

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Отчет по преддипломной практике

Студент

В.С. Шевцов

Руководитель

А.Н. Криштапович

Консультант от кафедры ЭВМ

А.М. Ковальчук

Нормоконтролер

Н.О. Туровец

МИНСК 2025

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К защите допустить:
Заведующий кафедрой ЭВМ
_____ Б.В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему

**ПАНЕЛЬ ДЛЯ НАСТРОЙКИ И УПРАВЛЕНИЯ ИГРОВЫМИ
СЕРВЕРАМИ**

БГУИР ДП 1–40 02 01 01 428 ПЗ

Студент

В.С. Шевцов

Руководитель

А.М. Ковальчук

Консультанты:

от кафедры ЭВМ

А.М. Ковальчук

по экономической части

В.Г. Горовой

Нормоконтролер

Н.О. Туровец

Рецензент

МИНСК 2025

Решением рабочей комиссии
допущен(а) к защите дипломного проекта
Председатель рабочей комиссии

_____ (_____)
Подпись Инициалы и фамилия

« ____ » _____ 2025 г.

СОДЕРЖАНИЕ

Введение.....	6
1 Обзор литературы	8
1.1 Исследование предметной области.....	8
1.2 Обзор аналогов	9
1.3 Обзор средств разработки	14
1.4 Постановка задачи	21
2 Системное проектирование.....	23
2.1 Обоснование выбора средств разработки	23
2.2 Общая структура приложения	24
3 Функциональное проектирование	29
3.1 Работа с базой данных	29
3.2 Интерфейсы сервисов.....	32
3.3 Data transfer objects.....	32
3.4 Классы общих данных	36
3.5 Контроллеры.....	38
3.6 Сервисы.....	42
3.7 Классы бизнес-логики	48
3.8 Классы инициализации.....	52

ВВЕДЕНИЕ

На сегодняшний день доля людей, играющих в игры различных жанров, составляет более 3.1 миллиарда человек [1], из которых около 47 процентов предпочитают многопользовательские (соревновательные, кооперативные и другие) игры [2].

Многопользовательские и кооперативные игры обладают рядом положительных аспектов, которые обогащают игровой опыт и способствуют развитию определенных навыков у игроков.

Совместная игра стимулирует общение между участниками, что способствует улучшению коммуникативных навыков. Игроки учатся ясно выражать свои мысли, слушать собеседников и эффективно передавать информацию, что положительно сказывается на их взаимодействии как в виртуальном мире, так и в реальной жизни.

Многопользовательские игры требуют от участников совместной работы для достижения общих целей, что усиливает социальное взаимодействие и способствует укреплению командного духа. Игроки учатся эффективно распределять роли и поддерживать друг друга в сложных ситуациях. Это особенно важно в играх, где успех зависит от слаженности действий команды.

Современные технологии позволяют использовать онлайн-игры не только для развлечения, но и в образовательных целях. Геймификация обучения становится все более популярной, так как игровой процесс способен повысить вовлеченность и мотивацию учащихся, облегчая усвоение сложных концепций.

Резюмируя всё вышесказанное, можно сделать вывод, что многопользовательские и кооперативные игры не только обогащают игровой процесс, но и способствуют развитию важных личностных и социальных навыков, делая их ценным инструментом для обучения и развлечения.

В современном мире индустрия онлайн-игр развивается стремительными темпами, что приводит к росту количества игровых серверов, используемых как любителями, так и профессиональными игровыми сообществами. Эффективное управление игровыми серверами становится важной задачей, требующей надежных и удобных инструментов. Администраторы серверов сталкиваются с множеством сложностей, включая настройку серверов, мониторинг их состояния, управление нагрузкой и обеспечение безопасности. Всё это требует от пользователей определенных знаний и навыков, что становится препятствием для игроков, желающих создать собственный сервер, но не обладающих достаточным опытом в администрировании. Для решения этих задач необходимо разработать удобную и функциональную систему управления, которая позволит автоматизировать ключевые процессы.

Одним из ключевых аспектов актуальности данной темы является экономическая и технологическая сторона вопроса. Развитие облачных технологий и контейнеризации позволяет автоматизировать процесс

развертывания игровых серверов, делая их управление более удобным и доступным. В то же время многие существующие решения требуют работы с командной строкой и сложными конфигурационными файлами, что затрудняет их использование для рядовых пользователей. Упрощение администрирования серверов не только снижает временные и финансовые затраты, но и способствует развитию любительских и профессиональных игровых сообществ. Также стоит отметить, что удобные инструменты управления серверами востребованы как среди энтузиастов, так и среди коммерческих организаций, предоставляющих игровые серверы в аренду. Возможность гибкой настройки ресурсов системы и контроля над серверами позволяет оптимизировать их работу и минимизировать риски, связанные с перегрузками и сбоями. Это особенно важно в условиях роста популярности многопользовательских игр, где стабильность серверов напрямую влияет на качество игрового процесса.

Дипломный проект посвящен разработке веб-панели управления игровыми серверами, которая обеспечит удобный интерфейс для их мониторинга и администрирования. Система будет представлять собой веб-приложение, которое позволит пользователям запускать, останавливать серверы, а также управлять их конфигурацией через браузер. Использование веб-технологий обеспечит доступность панели управления из любой точки мира.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Исследование предметной области

В рамках данного исследования были рассмотрены научные работы и статьи, посвященные изучению положительного влияния видеоигр на личные качества и навыки людей.

В университете Хьюстона провели исследование, чтобы узнать, как игры влияют на трудовую деятельность людей [3].

Исследование в течение 20 лет отслеживало привычки 23 трудоустроенных геймеров, преданных своим игровым консолям. Его цель заключалась в том, чтобы выяснить, существуют ли качества, которые игроки используют в виртуальном мире, и которые непреднамеренно переносятся в их профессиональную деятельность.

Респонденты отметили, что подходят к своей работе так же, как к вызовам, с которыми сталкиваются в любимых играх. Они хвалили свое увлечение за развитие терпения, настойчивости и мотивации. Некоторые заявили, что достижения в видеоиграх помогли им обрести уверенность в себе, что положительно сказалось на их карьере. Еще больше опрошенных отметили, что игры помогли им лучше осознавать свои навыки в командной среде. Некоторые даже освоили лидерские качества, например, один IT-специалист, принимавший участие в исследовании.

Одним из исследований, посвященных влиянию видеоигр, стал отчет The Power of Play[4]. Согласно ему, одним из ключевых аспектов, который делает видеоигры ценным элементом повседневной жизни, является их способность помогать людям справляться с эмоциональными и психологическими нагрузками. Многие игроки отмечают, что игровая среда дает им возможность отвлечься от повседневных проблем, структурировать мысли и найти эмоциональную разрядку. Кроме того, видеоигры создают безопасное пространство для самовыражения, позволяя пользователям преодолевать личные барьеры и повышать уверенность в себе.

В онлайн-играх математика и физика играют ключевую роль в создании увлекательных и реалистичных игровых миров. Математика используется в первую очередь для графики и 3D-моделирования. При отображении объектов в игровом мире применяются различные математические модели, такие как векторные и матричные вычисления, чтобы трансформировать, вращать и перемещать 3D-модели. Алгоритмы, основанные на математике, также активно используются для оптимизации путей, генерации случайных событий и обработки данных, что позволяет создавать разнообразные игровые сценарии. В искусственном интеллекте NPC, например, вычисляются оптимальные пути через игровые локации и принимаются решения на основе вероятностных моделей.

Физика, в свою очередь, помогает создать динамичные и реалистичные взаимодействия объектов в мире игры. Например, для имитации движений персонажей, транспортных средств или любых других объектов используется

математическое моделирование движения, основанное на законах физики, таких как гравитация, инерция и сила трения. Столкновения объектов также подчиняются законам физики, что делает их более натуральными и предсказуемыми для игрока. Кроме того, для симуляции разрушений или воздействия различных сил на объекты игры применяются более сложные физические уравнения, которые помогают создавать эффектные сцены, такие как обрушение зданий или взрывы.

Не стоит забывать и о важной роли сетевых технологий и математики в многопользовательских играх. Для того чтобы обеспечить бесперебойную работу игры для всех игроков, необходимы алгоритмы для оптимизации передачи данных, минимизации задержек и синхронизации событий в реальном времени. В дополнение к этому криптографические методы, использующие математические алгоритмы шифрования, играют важную роль в обеспечении безопасности данных игроков и защиты от взломов. В целом, математика и физика в онлайн-играх обеспечивают не только графическую составляющую, но и создание динамичной, интересной и безопасной игровой среды.

Помимо этого, игровая среда является благоприятной платформой для развития креативности. Многие игры предлагают пользователям создавать собственные миры, прорабатывать сценарии и решать нетривиальные задачи. Это развивает воображение, художественные и дизайнерские навыки, а также способность мыслить инновационно.

1.2 Обзор аналогов

Любой программный продукт разрабатывается для выполнения конкретных задач. Создаваемое решение не является уникальным, поскольку на момент разработки уже существует множество аналогов. Поэтому, чтобы создать конкурентоспособный сервис, необходимо провести анализ существующих решений, выявить их ключевые функции, преимущества и недостатки.

В контексте сравнения с разрабатываемой панелью будет использоваться понятие хостинг – сервис хранения данных, непосредственно к которому панель предоставляет доступ. Сравнение будет проводиться по максимальному количеству критериев, доступных без оплаты, так как для просмотра полного функционала в большинстве случаев нужно арендовать сервер.

1.2.1 Axenhost

Axenhost[5] – хостинг с наличием условно-бесплатной модели предоставления серверов помимо основных, платных серверов, а также обильным количеством игр. На рисунке представлена главная страница для конкретной игры хостинга.



Рисунок 1.1 – Главная страница игры хостинга

Преимущества:

- широкий список игр, для которых можно создать сервер;
- гибкий выбор ресурсов для создания сервера;
- возможность добавлять других людей в команды для группового управления сервером.

Недостатки:

- хранить можно только 1 резервную копию;
- нет наглядного управления настройками сервера;
- обилие рекламы;
- условно-бесплатная модель аренды очень ограничена валютой сайта, которая не позволит арендовать на некоторые игры даже минимальный тариф непрерывно;
- наличие так называемого премиума(помимо тарифов серверов), без которого не получить некоторых вещей даже на платных тарифах (например, использование sftp для подключения к файловой системе сервера);
- прослойка в виде валюты сайта для тарифов, что сильно размывает представление о реальной стоимости серверов.

1.2.2 Host-DW

Host-DW[6] – небольшой хостинг для terraria с несколькими тарифами.

Преимущества:

- наличие бесплатного тарифа;
- присутствие заранее настроенных заготовок для сервера.

Недостатки:

- при входе или регистрации пароль не замаскирован в поле ввода;
- номинально есть восстановление пароля, однако оно не работает;
- неудобная навигация по сайту, некоторый функционал не понятен интуитивно;

- бесплатный тариф максимально урезан: нет доступа к файлам, нельзя поставить свою карту, доступен только минимальный размер мира, количество игроков на сервере ограничено до 4;
- наличие только одной игры.

На рисунке 1.2 представлена панель управления сервером.

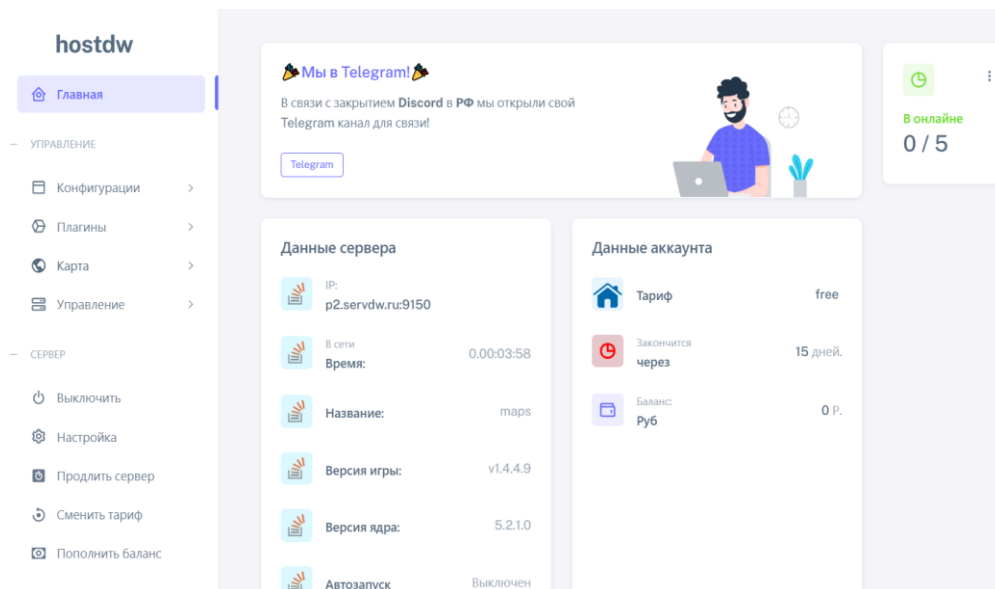


Рисунок 1.2 – Главная страница панели управления

1.2.3 Aternos

Aternos [7] – полностью бесплатный хостинг для minecraft. Предоставляет полноценное управление сервером с частичным ограничением доступа к его файлам. На рисунке 1.3 представлена главная страница панели со вкладкой навигации.

