Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инновационного непрерывного образования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

*К защите допустить*:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
09.07.2025

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему

# Разработка приложения управления организацией

БГУИР КР 1-40 05 01-10 № 193 ПЗ

Студент (подпись студента) С.В.Гальчинский

Курсовая работа
представлена на проверку
09.07.2025
_____
(подпись студента)

2025

# Реферат

БГУИР КР 1-40 05 01-10 № 193 ПЗ, гр. 894351

С.В.Гальчинский, Разработка приложения управления организацией, Минск: БГУИР - 2025.

Пояснительная записка 63393 с., 22 рис., 0 табл.

Ключевые слова: Игры, JavaScript, Node.js, React, Webpack, Babel

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

-

-

# Содержание



# Введение

В апреле 2015 года вышла видеоигра Grand Theft Auto V на ПК. Она получила оценку 10/10 от большинства игровых изданий, тираж GTA V превысил 75 миллионов копий. Интересно, что в 2016 году игра продалась даже большим тиражом, чем в 2015. Это говорит о том, что популярность GTA 5 с годами не падает, а только растет. И в начале января 2018 года появляется RAGE Multiplayer – автономный мультиплеер для Grand Theft Auto V, позволяющий размещать частные серверы с пользовательскими режимами игры по вашему выбору. Также в этот мультиплеер встроен Chromium Embedded Framework (далее CEF) — открытый фреймворк для встраивания в приложение браузерного движка из проекта Chromium. CEF позволяет разработчику добавлять в приложение элементы браузера. Актуальность данного курсового проекта обусловлена тем, что в 2020 году данная модификация для GTA V занимает первое место по просмотрам на Twitch - видеостриминговый сервис, специализирующийся на тематике компьютерных игр. Это говорит о том, что игрокам интересна данная модификация и большое количество разработчиков начинает разрабатывать сервера на основе RAGE Multiplayer. Целью проекта является создание приложения управлениями организациями для сервера на RAGE Multiplayer и изучение возможностей языка JS в Node.js. В процессе выполнения работы, мы выполним постановку задачи, изучим возможности языка JavaScript, и создадим React-приложение. Для создания приложения требуется следующее: • Сервер на RAGE Multiplayer • База данных с информацией об игроках и организациях • Создание пользовательского интерфейса • Создание классов организации и ее участника • Обработка запросов клиент-сервер / сервер-клиент • Создание системы контроля доступа с использованием битовой маски С точки зрения внутреннего устройства, необходимо использовать API RAGE Multiplayer для связи клиента и сервера, и создания браузера CEF. Для хранения информации об игроках и организации создадим реляционную базу данных и воспользуемся специальным языком SQL. Для создания приложения, будем использовать среду

разработки VS Code для JavaScript. Для разработки пользовательского интерфейса будем использовать React, серверная часть использует Node.js.

# 1 Описание проекта

**Сервер на RAGE Multiplayer**

Для создания приложения я использую собственный сервер на RAGE Multiplayer (рис.1) с аутентификацией и последующей авторизацией.

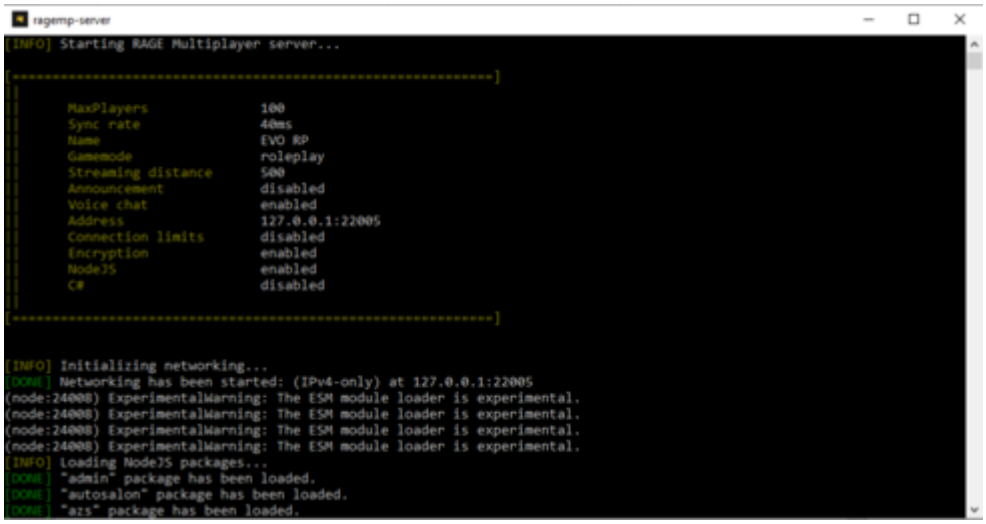

Рисунок 1 - Запуск сервера в консоли Windows

Чтобы воспользоваться приложением, нужно зайти на сервер с помощью клиента RAGE Multiplayer и авторизоваться (рис.2). После авторизации игрок выбирает персонажа, который появится в мире GTA V (рис.3). Когда он загрузится, появится возможность открыть приложение для управления организациями, нажав клавишу F3.

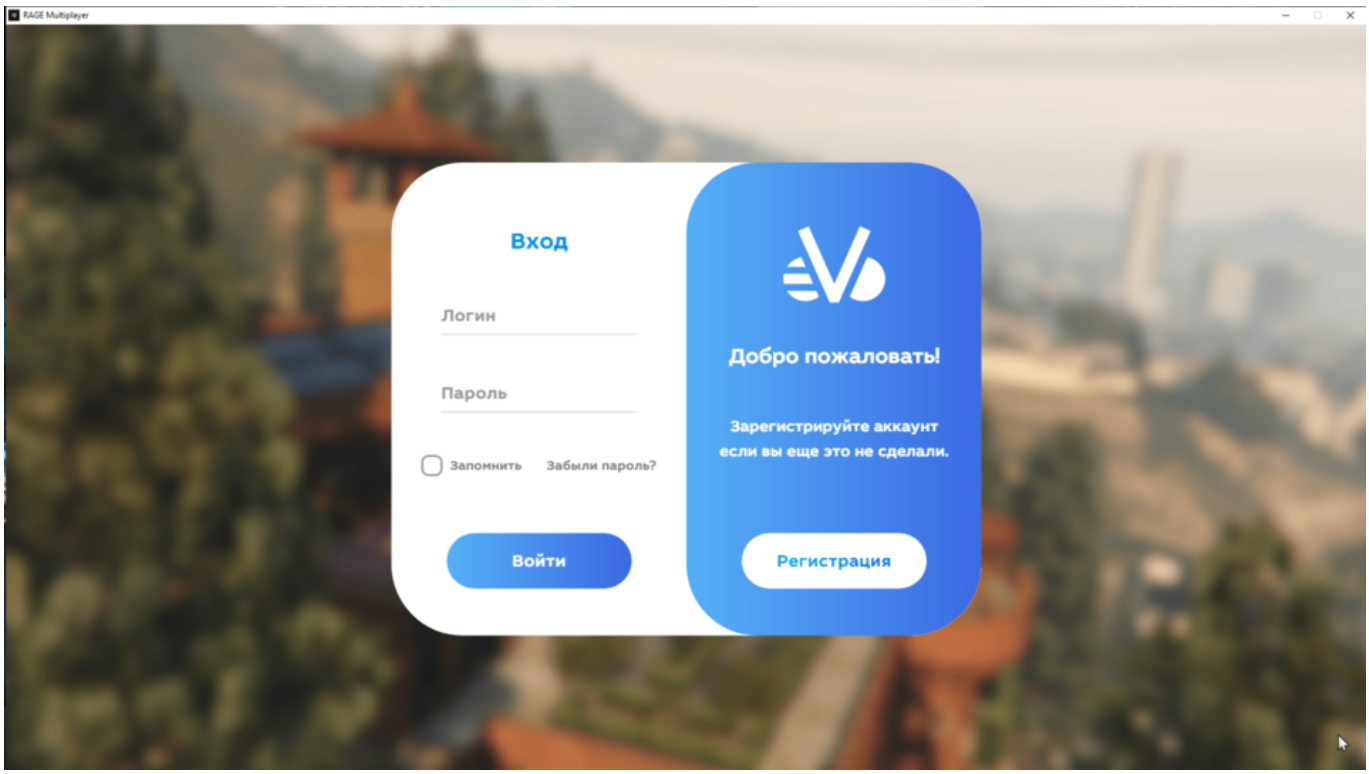

Рисунок 2 - Окно авторизации

Рисунок 3 - Персонаж в мире GTA V

**Приложение управления организацией**

После обработки события нажатия клавиши F3 с клиента идет запрос на сервер на список организаций, в которых состоит персонаж, и доступ к приложению управления этих организаций. Если персонаж не имеет доступа или не состоит в организациях, то сервер вернет отрицательный ответ. Если персонаж имеет доступ к более, чем одной организации, то сервер предлагает клиенту выбрать нужную организацию. Клиент, в свою очередь, открывает выбор организации с помощью CEF (рис.4).

Рисунок 4 – Выбор организации

Когда персонаж выберет организацию или персонаж имеет доступ только к одной организации, на клиенте открывается основное меню (рис.5) приложения с навигацией и статистикой по данной организации. В этом меню можно узнать количество транспорта, ролей и участников, а также открыть меню заказов, налогов и модификаций. В навигации можно перейти на следующие страницы: роли, участники, магазин, имущество и логи.

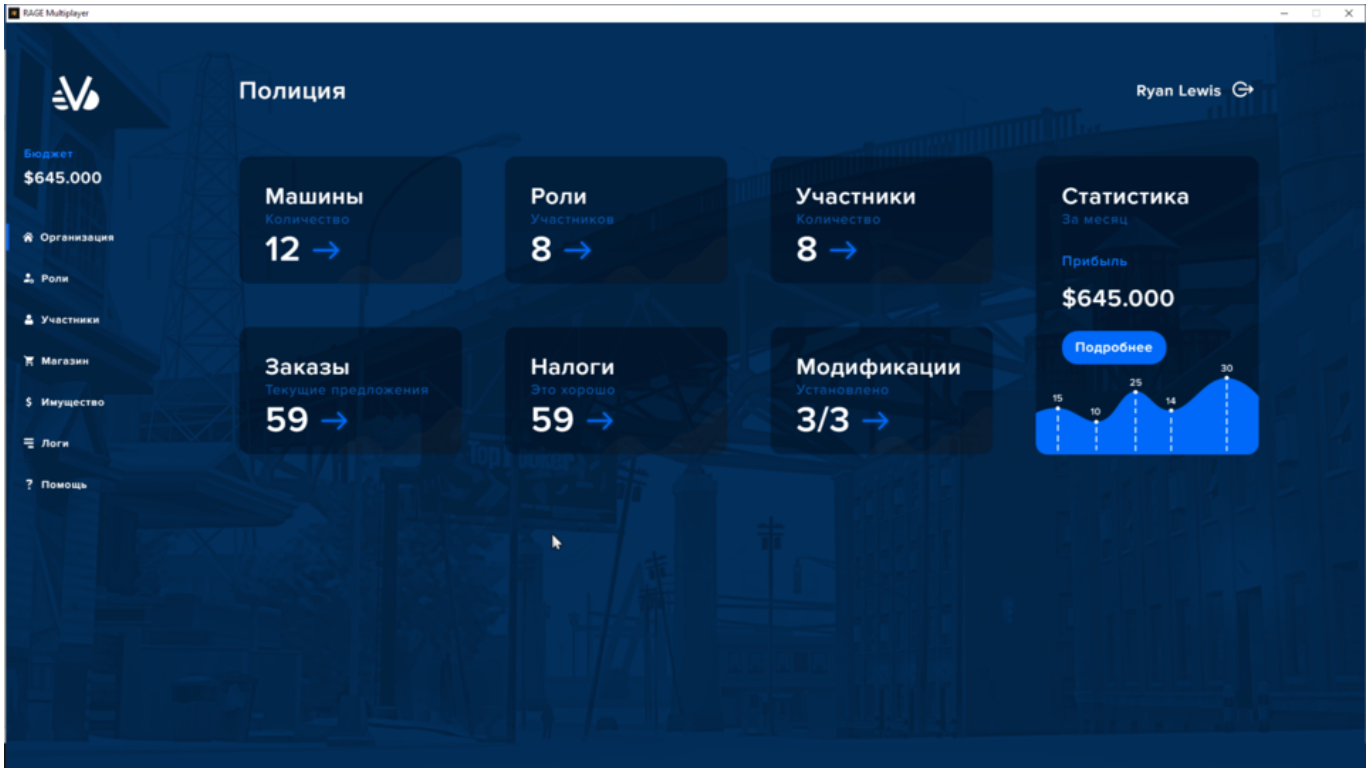


Рисунок 5 – Основное меню

В меню заказов (рис.6) можно посмотреть таблицу с историей всех заказов, а также отследить статус текущих. Таблицу можно сортировать по типу товара и по статусу доставки.

