

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 ОБЩИЙ РАЗДЕЛ.....	5
1.1 Описание предметной области.....	5
1.1.1 Аналогии.....	8
1.2 Сравнение технологий.....	9
1.2.1 Технологии для разработки игры	9
1.2.2 Технологии для разработки сервера	12
1.3 Требования.....	16
2 СПЕЦИАЛЬНЫЙ РАЗДЕЛ.....	17
2.1 Диаграмма прецедентов использования.....	17
2.2 Инфологическая модель.....	18
2.3 Даталогическая модель	19
2.4 Проект основных дисплейных фрагментов	20
2.5 Исключительные ситуации	23
2.6 Разработанные запросы.....	24
2.7 Обеспечение целостности БД.....	26
2.8 Разграничение прав доступа. Защита данных.....	26
2.9 Диаграмма классов сервера и клиента.....	27
2.10 Справочная система.	28
2.11 Тестирование приложения.....	29
2.11.1 Сервер.....	29
2.12 Клиент	32
Заключение	35
Список использованных источников	36
ПРИЛОЖЕНИЕ А Характеристики программного и аппаратного обеспечения.....	37
ПРИЛОЖЕНИЕ Б Руководство системного программиста.	38
ПРИЛОЖЕНИЕ В Листинг SQL-скрипта.	40
ПРИЛОЖЕНИЕ Г Листинг приложений.	43

Перечень сокращений и обозначений

БД – База данных;

JSON – JavaScript Object Notation;

SQL – Structured Query Language;

ПК – Персональный компьютер;

ВВЕДЕНИЕ

Онлайн видеоигры – это многомиллионная индустрия, которая затрагивает огромное количество пользователей на самых разных платформах – от портативных консолей и мобильных устройств до домашних консолей и персональных компьютеров. Онлайн-функционал так или иначе сейчас есть почти во всех современных играх. Он может быть от мультиплеера и общения разных пользователей до монетизации и DRM.

Данный проект актуален только в обучающих целях, так как в нём не планируется достаточно функционала для конкуренции с более популярными играми жанре. Несмотря на это, популярность игр с похожей структурой (Fallout Shelter, Stronghold Kingdoms) позволяет рассматривать потенциал некой «расширенной» версии этого проекта с большим функционалом в коммерческих целях. Также, этот проект можно перенести на мобильные платформы (iOS и Android), где подобные игры более популярны.

Доход индустрии бесплатных мобильных игр сейчас превышает совокупный доход индустрии ПК-приложений и консольных игр. Такой высокий доход является следствием очень низкого порога вхождения, нулевой начальной стоимостью и агрессивной монетизацией. Монетизация бесплатных мобильных игр со схожими механиками может происходить несколькими способами:

- Реклама
- Внутриигровые покупки
- Внутриигровые подписки
- Платная версия приложения

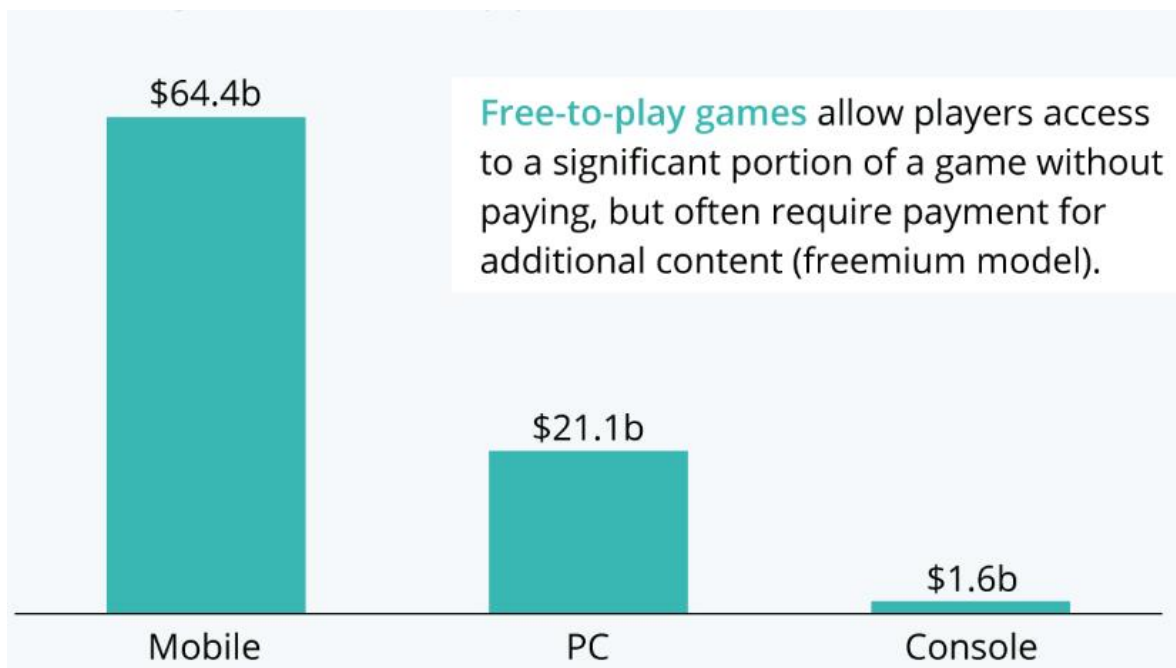


Рисунок 1 – Сравнение монетизации игровых индустрий

1 ОБЩИЙ РАЗДЕЛ

1.1 Описание предметной области

Видеоигра — компьютерная программа, обеспечивающая взаимодействие пользователя с игровыми системами с помощью графического интерфейсам и генерирующая изображение на основе пользовательского ввода.

Видеоигры могут быть классифицированы по нескольким признакам:

– **Жанр.** отражает наиболее существенные признаки игры. Жанр может быть как ярко выраженным (StarCraft – стратегия в реальном времени), так и допускать смешения (Mass Effect – Action-RPG). Также, смешению подвержены отдельные механики (Final Fantasy IV – Active Time Battle (ATB)).

Обычно выделяются следующие жанры:

- **Action** (экшен)
- **Adventure** (приключенческие)
- **Casual** (казуальные игры)
- **MMO**
- **Racing** (гоночные игры)
- **RPG** (ролевые игры)
- **Simulation** (симуляторы)
- **Sports** (спортивные игры)
- **Strategy** (стратегии)

– **Количество игроков.** Игры могут быть однопользовательские (рассчитанной на одного игрока) и многопользовательские (рассчитанной на два и более участника). По типу взаимодействия между игроками, многопользовательские игры можно разделить на кооперативные и соревновательные. В кооперативных играх, несколько игроков сотрудничают для преодоления препятствий или соревнуются против соперников, управляемых искусственным интеллектом. В соревновательных играх, игроки

(или команды игроков) соревнуются между собой. Многопользовательские игры могут вестись как на одном компьютере, так и по сети.

– **Визуальная составляющая.** Игры могут использовать графическое представление (двухмерное или трёхмерное) или использовать текстовый интерфейс.

Предметной областью в данной работе будет являться игра Starships, разработанная для данного курсового проекта.

Starships это игра в жанре симулятора, в которой игрок берёт на себя управление космическим кораблём и отправляет его на различные задания. У игрока нет возможности прямого управления кораблём – результаты заданий рассчитываются на сервере. Игрок может модифицировать и менять части корабля, покупать и продавать запчасти и вещи, собранные на миссиях, нанимать команду, ремонтировать корабль, а также создавать снаряжение для корабля. Большинство этих действий занимают реальное время, так что игра рассчитана на короткие игровые сессии (5-10 минут) несколько раз в день.

Игровой сервер – программное обеспечение, отвечающее за хранение информации об игровом мире и обработку запросов, поступающих как от игроков, так и администратора. Сервер отправляет клиентам достаточно информации для того чтобы они поддерживали свою достаточно точную версию игрового мира.

Виды серверов:

– **Выделенный сервер** – вид хостинга, при котором клиенту целиком предоставляется отдельная физическая машина (в противоположность виртуальному хостингу). Обычно используется для запуска приложений, которые не могут сосуществовать на одном сервере с другими проектами или имеют повышенные требования к ресурсам.

– **Listen server** – система, в которой один из клиентов берёт на себя роль сервера.

– **Peer-to-Peer** – компьютерная сеть, основанная на равноправии участников. Часто в такой сети отсутствуют выделенные серверы, а каждый

узел (peer) является как клиентом, так и выполняет функции сервера. В отличие от архитектуры клиент-сервера, такая организация позволяет сохранять работоспособность сети при любом количестве и любом сочетании доступных узлов. Участниками сети являются все пиры.

Тикрейт (tickrate) – частота, с которой сервер обновляет информацию и синхронизирует игроков. Серверы ограничивают частоту обновления информации для экономии трафика и процессорного времени клиентов и сервера. Обычно тикрейт установлен на значении в 30 Гц (Rainbow Six: Siege) или 60 Гц (Overwatch), но есть и исключения (12 Гц в закрытых матчах Call of Duty: Modern Warfare).

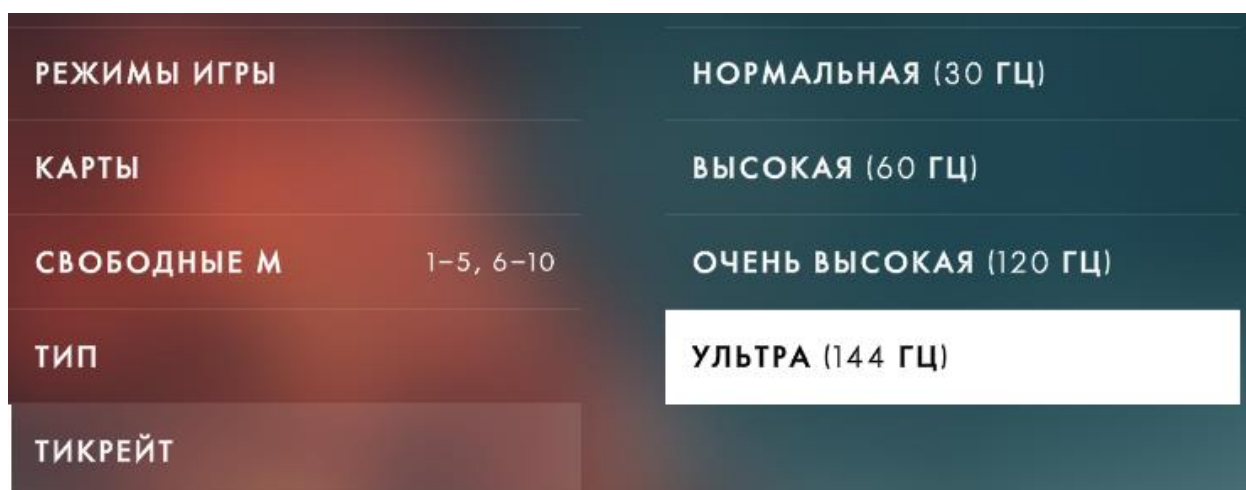


Рисунок 2 – Фильтр тикрейта при поиске сервера в Battlefield 4

1.1.1 Аналоги

Одним из коммерческих аналогов данного курсового проекта является бесплатная игра для iOS, Android, Microsoft Windows, Microsoft Xbox One, Sony PlayStation 4 и Nintendo Switch от Behaviour Interactive и Bethesda Game Studios.

В этой игре игрокам предлагается построить и управлять своим убежищем в качестве зрителя. Как и этот курсовой проект, Fallout Shelter является менеджмент-симулятором, рассчитанным на короткие (5-10 минут) игровые сессии несколько раз в день. Игровой процесс в обоих проектах заключается в балансировке некоторых величин для успешного выполнения разнообразных заданий.

В отличие от Fallout Shelter, данный курсовой проект требует постоянного интернет-соединения и не имеет монетизации.



Рисунок 3 – Fallout Shelter

1.2 Сравнение технологий

1.2.1 Технологии для разработки игры

В качестве клиента была выбрана игра с графическим интерфейсом. Так как время на написание курсовой работы ограничено, то использование фреймворков таких как MonoGame или LibGDX, или графической библиотеки (SFML, SDL) и тем более использование графических API вроде Vulkan, OpenGL или DirectX напрямую было неrationально из-за необходимости организации менеджмента сцен, ресурсов, памяти, систем ввода и вывода, интерфейса, инструментов создания контента и другого функционала, я решил использовать один из игровых движков общего назначения, которые уже включают в себя всё необходимое.

Мой выбор стоял между Unity (C#), Unreal Engine 4 (C++) и Godot (GDScript, VisualScript, C# и C++).

Сравнивать эти средства разработки можно по нескольким критериям, которые будут важны при создании проекта в соответствии с техническим заданием:

- **Лицензия:** Godot могут быть использованы в коммерческих целях без отчислений компании-производителю средств разработки. Помимо этого, Godot полностью бесплатен и имеет открытый исходный код. Unity бесплатна для студий с небольшим бюджетом (до \$100000). UE4 полностью бесплатна для игр, которые заработали меньше \$1000000, после этого – 5% с продаж.

