САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Чижова Надежда Александровна

Проектирование и реализация серверной части визуальной среды

разработки в облаке

Дипломная работа

Допущена к защите.

Зав. Кафедрой:

д.ф.-м.н., проф. Терехов А. Н.

Научный руководитель:

ст. преп. Литвинов Ю. В.

Рецензент:

к.ф.-м.н., доцент Кознов Д. В.

Санкт-Петербург

2013

SAINT PETERSBURG STATE UNIVERSITY

Mathematics & Mechanics Faculty

Software Engineering Chair

Chizhova Nadezhda

Design and implementation of back-end for

cloud-based visual IDE

Graduation Thesis

Admitted for defense.

Head of the chair:

Professor A.N. Terekhov

Scientific supervisor:

Senior lecturer Y.V. Litvinov

Reviewer:

Associate Professor D.V. Koznov

Saint Petersburg

2013

Оглавление

[Введение 4](#_Toc357360727)

[1. Постановка задачи 6](#_Toc357360728)

[2. Обзор существующих решений 7](#_Toc357360729)

[2.1 AppInventor 8](#_Toc357360730)

[2.2 Buildanapp 9](#_Toc357360731)

[2.3 Выводы 10](#_Toc357360732)

[3. Используемые технологии 11](#_Toc357360733)

[3.1 Microsoft.NET: F# 11](#_Toc357360734)

[3.2 Microsoft.NET: WebSharper 12](#_Toc357360735)

[3.3 Microsoft.NET: C# 13](#_Toc357360736)

[3.4 Windows Azure 13](#_Toc357360737)

[4. Требования к архитектуре 14](#_Toc357360738)

[5. Архитектура системы 16](#_Toc357360739)

[5.1. Клиент 18](#_Toc357360740)

[5.2. Генератор 20](#_Toc357360741)

[6. Практическая реализация 21](#_Toc357360742)

[6.1. Формат обмена данными 21](#_Toc357360743)

[6.2. Репозиторий 22](#_Toc357360744)

[6.3 Сайт и веб-сервер 22](#_Toc357360745)

[6.4. Сервер приложений 23](#_Toc357360746)

[7. Сложности реализации 24](#_Toc357360747)

[7.1 Сайт и веб-сервер 24](#_Toc357360748)

[7.2 Сервер приложений 28](#_Toc357360749)

[7.3 Взаимодействие с «облаком» 29](#_Toc357360750)

[8. Апробация 30](#_Toc357360751)

[Заключение 32](#_Toc357360752)

[Список литературы 33](#_Toc357360753)

[Приложения 34](#_Toc357360754)

[Приложение А 34](#_Toc357360755)

[Приложение В 36](#_Toc357360756)

**Введение**

По прогнозам Gartner Group[[1]](#footnote-1), человечество стоит на пороге формирования нового поколения людей, которые будут неразрывно связывать свой быт и отдых с сетью, получая доступ к ней посредством многочисленных мобильных устройств. Благодаря этому в последнее время наблюдается быстрое развитие и совершенствование различных онлайн-сервисов и электронных магазинов, которые предлагают не только самые современные модели мобильных устройств, но и сложные приложения, делающие нашу повседневную жизнь значительно проще. Согласно аналитикам из компании ABI Research, общий объем рынка мобильных приложений на конец 2012 года превысил 30 млрд. долл. США[[2]](#footnote-2).

В связи с этим, возникла идея создания конструктора, с помощью которого можно, не имея навыков программирования, создавать (в пределах нескольких часов или минут) различные приложения для мобильных телефонов. Подобная технология была бы очень востребована у обычных пользователей, не знакомых с программированием. Так появился проект QReal:Web – браузерная среда разработки для создания приложений для мобильных телефонов.

QReal:Web является системой, позволяющей посредством палитры элементов создать мобильное приложение и затем сгенерировать инсталляционный пакет для телефона. Сформированный файл автоматически сохраняется на компьютере пользователя и впоследствии может быть передан на мобильный телефон. В рамках данной дипломной работы требовалось спроектировать, разработать и апробировать серверную часть конструктора QReal:Web.

QReal:Web изначально проектировался как браузерное приложение, так как веб-приложения являются более доступными для пользователей, чем десктопные. В связи с тем, что разрабатываемый конструктор ориентирован на большое количество пользователей, появилась идея разрабатывать не просто веб-приложение, а веб-приложение, работающее в облаке. Облачные вычисления – технология обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как интернет-сервис. Использование облака – один из способов значительно снизить затраты на дальнейшую поддержку IT-решений в связи с тем, что облака позволяют не заботиться о конфигурации и сбоях оборудования, не требуют мощного компьютера на стороне пользователя и ускоряют процесс обмена данными.

Данный конструктор является первым шагом на пути разработки metaCASE системы, которая будет по описанию языка автоматически генерировать визуальный редактор, генератор и другие средства инструментальной поддержки. Описание языка будет задаваться в виде метамодели.

**1. Постановка задачи**

Разрабатываемый конструктор должен удовлетворять следующим требованиям:

* Не требовать от пользователя знаний и опыта программирования для возможности создания приложения. Приложение должно создаваться путем перемещения мышкой элементов палитры на экранную форму – все настраивается и включается при помощи различных элементов палитры и опций.
* Поддерживать основные лидирующие на рынке мобильные операционные системы, такие как Android OS, Windows Mobile и Apple iOS.