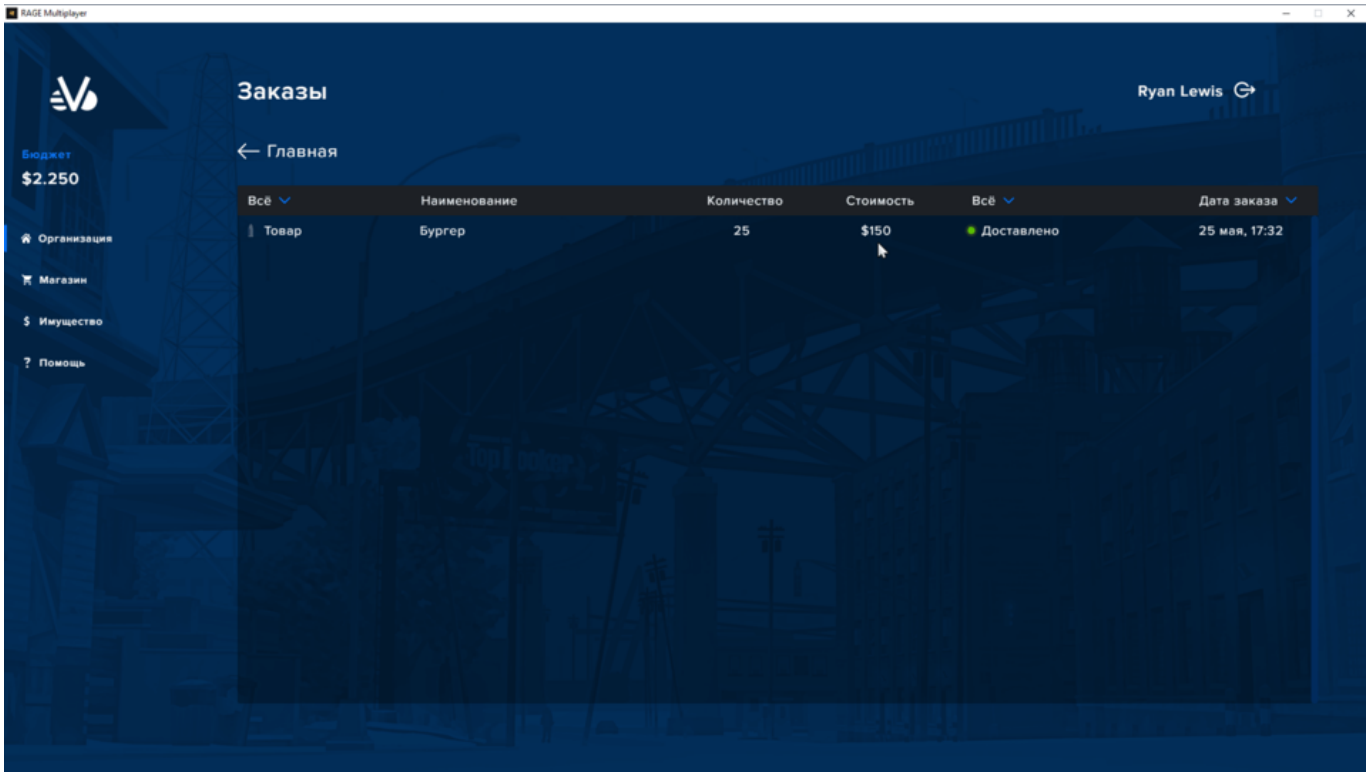

Рисунок 6 – Меню заказов

В меню налогов (рис.7) персонажу нужно оплатить налоги, если они имеются. Оплачивать налоги должен владелец бизнеса (магазин, АЗС). Также в этом меню можно посмотреть историю платежей.

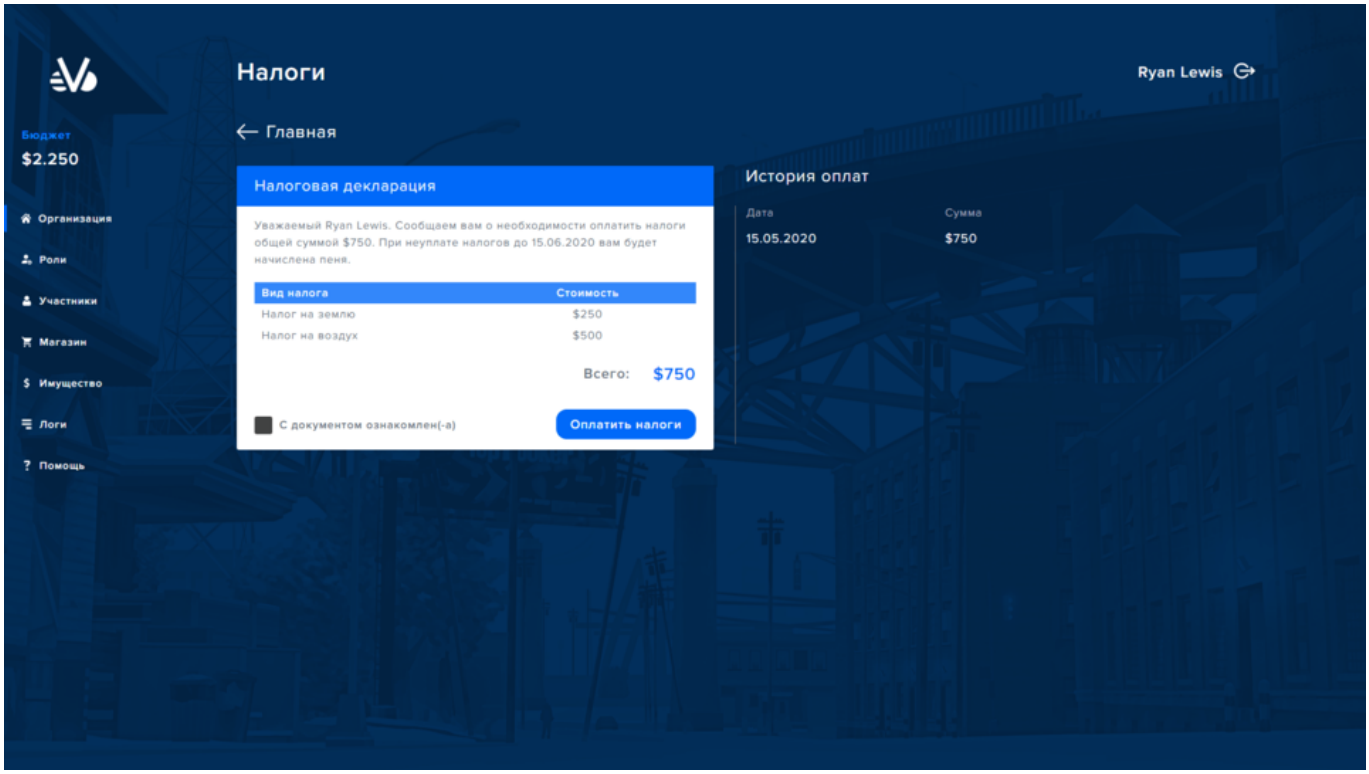

Рисунок 7 – Меню налогов

В меню модификаций (рис.8) персонаж может купить улучшения для своей организации. К примеру, для магазина персонаж может купить улучшенный сейф для того, чтобы уменьшить

шанс взлома грабителями. После покупки улучшения, оно добавляется в организацию на сервере.

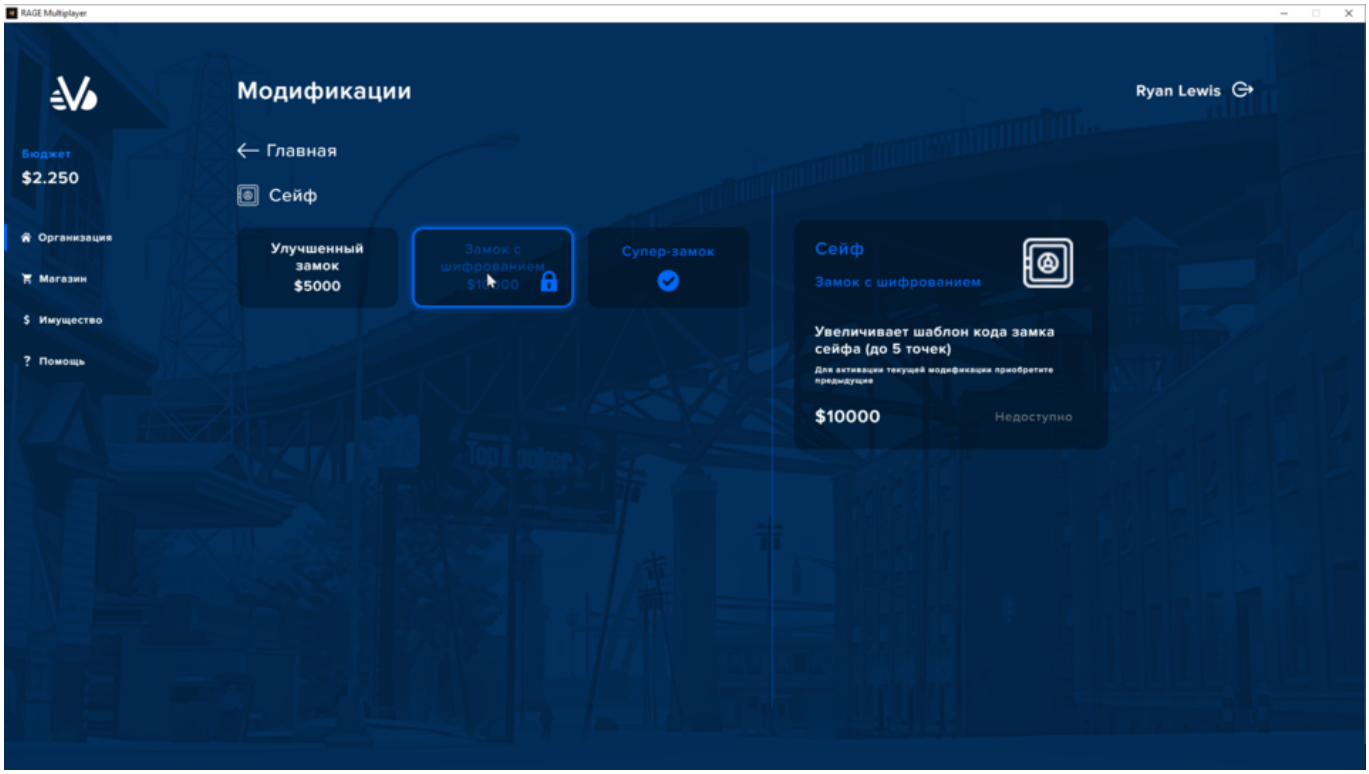


Рисунок 8 – Меню модификаций

В меню управления ролями (рис.9) можно изменить права доступа к определенной роли, добавить новую роль или удалить уже существующую. Пользователь может производить действия над ролями, только если имеет на это права и его роль выше (имеет больше прав), чем роль, над которой производятся действия.

