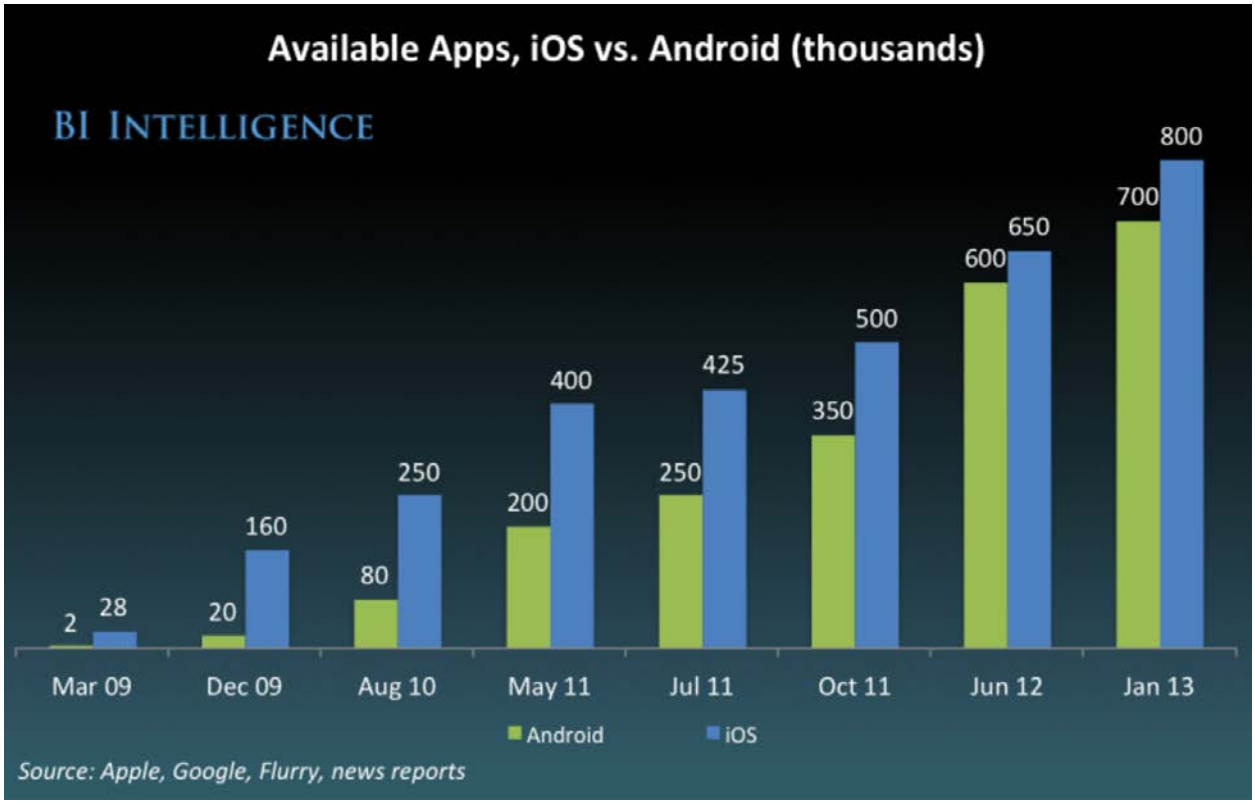


Рисунок 5. Объемы рынка мобильных приложений (тыс.)

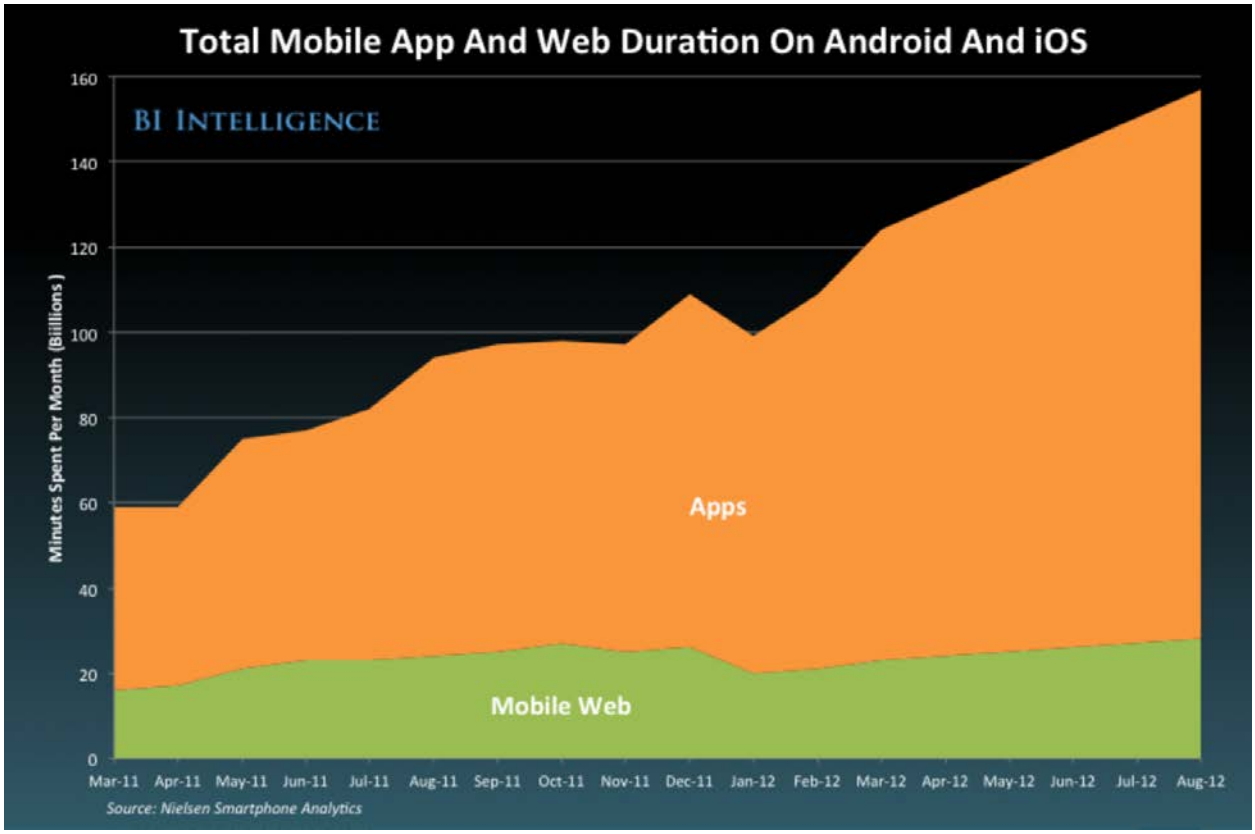


Рисунок 6. Распределение времени пользователя при использовании мобильных устройств

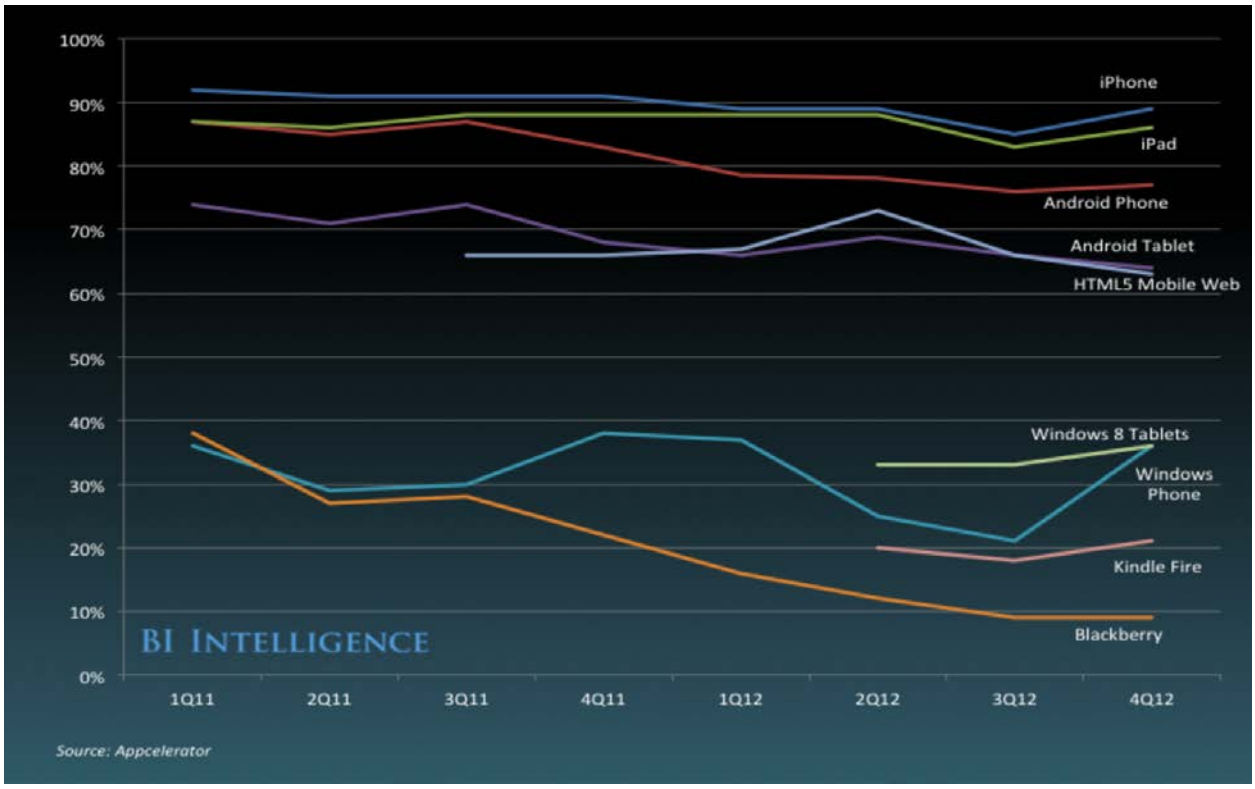


Рисунок 7. Распределение выпускаемых разработчиками мобильных приложений

- отсутствие оптимизации под конкретное устройство, что обычно ведет к потере производительности;
- отсутствие возможности централизованного распространения приложений: мобильные технологические платформы предоставляют “магазины приложений” (Apple AppStore, Google Play, Windows Marketplace) только для нативных продуктов;
- разные степени поддержки функционала HTML5 стандарта веб-браузерами мобильных операционных систем (рис. 4).

3. Гибридные приложения, то есть “приложения-обёртки”, которые получаются за счет внедрения HTML5 приложения внутрь нативного контейнера, что позволяет скомбинировать лучшие стороны обоих подходов.

   Существует два основных варианта сборки гибридных приложений:

   - локальная сборка - все HTML файлы разметки, а также необходимые файлы скриптов, стилей и ресурсов хранятся локально внутри мобильного приложения и не требуют соединения с сетью Интернет для полнофункциональной работы;
   - серверная сборка - внутри приложения посредством WebView компонента отображается контент, расположенный на заранее заданном сервере; в таком случае функционал приложения при отсутствии online-соединения сильно ограничен.

Согласно аналитическим отчетам [18] пользователи мобильных устройств отдают свое предпочтение нативным, нежели чем мобильным веб-приложениям (рис. 6). С другой стороны, эти же отчеты показывают нежелание разработчиков переходить на разработку HTML5 приложений (рис. 5, 7).

Созданная в рамках настоящей работы технология разработки мобильных

приложений подразумевает создание именно нативных приложений, но без необходимости написания программного кода и самостоятельной сборки готового приложения. Разработанная технологическая платформа подразумевает наличие клиентского мобильного приложения, которое может изменять структуру и содержание контента без необходимости повторной пересборки, а все изменения в структуре конфигурируются с помощью удаленного веб-приложения. Структура и содержание контента мобильного приложения хранятся локально стандартными для конкретной мобильной платформы способами. Далее приведен обзор существующих самых близких аналогов разработанной технологической платформы.