Для обеспечения выполнения описанных требований в рамках моей дипломной работы были поставлены следующие задачи:

1. сформулировать требования к формату обмена данными между клиентской и серверной частью;
2. обеспечить обмен данными между клиентской, серверной частью и генераторами;
3. изучить возможности технологии WebSharper;
4. разработать сайт, используя WebSharper, на котором пользователь может зарегистрироваться, спроектировать свое мобильное приложение и сгенерировать соответствующий файл для мобильного телефона;
5. спроектировать и реализовать репозиторий для хранения диаграмм (приложений, созданных пользователем) и сервер приложений.

**2. Обзор существующих решений**

На данный момент на рынке существует ряд онлайн-сервисов, позволяющих создавать мобильные приложения с помощью визуального редактора: AppInventor, iBuildApp, Apps.ru, Kanchoo и другие. Такие средства предлагают пользователю инструмент для задания внешнего вида приложения и простейшей логики в виде указания переходов между экранами приложения. Для проектирования приложений в рассмотренных конструкторах используются различные графические языки. Приложения строятся путем объединения стандартных компонент.

 Недостатком таких сервисов является то, что они не поддерживают возможность задания нетривиальной логики приложения, например: агрегацию данных, работу с внешними источниками данных, авторизацию и т.д. Таким образом, функционал большинства существующих сервисов ограничивается созданием информационных приложений, которые не способны на взаимодействие с сервером или внешними источниками данных.

 Кроме того, многие из представленных на рынке сервисов позволяют создавать приложения только для одной определённой мобильной платформы. Например, проект Kanchoo позволяет создавать приложения исключительно для платформы iOS, а конструктор AppInventor – для платформы Android.

 К тому же в процессе работы в некоторых конструкторах, например, в AppInventor, требуется непрерывное подключение телефона к рабочему компьютеру. При отсоединении телефона от компьютера до окончания работы над созданием приложения проделанная работа не сохранится.

**2.1 AppInventor**



Рис. 1. AppInventor

Визуальная среда разработки мобильных приложений AppInventor (показана на рис. 1) разработана в Google Labs, однако после закрытия лаборатории была передана в MIT (Массачусетский технологический институт). MIT запустил первую публичную бета-версию проекта в марте 2011 года[[3]](#footnote-3).

Данный программный продукт позволяет создавать приложения для мобильной платформы Android. Перед началом разработки необходимо настроить компьютер и телефон для работы с AppInventor (в процессе работы на телефоне должен быть запущен App Inventor Phone Application).

Для создания приложений в AppInventor используется визуальный язык программирования, подобный языку Scratch (визуальная объектно-ориентированная среда программирования для обучения школьников младших и средних классов). Приложения строятся объединением стандартных элементов, например Label (метка), который отображает текст на экране, Button (Кнопка), Canvas, в котором можно располагать изображения или анимацию; accelerometer (motion) sensor, который определяет, когда телефон трясется или переворачивается. Также есть элементы с помощью которых можно отправить сообщения, проиграть видео и многие другие.

**2.2 Buildanapp**



Рис. 2. Buildanapp

Конструктор Buildanapp[[4]](#footnote-4) (показана на рис. 2) позволяет разрабатывать приложения для мобильной платформы iPhone, Android, Blackberry и Mobile Web.

Построение приложения происходит путем прохождения цепочки из шести шагов. На каждом этапе пользователю предлагается выбрать цель приложения, необходимые элементы, количество и вид страниц в приложении и другие функции. BuildAnApp предоставляет функционал для использования электронной почты и списка рассылки. После создания приложения система предлагает загрузить конечное приложение на компьютер пользователя. Интересной особенностью является то, что владельцы имеют возможность с помощью BuildAnApp рекламировать свои созданные приложения, чтобы заработать на них.

**2.3 Выводы**

Таким образом, в результате проведенного анализа были выявлены основные недостатки существующих на данный момент веб-конструкторов для создания мобильных приложений, а именно:

* отсутствие возможности задания нетривиальной клиентской или серверной логики мобильного приложения;
* проектирование приложений исключительно для одной мобильной платформы;
* необходимость постоянного соединения рабочего компьютера и мобильного телефона в процессе создания приложения (для некоторых конструкторов);
* отсутствие средств для тестирования разработанного мобильного приложения на рабочем компьютере.

При проектировании конструктора QReal:Web были учтены и максимально устранены перечисленные недостатки.

**3. Используемые технологии**

QReal:Web изначально проектировался как облачный конструктор для создания мобильных приложений, в связи с чем в качестве основной платформы для реализации серверной части приложения был выбран язык программирования F# (функциональный язык, в котором эффективно реализована поддержка распараллеливания данных) и технология web-разработки WebSharper для взаимодействия с клиентской частью (фреймворк, позволяющий реализовывать на F# клиентскую часть). В качестве облака было решено использовать Windows Azure – облачную платформу от Microsoft.

Для возможности задания серверной логики приложения был реализован сервер приложений, для разработки которого в качестве языка программирования был выбран C#.

В качестве основной среды разработки использовалась Microsoft Visual Studio 2012.

**3.1 Microsoft.NET: F#**

Функциональный язык F#, разработанный в 2002 году в подразделении Microsoft Research, обеспечивает высокую производительность, и снижает объем написанного кода, в сравнении, например, с С#. С деталями использования данного языка можно ознакомиться в книгах Криса Смита [1] и Дона Сайма [2], а также в документации к F# [9].

