МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет
«Московский институт электронной техники»

Факультет микроприборов и технической кибернетики
Кафедра информатики и программного обеспечения вычислительных систем

Бакалаврская работа
по направлению 09.03.04 «Программная инженерия»

«Разработка мобильного приложения калькуляции стоимости ремонта автомобиля»
(шифр МП КРА)

Студент _____ Бычков А.А.
Научный руководитель,
Доцент кафедры ИПОВС, к.т.н. _____ Касимов Р.А.

Москва 2018

Содержание

Введение

Определение стоимости ремонта автомобиля представляет собой сложную задачу, для решения которой необходимы эксперты, обладающие знаниями в специфических областях автомобильного ремонта.

Во время прохождения практики в ООО «ДИМАВТО» было выявлено, что сложность определения стоимости ремонта «отпугивает» клиентов автосервиса. Для решения данной проблемы было принято решение о внедрении сервиса калькуляции ремонта автомобиля. Более того, особенностью данного сервиса должна была стать возможность потенциальных клиентов самостоятельно выполнять расчет, а уже затем, приняв решение воспользоваться автосервисом, записаться на обслуживание или договорится по телефону.

Данная задача возникает во многих автосервисах, однако в данном случае следовало принять во внимание основное направление деятельности ООО «ДИМАВТО» - ремонт топливной аппаратуры. Соответственно, должна присутствовать возможность дополнять перечень ремонтных работ, а также корректировать цены по сезонам.

Были изучены уже существующие программные решения других фирм и составлен перечень необходимых характеристик. Однако, исходя из поставленных требований, ни одно ныне существующее средство нельзя было назвать удовлетворительным решением обозначенной проблемы. В результате, актуальной является задача создания нового ПО, позволяющего оценить стоимость ремонта автомобиля и записаться на обслуживание непосредственно с мобильного устройства, в рамках ВКР бакалавра по направлению 09.03.04 «Программная инженерия» [1] (в соответствии с ГОСТ 19.701-90 [2], 19.505-78 [3] и 19.201-79 [4].

Пояснительная записка состоит из введения, трех разделов, заключения, списка литературы и двух приложений: руководства оператора и текста программы.

Исследовательский раздел включает в себя анализ существующих решений, выбор архитектуры МП КРА, схему данных МП КРА, алгоритм работы сервера и алгоритм работы клиента.

Конструкторский раздел содержит выбор языка программирования, выбор среды разработки, разработку web-сервиса, и разработку мобильного приложения.

Технологический раздел состоит из отладки web-сервиса, отладки мобильного приложения, тестирования web-сервиса и тестирования мобильного приложения.

1. Исследовательский раздел

1.1. Анализ существующих решений

В настоящее время уже существуют программные средства для самостоятельной оценки стоимости ремонта, исходя из видимых повреждений. Рассматривались следующие программы: Аудатэкс «AudaEnterprise Gold» [5], Аудатэкс «AUTOonline» [6], «Автонормы» SmileGroup [7] и «Автонормы» ATLIB.RU [8]. Компания «Аудатэкс» занимается подсчётом стоимости ремонта автомобилей начиная с 60-х годов XX века и зарекомендовала себя как международная компания по разработке качественного ПО. В настоящий момент компания продвигает два своих продукта «AudaEnterpriseGold» и «AUTOonline».

Первый из них рассчитан на крупные предприятия и мастерские. Программа поставляется в виде установочного носителя и запускается непосредственно с ПК, без необходимости в Интернет-соединении. Цена такого решения крайне высока. Это обусловлено тем, что, владея таким ПО можно производить неограниченное число расчётов, а, следовательно, обслуживать большое количество клиентов. Для снижения стоимости этого продукта, компания «Аудатекс» выпустила другую версию программы под названием «AUTOonline». Принцип работы с этой программой иной: программа доступна в виде сайта через Интернет и не устанавливается на ПК в автомастерской. В данном случае лицензируется факт наличия доступа к сервису и количество произведённых расчётов. Это даёт возможность использовать данное ПО малым и средним авторемонтным предприятиям, поскольку затраты на использование расчётного ПО прямо пропорционально количеству обслуживаемых клиентов. Стоит также отметить, что из-за условий лицензирования, ни один из продуктов «Аудатэкс» не удаётся предоставлять в пользование клиентам самостоятельно.

Также были изучены несколько бесплатных online калькуляторов стоимости ремонта. Среди них были «Автонормы» от компании «SmileGroup» и одноименная программа от сообщества «ATLIB.RU». Обе программы предоставляют схожий функционал: пользователь указывает на сайте данные о своем транспортном средстве, а программа демонстрирует ему перечень всевозможных ремонтных работ по категориям. Затем пользователь отмечает требуемые работы и, в случае с программой от «ATLIB.RU», указав

стоимости нормо-часа, получает расчётную стоимость всего ремонта. Эти сервисы невозможно использовать на предприятии авторемонта по нескольким причинам.

Во-первых, сам перечень ремонтных работ включает в себя все возможные виды работ, без учёта специализации конкретного автосервиса.

Во-вторых, посетителю такого сайта необходимо знать стоимость нормо-часа на авторемонтном предприятии, куда он хочет обратиться, хотя было бы удобнее, если она выставлялась автоматически.

В-третьих, у всех многих online решений отсутствует возможность вносить коррективы в базу норм времени. Однако все перечисленные выше решения не имеют мобильной версии. Это означает, что воспользоваться ими можно только со стационарного ПК. Некоторые мобильные устройства позволяют посещать обычные web-сайты, но это сопряжено с определёнными трудностями, вызванными в первую очередь использованием вместо мыши сенсорного экрана с недостаточной точностью для работы с мелкими элементами.

Целью работы является создание нового ПО, позволяющего оценить стоимость ремонта автомобиля и записаться на обслуживание непосредственно с мобильного устройства. Поскольку на сегодняшний день существует несколько популярных мобильных платформ, разработка имеет кроссплатформенную направленность.

1.1.1. Исследование программного решения «Аудатэкс AudaEnterprise Gold»

Предполагается доступ через Интернет с помощью браузера.

Имеется указать стоимость нормо-часа для каждого из видов работ.

Отдельная мобильная версия не предусмотрена. Имеется возможность подключения к сервису с мобильного устройства с web-браузером, однако макет экранных форм может нарушиться.

Возможность добавлять автомобили в базу самостоятельно отсутствует. Разработчик ПО сам добавляет в систему марки и модели машин по ходу работы.

ПО установлено на серверах разработчика и его модификация технически невозможна.

ПО лицензируется на один год по фиксированной цене в 4800 руб. Далее, исходя от количества рабочих мест в автосервисе и количества расчётов в месяц, формируется дополнительная ежемесячная плата. Минимальная цена: 2200 руб. в месяц.

1.1.2. Исследование программного решения «Автонормы SmileGroup»

Предполагается взаимодействие с сервисом через интернет с помощью браузера. Требуется постоянное подключение к сети.

Возможность указать стоимость нормо-часа отсутствует. Предполагается, что расчёт производится специалистом из автосервиса.

Мобильная версия не предусмотрена.

Возможность добавления автомобилей в базу отсутствует. Разработчик самостоятельно добавляет автомобили в перечень по мере поступления новой информации. Отсутствует четкий график.

ПО установлено на серверах разработчика и его модификация технически невозможна.

Программа предоставляется бесплатно.

1.1.3. Исследование программного решения «Автонормы Atlib.ru»

Предполагается взаимодействие с сервисом через интернет с помощью браузера. Требуется постоянное подключение к сети.

Имеется возможность указать стоимость нормо-часа для всех видов работ одновременно, что справедливо не для всех автосервисов.

Мобильная версия не предусмотрена.

Возможность добавления автомобилей в базу отсутствует. Разработчик самостоятельно добавляет автомобили в перечень по мере поступления новой информации. Отсутствует четкий график.

ПО установлено на серверах разработчика и его модификация технически невозможна.

Программа предоставляется бесплатно.

1.1.4. Общее сравнение существующих решений

Таблица 1 - Сравнение ключевых параметров решений

Параметры	Аудатэкс AudaEnterprise Gold	Аудатэкс AUTOonline	Автонормы SmileGroup	Автонормы ATLIB.RU	МП КРА
Режим доступа	Установка на компьютер	Online	Online	Online	Online / Локальная сеть
Возможность указать расценки нормо-часа	Да	Да	Нет	Да	Да
Наличие мобильной версии	Нет	Нет	Нет	Нет	Да
Расширяемая база автомобилей	Нет	Нет	Нет	Да	Да
Возможность последующей модификации	Нет	Нет	Нет	Нет	Да
Вариант лицензирования	Проприетарное ПО (непрозрачная политика ценообразования)	4800 руб. в год + от 2200 руб. в месяц	Не требуется	Не требуется	Не требуется

В результате анализа вышеперечисленных программ, был составлен перечень функций, которыми должно обладать искомое решение:

– возможность указать расценки нормо-часа. Для того чтобы расчёт выдавал результат в рублях, необходимо иметь в программе возможность указать стоимость нормо-часа;

- наличие мобильной версии. Ключевой функцией предполагаемого решения должна быть возможность выполнения расчётов по ремонту с мобильного устройства;
- расширяемая база автомобилей. Поскольку рынок автомобилей постоянно растёт, требуется возможность добавлять новые марки и модели машин, чтобы предоставлять клиентам актуальные услуги;
- возможность последующей модификации. Для повышения эффективности предоставленного клиентам калькулятора может возникнуть необходимость внести изменения в определенные экранные формы приложения.

1.2. Выбор архитектуры МП КРА

Для соответствия поставленным требованиям была выбрана архитектура *клиент-сервер*. Суть данной архитектуры заключается в том, что программа разделяется на две части: серверную, выполняющую какие-либо вычисления, хранящую в себе какие-либо данные и клиентскую, которая обращается к серверу за предоставляемыми сервисами. Обычно клиенты и серверы выполняются на разном оборудовании, обмениваясь при этом данными посредством компьютерной сети. Сервер предоставляет ресурсы, которыми могут воспользоваться клиенты. Клиенты не предоставляют никакие ресурсы, а лишь запрашивают ресурсы сервера.

В данном случае реализация решения в виде одного компонента нецелесообразно ввиду того, что нормы времени автомобильного ремонта представляют собой большой объём часто изменяющихся данных. Для того чтобы у клиентов всегда имелся доступ к самой последней информации, база данных размещается на сервере, а непосредственно программа расчёта на мобильном устройстве.

1.3. Схема данных МП КРА

Схема данных представлена на рис. 1. В начале данные поступают в мобильное приложение как результат соответствующего запроса. Затем данные демонстрируются пользователю для выбора интересующего его производителя автомобиля. После указания производителя, происходит фиксация выбранного значения, и демонстрируются модели автомобилей данного производителя. Если у данного производителя только одна модель, то

данный шаг пропускается. После указания модели демонстрируются доступные варианты исполнения выбранного автомобиля. Также, как и в случае с моделью, если вариант только один, то данный шаг пропускается.

На основании введенной информации система может предоставить пользователю перечень доступных ремонтных работ по категориям. Выбирая категории и отмечая требуемые работы, формируется список заказанных работ. При добавлении или удалении работы в список, происходит расчёт стоимости ремонта с учетом норм времени выбранных работ, категорий ремонта и стоимости нормо-часа.

1.4. Алгоритм работы сервера

На рис. 2 приведен алгоритм работы сервера. Алгоритм сервера представляет собой цикл, прослушивающий очередь входящих подключений. При подключении, клиент передаёт серверу запрос и ожидает ответ. В момент поступления запроса сервер последовательно проверяет его соответствие одному из критериев пяти типов запросов. В случае если запрос не соответствует ни одному из известных, то формируется сообщение об ошибке для клиента. Всего предусмотрено пять видов запросов:

- запрос списка производителей;
- запрос списка моделей;
- запрос списка вариантов исполнения;
- запрос списка категорий;
- запрос списка ремонтных работ.

При соответствии запроса одному из вышеуказанных, происходит извлечение соответствующей информации из базы данных сервера, сериализация и передача клиенту. После этого цикл начинается заново.

1.5. Алгоритм работы клиента

Схема алгоритма работы клиента приведена на рис. 3. При запуске приложения происходит подготовка списка производителей, путем отправки запроса на сервер и получении результата. Затем перечень демонстрируется пользователю для выбора одного варианта. После выбора, происходит запрос списка моделей автомобилей указанного

производителя. По получении ответа, список демонстрируется пользователю. Если ответ содержал только одно значение, то пользователю отображается значение без возможности выбора. Аналогичным образом происходит выбор варианта исполнения автомобиля. Затем приложение демонстрирует пользователю доступные ремонтные работы, разбитые по категориям. Выбирая их, пользователь формирует список заказанных работ. После того, как все необходимые работы выбраны, производится расчет итоговой стоимости работ и вывод на экран.

Выводы по исследовательскому разделу

В результате исследований, было принято решение о разработке собственного ПО, обладающего возможностью указать стоимость нормо-часа, расширяемой базой автомобилей и оформленного в виде мобильного приложения. В качестве архитектуры была выбрана клиент-серверная модель для оптимального информационного обмена между модулями ПО.

