МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

Кафедра современных технологий программирования

ДЕМЕНОВИЧ АРТУР ЮРЬЕВИЧ

**Разработка социальной семейной онлайн-игры**

**"Family adventure world"**

Дипломный проект

студента 5 курса специальности

1-40 01 01 «Программное обеспечение информационных технологий»

дневной формы получения образования

|  |  |
| --- | --- |
| «Допустить к защите»  Заведующий кафедрой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Рудикова Л.В.  \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ 2017 г. | Научный руководитель  Карканица Анна Викторовна, старший преподаватель кафедры современных технологий программирования |
|  |  |

Гродно, 2017

СОДЕРЖАНИЕ

[РЕЗЮМЕ 4](#_Toc484640178)

[SUMMARY 5](#_Toc484640179)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ И ТЕРМИНОВ 6](#_Toc484640180)

[ВВЕДЕНИЕ 7](#_Toc484640181)

[ГЛАВА 1 9](#_Toc484640182)

[РАЗРАБОТКА БРАУЗЕРНЫХ ИГР: ОТ СЮЖЕТА ДО РЕАЛИЗАЦИИ 9](#_Toc484640183)

[1.1 Идея игрового сюжета, персонажи и роли 9](#_Toc484640184)

[1.2 Браузерные игры 10](#_Toc484640185)

[1.3 Обзор игровых приложений для детей и их родителей 11](#_Toc484640186)

[1.4 Игровой жанр «Социальная семейная игра» 11](#_Toc484640187)

[1.5 Общая концепция социальной семейной игры и роли пользователей 12](#_Toc484640188)

[1.6 Выводы по главе 1 13](#_Toc484640189)

[ГЛАВА 2 14](#_Toc484640190)

[ПРОЕКТИРОВАНИЕ БРАУЗЕРНОЙ ИГРЫ «FAMILY ADVENTURE WORD» 14](#_Toc484640191)

[2.1 Общая концепция игрового приложения 14](#_Toc484640192)

[2.2 Стратегия развития персонажей 15](#_Toc484640193)

[2.3 Концептуальная модель данных 17](#_Toc484640194)

[2.4 Варианты использования игрового приложения 19](#_Toc484640195)

[2.5 Проектирование макетов пользовательского интерфейса 20](#_Toc484640196)

[2.6 Выбор архитектурного решения 23](#_Toc484640197)

[2.7 Обоснование выбора способа обмена сообщения между клиентом и сервером (RESTFul API) 24](#_Toc484640198)

[2.8 Технологии и средства реализации 24](#_Toc484640199)

[2.8.1 Платформа ASP.NET Web API 2 25](#_Toc484640200)

[2.8.2 Технология доступа к данным Entity Framework 26](#_Toc484640201)

[2.8.3 Библиотека разработки интерфейсов React 26](#_Toc484640202)

[2.8.4 Инструмент управления состояниями Redux 27](#_Toc484640203)

[2.9 Выводы по главе 2 28](#_Toc484640204)

[ГЛАВА 3 29](#_Toc484640205)

[ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ «FAMILY ADVENTURE WORLD» 29](#_Toc484640206)

[3.1 Распределение уровней реализации по технологиям 29](#_Toc484640207)

[3.2 Использование паттерна «Инверсия управления» 31](#_Toc484640208)

[3.3 Реализация модуля доступа к базе данных 33](#_Toc484640209)

[3.4 Реализация модуля формирования и обработки данных 36](#_Toc484640210)

[3.5 Реализация конструктора игровых достижений 37](#_Toc484640211)

[3.6 Реализация клиентского приложения 40](#_Toc484640212)

[3.7 Выводы по главе 3 42](#_Toc484640213)

[ЗАКЛЮЧЕНИЕ 43](#_Toc484640214)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 45](#_Toc484640215)

[ПРИЛОЖЕНИЯ 46](#_Toc484640216)

РЕЗЮМЕ

Деменович Артур Юрьевич

Дипломная работа − «Разработка социальной семейной онлайн-игры «Family adventure world»»

62 страниц, 24 иллюстраций, 2 приложение, 1 таблица.

**Ключевые слова** − социальная игра, семейная игра, браузерная игра, веб-приложение, Web API 2, React, Redux, Entity Framework.

**Объект исследования** − методы разработки игровых приложений, работающих в браузере.

**Предмет исследования** − технологии и программные средства для проектирования и программной реализации веб-ориентированных приложений.

**Цель дипломной работы** − разработка социально-ориентированной семейной игры «Семейный мир приключений», вовлекающей членов семей во взаимодействие между собой в игровой форме в процессе создания, назначения и выполнения заданий.

**Методы исследования** − объектно-ориентированный анализ и проектирование программных систем, структурный анализ, методы проектирования баз данных, методы проектирования информационных систем, технологии разработки клиент-серверных приложений, методы построения пользовательских интерфейсов.

**Авторская характеристика работы** − результатом работы является разработка и программная реализация игрового приложения «Семейный мир приключений», обеспечивающего новый вид взаимодействия между членами семьи в игровой форме в виде создания, назначения, выполнения заданий.

SUMMARY

Demenovich Artur

Graduate work − «Develop of social family online-game «Family adventure world»»

62 pages, 24 illustrations, 2 applications, 1 table.

**Keywords** − Social game, family game, browser game, web application, Web API 2, React, Redux, Entity Framework.

**Object of research** − methods for developing game applications that work in the browser.

**Subject of study** − technology and software tools for the design and software implementation of information systems, technology and means to implement web-based applications.

**The aim of the graduate work** − development of community-oriented family games «Family adventure world», involving family members interact with each other in a playful way in the process of creation, appointment and task execution.

**Research methods** − object-oriented analysis and design of software systems, structural analysis, database design methods, methods of designing information systems, technologies for developing client-server applications, methods for building user interfaces.

**Author's response work** − the result of the work is the development and software implementation of the game application «Family Adventure World» which provides a new kind of interaction between family members in a game form in the form of creating, assigning and performing tasks.

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.

Цель дипломной работы – разработка социально-ориентированной семейной игры «Семейный мир приключений», вовлекающей членов семей во взаимодействие между собой в игровой форме в процессе создания, назначения и выполнения заданий.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Продумать сюжет игры, выделить персонажей, разработать сценарий и логику игрового процесса.
2. Проанализировать подходы к реализации многопользовательских браузерных онлайн-игр.
3. Выполнить программную реализацию многопользовательского игрового приложения, обеспечивающего:
   1. возможность создания и управления игровыми элементами и персонажами (семья, члены семьи, задания, достижения и др.);
   2. возможность создания, редактирования, назначения заданий, отслеживания статуса их выполнения;
   3. возможность приобретения персонажами опыта и вознаграждений за выполнение заданий.

ГЛАВА 1

РАЗРАБОТКА БРАУЗЕРНЫХ ИГР: ОТ СЮЖЕТА ДО РЕАЛИЗАЦИИ

1.1 Идея игрового сюжета, персонажи и роли

Основной идеей игры «Семейный мир приключений» 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. Задание – это просьба от члена семьи в котором он описывает проблему и как можно помочь ему с этим или сделать это за него. Задание может представлять из себя как простое вида “Помыть посуду”, так и сложного “Ухаживать за растениями в доме в течении недели”. Сложность заданий может определятся не только контекстом, т.к. для родителей “Помыть посуду” это просто рутина, а для ребенка в виду физических особенностей это целое приключение, а также непосредственно помечая насколько задание будет трудно выполнить. За каждое выполненное задание пользователь получает опыт и деньги. Предлагается реализовать в игре систему достижений, которая позволить мотивировать игроков на достижение новых целей.

Дополнительными элементами игры являются инвентарь и список желаемого. Инвентарь – это место где хранятся все предметы игрока, каждый предмет может представлять из себя как обычную реальную вещь, так и виртуальную, например, игры или фильмы. Предлагается интегрировать игру с различными сервисами такими как Steam, Amazon, ebay, чтобы предоставить пользователю больший выбор реально существующих предметов, которые можно добавлять в список желаемого.

Если игрок обладает достаточным количеством опыта и денег, он сможет запрашивать данные предметы. Фиксированием факта получения предмета является фото-подтверждение игрока с предметом. После этого предмет можно будет переместить в инвентарь. Интеграция приложения с различными социальными сетями позволит делиться своими достижениями, вещами и списком желаемого.

Игру предлагается реализовать в виде приложения, работающего в браузере, чтобы отвязать игроков от конкретного компьютера и избавить от необходимости инсталляции игры на компьютер.

1.2 Браузерные 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 в таких играх является виртуальные вещи, которые никак не осязаемы игроками в реальной жизни.

1.3 Обзор игровых приложений для детей и их родителей

В сети на сегодняшний день в сети отсутствуют игровые приложения, которые бы могли объединить общим занятием детей и их родителей.

Рассмотрим некоторые примеры игр для детей, которые позиционируются, как игры, в которых ребенок должен постоянно выполнять какие-то задания: «Фиксики Мастера», «Ми-Ми-Мишки», «Лунтик», «Бумажки». Разработчики подобных игр позиционируют их как игры, с помощью которых ребенок научится быть наблюдательным, любознательным, изобретательным, находчивым, чтобы решать новые задачи.

Несомненным достоинством таких игр является то, что с их помощью ребенок познает, как устроен мир и развивается. Однако чаще всего родители используют такие игры просто, чтобы занять ребенка на продолжительное время. В итоге ребенок проводит за телефоном и планшетом все свое свободное время, общение в семье сводится на нет.

1.4 Игровой жанр «Социальная семейная игра»

Существует огромное количество игровых жанров, отличающихся разнообразием игровых персонажей, сюжетных линий, логикой игрового процесса и другими параметрами. Однако однозначного толкования понятия социальной или семейной компьютерной игры до сих пор нет. Поэтому нужно выделить основные принципы, которые будут определять рамки данного понятия.

Социальная семейная компьютерная игра – это игра, работающая в браузере, не требующая установки на компьютер, вовлекающая в игровой процесс членов семей (детей и их родителей) и призванная в игровой форме повысить эффективность взаимодействия детей и родителей на пути достижения общих целей.

Каждый участник игры имеет виртуального персонажа, который, можно сказать, является виртуальной копией реального человека. Взаимодействие между виртуальными членами семьи происходит в процессе выполнения игроками заданий, которые создаются, назначаются и выполняются игровыми персонажами. За выполнение заданий игрок получает награду в виде опыта и монет.

Такой игровой жанр, по нашему мнению, позволит вовлечь детей и родителей в эффективное взаимодействие и дисциплинирует обе стороны. При этом игроки не будут оторваны от полюбившихся ими гаджетов, однако теперь будут проводить время с большей пользой, выполняя родительские поручения и получая за это реальные вознаграждения.

1.5 Общая концепция социальной семейной игры и роли пользователей

Основной задачей разрабатываемого игрового приложения является задача превратить воспитательный процесс из скучных поручений, которые воспринимаются детьми как наказание, в интересные задания, похожи на те, которые ребенок выполняет в обычных компьютерных играх.

Для этого необходимо предусмотреть возможность создавать семью в игре, создавать задания для семьи, а также реализовать механизм получения наград и систему достижений. Таким образом, выделяются следующие роли пользователей:

* Dad (Папа)
* Mom (Мама)
* Son (Сын)
* Daughter (Дочь)
* Administrator (Администратор)

Пользователи Dad, Mom имеют возможность создания семьи и заданий. Son, Daughter могут выполнять задания. Administrator имеет возможность управлять системой достижений в игре, а также управлять пользователями.

1.6 Выводы по главе 1

Обзор существующих жанров и сюжетных линий компьютерных игр, а также соответствующих программных реализаций этих игр, позволил сделать вывод о том, что актуальной видится задача разработки семейной онлайн-игры с социальным компонентом. Основная идея игры – создание игровой среды, в которой дети и родители будут активно взаимодействовать.

Для реализации высказанной идеи была предложена сюжетная линия игры, выделены персонажи, представлена общая концепция игры и логика игрового процесса.

Принято решение о реализации компьютерной игры «Семейный мир приключений» в виде браузерного приложения.