- **Организация ввода.**

- По типу распознавания ввода, системы можно разделить на два типа – события (при нажатии каждой кнопки срабатывает соответствующее событие, на которое могут подписываться пользовательские компоненты) и опрос (каждый кадр, компоненты опрашивают систему ввода на предмет состояния отдельных кнопок). Пользовательский ввод в Unity реализован методом опроса, UE4 – методом событий, Godot реализует оба метода. Unity также поддерживает метод событий с помощью новой системы ввода в

Unity 2020.1, но она пока в стадии бета-теста и не рекомендуется к использованию в коммерческих продуктах.

- Unity позволяет опрашивать систему ввода, по строковому значению, соответствующему определённой кнопке, что упрощает реализацию переназначения кнопок, по сравнению с Godot и UE4 из-за специфики обработки нажатия кнопок в этих программных средствах.

- Godot и UE4 лучше устроена поддержка ввода с геймпадов (эти программные средства могут распознавать индекс устройства, с которого был произведён ввод, а также его тип (например, Dualshock 4, Xbox Wireless Controller или Nintendo Switch Pro Controller), в то время как Unity обращается к ним как к HID-устройствам, что существенно ограничивает удобство работы с геймпадами). Поддержка геймпадов не так существенна при разработке данной версии курсового проекта, но может усложнить перенос игры на другие платформы в будущем.

- **Физический движок.** В UE4 и Unity используется физический движок PhysX (PhysX 3.4 и PhysX 4.1 соответственно). Godot использует собственную реализацию 2D физики. В данном проекте физический движок, используемый средствами разработки не так важен для основного геймплейного цикла, но может использоваться для создания эффектов.

- **Архитектура взаимодействия игровых объектов, обеспечивающая низкое сцепление модулей.** UE4 и Unity используют компонентную архитектуру игровых объектов, что выгодно. Godot использует вложенные сцены.

- **Средства создания шейдеров.** Unity и UE4 поддерживают как визуальные средства создания шейдеров, что значительно ускоряет процесс их создания, так и языки шейдеров подобные HLSL, тогда как Godot использует язык подобный GLSL ES 3.0 и не предоставляет средств визуального создания шейдеров. В данном проекте, дизайн интерфейса будет важнее визуальных эффектов, так что данный пункт не так сильно влияет на выбор технологии,

- **Скриптовый язык.** Unity поддерживает C#, UE4 – C++ (а также систему Blueprint для визуального программирования), Godot – C# или

GDScript на выбор (а также VisualScript для визуального программирования и C++ для низкоуровневого доступа к движку). Сравнивать эти языки можно по нескольким характеристикам:

- **Среда программирования.** Существует множество интегрированных систем программирования для C# (Visual Studio, JetBrains Rider) и C++ (Visual Studio, Code::Blocks). Также, существуют текстовые редакторы, предоставляющие инструменты и расширения для более удобной работы с конкретным языком программирования и игровым движком (например, Visual Studio Code, Notepad++, Atom). В таких текстовых редакторах обычно отсутствуют средства отладки и компилятор, но они требуют меньше вычислительных ресурсов компьютера, но в данном проекте это не критично, так как все игровые движки предоставляют собственные средства отладки и компилятор.

- **Вспомогательные средства программирования.** Godot и UE4 предоставляют средства визуального программирования, что может быть удобно для написания небольших скриптов. Unity, на данный момент, не предоставляет встроенных средств визуального программирования. Существует превью версия в Unity 2020.1, но на данный момент она слишком нестабильна для использования в данном проекте. Также существуют сторонние решения (например, Bolt).

- **Типизация.** C# и C++ статически типизированные языки, в то время как GDScript – динамически типизирован.

- **Синтаксис.** C# и C++ – Си-подобные языки, в то время как синтаксис GDScript схож с синтаксисом Python и Lua (использование отступов для определения уровня вложения, схожесть многих ключевых слов и синтаксиса в целом).

Исходя из вышесказанного, а также полагаясь на собственный опыт разработки игр на этих движках, я выбрал средством программирования данного курсового проекта язык C#, игровой движок Unity, текстовый редактор – Visual Studio Code.

1.2.2 Технологии для разработки сервера

Так как для написания клиента был выбран язык C#, то будет рациональнее использовать один язык программирования для всего проекта, следовательно я буду рассматривать библиотеки для C#. При выборе Unity в качестве технологии разработки клиента, у нас есть несколько вариантов организации соединения с сервером.

Готовые решения:

- **UNet/Unity Networking Layer.** Встроенные в Unity системы организации мультиплеера. UNet – устаревшая система, поддержка которой постепенно прекращается. Система, которая приходит ей на замену – пока что на стадии бета-теста, и пока не готова для применения в приложениях. Также, новая система пока страдает от недостатка качественной документации.

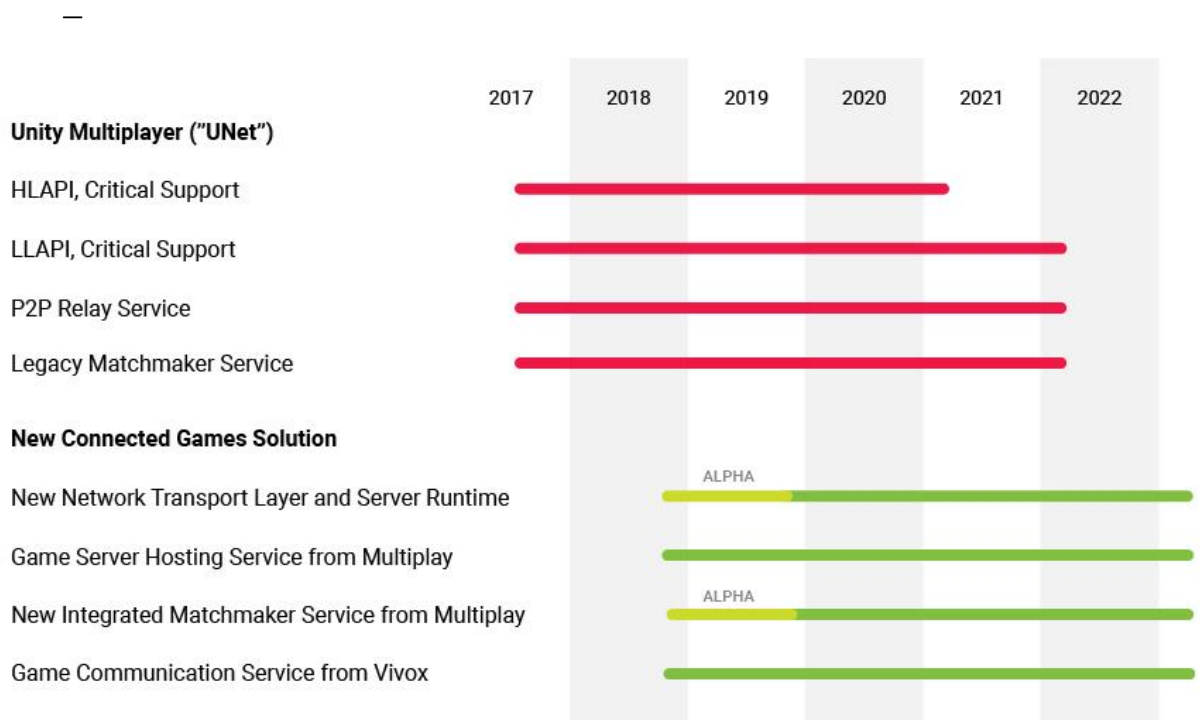


Рисунок 4 – График перехода с UNet на новую мультиплеерную систему

- **Photon.** У Unity также есть альтернативные способы организации мультиплеера. Один из них – популярная сторонняя библиотека Photon. Photon поддерживает такой функционал как . В данной курсовой работе это решение

не подойдёт из-за его направленности на сессионные мультиплеерные игры, геймплей которых происходит синхронно в реальном времени, так как соединение игроков происходит с помощью «комнат» ограниченного размера. Также, её минусом является платная лицензия, цена которой зависит от количества одновременных пользователей, что теоретически ограничит развитие игры в будущем.

– **Socket.io + Unity WWW.** Ещё одним популярным решением для организации мультиплеера в Unity является сервер, написанный на Node.js, с использованием библиотеки Socket.io и обращение к нему с помощью встроенного в Unity класса WWW. В данном курсовом проекте я решил не использовать это решение из-за сложности стека технологии. Использование нескольких языков программирования (JavaScript, PHP и C#) излишне усложнит проект и взаимодействие между клиентом и сервером.

– **Steamworks Multiplayer.** Поддерживает создание выделенных серверов и установление peer-to-peer соединения, а также подбор игроков. К тому же, Steamworks API поддерживает голосовой чат, список друзей, достижения и многое другое. В этом курсовом проекте его применение невозможно, так как Steamworks API использует функционал Steam, а следовательно требует публикации игры в Steam.

Так как мы пишем на C#, то можем воспользоваться системной библиотекой Socket для подключения по различным протоколам. Socket реализует два протокола:

UDP: Преимущества UDP заключаются в простоте как самого протокола, так и пакета, следовательно, он гораздо быстрее, чем TCP. Следовательно, он больше подходит для игр, в которых геймплей происходит в реальном времени. Главным недостатком является ненадёжность доставки сообщений, что может быть критично в случае недоставки сообщения о выполнении задания, получения вещей и подобных.

Биты	0 - 15	16 - 31
0-31	Порт отправителя (Source port)	Порт получателя (Destination port)
32-63	Длина датаграммы (Length)	Контрольная сумма (Checksum)
64-...	Данные (Data)	

Рисунок 5 – Структура UDP пакета

TCP: В противоположность UDP, TCP гарантирует доставку сообщений, что предпочтительнее в данном случае, но из-за сложности и размера пакета, он не подходит для обмена информацией между клиентом и сервером в реальном времени.

Бит	0 — 3	4 — 9	10 — 15	16 — 31
0	Порт источника, Source Port			Порт назначения, Destination Port
32	Порядковый номер, Sequence Number (SN)			
64	Номер подтверждения, Acknowledgment Number (ACK SN)			
96	Длина заголовка, (Data offset)	Зарезервировано	Флаги	Размер Окна, Window size
128	Контрольная сумма, Checksum			Указатель важности, Urgent Point
160	Опции (необязательное, но используется практически всегда)			
160/192+	Данные			

Рисунок 6 – Структура заголовка TCP пакета

База данных:

MySQL: Несмотря на то что MySQL подходит по параметрам безопасности, скорости и документации, его направленность на использование как БД для сайтов не вполне подходит для планируемой реализации сервера.

PostgreSQL: Так как сервер задумывается как один портативный исполняемый файл, то сложность и низкая скорость PostgreSQL не подходят под планируемые спецификации сервера.

SQLite: Портативность SQLite позволяет создать сервер, который автоматически создаёт все нужные файлы при первом запуске исполняемого

файла. SQLite имеет некоторые ограничения, из-за специфики реализации. Например, SQLite реализует не все SQL-запросы (RIGHT OUTER JOIN, FULL OUTER JOIN); ALTER TABLE поддерживается не целиком (только RENAME TABLE, ADD COLUMN, RENAME COLUMN) и некоторые другие. Также, он не поддерживает запросы GRANT и REVOKE, потому что они не имеют смысла во встраиваемой базе данных. Для данного курсового проекта эти ограничения не так важны, потому что клиенты не будут напрямую взаимодействовать с БД, а посылать сообщения серверу, который будет их обрабатывать и по надобности обращаться к БД. Отсутствие разных пользователей с разными правами также решается на сервере, так как все запросы проходят через него. Ограничения по количеству одновременных запросов в данной реализации проекта не играют большой роли, так как не ожидается много пользователей.