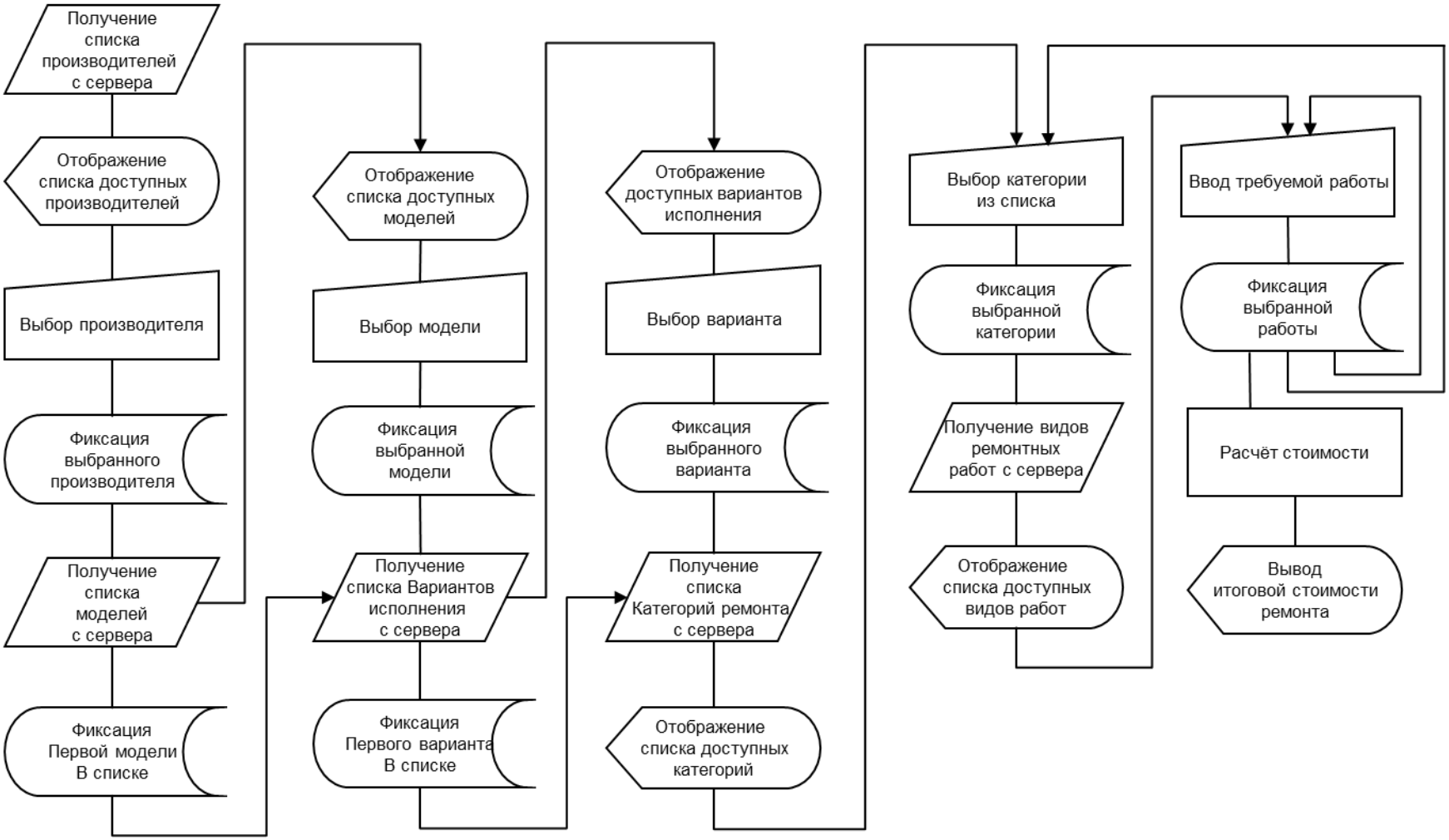

Рисунок 1 - Схема данных мобильного приложения

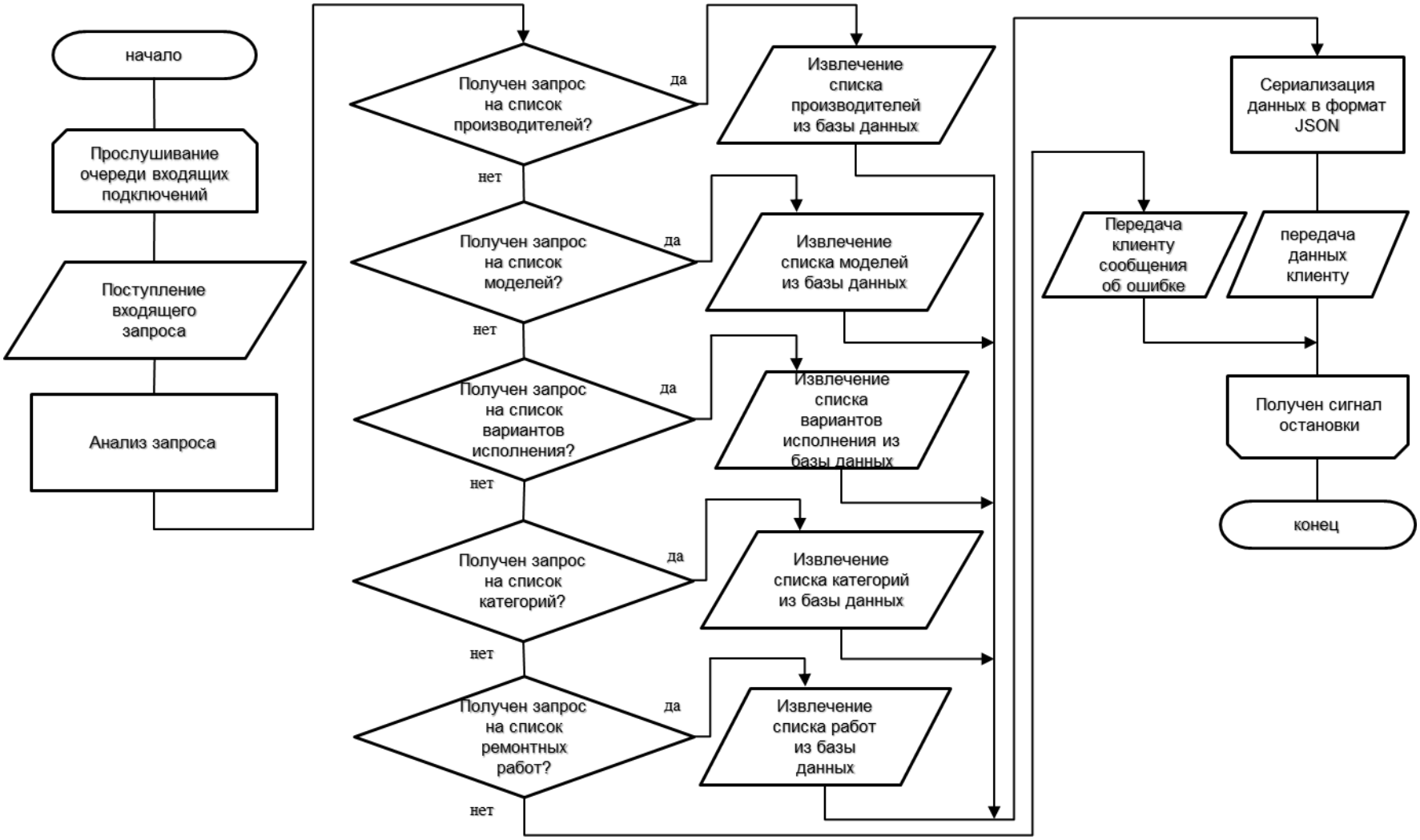

Рисунок 2 - Схема работы сервера

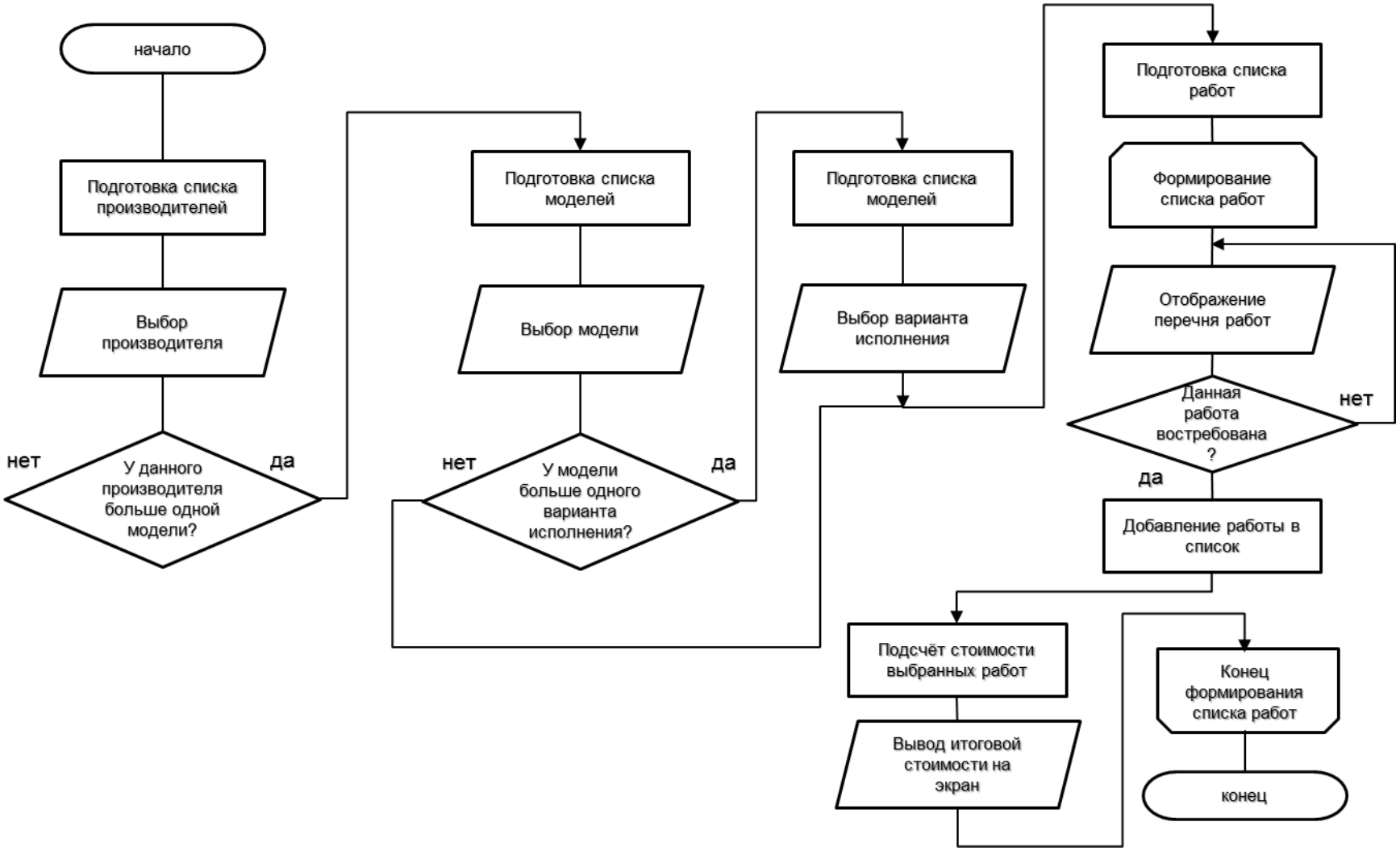

Рисунок 3 - Схема алгоритма работы клиента

2. Конструкторский раздел

2.1. Выбор языка программирования

Выбор языка программирования основывался на сравнении доступных средств языка и требований проекта. Учитывая множество доступных вариантов, необходимо выбрать именно тот язык, который даст возможность разработать ПО с минимальными накладными расходами и за сроки, указанные в техническом задании. Вначале были сформированы *ключевые особенности* проекта, на основании которых можно выделить предпочтительный язык программирования:

- клиент-серверная архитектура;
- одинаковый язык для серверной части и мобильного приложения;
- переносимость;
- строгая типизация данных;
- средства для работы с СУБД;
- наличие транслятора под платформу Windows Phone;
- наличие транслятора под платформу Android;
- наличие транслятора под платформу iOS.

2.1.1. Обзор возможностей языка C++

C++ - компилируемый, статически типизируемый язык программирования общего назначения [9]. Предусматривает как функциональную, так и объектно-ориентированную парадигму программирования. Содержит средства для низкоуровневых взаимодействий с памятью.

Изначально C++ разрабатывался как язык для встраиваемых решений и системного программирования, больших систем, а также для систем с ограниченными ресурсами. Основной целью разработчиков было обеспечение высокой производительности, эффективности и гибкости языка. C++ также нашёл применение во многих других областях, таких как прикладное ПО, серверное ПО и системы реального времени (телефонные коммутаторы и космические аппараты).

Данный язык серьёзно повлиял на другие языки, например, C# и Java.

C++ поддерживает компиляцию под различные платформы, что делает программы, написанные на нём *переносимыми*.

Для создания *web-сервисов* на языке C++ существуют сторонние библиотеки и фреймворки:

- gSOAP [10];
- Apache Axis [11];
- Pion [12];

Наиболее функциональным средством является решение «WSO2 Web Services Framework for C++» [13].

Для взаимодействия с различными базами данных в C++ используется программный интерфейс «ODBC» [14]. С помощью него осуществляется подключение к СУБД, выполнение SQL-запросов и получение результатов. В перечень совместимых СУБД входят:

- Oracle;
- PostgreSQL;
- MySQL;
- Microsoft SQL Server;
- SQLite;
- Sybase ASE;
- SAP HANA;
- DB2.

Несмотря на переносимость программ, написанных на C++, *мобильные платформы накладывают свои ограничения на формат исполняемого кода*. На сегодняшний день, только ОС Android допускает запуск программ на C++. Windows Phone и iOS такой возможности не предоставляют.

2.1.2. Обзор возможностей языка Objective-C

Objective-C – объектно-ориентированный язык общего назначения, построенный на основе языка C и парадигм «Smalltalk» [15]. Является основным языком программирования, используемым Apple в операционных системах «OS X» и «iOS». Также выступает в качестве

программного интерфейса в «Cocoa» и «Cocoa Touch». Поддерживает динамическое типизирование.

Для Objective-C существуют компиляторы под ОС «Linux», «NetBSD», «OpenBSD», «FreeBSD», «Solaris» и «Windows», как часть свободного проекта «GNUstep».

Objective-C содержит встроенные средства для работы с сетевыми сокетами. Тем не менее, реализация такого решения будет крайне затратна по времени разработки, поскольку все детали HTTP протокола придётся разрабатывать самостоятельно.

Objective-C имеет средства для работы со встроенной в ОС «Mac OS X» и «iOS» СУБД «EntropyDB» [16]. Средства языка позволяют работать с базой данных как с коллекцией строго типизированных объектов, не прибегая к необходимости выполнять SQL-запросы самостоятельно.

Помимо этого, имеется возможность использовать встраиваемую СУБД «SQLite» [17].

На сегодняшний день единственной *мобильной платформой*, поддерживающей программы на «Objective-C» является «iOS». Другие мобильный платформы не поддерживаются.

2.1.3. Обзор возможностей языка Java

Java – объектно-ориентированный язык общего назначения с поддержкой параллельных вычислений. Основной концепцией языка является переносимость кода [18]. Скомпилированный Java-код может выполняться на всех платформах, поддерживающих Java, без необходимости в перекомпиляции. Java приложения обычно компилируются в байт-код, который способен выполняться на любой Java-машине независимо от архитектуры процессора самого компьютера. Java занимает лидирующие позиции среди языков для клиент-серверной разработки.

Следуя своей главной концепции, Java программы могут выполняться на любой платформе, где установлена Java-машина. В неполный перечень таких платформ входят:

- Windows;
- Mac OS X;
- Linux;
- Solaris.

Помимо возможности выполнения кода в Java-машине, существуют специализированные устройства с процессорами, поддерживающими байт-код на аппаратном уровне. Использование таких устройств обеспечивает существенное повышение производительности.

Для реализации *web-сервисов* в Java имеется два встроенных решения: «Metro» [19] и «GlassFish» [20].

«Metro» представляет из себя набор классов для «JAX-WS», «JAXB», и «WSIT» для создания защищённых, надёжных, транзакционных web-сервисов и клиентов.

«GlassFish» представляет из себя совместимый с Java EE5 сервер приложений, с расширенными возможностями для приложений «Metro». Также имеются AJAX средства для Web 2.0.

Для работы с *базами данных* в Java используется интерфейс «JDBC» [21]. Работа с ним аналогична работе с «ODBC» в C++. Стыковка непосредственно с каждой из СУБД производится посредством специального JDBC-драйвера, предоставляемого разработчиком СУБД.

Помимо этого, в Java имеется встроенная СУБД – «Java DB» [22], которая в свою очередь является особой версией «Apache Derby».

Для разработки под *мобильные устройства* на базе Android существует комплект разработчика от Google под названием «Android SDK». В ОС Android используется нестандартный Java-байткод, поэтому компилирование Java-программ под Android стандартными средствами не представляется возможным. В состав «Android SDK» входят различные компиляторы и средства отладки для разработки под каждую из версий ОС Android.

Однако на других мобильных платформах выполнение Java программ невозможно ввиду отсутствия под них соответствующих Java-машин.

2.1.4. Обзор возможностей языка PHP

PHP – язык серверных сценариев, спроектированный для web-разработки, но также иногда используется как язык общего назначения [23]. PHP-код может быть встроен в HTML-код, или применяться в составе различных систем web-шаблонов, систем управления контентом (CMS) и web-фреймворков. PHP обычно обрабатывается PHP-интерпретатором,

который реализован в виде модуля для web-сервера, либо в виде отдельного исполняемого файла.

Учитывая тот факт, что PHP является интерпретируем языком, платформы, где его можно применять ограничиваются лишь требованиями PHP-интерпретатора. PHP-интерпретатор является свободным ПО, поэтому его исходный код доступен в открытом виде. Благодаря этому пакеты с интерпретатором PHP присутствуют, практически, во всех дистрибутивах Linux. На официальном сайте PHP доступны сборки для Windows.

PHP изначально проектировался *как серверный язык сценариев*, поэтому он содержит в себе богатые возможности для создания web-сервисов. К ним относятся:

- разбор HTTP-запроса по параметрам;
- сериализация и десериализация данных в форматы JSON и XML;
- работа с различными СУБД;

Для работы с *базами данных* в PHP существуют различные модули. Наиболее значимыми можно назвать «DBA», «dbx», «ODBC» и «PDO» [24].

«DBA» представляет из себя уровень абстракции между PHP и некоторыми файловыми базами данных.

«dbx» - конвенция написания модулей для взаимодействия с различными базами данных (где «x» - название СУБД).

«ODBC» - расширение, позволяющее программам на PHP взаимодействовать с СУБД, имеющими «ODBC» интерфейс. Таким образом обеспечивается широкий спектр поддерживаемых баз данных.

«PDO» - объекты данных PHP. Определяет простой и согласованный интерфейс для доступа к базам данных в PHP. Каждый драйвер базы данных, в котором реализован этот интерфейс, может представить специфичный для базы данных функционал в виде стандартных функций расширения. Но надо заметить, что само по себе расширение PDO не позволяет манипулировать доступом к базе данных. Чтобы воспользоваться возможностями PDO, необходимо использовать соответствующий конкретной базе данныхPDO драйвер. PDO обеспечивает абстракцию (доступа к данным). Это значит, что вне зависимости от того, какая конкретная база данных используется, вы можете пользоваться одними и теми функциями для выполнения запросов и выборки данных. PDO не абстрагирует саму базу данных, это расширение не переписывает SQL запросы и не эмулирует отсутствующий в

СУБД функционал. Если нужно именно это, необходимо воспользоваться полноценной абстракцией базы данных.

Несмотря на наличие средств для разработки прикладных приложений на PHP, возможности выполнения PHP кода на мобильных устройствах нет, ввиду отсутствия PHP-интерпретаторов для мобильных платформ.

2.1.5. Обзор возможностей языка Python

Python – широко применяемый высокоуровневый, интерпретируемый язык общего назначения [25]. Использует динамическую типизацию. Создавался с упором на читаемость кода, что позволяет описывать выражения меньшим числом строк, чем понадобилось бы, например, на C++ или Java.

Python поддерживает несколько парадигм программирования, включая объектно-ориентированное, императивное и функциональное программирование. Язык использует систему динамической типизации и автоматическое управление памятью. Имеет обширную библиотеку стандартных функций.

Интерпретаторы Python доступны для многих операционных систем, что обеспечивает его кроссплатформенность. С применением сторонних инструментов, таких как «Py2exe» [26] или «Pyinstaller» [27]. Программы на этом языке могут быть упакованы в самостоятельные приложения, что позволяет распространять программы на Python без необходимости установки интерпретаторов.

Python *не содержит встроенных средств для создания web-сервисов*, однако доступны сторонние расширения. Среди них можно отметить наиболее популярные:

- flask [28];
- Pywebsvcs [29];
- Webservice [30];

Python имеет встроенные средства для работы с базами данных с помощью интерфейсов «ADO» и «ODBC». Кроме того имеется возможность использования встроенных в язык СУБД «buzhug» [31] и «SnakeSQL» [32]. Отдельные разработчики СУБД, например «MySQL» предоставляют собственные модули для работы со своими базами данных в Python [33].

Для ведущих мобильных ОС (Android, iOS, Windows Phone) существуют интерпретаторы Python, что делает возможным разработку программ на Python для мобильных устройств. Однако сам по себе язык не содержит средств создания графических пользовательских интерфейсов, поэтому такие мобильные приложения ограничатся лишь режимом командной строки, что *не соответствует современным представлениям о мобильном* ПО.

2.1.6. Обзор возможностей языка C#

C# - объектно-ориентированный язык программирования со строгой типизацией [34]. Поддерживает императивную, декларативную, функциональную, обобщённую, объектно-ориентированную и компонентно-ориентированную парадигмы программирования. Разработан Microsoft как язык для платформы .Net как язык общего назначения. При разработке языка ключевыми параметрами выступали следующие принципы:

C# предполагался как простой, современный, объектно-ориентированный язык общего назначения.

Язык и его реализации, должны обеспечивать поддержку основных принципов разработки ПО, а именно контроль за типизацией, контроль границ массивов, выявление попыток использования не проинициализированных переменных и автоматическую сборку мусора. Важными факторами также являлись производительность, надёжность и сложность разработки.

Язык должен предусматривать возможность компонентной разработки в распределённой среде.

Переносимость исходного кода крайне важна: разработчики, знакомые с C или C++ должны иметь возможность разобраться в программе на языке C#.

C# должен обеспечить возможность разработки программ, как для полноценных, так и для встраиваемых систем с ограниченными ресурсами.

По своей архитектуре, C# в большей части отражает возможности низлежащей среды исполнения CLI. Большинство встроенных типов имеют точное соответствие в среде исполнения. Однако спецификация языка не указывает требования к коду, выдаваемому компилятором. Это означает, что нигде явно не прописано, что компилятор C# должен на

выходе выдавать код, нацеленный на CLI. Теоретически компилятор C# может выдавать машинный код как традиционные компиляторы C++ или Fortran.

Изначально существовала только Windows реализация среды исполнения под названием «.Net Framework». Однако свободная реализация «Mono» от компании «Xamarin» признана Microsoft как официальная реализация платформы .Net для Linux.

Для создания web-сервисов на языке C# используется фреймворк ASP.NET [35]. Данный фреймворк позволяет создавать современные web-сервисы с архитектурой MVC, при этом решая следующие задачи:

- размещение web-сервиса (внутри web-сервера IIS, nginx, либо как самостоятельный сервис);
- обработка HTTP запросов (сюда входят задачи маршрутизации, извлечение параметров, формирование ответов);
- работа с СУБД;
- сериализация и десериализация данных.

Изначально в состав .NET входит средство ADO.NET позволяющее получить доступ к данным, хранящихся в различных СУБД [36]. Однако в последнее время набирает популярность средство объектно-реляционного представления Entity Framework [37], позволяющее абстрагироваться от таблиц в базе данных и работать с ними как с программными объектами.

