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

ФАКУЛЬТЕТ ИУ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА ИУ7 «ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Разработка базы данных соревновательной игры

Студент	**ИУ7-61Б**		
	(группа)	(подпись, дата)	(И.О. Фамилия)
Руководитель курсового проекта			**Волкова Л.Л.**
		(подпись, дата)	(И.О. Фамилия)
Консультант			
		(подпись, дата)	(И.О. Фамилия)

2025 г.

РЕФЕРАТ

Расчетно-пояснительная записка 38 с., 13 рис., 14 источников, 1 прил.

БАЗЫ ДАННЫХ, СОРЕВНОВАТЕЛЬНАЯ ИГРА, C#, POSTGRESQL, WEB-СЕРВЕР.

Цель работы — разработка базы данных соревновательной игры.

В процессе работы изучены существующие модели баз данных, спроектирована собственная база данных, соответствующая поставленной цели, реализован Web-сервер на языке программирования C# с API для доступа и работы с информационной системой. Проведено исследование зависимости времени выполнения задачи при помощи хранимой процедуры и при составлении обычных запросов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью курсовой работы является разработка базы данных соревновательной игры.

Для достижения поставленной цели необходимо решить следующие задачи:

— формализовать соревновательную игру, описать предметную область;

— провести анализ существующих решений;

— формализовать сущности базы данных;

— спроектировать архитектуру базы данных и ограничения целостности;

— спроектировать процедуру выдачи наград;

— описать интерфейс доступа к базе данных;

— выбрать средства реализации базы данных и приложения;

— реализовать базу данных и приложение;

— описать методы тестирования разработанной функциональности и разработать тесты для проверки корректности работы приложения;

— исследовать зависимость времени выдачи наград от количества вознаграждаемых игроков в двух случаях: при помощи хранимой процедуры и при помощи обычных запросов.

1 Аналитическая часть

1.1 Формализация соревновательной игры

Соревновательная игра состоит из основной части, оцениваемая количеством очков (целым числом), и заключительной, где очки игроков сравниваются между собой.

Соревнование характеризуется сроками проведения (начало и конец), информацией о выдаваемых наградах, а также параметрами игры. Выдаваемые награды характеризуются собственно наградой, а также критерием выдачи исходя из таблицы лидеров. Возможны следующие критерии:

— место в таблице лидеров — игрок занял место в таблице лидеров в пределе указанного диапазона;

— ранг в таблице лидеров — доля игроков (число от 0 до 1), которых игрок опередил в таблице лидеров, лежит в пределе указанного диапазона.

Поскольку формат математической игры подвержен изменению, параметры игры представляют собой текстовый файл, содержащий всю необходимую информацию для расшифровки клиентским приложением.

В основной части игрок зарабатывает очки, решая математические примеры на скорость, указанные в параметрах соревнования. Результат, как и время подачи результата сохраняется в таблицу лидеров соответствующего соревнования.

Заключительная часть наступает по истечению срока соревнования. Составляется таблица лидеров, сортируя игроков сначала по убыванию игрового счёта, потом по возрастанию времени подачи последнего результата. Награды выдаются игрокам, исходя из таблицы лидеров.

База данных соревновательной игры создаётся для математической игры — уровни содержат информацию о примерах, выдаваемых пользователем клиентским приложением. Клиентское приложение осуществляет подсчёт очков и отправляет результат серверу.

В рамках поставленной цели требуется разработать базу данных, содержащую информацию о соревнованиях, об игроках, о их участии в соревнованиях, а также информацию о выданных наградах и выдаваемых наградах соревнования. Требуется, чтобы программа автоматически выполняла награждение игроков по истечению сроков соревнования. Также, приложение не должно позволять игроку участвовать вне сроков действия соревнования.

Желательно, чтобы награды имели представление в виде изображения. Администратор должен иметь возможность изменять его. При этом, загружаемое изображение должно дополнительно обрабатываться:

— иметь размер не более N по каждой стороне;

— иметь непрозрачный фон, заменить прозрачный фон на чёрный цвет;

— приводится к одному формату изображения.

Возможно, что игрок может настроить себе изображение профиля. Также, награда может иметь представление в игре — текстовый файл, описывающий дополнительные характеристики

награды.

1.2 Анализ существующих решений

Для сравнения были выбраны соревновательные и обучающие игры. Составлены следующие критерии:

— наличие временных соревнований (К1);

— внутриигровые вознаграждения за соревнования (К2);

— образовательная направленность (К3);

— доступность в РФ (К4).

Сравнение представлено в таблице 1.1.

Таблица 1.1 — Сравнение существующих решений

	К1	К2	К3	К4
Клавогонки	+	–	+	+
Математический тренажёр	–	–	+	+
Clash Royale	+	+	–	–
Предлагаемое решение	+	+	+	+

Таким образом, разрабатываемое решение не уступает существующим по выдвинутым критериям.

1.3 Формализация данных

Разрабатываемая база данных должна содержать информацию об игроках, профилей игроков, соревнований, результатах игры, наград. Информация об игровом уровне хранится как Сущности базы данных представлены в таблице 1.2

Таблица 1.2 — Описание сущностей базы данных

Сущность	Данные
Аккаунт	Логин, почта, права доступа, пароль
Профиль	Имя, описание, изображение
Соревнование	Имя, описания, даты начала и конца, описание уровня, выданы ли награды
Заявка об участии	Количество очков, время участия. Ссылается на соревнование и профиль
Тип награды	Имя, описание, редкость, изображение, внутриигровое представление
Награда	Дата выдачи. Ссылается на тип награды, соревнование и игрока
Награда соревнования	Ссылается на тип награды, соревнование, и критерий выдачи по месту или рангу.
Критерий выдачи по месту	Минимальное и максимальное место
Критерий выдачи по рангу	Минимальный и максимальный ранг

Диаграмма сущностей базы данных представлена на рисунке 1.1

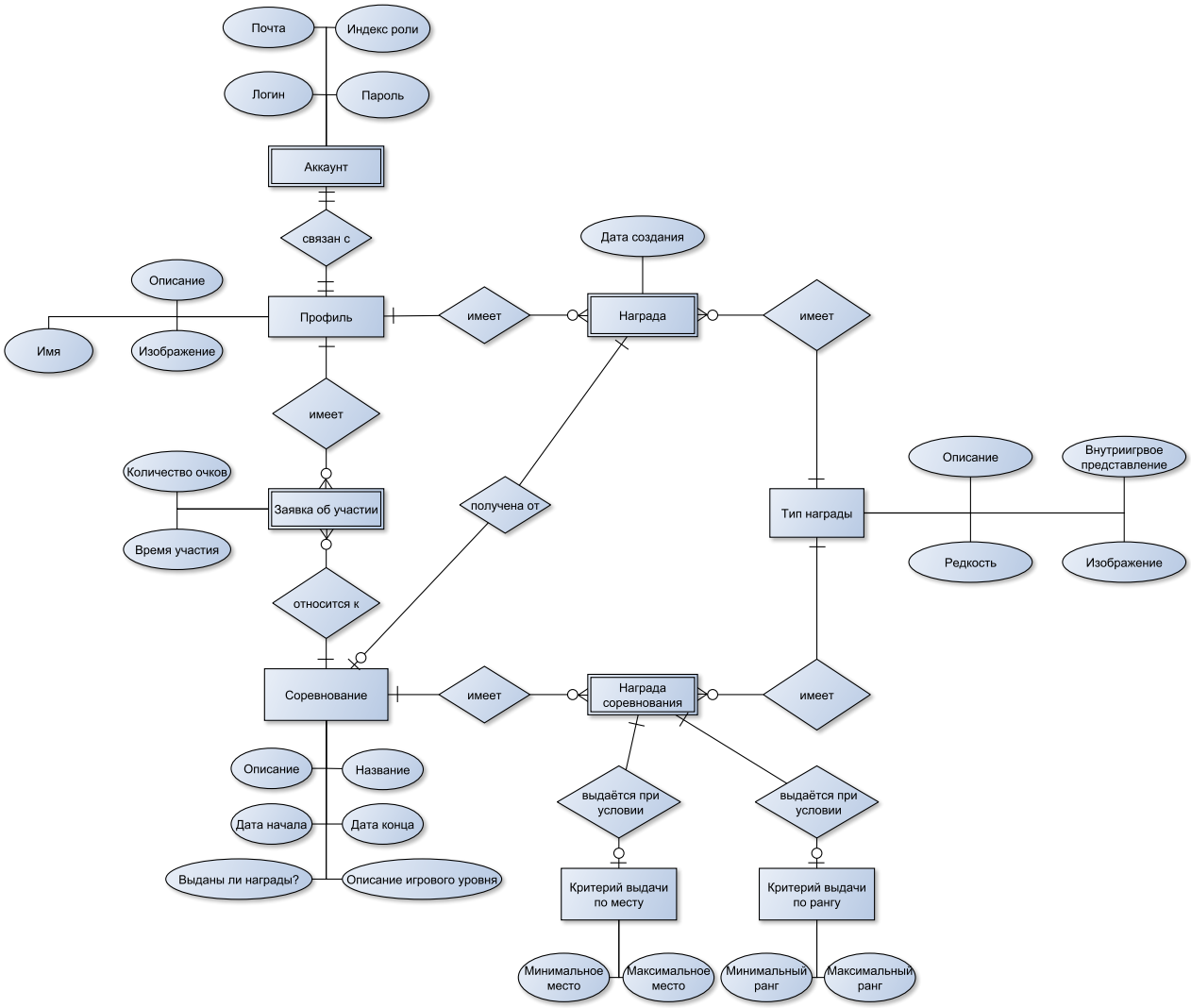

Рисунок 1.1 — Диаграмма сущностей базой данных в нотации Чена

1.4 Категории пользователя

В рамках задачи было выделено три категории пользователя:

— гость,

— игрок,

— администратор.

