Оглавление

[РЕЗЮМЕ 3](#_Toc483523856)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ И ТЕРМИНОВ 5](#_Toc483523857)

[ВВЕДЕНИЕ 6](#_Toc483523858)

[ГЛАВА 1 8](#_Toc483523859)

[АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 8](#_Toc483523860)

[1 Понятие браузерной игры 8](#_Toc483523861)

[1.1 Социальная семейная браузерная игра 8](#_Toc483523862)

[1.3 Требования к социальной семейной браузерной игрe 9](#_Toc483523863)

[1.4 Обзор существующих решений 9](#_Toc483523864)

[1.5 Выводы по главе 1 10](#_Toc483523865)

[ГЛАВА 2 11](#_Toc483523866)

[ПРОЕКТИРОВНАИЕ ВЕБ-ПРИЛОЖЕНИЯ «Family Adventure World» 11](#_Toc483523867)

[2.1 Обоснование выбора способа обмена сообщения между клиентом и сервером (RESTFul API) 11](#_Toc483523868)

[2.2 Общая характеристика системы 11](#_Toc483523869)

[2.3 Выбор архитектурного решения 12](#_Toc483523870)

[2.4 Выбор программной платформы и средств реализации 12](#_Toc483523871)

[2.4.1 Язык программирования C# и платформа ASP.NET Web API 2 15](#_Toc483523872)

[2.4.2 Технология доступа к данным Entity Framework 15](#_Toc483523873)

[2.4.3 Библиотека разработки интерфейсов React 16](#_Toc483523874)

[2.4.4 Инструмент управления состояниями Redux 16](#_Toc483523875)

[2.5 Анализ требований к построению пользовательского интерфейса 16](#_Toc483523876)

[2.6 Концептуальная модель данных 19](#_Toc483523877)

[2.7 Диаграмма вариантов использования 21](#_Toc483523878)

[2.8 Выводы по главе 2 22](#_Toc483523879)

[ГЛАВА 3 23](#_Toc483523880)

[ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ «Family Adventure World» 23](#_Toc483523881)

[3.1 Распределение уровней реализации по технологиям 23](#_Toc483523882)

[3.2 Использование паттерна Repository и Fluent Api для реализации DAL 24](#_Toc483523883)

РЕЗЮМЕ

Деменович Артур Юрьевич

Дипломная работа − «Разработка социально ориентированной семейной игры “Family Adventure World”», 57 страниц, 14 иллюстраций, 3 приложения, 1 таблица.

**Ключевые слова −** социальная игра, социально ориентированная семейная игра, модель данных, документо-ориентированная СУБД.

**Объект исследования −** методы моделирования данных в информационных системах, механизмы построения хранилища данных в документо-ориентированной СУБД, механизмы эффективного хранения, обработки и анализа данных.

**Предмет исследования −** технологии и программные средства для проектирования и программной реализации информационных систем, технологии и средства реализации web-ориентированных приложений.

**Цель дипломной работы −** разработка социально ориентированной семейной игры, обеспечивающей взаимодействие членов семей между собой в игровой форме в виде создания, назначения и выполнения заданий.

**Методы исследования −** объектно-ориентированный анализ и проектирование программных систем, структурный анализ, методы проектирования баз данных, методы проектирования информационных систем.

**Авторская характеристика работы −** результатом работы является разработка и программная реализация web приложения обеспечивающие новый вид взаимодействия между членами семьи в игровой форме в виде создания, назначения, выполнения заданий.

**Область применения −** ????

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ И ТЕРМИНОВ

БД – база данных.

СУБД − система управления базами данных.

РСУБД – реляционная система управления базами данных.

ИС – информационная система.

ПО – программное обеспечение.

JS (JavaScript) – прототипно-ориентированный сценарный язык программирования.

API (Application Programming Interface) − набор готовых классов, процедур, функций и структур, предоставляемых библиотекой или сервисом для использования во внешних программных продуктах.

DAO (Data Access Object) **–** объект, который предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения.

SQL (Structured Query Language)– структурированный язык запросов.

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком.

BSON (Binary JavaScript Object Notation) – формат обмена данными, бинарная форма представления простых структур данных.

MVC (Model-view-controller) – схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.

ВВЕДЕНИЕ

В настоящее время индустрия игровых приложений прочно захватывает умы человечества. Зачастую мы даже становимся жертвой этой индустрии. Достаточно вспомнить одно из уникальных и нашумевших явлений под названием Pokémon GO. Все мы, в той или иной степени, находимся под влиянием гаджетов, интернета, социальных сетей, компьютерных игр. Играют все, и взрослые, и дети, и стар и млад, тратя на это невероятное количество времени. В случае, когда речь идет о детях, выбор такого способа времяпрепровождения не является осознанным, так как технологии используют наши психологические уязвимости.

Тяжело представить современного ребенка и его родителей без смартфона или планшета. Каждый увлечен своим виртуальным миром, необходимость в общении в семье отсутствует, семьи разобщаются, между детьми и родителями возникает недопонимание, конфликты. Запретить Интернет и уничтожить все гаджеты невозможно. Проблему нужно решать другими способами. Предлагается разработка игрового приложения «Семейный мир приключений», которое помимо игровой функции будет иметь и социальный, и воспитательный характер. У родителей появится возможность превратить выполнение домашних обязанностей своих детей в увлекательную игру с призами и поощрениями.

Зачастую дать задание ребенку, будь то выполнение домашнего задания или помощь по дому, бывает сложно. Даже провести время на улице с друзьями для некоторых детей становится сложным заданием. Существует огромное количество мобильных приложений, которые позволяют составлять списки заданий на каждый день, отмечать их выполнение, но для ребенка это совсем не интересно. Предлагаем решить эту задачу креативно, разработав игровое приложение «Семейный мир приключений», которое не только развлечет, но и сплотит детей и их родителей.

Основной отличительной чертой игры является то, что она предоставляет окружение где все пользователи, являются персонажами под названием «Семья». У каждого пользователя есть уровень, опыт и деньги. Главы семей могут создавать задания и назначать их другим членам семьи. За каждое выполненное задание пользователь получает опыт и деньги. В игре реализована система достижений, которая позволит предать игре дополнительные цели.

Дополнительными элементами игры являются инвентарь и список желаемого. Так как система интегрирована с различными сервисами, то пользователю предоставляется большой выбор реально существующих предметов, которые можно добавлять в список желаемого. Когда игрок обладает достаточным количеством опыта и денег, он сможет запрашивать данные предметы. Фиксированием факта получения предмета является фото-подтверждение с предметом. После этого предмет можно будет переместить в инвентарь. Интеграция приложения с различными социальными сетями позволит делится своими достижениями, вещами и списком желаемого.

Цель дипломной работы – разработка социально ориентированной семейной игры, обеспечивающей взаимодействие членов семей между собой в игровой форме в виде создания, назначения и выполнения заданий.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Выполнить анализ предметной области на предмет выявления структуры и форматов исходных данных, механизмов и форм контроля выполнения заданий. Выработать требования к функционалу разрабатываемой системы.
2. Разработать архитектуру программной системы, осуществить выбор эффективных средств ее программной реализации.
3. Выполнить программную реализацию системы, обеспечивающей:
   1. возможность создания и управления семьей;
   2. возможность создания и редактирования заданий, а также их назначения для членов семьи;
   3. возможность получения опыта и наград за выполнение заданий;

ГЛАВА 1

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1 Понятие браузерной игры

Браузерные игры — категория онлайн-игр, в которых Web-браузер выступает либо в роли операционной оболочки для игр, позволяя играть в игру без установки на компьютере дополнительного ПО, либо служит контейнером для дополнительной виртуальной машины, которая непосредственно выполняет код игры (Java, Flash, Shockwave и аналогичные).

Игры данного типа чаще всего являются казуальными, что связано с ограничениями на размер. Кроме этого, браузерные игры пользуются популярностью у разработчиков азартных коммерческих игр, в частности интернет-казино, что обусловлено отсутствием процесса инсталляции игры на компьютер.

В большинстве случаев браузерные игры предоставляют игроку возможность создавать виртуального персонажа, а также возможно развивать его выполняя задания. Наградой таких игр является виртуальные внутреигровые вещи которые никак не влияют на игроков в реальной жизни.

1.1 Социальная семейная браузерная игра

На данный момент понятие социальной семейной браузерной игры отсутствует. Поэтому нужно выделить основные принципы, которые будут определять рамки данного понятия.

