



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

ФАКУЛЬТЕТ *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ

*«Разработка системы автоматизированного запуска
выделенных серверов Unreal Engine 4»*

Студент РК6-31М
(Группа)

Д. В. Боженко
(подпись, дата) (инициалы и фамилия)

Руководитель

Ф. А. Витюков
(подпись, дата) (инициалы и фамилия)

Москва, 2024 г.

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

УТВЕРЖДАЮ
Заведующий кафедрой РК6
А.П. Карпенко

« ____ » _____ 2024 г.

ЗАДАНИЕ
на выполнение курсового проекта

по дисциплине _____ Машинное обучение

Студент группы РК6-31М

_____ Боженко Дмитрий Владимирович
(Фамилия, имя, отчество)

Тема курсового проекта Разработка системы автоматизированного запуска выделенных серверов Unreal Engine 4

Направленность КП (учебная, исследовательская, практическая, производственная, др.) учебная
Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 5 нед., 50% к 11 нед., 75% к 14 нед., 100% к 16 нед.

Техническое задание: 1. Реализовать один из выбранных методов автоматизированного запуска выделенного сервера Unreal Engine;
2. Проанализировать доступные методы аутентификации для запуска выделенных серверов, выбрать наиболее подходящий и реализовать его;
3. Реализовать хранение данных о транзакциях на клиентской стороне;
4. Реализовать основу для балансировки клиентов по серверам, включающую регистрацию серверов в менеджере и мониторинг их загрузки по числу игроков.

Оформление курсового проекта:

Расчетно-пояснительная записка на 29 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

Дата выдачи задания «15» сентября 2024 г.

Руководитель НИР

(Подпись, дата)

Витюков Ф.А.
И.О. Фамилия

Студент

(Подпись, дата)

Боженко Д. В.
И.О. Фамилия

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре

АННОТАЦИЯ

В представленной работе рассмотрен поэтапный процесс реализации автоматизированного запуска выделенных серверов Unreal Engine 4. В каждом пункте работы описано проведенное исследование в области решаемой задачи и обоснование выбранного решения. Для ознакомления с принципами работы системы в каждом пункте работы подробно описан процесс реализации поставленной задачи.

Результаты представленной работы могут быть полезны для тех, кто желает изучить процесс разработки системы в ознакомительных целях.

В расчетно-пояснительной записке 34 листа, 17 рисунков.

СОДЕРЖАНИЕ

Аннотация	3
Перечень сокращений и условных обозначений	5
Введение	6
1. Автоматизация запуска выделенного сервера.....	7
1.1. Программная реализация демона	7
1.2. Запуск демона на Linux Debian.....	9
1.3. Реализация обратного сетевого взаимодействия	13
2. Валидация процесса запуска выделенных серверов.....	19
3. Хранение информации о транзакциях на стороне клиента	23
4. Реализация мониторинга состояния серверов.....	29
Заключение	33
Список использованных источников	34
Приложение А	35

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

UE – движок Unreal Engine 4.

TCP – протокол транспортного уровня, обеспечивающий надежную доставку данных

EOS – встроенная в движок подсистема, предоставляющая низкоуровневые программные инструменты

ВВЕДЕНИЕ

Одним из важнейших аспектов работы с Unreal Engine является запуск выделенных серверов, которые обеспечивают многопользовательскую работу, взаимодействие между клиентами и поддержание сетевой архитектуры. Однако автоматизация процесса запуска и управления такими серверами представляет собой сложную задачу, особенно в контексте масштабируемых и распределенных систем, где необходимо учитывать множество факторов, включая доступность ресурсов, производительность и отклик системы.

В рамках данной исследовательской работы рассматривается разработка и внедрение системы автоматического запуска выделенных серверов Unreal Engine на основе C++. Разработанная система позволяет пользователям инициировать запуск серверов по запросу с клиентского приложения. Такая автоматизация значительно упрощает управление серверами, снижает нагрузку на операционные команды и позволяет оптимизировать использование вычислительных ресурсов.

Также важным компонентом является аутентификация при запуске серверов, что обеспечивает безопасный доступ к системным ресурсам и позволяет гарантировать, что только авторизованные пользователи могут управлять запуском или подключением к выделенным серверам. Аутентификация играет критическую роль в предотвращении несанкционированного использования серверных мощностей и защиты пользовательских данных в многопользовательской среде.

Кроме того, внедрение возможностей работы в оффлайн-режиме повышает удобство использования системы для конечных пользователей. Работа в оффлайн-режиме позволяет пользователям сохранять доступ к ключевым функциям и данным, даже в случае временных перебоев с сетью.

1. АВТОМАТИЗАЦИЯ ЗАПУСКА ВЫДЕЛЕННОГО СЕРВЕРА

Для создания системы автоматизированного запуска выделенных серверов Unreal Engine (UE) необходимо разработать несколько ключевых компонентов, обеспечивающих слаженную работу системы. Основой является программа-демон (демон), которая будет отвечать за запуск экземпляров серверов. Демон работает в фоновом режиме, отслеживая запросы на создание новых серверных процессов и обеспечивая их корректное выполнение. Это позволяет автоматически управлять ресурсами и запускать новые серверные экземпляры в зависимости от текущих потребностей.

Не менее важен и менеджер серверов (менеджер) — компонент, который обрабатывает запросы клиентов и взаимодействует с программой-демоном. Менеджер серверов принимает входящие запросы от клиентских приложений, обрабатывает их и, при необходимости, направляет инструкции демону для запуска, завершения или обновления серверных процессов.

1.1. Программная реализация демона

Демон — это программа, которая запускается в фоновом режиме и выполняет поставленные ей задачи [1]. К основным задачам демона в рамках данной практической работы можно отнести умение принимать команды и реагировать на них; умение запускать по команде экземпляр выделенного сервера UE; умение собирать информацию системы о занятых и свободных вычислительных ресурсах; умение собирать информацию о состоянии всех запущенных выделенных серверов UE.

Для реализации поставленных задач была выбрана библиотека с открытым исходным кодом *Boost*, которая включает в себя классы и пространства имен для работы с сетевой частью и потоками, необходимые для реализации демона.

Для того, чтобы демон мог принимать входящие сообщения, был создан сетевой сокет, основанный на транспортном протоколе TCP [2]. Прослушивание по сокету является блокирующей операцией из-за наличия бесконечного цикла,

поэтому необходимо было создать отдельный поток и инициализировать в нем сокет, как показано в листинге (приложение А, листинг А.1).

С помощью класса *boost::asio::ip::tcp::endpoint* необходимо указать, по какому адресу и порту будет происходить прослушивание входящих соединений. Важно указать именно адрес 0.0.0.0, а не 127.0.0.1, так как при указании адреса 127.0.0.1 сервис будет доступен только в рамках localhost. При указании сетевого интерфейса 0.0.0.0 сервис будет доступен для любого внешнего подключения в пределах локальной сети.

Обработка команд была реализована в функции *handleIncomeQuery*, которая читает сообщения из сокета *clientSocket*, создаваемого для каждого входящего подключения (приложение А, листинг А.2).

В данном листинге можно увидеть пример, что при успешном чтении данных из *clientSocket* и получении команды “START”, вызывается функция *startServerInstance*. Важно отметить, что вызов функции является блокирующей операцией. Для этого, работа функции была вынесена в отдельный поток с помощью экземпляра класса *boost::thread*. Также необходимо было выполнить метод *detach*, который «отсоединяет» указанный поток, что позволяет ему полностью независимо существовать от вызвавшего его основного потока.

Как показано в листинге (приложение А, листинг А.3), запуск экземпляра сервера осуществлялся через создание экземпляра класса *boost::process::child*, который позволяет порождать дочерние процессы. Параметр конструктора *scriptPath* представляет собой путь до скрипта, запускающий процесс выделенного сервера; *boost::process::std_out > stdout* указывает, что вывод процесса необходимо перенаправлять в стандартный поток вывода; *boost::process::std_err > stderr* аналогично указывает на необходимость перенаправления потока ошибок.

Процесс запуска для безопасности приложения важно было обернуть в конструкцию *try catch*, так как это потенциально опасное место, где могут возникнуть ошибки и исключения. В блоке *try* демон запускает указанный

процесс. При успешном запуске экземпляра выделенного сервера в консоль выводится соответствующее сообщение. Также при возникновении исключений, они обрабатываются в блоке *catch* и выводятся в поток вывода ошибок для дальнейшей отладки.

1.2. Запуск демона на Linux Debian

Для запуска программы на Linux Debian для начала необходимо установить чистый дистрибутив без какого-либо графического интерфейса. Для более простой реализации в рамках разработки было принято решения использовать гипервизор VirtualBox. Для того, чтобы виртуальная машина была видна во внутренней сети, необходимо провести первоначальные сетевые настройки гипервизора.