По производительности F# сравним с C#, однако если сравнивать с языками, которые настолько же лаконичны, как F#, в частности с динамическими языками и языками сценариев, он имеет гораздо лучшие показатели. В 2010 году Microsoft выпустила первую версию продукта – Microsoft Visual Studio 2010, в которую встроена поддержка разработки на F#.

Парадигма функционального программирования предполагает отсутствие в программе неявной зависимости по данным, что в свою очередь облегчает распараллеливание данных. Например, в F# 2.0 реализованы asynchronous workflows – конструкции для организации параллельных и асинхронных вычислений. С помощью традиционных императивных языков программирования достаточно сложно и трудоемко написать распараллеливаемый код без ошибок. Подобное преимущество F# стало основным аргументом при выборе языка программирования для реализации конструктора QReal:Web в связи с тем, что QReal:Web был задуман как система, которая должна работать с большим количеством пользователей, а значит и со значительным потоком информации, которую необходимо обработать за ограниченный промежуток времени.

На данный момент F# широко используется в финансовой сфере, где требуются обработка больших данных и незамедлительное принятие решений, а также нужно строить быстрые алгоритмы, способные давать результат в как можно более короткое время.

На данный момент текущей версией языка F# является версия 3.0, которая поставляется в Microsoft Visual Studio 2012.

**3.2 Microsoft.NET: WebSharper**

WebSharper является HTML5-ориентированным веб-фреймворком, который позволяет создавать веб-приложения, полностью реализованные с использованием единственного языка программирования F#. F#-код клиентской части транслируется в JavaScript, а код серверной части выполняется на сервере обычным образом. С описанием использования данной технологии можно ознакомиться в документации, опубликованной в открытом доступе [7].

WebSharper позволяет повысить эффективность разработки приложения, благодаря возможности использования единого языка программирования для клиентской и серверной частей. В частности, использование языка F# и на серверной и на клиентской стороне обеспечивает прозрачную передачу данных (с использованием одинаковых типов для клиента и серверного кода).

**3.3 Microsoft.NET: C#**

Для реализации сервера приложений был выбран объектно-ориентированный язык программирования C#, разработанный компанией Microsoft в 2001 году. Данный язык поддерживает полиморфизм, перегрузку операторов, события, итераторы, исключения и многое другое. Также в связи с необходимостью продолжительной работы сервера приложений в режиме реального времени важным факторами при выборе языка программирования были наличие сборщика мусора, легкость добавления новых функциональных возможностей сервера и нетрудоемкость поддержки при эксплуатации. С# является одним из наиболее популярных языков программирования и достаточно прост для использования, в связи с чем и было принято решение использовать его для разработки сервера приложений.

С момента появления язык много раз обновлялся и его текущая версия 5.0.

**3.4 Windows Azure**

Windows Azure – это масштабируемая интернет-платформа для облачных вычислений, разработанная компанией Microsoft. Данная платформа включает в себя операционную систему для облачных вычислений и широкий набор служб для разработчиков программного обеспечения [5].

Microsoft Visual Studio 2012 имеет специальные средства для выкладывания проектов в Windows Azure [8]. Таким образом, при разработке проекта с помощью .NET возможна интеграция с облачной платформой без необходимости изменения какого-либо написанного кода.

**4. Требования к архитектуре**

 Исходя из проведенного анализа существующих подходов к визуальному созданию мобильных приложений были сформулированы следующие требования к архитектуре разрабатываемого конструктора.

1) Функциональные требования:

* Предоставление возможности сохранения результата, созданного пользователем.

Необходимо обеспечить пользователю возможность прерывать работу и возобновлять, используя сохраненный результат.

* Обеспечение взаимосвязи объектов.

Объекты, используемые пользователем, должны быть связаны между собой.

* Обеспечение возможности изменения свойств существующих объектов и связей между ними.
* Предоставление возможности удаления объектов и взаимосвязей.
* Обеспечение возможности пользователю перетаскивать на экран мышкой объекты, необходимые для конфигурации его приложения.

2) Нефункциональные требования:

* Простота использования.

Незнакомый с программированием человек должен быть способен написать несложное приложение.

* Расширяемость.

Необходимо обеспечить возможность масштабирования системы, для дальнейшего развития.

* Модульность системы. Слабая связность компонент.

Сбои/падения одной компоненты системы должны как можно меньше влиять на работу других компонент.

* Реализация интерфейсов для различных браузеров.

Интерфейс системы не должен зависеть от браузера пользователя.

* Расширяемость формата обмена данными между клиентской и серверной частями.

В случае появления новых элементов добавление новых узлов в передаваемый формат не должно представлять сложности.

3) Ограничения:

* Использование Windows Azure.
* Использование F# для реализации серверной части приложения.

**5. Архитектура системы**

По результатам исследования и анализа существующих конструкторов для создания мобильных приложений был предложен собственный подход.