В настоящий момент на рынке существует несколько платформ-конструкторов генерации готовых мобильных приложений. Был проведен анализ каждого из представленных решений по следующим критериям:

1. поддерживаемые мобильные платформы - для каких мобильных платформ конструктор позволяет создавать мобильные приложения;
2. тип создаваемого приложения - различают нативное (средствами операционной системы, используя стандартные UI элементы), либо "обёрнутое" web-приложение, когда приложение, по сути, состоит только из web-view элемента, который показывает заранее загруженный HTML+JS+CSS контент;
3. настройка внешнего вида - какие настройки внешнего вида приложения доступны в конструкторе.

Ниже приведен краткий обзор каждого из приложений. Таблица 1 показывает результаты проведенного анализа в соответствии с выбранными критериями.

Kitapps - attendify

Web-сайт: http://attendify.com/

Платформа-конструктор, позволяющая создавать iOS приложения для событий и социально-значимых мероприятий. Отличается высокой ценой создания приложения. Главным недостатком является малое количество шаблонов и почти полное отсутствие социальных инструментов.

My-apps.com

Веб-сайт: http://my-apps.com/

Платформа-конструктор, ориентированная на приложения для бизнес-организаций. Предлагаются решения для iOS и Android платформ. На данный момент существует 4 разных типа шаблонов в зависимости от типа компании, для которой создается приложение. Главным недостатком является малое количество шаблонов.

BiznessApps

Web-сайт: http://www.biznessapps.com/

Самая крупная платформа-конструктор, предлагающее решения сразу для нескольких (iOS, Android, Mobile Web) мобильных платформ. Структура приложения настраивается с помощью web-приложения, после чего появляется возможность собрать готовое приложение. Предлагаются шаблоны для бизнес-ориентированных предприятий. Основным недостатком платформы является то, что предлагаемое решение в итоге является не нативным приложением, а лишь оберткой мобильного web-приложения.

В итоге, главным преимуществом разработанного продукта является нативность создаваемых мобильных приложений, а также возможность создания произвольного шаблона отображения, что не предусмотрено ни в одном из рассмотренных продуктов.

Таблица 1

Сравнение существующих похожих решений

	Bizness Apps	**Kitapps Attendify**	**My-apps.com**
Мобильные платформы	■ iPhone ■ Android Phone ■ HTML5	■ iPhone	■ iPhone ■ Android ■ HTML5 ■ WPhone
Тип создаваемого приложения	wrapped web	native	native
Настройка внешнего вида	цветовая схема, готовые шаблоны экранов	цветовая схема, готовые шаблоны экранов	цветовая схема, два варианта расположения элементов, готовые шаблоны экранов
Цена	$60 /месяц	$400-600 /приложение /год	$33 /месяц

Аналогично, был проведен анализ решений для разработки мульти-платформенных приложений в соответствии со следующими критериями:

1. поддерживаемые мобильные платформы - для каких мобильных платформ конструктор позволяет создавать мобильные приложения;
2. тип создаваемого приложения - различают нативное (средствами операционной системы, используя стандартные UI элементы), либо "обёрнутое" web-приложение, когда приложение, по сути, состоит только из web-view элемента, который показывает заранее загруженный HTML+JS+CSS контент;

3. используемые языки программирования.

Ниже приведен краткий обзор каждого из приложений. Таблица 2 показывает результаты проведенного анализа в соответствии с выбранными критериями.

Adobe PhoneGap

Web-сайт: https://build.phonegap.com/

Технологическая платформа для создания мобильных приложений с использованием HTML, JavaScript и CSS. На выходе получается "обёрнутое" приложение с возможностью полноправно взаимодействовать с системными функциями (обычные браузерные обёртки не имеют доступа к системе в целом). Пользователю предлагается написать мобильное веб-приложение самостоятельно, после чего платформа поможет подготовить сборки для популярных мобильных платформ. Технологическая составляющая основана на открытой платформе Apache Cordova.

IBM Worklight

Web-сайт: http://www-03.ibm.com/software/products/en/worklight

Полноценный инструмент для разработки кросс-платформенных приложений. Позволяет создавать HTML5 и гибридные приложения для четырех популярных платформ. Основной упор делается на безопасность и стабильность работы. Основными потребителями продукта являются бизнесы Enterprise уровня, о чем говорит очень высокая цена использования. Предоставляется инструмент для разработки (IDE), а так же высокоуровневый API для кроссплатформенного доступа приложения к мобильной операционной системе. По факту использует Adobe PhoneGap, как технологическую платформу, и собственные плагины для IDE Eclipse.

Apache Cordova

Веб-сайт: https://cordova.apache.org/

Технологическая платформа, позволяющая создавать мобильные приложения, используя HTML, JavaScript и CSS. Предоставляет API для работы с системными функциями устройства, что вкупе с UI библиотеками в итоге позволяет создать почти полностью navite-приложение. Является платформой с открытым исходным кодом. Основное отличие от Adobe PhoneGap состоит в том, что Apache Cordova не предоставляет упрощенной возможности публикации приложений. Всю разработку нужно вести самостоятельно.

Appcelerator

Web-сайт: http://www.appcelerator.com/

Технологическая платформа, позволяющая создавать полноценные приложения. Предоставляет свои инструменты разработки, SDK и полноценный API для доступа к системе. Самое комплексное решение с точки зрения мульти-платформенной разработки из существующих на рынке. Требует усилий, связанных с разработкой приложения и изучением внутреннего языка разметки и стилей (подмножество XML и аналог CSS).

Таблица 2

Сравнение существующих платформ разработки приложений

	IBM Worklight	**Adobe PhoneGap**	**Apache Cordova**	**Appcelerator**
Мобильные платформы	■ iPhone ■ Android ■ WindowsPhone ■ Blackberry OS	■ iPhone ■ Android ■ webOS ■ WindowsPhone ■ Bada ■ Blackberry OS	■ Amazon Fire OS ■ Android ■ Bada ■ Blackberry OS ■ Firefox OS ■ iOS, Mac OS X ■ QT ■ Tizen ■ Ubuntu ■ Web OS ■ Windows ■ Windows Phone	■ iPhone ■ Android ■ HTML5 ■ WPhone ■ Blackberry OS
Тип создаваемого приложения	wrapped web + native	wrapped web + native	native + wrapped web	native + wrapped web

Используемые языки программирования	HTML, JavaScript, CSS	HTML, JavaScript, CSS	HTML, JavaScript, CSS	JavaScript, CSS-like + XML definitions of layouts
Цена	$39600 / год	1 приложение бесплатно $9.99 / мес	Беплатно	Платно, цены по запросу индивидуально для каждой компании

Все представленные решения предусматривают самостоятельное проектирование и разработку приложения. Суть всех продуктов заключается в экспорте написанной программы в упакованные под разные платформы мобильные приложения. Особенность же настоящего проекта заключается в отсутствии части написания программного кода: все взаимодействие между элементами интерфейса, структура и содержание контента конфигурируются на серверной стороне без необходимости непосредственной разработки.

Программная составляющая предусмотрена только в рамках интерфейса программной платформы, основываясь на котором сторонние разработчики смогут создавать свои модули для интеграции с системой контента. В течение настоящей работы одной из задач является разработка набора модулей для демонстрационного приложения, которые также будут использоваться как пример использования API технологической платформы.

3. ОПИСАНИЕ АРХИТЕКТУРЫ

3.1. Выбор основополагающего паттерна проектирования

Для реализации данной технологической платформы необходимо было определиться с ключевым архитектурным решением - выбор паттерна проектирования, который бы предусматривал высокий уровень абстракции, так как это диктуется главной особенностью настоящего проекта - кросс-платформенным решением для управления структурой и контентом мобильных клиентских приложений без необходимости повторной сборки.

Основные паттерны проектирования мобильных приложений обычно диктуются отдельными технологическими платформами. Компании-производители мобильных операционных систем создают собственные API для сторонних разработчиков согласно определенным паттернам, ожидая, что разрабатываемые приложения будут создаваться с учетом особенностей целевой платформы.

Одним из самых популярных на данный момент паттернов является Model-View-Controller [3] (MVC) (рис. 8). В рамках этого паттерна, все объекты в приложении принадлежат одной из трех ролей: модель (model), отображение (view), контроллер (controller). MVC определяет не только роли каждого из объектов, но также описывает способы их взаимодействия между

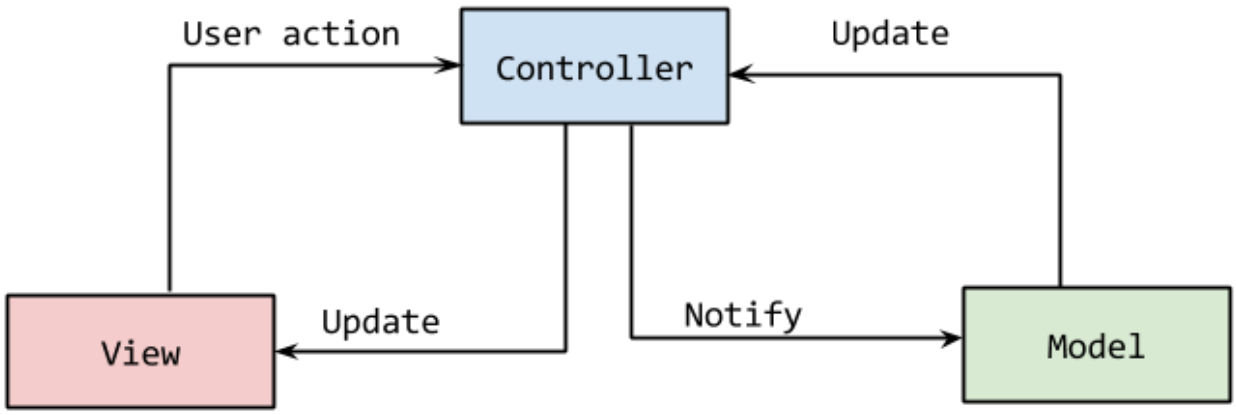


Рисунок 8. Model-View-Controller

собой. Объекты каждого из типов строго разделены между собой и могут общаться только по описанным интерфейсным каналам. Основные роли объектов в рамках паттерна Model-View-Controller:

1. **Model** - объекты модели инкапсулируют в себе представление данных какой-либо предметной области, а также определяют бизнес-логику и операции взаимодействия с данными (загрузку с удаленных ресурсов, из локальной базы, вычисления). Если модель спроектирована верно, то есть все данные и базовая логика заложены внутри объектов данной роли, главным преимуществом будет являться возможность простого переиспользования компонентов. Объекты модели могут также иметь связи между собой (например, один-к-одному, одним-ко-многим). В идеальном сценарии объекты модели должны быть полностью абстрагированы от объектов представления (view). При необходимости обновления представления, модель сообщает контроллеру о том, какие именно данные были обновлены, а уже контроллер определяет, какие именно представления и каким именно образом необходимо актуализировать.
2. **View** - объект представления, то есть того, что пользователь может непосредственно видеть на экране устройства. Объекты представления знают, каким образом они должны быть отрисованы на экране, а также определяют действия и события, на которые будут они могут реагировать при взаимодействии пользователя с приложением. Основное назначение view - отображение данных модели на экране и предоставление пользователю возможности для их создания и редактирования. При правильном проектировании, объекты представления являются переиспользуемыми и могут быть интегрированы в другие приложения. При обновлении данных модели контроллер соответственно обновляет представление, которое самостоятельно определяет, каким образом данные будут выглядеть на экране. При возникновении заданного события

взаимодействия с пользователем объект view посылает уведомление контроллеру, который в свою очередь может, например, обновить модель.

3. **Controller** - объект контроллера - главное связующее звено между моделью и ее представлением. Обычно, контроллер отвечает за актуализацию представления в соответствии с последними изменениями в модели, а также обрабатывает все события, генерируемые объектами view, преобразуя модель: внося изменения или создавая новые объекты. Таким образом, объекты представления являются полностью абстрагированы от конкретной предметной области и модели данных, которая, в свою очередь, независима от представления.

Основным преимуществом данного паттерна является независимость представления от модели, что в итоге дает высокую степень модульности и переиспользуемости компонентов. При проектировании архитектуры настоящей технологической платформы необходимо было, наоборот, связать модель и представление, минимизировав влияние контроллера. Такое требование объясняется универсальностью решения, которая невозможна при использовании контроллера в его классической роли, так как вся логика должна быть максимально описана на серверной стороне - что бы позволило сделать клиентские приложения независимыми и легко изменяемыми. Таким образом, использование MVC в чистом виде не представляется возможным в силу особенностей реализуемой системы.