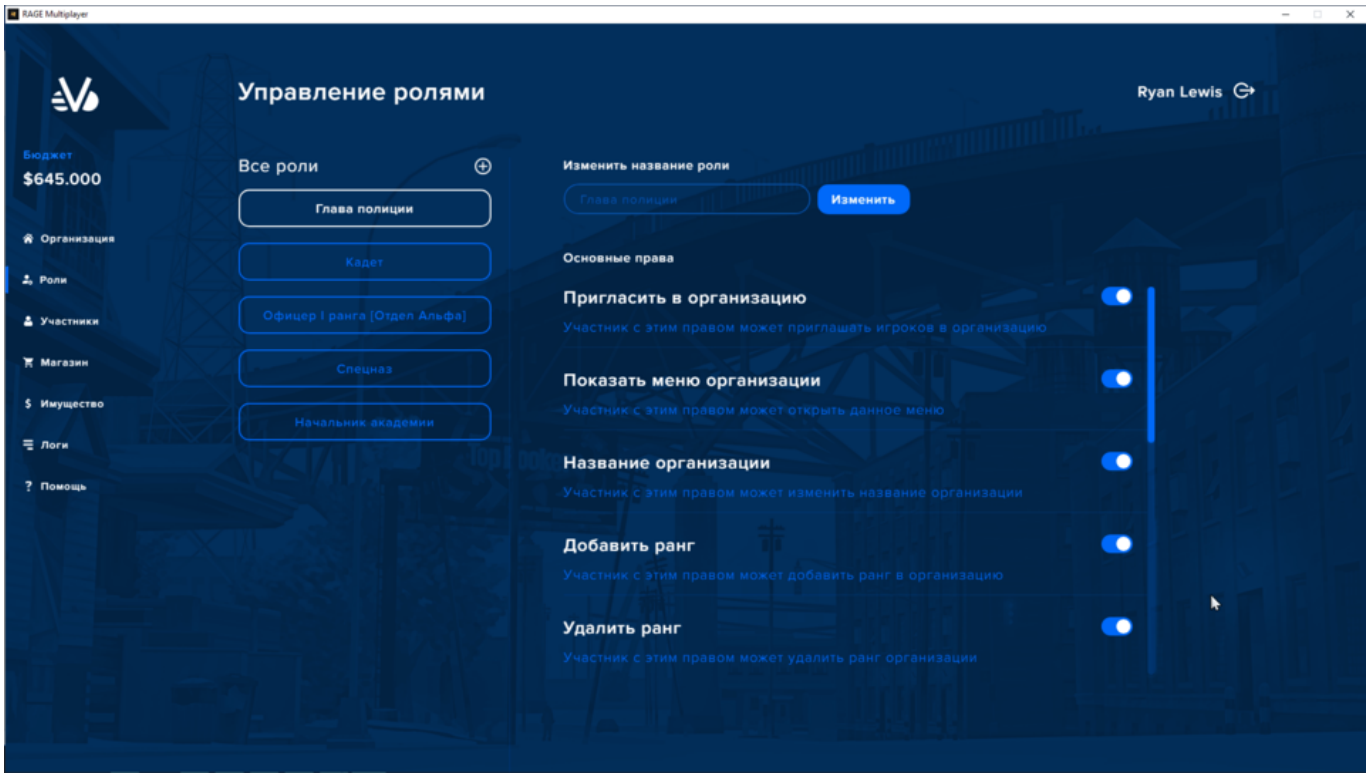


Рисунок 9 – Меню управления ролями

Права доступа реализуются с помощью битовой маски. Битовой маской называют комбинацию двоичных значений, которая используется для проверки и выборки единиц на нужных позициях. После изменений или добавления роли, данные о доступах хранятся в таблице базы данных в десятичном виде (рис.10).

| id | org_id | name | mask |
|---|---|---|---|
| 1 | 1 | Лидер фермы | 127 |
| 2 | 1 | Сборщик урожая | 1 |
| 3 | 1 | Сборщик урожая 2 | 1 |
| 4 | 1 | Водитель трактора | 1 |
| 5 | 1 | Водитель фургона | 1 |
| 6 | 1 | Пилот кукурузника | 1 |
| 9 | 4 | Глава полиции | 2047 |
| 10 | 4 | Кадет | 895 |

Рисунок 10 – Таблица organization_rank

В меню участников (рис.11) можно узнать кто состоит в организации, находится ли персонаж в сети, изменить роли персонажа или удалить его из организации, если есть права на эти действия. Список участников представляет собой таблицу, которую можно отсортировать по всем полям (имя, роли, статус).

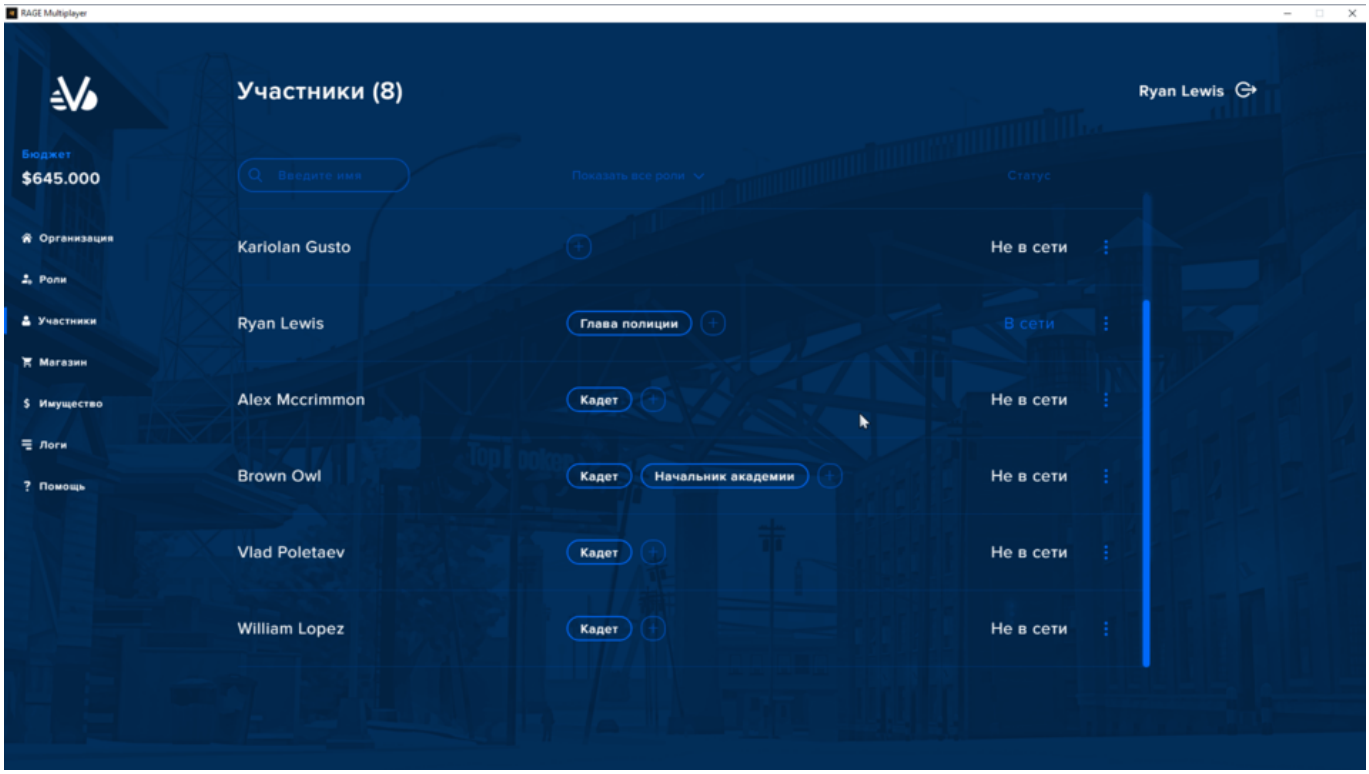


Рисунок 11 – Меню участников

Меню магазина делится на три категории:

Магазин транспорта
Магазин недвижимости
Магазин товаров

В магазине транспорта (рис.12) можно купить транспорт для организации. У каждой организации свой уникальный транспорт, характеристики и изображение которого можно посмотреть в карточке т/с. Чтобы купить транспорт, у персонажа должен быть доступ к этому действию, а также у организации должно хватать денежных средств и парковочных мест. Если какое-либо из условий не соблюдается, то сервер возвращает ответ на клиент с кодом ошибки.