ГЛАВА 2

ПРОЕКТИРОВАНИЕ БРАУЗЕРНОЙ ИГРЫ «FAMILY ADVENTURE WORD»

2.1 Общая концепция игрового приложения

В соответствии с сюжетной линией игры, описанной в главе 1, выделим основные игровые элементы, модели которых необходимо построить и реализовать, а также компоненты браузерного приложения, необходимые для отображения статуса игры и ее персонажей:

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

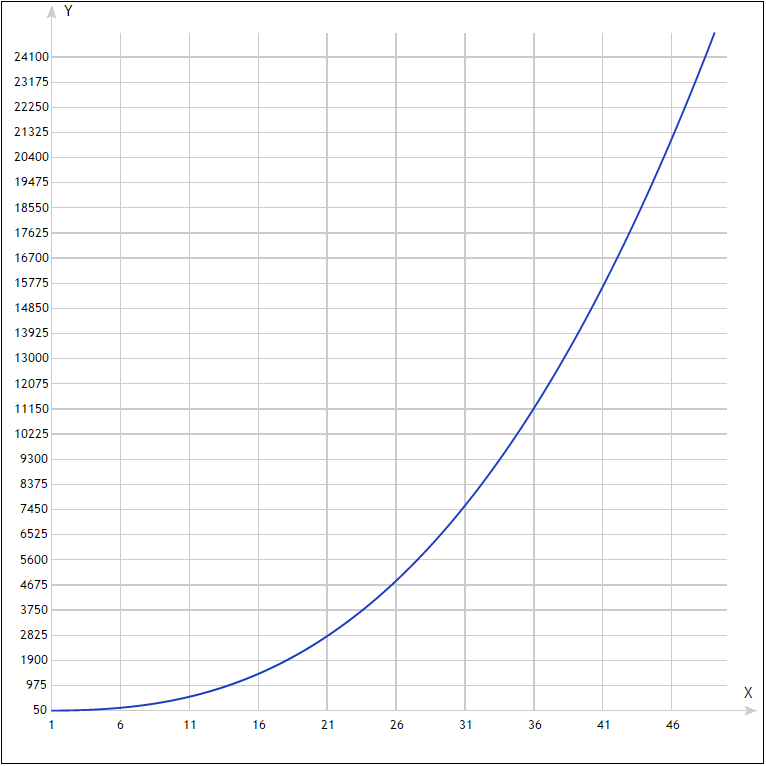
Для возможности создания и управления персонажами и игровыми элементами необходимо предусмотреть реализацию функций для:

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

Браузерная игра «Семейный мир приключений» будет реализовываться как многопользовательская онлайн-игра, поэтому необходимо предусмотреть разграничение прав доступа для ролей Dad, Mom, Son, Daughter, Administrator.

2.2 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 (рисунок 2.1).



**Р**исунок 2.1. 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. Такими факторами могут быть 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 |

**Таблица** 2.1. 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.

2.3 Концептуальная модель данных

Для построения концептуальной модели данных, выделим базовые сущности предметной области игрового приложения:

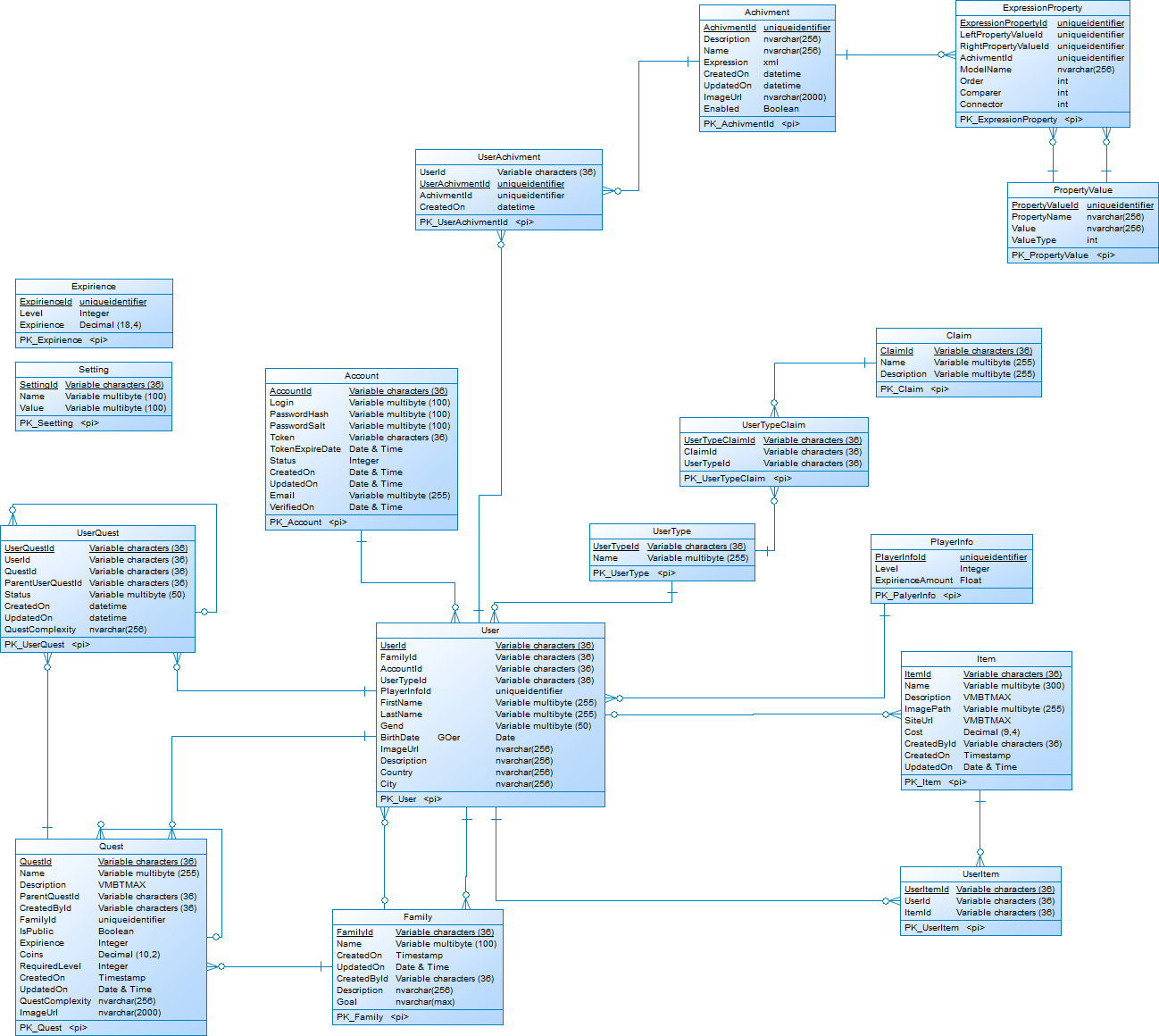
* User (Пользователь)
* PlayerInfo (Информация об игроке)
* Achivment (Достижения)
* ExpressionProperty (Свойство выражения)
* PropertyValue (Значение свойства)
* Quest (Задание)
* Family (Семья)
* Item (Вещь)
* Experience (Опыт)
* UserType (Тип пользователя)
* Claim (Права)

Каждая сущность 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 информацию, необходимую для восстановления условия достижения. Модель базы данных представлена на рисунке 2.2.



**Рисунок** 2.2. Концептуальная модель базы данных

2.4 Варианты использования игрового приложения

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 (рисунок 2.3).

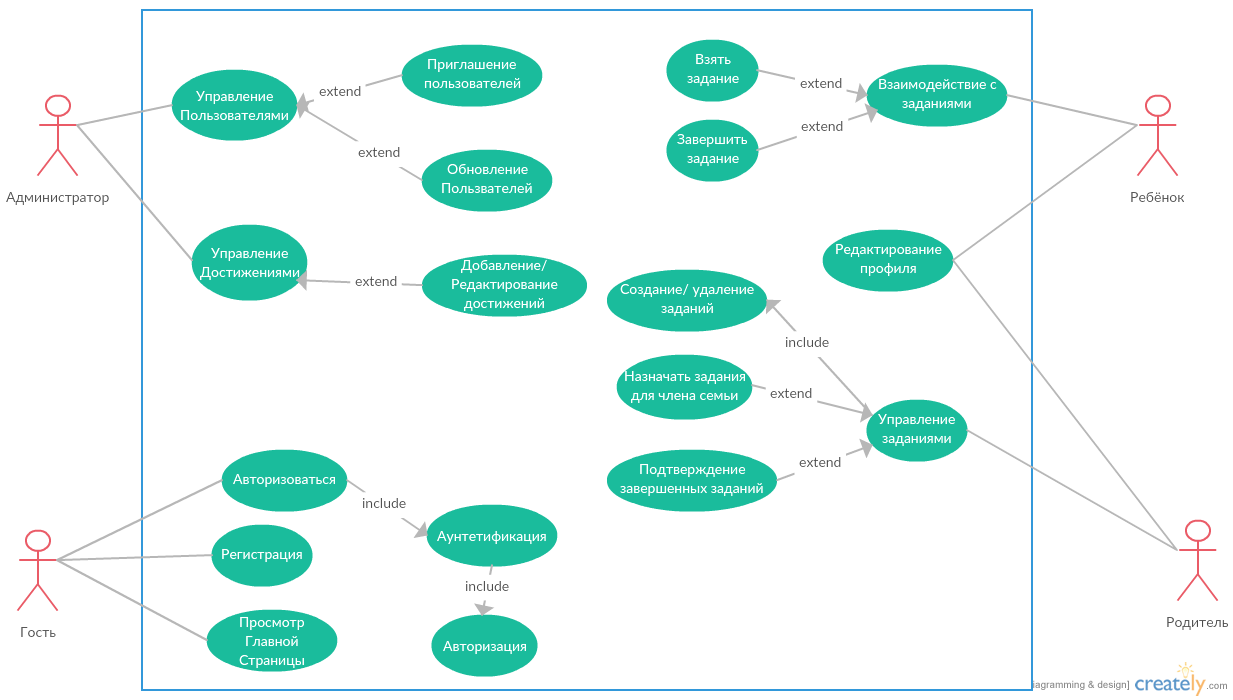


Рисунок 2.3. UML диаграмма вариантов использования

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.
   4. Администратор
   5. Управление достижениями (включает в себя создание и редактирование);
   6. Управление пользователями;
2. Родитель
   1. Управление заданиями (включает в себя создание и редактирование, а также назначение заданий);
   2. Управление семьей;
   3. Управление профилем.
3. Ребёнок
   1. Управление профилем;
   2. Управление заданиями (включает в себя функции выбора задания и управления статусом задания).

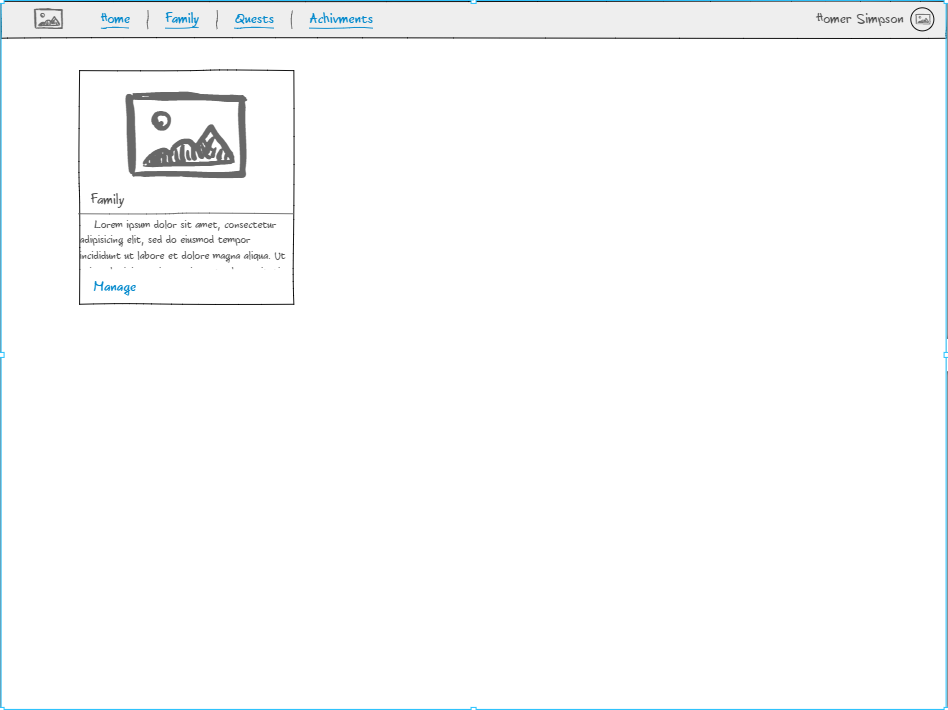
2.5 Проектирование макетов пользовательского интерфейса

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.

Учитывая все нюансы и пожелания конечных пользователей, был разработан интерфейс, часть которого представлена в приложении А. Макеты основных страниц приложения представлены на рисунках 2.4 – 2.7.



