|  |  |
| --- | --- |
|  | **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 НАУЧНО-ИСССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ***

***НА ТЕМУ***

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | РК6-31М | |  |  | Д. В. Боженко |
|  | (Группа) | |  | (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 научно-исследовательской работы**

по теме: *Разработка системы автоматизированного запуска выделенных серверов Unreal   
Engine 4*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_РК6-31М\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Боженко Дмитрий Владимирович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.) \_учебная\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

***Техническое задание:* 1.** Исследовать возможные методы автоматизации запуска выделенного сервера Unreal Engine, выбрать оптимальный метод и реализовать его, предоставив обоснование принятого решения.

**2.** Проанализировать доступные методы аутентификации для запуска выделенных серверов, выбрать наиболее подходящий и реализовать его.

**3**. Исследовать доступные методы хранения данных о транзакциях на клиентской стороне, выбрать оптимальный метод и реализовать его, обосновав выбор.

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на \_\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «15» сентября 2024 г.

|  |  |  |
| --- | --- | --- |
| **Руководитель НИР** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Витюков Ф.А.** |
|  | (Подпись, дата) | И.О. Фамилия |
| **Студент** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Боженко Д. В.** |
|  | (Подпись, дата) | И.О. Фамилия |

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

**СОДЕРЖАНИЕ**

[Перечень сокращений и условных обозначений 4](#_Toc182251090)

[Введение 5](#_Toc182251091)

[1. Автоматизация запуска выделенного сервера 6](#_Toc182251092)

[1.1. Реализация демона 6](#_Toc182251093)

[Заключение 8](#_Toc182251094)

[Список использованной литературы 9](#_Toc182251095)

[Приложение 10](#_Toc182251096)

# 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.

## Программная реализация демона

Демон – это программа, которая запускается в фоновом режиме и выполняет поставленные ей задачи. 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. 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 происходит 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 *сlientSocket*, создаваемого для каждого входящего подключения (приложение А, листинг А.2).

В данном листинге можно увидеть пример, что при успешном чтении данных из *сlientSocket* и получении команды *“START”,* вызывается функция *startServerInstance*. Важно отметить, что вызов функции является блокирующей операций. 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.

## 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*.

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>.*

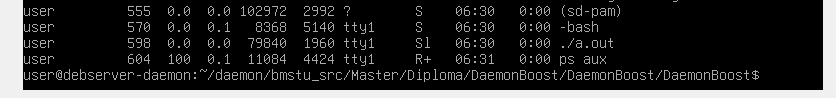


Рисунок 2 Просмотр работающего процесса через команду *ps aux*

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*. С помощью команды *netstat -tuln* проверялось, что сервис запущен, слушал входящие соединения и был доступен по сетевому интерфейсу 0.0.0.0.



Рисунок 3 Проверка всех сервисов, прослушивающих входящие соединения по протоколам *TCP/UDP*

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 севера, 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 результаты обработки двух команд “START” удаленно были запущены два экземпляра выделенных серверов, как показано на рисунке ниже.



Рисунок 4 Список запущенных выделенных серверов UE

На рисунке 4 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 c 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.

## 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 5).



Рисунок 5 А) Схема обратного сетевого взаимодействия напрямую с клиентом. Б) Схема обратного сетевого взаимодействия через менеджера серверов

Анализируя рисунок 5 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 АGameMode, 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 отсылает 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 сетевого взаимодействия клиента-инициатора с системой (рисунок 6).

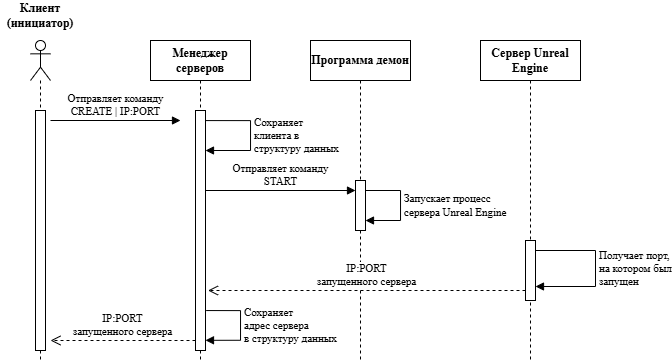


Рисунок 6 UML-диаграмма последовательности взаимодействия клиента-инициатора с системой

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 уделенной машине серверу (рисунок 7).