Все категории пользователя имеют возможность просматривать профили игроков, таблицы лидеров соревнования, а также собственно соревнований (данные о сроках проведения и наградах)

Гость имеет возможность авторизации и создания аккаунта. При авторизации гость может стать игроком или администратором.

Игрок может просматривать свои награды, редактировать свой профиль, а также участвовать в соревнованиях.

Администратор может:

— назначать и отзывать награды игрока;

— создавать и редактировать соревнования;

— отзывать нежелательные результаты, удаляя их;

— создавать и редактировать типы наград.

Игрок и администратор имеют возможность выйти из аккаунта, в последствии чего устанавливается категория пользователя "гость".

Диаграмма пользования базой данных представлена на рисунке 1.2.

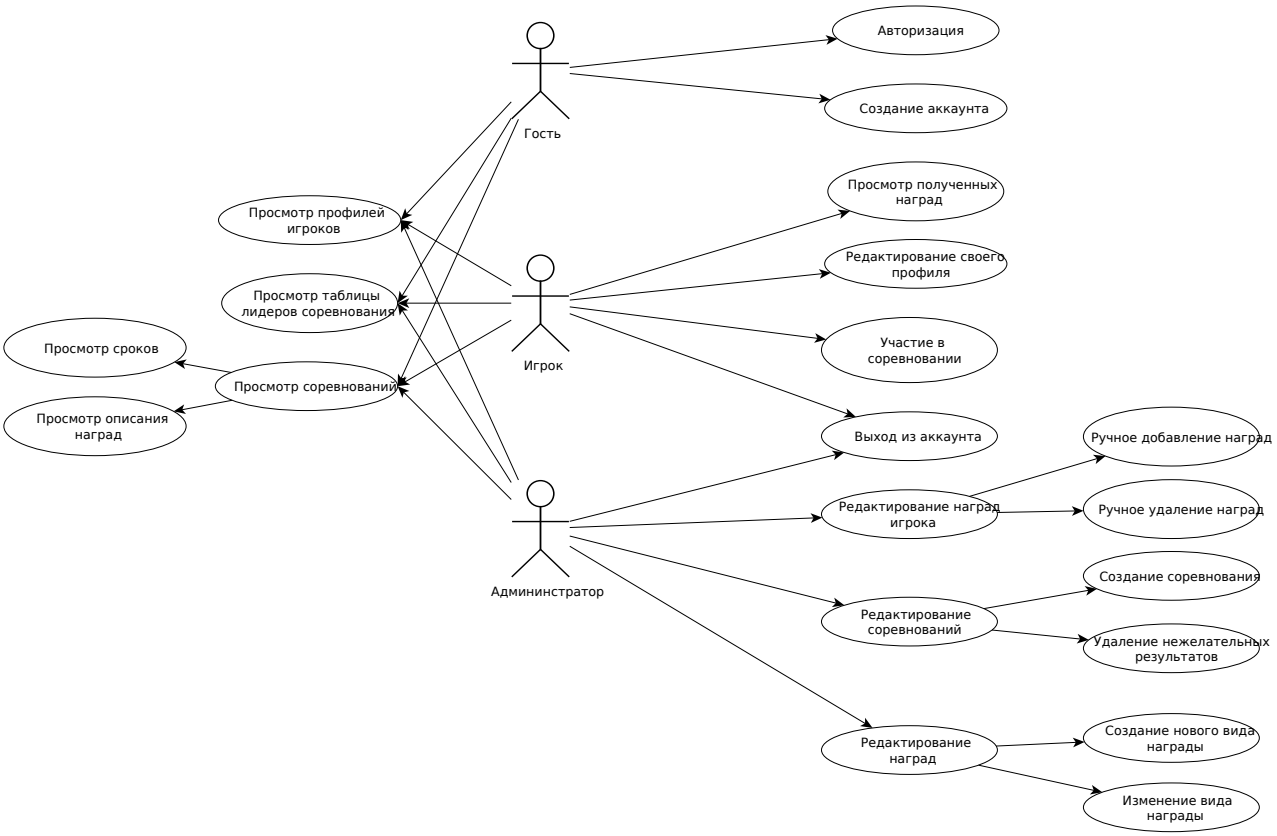

Рисунок 1.2 — Диаграмма пользования базой данных

1.5 Выбор модели данных

Основными моделями данных являются [1]:

— иерархическая модель;

— сетевая модель;

— реляционная модель.

Иерархическая модель позволяет строить БД с иерархической древовидной структурой [1]. В иерархической модели данных используется ориентация древовидной структуры от корня к листьям — любой дочерний узел имеет только один родительский узел. Исходя из этой особенности, реализация связей многие ко многим не поддерживается. Поскольку в рамках задачи присутствует отношение многие ко многим, данная модель не подходит для достижения

цели.

Сетевая модель данных основывается на графе зависимости. Она является наиболее полной с точки зрения реализации различных типов связей и ограничений целостности. Но т.к. наборы организованы с помощью физических ссылок, в этой модели не обеспечивается физическая независимость данных: изменение структуры данных требует изменения логики приложения [1].

Реляционная модель данных и представляет собой набор отношений, изменяющихся во времени. Существуют три части реляцонной базы данных [2]:

— структурная часть описывает, из каких объектов состоит реляционная модель;

— целостная часть определяет базовые требования целостности;

— манипуляционная часть описывает способы манипулирования данными.

Реляционная модель данных основана на понятии отношения — информационной модели реального объекта предметной области, формально представленной множеством однотипных кортежей. Кортеж отношения соответствует экземпляру объекта, свойства которого определяются значениями соответствующих атрибутов (полей) кортежа. Кортежи отношений могут быть связаны между собой с помощью внешних ключей — ссылок на соответствующие атрибуты.

В качестве модели данных была выбрана реляционная модель, поскольку в ней обеспечена физическая независимость данных и возможно реализовать отношение многие ко многим.

Вывод

Соревновательная игра была формализована. Проведён анализ существующих решений. Сущности базы данных были формализованы. Была выбрана реляционная модель данных в силу физической независимости данных, а также возможности реализовать отношение многие ко многим.

2 Конструкторская часть

Поскольку база данных соревновательной игры является общей для всех игроков, программное обеспечение должно обеспечить Web-интерфейс взаимодействия с базой данных. Дополнительно, возможно предусмотреть консольный интерфейс для проверки базового функционала.

2.1 Описание сущностей базы данных

Используются следующие сокращения для обозначения ограничений на поля сущности:

— PK — первичный ключ;

— FK — внешний ключ;

— U — значение уникально в рамках таблицы;

— NN — пустое значение поля недопустимо.

В таблицах 2.1-2.6 указаны описания полей таблиц и ограничения целостности.

Таблица 2.1 — Описание таблицы account

Поле	Тип	Описание	Ограничения
ID	int	Идентификатор	PK
login	varchar(32)	Логин игрока	U
password_hash	varchar	Хэш пароля	
email	varchar	Почта	
privilegy_level	int	Категория пользователя	
profile_image	bytea	Изображение профиля	
description	varchar	Описание профиля	

Дополнительные ограничения к таблице 2.1: login не содержит пробелов. Для обеспечения безопасности, поле "хэш пароля"у таблицы аккаунтов должно быть недоступно для всех категорий пользователей.

Таблица 2.2 — Описание таблицы competition

Поле	Тип	Описание	Ограничения
ID	int	Идентификатор	PK
competition_name	varchar(64)	Название соревнования	NN
description	varchar(128)	Описание соревнования	
start_time	timestamp	Время начала соревнования	NN
end_time	timestamp	Время конца соревнования	NN
level_data	bytea	Данные об уровне соревнования	
has_ended	bool	Закончилось ли соревнование? (выданы ли награды)	NN

Дополнительные ограничения к таблице 2.2: время конца всегда больше времени начала.

Дополнительные ограничения к таблице 2.3: пара значений competition_ID и account_ID уникально в рамках таблицы.

Таблица 2.3 — Описание таблицы player_participation

Поле	Тип	Описание	Ограничения
competition_ID	int	Идентификатор соревнования	FK, NN
account_ID	int	Идентификатор игрока	FK, NN
score	int	Очки	NN
last_update_time	timestamp	Последнее время обновления результата	

Таблица 2.4 — Описание таблицы reward_description

Поле	Тип	Описание	Ограничения
ID	int	Идентификатор	PK
reward_name	varchar(64)	Название награды	NN
description	varchar(128)	Описание награды	
icon_image	bytea	Изображение награды	
ingame_data	bytea	Представление награды в игре	

Тип condition_type_enum является перечисляемым типом, возможными значениями которого являются ’rank’ и ’place’.

Таблица 2.5 — Описание таблицы competition_reward

Поле	Тип	Описание	Ограничения
ID	int	Идентификатор	PK
reward_description_id	int	Идентификатор описания награды	FK, NN
competition_id	int	Идентификатор соревнования	FK, NN
condition_type	condition_type_enum	Тип критерия выдачи наград	NN
min_place	int	Минимальное место	
max_place	int	Максимальное место	
min_rank	decimal(4,3)	Минимальный ранг	
max_rank	decimal(4,3)	Максимальный ранг	

Дополнительные ограничения к таблице 2.5:

— если condition_type = rank, то соблюдается соотношение $0 \leq min\_rank \leq max\_rank \leq 1$ и соответствующим полям присвоено не пустое значение.

— если condition_type = place, то соблюдается соотношение $1 \leq min\_place \leq max\_place$ и соответствующим полям присвоено не пустое значение.

Таблица 2.6 — Описание таблицы player_reward

Поле	Тип	Описание	Ограничения
ID	int	Идентификатор	PK
reward_description_id	int	Идентификатор описания награды	FK, NN
player_id	int	Идентификатор игрока	FK, NN
competition_id	int	Идентификатор соревнования	FK
creation_date	timestamp	Дата создания награды	NN

На рисунке 2.1 представлена схема базы данных соревновательной игры.