**Рисунок** 2.4. Шаблон главной страницы

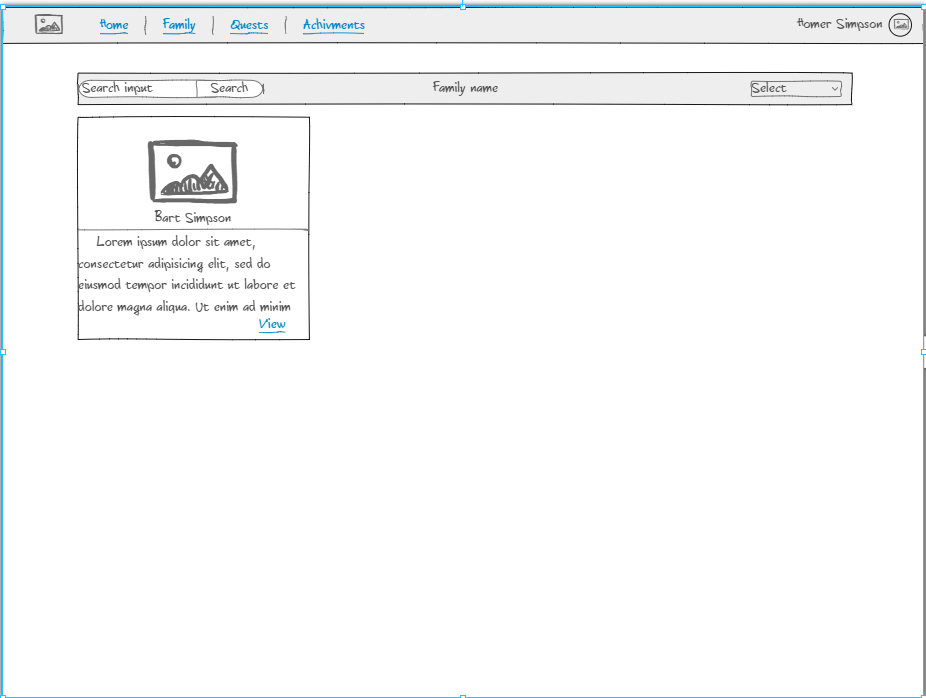


Рисунок 2.5. Шаблон главной страницы семьи

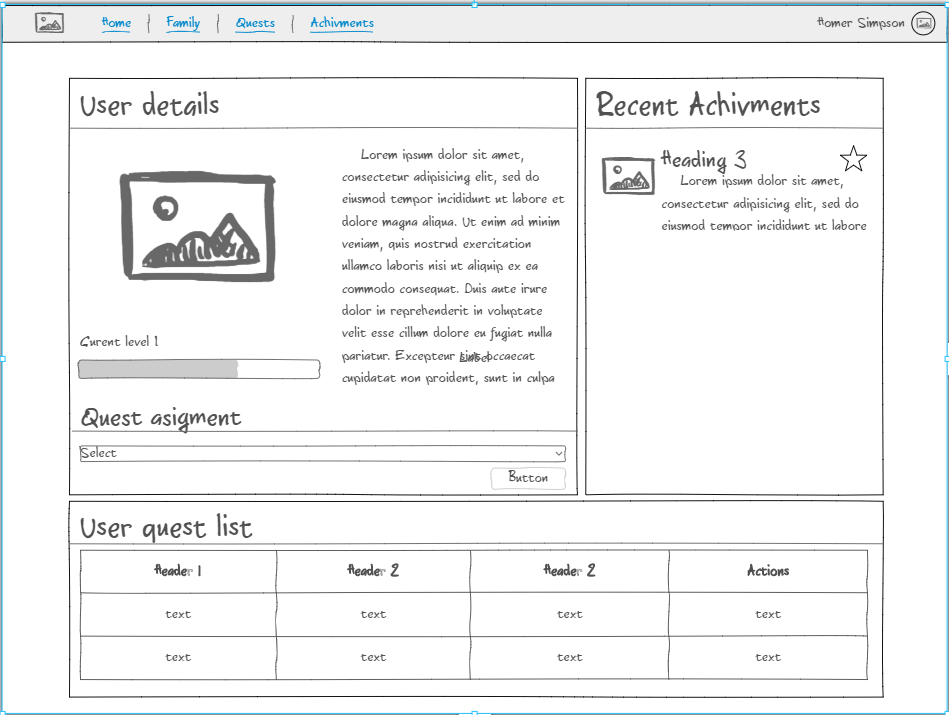


Рисунок 2.6. Шаблон страницы информации о члене семьи

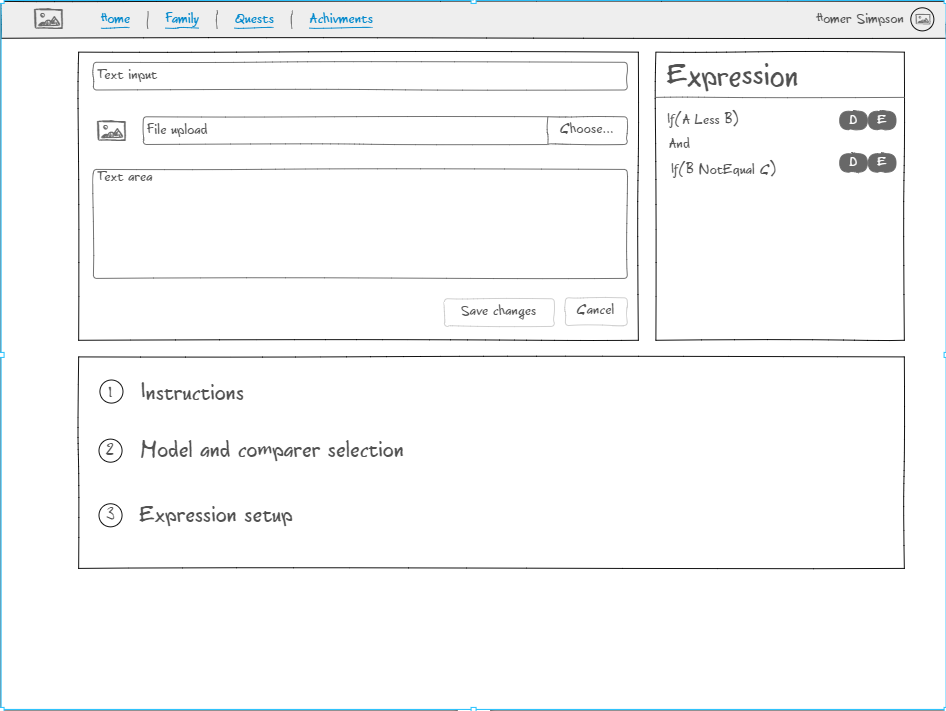


Рисунок 2.7. Шаблон редактора достижений

2.6 Выбор архитектурного решения

В качестве основы для построения браузерной многопользовательской игры была выбрана типичная 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.

2.7 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 [1].

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.

2.8 Технологии и средства реализации

Правильный выбор программной платформы является важнейшей задачей при построении любой компьютерной системы. 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.

Разрабатываемое игровое приложение является браузерным приложением, клиентская часть которого будет активно взаимодействовать с серверной. Для его реализации были выбраны следующие технологии, средства и языки программирования:

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

2.8.1 Платформа ASP.NET Web API 2

ASP.NET MVC представляет собой платформу для создания сайтов и веб-приложений с использованием паттерна 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.

2.8.2 **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, то 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 [3].

2.8.3 Библиотека разработки интерфейсов React

React — это библиотека для разработки интерфейсов, созданная Facebook. В последний год React 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.

В React синтаксис JSX — это не HTML, это всего лишь синтаксический сахар для javascript-функций. Этот синтаксис может транслироваться в чистый javascript, как непосредственно на странице, так и на этапе сборки. Для прекомпиляции можно использовать удобный и привычный инструмент (gulp, grunt, webpack, что больше нравится).

Функция render компонента в общем случае вызывается при любом изменении состояния и свойства компонента. Если для маленького компонента это не кажется страшным, то для более или менее сложного постоянная перерисовка «на каждый чих» кажется избыточной. Но в React используется, так называемый, виртуальный DOM (Virtual DOM). Фактически React работает с копией реального DOM’а в оперативной памяти. Все изменения вносятся в это виртуальное дерево, потом текущая копия сравнивается с предыдущей копией и, наконец, в реальный DOM вносятся только отличия между этими копиями.

2.8.4 Инструмент управления состояниями Redux

Данные в React «текут» через компоненты. Это называется «однонаправленный поток данных» — поток данных проходит в одном направлении, от родителя к ребенку. При этом не очевидно, как два компонента, не связанные отношением родитель-ребенок, будут взаимодействовать между собой.

В React не рекомендуется реализовывать прямое взаимодействие компонент-компонент. Это считается плохой практикой, приводит к ошибкам и спагетти-коду — старый термин для запутанного кода.

Для этого в дополнение к React используется 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.

Redux предлагает хранить все состояния приложения в одном месте, называемом «store» («хранилище»). Компоненты «отправляют» изменение состояния в хранилище, а не напрямую другим компонентам. Компоненты, которые должны быть в курсе этих изменений, «подписываются» на хранилище.

Хранилище может рассматриваться как «посредник» во всех изменениях состояния в приложении. С Redux компоненты не связываются друг с другом напрямую, все изменения должны пройти через единственный источник истины, через хранилище.

C Redux все компоненты получают свое состояние из хранилища. Также ясно, куда компонент должен отправить информацию об изменении состояния — опять же в хранилище. Компонент только инициирует изменение и не заботится об остальных компонентах, которые должны получить это изменение. Таким образом, Redux делает поток данных более понятным.

2.9 Выводы по главе 2

На этапе проектирования игрового приложения было принято решение о разработке браузерной игры, которая будет реализована в виде веб-приложения.

В качестве архитектурного решения была выбрана типичная трехуровневая архитектура, для реализации которой принято решение использовать следующий набор технологий и средств реализации: языки программирования C#, JavaScript, платформа ASP.NET Web API 2, СУБД MSSQL, библиотека React, инструмент управления состояниями Redux, технология доступа к данным Entity Framework.

Разработана концепция игрового сюжета, стратегия развития персонажей, выработаны правила оценки достижений игрока. Спроектирована модель базы данных приложения. Представлены макеты интерфейса основных страниц приложения и также схема реализации игрового процесса и варианты взаимодействия игрового приложения и его пользователей.

ГЛАВА 3

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ «FAMILY ADVENTURE WORLD»

3.1 Распределение уровней реализации по технологиям

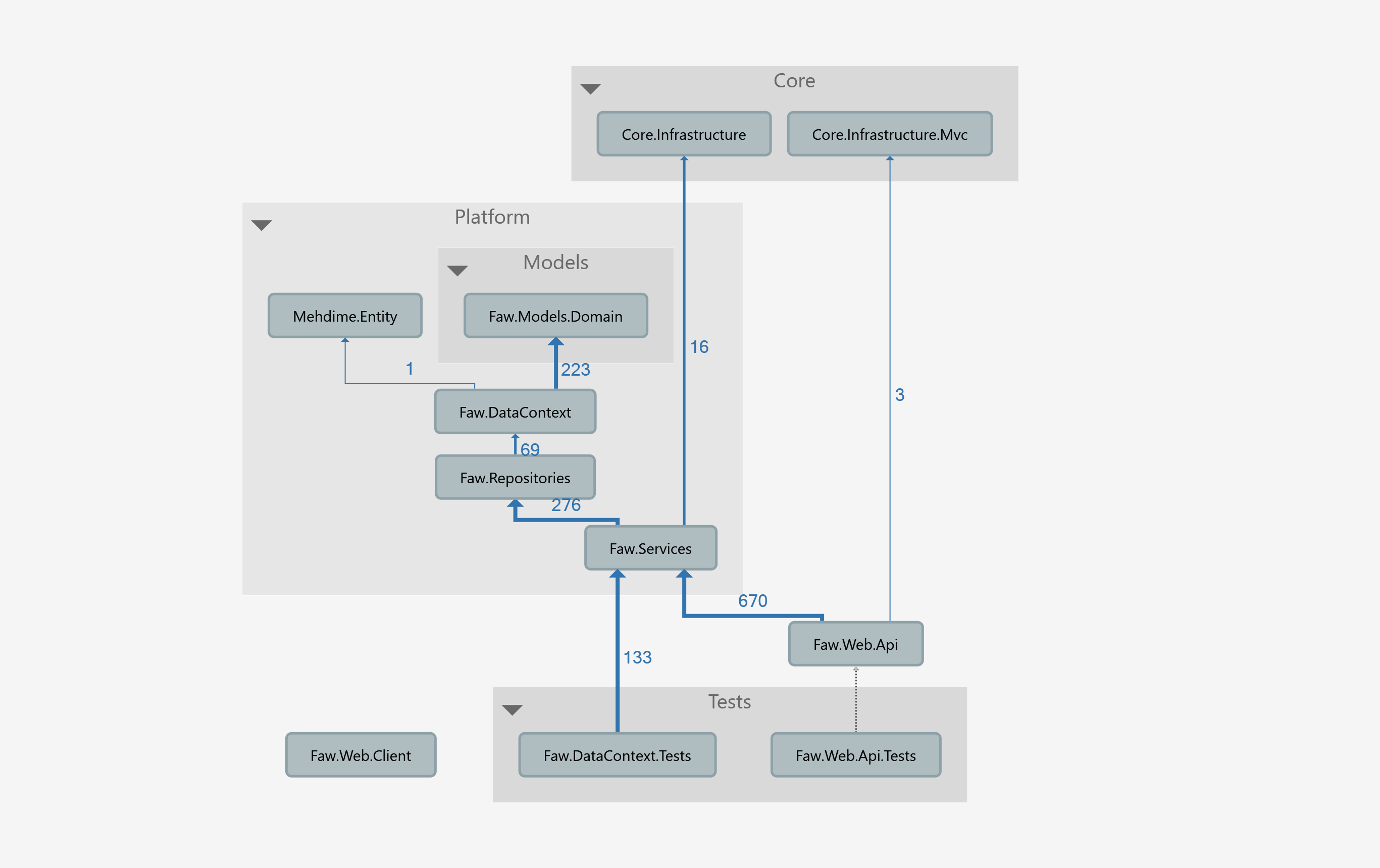
Реализация системы многомодульной структуры требует в первую очередь распределения уровней реализации в зависимости от требований, наложенных на них и, конечно же, актуальности технологий, которые можно использовать для осуществления поставленных целей. Выполним распределения уровней реализации по используемым 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 моделей 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 (рисунок 3.1).



