Министерство образования Республики Беларусь

Учреждение образования

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные Системы и Сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Программное средство

**«Многопользовательская игра 4 цвета»**

БГУИР КП 1-40 01 01 032 ПЗ

Студент: гр. 310902 Ширукова В.А.

Руководитель: Болтак С.В.

Минск 2025

Содержание

[Введение 5](#_Toc199129912)

[1 Анализ предметной области 6](#_Toc199129913)

[1.1 Обзор аналогов 6](#_Toc199129914)

[1.2 Постановка задачи 9](#_Toc199129915)

[2 Проектирование програмнного средства 10](#_Toc199129916)

[2.1 Структура программы 10](#_Toc199129917)

[2.2 Проектирование интерфейса программного средства 10](#_Toc199129918)

[2.3 Проектирование функционала программного средства 13](#_Toc199129919)

[3 Разработка программного средства 17](#_Toc199129923)

[3.1 Реализация серверной части приложения 17](#_Toc199129924)

[3.2 Функции сетевого взаимодействия 21](#_Toc199129928)

[3.3 Функции для обработки пользовательских данных 23](#_Toc199129932)

[4 Тестирование программного средства 26](#_Toc199129935)

[5 Руководство пользователя 27](#_Toc199129936)

[5.1 Интерфейс программного средства 27](#_Toc199129937)

[5.2 Управление программным средством 29](#_Toc199129940)

[Заключение 30](#_Toc199129941)

[Список использованных источников 31](#_Toc199129942)

[Приложение А. Текст программы 32](#_Toc199129943)

## **ВВЕДЕНИЕ**

В современном мире цифровые технологии стремительно развиваются, а интерактивные веб-приложения становятся неотъемлемой частью досуга и социального взаимодействия людей. Многопользовательские игры в реальном времени приобретают особую популярность, поскольку они позволяют людям общаться и развлекаться независимо от географического местоположения. Классические настольные игры, адаптированные для веб-платформ, сочетают в себе привычные правила с современными технологическими возможностями.

Несмотря на существование множества игровых платформ, многие из них не обеспечивают полноценного игрового опыта для популярных карточных игр или требуют установки дополнительного программного обеспечения. Существует потребность в разработке легковесных, доступных через браузер решений, которые обеспечивают стабильное соединение между игроками, корректную обработку игровой логики и интуитивно понятный пользовательский интерфейс. Особенно важными являются возможности создания игровых комнат, управления состоянием игры и синхронизации действий между всеми участниками в реальном времени.

Актуальность разработки многопользовательской веб-игры обусловлена растущим спросом на доступные онлайн-развлечения, которые не требуют установки клиентского программного обеспечения и могут работать на любых устройствах с веб-браузером. Использование современных веб-технологий, таких как Node.js для серверной части, Socket.IO для взаимодействия в реальном времени и Vue.js для создания отзывчивого пользовательского интерфейса, позволяет создать масштабируемое и производительное решение. Реализация игры 4 цвета как веб-приложения демонстрирует возможности современного веб-стека для создания сложных интерактивных приложений с поддержкой множественных одновременных соединений.

В рамках данной курсовой работы будет разработано программное средство «Многопользовательская игра 4 цвета» — веб-платформа, позволяющая пользователям создавать игровые комнаты, приглашать друзей и играть в популярную карточную игру через браузер. Система будет включать серверную часть на Node.js с использованием Express.js и Socket.IO для управления игровыми сессиями, а также клиентскую часть на Vue.js с анимированным интерфейсом и поддержкой всех правил классической игры UNO.

# 1 **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

# 1.1 Обзор аналогов

Программное средство Four Colors, предназначенное для простейшей игры.

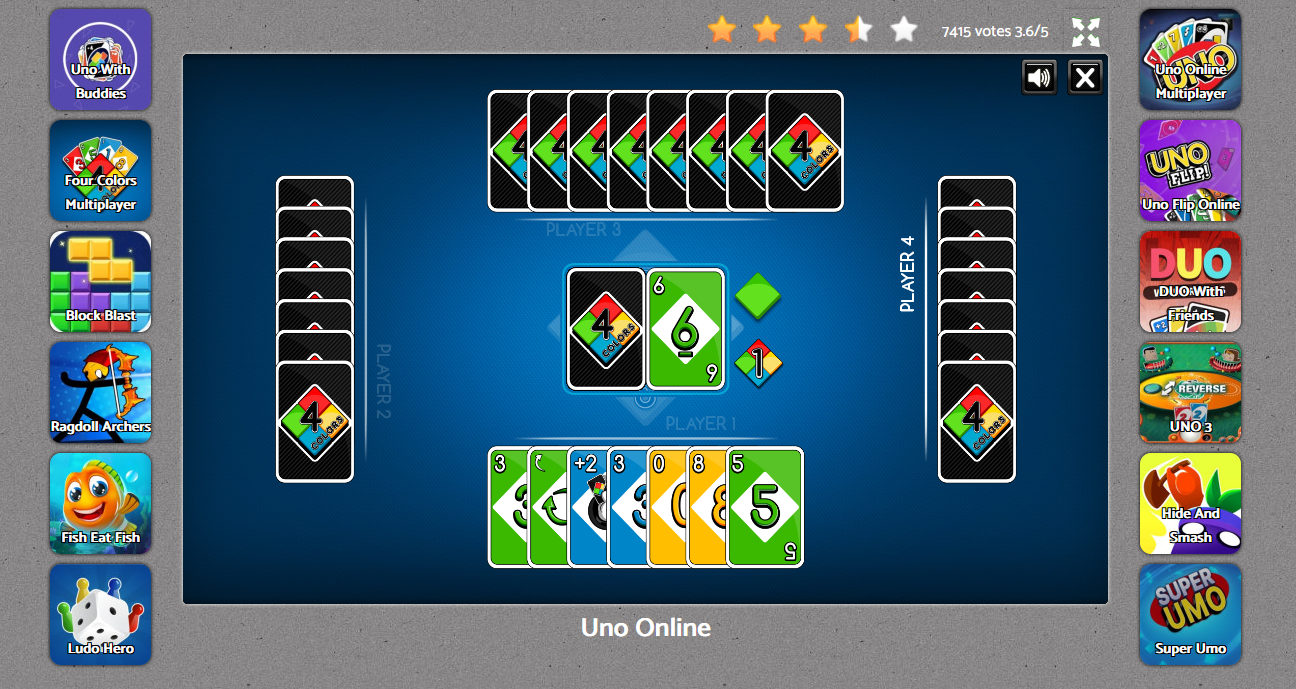


Рисунок 1.1 – Программное средство «Four Colors»

Four Colors представляет собой базовую реализацию популярной карточной игры UNO в веб-формате. Приложение предлагает минималистичный интерфейс с основными игровыми механиками, включающими стандартную колоду карт, базовые правила игры и простое управление.

Основные преимущества Four Colors включают легковесность приложения, быструю загрузку и понятную игровую логику, соответствующую классическим правилам UNO. Платформа не требует установки дополнительного программного обеспечения и работает непосредственно в веб-браузере.

Критические недостатки платформы значительно ограничивают ее практическое применение. Устаревший дизайн интерфейса не соответствует современным стандартам UX/UI и создает неприятные впечатления от использования. Отсутствие многопользовательского режима исключает возможность игры с друзьями в реальном времени, что является основной привлекательностью подобных игр. Дополнительно отметим отсутствие системы комнат для игроков, невозможность создания приватных игровых сессий, отсутствие чата между участниками и примитивная анимация карт без современных визуальных эффектов.