Общая архитектура разработанной системы состоит из нескольких компонент и представлена на рис. 3. При разработке архитектуры конструктора были использованы результаты, описанные в работах по проектированию приложений А. Г. Федорова [3], С. Д. Кузнецова [4] и книге Мартина Фаулера [6]. Клиентская часть состоит из дизайнера и эмулятора. Дизайнер предлагает средства для моделирования интерфейса приложения и задания необходимой логики. Эмулятор обеспечивает возможность проверки корректности работы создаваемого мобильного приложения. Серверная часть, включающая веб-сервер и необходимые базы данных, выполняет обработку необходимых данных и генерацию итогового файла, который затем передается на клиент и сохраняется на компьютере пользователя. За саму генерацию отвечает набор специально написанных генераторов приложений под различные платформы. В процессе работы, сформированное мобильное приложение взаимодействует с app сервером (сервером приложений).

Создание приложения происходит путем перетаскивания на рабочую область элементов палитры, таких как кнопка, изображение, текстовое поле и другие. Аналогичным образом происходит задание логики приложения и свойств различных элементов. Например, при добавлении новой кнопки, указывается название формы на которую следует перейти при ее нажатии и, при необходимости, условия для такого перехода. Подобные свойства элементов можно задать или изменить в любой момент при проектировании приложения Созданное приложение можно протестировать с помощью эмулятора и затем сгенерировать соответствующий файл. Сформированный файл (с расширением apk для Android и с расширением xap для Windows Phone) автоматически сохраняется на компьютере пользователя.

В связи с тем, что серверная часть приложения используется как репозиторий диаграмм, клиентская и серверная части взаимодействуют между собой посредством post-запросов (в отличие от get запросов, post запросы передают всю необходимую информацию в теле запроса, а не в заголовке, и таким образом предпочтительны для использования в системах, где необходимо передавать файлы большой длины).



Рис. 3. Основные модули конструктора

Для возможности свободного доступа пользователей к разработанному конструктору, прототип дизайнера и эмулятора выложен в «облако» Microsoft: Windows Azure[[5]](#footnote-5).

В рамках данной дипломной работы были реализованы сайт, веб-сервер, app сервер, все необходимые базы данных, а также обеспечена взаимосвязь между различными компонентами системы (клиентская часть, веб-сервер, генераторы и сервер приложений). Разработка клиентской части (дизайнера и эмулятора) и генераторов не являлась частью поставленной задачи и была проведена другими людьми в рамках соответствующих курсовых и дипломных работ. Тем не менее, их описание приводится здесь для общего понимания технологии работы реализованного конструктора.

## 5.1. Клиент

 Клиентская часть сервиса не зависит от браузера пользователя и состоит из двух компонент: дизайнера приложений и эмулятора. Дизайнер отвечает за создание интерфейса приложения и описание его бизнес-логики. Эмулятор позволяет запустить создаваемое приложение прямо в окне интернет-браузера.

 Дизайнер приложений включает в себя такие компоненты, как палитра элементов, редактор свойств, основная рабочая область, менеджер экранов и редактор клиентской логики приложения. Для задания клиентской логики на данный момент используется событийно-триггерная система, позволяющая задавать поведение приложения при реализации событий определённых типов, например “пришёл ответ на запрос об авторизации”, “сработал таймер”, “нажата кнопка” и т.д. На данный момент клиентская логика описывается путём компоновки элементарных блоков-действий. Пример формирования клиентской логики – задание перехода при авторизации на форму отображения карты, в случае ввода корректной пары “логин-пароль”, и на форму ошибки, в случае ввода некорректной пары “логин-пароль” показан на рис. 4.



Рис. 4. Пример задания клиентской логики

Действия могут быть как общими для всех типов приложений, например условное ветвление или переход на другой экран, так и специфичными для конкретных типов создаваемых приложений, например отправка запроса об авторизации на сервер. В дальнейшем планируется использование графических языков для задания и клиентской, и серверной логики.

Для запуска онлайн-эмулятора или генерации мобильного приложения, подлежащего установке на смартфон, разрабатываемое приложение экспортируется в XML специального вида, который передаётся эмулятору или на сервер в зависимости от того, запускается эмулятор или же генерируется готовое приложение.

 Эмулятор служит для запуска разрабатываемого приложения прямо в окне браузера. Это позволяет существенно облегчить отладку приложения благодаря тому, что не требуется каждый раз генерировать приложение и устанавливать его на смартфон.

## 5.2. Генератор

Генератор по полученным данным создает файл приложения под необходимую мобильную платформу. На текущий момент реализована генерация в популярные мобильные платформы Android и Windows Phone 7/8. В дальнейшем, планируется реализация поддержки и других мобильных операционных систем. Особенностью реализации блоков генераторов является их независимость от остальных компонент системы. Таким образом добавление в конструктор дополнительного блока для генерации кода в новую мобильную платформу является достаточно простой задачей.

 Генератор также как и серверная часть написан на языке F#. Сопоставление шаблонов в этом языке позволяет удобно работать со стандартными парсерами библиотеки классов платформы .NET и дает возможность быстро добавить разбор новых узлов. Кроме этого, скачивание нужных ресурсов из сети интернет и запись данных в файл описывается всего в несколько строчек.

**6. Практическая реализация**

**6.1. Формат обмена данными**

В задачах проектирования и реализации информационных систем одной из важных проблем является обмен данными между различными компонентами системы. В рамках моей дипломной работы необходимо было разработать формат обмена данными между клиентской частью и генератором. В связи с тем, что в классической разработке для мобильных приложений для генерации файла в мобильную платформу используется язык разметки XML, разработанный мной формат также является языком разметки, подобным формату, который используется при разработке приложений для операционной системы Android.