**Рисунок** 3.1. Диаграмма связанности компонентов системы

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 [2].

В приложении 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 (см. Приложение Б, Листинг 3.1).

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 возвращает уже созданный объект. Однако IoC-контейнер может возвращать только зарегистрированные объекты (Листинг 3.2).

Листинг 3.2 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 3.3).

Листинг 3.3 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 [3].

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

Для связи с базой данных использовался 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, то за основу для всех DTO-моделей была выделена абстрактная модель BaseEntity (Листинг 3.4), которая содержит в себе одно приватное поле \_entityId. Это поле 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 (Приложение В, Листинг 3.4).

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 (Листинг 3.5). В базовом классе описан маппинг первичного ключа.

**Листинг 3.5 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 (см. Приложение Б, Листинг 3.6), унаследованным от DbContext, свойства которого позволяют работать с объектами базы данных, используя LINQ to Entities.

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 общих для всех репозиториев методов:

* 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, то был разработан обобщённый абстрактный класс, который реализует все данные методы (см. Приложение Б, Листинг 3.7). Таким образом, любой репозиторий, реализуя данный класс получают базовый набор функций, а также доступ к контексту [4].

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 области. 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 осуществляет аутентификацию пользователя в системе;
* void Edit(User user) – метод, который используется для обновления информации о пользователе.

3.5 Реализация конструктора игровых 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 получает 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 данные, 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 и для того, чтобы данное выражение можно было редактировать и восстанавливать, была разработана базы данных (рисунок 3.2).



**Рисунок** 3.2. Концептуальная модель БД выражений

Так как выражение может быть сложным и состоять из нескольких условий, то между таблицами Achievement и ExpressionProperty установлена связь “один ко многим”. Каждая запись ExpressionProperty хранит в себе название модели (ModelName). Это позволяет создавать условия, используя 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 между выражениями. 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 о значении или названии свойства модели, а также ее типа. Каждое выражение ExpressionProperty содержит в себе ссылки на две PropertyValue для левой и правой частей выражения.

Для компиляции данных в выражения, которые будут использованы для проверки выполнения достижения, был выделен интерфейс IConditionExpressionBuilder (Листинг 3.8).

Листинг 3.8 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 методов: для генерации сложного выражения и одинарного (см. Приложение Б, Листинг 3.9).

Как видно из кода метод Build принимает 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.

Важным 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 того, что данное условие выполняется для конкретного пользователя или пользователей.

3.6 Реализация клиентского приложения

В современных реалиях красивый дизайн и удобный интерфейс является важной частью разработки. Дизайн сайта был выполнен в стиле Material Design. Для этого использовалась библиотека стилей Material Design Light.

Для реализации приложения была выбрана библиотека React в связке с Redux. Такая связка позволила разбивать приложения на компоненты, которые напрямую не зависят друг от друга, но любой компонент может подписаться на событие изменения данных и обновить свое представление. К примеру, вывод эффекта загрузки в момент, когда данные со списком заданий для пользователя загружаются с сервера (Листинг 3.10, рисунок 3.4).

**Листинг 3.10 Выбор компонента для отображения на основе информации о процессе загрузки данных с сервера**

<Cell col={12} className='-quest-list-container'>

{

props.userQuestsLoading

? (

<Loading />

)

: (

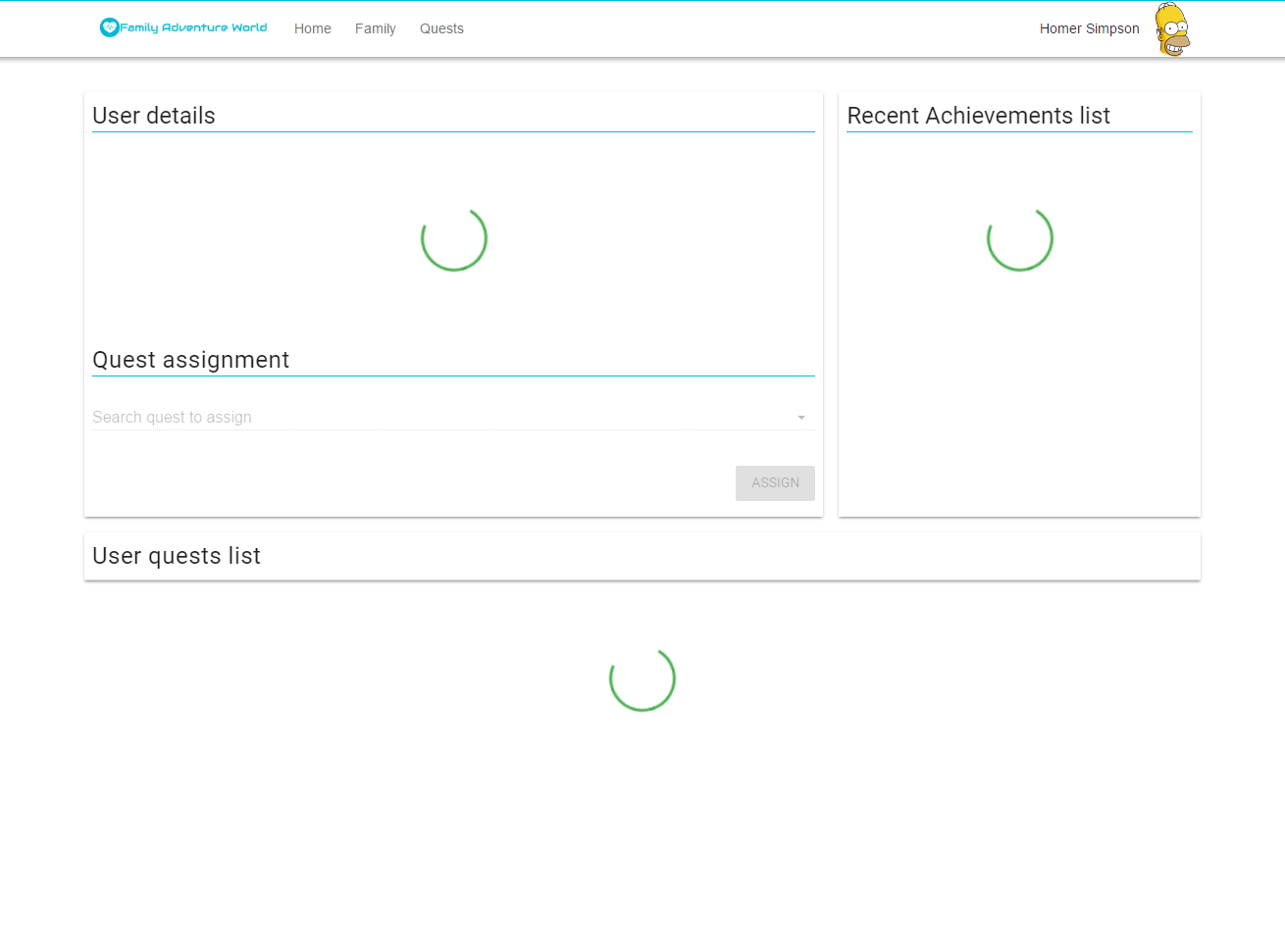
<MemberQuestList userQuests={props.userQuests} />

)

}

</Cell>

Так как все элементы приложения разбиты на компоненты, то и взаимодействие с приложением может быть разделено, к примеру, данные о пользователе загружаются быстро и работать с ними (редактировать) уже можно. В то же время информация о достижениях загружается дольше, но эта информация никак не влияет на данные о пользователе и компонент, который отвечает за манипуляцию и отображения этих данных. Таким образом вся страница не блокируется и компоненты, которые уже загрузились, позволяют работать с ними, не дожидаясь загрузки всех компонентов.



**Рисунок** 3.4. Эффект загрузки во время загрузки данных с сервера

Так как достаточно большую часть работы с приложением составляет заполнение информации, для чего нужны формы, то было решено использовать библиотеку react-redux-form. Это очень мощная библиотека, которая сразу написана с учетом связки React и Redux. Внутри себя каждая форма хранит состояние и, подключив эту форму к своему хранилищу, в любой момент можно получить состояние данной формы. Так же при помощи набора методов можно манипулировать состоянием не только через UI, а также и программно.

Для ограничения доступа пользователей к страницам, которые не доступны ему в виду ограничений, был разработан метод requireAuthorization (см. Приложение Б, Листинг 3.11).

Данный метод используется для каждой страницы в системе, в качестве параметров он принимает сам компонент, для которого используется, а также названия прав, которые нужно иметь, чтобы получить доступ к данной странице. В случае если пользователь не авторизован или у него не хватает прав чтобы зайти на данную страницу, он будет автоматически перенаправлен на страницу авторизации.

3.7 Выводы по главе 3

На данном этапе разработки было разработано браузерное игровое приложение «Семейный мир приключений», в рамках которого реализованы следующие компоненты:

* модуль управления личными данными пользователя;
* модуль управления семьей пользователя;
* модуль управления заданиями пользователя;
* модули просмотра информации о члене семьи с возможностью назначать задания;
* модуль управления игровыми достижениями;

Веб-приложение реализовано на базе трехуровневой архитектуры с использованием технологий .Net, JavaScript, React, Redux, Material Design, Scss, Entity Framework, Axios, React-mdl, Lodash.

ЗАКЛЮЧЕНИЕ

В результате работы над дипломным проектом разработано социально-ориентированное игровое приложение «Семейный мир приключений», которое вовлекает членов семей во взаимодействие между собой в игровой форме в процессе создания, назначения и выполнения заданий.

Для достижения поставленной цели был выполнен обзор многопользовательских игровых приложений для детей и их родителей, выделен отдельный жанр семейных игр. В духе этого жанра разработан сюжет семейной игры, выделены персонажи, предложена стратегия развития персонажей, а также система оценки игровых достижений.

Сюжет игры был реализован в виде веб-приложения, пользователи которого могут создавать семьи, добавлять членов семей, назначать друг другу задания, делиться достижениями, зарабатывать опыт и вознаграждения. Таким образом, есть надежда, что разработанное игровое приложение берет на себя социальную функцию сплочения детей и их родителей на пути достижения общих целей.