Вторым рассматриваемым паттерном является Model-View-ViewModel (MVVM)[19] (иногда также называют Model-View-Binder). В литературе сюда относят и Presentation (Application) Model [16], который является популярным паттерном проектирования среди разработки мобильных приложений для популярных платформ (например, Windows Phone).

Основной особенностью паттерна MVVM является совмещение в себе функциональных преимуществ ставшего уже классическим для мобильных

(и веб-) приложений подхода MVC (Model-View-Controller) с преимуществами связывания данных (data-binding), то есть завязки атрибутов View на отдельные свойства модели через создания так называемых Binding, которые часто также предусматривают внутренние (в основном примитивные) конвертеры (например, для преобразования объекта типа date в строку даты, или для форматирования строк и чисел). Такой подход позволяет минимизировать код, связанный с частью контроллера из MVC.

Паттерн MVVM широко применяется в программных продуктах компании Microsoft (WPF, Silverlight), однако, стоит заметить, что сам по себе подход независим от платформы. В рамках настоящей работы сделана попытка воспроизведения данного паттерна проектирования в совмещении с описанным ранее паттерном MVC с использованием собственных инструментов связывания данных. В упрощенном виде паттерн представлен на рис. 9. Основные компоненты:

1. View - непосредственное графическое представление в виде UI элементов на экране устройства. По своей роли совпадает с ролью из MVC.
2. Model - модель представления какой-либо предметной области, бизнес-логика, данные. По своей роли совпадает с ролью из MVC.
3. ViewModel - абстрактное представление View, открывающее атрибуты для связки, а также описывающее возможные события, которые будут генерироваться при взаимодействии пользователя с приложением. В какой-то степени отчасти повторяет функциональное назначение роли контроллера из паттерна Model-View-Controller.
4. DataBinding (Binder) - указатель-связка между данными и непосредственным их представлением на графическом интерфейсе. Связки возможны благодаря описанным в ViewModel свойствам, к

которым Model может иметь доступ напрямую, минуя посредников.

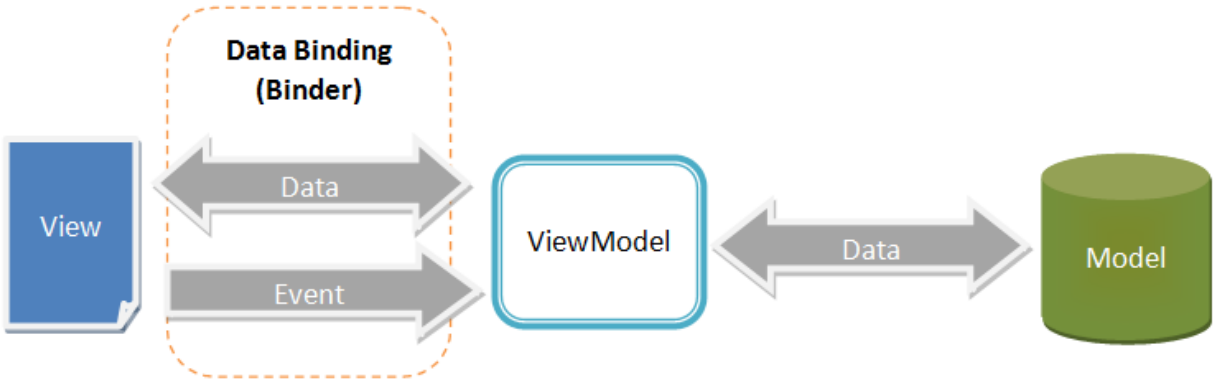


Рисунок 9. Паттерн Model-View-ViewModel (Binder)

Преимуществом данного подхода является наличие тесной связи между моделью и представлением посредством data-binding связки, что позволяет отображению динамически меняться при изменениях модели, минуя посредников. ViewModel здесь выступает, скорее, как контекст, который может обрабатывать события, а также указывает, какие именно данные должны быть связаны с отображением. Недостатком паттерна MVVM в рамках настоящей работы является необходимость для представления самостоятельно отслеживать изменения модели. Это ограничение удалось обойти при комбинации MVC и MVVM, таким образом применив при разработке технологической платформы основополагающего паттерна в виде модифицированного паттерна MVVM. Модификация заключается в присвоении ViewModel “обязанностей” роли контроллера из MVC, а именно - отслеживание изменений в модели и предоставление подходящей связки (data-binding) для представления. То есть комбинированный ViewModel и Controller выступает в роли менеджера указателей-связок, являясь посредником между моделью и представлением только по части преобразования значений в соответствии с указанным форматом, поиском ассоциированной связки, а также обработкой событий (например, нажатие кнопки) в соответствии с заданной конфигурацией реакции на них (рис. 10).

3.2. Реализация собственного паттерна проектирования

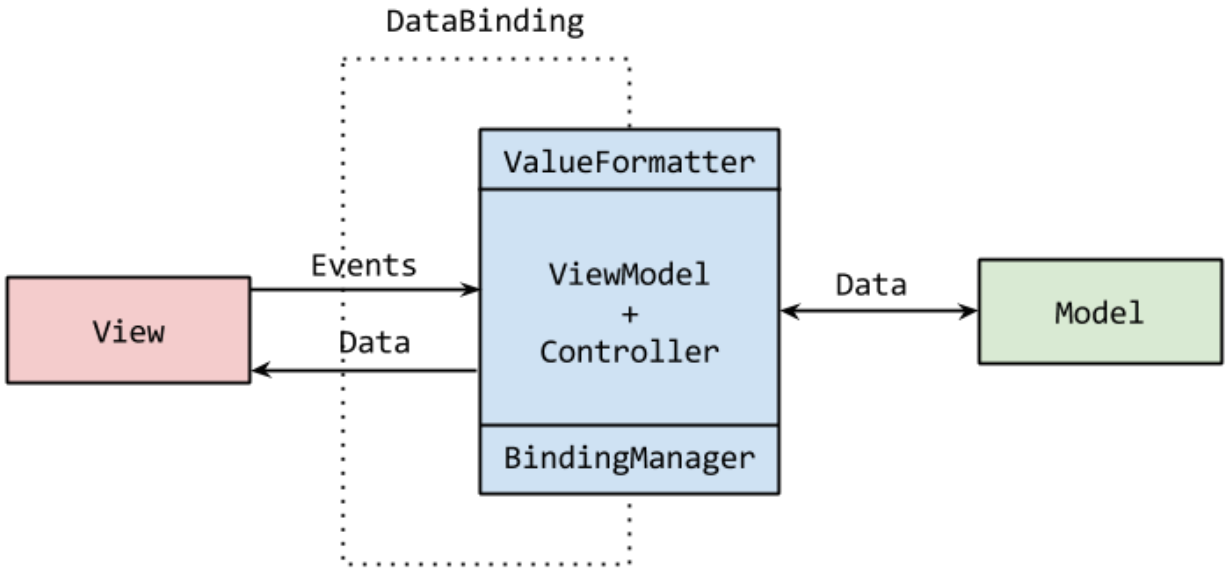


Рисунок 10. Комбинированный вариант MVC и MVVM

В рамках настоящего проекта был реализован модифицированный паттерн MVVM как в серверном, так и в клиентском решениях. Основными сущностями (см. Приложение 1) являются `View`, `Page`, `Action`, `Procedure`, `EntityDescription`, `Binding`.

`View`. Шаблон представления графического интерфейса пользователя в рамках клиентского приложения. Текущая реализация содержит следующие подклассы: `TextView, ImageView, Button, PdfView, WebView`. Архитектурное решение позволяет добавлять новые подклассы в модель, что в итоге автоматически масштабируется в веб-приложении - проектировщике интерфейса. Основные свойства класса описывают расположение на экране, размеры, цвет фона. Свойства каждого из подклассов описывают возможные параметры настройки отображения, а также предоставляют информацию об атрибутах, которые доступны для установления связей (data-binding) с моделью. Таблица 3 описывает доступные для настройки параметры отображения и атрибуты, для которых можно указывать связи.

Таблица 3

Доступные для настройки параметры отображения и атрибуты, предназначенные для связывания

Класс	Параметры отображения	Атрибуты для связывания

`View` - прямоугольная область	`Frame` - расположение на экране, включает в себя:
	`1) origin – x, y`
	`2) size – width, height`
	`Background color` - цвет фона	

`PdfView` - компонент отображения PDF файлов	Наследуются из `View`	`file` - pdf-файл, который будет отображаться
`ImageView` - компонент отображения изображений	Наследуются из `View` `ContentMode –` варианты отображения изображения: • `scale` - растягивание на всю площадь view • `fit` - растягивание с сохранением отношения сторон	`image_url` - URL картинки для отображения `image` - статическое изображение (placeholder)
`TextView` - компонент отображения текстовых данных	Наследуются из `View` `TextAlign –` расположение (выравнивание) текста: • `center` - по центру • `left` - по левому краю • `right` - по правому краю `TextStyle –` стиль (начертание) текста: • `regular` - обычный • `bold` - полужирный • `light` - тонкий • `italic` - с наклоном `TextSize –` размер (кегль) шрифта `TextColor –` цвет текста	`text` - текст для отображения
`Button` - компонент “кнопка”, доступная для нажатия пользователем	Наследуются из TextView	`title` - текст для отображения на кнопке `action` - реакция на нажатие кнопки

Концептуально данный класс и его подклассы представляют собой часть View паттерна MVVM, так как являются лишь шаблоном отображения

и хранят информацию о свойствах, доступных для связки. Список доступных `View` расширяется по мере внесения доработок в настоящий проект. В данной реализации он лишь подтверждает возможность существования данной части архитектуры.

Параметры отображения настраиваются при создании шаблонов отображения. Под “шаблоном” понимается созданное в рамках разработанного модуля веб-приложения для создания шаблонов (рис. 11) описание родительского view и, соответственно, его дочерних view с использованием вариантов параметров отображения. Созданный шаблон

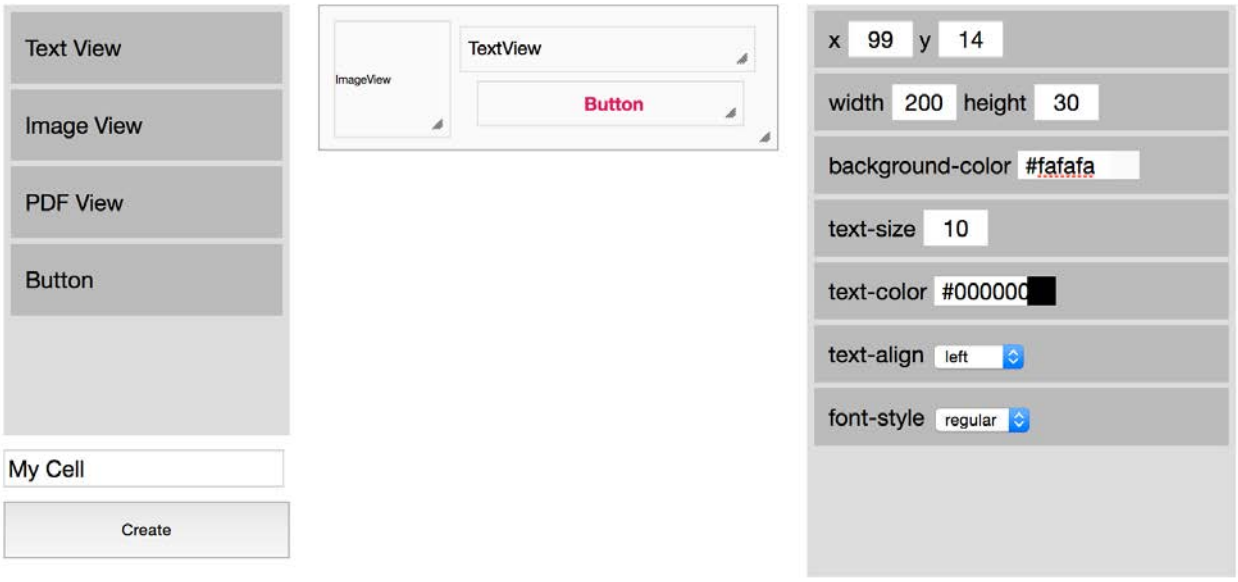


Рисунок 11. Модуль веб-приложения для создания описаний шаблонов view

может быть использован при настройки отображения отдельных страниц приложения как шаблон для всей страницы, так и как шаблон для отображения строки при использовании списков (как отдельного типа объекта Page).