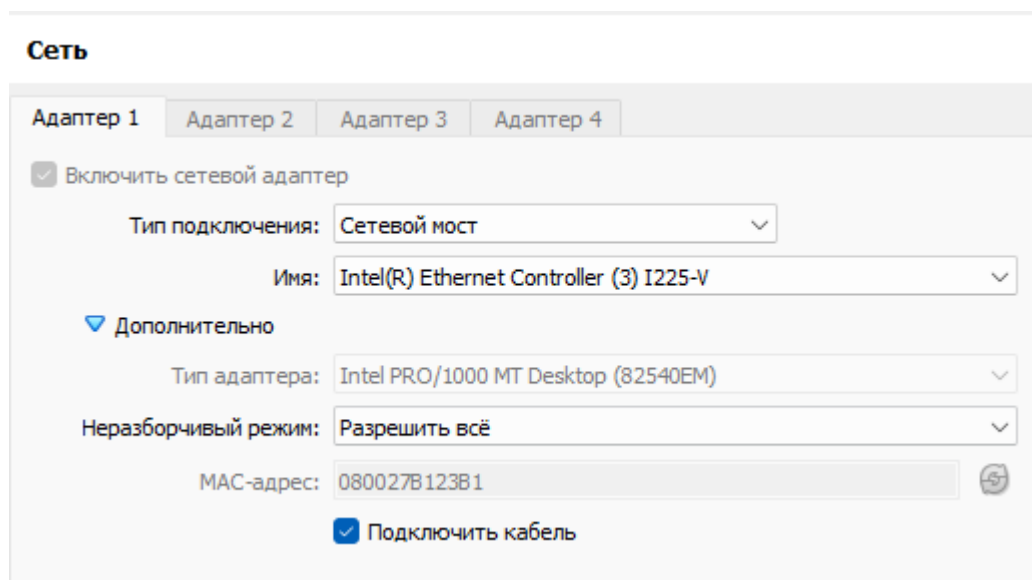


Рисунок 1 – Сетевые настройки гипервизора

Как видно на рисунке 1, в качестве типа подключения необходимо указать “Сетевой мост”; в качестве имени необходимо указать сетевой интерфейс, в данном случае технология *Ethernet*. В пункте “Неразборчивый режим” необходимо указать “Разрешить все”. После запуска ВМ необходимо проверить IP-адрес машины с помощью команды *ip a* (рисунок 2).

```

user@debsserver-daemon:~/daemon/bmstu_src/Master/Diploma/DaemonBoost/DaemonBoost/DaemonBoost$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:b1:23:b1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.12/24 brd 192.168.1.255 scope global dynamic enp0s3
        valid_lft 84546sec preferred_lft 84546sec
    inet6 2a00:1370:81a0:3b89:a00:27ff:feb1:23b1/64 scope global dynamic mngtmpaddr
        valid_lft 481sec preferred_lft 481sec
    inet6 fe80::a00:27ff:feb1:23b1/64 scope link
        valid_lft forever preferred_lft forever
user@debsserver-daemon:~/daemon/bmstu_src/Master/Diploma/DaemonBoost/DaemonBoost/DaemonBoost$ _

```

Рисунок 2 – Вывод сетевых параметров VM

Чтобы понять, что VM доступна в локальной сети и между основной машиной и VM доступны сетевые взаимодействия, была выполнена команда *ping 192.168.1.12* с основной машины (рисунок 3).

```

C:\Users\User>ping 192.168.1.12

Обмен пакетами с 192.168.1.12 по 32 байтами данных:
Ответ от 192.168.1.12: число байт=32 время=6мс TTL=64
Ответ от 192.168.1.12: число байт=32 время<1мс TTL=64
Ответ от 192.168.1.12: число байт=32 время<1мс TTL=64
Ответ от 192.168.1.12: число байт=32 время<1мс TTL=64

Статистика Ping для 192.168.1.12:
    Пакетов: отправлено = 4, получено = 4, потеряно = 0
    (0% потерь)
Приблизительное время приема-передачи в мс:
    Минимальное = 0мсек, Максимальное = 6 мсек, Среднее = 1 мсек

```

Рисунок 3 – Выполнение проверки на доступность VM в локальной сети

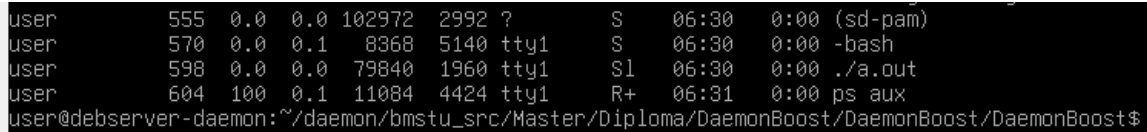
Далее, чтобы скомпилировать программу для начала была скачена и установлена библиотека *boost*. Для этого была использована команда *wget* для скачивания архива; для его разархивации была использована команда *tar -C /home/user/boost. /home/user/boost* – рабочая директория, где в дальнейшем будут лежать исходные и бинарные файлы библиотеки. После установки для компиляции программы необходимо в директории, где лежит программа, выполнить команду

```

g++ -I /home/user/boost/boost_1_82_0 DaemonBoost.cpp -L
/home/user/boost/boost_1_82_0/stage/lib -lboost_filesystem -lboost_system -
lboost_thread

```

и запустить файл `./a.out &`. Знак `&` означает, что процесс будет выполняться в фоновом режиме и не будет блокировать ввод других команд в терминал. Для того, чтобы убедиться в запуске процесса, можно выполнить команду `ps aux | grep ./a.out`, которая покажет *PID* запущенного процесса. Для завершения процесса использовалась команда `kill -9 <PID>`.



```
user      555  0.0  0.0 102972  2992 ?        S    06:30   0:00 (sd-pam)
user      570  0.0  0.1   8368   5140 tty1    S    06:30   0:00 -bash
user      598  0.0  0.0   79840   1960 tty1    Sl   06:30   0:00 ./a.out
user      604  100  0.1   11084   4424 tty1    R+   06:31   0:00 ps aux
user@debserver-daemon:~/daemon/bmstu_src/Master/Diploma/DaemonBoost/DaemonBoost/DaemonBoost$
```

Рисунок 4 – Просмотр работающего процесса через команду `ps aux`

Далее для того, чтобы из локальной сети можно было подключиться к процессу демона, необходимо изменить правила *firewall* и разрешить прослушивание по указанному порту и протоколу. Для открытия порта была использована программная утилита *firewalld*. Ниже представлен набор команд, который использовался для открытия порта.

```
systemctl start firewalld
systemctl enable firewalld
firewall-cmd --permanent --zone=public --add-port=8871/tcp
firewall-cmd --reload
```

Команда *start* запускает сервис, команда *enable* включает сервис при каждой загрузке системы, третья основная команда добавляет правило для порта 8871 и указывает, что прослушивание будет производиться по протоколу TCP. Четвертая команда применяет новые правила и перезапускает сервис. Если добавление было произведено успешно, то команда *firewall-cmd --list-ports* покажет добавленный порт в списке. С помощью команды *netstat -tuln* проверялось, что сервис запущен, слушал входящие соединения и был доступен по сетевому интерфейсу 0.0.0.0 (рисунок 5).

```

user@debserver-daemon:~/daemon/bmstu_src/Master/Diploma/DaemonBoost/DaemonBoost/DaemonBoost$ netstat
-tuln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:8871            0.0.0.0:*              LISTEN
tcp      0      0 0.0.0.0:22             0.0.0.0:*              LISTEN
tcp6     0      0 :::22                  :::*                    LISTEN
udp      0      0 0.0.0.0:68             0.0.0.0:*
user@debserver-daemon:~/daemon/bmstu_src/Master/Diploma/DaemonBoost/DaemonBoost/DaemonBoost$ _

```

Рисунок 5 — Проверка всех сервисов, прослушивающих входящие соединения по протоколам *TCP/UDP*

После выполнения отладки запуска сервиса было выполнено копирование файлов выделенного сервера под ядро UNIX с основной машины, на виртуальную машину Linux Debian. Было принято решение передать файлы по протоколу ssh. Для того, чтобы убедиться в работе ssh-сервера на Linux, была использована команда *service ssh status*. Для удаленного копирования с Windows на Linux на основной машине из терминала Power Shell была выполнена команда

```

pscp E:\Master\sem2\MMAPS\Releases\LinuxServer.zip
user@192.168.1.12:/home/user/dedicated-server

```

Для запуска выделенного сервера UE в целях безопасности нельзя использовать пользователя с правами *root*. Поэтому для этого был создан пользователь *user*, которому были прописаны права *sudo*. Также скрипту, который запускает экземпляр выделенного сервера, были выданы права с помощью выполнения команды

```

chmod +x /home/user/dedicated-server/LinuxServer/Lab4Server.sh.

```

После выполнения настроек и установок была проверена работоспособность системы. В результате обработки двух команд “START” удаленно были запущены два экземпляра выделенных серверов, как показано на рисунке ниже.

```

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:8871            0.0.0.0:*              LISTEN
tcp      0      0 0.0.0.0:22             0.0.0.0:*              LISTEN
tcp      0      0 0.0.0.0:1985            0.0.0.0:*              LISTEN
tcp6     0      0 :::22                  :::*                    LISTEN
udp      0      0 0.0.0.0:7777            0.0.0.0:*
udp      0      0 0.0.0.0:7778            0.0.0.0:*
udp      0      0 0.0.0.0:68             0.0.0.0:*
user@debserver-daemon:~/daemon/bmstu_src/Master/Diploma/DaemonBoost/DaemonBoost/DaemonBoost$ _

```

Рисунок 6 — Список запущенных выделенных серверов UE

На рисунке 6 видно, что запущено два сервиса на портах 7777/udp (порт по умолчанию для сервера Unreal Engine) и 7778/udp. Также можно заметить, что номер порта увеличивается с количеством запущенных экземпляров. Так как предполагается, что на одной машине с демоном будет запускаться несколько экземпляров выделенных серверов Unreal Engine, для них также были прописаны правила firewall с помощью команды *firewall-cmd --zone=public --permanent --port-add=7777-7797*. Запись команды в таком виде позволяет открыть сразу группу портов 7777-7797, что позволяет запустить до 20 экземпляров UE на одной машине. Также с помощью команды *ps aux* была проверен корректный запуск процессов Unreal Engine (рисунок 7).

```
user      650  0.0  0.0   2576   908 tty1    S    07:12   0:00 /bin/sh /home/user/dedicated-serv
user      657  6.6  5.2 1679872 209752 tty1    Sl   07:12   0:03 /home/user/dedicated-server/Linux
user      687  0.0  0.0   2576   936 tty1    S    07:12   0:00 /bin/sh /home/user/dedicated-serv
user      694  8.2  5.2 1679944 208680 tty1    Sl   07:12   0:02 /home/user/dedicated-server/Linux
user      722  0.0  0.1  11084   4328 tty1    R+   07:13   0:00 ps aux
user@debsserver-daemon:~/daemon/bmstu_src/Master/Diploma/DaemonBoost/DaemonBoost/DaemonBoost$ _
```

Рисунок 7 – Список запущенных процессов серверов Unreal Engine

Для проверки доступности запущенных серверов в рамках отладки с клиента Unreal Engine в консоли (по умолчанию открытие на клавишу ~) была выполнена команда *open 192.168.1.12:777* и *192.168.1.12:7778*, которая позволила подключиться к серверам Unreal Engine, запущенным на VM Linux Debian.