По своей структуре документ в формате XML представляет собой дерево элементов. Некоторые элементы имеют свои дополнительные атрибуты, которые в свою очередь могут иметь какие-то значения. Основным требованием, предъявляемым к формату обмена данными, являлась расширяемость существующего протокола. XML идеально удовлетворяет данному требованию благодаря простой древовидной структуре. Добавление новых узлов в передаваемый формат при появлении новых элементов не представляет сложности.

Разработанный формат обмена данными является единым, в рамках создаваемой системы, для генерации в различные мобильные платформы (Android и Windows Phone). Таким образом, на клиенте происходит формирование одного XML файла, в котором содержится вся информация, необходимая генератору для формирования итогового мобильного приложения и данный XML файл передается на сервер, где и происходит генерация.

Примеры XML документов, соответствующих разработанному формату, формируемые при создании приложений типа “Визитка” и “Мобильный помощник врача” находятся в приложениях.

**6.2. Репозиторий**

Для возможности работы пользователей на сайте (авторизация, создание приложений, сохранение незавершенных приложений и т.д.) был реализован репозиторий на основе базы данных Microsoft SQL Server 2008. В разработанном репозитории была создана таблица, хранящая пары “логин-пароль” для возможности авторизации пользователя на сайте, а также таблица для хранения спроектированных (или не до конца спроектированных) пользователем приложений, которые хранятся в базе данных в виде XML-файлов.

Дополнительно, для апробации разработанного сервера приложений, в репозитории сервера приложений была создана аналогичная таблица для хранения пары “логин-пароль” зарегистрированных ранее пользователей.

**6.3 Сайт и веб-сервер**

Изначально планировалась реализация как серверной, так и клиентской части исключительно посредством WebSharper.

С использованием данной технологии был разработан сайт, на котором пользователь может зайти в свой личный кабинет при помощи логина и пароля. Для хранения логина и пароля ранее зарегистрированных пользователей в базе данных заведена соответствующая таблица Users. Связь между приложением и базой данный осуществляется при помощи стандартных средств .Net – функций из класса SQLConnection.

В рамках дипломной работы был также реализован сервер разработки, который обеспечивает связь между клиентской частью сайта и генераторами. Серверная часть принимает данные, полученные от клиентской части, и выполняет генерацию итогового файла, который затем передается обратно на клиент и сохраняется на компьютере пользователя.

Дополнительно на веб-сервере предполагается работа репозитория для хранения диаграмм пользователя, благодаря которому каждый клиент в личном кабинете сможет редактировать ранее созданные диаграммы.

**6.4. Сервер приложений**

Для возможности задания серверной логики приложения реализован сервер приложений, который имеет возможность в режиме онлайн передавать пользователю данные, запрошенные из сторонних источников. Также сервер приложений может быть использован для агрегации данных из различных внешних систем, выполнения сложных вычислений и других задач.

Связь клиентской части, сервера и итогового мобильного приложения с сервером приложений происходит с помощью post-запросов.

**7. Сложности реализации**

WebSharper – достаточно новый продукт, первая версия которого была выпущена в 2008 году. По этой причине были выявлены следующие сложности при разработке с помощью данного фреймворка:

* низкое качество документации;
* практически полное отсутствие разумных примеров использования в открытом доступе, в связи с небольшим количеством пользователей;
* отсутствие готового решения для многих технических проблем.

**7.1 Сайт и веб-сервер**

В целях тестирования серверной части мной была предпринята попытка реализовать редактор для создания мобильных приложений также при помощи WebSharper. В результате анализа средств WebSharper для работы с графическими изображениями было решено использовать расширение Raphael, аналогичное JavaScript-библиотеке raphael.js, предназначенной для работы с векторной графикой. Однако в процессе разработки выяснилось, что такой подход к созданию редактора является довольно громоздким, поскольку создание примитивов требовало большого количества инструкций, и их комбинация требовала сложной работы с графикой. Затем, разработка редактора была поручена человеку, обладающему опытом работы с JavaScript, а не с WebSharper, в результате чего было решено сменить инструментарий.

Было принято решение, что реализацию редактора будет проще осуществить при помощи TypeScript – надстройки над языком JavaScript. Таким образом на серверной части приложения потребовались различные средства для возможности взаимодействия технологий TypeScript и WebSharper. Например:

* Через атрибут IFrame, который позволяет поместить в один сайт содержимое другого сайта, внутрь клиентской части, разработанной с помощью WebSharper, стало возможно поместить редактор, который фактически находится на другом сайте. Таким образом, при авторизации на сайте, который находится, например, по адресу http://localhost:1234, и входе в личный кабинет, пользователю отображается редактор, находящийся на сайте http://qreal.ru/MobileCreator/default.htm.
* Создание потока, который работает на сервере и обрабатывает запросы, обращенные к определенному порту.

В связи с практически полным отсутствием документации к WebSharper и примеров использования в открытом доступе, при разработке серверной части приложения возникли проблемы, описанные ниже.