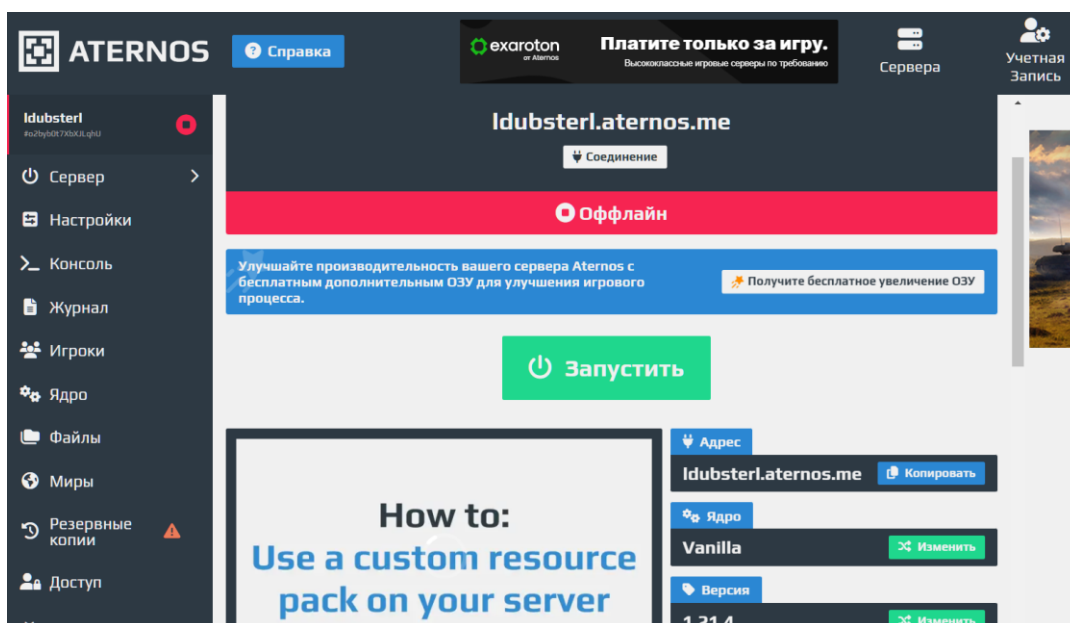


Рисунок 1.3 – Вкладка запуска сервера

Преимущества:

- полностью бесплатен;
- доступна смена версии ядра сервера без его пересоздания;
- есть вкладка для графической настройки сервера;
- есть dns-сервер для упрощения запоминания адреса игрового сервера;
- возможность настроить совместное управление сервером;
- имеется возможность делать резервные копии мира.

Недостатки:

- заранее не известны предоставляемые вычислительные мощности;
- на практике вычислительных ресурсов не хватает для комфортной игры нескольких человек;
- нет доступа к загрузке файлов на сервер, только редактирование существующих;
- обязательный просмотр рекламы при выполнении некоторых действий;
- только одна игра.

1.2.4 Hosting-minecraft

Hosting-minecraft [8] – очередной хостинг для серверов майнкрафта, который помимо аренды игровых серверов предоставляет полноценную аренду персонального сервера. На рисунках 1.4 – 1.5 показаны главная страница сайта, главная страница панели и вкладки управления панели.

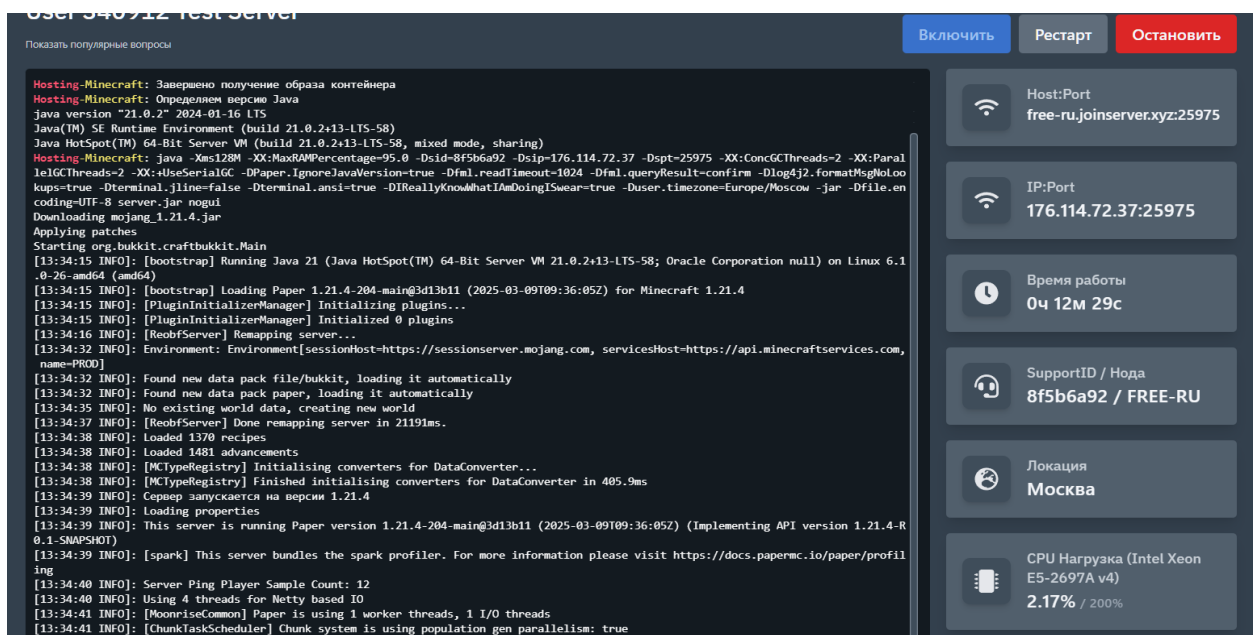


Рисунок 1.4 – Основная вкладка панели управления

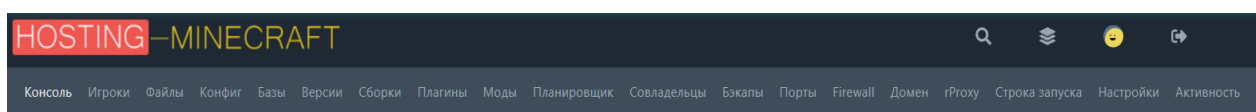


Рисунок 1.5 – Вкладки панели управления для краткого описания функционала

Преимущества:

- наличие тестового тарифа;
- подробное отображение использования ресурсов (процессор, память, диск);
- очень широкий функционал панели управления;
- возможность аренды VPS/VDS серверов помимо игровых;
- наличие списка готовых сборок, чтобы упростить установку модов;
- наличие функции подбора тарифа по критериям.

Недостатки:

- при использовании тестового сервера, его нужно пересоздавать через сутки;
- для изменения настроек нужно менять непосредственно файл с этими настройками;
- поддержка только одной игры.

1.2.5 XLGAMES.PRO

XLGAMES.PRO [9] – хостинг с большим списком игр и, так же, как и прошлый, помимо игровых серверов предоставляет VPS/VDS. На рисунке 1.6 представлен список игр, для которых хостинг предоставляет создание сервера.



































 Все сервера Серверов: 792 Игроков: 3528	 The Isle Серверов: 11 Игроков: 478	 DayZ Standalone R Серверов: 300 Игроков: 1968	 SPACE STATION 14 Серверов: 5 Игроков: 89	 FIVEM - GTA Серверов: 8 Игроков: 81
 Rust Серверов: 25 Игроков: 115	 Battlefield 4 EPS 60Hz Серверов: 30 Игроков: 251	 Battlefield 3 Серверов: 15 Игроков: 128	 PROJECT ZOMBOID Серверов: 31 Игроков: 118	 V Rising Серверов: 5 Игроков: 43
 Battlefield 4 Серверов: 16 Игроков: 55	 Arma Reforger Серверов: 14 Игроков: 51	 7 Days to Die Серверов: 20 Игроков: 44	 Call of Duty World at War Серверов: 2 Игроков: 11	 Conan Exiles Серверов: 29 Игроков: 21
 Minecraft Серверов: 13 Игроков: 13	 Euro Truck Simulator 2 Серверов: 4 Игроков: 8	 Farming Simulator 2025 Серверов: 77 Игроков: 26	 SOULMASK Серверов: 8 Игроков: 5	 Mordhau Серверов: 5 Игроков: 2
 Valheim Серверов: 51 Игроков: 7	 Farming Simulator 2022 Серверов: 35 Игроков: 7	 American Truck Simulator Серверов: 1 Игроков: 2	 Battlefield 4 EPS 120Hz Серверов: 2 Игроков: 1	 Battlefield Hardline Серверов: 3 Игроков: 1
 Space Engineers Серверов: 4 Игроков: 1	 Farming Simulator 2019 Серверов: 4 Игроков: 1	 ARK: Survival Ascended Серверов: 2 Игроков: 1	 Garrys Mod Серверов: 2 Игроков: 0	 Path Of Titans Серверов: 3 Игроков: 0
 Squad Серверов: 1 Игроков: 0	 Arma 3 Серверов: 1 Игроков: 0	 THE FRONT Серверов: 2 Игроков: 0	 Battlefield 4 ZEM Серверов: 1 Игроков: 0	 ProCon Layer Серверов: 19 Игроков: 0
 Battlefield 3 VU Серверов: 1 Игроков: 0	 Barotrauma Серверов: 5 Игроков: 0	 Palworld Серверов: 6 Игроков: 0	 Call of Duty 2 Серверов: 1 Игроков: 0	 Battlefield Bad Company 2 Серверов: 1 Игроков: 0
 Last Oasis Серверов: 1 Игроков: 0	 Night of the Dead Серверов: 1 Игроков: 0	 ARK: Survival Evolved Серверов: 2 Игроков: 0	 Terraria Серверов: 1 Игроков: 0	 Satisfactory Серверов: 1 Игроков: 0
 Icarus Серверов: 11 Игроков: 0	 Team Fortress 2 Серверов: 1 Игроков: 0	 Sons of the Forest Серверов: 1 Игроков: 0	 Enshrouded Серверов: 8 Игроков: 0	 Killing Floor Серверов: 1 Игроков: 0
 No One Survived Серверов: 1 Игроков: 0				

Рисунок 1.6 – Список игр, для которых можно заказать сервер

Преимущества:

- наличие VPS/VDS;
- большой список игр, для которых можно сделать сервер;
- несколько локаций для аренды сервера;
- возможность включить отображение своего сервера в общем мониторинге;

Недостатки:

- отсутствие бесплатного/пробного тарифа;
- отсутствие тарифов для маленьких серверов;

– в сравнении с разработкой по теме диплома – отсутствие Don't starve together в списке игр.

1.3 Обзор средств разработки

Разработка любого программного продукта требует тщательного выбора инструментов и технологий. В данном разделе рассматриваются основные средства разработки, которые могут быть использованы для создания данного проекта: фреймворки, библиотеки для бэкенд и фронтенд части, языки программирования, средства виртуализации и системы управления базами данных.

1.3.1 Фронтенд

Фронтенд-фреймворки и библиотеки представляют собой инструменты для упрощения и ускорения разработки пользовательских интерфейсов веб-приложений. Они включают в себя готовые решения для работы с DOM (Document Object Model), стилизацией, управлением состоянием и взаимодействием с сервером. Использование таких инструментов позволяет разработчикам создавать динамичные, адаптивные и интерактивные веб-приложения с минимальными затратами времени и ресурсов.

Список фреймворков:

- Angular [10] – фреймворк от Google, основанный на TypeScript. Он предоставляет полный набор инструментов для разработки сложных одностраничных приложений (SPA), включая встроенные модули для работы с формами, HTTP-запросами и маршрутизацией;

- Vue.js [11] – фреймворк, сочетающий в себе простоту и гибкость. Vue.js легко интегрируется в существующие проекты и позволяет создавать как небольшие виджеты, так и полнофункциональные веб-приложения;

- Svelte [12] – фреймворк, отличающийся подходом к рендерингу интерфейсов. В отличие от React, Angular и Vue, он не использует виртуальный DOM, а компилирует код на этапе сборки, что значительно снижает нагрузку на браузер;

- Next.js [13] – фреймворк, основанный на React, разработанный компанией Vercel. Он предоставляет возможности серверного рендеринга (SSR), статической генерации страниц (SSG) и автоматической оптимизации, что делает его отличным выбором для создания высокопроизводительных веб-приложений и SEO-оптимизированных сайтов;

- Bootstrap [14] – популярный CSS-фреймворк, который также включает JavaScript-компоненты. Разработанный Twitter, он упрощает создание адаптивных и стилизованных пользовательских интерфейсов благодаря готовым сеткам, кнопкам, модальным окнам и другим элементам UI. Хотя Bootstrap больше ориентирован на стилизацию, его часто используют в связке с другими JS-фреймворками.