Платформа .Net имеет официальную реализацию под Windows Phone. Это означает, что программы, написанные на C# можно перенести на этот вид мобильных устройств. Программный комплекс «Xamarin» даёт возможность перенести программы на C# и на остальные платформы. В случае с Android на мобильном устройстве предварительно развёртывается «Mono» - свободная реализация .Net, а затем целевое приложение. В случае с iOS специальный транслятор «Xamarin.iOS» выполняет преобразование исходного кода C# в исходный код Objective-C, который впоследствии компилируется под мобильное устройство.

Таким образом, на языке C# возможна кроссплатформенная разработка, в том числе и под мобильные устройства.

Для сравнения возможностей рассмотренных языков программирования, составлена таблица 2 («+» соответствие параметра, «-» несоответствие).

Таблица 2 - Сравнение возможностей языков программирования

Параметры	Обязательный параметр	Язык					
		C#	C++	Java	Objective-C	PHP	Python
Динамическая типизация	-	-	-	-	-	+	+
Макросы	-	-	+	-	-	-	-
Интерпретируемый	-	-	-	-	-	+	+
Сборщик мусора	-	+	-	+	+	+	+
Наличие транслятора под Windows Phone	+	+	-	-	-	-	-
Наличие транслятора под Android	+	+	+	+	-	-	-
Наличие транслятора под iOS	+	+	-	-	+	-	-

По итогам был выбран язык C#, как наиболее эффективный для реализации поставленной задачи.

2.2. Выбор среды разработки

С учётом выбранного языка оставалась задача выбора среды разработки, в которой будет происходить написание программы. С учётом того, что клиентская и серверная части разрабатываются на одном языке, ожидалось, что это можно будет возможно с применением одних и тех же инструментов.

2.2.1. Обзор среды Visual Studio

Visual Studio – интегрированная среда разработки от Microsoft [38]. Предназначена, в первую очередь, для разработки программ для Microsoft Windows, а также web-сайтов, web-приложений и web-сервисов. Visual Studio использует такие платформы разработки как Windows API, Windows Forms, Windows Presentation Foundation, Windows Store и Microsoft. Компиляторы, входящие в состав данной среды, создают как машинный, так и управляемый код.

Visual Studio включает в себя текстовый редактор кода с технологией автодополнения «IntelliSense», а также инструменты для рефакторинга кода. Среда включает в себя средства создания графических пользовательских интерфейсов, проектирования web-страниц и баз данных.

Visual Studio имеет возможность расширения с помощью сторонних модулей, благодаря чему доступен широкий ассортимент расширений, дополняющих среду различным функционалом.

Среда поддерживает различные языки программирования и, с помощью расширений, даёт возможность редактировать исходный код на, практически, любом языке программирования.

Список встроенных языков содержит:

- C;
- C++;
- C++/CLI;
- VB.NET;
- C#;
- F#;
- Python (устанавливается отдельно);
- Ruby (устанавливается отдельно);
- Node.js (устанавливается отдельно);
- M (устанавливается отдельно);
- XML/XSLT;
- HTML/XHTML;
- JavaScript;

– CSS.

Visual Studio имеет консоль управления модульным тестированием «NUnit» и содержит широкий ассортимент инструментов для разработки ASP.NET приложений. При создании ASP.NET проекта в Visual Studio используется специальный диалог, в котором можно указать основные параметры будущего приложения, благодаря чему задачи по первоначальной настройке и конфигурированию проекта будут выполнены автоматически. В состав среды разработки входит отладочный web-сервер «IIS Express», позволяющий отлаживать web-сайты без предварительной публикации.

При конструировании различных элементов ASP.NET Visual Studio предлагает технологию формирования кода «Scaffolding». С её помощью при создании новых элементов, например, контроллеров, Visual Studio вначале убедится, что установлены все необходимые для этого элемента библиотеки, а затем сформирует шаблон, дополнив который своим кодом, можно получить готовый контроллер.

В данной среде разработки присутствует редактор HTML файлов с поддержкой Razor синтаксиса. Кроме того, система автодополнения IntelliSense поддерживает наиболее популярные библиотеки Web-разработки, такие как jQuery и AngularJS.

Также имеется интеграция со средствами публикации приложений «Web Deploy», что делает процесс развёртывания web-приложений в интернете предельно простым.

В Visual Studio встроена *поддержка средств контроля версий* «Git» и «TFS». Благодаря этой интеграции, обеспечивается совместная работа над проектами и хранение предыдущих версий исходных кодов без необходимости переключаться между программами.

В перечень расширений для Visual Studio входят комплекты разработчиков для устройств Windows и Android, которые дополняют среду эмуляторами мобильных устройств для тестирования мобильных программ.

Visual Studio поддерживает разработку под Windows Phone после установки комплекта разработчика Windows Phone SDK, разработку под Android после установки расширения «Visual Studio for Android», либо «Xamarin for Visual Studio» и разработку приложений под iOS после установки расширения «Xamarin for Visual Studio» и подключения по сети к компьютеру Mac с установленной средой разработки Xcode.

Visual Studio 2015 предоставляется в различных вариантах поставки (см. табл. 3).

Таблица 3 - Варианты лицензирования Visual Studio

Редакция	Community	Professional	Enterprise
Описание	Бесплатный, полнофункциональный и расширяемый инструмент для разработчиков, создающих некорпоративные приложения	Профессиональные инструменты и службы для разработки, предназначенные для индивидуальных разработчиков или небольших групп.	Решение корпоративного уровня с расширенными возможностями для групп, работающих над проектами любого размера или сложности, включая углубленное тестирование и процесс разработки.
Цена	Бесплатно	$45/месяц	$250/месяц

2.2.2. Обзор среды Xamarin Studio

Xamarin Studio – интегрированная среда мобильной разработки созданная на базе «MonoDevelop» [39]. Основное направление разработки – Mono и .NET framework. Xamarin Studio включает в себя функции, аналогичные другим средам разработки, например, автодополнение кода, поддержку средств контроля версий, дизайнер Web-страниц и пользовательских интерфейсов.

Поддерживаемые языки разработки:

- Boo;
- C;
- C++;
- C#;
- CIL;
- D;
- F#;
- Java;

– Oxygene;
– Vala;
– VB.NET.

В состав Xamarin Studio входит средство модульного тестирования «NUnit».

Xamarin Studio позволяет вести ASP.NET разработку, обеспечивая при этом лишь базовые возможности, такие как создание проекта и запуск отладочного web-сервера.

Xamarin Studio поддерживает интеграцию с «Git», но данная среда *не содержит эмуляторов мобильных устройств*, хотя имеется возможность управлять эмуляторами Android из комплекта «Android SDK».

Xamarin Studio определяет проекты Windows phone, позволяет просматривать и редактировать их исходный код, однако возможности компилирования, отладки, развёртывания и публикации отсутствуют.

Xamarin Studio позволяет вести разработку приложений для Android. Сюда входит редактирование кода, визуальный редактор графического пользовательского интерфейса, компилирование приложения, развертывание на устройстве и создание установочных файлов.

Xamarin Studio позволяет вести разработку приложений под iOS. В случае, если среда установлена на компьютер с «Mac OS X», то возможен полный цикл разработки, включающий отладку и тестирование. В случае, если среда установлена на другую ОС, то требуется наличие сетевого подключения к компьютеру Mac.

Xamarin Studio предоставляется бесплатно.

2.2.3. Обзор среды Sharp Develop

SharpDevelop – бесплатная интегрированная среда разработки с открытым исходным кодом для разработки под платформы .NET и Mono [40]. Изначально разрабатывалась как бесплатная облегчённая альтернатива Visual Studio. Содержит все основные функции Visual Studio Express, включая функции управления проектами, редактирования кода, компилирования и отладки приложений. Для упрощения перехода на данную среду разработки, SharpDevelop поддерживает работу с файлами проектов Visual Studio. Присутствует возможность компилирования приложений для .NET Framework версий 2.0, 3.0, 3.5, 4.0 и .NET Compact Framework 2.0 и 3.5.

Поддерживаемые языки:

- C#;
- VB.NET;
- Boo;
- F#;
- IronPython;
- IronRuby.

SharpDevelop содержит дизайнер пользовательских интерфейсов, подсветку кода, средства автодополнения и возможность отладки приложений .NET Framework.

В SharpDevelop интегрировано средство модульного тестирования «NUnit» а также средства Web-разработки ASP.NET. В них входят:

- дизайнер HTML без средств форматирования исходного кода;
- дизайнер форм ASP.NET;
- автодополнение для ASP.NET, JavaScript и CSS;
- средства публикации сайта по протоколу FTP;
- управление web-сервером IIS;
- поддержка средств контроля версий.

Sharp develop поддерживает широкий спектр систем контроля версий:

- Subversion;
- Git;
- Mercurial;

В комплекте с SharpDevelop не предоставляются эмуляторы мобильных устройств, отсутствуют средства мобильной разработки, однако, данная среда, является свободным ПО и распространяется бесплатно.

2.2.4. Обзор среды Geany

Geany – простой графический текстовый редактор с базовыми функциями среды разработки [41]. Разрабатывался с целью создать максимально простую интегрированную среду разработки, с максимально возможной скоростью запуска и минимальным числом зависимостей от сторонних библиотек.

Перечень функций:

– автодополнение;
– подсветка синтаксиса;
– «свёртка» блоков текста;
– навигация по коду;
– встроенный эмулятор терминала;
– система сборки, основывающаяся на сторонних компиляторах;
– расширяемость с помощью подключаемых модулей;
– множественное выделение текста;
– настраиваемая система «горячих» клавиш.

Поддерживаемые языки:

– C;
– C++;
– C#;
– Java;
– JavaScript;
– PHP;
– HTML;
– LaTeX;
– CSS;
– Python;
– Perl;
– Ruby;
– Pascal;
– Haskell;
– Erlang;
– Vala.

Geany не поддерживает модульное тестирование.

Geany не имеет средств ASP.NET разработки. Доступно только редактирование статичных HTML файлов.

В Geany отсутствует встроенная поддержка контроля версий, эмуляторы мобильных устройств в комплект поставки не входят.

Разработка мобильных приложений не предусмотрена. Возможно только редактирование исходного кода.

Разработка мобильных приложений не предусмотрена. Возможно только редактирование исходного кода, сборка и развертывание посредством «Android SDK».

Разработка мобильных приложений iOS не предусмотрена.

Тип лицензии: GNU GPL v2.

По итогам изучения возможностей различных сред разработки, была составлена таблица 4 («+» соответствие параметра, «-» несоответствие).

Таблица 4 - Сравнение возможностей интегрированных сред разработки

Параметры	Обязательность параметра	Среды разработки			
		Visual Studio	Xamarin Studio	SharpDevelop	Geany
Модульное тестирование	+	+	+	+	-
Возможность Web-разработки	-	+	+	+	-
Поддержка средств контроля версий	-	+	+	+	-
Эмуляторы мобильных устройств	+	+	-	-	-
Разработка под Windows Phone	+	+	-	-	-
Разработка под Android	+	+	+	-	-
Разработка под iOS	+	+	+	-	-
Тип лицензии	+	Бесплатно (издание Community)	Бесплатно	Бесплатно	Бесплатно

В результате сравнения, была выбрана интегрированная среда разработки Visual Studio.

2.3. Разработка web-сервиса

Разработка web-сервиса осуществлялась по технологии ASP.NET в интегрированной среде разработки Visual Studio 2015.

2.3.1. Особенности технологии ASP.NET Core

ASP.NET Core [42] – это кроссплатформенная платформа для создания современных облачных web-приложений с применением .NET [43]. Платформа имеет модульную структуру с минимальными накладными расходами, благодаря чему обеспечивает гибкий подход к разработке решений. Поддерживается разработка в среде Windows, Mac и Linux. ASP.NET Core имеет полностью открытый исходный код, доступный на GitHub [44].

Первая версия ASP.NET появилась в 2001 году. С тех пор миллионы разработчиков использовали её для создания высококачественных web-приложений. В течении многих лет платформа непрерывно развивалась и дополнялась новыми возможностями.

В ASP.NET Core присутствует ряд архитектурных изменений, благодаря чему платформа стала более компактной и модульной. ASP.NET Core больше не зависит от классической библиотеки «System.Web.dll». Вместо этого функционал платформы предоставляется по частям в виде отдельных NuGet пакетов [45], что позволяет использовать только необходимые компоненты платформы. Также это повышает защищенность программного продукта, отключая невостребованные возможности, и позволяет сэкономить ресурсы.

ASP.NET Core создавался с учетом потребностей современных web-приложений, например единый процесс разработки web-интерфейсов и web-API, которые зачастую вместе используются в одном приложении. Ещё ASP.NET Core разработан с возможностью интеграции в существующие облачные среды: проект конфигурируется с помощью переменных среды. Также присутствует возможность подключения сервисов через внедрение зависимостей.

2.3.2. Особенности технологии DNX

.NET Execution Environment (DNX) [46] - это среда выполнения и одноименный комплект разработчика для написания .NET приложений на Windows, Mac и Linux. Он предоставляет хост-процесс, логику размещения CLR [47], и технологию обнаружения точки входа управляемого кода. DNX создавался для кроссплатформенных ASP.NET приложений, хотя он также может применяться в других типах проектов, например, кроссплатформенных консольных приложений.

2.3.3. Разработка моделей

Web-сервис разработан по концепции MVC [48]. Сначала был определён набор базовых сущностей (моделей) и база данных для хранения их в системе. Для реализации базы данных был использован инструмент Entity Framework [37]. В качестве методологии проектирования был выбран «Code First» [49]. Сущность данного метода заключается в том, что разработчику не нужно отдельно проектировать структуру базы данных - достаточно лишь описать сами объекты, а система сама определяет структуру таблиц, основываясь на конвенциях. Были созданы следующие модели (см. табл. 5-9):

Таблица 5 - CarManufacturer (Производитель (марка) автомобиля)

Свойство	Назначение	Тип данных
ID	Идентификатор	Целочисленный
Title	Название	Строковый
Models	Модели автомобилей, выпускаемые этим производителем	Коллекция типа CarModel

Таблица 6 - CarModel (Модель автомобиля)

Свойство	Назначение	Тип данных
ID	Идентификатор	Целочисленный

Продолжение таблицы 6 - CarModel (Модель автомобиля)

Title	Название	Строковый
Generations	Варианты исполнения данной модели	Коллекция типа CarGeneration
Manufacturer	Производитель	CarManufacturer

Таблица 7 - CarGeneration (Вариант исполнения автомобиля)

Свойство	Назначение	Тип данных
ID	Идентификатор	Целочисленный
Title	Название	Строковый
Model	Модель	CarModel
Category	Корневая категория ремонта	Category

Таблица 8 - Category (Категория ремонтных работ)

Свойство	Назначение	Тип данных
ID	Идентификатор	Целочисленное
Title	Название	Строковый
ParentCategory	Родительская категория	Category
ChildCategories	Дочерние категории	Коллекция типа Category
CarGenerations	Варианты исполнения автомобилей	Коллекция типа CarGeneration

Таблица 9 - Work (Описание выполняемых работ)

Свойство	Назначение	Тип данных
ID	Идентификатор	Целочисленное
Title	Название	Строковый
Time	Время работы в часах	Двойная точность

Продолжение таблицы 9 - Work (Описание выполняемых работ)

CarGeneration	Вариант исполнения автомобиля	CarGeneration
Category	Категория работы	Category

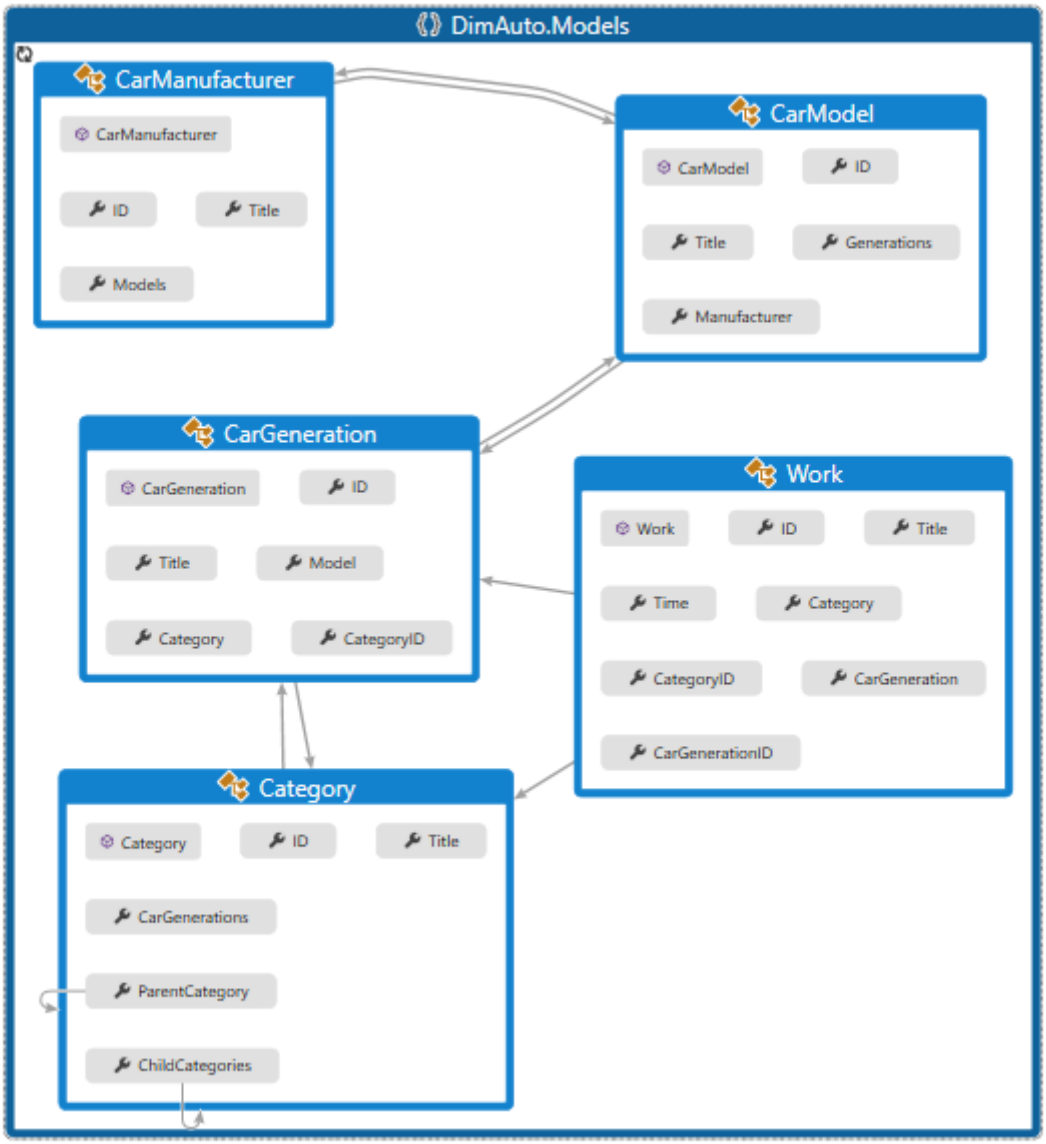

Рисунок 4 - Схема объектов (Visual Studio)

Проектирование базы данных происходит следующим образом. Entity Framework анализирует созданные объекты и формирует по таблице на каждый объект. Свойства примитивных типов становятся столбцами, в том случае если не содержат в своем имени ключевых слов, определяемых конвенциями. После этого анализируются связи между

объектами. К примеру, сущность «CarManufacturer» имеет свойство «Models», представленное коллекцией объектов «CarModel». В то же время объект «CarModel» имеет свойство «Manufacturer» сложного типа «CarManufacturer». На основании этого Entity Framework принимает решение, что между объектами «CarManufacturer» и «CarModel» имеется связь «Один-ко-многим». Для реализации этой связи в проектируемую таблицу «CarModels» добавляется столбец «CarManufacturerID», который впоследствии становится внешним ключом. Если в процессе анализа объектов определяется, что имеет место связь вида «многие-ко-многим», то Entity Framework создает «Joint-таблицу», в которой будет два столбца: первичный ключ из первой таблицы и первичный ключ из второй таблицы. Что примечательно, для формирования связей нет необходимости описывать зависимость объектов с обеих сторон. В случае, рассмотренном выше, достаточно было описать свойство «Models» в объекте «CarManufacturers». Для Entity Framework этого уже достаточно для того, чтобы предположить о существовании связи «один-ко-многим». В данном примере свойство «Manufacturer» в объекте «CarModel» добавлено в целях навигации. Навигационные свойства нужны для того, чтобы, обратившись к ним из кода, получить связанный объект. Внутри Entity Framework это реализовано двумя способами: «Lazy Loading» и «Eager Loading» [50]. Способ, используемый в конкретной ситуации определяется разработчиком. При обращении к таблице «CarManufacturers» в режиме «Lazy Loading» Entity Framework выполнит один запрос к базе данных на получение таблицы «CarManufacturers». При обращении к навигационному свойству «Models» конкретной записи из таблицы будет выполнен второй запрос к базе данных, в данном случае, к таблице «CarModels», с указанием значения внешнего ключа. Такой подход экономит память системы, но снижает скорость работы приложения при интенсивном обращении к сложным свойствам объектов. Если заранее известно, что придётся выполнить несколько запросов, то больше подойдёт метод «Eager Loading». Данный метод позволяет принудительно извлечь указанные свойства объектов при первом запросе.