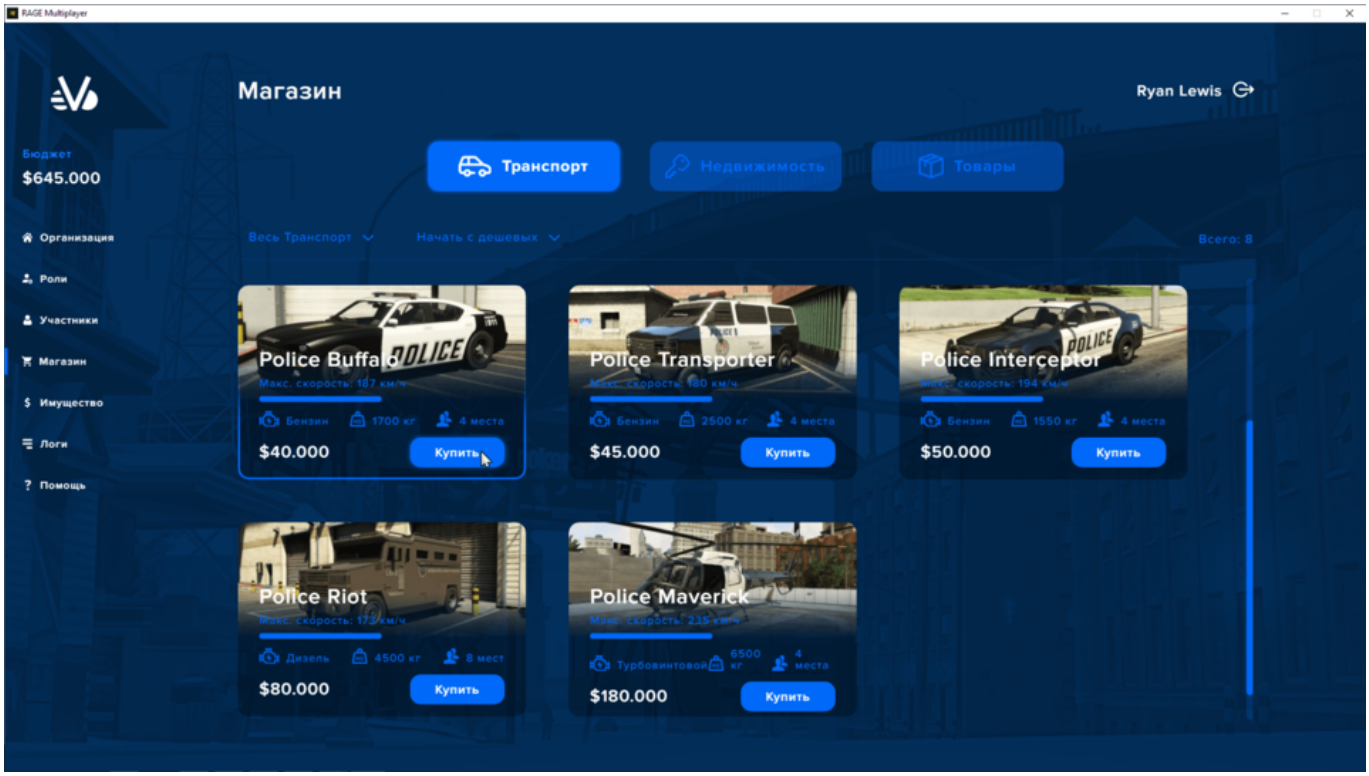


Рисунок 12 – Магазин транспорта

В магазине недвижимости (рис.13) можно купить склады для своей организации. В зависимости от вида организации, представлена разная недвижимость. К примеру, для магазинов можно купить склад для хранения товаров. В карточке недвижимости есть изображение и характеристика с указанием местоположения. Также как и с транспортом, если какое-либо из условий не соблюдается, то сервер возвращает ответ на клиент с кодом ошибки.

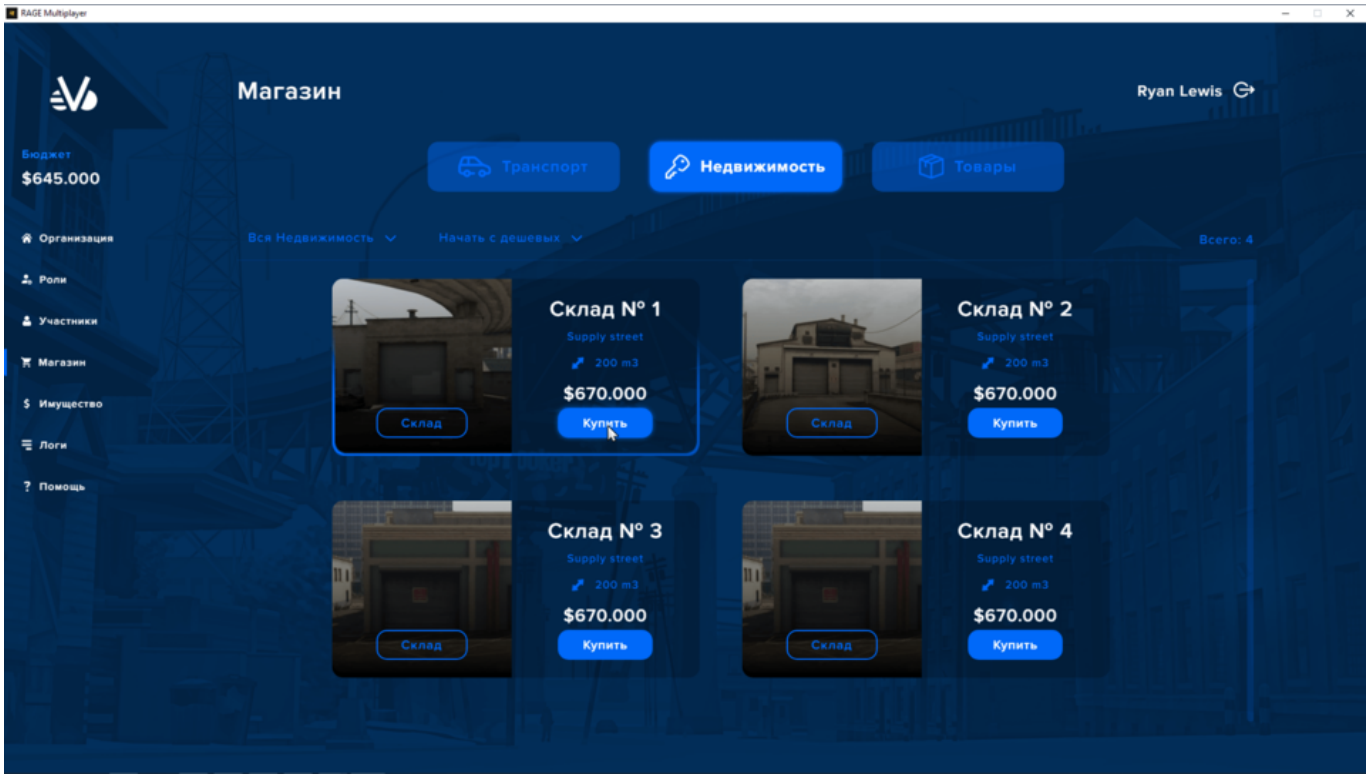

Рисунок 13 – Магазин недвижимости

В магазине товаров (рис.14) можно купить товары для продажи или использования внутри организации. Также можно перекупать товары у других организаций. При покупке товара, нужно выбрать куда его доставить и каким способом. После подтверждения оплаты можно отследить статус товара в меню заказов (рис.6).

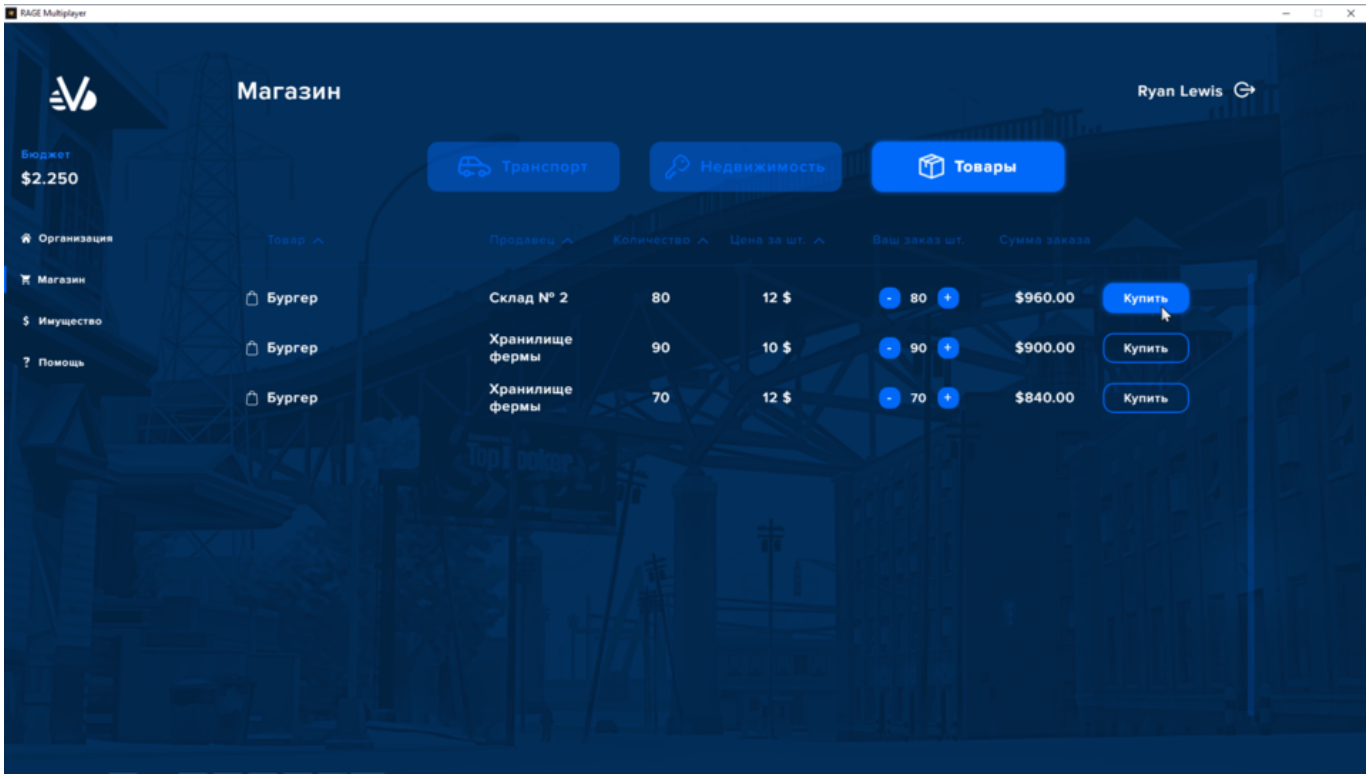

Рисунок 14 – Магазин товаров

В меню имущества также, как и в меню магазина, есть три категории: транспорт,

недвижимость и товары, которые делятся на товары на складе и товары на продаже.

В категории транспорта (рис.15) персонаж может просмотреть транспортное средство, которым владеет организация, а также отремонтировать его и установить модификации, если имеет права доступа на эти действия.

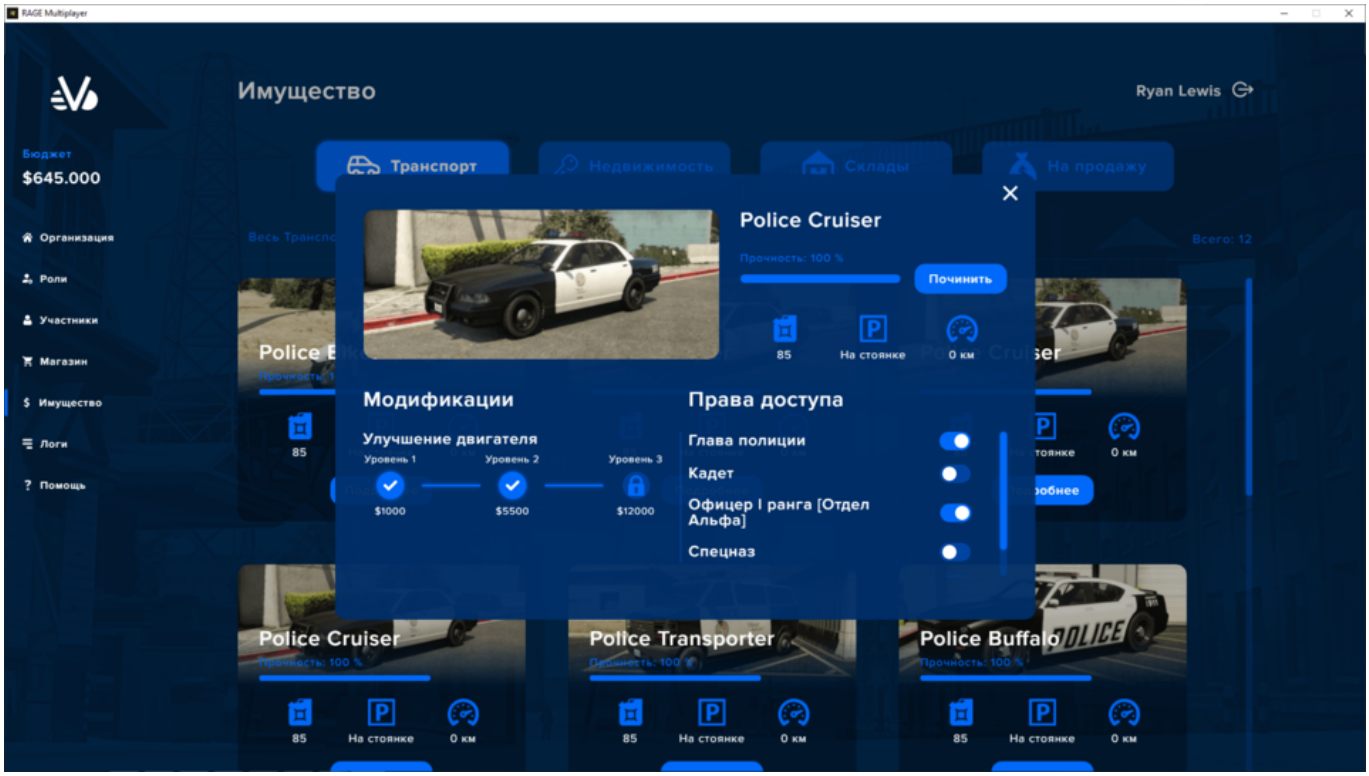


Рисунок 15 – Карточка т/с в меню имущества

В категории недвижимости персонаж может просмотреть чем владеет организация, а также проложить маршрут до каждого объекта.