1.3. Реализация обратного сетевого взаимодействия

Под обратным сетевым взаимодействием подразумевается отправка сообщения с запущенного экземпляра выделенного сервера на клиент UE. Данное сообщение содержит IP-адрес и порт, на котором запущен выделенный сервер, которое необходимо для того, чтобы клиент, который изъявил желание запустить матч, получил необходимую информацию об адресе подключения. В ходе проведения исследовательской работы было выявлено два способа, с помощью которых можно организовать вышеописанное обратное сетевое взаимодействие (рисунок 8).

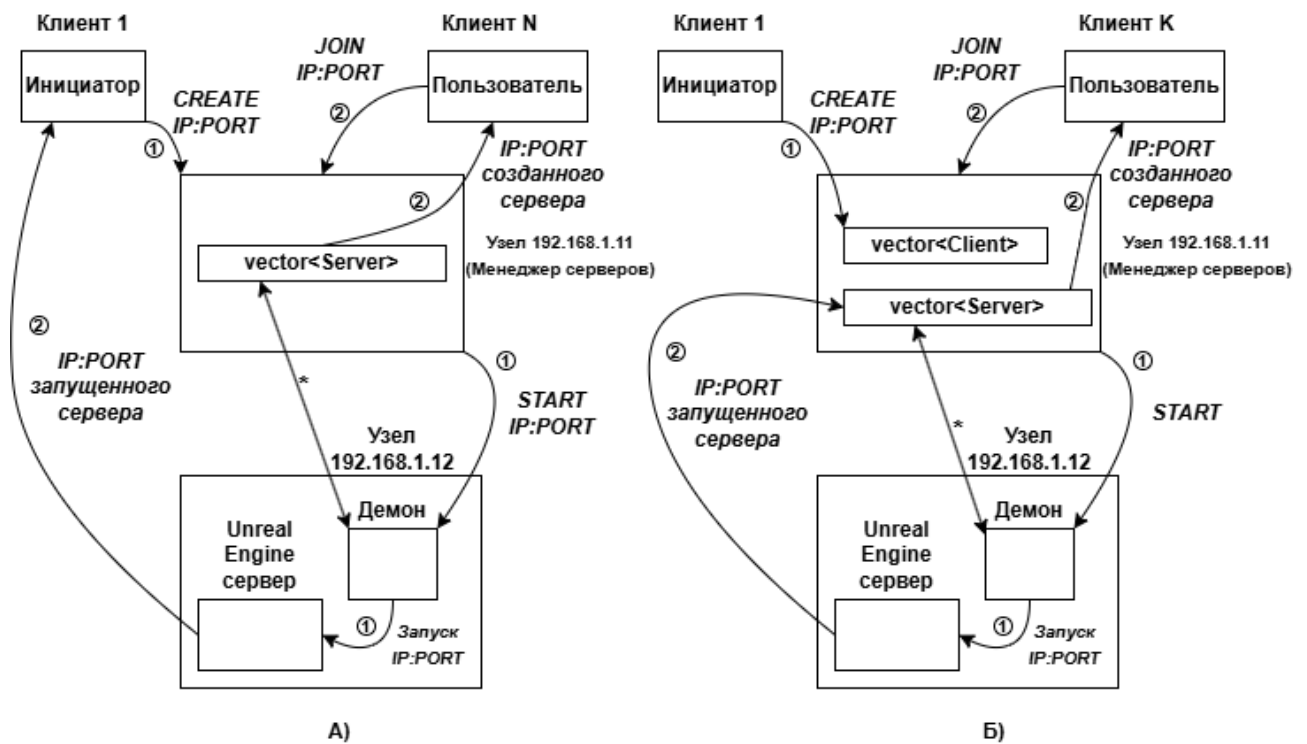


Рисунок 8 – А) Схема обратного сетевого взаимодействия напрямую с клиентом. Б) Схема обратного сетевого взаимодействия через менеджера серверов

Анализируя рисунок 8 можно увидеть два разных подхода к обратной отправке сообщения. Структура данных *vector<Server>* представляет собой список всех работающих экземпляров выделенных серверов. Структура данных *vector<Client>* представляет собой список всех клиентов, которые инициировали соединение с менеджером серверов. В первом подходе IP-адрес и порт запущенного экземпляра выделенного сервера сообщаются клиенту напрямую, без посредников.

К плюсам первого подхода можно отнести большую надежность доставки, так как сообщение проходит меньше сетевых узлов и вероятность потери сетевого пакета уменьшается. К минусам можно отнести отсутствие полного контроля над системой, так как сообщения о запуске сервера и отправке его IP-адреса не логируются централизованно через менеджера серверов. Также к минусам можно отнести вынужденную запись IP-адреса и порта клиента-инициатора в параметры запуска выделенного сервера Unreal Engine.

Во втором подходе сообщение с IP-адресом и портом передается обратно менеджеру серверов, а не напрямую клиенту, который изъявил желание начать сессию.

К плюсам второго способа можно отнести возможность лучшего контроля над системой, так как все действия с отправкой и записью сообщений централизованно логируются в программе менеджера серверов. К минусам такого подхода можно отнести большее потребление памяти, так как в программе необходимо хранить информацию о клиентах, которые выразили желание начать сессию.

В ходе выполнения исследовательской работы было принято решение в пользу второго способа сетевого взаимодействия, которое обеспечивает более предсказуемое и открытое поведение системы.

Для программной реализации вышеописанного подхода необходимо в классе, производного от *AGameMode*, в методе *BeginPlay* инициализировать отправку сообщения с указанным IP-адресом и портом, на котором запустился сервер. Можно предположить, что IP-адрес узла, на котором запускается выделенный сервер заранее известен в системе и может быть указан в конфиге приложения. Порт, на котором запуска приложение – заранее никогда не известен и должен быть получен программно не из конфига (приложение А, листинг А.4). Анализируя данный листинг, можно увидеть, что получение порта, по которому созданный сервер слушает входящие соединения, было реализовано с помощью метода *LowLevelGetNetworkNumber* класса *UNetDriver*. Данный метод возвращает строку, которую было необходимо разбить по символу разделителю двоеточия. Далее было необходимо получить порт и отправить полученную строку в программу сервера менеджеров. IP-адрес и порт, по которому слушает программа менеджера серверов была записана в конфиг приложения Unreal Engine.

В программе сервера менеджеров необходимо реализовать обработку сообщений с IP-адресом и портом, получаемых от запущенного выделенного

сервера. Для этого необходимо определить метод, который получает сообщения по сокету, выбирает нужного клиента из списка сохраненных в памяти приложения и отправляет ему полученное сообщение с информацией о подключении (приложение А, листинг А.5). Анализируя листинг, можно заметить, что отправка сообщений клиентам организована в соответствии с принципами работы очереди. Если по команде был запущен выделенный сервер, то необходимо отправить его IP-адрес и порт первому клиенту в очереди, который инициировал данный запуск (клиент с типом *ClientType::INITIATOR*).

Ниже представлена UML-диаграмма последовательности для успешного сценария сетевого взаимодействия клиента-инициатора с системой (рисунок 9).

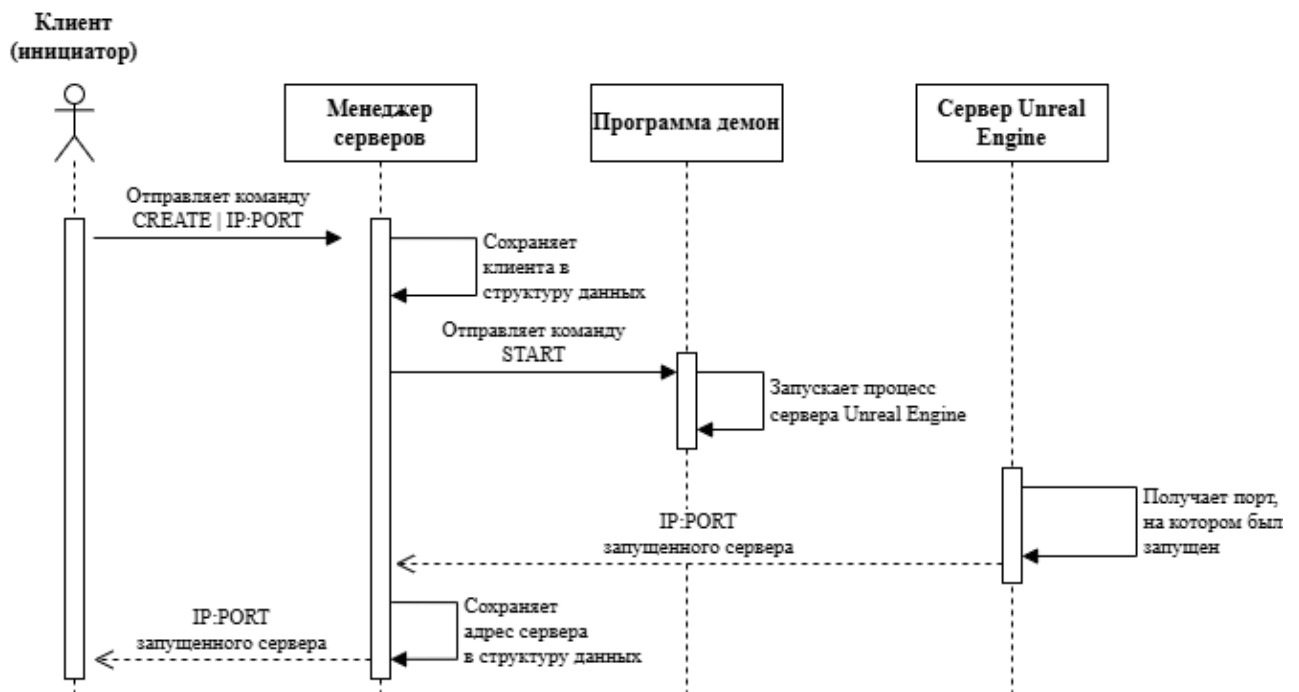


Рисунок 9 – UML-диаграмма последовательности взаимодействия клиента-инициатора с системой (успешный сценарий)