Иногда Entity Framework определяет связи не так, как этого ожидал разработчик. Для тонкой настройки предусмотрено расширение «Fluent API» [51]. С помощью него можно явно указать как желаемый объект надо связать с другим объектом, какие столбцы использовать в качестве внешних ключей и т.д.

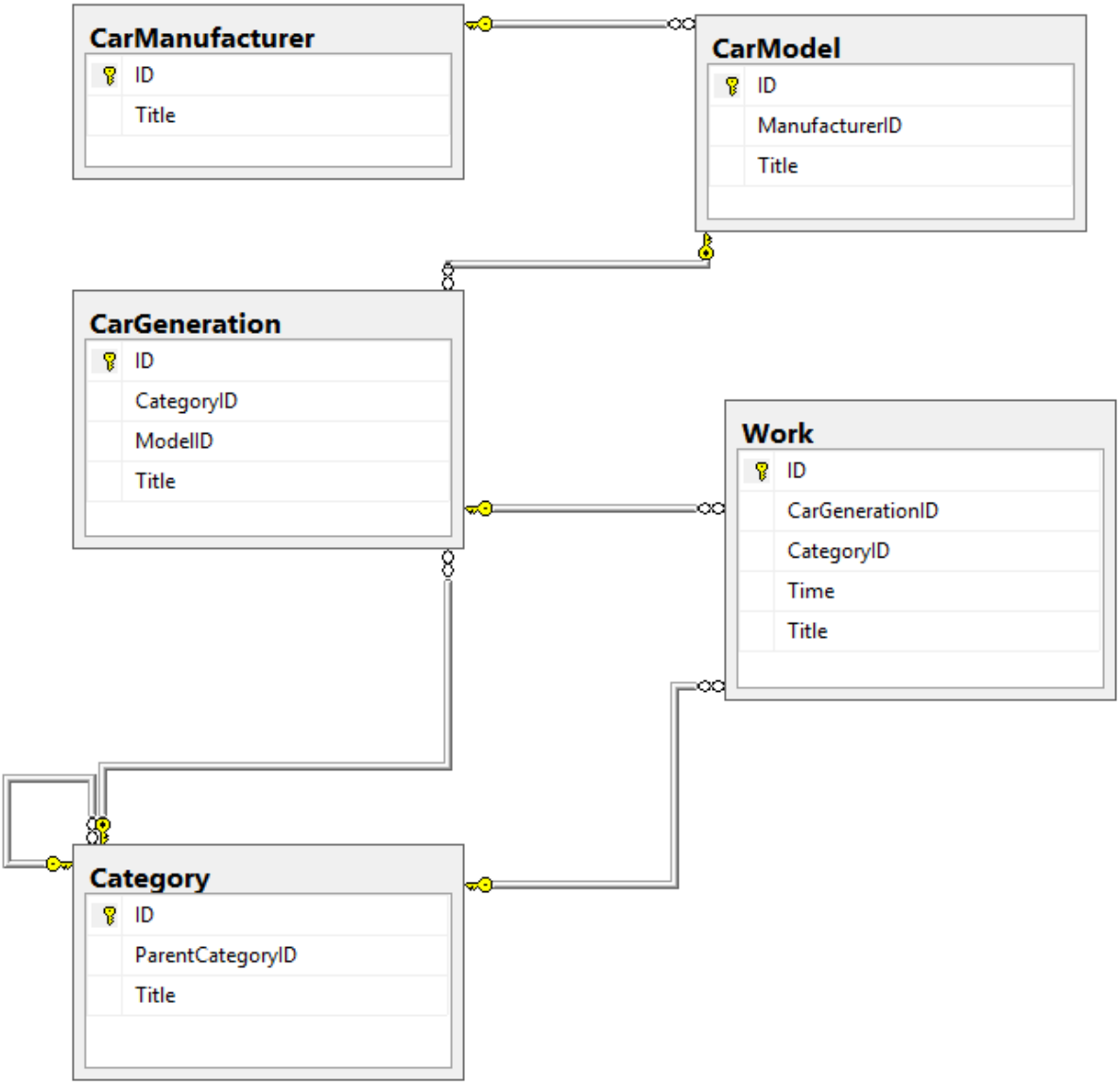

Рисунок 5 - Результирующая схема базы данных (SQL Management Studio)

Работа с базой данных через Entity Framework осуществляется с помощью LINQ-запросов [52] к объекту контекста данных.

2.3.4. Разработка представлений

Следующим этапом при создании web-сервиса была разработка представлений. В соответствии с техническим заданием нужно было разработать представления двух

категорий: пользовательский интерфейс для администратора, редактирующего базу данных и программный интерфейс (API) для мобильного приложения.

Разработка пользовательского интерфейса

Пользовательский интерфейс в приложениях ASP.NET представлен набором текстовых файлов с Razor-синтаксисом [53]. Данный синтаксис позволяет включить в традиционную HTML разметку [54] участки программного кода на C# или Visual Basic. При обращении к странице пользователем сервер выполняет участки с кодом, дополняя статичное HTML содержимое файла и возвращает результат в виде обычной HTML страницы.

Создание страниц с Razor-синтаксисом в интегрированной среде разработки Visual Studio происходит в автоматизированном режиме с помощью технологии «Scaffolding» [55]. Разработчик выбирает модель данных и даёт команду на создание контроллера. Пользовательский интерфейс при этом формируется автоматически, с учётом всех примитивных свойств модели. Для каждого свойства, вне зависимости от типа формируется поле текстового ввода, но с JavaScript-валидацией [56], запрещающей вводить в эти поля данные, отличающиеся по типу от свойств модели. Всего на каждую модель формируется по пять видов (см. табл. 10):

Таблица 10 - Автоматические формируемые виды и их назначение

Имя файла	Назначение
Create.cshtml	Интерфейс добавления объекта. Содержит текстовые поля и кнопку «Сохранить»
Edit.cshtml	Интерфейс редактирования объекта. Содержит текстовые поля и кнопки «Сохранить» и «Отмена»
Delete.cshtml	Интерфейс удаления объекта. Содержит текст с предупреждением и кнопки «Удалить» и «Отмена
Details.cshtml	Интерфейс подробных сведений об объекте. Демонстрирует содержимое примитивных свойств объекта. Впоследствии может быть доработан, путём добавления сложных свойств вручную

Продолжение таблицы 10 - Автоматические формируемые виды и их назначение

Index.cshtml	Интерфейс перечня объектов данного типа. Представлен таблицей. Столбцами являются примитивные свойства объекта. Также содержит три дополнительных столбца со ссылками на интерфейсы подробных сведений, редактирования и удаления объекта. Над таблицей имеется ссылка на интерфейс добавления нового объекта

После того, как виды были сформированы, разработчик, при необходимости, корректирует их вручную с помощью Visual Studio.

Разработка программного интерфейса (API)

В первую очередь при разработке программного интерфейса web-сервиса учитывались требования технического задания. Исходя из них, данные, передаваемые между web-сервисом и мобильным устройством должны быть представлены в виде объектов JSON [57], передаваемых по протоколу HTTP [58].

Традиционно для передачи данных в web-сервисах применялся протокол SOAP [59]. Данный протокол представлял данные в виде XML-документов [60] в специальной SOAP-обертке. Несмотря на то, что XML подходит для большинства случаев, он имеет недостатки, которые в определенных проектах могут быть существенными. Одним из таких недостатков является увеличение количества передаваемых данных при неизменном количестве информации.

JSON – это открытый текстовый формат передачи данных. Как и XML, он человекочитаем, платформонезависим и имеет широкий перечень реализаций. Данные, передаваемые в этом формате имеют небольшой объём и могут быть с легкостью обработаны в большинстве языков программирования. Исходя из этого, можно сказать, что этот формат можно применять практически в любой системе, где требуется передача структурированных данных.

Формат передачи данных JSON был разработан как подмножество представления объектов литералами в языке JavaScript. Однако, несмотря на мягкие правила синтаксиса литералов в JavaScript, следует помнить, что в JSON есть дополнительные ограничения.

Например, имя объекта обязано быть представленным в виде допустимой строки. Строки в JSON обязаны ограничиваться кавычками. В то же время в JavaScript свойства объектов могут быть ограничены кавычками, апострофами, или вообще не ограничиваться, если их имена не совпадают с ключевыми словами языка. Также значения свойств в JSON ограничено несколькими допустимыми типами, в то время как в JavaScript свойства объектов могут быть как литералами, объектами, выражениями или даже вызовами других функций.

Сообщение, форматированное по стандарту JSON, состоит из корневого объекта или массива. Элементами массива и свойствами объекта могут быть другие объекты, массивы, строки, числа, логические значения (истина и ложь) или «null».

Одной из проблемных частей JSON является отсутствие специального литерала для представления дат и времени. Объяснением этого выступает тот факт, что в JavaScript тоже отсутствует литерал такого вида: поддержка дат и времени в этом языке реализована специальным объектом «Date». Многие приложения, применяющие JSON, для передачи дат и времени часто используют строковые либо числовые литералы. Если используется строка, то ее следует закодировать по формату ISO 8601 [61]. Если используется число, то обычно оно отражает количество миллисекунд, прошедших с начала Unix-эпохи (01.01.1970). Однако стоит учитывать, что это просто правило, а не часть стандарта. При обмене данными с web-сервисом необходимо заранее обговорить формат даты и времени. Например, в Microsoft ASP.NET не используется ни одно из вышеупомянутых правил. Вместо этого происходит преобразование .NET объекта «DateTime» в строку в формате «\/Date(количество_тиков)\/», где «количество_тиков» представляет количество миллисекунд с Unix-эпохи.

JSON как и XML может использоваться для представления объектов в памяти компьютера в текстовом формате, читаемом людьми. Более того, оба этих формата изоморфны друг другу. Многие web-сервисы позволяют получать данные как в JSON, так и в XML формате. Тем не менее, нельзя однозначно сказать, что один формат лучше другого. При разработке решения следует внимательно изучить характер передаваемых данных, чтобы принять правильное решение. Например XML уходит корнями в системы разметки документов, что даёт ему явные преимущества в этом направлении (как например формат XHTML [62]). JSON произошел из программирования, где было важно отображать структуру вложенных объектов. Приведём наиболее значимые отличия (см. табл. 11):

Таблица 11 - Сравнение возможностей XML и JSON

Характеристика	XML	JSON
Типы данных	Не типизирует данные. Разработчику требуется дополнительно оформить XML-схему, если необходима типизация	Предоставляет скалярные типы данных и возможность отобразить структуру данных в виде массивов и объектов
Поддержка массивов	Для передачи массивов можно ввести правило, по которому элементы массива располагаются внутри другого элемента, имя которого является множественной формой названия типа элементов массива.	Встроенная поддержка массивов
Поддержка объектов	Для передачи объектов можно ввести правило, по которому определенные свойства передаются как атрибуты, а другие как вложенные элементы	Встроенная поддержка объектов
Поддержка значения «null»	Требует использования специального объекта «xsi:null», расположенного в специальном пространстве имен	Изначально распознаёт значение «null»
Комментарии	Встроенная поддержка комментариев	Не поддерживаются

Продолжение таблицы 11 - Сравнение возможностей XML и JSON

Пространства имен	Поддерживает пространства имен, что предотвращает коллизии при объединении документов. Также возможно беспрепятственное расширение стандарта.	Пространства имен не предусмотрены. Коллизии имен обычно устраняются путём разнесения объектов в разные узлы, или добавлением специальных префиксов к именам.
Принципы форматирования	Сложные. Требуются затраты на определения того, как соотнести объекты и XML-представление. Неоднозначность в использовании атрибутов	Простые. Предусмотрен один способ представления объектов. Единственное разногласие касается представления дат и времени
Размер документа	Документы имеют большой объем. Это особенно заметно, если не используется атрибуты	Форматирование представлено отдельными символами, поэтому большую часть объема представляют сами данные в текстовом виде.

Работа с данными в формате JSON на языке C# довольно тривиальна. При использовании библиотеки «Newtonsoft.Json» можно в один оператор получить JSON текст из сложного объекта и наоборот.

2.3.5. Разработка контроллеров

Контроллер – главная алгоритмическая часть приложения MVC. Именно здесь происходит основная работа с данными, представленными моделями, и подготовка их к отображению с помощью видов. Контроллер – это связующее звено между данными и интерфейсом.

При использовании механизма «Scaffolding» Visual Studio автоматически создает контроллер и вид с базовыми возможностями по отображению, добавлению, изменению и удалению данных. Затем контроллер можно доработать, снабдив его дополнительными функциями.

Контроллер для пользовательского интерфейса web-сервиса не потребовал каких-либо изменений – результат операции «Scaffolding» соответствовал требованиям технического задания.

Во время разработки контроллера для программного интерфейса возникла проблема рекурсивного обхода навигационных свойств объектов. Рассмотрим контроллер производителей автомобилей «CarManufacturersController». При выполнении операции «Index» контроллеру необходимо извлечь из базы данных все объекты «CarManufacturer», сериализовать их в формат JSON и вернуть запрашивающей стороне. Однако внутри каждого объекта «CarManufacturer» есть навигационное свойство «Models», при обращении к которому произойдет извлечение из базы соответствующих моделей автомобилей в виде объектов «CarModel». Каждый объект «CarModel», в свою очередь имеет обратную ссылку на производителя, представленного объектом «CarModel». В итоге при попытке сериализовать результат первого запроса возникнет бесконечная рекурсия, которая закончится переполнением программного стека и аварийным завершением программы. Для решения данной проблемы используется подход с применением специальных объектов передачи данных DTO [63]. Такие объекты представляют из себя «урезанные» версии объектов моделей, без «лишних» навигационных свойств. В ходе работ были созданы DTO объекты для всех случаев, где возникала рекурсия.

Для использования DTO объектов требуется реализовать преобразование типов данных из полных объектов в DTO. В данном решении использовалась библиотека «AutoMapper» [64]. Принцип её работы основан на том, что свойства двух объектов с одинаковыми именами эквивалентны. Библиотека работает во время исполнения программы и динамически определяет структуру объектов с помощью языковой функции C# Reflection. Для успешной работы этой технологии разработчику необходимо объявить перечень допустимых преобразовании в блоке инициализации программы. В ASP.NET для этого наилучшим образом подходит файл «Startup.cs». После этого в контроллере можно выполнить преобразование полноценного объекта «CarManufacturer» в

«CarManufacturerDTO», передав результат запроса к базе данных в статическую функцию библиотеки «AutoMapper».

2.4. Разработка мобильного приложения

Разработка мобильного приложения производилась с применением технологии Xamarin [65]. Как и в случае с web-сервисом был выбран паттерн MVC. При кроссплатформенной разработке такой подход позволяет выделить модели и контроллеры в платформонезависимую часть, а виды (интерфейсы) описать под каждую платформу отдельно. В результате бизнес-логика приложения описывается один раз, что экономит время разработки, позволяет уменьшить различия в поведении приложения на различных платформах, но в то же время предоставить пользователям привычный интерфейс.

Специфика кроссплатформенной разработки для стационарных ПК и мобильных устройств имеет одно важное отличие. Пользовательские интерфейсы операционных систем для настольных компьютеров разработаны по одному и тому же принципу. Это окно, с заголовком, со строкой меню, полосами вертикальной и горизонтальной прокрутки, и строкой состояния. В зависимости от приложения некоторые из вышеперечисленных элементов могут отсутствовать, но это не сильно влияет на взаимодействие пользователя с приложением.

В мобильных ОС, ввиду аппаратных различий самих устройств, концепции графических пользовательских интерфейсов сильно отличаются. Для примера рассмотрим навигацию внутри приложения. Из-за малых размеров экрана, разработчики разделяют весь функционал приложения на несколько экранов, сменяющих друг друга при необходимости. Когда возникает потребность у пользователя вернуться на предыдущий экран, то способ зависит от используемой платформы. В ОС Android и Windows Phone на мобильном устройстве имеется аппаратная клавиша «Назад». С помощью этой клавиши можно выполнять возврат к предыдущему экрану, либо отменить выполняемое действие. В iOS на устройстве нет такой клавиши. Единственной навигационной клавишей является кнопка «Домой», которая возвращает систему на начальный экран. Поэтому в приложениях для iOS кнопка «Назад» реализуется в виде элемента пользовательского интерфейса. Именно из-за различия в интерфейсах мобильных ОС была выбрана технология Xamarin.