Результаты работы над проектом были представлены на конференциях:

* Деменович, А.Ю. Реализация социального аспекта в разработке игрового веб-приложения «Семейный мир приключений» / А. Ю. Деменович, А.В. Карканица // Программное обеспечение информационных технологий : 53-я Научная конференция аспирантов, магистрантов и студентов БГУИР, Минск, 02-06 мая 2017 г.
* Деменович, А.Ю. Игровое веб-приложение «Семейный мир приключений» как воспитательный и образовательный инструмент / А. Ю. Деменович, А.В. Карканица // Психолого-педагогические проблемы обучения и воспитания школьников и студентов : XХIV Международная студенческая научно-практическая конференция «От идеи – к инновации», Мозырь, 27 апреля 2017 г.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

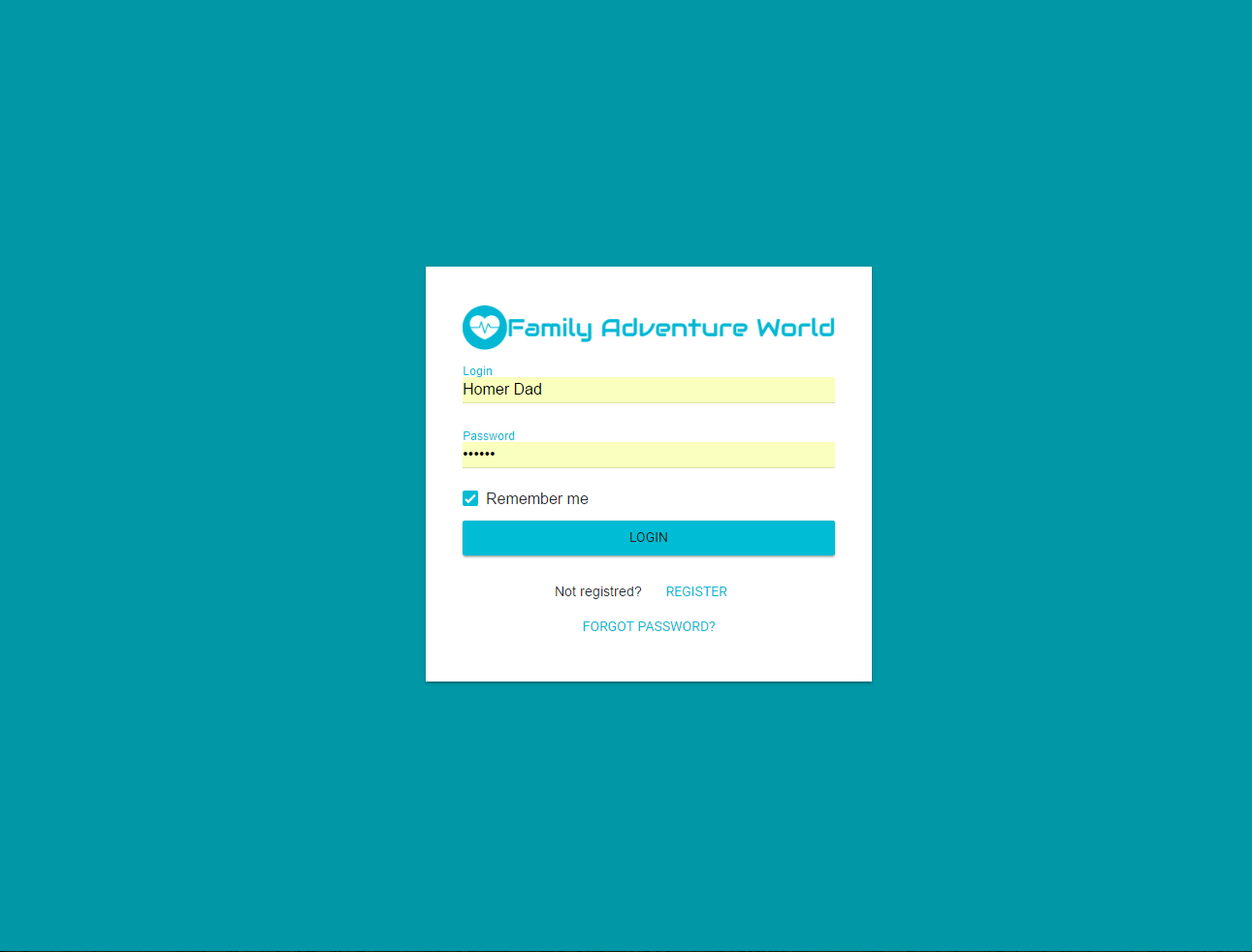


Рисунок А 1 – Страница авторизации

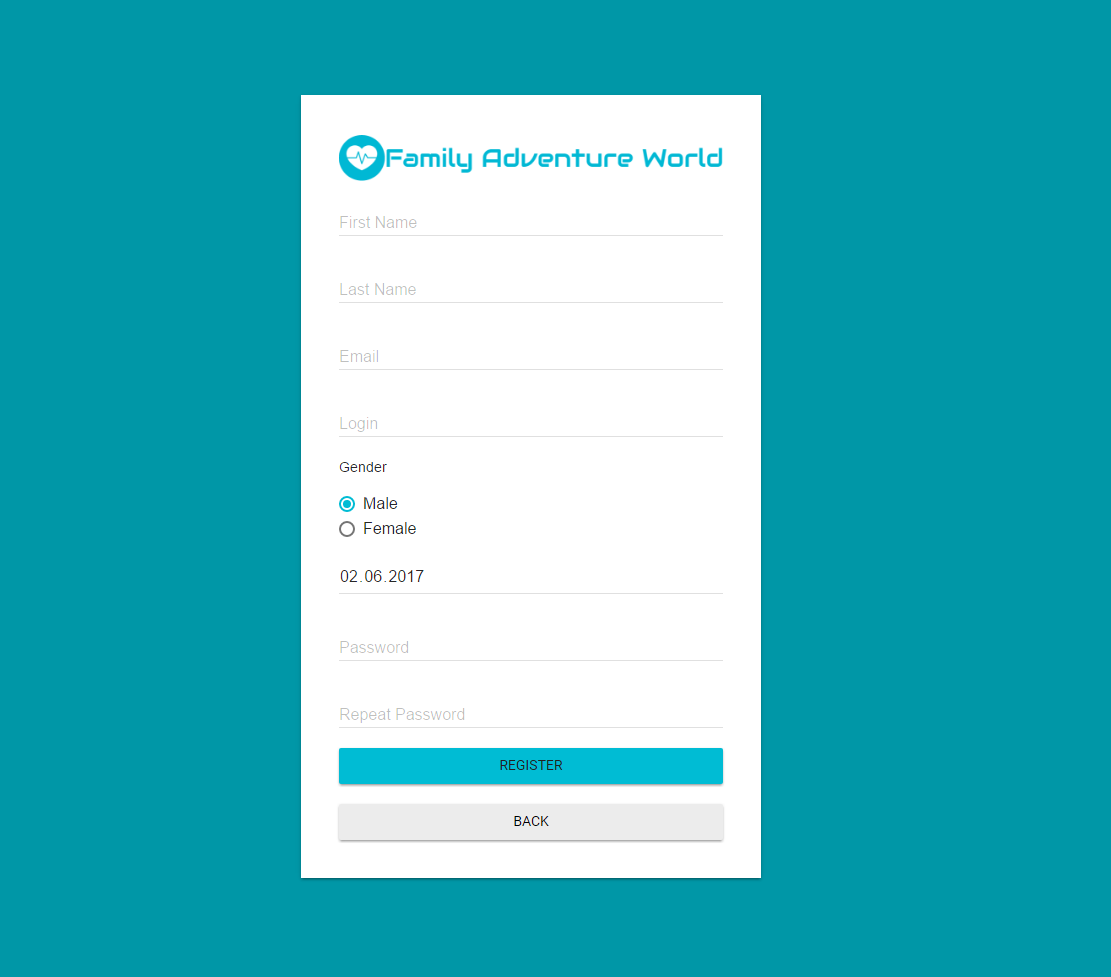


Рисунок А 2 – Страница регистрации

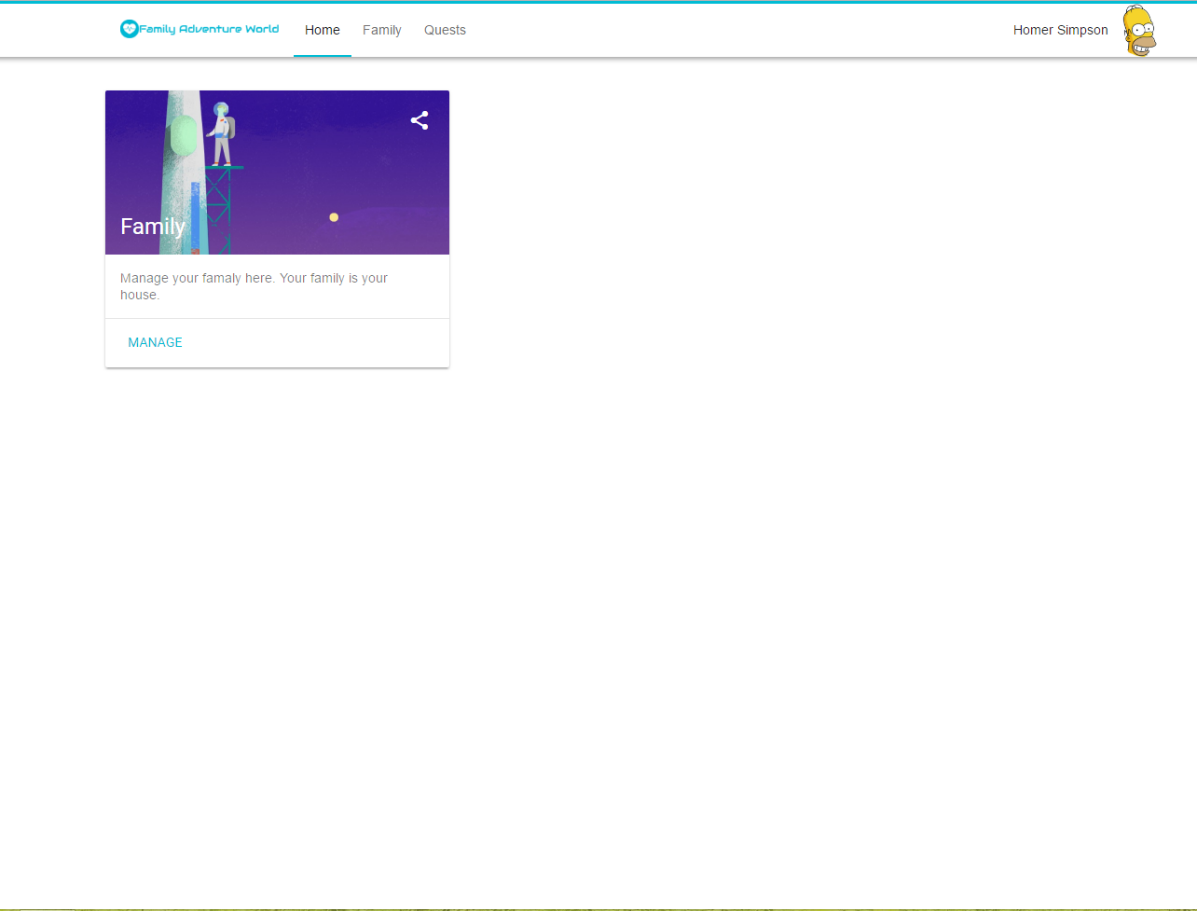


Рисунок А 3 – Главная страница сайта

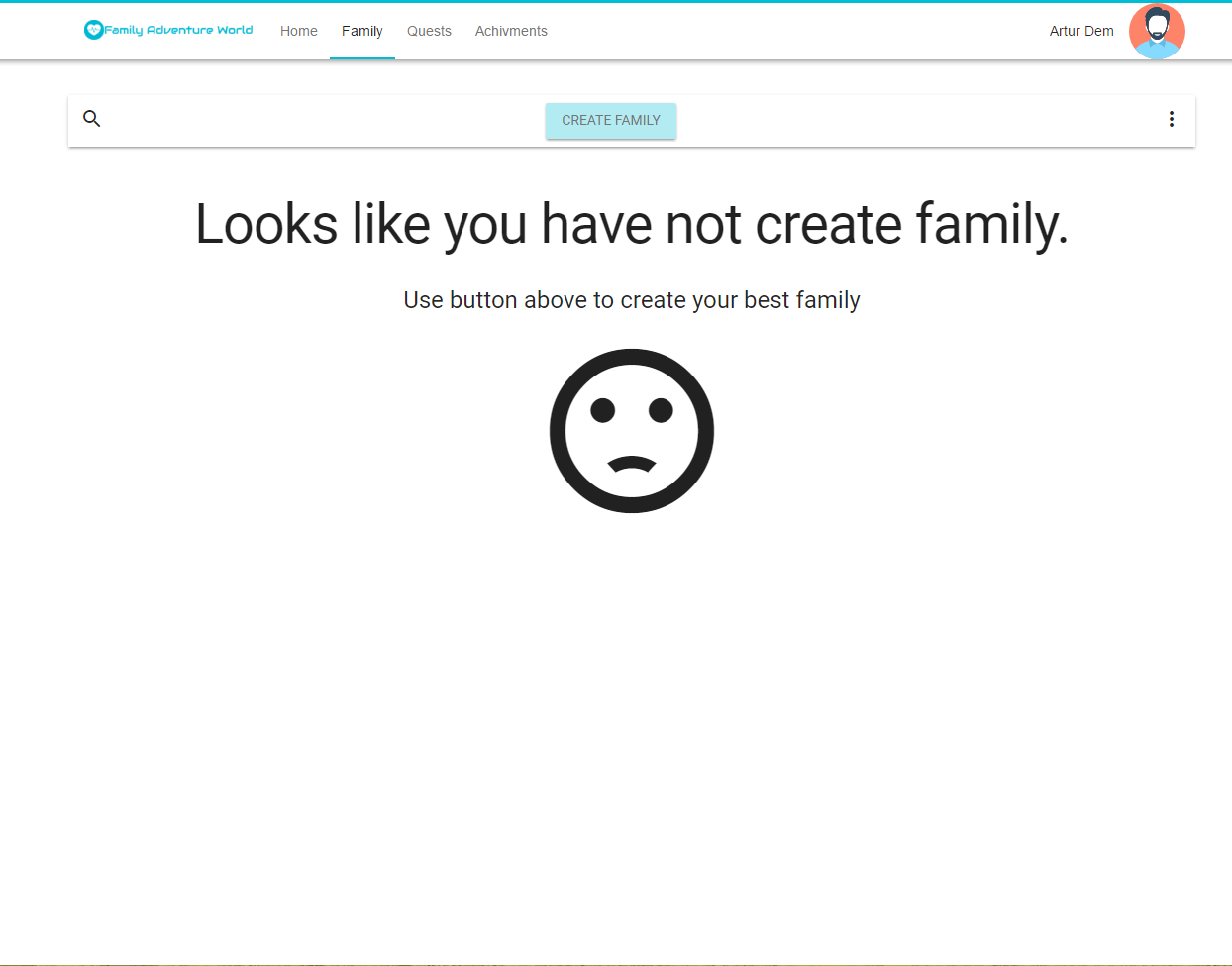


Рисунок А 4 – Страница семьи (семья не создана)