В итоге вышеописанного взаимодействия в логах программы менеджера серверов появляются следующие логи (рисунок 10).

```

[2024-08-26 22:15:21] Start listening on 0.0.0.0:8870

Got data from client: CREATE
Sending command to daemon: Start
Added client to queue: [CLIENT_ADDRESS=127.0.0.1:54578,CLIENT_TYPE=INITIATOR]
Got URI form started DedicatedServer: 127.0.0.1:7777
Senging data to cleint: 127.0.0.1:7777
  
```

Рисунок 10 – Логи менеджера при успешной обработке запросов

В логах программы-демона фиксируются события запуска экземпляров выделенных серверов и ошибки, если они возникают во время обработки команд запуска (рисунок 11).

```
Start listening on 0.0.0.0:8871
Handle income query: Start
Starting server instance...

E:\SummerPracticeEOS\github\bmstu_src\bmstu_src\Master\Diploma\DaemonBoost\DaemonBoost\Daemo
ticeEOS\github\bmstu_src\bmstu_src\Master\Diploma\Client\Binaries\Win64\Lab4Server.exe -log
Server instance started successfully!
```

Рисунок 11 — Логи демона после запуска выделенного сервера

На клиенте была реализована удобная обработка сообщения с информацией о подключении, чтобы пользователь мог подключиться по указанному IP-адресу через графический интерфейс.

Обработка команд была реализована в отдельном потоке, так как прослушивание по сокету является блокирующей операцией. Вследствие этого, важно упомянуть, что управление виджетами средствами C++ возможно только из главного потока, и невозможно из дочернего, который был создан специально для прослушивания сообщений. Для решения такой проблемы в UE был найден и использован метод *AsyncTask* [3], который позволяет асинхронно запустить задачу в указанном потоке (приложение А, листинг А.6).

В итоге, когда обработка выполняется, у пользователя на экране появляется виджет, с помощью которого он может подключиться к запущенному на выделенной машине серверу (рисунок 12).

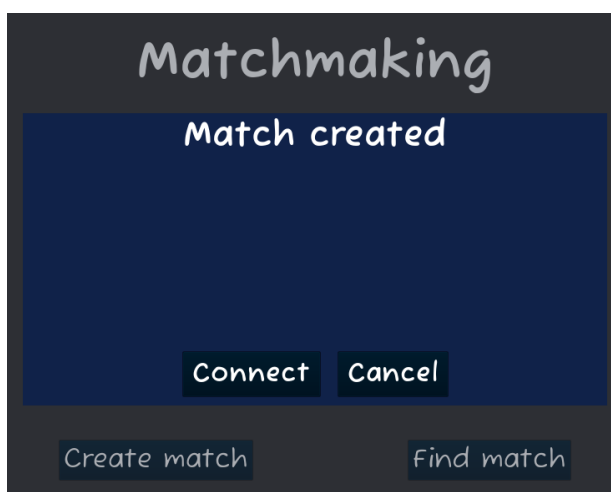
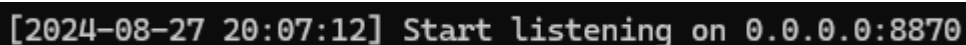


Рисунок 12 — Виджет управления подключением на клиенте

Для того, чтобы работа системы была понятна разработчику, и чтобы работу системы можно было легче отладить, почти каждое действие, происходящее в системе, должно быть зафиксировано. Как правило, в качестве места, куда записываются логи выбирают файл, где каждое действие записывается в формате "<TimeStamp>: <Выполненное действие>".

В программах менеджера серверов был реализован простой класс логгера через синглтон Майерса. Синглтон Майерса позволяет через статическую переменную в классе единожды, при первой инициализации, создать экземпляр логгера и переиспользовать уже созданный экземпляр в других местах программы.

Пример записи событий в журнал программой с помощью созданного логгера представлен ниже (рисунок 13).



```
[2024-08-27 20:07:12] Start listening on 0.0.0.0:8870
```

Рисунок 13 – Запись события в журнал программы

2. ВАЛИДАЦИЯ ПРОЦЕССА ЗАПУСКА ВЫДЕЛЕННЫХ СЕРВЕРОВ

Аутентификация для контроля возможности запуска выделенных серверов в приложении — важный компонент, обеспечивающий доступ к серверным ресурсам только для авторизованных пользователей. Реализация этого механизма может быть выполнена разными способами, включая использование готовых решений, таких как Epic Online Services (EOS), или разработку собственного решения с нуля. Важно учитывать достоинства и недостатки каждого из этих подходов, чтобы выбрать оптимальный метод для конкретного проекта.

Существует два основных подхода для решения задачи аутентификации: использование готового API, такого как EOS, или создание собственного решения. EOS предлагает готовый механизм аутентификации и управления покупками с помощью программных интерфейсов `PurchaseInterface` и `StoreInterface`, которые легко интегрируются в любой проект UE и обеспечивают безопасность за счет стандартизированных, протестированных методов. В качестве альтернативы, собственное решение позволяет разработчику самостоятельно настроить всю логику аутентификации, учитывая любые специфические требования.

Разработка собственного решения имеет свои плюсы и минусы. К её преимуществам можно отнести полный контроль над процессом аутентификации и возможность гибкой настройки всех аспектов, что особенно полезно при сложных или уникальных требованиях системы. Однако создание собственного решения сопряжено с рисками: велика вероятность появления уязвимостей и багов, которые сложнее выявить и устранить без ресурсов и опыта, как у команды EOS. Самостоятельная разработка потребует больше времени на тестирование и оптимизацию, чтобы соответствовать уровню безопасности, который предлагает EOS.

Подход с использованием EOS также имеет свои особенности. Основное преимущество EOS — это безопасность и стабильность. Платформа прошла

множество тестов в реальных условиях, а ее функциональность поддерживается экспертами в области безопасности. Кроме того, EOS обеспечивает быструю и надежную интеграцию, избавляя разработчиков от необходимости проектировать аутентификацию с нуля. Тем не менее, использование EOS может ограничивать гибкость, так как система стандартна и не всегда позволяет детально настроить каждый аспект под уникальные требования.

Таким образом, для задачи аутентификации, контролирующей доступ к выделенным серверам, EOS представляет собой предпочтительное решение. Оно минимизирует риски, связанные с уязвимостями и багами, и предоставляет разработчику надежную, проверенную инфраструктуру.

Для начала работы с представленными программными интерфейсами в настройках проекта EOS необходимо добавить предложение. Предложение – это сущность, хранящаяся в системе EOS, которую пользователь может приобрести и получить права на совершение определенного действия. В рамках поставленной задачи наличие у пользователя такого предложения, которое он получил через систему транзакций в интерфейсе проекта, означает, что у него есть право на создание выделенных серверов UE. В настройках проекта важно знать ID предложения, которое сгенерировала система EOS, и использовать его для выполнения операций с транзакциями.

Чтобы пользователь мог совершить покупку, он обязательно предварительно должен пройти аутентификацию в системе EOS. В методе *StartPurchase* описан реализованный процесс приобретения предложения и проверки результата транзакции (приложение А, листинг А.7). Анализируя листинг, можно увидеть, что метод *Checkout* является асинхронным и не блокирует основной поток приложения. Для проверки результата транзакции в параметры функции передается лямбда-функция, которая вызывается при получении результата из системы EOS. В случае успеха транзакции далее выполняется функция *QueryReceipts*, которая устанавливает за пользователем право запускать матч (приложение А, листинг А.8). В случае совпадения ID

предложения, найденного в транзакциях, булева переменная, определенная в классе `UGameInstance`, принимает значение `true`. Важно сохранять такие переменные именно в экземпляр класса, производного от `UGameInstance`, так как экземпляр данного класса существует в памяти на протяжении всего жизненного цикла приложения UE.

Управление запуском выделенного сервера осуществляется через виджет, представленный ниже (рисунок 14).

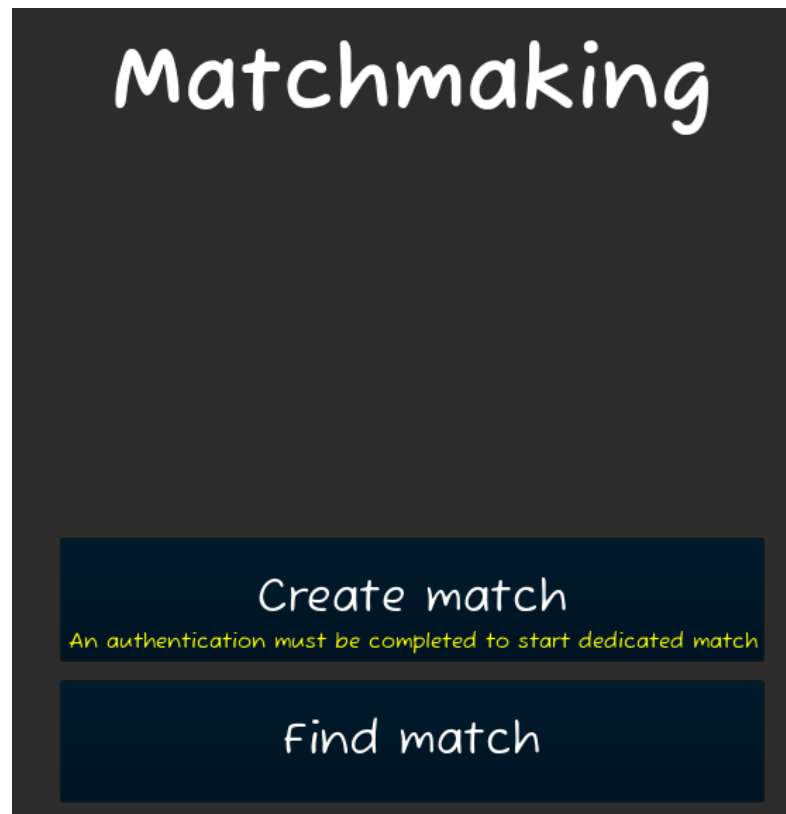


Рисунок 14 – Виджет управления запуском выделенных серверов

Если пользователь авторизовался и ранее не выполнил вышеописанный процесс с транзакцией, в интерфейсе рядом с элементом управления, отвечающим за запуск сервера, будет показано информационное сообщение о необходимости провести процесс аутентификации. При попытке нажать на элемент без подтвержденного права на запуск выделенного сервера система выведет соответствующее предупреждающее сообщение на экран пользователя. Управление элементом средствами C++ представлено ниже (приложение А, листинг А.9).