Библиотеки:

– React [15] – библиотека, разработанная Facebook (ныне Meta), предназначенная для построения пользовательских интерфейсов. React использует виртуальный DOM, что повышает производительность, и компонентный подход, упрощающий разработку сложных интерфейсов;

– SolidJS [16] – это реактивная библиотека для создания пользовательских интерфейсов, которая фокусируется на высокой скорости работы. В отличие от React, она не использует Virtual DOM, а вместо этого применяет реактивные сигналы, что значительно уменьшает количество повторных ререндеров. SolidJS компилирует код в нативный JavaScript, устраняя лишние вычисления и обеспечивая максимальную эффективность;

– Alpine.js [17] – это минималистичная библиотека, предназначенная для создания реактивных интерфейсов с декларативным синтаксисом. Она позволяет управлять состоянием и поведением элементов прямо в HTML, без сложного JavaScript-кода. Благодаря своему небольшому размеру (около 10KB) Alpine.js идеально подходит для небольших проектов, интерактивных компонентов и улучшения статических страниц.

1.3.2 Бэкенд

Бэкенд-фреймворки представляют собой набор инструментов и библиотек, предназначенных для упрощения разработки серверной части веб-приложений. Они помогают разработчикам управлять базами данных, обрабатывать HTTP-запросы, реализовывать аутентификацию, управлять бизнес-логикой и обеспечивать безопасность.

Бэкенд-фреймворки могут быть полноценными (full-stack), предоставляющими все необходимое для разработки (например, Django, Spring Boot), или минималистичными (микрофреймворками), которые дают лишь базовый функционал, оставляя разработчику свободу выбора дополнительных инструментов (например, Flask, Express.js).

Популярные бэкенд-фреймворки для разных языков:

– ASP.NET Core (C#) [18] – мощный и производительный фреймворк от Microsoft, предназначенный для создания веб-приложений и API. Он поддерживает кроссплатформенность, высокую масштабируемость и интеграцию с облачными сервисами;

– Spring Boot (Java) [19] – один из самых популярных Java-фреймворков, предоставляющий удобные инструменты для быстрого создания веб-приложений и микросервисов. Он включает мощные средства работы с базами данных, безопасность и автоматическую настройку;

– Django (Python) [20] – full-stack фреймворк, использующий концепцию "батарейки в комплекте", что означает наличие встроенных инструментов для аутентификации, работы с ORM, админ-панели и многого другого. Он обеспечивает быструю разработку и безопасность;

– Flask (Python) [21] – микрофреймворк, который предоставляет базовый набор инструментов для веб-разработки, оставляя разработчику свободу выбора дополнительных библиотек. Подходит для создания простых API и небольших веб-приложений;

– Express.js (JavaScript, Node.js) [22] – минималистичный фреймворк для Node.js, который широко используется для создания API и серверных приложений. Он прост в освоении, имеет огромное сообщество и гибкую архитектуру;

– Ruby on Rails (RoR) [23] – серверный фреймворк для языка Ruby, предназначенный для быстрой разработки веб-приложений. Он следует принципам "конвенция важнее конфигурации" и DRY (Don't Repeat Yourself), что позволяет минимизировать объем кода и ускорить процесс разработки;

– Laravel (PHP) [24] – современный PHP-фреймворк, который предоставляет удобный синтаксис, мощные инструменты для работы с базами данных (Eloquent ORM) и встроенные механизмы аутентификации, маршрутизации и тестирования.

1.3.3 Языки программирования

Выбор языка программирования зависит от выбранного языка для бэкенда или фреймворка, а также от требований проекта. Для разработки веб-панели управления серверами могут быть использованы следующие языки:

– JavaScript [25] – основной язык программирования для фронтенда, отвечающий за динамическое поведение страниц. С его помощью можно создавать интерактивные элементы, анимации, валидацию форм и взаимодействие с сервером через API. Также, хотя JavaScript изначально был языком для фронтенда, благодаря платформе Node.js он активно используется и в бэкенде. Он позволяет создавать быстрые и масштабируемые серверные приложения, поддерживает асинхронную обработку запросов и широко применяется в разработке REST API и микросервисов;

– Python [26] – универсальный язык, популярный в веб-разработке благодаря фреймворкам Django и Flask. Отличается простотой синтаксиса, богатой экосистемой и используется как для классических веб-приложений, так и для работы с машинным обучением, анализом данных и автоматизацией;

– PHP [27] – один из старейших языков для веб-разработки, широко применяемый для серверной части сайтов. Основным фреймворком – Laravel, а также популярные CMS, такие как WordPress и Drupal, работают на PHP;

– Java [28] – мощный объектно-ориентированный язык, который часто используется для построения масштабируемых корпоративных приложений. В веб-разработке применяется с фреймворками Spring Boot и Jakarta EE, обеспечивая надежность и высокую производительность;

– C# [29] – язык, активно используемый в среде Microsoft для разработки веб-приложений на платформе ASP.NET Core. Отличается высокой производительностью, безопасностью и глубокой интеграцией с облачными решениями Microsoft Azure;

– Go (Golang) [30] – язык, разработанный Google, известный своей высокой производительностью и удобством работы с многопоточностью. Широко применяется в микросервисной архитектуре, облачных сервисах и веб-приложениях;

– Ruby [31] – язык, известный благодаря фреймворку Ruby on Rails, который позволяет быстро создавать веб-приложения с удобной структурой и минимальным количеством кода.

– Rust [32] – современный язык с упором на безопасность и производительность. Используется в высоконагруженных веб-приложениях, разработке блокчейнов и системного программирования.

1.3.4 Средства виртуализации и контейнеризации

Виртуализация и контейнеризация – технологии, позволяющие изолировать и управлять вычислительными средами для эффективного использования ресурсов. Виртуализация создает полноценные виртуальные машины с собственной операционной системой, обеспечивая гибкость и безопасность при развертывании программ. Контейнеризация, в свою очередь, использует легковесные контейнеры, работающие на общем ядре ОС, что делает их более производительными и удобными для развертывания приложений в облаке и микросервисной архитектуре. Эти технологии позволяют оптимизировать ресурсы, ускорять развертывание и упрощать управление инфраструктурой.

Средства контейнеризации:

– Docker [33] – одна из самых популярных платформ для контейнеризации, позволяющая разрабатывать, упаковывать и развертывать приложения в легковесных изолированных контейнерах. Docker обеспечивает кроссплатформенную совместимость, высокую гибкость и удобство в управлении зависимостями. Он широко используется в DevOps-процессах, облачных сервисах и CI/CD-пайплайнах;

– Podman [34] – инструмент контейнеризации, который предлагает функциональность, схожую с Docker, но работает без центрального демона, что повышает безопасность. Он позволяет запускать контейнеры без root-доступа и поддерживает стандарт OCI, что делает его удобным выбором для интеграции с существующими системами безопасности Linux;

– LXC (Linux Containers) [35] – технология контейнеризации, ориентированная на создание окружений, максимально приближенных к полноценным виртуальным машинам. В отличие от Docker, LXC предоставляет более низкоуровневый доступ к системе, что делает его удобным для изоляции целых операционных систем и использования в серверной инфраструктуре.

Технологии виртуализации:

– OpenVZ [36] – система виртуализации на уровне операционной системы, которая позволяет запускать несколько изолированных контейнеров (виртуальных сред) на одном сервере. В отличие от Docker, OpenVZ использует общее ядро ОС, что снижает накладные расходы на ресурсы, но ограничивает возможности работы с разными версиями операционных систем. Это решение популярно среди хостинг-провайдеров благодаря своей эффективности;

– Hyper-V [37] – это технология виртуализации, разработанная Microsoft, которая позволяет создавать и управлять виртуальными машинами (VM) на операционных системах Windows. Она предоставляет возможность запускать несколько изолированных операционных систем на одном физическом сервере, эффективно распределяя вычислительные ресурсы;

– KVM (Kernel-based Virtual Machine) [38] – это технология аппаратной виртуализации, встроенная в ядро Linux, позволяющая запускать полноценные виртуальные машины с высокой производительностью. В отличие от контейнеризации, KVM создает изолированные окружения с собственным ядром ОС, что делает его подходящим для традиционной виртуализации серверов и облачных вычислений.

1.3.5 Системы управления базами данных

Системы управления базами данных (СУБД) – программное обеспечение, которое помогает создавать, управлять и взаимодействовать с базами данных. Она позволяет эффективно хранить, организовывать, извлекать и обновлять данные. СУБД по способу хранения делятся на 2 типа: реляционные и нереляционные (nosql). В данном разделе представлены самые популярные системы обоих типов.

Реляционные:

– MySQL [39] – это одна из самых популярных реляционных СУБД с открытым исходным кодом. Она используется для хранения данных в структурированном виде, поддерживает SQL-запросы для извлечения, обновления и удаления информации. MySQL известна своей высокой производительностью, надежностью и простотой в использовании;

– Microsoft SQL Server (MSSQL) [40] – это коммерческая реляционная СУБД, разработанная Microsoft, предназначенная для хранения и управления большими объемами данных. MSSQL используется в корпоративных приложениях и облачных сервисах, обеспечивая высокую степень безопасности, масштабируемости и надежности;

– PostgreSQL [41] – это объектно-реляционная СУБД с открытым исходным кодом, которая известна своей гибкостью, надежностью и поддержкой сложных запросов. PostgreSQL поддерживает стандарты SQL, но также предоставляет дополнительные возможности, такие как хранение и обработка данных типов JSON и XML, а также поддержку пользовательских типов данных.

Нереляционные СУБД:

– MongoDB [42] – это документно-ориентированная СУБД, которая хранит данные в формате BSON (бинарный JSON). В отличие от традиционных реляционных баз данных, MongoDB позволяет гибко управлять неструктурированными и полуструктурированными данными, что делает её подходящей для работы с большими объемами разнообразных данных;

– Cassandra [43] – это распределенная колонка-ориентированная СУБД, предназначенная для обработки больших объемов данных в реальном времени. Она позволяет эффективно работать с данными, которые необходимо

хранить в распределенном виде, и предлагает высокую доступность и отказоустойчивость;

– Redis [44] – это хранилище данных в формате ключ-значение, которое используется для кэширования, обработки данных в реальном времени и улучшения производительности приложений. Redis часто применяется для хранения сессий, очередей сообщений и кэшированных данных, что делает его идеальным для приложений с высокой нагрузкой и требованиями к скорости отклика.

1.3.6 Стили архитектуры веб-приложений

При разработке веб-приложений используются различные архитектурные стили, каждый из которых определяет принципы построения структуры кода, взаимодействия компонентов и организации данных. Выбор подходящего стиля зависит от требований к проекту, уровня масштабируемости, удобства сопровождения и скорости разработки. Среди наиболее распространенных подходов можно выделить RESTful API, MVC и микросервисную архитектуру.

REST (Representational State Transfer) [45] представляет собой стиль архитектуры, основанный на принципах взаимодействия по протоколу HTTP. В этом подходе сервер предоставляет доступ к ресурсам, используя унифицированные HTTP-методы, такие как GET, POST, PUT, DELETE. Данные обычно передаются в формате JSON или XML, что делает API удобным для интеграции с различными клиентскими приложениями.

Основным преимуществом REST является его простота и масштабируемость. Благодаря отсутствию состояния (stateless) серверу не нужно хранить информацию о предыдущих запросах, что облегчает распределение нагрузки и упрощает горизонтальное масштабирование. REST также поддерживает кэширование, что повышает производительность и снижает нагрузку на сервер.

MVC [46] – классический архитектурный шаблон, который разделяет приложение на три уровня: модель (Model), представление (View) и контроллер (Controller).

Модель отвечает за работу с данными, бизнес-логику и взаимодействие с базой данных. Представление формирует интерфейс, отображая данные пользователю. Контроллер обрабатывает входящие запросы, обновляет модель и выбирает нужное представление.

Основное преимущество стиля – четкое разделение логики, что делает код более читаемым и поддерживаемым.

1.3.7 Способы организации кода внутри приложения

При разработке приложений используются различные стили организации кода, которые помогают структурировать логику, упростить поддержку и масштабирование. Вот основные из них:

Микросервисный стиль [47] проектирования предполагает разбиение приложения на небольшие независимые сервисы, каждый из которых

выполняет свою конкретную задачу. Взаимодействие между сервисами происходит через API или асинхронные сообщения (например, через Kafka, RabbitMQ или Redis Pub/Sub).

Одним из ключевых преимуществ микросервисов является гибкость и масштабируемость. Разные части системы могут разрабатываться и разворачиваться независимо, что упрощает обновление и поддержку кода. Также можно использовать разные технологии и базы данных в зависимости от требований каждого сервиса.

Монолитный стиль[49] проектирования предполагает, что все компоненты приложения находятся в единой кодовой базе и работают как единое целое. Вся логика – от пользовательского интерфейса до базы данных – тесно связана и разрабатывается в рамках одного проекта.

Одним из ключевых преимуществ монолита является простота разработки, отладки и развертывания. Все части системы находятся в одном месте, что упрощает управление зависимостями и ускоряет процесс разработки. Кроме того, монолитные приложения обычно требуют меньше инфраструктурных затрат по сравнению с распределенными системами.

Однако с ростом сложности системы монолит может стать трудным для масштабирования, поскольку любые изменения требуют развертывания всего приложения. Этот стиль проектирования чаще всего используется в небольших и средних проектах, где важны скорость разработки и минимальные накладные расходы на поддержку.

Слоистая архитектура (Layered Architecture)[50] основана на разделении приложения на несколько уровней, каждый из которых выполняет свою роль. Как правило, выделяются три основных слоя:

- презентационный (UI) – отвечает за взаимодействие с пользователем;
- бизнес-логика (Application/Domain) – содержит основные правила обработки данных;
- доступ к данным (Data Access Layer) – взаимодействует с базой данных.