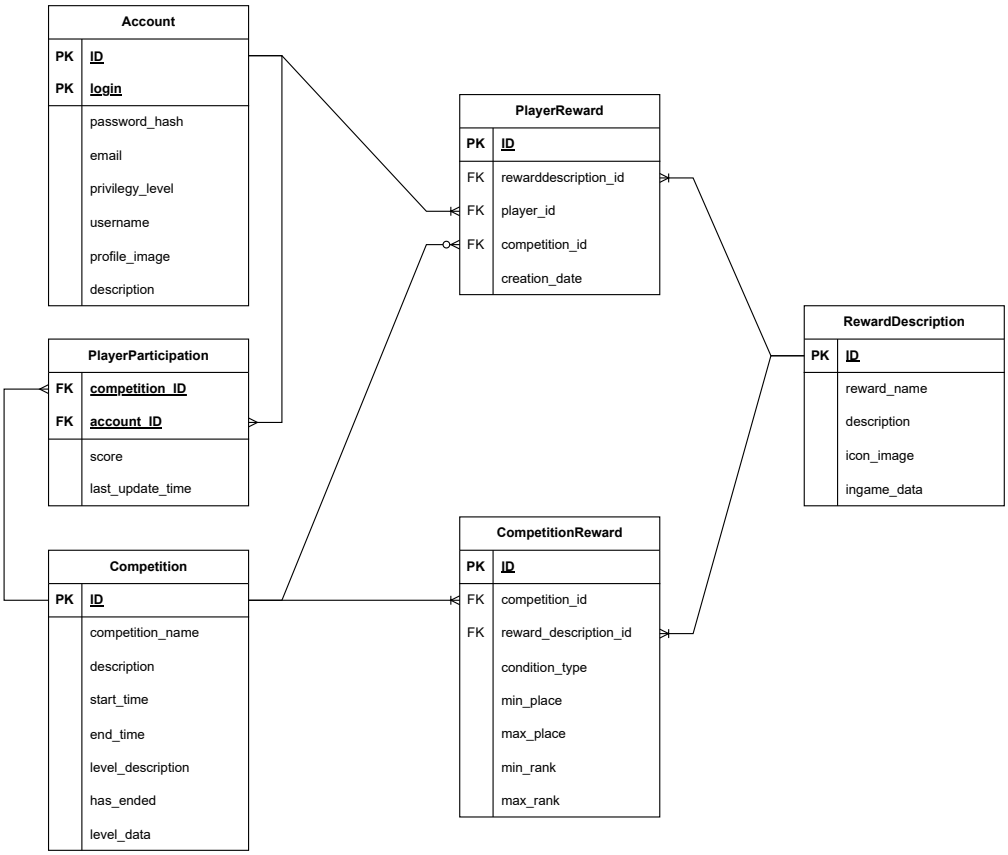

Рисунок 2.1 — Схема базы данных соревновательной игры

2.2 Функциональная модель

При истечении срока соревнования, игрокам должны быть выданы награды, исходя из таблицы лидеров. Награды соревнования выдаются всем участникам в таблице лидеров, которые подошли под критерий выдачи награды. После этого, для соревнования устанавливается, что награды за него выданы, чтобы избежать повторных вознаграждений.

Схема хранимой процедуры выдачи наград представлена на рисунках 2.2, 2.3 2.4.