1.3 Требования

От сервера требуется обработка следующих запросов:

- Изменение публичного имени игрока. У каждого игрока будет публичное имя, отличное от логина и видимое другими игроками, которое он сможет менять по желанию;
- Отправка корабля игрока на задание;
- Покупка модулей корабля;
- Продажа вещей, полученных на заданиях;
- Расчёт результатов задания;
- Получение списка заданий;

Также, сервер должен реализовывать следующие команды администратора:

- Создание бэкапа БД;
- Загрузка файлов с новыми игровыми сущностями (предметами, модулями корабля, миссиями и т.д.);
- Обработка SQL команд;

2 СПЕЦИАЛЬНЫЙ РАЗДЕЛ

2.1 Диаграмма прецедентов использования

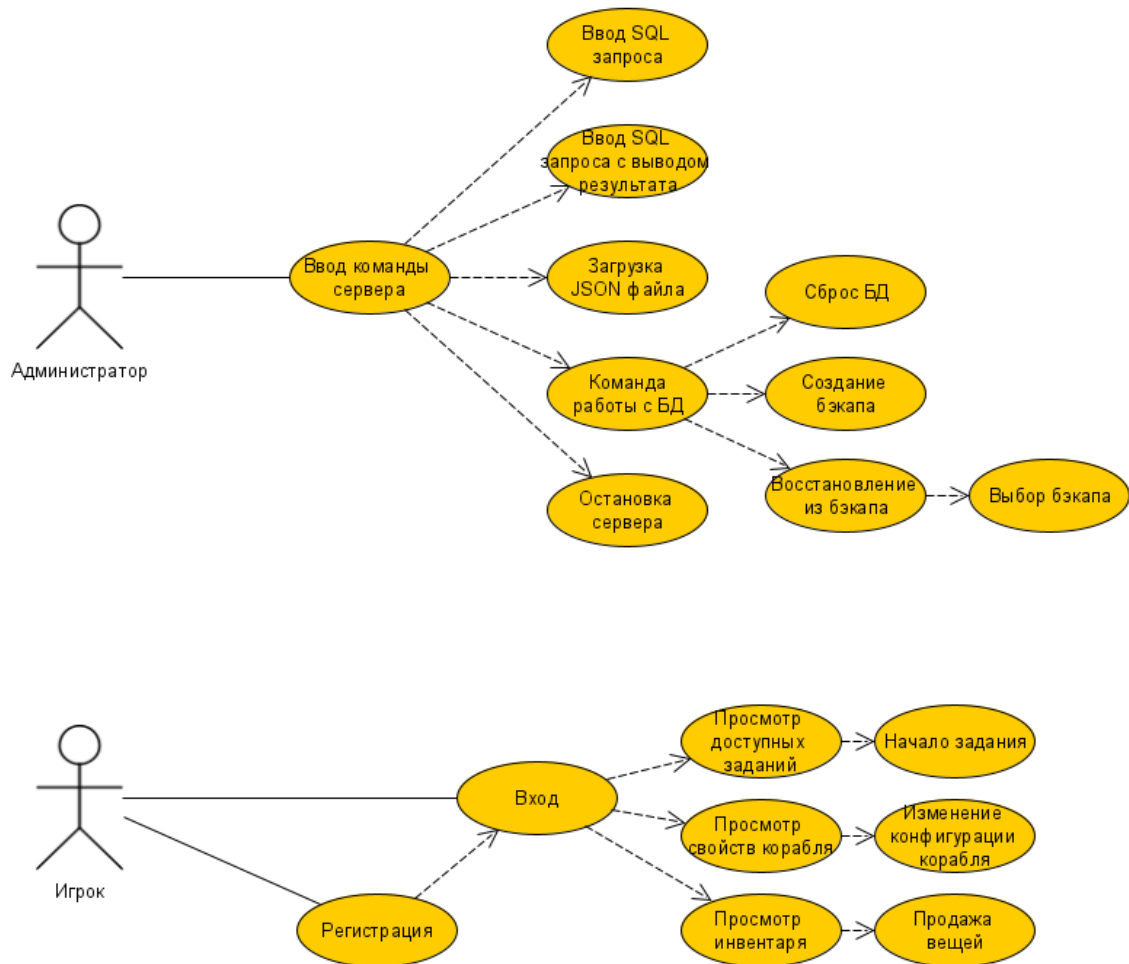


Рисунок 7 – UML диаграмма

2.2 Инфологическая модель

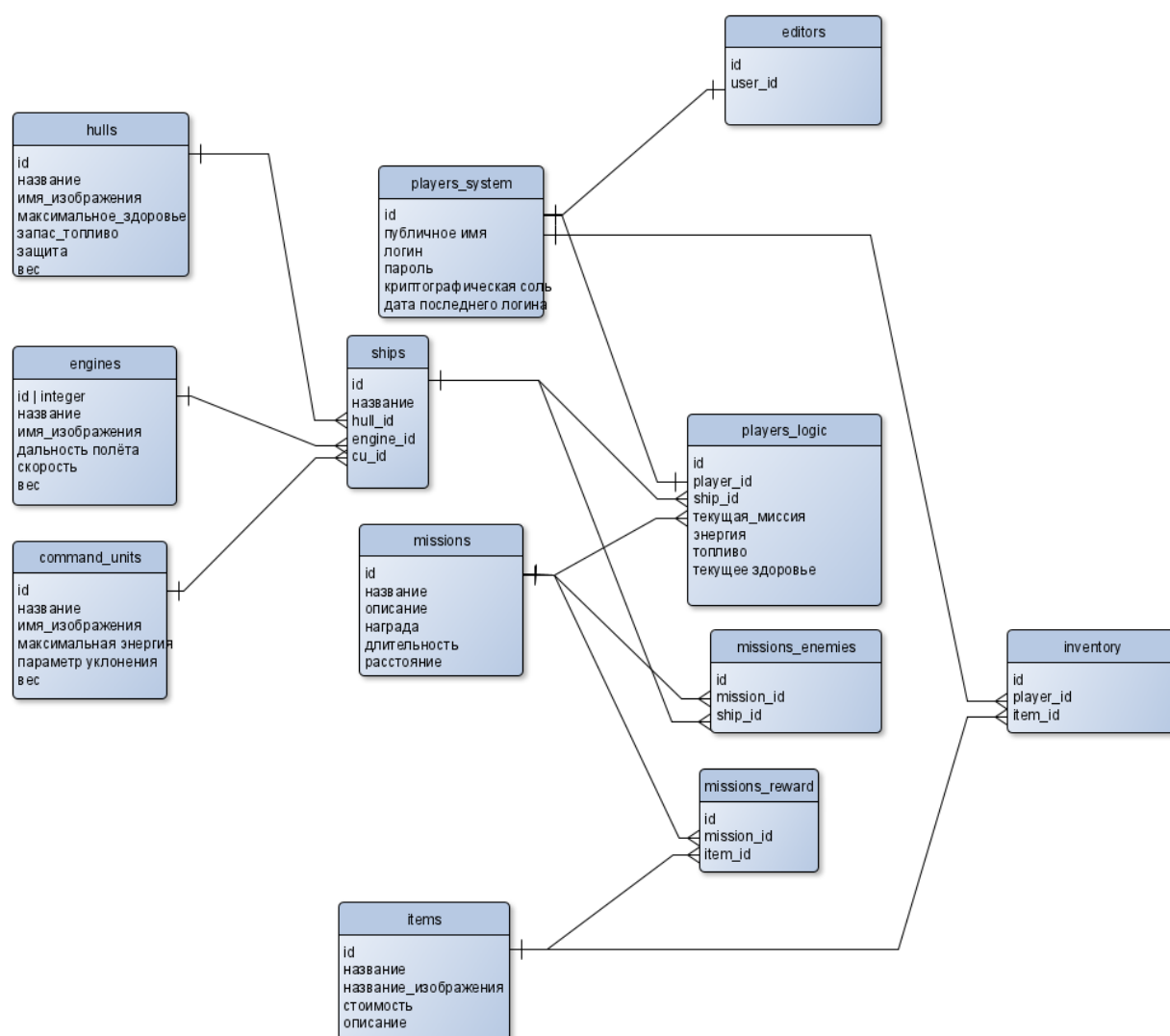


Рисунок 8 – Инфологическая модель БД

2.3 Даталогическая модель

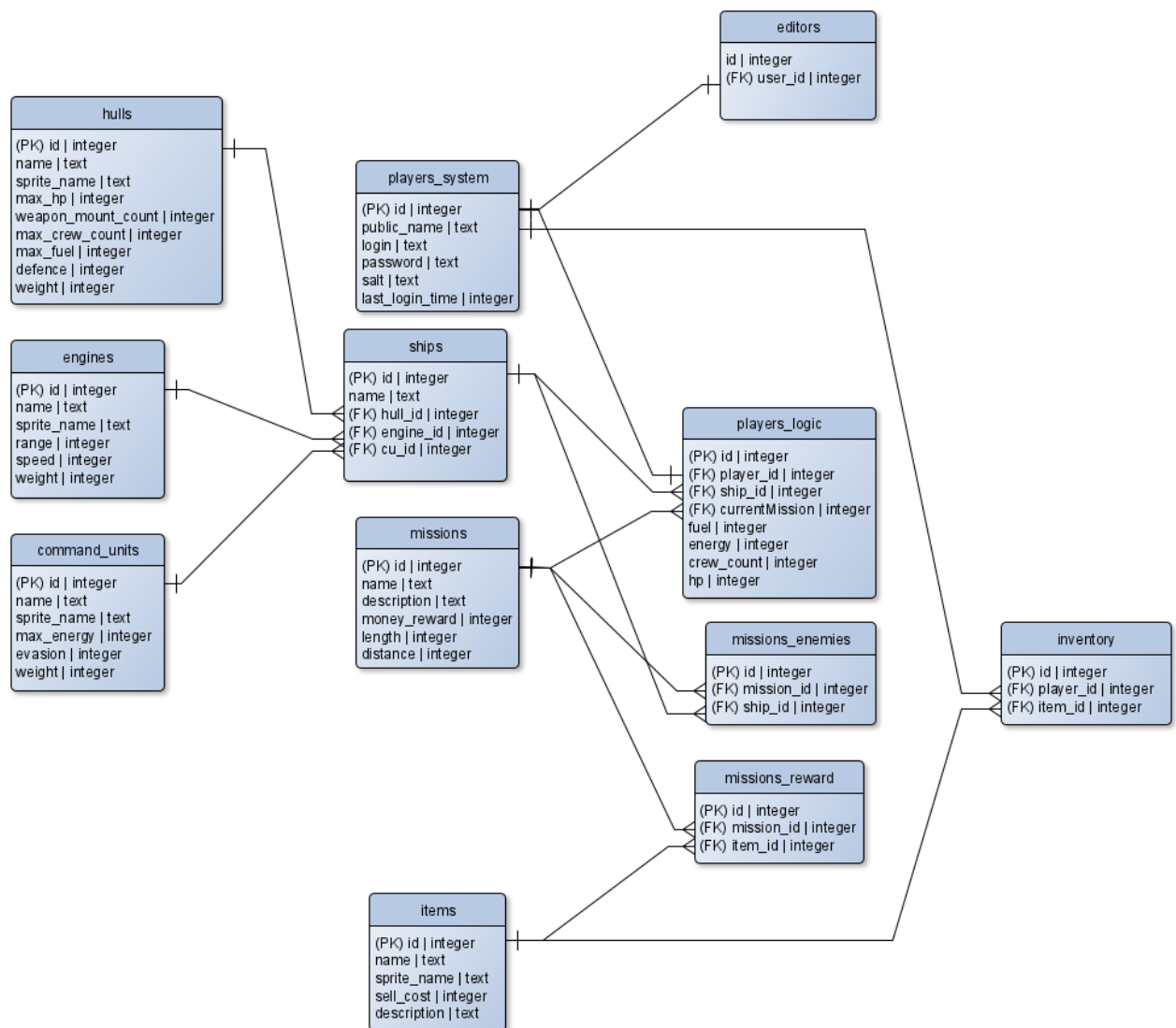


Рисунок 9 – Даталогическая модель БД

2.4 Проект основных дисплейных фрагментов

В клиенте есть два основных окна – окно авторизации и главное игровое окно. При запуске пользователю демонстрируется окно авторизации и после успешного входа в свой аккаунт или регистрации, ему демонстрируется главное окно.

Сервер не имеет графического интерфейса.

Прототипы выполнены с помощью дизайнера Windows Forms, в то время как сам интерфейс приложения реализован с помощью встроенных инструментов проектирования интерфейсов Unity.

Экран авторизации имеет два режима – регистрация и авторизация. Находясь в режиме авторизации, пользователь, нажав на кнопку «Register» может показать поля, необходимые для регистрации нового аккаунта (публичное имя и подтверждение пароля). В этом режиме, поля «Login» и «Password» используются в ходе регистрации.