Основное преимущество такого подхода – четкое разделение ответственности. Код становится более организованным, а каждый слой может разрабатываться и тестироваться отдельно, однако слоистая архитектура может привести к избыточности кода и усложнению взаимодействия между слоями. Кроме того, при некорректном проектировании слои могут становиться слишком зависимыми друг от друга, что усложняет внесение изменений.

Чистая архитектура (Clean Architecture)[51], предложенная Робертом Мартином, строится на принципе разделения приложения на независимые уровни, где бизнес-логика не зависит от фреймворков, баз данных и других внешних инструментов.

Приложение делится на четыре главных слоя:

- entities – основные сущности и правила бизнеса;
- use Cases – сценарии использования, в которых описаны основные процессы системы;

- interface Adapters – адаптеры, связывающие бизнес-логику с внешним миром;
- frameworks & Drivers – внешний уровень, включающий базы данных, API, UI и другие зависимости.

Главное преимущество чистой архитектуры – гибкость и тестируемость. Можно легко менять инфраструктуру, не затрагивая бизнес-логику, а код остается структурированным и понятным.

1.3.8 Способы авторизации и аутентификации

Авторизация играет ключевую роль в защите данных и управлении доступом пользователей к различным частям веб-приложения. Существует множество методов аутентификации и авторизации, но три наиболее распространенных подхода – это JWT (JSON Web Token), OAuth 2.0 и сессионная аутентификация. Каждый из них имеет свои особенности, преимущества и области применения.

JWT представляет собой самодостаточный токен, который содержит зашифрованные данные о пользователе. Он передается клиенту после входа и добавляется к каждому запросу в заголовке *Authorization*.

Этот метод удобен для масштабируемых и распределенных систем, так как серверу не нужно хранить состояние сессии, однако токены трудно отозвать до истечения срока, поэтому используются короткоживущие токены и refresh-токены.

OAuth 2.0 используется для авторизации через сторонние сервисы (например, Google, Facebook) и доступа к API. Вместо передачи учетных данных пользователь получает токен доступа, который позволяет работать с защищенными ресурсами.

Этот метод безопасен и гибок, так как позволяет ограничивать доступ к определенным данным. Однако его реализация сложнее, так как требует настройки серверов авторизации и управления разрешениями.

При сессионной авторизации сервер создает сессию после входа пользователя и отправляет ему cookie с session ID. Клиент автоматически передает этот идентификатор при каждом запросе, что позволяет серверу проверять аутентификацию.

Этот метод простой в реализации и позволяет управлять сессиями, но хуже масштабируется, так как требует хранения данных на сервере. В больших системах сессии часто хранятся в Redis для обеспечения высокой производительности.

1.4 Постановка задачи

Темой проекта является разработка веб-панели управления игровыми серверами, которая обеспечит удобный интерфейс для их мониторинга и администрирования. Система будет представлять собой веб-приложение, которое позволит пользователям запускать, останавливать серверы, а также управлять их конфигурацией через браузер.

Основной целью разработки является упрощение управления сервером для обычного пользователя, который не обладает глубокими знаниями в области его создания и настройки. Панель позволит минимизировать необходимость работы с командной строкой и сложными конфигурационными файлами, предоставляя интуитивно понятный интерфейс для взаимодействия с игровыми серверами. Данную панель можно использовать как локально для создания частного сервера на своём персональном компьютере, так и расположить, например, в кластере центра обработки и хранения данных с целью получения прибыли от сдачи в аренду серверов, так как особенности реализации позволяют гибко настраивать ограничения на использование ресурсов системы и, например, ввести тарифы в зависимости от требуемых ресурсов.

Разрабатываемая веб-панель будет полезна как для небольших сообществ, так и для обычных пользователей, желающих запустить игровой сервер без необходимости разбираться в сложных технических аспектах. Она позволит сократить временные затраты на администрирование, автоматизировать рутинные процессы и повысить удобство работы с серверами.

Исходя из всего вышесказанного разрабатываемое программное приложение будет состоять из 2 частей: серверной и клиентской.

В серверной части будет реализован следующий функционал:

- модуль регистрации, аутентификации и авторизации пользователя для получения доступа к своим серверам;
- настройка контейнеров, в которых будут запускаться сервера;
- блок создания и настройки серверов перед их использованием;
- модуль взаимодействия пользователя с сервером в реальном времени;
- блок управления личным профилем пользователя.

Клиентская часть состоит из:

- страницы для регистрации и аутентификации;
- страница профиля пользователя;
- страница для выбора или создания сервера;
- страницы запуска, настройки сервера и управления его файловой системой.

Таким образом, дипломный проект направлен на создание удобной отказоустойчивой системы управления игровыми серверами, которая позволит пользователям эффективно контролировать и настраивать игровые процессы, обеспечивая высокую стабильность и производительность серверов.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

2.1 Обоснование выбора средств разработки

2.1.1 В качестве фреймворка для разработки клиентской части(фронтенда) был выбран React. Выбор пал на эту технологию благодаря ее гибкости, высокому уровню абстракции и широким возможностям построения сложных пользовательских интерфейсов. Использование компонентного подхода упрощает создание повторно используемых элементов интерфейса, что позволяет легко поддерживать и развивать кодовую базу.

Благодаря виртуальному DOM React обеспечивает высокую производительность при обновлении интерфейса, что особенно важно для интерактивных веб-приложений с динамически изменяющимися данными. Также React хорошо интегрируется с различными состояниями управления, такими как Redux, Zustand или React Query, что делает управление данными внутри приложения более предсказуемым и удобным.

Еще одно важное преимущество React – его гибкость в выборе стратегии рендеринга. Можно разрабатывать как классическое SPA (Single Page Application), так и использовать серверный рендеринг (SSR) или статическую генерацию страниц (SSG). Это делает React подходящим для различных типов приложений, включая сложные корпоративные решения, админ-панели и маркетплейсы.

2.1.2 Для серверной части приложения использован ASP.NET Core благодаря его высокой производительности, кроссплатформенности и мощной экосистеме. Этот фреймворк является одним из самых быстрых веб-фреймворков, обеспечивая минимальные накладные расходы при обработке HTTP-запросов, благодаря встроенному веб-серверу Kestrel.

Одна из ключевых особенностей ASP.NET Core – его масштабируемость. Он отлично подходит как для монолитных приложений, так и для микросервисной архитектуры. Встроенная поддержка Dependency Injection упрощает управление зависимостями, что делает код более модульным и тестируемым.

Безопасность в ASP.NET Core реализована на высоком уровне. Фреймворк предоставляет встроенную защиту от XSS, CSRF и SQL-инъекций, а также мощные механизмы аутентификации и авторизации через Identity, JWT и OAuth. Это позволяет разрабатывать защищенные веб-приложения без необходимости встраивания дополнительных решений.

Еще один важный аспект – интеграция с облачными сервисами и контейнеризация. ASP.NET Core хорошо работает в Docker и Kubernetes, а также легко интегрируется с облачными платформами, такими как Azure и AWS. Это делает его отличным выбором для развертывания и масштабирования приложений.

2.1.3 В качестве основной базы данных выбрана PostgreSQL, так как она сочетает высокую производительность, надежность и богатый функционал. В отличие от MySQL, PostgreSQL поддерживает расширенные индексы, такие как GIN и BRIN, что позволяет значительно ускорять сложные запросы.

Поддержка JSONB делает PostgreSQL отличным выбором для хранения полу-структурированных данных, что позволяет комбинировать реляционную и NoSQL-модель в одной базе данных. Это особенно полезно, если в приложении необходимо хранить динамические структуры данных без жесткой схемы.

Кроме того, PostgreSQL обладает встроенной поддержкой репликации и горизонтального масштабирования, что позволяет эффективно работать с большими объемами данных. В сочетании с ACID-совместимостью и транзакционной целостностью это делает PostgreSQL оптимальным выбором для высоконагруженных систем.

2.1.4 Redis используется для ускорения работы приложения за счет кеширования данных в памяти. Это снижает нагрузку на основную базу данных, так как часто запрашиваемая информация извлекается из Redis, а не из PostgreSQL.

Благодаря своей структуре данных Redis может использоваться не только для кеширования, но и в качестве механизма Pub/Sub для организации обмена сообщениями между микросервисами. Это делает его удобным инструментом для реализации real-time функционала, например, чатов, уведомлений или мониторинга событий в приложении.

Еще одно важное применение Redis – хранение сессий пользователей. Это особенно полезно в масштабируемых приложениях, работающих на нескольких серверах, где стандартное хранение сессий в памяти веб-сервера может быть недостаточно надежным.

Выбранный стек технологий обеспечивает высокую производительность, надежность и удобство разработки. React позволяет быстро создавать динамичные интерфейсы, ASP.NET Core обеспечивает мощную серверную логику и безопасность, PostgreSQL гарантирует надежное хранение данных, а Redis ускоряет работу и уменьшает нагрузку на сервер.

2.2 Общая структура приложения

Так как приложение обладает большим количеством функционала, для упрощения его тестирования, масштабируемости и отладки следует построить определенную структуру: разделить проект на блоки, содержащие связанную между собой логику.

В этом подразделе будут проанализированы ключевые компоненты веб-приложения, их функции и способы взаимодействия. Четкое определение структуры системы играет важную роль в разработке, так как формирует основу для последующей реализации всех возможностей панели управления серверами.

Разработанная структурная схема представлена на чертеже ГУИР 400201.428 С1.

2.2.1 Блок регистрации

Блок регистрации отвечает за создание новых учетных записей и управление данными пользователей на этапе их первичной аутентификации. Он обеспечивает безопасный и удобный процесс регистрации, включая ввод пользовательских данных, их валидацию, проверку уникальности и сохранение в базе данных.

При регистрации пользователи указывают адрес электронной почты, имя пользователя и пароль. Для предотвращения создания дублирующих учетных записей реализована проверка уникальности email и имени пользователя. Кроме того, для повышения безопасности осуществляется валидация пароля, включая проверку его сложности (длина, наличие различных символов) и возможность использования менеджера паролей.

Чтобы защитить систему от злоумышленников, применяется подтверждение email. После ввода данных пользователю отправляется код или ссылка для активации учетной записи, что предотвращает регистрацию подложных аккаунтов. В случае незавершенной регистрации система может автоматически очищать неактивные учетные записи через заданный промежуток времени.

Все передаваемые данные шифруются, а пароли хранятся в защищенном виде с использованием алгоритма хеширования BCrypt. Это гарантирует, что даже при утечке базы данных злоумышленники не смогут получить доступ к паролям пользователей.

В случае ошибок пользователю предоставляются подробные уведомления, позволяющие корректировать введенные данные. Это улучшает пользовательский опыт и снижает вероятность возникновения проблем при регистрации.

Блок регистрации тесно интегрируется с системой авторизации и управления пользователями, обеспечивая надежную защиту учетных записей и удобство работы с платформой.

2.2.2 Блок аутентификации и авторизации

Блок авторизации и аутентификации отвечает за проверку учетных данных пользователей и управление их доступом к системе. Он обеспечивает безопасный вход в систему, обработку токенов доступа и контроль уровня привилегий.

Процесс аутентификации начинается с ввода логина email и пароля. Введенные данные сверяются с сохраненными в базе.

После успешной аутентификации система генерирует JWT access и refresh токены, которые передаются клиенту и используются для последующих запросов. Для повышения устойчивости ко взлому access токен создается на короткий срок, а по его истечению нужно провести повторную аутентификацию, что очень неудобно для пользователя. Чтобы решить

данную проблему используется refresh токен с гораздо большим сроком жизни. Из-за того, что он используется только для повторной аутентификации, перехватить его гораздо сложнее.

Авторизация управляет доступом к различным разделам системы, проверяя права пользователей и применяя ролевую модель (RBAC). Это позволяет разграничить доступ к функциям панели управления в зависимости от уровня привилегий (администратор, пользователь).

Таким образом, блок авторизации и аутентификации обеспечивает безопасное управление пользователями, защищая систему от несанкционированного доступа и повышая удобство работы.

2.2.3 Блок создания и удаления серверов

Данный блок отвечает за создание, настройку и удаление игровых серверов, поддерживая различные типы игр. Он обеспечивает удобный интерфейс для пользователей, позволяя быстро развернуть сервер с нужными параметрами и управлять его конфигурацией.

Перед созданием сервера пользователь задает первичные параметры, включая название, описание и (при необходимости) пароль для защиты доступа. После этого система предлагает изменить стандартные настройки, если пользователю требуется кастомизация. Эти настройки включают конфигурацию игрового мира, параметры администрирования и дополнительные опции, специфичные для конкретных игр.

Для большинства игр процесс создания сервера единообразен, позволяя выбрать конфигурацию мира и задать параметры администратора. Однако для Don't Starve Together предусмотрена дополнительная возможность установки модов. Пользователь может указать ссылки на моды из Steam Workshop и при необходимости загрузить индивидуальные конфигурации, если стандартные настройки мода его не устраивают.

После завершения процесса в базе данных создается запись о новом сервере, позволяя пользователю управлять несколькими серверами одновременно. Это дает гибкость в использовании платформы, позволяя запускать и администрировать несколько игровых сред с разными настройками.