`Page`. Представляет собой отображение отдельного экрана (страницы) в рамках клиентского мобильного приложения. Является представлением части ViewModel в рамках паттерна MVVM. Архитектура разработанной технологической платформы предусматривает добавление новых типов

экранов без значительных затрат. На данный момент момент реализованы следующие типы экранов:

- `Custom` - произвольный экран с отображением шаблона. Предусматривает связку на свойство `view`, где необходимо указать созданный ранее шаблон отображения, а также связку на свойство `item` - контекст данных для отображения.

- `List` - экран для отображения списка элементов. Предусматривает связку на свойство `item` - шаблон отображения строки элемента и связку на свойство `items` - контекст (массив) данных для отображения в списке.

- `Pdf` - экран для отображения файла формата PDF. Предусматривается связку на свойство `file,` необходимый для отображения на экране.

Итого, в рамках объекта Page предусматривается два типа указателей связок: контекст (data-context) и отображение (view). Первый тип связи определяется одним из двух вариантов поставщиков данных: выполнение заданной процедуры (объекта Procedure) или получение заданных данных согласно описанию объекта EntityDescription. Специфика работы каждого из вариантов описана ниже. Второй тип - view - ссылка на шаблон отображения, созданного с помощью модуля описания в рамках созданного веб-приложения.

При открытии страницы в результате сгенерированного события объектом Action в контексте данных также доступна информация из payload, переданного вместе с информацией о событии, указанной в объекте Action.

`Action`. Описание реакции элементов интерфейса на взаимодействие пользователя. Например, нажатие кнопки, либо успешный ввод данных. Является составной частью связки view с моделью, направленной на

генерацию событий и реакцию на них. В настоящем проекте предусмотрены три типа реакций на событие:

1. открытие внутренней URL - открывает заданный URL во внутреннем (не покидая приложения) веб-браузере;

2. открытие внешней URL - открывает заданный URL в стандартном браузере мобильной операционной системы;

3. переход на другой экран - производит переход на заданный Page с указанием payload - прикрепленных данных, которые будут расширять контекст открываемой страницы; с точки зрения структуры данных, payload - это набор пар ключ-значение, где значение может являться ссылкой на какое-либо поле текущего контекста данных.

Procedure. Составная часть связки ViewModel с Model. Является одним из источников контекста данных (Data Context), используемого при создании связок у объектов Page. В настоящем проекте Procedure (процедура) - создаваемые разработчиком динамические скрипты, которые при интерпретации с указанными аргументами генерируют объекты для их последующего представления. Созданные процедуры могут быть использованы в рамках приложения при определении контекста данных. При описании новой процедуры разработчику предлагается указать набор параметров (имя, тип, значение по умолчанию), которые также могут быть опциональными, и разместить написанный на языке Python исполнимый код, который будет интерпретирован при обращении клиентским мобильным приложением. Описанные параметры могут быть получены из мобильной операционной системы пользователя (в настоящее время доступно местоположение устройства, как один из вариантов параметра), либо будут заданы на основе контекста страницы, которая использует данную процедуру в качестве поставщика данных, или принимать произвольные значения. Загруженный программный код должен иметь основную функцию с именем

`func` и набором параметров, заданным при описании процедуры. Возвращаемое значение должно быть списком объектов (`list` из `dict`), чьи атрибуты могут быть использованы в указателях связках (data-binding) в частях шаблона отображения. При написании исполняемого скрипта разработчики могут пользоваться разработанным в рамках настоящей работы API, описание которого приводится в разделе 3.2. Данные, поставляемые процедурами принимаются как динамические и не могут быть доступны без активного интернет-соединения на клиентском мобильном приложении.

EntityDescription. Аналогичная по назначению Procedure составляющая модели. Основное отличие состоит в том, что контент статический и создается модератором/администратором приложения в рамках серверного веб-приложения, а затем синхронизируется с клиентскими приложениями с помощью локального хранения и кеширования на удаленной стороне. Особенностью этой модели является runtime-генерация классов (согласно созданному описанию), а также создание их экземпляров, которые хранятся в локальном хранилище посредством стандартных для конкретной платформы методов хранения объектов (в основном - ORM решения). Подробное описание runtime-генерации классов, а также особенностей описания модели и создания объектов на ее основе, приведено в разделе 3.6.

Binding. Основная сущность-связка, которая обеспечивает соединение свойств View и атрибутов модели, полученной через EntityDescription или Procedure, а также используется для установки связей между объектами Page и шаблонами отображения и между объектами Procedure и их аргументами (при их наличии). Текущая архитектура абстракции предусматривает полное переиспользование компонентов связки. В настоящем проекте используются связи следующих типов:

1. TypeValue - связь между значением атрибута элемента модели (data context) и параметром свойства, доступным для связывания у

элемента отображения (view). Параметрами данного типа связи являются:

- view - объект шаблона отображения, на свойство которого будет установлена связь;
- property - свойство элемента отображения, к которому проводится связь (например, text у TextView);
- value - значение, которое будет принимать указанное свойство; при установке значения может быть использован любой элемент текущего контекста данных (payload для объекта Page или атрибут модели данных); полное описание вариантов установки значений для Binding находится в разделе 3.1.1.

2. TypeTemplate - связь между значением атрибута объекта Page, доступным для связывания с шаблоном, и объектом шаблона отображения (например, item у объекта Page с типом List). Параметрами данного типа связи являются:
   - property - имя атрибута объекта Page, для которого осуществляется связь;
   - view - объект шаблона отображения, созданный с помощью конструктора описания в рамках разработанного веб-приложения.
3. TypeProcedure - связь между объектом Procedure, как элементом контекста данных, и атрибутом объекта Page, доступным для указания контекста данных (например, свойство items объекта Page с типом List). Параметрами данного типа связи являются:
   - procedure - процедура, используемая для генерации объектов контекста данных;

- property - имя атрибута объекта Page, для которого осуществляется связь.

4. TypeArgument - используется в комбинации с TypeProcedure типом связи. Содержит информацию о значениях аргументов процедуры, передаваемых при ее вызове в контексте отдельного объекта Page. Параметрами данного типа связи являются:

    - procedure - процедура, для которой предназначен данный аргумент;
    - argument - аргумент, значение которого будет указано в связи;
    - value - значение аргумента, указанное произвольно или с использованием элементов текущего контекста (заданного payload объекта Page); полное описание вариантов установки значений приводится в разделе 3.1.1.

5. TypeAction - связь между элементом шаблона отображения, и вариантом реакции на событие, которое он генерирует при пользовательском взаимодействии (например, нажатие на кнопку - элемент отображения Button). Параметрами данного типа связи являются:

    - action - объект Action, содержащий в себе информацию о варианте реакции на событие и значение, которое будет использовано для этого перехода (либо payload для открытия страницы); полное описание вариантов установки значений приводится в разделе 3.1.1.;
    - view - элемент отображения, который генерирует событие в результате пользовательского взаимодействия.

6. TypeEntity - связь между описанием модели данных EntityDescription

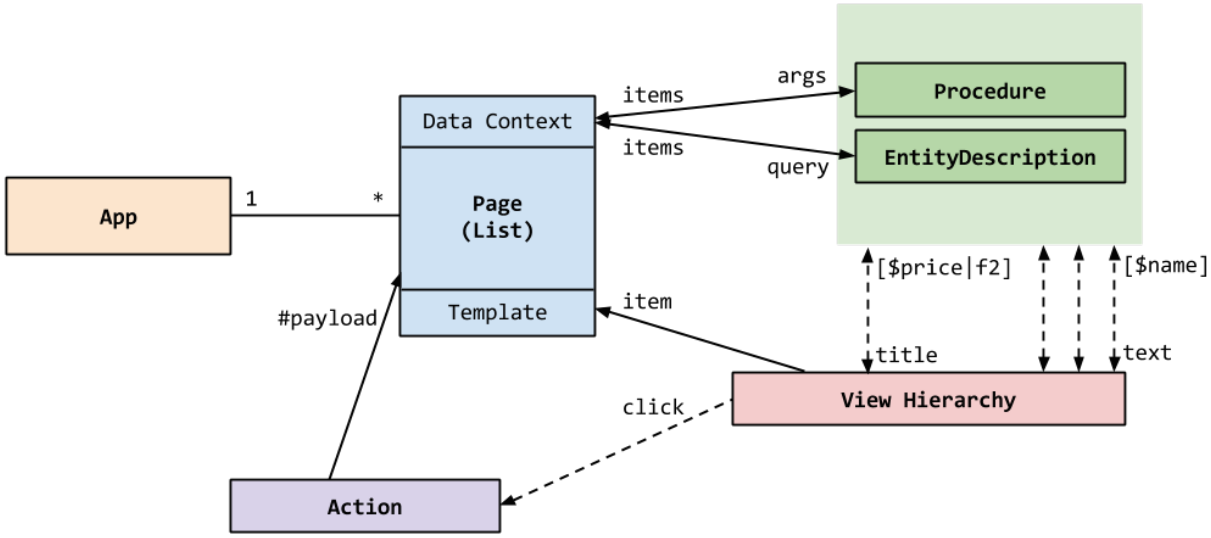


Рисунок 12. Схема связей для объекта Page типа List

и атрибутом объекта Page, для которого доступна установка связи контекста данных (например, items у объекта Page с типом List). При установке связи в последствии значением атрибута объекта Page будет являться список объектов, соответствующих описанию модели EntityDescription и созданных в рамках модуля разработанного веб-приложения. Параметрами данного типа связи являются:

- entityDescription - описание модели данных, в соответствии с которым будут передаваться объекты;
- query - опциональный запрос на языке SQL (фильтр - WHERE), который будет выполнен при получении данных из локального хранилища; полное описание вариантов установки значения запроса приводится в разделе 3.1.1.

Пример схемы связей объекта Page с типом List изображен на рис. 12. Все изображенные связи (кроме связи `App(1)—(*)Page`) являются объектами типа Binding с разными типами и набором параметров, описанными выше. Сериализованная схема будет являться входными данными для клиентского мобильного приложения, которое будет строить свою структуру и

содержание контента в соответствии с описанием схемы.

3.2.1. Варианты установки значений для объектов связей

В рамках разработанной технологической платформы создания мобильных приложений основополагающим объектом связей является Binding - сущность, отвечающая за присвоение значений между частями контекста данных и элементами отображения с опциональной фильтрацией получаемых объектов (при указании условий для запроса к локальной базе данных приложения или при указании параметров исполняемой процедуры), процедурами и значениями их аргументов, а также дополнительных данных (payload) при открытии страницы (page) в результате пользовательского взаимодействия.

Контекст данных можно разделить на два типа:

1. Контекст данных самой страницы (page) отображения: пары ключ-значения, переданные при выполнении инструкций объекта Action при реакции на событие.
2. Контекст данных, определенный после выполнения процедуры или при получении значений объектов модели в соответствии с каким-либо описанием EntityDescription.

При использовании значений из первого контекста ключевым символом является символ “#”, второго - символ “$”. В общем случае устанавливаемое значение принимает следующий вид:

```
Some text [$some_property], [#payload_key], format [$date|time]
```

где интерес представляют значения, заключенные в `[ ]`, которые учитываются при обработке значения. При преобразовании значений поиск в строке осуществляется регулярными выражениями вида: `\[\$[A-Za-z]+(?:\|[A-Za-z]+)?\]` и `\[#[A-Za-z]+(?:\|[A-Za-z]+)?\]`, затем определяется источник необходимого значения (объект модели или payload

текущей страницы) и происходит подстановка с возможным преобразованием, если оно указано после символа “|”, который используется как разделитель между именем атрибута и функцией форматирования значения. Доступные функции форматирования значений указаны в таблице 4. Таким образом, преобразованные с заданными функциями значения замещаются в исходной строке, формируя результирующее значение, которое становится значением свойства, указанного в целевом объекте отдельного Binding: атрибуты элементов отображения (например, `text` у `TextView` или `title` у `Button`), значения пар в составе payload, передаваемого в объект Page при реакции на Action, части значения URL (или полное значение), указанные при реакции на Action, запрос к локальной базе данных при формировании контекста из модели EntityDescription, аргументы, необходимые для вызова процедуры с целью формирования контекста данных.