Социальная семейная браузерная игра – это игра, работающая в браузере и не требующая установки на компьютер, каждый участник игры имеет виртуального персонажа, который является виртуальной копией реального человека. Взаимодействие между виртуальными членами семьи происходит в виде создания/выполнения заданий за который игрок получает награду в виде опыта и монет.

Данный тип игры позволит по-новому взаимодействовать с членами семьи, в частности с детьми, которые в современном мире привыкли проводить большую часть времени в гаджетах. А выполнение поручений от родителей для таких детей звучит как наказание.

1.3 Требования к социальной семейной браузерной игрe

Основной задачей разрабатываемой системы видится задача превратить воспитательный процесс из поручений, которые воспринимаются детьми как наказания в интересные задания, которые похожи на те задания, которые выполняет ребенок в обычных играх.

Для этого необходимо предусмотреть возможность создавать семью в игре, создавать задания для семьи, а также систему получение наград и достижений.

Таким образом, были выделены следующие роли пользователей системы:

* Dad
* Mom
* Son
* Daughter
* Administrator

Dad, Mom имеют возможность создания семьи и создания заданий. Son, Daughter имеют возможность выполнять задания. Administrator имеет возможность создавать достижения в системе а так же управлять пользователями.

1.4 Обзор существующих решений

В сети на сегодняшний день отсутствует что-либо похожее или с близкой по идее концепцией. Ниже будут рассмотрены отдаленные примеры развивающих игр который помогают только отдалится от родителей.

Игра Фиксики Мастера — задания, которые вашему ребёнку никогда не надоест.

Вместе с любимыми героями ваш малыш научится быть:

* наблюдательным, чтобы найти неисправные предметы
* любознательным, чтобы понять, как устроены разные вещи
* изобретательным, чтобы починить выбранный предмет
* находчивым, чтобы решать новые.

Плюсы данной игры заключаются в том, что ваш ребенок познает как устроен мир и развивается. Но в этой игре, не хватает социальной составляющей общение с родителями полностью отсутствует, ребенок будет проводить за планшетом или телефоном очень много времени.

Все последующие игры имеют схожую направленность и одинаковые недостатки, которые были описаны выше.

1. Ми-ми-мишки - https://play.google.com/store/apps/details?id=com.imult.bebebearsfree2play&hl=ru
2. Лунтик: Детские игры -https://play.google.com/store/apps/details?id=com.PSVStudio.LuntikMinigames&hl=ru
3. Бумажки -https://play.google.com/store/apps/details?id=com.imult.papermates&hl=ru

1.5 Выводы по главе 1

ГЛАВА 2

ПРОЕКТИРОВНАИЕ ВЕБ-ПРИЛОЖЕНИЯ «Family Adventure World»

2.1 Обоснование выбора способа обмена сообщения между клиентом и сервером (RESTFul API)

Главным критерием выбора архитектуры RESTFul API был её основной принцип - «Независимость от состояния (Statelessness)». Проще говоря, RESTful сервер не должен отслеживать, хранить и тем более использовать в работе текущую контекстную информацию о клиенте. С другой стороны, клиент должен взять эту задачу на себя.

Как это улучшит процесс разработки приложения? При реализации данного подхода сервер не будет зависеть от конкретного клиента и от данных, которые он присылает. Это необходимо для того, чтобы пользователь мог не только сам участвовать в игровом процессе, но и при желании, смог разработать собственное клиентское приложение, которое сможет пользоваться данными сервера.

2.2 Общая характеристика системы

В соответствии с требованиями, предъявляемыми к разрабатываемой системе, и с учетом необходимого функционала были выделены следующие компоненты разрабатываемой системы:

* персонализированная контрольная панель текущего состояния семьи на основе доступной информации;
* члены семьи;
* персонализированная контрольная панель текущего состояния заданий пользователя/семьи.
* персонализированная контрольная панель текущего состояния члена семьи;

Основными функциональными возможностями разрабатываемой системы являются:

* управление членами семьи;
* управление заданиями;
* управление достижениями;

Под управлением понимаются такие функциональные возможности, как функции создания, редактирования и удаления данных.

В системе необходимо реализовать механизм разграничения прав доступа к информации в соответствии с выделенными ролями: Dad, Mom, Son, Daughter, Administrator.

2.3 Выбор архитектурного решения

В качестве основы для построения программной системы была выбрана трехуровневая архитектура:

**Уровень представления данных.** Уровень приложения с интерфейсом пользователя, главная задача которого представление результатов, понятных пользователю. Для него организован набор функциональных возможностей, которые будут доступны пользователю в момент авторизации.

**Уровень бизнес-логики.** Уровень, координирующий работу системы, который обрабатывает команды, выполняет логические решения и вычисления, а также расчеты. На данном уровне располагаются основные функциональные компоненты и модули системы. Каждый модуль отвечает за выполнение соответствующей задачи, либо набора задач.

**Уровень хранения данных.** Уровень, обеспечивающий хранение и извлечение информации из базы данных и файловой системы.

2.4 Выбор программной платформы и средств реализации

Правильный выбор программной платформы является важнейшей задачей при построении любой компьютерной системы и разработке методики ее использования. Неправильный выбор приводит не только к удорожанию проекта, но и усложняет поддержку жизненного цикла целевой системы. От выбора платформы так же зависит время отладки кодов программ, возможность использования готовых кодов, качество сервисной поддержки, объем документирования проекта и т.д. Поэтому вопрос выбора платформы для разработки требует отдельного рассмотрения.

Современные сложные автоматизированные системы и проекты, обеспечивающие их создание, характеризуются, как правило, следующими особенностями:

* наличие совокупности тесно взаимодействующих компонентов – подсистем, имеющих свои локальные задачи и цели функционирования;
* иерархическую структуру взаимосвязей компонентов, обеспечивающую устойчивость функционирования системы;
* необходимость достаточно длительного сосуществования старых приложений и вновь разрабатываемых БД и приложений;
* наличие потребности как в традиционных приложениях, связанных с обработкой транзакций и решением регламентных задач, так и в приложениях аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема;
* функционирование в неоднородной операционной среде на нескольких вычислительных платформах;
* существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению информационных систем.

Учитывая особенности современных информационных систем, выделяют следующие основные критерии выбора средств программирования (СП).

1. Обеспечение целостности проекта и контроль за его состоянием. Данное требование означает наличие единой технологической среды создания, сопровождения и развития ИС, а также целостность базы проектных данных.
2. Независимость от программно-аппаратной платформы и СУБД. Требование определяется неоднородностью среды функционирования ИС.
3. Поддержка одновременной работы групп разработчиков. Развитые СП должны обладать возможностями разделения полномочий персонала разработчиков и объединения отдельных работ в общий проект. Должна обеспечиваться одновременная работа проектировщиков БД и разработчиков приложений (разработчики приложений в такой ситуации могут начинать работу с базой данных, не дожидаясь полного завершения ее проектирования CASE-средствами).
4. Возможность разработки приложений "клиент-сервер" требуемой конфигурации. Подразумевается сочетание наличия развитой графической среды разработки приложений (многооконность, разнообразие стандартных графических объектов, разнообразие используемых шрифтов и т.д.) с возможностью декомпозиции приложения на "клиентскую" часть, реализующую пользовательский экранный интерфейс и "серверную" часть.
5. Простота использования. Здесь учитывается доступность пользовательского интерфейса, время, необходимое для обучения.

Согласно предложенной схеме функционирования системы и руководствуясь предъявляемыми критериями выбора СП, были выбраны следующие средства реализации:

1. Язык программирования C#
2. Язык программирования JavaScript
3. Платформа ASP.NET Web API 2
4. Хранилище данных MSSQL
5. Библиотека разработки интерфейсов React
6. Инструмент управления состояниями Redux
7. Технология доступа к данным Entity Framework
8. Динамический язык стилей Scss

2.4.1 Язык программирования C# и платформа ASP.NET Web API 2

На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей. C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java.

ASP.NET MVC представляет собой платформу для создания сайтов и веб-приложений с использованием паттерна (или шаблона) MVC (model - view - controller).

Web API представляет иной способ построения приложения ASP.NET несколько отличный от ASP.NET MVC. Web API представляет собой веб-службу, которая может взаимодействовать с различными приложениями. При этом приложение может быть веб-приложением ASP.NET, либо может быть мобильным или обычным десктопным приложением.

Также надо отметить, что платформа Web API 2 не является частью фреймворка ASP.NET MVC и может быть задействована как в связке с MVC, так и в соединении с Web Forms. Поэтому в Web API имеется своя система версий. Так, первая версия появилась с .net 4.5. А вместе с .NET 4.5.1 и MVC 5 вышла Web API 2.0.