В категории товаров на складе находятся все товары, которые есть в организации, включая товары на продаже. В этом меню можно выставить товары на продажу, если организация имеет права на продажу товара и персонаж имеет доступ к этому действию. При выставлении товара на продажу нужно выбрать количество и стоимость одной единицы.

В категории товаров на продаже (рис.16) находятся все товары, которые выставлены на продажу. В этом меню можно вернуть товары на склад, т.е. отменить продажу или вернуть только часть товаров.

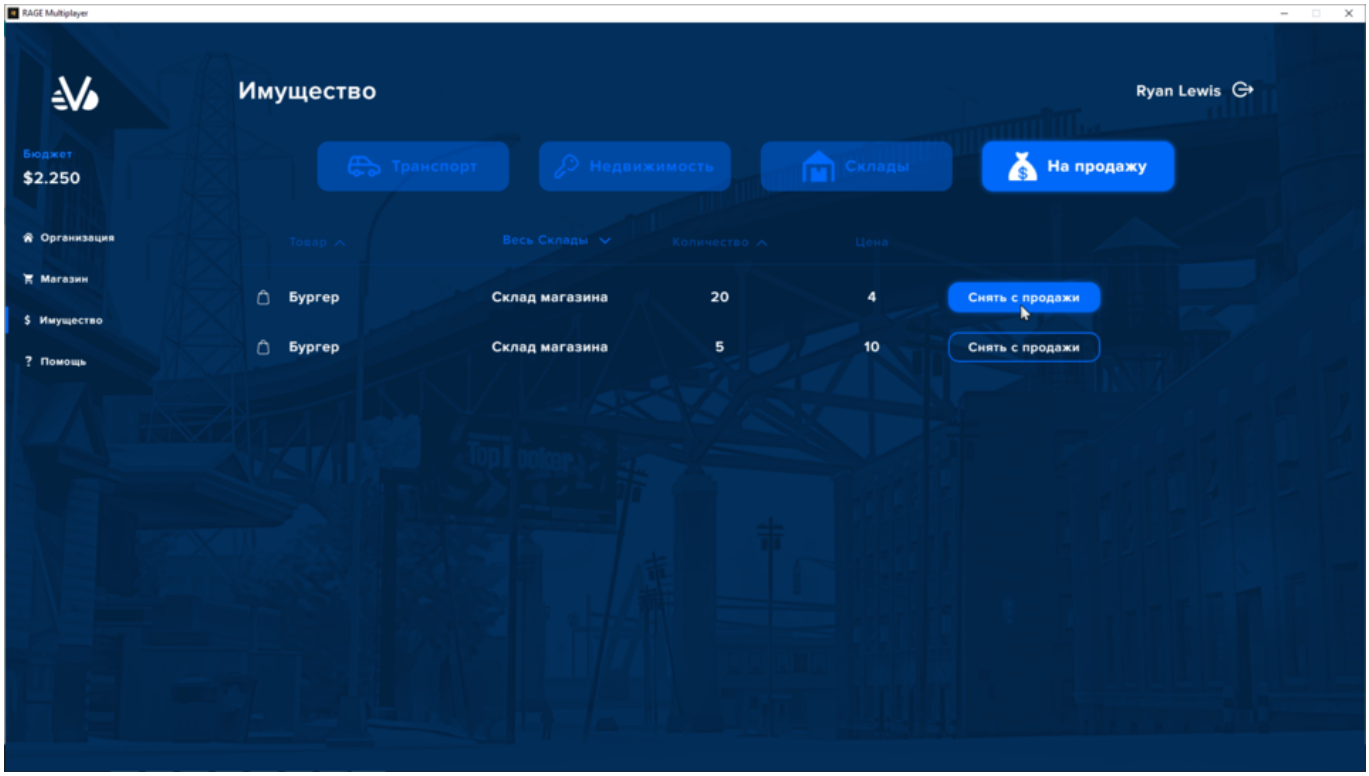

Рисунок 16 – Товары на продажу в меню имущества

На странице логов можно увидеть все основные действия, которые совершали пользователи в приложении данной организации. Доступ к этой странице имеют только владельцы (или лидеры) организаций.

# 2 Обоснование выбора технологий

Для написания курсового проекта использовался язык программирования JavaScript, используя принципы объектно-ориентированного программирования.

Для создания сервера использовался RAGE Multiplayer. Серверная часть этого мультиплеера использует Node.js. Клиентская часть на чистом JS. Связь между серверной и клиентской частью обеспечивает API, предоставляемое мультиплеером. Приложение написано в CEF. CEF позволяет делать обертку над нативными функциями в пространстве объектов виртуальной машины JavaScript. Ресурсоемкие операции по обработке больших массивов данных можно переложить на более строгие и быстрые языки программирования. CEF позволяет обрабатывать события навигации, скачивания файлов и так далее.

Серверная часть написана на Node.js. Node.js - среда выполнения JavaScript, основанная на JavaScript движке V8 из Chrome.

Почему Node.js?

Node.js использует управляемую событиями, неблокирующую модель ввода-вывода, которая делает ее легкой и эффективной
Пакетная экосистема Node.js, npm, является самой большой экосистемой библиотек с открытым исходным кодом в мире
V8 может работать автономно или может быть встроен в любое приложение C++. Благодаря этому, вы можете написать свой собственный код на C++, и сделать его

доступным для JavaScript

Клиентская часть написана в виде SPA (Single page application) на React. React - JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. Эта библиотека стала одной из самых любимых и востребованных технологий, а также самой трендовой технологией на StackOverflow.

Почему React?

- Virtual DOM может повысить производительность высоконагруженных приложений, что может снизить вероятность возникновения возможных неудобств и улучшает пользовательский опыт
- Использование изоморфного подхода помогает производить рендеринг страниц быстрее, тем самым позволяя пользователям чувствовать себя более комфортно во время работы с приложением
- Использования наиболее трендовых технологий разработки

Используя передовые возможности, такие как Virtual DOM или изоморфный JavaScript, React разработчики могут с высокой скоростью создавать высокопроизводительные приложения, несмотря на уровень их сложности. Возможность с легкостью заново использовать уже имеющийся код повышает скорость разработки, упрощает процесс тестирования, и, как результат, понижает затраты. Тот факт, что эта библиотека разрабатывается и поддерживается высококвалифицированными разработчиками и набирает все большую популярность с каждым годом, дает основания надеяться, что тенденция к дальнейшим улучшениям продолжится.

ООП представляет программное обеспечение как совокупность взаимодействующих объектов, а не набор функций или просто список команд (как в традиционном представлении). В ООП, каждый объект может получать сообщения, обрабатывать данные, и отправлять сообщения другим объектам. Каждый объект может быть представлен как маленькая независимая машина с отдельной ролью или ответственностью.

ООП способствует большей гибкости и поддерживаемости в программировании, и широко распространена в крупномасштабном программном инжиниринге. Так как ООП настоятельно подчеркивает модульность, объектно-ориентированный код проще в разработке и проще для понимания впоследствии. Объектно-ориентированный код способствует более точному анализу, кодированию и пониманию сложных ситуаций и процедур, чем методы программирования с меньшей модульностью.

**Терминология ООП**

**Пространство имён** - контейнер, который позволяет разработчикам связать весь функционал под уникальным, специфичным для приложения именем.

**Класс** - определяет характеристики объекта. Класс является описанием шаблона свойств и методов объекта.

**Объект** - экземпляр класса.

**Свойство** - характеристика объекта, например, цвет.

**Метод** - возможности объекта, такие как ходьба. Это подпрограммы или функции, связанные с классом.

**Конструктор** - метод, вызываемый в момент создания экземпляра объекта. Он, как правило, имеет то же имя, что и класс, содержащий его.

**Абстракция** - совокупность комплексных наследований, методов и свойств объекта должны адекватно отражать модель реальности.

Принципы объектно-ориентированного программирования:

Наследование. Возможность выделять общие свойства и методы классов в один класс верхнего уровня (родительский). Классы, имеющие общего родителя, различаются между собой за счет включения в них различных дополнительных свойств и методов.
Инкапсуляция. Свойства и методы класса делятся на доступные из вне (опубликованные) и недоступные (защищенные). Защищенные атрибуты нельзя изменить, находясь вне класса. Опубликованные же атрибуты также называют интерфейсом объекта, т. к. с их помощью с объектом можно взаимодействовать. По идеи, инкапсуляция призвана обеспечить надежность программы, т.к. изменить существенные для существования объекта атрибуты становится невозможно.
Полиморфизм. Полиморфизм подразумевает замещение атрибутов, описанных ранее в других классах: имя атрибута остается прежним, а реализация уже другой. Полиморфизм позволяет специализировать (адаптировать) классы, оставляя при этом единый интерфейс взаимодействия.

Преимущества ООП:

Возможность лёгкой модификации (при грамотном анализе и проектировании)
Возможность отката при наличии версий
Более легкая расширяемость
«Более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку.
Сокращение количества межмодульных вызовов и уменьшение объемов информации, передаваемой между модулями.
Увеличивается показатель повторного использования кода.

Отличия ООП в JavaScript.

Объект в JavaScript — это просто коллекция пар ключ-значение (и иногда немного внутренней магии). Однако, в JavaScript нет концепции класса. К примеру, объект с свойствами {name: Linda, age: 21} не является экземпляром какого-либо класса или класса Object. И Object, и Linda являются экземплярами самих себя. Они определяются непосредственно собственным поведением. Тут нет слоя метаданных (т.е. классов), которые говорили бы этим объектам как нужно себя вести.

Модель прототипного ОО приносит несколько новых динамичных и экспрессивных путей решения старых проблем. В ней также представлены мощные модели расширения и повторного использования кода (а это и интересует людей, которые говорят об объектно-ориентированном программировании). Однако, эта модель даёт меньше гарантий. Например, нельзя полагаться, что объект x всегда будет иметь один и тот же набор свойств.

В отличие от Java и C#, в JavaScript нет ключевого слова private. Однако в JavaScript есть определенная договоренность по использованию «приватных» значений — добавление нижнего подчеркивания перед словом. Однако в ES6 нам доступен вызов объекта WeakMap,

который позволяет создавать приватные свойства.