Рисунок 1.2 – Программное средство «scuffeduno.online»

Платформа scuffeduno.online позиционирует себя как неформальная реализация игры UNO с упором на простоту и доступность. Сервис предлагает возможность быстрого создания игровых комнат и базовую многопользовательскую функциональность.

**Преимущества** включают наличие многопользовательского режима, возможность создания комнат с уникальными кодами для приглашения друзей, базовую систему чата и относительно стабильное соединение между игроками. Платформа поддерживает основные правила UNO и позволяет играть до 8 человек одновременно.

**Недостатки** проявляются в нестабильной работе сервера, частых разрывах соединения при большой нагрузке, отсутствии системы регистрации пользователей и сохранения игровой статистики. Интерфейс выглядит незавершенным с примитивной графикой карт, отсутствием анимаций и звуковых эффектов. Также отмечается ограниченная функциональность чата, отсутствие системы рейтингов и невозможность настройки правил игры.

Buddy Board Games представляет наиболее профессиональную реализацию онлайн UNO среди рассматриваемых аналогов. Платформа интегрирована в более крупную экосистему настольных игр и предлагает расширенную функциональность.

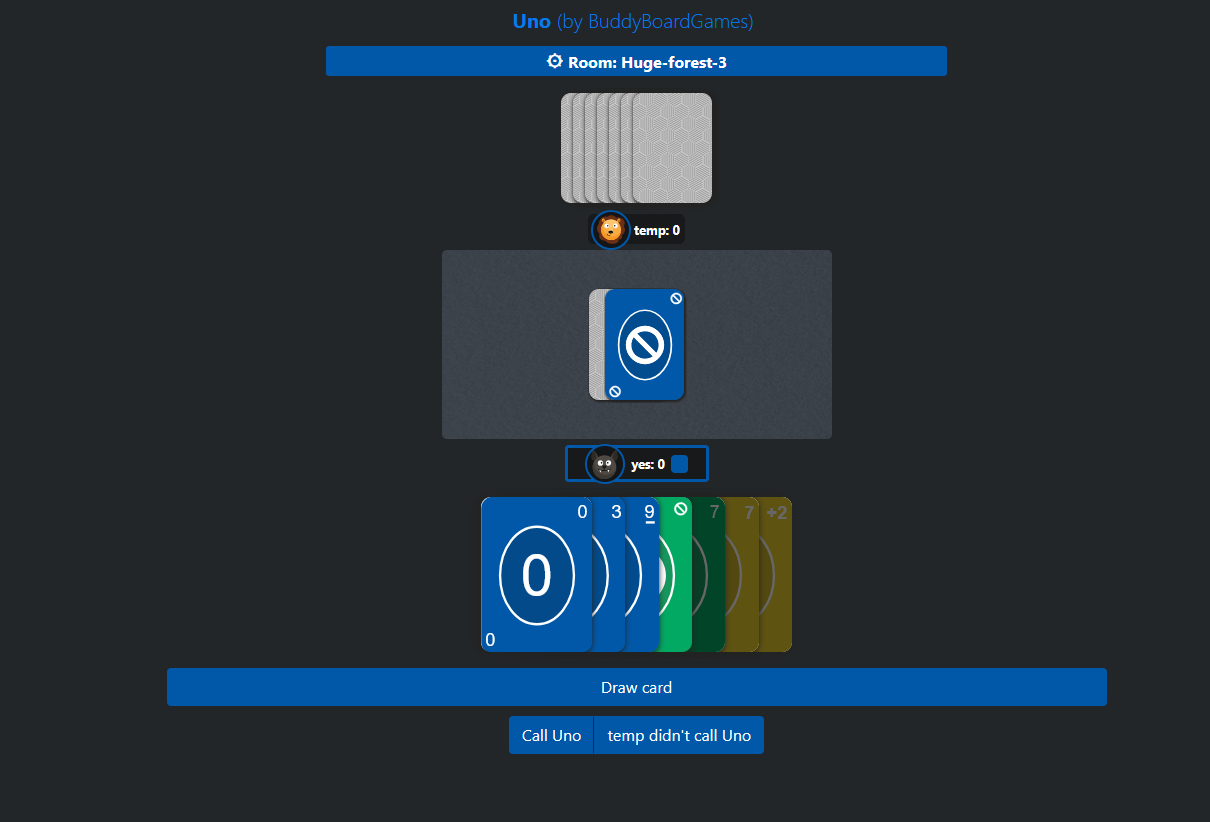


Рисунок 1.3 – Программное средство «buddyboardgames.com/uno»

Преимущества включают качественный графический дизайн с анимированными картами, стабильную многопользовательскую архитектуру, систему регистрации и профилей пользователей, встроенный голосовой и текстовый чат, возможность настройки правил игры и систему рейтингов. Платформа поддерживает создание как публичных, так и приватных комнат, имеет систему друзей и уведомлений.

Недостатки связаны с коммерческой моделью, требующей оплаты за доступ к полному функционалу, сложностью интерфейса для новых пользователей и необходимостью обязательной регистрации. Дополнительно отмечается отсутствие мобильной оптимизации, ограниченные возможности кастомизации интерфейса и зависимость от стабильности внешних серверов, что может создавать проблемы с доступностью сервиса.

# 1.2 Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства «Многопользовательская игра 4 цвета» — современной веб-платформы для онлайн-игры в популярную карточную игру UNO.

В процессе реализации будет разработан функционал для создания полноценной игровой экосистемы, объединяющей игроков в реальном времени через веб-браузер, с акцентом на удобный пользовательский интерфейс, стабильное сетевое взаимодействие и корректную реализацию игровой механики.

В клиентской части приложения планируется реализовать следующие функции:

* Создание и подключение к игровым комнатам с уникальными кодами;
* Интерактивный игровой интерфейс с анимированными картами;
* Отображение игрового состояния (карты в руке, колода, карты других игроков);
* Реализация всех правил классической игры UNO/4 цвета.

В серверной части приложения планируется реализовать функции:

* Управление игровыми сессиями и комнатами через WebSocket-соединения;
* Обработка игровой логики и валидация ходов игроков;
* Синхронизация состояния игры между всеми участниками;
* Система управления пользователями и игровыми комнатами;
* Масштабируемая архитектура для поддержки множественных одновременных игр.

Технологический стек разработки:

* Frontend: Vue.js для создания реактивного пользовательского интерфейса, GSAP для анимаций карт и игровых элементов;
* Backend: Node.js с Express.js для создания веб-сервера и API;
* Real-time коммуникация: Socket.IO для обеспечения взаимодействия в реальном времени;
* Архитектура: Модульная структура с разделением на игровые сервисы, контроллеры и компоненты интерфейса.

Приложение должно обеспечивать стабильную работу, корректную обработку всех игровых сценариев и приятный пользовательский опыт с современным интерфейсом и плавными анимациями.

# 2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## **Структура программы**

Программное средство «Многопользовательская игра 4 цвета» построено на клиент-серверной архитектуре с модульным разделением компонентов.

Серверная часть (Node.js):

* Server Module – основной модуль на Express.js для HTTP-запросов и маршрутизации;
* GameService Module – игровая логика, управление комнатами и валидация ходов;
* SocketHandler Module – WebSocket-соединения через Socket.IO;
* RoomManager Module – создание и управление игровыми комнатами.