Таблица 4.

Доступные функции форматирования значений для преобразования в рамках установленных связей

Название функции	Тип входного значения	Описание функции
date	datetime	Вывод только даты
time	datetime	Вывод только времени
fXX	double	Округление нецелого значения до XX знаков после запятой
ceil	double	Округление до целого в верхнюю сторону
floor	double	Округление до целого в меньшую сторону
round	double	Округление по правилам округления
len	string	Длина строки

3.3. Описание работы исполняемых процедур, как инструмента динамической генерации контекста данных

Один из источников формирования контекста данных в рамках данной технологической платформы являются объекты типа Procedure (процедура), которые описываются набором параметров и программным кодом на языке Python. Заданные процедуры могут быть использованы в связях при установке data-context элементам Page, например, при задании `items` для объекта Page с типом List.

Параметры (аргументы), используемые при описании процедур характеризуются следующими свойствами:

1. Имя параметра - при передаче параметров в функцию, написанную на языке программирования Python используется механизм передачи keyword-args (`**kwargs`), при котором аргументы передаются в виде пар ключ-значение. Такое решение обеспечивает необходимый для создания универсальных процедур уровень абстракции.

2. Тип - может быть custom, то есть передаваемое извне произвольное значение (в соответствии с преобразованием), либо device-specific, то есть зависит от устройства и настроек пользователя. На данный момент доступен из второго типа доступно только “местоположение пользователя”.

3. Значение по умолчанию - нужно в случае, если значение извне не было передано.

4. Флаг опциональности - значение параметра может быть опциональным, в таком случае будет использоваться значение по умолчанию.

Программный код в конечном счете должен объявлять глобальную

функцию `func` с параметрами, указанными при создании описания процедуры (порядок в данном случае не важен), и возвращаемым значением в виде массива объектов модели, чьи свойства могут быть использованы как составные части объектов Binding. При разработке исполнимого скрипта допускается использование всех стандартных функций и модулей Python 2.7 плюс разработанного в рамках настоящей технологической платформы программного API-обёртки для доступа к внешним ресурсам, кэшированию и хранению объектов на серверной базе данных. Описание API приводится в Приложении 2. Примеры программных скриптов на языке Python с использованием описанного API приводится в Приложениях 3, 4. Указанные примеры демонстрируют полный цикл получения и преобразования контента, готового к отображению (с использованием запросов удаленных ресурсов, а также соединения с удаленной MySQL базой данных). Также в рамках настоящей работы были разработаны процедуры, доступные для общего использования, получающие данные из различных социальных сетей:

1. get_twitter_feed - получение ленты сообщений пользователя из социальной сети Twitter (http://twitter.com). Имя пользователя передается в качестве аргумента.

2. get_instagram_feed - получение ленты фотографий пользователя из социальной сети Instagram (http://intagram.com/). Имя пользователя передается в качестве аргумента.

3. get_vk_feed - получение новостей со стены группы из социальной сети ВКонтакте (http://vk.com/). Идентификатор группы передается в качестве аргумента.

3.4. Суммирующее описание предлагаемого архитектурного решения

Предлагаемое архитектурное решение было полностью реализовано в виде веб-приложения для настройки структуры и содержания контента. Уникальность описываемой архитектуры заключается в возможности переиспользования ее компонентов, а также, при наличии клиентского мобильного приложения, в полной мере реализующего всю логику серверной части, решение позволяет модифицировать структуру и содержание контента без необходимости повторной сборки и компиляции программы. При использовании технологии практически полностью исключается написание программного кода при разработке мобильного приложения: настройка структуры и непосредственная генерация контента осуществляется с помощью разработанного веб-приложения, использование которого не требует специальной квалификации пользователя. Применение навыков программирования потребуется только для написания собственных скриптов-генераторов динамического контента (Procedure), однако их создание не обязательно должно быть связано непосредственно с разработкой приложения на основе данной технологии: в рамках технологической платформы разработчики могу размещать свои исполняемые скрипты доступные создателям приложений. В рамках данной работы было реализовано несколько общих скриптов, направленных на получение данных пользователей из социальных сетей, и доступных для использования в любом разрабатываемом приложении.

Таким образом, для создания мобильного приложения с использованием разработанной технологии, необходимо, взаимодействуя с веб-приложением, выполнить следующие шаги:

1. описать предметную область, создав соответствующие описания модели EntityDescription - необходимо определить набор классов-сущностей, свойства объектов которых будут использованы при

отображении данных;

2. при необходимости добавить свои скрипты генерации динамического контента (либо выбрать из каталога имеющихся процедур);
3. создать шаблоны отображения view, настроив свойства каждого элемента;
4. создать набор объектов Page (экранов приложения) с использованием доступных типов, указав контекст данных (объекты модели в соответствии с описанием какой-либо EntityDescription, получаемые с использованием заданного запроса фильтрации, или процедура генерации динамического контента с заданными параметрами) и шаблон их отображения;
5. расставить связи (data-bindings) между свойствами объектов модели и атрибутами элементов отображения;
6. определить возможности перехода между экранами, используя связь Action у элемента Button, передавая необходимые части контекста данных следующему объекту Page;
7. создать объекты модели в соответствии с созданным описанием EntityDescription;
8. после этого полученные структура и содержание контента могут быть синхронизированы демонстрационным приложением, которое зафиксирует все данные локально в момент исполнения и сможет продолжить свою работу без активного интернет-соединения.

В рамках настоящей работы было также разработано демонстрационное мобильное клиентское приложение, которое способно синхронизировать произвольные структуру и содержание контента, конфигурируя на их основе локальное хранилище данных, что в итоге позволяет использовать полностью

нативное приложение без необходимости в соединении с сетью Интернет (кроме вызова процедур для динамической генерации контента). Описание разработанного демонстрационного приложения приводится в разделе 3.7.

3.5. Описание специфики runtime-генерации классов

Для описанной в данной работе архитектуры, в силу своей динамичности, невозможно заранее определить структуру и модель данных. Архитектура данного проекта предусматривает создание описания модели данных (под моделью понимается набор классов, как название плюс описание всех полей: название и тип), хранение ее описания и объектов, созданных на основе этой модели, в базе данных, а так же передача модели на клиентское приложение, которое должно сконфигурировать на ее основе локальное хранилище и представление объектов в форме каких-либо структур, доступных на платформе конкретного мобильного приложения. Таким образом, речь идет о формировании модели данных в момент выполнения (at runtime), что накладывает определенные ограничения, так как не все языки программирования позволяют runtime генерацию и описание классов. В рамках данной работы рассматривались возможности формирования модели данных в момент исполнения. Основные исследовательские данные были получены на основе платформы iOS и возможностей языка Objective-C, а также дополнительно был проведен анализ возможностей разработки подобных решений на таких мобильных платформах, как Windows Phone (язык программирования C#) и Android (язык программирования Java).

Выбор структуры сильно зависит от скорости работы, ведь именно скорость работы является основным преимуществом native-приложений перед html5- и hybrid-приложениями. В рамках настоящей работы был проведен сравнительный анализ потребления процессорного времени (CPU time) при обращении к полям объектов разными способами для разных структур данных. Замер производился посредством стандартной функции языка C - `clock_t clock(void),` которая возвращает количество потребленного процессорного времени приложением на момент вызова метода [4]. Было выбрано несколько вариантов возможного представления

объектов модели, а также способов обращения к их полям. Далее приводятся рассмотренные варианты и результаты проведенного сравнительно анализа.

1. Самым "наивным" подходом является представление объектов в виде такой структуры данных, как словарь (dictionary, map), где поля представлялись бы в виде пар ключ-значение. В рамках Objective-C за эту структуру данных отвечает класс `NSDictionary`, объекты которого могут являться представлением частей модели данных, описанной в приложении.

2. Обращение к полю объекта класса посредством стандартного метода всех объектов Objective-C `-(id)valueForKey:(NSString *)key`, который производит поиск значения поля по его строковому названию.

3. Обращение к значению поля объекта с помощью средств, доступных через библиотеку `<objc/runtime.h>`, которая является основным инструментом для работы с классами, объектами, свойствами, переменными и методами языка Objective-C.

В таблице 5 приведены результаты анализа, который заключался в замере затраченного процессорного времени на получение значения поля объекта тем или иным способом. Результаты рассчитывались 10 раз для 10000000 обращений (для получения достоверных средних значений) для получения строкового (NSString) значения поля объекта полноценного сформированного runtime Objective-C класса, либо через структуру NSDictionary. Диаграмма (рис. 13) показывает затраченное процессорное время на каждом из испытаний для каждого способа.

Таблица 5

Результаты анализа затраченного процессорного времени на получение значения поля объекта

Способ обращения к полю объекта	Затраченное процессорное время
Обращение через ключ к структуре NSDictionary	3641173
Обращение через стандартный метод Objective-C объекта `-(id)valueForKey:(NSString *)key`	3400599
Обращение через runtime с помощью метода `id object_getIvar(id, Ivar)` напрямую	371471
Обращение через runtime с помощью метода `id object_getIvar(id, Ivar)` через собственый метод объекта	1125425
Обращение через runtime с помощью метода `id object_getIvar(id, Ivar)` через статический метод	1840793

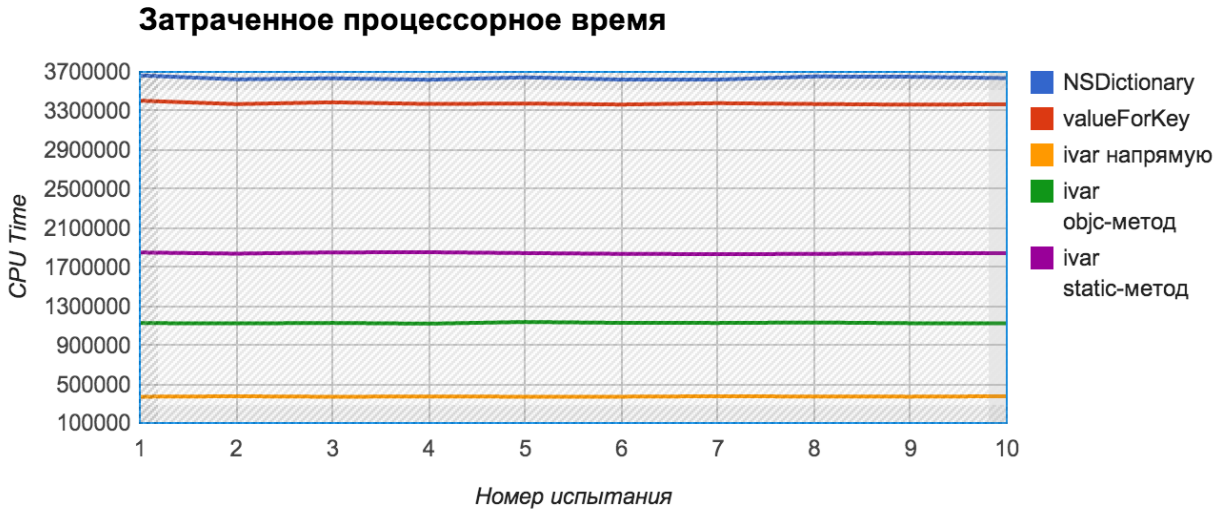


Рисунок 13. Затраченное на получение значения поля объекта процессорное время

Согласно полученным результатам сравнительного анализа, был выбран способ получения значений полей объекта напрямую через runtime-reflection посредством Objective-C метода аксессора (второй по производительности подход). Метод аксессор в данном случае был использован для удобства. Сигнатура метода имеет следующий вид: –

`(id)getValueForKey:(NSString *)key;` и возвращает значение, ассоциированное с указанным именем поля. Метод является аналогом стандартного метода `—(id)valueForKey:(NSString *)key,` однако работает заметно быстрее потому, что сразу производит поиск необходимого значения поля непосредственно у объекта класса, вместо предварительного поиска по методам [11].