В случае, когда пользователь выполнил процесс приобретения предложения, система удалит подсказку и позволит пользователю запустить выделенный сервер (рисунок 15).

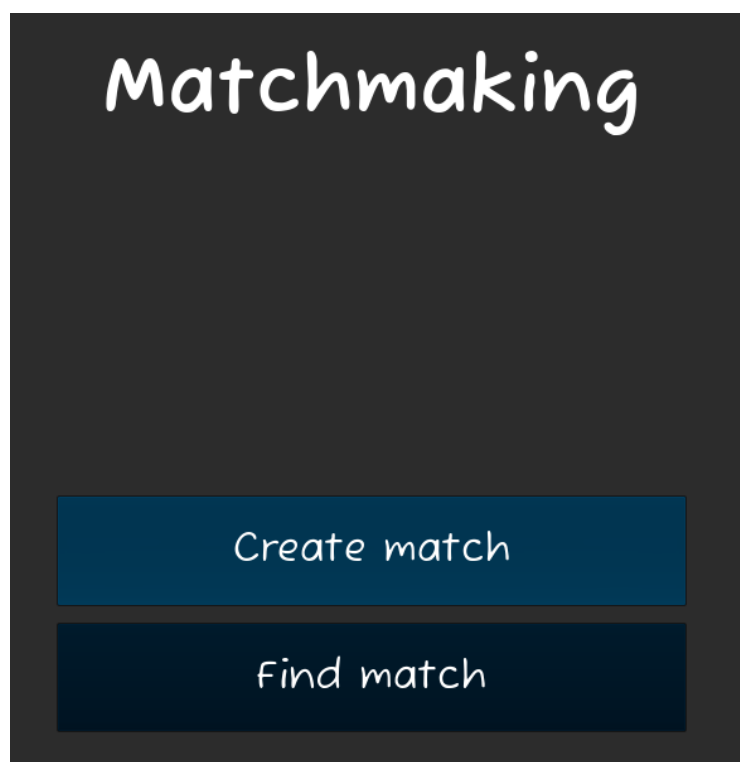


Рисунок 15 – Управляющий виджет после успешного проведения транзакции

3. ХРАНЕНИЕ ИНФОРМАЦИИ О ТРАНЗАКЦИЯХ НА СТОРОНЕ КЛИЕНТА

Хранение информации о транзакциях на стороне клиента — это решение, которое часто требует балансировки между безопасностью и удобством для пользователя. С одной стороны, безопасность данных критически важна, особенно если речь идет о сохранении информации о покупках и правах доступа. С другой стороны, удобство пользователей также играет ключевую роль, так как от этого зависит общий опыт работы с приложением. В современных условиях разработки необходимо учитывать оба аспекта, чтобы создать надежное и в то же время дружелюбное пользователю приложение.

Хотя хранение данных о транзакциях на устройстве клиента может показаться менее безопасным решением, этот подход имеет важное преимущество: пользователи могут работать в оффлайн-режиме и сохранять доступ на свои права в приложении без постоянного подключения к сети.

Хранение релевантной информации на стороне клиента сталкивается, как правило, с двумя основными проблемами: защитой от несанкционированного чтения файлов и предотвращением их распространения. Помимо риска того, что третьи лица могут получить доступ к информации о транзакциях, существует и проблема распространения данных — когда сами файлы могут быть перенесены на другое устройство или переданы другим пользователям. Решением для этих задач является использование криптографических методов в сочетании с механизмами, обеспечивающими привязку данных к конкретному устройству. Ниже представлен рисунок с схемой, на которой изображены часто используемые методы защиты от передачи данных между клиентами (рисунок 16).

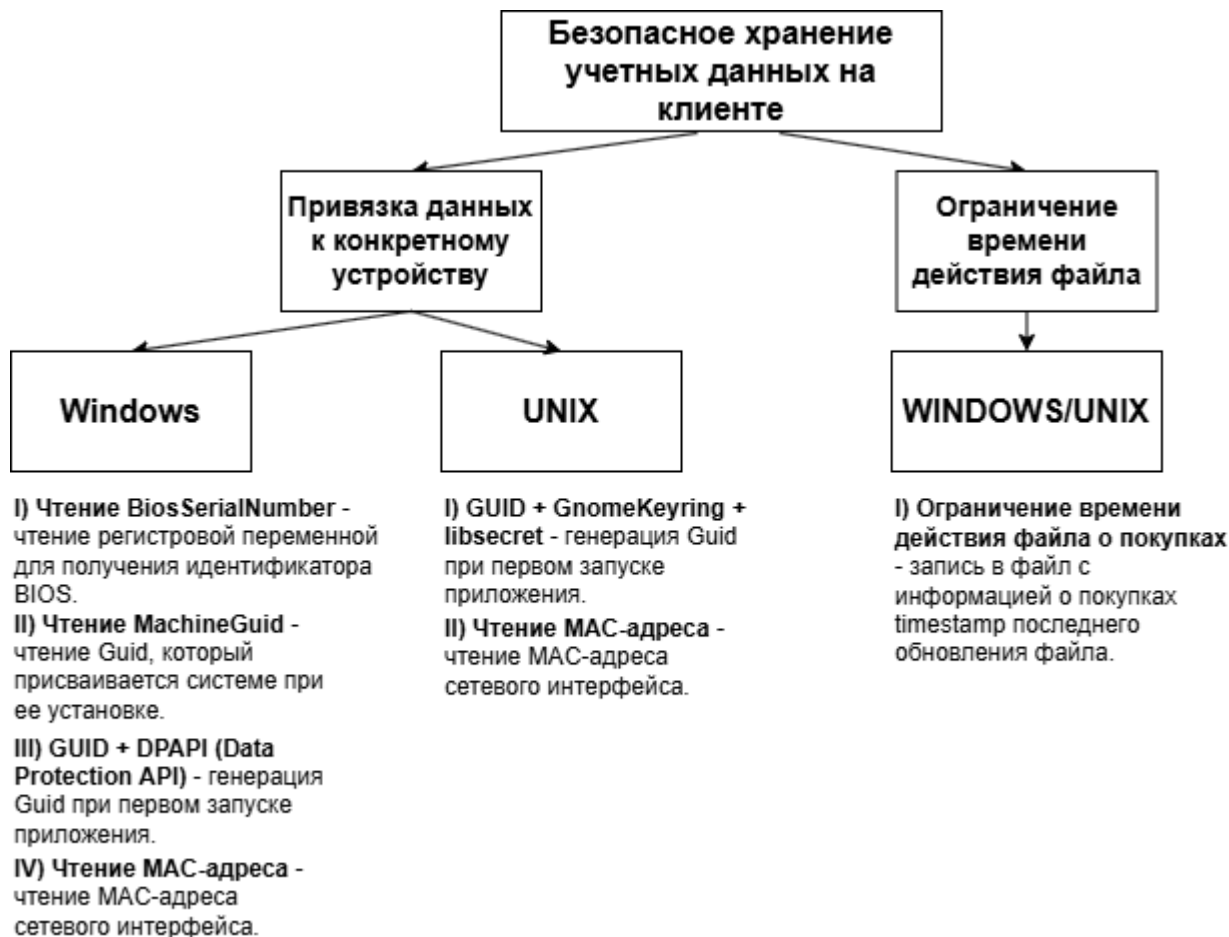


Рисунок 16 – Основные способы защиты от распространения файлов между клиентами

На рисунке 16 представлены основные способы защиты от передачи важных файлов между клиентами. Каждый из способов имеет свои преимущества и недостатки. Все способы защиты оценивались с учетом критерия безопасности и неудобств, которые они могут причинить пользователю приложения.

BIOS Serial Number. Решение через BIOS Serial Number представляет собой чтение регистровой переменной – уникального идентификатора BIOS. Такой подход обеспечивает доступ к файлам только на исходном устройстве. К преимуществам метода можно отнести то, что он предоставляет надежный уровень защиты, так как файлы будут жестко привязаны только к одному устройству. К недостаткам можно отнести: неудобство для пользователя, если

устройство заменено или отремонтировано; ограничение для клиентов на UNIX-подобных системах, так как данный подход будет работать только на системе Windows; требование запуска приложения от имени администратора, так как приложению будет необходимо читать переменные регистра.

MachineGuid. Аналогичное решение, как и чтение BIOS Serial Number, которое обладает теми же преимуществами и недостатками.

MAC-адрес. Чтение MAC-адреса позволяет однозначно идентифицировать пользователя, так как MAC-адрес уникален для каждого сетевого интерфейса. К преимуществам данного метода можно отнести: работоспособность на устройстве с любой операционной системой. К недостаткам можно отнести: неудобство для пользователя, так как при смене сетевой карты или сетевого интерфейса доступ будет потерян; MAC-адрес гораздо легче подделать, чем BIOS Serial Number или MachineGuid.

Guid. Реализация через Guid [5] представляет собой генерацию уникального идентификатора Guid при первом запуске приложения и дальнейшее его хранение в зашифрованном виде в файлах приложениях. Шифрование файлов также можно осуществить несколькими способами. Операционная система Windows предоставляет *Data Protection API*, которое делает возможным шифрование информации на основе данных учетной записи пользователя от Microsoft. Такой подход к шифрованию обеспечивает надежную защиту, но, очевидно, он не будет работать на клиенте с UNIX-подобной операционной системой. Еще одним очевидным недостатком является то, что при смене учетной записи Microsoft пользователь потеряет доступ к данным файлам, что причиняет неудобство в использовании.