Клиентская часть (Vue.js):

* App Module – корневой компонент приложения;
* GameBoard Component – игровое поле и отображение карт;
* GameRoom Component – управление комнатами и список игроков;
* CardComponent – отображение и анимация карт;
* UI Components – кнопки, попапы, уведомления.

Модули взаимодействия:

* SocketClient Module – WebSocket-соединение с сервером;
* EventHandler Module – обработка игровых событий;
* StateManager Module – управление состоянием игры.

Такая структура обеспечивает масштабируемость, независимое развитие частей системы и поддержку множественных игровых сессий в реальном времени.

## **Проектирование интерфейса программного средства**

Для разработки интерфейса многопользовательской веб-игры «4 цвета» будет использоваться современный декларативный подход с применением Vue.js — прогрессивного JavaScript-фреймворка для создания пользовательских интерфейсов и одностраничных приложений.

* + 1. Меню входа

Интерфейс меню приложения изображен как рисунке 2.1.

Экран входа построен на основе компонента RoomEntry, который включает:

* Поле ввода с валидацией кода комнаты;
* Визуальную обратную связь при вводе данных;
* Анимированные переходы между состояниями.

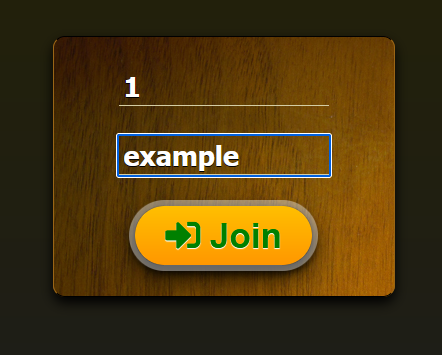


Рисунок 2.1 **–** Главное окно программного средства

Интерфейс входа в приложение «4 цвета» организован согласно принципам современного веб-дизайна и включает следующие основные компоненты:

* Поле ввода кода комнаты — центральная область для ввода идентификатора игровой комнаты;
* Кнопка подключения — элемент для присоединения к существующей игровой сессии;
* Фоновое оформление — стилизованный деревянный текстурный фон для создания атмосферы настольной игры.
  + 1. Игровой интерфейс

Интерфейс игрового процесса включает следующие основные Vue.js компоненты:

* PlayerHand — компонент отображения карт в руке игрока в нижней части экрана;
* GameBoard — центральная игровая область с колодой и картой сброса;
* OpponentCards — отображение карт противников в верхней части экрана;
* GameControls — элементы управления игровыми действиями.

Интерфейс игрового процесса изображен на рисунке 2.2.



Рисунок 2.2 **–** Интерфейс игрового процесса

Игровое поле организовано следующим образом:

* Верхняя панель — отображение карт противника и информации о ходе;
* Центральная область — колода карт и область сброса с текущей активной картой;
* Нижняя панель — карты игрока с возможностью выбора и разыгрывания;
* Боковые элементы — индикаторы состояния игры и счетчики карт.

Таким образом, система интерфейса в приложении «4 цвета» реализована с использованием компонентной архитектуры Vue.js, обеспечивающей интуитивный переход от входа в игру к полноценному игровому процессу с поддержкой всех правил классической карточной игры UNO.

## **Проектирование функционала программного средства**

Ключевым аспектом разработки серверной части многопользовательской игры является обеспечение корректной обработки игровых событий, синхронизации состояния между клиентами и управления игровыми сессиями в реальном времени. Учитывая специфику многопользовательской веб-игры «4 цвета», программное средство должно содержать следующий список функций:

* основные игровые функции:
  1. функции создания и управления игровыми комнатами;
  2. функция обработки игровых ходов и валидации правил UNO;
  3. функция синхронизации игрового состояния между участниками в реальном времени.
* серверные функции:

1. функция создания уникальных кодов игровых комнат;
2. функция управления жизненным циклом игровых сессий;
3. функция обеспечения безопасности и предотвращения читерства.

* дополнительные функции:

1. функции управления подключениями игроков и WebSocket-соединениями;
2. функции отображения игрового интерфейса и анимации карт;
3. функция обработки ошибок сети и отображение уведомлений игрокам;
4. функции чата для общения между участниками игры.

Такая структура обеспечивает полный функционал многопользовательской карточной игры с поддержкой real-time взаимодействия и современным веб-интерфейсом.

### Функция создания игровой комнаты

Функция createRoom является ключевым компонентом серверной части приложения, отвечающим за инициализацию новых игровых сессий и подготовку начального игрового состояния. Данная функция обеспечивает создание уникальных игровых комнат с автоматической генерацией кодов доступа и настройкой параметров игры согласно правилам UNO. Блок-схема данной процедуры изображена на рисунке 2.3.



Рисунок 2.3 **–** Блок-схема функции createRoom

### Функция управления подключениями игроков

Функция handlePlayerConnection является центральным компонентом серверной архитектуры, отвечающим за обработку подключений игроков через WebSocket-соединения и управление их жизненным циклом в рамках игровых сессий. Данная функция обеспечивает установление соединения между клиентом и сервером, регистрацию игроков в игровых комнатах и поддержание стабильного взаимодействия в реальном времени. Блок-схема данной функции изображена на рисунках 2.4.



Рисунок 2.4 **–** Блок-схема функции handlePlayerConnection

Таким образом, функция handlePlayerConnection реализует комплексный механизм управления сетевыми соединениями в многопользовательской игровой среде

### Функция синхронизации игрового состояния

Функция syncGameState является критически важным компонентом серверной архитектуры, отвечающим за поддержание актуального и консистентного игрового состояния между всеми участниками многопользовательской сессии. Данная функция обеспечивает передачу персонализированных игровых данных каждому клиенту через WebSocket-соединения с минимальными задержками и гарантированной доставкой. Блок-схема данной функции изображена на рисунках 2.5.



Рисунок 2.5 **–** Блок-схема функции syncGameState

Таким образом, функция syncGameState представляет собой ключевой элемент архитектуры многопользовательской игры, обеспечивающий бесперебойную синхронизацию игрового состояния между всеми участниками сессии.

# 3 **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

## **3.1 Реализация серверной части приложения**

Серверная часть многопользовательской игры «4 цвета» реализована на Node.js с использованием Express.js для создания RESTful API и Socket.IO для обеспечения взаимодействия в реальном времени. Архитектура построена на основе модульного подхода с четким разделением ответственности между компонентами системы.

Основными компонентами серверной части являются:

* GameService — модуль управления игровой логикой и состоянием игровых сессий;
* SocketHandler — модуль обработки WebSocket-соединений и событий реального времени;
* RoomManager — модуль управления игровыми комнатами и участниками;
* ValidationModule — модуль проверки корректности игровых действий.

Сервер инициализируется с настройкой CORS-политики для обеспечения безопасного взаимодействия с клиентскими приложениями и поддержкой JSON-сериализации для передачи игровых данных. Использование Map-структур для хранения активных игровых комнат обеспечивает быстрый доступ к данным и эффективное управление памятью при множественных одновременных сессиях.

### Функция создания игровой комнаты

Функция **createRoom** является ключевым компонентом инициализации новых игровых сессий в системе. Данная функция отвечает за генерацию уникальных идентификаторов комнат, создание начального игрового состояния и подготовку всех необходимых структур данных для проведения многопользовательской партии.