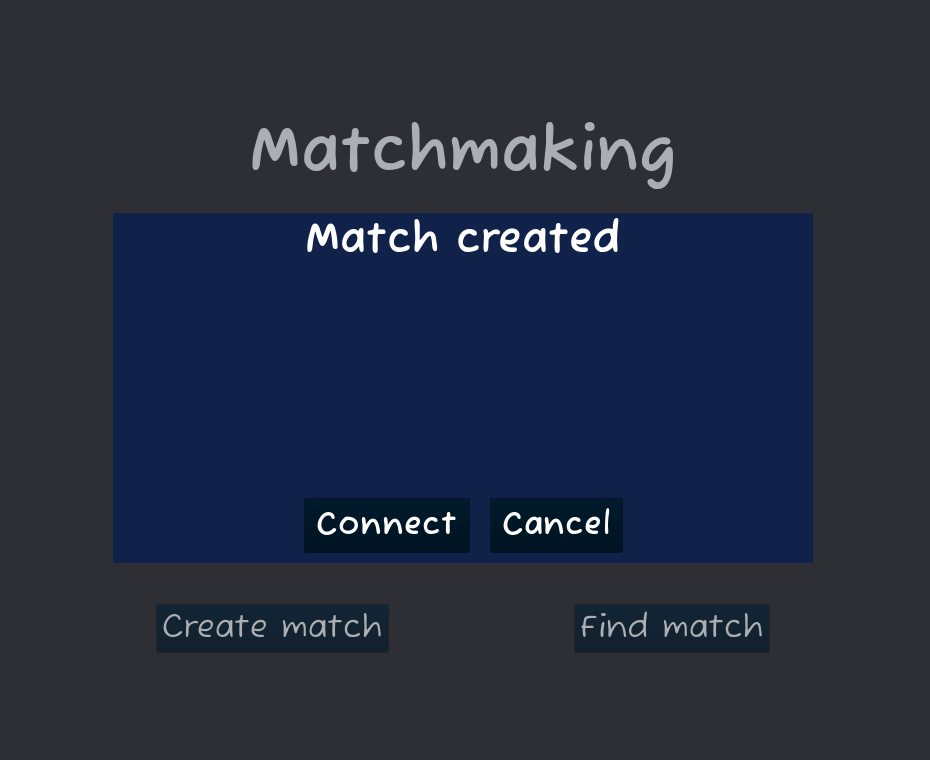


Рисунок 7 Виджет управления подключением на клиенте

# Аутентификация 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 интегрируется 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, булевая 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 8).

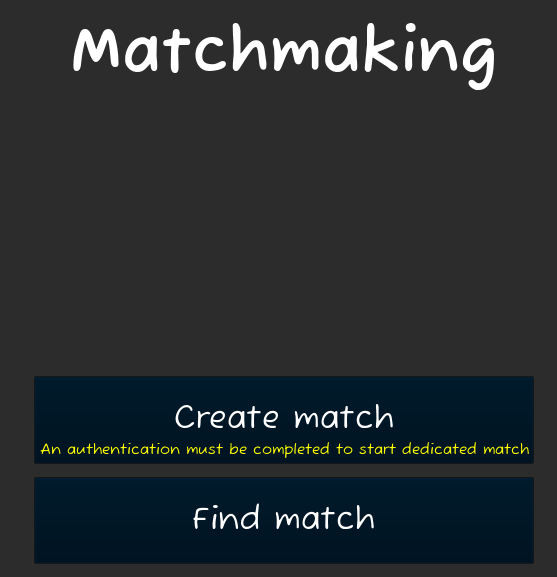


Рисунок 8 Виджет управления запуском выделенных серверов

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 9).

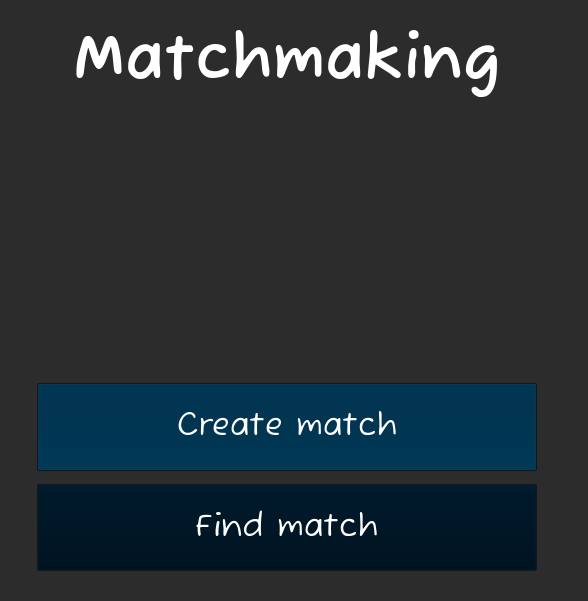


Рисунок 9 Состояние управляющего виджета после успешного проведения транзакции

# 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 использованной литературы

1. Cross-Compiling for Linux. Unreal Engine Documentation [Электронный ресурс] // Режим доступа: <https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/Linux/GettingStarted/> (дата обращения 15.02.2024);
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> (дата обращения 20.03.2024);
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 25.03.2024);
4. Boost.Asio. Boost C++ Libraries [Электронный ресурс] // Режим доступа: <https://www.boost.org/doc/libs/1_76_0/doc/html/boost_asio.html> (дата обращения 10.04.2024);
5. «Boost.Asio C++ Network Programming». Habr [Электронный ресурс] // Режим доступа: <https://habr.com/ru/articles/195794/> (дата обращения 15.04.2024);

# 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();

}