



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ (ИУ)

КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

***Разработка базы данных для хранения и обработки
данных приложения для отслеживания времени
прохождения игр***

Студент группы ИУ7-64Б

Золотухин А. В.

Руководитель курсовой работы

Павельев А. А.

2023 г.

РЕФЕРАТ

Курсовая работа представляет собой реализацию базы данных, позволяющую собирать у пользователей информацию об играх: время прохождения игры (время прохождения сюжета, время 100% прохождения), отзывы и оценки — а также приложение, предоставляющее интерфейс для доступа к базе данных. Готовый продукт позволяет просматривать, изменять, добавлять и удалять данные из базы.

В качестве системы управления базой данных используется Sql Server, которая подключается к приложению, реализованному на языке программирования C#.

Ключевые слова: база данных, Sql Server, система управления базами данных, интерфейс.

Расчетно-пояснительная записка к курсовой работе содержит 42 страницы, 21 иллюстраций, 3 таблицы, 11 источников.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 АНАЛИТИЧЕСКАЯ ЧАСТЬ	6
1.1 Формализация задачи	6
1.2 Типы пользователей и доступная им функциональность	7
1.3 Анализ существующих аналогов	10
1.4 Анализ существующих видов баз данных	12
1.4.1 Кэширование данных	13
2 КОНСТРУКТОРСКАЯ ЧАСТЬ	15
2.1 Описание таблиц разрабатываемой базы данных	15
2.2 Схема алгоритма преобразования строки в xml формат	18
2.3 Диаграмма классов приложения	19
3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ	21
3.1 Выбор системы управления базой данных	21
3.2 Выбор языка программирования	22
3.3 Реализация ролевой модели	22
3.4 Реализация процедуры преобразования строки в xml формат и вставки в таблицу	23
3.5 Реализация модели «Отзыв»	24
3.6 Демонстрация интерфейса приложения	25
3.7 Тестирование приложения	34
4 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ	35
4.1 Технические характеристики устройства	35
4.2 Описание исследования	35
4.3 Результаты исследования	36
4.4 Вывод	37
ЗАКЛЮЧЕНИЕ	39
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	40

ВВЕДЕНИЕ

Легко проверить, сколько времени уходит на просмотр фильма или телепередачи, так почему бы не сделать то же самое и с видеоиграми? Никто не хочет покупать за немаленькие деньги слишком короткую игру или 200-часовую RPG, на которую нет столько времени.

Было бы неплохо, если бы существовало такое приложение или сайт, на котором люди могли бы посмотреть примерное время прохождения игры.

Цель курсовой работы — разработать базу данных, в которой можно посмотреть информацию о статистике времени прохождения игр.

Для достижения поставленной цели необходимо решить следующие задачи:

- анализ существующих решений;
- формализация задачи и данных;
- анализ существующих способов хранения данных;
- анализ существующих систем управления базами данных;
- реализация спроектированной базы данных;
- реализация методов и классов, которые обеспечат доступ к базе данных;
- исследование зависимости преобразования данных от объема и от реализации алгоритма.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

В данном разделе будут формализованы задача и данные. Будет разработана ER-диаграмма разрабатываемой системы. Будут проанализированы существующие аналоги. Будут рассмотрены типы пользователей и функционал, который будет им доступен. Будут проанализированы способы организации хранения данных, а также будет выбран наилучший подход к хранению данных для решения поставленной задачи.

1.1 Формализация задачи

Для решения поставленной цели необходимо спроектировать и реализовать базу данных для отображения статистической информации о времени прохождения игр и приложение, которое позволит получать доступ, изменять, добавлять и удалять данные. В базе данных необходимо реализовать три типа ролей: авторизованный пользователь, гость и администратор.

В базе данных должна храниться информация об: играх, отзывах об играх, временных отметках о времени прохождения игры, данные для авторизации пользователей, платформах запуска игр, связи между игрой и платформами, на которых запускается игра.

В таблице 1.1 представлены сущности базы данных и их атрибуты.

Таблица 1.1 – Категории и сведения о данных

Сущность	Атрибуты
Игры	ID игры, название игры, дата выхода, разработчик, издатель, жанры
Временная отметка	ID игры, имя пользователя, количество часов, количество минут, тип прохождения.
Отзыв	ID игры, имя пользователя, текст отзыва, оценка, дата публикации.
Пользователь	Имя пользователя, пароль.
Платформа	ID платформы, название, средняя стоимость
Связь платформ и игр	ID платформы, ID игры.

Согласно таблице 1.1, на рисунке 1.1 изображена ER-диаграмма базы данных.

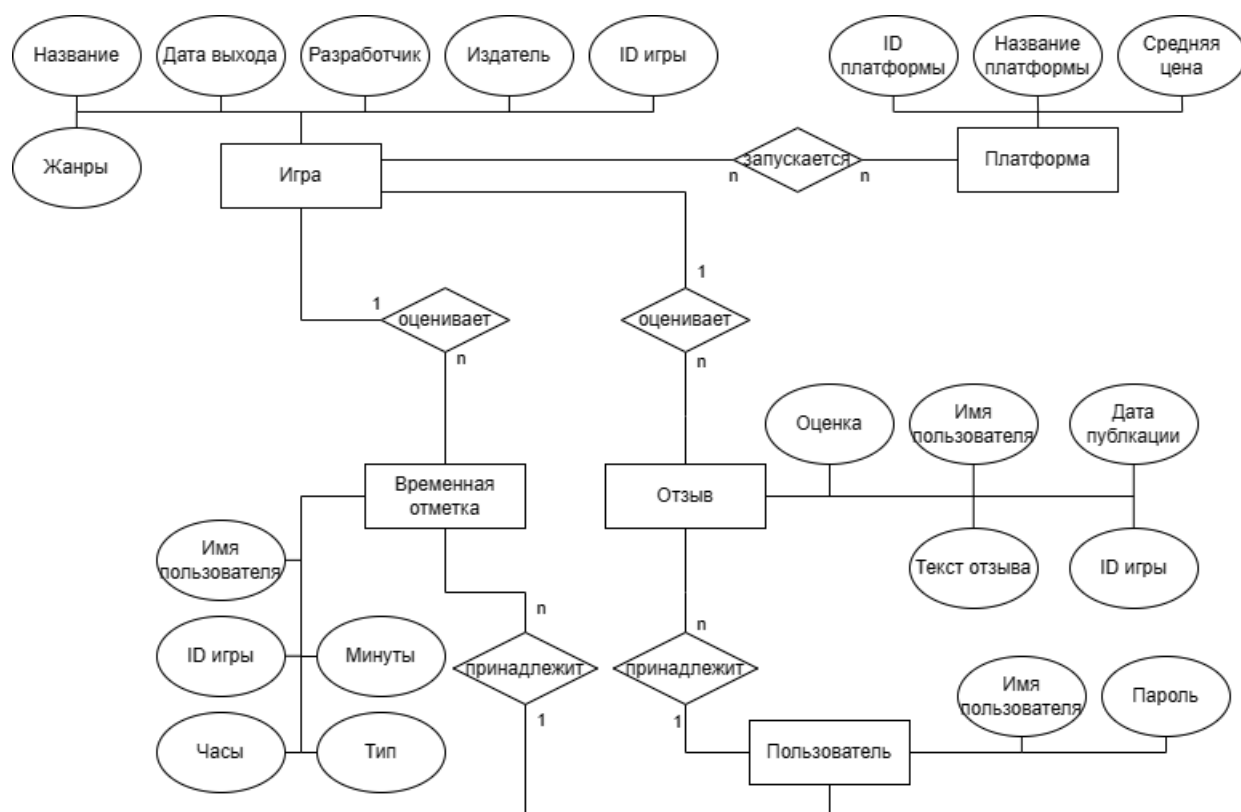


Рисунок 1.1 – ER-диаграмма

1.2 Типы пользователей и доступная им функциональность

Одной из задач курсовой работы является разработка приложения, которым будет возможность авторизации. Таким образом, необходимо разработать ролевую модель, состоящую из авторизованных пользователей, неавторизованных пользователей (гостей) и администраторов.

Каждый тип пользователя имеет свою специфическую функциональность, которая может быть описана с помощью диаграмм прецедентов (Use Case диаграммы).

Набор функций, которые могут быть использованы неавторизованными пользователями:

- аутентификация,
- регистрация,

- просмотр различной статистики о времени прохождения игры по разным категориям,
- просмотр отзывов и оценок,
- просмотр информации об игре.

На рисунке 1.2 представлена Use case диаграмма для неавторизованного пользователя.



Рисунок 1.2 – Диаграмма прецедентов для неавторизованного пользователя

Набор функций, которые могут быть использованы авторизованными пользователями:

- просмотр информации об игре,
- просмотр различной статистики о времени прохождения игры по разным категориям,

- добавление отзывов и оценок,
- добавление информации о времени прохождения игры,
- просмотр отзывов и оценок,
- выход.

На рисунке 1.3 представлена Use case диаграмма для неавторизованного пользователя.