Процесс создания игровой комнаты начинается с генерации уникального 6-символьного кода, состоящего из букв и цифр в верхнем регистре. Для обеспечения уникальности выполняется проверка на совпадение с существующими кодами в глобальной Map-структуре gameRooms. В случае обнаружения дубликата функция рекурсивно вызывает себя для генерации нового кода.

function createRoom(hostPlayerId) {

// Генерация уникального кода комнаты

const roomCode = Math.random().toString(36).substring(2, 8).toUpperCase();

// Проверка на уникальность кода

if (gameRooms.has(roomCode)) {

return createRoom(hostPlayerId);

}

// Создание объекта игровой комнаты

const newRoom = {

code: roomCode,

hostId: hostPlayerId,

players: [hostPlayerId],

gameState: 'waiting',

deck: [],

currentCard: null,

currentPlayer: 0,

direction: 1,

playerHands: {},

createdAt: new Date()

};

// Инициализация и перемешивание колоды карт

newRoom.deck = generateDeck();

newRoom.deck = shuffleDeck(newRoom.deck);

// Добавление комнаты в глобальный реестр

gameRooms.set(roomCode, newRoom);

return newRoom;

}

**Структура объекта игровой комнаты:**

Созданная комната содержит следующие ключевые параметры:

* **code** — уникальный идентификатор для присоединения игроков;
* **hostId** — идентификатор создателя комнаты;
* **players** — массив участников с начальным элементом-организатором;
* **gameState** — состояние игры ('waiting', 'playing', 'finished');
* **deck** — массив карт для раздачи игрокам;
* **currentCard** — активная карта на игровом столе;
* **currentPlayer** — индекс игрока, который должен сделать ход;
* **direction** — направление хода (1 для по часовой, -1 для против);
* **playerHands** — объект для хранения карт каждого игрока;
* **createdAt** — временная метка создания комнаты.

**Инициализация игровой колоды:**

Функция автоматически создает полную колоду из 108 карт UNO, включающую числовые карты (0-9) в четырех цветах, специальные карты действий (Пропуск хода, Смена направления, Взять две карты) и универсальные карты (Выбор цвета, Взять четыре карты). Колода перемешивается с использованием алгоритма Фишера-Йетса для обеспечения случайного распределения карт.

Созданная комната регистрируется в глобальной Map-структуре gameRooms с ключом в виде сгенерированного кода, что обеспечивает быстрый доступ при присоединении игроков и управлении игровыми сессиями. Функция возвращает объект созданной комнаты для дальнейшего использования в игровой логике.

### Функция управления подключениями игроков

Функция handlePlayerConnection отвечает за обработку подключений игроков через WebSocket-соединения и управление их участием в игровых сессиях. Данная функция обеспечивает установление соединения между клиентом и сервером, регистрацию игроков в комнатах и поддержание стабильного взаимодействия в реальном времени.

При установлении соединения сервер присваивает уникальный идентификатор сокета и регистрирует обработчики для основных игровых событий: присоединение к комнате (joinRoom), выполнение игрового хода (makeMove) и отключение от сервера (disconnect).

function handlePlayerConnection(socket) {

console.log(`Игрок подключился: ${socket.id}`);

// Обработка присоединения к комнате

socket.on('joinRoom', (roomCode, playerName) => {

const room = gameRooms.get(roomCode);

if (!room) {

socket.emit('error', 'Комната не найдена');

return;

}

if (room.players.length >= 10) {

socket.emit('error', 'Комната переполнена');

return;

}

// Добавление игрока в комнату

room.players.push({

id: socket.id,

name: playerName,

cards: [],

isReady: false

});

socket.join(roomCode);

socket.currentRoom = roomCode;

// Уведомление всех игроков о новом участнике

io.to(roomCode).emit('playerJoined', {

playerId: socket.id,

playerName: playerName,

totalPlayers: room.players.length

});

// Обработка отключения игрока

socket.on('disconnect', () => {

handlePlayerDisconnect(socket);

});}

Использование Socket.IO обеспечивает автоматическую маршрутизацию сообщений только между участниками конкретной игровой сессии, что критически важно для корректной работы многопользовательской игры.

### Функция обработки игрового хода

Функция **processPlayerMove** является центральным компонентом игровой логики, отвечающим за обработку и валидацию игровых действий участников. Данная функция обеспечивает корректное выполнение ходов согласно правилам игры UNO, обновление игрового состояния и передачу результатов всем участникам сессии.

function processPlayerMove(socket, moveData) {

const { roomCode, cardIndex, chosenColor } = moveData;

const room = gameRooms.get(roomCode);

if (!room) {

socket.emit('error', 'Игровая комната не найдена');

return;}

// Проверка очередности хода

const currentPlayerIndex = room.currentPlayer;

const currentPlayer = room.players[currentPlayerIndex];

if (currentPlayer.id !== socket.id) {

socket.emit('error', 'Сейчас не ваш ход');

return;

}

// Валидация выбранной карты

const playerHand = currentPlayer.cards;

if (cardIndex < 0 || cardIndex >= playerHand.length) {

socket.emit('error', 'Некорректный индекс карты');

return;

}

const selectedCard = playerHand[cardIndex];

// Проверка соответствия правилам UNO

if (!isValidMove(selectedCard, room.currentCard)) {

socket.emit('error', 'Недопустимый ход по правилам игры');

return;

}

// Выполнение хода

executeMove(room, currentPlayerIndex, cardIndex, selectedCard, chosenColor);

// Проверка условий победы

if (playerHand.length === 1) {

announceWinner(room, currentPlayerIndex);

return;

}

// Переход к следующему игроку

determineNextPlayer(room, selectedCard);

// Синхронизация состояния

syncGameState(roomCode);

}

Таким образом, функция processPlayerMove обеспечивает полную обработку игровой логики UNO, гарантируя соблюдение правил игры, корректное обновление состояния и справедливое определение результатов игровых действий участников.

## **Функции сетевого взаимодействия**

### **3.2.1** Функция обработки сетевого взаимодействия

Функция handleNetworkCommunication является ключевым компонентом серверной архитектуры, отвечающим за управление WebSocket-соединениями и обеспечение взаимодействия в реальном времени между клиентами многопользовательской игры. Данная функция реализует полный цикл сетевой коммуникации от установления соединения до обработки игровых событий.

function handleNetworkCommunication(server) {

// Инициализация Socket.IO сервера

const io = new Server(server, {

cors: {

origin: "\*",

methods: ["GET", "POST"],

credentials: true

},

pingTimeout: 60000,

pingInterval: 25000

});

// Обработка новых соединений

io.on('connection', (socket) => {

console.log(`Клиент подключился: ${socket.id}`);

// Регистрация обработчиков событий

socket.on('joinRoom', (data) => {

handleJoinRoom(socket, data, io);

});

socket.on('gameAction', (data) => {

handleGameAction(socket, data, io);

});

socket.on('chatMessage', (data) => {

handleChatMessage(socket, data, io);

});

socket.on('disconnect', (reason) => {

handleDisconnect(socket, reason, io);

});

// Отправка подтверждения подключения

socket.emit('connectionEstablished', {

socketId: socket.id,

timestamp: new Date().toISOString()

});

});

return io;