2.4.1. Разработка моделей

Поскольку разработка мобильного приложения велась на том же языке, что и web-сервис, удалось сократить затраты на разработку моделей, повторно использовав DTO-объекты web-сервиса. Таким образом была обеспечена полная совместимость принимающей и передающей частей решения. Сериализованные в JSON объекты на сервере безошибочно десереализуются в клиенте, с использованием тех же самых моделей.

2.4.2. Разработка представлений

По причинам, описанным во введении, разработка пользовательских интерфейсов под каждую платформу ввелась отдельно. Далее пойдёт детальное рассмотрение методик.

Windows Phone

Разработка пользовательских интерфейсов для Windows Phone производится с помощью инструментов, входящих в комплект разработчика (SDK) Windows Phone. Технология XAML, по которой описываются интерфейсы в Windows Phone имеет много общего с технологией WPF (Windows Presentation Foundation), по которой создаются интерфейсы традиционных Windows приложений. Наиболее правильно было бы сказать, что пакет разработчика для Windows Phone добавляет к уже существующим инструментам Visual Studio элементы пользовательского интерфейса телефона.

XAML – расширяемый язык разметки (Extensible Markup Language) является декларативным языком [66]. XAML позволяет инициализировать объекты и задавать их свойства с помощью языковых структур, отражающих иерархические взаимосвязи между несколькими объектами. С помощью XAML создаются пользовательские интерфейсы, состоящие из различных элементов. Затем в отдельном файле на языке C# производится описание реакция этих элементов на различные элементы. Также в этом файле можно программно управлять свойствами элементов. Язык XAML позволяет отделить программную и визуальную часть интерфейса, что даёт возможность редактировать интерфейс в различных дизайнерских средах, например, Microsoft Expression Blend.

XAML файлы обязательно содержат один-единственный корневой элемент, определяющий какой тип объекта, описываемого с помощью данного файла. XAML определяет два способа объявления объектов:

Напрямую, описывая объекты в виде элементов. Для описания одного объекта используются открывающий и закрывающий теги, между которыми можно разместить дочерние элементы.

Косвенно, описывая атрибуты элементов. Для описания объекта используется строковое значение. В теории такой подход уместен для описания любых элементов, кроме корневых. Данная операция является косвенной по отношению к XAML-процессору, поскольку ему должно быть известно, как создать этот объект, какие начальные значения присвоить его свойствам и т.д.

Прямое описание объекта в виде элемента

Чтобы объявить объект с помощью синтаксиса элемента, необходимо написать два тега, используя следующий шаблон, где «имя_объекта» следует заменить на имя класса, объект которого нужно создать:

```
<имя_объекта>
</имя_объекта>
```

Многие объекты пользовательского интерфейса могут содержать потомков. В таком случае описание будет происходить так:

```
<имя_объекта>
<имя_потомка>
</имя_потомка>
</имя_объекта>
```

Для удобства, если объект не содержит в себе вложенных объектов, то его описание можно сократить до одного тега:

```
<имя_объекта>
```

```
<имя_потомка />
</имя_объекта>
```

Инициализирующий текст

Иногда требуется описать элемент вместе с содержащимся внутри него текстом, который является начальным значением для конструктора. В XAML такая технология называется инициализирующим текстом. Теоретически использование инициализирующего текста — это то же самое, что вызов конструктора с параметрами, однако внутри XAML-процессора это происходит немного по-другому.

Задание свойств

Существует несколько способов задания свойств объектов в XAML:

- использование атрибутов;
- использование дочерних элементов;
- использование содержимого;
- использование коллекций.

Как и в случае с объявлением объектов, перечень доступных вариантов может варьироваться в зависимости от задаваемого объекта. Некоторые свойства могут быть заданы только одним из вышеперечисленных способов. Другие свойства могут допускать варианты. Например, свойство которое может быть задано как содержимое, также может быть задано как дочерний элемент типа «Содержимое»

Свойства доступные только для чтения не могут быть заданы никаким способом, ни через код, ни с помощью XAML. Использование коллекций может создать видимость того, что происходит задание свойств, предназначенных только для чтения, но на деле это не так.

Задание свойств с помощью атрибутов

С помощью следующего синтаксиса можно задавать свойства объектов. Где «имя_объекта» - имя класса, объект которого создается, «имя_свойства» - имя задаваемого свойства, а «значение_свойства» - значение:

```
<имя_объекта имя_свойства="значение_свойства" … />
```

или

```
<имя_объекта имя_свойства="значение_свойства">
… <!—дочерние элементы -->
</имя_объекта>
```

Следующий пример задаёт элемент прямоугольник (Rectangle) с четырьмя атрибутами: имя, ширина, высота и заливка (Name, Width, Height и Fill соответственно):

```
<Rectangle Name="rectangle1" Width="100" Height="100" Fill="Blue" />
```

То же самое на языке C# выглядело бы следующим образом:

```
Rectangle rectangle1 = new Rectangle();
rectangle1.Width=100.0;
rectangle1.Height=100.0;
rectangle1.Fill = new SolidColorBrush(Colors.Blue);
```

Первая строка – вызов конструктора по-умолчанию. Стоит заметить, что значение свойства «Name» используется в качестве имени объекта, которому будет присвоен результат выполнения конструктора. Остальные строки выполняют присваивания значений свойствам.

Задание свойств с помощью элементов свойств

Многие свойства объектов могут быть описаны с помощью других элементов. Для того, чтобы это сработало, необходимо, чтобы присваиваемое значение в свою очередь было объектом, который можно инстанциировать с помощью XAML, а затем «заполнить» им свойство родителя.

Чтобы использовать данный синтаксис, необходимо описать вложенный элемент по шаблону «<объект.свойство>». По стандарту XML, такой элемент будет просто

вложенным тегом, с точкой в названии. Однако с позиции XAML точка в названии определяет элемент как описывающий свойство своего родителя. В следующем примере «свойство» - имя задаваемого свойства, а «объект_являющийся_свойством» - новый объект, который станет свойством родительского объекта:

```
<объект>
<объект.свойство>
объект_являющийся_свойством
</объект.свойство>
 </объект>
```

Возвращаясь к примеру, с прямоугольником:

```
<Rectangle
Name="rectangle1"
Width="100"
Height="100"
>
<Rectangle.Fill>
<SolidColorBrush Color="Blue"/>
</Rectangle.Fill>
</Rectangle>
```

Задание свойств как содержимого

Некоторые типы объектов в XAML допускают задание свойств как содержимого. Это значит, что можно опустить объявление дополнительных элементов, определяющих это свойство и сразу объявить его внутри тега родителя. Например, свойство «Child» (Потомок) объекта «Borer» (Рамка) может быть задан как содержимое. Пример:

```
<Border>
<Button …/>
</Border>
```

Это допустимо в тех случаях, когда свойство допускает свободную объектную модель, когда тип свойства объявлен как строковый, либо «Object». Например, в XAML допускается размещать текст как содержимое элемента, явно не создавая при этом элемент «String»:

```
<TextBlock>Здравствуйте!</TextBlock>
```

Задание свойств через коллекции

В XAML существуют несколько вариаций синтаксиса коллекций. Сперва может показаться, что с помощью XAML можно задавать свойства или коллекции доступные только для чтения. На самом деле это не так. XAML позволяет добавлять элементы в уже существующие коллекции, неявно вызывая метод «Add». Более того, иногда коллекции можно пополнять как содержимое, например, в случае с вертикальным списком:

```
<StackPanel>
<TextBlock>Здравствуй</TextBlock>
<TextBlock>Мир</TextBlock>
</StackPanel>
```

Использование визуального редактора

Крайне полезной возможностью интегрированной среды разработки Visual Studio можно считать визуальный редактор XAML. Если корневой элемент документа относится к визуализируемому типу (например, страница), то над текстовым редактором возникает окно предпросмотра.

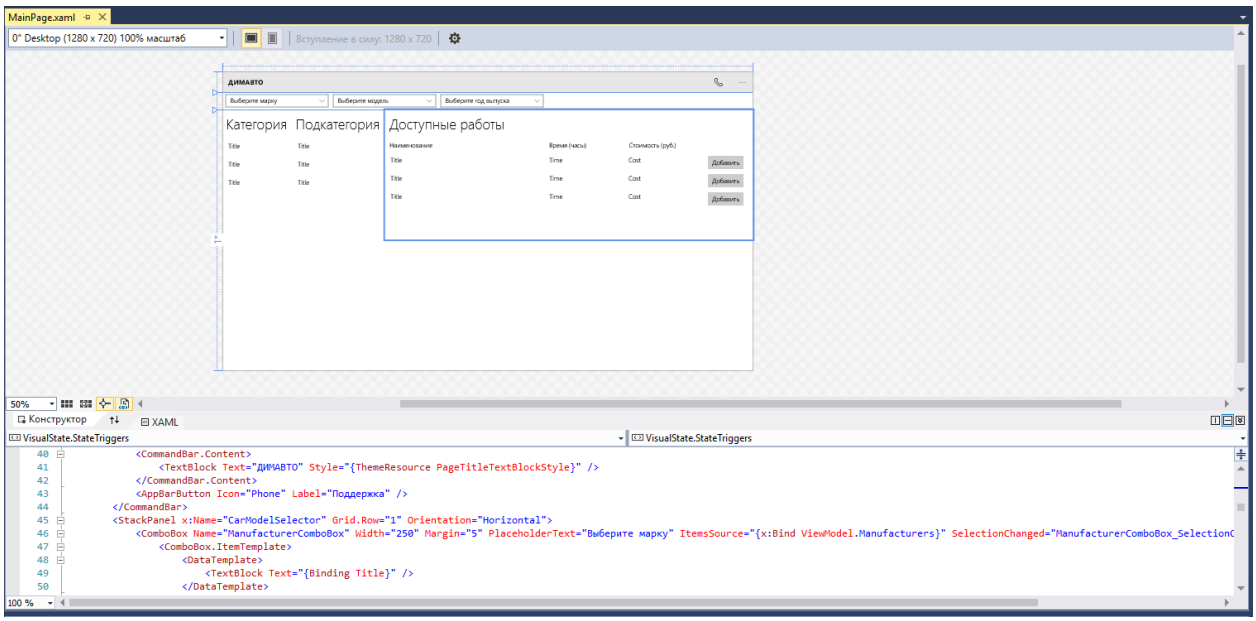

Рисунок 6 - Окно визуального редактора XAML в Visual Studio

Android

Разработка пользовательского интерфейса под Android концептуально схожа с разработкой под Windows Phone, однако имеет свои характерные особенности. Во-первых, для разработки интерфейсов из среды Visual Studio требуется специальная надстройка Visual Studio Integration [67] из комплекта Xamarin. Во-вторых, разные версии ОС Android включали в свой состав различные наборы элементов пользовательского интерфейса. Это означает, что в определённый момент разработки может оказаться, что один и тот же экран пользовательского интерфейса для разных платформ придется реализовывать отдельно, каждый из своего набора элементов.

Пользовательский интерфейс в ОС Android состоит из XML файлов, описывающих взаимное расположение элементов на экране. Описание реакции на события происходит в файлах с программным кодом, называемых операциями (Activity) [68] или фрагментами (Fragments) [69]. Строго говоря, концепция Android приложений не вписывается в паттерн MVC – в приложении не предусмотрено механизмов, обеспечивающих привязку элементов моделей с элементами пользовательского интерфейса. Редактирование XML файлов можно проводить как в визуальном редакторе, так и в текстовом режиме. Существенной недоработкой данной системы можно назвать то, что некоторые элементы пользовательского интерфейса не удается визуализировать на компьютере разработчика. В

таком случае, чтобы произвести предварительный просмотр, необходимо выполнить тестовое развертывание приложения на реальном устройстве, или эмуляторе.

iOS

Разработка пользовательских интерфейсов для iOS разительно отличается от остальных платформ. Проектирование основано на разработке общей схемы возможных переходов между экранами приложения, т.н. «Storyboard» [70]. Разработчик с помощью средств визуальной разработки размещает на экране прямоугольники, олицетворяющие экраны приложения. Затем, увеличивая масштаб, приближает вид к одному из них и снабжает его элементами управления из коллекции всех возможных.

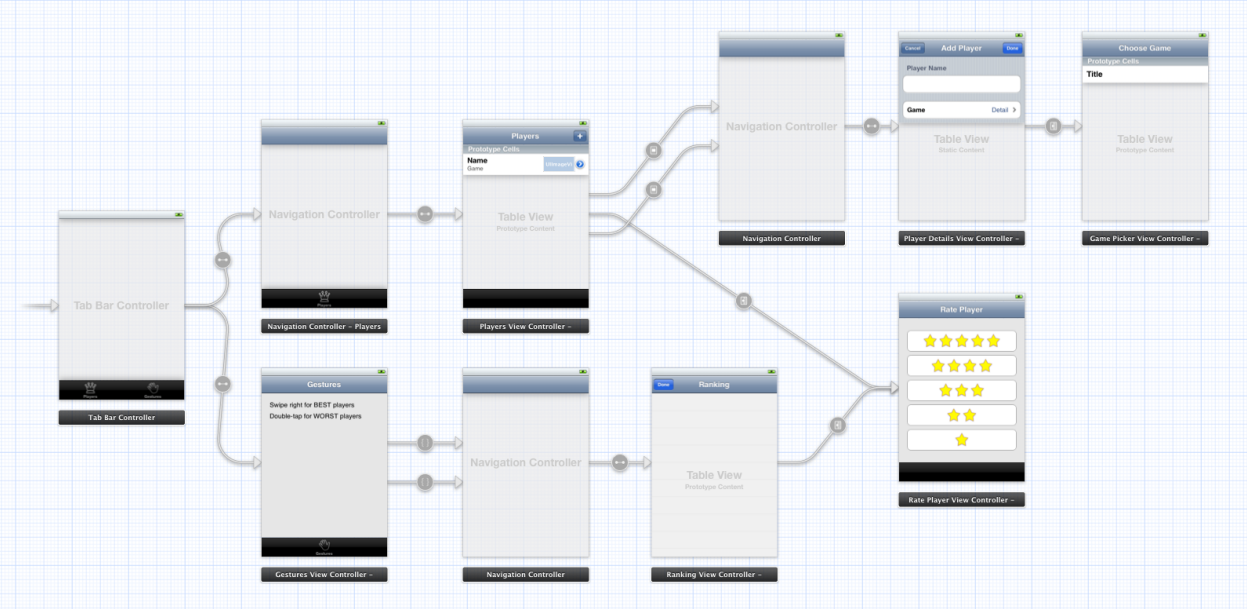

Рисунок 7 - Скриншот Storyboard мобильного приложения для iOS

Составление интерфейса средствами текстовой верстки в iOS недоступно. Для программирования переходов между экранами, разработчик непосредственно в средстве визуального проектирования проводит стрелку от элемента интерфейса к другому экрану. Такая стрелка в документации носит название «Segue» [71]. Тем самым создается обработчик события касания элемента выполняющий переход к другому экрану. В остальном концепция построения интерфейсов в iOS соответствует паттерну MVC – данные из моделей с помощью контроллеров «связывают» с элементами интерфейса.

2.4.3. Разработка контроллеров

Мобильное приложение, обрабатывающее данные по схеме, представленной на рис. 3, может состоять из одного контроллера, в задачи которого входит:

- получение фильтрующих категорий от сервера;
- обработка пользовательского выбора категории;
- запрос списка ремонтных работ по выбранным категориям;
- заполнение списка выбранных ремонтных работ;
- подсчёт итоговой стоимости.

Далее следуют детальные методики программирования под мобильные устройства, в зависимости от используемой платформы. Технология Xamarin обеспечивает переносимость программного кода между устройствами, однако у каждого из них остаются свои особенности, которые следует учитывать в процессе разработки.

Windows Phone

При программировании контроллеров в Windows Phone, технология Xamarin не применяется. Поскольку все инструменты для мобильной разработки под эту платформу входят в состав Windows Phone SDK и интегрируются в Visual Studio, то процесс программирования не имеет отличий от разработки под традиционную ОС Windows. «Привязка» данных в моделях в XAML интерфейсу осуществляется с помощью механизма свойств-зависимостей (Dependency Properties). Контроллер отвечает за работу с данными, хранящимися в его свойствах и не представляет о существовании пользовательского интерфейса. Вместо этого, определенные элемента интерфейса «привязаны» к конкретным свойствам контроллера. Это означает, что, при изменении этих свойств, интерфейс самостоятельно запросит их новые значения из контроллера и отобразит их пользователю. Существуют также варианты двухсторонней «привязки», когда контроллер в свою очередь тоже проверяет элементы пользовательского интерфейса и при обнаружении изменений, считывает и обрабатывает их.

Android

C# не является официально поддерживаемым Google языком программирования для Android. Поэтому здесь применяется технология «Xamarin.Android». Суть её заключается в поддержке ОС Android исполнения программного кода на C++. Разработчики Xamarin (в прошлом Novell) разработали специальную версию библиотеки «Mono» [72], которая добавляет в ОС Android возможность выполнять байт-код C#.

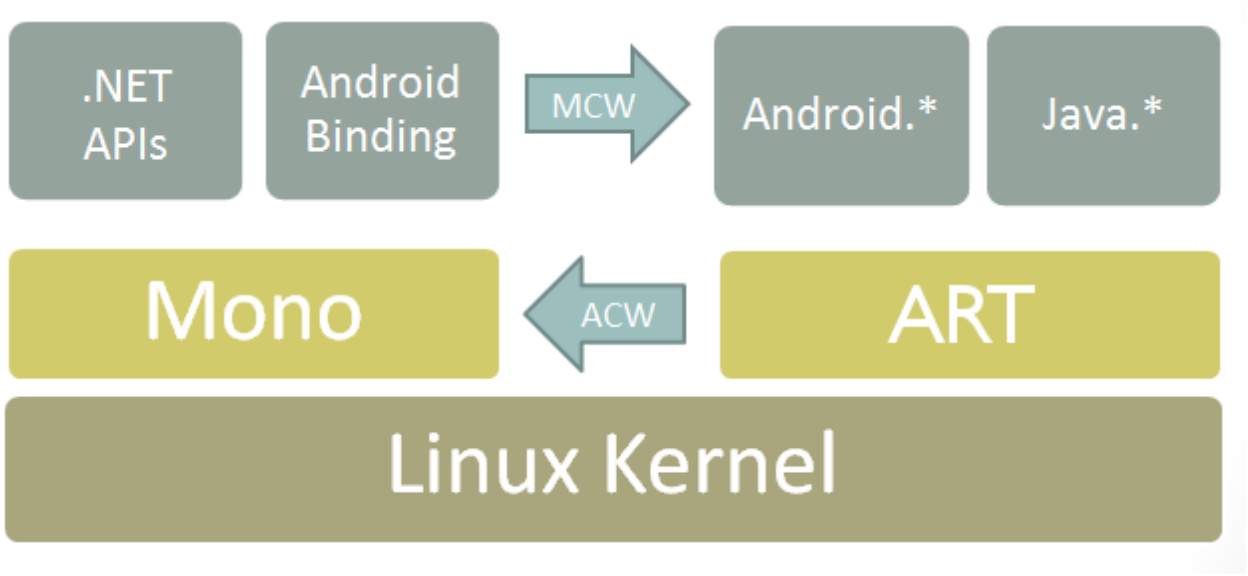

Рисунок 8 - Архитектура Xamarin на платформе Android

При этом оставался вопрос отрисовки пользовательского интерфейса, который должен был выполнятся в среде исполнения Android (ART), а не Mono. Для решения данного вопроса в Xamarin существуют программные обертки для управляемого кода (Managed Callable Wrappers) [73], с помощью которых их программы на C# можно сделать вызов к системным библиотекам Android на Java, для получения нужных графических объектов.