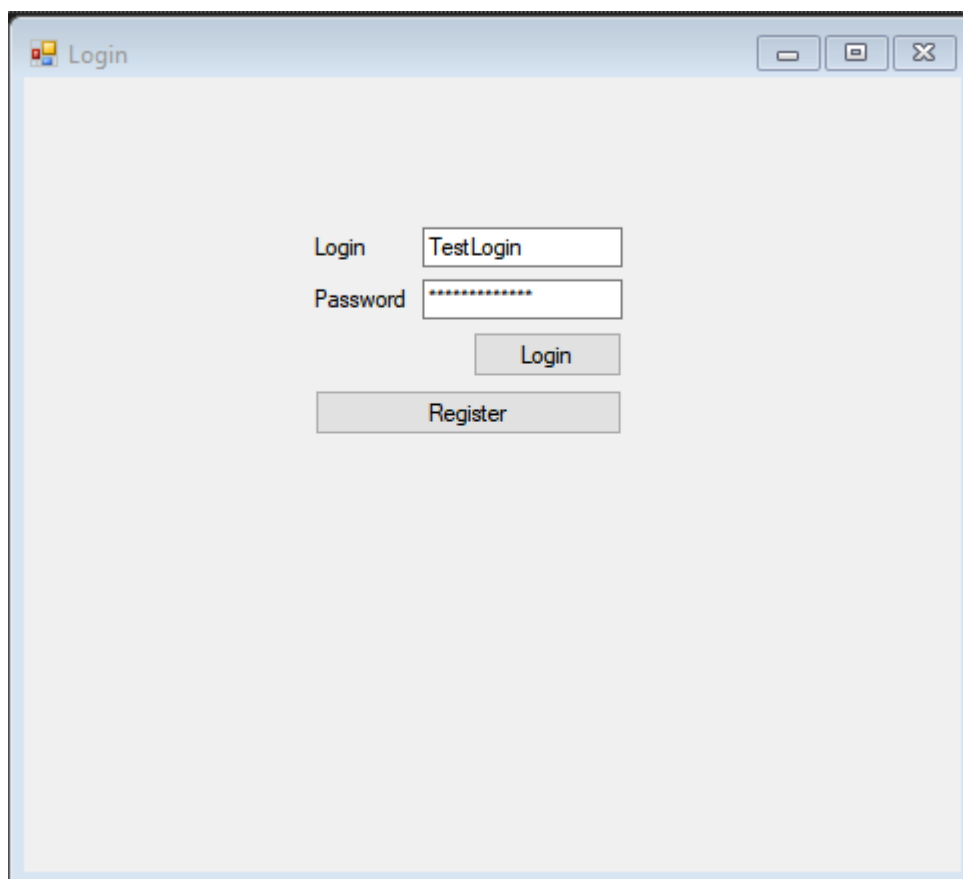


Рисунок 10 – Экран авторизации

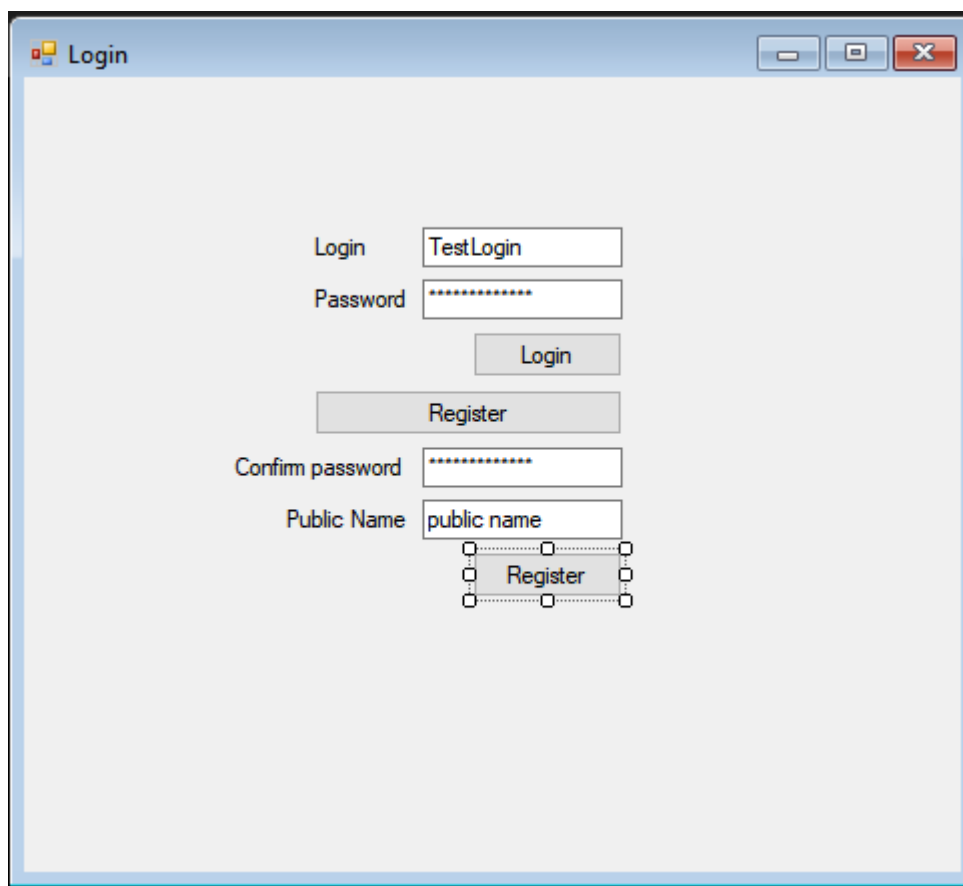


Рисунок 11 – Экран регистрации

Главный экран игры имеет структуру, описанную на рисунке 11. На информационной панели расположены некоторые важные игровые данные (деньги, энергия, топливо и т.д.), с возможностью добавить ещё при потенциальном расширении игры в будущем. Главная панель – контейнер для вкладок, содержащих элементы интерфейса для отображения какой-либо определённой информации (например, задания, инвентарь или детали корабля). Вкладка выбирается с помощью кнопок в левой части экрана. Это решение было также обусловлено возможностью расширения функционала в будущем.

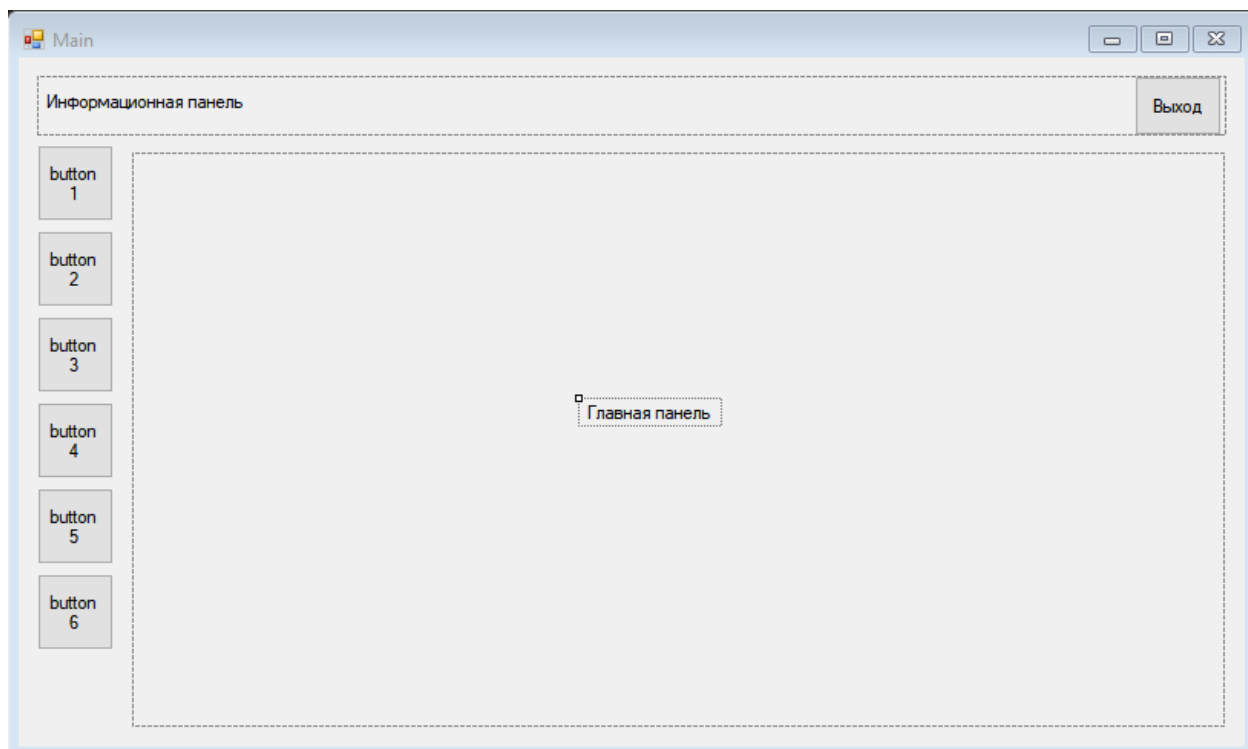


Рисунок 12 – Главная игровая панель

2.5 Исключительные ситуации

Сервер реализует исключительные ситуации, описанные в таблице 1. Клиент реализует исключительные ситуации, описанные в таблице 2.

Таблица 1 – Исключительные ситуации сервера

Ситуация	Способ обработки
БД заблокирована другим процессом/другим сообщением	Вывод сообщений «database is locked» администратору
Ввод некорректной команды (или аргументов) администратором	Вывод сообщения администратору
Неправильно введенный SQL запрос в серверной команде «sql» или «sqlq»	Вывод исключения на экран
Некорректный аргумент команды «load»	Вывод на экран сообщения о невозможности найти введенный системным администратором файл
Невозможность инициализировать БД	Вывод сообщения администратору, остановка сервера
Некорректный запрос от клиента	Сообщение администратору
Некорректный логин или пароль	Отправка сообщения клиенту об отказе входа в аккаунт.

Таблица 2 – Исключительные ситуации клиента

Ситуация	Способ обработки
Отключение сервера во время работы клиента	Вывод сообщения игроку
Невозможность подключиться к серверу для обработки запроса	Вывод сообщения игроку

2.6 Разработанные запросы

Программы реализуют запросы, перечисленные в таблице 3.

Таблица 3 – Спипок запросов

Запрос	Описание
<code>SELECT * FROM engines</code>	Получение списка доступных двигателей для корабля.
<code>SELECT * FROM hulls</code>	Получение списка доступных корпусов для корабля.
<code>SELECT * FROM command_units</code>	Получение списка доступных командных центров для корабля.
<code>SELECT * FROM missions</code>	Получение списка доступных заданий
<code>select items.* from items inner join inventory on items.id = inventory.item_id where inventory.player_id = @playerID</code>	Получение инвентаря игрока
<code>SELECT id, login, password, salt FROM players_system WHERE login=@username</code>	Получение информации для авторизации
<code>UPDATE players_system SET last_login_time = \"{DateTime.Now.ToString()}\" WHERE id = {data.GetInt32(0)}</code>	Установление даты последнего логина
<code>INSERT INTO players_system (login, password, salt, public_name, last_login_time) VALUES (@username, @password, @salt, @name, @lastLogin); SELECT last_insert_rowid()</code>	Регистрация нового аккаунта

Ниже приведён код формирования запроса на добавление записи из JSON файла в БД на языке C#.

```
private static void InsertInDB<T>(T obj) {  
    string sql = $"INSERT INTO {obj.GetType().Name}(";  
    var fields = obj.GetType().GetFields();  
    for (int i = 0; i < fields.Length; i++) {  
        sql += fields[i].Name;  
        if (i != fields.Length - 1) {  
            sql += ",";  
        }  
    }  
  
    sql += ") VALUES (";
```



```

    for (int i = 0; i < fields.Length; i++) {
        if (fields[i].GetValue(obj).GetType() == typeof(string)) {
            sql += $"{fields[i].GetValue(obj).ToString()}\"";
        } else {
            sql += fields[i].GetValue(obj).ToString();
        }
        if (i != fields.Length - 1) {
            sql += ",";
        }
    }

    sql += ")";

    DBController.ExecuteNonQuery(sql);
}

```

2.7 Обеспечение целостности БД

Обеспечение целостности достигается за счёт обработки игровой логики на сервере. Клиент посылает только сообщения о том, что он собирается сделать (продать предмет, заменить часть корабля, отправиться на задание и т.д.), а уже сервер самостоятельно производит данные действия, проверяя возможность их совершения.

Также, на данные вводимые системным администратором (как вручную, с помощью команд `sql` и `sqlq`, так и с помощью JSON-файлов) введены ограничения – ни одно из полей в базе данных не может быть пустым.

2.8 Разграничение прав доступа. Защита данных

Разграничение прав доступа реализуется только в клиенте. Каждый пользователь имеет доступ только к информации, относящейся к своему аккаунту. Работа с сервером подразумевается только с компьютера, на котором запущен сервер одним системным администратором, а не удалённо, поэтому отсутствие авторизации и разграничения прав доступа на серверной стороне не является серьёзной проблемой в данной реализации. Администратор сервера имеет доступ ко всем данным пользователей, как логическим (игровая информация вроде текущего количества здоровья и), так и системным (`id` игроков, их логин и публичные имена, а также дата последнего логина).

Защита информации пользователей на сервере осуществляется с помощью хэширования с помощью криптографической соли. Хэширование происходит на стороне сервера. Так как в игре отсутствует система монетизации и в БД не содержится информация о реальном имени, адресе и банковской информации, то единственной защищённой информацией в базе данных являются пароли пользователей.