Также ключ шифрования можно хранить в коде самого приложения. Ключ шифрования будет непросто получить несанкционированным способом, если файлы приложения будут доступны пользователю только в виде бинарных файлов без доступа к исходному коду.

Ограничение времени действия файла. Подход представляет собой запись в файл информации о последнем обновлении файла, т.е. когда пользователь был последний раз в сети. При каждом обращении к файлу необходимо проверять дату последнего обновления. В случае, если файлы давно не обновлялись, пользователь потеряет к ним доступ. Такой подход играет роль дополнительной защиты от несанкционированного доступа на время. Однако такой подход явно причиняет неудобство пользователю, в случае если он давно не заходил в сеть.

После проведения анализа всех вышеперечисленных методов было принято решение выполнить реализацию именно через генерацию уникального идентификатора Guid, с дальнейшим хранением его в файлах приложения и защитой информации с помощью шифрования. Такой подход не требует жесткой привязки к конкретному устройству, не требует запуска приложения от прав администратора, не привязан к конкретному типу операционной системы. Хотя такой подход и обеспечивает не самую надежную защиту от распространения и прочтения данных несанкционированным способом, однако он обеспечивает соблюдение критерия удобства использования, что, как уже отмечалось выше, является важным аспектом при разработке приложения.

Изначально для реализации описываемого процесса по использованию информации о транзакциях в оффлайн-режиме необходимо сгенерировать уникальный идентификатор приложения – Guid. Вероятность того, что в мире будет сгенерировано два одинаковых ключа крайне мала [4]. Поэтому Guid является отличным решением в качестве использования уникального идентификатора для приложения.

Если приложению при инициализации не удалось успешно найти файл по пути */Saved/AppData/AppGuid.dat*, то считается, что Guid еще не был сформирован и записан в файл (приложение А, листинг А.10). Также стоит отметить, что директория *Saved* была выбрана для хранения не случайным образом. Данная директория обычно содержит все не бинарные файлы, которые UE самостоятельно сохраняет и использует при работе. Данная директория

никогда автоматически не очищается движком, следовательно, исключается шанс, что релевантная информация о покупках может быть удалена программно.

Запись информации о транзакциях выполняется сразу после того, как пользователь совершил транзакцию, и при первом запуске приложения, когда пользователь вошел в режим онлайн.

Для каждой транзакции генерируется отдельный файл (приложение А, листинг А.11). В содержимое файла записывается Guid приложения, которое было сгенерировано ранее и ID предложения, которое было приобретено в рамках транзакции. Для того, что решить проблему уникальности файлов, было принято решение записывать имя файла в формате "<OfferID>_<PlayerID>". Причем OfferID следует хранить не в открытом виде в названии файла, а, например, в виде контрольной суммы SHA256.

Когда пользователь не прошел аутентификацию в системе EOS и нажимает на кнопку создать матч в оффлайн-режиме, система проверяет наличие файлов с информацией о транзакции с нужным OfferID в директории */Saved/Purchases/* (приложение А, листинг А.12).

Для дополнительной защиты пользователь должен ввести свой логин, который соответствует его логину в системе EOS. Введенный пользователем логин сравнивается с логином, который предоставила система EOS на момент создания и записи файла о транзакции в онлайн режиме. Хотя это отнюдь не полноценная аутентификация, но она может служить дополнительным препятствием для тех пользователей, кто пытается подделать учетную запись или обойти привязку данных к пользователю. Это может быть полезно в том случае, если пользователь часто предпочитает решения, которые не требуют лишних усилий. Добавление логина в имя файла и проверка его при входе в оффлайн-режиме может использовать этот принцип в качестве дополнительной меры защиты. Большинство пользователей будут интуитивно использовать свой обычный логин, не задумываясь о его точном написании или возможных вариациях, что осложняет попытки доступа к чужим данным. Такая проверка

добавляет уровень "ленивой безопасности", поскольку пользователям проще ввести свой привычный логин, чем пытаться обойти систему.

4. РЕАЛИЗАЦИЯ МОНИТОРИНГА СОСТОЯНИЯ СЕРВЕРОВ

Для эффективной балансировки количества игроков по серверам и распределения серверов по виртуальным машинам необходимо создать фундаментальную инфраструктуру мониторинга и управления. Такой подход требует реализации системы, которая будет отслеживать состояние серверов и виртуальных машин, а также обеспечивать их регистрацию в менеджере серверов. Эти шаги являются ключевыми для сбора данных о нагрузке, доступных ресурсах и состоянии элементов инфраструктуры, что позволит в дальнейшем принимать оптимальные решения для балансировки и масштабирования.

Для начала реализации такого требования было принято решение разделить обработку клиентских сокетов и сокетов серверов UE. Это обусловлено тем, что данные типы сокетов имеют различную логику обработки: клиентские сокет и серверные сокет обмениваются данными с разной интенсивностью и требуют специфических подходов к обработке сообщений. Такое разделение позволяет оптимизировать работу с каждым типом соединения и эффективно управлять нагрузкой (приложение А, листинг А.13).

Для пояснения кода, представленного в листинг А.13 для начала необходимо дать пояснения к принципам работы TCP-сокета, а именно долгоживущего сокета (long-lived socket). Его характерной чертой является то, что он сохраняет соединение на протяжении всей сессии, пока оно явно не разрывается одной из сторон (клиентом либо сервером). Этот подход позволяет избежать создания нового сокета для каждого сообщения, обеспечивая более эффективное взаимодействие и снижая накладные расходы на установление соединения.

Для начала передачи сообщений по протоколу TCP клиентская сторона устанавливает соединение с принимающей стороной. Как только соединение было установлено сервер записывает в память сокет клиента и в бесконечном цикле пытается прочитать сообщения из клиентского сокета. Пока попытки

чтения не прекращаются, клиентский сокет никогда не будет явно пытаться разорвать существующее соединение. Как только клиент присылает сообщение, сервер обрабатывает его в отдельном потоке и далее продолжает пытаться слушать следующие входящие сообщения. Если клиентский сокет явно разорвал соединение с сервером, то вызовется исключение типа *boost::system::system_error*, управление передается в блок *catch* и сервер явно разрывает соединение для двоих сторон выполнив функции *socket->shutdown(boost::asio::ip::tcp::socket::shutdown_both)* и *socket->close()*.

Для обработки команд, которые приходят от запущенного выделенного сервера был разработан протокол следующего вида: *<имя_команды>, <ключ1=значение1>, ..., <ключN=значениеN>*. Первым словом до разделителя, всегда идет тип команды, которую надо обработать. Далее через запятую идут пары ключ значения, которые являются опциональными и предназначены для передачи дополнительной информации к выполняемой команде. Пример команды, которая отправляется с выделенного сервера UE после его запуска для его регистрации в системе имеет вид

REGISTER_SERVER,uuid=ae34b65e4a45cd1a2,uri=127.0.0.1:7777,current_
players=1,max_players=10

Код функции обработки команд представлен в листинге А.14.

В команде приходят такие данные как *uuid* – уникальный идентификатор сервера, *uri* – адрес подключения, *current_players* – текущее количество игроков, *max_players* – максимально возможное количество игроков.

Также помимо регистрации сервера необходимо было обработать событие, когда меняется текущее количество игроков. В классе *AGameMode* есть виртуальный метод *virtual void PostLogin(APlayerController* NewPlayer)*, который вызывается каждый раз как к серверу подключается новый пользователь. Для реализации отправки команды был переопределен данный метод в производном классе и добавлена логика по отправке сообщения в менеджер с актуальным количеством текущих игроков на сервере.

Аналогичную логику необходимо было реализовать и тогда, когда пользователь отключается от сервера. Для реализации данной задачи был переопределен и использован метод *virtual void Logout(AController* Exiting)*, который выполняется каждый раз, как пользователь отключается от сервера. Пример логов менеджера представлен ниже на рисунке 17.

```
[2024-12-18 22:12:34] Got data from dedicated server: REGISTER_SERVER,
1:7777,current_players=0,max_players=10

Char '=' not found. Probably message command type: REGISTER_SERVER
[2024-12-18 22:12:34] Registering server job started...

Registered server with uuid = 1D6B4B5D4194BE7B571B349038436F24
Sending uri of started dedicated server to client: 127.0.0.1:7777
Senging data to cleint: 127.0.0.1:7777
[2024-12-18 22:12:34] Registering server job finished...

[2024-12-18 22:12:36] Got data from dedicated server: UPDATE_SERVER,uu
s=1

Char '=' not found. Probably message command type: UPDATE_SERVER
[2024-12-18 22:12:36] Updating server job started...

Old value: 0
New value: 1
[2024-12-18 22:12:36] Updating server job finished...

[2024-12-18 22:12:57] Got data from dedicated server: UPDATE_SERVER,uu
s=2

Char '=' not found. Probably message command type: UPDATE_SERVER
[2024-12-18 22:12:57] Updating server job started...

Old value: 1
New value: 2
[2024-12-18 22:12:57] Updating server job finished...
```

Рисунок 17 – Логи менеджера при выполнении команд

Анализируя рисунок 17, можно увидеть, что вначале сервер получил команду на регистрацию сервера и сохранение его в память. Далее к серверу подключались пользователи и началось выполнение команд *UPDATE_SERVER*, которое выводит как дополнительную информацию старое и новое значение изменившегося параметра.

В результате реализации была добавлена логика, которая обеспечивает установление долгоживущих соединений с серверами, их регистрацию в системе и обновление состояния в режиме реального времени. Такой подход позволяет эффективно управлять серверами и поддерживать актуальную информацию об их состоянии. Выбранное решение базируется на оптимальных принципах работы с сетевыми соединениями, что гарантирует стабильность и производительность системы. Такой подход также гарантирует, что любое количество серверов, которое подключилось к менеджеру, будет зарегистрировано в системе и обновляться в режиме реального времени.