3.6. Описание runtime возможностей программных платформ

Так как основная логика создания описания структуры и содержания контента располагается на серверной стороне (клиентское приложение должно быть максимально лишено собственной "логики", что позволяло бы обеспечить заданный уровень динамичности), в первую очередь необходимо было разобраться, каким образом будут описываться создаваемые модель данных используемой предметной области. Язык программирования серверной платформы Python был выбран не случайно: в силу "врожденной" динамической типизации создание описаний классов, добавление их объектам методов и полей является свойством платформы, что определенно облегчило работу. Программная платформа Google AppEngine, которая обеспечивает функционирование разработанного в рамках настоящей работы веб-приложения, предоставляет NoSQL хранилище данных [8], по своей структуре соответствующее полноценному ORM решению, и предоставляет все возможности для хранения как описаний моделей, так и экземпляров классов отдельной взятой модели. Встроенный метод `type(name, bases, dict)` позволяет объявить новый тип, указав имя, набор базовых классов, а также словарь для объявления полей и методов класса. Runtime-generated классы в настоящей работе наследуются от `ndb.Model` - класса, обеспечивающего хранение объектов в рамках NoSQL хранилища на платформе Google AppEngine. В настоящем проекте реализовано динамическое создание классов, а так же объектов этих классов и их непосредственное хранение в NoSQL хранилище Google AppEngine - Cloud Datastore.

Динамическое runtime- создание классов на платформе iOS на языке Objective-C доступно с использованием методов упомянутой выше библиотеки `<objc/runtime.h>`. Objective-C является Open-Source (не смотря на авторство компании Apple) языком [13, 18], который по своему

устройству изначально является runtime-ориентированным: например, решение о вызове того или иного метода (отсылке сообщения, в данном конкретном случае), будет принято непосредственно во время исполнения, нежели чем во время компиляции (compiling) или связывания (linking). Это позволяет динамически определять метод, который будет вызван при посылки того или иного сообщения, а также, например, подменять реализацию любого метода своим собственным. По факту, Objective-C Runtime - Runtime Library, написанная на C и Assembler, которая добавляет Объектно-ориентированную сущность и возможности языку C, что в конечном счете формирует Objective-C, то есть производит подгрузку классов, обеспечивает создание dispatch таблиц методов и перенаправление сообщений. Все создаваемые в рамках настоящей работы динамически классы являются подклассами класса Entity (в свою очередь является подклассом `NSManagedObject`, использующимся для удобного хранения создаваемых объектов в локальном ORM хранилище платформы iOS), который предоставляет удобные методы для прямого доступа к полям. В настоящем проекте в рамках разработанного прототипа мобильного приложения реализовано динамическое создание классов по описанной серверным приложением модели данных, определение на их основе структуры ORM базы данных, с помощью которой осуществляется их непосредственное хранение.

Также в настоящей работе одной из задач являлось проведение анализа возможности реализации подобного решения и построения динамической runtime архитектуры на других лидирующих [7] мобильных платформах: Android (Java) и Windows Phone OS (C#, .NET).

Программная платформа .NET, используемая для разработки мобильных приложений для мобильной операционной системы Windows Phone содержит стандартную библиотеку System.Reflection.Emit [6], которая позволяет определять assemblies - классы с заданной моделью данных в

момент исполнения - и регистрировать их в текущем runtime-окружении для создания экземпляров. Из отличительных особенностей библиотеки можно выделить возможность генерации исполняемых файлов с сгенерированными классами, которые можно использовать повторно либо в других приложениях. Платформа позволяет определять динамические классы, конфигурировать набор типизированных полей и добавлять методы, а так же настраивать на основе созданных типов модель и структуру локальной ORM базы данных (использующей LINQ), что обеспечивает хранение создаваемых экземпляров.

Программная платформа Android с использованием языка Java так же предусматривает возможность динамического создания классов в соответствии со структурой модели. Однако класс получается скомпилированным - то есть полноценная динамика невозможна, но это и не рассматривалось в рамках данной работы. В соответствии с поставленными задачами - динамически созданного скомпилированного класса будет достаточно. За динамическую компиляцию отвечает появившийся в Java 1.6 пакет `javax.tools` [15]. Он предоставляет API доступа к JavaCompiler, который позволяет скомпилировать класс во время исполнения и в комбинации с ClassLoader-ом может предоставить доступ к созданию экземпляров, вызову методов для них, а также доступу к полям. Не смотря на то, что Android не предоставляет стандартных ORM решений для хранения объектов, можно либо использовать open-source решения (такие, как, например, ORMLite [14]), либо использовать простую обертку над встроенным API для работы с SQLite. Подобное решение в настоящее время применяется в большом Enterprise решении - Hibernate [9], которое в свою очередь основано на JavaAssist [10] - библиотеке, которая является программной оберткой над ClassLoader в комбинации с инструментами из `javax.tools`.

3.7. Динамическое описание модели

При описании модели необходимо указать уникальное в пределах одного приложения имя модели, а также указать набор атрибутов (имя-тип), которыми будут обладать экземпляры описанного класса.

Текущая реализация динамического описания модели предусматривает следующие типы атрибутов объекта:

1. строковое значение (string);
2. целое число (long);
3. нецелое число (double);
4. дата и время (datetime);
5. булевое значение (boolean);
6. изображение (image - хранится как ссылка на изображение в файловом хранилище).

Отдельного внимания требует вопрос обновления модели, так как именно этот немаловажный аспект является "узким местом" любой универсальной кросс-платформенной системы. В рамках настоящей работы предусмотрены следующие варианты изменения созданной модели (с допущениями):

1. добавление атрибута;
2. удаление атрибута;
3. изменение типа атрибута: варианты преобразований значений описаны в таблице 6;
4. изменение имени атрибута: при этом будет произведено создание

атрибута с новым значением, копирование значений, а затем удаление старого атрибута.

Таблица 6

Варианты преобразований значений атрибутов при смене типа

	string	long	double	datetime	boolean	image
string		попытка преобразования, либо 0	попытка преобразования, либо 0.0	попытка преобразования по ISO-8601 [5], либо NULL	True, если значение НЕ NULL, в противном случае - False	NULL
long	строковое представление, либо NULL		приведение типов, либо 0.0	преобразование unix timestamp, либо NULL	True, если больше 0, в противном случае - False	NULL
double	строковое представление, либо NULL	приведение типов, либо 0		NULL	True, если больше 0, в противном случае - False	NULL
datetime	представление по ISO-8601, либо NULL	unix timestamp, либо 0	0.0		False	NULL

`boolean`	NULL	1, если True и 0, если False, либо False	1, если True и 0, если False, либо False	NULL		NULL
`image`	NULL	0	0.0	NULL	False	

За каждым описанием модели закрепляется номер версии, а так же список изменений, которые необходимо произвести клиенту, чтобы привести модель в актуальное состояние. При изменении типа для каждой пары типов предлагается выбор способа преобразования значений в соответствии с таблицей 6. При синхронизации структуры и содержания контента, клиентское приложение выполняет описанные преобразования модели (при несовпадении версий) и приводит ее в актуальное состояние инкрементально. Это позволяет безопасно выполнять преобразования в фоновом режиме - при прерывании, миграция продолжится при следующей синхронизации (например, при запуске приложения) с последней успешно примененной версии.

3.8. Описание разработанного демонстрационного мобильного приложения

Одной из задач настоящей работы была разработка нативного мобильного приложения, способного изменять структуру и содержание контента без необходимо повторной компиляции и сборки.

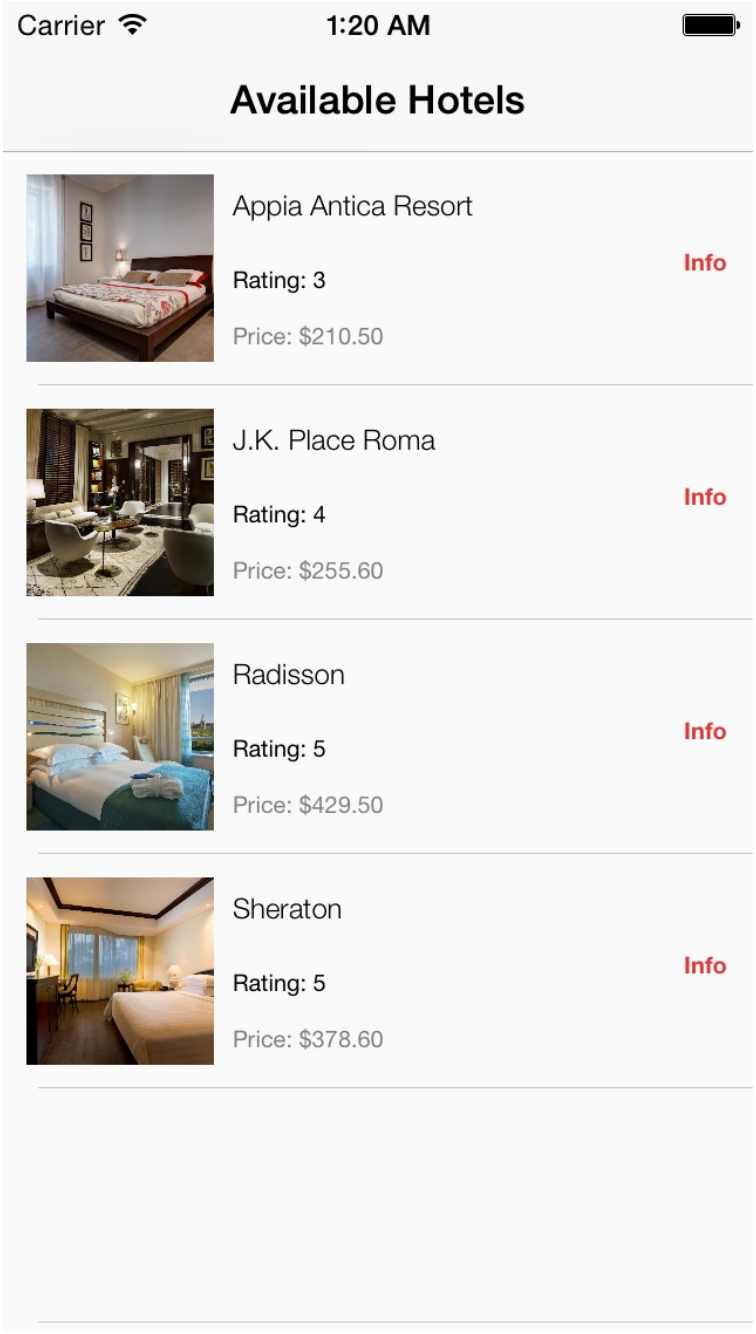


Рисунок 14. Разработанное мобильное приложение

Было разработано демонстрационное мобильное приложение на платформе iOS, которое в полной мере соответствует предложенной в данной работе технологии разработки кросс-платформенных приложений (рис. 14). При первом запуске необходимо указать идентификатор разработанного в рамках веб-конструктора приложения, чья структура и содержание контента будет синхронизированы мобильным клиентом. Модель данных клиентского решения полностью совпадает с моделью данных серверной части (Приложение 1), что в последствии позволяет модифицировать клиентское приложение при синхронизации данных с сервером. Синхронизация происходит в фоном режиме при запуске приложения, а так же при получении уведомления об изменении структуры или модели данных. Уведомления рассылаются всем установленным мобильным приложениям, после получения которых они самостоятельно скачивают с сервера актуальные настройки конфигурации. На рис. 15 изображена диаграмма последовательностей взаимодействия клиентского приложения с серверной частью.