Блок управления серверами играет ключевую роль в работе панели, обеспечивая гибкость, удобство и расширенные возможности конфигурирования для пользователей.

2.2.4 Блок запуска и взаимодействия с сервером

Данный блок отвечает за запуск, мониторинг и взаимодействие с игровыми серверами в реальном времени. Он управляет процессом выделения необходимых ресурсов, выбора свободных портов, создания Docker-контейнеров и обеспечением связи между пользователем и сервером.

При инициализации запуска система определяет свободный диапазон портов для нового экземпляра сервера, чтобы избежать конфликтов между разными процессами. Затем создается Docker-контейнер с соответствующими

параметрами, включая указанные пользователем настройки. После успешного запуска сервера в базе данных добавляется запись с информацией о нем, что позволяет пользователю управлять своим сервером через интерфейс панели.

Для взаимодействия пользователя с сервером в реальном времени используется технология SignalR, предоставляющая удобный механизм для двусторонней связи между клиентом и сервером через веб-сокеты.

При запуске сервера или обновлении страницы с консолью клиентское приложение устанавливает подключение к серверу SignalR. Он добавляет подключение в группу пользователей, идентифицируя его с помощью уникального connection ID.

Все команды, введенные пользователем в консоли, передаются через веб-сокеты в SignalR, а затем отправляются в соответствующий контейнер сервера. Ответы сервера (например, вывод консоли) передаются обратно через SignalR и отображаются в пользовательском интерфейсе в режиме реального времени, что позволяет пользователю следить за процессами, управлять настройками и вводить команды без необходимости перезапуска страницы.

Для эффективного управления подключениями используется локальный кэш, основанный на потокобезопасном словаре. Он содержит:

- connection ID клиента (уникальный идентификатор подключения к SignalR);
- session ID веб-сокета (идентификатор активной сессии);
- идентификатор контейнера в Docker;
- время последнего использования консоли.

Кэш автоматически очищается при его переполнении, а идентификаторы сбрасываются при выходе за пределы типа, хранящего их значения. Очистка выполняется на основе частоты использования консоли: если пользователь не взаимодействует с консолью длительное время, подключение разрывается, и ресурсы освобождаются. При повторном открытии консоли подключение автоматически возобновляется.

При удалении сервера выполняется полное очищение всех связанных данных:

- остановка контейнера в Docker и освобождение его портов;
- удаление записи о сервере из базы данных;
- очистка кэша подключений, связанных с данным сервером;
- удаление сервера из списка пользователя, что делает его недоступным в интерфейсе панели.

2.2.5 Блок настройки сервера

Блок настройки предоставляет пользователям возможность изменять конфигурацию сервера после его создания, адаптируя параметры как под игровой процесс, так и под административные задачи. Изменения могут касаться основных данных сервера, таких как название, описание и пароль, а также более специфичных настроек, зависящих от конкретной игры.

Пользователь может в любой момент:

- изменить название, описание и пароль сервера;

- настроить параметры доступа, например, ограничить вход на сервер определенными условиями;

- изменить режим работы сервера, если игра поддерживает разные типы игровых миров или сессий.

После внесения изменений система автоматически применяет их, и в зависимости от типа настроек сервер может потребовать перезапуска.

Для разных игр доступны специфические параметры, обеспечивающие гибкость настройки:

Некоторые настройки для Minecraft:

- трансляция логов сервера всем операторам, что позволяет администраторам отслеживать события в реальном времени;

- настройка проверки подлинности лицензии, позволяющая включать или отключать требование официальной учетной записи;

- изменение параметров генерации мира, включая сид, размер и дополнительные игровые механики.

Для Don't Starve Together:

1 Настройка и управление модами: можно добавлять, удалять и редактировать моды, даже если изначально они не были указаны при создании сервера.

2 Изменение конфигурации модов: некоторые моды позволяют гибко настраивать свои параметры, и эти настройки можно редактировать без необходимости переустановки.

3 Изменение параметров сложности и генерации мира, что влияет на геймплей и динамику выживания.

Для Terraria:

- включение/выключение встроенного античита, обеспечивающего защиту от нечестной игры;

- настройка PvP-режима позволяет включать или отключать возможность сражений между игроками;

- изменение уровня сложности мира, например, переключение между классическим, экспертным и мастер-режимом.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе рассматриваются ключевые классы приложения, их методы и взаимодействие, формирующие основу работы панели управления.

Учитывая то, что приложение разрабатывается с использованием ASP.NET Core и Restful API и разделением кода по принципам чистой архитектуры, классы можно разделить на следующие группы:

- классы работы с базой данных;
- интерфейсы сервисов;
- data transfer objects;
- классы, содержащие общие данные;
- контроллеры;
- сервисы;
- классы бизнес-логики;
- классы инициализации.

В следующих подразделах подробно рассматриваются эти группы, их структура и методы и принципы взаимодействия в рамках архитектуры приложения.

3.1 Работа с базой данных

3.1.1 `PanelDbContext`

`PanelDbContext` представляет собой контекст базы данных, который наследуется от класса `DbContext`. Он управляет доступом к базе данных и предоставляет наборы данных для работы с моделями.

Содержит в себе следующие свойства:

- `Users` – коллекция пользователей, представленная объектом `DbSet<User>`, позволяет выполнять операции чтения и записи данных пользователей в базу данных;
- `Tokens` – коллекция токенов обновления (`refresh tokens`), представлена объектом `DbSet<RefreshToken>`, хранит и управляет данными о токенах(время действия, сам токен и какому пользователю он принадлежит);
- `RunningServers` – коллекция запущенных серверов, представлена объектом `DbSet<RunningServer>`, отвечает за хранение информации о запущенных игровых серверах;

Класс имеет конструктор `PanelDbContext (DbContextOptions<PanelDbContext> options)`, который принимает параметры конфигурации для базы данных и передает их в базовый класс `DbContext`. В частности, нужен для передачи строки подключения.

Также в конструкторе вызывается метод `Database.EnsureCreated()`, который проверяет существование базы данных и создает её при необходимости, если она отсутствует.

3.1.2 Repository<T>

Этот класс представляет собой универсальный репозиторий, реализующий интерфейс IRepository<T> для управления сущностями, наследуемыми от AbstractEntity. Он использует DbContext для взаимодействия с базой данных и поддерживает основные CRUD-операции.

Класс содержит свойство Entities, предоставляющее доступ к данным сущности T в виде IQueryable<T>, а также несколько методов:

- AddAsync(T entity) – выполняет асинхронное добавление новой сущности в базу данных, с возможностью кэширования данных в Redis;
- DeleteAsync(T entity) – удаляет сущность из базы данных, предусматривая очистку соответствующего кэша;
- GetByIdAsync(int id) – осуществляет поиск сущности по идентификатору с возможностью предварительного получения данных из Redis;
- UpdateAsync(T entity) – обновляет данные сущности в базе, с возможностью синхронизации изменений в Redis.

3.1.3 UnitOfWork

Этот класс реализует шаблон Unit of Work, инкапсулируя работу с базой данных через PanelDbContext и управляя репозиториями для различных сущностей.

В классе хранится:

- _context – экземпляр PanelDbContext, отвечающий за взаимодействие с базой данных;
- _repositories – коллекция, использующая Hashtable для хранения репозитория, чтобы исключить их повторное создание;
- _cache – механизм кэширования через IDistributedCache.

Ключевые методы:

- Repository<T>() – возвращает репозиторий для указанной сущности T, создавая его при необходимости;
- Save() – сохраняет изменения в базе данных;
- Dispose() – освобождает ресурсы контекста базы данных.

Этот класс обеспечивает централизованное управление репозиториями и транзакциями, упрощая работу с базой данных.

3.1.4 AbstractEntity

Данный класс содержит свойство Id. Данный класс используется в качестве ограничения входного параметра Repository и для исключения дублирования данного поля в каждой модели.

3.1.5 User

Этот класс представляет сущность пользователя и наследуется от AbstractEntity.

Он хранит основные данные учетной записи, а также связан с серверами и токенами обновления.

Основные свойства:

- Email – адрес электронной почты пользователя;
- Password – хешированный пароль;
- PasswordSalt – соль для хеширования пароля;
- FtpPassword – пароль для доступа по FTP;
- Role – роль пользователя в системе, по умолчанию "User";
- MinecraftServerExecutable – путь к исполняемому файлу сервера Minecraft;
- DSTServerExecutable – путь к исполняемому файлу сервера Don't Starve Together;
- Ts – временная метка последнего обновления данных пользователя;
- RefreshTokens – список связанных токенов обновления;
- RunningServers – список серверов, принадлежащих пользователю.

Данный класс используется для хранения и управления учетными записями пользователей в системе.

3.1.6 RunningServer

Этот класс представляет сущность запущенного игрового сервера и наследуется от `AbstractEntity`. Он содержит информацию о запущенном экземпляре сервера, привязанном к конкретному пользователю.

Основные свойства:

- UserId – идентификатор пользователя, которому принадлежит сервер.
- ContainerName – имя Docker-контейнера, в котором запущен сервер.
- Port – номер порта, используемого сервером.
- ServerType – тип игрового сервера, представленный в виде перечисления `ServerTypes`.
- ConnectionId – уникальный идентификатор веб-сокета соединения, по умолчанию 0.

Этот класс используется для отслеживания активных игровых серверов, их параметров и связей с пользователями.

3.1.7 RefreshToken

Этот класс представляет сущность токена обновления и наследуется от `AbstractEntity`. Он используется для управления механизмом обновления JWT-токенов, предоставляя возможность пользователям оставаться в системе без повторного ввода учетных данных.

Основные свойства:

- UserId – идентификатор пользователя, которому принадлежит токен.
- TokenHash – хеш токена обновления, обеспечивающий его защиту.

- TokenSalt – соль для хеширования токена, повышающая его криптостойкость.

- IssueDate – дата и время выдачи токена.

- ExpiryDate – дата и время истечения срока действия токена.

Этот класс используется для безопасного хранения и проверки токенов обновления, что позволяет пользователям повторно аутентифицироваться без необходимости повторного ввода пароля.

3.2 Интерфейсы сервисов

Данная группа классов представляет собой абстракции для сервисов, что позволяет отделить их реализацию от бизнес-логики приложения. Такой подход соответствует принципам чистой архитектуры, обеспечивая гибкость, тестируемость и слабую связанность между компонентами.

В чистой архитектуре ключевая идея заключается в разделении кода на слои, где высокоуровневые модули (например, бизнес-логика) не зависят от низкоуровневых деталей (конкретных реализаций сервисов, баз данных и инфраструктуры). Вместо этого взаимодействие между слоями осуществляется через абстракции (интерфейсы), что позволяет заменять конкретные реализации без влияния на основную логику приложения.

Благодаря внедрению зависимостей (dependency injection) сервисы передаются в зависимости от контекста исполнения, а не жестко фиксируются в коде. Это упрощает модульное тестирование (можно подменять реальные сервисы заглушками), а также облегчает поддержку и расширение системы.

Методы этих абстракций полностью повторяют методы одноименных классов, их реализующих, поэтому их дублирование в описании нецелесообразно. Вместо этого акцент делается на принципах проектирования, позволяющих легко заменять реализацию конкретного сервиса (например, хранилища данных или механизма аутентификации) без нарушения работы других компонентов.

3.3 Data transfer objects

Data transfer object (DTO) – это объект, предназначенный для передачи данных между различными слоями приложения, чаще всего между бэкендом и фронтендом или между разными сервисами. DTO используется для изоляции внутренних структур данных от внешних слоев приложения, упрощения сериализации и уменьшения объема передаваемых данных.

Основная цель использования DTO – это разделение моделей домена и представления данных, а также оптимизация передачи информации. Прямое использование доменных моделей (например, сущностей базы данных) в API или между слоями системы может привести к:

- лишним данным в ответах – передаются все свойства модели, даже если они не нужны;

- уязвимостям – например, возможность изменения полей, которые должны оставаться неизменными;

- сильной связности между слоями – изменение модели базы данных может затронуть все слои приложения.

DTO помогает избежать этих проблем, предоставляя упрощенные структуры данных, содержащие только нужные поля.

Вместе с использованием библиотеки MediatR и ее интерфейса `IRequest`, DTO помогает улучшить архитектуру приложения и упростить взаимодействие между слоями. MediatR реализует паттерн посредник, который позволяет обрабатывать запросы, инкапсулируя логику запроса в одном месте и разделяя компоненты, взаимодействующие с данным запросом.

`IRequest` является интерфейсом, который используется для определения запроса в системе с помощью MediatR. Запрос включает DTO, которые будут переданы через медиатор и обработаны соответствующими обработчиками. Преимущества такого подхода:

- изоляция логики: запросы и обработчики запросов изолируют бизнес-логику от других частей системы, таких как контроллеры или сервисы. DTO используется как объект данных, который передается через MediatR;

- упрощение взаимодействия: взаимодействие между различными слоями или сервисами осуществляется через медиатор, что позволяет избежать жесткой связи между компонентами. DTO передает только необходимые данные, исключая ненужную информацию;

- оптимизация обработки данных: вместо передачи полной сущности доменной модели, DTO передает только нужные поля, что позволяет уменьшить объем данных и повысить производительность, особенно при сложных запросах.