Как было сказано в предыдущем разделе, подсистема пользовательских интерфейсов Android не располагает к применению паттерна MVC, ввиду того, что нет встроенной возможности «связывания» графических элементов и свойств контроллера. Для преодоления данного неудобства для каждой страницы интерфейса был создан вспомогательный контроллер, который осуществляет принудительное обновление элементов интерфейса при изменении значений в контроллере.

iOS

Разработка под iOS связана с дополнительными трудностями, вызванными лицензионными ограничениями Apple. По лицензии приложения для iOS можно создавать только в интегрированной среде разработки Xcode [74], которая в свою очередь может быть установлена только на фирменные компьютеры Apple, такие как Mac Mini, Mac Book Pro и Mac Pro. Кроме того, архитектура iOS не предусматривает возможности написания программ на других языках, кроме официально поддерживаемых Apple. Ввиду всего этого, технология «Xamarin.iOS» предлагает следующее решение. Разработчик пишет код на C# на компьютере с установленной средой разработки Visual Studio. Затем, при сборке приложения, программа транслируется в исходные коды на C# и передаются по локальной сети на компьютер Apple, с установленной средой разработки Xcode. Там происходит окончательное компиляция программы и развертывание на мобильном устройстве или эмуляторе.

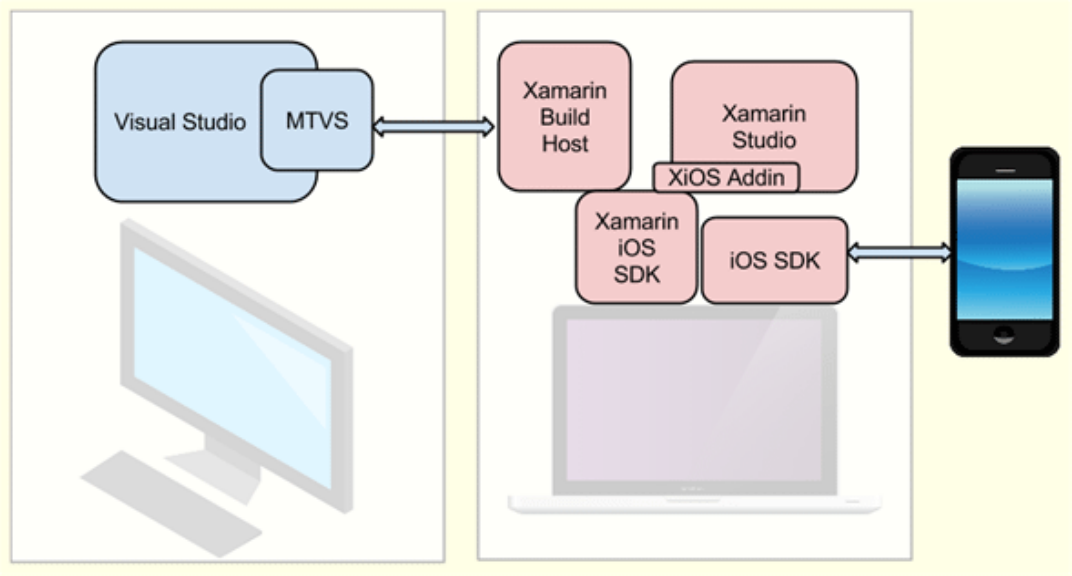

Рисунок 9 - Процесс сборки приложения под iOS с помощью Xamarin

Выводы по конструкторскому разделу

В качестве языка программирования был выбран C#, поскольку он сочетает в себе требуемые параметры и простоту использования. Среда разработки Visual Studio отвечает всем требованиям поставленной задачи. Паттерн MVC органично вписался как в архитектуру web-сервиса, так и клиента.

3. Технологический раздел

Для тестирования приложения использовались различные методики, рассчитанные как на испытание корректности программы, так и на отказоустойчивость развернутой системы.

3.1. Отладка web-сервиса

Для отладки web-сервиса применялись средства Visual Studio и функциональное тестирование с помощью HTTP анализатора Fiddler.

3.1.1. Отладка средствами Visual Studio

Интегрированная среда разработки Microsoft Visual Studio 2015 [75] предоставляет богатые возможности отладки программ.

Большинство синтаксических ошибок в коде выявляются ещё на этапе компиляции. Однако другие виды ошибок требуют отладки программы. Для этого выполняемая программа изучается с целью определения хода выполнения программы и контрольных значений данных. Используя инструменты отладки Visual Studio можно детально изучить как работает программа, выполняя обход каждого выражения и контролируя при этом значения переменных.

Для запуска процесса отладки необходимо подключить отладчик Visual Studio к выполняемому процессу в ОС. Кроме того, выполняемая программа должна быть скомпилирована с разрешением отладки. При запуске программы непосредственно из среды разработки отладчик присоединяется к процессу автоматически. В случае ручного подключения к процессу, необходимо определить хост-процесс web-сервера, выполняющего ASP [43] приложение. Обычно, этот процесс имеет имя «Aspnet_wp.exe», «dnx.exe» или «W3wp.exe» в зависимости от конфигурации web-сервера IIS [76].

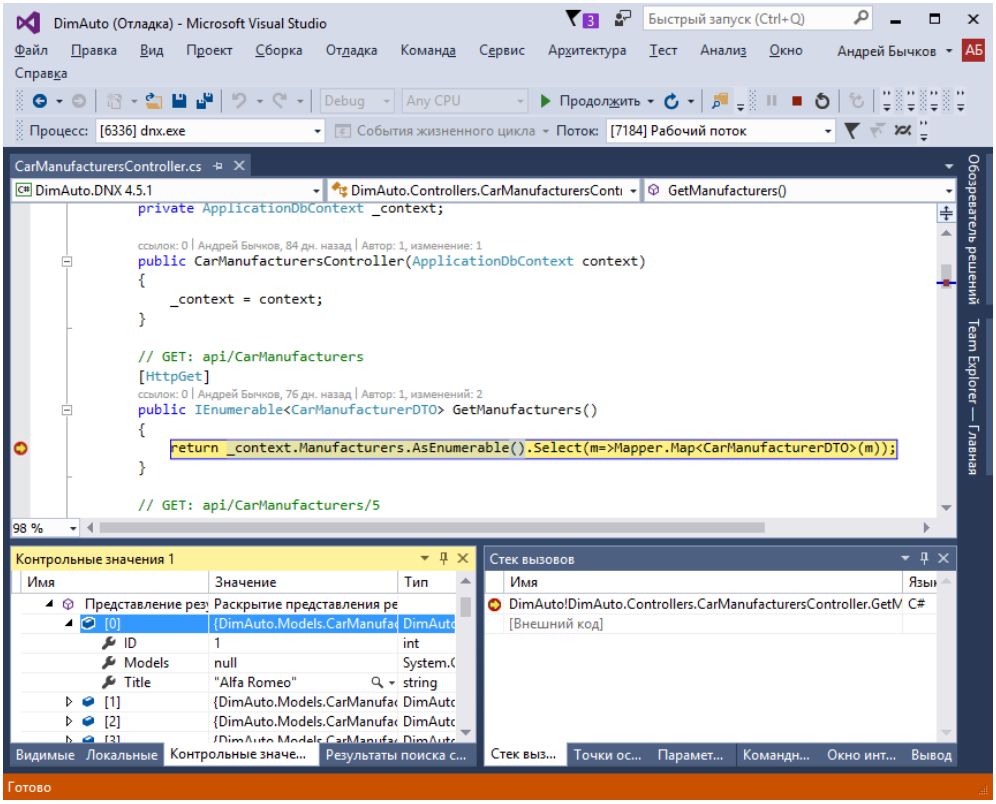

Рисунок 10 - Отладка в Visual Studio

3.1.2. Отладка средствами Fiddler

Программа Fiddler [77] является бесплатным отладочным прокси-сервером, совместимым с любым браузером, ОС и платформой. В основные возможности программы входит web отладка, тестирование производительности, запись HTTP и HTTPS трафика, управление web-сессиями и проверка безопасности. Кроме того, программа расширяема с помощью дополнений из каталога на сайте разработчика.

Принцип работы данной программы заключается в том, что она пропускает клиентские запросы к web-серверу через себя, выступая в качестве прокси-сервера. Благодаря этому появляется возможность анализировать запросы и ответы к ним со всеми подробностями. Помимо этого, имеется возможность формировать запросы к web-серверу самостоятельно непосредственно из Fiddler, без использования клиентской части приложения. Это полезно в тех случаях, если в клиентском приложении содержится ошибка, препятствующая формированию корректных запросов, или какая-либо часть функционала ещё не реализована.

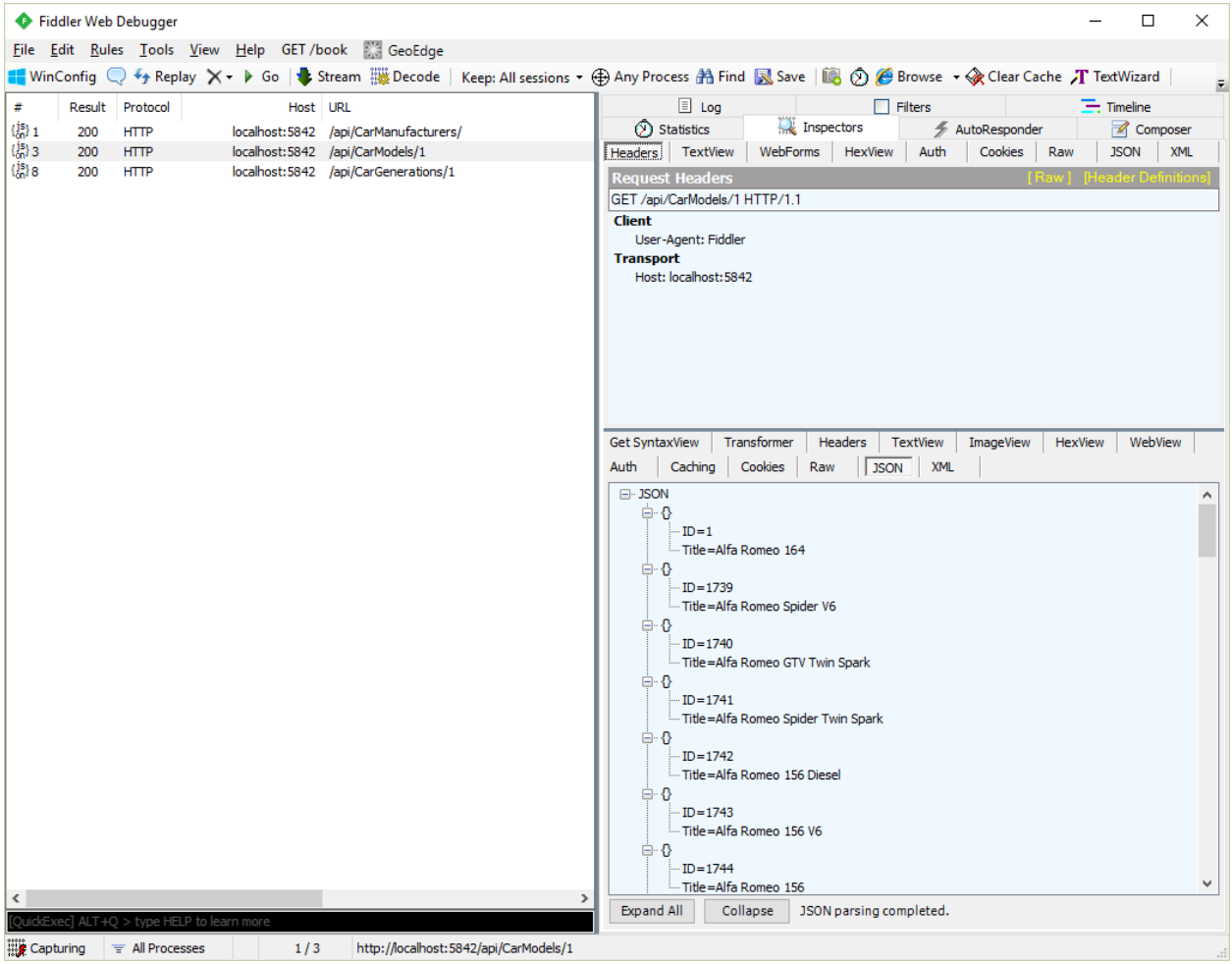

Рисунок 11 - Отладка в Fiddler

3.2. Отладка мобильного приложения

Отладка мобильных приложений связана с дополнительными трудностями, вызванными различиями в аппаратном обеспечении компьютера разработчика и мобильного устройства. Несмотря на то, что многие комплекты разработчиков (SDK) для мобильных устройств поставляются с эмуляторами мобильных платформ, зачастую этого бывает недостаточно. Эмулятор не даёт в полной мере протестировать корректность работы программы на устройстве. С помощью эмулятора можно лишь проверить, что программа не имеет явных ошибок в своей структуре. Многие аппаратные особенности мобильных устройств (такие как сенсорный экран с множественным касанием, акселерометр, компас и т.д.) невозможно качественно сымитировать.

Для отладки кроссплатформенной части мобильного приложения применялся отладчик Visual Studio. Для отладки платформозависимого кода применялись

специфические средства. В случае с платформой Windows использовался отладчик из комплекта Windows Phone SDK [78]. В случае с Android применялся инструмент ADB Logcat [79]. Для iOS применялся GDB [80].

3.2.1. Отладка средствами Windows Phone SDK

В состав Windows Phone SDK входит отладчик, который интегрируется со средой разработки Visual Studio и позволяет проводить отладку программ как на эмуляторах Windows Phone, так и на реальном устройстве.

Эмулятор Windows Phone создан на базе технологии виртуализации от Microsoft Hyper-V [81], которая в свою очередь использует технологии аппаратной виртуализации Intel VT [82], либо AMD-V [83], в зависимости от типа процессора, установленного в компьютере разработчика. Такой подход обеспечивает приемлемый уровень производительности эмулятора, близкий к настоящему смартфону. В комплект SDK входят пять образов для виртуальной машины, каждый из которых представляет определенное семейство смартфонов на базе Windows Phone. Образы отличаются размерами экрана и объемами доступной оперативной памяти, что соответствует устройствам разных ценовых категорий: от бюджетных до аппаратов премиум класса. Помимо вышеуказанного эмулятор использует специальную оболочку XDE [84], имитирующую облик устройства. Эта оболочка даёт возможность эмулировать нажатие функциональных клавиш смартфона (таких как клавиша блокировки экрана, громкость, назад и т.д.) и симуляцию некоторых условий использования телефона, например, плохие условия приёма.

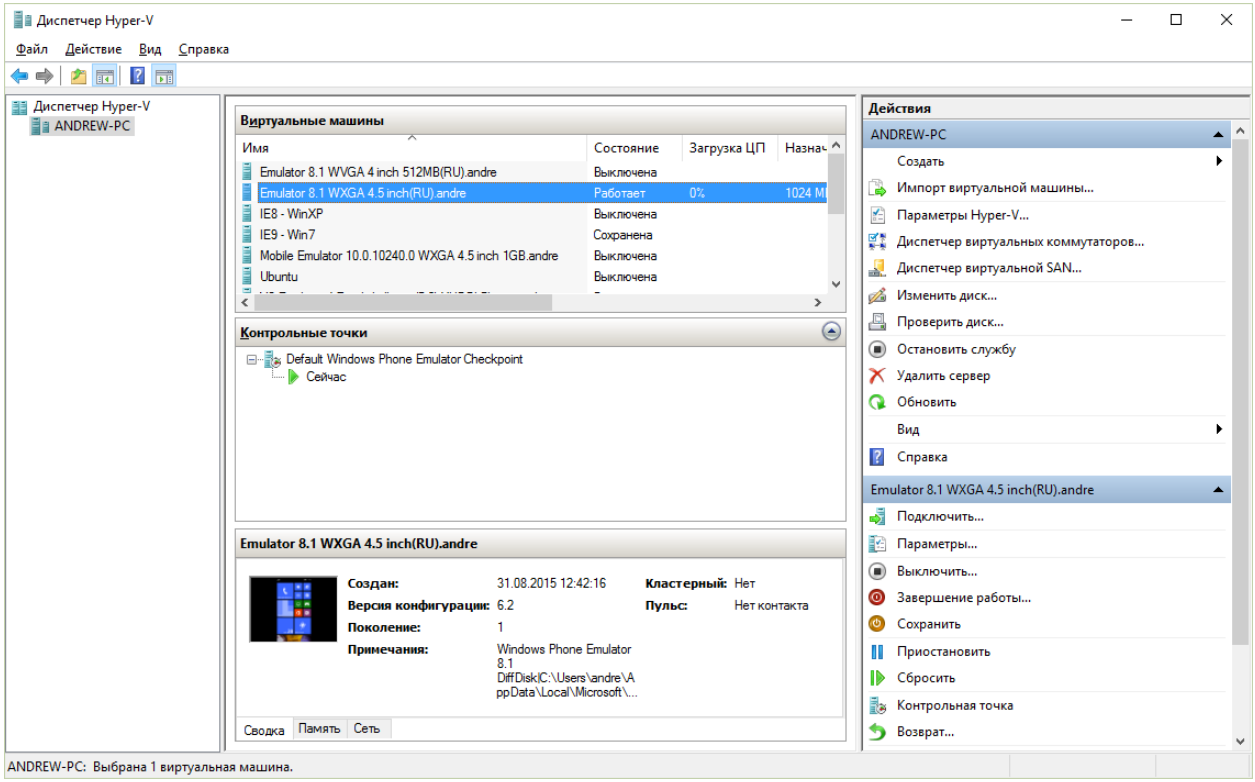

Рисунок 12 - Консоль Hyper-V с выделенным эмулятором Windows Phone

Рисунок 13 - XDE оболочка эмулятора Windows Phone

Для отладки на реальном устройстве с ОС Windows Phone необходимо предварительно получить лицензию разработчика. В целях предотвращения распространения некачественного ПО среди мобильных устройств, корпорация Microsoft реализовала возможность установки на смартфоны программы, полученные только из «Магазина Windows» [85]. Все программы, размещаемые в «Магазине Windows» предварительно тестируются сотрудниками Microsoft на предмет некачественного, либо вредоносного ПО. Эмулятор Windows Phone не содержит данного ограничения, поскольку предназначен для отладки программ до того, как их отправляют в «Магазин Windows». Для отладки на реальном устройстве его необходимо предварительно разблокировать. Это делается с помощью специальных инструментов, входящих в состав Windows Phone. При наличии у разработчика лицензии на разработку, он получает возможность устанавливать на свое мобильное устройство до пяти программ в обход «Магазина Windows».