Однако данный подход имеет и свои минусы: ограниченное количество потоков может стать узким местом при большом числе подключений, а также наблюдается повышенная нагрузка на ЦП из-за выделения отдельного потока для каждого соединения. Эти аспекты указывают на потенциал для дальнейшей оптимизации, например, через использование пула потоков или асинхронной обработки соединений. Такой подход позволит снизить нагрузку на процессор и увеличить масштабируемость системы.

ЗАКЛЮЧЕНИЕ

В ходе работы была разработана базовая версия системы автоматизированного запуска выделенных серверов UE, которая упрощает управление серверной инфраструктурой. Основой системы стала программа-демон, запускающая экземпляры серверов в автоматическом режиме, и менеджер серверов, который обрабатывает запросы от клиентов и взаимодействует с демоном.

Для обеспечения безопасности и удобства пользователей была реализована аутентификация через API транзакций EOS, включающая использование интерфейсов PurchaseInterface и StoreInterface. Это позволило организовать безопасный процесс аутентификации и управление покупками, предоставляя пользователям доступ к серверным услугам только при подтверждении их прав. Дополнительно был внедрен защищенный оффлайн доступ к данным о покупках, что дает пользователям возможность пользоваться системой даже при временных сбоях в подключении к сети.

Таким образом, разработанная система предоставляет функциональные возможности для запуска серверов в автоматизированном режиме, безопасно управляет доступом через аутентификацию и поддерживает оффлайн-режим, что делает её эффективным и надежным решением для многопользовательских приложений на базе UE.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Процесс разработки и тестирования демонов. Habr [Электронный ресурс] // Режим доступа: <https://habr.com/ru/companies/badoo/articles/254087/> (дата обращения 15.09.2024).
2. TCP Socket Listener, Receive Binary Data From an IP/Port into UE4. Unreal Community [Электронный ресурс] // Режим доступа: [https://unrealcommunity.wiki/tcp-socket-listener-receive-binary-data-from-an-ip/port-into-ue4-\(full-code-sample\)-1eefbvdk](https://unrealcommunity.wiki/tcp-socket-listener-receive-binary-data-from-an-ip/port-into-ue4-(full-code-sample)-1eefbvdk) (дата обращения 25.09.2024).
3. UE5 Multithreading With FRunnable And Threat Workflow. Algosyntax [Электронный ресурс] // Режим доступа: <https://store.algosyntax.com/tutorials/unreal-engine/ue5-multithreading-with-frunnable-and-thread-workflow/> (дата обращения 08.10.2024).
4. Habr. Первичный ключ – GUID [Электронный ресурс] // Режим доступа: <https://habr.com/ru/articles/265437/> (дата обращения 13.10.2024).
5. Best Practices for Using GUIDs. Microsoft Documentation [Электронный ресурс] // Режим доступа: <https://learn.microsoft.com/ru-ru/> (дата обращения 01.11.2024).

ПРИЛОЖЕНИЕ А

Листинг 1 – Создание сокета, принимающего входящие сообщения

```
void createAcceptThread()
{
    const unsigned int port = 8871;
    // Создание контекста
    boost::asio::io_context context;
    boost::asio::ip::tcp::endpoint endpoint(boost::asio::ip::tcp::v4(), port);

    // Инициализация сокета для прослушки входящих соединений
    boost::asio::ip::tcp::acceptor acceptor(context, endpoint);
    std::cout << "Start listening on 0.0.0.0:" << port << std::endl;

    while (true)
    {
        boost::this_thread::sleep(boost::posix_time::seconds(1));
        boost::shared_ptr<boost::asio::ip::tcp::socket> clientSocket(new
boost::asio::ip::tcp::socket(context));
        acceptor.accept(*clientSocket);
        boost::thread(boost::bind(handleIncomeQuery, clientSocket));
    }
}
```

Листинг 2 – Метод обработки входящих команд

```
void handleIncomeQuery(boost::shared_ptr<boost::asio::ip::tcp::socket> socket)
{
    bool bIsReading(true);
    while (bIsReading)
    {
        char data[512];
        size_t bytesRead = socket->read_some(boost::asio::buffer(data));
        if (bytesRead > 0)
        {
            const std::string message = std::string(data, bytesRead);
            std::cout << "Handle income query: " << message << std::endl;
            if (message == "Start")
            {
                std::cout << "Starting server instance..." << std::endl;
                boost::thread(startServerInstance).detach();
            }
            bIsReading = false;
        }
    }
}
```

Листинг 3 – Функция запуска экземпляра сервера

```
void startServerInstance()
{
    #ifdef _WIN32
        const std::string scriptPath =
"E:\\Master\\sem2\\MMAPS\\Releases\\WindowsServer\\Lab4ServerPackaged.bat";
    #else
        const std::string scriptPath = "/home/user/dedicated-
server/LinuxServer/Lab4Server.sh";
    #endif // _WIN32
    try
```

```

    {
        boost::process::child childThreat(scriptPath, boost::process::std_out >
stdout, boost::process::std_err > stderr);
        childThreat.wait();
        if (childThreat.exit_code() == 0)
        {
            std::cout << "Server instance started successfully!" << std::endl;
        }
        else
        {
            std::cerr << "Error, while starting server instance. Exit code: " <<
childThreat.exit_code() << std::endl;
        }
    }
    catch (const std::exception& exception)
    {
        std::cerr << "Exception occured, while starting server instance: " <<
exception.what() << std::endl;
    }
}

```

Листинг 4 – Программное получение порта, на котором запущен сервер

```

FString LocalNetworkAddress = NetDriver->LowLevelGetNetworkNumber();
if (LocalNetworkAddress.IsEmpty())
{
    UE_LOG(LogTemp, Error, TEXT("NetworkAddressString is empty"));
    return;
}
FString AdressString;
FString PortString;
if (!LocalNetworkAddress.Split(TEXT(":"), &AdressString, &PortString))
{
    UE_LOG(LogTemp, Error, TEXT("Failed to get port"));
    return;
}
int32 Port = FCString::Atoi(*PortString);
UE_LOG(LogTemp, Log, TEXT("Server is listening on port: %d"), Port);
SendUriToServerManager(Port);

```

Листинг 5 – Обработка адреса запущенного выделенного сервера

```

void TcpServer::ProcessDataFromDaemon(std::string& message)
{
    // Обработка присланного URI от DedicatedServer
    std::cout << "Got URI form started DedicatedServer: " << message <<
std::endl;

    auto initiatorConnectedClients = boost::adaptors::filter(_connectedClients,
[] (const ClientInfo& clientInfo)
    {
        return clientInfo.UserType == ClientType::INITIATOR;
    });

    if (initiatorConnectedClients.empty())
    {
        std::cout << "Connected clients queue is empty. No client to send
IP:PORT to" << std::endl;
        return;
    }

    ClientInfo& firstInitiatorInQueue = initiatorConnectedClients.front();

```

```

std::cout << "Senging data to cleint: " << message << std::endl;
SendDataToSocket(firstInitiatorInQueue.Socket, message);
_connectedClients.erase(_connectedClients.begin());
}

```

Листинг 6 — Работа с интерфейсом пользователя в дочернем потоке

```

const FString receivedStringData =
FromBinaryArrayToString(receivedData);

GameInstance->SetConnectAddress(receivedStringData);
UMatchmakingConnectWidget* matchmakingConnectWidget = GameInstance-
>MatchMakingConnectWidget;
AsyncTask(ENamedThreads::GameThread, [matchmakingConnectWidget]()
{
    matchmakingConnectWidget->AddToViewport();
});

```

Листинг 7 — Реализации транзакции с приобретением предложения

```

void ULab4GameInstance::StartPurchase()
{
    FUniqueNetIdPtr userUniqueId = IdentityPtr->GetUniquePlayerId(0);
    if (!userUniqueId.IsValid())
    {
        UE_LOG(LogTemp, Error, TEXT("Error, while purchasing item.
UnserUniqueId is invalid"))
        return;
    }

    if (!UserPurchaseInterface.IsValid())
    {
        UE_LOG(LogTemp, Error, TEXT("PurchaseInterface pointer is invalid"))
        return;
    }

    FPurchaseCheckoutRequest checkoutRequest = {};
    checkoutRequest.AddPurchaseOffer(TEXT("DedicatedMatchStart"), OfferId, 1);

    UserPurchaseInterface->Checkout(*userUniqueId, checkoutRequest,
FOnPurchaseCheckoutComplete::CreateLambda([this, userUniqueId](const
FOnlineError& Result, const TSharedRef<FPurchaseReceipt>& Receipt)
    {
        if (Result.bSucceeded)
        {
            GetUserReceipts(userUniqueId, true);
            return;
        }

        UE_LOG(LogTemp, Error, TEXT("Failed to complete checkout: %s"),
*(Result.ErrorRaw));
    }));
}