3.3.1 Login

Класс `Login` используется для передачи данных от клиента к серверу при попытке входа в систему и реализует интерфейс `IRequest`. Он содержит два свойства:

- `Email` – адрес электронной почты пользователя, используемый для идентификации;

- `Password` – пароль пользователя, введенный для аутентификации.

3.3.2 SignUp

Класс `SignUp` реализует интерфейс `IRequest<ActionResult>` и представляет собой запрос, используемый для регистрации нового пользователя в системе. Он содержит все необходимые данные для создания учетной записи и назначения начальных параметров пользователя:

- `Email` – строка, представляющая адрес электронной почты пользователя, который будет использоваться для регистрации.

- `Password` – строка, содержащая пароль пользователя для аутентификации.

- `ConfirmPassword` – строка, предназначенная для подтверждения пароля и проверки, что оба введенных пароля совпадают.
- `Ts` – дата и время, когда был инициирован процесс регистрации, что может быть полезно для аудита или отслеживания времени создания аккаунта.
- `Role` – строка, определяющая роль пользователя в системе (например, «User», «Admin»), которая может влиять на права доступа и функционал в приложении.

3.3.3 Tokens

Класс `Tokens` реализует интерфейс `IRequest<IActionResult>` и представляет собой запрос, содержащий данные для передачи токенов доступа и обновления. Этот класс используется для возврата информации о токенах, которые необходимы для аутентификации и авторизации пользователя в системе. Его свойства:

- `AccessToken` – строка, представляющая токен доступа, который используется для авторизации пользователя и предоставления ему доступа к защищенным ресурсам;
- `RefreshToken` – строка, представляющая токен обновления, который используется для получения нового токена доступа без необходимости повторной аутентификации пользователя.\

3.3.4 CreateDSTServerRequest

Класс `CreateDSTServerRequest` реализует интерфейс `IRequest<IActionResult>` и представляет собой запрос, предназначенный для создания нового сервера Don't Starve Together (DST). Этот класс содержит все необходимые данные для настройки и инициализации нового сервера в системе. Они представлены в качестве следующих свойств:

- `Id` – уникальный идентификатор пользователя, отправившего запрос;
- `ServerName` – строка, представляющая название сервера. Это имя будет отображаться в системе и может быть использовано пользователями для идентификации сервера;
- `ServerDescription` – строка, описывающая сервер. Это поле предоставляет дополнительную информацию о сервере, например, его особенности или цели;
- `ServerPassword` – строка, представляющая пароль для доступа к серверу, если сервер требует аутентификации для подключения.

Также для всех этих параметров имеется конструктор `CreateDSTServerRequest(int id, string serverName, string serverDescription, string serverPassword)`.

3.3.5 CreateMinecraftServerRequest

Класс `CreateMinecraftServerRequest` реализует интерфейс `IRequest<IActionResult>` и предназначен для запроса, который

инициализирует создание нового сервера Minecraft. Он содержит необходимые данные для создания и настройки игрового сервера в системе.

Свойства класса:

- `Id` – уникальный идентификатор пользователя, отправившего запрос;
- `ServerExecutableName` – строка, представляющая имя исполнимого файла для запуска сервера Minecraft. Это имя указывает на конкретный исполняемый файл, который будет использован для инициализации сервера.

Также содержит конструктор `CreateMinecraftServerRequest(string serverExecutableName, int id)`.

3.3.6 GetContentRequest

Класс `GetContentRequest` реализует интерфейс `IRequest <IActionResult>` и предназначен для запроса получения контента, связанного с конкретным пользователем и файлом. Этот класс содержит информацию, необходимую для извлечения данных о содержимом файла из файловой системы. Свойства:

- `UserId` – целое число, представляющее уникальный идентификатор пользователя, который запрашивает контент;
- `Path` – строка, представляющая путь до файла в файловой системе, который требуется получить. Путь указывает точное местоположение файла или директории в системе.

Конструктор: `GetContentRequest(int userId, string path)`.

3.3.7 GetServerSettingsRequest

Класс `GetServerSettingsRequest` реализует интерфейс `IRequest <IActionResult>` и предназначен для запроса получения настроек сервера для конкретного пользователя и типа сервера. Этот класс содержит информацию, необходимую для получения данных конфигурации сервера в системе. Свойства:

- `UserId` – целое число, представляющее уникальный идентификатор пользователя, для которого запрашиваются настройки сервера;
- `ServerType` – перечисление `ServerTypes`, которое указывает тип сервера, настройки которого должны быть получены. Это позволяет системе обрабатывать запросы для различных типов серверов.

Конструктор: `GetServerSettingsRequest(int userId, ServerTypes type)`.

3.3.8 UpdateFileRequest

Класс `UpdateFileRequest` реализует интерфейс `IRequest <IActionResult>` и предназначен для запроса на обновление содержимого

файла. Этот класс используется для передачи информации о файле, который требуется обновить, а также для передачи нового контента.

- `Id` – целое число, представляющее уникальный идентификатор пользователя, отправившего запрос;

- `Path` – строка, представляющая путь к файлу, который необходимо обновить в файловой системе.

- `Content` – массив строк, содержащий новое содержимое файла, которое должно быть записано в указанный файл.

Конструктор: `UpdateFileRequest(int id, string path, string[] content)`.

3.4 Классы общих данных

Данный раздел посвящен классам, которые нельзя явно отнести к какому-то определенной группе в виду того, что они содержат разнообразные данные

3.4.1 Token

Класс `Token` представляет собой объект для передачи данных между слоем приложения и слоем инфраструктуры чистой архитектуры системы для передачи информации о пользователе и его токене доступа. Этот класс используется для упрощения и стандартизации обмена данными, связанными с авторизацией.

- `UserId` – целое число, представляющее уникальный идентификатор пользователя. Это значение используется для привязки токена к конкретному пользователю;

- `JwtToken` – строка, содержащая JWT (JSON Web Token) токен. В зависимости от контекста использования класса, может содержать и токен доступа, и токен обновления.

3.4.2 ServerTypes

Перечисление `ServerTypes` используется для обозначения различных типов серверов, которые могут быть развернуты в приложении. Оно предоставляет набор предустановленных значений, которые служат для классификации серверов в зависимости от их назначения или типа игры.

1 `Minecraft`: представляет тип сервера для игры `Minecraft`. Это может быть как обычный `Minecraft` сервер, так и сервер с настроенными модами или специфическими конфигурациями.

2 `DstMaster`: представляет тип сервера для игры `Don't Starve Together` (DST) в режиме мастер-сервера. Этот сервер управляет игровым миром и может соединять игроков в одну игровую сессию.

3 `DstCaves`: представляет тип сервера для пещер в `Don't Starve Together`. Это отдельный тип сервера, где игроки могут исследовать подземелья, добавленные в игру.

Это перечисление используется для различения серверов по их типу и назначению, что позволяет программно определять, какие конфигурации или параметры применимы к каждому типу сервера. Например, настройки и операции для Minecraft сервера будут отличаться от настроек и операций для серверов DST.

3.4.3 ManageMode

Перечисление ManageMode используется для определения действия, которое должно быть выполнено с FTP пользователем. Это перечисление предоставляет два возможных состояния, которые помогают управлять пользователями FTP в системе.

1 `Create` указывает, что необходимо создать нового FTP пользователя. Это действие подразумевает добавление нового пользователя с необходимыми правами и настройками в системе.

2 `Delete` указывает, что необходимо удалить существующего FTP пользователя. Это действие подразумевает удаление учетной записи пользователя, включая его доступ и настройки, из системы.

3.4.4 BaseResponse

Класс BaseResponse представляет собой базовую структуру для ответа на запрос в приложении. Он используется для передачи информации о результате выполнения операции, например, успешности операции и сообщения, связанного с результатом. Имеет следующие поля:

1 `isSuccess` – булевый параметр, указывающий на успешность выполнения запроса. По умолчанию установлено значение `true`, что означает успешное выполнение операции. Если операция завершена с ошибкой, это поле может быть установлено в `false`.

2 `message` – строка, содержащая сообщение, которое поясняет результат запроса. Это сообщение может содержать дополнительную информацию, например, описание ошибки или успешного выполнения.

Конструкторы:

1 Первый конструктор принимает одно сообщение и устанавливает значение `isSuccess` по умолчанию как `true`. Это может быть полезно для случаев, когда операция прошла успешно, и требуется только передать сообщение.

2 Второй конструктор позволяет задать как успешность выполнения (`isSuccess`), так и сообщение. Это дает больше гибкости, например, в случае ошибки, когда нужно передать конкретное сообщение о причине неудачи.

3.4.5 ConnectionInfo

`ConnectionInfo` – это структура, которая представляет информацию о подключении клиента к серверу. Этот класс используется для хранения данных, связанных с конкретным клиентским подключением, включая

информацию о процессе, привязанном к контейнеру Docker, а также временные метки.

Свойства:

- `HubConnectionId` – уникальный идентификатор соединения клиента в SignalR, используемый для связи клиента с сервером;
- `AttachProcess` – процесс, связанный с подключением, используется для выполнения команд в контейнере Docker;
- `LastUsingTime` – время последнего использования соединения. Это поле помогает отслеживать, когда соединение было использовано в последний раз;
- `ContainerName` – имя контейнера, с которым связано подключение. Это важно для идентификации серверов и выполнения команд внутри конкретных контейнеров;
- `Handler` – обработчик событий для получения данных о процессе (например, логов, вывода ошибок).

3.5 Контроллеры

Контроллеры в ASP.NET Core REST-приложениях служат основным звеном между клиентом и сервером, обрабатывая входящие HTTP-запросы и возвращая соответствующие HTTP-ответы. Они обеспечивают маршрутизацию, связывая запросы с конкретными методами, которые выполняют бизнес-логику и взаимодействуют с сервисами и базой данных.

Одной из ключевых задач контроллеров является обработка входных данных. Они принимают параметры запроса, данные из тела запроса или заголовков и выполняют их валидацию. Это позволяет защитить приложение от некорректных данных, обеспечивая их соответствие ожидаемым форматам.

Контроллеры также управляют формированием ответов. Они возвращают данные в виде JSON или другого формата, соответствующего требованиям API. В зависимости от результата выполнения операции контроллер может отправить успешный ответ с данными или ошибку с соответствующим HTTP-статусом.

Еще одной важной функцией контроллеров является взаимодействие с сервисами и слоями бизнес-логики. Контроллеры не должны содержать сложную логику – вместо этого они вызывают соответствующие сервисы, которые выполняют основную работу, например, обработку данных, обращение к базе или взаимодействие с внешними API. Это помогает разделить ответственность между слоями приложения, улучшая его поддержку и тестируемость.

Контроллеры также поддерживают различные механизмы безопасности, такие как аутентификация и авторизация. Они могут проверять права доступа пользователя перед выполнением запроса, запрещая выполнение операций, если у пользователя недостаточно прав. Это обеспечивает защиту API и предотвращает несанкционированный доступ.

Кроме того, контроллеры могут управлять кэшированием и оптимизацией запросов, например, используя кэширование ответов для уменьшения нагрузки на сервер. Это особенно важно для REST API, где может быть большое количество однотипных запросов.

Таким образом, контроллеры в ASP.NET Core REST-приложениях играют центральную роль в обработке запросов, управлении входными и выходными данными, обеспечении безопасности и организации взаимодействия между клиентом и сервером.

3.5.1 BaseController

`BaseController` является базовым контроллером, который наследуется от `ControllerBase`. Он предназначен для предоставления общих механизмов аутентификации и авторизации, используемых в производных контроллерах, упрощая доступ к данным пользователя, который выполняет запрос. Свойства:

1 `UserId` – свойство, которое получает идентификатор пользователя из `NameIdentifier` в `ClaimsIdentity`. Используется для определения текущего пользователя, выполняющего запрос.

2 `Role` – свойство, извлекающее роль пользователя из `Role`. Позволяет контроллерам управлять доступом к определенным ресурсам на основе ролей.

Единственный метод `FindClaim(string claimName)` получает значение клейма пользователя по его названию. Он извлекает `ClaimsIdentity` из `HttpContext.User.Identity`, находит в нем запрашиваемый клейм и возвращает его значение. Если клейм отсутствует, возвращает `null`.

Этот контроллер служит основой для других контроллеров, предоставляя удобный доступ к данным аутентифицированного пользователя, что помогает сократить дублирование кода и улучшить читаемость контроллеров, работающих с идентификацией и авторизацией.

3.5.2 AuthenticationController

`AuthenticationController` является контроллером, отвечающим за обработку запросов, связанных с аутентификацией и управлением учетными записями пользователей. Он наследуется от `BaseController`, что позволяет использовать данные текущего пользователя, такие как `UserId` и `Role`.

Контроллер помечен атрибутами `[Route("api/[controller]")]` и `[ApiController]`, что определяет маршрут к нему (`api/authentication`) и включает встроенные проверки модели, упрощая обработку запросов.

Поле `_mediator` хранит экземпляр `IMediator`, который используется для отправки запросов обработчикам без непосредственного

вызова сервисов. Это снижает связанность контроллера с конкретными реализациями бизнес-логики.

Конструктор `AuthenticationController(IMediator mediator)` принимает объект `IMediator` через внедрение зависимостей (`Dependency Injection`). Он инициализирует `_mediator`, позволяя использовать его во всех методах контроллера.