**Технические проблемы**

* + Реализация ответов на post-запросы, пришедшие от клиента
		- Для получения данных было необходимо обработать запрос дважды. Запрос, пришедший с клиента, обработанный с помощью InputStream на сервере отображался как пустой. При повторной обработки запроса сервером считывались реальные данные пришедшие от клиента. Обнаружить решение проблемы с помощью WebSharper не удалось, в связи с чем для устранения возникшей ситуации потребовалось ввести переменную, которая меняет значение в зависимости от количества запуска обработчика запроса.
		- Отсутствие возможности на клиенте получить данные из запроса, присланные сервером. Для решения проблемы потребовалось вставить в программный код сервера следующие строки:
		ctx.Response.Headers.Add("Access-Control-Allow-Origin", "\*")
		ctx.Response.Headers.Add("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept")

Для того, чтобы пользователь мог принять ответ на post-запрос, к ответу на запрос добавляются следующие заголовки: Access-Control-Allow-Origin со значением '\*' и Access-Control-Allow-Headers со значением "Origin, X-Requested-With, Content-Type, Accept". Данный заголовки позволяют клиенту принимать данные, посылаемые сервером. В случае, если данные заголовки не используются, то при попытке компиляции проекта отображается ошибка:

“XMLHttpRequest cannot load <http://localhost:12345/>. Origin <http://localhost:61082> is not allowed by Access-Control-Allow-Origin”

* + Реализация автоматического скачивания файлов, в связи с необходимостью отсутствия явной ссылки на файл (в случае наличия явной ссылки на файл, любой, у кого есть ссылка, может распространить данный файл всем, что не безопасно). Первоначально планировалось сгенерированный на сервере файл либо передавать побайтово, таким образом создавая файл на клиенсткой стороне, либо передавать клиенту файл с определенными пометками (то есть указывая тип передачи как бинарный файл). Однако посредством WebSharper подобную передачу файла на клиентскую часть, реализованную на TypeScript, осуществить не удалось.

**Проблемы генерации из WebSharper в JavaScript**

* В F# возможен синтаксис: type MyClass (arg1, arg2) as this = …
При использовании такой конструкции к экземпляру класса в дальнейшем возможно обращение по имени this, в связи с чем подобный синтаксис достаточно удобен. Однако, в случае с генерацией данной конструкции в JavaScript (код WebSharper транслируется в JavaScript) происходило создание ссылки (синтаксис “as this”), которая мешала компиляции (компилятор выводил ошибку о невозможности компиляции ввиду создания ссылки). Таким образом компиляция была невозможна. Причем данная проблема возникла не у самого класса, а у его наследника, который транслировался, в результате чего описанную проблему было сложно обнаружить.
* type VisiblePrimitive не содержит у метода DrawRect атрибут [<JavaScript>]. Таким образом, при использовании Raphael происходила корректная генерация кода WebSharper в код JavaScript, однако JavaScript в свою очередь не запускался.

**Решения описанных проблем**

* Для корректной реализации взаимодействия посредством post-запросов был реализован отдельный файл NetExtensions. Данный файл содержит класс для создания сервера, отвечающего на post-запросы.
* При попытке разработать редактор при помощи WebSharper была реализована событийная обработка (нажатия на кнопки мыши и перемещения элементов по экрану), хранение примитивов и методов для их рисования посредством Raphael в файле проекта Primitive.fs и их генерации в XML файл. Для решения описанной проблемы с методом DrawRect была реализована структура кода, в которой всё наследуется от Primitive. Данный класс имеет наследника VisiblePrimitive для тех элементов, которые отображаются на форме.

**7.2 Сервер приложений**

Для тестирования сервера приложений был также реализован класс Sender – эмулятор клиента, отправляющего запросы и принимающего ответы на них.

Идентификация пользователя на сервере приложений реализована с помощью cookie – небольшого набора данных, передаваемых между клиентской и серверной частями, уникальных для каждого пользователя. При посещении веб-сайта серверная часть приложения оставляет на компьютере пользователя cookie для аутентификации клиента при его возвращении на сайт. Для хранения cookie реализована отдельная таблица в базе данных, хранящая уникальный идентификатор клиента, соответствующий уникальному идентификатору клиента из таблицы пользователей, и cookie данного пользователя в текущей сессии.

Для обработки запросов, обращенные к определенному порту, был реализован бесконечный цикл, создающий поток, отвечающий на запросы:
for (; ; ) {
 HttpListenerContext ctx = listener.GetContext();
 new Thread(new Worker(ctx).ProcessRequest).Start();
}

Так же, как и при разработке веб-сервера, при реализации сервера приложений возникли проблемы на клиентской части с получением данных из post-запроса. Используя стандартные средства обработки post-запросов извлечь присланную информацию также не получилось. Для решения проблемы потребовалось, как и в случае с веб-сервером вставить следующие строки кода в программу:

ctx.Response.Headers.Add("Access-Control-Allow-Origin", "\*")
ctx.Response.Headers.Add("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept")

**7.3 Взаимодействие с «облаком»**

Для возможности свободного доступа пользователей к разработанному конструктору, прототип дизайнера и эмулятора выложен в облако Microsoft: Windows Azure.

При попытке выложить редактор в «облако» возникли проблемы, связанные с тем, что в исходном проектном файле было указано компилировать TypeScript в JavaScript с учетом AMD только при отладке (в дебаге), а в релизе компилировать стандартно. Однако в «облако» сначала выкладывался код, собранный именно в релизе. Таким образом возникали ошибки, которые удалось исправить добавлением описанной опции при компиляции в релизе.