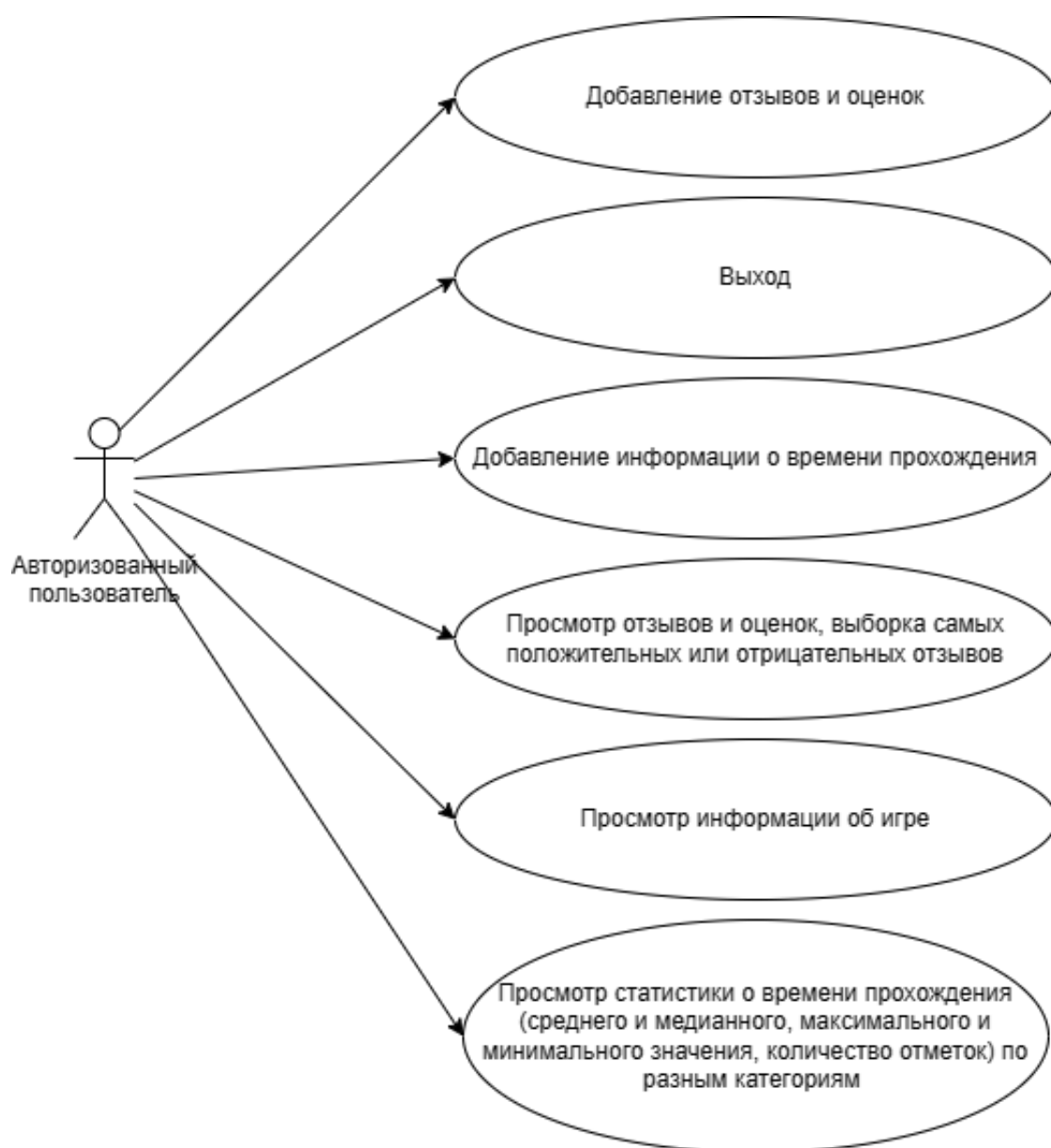


Рисунок 1.3 – Диаграмма прецедентов для авторизованного пользователя

Набор функций, которые могут быть использованы администратором:

- ВЫХОД,
- просмотр и изменение, добавление, удаление всей информации, доступной в базе данных.

На рисунке 1.4 представлена Use case диаграмма для администратора.

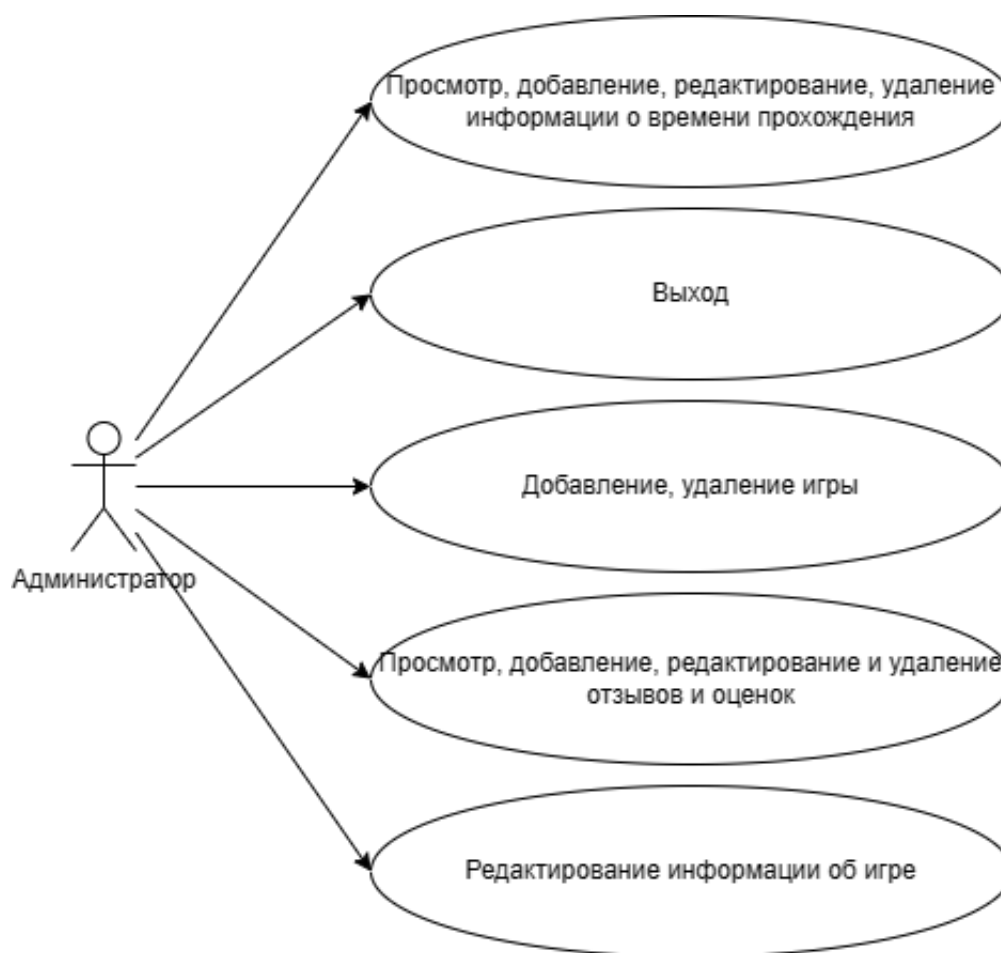


Рисунок 1.4 – Диаграмма прецедентов для администратора

1.3 Анализ существующих аналогов

Для того чтобы проанализировать существующие аналогичные решения, были определены следующие критерии:

1. Возможность указать время прохождения игры;
2. Возможность оставлять оценки и отзывы;

3. Наличие интерфейса на русском языке.

Steam [1] — один из самых популярных магазинов видео игр. Он позволяет не только покупать игры, но и оставлять обзоры на них. Также Steam считает время проведенное в игре, но не может собирать информацию времени прохождения (рисунок 1.5). Имеет русский интерфейс.

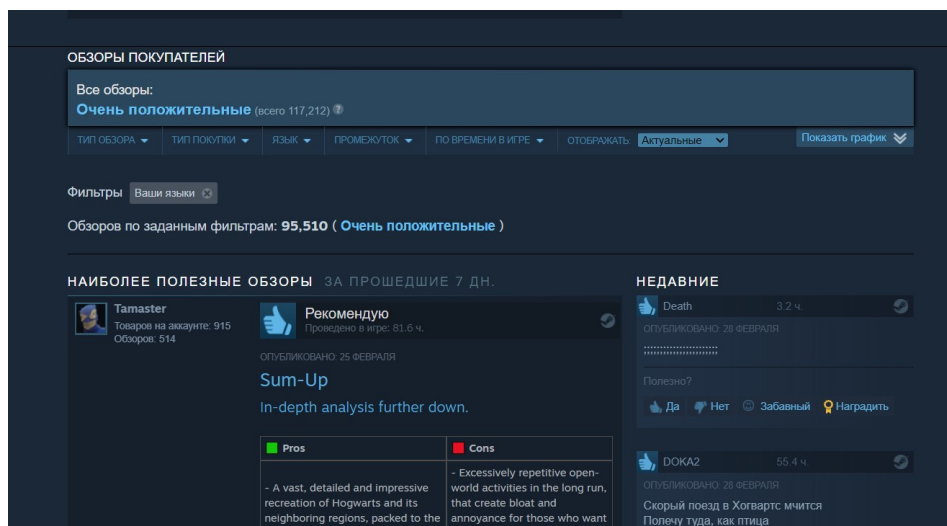


Рисунок 1.5 – Приложение Steam

HowLongToBeat [2] — сайт, на котором можно оставлять обзоры и оценки видеоигр, а также оставлять время прохождения игры по 2 категориям: основной сюжет, 100% прохождение (рисунок 1.6). Данный сайт имеет серьезный недостаток, он полностью на английском языке.

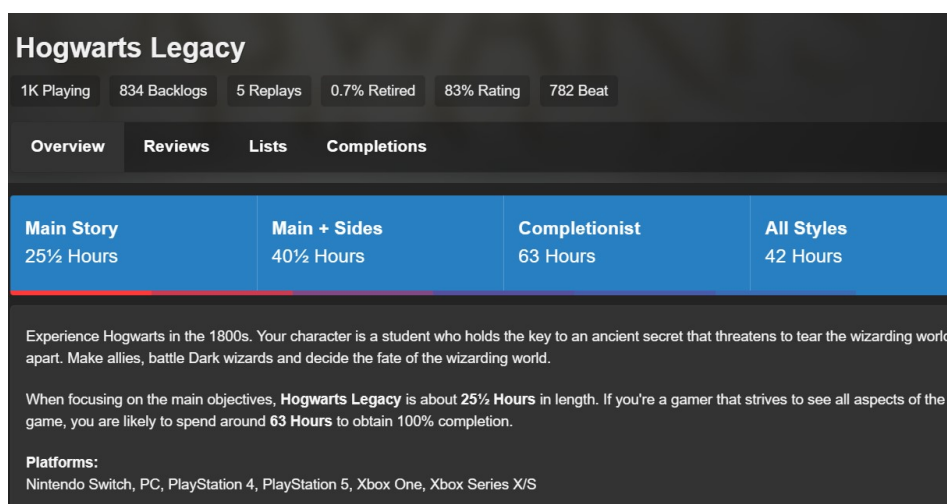


Рисунок 1.6 – Сайт HowLongToBeat

В таблице 1.2 сравниваются существующие аналоги в соответствии с критериями, перечисленными в пункте 1.3. В таблице приняты следующие обозначения: время прохождения — возможность указать время прохождения игры, отзывы и оценки — возможность оставлять оценки и отзывы, русский язык — наличие интерфейса на русском языке.