Методы:

– `Login(Login request) (HttpPost("Login"))` – получает запрос с учетными данными (`Login`), передает его в `IMediator` для обработки и возвращает результат;

– `SignUp(SignUp request) (HttpPost("SignUp"))` – аналогично, обрабатывает запрос на регистрацию (`SignUp`) и отправляет его в `IMediator`;

– `GetNewAccessToken(GetAccessToken request) (HttpPost("Reauthorize"))` – запрашивает новый токен доступа (`GetAccessToken`) через `IMediator`;

– `Logout() (HttpPost("Logout"))` – отправляет команду выхода из системы (`LogoutRequest`), передавая `UserId` текущего пользователя;

– `Verify() (HttpGet("Verify"))` – проверяет, аутентифицирован ли пользователь. Если да, возвращает `true`, если он администратор, иначе `false`. В противном случае возвращает `Unauthorized()`.

Этот контроллер служит точкой входа для всех операций, связанных с аутентификацией, и обеспечивает их обработку с использованием паттерна CQRS через `IMediator`.

3.5.3 ServerSelectionController

`ServerSelectionController` – это контроллер, отвечающий за управление игровыми серверами. Он наследуется от `BaseController`, что позволяет использовать информацию о текущем пользователе, например `UserId`.

Контроллер помечен атрибутами `[Route("api/[controller]/[action]")]` и `[ApiController, Authorize]`, что указывает на необходимость аутентификации и определяет маршруты методов.

Поля:

– `_config` – объект `IConfiguration`, используемый для работы с конфигурацией приложения;

– `_mediator` – объект `IMediator`, через который контроллер отправляет команды в соответствующие обработчики.

Конструктор `ServerSelectionController(IConfiguration config, IMediator mediator)` инициализирует `_config` и `_mediator`, используя внедрение зависимостей.

Методы:

- `CreateMinecraftServer(string serverVersion)` (`HttpPut`) – создает сервер Minecraft, передавая указанную версию и `UserId`;
- `CreateDSTServer(CreateDSTServerRequest request)` (`HttpPut`) – создает сервер Don't Starve Together, используя параметры из `CreateDSTServerRequest`;
- `DeleteMinecraftServer()` (`HttpDelete`) – удаляет сервер Minecraft текущего пользователя;
- `DeleteDSTServer()` (`HttpDelete`) – удаляет сервер Don't Starve Together текущего пользователя;
- `StartMinecraftServer()` (`HttpGet`) – запускает сервер Minecraft пользователя;
- `StartDSTServer()` (`HttpGet`) – запускает сервер Don't Starve Together пользователя;
- `GetServerStatus()` (`HttpGet`) – получает статус запущенного сервера;
- `GetSettings(ServerTypes type)` (`HttpGet`) – получает настройки сервера указанного типа (Minecraft, DstMaster или DstCaves);
- `UpdateFile(UpdateFileRequest request)` (`HttpPatch`) – обновляет содержимое файла на сервере пользователя;
- `GetFolderContent(string path)` (`HttpGet`) – получает список файлов и папок по указанному пути в файловой системе сервера.

Этот контроллер предоставляет API для управления серверами, обеспечивая их создание, удаление, запуск, обновление файлов и получение информации о статусе. Все операции выполняются через `IMediator`, что снижает связанность контроллера с бизнес-логикой.

3.5.4 ConsoleHub

Данный хаб `SignalR` напрямую нельзя отнести к контроллерам, однако для веб-сокетов он выполняет по сути ту же функцию, что и контроллеры для `http`-запросов. Отличается он лишь тем, что запросы и ответы приходят в реальном времени, поэтому было решено отнести его к этой группе.

`ConsoleHub` – это класс, реализующий логику для взаимодействия с клиентами через `SignalR`. Он отвечает за подключение, управление командной строкой серверов, выполнение команд и отправку логов от сервера к клиенту.

Поля:

1 `_connections` – статическое поле, представляющее кэш подключений клиентов. Оно хранит информацию о текущих соединениях, включая идентификаторы и процессы, связанные с ними, а также обработчики, использующиеся для отправки данных из процесса клиенту.

2 `_unitOfWork` – объект для работы с репозиториями и сохранением изменений в базе данных через паттерн `Unit of Work`.

3 `_processManager` – интерфейс для взаимодействия с операционной системой, в частности для управления процессами, например, для запуска команд.

4 `_logger` – объект для логирования, используется для записи информации и ошибок, связанных с подключением и выполнением команд.

Методы:

1 `OnConnectedAsync()`. Этот метод вызывается, когда клиент успешно подключается. Он проверяет наличие сервера с указанным контейнером в базе данных. Если сервер найден, он либо создаёт новый процесс для подключения к контейнеру через команду `docker attach`, либо использует существующий процесс, если такой уже есть. Логи отправляются клиенту, и информация об успешном подключении передается обратно клиенту. После завершения этих действий обновляется запись о сервере в базе данных.

2 `SendLogsToClient(uint id, IClientProxy client)`. Этот метод передает логи от контейнера (через вывод стандартных данных процесса) в клиентский интерфейс. Он привязывает обработчик события `OutputDataReceived` к процессу, чтобы передавать логи пользователю через `SignalR`.

3 `ReceiveCommandFromClient(uint connectionId, string command)`. Метод для получения команды от клиента. Если команда не пуста, она отправляется в процесс контейнера через стандартный ввод. Если команда – `stop`, то контейнер останавливается, и подключение от клиента разрывается.

4 `OnDisconnectedAsync(Exception exception)`. Этот метод вызывается, когда клиент отключается. Он проверяет, был ли пользователь активен в течение последних 10 минут. Если нет, удаляет информацию о соединении из кэша и обновляет запись в базе данных, сбрасывая `Id` подключения, для оптимизации использования вычислительных ресурсов сервера. Если клиент был активен, пересоздается только подключение к хабу `SignalR` в кэше.

5 `WaitForContainerStopAsync(string containerName, int timeoutSeconds = 10)`. Этот метод следит за состоянием контейнера и ожидает его остановки в течение заданного времени. Он периодически проверяет статус контейнера с помощью команды `docker inspect`. Если контейнер не останавливается в пределах указанного времени, выполняется принудительная остановка через команду `docker kill`.

3.6 Сервисы

В контексте чистой архитектуры сервисы представляют собой слой бизнес-логики, который инкапсулирует ключевые процессы и правила работы приложения. Они находятся между контроллерами и репозиториями, обеспечивая выполнение операций, независимых от инфраструктуры (например, базы данных или API).

Сервисы представляют собой классы, реализующие интерфейсы, что делает их удобными для внедрения зависимостей (Dependency Injection).

3.6.1 AuthenticationService

Этот класс представляет собой сервис, отвечающий за аутентификацию и управление пользователями.

Он реализует интерфейс `IAuthenticationService` и используется для обработки запросов, связанных с входом, выходом и регистрацией пользователей.

Сервис взаимодействует с тремя зависимостями:

- `ITokenService` – сервис генерации токенов, используемый для создания JWT-токенов при успешной аутентификации.
- `IUnitOfWork` – паттерн Unit of Work, позволяющий управлять транзакциями и работой с репозиториями.
- `IFtpManager` – сервис для управления FTP-пользователями, который создает FTP-аккаунт при регистрации.

Методы:

1 `LoginAsync(Login loginRequest)` – проверяет существование пользователя, валидирует пароль и, если данные верны, создает JWT-токены, возвращая их клиенту.

2 `LogoutAsync(int userId)` – удаляет refresh-токен пользователя, чтобы предотвратить дальнейшее обновление access-токена.

3 `SignUpAsync(SignUp signupRequest)` – проверяет, существует ли пользователь с таким email, сравнивает пароли, хеширует их и создает новую запись пользователя. При этом также создается FTP-аккаунт с уникальным паролем.

Все методы возвращают `ActionResult`, что позволяет легко интегрировать сервис с API-контроллерами.

3.6.2 TokenService

Класс `TokenService` отвечает за генерацию и управление токенами аутентификации и обновления (JWT-токенами и refresh-токенами) в системе. Он реализует интерфейс `ITokenService` и использует базу данных `PanelDbContext` для работы с сущностями пользователей и refresh-токенами.

Методы:

1 `GenerateAccessTokenAsync(int userId)` генерирует access-token для пользователя с данным `userId`. Он проверяет, существует ли пользователь в базе данных, и если да, то создает новый токен с использованием `TokenBuilder`. Если пользователь не найден, возвращает пустую строку.

2 `GenerateTokensAsync(int userId)` генерирует access-token и refresh-token для пользователя. Включает хеширование refresh-токена и сохранение его в базе данных для пользователя. Если у пользователя уже есть

существующие refresh-токены, они удаляются перед добавлением нового. Возвращает объект Tokens, который содержит оба токена.

3 RemoveRefreshTokenAsync(User user) удаляет refresh-token для указанного пользователя из базы данных. Этот метод используется при выходе пользователя или обновлении токенов.

4 ValidateRefreshTokenAsync(Token refreshTokenRequest) валидирует refresh-token, проверяя его соответствие с хешированным значением в базе данных, а также проверку срока действия токена. Если токен истек или неверен, возвращает сообщение с ошибкой. В случае успешной валидации возвращает userId для дальнейшего использования.

Зависимости:

- PanelDbContext _db – контекст базы данных, используется для работы с таблицами пользователей и refresh-токенов;

- TokenBuilder – класс, отвечающий за создание токенов (как access, так и refresh);

- PasswordBuilder – класс, используемый для хеширования паролей и токенов.

Этот сервис предоставляет функционал для обработки токенов, который широко используется для аутентификации пользователей в приложении, особенно для работы с сессиями, а также для управления сроком действия и безопасностью токенов.

3.6.3 PasswordBuilder

Класс PasswordBuilder используется для создания и хеширования паролей с применением криптографических алгоритмов. Он предоставляет два статических метода, которые позволяют безопасно генерировать соль и хешировать пароли с помощью алгоритма PBKDF2 (Password-Based Key Derivation Function 2).

Методы:

1 GetSecureSalt(). Этот метод генерирует случайную соль для хеширования пароля. Соль представляет собой случайный набор байтов, получаемый с использованием криптографически безопасного генератора случайных чисел (RandomNumberGenerator.GetBytes). Возвращается массив байтов длиной 32 байта, который может быть использован для хеширования пароля с повышенной безопасностью.

2 HashUsingPbkdf2(string password, byte[] salt). Этот метод выполняет хеширование пароля с использованием алгоритма PBKDF2, который является криптографически стойким методом для извлечения ключа из пароля. В качестве входных данных используется сам пароль и соль, переданная в метод. Используется хеш-функция HMACSHA256 с 100000 итерациями, что делает процесс хеширования более защищенным от атак. Результатом является строка в формате Base64, представляющая собой хешированное значение пароля.

Этот класс является ключевым элементом системы для защиты паролей пользователей, так как позволяет безопасно генерировать соль и хешировать пароли с минимальными шансами на их компрометацию.

3.6.4 TokenBuilder

Класс `TokenBuilder` предоставляет функциональность для генерации токенов JWT, которые используются для аутентификации и авторизации пользователей в системе.

Константы:

1 `Issuer` – строка, определяющая издателя токена. В данном случае это "LibraryAPI", что означает, что токены выпускаются системой с таким именем.

2 `Audience` – строка, определяющая аудиторию токена. Здесь указано "LibraryUser", что предполагает, что токен предназначен для пользователей системы.

Методы:

1 `GenerateAccessToken(int userId, string role)` генерирует JWT, который используется для аутентификации пользователя. Токен содержит информацию о пользователе (например, его ID и роль) в виде клеймов (claims).

В процессе создания токена:

1.1 `Claims` – создается объект `ClaimsIdentity`, который включает два клейма: `NameIdentifier` (ID пользователя) и `Role` (роль пользователя).

1.2 `SigningCredentials` – определяются учетные данные для подписи токена. Для этого используется симметричный ключ (секрет), указанный в передаваемой константе при запуске приложения `Secret`, и алгоритм подписи `HMACSHA256`.

1.3 `TokenDescriptor` – создается объект `SecurityTokenDescriptor`, который содержит информацию о токене, такую как издатель, аудитория, время истечения срока действия (15 минут) и учетные данные для подписи.

1.4 `Token` – с использованием `JwtSecurityTokenHandler` создается сам токен, который затем сериализуется в строку и возвращается.

2 `GenerateRefreshToken()` генерирует рефреш-токен. Рефреш-токен используется для получения нового access-токена после того, как текущий истек.

В процессе генерации рефреш-токена:

2.1 Генерируется массив случайных байтов длиной 32 байта с использованием криптографически безопасного генератора случайных чисел.

2.2 Эти байты кодируются в строку Base64 и возвращаются как результат.

Зависимости:

- `JwtSecurityTokenHandler` – класс для создания и валидации JWT;
- `SecurityTokenDescriptor` – объект для описания параметров токена;
- `ClaimsIdentity` – объект, представляющий информацию о пользователе в виде клеймов;
- `SigningCredentials` – объект, содержащий информацию для подписи токена;
- `SecurityAlgorithms.HmacSha256Signature` – алгоритм подписи для HMAC-SHA256;
- `RandomNumberGenerator` – криптографически безопасный генератор случайных чисел, используемый для генерации рефреш-токенов.