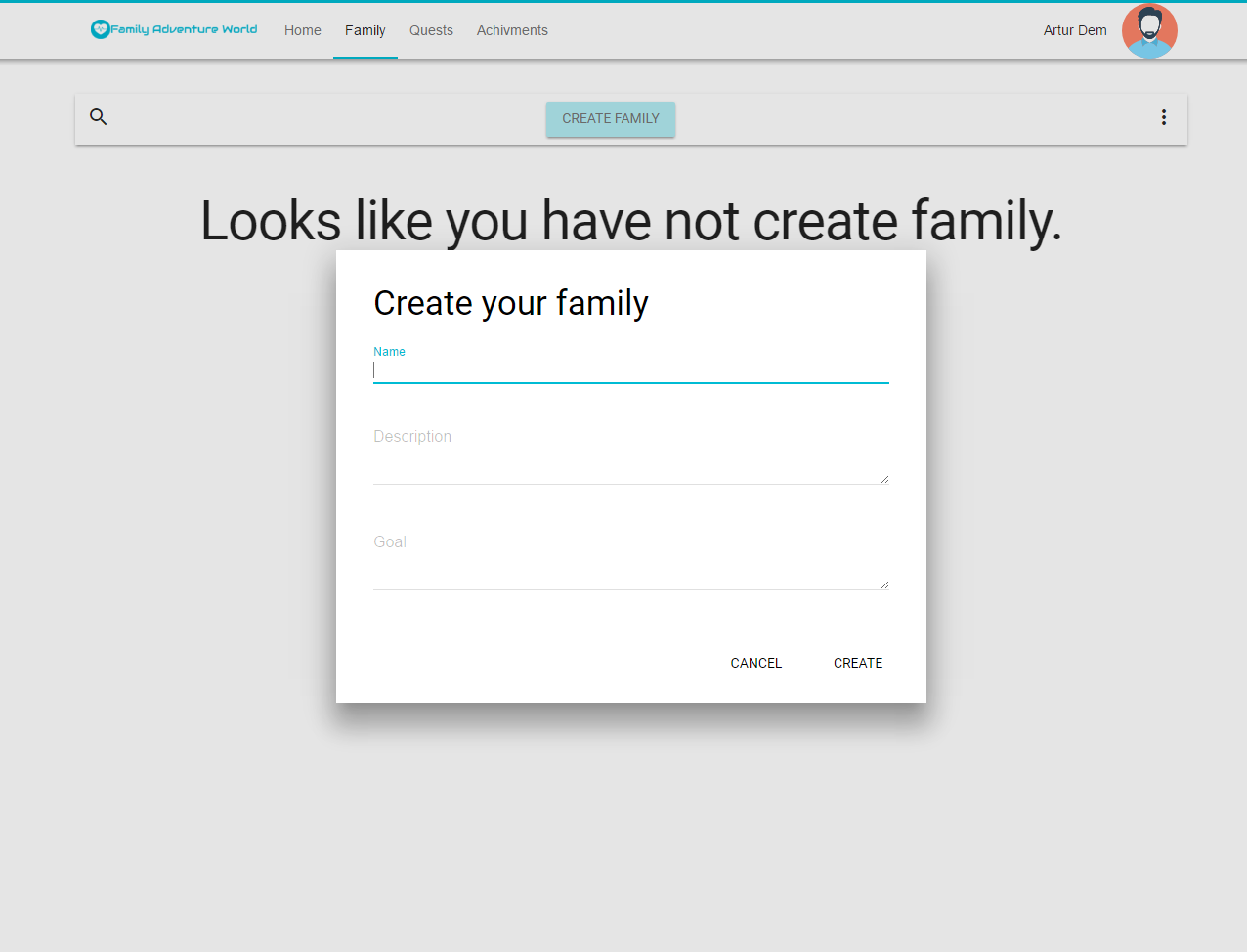


Рисунок А 5 – Диалог создания семьи

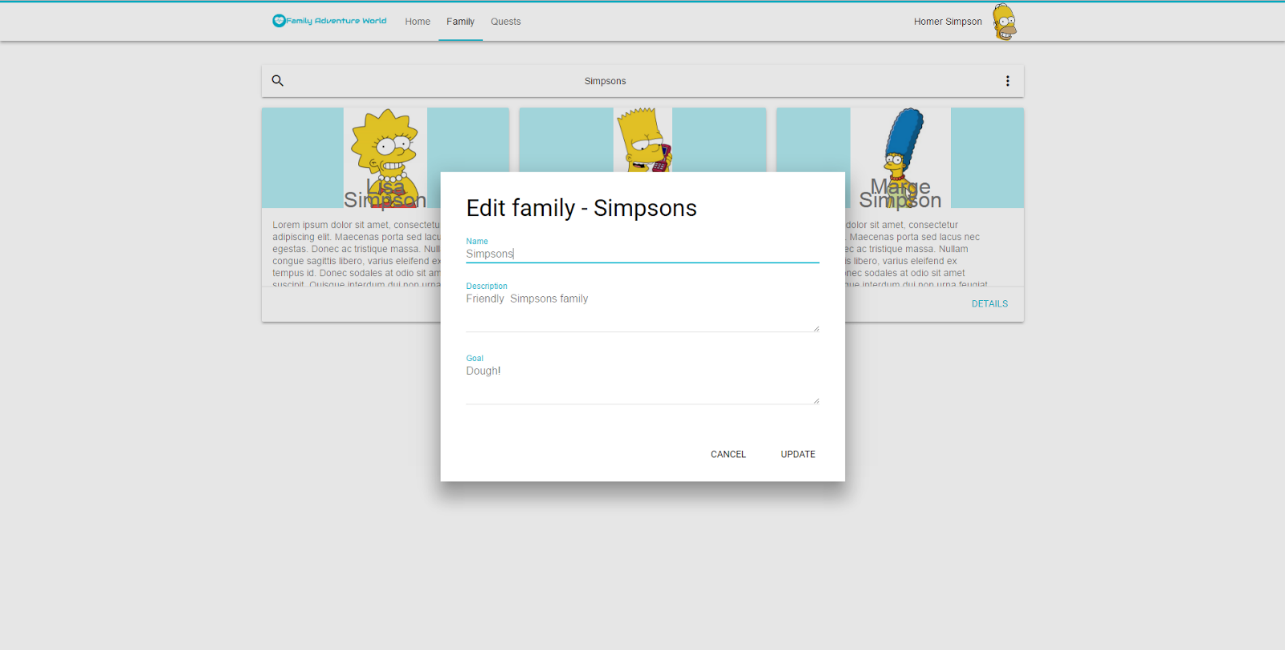


Рисунок А 6 – Диалог редактирования семьи

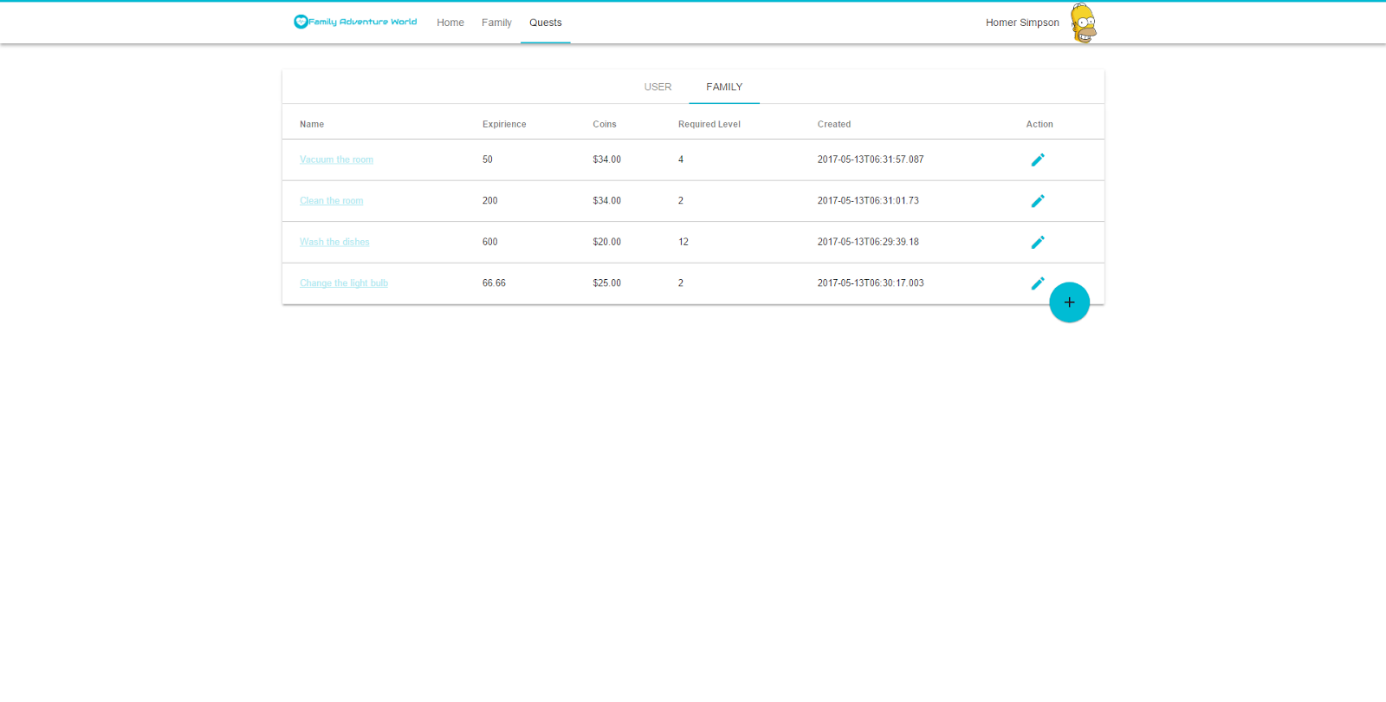


Рисунок А 7 – Страница просмотра заданий

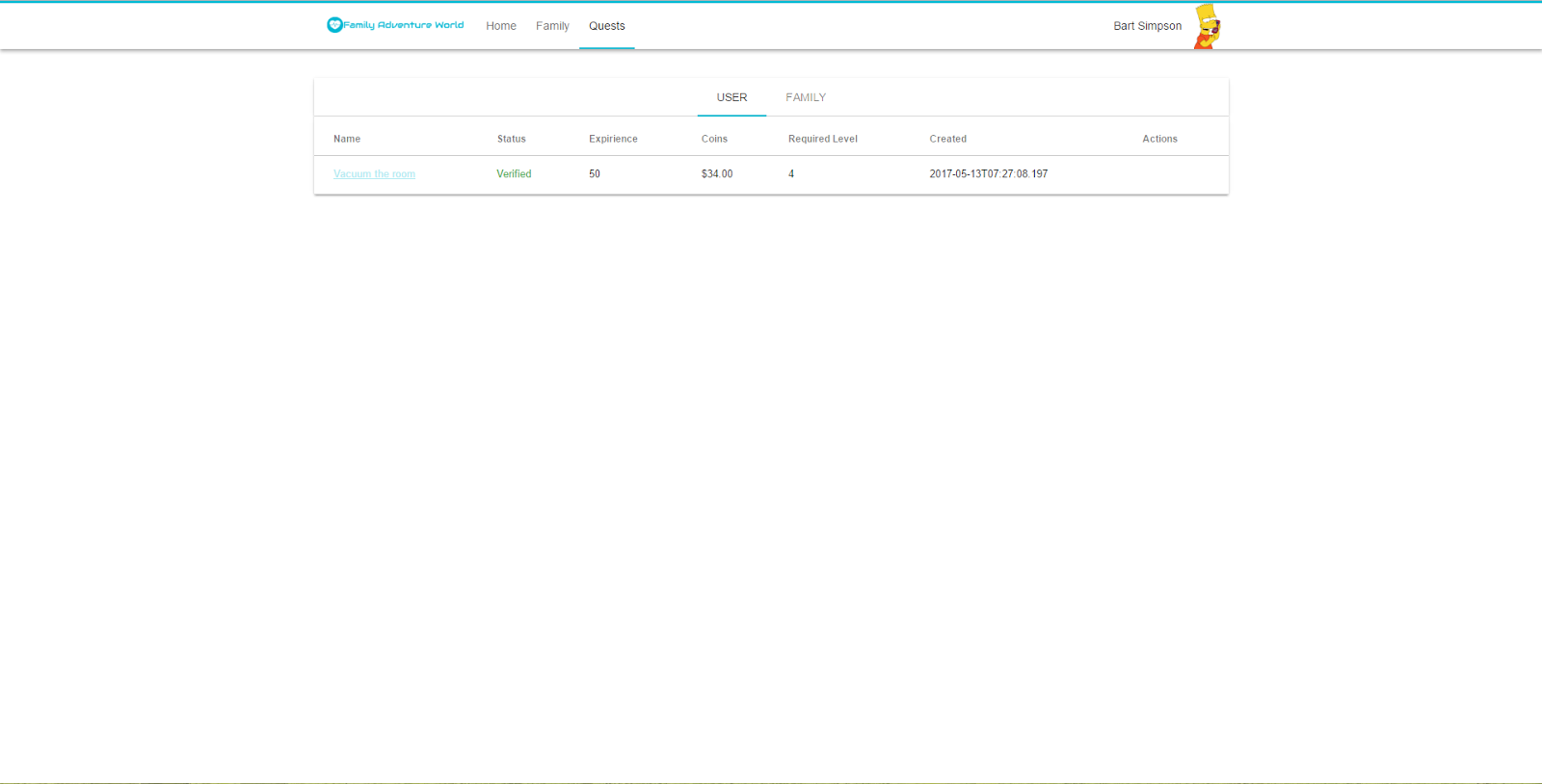


Рисунок А 8 – Страница просмотра пользовательских заданий

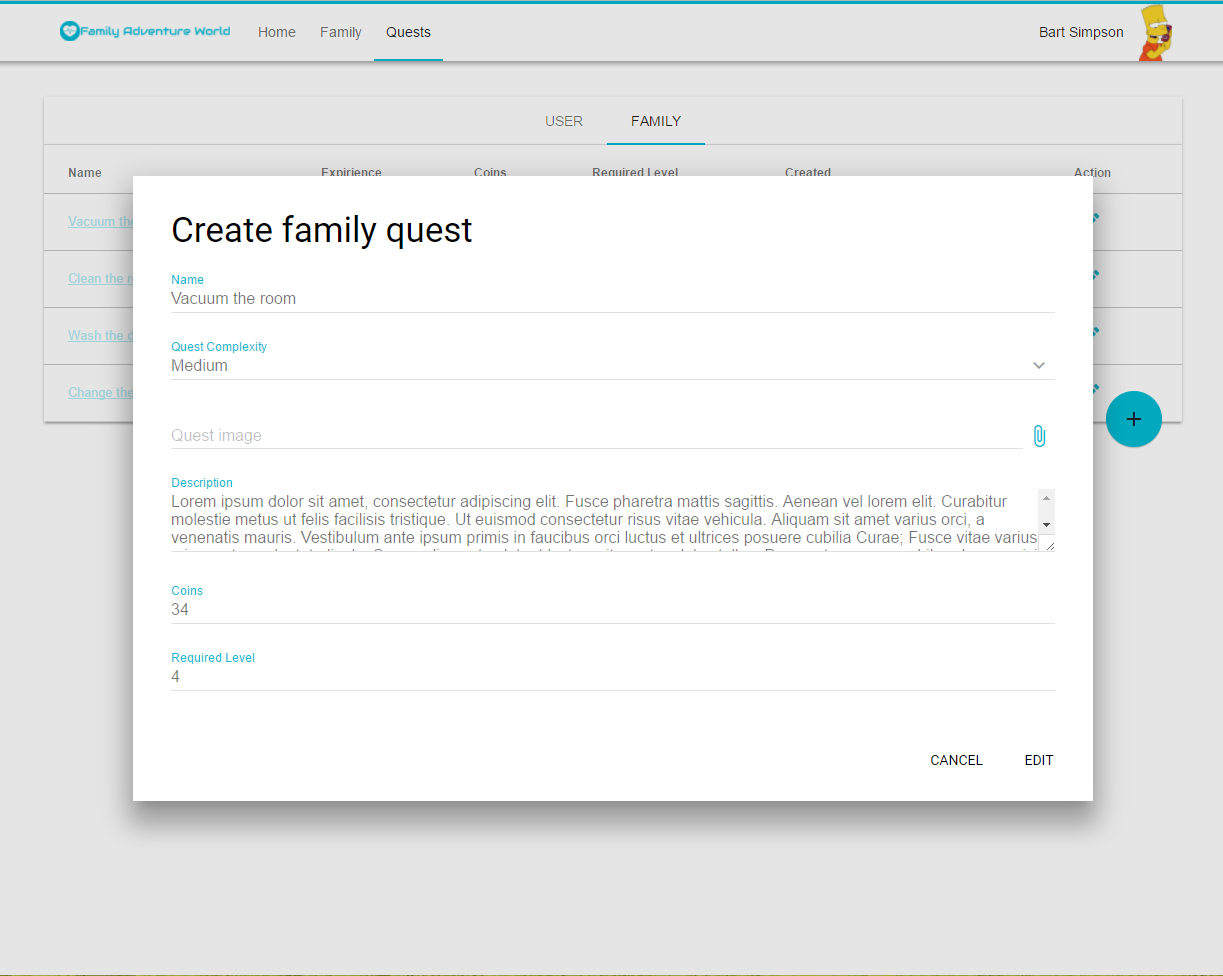


Рисунок А 9 – Диалог создания/редактирования задания

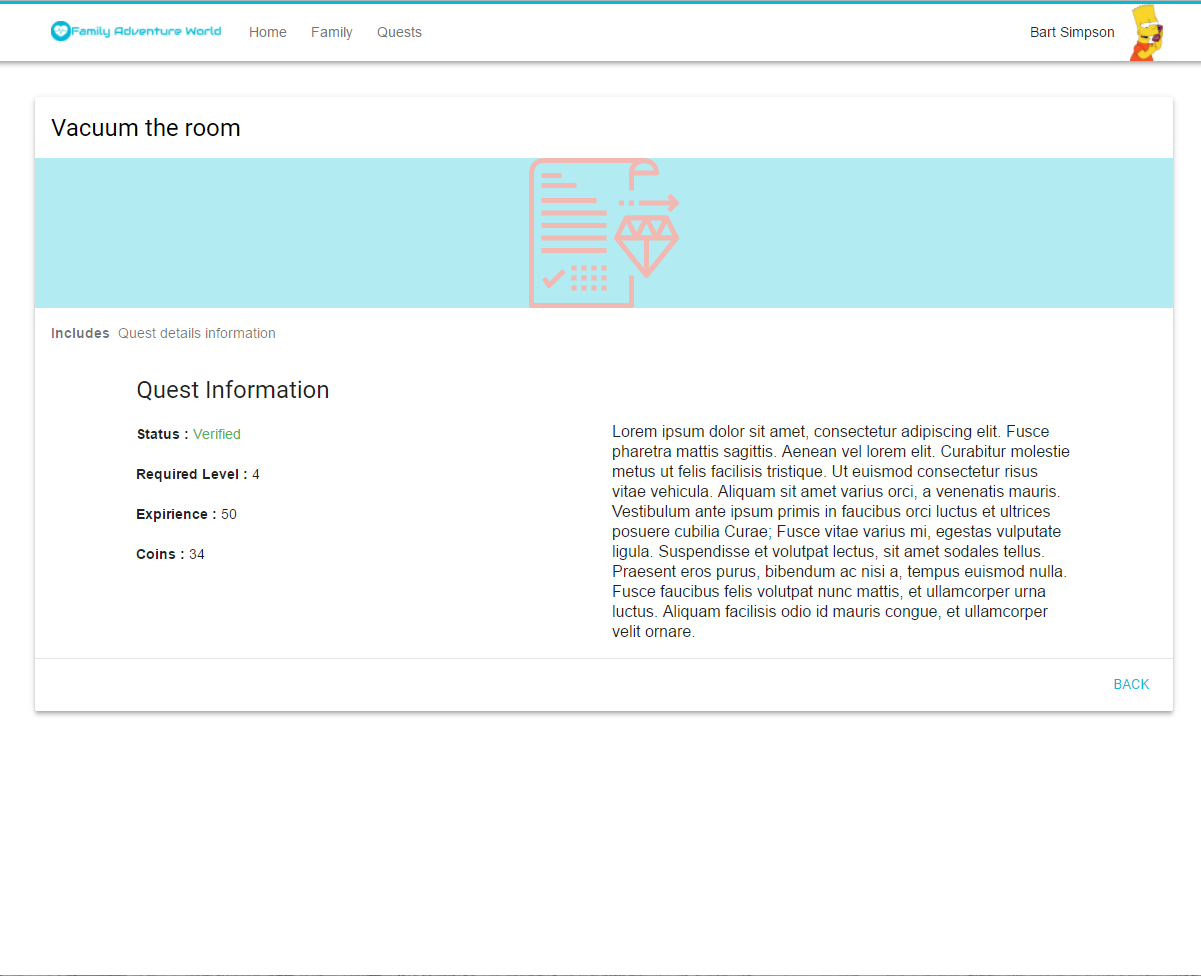


Рисунок А 10 – Страница информации о задании

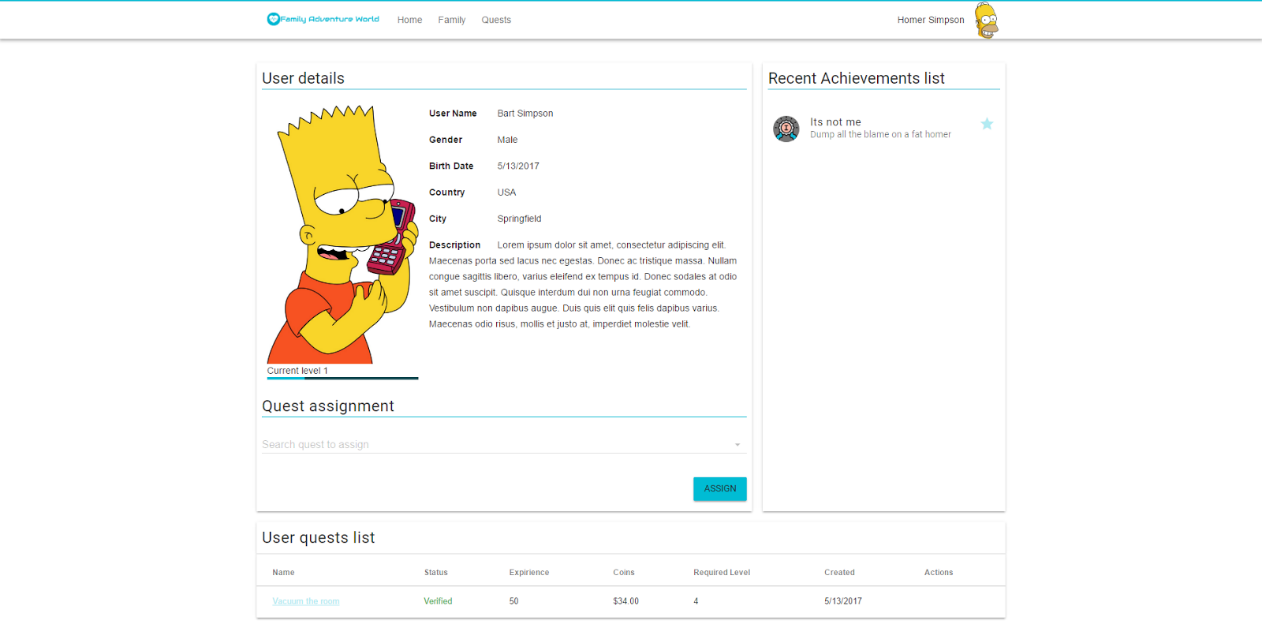


Рисунок А 11 – Страница просмотра информации о члене семьи

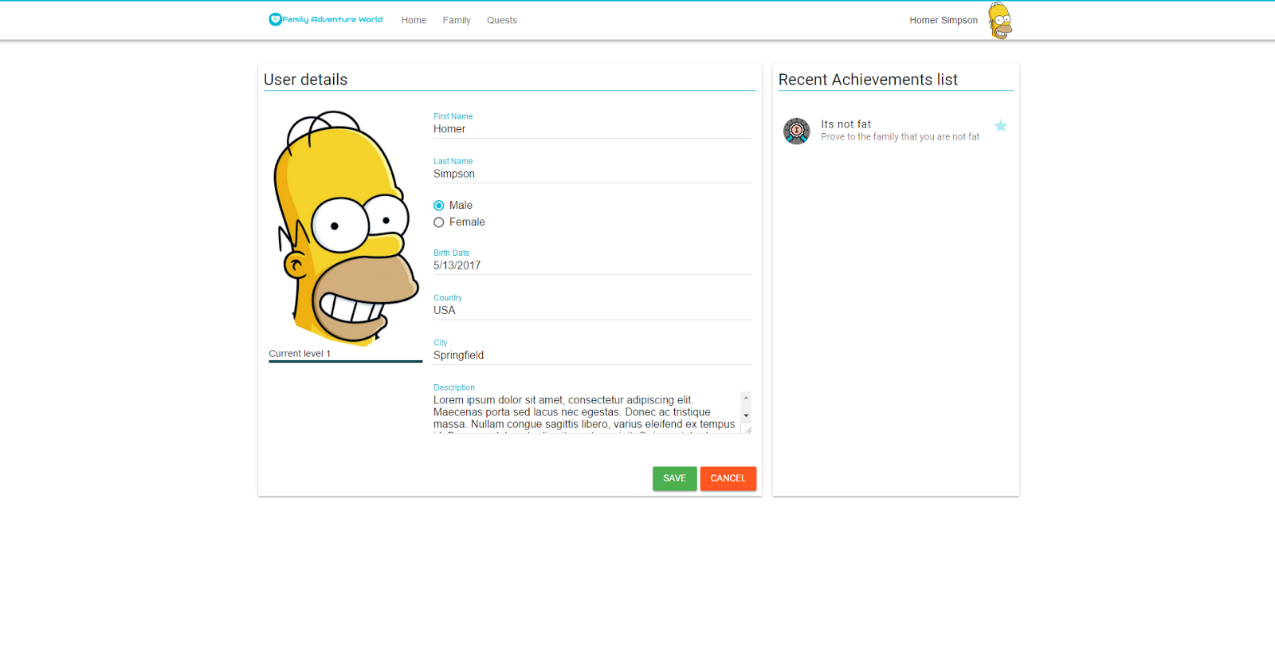


Рисунок А 12 – Страница редактирования профиля