```

Листинг 8 – Проверка результатов транзакции

```
void ULab4GameInstance::GetUserReceipts(FUniqueNetIdPtr userUniqueId, bool
bShouldFinalize)
{
    TArray<FPurchaseReceipt> userReceipts;
    UserPurchaseInterface->GetReceipts(*userUniqueId, userReceipts);

    for (const FPurchaseReceipt& userReceipt : userReceipts)
    {
        UE_LOG(LogTemp, Log, TEXT("User receipt transaction Id: %s"),
*(userReceipt.TransactionId))
        if (userReceipt.TransactionState !=
EPurchaseTransactionState::Purchased)
        {
            UE_LOG(LogTemp, Error, TEXT("Transaction with ID: %s is not
purchased yet"), *(userReceipt.TransactionId))
            continue;
        }

        for (const FPurchaseReceipt::FReceiptOfferEntry& offerEntry :
userReceipt.ReceiptOffers)
        {
            UE_LOG(LogTemp, Log, TEXT("OfferId: %s, quantity: %d"),
*(offerEntry.OfferId), offerEntry.Quantity)

            if (!bCanStartDedicatedMatch && offerEntry.OfferId ==
OfferIdAudience)
            {
                bCanStartDedicatedMatch = true;
                SavePurchaseToFile(offerEntry.OfferId);
                UE_LOG(LogTemp, Log, TEXT("OfferId %s has been found.
Set bCanStartDedicatedMatch = true"), *OfferId)
            }
            if (!bShouldFinalize) continue;

            // Finalizing purchased transations
            for (const FPurchaseReceipt::FLineItemInfo& offerLineItem :
offerEntry.LineItems)
            {
                UE_LOG(LogTemp, Log, TEXT("OfferLineItem name: %s"),
*(offerLineItem.ItemName))
                FString InReceiptValidationInfo =
offerLineItem.ValidationInfo;

                if (InReceiptValidationInfo.IsEmpty())
                {
                    UE_LOG(LogTemp, Log, TEXT("OfferLineItem name has
empty validation info: %s. Skip finalizing"))
                    continue;
                }

                UserPurchaseInterface-
>FinalizeReceiptValidationInfo(*userUniqueId, InReceiptValidationInfo,
FOnFinalizeReceiptValidationInfoComplete::CreateLambda([userReceipt,
offerEntry](const FOnlineError& Result, const FString& ValidationInfo)
                {
                    if (Result.bSucceeded)
                    {
                        UE_LOG(LogTemp, Log, TEXT("Successfully
finilized purchase: TransactionId: %s, OfferId: %s"),
*(userReceipt.TransactionId), *(offerEntry.OfferId))
                    }
                })
            }
        }
    }
}
```

```

        return;
    }

    UE_LOG(LogTemp, Error, TEXT("Error, while
finalizing purchase: %s"), *(Result.ErrorRaw))
    ));
    }
}

if (bCanStartDedicatedMatch)
{
    m_pMainMenu->SetMatchmakingHintTextVisibility(false);
}
}

```

Листинг 9 – Управление запуском выделенного сервера через виджет

```

void UMainMenu::OnMatchmakingCreateButtonClicked()
{
    UE_LOG(LogTemp, Warning, TEXT("Create matchmaking button clicked"))
    ULab4GameInstance* gameInstance = GetGameInstance<ULab4GameInstance>();
    if (gameInstance == nullptr) return;

    if (gameInstance->GetIfCanStartDedicated())
    {
        gameInstance->CreateSocketConnection();
        gameInstance->InitializeReceiveSocketThread();
        gameInstance-
>SendMessageToHostSocket(FString::Printf(TEXT("CREATE")));

        return;
    }
    if (gameInstance->GetIsLanGame() && !gameInstance-
>GetIfCanStartDedicated())
    {
        gameInstance->MatchmakingInputWidget->AddToViewport();
        return;
    }

    UE_LOG(LogTemp, Log, TEXT("Starting purchasing process from main menu"))
    gameInstance->StartPurchase();
}

```

Листинг 10 – Генерация уникального идентификатора приложения

```

void ULab4GameInstance::GenerateGuidAndSave()
{
    FString appGuidString = FGuid::NewGuid().ToString();
    UE_LOG(LogTemp, Log, TEXT("Generated app GUID"))

    SaveBase64EncodedData(appGuidString,
FPaths::Combine(FPaths::ProjectSavedDir(), TEXT("AppData"),
TEXT("AppGUID.dat")));
}

void ULab4GameInstance::SaveBase64EncodedData(const FString& Data, const
FString& FilePath)
{
    FString encodedData = FBase64::Encode(Data);
    FString filePath = FilePath;
}

```

```

        FFileHelper::SaveStringToFile(encodedData, *filePath);
        UE_LOG(LogTemp, Log, TEXT("Saved encoded GUID to directory"))
    }
}

```

Листинг 11 – Сохранение информации о транзакции в отдельный файл

```

void ULab4GameInstance::SavePurchaseToFile(const FString& PurchaseOfferId)
{
    if (!CheckIfGuidExists())
    {
        UE_LOG(LogTemp, Error, TEXT("App GUID not found"))
        return;
    }

    FString playerLogin = GetPlayerName();
    FString offerIdHash = GetSHA256Hash(PurchaseOfferId);
    FString appGuid =
    LoadBase64EncodedData(FPaths::Combine(FPaths::ProjectSavedDir(),
    TEXT("AppData"), TEXT("AppGUID.dat")));
    FString fileName = offerIdHash + TEXT("_") + playerLogin + TEXT(".dat");
    FString filePath = FPaths::Combine(FPaths::ProjectSavedDir(),
    TEXT("Purchases"), fileName);

    if (FPlatformFileManager::Get().GetPlatformFile().FileExists(*filePath))
    {
        UE_LOG(LogTemp, Log, TEXT("Offer with ID: %s already exists, hash:
%s. Cancel saving data..."), *PurchaseOfferId, *offerIdHash);
        return;
    }

    FString fileContent = FString::Printf(TEXT("AppGUID=%s\nOfferId=%s"),
    *appGuid, *PurchaseOfferId);
    SaveBase64EncodedData(fileContent, filePath);
    UE_LOG(LogTemp, Log, TEXT("Saved information about offer: [OfferId=%s,
UserLogin=%s]"), *PurchaseOfferId, *playerLogin);
}

```

Листинг 12 – Чтение файлов с информацией о транзакции

```

void ULab4GameInstance::SavePurchaseToFile(const FString& PurchaseOfferId)
{
    if (!CheckIfGuidExists())
    {
        UE_LOG(LogTemp, Error, TEXT("App GUID not found"))
        return;
    }

    FString playerLogin = GetPlayerName();
    FString offerIdHash = GetSHA256Hash(PurchaseOfferId);
    FString appGuid =
    LoadBase64EncodedData(FPaths::Combine(FPaths::ProjectSavedDir(),
    TEXT("AppData"), TEXT("AppGUID.dat")));
    FString fileName = offerIdHash + TEXT("_") + playerLogin + TEXT(".dat");
    FString filePath = FPaths::Combine(FPaths::ProjectSavedDir(),
    TEXT("Purchases"), fileName);

    if (FPlatformFileManager::Get().GetPlatformFile().FileExists(*filePath))
    {
        UE_LOG(LogTemp, Log, TEXT("Offer with ID: %s already exists, hash:
%s. Cancel saving data..."), *PurchaseOfferId, *offerIdHash);
    }
}

```



```

        return;
    }

    FString fileContent = FString::Printf(TEXT("AppGUID=%s\nOfferId=%s"),
*appGuid, *PurchaseOfferId);
    SaveBase64EncodedData(fileContent, filePath);
    UE_LOG(LogTemp, Log, TEXT("Saved information about offer: [OfferId=%s,
UserLogin=%s]"), *PurchaseOfferId, *playerLogin);
}

```

Листинг 13 – Обработка данных из сокета выделенного сервера Unreal Engine

```

void
TcpServer::ReadDataFromServerSocket(boost::shared_ptr<boost::asio::ip::tcp::socket> socket)
{
    try
    {
        while (true)
        {
            char data[512];

            size_t bytesRead = socket->read_some(boost::asio::buffer(data));
            if (bytesRead > 0)
            {
                std::string message = std::string(data, bytesRead);
                ProcessDataFromServer(message, socket);
            }
        }
    }
    catch (const boost::system::system_error& error)
    {
        if (error.code() == boost::asio::error::eof || error.code() ==
boost::asio::error::connection_reset)
        {
            std::cout << "Finish reading from client socket, client disconnected
cleanly" << std::endl;
        }
        else
        {
            std::cout << "Socket error occured: " << error.what() << std::endl;
        }
    }

    socket->shutdown(boost::asio::ip::tcp::socket::shutdown_both);
    socket->close();
    std::cout << "Client socket closed" << std::endl;
}

```

Листинг 14 – Обработка входящих команд по установленному протоколу

```

void TcpServer::ProcessDataFromServer(std::string& message,
boost::shared_ptr<boost::asio::ip::tcp::socket> socket)
{
    Logger::GetInstance() << "Got data from dedicated server: " << message <<
std::endl;

    std::string commandType =
CommandsHelper::GetCommandTypeFromMessage(message);
    std::unordered_map<std::string, std::string> commandKeyValuePairs =
CommandsHelper::GetKeyValuePairs(message);

    if (commandType == "REGISTER_SERVER")
    {

```

```

        Logger::GetInstance() << "Registering server job started..." <<
std::endl;
        ServerInfo newServer;

        newServer.Uuid = commandKeyValuePairs["uuid"];
        newServer.Uri = commandKeyValuePairs["uri"];
        newServer.MaxPlayers =
atoi(commandKeyValuePairs["max_players"].c_str());
        newServer.CurrentPlayers =
atoi(commandKeyValuePairs["current_players"].c_str());

        _runningServers.insert(_runningServers.begin(), newServer);
        std::cout << "Registered server with uuid = " << newServer.Uuid <<
std::endl;
        SendConnectionStringToClient(newServer.Uri);

        Logger::GetInstance() << "Registering server job finished..." <<
std::endl;
    }
    if (commandType == "UPDATE_SERVER")
    {
        Logger::GetInstance() << "Updating server job started..." << std::endl;
        std::string updatedInstanceUuid = commandKeyValuePairs["uuid"];

        auto registeredServer = boost::find_if(_runningServers,
[&updatedInstanceUuid](const ServerInfo& serverInfo)
        {
            return serverInfo.Uuid == updatedInstanceUuid;
        });

        if (registeredServer == _runningServers.end())
        {
            std::cout << "Server with such uuid: " << updatedInstanceUuid << "
not found" << std::endl;
            Logger::GetInstance() << "Updating server job finished..." <<
std::endl;
            return;
        }

        ServerInfo& foundRegisteredServer = *registeredServer;
        unsigned int currentPlayers =
atoi(commandKeyValuePairs["current_players"].c_str());

        std::cout << "Old value: " << foundRegisteredServer.CurrentPlayers <<
std::endl;
        foundRegisteredServer.CurrentPlayers = currentPlayers;
        std::cout << "New value: " << foundRegisteredServer.CurrentPlayers <<
std::endl;

        Logger::GetInstance() << "Updating server job finished..." << std::endl;
    }
}

```