2.9 Диаграмма классов сервера и клиента

На рисунках 13 и 14 изображены структуры (классы, методы, переменные и их связи) сервера и клиента соответственно.

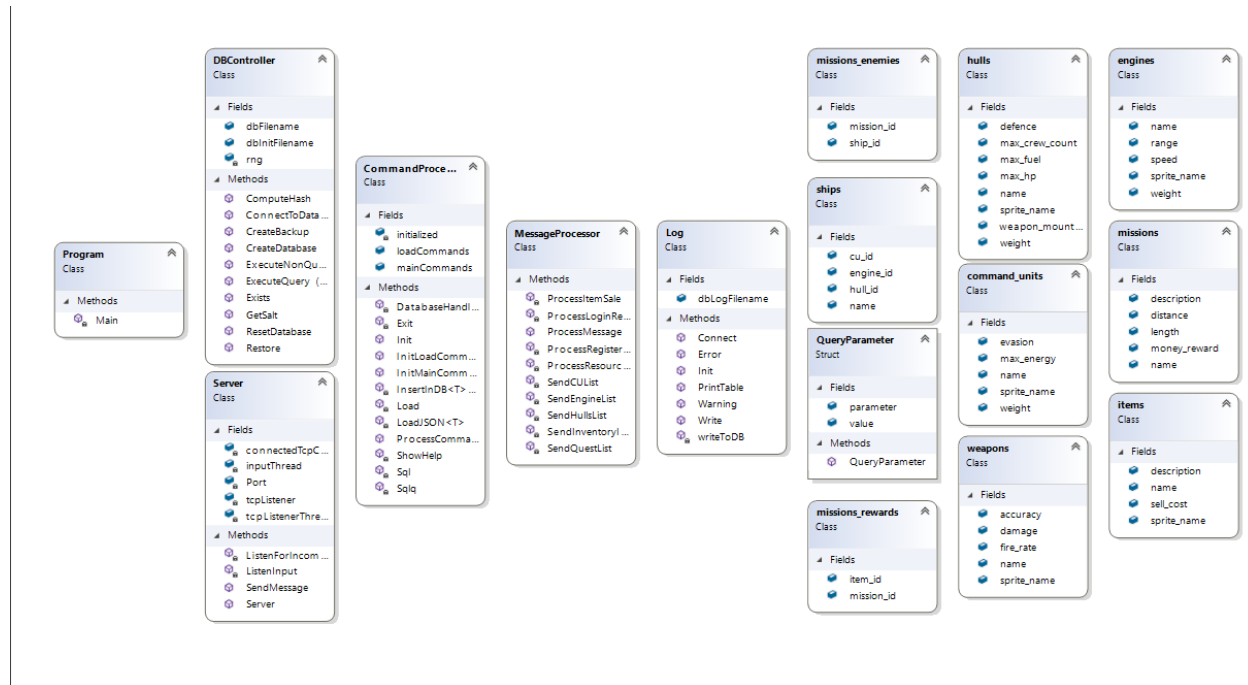


Рисунок 13 – Диаграмма классов сервера.

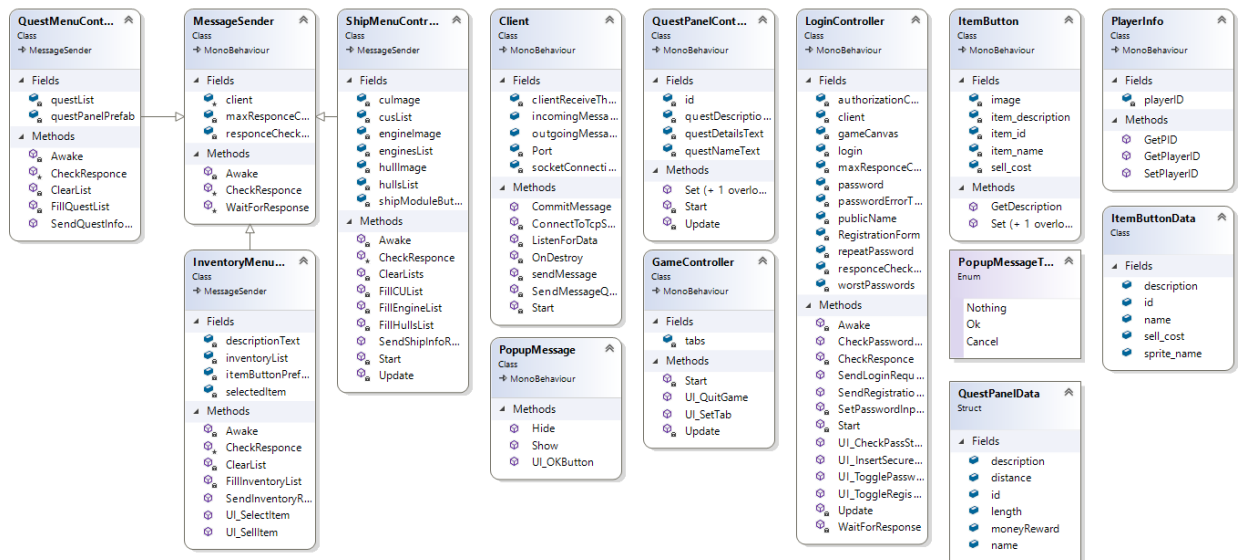


Рисунок 14 – Диаграмма классов клиента.

2.10 Справочная система.

Справочная система сервера доступна по команде «help». Системный администратор может ввести эту команду для получения информации обо всех доступных ему командах. Данная документация заполняется при добавлении команды в словарь командного процессора.

Пример регистрации команд в системе:

```
mainCommands.Add("sql", new Tuple<string, Action<string[]>>("executes sql command", Sql));
mainCommands.Add("sqlq", new Tuple<string, Action<string[]>>("executes sql commands and prints result", Sqlq));
mainCommands.Add("exit", new Tuple<string, Action<string[]>>("shuts down the server", Exit));
mainCommands.Add("load", new Tuple<string, Action<string[]>>("loads JSON file of a specified type", Load));
mainCommands.Add("help", new Tuple<string, Action<string[]>>("shows this message", ShowHelp));
mainCommands.Add("db", new Tuple<string, Action<string[]>>("handles database actions", DatabaseHandling));
```

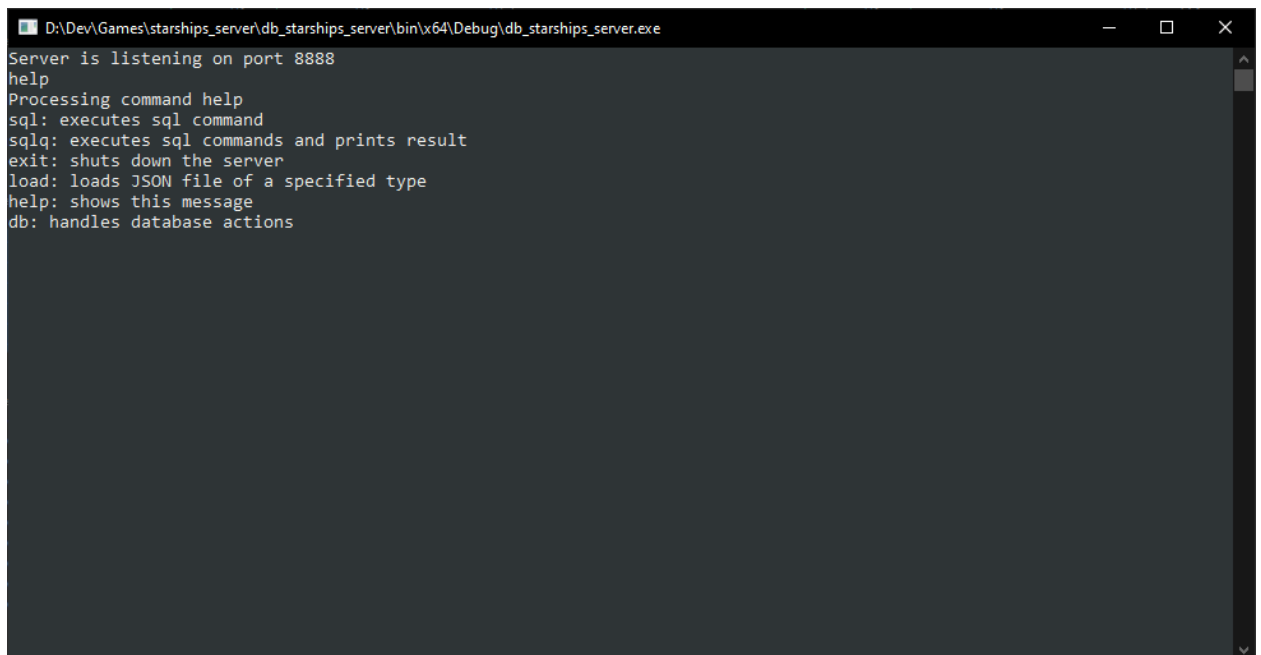
A screenshot of a Windows command prompt window. The title bar shows the file path: D:\Dev\Games\starships_server\db_starships_server\bin\x64\Debug\db_starships_server.exe. The window content shows the following text:
Server is listening on port 8888
help
Processing command help
sql: executes sql command
sqlq: executes sql commands and prints result
exit: shuts down the server
load: loads JSON file of a specified type
help: shows this message
db: handles database actions

Рисунок 15 – Вывод справки.

2.11 Тестирование приложения

2.11.1 Сервер

Первичная инициализация БД. При первом запуске сервера, автоматически создаётся пустая БД, используя SQL файл, указанный в переменной `dbInitFilename`.

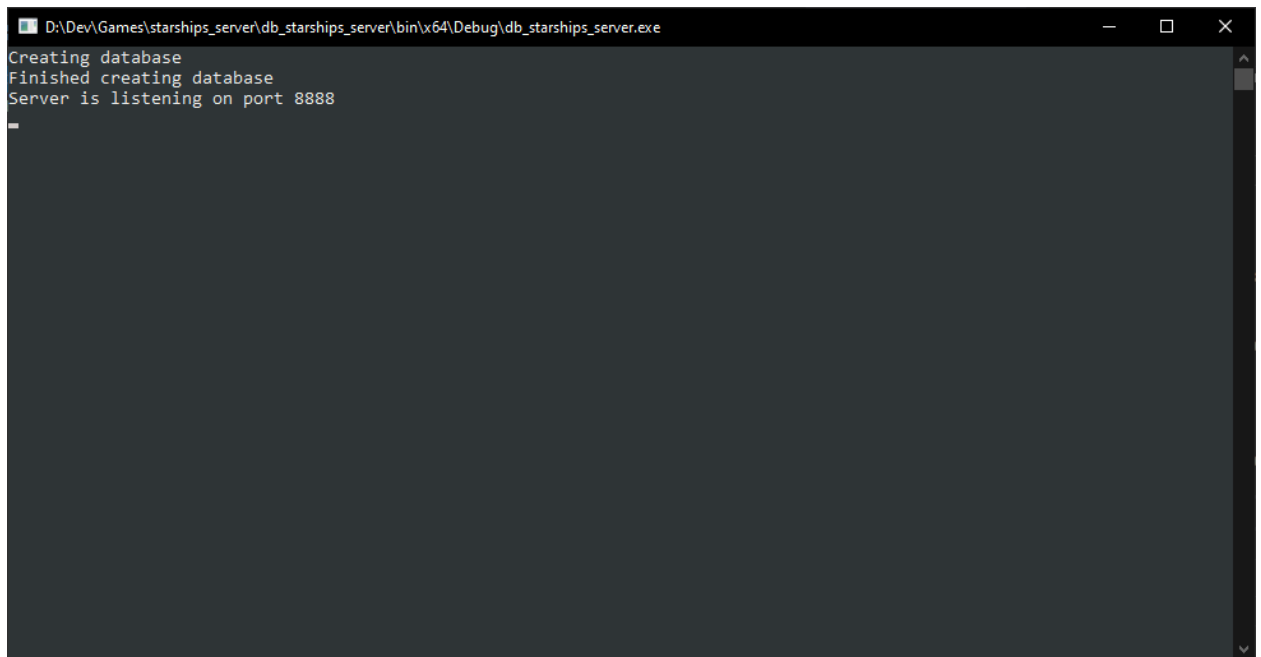


Рисунок 16 – Создание БД при запуске сервера

Создание бэкапа БД. Сервер поддерживает создание бэкапов и восстановление из них. Для создания бэкапа используется команда `db` с аргументом `backup`. Также, дополнительными аргументами можно указать комментарий к бэкапу.