2.4.2 **Технология доступа к данным Entity Framework**

**Entity Framework** представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работает с объектами.

Одним из преимуществ EF является удобство конфигурации моделей. Для этого EF предоставляет Fluent API.

2.4.3 Библиотека разработки интерфейсов React

React — это библиотека для разработки интерфейсов, созданная Facebook. В последний год он приобрел особенную популярность, о нем постоянно пишут, многие известные компании используют его в своих проектах.

В React используется так называемый компонентный подход. В React нет контроллеров, вьюшек, моделей, шаблонов и т.д. — все есть компонент. Компоненты можно и нужно переиспользовать, наследовать друг от друга, компоновать. Компонент — это своего рода строительная единица, из которой собирается интерфейс.

2.4.4 Инструмент управления состояниями Redux

Redux — это инструмент управления как состоянием данных, так и состоянием интерфейса в JavaScript-приложениях. Он подходит для одностраничных приложений, в которых управление состоянием может со временем становиться сложным. Redux не связан с каким-то определенным фреймворком, и хотя разрабатывался для React, может использоваться с Angular или jQuery.

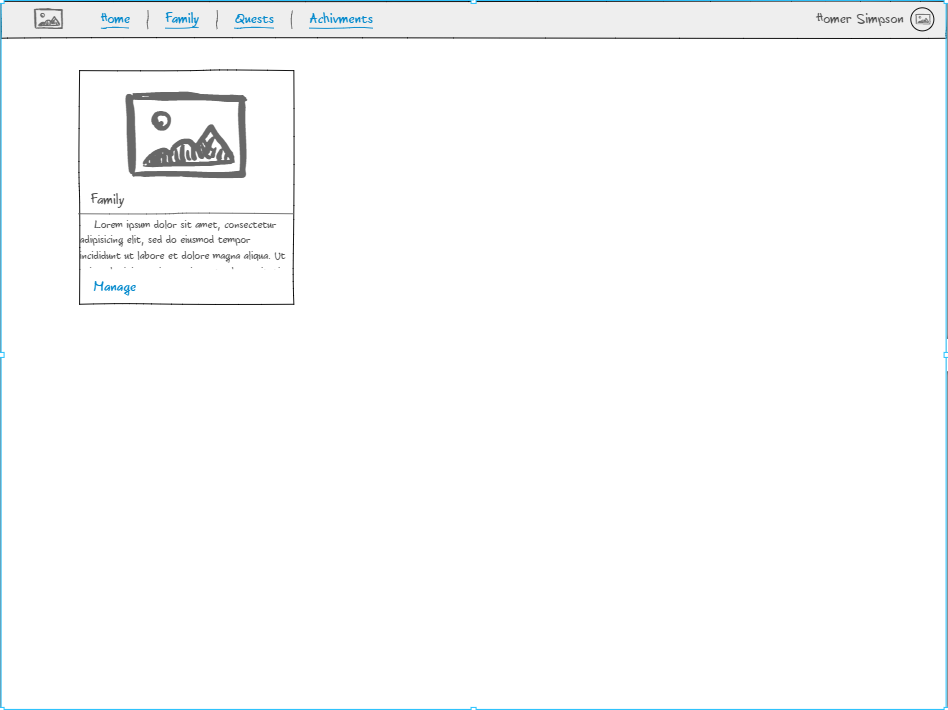
2.5 Анализ требований к построению пользовательского интерфейса

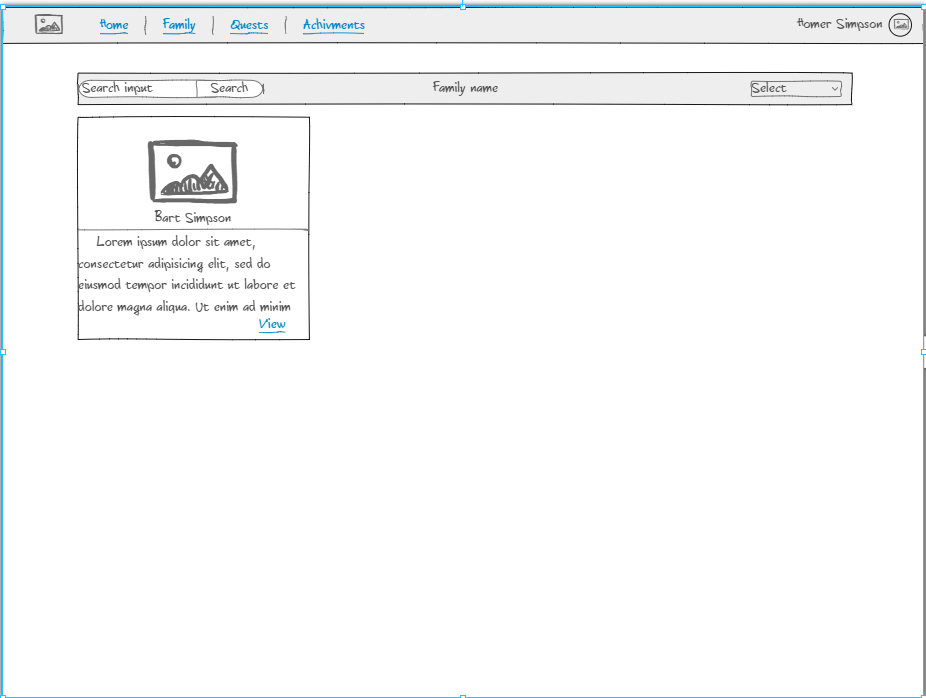
Рассуждая обо всей функциональности и о технологиях, используемых для реализации семейной социальной игры, не стоит забывать и о таком важном для любой системы, ориентированной на массового пользователя показателе, как удобство пользовательского интерфейса.

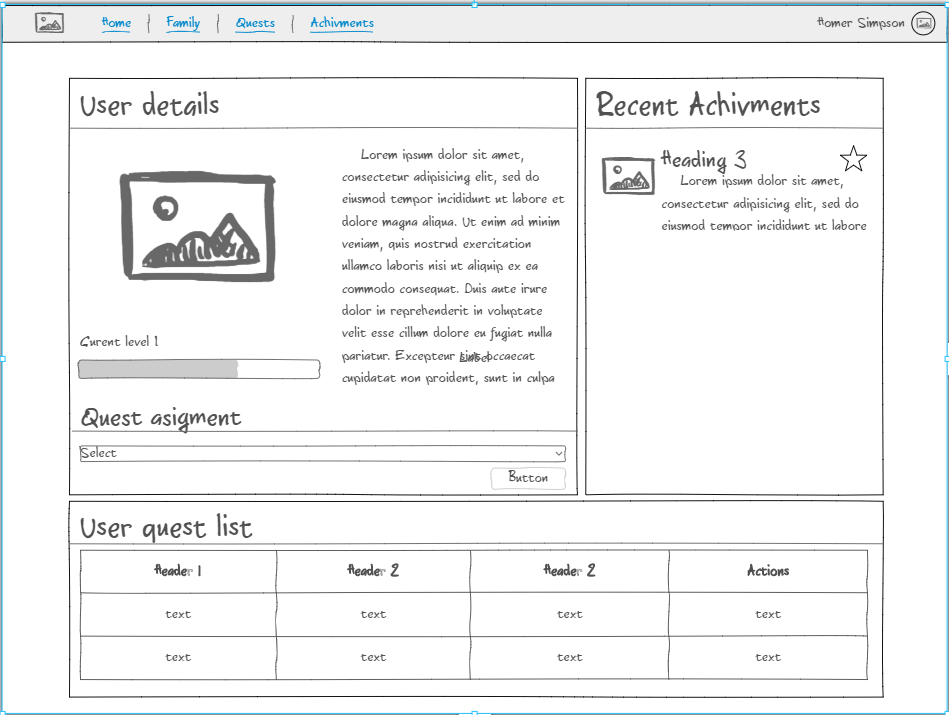
Пользовательский интерфейс зачастую играет едва ли не самую значимую роль в том, будет пользователь использовать систему, или предпочтет менее функциональный, но более понятный, красивый и дружественный интерфейс другого приложения.