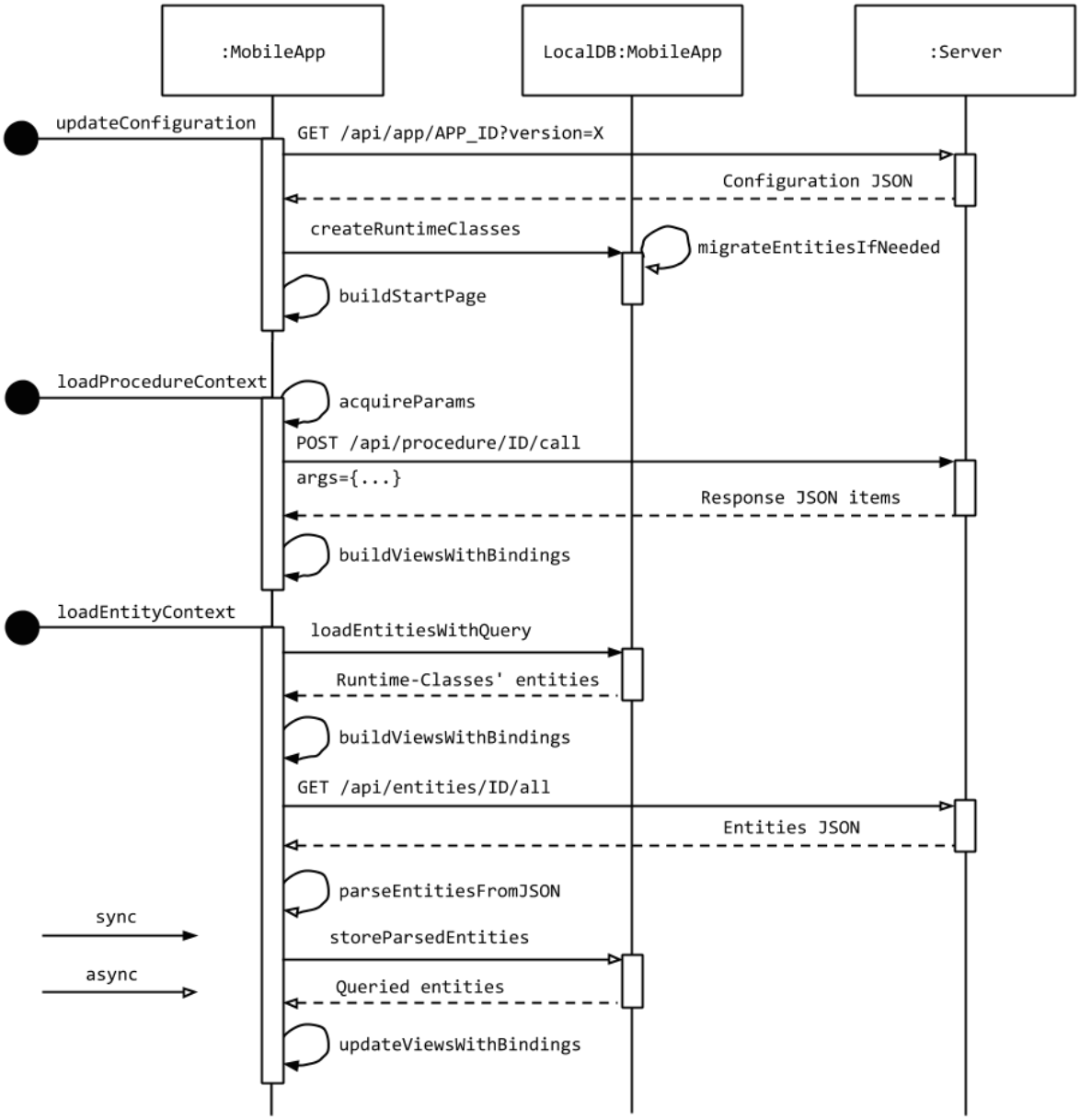


Рисунок 15. Диаграмма последовательностей взаимодействия мобильного приложения с серверной частью

4. ЗАКЛЮЧЕНИЕ

В настоящей работе предложена и реализована архитектура технологии разработки кросс-платформенных приложений с динамической структурой и содержанием контента.

Обзор существующих решений выделяет три основных способа разработки кросс-платформенный приложений: нативная разработка (в зависимости от конкретной мобильной платформы), HTML5 разработка и гибридное решение, комбинирующее оба подхода. Анализ показывает, что нативные приложения превосходят HTML5 и гибридные типы разработок, однако все кросс-платформенные аналоги такого вида предусматривают непосредственное разработку мобильного приложения, то есть написание программного кода. Уникальность же настоящей работы заключается в том, что для создания мобильного приложения, ориентированного на контент, не требуется наличие навыков программирования: структура и содержание контента полностью настраиваются в созданном веб-приложении - конструкторе.

Выбранный основополагающий архитектурный паттерн представляет собой собственную реализацию комбинированного решения, учитывающая плюсы Model-View-Controller и Model-View-ViewModel подходов к проектированию и разработке мобильных приложений. Использование данного комбинированного решения позволяет добиться высокого уровня абстракции, что делает отдельные компоненты полностью взаимозаменяемыми и переиспользуемыми. Структура Binding обеспечивает связи между элементами разработанной модели и является, по сути, центральным ядром системы.

В рамках настоящего проекта также разработана программная платформа, позволяющая сторонним разработчикам размещать исполняемые скрипты, результаты выполнения которых могут быть использованы как

источники контекста данных, доступных для отображения посредством связывания.

Отдельное внимание было удаление runtime-генерации классов и конфигурации встроенного локального хранилища в соответствии с заданным описанием модели данных. Использование именно runtime-создаваемых классов дает заметный прирост в производительности в сравнении с использованием структуры данных “словарь”, как способа представления экземпляров модели. Проведенный анализ вариантов реализации подобного подхода на других мобильных платформах показывает возможную масштабируемость решения. Немаловажным является возможность изменения описания модели, а также предусмотренные варианты миграции. Реализованный метод устанавливает высокий уровень гибкости модели, что немаловажно при разработке динамических систем такого типа.

Разработанное демонстрационное мобильное клиентское приложение доказывает работоспособность предложенной архитектуры и описанной технологии создания кросс-платформенных приложений.

Дальнейшее развитие технологии подразумевает создание клиентских приложений на основные мобильные платформы (Android, Windows Phone). Задача облегчается тем фактом, что для функционирования системы достаточно разработать только одно приложение, которое бы реализовывало все аспекты предложенной технологии. В силу динамичности решения, такое приложение может самостоятельно подстраиваться под модель данных и структуру контента, которые впоследствии могут изменяться без необходимости повторной компиляции или сборки. Так же предусматривается разработка встраиваемого компонента (SDK), который сторонние разработчики мобильных приложений смогут добавлять в свои проекты для быстрого прототипирования динамических модулей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. An Era of growth: The cross-platform report [Электронный ресурс] / nielsen аналитическая компания. Режим доступа http://www.nielsen.com/us/en/reports/2014/an-era-of-growth-the-cross-platform-report.html, по запросу (дата обращения 02.03.2014)

2. Brands' mobile apps aren't just about the discounts [Электронный ресурс] / eMarketer аналитическая компания. Режим доступа http://www.emarketer.com/Article/Brands-Mobile-Apps-Arent-Just-About-Discounts/1010100, свободный (дата обращения 02.03.2014)

3. Cocoa Core Competencies. Model-View-Controller [Электронный ресурс] / Apple компания разработчик мобильной ОС. Режим доступа https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html, свободный (дата обращения 11.03.2014)

4. clock [Электронный ресурс] / CPlusPlus - портал описания функций языка C++. Режим доступа http://www.cplusplus.com/reference/ctime/clock/, свободный (дата обращения 27.04.2014)

5. Date and Time formats [Электронный ресурс] / W3C - организация-разрабочтик стандарта WWW. Режим доступа http://www.w3.org/TR/NOTE-datetime, свободный (дата обращения 18.04.2014)

6. Emitting Dynamic Methods and Assemblies [Электронный ресурс] / MSDN портал описание библиотек платформы .NET. Режим доступа http://msdn.microsoft.com/en-us/library/8ffc3x75(v=vs.110).aspx, свободный (дата обращения 10.05.2014)

7. Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013 [Электронный ресурс] / Gartner аналитическая

компания. Режим доступа http://www.gartner.com/newsroom/id/2665715, свободный (дата обращения 28.03.2014)

8. Google Developers. Python NDB Overview [Электронный ресурс] / Google компания разработчик ПО. Режим доступа https://developers.google.com/appengine/docs/python/ndb/, свободный (дата обращения 15.04.2014)
9. Hibernate [Электронный ресурс] / Hibernate компания разработчик Enterptise решений для языка Java. Режим доступа http://hibernate.org, свободный (дата обращения 15.05.2014)
10. Javaassist [Электронный ресурс] / Javaassist портал описаня библиотеки. Режим доступа http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/, свободный (дата обращения 15.05.2014)
11. Key-Value Coding Programming Guide. Accessor Search Implementation Details [Электронный ресурс] / Apple компания-разработчик мобильной ОС. Режим доступа https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/KeyValueCoding/Articles/SearchImplementation.html, свободный (дата обращения 15.04.2014)
12. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development options [Электронный ресурс] / salesforce компания разработчик облачных решений для бизнеса. Режим доступа https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options, свободный (дата обращения 02.04.2014)
13. Open Source Reference Library [Электронный ресурс] / Apple компания-разработчик мобильной ОС. Режим доступа http://opensource.apple.com/, свободный (дата обращения 25.04.2014)

14. OrmLite - Lightweight Object Relational Mapping (ORM) Java Package [Электронный ресурс] / OrmLite библиотека для языка Java. Режим доступа http://ormlite.com, свободный (12.05.2014)

15. Package javax.tools [Электронный ресурс] / Java Platform SE 7 портал описания библиотек языка Java. Режим доступа http://docs.oracle.com/javase/7/docs/api/javax/tools/package-summary.html, свободный (дата обращения 15.05.2014)

16. Presentation Model [Электронный ресурс] / Martin Fowler. Режим доступа http://martinfowler.com/eaaDev/PresentationModel.html, свободный (дата обращения 10.03.2014)

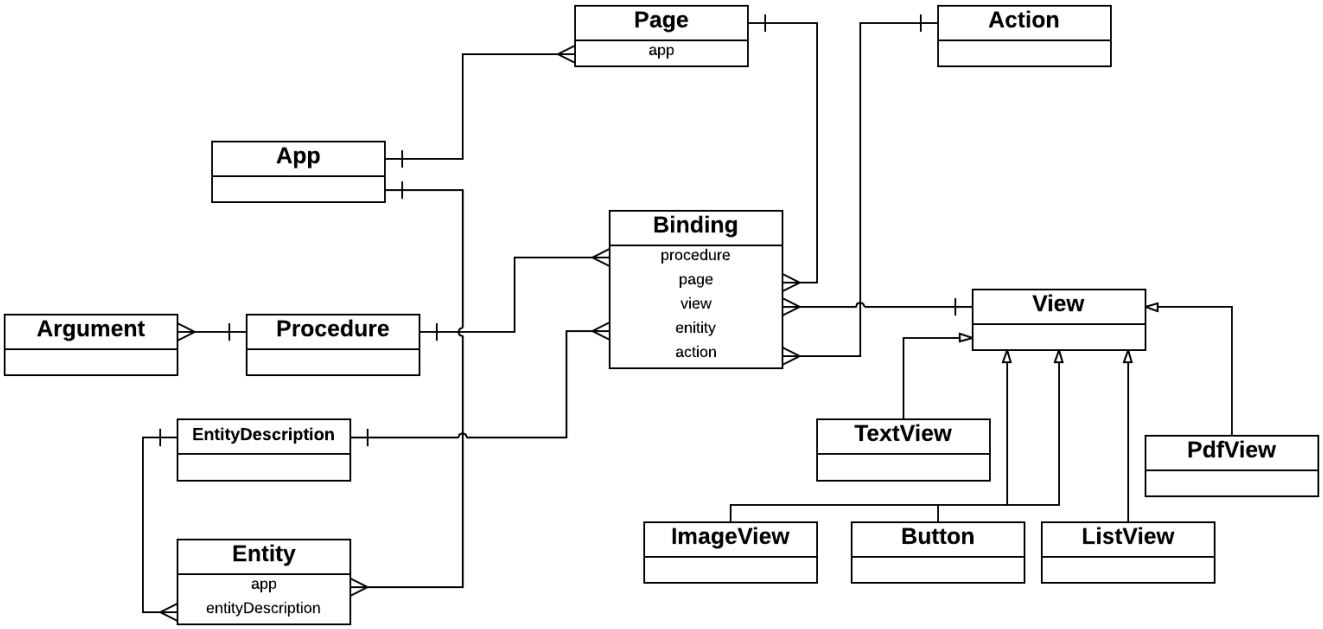
17. The Future of Mobile Development: HTML5 Vs. Native Apps [Электронный ресурс] / Business Insider аналитический портал. Режим доступа http://www.businessinsider.com/html5-vs-native-apps-for-mobile-2013-6?op=1, свободный (дата обращения 05.04.2014)

18. Understanding the Objective-C Runtime [Электронный ресурс] / Colin Wheeler. Режим доступа http://cocoasamurai.blogspot.ru/2010/01/understanding-objective-c-runtime.html, свободный (дата обращения 25.04.2014)

19. WPF Apps with the Model-View-ViewModel Design Pattern [Электронный ресурс] / MSDN Magazine портал для разработчиков. Режим доступа http://msdn.microsoft.com/en-us/magazine/dd419663.aspx, свободный (дата обращения 10.03.2014)

ПРИЛОЖЕНИЕ 1

МОДЕЛЬ ДАННЫХ СЕРВЕРНОГО РЕШЕНИЯ



ПРИЛОЖЕНИЕ 2

ОПИСАНИЕ API ДЛЯ СОЗДАНИЯ СКРИПТОВ ДЛЯ PROCEDURE

Основной пакет для доступа к API - `apps_api,` включает в себя следующие модули:

`external.py`

`fetch(url, method='GET', payload=None, headers={}, timeout=10)`

Совершает обращение по указанному URL.