Таблица 1.2 – Сравнение существующих аналогов

Сервис	Время прохождения	Отзывы и оценки	Русский язык
Steam	-	+	+
HowLongToBeat	+	+	-

Таким образом, ни одно из существующих решений не обладает всеми функциями, которые должны быть реализованы в разрабатываемом приложении. В частности, Steam не позволяет посмотреть среднее время игры, а HowLongToBeat не имеет русского интерфейса.

1.4 Анализ существующих видов баз данных

Для решения поставленной цели необходимо разработать базу данных (БД). БД — это упорядоченный набор структурированной информации или данных,

которые обычно хранятся в электронном виде в компьютерной системе [3].

Классификация реляционных баз данных по способу хранения

Реляционные базы данных по способу хранения делятся на две группы: строковые и колоночные.

Строковые базы данных

Строковые базы данных, в которых записи хранятся построчно, в основном используются в транзакционных системах (англ. OLTP [4]), характеризующихся большим количеством коротких операций вставки, обновления и удаления. OLTP-системы лучше всего для быстрой обработки запросов и поддержания целостности данных в средах с множественным доступом.

Колоночные базы данных

Колоночные базы данных - это базы данных, в которых записи хранятся в "колонок". Этот тип в основном используется в аналитических системах (OLAP [5]), характеризующихся низкими объемами транзакций и запросами, которые часто бывают сложными и включают агрегирование. Мерой эффективности таких систем является время ответа. Наличие агрегированных исторических данных позволяет системам OLAP применять интеллектуальные аналитические методы.

1.4.1 Кэширование данных

Чтобы уменьшить время отклика, можно использовать механизм кэширования данных, для реализации которого подходят NoSQL [6] *in-memory* базы данных, хранящие данные в оперативной памяти, что обеспечивает более быстрый доступ, по сравнению с традиционными реализациями.

Для сокращения времени отклика можно использовать механизм кэширования данных, который может быть реализован с помощью NoSQL [6] *in-memory* базы данных, которые обеспечивают более быстрый доступ, чем традиционные приложения, за счет хранения данных в оперативной памяти.

Проблема синхронизации данных

Приложение записывает данные в кэш и базу данных, но они никак не синхронизированы, что приводит к несоответствиям в данных. Например, может возникнуть ситуация, когда данные удаляются из хранилища и одновременно должны быть удалены из кэша. Эту проблему можно решить с помощью создания триггеров. Триггеры могут запускать изменения или удаления и синхронизировать данные в кэше.

Проблема «холодного старта»

Когда кэш создается впервые, он пуст и не содержит никаких данных. Все запросы отправляются непосредственно в базу данных, и только через некоторое время кэш содержит необходимую информацию. При перезапуске предыдущее состояние кэша может быть восстановлено из журнала событий, хранящегося на диске. При перезапуске кэша данные должны быть синхронизированы с хранилищем - может оказаться, что некоторые данные в кэше устарели на момент перезапуска.

2 КОНСТРУКТОРСКАЯ ЧАСТЬ

В данном разделе будут представлены этапы проектирования системы. Будут описаны таблицы и их атрибуты, проектируемой базы данных. Будет разработана ER-диаграмма базы данных. Будет разработан алгоритм преобразования информации о жанрах игры из строчного представления в xml формат. Будет разработана UML диаграмма компонента бизнес-логики и компонента доступа к базе данных.

2.1 Описание таблиц разрабатываемой базы данных

Согласно ER-диаграмме, показанной на рисунке 1.1, в базе данных выделены и реализованы следующие таблицы:

1. таблица с играми Games;
2. таблица с временными отметками TimeRecords;
3. таблица с отзывами Reviews;
4. таблица с пользователями Users;
5. таблица с платформами Platforms;
6. таблица связка платформ и игр GamePlatform;

На рисунке 2.1 представлена ER-диаграмма базы данных.

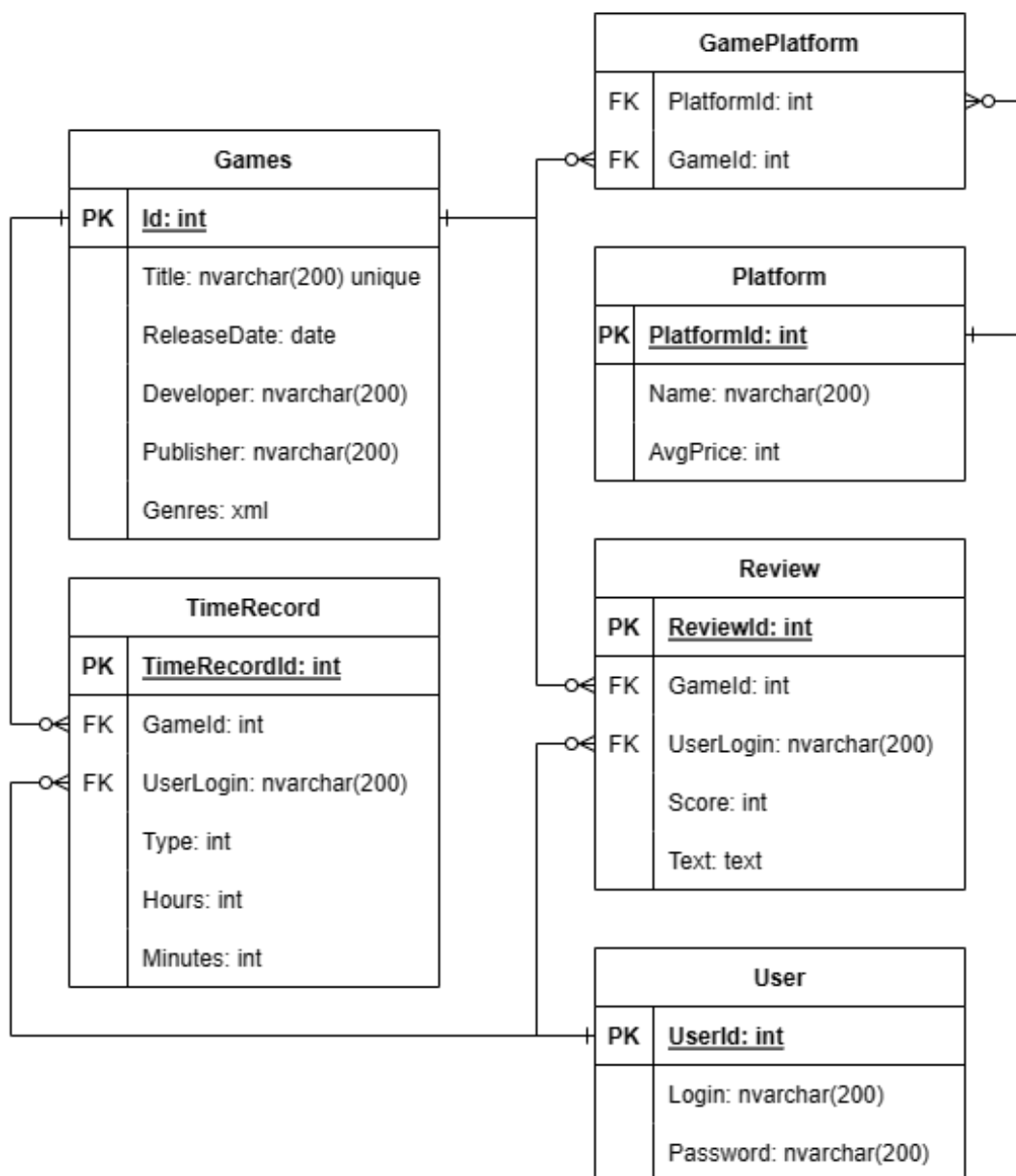


Рисунок 2.1 – Диаграмма сущность-связь базы данных

В таблице Games, которая содержит информацию об играх, представлены следующие атрибуты:

1. Id — идентификатор игры, первичный ключ, int;
2. Title — уникальное название игры, string;
3. ReleaseDate — дата выхода игры, date;
4. Developer — разработчик игры, string;
5. Publisher — издатель, string;

6. Genres — жанры, xml.

В таблице Platforms, которая содержит информацию о платформах, представлены следующие атрибуты:

1. Id — идентификатор платформы, первичный ключ, int;
2. Name — уникальное название платформы, string;
3. AvgPrice — средняя цена платформы, int.

В таблице GamePlatform, которая содержит информацию о связи игр и платформ, представлены следующие атрибуты:

1. GameId — идентификатор игры, int;
2. PlatformId — идентификатор платформы, int.

В таблице TimeRecords, которая содержит информацию о временных отметках на игре, представлены следующие атрибуты:

1. Id — идентификатор, int;
2. GameId — идентификатор игры, int;
3. UserLogin — имя пользователя который оставил эту отметку, string;
4. Hours — количество полных часов потраченных на прохождение игры, int;
5. Minutes — количество полных минут потраченных на прохождение игры, int;
6. Type — тип прохождения, int.

В таблице Reviews, которая содержит информацию об отзывах, представлены следующие атрибуты:

1. Id — идентификатор, int;
2. GameId — идентификатор игры на которую оставлен отзыв;

3. `UserLogin` — имя пользователя оставившего отзыв, `string`;
4. `Text` — текст отзыва, `string`;
5. `PublicationDate` — дата публикации отзыва, `date`.

В таблице `Users`, которая содержит информацию о пользователях, представлены следующие атрибуты:

1. `Login` — имя пользователя, `string`;
2. `Password` — пароль, `string`.

2.2 Схема алгоритма преобразования строки в xml формат

На рисунке 2.2 представлена преобразования строки в xml формат. Сначала алгоритм преобразует данные из одной строки, в которой содержится информация о жанрах, в массив строк, каждая из которых описывает отдельный жанр. Затем каждый из элементов полученного массива будет записан в строку в xml формате. Информация в xml формате может быть успешно вставлена в таблицу.