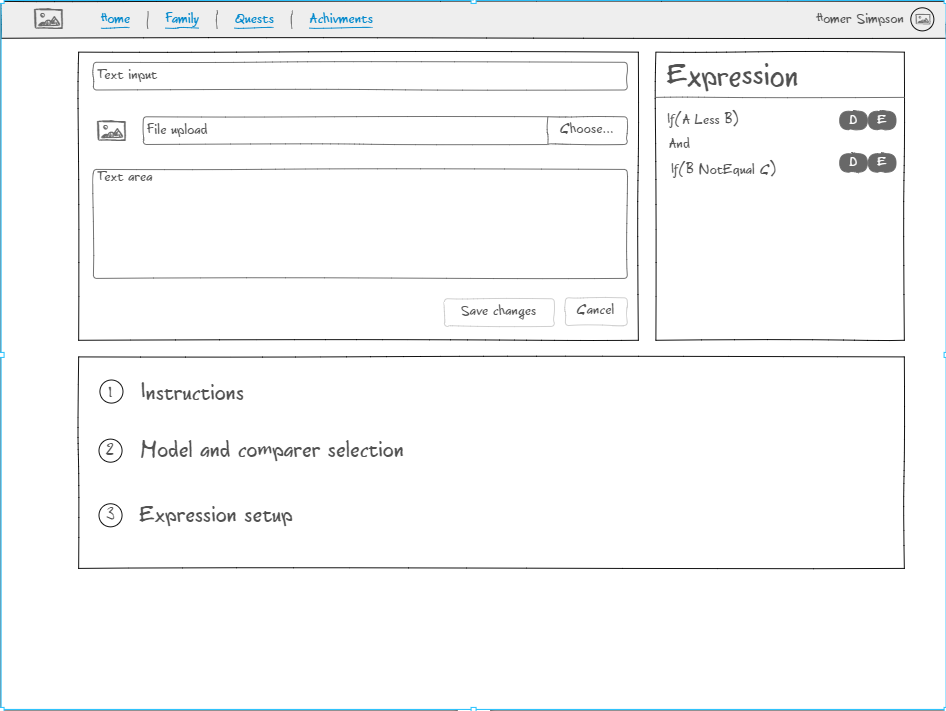
На данном этапе проектирования приложения ставится задача выяснить, каким образом должен быть построен интерфейс, какие моменты могут показаться пользователю неудобными и непонятными, а какие наоборот помогут лучше сориентироваться. В этом случае в первую очередь стоит обратиться к примерам приложений, которые уже давно зарекомендовали себя в повседневном использовании и не вызывают особых трудностей с обращением.

Зачастую веб-приложения оказываются не очень удобными для использования из-за перегруженности их дизайна: пользователь начинает теряться в большом количестве кнопок, ссылок, открытых страниц и как итог весь процесс использования приложения сводится к поиску необходимой информации среди огромного ее потока. Необходимо было создать приложение с интерфейсом, организованным таким образом, чтобы пользователю всегда было понятно, на какую кнопку нужно нажать, как попасть на определенную страницу или как выполнить определенное действие. Учитывая все нюансы и пожелания конечных пользователей, был разработан интерфейс, часть которого представлена в приложении А. Схема интерфейса представлена на рисунках …









2.6 Концептуальная модель данных

Анализ предметной области позволил выделить следующие базовые сущности модели данных:

* User
* PlayerInfo
* Achivment
* ExpressionProperty
* PropertyValue
* Quest
* Family
* Item
* Experience
* UserType
* Claim

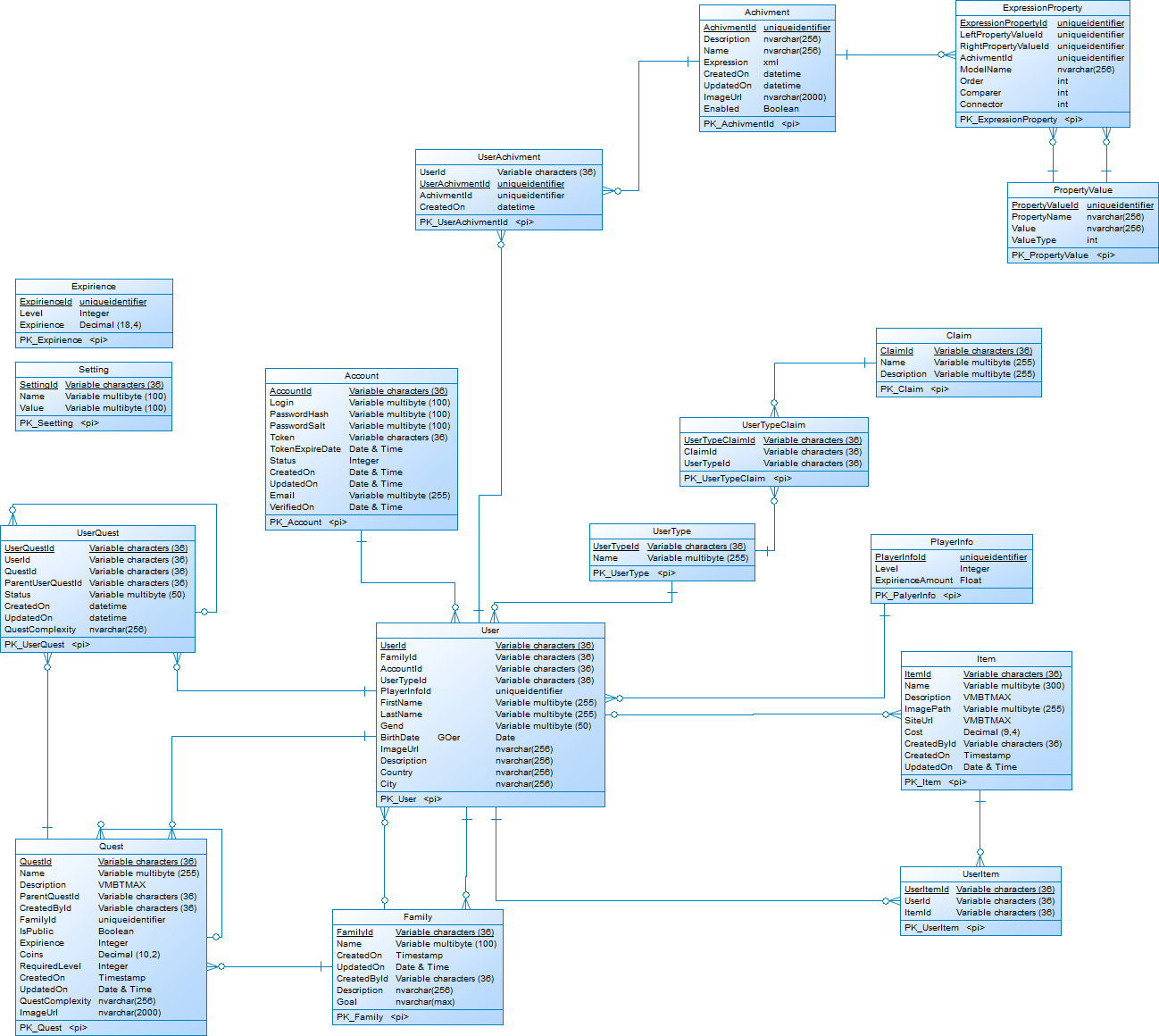
Каждый такой объект характеризуется своим набором атрибутов.

**User** является центральной таблицей в системе, в этой сущности хранится личная информация о пользователе, в свою очередь вся техническая информация такая как Login/Password и т.д. хранится в таблице **Account**. Информация о пользователе как игроке хранится отдельно в таблице **PlayerInfo**.

В системе присутствуют права, информация о правах хранится в таблице Claim. Права в системе привязываются к типу пользователя. Каждый пользователь может иметь один тип.