Getter-ы и setter-ы обычно используются в классических объектно-ориентированных языках для обеспечения инкапсуляции. Они не особо нужны в JavaScript, но, у нас динамический язык. С любой точки зрения, они позволяет обеспечить proxy для запросов на чтение и запись свойств.

Наследование в JavaScript осуществляется через клонирование поведения объекта и расширение его специализированным поведением. Объект, поведение которого клонируют, называется прототипом. Прототип — это обычный объект, который делится своим поведением с другими объектами — в этом случае он выступает в качестве родителя. Концепт клонирования поведения не означает, что вы будете иметь две различные копии одной и той же функции или данных. На самом деле JavaScript реализует наследование через делегирование, т.е. все свойства хранятся в родителе, а доступ к ним расширен через ребёнка.

Программистам часто приходится искать компромисс между повторным использованием кода и его масштабируемостью. Поведение в ООП прописано в абстрактных классах, но его можно в какой-то степени настроить во время создания экземпляров класса. Это способствует лучшему повторному использованию кода, что экономит разработчикам много времени.

# 3 Инструментарий

**Обоснование используемых инструментов**

Для написания кода использовался редактор исходного кода Visual Studio Code. Visual Studio Code отличный выбор для программиста, имеет необходимый минимум:

- неплохую документацию
- автодополнение кода
- подсветка синтаксиса
- встроенный отладчик
- расширение функционала за счет плагинов
- управление системой контроля версий git
- кроссплатформенный
- бесплатный, с открытым исходным кодом

Также редактор адаптирован для Веб-разработки и вполне подойдет для серьезных проектов как основной инструмент редактирования кода.

Для сборки React-приложения использовались Webpack и Babel.

Webpack используется для компиляции JavaScript-модулей. Этот инструмент часто называют «бандлером» (от bundler) или «сборщиком модулей». Он сделан в основном для JavaScript, но он может преобразовывать внешние ресурсы, такие как HTML, CSS и изображения, если включены соответствующие загрузчики. После его установки работать с ним можно, используя интерфейс командной строки или его API. Вот как выглядит то, что делает Webpack (рис.17).

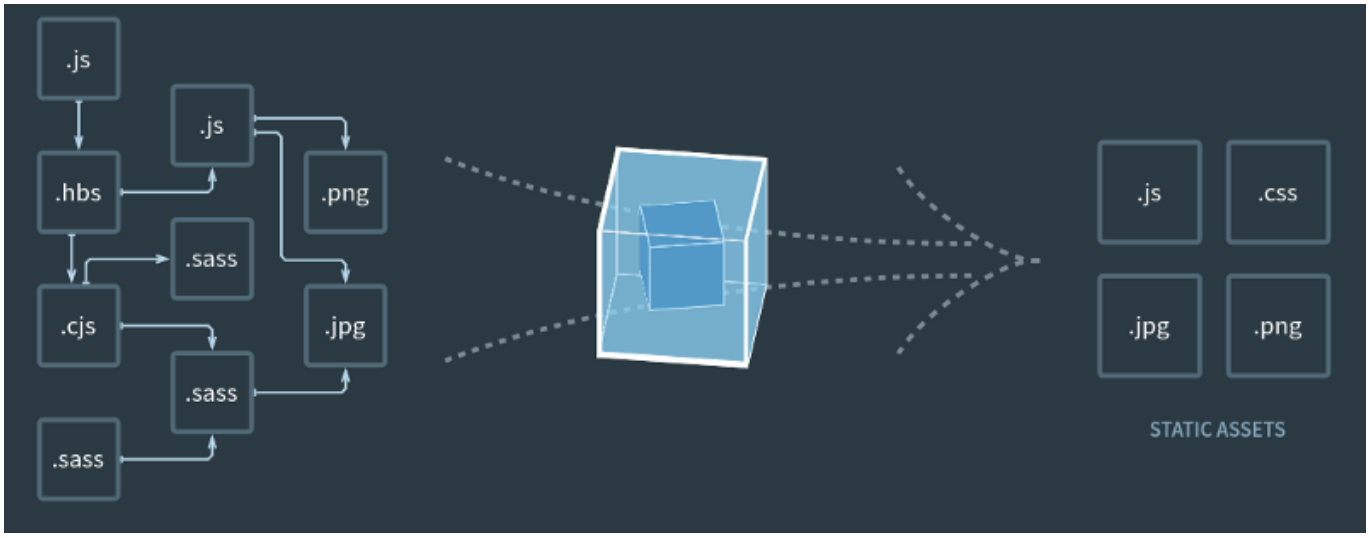


Рисунок 17 – Сборка Webpack

Babel — это транспилятор, который, в основном, используется для преобразования конструкций, принятых в свежих версиях стандарта ECMAScript, в вид, понятный как современным, так и не самым новым браузерам и другим средам, в которых может выполняться JavaScript. Babel, кроме того, умеет преобразовывать в JavaScript и JSX-код, используя @babel/preset-react. Именно благодаря Babel, при разработке React-приложений, можно пользоваться JSX.

В качестве СУБД используется MySQL — это сервер баз данных, используемый в разных приложениях. SQL означает язык структурированных запросов - Structured Query Language, который MySQL использует для коммуникации с другими программами. Сверх того, MySQL имеет свои собственные расширенные функции SQL для того, чтобы обеспечить пользователям дополнительный функционал.

Для управления этой СУБД используется phpMyAdmin — веб-приложение с открытым кодом, написанное на языке PHP и представляющее собой веб-интерфейс для администрирования СУБД MySQL. phpMyAdmin позволяет через браузер и не только осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных.

Для связи БД сервера (рис.18) и серверной части на Node.js используется модуль npm mysql.

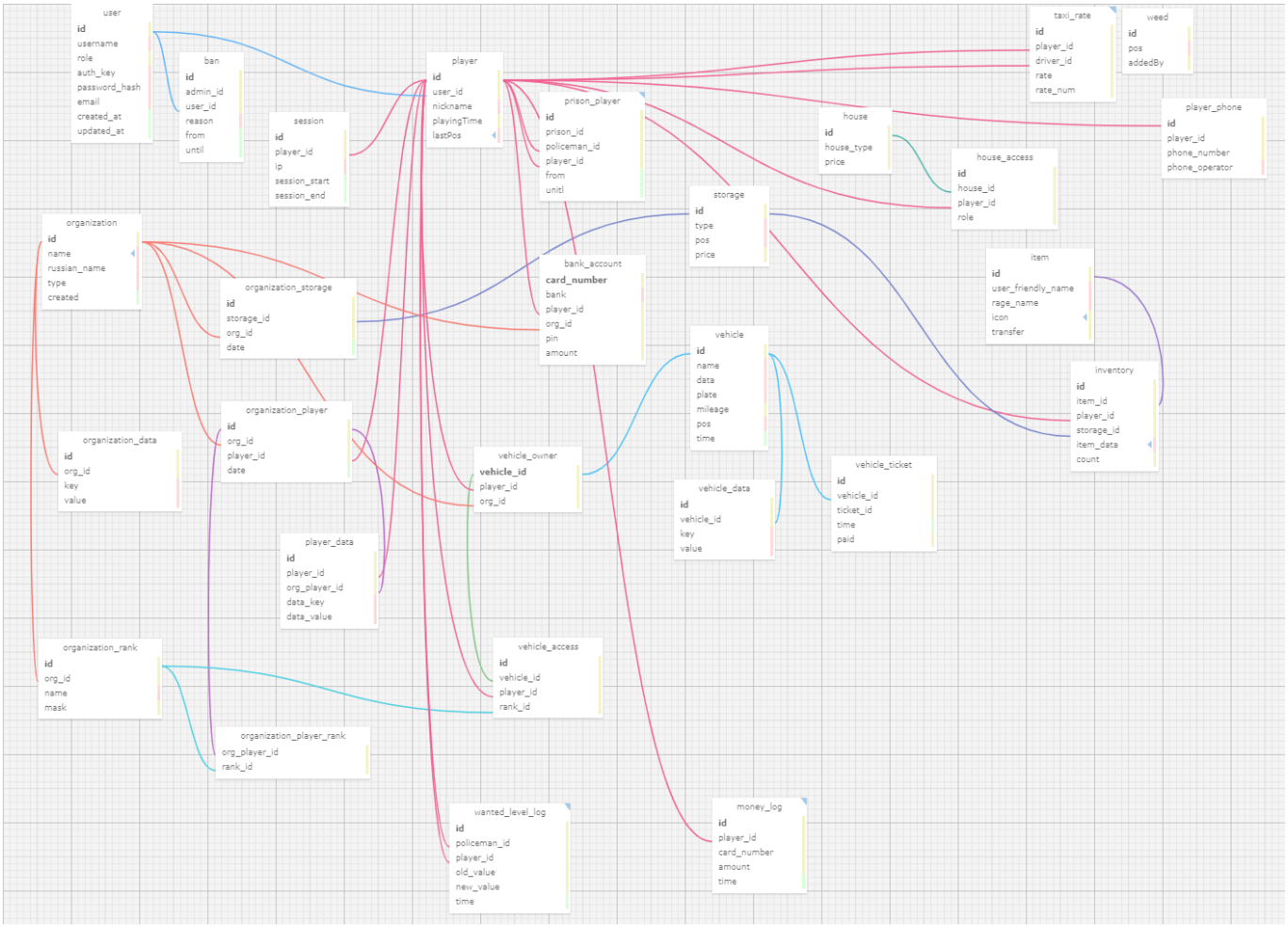

Рисунок 18 – схема базы данных сервера

**Использование системы контроля версий GIT**

Системы контроля версий (СКВ, VCS, Version Control Systems) позволяют разработчикам сохранять все изменения, внесённые в код. СКВ также дают возможность нескольким разработчикам работать над одним проектом и сохранять внесённые изменения, чтобы убедиться, что все могут следить за тем, над чем они работают. Существует три типа СКВ: локальная, централизованная и распределённая.

Git — распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать совместно с другими разработчиками. Она была разработана в 2005 году Линусом Торвальдсом, создателем Linux, для того чтобы другие разработчики могли вносить свой вклад в ядро Linux. Git известен своей скоростью, простым дизайном, поддержкой нелинейной разработки, полной децентрализацией и возможностью эффективно работать с большими проектами.

Git стоит отдельно от других СКВ из-за подхода к работе с данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

Преимущества Git

Бесплатный и open-source. Это значит, что его можно бесплатно скачать и вносить любые

изменения в исходный код
Небольшой и быстрый. Он выполняет все операции локально, что увеличивает его скорость. Кроме того, Git локально сохраняет весь репозиторий в небольшой файл без потери качества данных
Резервное копирование. Git эффективен в хранении бэкапов, поэтому известно мало случаев, когда кто-то терял данные при использовании Git
Простое ветвление. В других СКВ создание веток— утомительная и трудоёмкая задача, так как весь код копируется в новую ветку. В Git управление ветками реализовано гораздо проще и эффективнее

GitHub — это облачное хранилище файлов. Смысл в том, что на этом сервисе вы можете разместить какие-то файлы со своего компьютера и хранить их на удаленном сервере. Причем, делать вы это можете совершенно бесплатно. Обычно он используется вместе с Git и даёт разработчикам возможность сохранять их код онлайн, а затем взаимодействовать с другими разработчиками в разных проектах.