**8. Апробация**

Была произведена апробация конструктора на примере создания приложений “Визитка” и “Мобильный помощник врача”.

 “Визитка” – приложение, отображающее общую информацию об организации, контакты и местоположение предприятия на карте. Приложение состоит из двух форм. Первая форма содержит логотип предприятия (изображение) и кнопку, при нажатии на которую происходит переход на вторую форму, где расположена карта с местоположением организации.

Апробация взаимодействия с сервером приложений была произведена на примере “Мобильный помощник врача” – приложения, отображающего местоположение пациентов определенного врача на карте. Приложение состоит из формы авторизации и формы с картой. При входе в приложение открывается форма авторизации. При попытке авторизации на сервер приложений отправляется соответствующий запрос. Если при попытке авторизации введена пара “логин-пароль”, которой нет в базе, происходит переход на форму с информацией о некорректности введенных данных. Если при попытке авторизации введена пара “логин-пароль”, существующая в базе, происходит переход на форму с картой, на которой отображается местоположение пациентов. Координаты местоположения пациентов для каждого врача хранятся в базе данных сервера приложений и передаются на клиентскую часть одновременно с информацией об успешности авторизации. Приложение опрашивает с заданной периодичностью (например, 5 секунд) сервер приложений на предмет появления новых записей в таблице с координатами пациентов для текущего пользователя и, если появилась новая информация, на карте отобразится новая точка с соответствующими координатами.

На рис. 5 показаны три формы приложения “Мобильный помощник врача”: форма авторизации, форма с информацией об ошибке при вводе логина и пароля и форма с отображением местоположения пациентов.



Рис. 5. Пример работы клиентской логики

Для данного приложения в базу данных были добавлены таблицы, содержащие информацию о логине и пароле пользователей (врачей), а также таблица, содержащая координаты пациентов для каждого врача.

**Заключение**

В рамках данной работы были изучены существующие подходы к визуальному созданию мобильных приложений. В результате проведенного анализа было разработано собственное решение, описанное в данной работе. Для возможности взаимодействия клиентской части, сервера разработки, сервера приложений и генераторов были разработаны форматы обмена данными.

 Также был реализован прототип серверной части (веб-сервер и сервер приложений) онлайн-конструктора приложений для мобильных телефонов, требующих возможности задания клиентской логики, и проведена апробация созданного решения на примере приложений «Визитка» и «Мобильный помощник врача».

Таким образом, результатами данной дипломной работы являются:

* архитектура конструктора QReal:Web
* формат обмена данными между различными частями конструктора
* обеспечение взаимосвязи между клиентской частью, сервером разработки, сервером приложений и генераторами
* сайт и сервер разработки
* сервер приложений
* интеграция клиентской части конструктора в «облако» Microsoft: Windows Azure
* апробация конструктора QReal:Web

Конструктор QReal:Web стал победителем регионального этапа конкурса Microsoft Imagine Cup в Санкт-Петербурге, результаты работы были также представлены автором данной работы на конференции “СПИСОК-2013”.

**Список литературы**

[1] Chris Smith, Programming F# // Издательство O'Reilly Media. 2009, 416С

[2] Syme Don, Expert F# // Издательство Apress, 2007,609С

[3] Елманова Н., Федоров А.Г. / Статья от июня 2002 г.: Архитектура современных web-приложений // КомпьютерПресс. 2002. URL:

<http://www.compress.ru/article.aspx?id=10951&iid=440>

[4] Кузнецов С.Д. Общая классификация архитектур информационных приложений. Проектирование и разработка корпоративных информационных систем // Центр информационных технологий. 1998.

URL: <http://citforum.ru/cfin/prcorpsys/index.shtml>

[5] Мартынов Д.Н., Федоров А.Г. Windows Azure: Облачная платформа Microsoft, 2010.

URL:<http://download.microsoft.com/documents/rus/msdn/Windows_Azure_web.pdf>

[6] Фаулер Мартин, Райс Девид, Фоммел Мэттью, Хайет Эдвард, Ми Роберт, Стаффорд Рэнди. Архитектура корпоративных программных приложений // Издательство “Вильямс”. 2007, 544С

[7] Документация по WebSharper, URL: <http://websharper.com/WebSharper.pdf> (дата последнего обращения 20.05.2013)

[8] Документация по Windows Azure, URL: <http://www.windowsazure.com/ru-ru/> (дата последнего обращения 29.04.2013)

[9] Документация по F#, URL: <http://msdn.microsoft.com/ru-ru/vstudio/hh386303.aspx>

(дата последнего обращения 20.05.2013)

**Приложения**

**Приложение А**

**“Визитка”**

<application name = "patients">

 <logic>

 <action control-id = "button4">

 <transition form-id = "About"/>

 </action>

 <action control-id = "button5">

 <transition form-id = "Map"/>

 </action>

 <trigger form-id = "main" event = "onShow"></trigger>

 <trigger form-id = "main" event = "onTimer"></trigger>

 <trigger form-id = "main" event = "onLoginResponse"></trigger>

 <trigger form-id = "main" event = "onPatientsResponse"></trigger>

 <trigger form-id = "About" event = "onShow"></trigger>

 <trigger form-id = "About" event = "onTimer"></trigger>

 <trigger form-id = "About" event = "onLoginResponse"></trigger>

 <trigger form-id = "About" event = "onPatientsResponse"></trigger>

 <trigger form-id = "Map" event = "onShow"></trigger>

 <trigger form-id = "Map" event = "onTimer"></trigger>

 <trigger form-id = "Map" event = "onLoginResponse"></trigger>

 <trigger form-id = "Map" event = "onPatientsResponse"></trigger>

 </logic>

 <forms>

 <form form\_name = "main">

 <LinearLayout

 orientation = "vertical"