Рисунок 2.2 – Схема алгоритма преобразования строки в xml формат

2.3 Диаграмма классов приложения

Диаграмма классов компонента доступа к базе данных и компонента бизнес-логики показана на рисунке 2.3.

Доступ к компоненту бизнес-логики осуществляется через интерфейсы сервисов. Доступ к компоненту базы данных осуществляется через Repository классы. Компонент бизнес-логики связан с компонентом доступа к базе данных через Repository интерфейсы. Таким образом, бизнес-логика не зависит от реализации компонента доступа к базе данных.

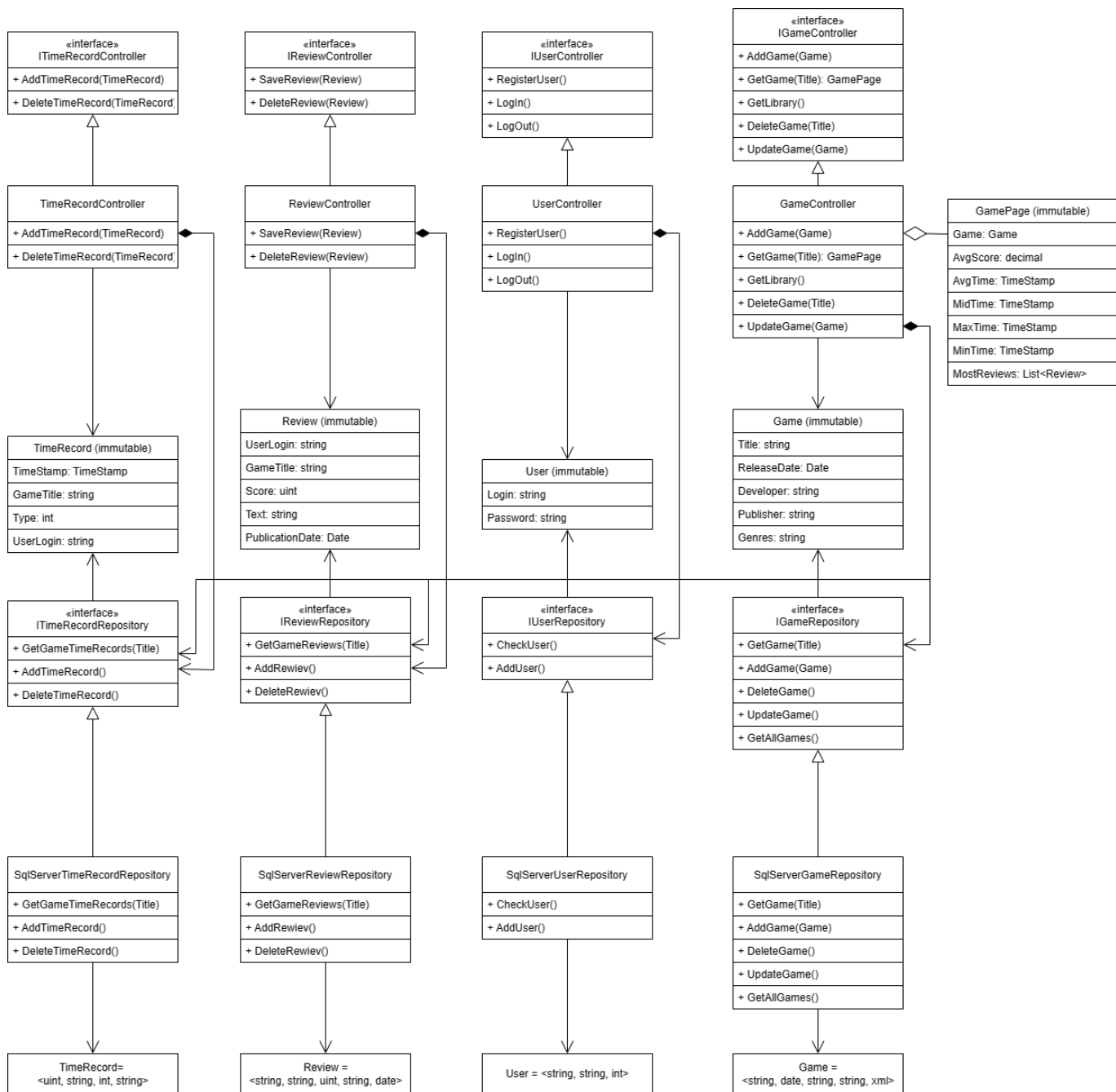


Рисунок 2.3 – Диаграмма классов

3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

В этом разделе будут выбраны средства реализации программного обеспечения. Будет выбран язык программирования, системы управления базами данных и программное обеспечение для графического интерфейса. Будут реализован код базы данных и будет реализовано взаимодействие между приложением и базой данных. Будет приведена демонстрация работы программы, реализующей функции каждой роли.

3.1 Выбор системы управления базой данных

Система управления базами данных — это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [3].

Предполагается, что реализуемая база данных будет хранить большое количество информации.

Система управления базой данных должна удовлетворять следующим требованиям.

1. Система управления базой данных должна предоставлять такие типы данных как строки, целые числа, даты и xml, чтобы можно было хранить информацию.
2. Для загрузки xml файла из приложения в системе управления базой данных должна быть реализована возможность создания хранимых процедур.
3. Система управления базой данных должна обеспечивать надежное хранение данных, то есть данные не должны потеряться даже, если в системе произойдет сбой.

В качестве системы управления базой данных был выбран SQL Server, так как он удовлетворяет всем выделенным требованиям.

3.2 Выбор языка программирования

В качестве языка программирования был выбран C# [7], так как он является полностью объектно-ориентированным, что позволяет использовать наследование, интерфейсы и абстракцию. Также стандартная библиотека C# включает пространство имен System.Data.Linq.Mapping [8], которое содержит классы для формирования объектной модели LINQ to SQL, представляющей структуру и содержимое реляционной базы данных.

Для упаковки приложения как полноценного продукта была выбрана система контейнеризации Docker [9]. Docker используется для создания изолированной среды для программного обеспечения, которое может быть развернуто на различных устройствах без дополнительной настройки.

3.3 Реализация ролевой модели

Необходимо реализовать ролевую модель в соответствии с пунктом 1.2. Роль — это разрешение, предоставляемое группе пользователей для доступа к данным.

Ролевая модель реализована как на уровне приложения, так и на уровне базы данных 3.1. В листинге 3.1 приведено создание ролей для авторизованного и неавторизованного пользователей.

Листинг 3.1 – Ролевая модель на уровне базы данных

```
1 create login [AuthorizedUser] with password = 'UserPassword1'
2 create user AU for login [AuthorizedUser]
3 create role [AuthorizedUserRole]
4 alter role [AuthorizedUserRole] add member AU
5 deny delete on Reviews to [AuthorizedUserRole];
6 deny update on Reviews to [AuthorizedUserRole];
7 deny delete on TimeRecords to [AuthorizedUserRole];
8 deny update on TimeRecords to [AuthorizedUserRole];
9 deny insert on Games to [AuthorizedUserRole];
10 deny delete on Games to [AuthorizedUserRole];
11 deny update on Games to [AuthorizedUserRole];
12 deny delete on Users to [AuthorizedUserRole];
13 deny update on Users to [AuthorizedUserRole];
14 deny alter to [AuthorizedUserRole];
15 grant insert on Reviews to [AuthorizedUserRole];
16 grant insert on TimeRecords to [AuthorizedUserRole];
17 grant insert on Users to [AuthorizedUserRole];
18 grant select on Users to [AuthorizedUserRole];
19 grant select on Games to [AuthorizedUserRole];
20 grant select on Reviews to [AuthorizedUserRole];
21 grant select on TimeRecords to [AuthorizedUserRole];
22
23 create login [GuestUser] with password = 'GuestPassword1'
24 create user GU for login [GuestUser]
25 create role [GuestUserRole]
26 alter role [GuestUserRole] add member GU
27 deny delete on Reviews to [GuestUserRole];
28 deny update on Reviews to [GuestUserRole];
29 deny delete on TimeRecords to [GuestUserRole];
```

```

30 deny update on TimeRecords to [GuestUserRole];
31 deny insert on Games to [GuestUserRole];
32 deny delete on Games to [GuestUserRole];
33 deny update on Games to [GuestUserRole];
34 deny delete on Users to [GuestUserRole];
35 deny update on Users to [GuestUserRole];
36 deny alter to [GuestUserRole];
37 deny insert on Reviews to [GuestUserRole];
38 deny insert on TimeRecords to [GuestUserRole];
39 grant insert on Users to [GuestUserRole];
40 grant select on Users to [GuestUserRole];
41 grant select on Games to [GuestUserRole];
42 grant select on Reviews to [GuestUserRole];
43 grant select on TimeRecords to [GuestUserRole];

```

3.4 Реализация процедуры преобразования строки в xml формат и вставки в таблицу

В листинге 3.2 представлена реализация процедуры преобразования строки с перечислением жанров в xml формат и вставки в таблицу Games на уровне базы данных.

Листинг 3.2 – Процедура преобразования строки в xml формат и вставки в таблицу на уровне базы данных

```

1 create or alter procedure InsertGameProcedure
2 (
3     @title nvarchar(200),
4     @releaseDate date,
5     @developer nvarchar(200),
6     @publisher nvarchar(200),
7     @genres nvarchar(200)
8 )
9 as
10 begin
11     delete temp where 0 = 0
12     insert into temp
13     select @title, genre_split.value
14     from string_split(@genres, ' ') as genre_split
15
16     insert into Games
17     values(@title, @releaseDate, @developer, @publisher,
18     (
19         select temp.genre
20         from temp
21         for xml path(''), type, root('Genre')
22     ))
23     delete temp where 0 = 0
24 end

```

В листинге 3.3 приведена реализация процедуры преобразования строки с перечислением жанров в xml формат и вставки в таблицу Games на уровне приложения.