`url` - URL, по которому необходимо выполнить запрос;

`method` - название HTTP метода для запроса;

`payload` - данные, которые должны быть включены в тело запроса;

`headers` - словарь HTTP-заголовков, которые необходимо прикрепить к запросу;

`timeout` - время, по прошествии которого невыполненный запрос будет прерван.

Возвращаемое значение - строка ответа от сервера

`cache.py`

`cache_obj(key, value, time=0)`

Кэширует заданное значение в памяти сервера.

`key` - ключ, по которому будет произведено кэширование значение

`value` - объект, который необходимо закэшировать

`time` - время, в течение которого объект будет храниться в кэше (либо всегда).

`get_cached_obj(key)`

Возвращает закэшированное значение. либо None, если по заданному ключу объект не был найден.

`key` - ключ, по которому необходимо осуществлять поиск.

`storage.py`

`class Model`

Класс, наследуясь из которого можно получать хранимые в серверной базе данных объекты. Подробная документация об использовании класса Model находится по ссылке: `https://developers.google.com/appengine/docs/python/ndb/`

ПРИМЕР 1 ИСПОЛНЯЕМОГО СКРИПТА ДЛЯ PROCEDURE НА ЯЗЫКЕ PYTHON

Получение и преобразование объектов из удаленной MySQL базы данных

```
import MySQLdb
import datetime

__author__ = 'Quiker'


def func(_id=None):
    def _serialize_value(val):
        if isinstance(val, datetime.datetime):
            return str(val).replace(' ', 'T')
        else:
            return val

    db = MySQLdb.connect(host='173.194.247.52',
                         passwd='rootpswd', user='root',
                         use_unicode=True, charset='utf8')
    cursor = db.cursor()
    cursor.execute('use app;')

    cursor.execute('SELECT * FROM Event %s;' % (('WHERE _id =
%ld' % long(_id)) if _id is not None else ''))

    rows = cursor.fetchall()

    result = []

    for row in rows:
        result.append(dict(zip(('_id', 'name', 'address',
'descr', 'date'), map(_serialize_value, row))))

    db.close()

    return result
```

ПРИМЕР 2 ИСПОЛНЯЕМОГО СКРИПТА ДЛЯ PROCEDURE НА ЯЗЫКЕ PYTHON

Получение и преобразование объектов-фотографий из социальной сети Instagram для заданного пользователя

```
import apps_api
import json
import urllib

__author__ = 'quiker'


def func(user):
    client_id = '4cf9d457f4e94932b2bd3c72484c6722'
    # user = '535783772'

    url = 'https://api.instagram.com/v1/users/%s/media/recent/?
%s'
                % (user, urllib.urlencode({
                          'client_id': client_id
                         }))

    result = json.loads(apps_api.fetch(url))
    data = result['data']

    photos = list()

    for d in data:
        photos.append({
            'image': d['images']['standard_resolution']['url']
        })

    return photos
```

ПРИМЕР СТРУКТУРЫ ПРИЛОЖЕНИЯ В ФОРМАТЕ JSON


```
{
    "app_key": "541288da8b8584726550a6698ec289bffa619b8c",
    "entity_descriptions": [
        {
            "id": "5066549580791808",
            "name": "Event",
            "properties": {
                "date": "datetime",
                "descr": "text",
                "name": "text",
                "address": "text"
            },
            "version": 3,
            "changes": [
                {
                    "to": 2,
                    "change": {
                        "type": 4,
                        "from": "description",
                        "to": "descr"
                    }
                },
                {
                    "to": 3,
                    "change": {
                        "type": 1,
                        "col": "address",
                        "col_type": "text"
                    }
                }
            ]
        }
    ],
    "name": "Отделение ПИ",
    "pages": [
        {
            "id": "6262818231812096",
            "type": 1,
            "name": "Event List",
            "bindings": [
                {
                    "id": "4855443348258816",
                    "query": "",
                    "entity": "5066549580791808",
                    "type": 2,
                    "property": "items"
                },
                {
                    "id": "5277655813324800",
                    "value": "Info",
                    "view": "5207287069147136",
                    "type": 0,
                    "property": "title"
```

```
},
{
    "id": "5418393301680128",
    "value": "[$name]",
    "view": "4644337115725824",
    "type": 0,
    "property": "text"
},
{
    "view": "5207287069147136",
    "id": "5840605766746112",
    "action": {
        "id": "4714705859903488",
        "payload": {
            "event_id": "[$_id]"
        },
        "type": 3,
        "value": "6333186975989760"
    },
    "type": 4,
    "property": "action"
},
{
    "id": "5981343255101440",
    "view": {
        "subviews": [
            {
                "subviews": [],
                "background_color": "0xfafafa",
                "id": "4644337115725824",
                "font": {
                    "style": 1,
                    "size": 12
                },
                "text_color": "0x0",
                "frame": {
                    "origin": {
                        "y": 9,
                        "x": 12
                    },
                    "size": {
                        "height": 34,
                        "width": 234
                    }
                },
                "type": 3,
                "align": 0
            },
            {
                "subviews": [],
                "background_color": "0xfafafa",
                "id": "5770237022568448",
                "font": {
                    "style": 2,
                    "size": 10
                },
                "text_color": "0x0",
                "frame": {
```

```
                    "origin": {
                        "y": 53,
                        "x": 12
                    },
                    "size": {
                        "height": 25,
                        "width": 209
                    }
                },
                "type": 3,
                "align": 0
            },
            {
                "subviews": [],
                "background_color": "0xfafafa",
                "id": "5207287069147136",
                "font": {
                    "style": 1,
                    "size": 10
                },
                "text_color": "0x4f6bdb",
                "frame": {
                    "origin": {
                        "y": 30,
                        "x": 202
                    },
                    "size": {
                        "height": 30,
                        "width": 100
                    }
                },
                "type": 5,
                "align": 2
            }
        ],
        "background_color": "0xfafafa",
        "id": "6473924464345088",
        "name": "Event Cell",
        "frame": {
            "origin": {
                "y": 0,
                "x": 0
            },
            "size": {
                "height": 100,
                "width": 320
            }
        },
        "type": 0
    },
    "type": 3,
    "property": "item"
},
{
    "id": "6544293208522752",
    "value": "[$address]",
    "view": "5770237022568448",
    "type": 0,
```

```
            "property": "text"
        }
    ],
    "isStart": true
},
{
    "id": "6333186975989760",
    "type": 0,
    "name": "Event Page",
    "bindings": [
        {
            "id": "4925812092436480",
            "query": "_id == [#event_id]",
            "entity": "5066549580791808",
            "type": 2,
            "property": "item"
        },
        {
            "id": "5136918324969472",
            "value": "RSVP",
            "view": "5348024557502464",
            "type": 0,
            "property": "title"
        },
        {
            "id": "5488762045857792",
            "value": "[$name]",
            "view": "4785074604081152",
            "type": 0,
            "property": "text"
        },
        {
            "view": "5348024557502464",
            "id": "5699868278390784",
            "action": {
                "id": "4573968371548160",
                "payload": {},
                "type": 2,
                "value": "http://myrsvp.com/?event=[$_id]"
            },
            "type": 4,
            "property": "action"
        },
        {
            "id": "6051711999279104",
            "view": {
                "subviews": [
                    {
                        "subviews": [],
                        "background_color": "0xfafafa",
                        "id": "4785074604081152",
                        "font": {
                            "style": 1,
                            "size": 10
                        },
                        "text_color": "0x0",
                        "frame": {
                            "origin": {
```

```
                    "y": 20,
                    "x": 43
                },
                "size": {
                    "height": 36,
                    "width": 226
                }
            },
            "type": 3,
            "align": 1
        },
        {
            "subviews": [],
            "background_color": "0xfafafa",
            "id": "5910974510923776",
            "font": {
                "style": 0,
                "size": 10
            },
            "text_color": "0x0",
            "frame": {
                "origin": {
                    "y": 81,
                    "x": 13
                },
                "size": {
                    "height": 178,
                    "width": 285
                }
            },
            "type": 3,
            "align": 0
        },
        {
            "subviews": [],
            "background_color": "0xfafafa",
            "id": "5348024557502464",
            "font": {
                "style": 1,
                "size": 10
            },
            "text_color": "0xa63030",
            "frame": {
                "origin": {
                    "y": 276,
                    "x": 108
                },
                "size": {
                    "height": 30,
                    "width": 100
                }
            },
            "type": 5,
            "align": 1
        }
    ],
    "background_color": "0xfafafa",
    "id": "6192449487634432",
```

```
                        "name": "Single Event Template",
                        "frame": {
                            "origin": {
                                "y": 0,
                                "x": 0
                            },
                            "size": {
                                "height": 480,
                                "width": 320
                            }
                        },
                        "type": 0
                    },
                    "type": 3,
                    "property": "view"
                },
                {
                    "id": "6614661952700416",
                    "value": "[$descr]\\nWill held at: [$address]",
                    "view": "5910974510923776",
                    "type": 0,
                    "property": "text"
                }
            ],
            "isStart": false
        }
    ]
}
```

ПРИЛОЖЕНИЕ 6

ОПИСАНИЕ СЕРВЕРНОГО API ТЕХНОЛОГИЧЕСКОЙ ПЛАТФОРМЫ

1. Получение конфигурации структуры приложения

GET /api/app/APP_ID
APP_ID - идентификатор приложения

Параметры:
* version (опционально) - номер последней синхронизированной локальной версии конфигурации, хранимой на клиенте

Результат:
JSON-сериализованное описание конфигурации и структуры приложения, включающее описание используемых страниц (Page) отображения, их связей, шаблонов, контекстов данных, а также описание моделей данных, необходимых для создания в рамках локального хранилища
Пример результата представлен в приложении 5.

2. Вызов удаленной процедуры

POST /api/procedure/PROCEDURE_ID/call
PROCEDURE_ID - идентификатор процедуры для вызова

Параметры:
* args - JSON-сериализованный набор пар ключ-значение для аргументов процедуры

Результат:
JSON-сериализованный массив элементов типа dictionary, представляющих собой объекты модели, используемые для отображения посредством связей.

3. Получение объектов модели в соответствии с описанием EntityDescription

GET /api/entities/ENTITY_DESCRIPTION_ID/all
ENTITY_DESCRIPTION_ID - идентификатор описания модели

Результат:

JSON-сериализованный массив элементов модели, используемых для отображения посредством связей и предназначенных для хранения в рамках локальной базы данных.

ПРИЛОЖЕНИЕ 7

ПРОГРАММНЫЙ КОД РАЗРАБОТАННОГО РЕШЕНИЯ

Ввиду большого объема приложения код разработанного решения приводится в электронном виде на компакт-диске, прикрепленном к настоящему документу.