Этот класс является важным элементом для реализации механизма аутентификации в приложении с использованием JWT и рефреш-токенов, обеспечивая безопасность и удобство работы с токенами.

3.6.5 `OsInteractionService`

`OsInteractionsService` реализует интерфейс `IOsInteractionsService` и предоставляет функциональность для взаимодействия с операционной системой, в частности для выполнения команд в оболочке и поиска свободных портов.

Поля:

1 `_shellName` – строка, указывающая имя оболочки, которая будет использоваться для выполнения команд. В данном случае это `/bin/bash`.

2 `_baseArguments` – строка, представляющая базовые аргументы, которые будут добавляться к каждой команде, передаваемой в оболочку. В данном случае это параметр `-c`, который позволяет передавать команду для выполнения.

Методы:

1 `ExecuteCommand(string command, string workingDirectory = "")` синхронно выполняет команду в операционной системе через оболочку (например, `/bin/bash`). Команда передается как строка, а опционально можно указать рабочую директорию, в которой будет выполняться команда. Метод возвращает строку с результатом выполнения команды. Процесс выполняется с использованием метода `CreateCmdProcess`, и после завершения процесса, результат читается из стандартного вывода.

2 `ExecuteCommandAsync(string command, string workingDirectory = "")` метод аналогичен методу `ExecuteCommand`, но выполняется асинхронно, что позволяет не блокировать выполнение других операций в приложении. Он возвращает строку с результатом выполнения команды. Используется асинхронный метод `WaitForExitAsync` для ожидания завершения процесса.

3 `CreateCmdProcess(string cmdArguments, string directory = "")` создает и конфигурирует объект `Process` для выполнения команд в оболочке. Процесс будет выполняться с указанными аргументами и в рабочей директории. Параметры:

3.1 `cmdArguments` – команда, которая будет передана для выполнения.

3.2 `directory` – рабочая директория для выполнения команды. Метод возвращает объект `Process`, который настроен для выполнения команды с редиректами стандартного ввода, вывода и ошибок.

4 `FindFreePortInRange(int rangeStart, int rangeEnd)` находит первый свободный порт в указанном диапазоне портов. Диапазон портов определяется параметрами `rangeStart` и `rangeEnd`. Метод пытается открыть TCP-сокеты на каждом порту в указанном диапазоне и проверяет, доступен ли он для использования. Если порт свободен, метод возвращает его, иначе продолжает поиск. Если не удастся найти свободный порт в указанном диапазоне, поиск продолжается в диапазоне с 1025 по 65535. Метод использует класс `TcpListener` для проверки доступности порта.

3.6.6 HubClientsCache

`HubClientsCache` – это класс, который управляет кэшированием и отслеживанием соединений между клиентами и процессами. Он хранит информацию о подключениях, а также предоставляет методы для добавления, удаления и обновления соединений.

Поля:

1 `_maxConnectionsAmount` – максимальное количество одновременных подключений, которые могут быть сохранены в кэше. Если это количество превышено, старые записи будут удалены.

2 `_currentRecordsAmount` – текущее количество записей в кэше.

3 `_id` – уникальный идентификатор для каждого подключения. Этот идентификатор увеличивается с каждым новым подключением.

4 `_connectionIds` – словарь, хранящий связь между идентификаторами подключений `SignalR (hubConnectionId)` и идентификаторами процессов в кэше.

5 `_connections` – основной словарь, который хранит информацию о процессе, связанном с подключением.

Методы:

1 `GetConnection(uint id)` – извлекает информацию о подключении по его уникальному идентификатору. Обновляет время последнего использования.

2 `GetConnectionId(string hubConnectionId)` – возвращает уникальный идентификатор процесса по `hubConnectionId`.

3 `GetLastUsingTime(uint id)` – возвращает время последнего использования для указанного идентификатора.

4 `SetHandler(uint id, DataReceivedEventHandler handler)` – назначает обработчик события для вывода данных из процесса.

5 `SetConnectionId(string hubConnectionId, uint id)` – связывает идентификатор соединения SignalR с уникальным идентификатором процесса.

6 `AddConnection(ConnectionInfo connInfo)` – добавляет новое подключение в кэш. Если количество записей превышает максимальный лимит, удаляются неактуальные записи.

7 `RemoveHubConnection(string hubId)` – удаляет соединение по `hubConnectionId` из словаря `_connectionIds`.

8 `RemoveConnection(string hubId)` – удаляет информацию о подключении и процессе из кэша. Если процесс связан с контейнером, он убивается.

9 `RemoveServerConnection(string hubId, out ProcessInfo processInfo)` – удаляет информацию о процессе и завершает его (убивает процесс, связанный с контейнером).

10 `RemoveIrrelevantRecords()` – удаляет записи, которые не использовались более 10 минут для освобождения места в кэше.

3.6.7 FtpManager

Класс `FtpManager` отвечает за управление FTP-пользователями, используя команды, которые выполняются через интерфейс `IOsInteractionsService`. Это позволяет создавать и удалять пользователей FTP с помощью утилиты `htpasswd`.

Класс содержит метод `ManageFTPUser(string username, ManageMode mode)` управляющий FTP-пользователями в зависимости от переданного режима. Если режим — создание пользователя, то используется опция `-d` для добавления нового пользователя, если режим — удаление пользователя, то используется опция `-D`. Генерируется случайный пароль, который затем используется для управления пользователем через команду `htpasswd`.

3.7 Классы бизнес-логики

В данном разделе описаны классы с основной логикой приложения. Все эти классы реализуют интерфейс `IRequestHandler<TRequest, TResponse>` – интерфейс из библиотеки `MediatR`, который используется для обработки запросов в шаблоне CQRS (Command-Query Responsibility Segregation).

3.7.1 CreateDSTServerHandler

Класс `CreateDSTServerHandler` обрабатывает запрос на создание сервера DST (Don't Starve Together) для пользователя. Он использует данные

конфигурации, взаимодействует с репозиториями данных, а также выполняет действия по созданию директорий, копированию файлов и генерации необходимых настроек для нового сервера.

Содержит единственный метод `Handle(CreateDSTServerRequest request, CancellationToken cancellationToken)`. Данный метод обрабатывает запрос на создание нового сервера DST для указанного пользователя. Он выполняет следующие шаги:

- проверяет, существует ли пользователь с данным `Id`;
- проверяет, был ли уже создан сервер для этого пользователя;
- создает необходимые каталоги и копирует файлы шаблонов для сервера;
- обновляет данные пользователя и сохраняет настройки для созданного сервера в файлы конфигурации;
- создает исполняемые скрипты для запуска сервера.

Возвращает объект `OkObjectResult` с сообщением об успешном создании сервера, если все шаги прошли успешно. Если пользователь не найден или сервер уже существует, возвращает ошибку.

3.7.2 DeleteDSTServerHandler и DeleteMinecraftServerHandler

Класс `DeleteDSTServerHandler` обрабатывает запрос на удаление сервера Don't Starve Together (DST).

Метод `Handle>DeleteDSTServer request, CancellationToken cancellationToken)` выполняет удаление игрового сервера DST пользователя. Шаги:

- ищет пользователя по `Id`;
- Удаляет файлы и папки, связанные с сервером;
- Обновляет информацию в базе данных;
- Возвращает результат операции.

`DeleteMinecraftServerHandler` работает аналогично, но для серверов Minecraft.

3.7.3 CreateMinecraftServerHandler

Класс `CreateMinecraftServerHandler` обрабатывает запрос на создание сервера Minecraft.

Поля:

- `_unitOfWork` – объект для работы с базой данных;
- `_config` – объект конфигурации, содержащий пути к директориям;
- `_processManager` – сервис для выполнения команд в операционной системе.

Метод `Handle(CreateMinecraftServerRequest request, CancellationToken cancellationToken)` выполняет создание нового сервера Minecraft для пользователя. Принцип работы:

- проверяет, существует ли пользователь и нет ли у него уже созданного сервера;
- создает папку для сервера, если она отсутствует;
- устанавливает сервер в зависимости от типа (forge, fabric или vanilla);
- записывает файл eula.txt с согласием на лицензионное соглашение;
- обновляет информацию о пользователе и сохраняет изменения в базе данных.

3.7.4 StartMinecraftServerHandler

Класс `StartMinecraftServerHandler` отвечает за запуск сервера Minecraft в контейнере Docker.

Поля:

- `_config` – объект `IConfiguration`, содержащий настройки приложения, такие как директории серверов.
- `_processManager` – сервис `IOsInteractionsService`, используемый для управления процессами и выполнения команд.
- `_unitOfWork` – объект `IUnitOfWork`, обеспечивающий доступ к базе данных и управление транзакциями.
- `_scopeFactory` – объект `IServiceScopeFactory`, который позволяет создавать скоупы `IServiceScope` для работы с `IUnitOfWork` в асинхронных задачах.

Методы:

1 `Handle(StartMinecraftServer request, CancellationToken cancellationToken)` запускает Minecraft-сервер в Docker-контейнере. Принцип работы:

- получает пользователя из базы данных по `request.Id`;
- проверяет, есть ли у пользователя созданный сервер;
- определяет путь к файлам сервера и ищет свободный порт;
- создаёт процесс `docker run` для запуска сервера;
- добавляет информацию о запущенном сервере в базу данных;
- запускает асинхронную задачу, которая удаляет запись о сервере при завершении его работы;
- ожидает запуска контейнера и возвращает `OkObjectResult` с адресом сервера.

2 `WaitForContainerAsync(string containerName, int timeoutSeconds = 30)` ожидает, пока контейнер с указанным именем (`containerName`) будет запущен.

3.7.5 GetContentHandler

Класс `GetContentHandler` реализует интерфейс `IRequestHandler<GetContentRequest, IActionResult>` и предназначен для получения содержимого директорий и файлов, связанных с пользователем.

Поля:

- `_readableFileExtensions` – статический массив допустимых расширений файлов (.txt, .json, .log), содержимое которых можно прочитать и вернуть.

- `_unitOfWork` – объект `IUnitOfWork`, используемый для работы с базой данных.

- `_configuration` – объект `IConfiguration`, содержащий настройки сервера, включая путь к директориям пользователей.

Метод `Handle(GetContentRequest request, CancellationToken cancellationToken)` возвращает список файлов и папок в указанной директории или содержимое файла, если он поддерживается.

3.7.6 GetServerSettingsHandler

Класс `GetServerSettingsHandler` предназначен для получения настроек Minecraft-сервера пользователя.

Поля:

- `_unitOfWork` – объект `IUnitOfWork`, используемый для работы с базой данных.

- `_configuration` – объект `IConfiguration`, содержащий настройки сервера, включая путь к директориям пользователей.

Метод `Handle(GetServerSettingsRequest request, CancellationToken cancellationToken)` возвращает содержимое файла настроек сервера.

3.7.7 GetServerStatusHandler

`GetServerStatusHandler` используется для получения статуса работающего сервера пользователя.

Поля:

- `_unitOfWork` – объект `IUnitOfWork`, предоставляющий доступ к репозиториям данных (в данном случае для работы с сущностью `RunningServer`).

- `_processManager` – объект `IOsInteractionsService`, который предоставляет методы для взаимодействия с операционной системой, в том числе для получения информации о внешнем IP-адресе.

Метод `Handle(GetServerStatusRequest request, CancellationToken cancellationToken)` возвращает информацию о статусе сервера(включен ли) и его адрес.

3.7.8 UpdateFileHandler

`UpdateFileHandler` используется для обновления файлов сервера пользователем.

Поля:

`_config` – объект `IConfiguration`, который предоставляет доступ к конфигурации приложения, в частности к директориям, используемым для хранения файлов серверов.

`_unitOfWork` – объект `IUnitOfWork`, предоставляющий доступ к репозиториям данных для работы с сущностью `User`.

Метод `Handle(UpdateFileRequest request, CancellationToken cancellationToken)` обрабатывает запрос на обновление содержимого файла.

3.8 Классы инициализации

Данные классы являются классами инициализации программы после ее запуска.

3.8.1 `ServiceCollectionExtensions`

`ServiceCollectionExtensions` содержит методы-расширения для добавления различных слоев и сервисов в контейнер зависимостей `IServiceCollection`. Эти методы позволяют упростить регистрацию сервисов, репозиториях и контекста базы данных в приложении.

Методы:

1 `AddInfrastructureLayer(this IServiceCollection services, IConfiguration config)` инициализирует добавление всех компонентов инфраструктуры.

2 `AddServices(this IServiceCollection services)` регистрирует необходимые сервисы в DI контейнере.

3 `AddDbContext(this IServiceCollection services, IConfiguration configuration)` регистрирует контекст базы данных с использованием строки подключения из конфигурации.

4 `AddRepositories(this IServiceCollection services)` регистрирует репозитории для работы с данными:

- `IUnitOfWork` – для управления транзакциями и репозиториями.

- `IRepository<>` – обобщённый репозиторий для работы с сущностями базы данных.

3.8.2 `Program`

Класс `Program` конфигурирует и запускает веб-приложение, используя `ASP.NET Core`. Он настраивает необходимые сервисы, такие как аутентификация, CORS, Swagger, а также подключение к базам данных, а также настраивает различные `middleware` для обработки запросов.

Класс содержит один главный метод `Main`, который запускает приложение и настраивает все необходимые компоненты через `WebApplication`.