}

### **3.2.2** Функция обработки игровых событий

Процедура AcceptClick добавляют ключевые элементы в файл, включая фамилию, ID, пароль из соответствующих компонентов формы. Эти элементы могут быть изменены позже с помощью с других модулей,которые организуют работу с следующими процедурами и функциями.

void \_\_fastcall TAddUserForm::AcceptClick(TObject \*Sender)

{

TStringList \*data = new TStringList();

try

{

// Загрузка существующих данных из файла

if (FileExists("UserData.txt"))

{

data->LoadFromFile("UserData.txt");

}

// Заполнение новых данных

TStringList \*newData = new TStringList();

try

{

newData->Add(IDEdit->Text + ";" + SurnameEdit->Text + ";" + PasswordEdit->Text); // Объединение данных в одну строку

// Добавление новых данных в основной список

data->Add(newData->Text); // Добавляем новую строку

}

\_\_finally

{

delete newData; // Освобождение памяти

}

// Сохранение всех данных обратно в файл

data->SaveToFile("UserData.txt");

}

\_\_finally

{

delete data; // Освобождение памяти

}

// Закрытие формы после записи

this->Close();

### **3.2.3** Функция обработки игровых событий

Система маршрутизации событий обеспечивает корректную передачу игровых действий между участниками. При получении события от клиента функция выполняет валидацию данных, определяет целевую игровую комнату и транслирует событие соответствующим участникам.

function handleGameAction(socket, actionData, io) {

const { roomCode, action, cardData, targetPlayer } = actionData;

// Валидация входящих данных

if (!roomCode || !action) {

socket.emit('error', 'Некорректные данные действия');

return;

}

// Получение игровой комнаты

const room = gameRooms.get(roomCode);

if (!room) {

socket.emit('error', 'Игровая комната не найдена');

return;

}

// Проверка принадлежности игрока к комнате

const player = room.players.find(p => p.socketId === socket.id);

if (!player) {

socket.emit('error', 'Игрок не найден в комнате');

return;

}

// Обработка игрового действия

const result = processGameAction(room, player, action, cardData);

if (result.success) {

// Рассылка обновления всем участникам комнаты

io.to(roomCode).emit('gameUpdate', {

action: action,

player: player.name,

gameState: room.gameState,

timestamp: new Date().toISOString()

});

// Персонализированные данные для конкретного игрока

socket.emit('playerUpdate', {

hand: player.cards,

canPlay: room.currentPlayer === player.index

});

} else {

socket.emit('actionError', result.error);

}

}

# 3.3 Функции для обработки пользовательских данных

Обеспечение безопасности многопользовательской игры является критически важным аспектом разработки, поскольку система должна защищать игровые сессии от различных типов атак и предотвращать читерство. В рамках серверной части реализован комплекс мер безопасности, включающий валидацию входящих данных, защиту от DoS-атак и проверку корректности игровых действий.

### **3.3.1** Валидация входящих данных

Функция validateGameInput отвечает за проверку всех входящих от клиентов данных на соответствие ожидаемому формату и допустимым значениям. Данная функция предотвращает передачу вредоносных данных и обеспечивает стабильность игровых сессий.

function validateGameInput(inputData, expectedSchema) {

const errors = [];

// Валидация обязательных полей

for (const field of expectedSchema.required) {

if (!inputData.hasOwnProperty(field)) {

errors.push(`Отсутствует обязательное поле: ${field}`);

}}

// Валидация типов данных

for (const [fieldName, fieldValue] of Object.entries(inputData)) {

const schema = expectedSchema.properties[fieldName];

if (!schema) continue;

// Проверка типа

if (typeof fieldValue !== schema.type) {

errors.push(`Некорректный тип поля ${fieldName}: ожидается ${schema.type}`);

continue;}

// Проверка ограничений для строк

if (schema.type === 'string') {

if (schema.maxLength && fieldValue.length > schema.maxLength) {

errors.push(`Поле ${fieldName} превышает максимальную длину ${schema.maxLength}`);

}

if (schema.pattern && !schema.pattern.test(fieldValue)) {

errors.push(`Поле ${fieldName} не соответствует требуемому формату`); }

// Проверка диапазона для чисел

if (schema.type === 'number') {

if (schema.min !== undefined && fieldValue < schema.min) {

errors.push(`Поле ${fieldName} меньше минимального значения ${schema.min}`);

}

if (schema.max !== undefined && fieldValue > schema.max) {

errors.push(`Поле ${fieldName} больше максимального значения ${schema.max}`);

}

}

// Проверка допустимых значений

if (schema.enum && !schema.enum.includes(fieldValue)) {

errors.push(`Недопустимое значение для поля ${fieldName}: ${fieldValue}`);

}

}

return {

isValid: errors.length === 0,

errors: errors

};

}

Таким образом, даная функция реализует валидацию входных данных.

### **3.3.2 Функция защиты от DoS-атак и спама**

Функция rateLimitingMiddleware реализует ограничение частоты запросов для предотвращения DoS-атак и спама. Система отслеживает количество запросов от каждого клиента и применяет временные блокировки при превышении лимитов.

class RateLimiter {

constructor() {

this.clients = new Map();

this.limits = {

joinRoom: { maxRequests: 5, windowMs: 60000 }, // 5 попыток за минуту

makeMove: { maxRequests: 30, windowMs: 60000 }, // 30 ходов за минуту

sendMessage: { maxRequests: 10, windowMs: 60000 } // 10 сообщений за минуту

checkRateLimit(socketId, action) {

const clientData = this.clients.get(socketId) || {

requests: {},

blockedUntil: 0};

// Проверка блокировки

if (clientData.blockedUntil > now) {

return {

allowed: false,

reason: 'Клиент временно заблокирован',

retryAfter: Math.ceil((clientData.blockedUntil - now) / 1000)};

const limit = this.limits[action];

if (!limit) {

return { allowed: true };

// Инициализация счетчика для действия

if (!clientData.requests[action]) {

clientData.requests[action] = [];}

const requests = clientData.requests[action];

// Очистка старых запросов

const windowStart = now - limit.windowMs;

clientData.requests[action] = requests.filter(timestamp => timestamp > windowStart);

// Проверка лимита

if (clientData.requests[action].length >= limit.maxRequests) {

// Блокировка клиента на 5 минут при превышении лимита

clientData.blockedUntil = now + 300000;

this.clients.set(socketId, clientData);

return {

allowed: false,

reason: 'Превышен лимит запросов',

retryAfter: 300

// Добавление текущего запроса

clientData.requests[action].push(now);

this.clients.set(socketId, clientData);

return { allowed: true };

// Очистка данных отключившихся клиентов

cleanup(activeSocketIds) {

for (const [socketId] of this.clients) {

if (!activeSocketIds.includes(socketId)) {

this.clients.delete(socketId);}}}}

Таким образом, система безопасности многопользовательской игры обеспечивает защиту от основных типов атак и предотвращает читерство через комплексную валидацию данных, ограничение частоты запросов и проверку корректности игровых действий.

# 4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В процессе тестирования многопользовательской игры «4 цвета» была выявлена критическая проблема с синхронизацией игрового состояния при одновременном выполнении действий несколькими игроками. При быстрых последовательных ходах игроков система некорректно обрабатывала очередность и допускала рассинхронизацию игрового поля между клиентами.