3.2.2. Отладка средствами ADB Logcat

В состав компилятора Xamarin [65] входит комплект разработчика Android SDK [86]. Комплект разработчика для Android не имеет возможности полной интеграции с Visual Studio. Несмотря на попытку разработчиков Xamarin реализовать хотя-бы базовую интеграцию Android SDK со средствами разработки Visual Studio, полученное решение сложно назвать удобным. Программисту доступен лишь вывод трассировщика ADB [87], и построчное выполнение отдельных участков программы с просмотром контрольных значений. Механизм перехвата необработанных исключений работает крайне неудовлетворительно. При возникновении исключения в пользовательском коде, оно «поднимается» на уровень Mono Runtime [73], печатается в виде ошибки в журнал, но не доходит до Visual Studio по причине того, что ОС Android уже обработала его просто завершив работу приложения. В таких случаях приходилось построчно выполнять тот модуль программы, на котором она завершилась с ошибкой. Однако тут разработчика поджидает ещё одна проблема: отладчику не всегда удаётся сопоставить выполняемый код в устройстве с исходным кодом на компьютере разработчика.

Android SDK содержит в себе программную среду, целью которой является эмуляция различных устройств. Характерной особенностью Android является присутствие на рынке большого количества устройств с различными версиями ОС. Иными словами,

производители смартфонов разрабатывают устройства на базе разных версий ОС. Поэтому в Android SDK эмуляторы классифицируются не по производительности, а по версии ОС. Однако производительность эмуляторов, входящих в состав Android SDK не даёт возможности хотя-бы приблизительно оценить скорость работы программы. Android SDK эмулирует устройства с процессорами архитектуры ARM [88] без применения технологий виртуализации. Учитывая, что разработка ведётся на компьютерах с процессорами архитектуры x64 [89], либо x86 [90], такой подход крайне неэффективен, хотя позволяет использовать для разработки самый простой компьютер с недорогим процессором. Скорость работы приложений на этих эмуляторах настолько низкая, что иногда не понятно: работает ли программа или «зависла».

Исходя из этого пришлось искать другие эмуляторы. Выбор был сделан в пользу программного пакета «Visual Studio Emulator for Android» от корпорации Microsoft [91]. Данный эмулятор использует тот-же подход, что и эмулятор для Windows Phone: в качестве основы выступает виртуальная машина Hyper-V, а оболочкой, имитирующей телефон, является XDE. В данном случае образы разделены, как и в Android SDK по версиям ОС, хотя примечательной особенностью является упоминание моделей телефонов, на которых установлена данная версия ОС в перечне образов. Исходя из всего прочего эти эмуляторы удобны в использовании, поскольку ими можно управлять непосредственно из Visual Studio.

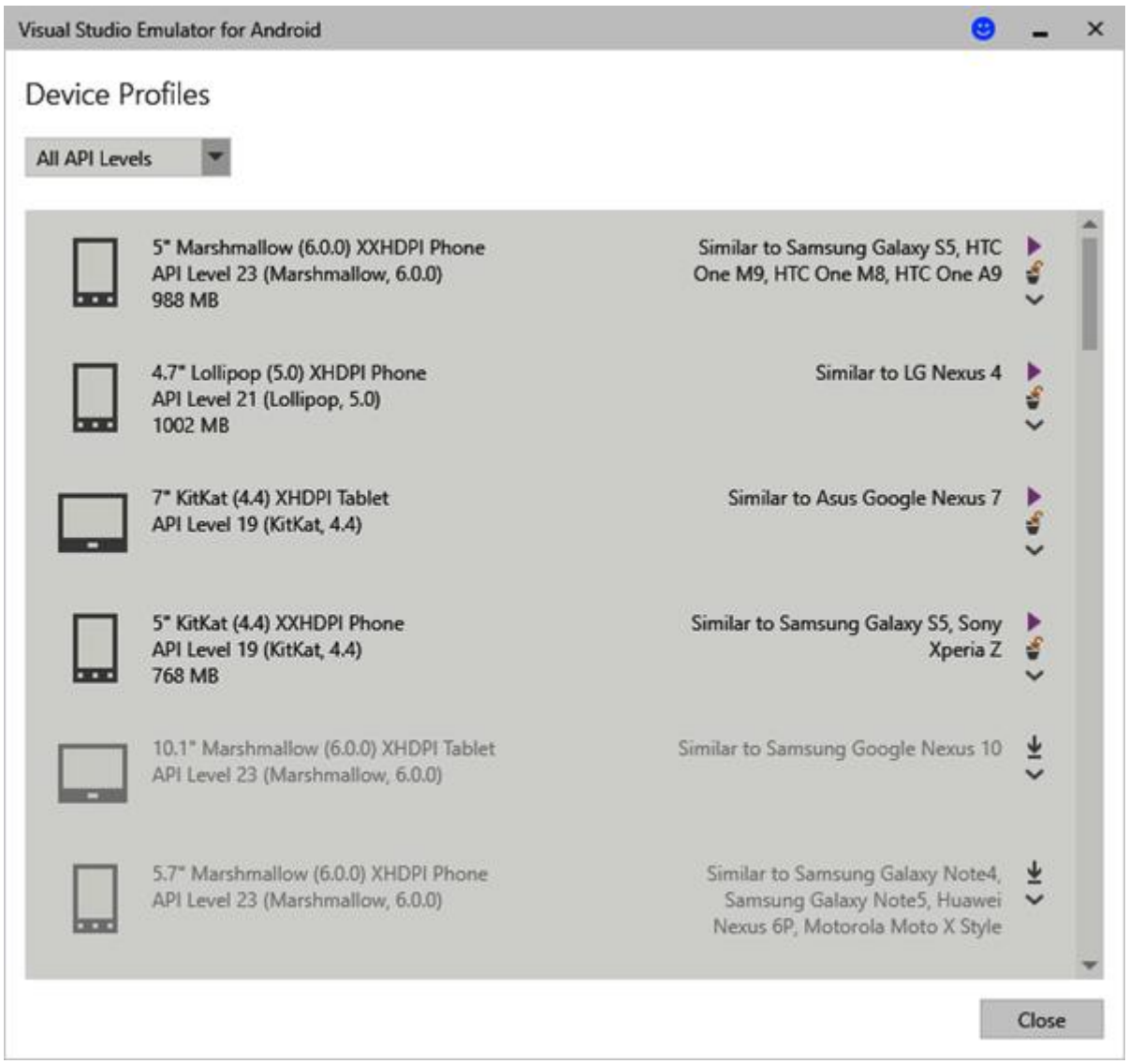

Рисунок 14 - Перечень образов в эмуляторе «Visual Studio Emulator for Android»

Рисунок 15 - XDE оболочка эмулятора Visual Studio Emulator for Android

Отладка на устройстве не вызывает каких-либо дополнительных затруднений. В Android разрешается установка приложений из любых источников после включения соответствующей функции в параметрах смартфона.

3.2.3. Отладка средствами GDB

Отладка под iOS сопряжена с рядом дополнительных трудностей, вызванных лицензионными ограничениями со стороны Apple. Компилирование, выполнение и отладка приложений для iOS может выполняться только на оригинальном оборудовании от Apple. Таким образом разработчику недостаточно иметь компьютер под управлением Windows с установленной средой разработки Visual Studio. Компания Xamarin предоставляет оригинальное решение, основывающееся на выстраивании «тандема» из Windows-компьютера и Mac OS-компьютера. В качестве компьютера с Mac OS может выступать любой компьютер компании Apple с ОС OS X версии 10.10 и выше. Компьютеры соединяются через локальную сеть. Процесс отладки выглядит следующим образом:

разработчик запускает код на исполнение на компьютере с Visual Studio. Visual Studio направляет файлы с исходными кодами в компилятор Xamarin, который в свою очередь транслирует их в файлы исходного кода Objective-C и передаёт по локальной сети на компьютер с Mac OS. На компьютере с Mac OS с помощью компилятора из набора среды разработки Xcode [92] выполняется сборка программы для iOS. Затем эта программа развёртывается либо на эмулятор, либо на подключенное к компьютеру мобильное устройство. Далее процесс отладки почти не отличается от отладки на Android с помощью ADB. Разница заключается лишь в службе трассировки, в качестве которой выступает GDB.

Рисунок 16 - Интегрированная среда разработки Xcode

Эмуляторы в составе Xcode обеспечивают наибольшее сходство с реальным оборудованием из всех вышеперечисленных. При появлении нового устройства с обновлённой ОС, компания Apple выпускает обновление для предыдущего поколения смартфонов, благодаря чему актуальная версия ОС имеет один и тот же номер на разных устройствах. Поэтому в эмуляторы предоставляются только для самой свежей версии ОС. Пользователю даётся возможность выбрать размеры экрана эмулируемого устройства.

Рисунок 17 - Интерфейс эмулятора iOS

Установка приложений на устройства с iOS допускается только из службы «App Store» [93]. Для отладки, как и в случае с Windows Phone, требуется предварительная разблокировка устройства. Для этого разработчику требуется лицензия от Apple. Разблокировка осуществляется через соответствующее окно среды разработки Xcode.

3.3. Тестирование web-сервиса

Для тестирования web-сервиса были выбраны методики, которые дают возможность протестировать серверную часть, не затрагивая при этом клиентскую программу.

3.3.1. Модульное тестирование

Для проверки корректности бизнес-логики был использован подход известный как «модульное тестирование». Учитывая тот факт, что при разработке ПО использовался объектно-ориентированный подход, вся бизнес логика программы состоит из отдельных объектов, каждый из которых выполняет свою роль. Зная устройство каждого из объектов

можно составить тест, полностью покрывающий логику одного объекта, не затрагивая при этом другие.

По концепции MVC [94] каждый модуль называется контроллером, работающим с данными конкретного типа – моделью. Всего в программе 5 контроллеров:

- контроллер производителей автомобилей;
- контроллер моделей автомобилей;
- контроллер вариантов исполнения автомобилей;
- контроллер категорий ремонтных работ;
- контроллер ремонтных работ.

Каждый из этих контроллеров является CRUD-модулем [95] т.е. отвечает за операции создания, получения и изменения и удаления данных. Поэтому минимальное количество тестов для каждого из таких модулей равно четырем. Далее из-за возможных различий в путях выполнения программы количество тестов может увеличиваться:

- контроллер производителей автомобилей:
  - добавление производителя;
  - получение полного списка производителей;
  - получение производителя по идентификатору;
  - изменение данных производителя;
  - удаление производителя по идентификатору;
- контроллер моделей автомобилей:
  - добавление модели;
  - получение полного списка моделей определённого производителя;
  - получение модели по ее идентификатору;
  - изменение данных модели;
  - удаление модели по идентификатору;
- контроллер вариантов исполнения автомобилей:
  - добавление варианта исполнения;
  - получение полного списка вариантов исполнения по идентификатору модели;
  - получение варианта исполнения по его идентификатору;
  - изменение данных варианта исполнения;
  - удаление варианта исполнения по идентификатору;

- контроллер категорий ремонтных работ:
  - добавление категории работ;
  - получение категории по идентификатору;
  - изменение данных категории;
  - удаление категории по идентификатору;
- контроллер ремонтных работ:
  - добавление работы;
  - получение списка работ по идентификатору варианта исполнения автомобиля и идентификатору категории;
  - изменение данных ремонтных работ;
  - удаление ремонтных работ по идентификатору.

В ходе проведения данной методики тестирования не было обнаружено никаких ошибок. Это свидетельствует о том, что все модули были качественно спроектированы, а все ошибки кодирования были исправлены на этапе отладки.

3.3.2. Нагрузочное тестирование

Для выявления соответствия системных требований программы требованиям к системе проводилось нагрузочное тестирование. В качестве тестового сценария был использован тест-кейс из функционального теста. С помощью инструментов нагрузочного тестирования Visual Studio имитировалось одновременное использование системы 25-ю пользователями, что составило примерно 170 запросов в секунду. В результате тестирования была обнаружена примечательная особенность платформы ASP.NET, названная авторами «прогрев». Заключается она в следующем: при «простое системы» (отсутствии запросов от пользователей в течение десяти минут) web-сервер удаляет кэш сценариев, высвобождая тем самым оперативную память сервера. При первом запросе к ASP сценарию, платформа первым делом проверяет, имеется ли кэшированная версия сценария в памяти компьютера. Если таковая имеется, то она выполняется и результат возвращается пользователю. Если кэш отсутствует, то сценарий преобразуется из байт-кода в машинный и помещается в кэш, что занимает несколько более длительное время. При нагрузочном тестировании, пока ни один сценарий ещё не выполнялся, первые запросы выполняются дольше, однако, как только запросы начинают повторяться, время отклика снижается. Период кэширования как

раз и попадает на «прогрев». В установившемся режиме время отклика на системе соответствующей рекомендуемым требованиям в техническом задании составляет не более 0,08 с, что является удовлетворительным результатом.

Таблица 12 - Результаты нагрузочного тестирования web-сервиса

Количество пользователей	Количество запросов в секунду	Среднее время отклика системы (с)
25	168	0.08
50	331	0.12
100	610	0.14
200	1138	0.19

Рисунок 18 - Результаты нагрузочного тестирования web-сервиса

3.4. Тестирование мобильного приложения

Для тестирования мобильного приложения использовались методики, схожие с теми, что применялись для тестирования web-сервиса. Разницу составляла иная среда исполнения. В данном случае тесты необходимо было проводить на мобильном оборудовании для максимальной приближенности к реальной ситуации.

3.4.1. Модульное тестирование

Отличительной особенностью Visual Studio 2015 также является наличие нового инструмента составления тестов «IntelliTest» [96]. Раньше при составлении модульного теста разработчиком приходилось самостоятельно описывать наборы входных и выходных данных для покрытия всей логики модуля. С появлением «IntelliTest» интегрированная среда разработки Visual Studio 2015 способна составить модульные тесты автоматически. «IntelliTest» сканирует исходный код модуля в поисках выражений и логических разветвлений. Затем генерируются такие данные, чтобы вызвать все возможные исключения в каждом выражении. При изменении исходных кодов разработчиком, «IntelliTest» снова сканирует их для поддержания тестов в актуальном состоянии.

Модульное тестирование мобильных приложений необходимо выполнять в среде мобильного устройства (на реальном смартфоне или в эмуляторе). Это связано с тем, что объекты бизнес логики могут использовать различные API, доступные исключительно в мобильной ОС (например, геопозиционирование, телефонные вызовы, и т.д.). Стандартные инструменты модульного тестирования не всегда поддерживают такую возможность. Рассмотрим методики модульного тестирования по платформам:

Windows Phone

Для проведения модульного тестирования на Windows Phone применяется обновлённая версия инструмента «MSTest» [97], которая, начиная с Visual Studio 2015 поддерживает тестирование на мобильных устройствах. Тест составляется разработчиком на компьютере. Затем тестовые сценарии добавляются к установочному пакету приложения и развертываются на устройстве (или эмуляторе). Затем производится серия тестов, а результаты отправляются в интегрированную среду разработки на компьютере разработчика.

Android и iOS

Поскольку разработка приложений для Android и iOS обеспечивается Xamarin, то инструменты для модульного тестирования программ на C# не входят в состав официальных

пакетов разработчика (SDK) для этих платформ. Xamarin предоставляет модифицированную версию пакета «NUnit» [98] - «NUnitLite» [99]. Основной особенностью данной версии является то, что тесты запускаются и проводятся непосредственно на мобильном устройстве. Разработчик добавляет объекты бизнес-логики в специальный проект на компьютере и снабжает их тестами. Затем на базе проекта формируется установочный пакет, который развертывается на мобильном устройстве и запускается. После этого разработчик берет в руки смартфон с установленными тестами, вручную запускает их и анализирует результаты на экране устройства. Возврат результатов на компьютер не предусмотрен.

Рисунок 19 - Результаты тестов на экране мобильного устройства

Рисунок 20 - Главное меню «NUnitLite»

3.4.2. Функциональное тестирование

Для определения соответствия реализованных в приложении функций требованиям технического задания проводилось функциональное тестирование. Для повышения эффективности тестирования использовались автоматизированные тесты пользовательского интерфейса (UI Tests). В связи с различиями тестируемых платформ, использовались разные инструменты.

В качестве тест-кейса применялся следующий сценарий:

Требуется произвести замену левой блок-фары на автомобиле ВАЗ 21099i 2003 года выпуска. После проведения работ необходимо удостоверится в работоспособности работы освещения. Ожидаемая стоимость работ: 700 руб.

Windows Phone

Для проведения автоматизированных тестов пользовательских интерфейсов на платформе Windows Phone предусмотрен инструмент «Coded UI Tests» [100]. Инструмент доступен разработчику в виде специального шаблона проекта Visual Studio, добавляемого в

проект с уже существующим приложением. Инструмент позволяет в интерактивном режиме указать тестируемые элементы пользовательского интерфейса, указать тестирующее воздействие на них и ожидаемые результаты. Для выполнения вышеуказанного сценария был создан тест, который дожидается появления главного меню приложения, выбирает в раскрывающемся списке «Производитель» марку «ВАЗ» и дожидается загрузки перечня моделей. После этого осуществляется выбор модели «21099i» в раскрывающемся списке «Модель». По окончании загрузки перечня вариантов исполнения, производится выбор варианта «2003». Затем, проверяется, стала ли активна кнопка «Далее» на экране. Если кнопка доступна, то осуществляется переход к другому экрану приложения, нажатием на неё. При появлении нового экрана, тестирующее приложение выбирает категорию ремонта «Электрооборудование» и подкатегорию «Фары». После этого, из перечня доступных работ выбирается «Блок фара левая - с/у (при снятой аккумуляторной батарее)» в количестве 1 шт. Аналогичным образом выбираются работы «Батарея аккумуляторная - с/у» (категория «Электрооборудование», подкатегория «Батарея аккумуляторная») и «Электрооборудование - проверка работы освещения, световой сигнализации и контрольных приборов (на автомобиле)» (категория «Техническое обслуживание», подкатегория «Контрольно-диагностические работы»). После этого анализируется значение поля «Итого». Если значение отлично от «700», либо на любом этапе теста произошла ошибка, тест считается проваленным.