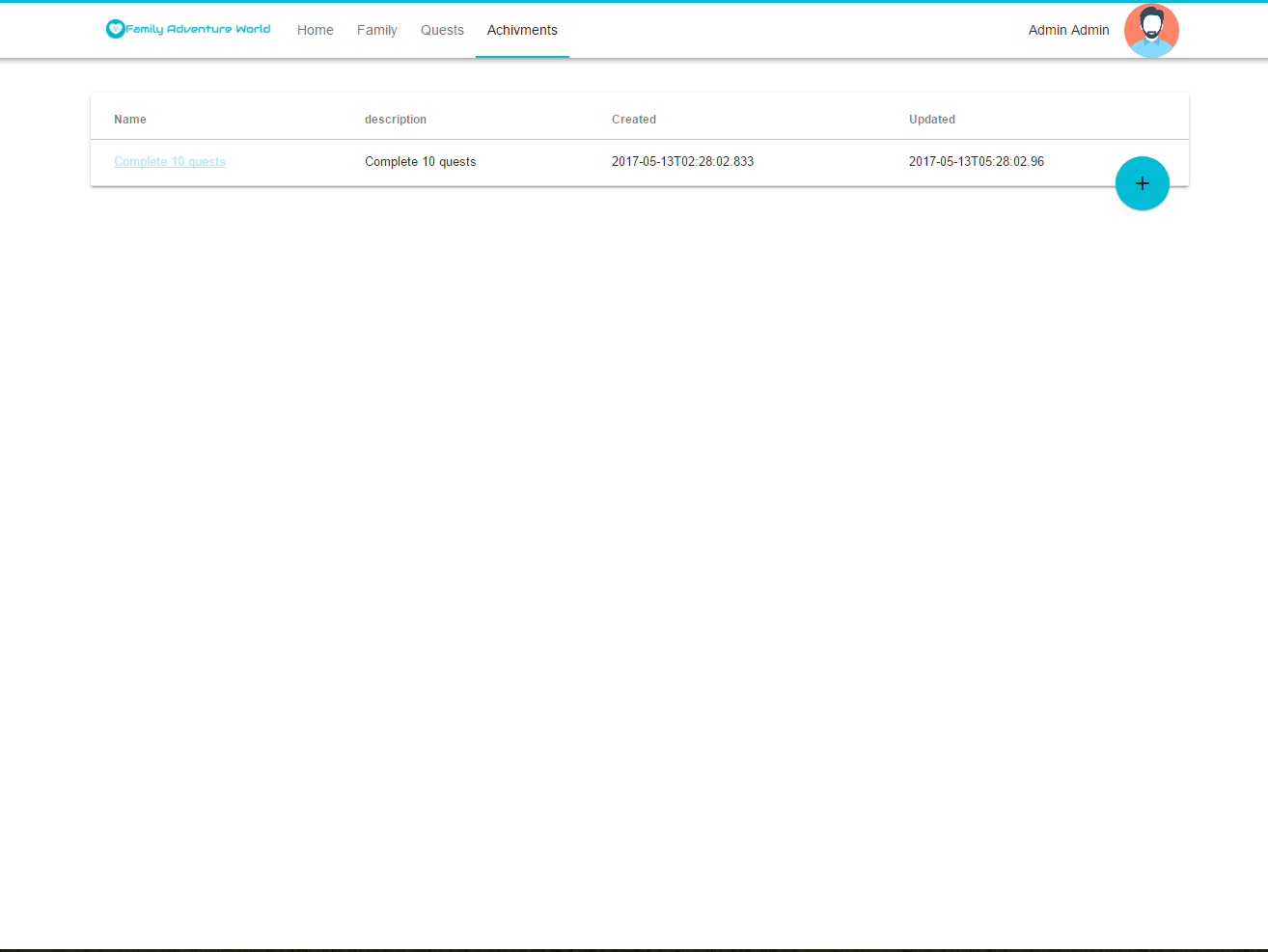


Рисунок А 13 – Страница просмотра достижений

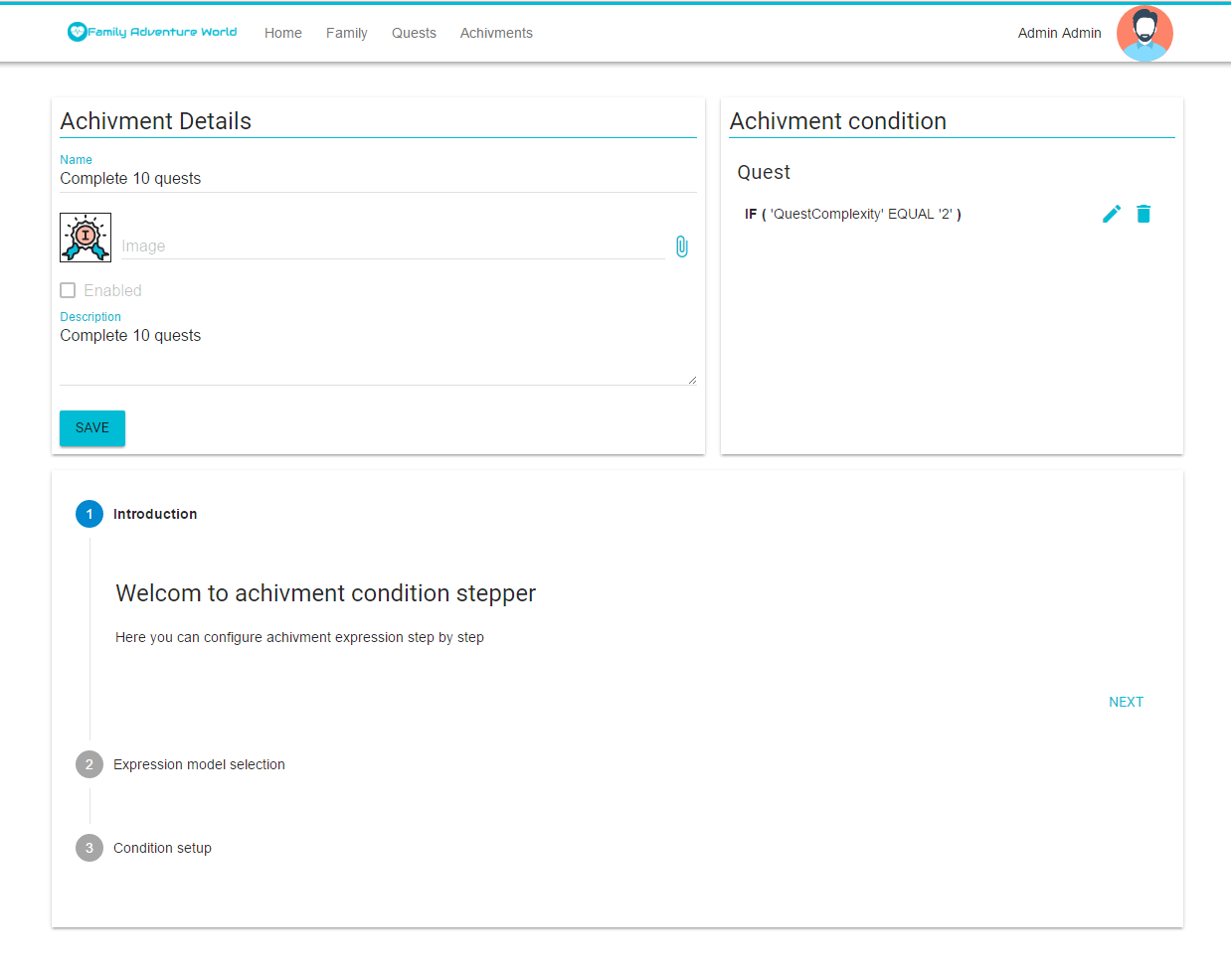


Рисунок А 14 – Страница редактирования достижений

ПРИЛОЖЕНИЕ Б

Листинги программы

**Листинг 3.1 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;

}

}

Листинг 3.4 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

}

Листинг 3.6 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());

}

}

Листинг 3.7 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);

}

}

Листинг 3.9 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);

}

}

Листинг 3.11 Реализация метода requireAuthorization.

import React from 'react'

import { connect } from 'react-redux'

import { browserHistory } from 'react-router'

export default function requireAuthorization (Component, allowedClaims) {

class AuthenticatedComponent extends React.Component {

checkAuth (isAuthenticated, claims) {

return isAuthenticated && (!allowedClaims || allowedClaims.some(r => claims.indexOf(r) !== -1))

}

componentWillMount () {

if (!this.checkAuth(this.props.isAuthenticated, this.props.claims)) {

browserHistory.push('/account/Login')

}

}

render () {

return (<Component {...this.props} />)

}

}

const mapStateToProps = (state) => ({

claims: state.user.credentials ? state.user.credentials.claims : [],

isAuthenticated: state.user.isAuthenticated

})

return connect(mapStateToProps)(AuthenticatedComponent)

}