При тестировании с четырьмя активными игроками было обнаружено, что если два игрока практически одновременно пытались разыграть карты (интервал менее 100 мс), серверная часть принимала оба хода, что приводило к нарушению правил игры и различному отображению игрового состояния у разных участников. Логи сервера показывали следующие ошибки:

[ERROR] Race condition detected: Player 2 and Player 3 moves processed simultaneously

[ERROR] Game state inconsistency: currentPlayer mismatch between clients

[WARNING] Socket.IO event flooding detected from multiple clients

Для устранения данной проблемы была реализована система мьютексов (mutex) для каждой игровой комнаты, обеспечивающая атомарность обработки игровых действий:

**const** roomLocks = **new** Map();

**class** GameRoomLock {

constructor() {

**this**.locked = false;

**this**.queue = [];

}

**async** acquire() {

**return** **new** Promise((resolve) => {

**if** (!**this**.locked) {

**this**.locked = true;

resolve();

} **else** {

**this**.queue.push(resolve);

}

});

}

release() {

**if** (**this**.queue.length > 0) {

**const** next = **this**.queue.shift();

next();

} **else** {

**this**.locked = false;

}

}

}

}

Таким образом, проблема синхронизации игрового состояния была успешно решена путем внедрения системы блокировок и версионирования, что обеспечило корректную работу многопользовательской игры при любых сценариях использования.

# 5 **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

# 5.1 Интерфейс программного средства

Качественно разработанный интерфейс способствует повышению удобства и интуитивности использования программного продукта. В данном программном средстве интерфейс выполнен в современном стиле с использованием Vue.js и адаптирован для комфортной игры в карточную игру UNO в многопользовательском режиме.

### Окно входа в игру

Окно входа представляет собой начальный экран приложения, который позволяет пользователю присоединиться к существующей игровой комнате. Интерфейс выполнен в стиле настольной игры с деревянной текстурой фона, что создает атмосферу реальной карточной игры.

Основные элементы окна входа:

* Поле ввода кода комнаты — центральное текстовое поле для введения 6-символьного кода игровой сессии;
* Кнопка "Join" — зеленая кнопка с иконкой входа для подключения к игровой комнате;
* Фоновое оформление — стилизованная деревянная поверхность для создания атмосферы настольной игры.

Интерфейс окна входа изображен на рисунке 5.1.

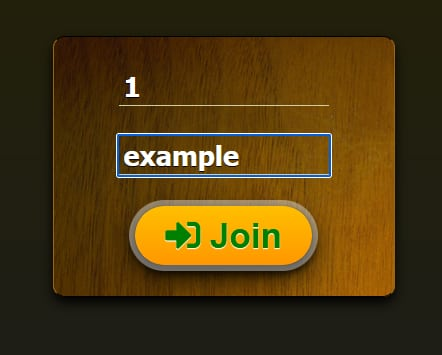


Рисунок 5.1 – Интерфейс главного окна приложения

### Игровое окно

Игровое окно является основным интерфейсом для проведения партии в UNO. Все элементы расположены логично и обеспечивают удобный доступ к игровым функциям без загромождения экрана.

Основные компоненты игрового интерфейса:

**Область карт противника** — верхняя часть экрана отображает карты других игроков (рубашкой вверх) и информацию об их ходе

**Центральная игровая зона** — содержит колоду для взятия карт (слева) и стопку сброса с текущей активной картой (справа)

**Панель карт игрока** — нижняя область экрана с картами в руке текущего игрока, отображаемыми веером

**Информационная панель** — отображение имен игроков, их количества карт и индикации текущего хода

Интерфейс игрового окна изображен на рисунке 5.2.



Рисунок 5.2 – Интерфейс главного окна приложения

В игровом окне реализованы следующие визуальные элементы:

* Карты в руке игрока отображаются с полной детализацией (номер, цвет, специальные символы);
* Активная карта в центре показывает текущий цвет и значение для определения возможных ходов;
* Колода карт представлена стандартной рубашкой UNO для взятия дополнительных карт.

# Управление программным средством

Программное средство «Многопользовательская игра 4 цвета» обеспечивает интуитивно понятное управление через веб-интерфейс с поддержкой как мыши, так и сенсорного ввода на мобильных устройствах.

**Подключение к игре**

Для начала игры пользователь должен получить код игровой комнаты от организатора или создать новую сессию:

* В поле ввода кода комнаты введите 6-символьный идентификатор игровой сессии
* Нажмите кнопку "Join" для подключения к комнате
* При успешном подключении произойдет автоматический переход к игровому интерфейсу

**Игровой процесс**

Управление в игре осуществляется с помощью простых действий:

* **Выбор и разыгрывание карт;**
* Кликните на выбранную карту повторно или перетащите её в центральную область для разыгрывания;
* Система автоматически проверит возможность хода согласно правилам UNO.

**Взятие карт из колоды:**

* При отсутствии подходящих карт кликните на колоду слева для взятия новой карты;
* Взятая карта автоматически добавится в руку игрока;

**Специальные действия:**

* При разыгрывании карт "Выбор цвета" появится диалог выбора нового цвета;
* Система автоматически применяет эффекты специальных карт (пропуск хода, смена направления, взять две карты).

**Окончание игры:**

* При достижении одной карты в руке игрока система автоматически проверяет условия победы;
* Победитель определяется при полном избавлении от всех карт;
* Результаты игры отображаются в специальном окне с возможностью начать новую партию.

Интерфейс обеспечивает плавную работу на различных устройствах и браузерах, адаптируясь под размер экрана и метод ввода пользователя. Адаптивный дизайн позволяет комфортно играть как на настольных компьютерах, так и на планшетах и смартфонах.

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы было разработано программное средство «Многопользовательская игра 4 цвета», представляющее собой современную веб-платформу для онлайн-игры в карточную игру UNO с поддержкой взаимодействия в реальном времени.

Разработанное приложение успешно реализует архитектуру клиент-сервер с использованием современных веб-технологий: серверная часть построена на Node.js с Express.js и Socket.IO для обеспечения real-time коммуникации, клиентская часть реализована на Vue.js с применением компонентного подхода и реактивного программирования.

В процессе работы были достигнуты следующие результаты:

Техническая реализация:

* Создана масштабируемая серверная архитектура с модульным разделением ответственности между компонентами;
* Реализован полный цикл управления игровыми сессиями от создания комнат до синхронизации состояния между участниками;
* Обеспечена корректная обработка всех игровых правил UNO с валидацией ходов и предотвращением читерства;
* Внедрена система безопасности с защитой от DoS-атак, валидацией входящих данных и rate limiting.

Функциональные возможности:

* Создание и управление игровыми комнатами с уникальными кодами доступа;
* Поддержка одновременных подключений до 10 игроков в одной сессии;
* Автоматическая синхронизация игрового состояния между всеми участниками;
* Интуитивно понятный пользовательский интерфейс с анимированными картами;
* Система обработки ошибок и восстановления соединений.