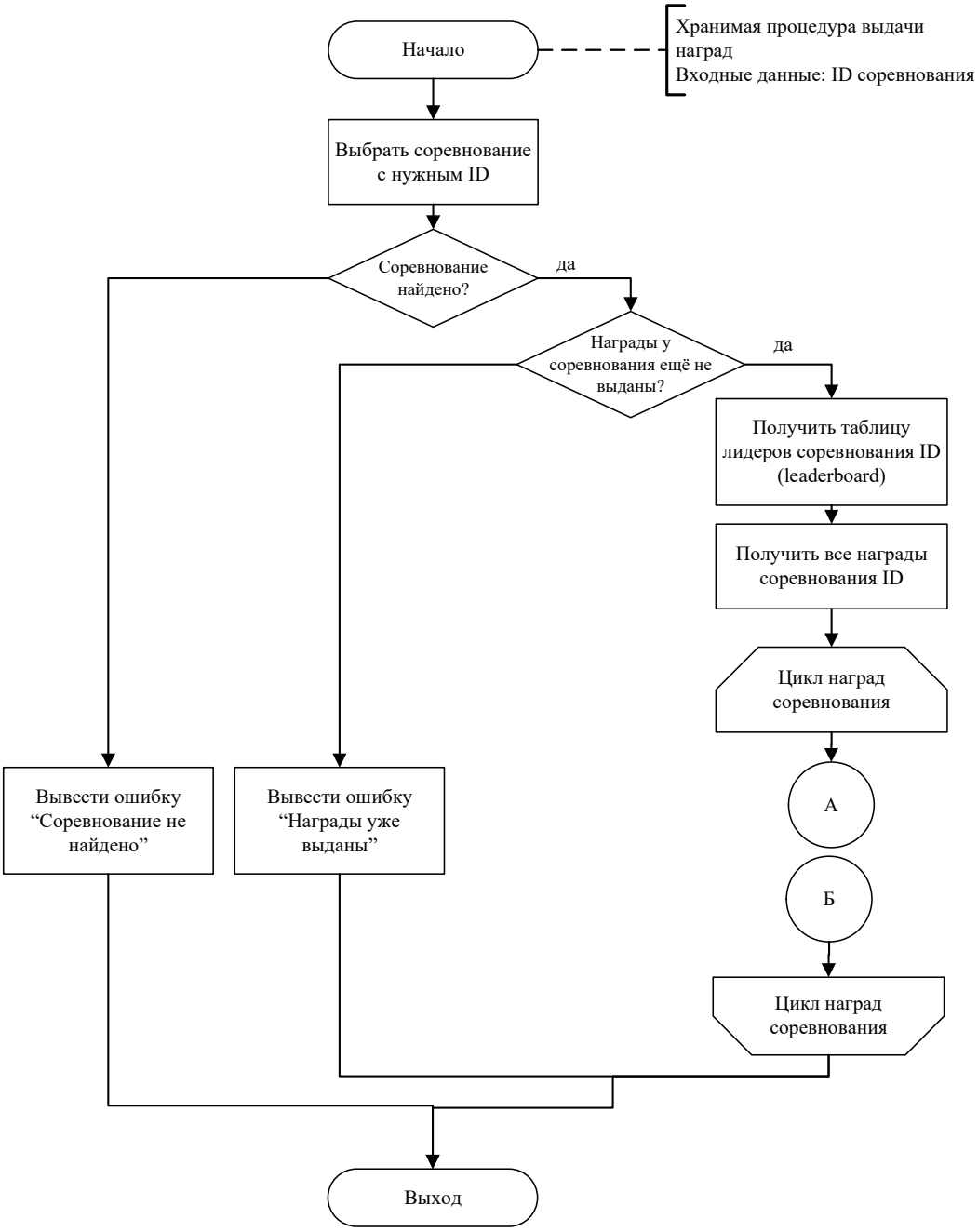

Рисунок 2.2 — Хранимая процедура выдачи наград

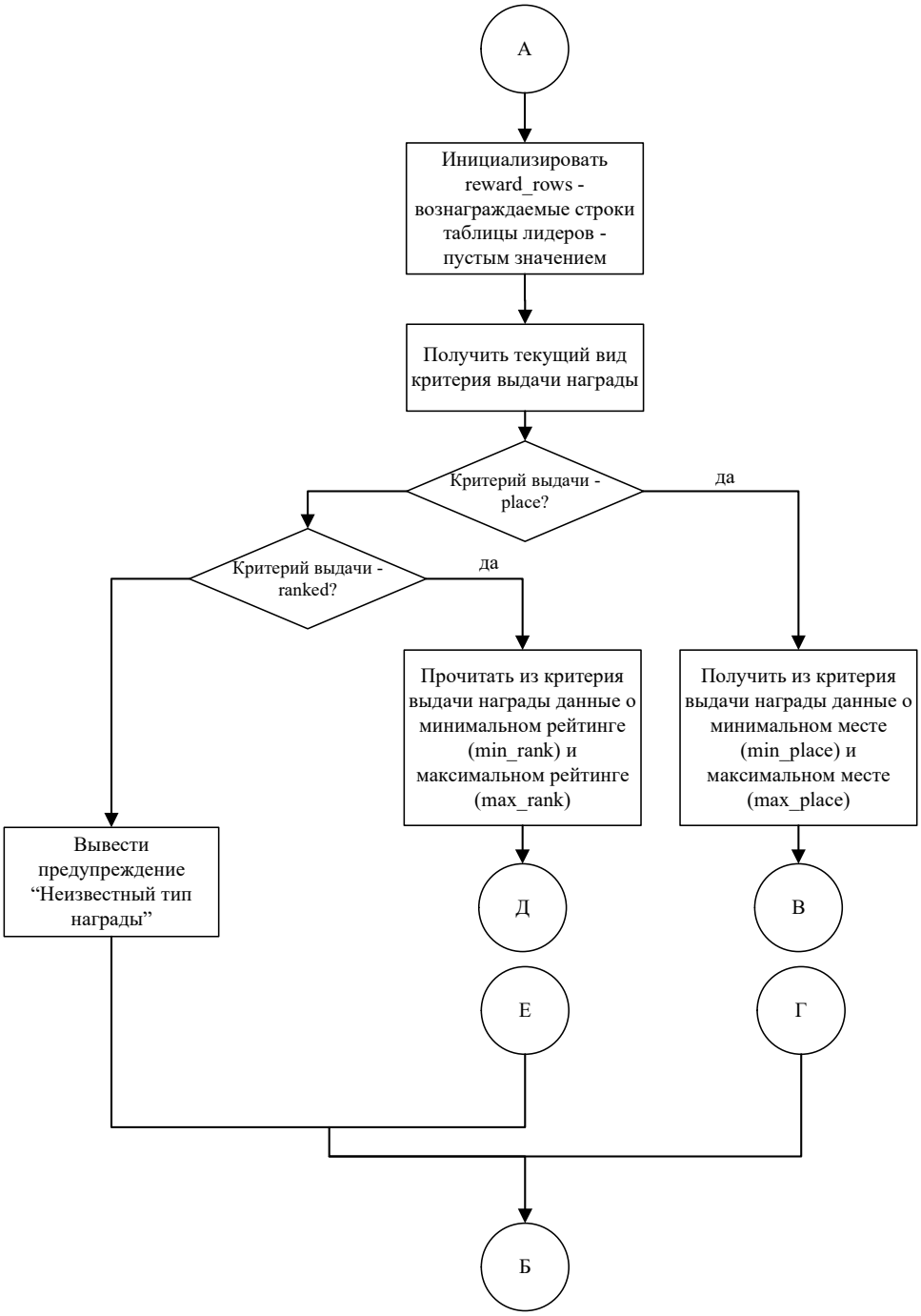

Рисунок 2.3 — Хранимая процедура выдачи наград

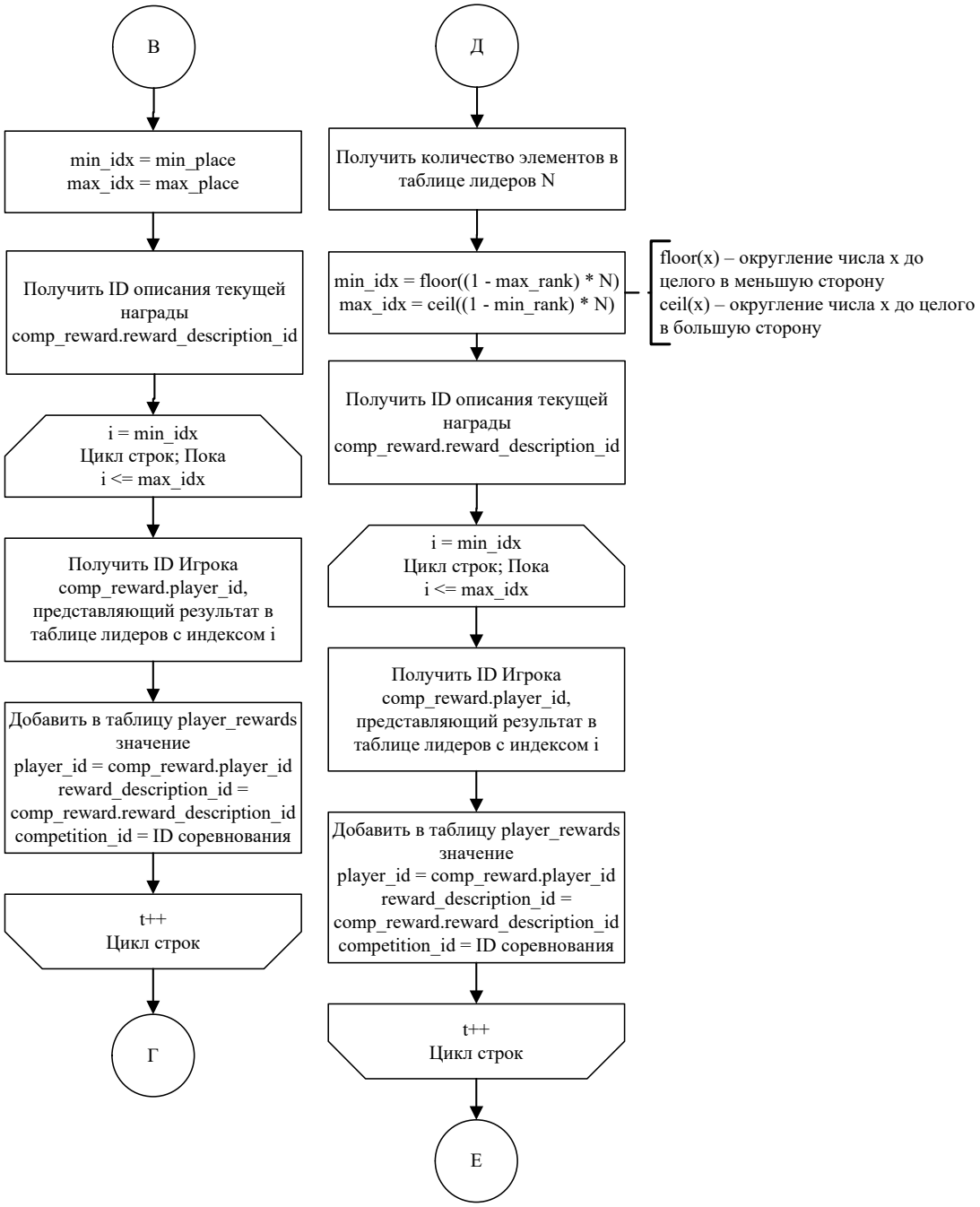

Рисунок 2.4 — Хранимая процедура выдачи наград

Доступ на чтение поля password_hash таблицы account ограничен. Следует определить функцию на уровне базы данных, которая сравнивает хэши паролей, чтобы предоставить возможность определять правильность пароля. Схема представлена на рисунке 2.5

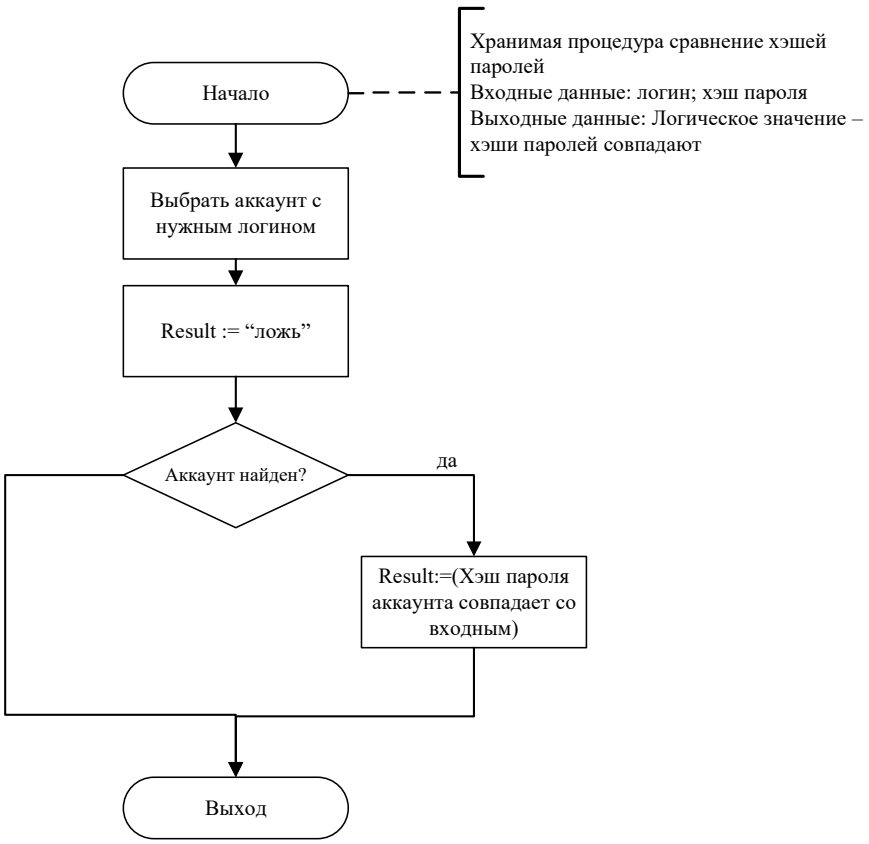

Рисунок 2.5 — Хранимая процедура сравнения хэшей паролей

Поскольку соревнование автоматически завершается по истечению срока, в программе должен быть предусмотрен отдельный процесс, который подключается к базе данных и вызывает хранимую процедуру выдачи наград, что требует дополнительного компонента постановки задач с отложенным выполнением. Процессу, который стоит за данным компонентом, назначен термин "демон выдачи наград".

У администратора имеется возможность загрузить на сервер изображение для типа награды в виде файла. Для обеспечения целостности, должна быть выполнена проверка содержимого, что данный файл является изображением.

2.3 Ролевая модель

В рамках задачи было выделено четыре роли на уровне базы данных:

— гость,

— игрок,

— админинстратор,

— демон выдачи наград.

Гость имеет следующие возможности:

— читать данные из таблиц competition, competition_reward, player_par-ticipation, reward_descri

— читать из таблицы account все поля, кроме password_hash;

— добавлять новые данные в таблицу account (создавать новый аккаунт);

— вызвать хранимую процедуру check_password_hash.

Игрок имеет все возможности, что имеет гость. Дополнительно, он имеет следующие возможности:

— обновлять таблицу account по полям username, description, profile_image;

— добавлять данные и обновлять таблицу player_participation;

— читать таблицу player_reward

Админ имеет все возможности, что имеет гость. Дополнительно, он имеет следующие возможности:

— удалять данные из player_participation;

— добавлять, изменять, удалять данные из player_reward;

— добавлять, изменять данные из таблиц competition, competition_reward, reward_description.

Демон выдачи наград имеет следующие возможности:

— вызвать хранимую процедуру grant_rewards();

— читать и изменять данные из таблицы competition;

— читать данные из таблиц player_participation, competition_reward;

— добавлять данные в таблицу player_reward.

Вывод

Спроектирована архитектура базы данных и ограничения целостности. Спроектированы хранимые процедуры выдачи наград и проверки хэша пароля.

3 Технологическая часть

В качестве реляционной СУБД была выбрана PostgreSQL [3], поскольку в ней представлены все необходимые возможности, в частности создание хранимых процедур.

Для реализации был выбран язык C# версии 12, поскольку в нём представлены все необходимые возможности для создания Web-интерфейса взаймодействия с приложением, в частности, с помощью фреймворка ASP.NET Core [4].

Для реализации внедрения зависимостей использовалась библиотека Microsoft.Dependency-Injection [5]

Для связи объектов базы данных с кодом программы использовался фреймворк EF Core [6].