Данный тест компилируется и собирается в отдельный пакет и устанавливается на мобильное устройство (или эмулятор) вместе с основным приложением. Далее Visual Studio осуществляет запуск тестов и сбор результатов. При этом на экране смартфона можно наблюдать этапы прохождения теста.

Android

На платформе Android функциональное тестирование проводится с помощью инструмента «Espresso» [101], поставляемого в комплекте с «Android Testing Support Library». Тест описывается в виде программного класса на языке Java [102], который использует библиотеку «android.support.test.InstrumentationRegistry». Основная концепция описания тестового класса соответствует принципам «JUnit» [103]. Внутри класса создаются методы с характерными именами, отвечающие за подготовку к тесту, выполнение теста и

возврат системы в исходное состояние. Анализ результата происходит в конце тестирующего метода – до возврата системы в исходное состояние. К недостаткам «Espresso» можно отнести отсутствие возможности интерактивного выбора тестируемых компонентов пользовательского интерфейса. Для определения компонентов используется технология «Hamcrest matchers» [104]. Эта технология позволяет описать тестируемый элемент пользовательского интерфейса по одному из его свойств, например, текстовому содержимому, или идентификатору внутри макета. Ещё одним недостатком является плохая интеграция пакета «Espresso» с графической подсистемой Android. Для успешного прохождения теста, необходимо отключить все улучшения интерфейса, такие как анимации окон, анимации переходов внутри приложения и анимации масштабирования. Анимации создают дополнительные задержки, которые расцениваются тестирующим приложением как отсутствие ожидаемого значения в тестируемых элементах. Готовый тест запускают с помощью специальной командной ADB (Android Debug Bridge). Таким образом тестирующий код выполняется на компьютере разработчика. На нем же осуществляется сбор результатов.

iOS

Для тестирования пользовательских интерфейсов приложений на C# на мобильных устройствах с операционной системой iOS применяется «Xamarin.UITest» [105]. Это решение позволяет создавать тесты с применением ранее описанной технологии «NUnit». Тест-кейсы описываются в виде модульных тестов с доступом к элементам пользовательского интерфейса. Для обращения к элементам используется подход аналогичный с Android: указываются свойства тестируемых объектов. Стоит учесть, что такой метод описания исключает тестирование корректности макета приложения. Например, кнопка, ошибочно расположенная за границами экрана не доступна пользователю, однако автоматический тест без труда сможет воспользоваться ей, вызвав её по внутреннему идентификатору. Особо стоит отметить, что для возможности проведения UI тестов, в целевое приложение должна быть внедрена специальная библиотека «Xamarin Test Cloud Agent» [106], обеспечивающая связь тестирующей среды на компьютере разработчика с интерфейсом приложения. Для составления тестов разработчиками Xamarin рекомендуется REPL (read-eval-print-loop) [107] подход. Суть его состоит в том, что при

составлении теста разработчик взаимодействует с интерфейсом программы не через графическую оболочку, а с точки зрения программы. Запуск теста и оценка результатов происходит на компьютере разработчика.

Выводы по технологическому разделу

Отладка web-сервиса проводилась средствами интегрированной среды разработки Visual Studio и web-отладчика Fiddler. Отладка мобильного приложения проводилась средствами из комплектов разработчика, под каждую мобильную платформу соответственно.

Тестирование web-сервиса включало в себя модульное и нагрузочное тестирование. Мобильное приложение было подвержено модульному и функциональному тестированию. По результатам, в программу были внесены необходимые исправления.

Заключение

В результате проведенной работы были изучены аналогичные решения от других фирм, актуальные языки программирования и среды разработки. В результате исследования возможностей программных средств других компаний, было принято решение о разработке собственного ПО, удовлетворяющего всем требованиям. Разработанное ПО позволяет потенциальным клиентам автосервиса ООО «ДИМАВТО» бесплатно загрузить мобильное приложение из Интернета и произвести необходимые расчеты самостоятельно, без помощи специалиста по автомобильному ремонту. Если помощь специалиста все-же потребуется, то в программе предусмотрена возможность вызова по телефону.

Разработанное ПО состоит из двух кроссплатформенных модулей: серверного и клиентского. Язык программирования C# был выбран в качестве основного, поскольку на нем было оптимально описывать как серверную, так и клиентскую часть приложения, минимизировав при этом возможные нестыковки. Для передачи данных между частями приложения используется формат JSON. Основным паттерном проектирования как в сервере, так и в клиенте является MVC.

Серверный модуль написан на языке C# с применением кроссплатформенной технологии ASP.NET Core, что позволяет подобрать оптимальный тарифный план хостинга, не взирая на ОС сервера. Клиентский модуль также написан на языке C#, но с применением технологии Xamarin, что дает возможность использовать данный модуль на современных мобильных платформах Windows, Android и iOS, тем самым максимально увеличив количество потенциальных клиентов.

Для работы с базами данных использовалось программное средство Entity Framework, благодаря чему удалось частично автоматизировать этап даталогического проектирования базы данных.

По итогам разработки было проведено модульное, функциональное и нагрузочное тестирование, по результатам которых, в программу были внесены необходимые исправления.

Список литературы

1. Гагарина Л.Г., Касимов Р.А., Федотова Е.Л., Черников Б.В., Коваленко Д.Г., Чжо З.Е. Методические указания по подготовке выпускной квалификационной работы по направлению подготовки бакалавров 09.03.04 «Программная инженерия». М.: МИЭТ, 2016.
2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.
3. ГОСТ 19.201-78. Техническое задание, требования к содержанию и оформлению.
4. ГОСТ 19.505-79. Руководство оператора. Требования к содержанию и оформлению.
5. Audatex Australia | a Solera company [Электронный ресурс] URL: http:// www.audatex.com.au
6. Audatex [Электронный ресурс] URL: http://www.audatex.ru
7. Автонормы Online [Электронный ресурс] URL: http://autonorms.smile-group.ru
8. Ремонт авто своими руками: статьи, инструкции, видео, отчёты по ремонту автомобилей своими руками [Электронный ресурс] URL: https://www.atlib.ru
9. A Brief Description - C++ Information [Электронный ресурс] URL: http:// www.cplusplus.com/info/description
10. gSOAP: SOAP/XML Web Services and XML Data Bindings for C and C++ [Электронный ресурс] URL: http://www.cs.fsu.edu/~engelen/soap.html]
11. WebServices - Axis [Электронный ресурс] URL: http://axis.apache.org/axis/]
12. pion 0.1 : Python Package Index [Электронный ресурс] URL: https:// pypi.python.org/pypi/pion]
13. WSO2 Web Services Framework for C++ | WSO2 Inc [Электронный ресурс] URL: http://wso2.com/products/web-services-framework/cpp/

14. Microsoft Open Database Connectivity (ODBC) [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/library/ms710252(v=vs.85).aspx
15. About Objective-C [Электронный ресурс] URL: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html
16. Google Code Archive - Long-term storage for Google Code Project Hosting. [Электронный ресурс] URL: https://code.google.com/archive/p/entropydb/
17. SQLite Home Page [Электронный ресурс] URL: https://sqlite.org/
18. Oracle Technology Network for Java Developers | Oracle Technology Network | Oracle [Электронный ресурс] URL: http://www.oracle.com/technetwork/java/index.html
19. Java Web services: Introducing Metro [Электронный ресурс] URL: http://www.ibm.com/developerworks/library/j-jws9/index.html
20. GlassFish Server [Электронный ресурс] URL: https://glassfish.java.net/
21. Java SE Technologies - Database [Электронный ресурс] URL: http://www.oracle.com/technetwork/java/javase/jdbc/index.html
22. Java DB [Электронный ресурс] URL: http://www.oracle.com/technetwork/java/javadb/overview/index.html
23. PHP: Preface - Manual [Электронный ресурс] URL: http://php.net/manual/en/preface.php
24. PHP: PDO - Manual [Электронный ресурс] URL: http://php.net/manual/ru/book.pdo.php
25. Welcome to Python.org [Электронный ресурс] URL: https://www.python.org/
26. FrontPage - py2exe.org [Электронный ресурс] URL: http://py2exe.org/
27. Welcome to PyInstaller official website [Электронный ресурс] URL: http://www.pyinstaller.org/
28. Welcome | Flask (A Python Microframework) [Электронный ресурс] URL: http://flask.pocoo.org/
29. Python Web Services [Электронный ресурс] URL: http://pywebsvcs.sourceforge.net/

30. webservice 0.3: Python Package Index [Электронный ресурс] URL: https://pypi.python.org/pypi/webservice/0.3
31. buzhug, a pure-Python database engine [Электронный ресурс] URL: http://buzhug.sourceforge.net/
32. SnakeSQL -- Pure Python SQL database supporting NULLs and Joins [Электронный ресурс] URL: http://pythonweb.org/projects/snakesql/
33. MySQL : Download Connector/Python [Электронный ресурс] URL: http://dev.mysql.com/downloads/connector/python/
34. C# [Электронный ресурс] URL: https://msdn.microsoft.com/ru-ru/library/kx37x362.aspx
35. The Official Microsoft ASP.NET Site [Электронный ресурс] URL: http://www.asp.net/
36. ADO.NET [Электронный ресурс] URL: https://msdn.microsoft.com/ru-ru/library/e80y5yhx(v=vs.110).aspx
37. Entity Framework [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/data/ef.aspx
38. Visual Studio — Microsoft Developer Tools [Электронный ресурс] URL: https://www.visualstudio.com/ru-ru/visual-studio-homepage-vs.aspx
39. Introducing Xamarin Studio - Xamarin [Электронный ресурс] URL: http://developer.xamarin.com/guides/cross-platform/getting_started/introducing_xamarin_studio/
40. SharpDevelop @ic#code [Электронный ресурс] URL: http://www.icsharpcode.net/OpenSource/SD/
41. Geany : Home Page [Электронный ресурс] URL: http://www.geany.org/
42. ASP.NET Core - Documentation [Электронный ресурс] URL: https://docs.asp.net
43. ASP.NET [Электронный ресурс] URL: http://www.asp.net
44. How people build software - GitHub [Электронный ресурс] URL: https://github.com/
45. NuGet Gallery | Home [Электронный ресурс] URL: http://www.nuget.org/

46. NET Execution Environment (DNX) [Электронный ресурс] URL: https://docs.asp.net/en/latest/dnx/index.html
47. Common Language Runtime (CLR) [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/library/8bs2ecf4(v=vs.110).aspx
48. Model-View-Controller [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/library/ff649643.aspx
49. Code First в Entity Framework для новой базы данных [Электронный ресурс] URL: https://msdn.microsoft.com/ru-ru/data/jj193542.aspx
50. Entity Framework Loading Related Entities [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/data/jj574232.aspx
51. Entity Framework Fluent API - Configuring/Mapping Properties & Types [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/data/jj591617.aspx
52. LINQ (Language-Integrated Query) [Электронный ресурс] URL: https://msdn.microsoft.com/ru-ru/library/bb397926.aspx
53. Introduction to ASP.NET Web Programming Using the Razor Syntax (C#) | The ASP.NET Site [Электронный ресурс] URL: http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-c
54. W3C HTML [Электронный ресурс] URL: http://www.w3.org/html/
55. Code First with Entity Framework 5 using MVC4 and MVC Scaffold - CodeProject [Электронный ресурс] URL: http://www.codeproject.com/Articles/468777/Code-First-with-Entity-Framework-using-MVC-and
56. jQuery Validation Plugin | Form validation with jQuery [Электронный ресурс] URL: http://jqueryvalidation.org/
57. JSON [Электронный ресурс] URL: http://www.json.org/
58. Hypertext Transfer Protocol - HTTP/1.1 [Электронный ресурс] URL: https://www.w3.org/Protocols/rfc2616/rfc2616.html
59. An Introduction to JavaScript Object Notation (JSON) in JavaScript and.NET [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/library/bb299886.aspx

60. Extensible Markup Language (XML) 1.0 (Fifth Edition) [Электронный ресурс] URL: https://www.w3.org/TR/REC-xml/
61. ISO 8601:2004 - Data elements and interchange formats -- Information interchange -- Representation of dates and times [Электронный ресурс] URL: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=40874
62. XHTML 1.0: The Extensible HyperText Markup Language (Second Edition) [Электронный ресурс] URL: https://www.w3.org/TR/xhtml1/
63. Data Transfer Object [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/library/ff649585.aspx
64. AutoMapper [Электронный ресурс] URL: http://automapper.org/
65. Mobile App Development & App Creation Software - Xamarin [Электронный ресурс] URL: https://www.xamarin.com/
66. XAML Overview [Электронный ресурс] URL: https://msdn.microsoft.com/ru-ru/library/cc189036(v=vs.95).aspx
67. Xamarin for Visual Studio - Build native mobile apps in C# for iOS, Android, Mac and Windows - Xamarin [Электронный ресурс] URL: https://www.xamarin.com/visual-studio
68. Операции | Android Developers [Электронный ресурс] URL: http://developer.android.com/intl/ru/guide/components/activities.html
69. Creating a Fragment | Android Developers [Электронный ресурс] URL: http://developer.android.com/intl/ru/training/basics/fragments/creating.html
70. Storyboard [Электронный ресурс] URL: https://developer.apple.com/library/ios/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html
71. Adding a Segue Between Scenes in a Storyboard [Электронный ресурс] URL: https://developer.apple.com/library/ios/recipes/xcode_help-IB_storyboard/Chapters/StoryboardSegue.html
72. The Mono Runtime | Mono [Электронный ресурс] URL: http://www.mono-project.com/docs/advanced/runtime/
73. Architecture - Xamarin [Электронный ресурс] URL: https://developer.xamarin.com/guides/android/under_the_hood/architecture/

74. Xcode - What's New - Apple Developer [Электронный ресурс] URL: https://developer.apple.com/xcode/
75. Visual Studio – Microsoft [Электронный ресурс] URL: https://www.visualstudio.com/ru-ru/visual-studio-homepage-vs.aspx
76. Home: The Official Microsoft IIS Site [Электронный ресурс] URL: http://www.iis.net
77. Fiddler free web debugging proxy [Электронный ресурс] URL: http://www.telerik.com/fiddler
78. Windows and Windows Phone SDK Archive [Электронный ресурс] URL: https://developer.microsoft.com/en-us/windows/downloads/sdk-archive
79. logcat | Android Developers [Электронный ресурс] URL: https://developer.android.com/tools/help/logcat.html
80. GDB – Xamarin [Электронный ресурс] URL: https://developer.xamarin.com/guides/android/advanced_topics/gdb/
81. Обзор Hyper-V [Электронный ресурс] URL: https://technet.microsoft.com/ru-ru/library/hh831531.aspx
82. Технология Intel® Virtualization (Intel® VT) [Электронный ресурс] URL: ttp://www.intel.ru/content/www/ru/ru/virtualization/virtualization-technology/intel-virtualization-technology.html
83. Виртуализация AMD [Электронный ресурс] URL: http://www.amd.com/ru-ru/solutions/servers/virtualization
84. Windows Phone Emulator overview [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/library/hh134796(v=expression.40).aspx
85. Все о приложениях — Магазин Microsoft [Электронный ресурс] URL: https://www.microsoft.com/ru-ru/store/apps
86. Download Android Studio and SDK Tools | Android Developers [Электронный ресурс] URL: https://developer.android.com/sdk/index.html
87. Android Debug Bridge | Android Developers [Электронный ресурс] URL: http://developer.android.com/tools/help/adb.html

88. ARM - The Architecture For The Digital World [Электронный ресурс] URL: http://www.arm.com/
89. Архитектура AMD64 (EM64T) [Электронный ресурс] URL: http://www.viva64.com/ru/a/0029/
90. x86 Architecture - Windows 10 hardware dev [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/library/windows/hardware/ff561502(v=vs.85).aspx
91. Visual Studio Emulator for Android [Электронный ресурс] URL: https://www.visualstudio.com/en-us/features/msft-android-emulator-vs.aspx
92. Xcode - IDE - Apple Developer [Электронный ресурс] URL: https://developer.apple.com/xcode/ide/
93. App Store [Электронный ресурс] URL: http://appstore.com
94. Общие сведения о ASP.NET MVC [Электронный ресурс] URL: https://msdn.microsoft.com/ru-ru/library/dd381412(v=vs.108).aspx
95. James M. Managing the Database Environment. New Jersey: Prentice-Hall, Inc, 1983.
96. Создание модульных тестов для своего кода [Электронный ресурс] URL: https://msdn.microsoft.com/library/dn823749(v=vs.140).aspx
97. Maintain code health with unit testing in Visual Studio Team Services [Электронный ресурс] URL: https://msdn.microsoft.com/library/vs/alm/test/developer-testing/developer-testing
98. NUnit – Home [Электронный ресурс] URL: http://nunit.org/
99. Unit Testing – Xamarin [Электронный ресурс] URL: https://developer.xamarin.com/guides/ios/deployment,_testing,_and_metrics/touch.unit/
100. Coded UI Tests [Электронный ресурс] URL: https://msdn.microsoft.com/en-us/library/dd286726.aspx
101. Testing UI for a Single App | Android Developers [Электронный ресурс] URL: http://developer.android.com/training/testing/ui-testing/espresso-testing.html
102. Что такое Java? [Электронный ресурс] URL: http://java.com/ru/about/whatis_java.jsp
103. JUnit – About [Электронный ресурс] URL: http://junit.org/junit4/

104. Hamcrest [Электронный ресурс] URL: http://hamcrest.org/
105. Introduction to Xamarin.UITest – Xamarin [Электронный ресурс] URL: https://developer.xamarin.com/guides/testcloud/uitest/intro-to-uitest/
106. Introduction to Xamarin Test Cloud – Xamarin [Электронный ресурс] URL: https://developer.xamarin.com/guides/testcloud/introduction-to-test-cloud/
107. Read-eval-print loop | Article about read-eval-print loop by The Free Dictionary [Электронный ресурс] URL: http://encyclopedia2.thefreedictionary.com/Read-Eval-Print+Loop
108. Microsoft.NET - Home [Электронный ресурс] URL: https://www.microsoft.com/net/default.aspx

Приложение 1. Руководство оператора

Приложение 2. Текст программы