Таким образом, поставленные в начале работы цели были достигнуты, создано стабильно работающее многопользовательское веб-приложение, соответствующее современным стандартам разработки и готовое к практическому использованию. Работа продемонстрировала возможности современного веб-стека для создания интерактивных игровых приложений с поддержкой множественных одновременных соединений и взаимодействием в реальном времени.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Официальная документация Node.js [Интернет-ресурс]. – Режим доступа: <https://nodejs.org/en/docs/>.
2. Официальная документация Vue.js [Интернет-ресурс]. – Режим доступа: <https://vuejs.org/guide/>.
3. Документация Socket.IO [Интернет-ресурс]. – Режим доступа: <https://socket.io/docs/v4/>.
4. Документация npm (Node Package Manager) [Интернет-ресурс]. – Режим доступа: <https://docs.npmjs.com/>.
5. Документация Express.js [Интернет-ресурс]. – Режим доступа: <https://expressjs.com/en/4x/api.html>.
6. MDN Web Docs - Socket.IO [Интернет-ресурс]. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>.
7. Официальные правила игры UNO [Интернет-ресурс]. – Режим доступа: <https://www.mattelgames.com/en-us/cards/uno>.
8. GitHub - Socket.IO Examples [Интернет-ресурс]. – Режим доступа: <https://github.com/socketio/socket.io/tree/master/examples>.
9. Документация Webpack [Интернет-ресурс]. – Режим доступа: <https://webpack.js.org/concepts/>.
10. Хавербеке М. Выразительный JavaScript / [Текст]. – 3-е изд. – СПб.: Питер, 2019. – 448 с. – ISBN 978-5-4461-1226-5

## **ПРИЛОЖЕНИЕ А**

**Текст программы**

let express = require('express');

let application = express();

let server = require('http').Server(application);

let GameServiceFactory = require('./node\_src/GameServiceFactory.js');

let GameServiceRepository = require('./node\_src/GameServiceRepository.js');

let UnitTest = require('./node\_src/UnitTest.js');

//Perform unit tests on some required logic

let unitTest = new UnitTest();

application.get('/', function(request, response){

response.sendFile(\_\_dirname + '/client/index.html');

});

application.use('/client', express.static(\_\_dirname + '/client'));

//server.listen(3000);

server.listen(80);

console.log('============= Server started ===============');

let io = require('socket.io')(server, {});

let gameServiceRepository = new GameServiceRepository();

let gameServiceFactory = new GameServiceFactory();

io.sockets.on('connection', function(socket) {

console.log('Socket connection');

socket.on('create', function(room) {

console.log('Join room: ' + room + ' socketId: ' + socket.id);

socket.join(room);

let gameService = gameServiceRepository.findById(room);

if(!gameService){

gameService = gameServiceFactory.create("UNO", room);

gameServiceRepository.insert(gameService);

}

socket.use(function(packet){

gameService.handleAction(socket, packet[0], packet[1]);

Object.keys(io.sockets.sockets).forEach(function(id) {

let data = gameService.getClientResponseData(id);

if(data){

io.to(id).emit('state', data);

}

});

});

});

});

{

"name": "game",

"version": "1.0.0",

"description": "",

"main": "index.js",

"repository": "https://github.com/houseofbits/nodejs-uno-game",

"scripts": {

"build": "vue-cli-service build --mode=development",

"serve": "vue-cli-service serve"

},

"author": "",

"license": "ISC",

"dependencies": {

"@fortawesome/fontawesome-free": "^5.13.0",

"express": "^4.17.1",

"gsap": "^3.2.6",

"socket.io": "^2.3.0",

"vue": "^2.6.11",

"vue-cookies": "^1.7.0",

"vue-socket.io": "^3.0.7"

},

"devDependencies": {

"@vue/cli": "^4.3.1",

"@vue/cli-service": "^4.3.1",

"vue-template-compiler": "^2.6.11"

},

"vue": {

"publicPath": "client",

"outputDir": "client",

"pages": {

"index": {

"entry": "src/main.js",

"template": "src/index.html",

"filename": "index.html"

}

}

}

}

<template>

<div class="frame" :style="{width:config.boardWidth+'px', height:config.boardHeight+'px'}">

<Authorize :client="state.client" :socket="socket"></Authorize>

<Board v-if="state.client.code">

<Scores :clients="state.clients" :position="config.scoresPos"></Scores>

<Card v-for="card in state.game.cards"

:ref="'card'+card.id"

:card="card"

:key="card.id"

:clickHandler="cardOnClick"

:transitionFinishHandler="cardTransitionFinish"

:timeline="config.timeline"

:active="state.client.turn"></Card>

<div v-if="config.playersInitialized && !state.game.winner && state.game.ready" >

<NamePlate v-for="(client, index) in state.clients" :key="'client\_'+index" :client="client" :position="namePosition(client.name)"></NamePlate>

</div>

</Board>

<div class="board-overlay" v-if="overlayVisible">

<PopupReady v-if="!state.game.winner && !state.game.ready" :buttonHandler="ready" :showButton="!state.client.ready"></PopupReady>

<PopupWon v-if="state.game.winner" :buttonHandler="ready" :winner="state.game.winner" :showButton="!state.client.ready"></PopupWon>

<PopupSpecial v-if="config.specialCard" :clickHandler="playCardSpecial" :card="config.specialCard"></PopupSpecial>

<PopupTake v-if="!config.specialCard && state.client.takeOrLeave" :card="state.client.takeOrLeave" :takeHandler="takeCard" :leaveHandler="playCardTOL"></PopupTake>

</div>

</div>

</template>

<script>

import Board from "./components/Board"

import Card from "./components/Card"

import PopupReady from "./components/PopupReady"

import PopupWon from "./components/PopupWon"

import PopupSpecial from "./components/PopupSpecial"

import PopupTake from "./components/PopupTake"

import NamePlate from "./components/NamePlate"

import Scores from "./components/Scores"

import Authorize from "./components/Authorize"

import testDataNew from "../public/testDataNew.json"

import ConfigMixin from "./mixins/Config"

import ClientMixin from "./mixins/Client"

const OWNER\_DRAW\_DECK = "draw"

const OWNER\_DISCARD\_DECK = "dsc"

export default {

name: "UnoGame",

props: {

socket: { type: Object },

},

mixins: [

ConfigMixin,

ClientMixin

],

components: {

Board,

Card,

Authorize,

PopupReady,

PopupWon,

PopupSpecial,

PopupTake,

NamePlate,

Scores

},

data: function() {

return {

state:{

clients:[],

game:{

cards:[],

events:[]

},

}

}

},

computed:{

overlayVisible:function(){

return this.state.client.code

&& ((!this.state.game.winner && !this.state.game.ready)

|| this.state.game.winner

|| this.config.specialCard

|| (!this.config.specialCard && this.state.client.takeOrLeave)

);

}

},

methods:{

playCard:function(card){

if(card.type === 'kc' || card.type === 'kg'){

this.config.specialCard = card;

}else{

this.state.client.turn = false;

this.cardSetOwner(card, OWNER\_DISCARD\_DECK);

this.config.specialCard = false;

this.socket.emit('place', {'client': this.state.client, card:card});

}

},

playCardSpecial:function(cardId, type){

let card = this.state.game.cards.find(function(elem){

return elem.id == cardId;

});

if(typeof card !== 'undefined'){

card.type = type;

this.playCard(card);

}

},

playCardTOL:function(cardId){

let card = this.state.game.cards.find(function(elem){

return elem.id == cardId;

});

if(typeof card !== 'undefined'){

this.playCard(card);

}

},

takeCard:function(){

this.socket.emit('take', {'client': this.state.client});

},

cardSetOwner:function(card, owner){

switch(owner){

case OWNER\_DRAW\_DECK:

this.transitionToDrawDeck(card);

break;

case OWNER\_DISCARD\_DECK:

this.transitionToDiscardDeck(card);

break;

default:

this.transitionToHand(card, owner);