Листинг 3.3 – Метод преобразования строки в xml формат и вставки в таблицу на уровне приложения

```

1 public void AddGameXMLApp(Game game)
2 {
3     DataContext db = new DataContext(adminConnection);
4     Table<GamesXML> gameTable = db.GetTable<GamesXML>();

```

```

5  XmlDocument xml = GetXml(game.Genres);
6  GamesXML g = new GamesXML()
7  {
8      Title = game.Title,
9      ReleaseDate = game.ReleaseDate,
10     Developer = game.Developer,
11     Publisher = game.Publisher,
12     Genres = xml.InnerXml,
13 };
14 gameTable.InsertOnSubmit(g);
15 try
16 {
17     db.SubmitChanges();
18 }
19 catch (DuplicateKeyException e)
20 {
21     throw new GameAlreadyExistsException(e.Message);
22 }
23 }
24
25 XmlDocument GetXml(string genres)
26 {
27     string[] genreSplit = genres.Split(' ');
28     XmlDocument xml = new XmlDocument();
29     XmlElement root = xml.CreateElement("Genres");
30     xml.AppendChild(root);
31     foreach (var genre in genreSplit)
32     {
33         xml["Genres"].AppendChild(xml.CreateElement("genre")).InnerText = genre;
34     }
35 }

```

3.5 Реализация модели «Отзыв»

В листинге 3.4 приведено создание таблицы Reviews в базе данных, а также создание ограничений целостности.

Листинг 3.4 – Таблица Reviews

```

1 create table Reviews
2 (
3     Gameld int,
4     UserLogin nvarchar(200),
5     [Text] nvarchar(1000),
6     Score int,
7     PublicationDate date,
8     primary key(Gameld, UserLogin)
9 );

```

В листинге 3.5 представлена реализация модели Review, которая реализует соответствующую сущность в компоненте бизнес-логики.

Листинг 3.5 – Модель Review

```

1 public class Review
2 {
3     public int Gameld { get; }
4     public string UserLogin { get; }
5     public uint Score { get; }
6     public string Text { get; }
7     public DateTime PublicationDate { get; }
8     public Review(int gameld, string userLogin, uint score, string text, DateTime publicationDate)
9     {
10         Gameld = gameld;
11         UserLogin = userLogin;
12         Score = score;
13         Text = text;
14         PublicationDate = publicationDate;
15     }
16 }

```

В листинге 3.6 приведена реализация репозитория, который предоставляет доступ к данным таблицы Reviews из базы данных.

Листинг 3.6 – Репозиторий SqlServerReviewRepository доступа к таблице Reviews из базы данных

```

1 [Table]
2 public class Reviews
3 {
4     [Column(IsPrimaryKey = true)]
5     public int Gameld { get; set; }
6     [Column(IsPrimaryKey = true)]
7     public string UserLogin { get; set; }
8     [Column]
9     public string Text { get; set; }
10    [Column]
11    public int Score { get; set; }
12    [Column]
13    public DateTime PublicationDate { get; set; }
14 }
15
16 public class SqlServerReviewRepository : SqlServerRepository, IReviewRepository
17 {
18     public SqlServerReviewRepository() : base() { }
19
20     public void AddReview(Review review)
21     {
22         DataContext db = new DataContext(userConnection);
23         Table<Reviews> reviewTable = db.GetTable<Reviews>();
24         Reviews r = new Reviews()
25         {
26             Gameld = review.Gameld,
27             UserLogin = review.UserLogin,
28             Score = (int)review.Score,
29             Text = review.Text,
30             PublicationDate = review.PublicationDate
31         };
32         reviewTable.InsertOnSubmit(r);
33         db.SubmitChanges();
34     }
35
36     public void DeleteReview(Review review)
37     {
38         DataContext db = new DataContext(adminConnection);
39         Reviews rev = db.GetTable<Reviews>().Where(r => r.Gameld == review.Gameld && r.UserLogin ==
40             review.UserLogin).First();
41         db.GetTable<Reviews>().DeleteOnSubmit(rev);
42         db.SubmitChanges();
43     }
44
45     public List<Review> GetReviews(int id)
46     {
47         DataContext db = new DataContext(guestConnection);
48         IQueryable<Reviews> gameReviews = from r in db.GetTable<Reviews>()
49             where r.Gameld == id
50             select r;
51         List<Review> reviews = new List<Review>();
52         foreach (var rev in gameReviews)
53             reviews.Add(new Review(rev.Gameld, rev.UserLogin, (uint)rev.Score, rev.Text, rev.PublicationDate));
54         return reviews;
55     }
56 }

```

3.6 Демонстрация интерфейса приложения

На рисунке 3.1 показан интерфейс приложения при запуске, библиотека видеоигр. Пользователь не авторизован. Пользователь может только авторизовать-

ся, зарегистрироваться и посмотреть информацию и статистику об игре.

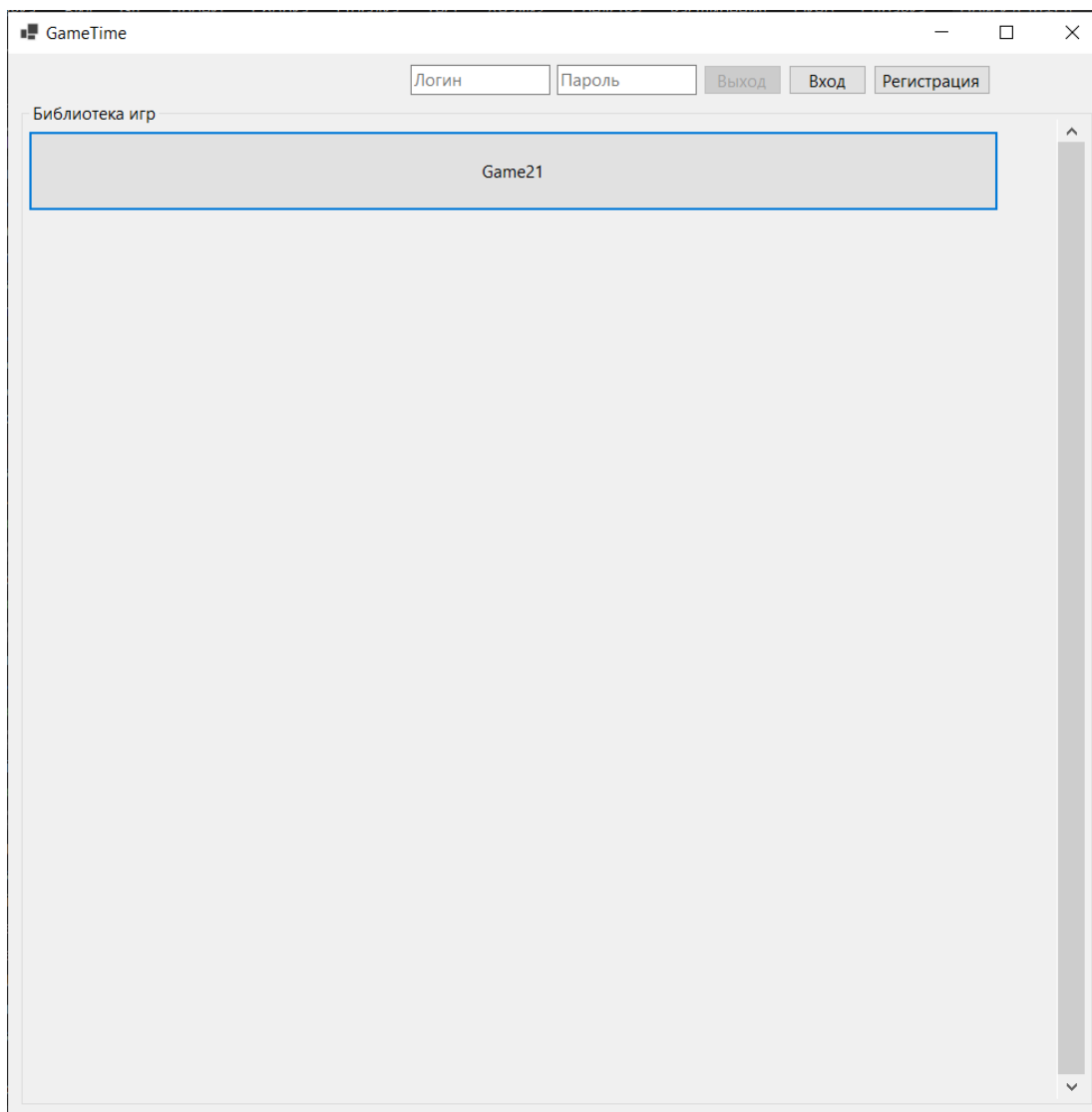


Рисунок 3.1 – Библиотека видеоигр

На рисунках 3.2, 3.3, 3.4 показана страница игры, которая доступна неавторизованному, авторизованному пользователю и администратору соответственно.