Для восстановления БД из бэкапа используется команда `db` с аргументом `restore`. При вводе этой команды, администратору печатается список всех доступных ему бэкапов, из которого он может выбрать нужный.

```
D:\Dev\Games\starships_server\db_starships_server\bin\x64\Debug\db_starships_server.exe
Server is listening on port 8888
db backup
Processing command db backup
Backup starships.db-01-08-2020-07-08-22.backup is created
db backup Test For Screenshot
Processing command db backup Test For Screenshot
Backup starships.db-01-08-2020-07-08-52TestForScreenshot.backup is created
db restore
Processing command db restore
Restoring database
Backup starships.db-01-09-2020-07-09-12-restore.backup is created
Select backup
0. .\starships.db-01-04-2020-07-04-25.backup
1. .\starships.db-01-08-2020-07-08-22.backup
2. .\starships.db-01-08-2020-07-08-52TestForScreenshot.backup
3. .\starships.db-01-09-2020-07-09-12-restore.backup
4. .\starships.db-27-34-2020-11-34-31-restore.backup
5. .\starships.db-27-35-2020-02-35-26-restore.backup
6. .\starships.db-27-35-2020-11-35-03-restore.backup
7. .\starships.db-27-59-2020-02-59-41-restore.backup
2
Restored database from .\starships.db-01-08-2020-07-08-52TestForScreenshot.backup
```

Рисунок 17 – Работа с бэкапами

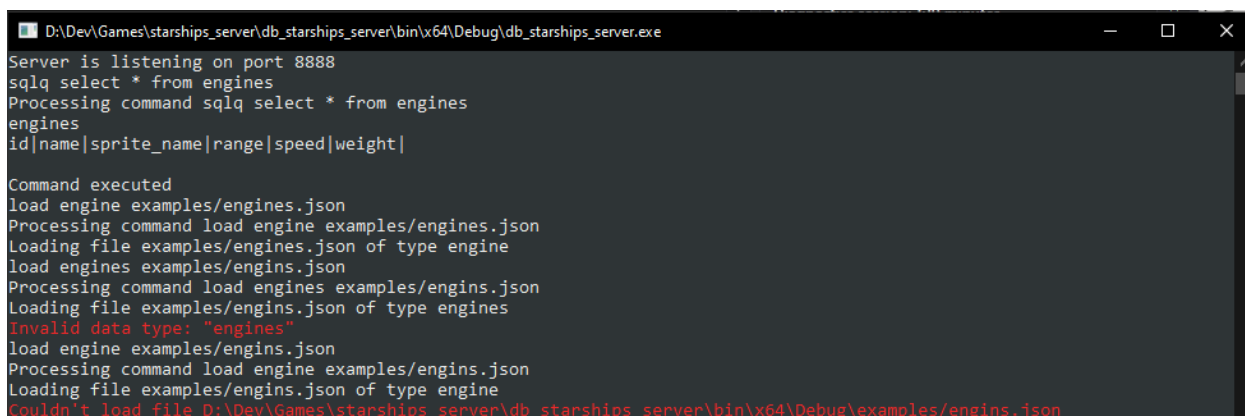
Обработка SQL команд. Сервер поддерживает две команды для ввода SQL команд – `sql` и `sqlq`. Первая команда предназначена для запросов, которые не возвращают значения (запросы типов DML, DDL, DCL и TPL). Вторая предназначена для выполнения DQL запросов и вывода на экран результатов в виде таблицы. На рисунке 18 показана работа этих команд, а также обработка некорректно введённого SQL запроса.

```
D:\Dev\Games\starships_server\db_starships_server\bin\x64\Debug\db_starships_server.exe
Server is listening on port 8888
sqlq select * from hulls
Processing command sqlq select * from hulls
hulls
id|name|sprite_name|max_hp|weapon_mount_count|max_crew_count|max_fuel|defence|weight|
Command executed
sql insert into hulls (name, sprite_name, hax_hp, weapon_mount_count, max_crew_count, max_fuel, defence, weight) values
("hull name", "sprite.png", 10, 3, 1500, 100, 7, 12)
Processing command sql insert into hulls (name, sprite_name, hax_hp, weapon_mount_count, max_crew_count, max_fuel, defence, weight) values ("hull name", "sprite.png", 10, 3, 1500, 100, 7, 12)
SQL logic error
table hulls has no column named hax_hp
Command executed
sql insert into hulls (name, sprite_name, max_hp, weapon_mount_count, max_crew_count, max_fuel, defence, weight) values
("hull name", "sprite.png", 10, 3, 1500, 100, 7, 12)
Processing command sql insert into hulls (name, sprite_name, max_hp, weapon_mount_count, max_crew_count, max_fuel, defence, weight) values ("hull name", "sprite.png", 10, 3, 1500, 100, 7, 12)
Command executed
sqlq select * from hulls
Processing command sqlq select * from hulls
hulls
id|name|sprite_name|max_hp|weapon_mount_count|max_crew_count|max_fuel|defence|weight|
1|hull name|sprite.png| 10|          3|        1500|    100|    7|    12|
Command executed
```

Рисунок 18 – Команды `sql` и `sqlq`

Загрузка файлов с новыми игровыми сущностями (предметами, модулями корабля, миссиями и т.д.). Вместо использования команд `sql` и `sqlq` для загрузки новой информации на сервер, администратор может загружать JSON файлы с данными об игровых сущностях с помощью команды `load`. Она принимает два аргумента – тип загружаемой сущности и путь до JSON файла. Администратор может создавать эти файлы вручную в текстовом редакторе, но, в теории, может быть создан отдельный инструмент для их создания или экспорт из Excel.

На рисунке 19 показана работа команды `load` для загрузки новых двигателей для кораблей из файла `engines.json`, а также обработка исключительных ситуаций (неправильно указанный тип и неправильно указанное имя файла).



```
D:\Dev\Games\starships_server\db_starships_server\bin\x64\Debug\db_starships_server.exe
Server is listening on port 8888
sqlq select * from engines
Processing command sqlq select * from engines
engines
id|name|sprite_name|range|speed|weight|
Command executed
load engine examples/engines.json
Processing command load engine examples/engines.json
Loading file examples/engines.json of type engine
load engines examples/engines.json
Processing command load engines examples/engines.json
Loading file examples/engines.json of type engines
Invalid data type: "engines"
load engine examples/engines.json
Processing command load engine examples/engines.json
Loading file examples/engines.json of type engine
Couldn't load file D:\Dev\Games\starships_server\db_starships_server\bin\x64\Debug\examples/engins.json
```

Рисунок 19 – Загрузка деталей корабля в БД из JSON файла

2.12 Клиент

Регистрация нового аккаунта. При регистрации нового аккаунта запрос с логином и паролем отправляется на сервер.

В целях тестирования при разработке курсового проекта, а также демонстрации, логин, пароль и публичное имя показываются администратору сервера, что недопустимо в реальной программе. Для этого необходимо добавить исключительные ситуации в систему вывода сообщений.

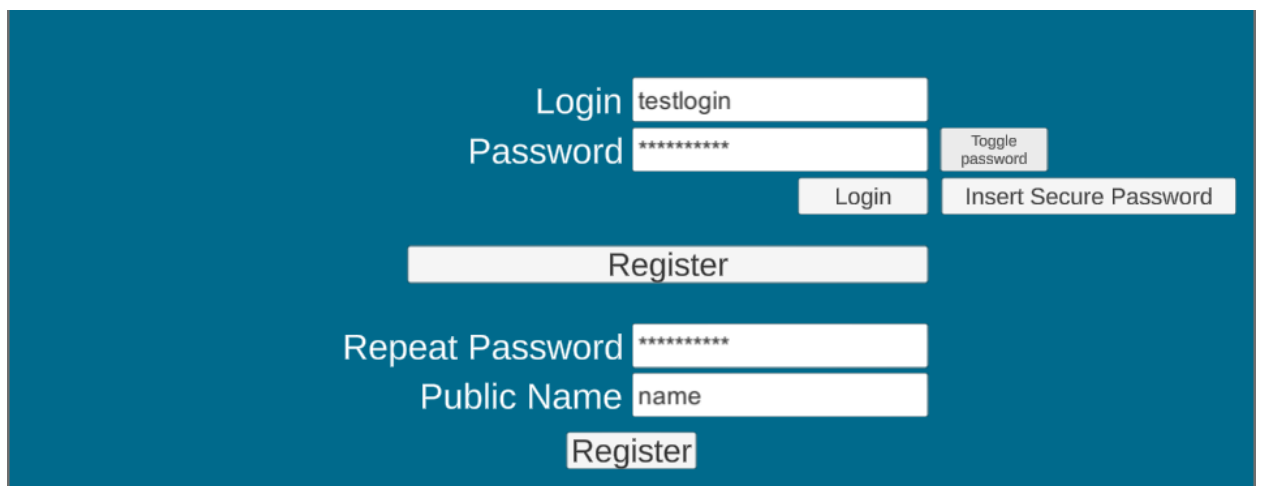
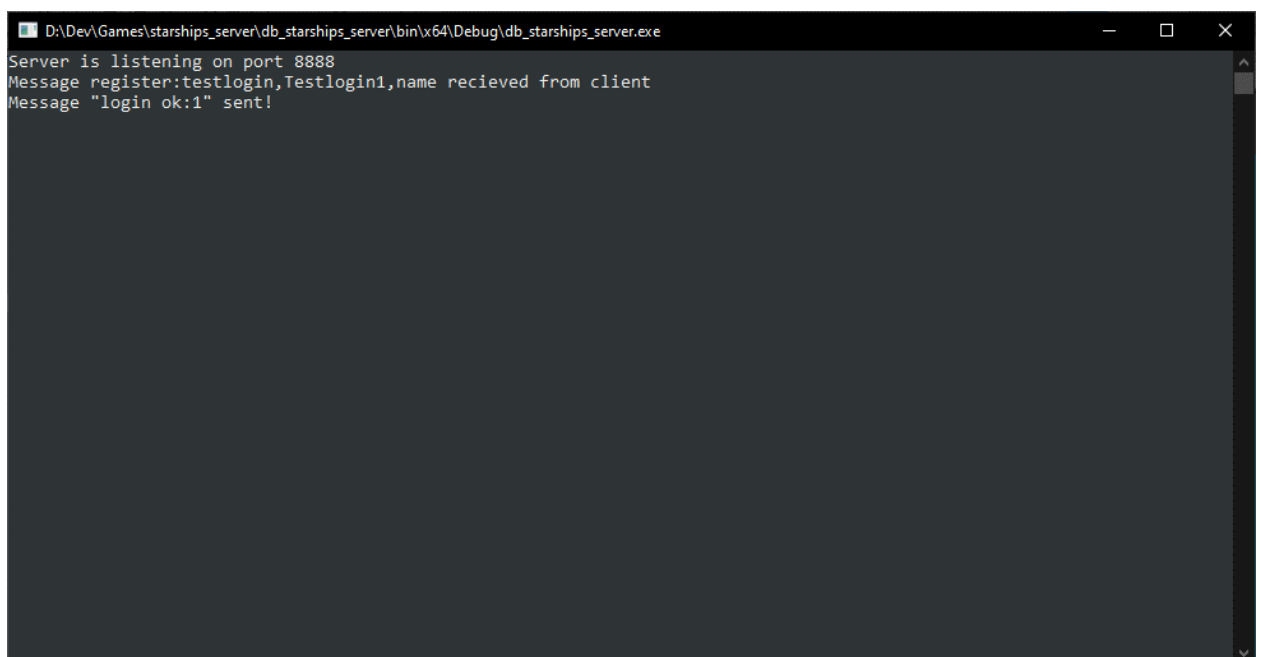


Рисунок 20 – Экран регистрации



```
D:\Dev\Games\starships_server\db_starships_server\bin\x64\Debug\db_starships_server.exe
Server is listening on port 8888
Message register:testlogin,Testlogin1,name recieved from client
Message "login ok:1" sent!
```

Рисунок 21 – Сообщение, получаемое сервером

Вход в аккаунт. Здесь применяется система, схожая с системой регистрации, но с другим префиксом и немного отличающейся информацией, посылаемой в сообщении. В данном случае, существует такая же проблема безопасности, с тем же решением, что и в системе регистрации.