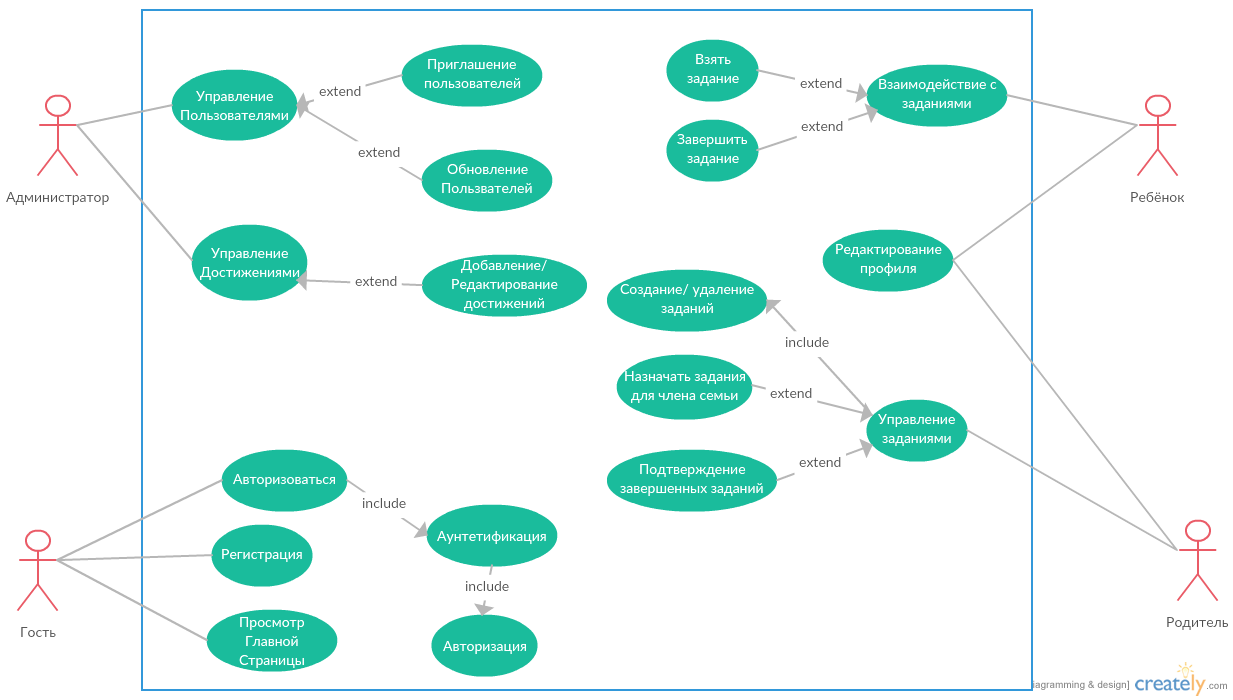
Таблица **Expirience** предназначена для хранения списка доступных уровней и значений необходимых для определения требований достижения уровня.

Таблица **Achivment** хранит информацию о достижениях, она связана с таблицей **ExpressionProperty** которая в свою очередь связана с **PropertyValue**. Последние хранят информация необходимую для восстановления условия достижения.



2.7 Диаграмма вариантов использования

На этапе проектирования приложения была построена UML диаграмма вариантов использования игры. Диаграммы вариантов использования отображает то, что система должна делать, не указывая сами применяемые методы.

На следующем рисунке … изображена полученная диаграмма.

Как видно из рисунка, в качестве актеров выступают Ребёнок, Родитель, Администратор и Гость.

Для каждого из актеров доступны свои варианты использования:

1. Гость
   1. Авторизация
   2. Регистрация
   3. Просмотр главной страницы
2. Администратор
   1. Управление достижениями. Это включает в себя создание и редактирование.
   2. Управление пользователями
3. Родитель
   1. Управление заданиями. Включает в себя создание и редактирование, а также назначение заданий.
   2. Управление семьей
   3. Управление профилем
4. Ребёнок
   1. Управление профилем
   2. Управление заданиями. Включает в себя функции выбора задания и управления статусом задания.

2.8 Выводы по главе 2

На данном этапе разработки были определены основные функциональные модули, отражающие структуру разрабатываемой системы, спроектирована конечная архитектура приложения и концептуальная модель БД. Так же определены основные функциональные возможности разрабатываемой системы.

На этапе проектирования были сформулированы основные требования для выбора платформы и средств реализации.

Согласно спроектированным модулям, общей схеме функционирования системы и руководствуясь предъявляемыми критериями выбора СП, были выбраны следующие средства реализации:

* Язык программирования C#
* Язык программирования JavaScript
* Платформа ASP.NET Web API 2
* Хранилище данных MSSQL
* Библиотека разработки интерфейсов React
* Инструмент управления состояниями Redux
* Технология доступа к данным Entity Framework
* Динамический язык стилей Scss

ГЛАВА 3

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ «Family Adventure World»

3.1 Распределение уровней реализации по технологиям

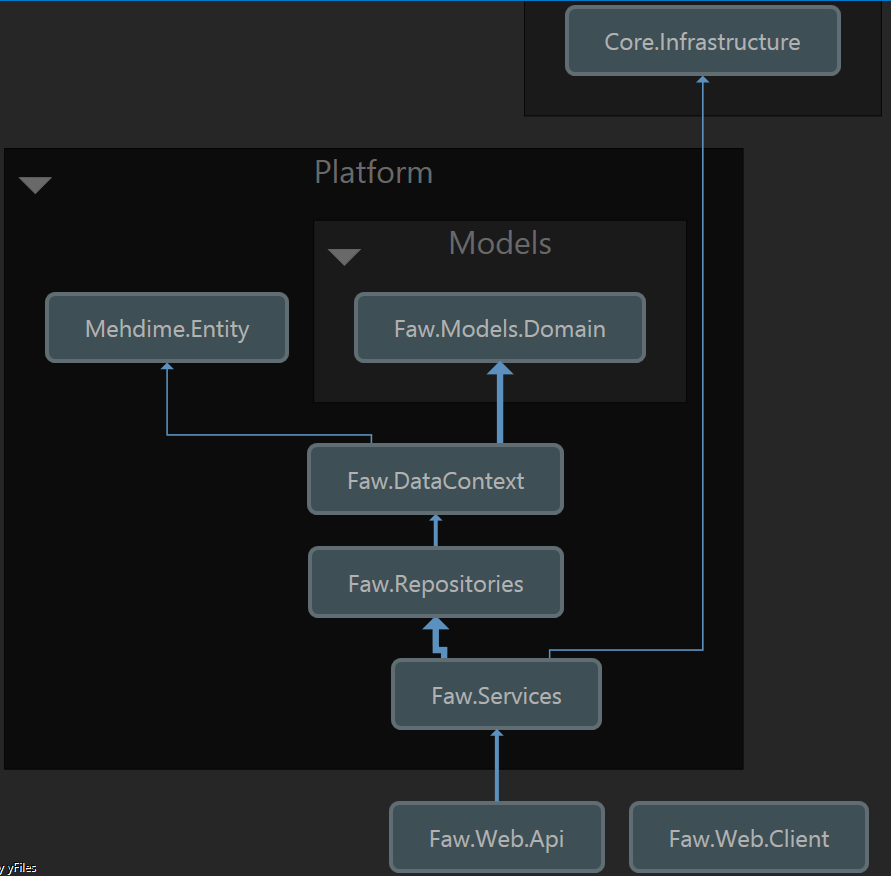
Реализация системы многомодульной структуры требует в первую очередь распределения уровней реализации в зависимости от требований, наложенных на них и, конечно же, актуальности технологий, которые можно использовать для осуществления поставленных целей. Выполним распределения уровней реализации по используемым технологиям.

Для начала рассмотрим самый первый, и в свою очередь один из самых важных уровней – уровень данных. Для хранения данных используется MS SQL.

Второй уровень реализации – это уже программная часть, предназначенная для работы с базой данных и соответствующими сущностями базы, так называемая DAO-прослойка приложения. Для реализации организации доступа к данным БД был выбран .Net Фреймворк Entity Framework, предоставляющий достаточно мощные и развитые средства, позволяющие не сосредотачиваться на подробностях синхронизации моделей БД и соответствующих им моделях системы. А также предоставляющее развитое Fluent API для описания связей между моделями.

Очередной этап разработки – это налаживание сервисов – классов, отвечающих за обработку данных, выполнение определенных действий, реализацию алгоритмов системы. Для разработки на этом этапе использовались стандартные средства .Net.

Использование Web Api позволило разделить данные, представление и обработку действий пользователя на три отдельных компонента. Важно отметить, что как представление, так и поведение зависят от модели. Однако модель не зависит ни от представления, ни от поведения. Это одно из ключевых достоинств подобного разделения. Оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели, что в итоге и позволяет использовать части одного модуля в пределах другого, и при том даже полностью изменить способ представления этого приложения в соответствии с поставленными целями.



Говоря об Web Api, становится вполне понятно, что очередным, завершающим этапом разработки является стадия разработки пользовательского представления. На сегодняшний день существует очень большое количество средств, предназначенных для разработки графических интерфейсов пользователей для веб-приложений. Каждое из этих средств отличается своим набором положительных и отрицательных факторов. Для реализации пользовательского представления разработанной системы используются библиотека React, инструмент управления состояниями Redux, фреймворк Material Design Light, динамический язык стилей Scss. Немаловажную роль также сыграли технологии CSS3 и HTML5, которые значительно расширили и упростили возможности создания и оформления пользовательского интерфейса.