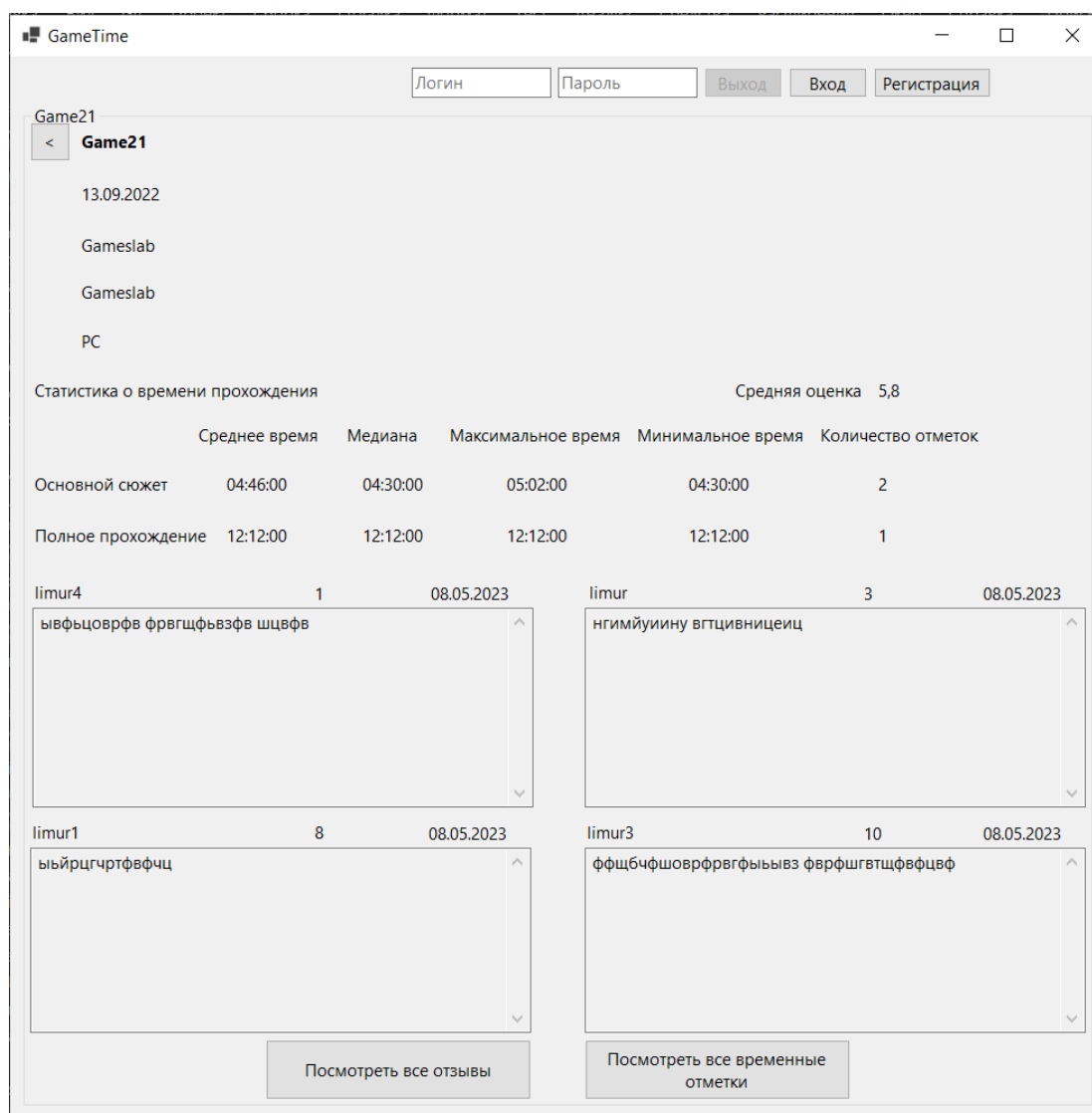


Рисунок 3.2 – Страница игры, которую видит не авторизованный пользователь

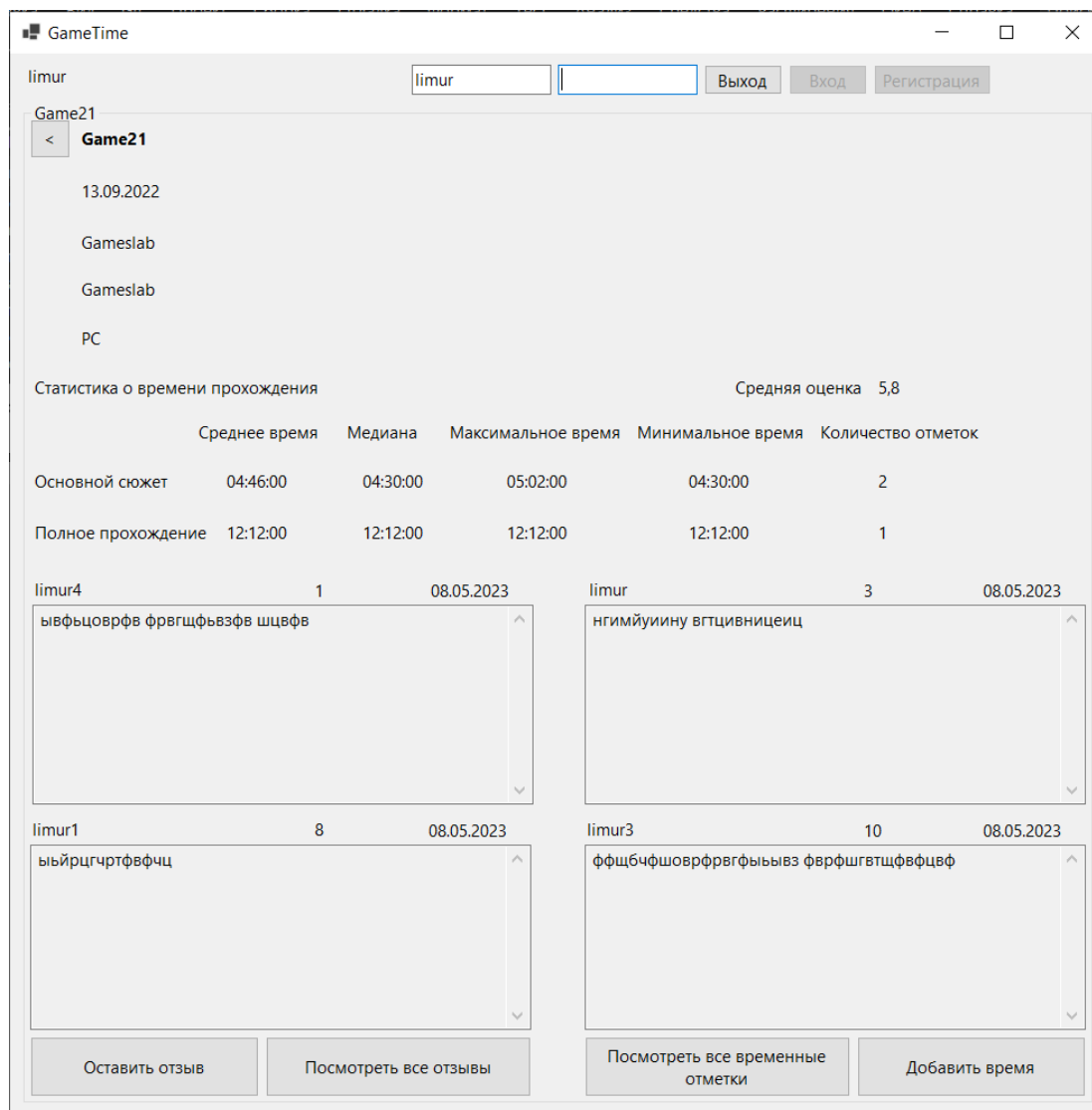


Рисунок 3.3 – Страница игры, которую видит авторизованный пользователь

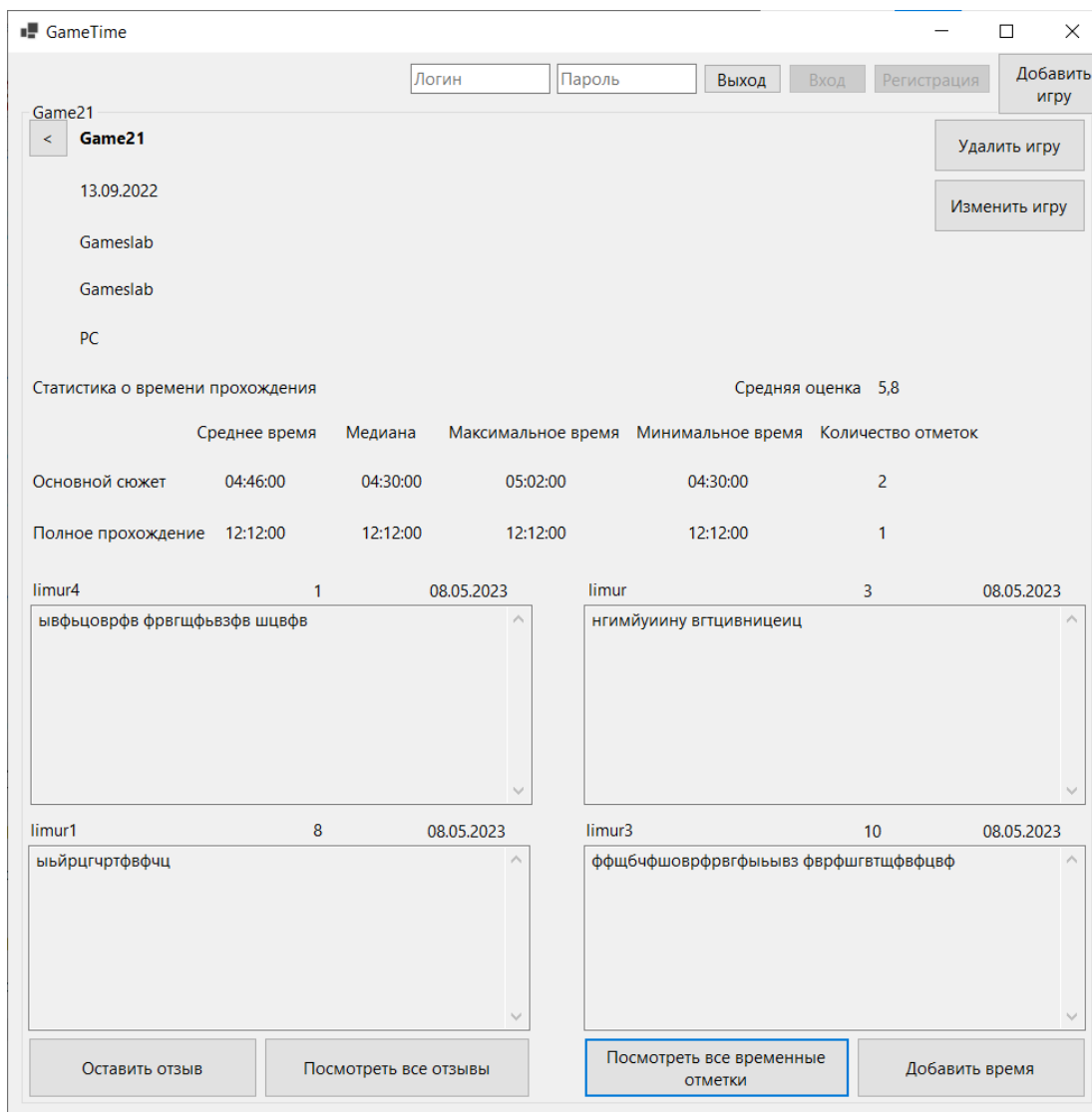


Рисунок 3.4 – Страница игры, которую видит администратор

На рисунке 3.5 показан интерфейс окна, в котором можно добавить временную отметку игре.