Для обработки изображений была выбрана библиотека Magick.NET [7].

Для хранения и выполнения отложенных задач использовалась библиотека Quartz.NET [8].

3.1 Компоненты программного обеспечения

На рисунке 3.1 представлена диаграмма компонентов приложения.

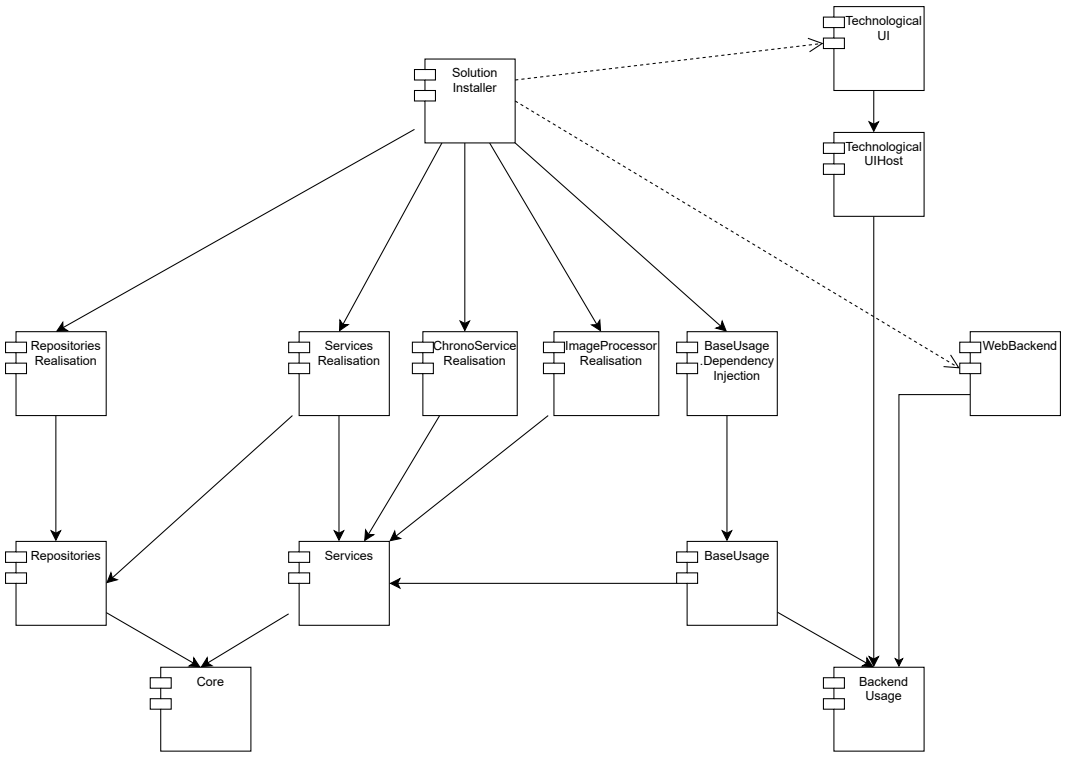

Рисунок 3.1 — Диаграмма компонентов приложения

Модуль Core содержит определение базовых объектов в виде классов, с которыми работает компонент доступа к базе данных и компонент бизнес-логики.

Модуль Repositories содержит интерфейсы, определяющие взаимодействия с базой данных.

Модуль RepositoriesRealisation содержит реализацию уровня взаимодействия с базой данных.

Модуль Services содержит интерфейсы, определяющие уровень бизнес-логики, а также сервиса постановки задач с отложенным ожиданием и сервиса обработки изображений.

Модуль ChronoServiceRealisation содержит реализацию сервиса постановки задач с отложенным ожиданием.

Модуль ImageProcessorRealisation содержит реализацию сервиса обработки изображений.

Модуль ServicesRealisation содержит реализацию уровня бизнес-логики на основе интерфейсов компонентов доступа к базе данных.

Модуль BackendUsage содержит интерфейсы, определяющие взаимодействие с приложением в общем виде.

Модуль BaseUsage содержит реализацию взаимодействия с приложением в общем виде на основе интерфейсов уровня бизнес-логики.

Модуль WebBackend реализует сетевое взаимодействие на основе интерфейсов взаимодействия с приложением в общем виде.

Модуль TechnologicalUIHost реализует взаимодействие в виде консольного интерфейса на основе интерфейсов взаимодействия с приложением в общем виде. Также он определяет интерфейс, описывающий консольный ввод-вывод.

Модуль TechnologicalUI содержит реализацию консольного ввода-вывода и предоставляет консольный интерфейс.

Модуль SolutionInstaller настраивает внедрение зависимостей, сопоставляя всем интерфейсам взаимодействия базы данных, бизнес – логики и т.д. с их реализациями.

3.2 Тестирование

Для реализации компонента бизнес – логики были разработаны модульные тесты. Для имитирования объектов слоя доступа к базы данных использовалась библиотека Moq [9]. Составлены 76 тестов, обеспечивающие покрытие кода — 82.31%. Все тесты пройдены успешно.

Для реализации компонента взаимодействия с базой данных были разработаны интеграционные тесты. Для создания независимых образов базы данных использовалась библиотека Testcontainers [10]. Составлено 74 теста, обеспечивающие покрытие кода — 70.34%. Все тесты пройдены успешно.

Для реализации компонента обработки изображений были разработаны функциональные тесты. На вход компоненту поступают файлы из папки Tests с различными названиями:

— если файл имеет префикс neg, то обработка должна выдать исключение; результат не сохраняется.

— в противном случае, ожидается, что обработка изображения должна пройти успешно; при этом результат сохраняется в соседнюю папку Results.

Составлено 9 тестов, которые обеспечивают 100% покрытие кода класса обработки изображений. Все тесты пройдены успешно.

Для реализации компонента отложенных задач были разработаны интеграционные тесты. Составлено 7 тестов, которые обеспечивают 100% покрытие кода класса отложенных задач. Все тесты пройдены успешно.

3.3 Примеры работы

На рисунках 3.2 – 3.5 приведены примеры работы программы, используя Web — интерфейс.