3.2 Использование паттерна «Инверсия управления»

Для того, чтобы уменьшить связность между различными слоями, был применен паттерн объектно-ориентированного программирования «Инверсия управления» (англ. Inversion of Control, IoC).

Инверсия управления (англ. Inversion of Control, IoC) — важный принцип объектно-ориентированного программирования, используемый для уменьшения связанности в компьютерных программах. Модули верхнего уровня не должны зависеть от модулей нижнего уровня, оба должны зависеть от абстракции. Каждый компонент системы должен быть как можно более изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

IoC-контейнер определяет поведение того, как он создает объекты и только в нем определяется отношение «интерфейс-реализация». Больше нигде в приложении этого нет. Фактически, когда IoC-контейнер сталкивается с какой-то зависимостью, например в конструкторе, то он пытается вместо этого интерфейса сам подставить нужную реализацию.

В системе всё разбито на различные сборки. Сборка Services (interfaces) является точкой входа в систему, с ней могут работать различные приложения, API сервисы и т.д. Эти интерфейсы реализует сборка Services (implementation), которая выполняет основную бизнес логику приложения, она же зависит от Repositories (interfaces). А реализация Repositories (implementation) зависит от базы данных.

Для того, чтобы изолировать компоненты системы друг от друга и не полагаться на детали реализации, необходимо что бы каждая сущность в системе обладала своим собственным интерфейсом или абстракцией, в которой бы были вынесены только те функции, которые могут быть использованы другими компонентами. Компоненты должны зависеть только от абстракций, а абстракции не должны зависеть от реализации.

Листинг 3.2.1 Пример использования IoC-контейнера

public class UserService : Service, IUserService

{

private readonly IUserRepository \_userRepository;

private readonly IAccountRepository \_accountRepository;

private readonly IPlayerInfoRepository \_playerInfoRepository;

private readonly IUserTypeQueryService \_userTypeQueryService;

private readonly IAccountQueryService \_accountQueryService;

public UserService(

IUserRepository userRepository,

IMapper mapper,

IAccountRepository accountRepository,

IDbContextScopeFactory contextScopeFactory,

IUserTypeQueryService userTypeQueryService,

IAccountQueryService accountQueryService,

IPlayerInfoRepository playerInfoRepository,

IMapper mapper)

: base(mapper, contextScopeFactory)

{

\_userRepository = userRepository;

\_accountRepository = accountRepository;

\_userTypeQueryService = userTypeQueryService;

\_accountQueryService = accountQueryService;

\_playerInfoRepository = playerInfoRepository;

}

}

Как видно из реализации, сущность UserService имеет свой собственный интерфейс IUserService, посредством которого осуществляется доступ к этому сервису. Так же он использует другие компоненты системы, такие как UserTypeQueryService, AccountQueryService и др. которые передаются параметрами в конструктор. Следовательно, UserService зависит только от интерфейсов IUserTypeQueryService, IAccountQueryService и др., соответственно за этими интерфейсами может скрываться все что угодно.

Благодаря такой реализации удалось уменьшить связность каждого компонента до минимума. Однако сам UserService не поймет какой объект использовать, например, вместо IUserTypeQueryService. Для этого был использован IoC-контейнер, который представляет собой некую абстрактную фабрику, которая собирает внутри себя объекты, во время исполнения программы, и при обращении к которой она возвращает уже созданный объект. Однако IoC-контейнер может возвращать только зарегистрированные объекты.

Листинг 3.2.2 Пример регистрации объектов в IoC-контейнер

public class NinjectServiceModule : NinjectModule

{

public override void Load()

{

Bind<IUserService>().To<UserService>();

}

}

Для использования в системе был использована библиотека Ninject. При помощи класса NinjectModule в системе на каждом уровне можно создавать модули регистрации интерфейсов и их реализаций. В системе для каждого уровня написан свой модуль.

Листинг 3.2.3 Пример регистрации NinjectModule для всего проекта

private static void RegisterServices(IKernel kernel)

{

kernel.Load(AppDomain.CurrentDomain.GetAssemblies());

Mapper.Initialize(cfg => cfg.AddProfiles(AppDomain.CurrentDomain.GetAssemblies()));

kernel.Bind<IMapper>().ToMethod(x => Mapper.Configuration.CreateMapper()).InSingletonScope();

}

При помощи метода Load в IoC контейнер загружаются все зависимости которые в последующем будут автоматически передаваться в конструкторы классов.

3.3 Реализация модуля доступа к базе данных

Одной из систем объектно-реляционного отображения является Entity Framework (EF) от Microsoft. Основной идея которого заключается в том, чтобы между объектной моделью и БД поместить еще один слой абстракции – это модель сущностей или концептуальную модель (Entity Data Model). Согласно этой модели и будут определяться правила отражения объектов на БД. Создать эту модель можно несколькими способами: "Database First", "Model First", "Code First". Так как база данных уже создана, то был выбран метод Database First.

Database First позволяет реконструировать модель на основе существующей базы данных, так же может автоматически создать модель данных, состоящую из классов и свойств, соответствующих объектам базы данных (таким, как таблицы и столбцы). Информация о структуре базы, модель данных и маппинг их друг на друга содержится в XML в файле edmx. Visual Studio предоставляет графический дизайнер Entity Framework, с помощью которого можно просматривать и редактировать edmx файл.

Для доступа к данным используется паттерн Repository который позволяет абстрагироваться от конкретной базы данных и в любой момент заменять их или использовать различные базы данных для разных целей.

Для связи с базой данных использовался Entity Framework подход Data Base First. При помощи “Fluent Api” были описаны соответствия между классами для базы данных.

Так как почти все сущности должны содержать уникальный идентификатор за основу для всех DTO моделей была выделена абстрактная модель BaseEntity. Которая содержит в себе одно приватное поле \_entityId которое всегда инициализируется при создании модели в новое значение при помощи статического метода NewGuid класса Guid. Так же в классе присутствует публичное свойство EntityId которое позволяет получить значение приватного поля или задать другое значение. В последующем все DTO модели будут наследоваться он данного класса.

Листинг 3.3.1 Реализация абстрактного класса BaseEntity

[Serializable]

public abstract class BaseEntity : ICloneable

{

#region Fields

private Guid \_entityId = Guid.NewGuid();

#endregion Field

#region Public Properties

[DataMember]

public Guid EntityId

{

get { return \_entityId; }

set { \_entityId = value; }

}

#endregion Public Properties

#region ICloneable Members

public virtual object Clone()

{

return this.MemberwiseClone();

}

#endregion ICloneable Members

}

В Fluent Api существует обобщенный класс EntityTypeConfiguration<TEntityType> который позволяет в конструкторе описывать маппинг модели для баз данных.

Используя модель BaseEntity и EntityTypeConfiguration<TEntityType> был выделен класс BaseEntityTypeConfiguration<T> где T является наследником от базового класса BaseEntity. В базовом классе описан маппинг первичного ключа.

Листинг 3.3.2 Реализация обобщенного класса BaseEntityTypeConfiguration<T>

public class BaseEntityTypeConfiguration<T> : EntityTypeConfiguration<T> where T : BaseEntity

{

protected BaseEntityTypeConfiguration(string keyColumnName)

{

if (keyColumnName == null) throw new ArgumentNullException(nameof(keyColumnName));

Property(e => e.EntityId)

.HasColumnName(keyColumnName)

.HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);

HasKey(e => e.EntityId);

}

}

Далее разрабатывался слой доступа к данным (Data Access Object Layer), который обеспечивает взаимодействие клиентского кода со слоем «Модуль формирования и обработки данных». Доступ к базе данных представлен классом FawDataContext, унаследованным от DbContext, свойства которого позволяют работать с объектами базы данных, используя LINQ to Entities.