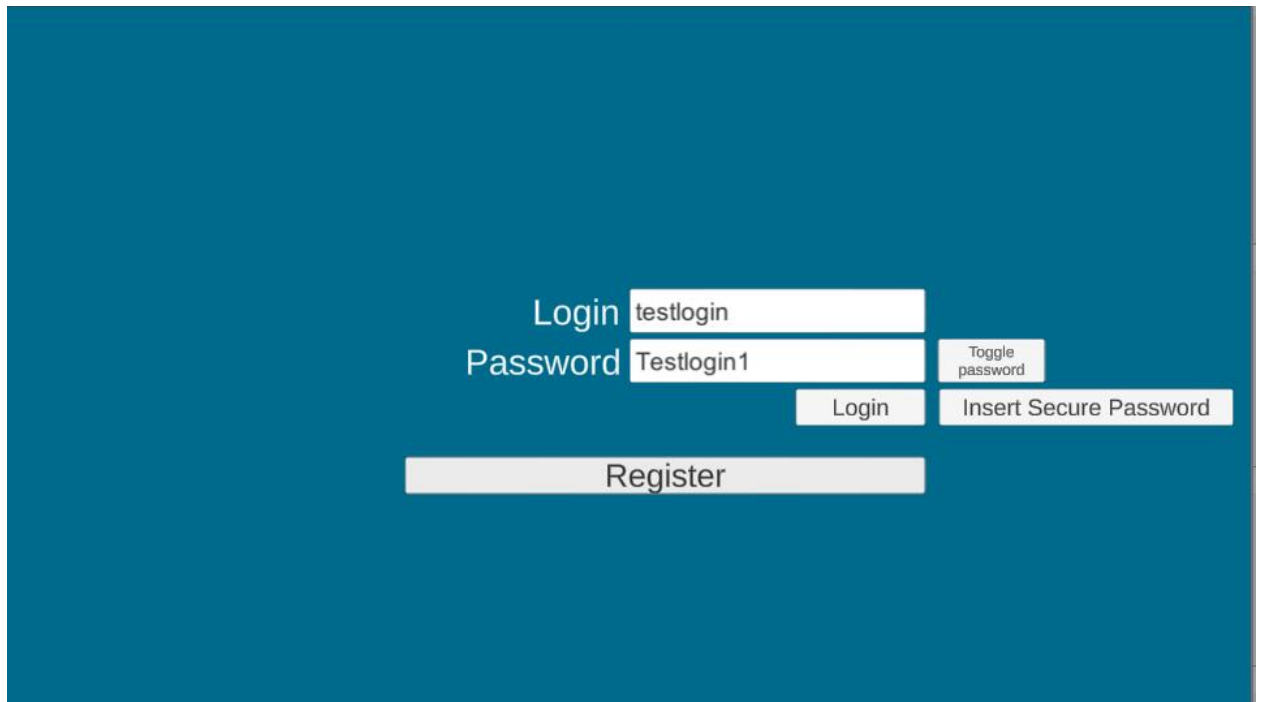


Рисунок 22 – Экран входа

```
Server is listening on port 8888
Message login:testlogin,Testlogin1 recieved from client
Message "login ok:1" sent!
```

Рисунок 23 – Сообщение, получаемое сервером

Получение информации с сервера. Для получения информации с сервера, клиент должен отправить запрос и ждать ответа. В целях обеспечения безопасности, сервер отправляет не SQL запросы, а запросы в специальном формате, а уже сам сервер безопасно обращается к БД.



Рисунок 24 – Данные, полученные от сервера

```
Message req:quests recieved from client  
Message "quests:  
mission 1,Example mission 1,100,60,1000  
mission 2,Example mission 2,200,120,1200  
" sent!
```

Рисунок 25 – Данные, отправленные сервером

ЗАКЛЮЧЕНИЕ

В ходе данного курсового проекта было разработано два приложения – игровой сервер и клиент, работающий с этим сервером. Для разработки сервера была использована интегрированная среда разработки Microsoft Visual Studio 2019 Community, язык программирования C#, а также следующие библиотеки:

- SQLite.Net – для работы с БД;
- Newtonsoft JSON – для работы с JSON файлами;

Также, использовались встроенные модули языка C# Threading и Sockets для организации многопоточной обработки данных и работой с протоколом TCP соответственно.

Для разработки клиента был использован игровой движок Unity, а также встроенный модуль C# Sockets для обеспечения соединения с сервером по протоколу TCP.

Для написания SQL скриптов был использован текстовый редактор Visual Studio Code и расширение MySQL Syntax для упрощения работы с SQL. Также, было использовано расширение vscode-sqlite для просмотра БД.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. C# 4.0: Полное руководство.: Пер. С англ. – М: ООО «И.Д. Вильямс», 2014. ISBN 978-5-8459-1684-6
2. Соловьев, И. В. Проектирование информационных систем. Фундаментальный курс / И.В. Соловьев, А. А. Майоров. - М. : Академический проект, 2009. - 398 с.
3. Соловьев, И. В. Проектирование информационных систем. Фундаментальный курс: Учебное пособие для высшей школы / И. В. Соловьев, А.А. Майоров; Под ред. В.П. Савиных. - М. : Академический проспект, 2009. - 398 с.
4. msdn.microsoft.com. Документация C#.
5. sqlite.org. Документация СУБД SQLite и примеры внедрения в приложение .Net Framework.
6. Рик ван дер Ланс: The SQL Guide to SQLite, 2009 ISBN 978-0557076765
7. Грофф Джеймс Р., Вайнберг Пол Н. SQL. Полное руководство. 2018
8. store.steampowered.com. Список игровых жанров и примеры приложений в данной предметной области.
9. wikipedia.org. Описание предметной области.
10. unity.com. Документация и обучающие статьи игрового движка Unity.

ПРИЛОЖЕНИЕ А

Характеристики программного и аппаратного обеспечения

Клиент:

Операционная система: Windows Vista SP1+

Процессор: требуется поддержка набора инструкций SSE2.

ОЗУ: 256 MB

Видеокарта: Видеокарта с поддержкой DirectX 10 (shader model 4.0).

Запоминающее устройство: 128 MB

Сервер:

Операционная система: Windows Vista SP2+ (64 бита)/ Windows Server 2008 SP2+ (64 бита);

Процессор: 1 ГГц;

Требования к запоминающему устройству сервера зависят от количества пользователей, количества игровых сущностей и частоте создания резервных копий.

ПРИЛОЖЕНИЕ Б

Руководство системного программиста

Работа с БД.

Работа с базой данных на сервере осуществляется с помощью команды db.

Аргументы команды db:

- **reset.** При первом запуске программа создаёт пустую БД, с помощью файла «created.sql». Системный администратор может вручную вернуть БД в данное состояние с помощью команды reset.

- **backup <сообщение>.** Команда для создания резервной копии БД. Сервер создаёт копию текущей версии БД и сохраняет дату её создания. Также, администратор может прикрепить в резервной копии сообщение.

- **restore.** Вызывает меню восстановления БД из резервной копии.

Также, сервер ведёт лог всех команд и запросов. Дата, тип и содержание каждого запроса, команды и сообщения хранится в базе данных log.db.

Использование SQL.

Для выполнения SQL запросов к БД, администратор может воспользоваться командами sql и sqlq. Первая команда предназначена для запросов, которые не возвращают значения (запросы типов DML, DDL, DCL и TPL). Вторая предназначена для выполнения DQL запросов и вывода на экран результатов в виде таблицы.

Добавление новых игровых сущностей

Список таблиц, содержащих игровую логику:

Таблица Б.1 – Таблицы БД

Название	Назначение
hulls	Корпусы кораблей
engines	Двигатели кораблей
command_units	Модули управления кораблей
weapons	Оружие кораблей
items	Предметы
missions	Задания
ships	Корабли
missions_reward	Награды за задания
missions_enemies	Враги, встречающиеся на заданиях

ПРИЛОЖЕНИЕ В

Листинг SQL-скрипта

```
CREATE TABLE hulls(  
    id integer PRIMARY KEY,  
    name TEXT NOT NULL,  
    sprite_name TEXT NOT NULL,  
    max_hp integer NOT NULL,  
    weapon_mount_count integer NOT NULL,  
    max_crew_count integer NOT NULL,  
    max_fuel INTEGER NOT NULL,  
    defence INTEGER NOT NULL,  
    weight INTEGER NOT NULL  
);
```

```
CREATE TABLE engines (  
    id INTEGER PRIMARY KEY,  
    name TEXT NOT NULL,  
    sprite_name TEXT NOT NULL,  
    range INTEGER NOT NULL,  
    speed INTEGER NOT NULL,  
    weight INTEGER NOT NULL  
);
```

```
CREATE TABLE command_units (  
    id INTEGER PRIMARY KEY,  
    name TEXT NOT NULL,  
    sprite_name TEXT NOT NULL,  
    max_energy INTEGER NOT NULL,  
    evasion INTEGER NOT NULL,  
    weight INTEGER NOT NULL  
);
```

```
CREATE TABLE weapons (  
    id INTEGER PRIMARY KEY,  
    name TEXT NOT NULL,  
    sprite_name TEXT NOT NULL,  
    damage INTEGER NOT NULL,  
    accuracy INTEGER NOT NULL,  
    fire_rate INTEGER NOT NULL  
);
```

```
CREATE TABLE items (  
    id INTEGER PRIMARY KEY,  
    name TEXT NOT NULL,  
    sprite_name TEXT NOT NULL,  
    sell_cost INTEGER NOT NULL,  
    description TEXT NOT NULL  
);
```

```
CREATE TABLE players_system (  
    id INTEGER PRIMARY KEY,  
    public_name TEXT NOT NULL,  
    login TEXT NOT NULL,  
    password TEXT NOT NULL,  
    salt TEXT NOT NULL,  
    last_login_time INTEGER NOT NULL  
);
```



```

CREATE TABLE missions (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    description TEXT NOT NULL,
    money_reward INTEGER NOT NULL,
    length INTEGER NOT NULL,
    distance INTEGER NOT NULL
);

CREATE TABLE ships (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,

    hull_id INTEGER,
    engine_id INTEGER,
    cu_id INTEGER,
    FOREIGN KEY(cu_id) REFERENCES command_units(id),
    FOREIGN KEY(engine_id) REFERENCES engines(id),
    FOREIGN KEY(hull_id) REFERENCES hulls(id)
);

CREATE TABLE players_logic (
    id INTEGER PRIMARY KEY,

    player_id INTEGER,
    ship_id INTEGER,
    currentMission INTEGER,
    fuel INTEGER NOT NULL,
    energy INTEGER NOT NULL,
    crew_count INTEGER NOT NULL,
    hp INTEGER NOT NULL,
    FOREIGN KEY(currentMission) REFERENCES missions(id),
    FOREIGN KEY(ship_id) REFERENCES ships(id),
    FOREIGN KEY(player_id) REFERENCES players_system(id)
);

CREATE TABLE players_weapons (
    id INTEGER PRIMARY KEY,
    player_id INTEGER,
    weapon_id INTEGER,
    FOREIGN KEY(weapon_id) REFERENCES weapons(id),
    FOREIGN KEY(player_id) REFERENCES hulls(id)
);

CREATE TABLE missions_reward (
    id INTEGER PRIMARY KEY,
    mission_id INTEGER,
    item_id INTEGER,
    FOREIGN KEY(item_id) REFERENCES items(id),
    FOREIGN KEY(mission_id) REFERENCES missions(id)
);

CREATE TABLE missions_enemies (
    id INTEGER PRIMARY KEY,
    mission_id INTEGER,
    ship_id INTEGER,
    FOREIGN KEY(ship_id) REFERENCES ships(id),
    FOREIGN KEY(mission_id) REFERENCES missions(id)
);

```

```
CREATE TABLE editors (  
  id INTEGER PRIMARY KEY,  
  user_id INTEGER,  
  
  FOREIGN KEY(user_id) REFERENCES players_system(id)  
);  
  
CREATE TABLE inventory (  
  id INTEGER PRIMARY KEY,  
  player_id INTEGER,  
  item_id INTEGER,  
  
  FOREIGN KEY(item_id) REFERENCES items(id)  
  FOREIGN KEY(player_id) REFERENCES players_system(id)  
);
```

ПРИЛОЖЕНИЕ Г Листинг приложений.