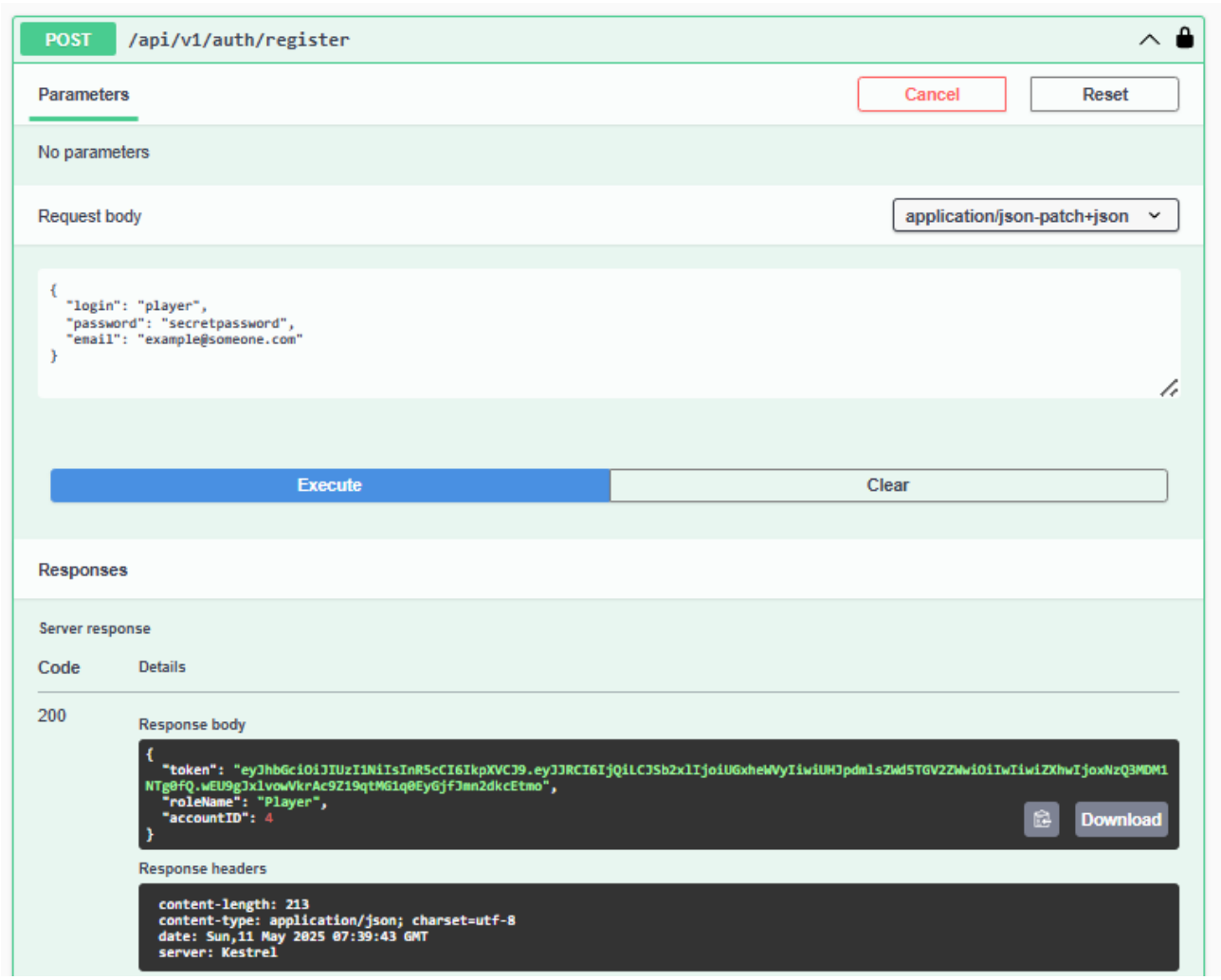

Рисунок 3.2 — Демонстрация регистрации в приложении

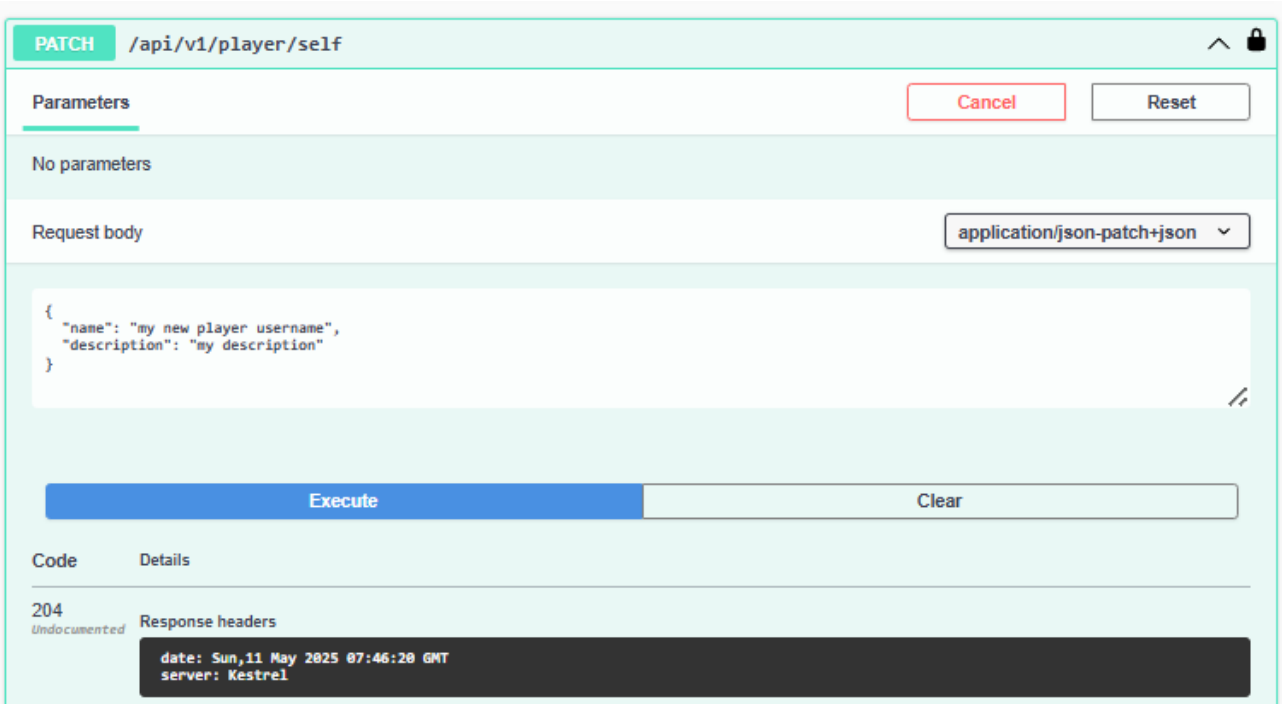

Рисунок 3.3 — Демонстрация изменение профиля

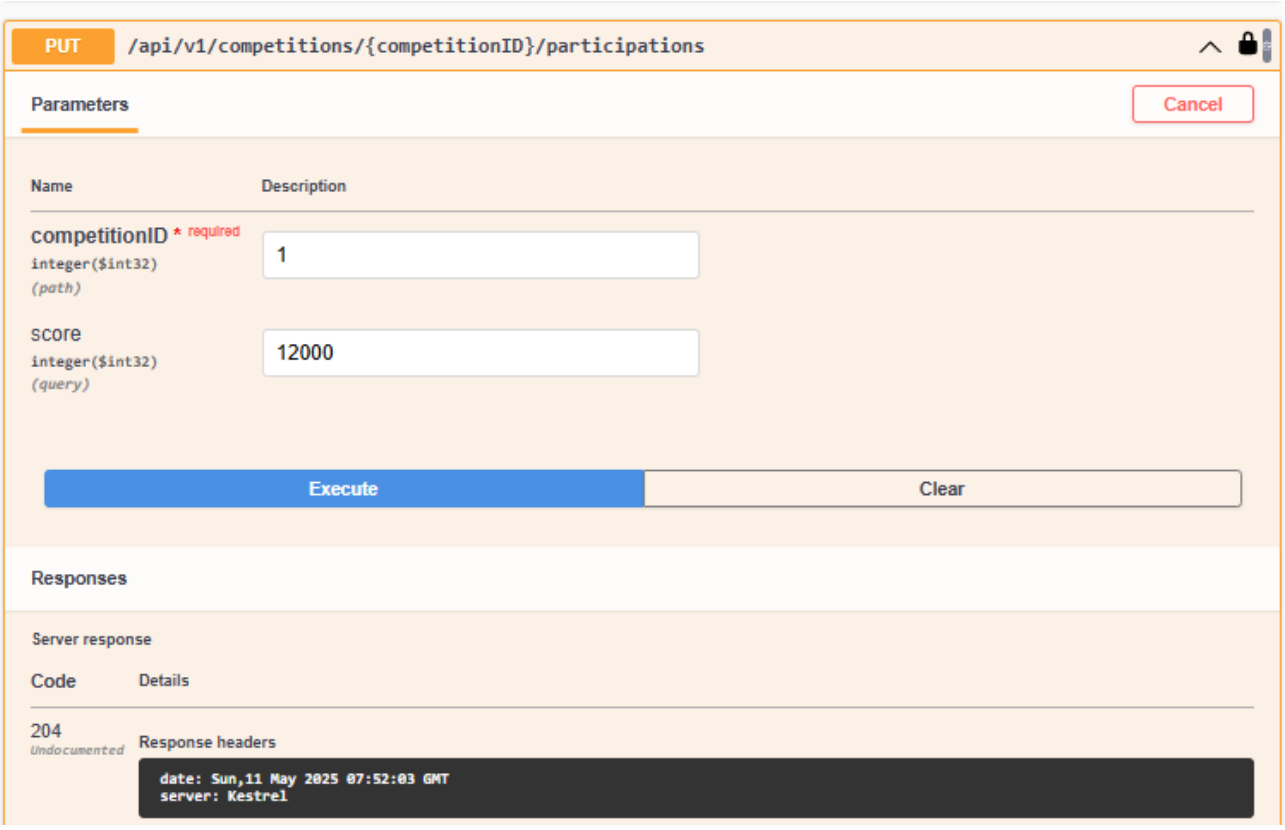

Рисунок 3.4 — Демонстрация участия в соревновании

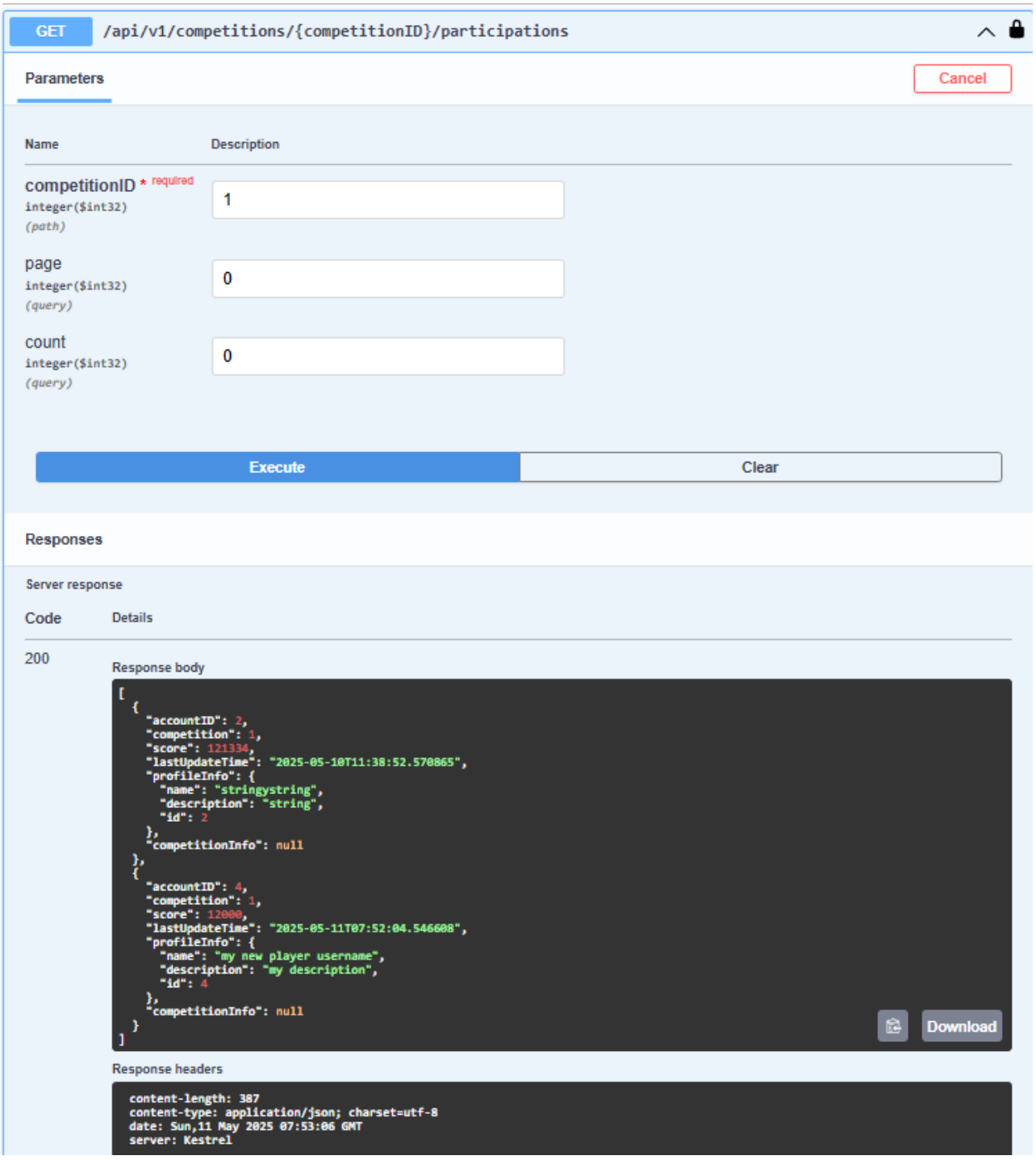

Рисунок 3.5 — Демонстрация просмотра таблицы лидеров

Вывод

Были выбраны средства реализации базы данных и приложения. Реализованы все компоненты Описаны методы тестирования разработанного функционала и разработать тесты для проверки корректности работы приложения. Все тесты пройдены успешно.