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 background = "#ffffff">

 <ImageView

 layout\_width = "wrap\_content"

 layout\_height = "wrap\_content"

 src = "https://dl.dropboxusercontent.com/u/10802739/ic/Logo.jpg"

 layout\_gravity = "center\_horizontal"

 layout\_marginTop = "10px"/>

 <TextView

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "10px"

 padding = "0px"

 text = "Hello!"

 textSize = "26px"/>

 <ImageView

 layout\_width = "wrap\_content"

 layout\_height = "wrap\_content"

 src = "https://dl.dropboxusercontent.com/u/10802739/ic/img1\_min.jpg"

 layout\_gravity = "center\_horizontal"

 layout\_marginTop = "10px"/>

 <Button

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "20px"

 text = "About"

 textSize = "15px"

 id = "button4"/>

 <Button

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "20px"

 text = "Show on map"

 textSize = "15px"

 id = "button5"/>

 </LinearLayout>

 </form>

 <form form\_name = "About">

 <LinearLayout

 orientation = "vertical"

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 background = "#ffffff">

 <WebView

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 id = "webview7"

 url = "http://www.imaginecup.com/#?fbid=vy20mUumfI0"/>

 </LinearLayout>

 </form>

 <form form\_name = "Map">

 <LinearLayout

 orientation = "vertical"

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 background = "#ffffff">

 <Map

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 id = "map9"/>

 </LinearLayout>

 </form>

 </forms>

</application>

**Приложение В**

**“Мобильный помощник врача”**

<application name = "patients">

 <logic>

 <action control-id = "button5">

 <login-request

 url = "http://nb.infolan.me:54321"

 login-id = "editText2"

 password-id = "editText4"/>

 </action>

 <trigger form-id = "main" event = "onShow"></trigger>

 <trigger form-id = "main" event = "onTimer"></trigger>

 <trigger form-id = "main" event = "onLoginResponse">

 <if condition = "loginSuccess">

 <then>

 <transition form-id = "Map"/>

 </then>

 <else>

 <transition form-id = "Failed"/>

 </else>

 </if>

 </trigger>

 <trigger form-id = "main" event = "onPatientsResponse"></trigger>

 <trigger form-id = "Map" event = "onShow"></trigger>

 <trigger form-id = "Map" event = "onTimer">

 <patients-request url = "http://nb.infolan.me:54321"/>

 </trigger>

 <trigger form-id = "Map" event = "onLoginResponse"></trigger>

 <trigger form-id = "Map" event = "onPatientsResponse">

 <showmap id = "map7"/>

 </trigger>

 <trigger form-id = "Failed" event = "onShow"></trigger>

 <trigger form-id = "Failed" event = "onTimer"></trigger>

 <trigger form-id = "Failed" event = "onLoginResponse"></trigger>

 <trigger form-id = "Failed" event = "onPatientsResponse"></trigger>

 </logic>

 <forms>

 <form form\_name = "main">

 <LinearLayout

 orientation = "vertical"

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 background = "#ffffff">

 <TextView

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "10px"

 padding = "0px"

 text = "Login:"

 textSize = "26px"/>

 <EditText

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "10px"

 id = "editText2"

 padding = "0px"

 text = "px"/>

 <TextView

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "10px"

 padding = "0px"

 text = "Password:"

 textSize = "26px"/>

 <EditText

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "10px"

 id = "editText4"

 padding = "0px"

 text = "px"/>

 <Button

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "20px"

 text = "Button"

 textSize = "15px"

 id = "button5"/>

 </LinearLayout>

 </form>

 <form form\_name = "Map">

 <LinearLayout

 orientation = "vertical"

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 background = "#ffffff">

 <Map

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 id = "map7"/>

 </LinearLayout>

 </form>

 <form form\_name = "Failed">

 <LinearLayout

 orientation = "vertical"

 layout\_width = "fill\_parent"

 layout\_height = "fill\_parent"

 background = "#ffffff">

 <TextView

 layout\_width = "fill\_parent"

 layout\_height = "wrap\_content"

 layout\_marginTop = "10px"

 padding = "0px"

 text = "Login failed"

 textSize = "26px"/>

 <ImageView

 layout\_width = "wrap\_content"

 layout\_height = "wrap\_content"

 src = "http://3before30.com/wp-content/uploads/2012/05/failed.png"

 layout\_gravity = "center\_horizontal"

 layout\_marginTop = "10px"/>

 </LinearLayout>

 </form>

 </forms>

</application>

1. <http://www.gartner.com/technology/mobile/> [↑](#footnote-ref-1)
2. <http://www.abiresearch.com/press/cumulative-mobile-app-revenues-set-to-exceed-30-bi> [↑](#footnote-ref-2)
3. <http://appinventor.mit.edu/> [↑](#footnote-ref-3)
4. <http://buildanapp.com/> [↑](#footnote-ref-4)
5. <http://qreal-web.azurewebsites.net/> [↑](#footnote-ref-5)