Сервер:

Program.cs

```
class Program {
    static void Main(string[] args) {
        Log.Init();
        if (!DBController.Exists()) {
            Log.Write("Database not found, creating...", "system");
            DBController.CreateDatabase();
        }

        Log.Write("Server starting", "system");
        CommandProcessor.Init();
        Server server = new Server(8888);
    }
}
```

Server.cs

```
public class Server{

    private TcpListener tcpListener;
    private TcpClient connectedTcpClient;

    private Thread tcpListenerThread;

    private int Port;
    private Thread inputThread;

    // Use this for initialization
    public Server(int port) {
        Port = port;
        // Start TcpServer background thread
        tcpListenerThread = new Thread(new
ThreadStart(ListenForIncommingRequests));
        tcpListenerThread.IsBackground = true;
        tcpListenerThread.Start();

        inputThread = new Thread(new ThreadStart(ListenInput));
        inputThread.Start();
    }

    private void ListenForIncommingRequests() {
        try {
            tcpListener = new TcpListener(IPAddress.Any, Port);
            tcpListener.Start();

            Console.WriteLine($"Server is listening on port {Port}");
            Byte[] bytes = new Byte[1024];

            while (true) {
                using (connectedTcpClient =
tcpListener.AcceptTcpClient()) {
                    using (NetworkStream stream =
connectedTcpClient.GetStream()) {
                        int length;
                        while ((length = stream.Read(bytes, 0,
bytes.Length)) != 0) {
                            var incommingData = new
byte[length];

```

```

0, length);
Encoding.ASCII.GetString(incommingData);
Array.Copy(bytes, 0, incommingData,
string clientMessage =
MessageProcessor.ProcessMessage(clientMessage, this);
}
}
}
}
}
catch (SocketException socketException) {
Console.WriteLine("SocketException " +
socketException.ToString());
}
}
public void SendMessage(string message) {
if (connectedTcpClient == null) {
return;
}
try {
NetworkStream stream = connectedTcpClient.GetStream();
if (stream.CanWrite) {
byte[] serverMessageAsByteArray =
Encoding.ASCII.GetBytes(message);
stream.Write(serverMessageAsByteArray, 0,
serverMessageAsByteArray.Length);
Console.WriteLine($"Message \"{message}\" sent!");
}
}
catch (SocketException socketException) {
Console.WriteLine("Socket exception: " + socketException);
}
}
}
}

```

DBController.cs

```

struct QueryParameter {
public string parameter;
public object value;

public QueryParameter(string p, object v) {
parameter = p;
value = v;
}
}

class DBController {
public static string dbFilename = "starships.db";
public static string dbInitFilename = "44owed44m.sql";

static RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();

public static void CreateDatabase() {
Log.Write("Creating database");
if (!File.Exists(dbInitFilename)) {
Console.WriteLine($"Couldn't find {dbInitFilename}");
Console.ReadLine();
Environment.Exit(1);
}
string dbInitCommand = File.ReadAllText(dbInitFilename);
SQLiteConnection.CreateFile(dbFilename);
ExecuteNonQuery(dbInitCommand, false);
}
}

```

```

        Log.Write("Finished creating database");
    }

    public static void ResetDatabase() {
        if (!File.Exists(dbInitFilename)) {
            Log.Error($"Cannot find SQL file to recreate initial state of database at
{Path.Combine(Directory.GetCurrentDirectory(), dbInitFilename)}");
            return;
        }

        string confirmationString = GetSalt(10);
        Log.Warning($"You are about to reset database. This will erase all
information including user data. Write {confirmationString} to confirm");
        string input = Console.ReadLine();
        if (input == confirmationString) {
            if (File.Exists(dbFilename)) {
                try{
                    File.Delete(dbFilename);
                }catch(IOException e) {
                    Log.Errorreceived;
                    return;
                }
            }
            CreateDatabase();
        }
    }

    public static void CreateBackup(string msg = "") {
        if (File.Exists(dbFilename)) {
            string backupFilename = $"{dbFilename}-{DateTime.Now.ToString("dd-mm-
yyyy-hh-mm-ss")}{msg}.backup";
            File.Copy(dbFilename, backupFilename);

            Console.WriteLine($"Backup {backupFilename} is created");
        }
    }

    public static void Restore() {
        Log.Write("Restoring database");
        CreateBackup("-restore");

        var files = Directory.GetFiles(".", "*.backup");

        Console.WriteLine("Select backup");
        for (int received = 0; received < files.Length; i++) {
            Console.WriteLine($"{i}. {files[i]}");
        }

        int selectedBackup = 0;
        do {
            try {
                selectedBackup = Convert.ToInt32(Console.ReadLine());
            }
            catch (Exception e) {
                Console.WriteLinereceived;
            }
        } while (selectedBackup >= files.Length || selectedBackup < 0);

        File.Delete(dbFilename);
        File.Move(files[selectedBackup], dbFilename);

        Console.WriteLine($"Restored database from {files[selectedBackup]}");
    }
}

```

```

        public static void ExecuteNonQuery(string query, QueryParameter[] parameters,
bool record = true) {
            using (SQLiteConnection connection = ConnectToDatabase()) {
                connection.Open();
                using (SQLiteCommand command = new SQLiteCommand(query, connection)) {
                    foreach (var v in parameters) {
                        command.Parameters.AddWithValue(v.parameter, v.value);
                    }
                    command.Prepare();
                    try {
                        command.ExecuteNonQuery();
                    }
                    catch (SQLiteException e) {
                        Log.Error(e.Message);
                        connection.Close();
                        return;
                    }
                    if (record)
                        Log.Write(query, "sqlp");
                    connection.Close();
                }
            }
        }

        public static SQLiteDataReader ExecuteQuery(string query, QueryParameter[]
parameters, SQLiteConnection connection, bool record = true) {
            SQLiteDataReader data = null;
            connection.Open();
            SQLiteCommand command = new SQLiteCommand(query, connection);
            foreach (var v in parameters) {
                command.Parameters.AddWithValue(v.parameter, v.value);
            }
            command.Prepare();
            try {
                data = command.ExecuteReader();
            }
            catch (SQLiteException e) {
                Log.Error(e.Message);
                return null;
            }
            if (record) Log.Write(query, "sqlqp");
            return data;
        }
    }
}

```

MessageProcessor.cs

```

class MessageProcessor {
    public static void ProcessMessage(string input, Server server) {
        foreach (var msg in input.Split('\n')) {
            if (msg == string.Empty)
                continue;
            Console.WriteLine($"Message {msg} received from client");

            string msgType = msg.Split(':')[0];

            switch (msgType) {
                case "login":
                    ProcessLoginRequest(msg, server);
                    break;
                case "register":
                    ProcessRegisterRequest(msg, server);
                    break;
                case "req":
                    ProcessResourceRequest(msg, server);

```

```

        break;
    case "sell":
        ProcessItemSale(msg, server);
        break;
    default:
        Log.Error($"Invalid message received: {msg}");
        break;
    }
}

private static void SendQuestList(Server server) {
    SQLiteConnection connection = DBController.ConnectToDatabase();
    SQLiteDataReader data = DBController.ExecuteQuery("SELECT * FROM missions",
connection);
    string msg = "quests:\n";

    while (data.Read()) {
        msg +=
        $"{data.GetString(1)},{data.GetString(2)},{data.GetInt32(3)},{data.GetInt32(4)},{data.Get
Int32(5)}\n";
    }
    connection.Close();
    server.SendMessage(msg);
}

private static void SendInventoryInformation(Server server, int playerID) {
    SQLiteConnection connection = DBController.ConnectToDatabase();
    string query = "select items.* from items inner join inventory on items.id =
inventory.item_id where inventory.player_id = @playerID";
    QueryParameter[] parameters = new QueryParameter[1];
    parameters[0] = new QueryParameter("@playerID", playerID);
    SQLiteDataReader data = DBController.ExecuteQuery(query, parameters,
connection);
    string msg = "inventory:\n";

    while (data.Read()) {
        msg +=
        $"{data.GetInt32(0)},{data.GetString(1)},{data.GetString(2)},{data.GetInt32(3)},{data.Get
String(4)}\n";
    }
    connection.Close();
    server.SendMessage(msg);
}

private static void ProcessRegisterRequest(string msg, Server server) {
    string[] tokens = msg.Substring("register:".Length).Split(',');
    string username = tokens[0];
    string password = tokens[1];
    string publicName = tokens[2];
    string lastLogin = DateTime.Now.ToString();
    string salt = DBController.GetSalt();

    string query = $"INSERT INTO players_system (login, password, salt,
public_name, last_login_time) VALUES (@username, @password, @salt, @name, @lastLogin);
SELECT last_insert_rowid()";

    QueryParameter[] parameters = new QueryParameter[5];
    parameters[0] = new QueryParameter("@username", username);
    parameters[1] = new QueryParameter("@password",
DBController.ComputeHash(password + salt));
    parameters[2] = new QueryParameter("@salt", salt);
    parameters[3] = new QueryParameter("@name", publicName);
    parameters[4] = new QueryParameter("@lastLogin", lastLogin);

```

```

        SQLiteConnection connection = DBController.ConnectToDatabase();
        SQLiteDataReader data = DBController.ExecuteQuery(query, parameters,
connection);
        while (data.Read()) {
            server.SendMessage($"login ok:{data.GetInt32(0)}");
            return;
        }

        connection.Close();
    }
}

```

Клиент:

Client.cs

```

public class Client : MonoBehaviour
{
    private TcpClient socketConnection;
    private Thread clientReceiveThread;
    [SerializeField] private int Port;

    public List<string> incomingMessageQueue;
    public List<string> outgoingMessageQueue;
    void Start () {
        ConnectToTcpServer();
        StartCoroutine(SendMessageQueue());
    }
    private void ConnectToTcpServer () {
        try {
            clientReceiveThread = new Thread (new ThreadStart(ListenForData));
            clientReceiveThread.IsBackground = true;
            clientReceiveThread.Start();
        }
        catch (Exception e) {
            Debug.Log("On client connect exception " + e);
            PopupMessage.Show("Couldn't connect to the server");
        }
    }

    private void ListenForData() {
        try {
            socketConnection = new TcpClient("localhost", Port);
            Byte[] bytes = new Byte[1024];
            while (true) {
                using (NetworkStream stream = socketConnection.GetStream()) {
                    int length;
                    while ((length = stream.Read(bytes, 0, bytes.Length)) != 0) {
                        var incomingData = new byte[length];
                        Array.Copy(bytes, 0, incomingData, 0, length);
                        string serverMessage = Encoding.ASCII.GetString(incomingData);
                        Debug.Log("server message received as: " + serverMessage);
                        incomingMessageQueue.Add(serverMessage);
                    }
                }
            }
        }
    }
}

```



```

        }
    }
}

catch (SocketException socketException) {
    Debug.Log("Socket exception: " + socketException);
    PopupMessage.Show(socketException.ToString());
}

}

public void CommitMessage(string msg) {
    outgoingMessageQueue.Add(msg);
}

private IEnumerator SendMessageQueue(){
    while(true){
        foreach(var v in outgoingMessageQueue){
            sendMessage(v);
        }
        outgoingMessageQueue.Clear();
        yield return new WaitForEndOfFrame();
    }
}
}

```

MessageSender.cs

```

public class MessageSender : MonoBehaviour
{
    protected Client client;
    [SerializeField] float responseCheckTimer;
    [SerializeField] int maxResponseChecks;

    protected IEnumerator WaitForResponse(int messageCount = 1, bool hidePopupMessage = false){
        int respondedMessagesCount = 0;
        int responseChecksCount = 0;
        while(responseChecksCount != maxResponseChecks){
            print($"{responseChecksCount}/{maxResponseChecks}");
            if(client.incomingMessageQueue.Count != 0){
                if(CheckResponse()){
                    respondedMessagesCount++;
                    if(respondedMessagesCount >= messageCount)
                        break;
                }
            }else{
                responseChecksCount++;
                yield return new WaitForSecondsRealtime(responseCheckTimer);
            }
        }

        if(hidePopupMessage){

```

```

        PopupMessage.Hide();
    }
}

virtual protected bool CheckResponse(){
    throw new System.NotImplementedException();
}
}

```

LoginController.cs

```

public class LoginController : MonoBehaviour
{
    private void Awake() {
        client = GameObject.Find("Client").GetComponent<Client>();
    }

    public void SendLoginRequest(){
        client.CommitMessage($"login:{login.text},{password.text}");
        StartCoroutine(WaitForResponse());
    }

    public void SendRegistrationRequest(){
        client.CommitMessage($"register:{login.text},{password.text},{publicName.text}");
        StartCoroutine(WaitForResponse());
    }

    IEnumerator WaitForResponse(){
        int responseChecksCount = 0;
        while(responseChecksCount != maxResponseChecks){
            if(client.incomingMessageQueue.Count != 0){
                if(CheckResponse()){
                    break;
                }
            }else{
                responseChecksCount++;
                yield return new WaitForSecondsRealtime(responseCheckTimer);
            }
        }
    }

    bool CheckResponse(){
        foreach(var v in client.incomingMessageQueue){
            if(v.Contains("login ok")){
                print("login successful");
                client.incomingMessageQueue.Remove(v);
                authorizationCanvas.enabled = false;
                gameCanvas.enabled = true;

                GetComponent<PlayerInfo>().SetPlayerID(System.Convert.ToInt32(v.Split(':')[1]));
            }
        }
    }
}

```

```

        return true;
    }

    if(v == "incorrect password"){
        print("incorrect password");
        client.incomingMessageQueue.Remove(v);
        return true;
    }

    if(v == "incorrect login"){
        print("incorrect login");
        client.incomingMessageQueue.Remove(v);
        return true;
    }
}

return false;
}

public bool CheckPasswordStrength(){
    passwordErrorText.text = string.Empty;
    bool flag = true;
    if(password.text.Length < 6){
        passwordErrorText.text += "- the password is too short\n";
        flag = false;
    }

    if(!Regex.IsMatch(password.text, @".*(?=.*([\d]+).*)(?=.*[a-z].*).*)"){
        passwordErrorText.text += "- must contain number and a letter\n";
        flag = false;
    }

    foreach(var v in worstPasswords){
        if(password.text == v){
            passwordErrorText.text += "- this password is too common\n";
            flag = false;
        }
    }

    return flag;
}
}

```