Листинг 3.3.3 Реализация класса доступа к базе данных.

public class FawDataContext : DbContext

{

static FawDataContext()

{

Database.SetInitializer<FawDataContext>(null);

}

public FawDataContext(string nameOrConnectionString) : base(nameOrConnectionString)

{

}

public FawDataContext(DbConnection existingConnection) : base(existingConnection, true)

{

}

public DbSet<User> Users { get; set; }

public DbSet<Quest> Quests { get; set; }

public DbSet<Account> Accounts { get; set; }

public DbSet<Claim> Claims { get; set; }

public DbSet<Family> Families { get; set; }

public DbSet<Item> Items { get; set; }

public DbSet<UserType> UserTypes { get; set; }

public DbSet<PlayerInfo> PalyerInfos { get; set; }

public DbSet<UserQuest> UserQuests { get; set; }

public DbSet<Expirience> Expiriences { get; set; }

public DbSet<Achivment> Achivments { get; set; }

public DbSet<UserAchivment> UserAchivments { get; set; }

public DbSet<ExpressionProperty> ExpressionProperties { get; set; }

public DbSet<PropertyValue> PropertyValues { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

modelBuilder.Configurations.Add(new AccountEntityConfiguration());

modelBuilder.Configurations.Add(new ClaimEntityConfiguration());

modelBuilder.Configurations.Add(new FamilyEntityConfiguration());

modelBuilder.Configurations.Add(new ItemEntityConfiguration());

modelBuilder.Configurations.Add(new QuestEntityConfiguration());

modelBuilder.Configurations.Add(new SettingEntityConfiguration());

modelBuilder.Configurations.Add(new UserEntityConfiguration());

modelBuilder.Configurations.Add(new UserTypeEntityConfiguration());

modelBuilder.Configurations.Add(new PalyerInfoEntityConfiguration());

modelBuilder.Configurations.Add(new UserQuestEntityConfiguration());

modelBuilder.Configurations.Add(new ExpirienceEntityConfiguration());

modelBuilder.Configurations.Add(new AchivmentEntityConfiguration());

modelBuilder.Configurations.Add(new UserAchivmentEntityConfiguration());

modelBuilder.Configurations.Add(new ExpressionPropertyEntityConfiguration());

modelBuilder.Configurations.Add(new PropertyValueEntityConfiguration());

}

}

Для того, чтобы на прямую не обращаться к Entities и уменьшить связность приложения, был применен паттерн программирования Repository. Который полностью скрывает работу класса FawDataContext и предоставляет лишь объект для работы с конкретной сущностью из базы данных.

Для реализации паттерна Repository был выделен обобщенный интерфейс IRepository<T>. Который содержит в себе описание для общих во всех репозиториях методах.

* void Insert(T entity); - Добавление сущности в базу данных
* void Update(T entity); - Обновление сущности в базе данных
* void Delete(Guid entityId); - Удаление сущности из базы данных по первичному ключу.
* T GetById(Guid entityId); - Получение сущности из базы данных по первичному ключу
* IQueryable<T> GetWhere(Func<T, bool> predicate); - Получение запроса сущностей из базы данных по определенному условию который задает функция предиката для данной модели.
* IQueryable<T> Get(); - Получение запроса всех сущностей из базы данных
* T Save(T entity); Сохранение сущности в базу данных
* void DeleteRange(IEnumerable<Guid> guids); Удаление всех записей из базы данных которые находятся в списке уникальных модификаторов переданных в функцию

Так как реализовывать данные методы в каждом репозитории это однотипная задача был реализован обобщённый абстрактный класс, который реализует все данные методы. Таким образом любой репозиторий реализуя данный класс получают базовый набор функций, а также доступ к контексту.

Листинг 3.3.4 Реализация класса доступа к базе данных.

public abstract class Repository<TEntity> : IRepository<TEntity> where TEntity : BaseEntity, new()

{

private readonly IAmbientDbContextLocator \_ambientDbContextLocator;

protected FawDataContext DbContext

{

get

{

return \_ambientDbContextLocator.Get<FawDataContext>();

}

}

protected Repository(IAmbientDbContextLocator dataContext)

{

\_ambientDbContextLocator = dataContext;

}

public virtual void Insert(TEntity entity)

{

DbContext.Set<TEntity>().Add(entity);

}

public virtual void Delete(Guid entityId)

{

var entityToDelete = GetById(entityId);

if (entityToDelete == null)

throw new ObjectNotFoundException();

Delete(entityToDelete);

}

public virtual TEntity GetById(Guid entityId)

{

return DbContext.Set<TEntity>().FirstOrDefault(x => x.EntityId == entityId);

}

public virtual IQueryable<TEntity> GetWhere(Func<TEntity, bool> predicate)

{

return DbContext.Set<TEntity>().Where(predicate).AsQueryable();

}

public IQueryable<TEntity> Get()

{

return DbContext.Set<TEntity>();

}

public virtual void Update(TEntity entityToUpdate)

{

DbContext.Entry(entityToUpdate).State = EntityState.Modified;

}

private void Delete(TEntity entity)

{

DbContext.Entry(entity).State = EntityState.Deleted;

}

public virtual TEntity Save(TEntity entity)

{

return DbContext.InsertOrUpdate(entity, entity.EntityId);

}

public virtual void DeleteRange(IEnumerable<Guid> guids)

{

DbContext.Set<TEntity>().RemoveRange(guids);

}

}

3.4 Реализация модуля формирования и обработки данных

Модуль формирования и обработки данных представляет собой объединение определенного количества сервисов, каждый из которых реализует свою бизнес-логику. Бизнес-логика – совокупность правил, принципов, зависимостей поведения объектов предметной. Иначе можно сказать, что бизнес-логика — это реализация правил и ограничений автоматизируемых операций.

Данный модуль был разработан для того, чтобы манипулировать получаемыми объектами из нижнего слоя или, иными словами, он организует и манипулирует сервисами, каждый из которых выполняет логически выделенный объем бизнес логики.

Примером такого сервиса может служить класс UserService, в котором реализованы следующие методы:

* Guid Register(User user) – метод позволяющий зарегистрировать нового пользователя в системе
* UserVerifyResult Verify(Guid token) – метод проверяющий токен который был выслан пользователю по почте для подтверждения аккаунта
* bool Authenticate(string emailOrlogin, string password) – метод который который осуществляет аунтетификацию пользователя в системе
* void Edit(User user) – метод который используется для обновления информации о пользователе

3.5 Реализация конструктора достижений

Практически во всех играх в том или ином виде присутствуют достижения. Каждый игрок за выполнение заданий или просто за определенный набор действий и даже просто по истечению времени получают достижения. Достижения в игре позволяют добавить дополнительный интерес к процессу, а также другие цели, которые являются необязательными, но не менее интересными.

Для реализации данной возможности в системе на серверной стороне были выбраны деревья выражений (ExpressionTree). При помощи специального api присутствует возможность составлять выражения во время выполнения программы, выражения, которые не требуют перекомпиляции проекта. Но для того что бы создать выражение требуются данный которые бы описывали данное выражение, а также что бы эти данные могли передаваться через web api и сохраняться в базе.



Для сохранения условия выражения в базе данных для того что бы данное выражение можно было редактировать и восстанавливать была разработана структура базы данных (рис …). Так как выражение может быть сложным и состоять из нескольких условий связь между таблицами Achievement и ExpressionProperty является 1 \* n (один ко многим). Каждая запись ExpressionProperty хранит в себе название модели (ModelName) это позволяет создавать условия использую несколько различных моделей. Примером может быть условие для получения достижения за выполнение сложного задания на определенном уровне, сложность задания хранится в модели задания (Quest) а уровень игрока хранится в информации об игроке (PlayerInfo). Так для этого условия будет создано две записи в таблице ExpressionProperty которые будут иметь ссылки на одну запись достижения (Achievement).

Так же в модели ExpressionProperty содержатся Comparer и Connector. Comparer содержит в себе компаратор который будет применен для сравнения двух частей выражения. Connector является связывающим звеном одного выражения с другим, он является не обязательным и если он не указан то это является показателем того что это последнее выражение в списке и далее не последует других.

Таблица PropertyValue является хранилищем для информации об значении или названии свойства модели, а также их типа. Каждое выражение (ExpressionProperty) содержит в себе ссылки на две PropertyValue для левой и правой части выражения.

Для компиляции данных в выражения, которые будут использованы для проверки выполнения достижения был выделен интерфейс IConditionExpressionBuilder

Листинг 3.4.1 Интерфейс **IConditionExpressionBuilder**.

public interface IConditionExpressionBuilder

{

Expression<Func<T, bool>> Build<T>(IEnumerable<ExpressionProperty> expressionProperties);

Expression<Func<T, bool>> Build<T>(ExpressionProperty expressionProperty);

}

Интерфейс содержит в себе два определения обобщенных метода для генерации сложного выражения, а также одинарного.