Также GitHub может похвастаться контролем доступа, багтрекингом, управлением задачами и вики для каждого проекта. Цель GitHub — содействовать взаимодействию разработчиков.

К проекту, загруженному на GitHub, можно получить доступ с помощью интерфейса командной строки Git и Git-команд. Также есть и другие функции, такие как документация, запросы на принятие изменений (pull requests), история коммитов, интеграция со множеством популярных сервисов

Этот проект на GitHub: github.com/Stazeg/organization-ui

# 4 Архитектурный шаблон проектирования MVC

Шаблон проектирования MVC (рис.19) предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо. Компоненты MVC:

Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.
Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.
Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения.

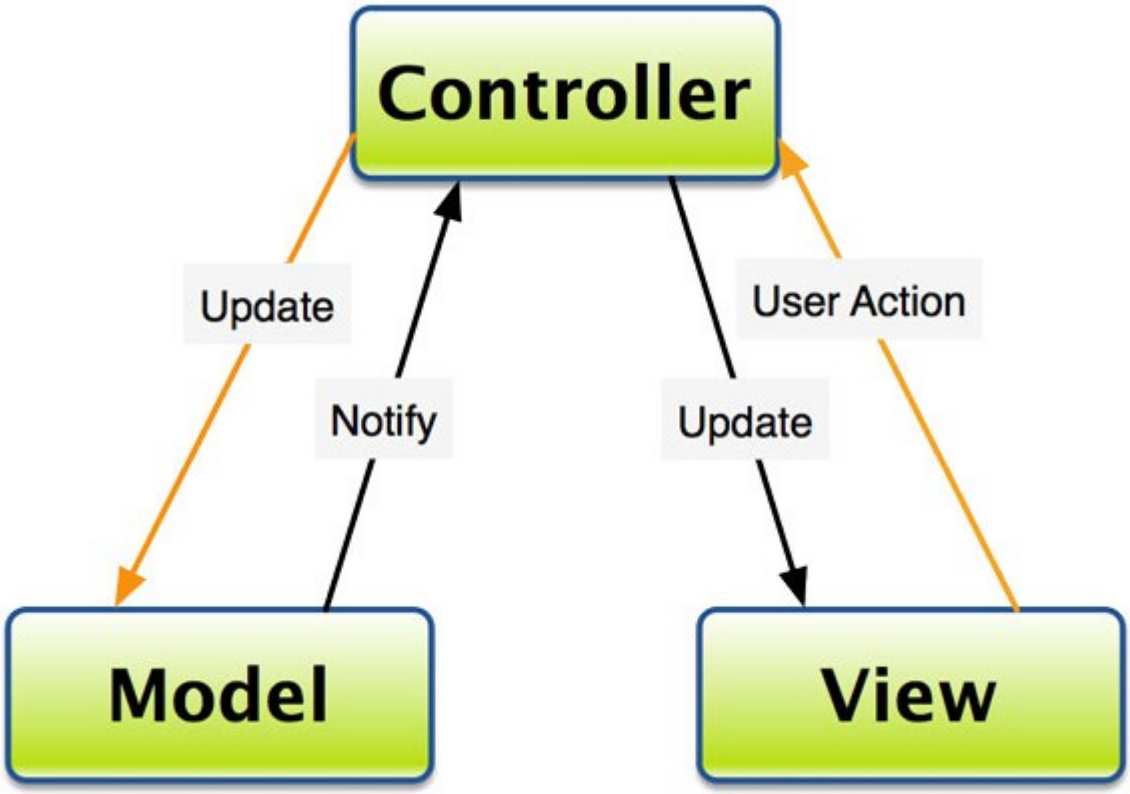


Рисунок 19 – Шаблон проектирования MVC

Поскольку MVC не имеет строгой реализации, то реализован он может быть по-разному. Нет общепринятого определения, где должна располагаться бизнес-логика. Она может находиться как в контроллере, так и в модели. В последнем случае, модель будет содержать все бизнес-объекты со всеми данными и функциями. Некоторые фреймворки жестко задают где должна располагаться бизнес-логика, другие не имеют таких правил.

Также не указано, где должна находиться проверка введённых пользователем данных. Простая валидация может встречаться даже в представлении, но чаще они встречаются в контроллере или модели.

Самое очевидное преимущество, которое мы получаем от использования концепции MVC — это чёткое разделение логики представления (интерфейса пользователя) и логики приложения.

Поддержка различных типов пользователей, которые используют различные типы устройств является общей проблемой наших дней. Предоставляемый интерфейс должен различаться, если запрос приходит с персонального компьютера или с мобильного телефона. Модель возвращает одинаковые данные, единственное различие заключается в том, что контроллер выбирает различные виды для вывода данных.

Помимо изолирования видов от логики приложения, концепция MVC существенно уменьшает сложность больших приложений. Код получается гораздо более структурированным, и, тем самым, облегчается поддержка, тестирование и повторное использование решений.

Наиболее распространенные виды MVC-паттерна, это:

Model-View-Controller. Основная идея этого паттерна в том, что и контроллер, и

представление зависят от модели, но модель никак не зависит от этих двух компонент.
Model-View-Presenter. Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель.
Model-View-View Model. Данный подход позволяет связывать элементы представления со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя.

Реализация MVVM и MVP-паттернов, на первый взгляд, выглядит достаточно простой схожей. Однако, для MVVM связывание представления с View-моделью осуществляется автоматически, а для MVP — необходимо программировать. MVC, по-видимому, имеет больше возможностей по управлению представлением.

Недостатки концепции MVC:

Необходимость использования большего количества ресурсов. Сложность обусловлена тем, что все три фундаментальных блока являются абсолютно независимыми и взаимодействуют между собой исключительно путем передачи данных. Controller должен всегда загрузить (и при необходимости создать) все возможные комбинации переменных и передать их в Model. Model, в свою очередь, должен загрузить все данные для визуализации и передать их во View. Например, в модульном подходе, модуль может напрямую обрабатывать переменные окружения и визуализировать данные без загрузки их в отдельные секции памяти.
Усложнен механизм разделения программы на модули. В концепции MVC наличие трех блоков (Model, View, Controller) прописано жестко. Соответственно каждый функциональный модуль должен состоять из трех блоков, что в свою очередь, несколько усложняет архитектуру функциональных модулей программы.
Усложнен процесс расширения функционала. Проблема очень схожа с вышеописанной. Недостаточно просто написать функциональный модуль и подключить его в одном месте программы. Каждый функциональный модуль должен состоять из трех частей, и каждая из этих частей должна быть подключена в соответствующем блоке.

MVC до сих пор остается, пожалуй, лучшим способом разработки серверной части приложений. Корень проблемы в том, что принципы и декомпозиции, которые MVC представляет на сервере, не такие же, как на клиенте. Когда мы переносим MVC на клиент, начинаются проблемы. Контроллеры напоминают то, что мы называем «code-behind». Контроллер сильно зависит от представления. Кроме этого, если вспомнить принцип единственной ответственности, это прямое нарушение правил. Код клиентского контроллера в определенной степени имеет дело и с обработкой событий, и с бизнес-логикой.

Но разработчики упустили из вида важнейший вклад, который принес миру React, — компонентно-ориентированную архитектуру. React не изобрел компонентный подход, но перенес идею на новый уровень. Компоненты представляют из себя: представление + обработка событий + состояние интерфейса.

Диаграмма (рис.20) ниже показывает, как фактически разделить стандартную MVC модель, чтобы получить компоненты. Выше линии осталось именно то, что пытается решить Flux: управление состоянием приложения и бизнес-логикой.

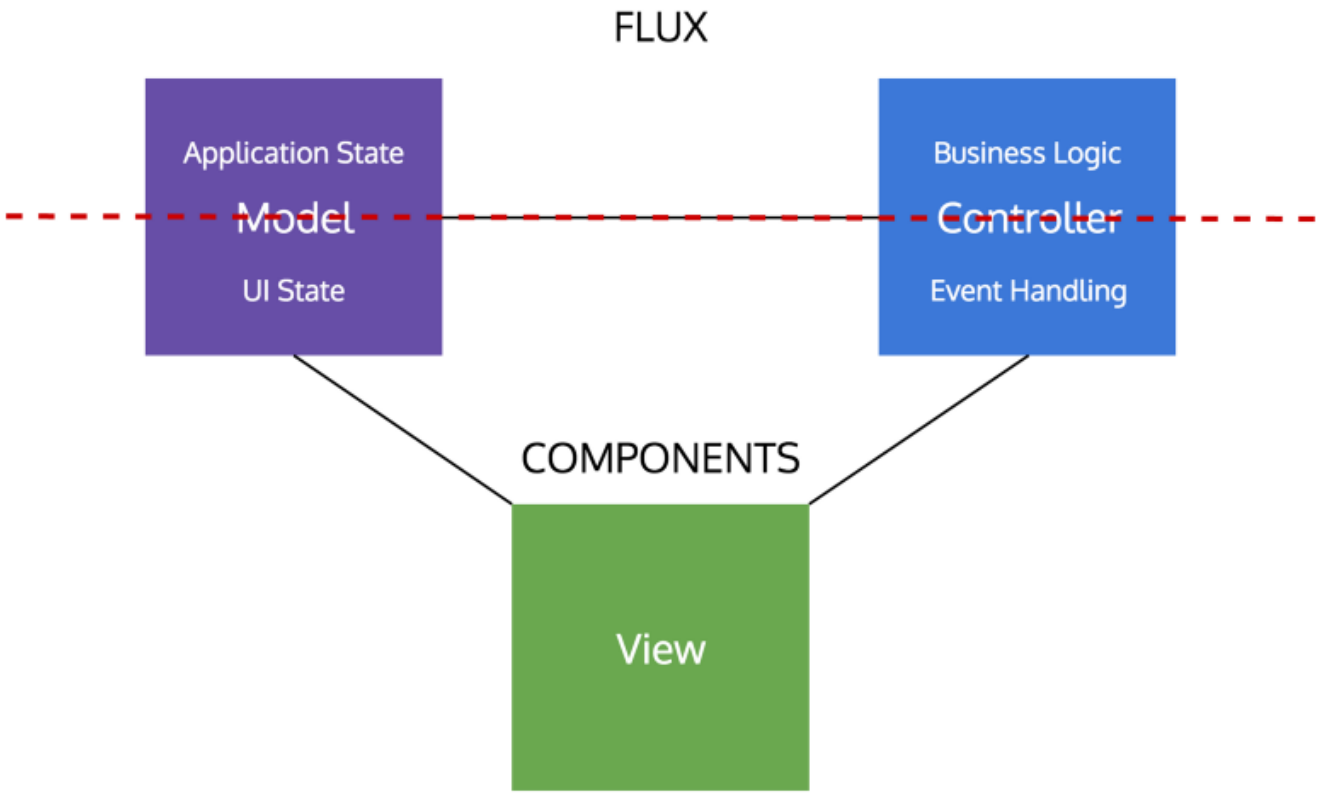


Рисунок 20 – Архитектура Flux

Одновременно с популярностью React и компонентно-ориентированной архитектуры мы также увидели растущую популярность однонаправленной архитектуры для управления состоянием приложения.

Одной из причин их успешного совместного применения стало то, что они вдвоем полностью покрывают классический MVC подход. Они к тому же обеспечивают намного лучшее разделение ответственности при построении фронтенд-архитектуры.

MVC был необходим в самом начале, потому что фронтенд-приложения становились все больше и сложнее, а мы не знали, как их структурировать. Я думаю, он справился со своим предназначением и заодно преподал хороший урок о том, как брать хорошую практику из одного контекста (сервера) и применять ее в другом (клиенте).