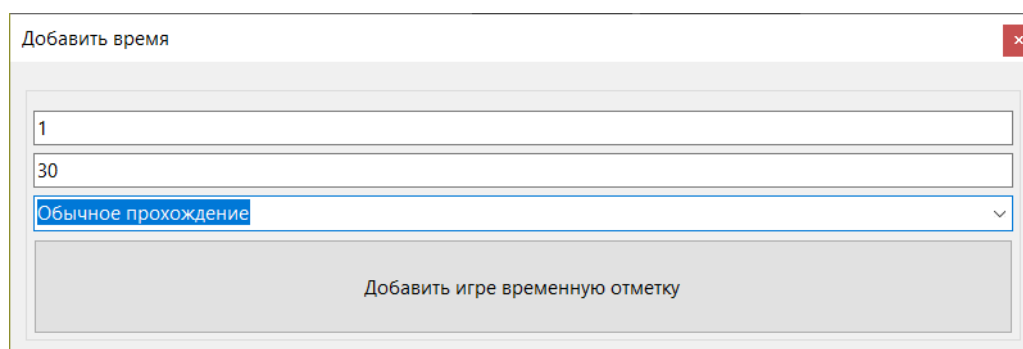
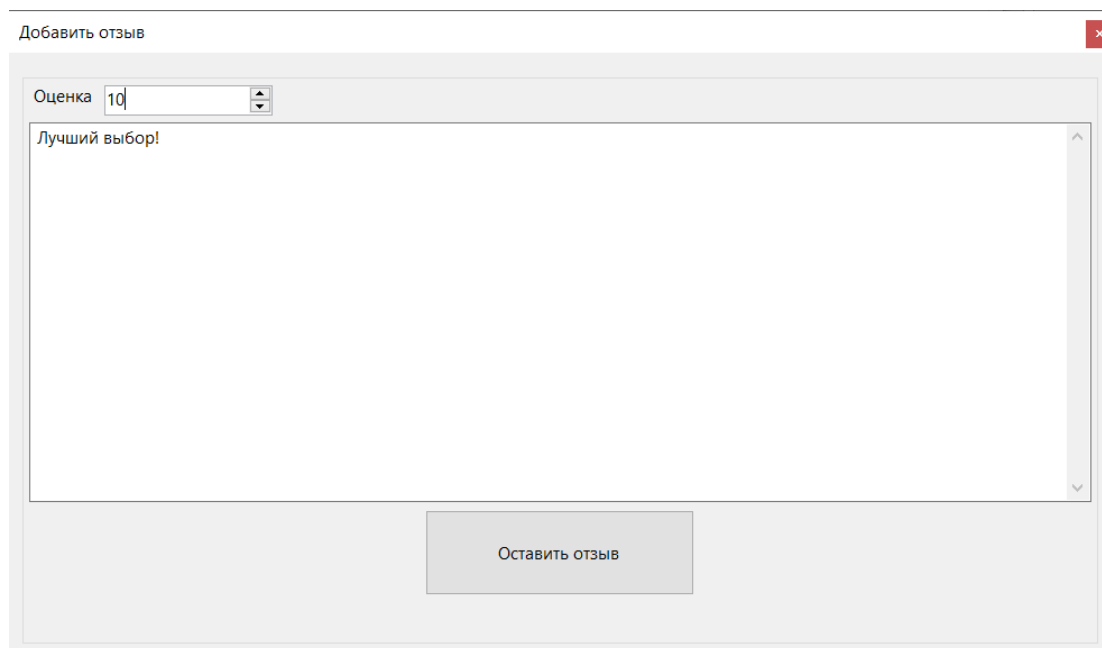


Рисунок 3.5 – Окно для добавления временной отметки игре

На рисунке 3.6 показан интерфейс окна, в котором можно добавить отзыв об игре.



Добавить отзыв

Оценка 10

Лучший выбор!

Оставить отзыв

Рисунок 3.6 – Окно для добавления отзыва об игре

На рисунке 3.7 показан интерфейс окна, в котором можно посмотреть все временные отметки игры. Администратору доступно удаление конкретной, что показано на рисунке 3.8.

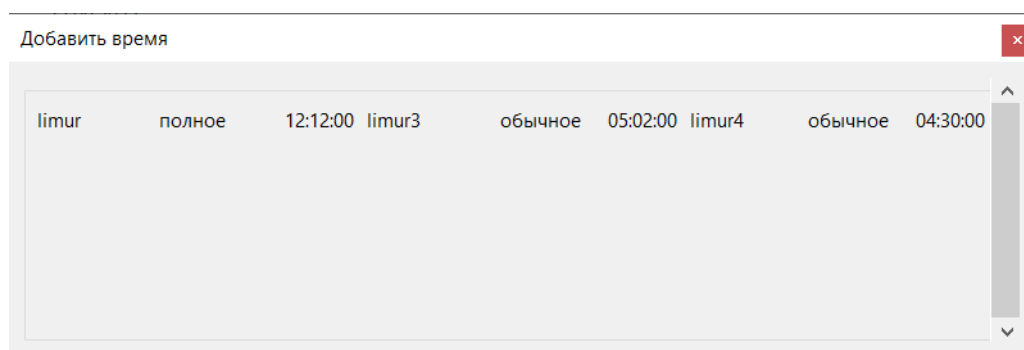


Рисунок 3.7 – Все временные отметки игры

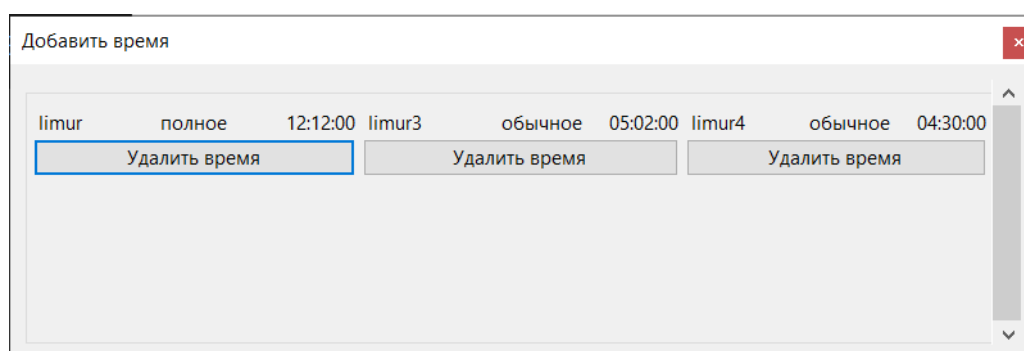


Рисунок 3.8 – Все временные отметки игры (администратор)

На рисунке 3.9 показан интерфейс окна, в котором можно посмотреть все отзывы игры. Администратору доступно удаление конкретного, что показано на рисунке 3.10.

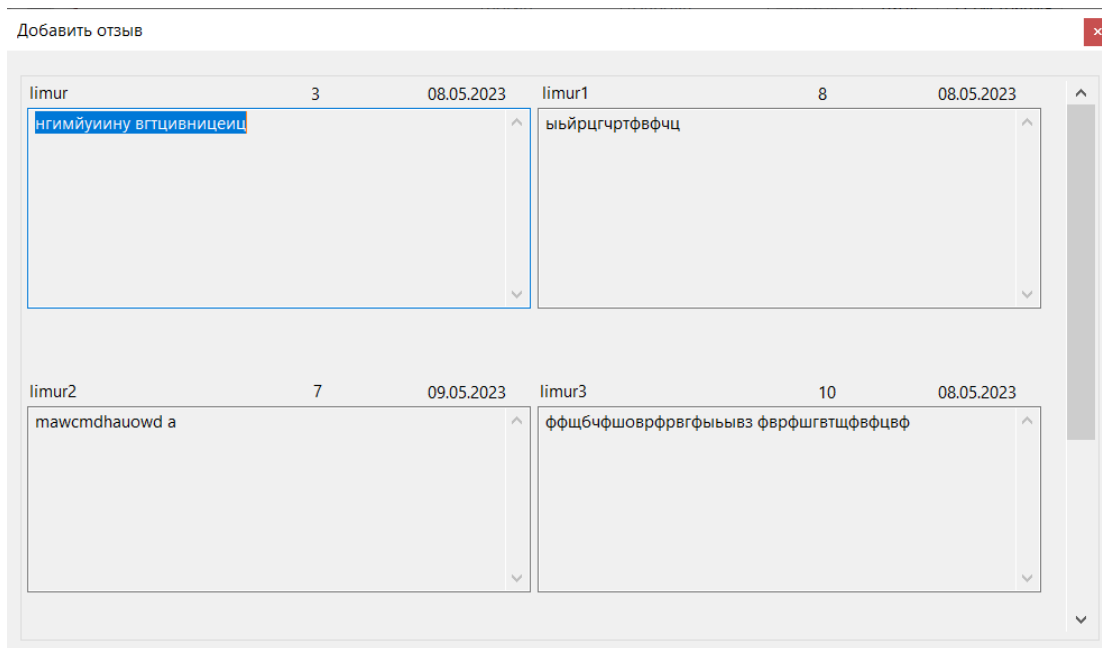


Рисунок 3.9 – Все отзывы игры

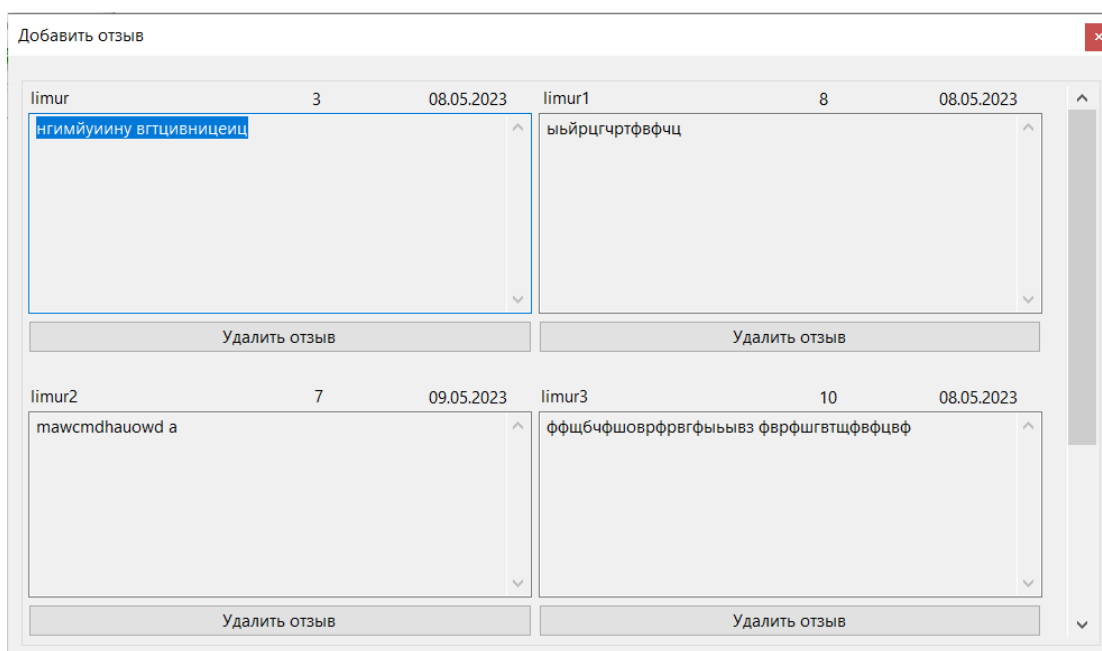


Рисунок 3.10 – Все отзывы игры (администратор)

На рисунке 3.11 представлен интерфейс окна для добавления или изменения игры.