4 Исследовательская часть

Цель исследования - сравнить зависимость времени выдачи наград от количества вознаграждаемых игроков в двух случаях: при помощи хранимой процедуры и при помощи обычных запросов.

Исследование выполнялись на вычислительной машине со следующими характеристиками:

— процессор: Intel(R) Core(TM) i7-7700K 4.20 ГГц 4.20 ГГц. [11];

— количество ядер процессора: 4, количество потоков: 8;

— объём оперативной памяти: 32 ГБ.

Описание способов выдачи наград:

— на стороне базы данных — вызов хранимой процедуры;

— на стороне приложения — реализация логики добавления наград с помощью Entity Framework Core.

Для создания различных входных данных, создаются независимые варианты базы данных, инициализируемые различными параметрами. Для создания независимых образов PostgreSQL использовалась библиотека Testcontainers.Postgresql [10]

Исследование проводилось в два этапа: подготовка данных и замеры времени.

На каждую подготовку данных выделялся отдельный образ PostgreSQL, которым был инициализирован сценарием (см. приложение А). Программа заполняла СУБД нужными данными, после чего делала снимок базы данных при помощи pg_dump [12] в файл. Для заполнения базы данных случайными данными использовалась библиотека Bogus [13].

Если снимок данных подготавливался для N выдаваемых наград, генерировалось:

— 1 соревнование;

— $N/2$ участников и записей в таблицы лидеров;

— 5 критериев наград, каждый из которых вознаграждает $\frac{N}{5}$ участников.

Критерии выдачи награды генерировались с равной вероятностью двух возможных видов: по диапазону доли в таблице лидеров, по диапазону мест. Параметр $N \in \{250, 500, 1000, 2000, 3000, 4000, 5000\}$.

На каждый замер времени выделялся отдельный образ PostgreSQL, который был инициализирован снимком базы данных, полученным на этапе подготовке данных. После этого, выполнялся один замер времени выдачи наград на стороне БД/приложения. Для измерения реального времени работы использовался класс Stopwatch [14].

Для каждого параметра N подготавливалось 10 вариантов входных данных. Для каждого варианта входных данных было произведено 30 замеров. Итоговое время работы для одного параметра N является усреднённым временем по 300 значениям.

В таблице 4.1 приведены результаты исследования.

Таблица 4.1 — Время выполнения от количества дополнительных потоков

Кол-во наград	Время, мс	
	На стороне базы данных	На стороне приложения
250	26.2485	43.8758
500	35.4606	64.9545
1000	45.4333	107.0000
2000	67.2545	171.0727
3000	90.6091	242.9091
4000	116.9364	342.6629
5000	128.8848	365.5333

При помощи метода наименьших квадратов были найдены коэффициенты полинома 3 степени, аппроксимирующие данные точки. Время на стороне базы данных аппроксимирует функция формулы (4.1); время на стороне приложения — формула (4.2).

$$t_1(N) = -5.774 \cdot 10^{-10} \cdot N^3 + 3.743 \cdot 10^{-6} \cdot N^2 + 1.684 \cdot 10^{-2} \cdot N + 24.066 \tag{4.1}$$

$$t_2(N) = -2.76 \cdot 10^{-9} \cdot N^3 + 1.888 \cdot 10^{-5} \cdot N^2 + 4.087 \cdot 10^{-2} \cdot N + 38.777 \tag{4.2}$$

Исходя из того, что коэффициенты при N^3 и N^2 достаточно малы ($< 10^{-4}$), зависимость времени выдачи наград от количества выдаваемых наград близка к линейной зависимости.

По результатам исследования, время работы на стороне сервера меньше, чем время работы на стороне приложения. При этом, линейный коэффициент роста времени на стороне приложения в 2.427 раза больше, чем на стороне сервера.

По данным таблицы 4.1 построены графики, представленные на рисунке 4.1.

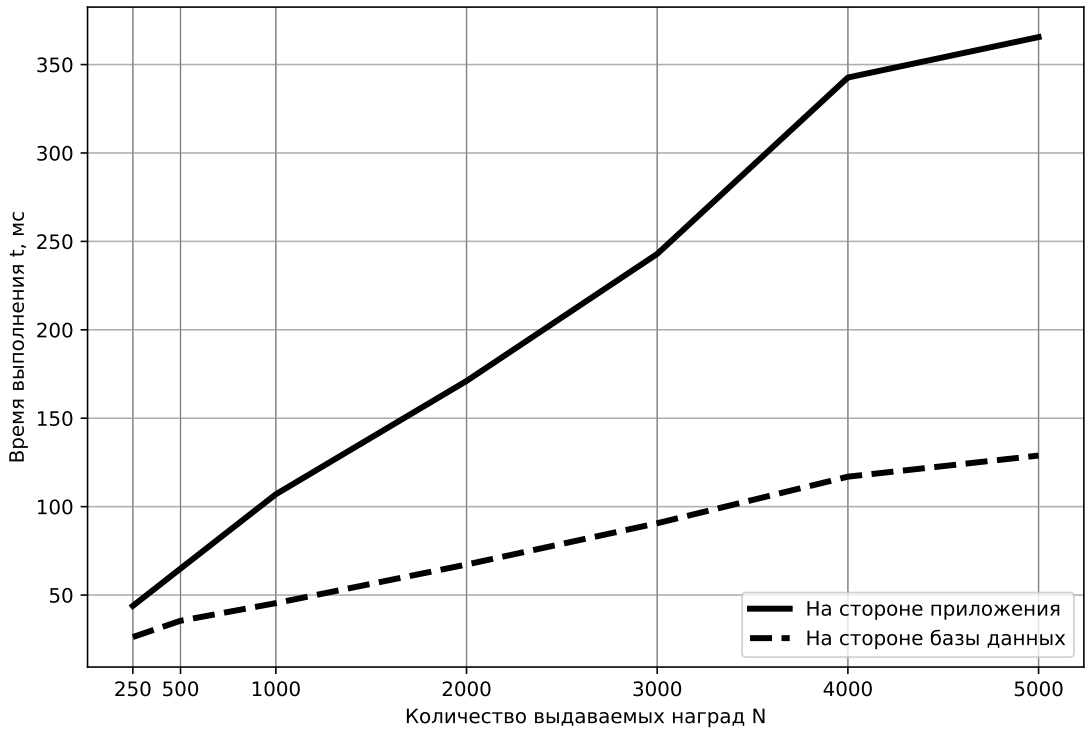

Рисунок 4.1 — Зависимость времени выдачи наград от количества выдаваемых наград

Вывод

Проведено исследование времени выдачи наград от количества вознаграждаемых игроков в двух случаях: при помощи хранимой процедуры и при помощи обычных запросов. Время работы на стороне сервера меньше, чем время работы на стороне приложения. Зависимость времени от количества наград в обоих случаях близка к линейной, при этом линейный коэффициент роста времени на стороне приложения в 2.427 раза больше, чем на стороне сервера. Следовательно, использование хранимой процедуры для вознаграждения пользователей целесообразно для увеличения скорости работы приложения.

ЗАКЛЮЧЕНИЕ

Были выполнены следующие задачи:

— соревновательную игру была формализована, предметная область описана;

— проведён анализ существующих решений;

— формализованы сущности базы данных;

— архитектуру базы данных спроектирована вместе с ограничениями целостности;

— процедуру выдачи наград была спроектирована;

— средства реализации базы данных и приложения;

— описан интерфейс доступа к базе данных;

— выбраны средства реализации базы данных и приложения, после чего они были реализованы;

— описаны методы тестирования разработанного функционала, разработаны тесты для проверки корректности работы приложения.

Исследование показало, что использование хранимой процедуры для вознаграждения пользователей целесообразно для увеличения скорости работы — линейный коэффициент роста времени на стороне приложения в 2.427 раза больше, чем на стороне сервера.

Была разработана база данных соревновательной игры.

Поставленные задачи выполнены. Цель работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Карпова И. П.* Базы данных : Учебное пособие. — Санкт-Петербург : Питер, 2013. — С. 240. — ISBN 978-5-496-00546-3.
2. *Мирошниченко Г. А.* Реляционные базы данных. Практические приемы оптимальных решений. — БХВ-Петербург, 2005.
3. Документация PostgreSQL [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/ (дата обращения: 05.05.2025).
4. Документация ASP.NET Core [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0 (дата обращения: 08.05.2025).
5. Документация по внедрению зависимостей [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection (дата обращения: 07.05.2025).
6. Документация Entity Framework Core [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/ef/core/ (дата обращения: 08.05.2025).
7. Документация библиотеки Magick.NET [Электронный ресурс]. — Режим доступа: https://github.com/dlemstra/Magick.NET/tree/main/docs (дата обращения: 06.05.2025).
8. Документация библиотеки Quartz.NET [Электронный ресурс]. — Режим доступа: https://www.quartz-scheduler.net/documentation/ (дата обращения: 05.05.2025).
9. Документация библиотеки Moq [Электронный ресурс]. — Режим доступа: https://github.com/devlooped/moq/wiki/Quickstart (дата обращения: 07.05.2025).
10. Документация Testcontainers.Postgresql [Электронный ресурс]. — Режим доступа: https://dotnet.testcontainers.org/modules/postgres/ (дата обращения: 05.05.2025).
11. Спецификация процессора Intel Core i7 [Электронный ресурс]. — Режим доступа: https://www.intel.com/content/www/us/en/products/sku/97129/intel-core-i77700k-processor-8m-cache-up-to-4-50-ghz/specifications.html (дата обращения: 15.10.2024).
12. Документация команды pgdump в PostgreSQL [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/current/app-pgdump.html (дата обращения: 05.05.2025).
13. Описание библиотеки Bogus [Электронный ресурс]. — Режим доступа: https://www.nuget.org/packages/bogus (дата обращения: 05.05.2025).
14. Документация класса System.Diagnostics.Stopwatch [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/api/system.diagnostics.stopwatch?view=net-8.0 (дата обращения: 05.05.2025).