# 5 Система контроля доступа с использованием битовой маски

Для проверки прав доступа к определенным методам в приложении управления организацией используются побитовые операторы. Побитовые операторы интерпретируют операнды как последовательность из 32 битов (нулей и единиц). Они производят операции, используя двоичное представление числа, и возвращают новую последовательность из 32 бит (число) в качестве результата. Примеры представления чисел в двоичной системе:

```
mask = 0;    // 00000000000000000000000000000000
mask = 1;    // 00000000000000000000000000000001
mask = 2;    // 00000000000000000000000000000010
```

```
mask = 3;    // 00000000000000000000000000000011
mask = 255;// 00000000000000000000000011111111
```

Побитовые операторы работают следующим образом:

1. Операнды преобразуются в 32-битные целые числа, представленные последовательностью битов. Дробная часть, если она есть, отбрасывается.
2. Для бинарных операторов – каждый бит в первом операнде рассматривается вместе с соответствующим битом второго операнда: первый бит с первым, второй со вторым и т.п. Оператор применяется к каждой паре бит, давая соответствующий бит результата.
3. Получившаяся в результате последовательность бит интерпретируется как обычное число.

Каждой роли соответствует ряд доступов в приложении управления организации и на самом сервере. Например, сотрудник Полиции может лишь просматривать роли и участников, а Глава Полиции – ещё и редактировать их, и тому подобное.

В итоге доступы хранятся в 10-ной системе, а для их проверки используется двоичная. Такая интерпретация доступов позволяет «упаковать» много информации в одно число. Это экономит память, а кроме этого – это удобно, поскольку в дополнение к экономии – по такому значению очень легко проверить, имеет ли персонаж заданную комбинацию доступов.

У каждого класса участника организации есть метод getMaskOrder, который возвращает константу с названиями методов, к которым можно изменить доступ, и их описания для клиента. С помощью этого метода, при запуске сервере создается 10-ное значение для каждого метода, который вернул getMaskOrder, в методе createMaskOrderValues.

Также при старте сервера из базы данных загружается значение маски каждой роли и присваивается каждому участнику. Если у участника несколько ролей, то его маска будет складываться из всех масок его ролей. Если в приложении изменяются права доступа для роли (рис.21), то на сервере вызывается метод refreshMask, для того чтобы обновить маску у всех персонажей, у которых эта роль имеется.

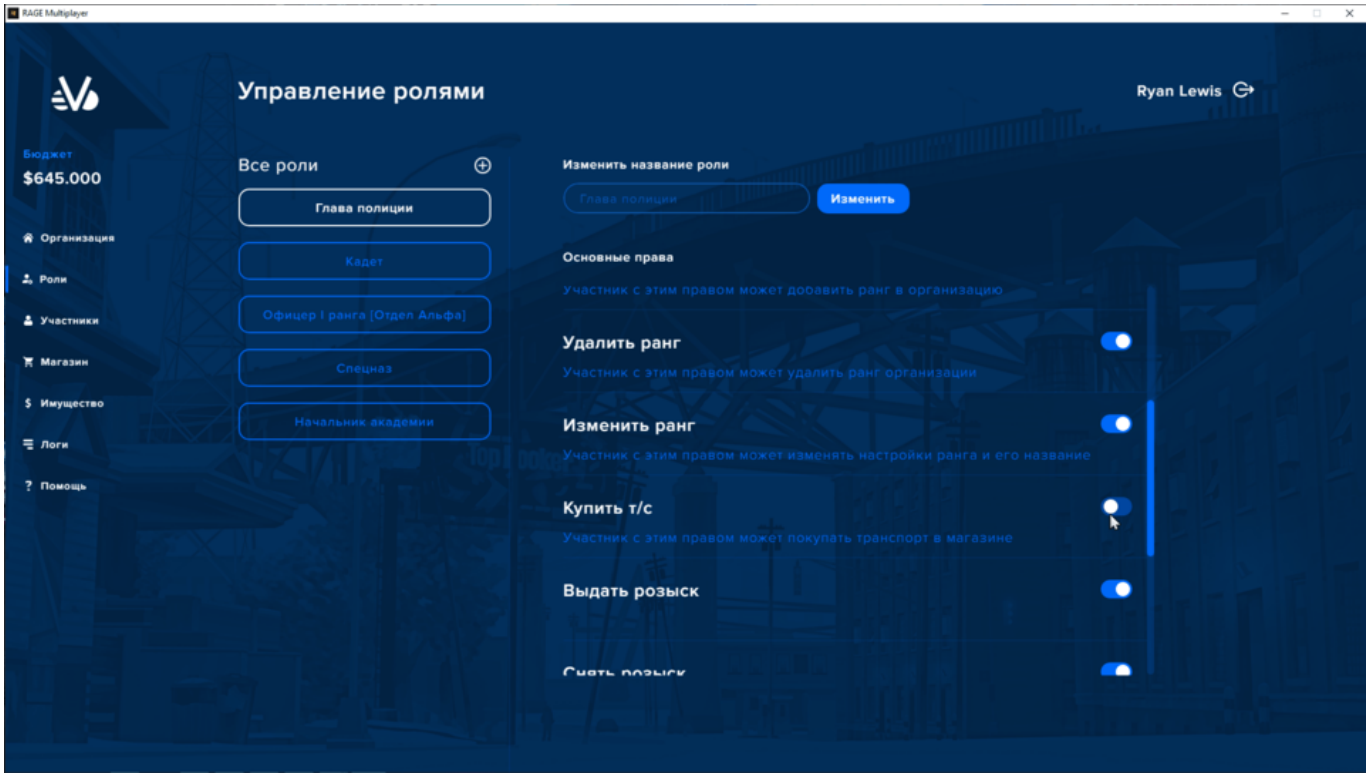

Рисунок 21 – Изменение прав доступа для роли Глава полиции

При вызове метода, который имеет проверку на права доступа, вызывается метод checkAccess, который проверяет доступ персонажа к данному методу с помощью побитовой операции &. checkAccess возвращает true/false значение и выполняет метод, при значении true или выдает ошибку прав доступа (рис.22) при значении false.

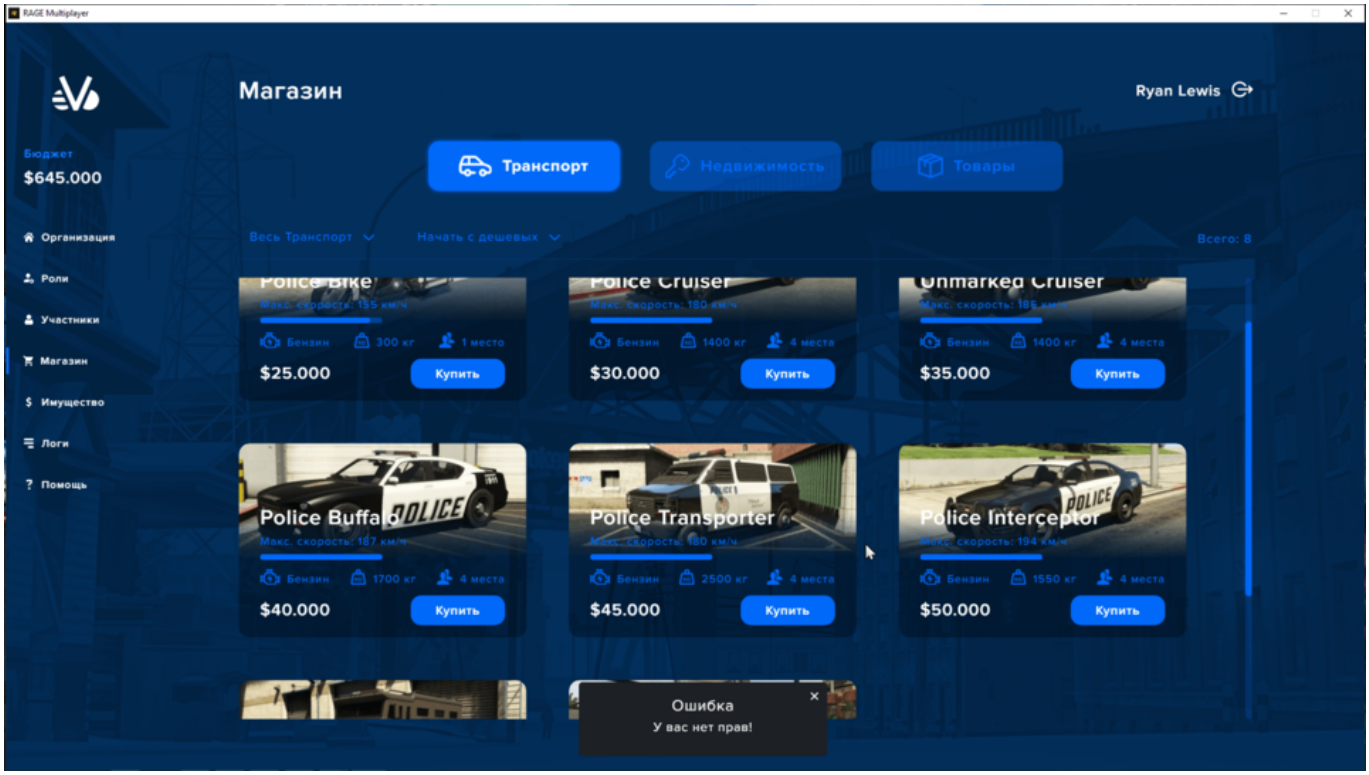

Рисунок 22– Уведомление об отсутствии прав на покупку т/с

Битовой маской называют как раз комбинацию двоичных значений, которая используется

для проверки и выборки единиц на нужных позициях. Маски могут быть весьма удобны. В частности, их используют в методах, чтобы одним параметром передать несколько «флагов», т.е. однобитных значений.

Применение масок налагает определённые ограничения. В частности, побитовые операторы в JavaScript работают только с 32-битными числами, а значит, к примеру, 33 доступа уже в число не упакуешь. Да и работа с двоичной системой счисления – как ни крути, менее удобна, чем с десятичной или с обычными логическими значениями true/false.

Поэтому основная сфера применения масок – это быстрые вычисления, экономия памяти.

## Заключение

По итогам работы, было создано приложение управления организацией для RAGE Multiplayer в GTA V. В процессе разработки были изучены как общие приёмы программирования на Node.js, так и назначение отдельных модулей, а также приемы программирования на языке JavaScript и фреймворке React. Были изучены API RAGE Multiplayer, возможности реляционной базы данных и принципы SPA приложений на React. Была разработана система контроля доступа с использованием битовой маски, изучены побитовые операторы и область их применения. Для работы данного приложения были написаны классы организации и участника организации. На данном сервере уже могут играть люди, управляя своей организацией с помощью данного приложения. На сервере проводились тестирования приложения многими игроками и ошибок обнаружено не было. В дальнейшем планируется увеличить количество возможных игроков путем добавления нового функционала и внедрения полноценной экономики. Убедившись в корректности работы приложения после пройденных тестов, можно сказать, что результат курсового проекта полностью соответствует всем исходным требованиям.

## Список использованных источников

## Приложения

1. [Задание] **Задание** 5ee12864eeb48_ООП_Задание.docx
2. [электронный документ] 5ee2ec3270603_ООП_Задание.docx
3. [Приложение] **Приложение** 5ee2ee62c4b85_ООП_Курсовая.docx