Листинг 3.4.2 Реализация интерфейса **IConditionExpressionBuilder**.

public class ConditionExpressionBuilder : IConditionExpressionBuilder

{

public Expression<Func<T, bool>> Build<T>(IEnumerable<ExpressionProperty> expressionProperties)

{

if (expressionProperties.All(x => x.ModelName != typeof(T).Name))

throw new InvalidCastException(

$"All expression property should have same type and be equeal to {typeof(T).Name} type");

var firstExpression = expressionProperties.OrderByDescending(x => x.Order).First();

var expression = GetPropertyExpression<T>(firstExpression);

if(expressionProperties.Count() == 1)

return Expression.Lambda<Func<T, bool>>(expression);

foreach (var expressionProperty in expressionProperties)

{

var currentExpression = GetPropertyExpression<T>(expressionProperty);

switch (expressionProperty.Connector)

{

case Connector.Or:

expression = Expression.OrElse(expression, currentExpression);

break;

case Connector.And:

expression = Expression.AndAlso(expression, currentExpression);

break;

}

}

return Expression.Lambda<Func<T, bool>>(expression);

}

public Expression<Func<T, bool>> Build<T>(ExpressionProperty expressionProperty)

{

return Expression.Lambda<Func<T, bool>>(GetPropertyExpression<T>(expressionProperty));

}

private Expression GetPropertyExpression<T>(ExpressionProperty expressionProperty)

{

var typeParam = Expression.Parameter(typeof(T));

var leftExpression = GetPropertyValueExpression(expressionProperty.LeftPropertyValue, typeParam);

var righExpression = GetPropertyValueExpression(expressionProperty.RightPropertyValue, typeParam);

Expression currentExpression;

switch (expressionProperty.Comparer)

{

case Comparer.Less:

currentExpression = Expression.LessThan(leftExpression, righExpression);

break;

case Comparer.Equal:

currentExpression = Expression.Equal(leftExpression, righExpression);

break;

case Comparer.Greater:

currentExpression = Expression.GreaterThan(leftExpression, righExpression);

break;

case Comparer.LessOrEqual:

currentExpression = Expression.LessThanOrEqual(leftExpression, righExpression);

break;

case Comparer.GreaterOrEqual:

currentExpression = Expression.GreaterThanOrEqual(leftExpression, righExpression);

break;

default:

currentExpression = Expression.Equal(leftExpression, righExpression);

break;

}

return currentExpression;

}

private Expression GetPropertyValueExpression(PropertyValue propertyValue, Expression parameterExpression)

{

if (string.IsNullOrEmpty(propertyValue.PropertyName))

{

return Expression.Constant(propertyValue.Value);

}

return Expression.Property(parameterExpression, propertyValue.PropertyName);

}

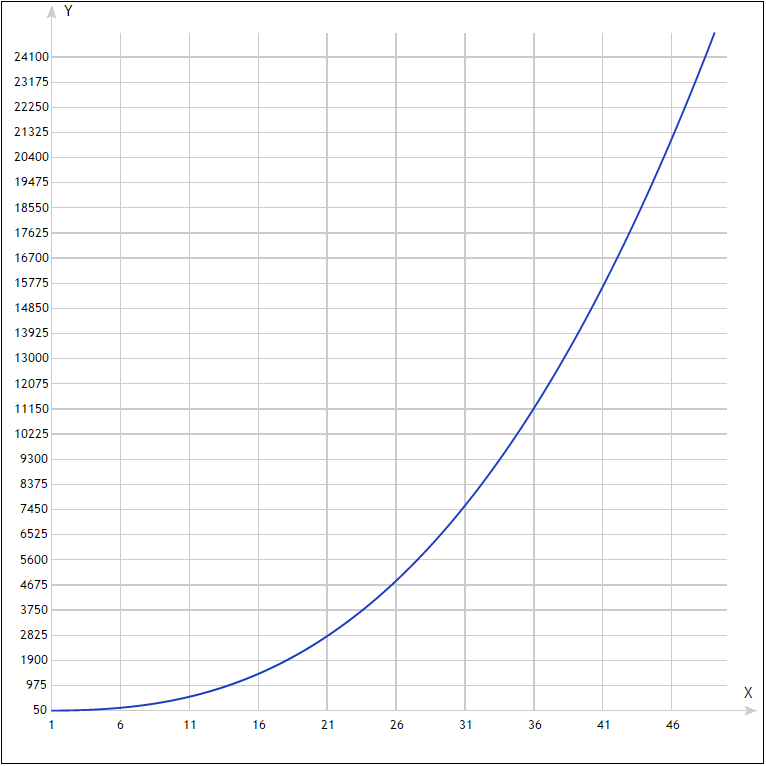
}

Как видно из кода в листинге 3.4.2 метод Build принимает в себя массив ExpressionProperties только одного типа. Для того что бы сгенерировать выражение для нескольких моделей нужно сгруппировать все ExpressionProperties по полю ModelName и для каждой группы вызвать метод Build.

Основным нюансом является то что каждая модель, которая используется для построения выражения должна быть связана с пользователем, и сервисы, которые используют в себе эти выражения должны добавлять к каждому выражению дополнительные условия для проверки что данное условия выполняется для конкретного пользователя или пользователей.

3.6 Стратегия развития персонажей

Важная задача в разработке игры — это правильный выбор стратегии по развитию виртуальных персонажей, а именно стратегии набора опыта. Первым шагом реализации этой задачи является определение таблицы опыта, которая станет дорогой игроков к вершинам. В большинстве случаев функция получения опыта описывается кривой, точки которой вычисляются по формуле: прим.(Рис. 1), где baseExpirience - константа, указывающая количество опыта требуемого для получения уровня, level - это уровень, для которого рассчитывается количество опыта, а factor - это константа, которая определят на сколько больше опыта нужно получить для указанного уровня. Чем больше это значение, тем более изогнутой получится кривая. Таким образом, набор опыта усложняется плавно по мере получения уровня, если правильно выбрать значение factor.



**(Рис. 1) График таблицы опыта при baseExpirience = 50 и factor = 2.6**

Так как единственный способ получения опыта в игре — это выполнение заданий, то ключевым моментом является правильно полученная формула для вычисления количества опыта, который игрок будет получать за задания. Зачастую, в играх присутствует большое количество факторов, которые влияют на результирующее значение: разница в уровнях персонажа; боевая единица, которую убивает игрок; разница в их характеристиках; сложность боевой единицы; время, которое игрок тратит на задание и другие. Для вычисления базового опыта в данной игре будет использоваться простая формула:, где expirienceToNexLevel - количество опыта, требуемого для перехода на следующий уровень, а coefficient - коэффициент который вычисляется случайным образом в зависимости от сложности задания.

|  |  |
| --- | --- |
| **Уровень сложности задания** | **Диапазон значений** |
| Легкий | 8-10 |
| Средний | 6-8 |
| Сложный | 3-6 |
| Невозможный | 1-3 |

**(Табл. 1) Диапазон значений для различных уровней сложности заданий**

При завершении задания количество опыта не обязательно должно быть равным тому, которое было вычислено при создании задания. На результирующие значение могут влиять такие факторы, как разница во времени выполнения аналогичного задания другими игроками; разница в опыте и возрасте игрока; количество выполнений одного и того же задания подряд и другие факторы. Некоторые из них являются очень важными. Так разница в возрасте игроков может быть существенной и это необходимо учитывать. Например, мытье посуды займет у ребенка займет больше времени, чем у взрослого. На данном этапе вывести финальную формулу с учетом различных факторов довольно сложно, так как игровое приложение активно развивается, появляются новые факторы, влияющие на результат.

Таким образом, имея таблицу опыта и формулу вычисления количества опыта за выполнение задания, в системе появляется возможность контролировать игровой процесс. Сложной задачей будет собрать все факторы воедино и определить тонкую грань, которая позволит балансировать между количеством заданий, которые игроку требуется выполнить для достижения новых уровней, и скоростью получения новых уровней.

3.7 Выводы по главе 3

В этой главе была описана архитектура системы, принципы взаимодействия модулей с базой данных и между собой. Рассмотрены некоторые трудности возникшие при реализации приложения.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. RESTful API [Электронный ресурс] – Режим доступа: http://habrahabr.ru/post/144011/
2. Dependency Injection [Электронный ресурс] – Режим доступа: http://habrahabr.ru/post/166287/
3. Entity Framework [Электронный ресурс] – Режим доступа: https://msdn.microsoft.com/ru-ru/library/bb399567%28v=vs.110%29.aspx
4. Фримен, Фримен, Сьерра: Паттерны проектирования, Авторы: Фримен Эрик, Фримен Элизабет, Сьерра Кэтти, Бейтс Берт, Издательство: Питер, 2014 г.
5. Совершенный код. Мастер-класс / Пер. С англ. – М. : Издательство «Русская редакция», 2015. – 896 стр. : ил.
6. Буч, Г. UML / Г. Буч, А. Якобсон – Санкт-Петербург: Издательство «Питер», 2006 – 736 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Внешний вид программы