ПРИЛОЖЕНИЕ А

Сценарий создания базы данных приведён в листинге 4.1

Листинг 4.1 — Сценарий создания базы данных

```
create table if not exists account(
    id int generated always as identity primary key,
    login varchar(32) unique not null,
    username varchar(32) not null,
    email varchar(32),
    password_hash varchar,
    privilegy_level int,
    description varchar(128),
    profile_image bytea,
    check (login not like '% %')
);

create table if not exists reward_description(
    id int generated always as identity primary key,
    reward_name varchar(64) not null,
    description varchar(128),
    icon_image bytea,
    ingame_data bytea
);

create table if not exists competition(
    id int generated always as identity primary key,
    competition_name varchar(64) not null,
    description varchar(128),
    start_time timestamp not null,
    end_time timestamp not null,
    level_data bytea,
    has_ended bool not null default(false),
    check(end_time > start_time)
);

create table if not exists player_participation(
    competition_id int references competition(id) not null,
    account_id int references account(id) not null,
    score int not null,
    last_update_time timestamp not null default(now())
);

alter table player_participation add constraint comp_uniqueness unique (competition_id,
    account_id);

create type condition_type_enum as enum ('place', 'rank');
```

```
42
43 create table competition_reward (
44     id int generated always as identity primary key,
45     reward_description_id int references reward_description(id) not null,
46     competition_id int references competition(id) not null,
47     condition_type condition_type_enum not null,
48     min_place int,
49     max_place int,
50     min_rank decimal(4,3),
51     max_rank decimal(4,3)
52 );
53
54 alter table competition_reward
55 add constraint chk_place_min check (
56     condition_type <> 'place' or (min_place is not null and min_place >= 1)
57 ),
58 add constraint chk_place_max check (
59     condition_type <> 'place' or (max_place is not null and max_place >= min_place)
60 ),
61 add constraint chk_rank_min check (
62     condition_type <> 'rank' or (min_rank is not null and min_rank >= 0)
63 ),
64 add constraint chk_rank_max check (
65     condition_type <> 'rank' or (max_rank is not null and max_rank <= 1)
66 ),
67 add constraint chk_rank_range check (
68     condition_type <> 'rank' or (max_rank >= min_rank)
69 );
70
71 create table if not exists player_reward(
72     id int generated always as identity primary key,
73     reward_description_id int references reward_description(id) not null,
74     player_id int references account(id) not null,
75     competition_id int references competition(id),
76     creation_date timestamp default(now())
77 );
78
79
80
81 create or replace function grant_place_rewards(sortedtable refcursor, min_val int, max_val
       int)
82 RETURNS SETOF player_participation -- Use SETOF to return multiple rows
83 LANGUAGE plpgsql
84 AS $$
85 DECLARE
86     row_data player_participation;
87 BEGIN
```

```
88      -- Move to the starting position
89      MOVE ABSOLUTE (min_val - 1) IN sortedtable;
90      -- Fetch and return rows in a loop
91      FOR i IN 1..(max_val - min_val + 1) LOOP
92          FETCH NEXT FROM sortedtable INTO row_data;
93          EXIT WHEN NOT FOUND;
94          RETURN NEXT row_data;
95      END LOOP;
96      RETURN;
97  END;
98  $$;
99
100 create or replace function grant_rank_rewards(sortedtable refcursor, min_rank float,
        max_rank float)
101 RETURNS SETOF player_participation
102 LANGUAGE plpgsql
103 AS $$
104 DECLARE
105     row_cnt int := 0;
106     min_val int := 0;
107     max_val int := 0;
108 BEGIN
109     -- Move to the end
110     MOVE ABSOLUTE 0 IN sortedtable;
111     MOVE FORWARD ALL FROM sortedtable;
112     -- Get the current position (which is the count)
113     GET DIAGNOSTICS row_cnt = ROW_COUNT;
114     MOVE ABSOLUTE 0 IN sortedtable;
115     --RAISE NOTICE 'deepsec row count %', row_cnt;
116     min_val := floor((1.0 - max_rank) * row_cnt) + 1;
117     max_val := ceil((1.0 - min_rank) * row_cnt) + 1;
118     --RAISE NOTICE '% %', min_val, max_val;
119     RETURN QUERY (select * from grant_place_rewards(sortedtable, min_val, max_val));
120 END;
121 $$;
122
123 -- deepseek my beloved goat
124 CREATE OR REPLACE PROCEDURE grant_rewards(comp_id int)
125 LANGUAGE plpgsql
126 AS $$
127 DECLARE
128     creward competition_reward%rowtype;
129     comp_valid_id int := null;
130     is_granted bool := false;
131     reward_type condition_type_enum;
132     place_cursor REFCURSOR := 'place_cursor';
133     reward_cursor REFCURSOR := 'reward_cursor';
```

```
    player_rec player_participation%rowtype;
BEGIN
    -- Check if rewards already granted
    SELECT c.id, c.has_ended
    INTO comp_valid_id, is_granted
    FROM competition c
    WHERE c.id = comp_id;
    IF comp_valid_id is null THEN
        RAISE EXCEPTION 'There is no such competition' USING HINT = 'No competition with
            ID';
    ELSIF is_granted THEN
        RAISE EXCEPTION 'Rewards have already been granted' USING HINT = 'Rewards already
            granted.';
    ELSE
        UPDATE competition c SET has_ended = true WHERE c.id = comp_id;
        -- Open the base cursor for place-based ordering
        OPEN place_cursor SCROLL FOR
            SELECT * FROM player_participation p
            WHERE p.competition_id = comp_id
            ORDER BY p.score DESC, p.last_update_time ASC;

        FOR creward IN
            SELECT * FROM competition_reward c WHERE c.competition_id = comp_id
        LOOP
            reward_type := creward.condition_type;
            CASE
                WHEN reward_type = 'rank' THEN
                    OPEN reward_cursor FOR
                        SELECT * FROM grant_rank_rewards(place_cursor, creward.min_rank,
                            creward.max_rank);

                WHEN reward_type = 'place' THEN
                    OPEN reward_cursor FOR
                        SELECT * FROM grant_place_rewards(place_cursor, creward.min_place,
                            creward.max_place);

                ELSE
                    RAISE WARNING 'Processing %: Unrecognized reward type %; skipping',
                        creward.id, reward_type;
                    CONTINUE; -- Skip to next reward
            END CASE;
            -- Process the reward cursor (common for both types)
            LOOP
                FETCH reward_cursor INTO player_rec;
                EXIT WHEN NOT FOUND;

                INSERT INTO player_reward(reward_description_id, player_id, competition_id)
```

```
                VALUES (creward.reward_description_id, player_rec.account_id, comp_id);
            END LOOP;

            -- Clean up and reset for next iteration
            CLOSE reward_cursor;
            MOVE ABSOLUTE 0 IN place_cursor;
        END LOOP;

        -- Final clean up
        CLOSE place_cursor;
    END IF;
END;
$$;

CREATE OR REPLACE FUNCTION public.check_password_hash(
    p_login VARCHAR,
    p_input_hash VARCHAR
)
RETURNS BOOLEAN AS $$
DECLARE
    v_stored_hash VARCHAR;
    v_result BOOLEAN;
BEGIN
    SELECT password_hash FROM account WHERE login = p_login INTO v_stored_hash;
    SELECT v_stored_hash = p_input_hash INTO v_result;

    RETURN v_result;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

-- GUEST

REVOKE SELECT, UPDATE, DELETE ON account FROM PUBLIC;
CREATE ROLE guest WITH LOGIN PASSWORD 'guest_password';
GRANT SELECT ON competition, competition_reward, player_participation, reward_description
    TO guest;
GRANT SELECT (id, login, username, email, privilegy_level, description, profile_image) ON
    account TO guest;
GRANT INSERT (login, username, email, password_hash, privilegy_level) ON account TO guest;
GRANT EXECUTE ON FUNCTION check_password_hash(varchar, varchar) TO guest;
ALTER ROLE guest WITH INHERIT;

-- PLAYER

CREATE ROLE player WITH LOGIN PASSWORD 'player_password';
GRANT guest TO player;
GRANT UPDATE (username, description, profile_image) ON account TO player;
```

```
221 GRANT SELECT, INSERT, UPDATE ON player_participation TO player;
222 GRANT SELECT ON player_reward TO player;
223 ALTER ROLE player INHERIT;
224
225 -- ADMIN
226
227 CREATE ROLE admin WITH LOGIN PASSWORD 'admin_password';
228 GRANT guest TO admin;
229 GRANT SELECT, DELETE ON player_participation TO admin;
230 GRANT INSERT, UPDATE, DELETE ON player_reward TO admin;
231 GRANT INSERT, UPDATE ON competition, competition_reward, reward_description TO admin;
232 ALTER ROLE admin INHERIT;
233
234 -- REWARD_GRANTER
235
236 CREATE ROLE reward_granter WITH LOGIN PASSWORD 'reward_granter';
237 GRANT EXECUTE ON PROCEDURE grant_rewards(integer) TO reward_granter;
238 GRANT SELECT, UPDATE ON TABLE competition TO reward_granter;
239 GRANT SELECT ON TABLE player_participation TO reward_granter;
240 GRANT SELECT ON TABLE competition_reward TO reward_granter;
241 GRANT INSERT ON TABLE player_reward TO reward_granter;
```

ПРИЛОЖЕНИЕ Б

Презентация состоит из ... слайдов