Добавить игру

Название

Разработчик

Издатель

Жанры

9 мая 2023 г.

Добавить игру

Рисунок 3.11 – Окно добавления игры

3.7 Тестирование приложения

Тестирование проводилось по методологии черного ящика. Для тестирования компонента бизнес-логики была использована такая технология тестирования, как unit-тесты. Для тестирования взаимодействие компонента доступа к данным и базы данных были написаны интеграционные тесты. Система тестирования была реализована с помощью фреймворка MS Test [10], которая позволяет быстро и автоматически протестировать отдельные компоненты приложения независимо от остальной его части.

4 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

В данном разделе будут представлены характеристики компьютера, на котором проводилось исследование. Также в данном разделе будут приведено описание и постановка исследования. Будут приведены результаты сравнения времени вставки игры в таблицу в зависимости от количества игр и от того, где происходит преобразование строки, содержащей информацию о жанрах, в xml-формат на уровне приложения или на уровне базы данных.

4.1 Технические характеристики устройства

Технические характеристики устройства, на котором выполнялись измерения:

- процессор — AMD Ryzen 5 4600H с Radeon Graphics 3.00 ГГц [11];
- операционная система — Windows 10 Корпоративная 22H2;
- оперативная память — 24 Гб;
- количество логических ядер — 12;
- ёмкость диска — 512 GB;

Во время тестирования ноутбук был включен в сеть питания и нагружен только приложениями встроенными в операционную систему и системой тестирования. Сторонние приложения запущены не были.

4.2 Описание исследования

Цель эксперимента - исследовать зависимость по времени добавления игр в таблицу от их количества и от реализации алгоритма преобразования строки, содержащей информацию о жанрах, в xml формат — на уровне приложения или на уровне базы данных.

В ходе исследования анализируется время, затрачиваемое на вставку игр в таблицу в зависимости от их количества и от реализации алгоритма преобразования строки, содержащей информацию о жанрах, в xml формат. Для каждого

количества игр было проведено 50 измерений и вычислено среднее арифметическое.

В исследовании для каждой реализации алгоритма преобразования строки в xml формат измеряется время вставки различного количества игр в таблицу. Каждой игре добавляется 20 жанров.

4.3 Результаты исследования

В таблице 4.1 приведены результаты исследования.

Таблица 4.1 – Результаты исследования зависимости времени (мс), затрачиваемого на вставку в игр таблицу, от количества и реализации алгоритма

Количество игр	Реализация алгоритма	
	на стороне базы данных	на стороне приложения
100	1116,7	678,8
200	1347,0	1358,1
300	2045,2	2585,2
400	4576,3	4851,6
500	9162,1	8373,2
600	10861,2	13313,7
700	14750,6	18671,4
800	15918,4	21309,9
900	23513,6	31194,5
1000	39439,7	45007,0

На рисунке 4.1 представлены графики зависимости времени вставки игр в таблицу от реализации алгоритма и от количества добавляемых игр.

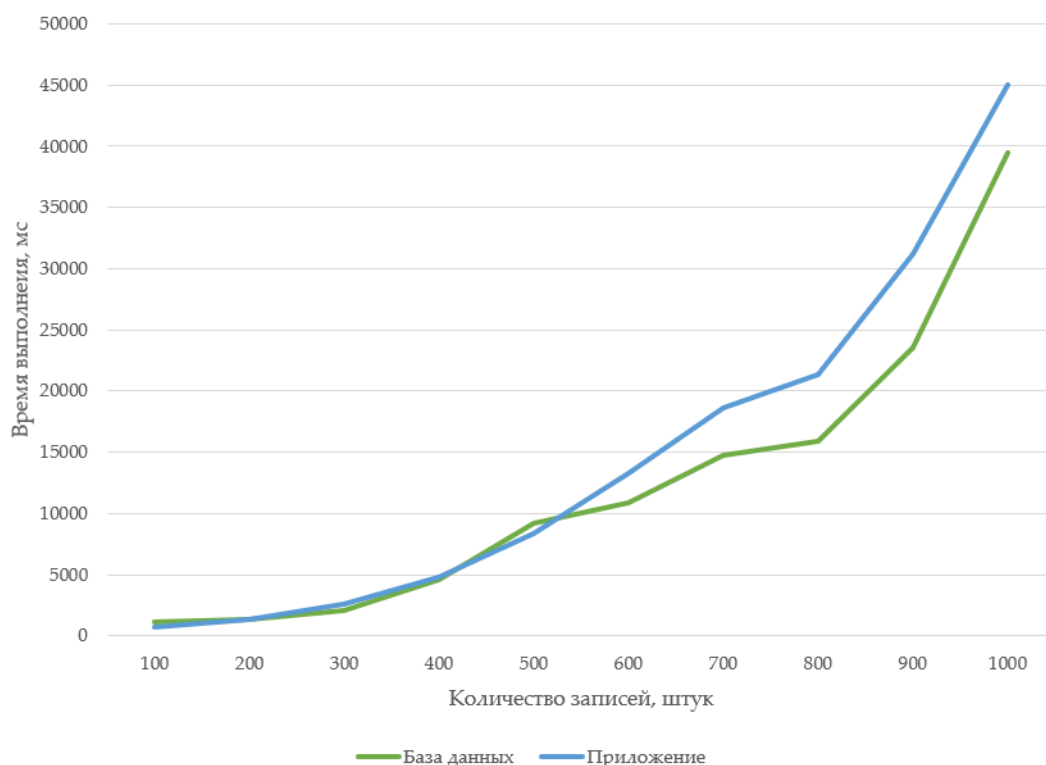


Рисунок 4.1 – Зависимость времени вставки игр в таблицу от реализации алгоритма и от количества добавляемых игр

Результаты показывают, что добавление игр в таблицу с помощью алгоритма преобразования строки в xml формат, реализованного на уровне базы данных, более эффективно, чем добавление игр с помощью алгоритма, реализованного на уровне приложения. Для добавления 800 игр реализация на уровне базы данных примерно на 25% быстрее, чем реализация на уровне приложения.

При реализации вставки на стороне базы данных в процедуру передается одна запись, и СУБД выполняет все вычисления, в то время как приложение должно сначала получить таблицу, в которую должна быть выполнена вставка, затем преобразовать строки в формат xml и, наконец, вставить преобразованную запись в таблицу.

4.4 Вывод

Результаты показывают, что при фиксированном количестве игр для вставки игр в таблицу требуется больше времени, когда алгоритм выполняется на уровне приложения, чем когда он выполняется на уровне базы данных. Таким образом, видно, что на время выполнения влияет не то, выполняется ли алгоритм

на уровне базы данных или на уровне приложения, а объем данных, которые передаются из базы данных в приложение и обратно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были решены все поставленные задачи и достигнута поставленная цель. Была разработана база данных, позволяющая отображать статистическую информацию о времени прохождения игры.

В ходе исследования были проанализированы методы хранения данных, рассмотрены существующие СУБД, получены знания в области проектирования баз данных и приложений. Также было разработано программное обеспечение для обеспечения доступа к данным.

Исследование показало, что на время выполнения операций, требующих доступа к базе данных, зависит не от того, на каком уровне выполняются вычисления, а объем данных, передаваемых во время этих вычислений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Steam [Электронный ресурс]. Режим доступа: <https://store.steampowered.com/> (дата обращения: 25.03.2023).
2. HowLongToBeat [Электронный ресурс]. Режим доступа: <https://howlongtobeat.com/> (дата обращения: 25.03.2023).
3. Что такое база данных | Oracle Россия и СНГ [Электронный ресурс]. Режим доступа: <https://www.oracle.com/ru/database/what-is-database/> (дата обращения: 25.03.2023).
4. What is OLTP? | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/cloud/learn/oltp> (дата обращения: 25.03.2023).
5. What is OLAP? | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/cloud/learn/olap> (дата обращения: 25.03.2023).
6. Что такое NoSQL? | Amazon AWS [Электронный ресурс]. Режим доступа: <https://aws.amazon.com/ru/nosql/> (дата обращения: 25.03.2023).
7. Документация C# [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 03.04.2023).
8. System.Data.Linq.Mapping Пространство имен [Электронный ресурс]. Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/api/system.data.linq.mapping?view=netframework-4.8> (дата обращения: 25.03.2023).
9. Docker: Empowering App Development for Developers [Электронный ресурс]. Режим доступа: <https://www.docker.com/> (дата обращения: 03.04.2023).
10. Модульное тестирование кода C# с использованием MSTest и .NET [Электронный ресурс]. Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-with-mstest> (дата обращения: 03.04.2023).
11. AMD Ryzen™ 5 4600H [Электронный ресурс]. Режим доступа: <https://www.amd.com/ru/products/apu/amd-ryzen-5-4600h> (дата обращения: 25.03.2023).