};

},

transitionToDrawDeck:function(card){

card.owner = OWNER\_DRAW\_DECK;

card.transform.x = this.config.drawPos.x + (5 - (Math.random()\*10));

card.transform.y = this.config.drawPos.y + (5 - (Math.random()\*10));

card.transform.angle = (5 - (Math.random()\*10));

card.transform.z = 150 + parseInt(card.moveId);

card.transform.scale = 1;

card.transform.d = 0;

},

transitionToDiscardDeck:function(card){

card.owner = OWNER\_DISCARD\_DECK;

card.transform.x = this.config.discardPos.x + (10 - (Math.random()\*20));

card.transform.y = this.config.discardPos.y + (10 - (Math.random()\*20));

card.transform.angle = (10 - (Math.random()\*20));

card.transform.z = 200;

card.transform.scale = 1;

card.transform.d = 0.4;

},

transitionToHand:function(card, owner){

if(typeof this.config.players[owner] !== 'undefined'){

let config = this.config.players[owner];

card.owner = owner;

card.transform.x = config.x + (5 - (Math.random()\*10));

card.transform.y = config.y + (5 - (Math.random()\*10));

card.transform.z = 300;

card.transform.angle = (10 - (Math.random()\*20));

card.transform.scale = config.scale;

card.transform.d = 0.3;

}

},

cardOnClick:function(card){

if(card.owner == OWNER\_DRAW\_DECK && card.nextMoveValid && this.state.client.turn){

this.takeCard();

this.cardSetOwner(card, this.self);

this.state.client.turn = false;

}else if(card.owner == this.self && card.nextMoveValid){

this.playCard(card);

}

},

cardTransitionFinish:function(card){

for(let i=0; i<this.state.clients.length; i++){

this.updateHand(this.state.clients[i].name);

}

},

initDeck:function(data){

this.state.game.cards = data;

for(let i=0; i<this.state.game.cards.length; i++){

this.cardSetOwner(this.state.game.cards[i], this.state.game.cards[i].owner);

}

},

updateHand:function(player){

let clientCards = [];

for(let i=0; i<this.state.game.cards.length; i++){

if(this.state.game.cards[i].owner == player){

clientCards.push(this.state.game.cards[i]);

}

}

clientCards.sort(function(a,b){

if(a.type < b.type)return -1;

if(a.type > b.type)return 1;

return 0;

});

let config = this.config.players[player];

let scale = (player === this.self)?1.2:this.config.opponentCardScale;

let pivotSetting = (player === this.self)?400:200;

let angleRangeDiv = (player === this.self)?7:9;

let maxAngle = (player === this.self)?50:35;

let cardsCount = clientCards.length;

let angleRange = (cardsCount \* 30) / angleRangeDiv;

angleRange = Math.min(angleRange, maxAngle);

let angleMin = -(angleRange / 2);

let angleMax = (angleRange / 2);

let angleStep = (angleMax - angleMin) / (cardsCount - 1);

let angle = angleMin;

let posZ = 50;

for(let i=0; i<clientCards.length; i++){

let cosi = (1 - Math.cos(angle \* (Math.PI / 180))) \* pivotSetting;

let sini = (Math.sin(angle \* (Math.PI / 180))) \* pivotSetting;

clientCards[i].transform.angle = angle;

if(typeof config !== 'undefined'){

clientCards[i].transform.x = config.x + sini;

clientCards[i].transform.y = config.y + cosi;

}

clientCards[i].transform.z = posZ;

clientCards[i].transform.scale = scale;

clientCards[i].transform.d = 0.05;

posZ++;

angle += angleStep;

}

},

updateDiscardDeck:function(){

let clientCards = [];

for(let i=0; i<this.state.game.cards.length; i++){

if(this.state.game.cards[i].owner == OWNER\_DISCARD\_DECK){

clientCards.push(this.state.game.cards[i]);

}

}

clientCards.sort(function(a,b){

if(parseInt(a.moveId) < parseInt(b.moveId))return -1;

if(parseInt(a.moveId) > parseInt(b.moveId))return 1;

return 0;

});

let posZ = 0;

for(let i=0; i<clientCards.length; i++){

clientCards[i].transform.z = posZ;

posZ++;

}

},

processEvents:function(){

for(let i=0; i<this.state.game.events.length; i++){

let event = this.state.game.events[i];

if(typeof this.state.game.cards[event.cardId] !== 'undefined'){

this.cardSetOwner(this.state.game.cards[event.cardId], event.newOwner);

}

}

},

updateState:function(cards){

for(let i=0; i<this.state.game.cards.length; i++){

if(typeof cards[i] === 'undefined')continue;

this.state.game.cards[i].moveId = cards[i].moveId;

if(cards[i].owner !== this.state.game.cards[i].owner){

this.cardSetOwner(this.state.game.cards[i], cards[i].owner);

}

this.state.game.cards[i].owner = cards[i].owner;

this.state.game.cards[i].type = cards[i].type;

this.state.game.cards[i].nextMoveValid = cards[i].nextMoveValid;

}

},

gameStateResponse(response){

//console.log(response);

this.state.client = response.client;

this.state.clients = response.clients;

this.state.game.events = response.game.events;

this.state.game.winner = response.game.winner;

this.state.game.ready = response.game.ready;

if(!this.config.initialized && response.game.cards.length > 0){

this.config.initialized = true;

this.initDeck(response.game.cards);

}else if(this.config.initialized && response.game.events.length > 0){

if(!this.config.playersInitialized){

this.config.playersInitialized = true;

this.initClientsConfig(this.state.clients);

}

this.processEvents();

}

this.updateState(response.game.cards);

this.updateDiscardDeck();

},

},

mounted:function () {

this.socket.on('state', this.gameStateResponse);

//this.gameStateResponse(testDataNew);

},

beforeDestroy:function () {

}

}

</script>

<style scoped>

.frame{

top:5px;

position:relative;

width:800px;

margin:0 auto;

}

.board-overlay{

position:absolute;

width:100%;

height:100%;

transform: translateZ(500px);

}

</style>

import Vue from 'vue'

import ClientApp from './clientApp.vue'

import VueCookies from 'vue-cookies'

import  '../node\_modules/@fortawesome/fontawesome-free/css/all.css'

Vue.use(VueCookies)

new Vue({

    el: '#clientApp',

    render: h => h(ClientApp)

});

<!DOCTYPE html>

<html lang="en">

<head>

    <title>UNO Node.js</title>

    <meta charset="UTF-8"

        name="viewport"

        content="width=device-width, initial-scale=1">

</head>

<body>

    <div id="clientApp"></div>

</body>

</html>

<template>

<div>

<UnoGame :socket="socket"></UnoGame>

</div>

</template>

<script>

import socket from 'socket.io-client'

import UnoGame from "./UnoGame"

export default {

name: "clientApp",

data: function() {

return {

socket: socket("http://" + window.location.hostname + ":" + window.location.port),

}

},

components: {

UnoGame

},

}

</script>

<style scoped>

</style>

<style>

body {

background: #1b1b21 url('../public/img/page\_bg.png') repeat-x;

margin:0 auto;

width:100%;

font: 12px normal Verdana, Arial, Helvetica, sans-serif;

user-select: none;

overflow: hidden;

transform: scale(1.0);

}

a{

text-decoration:none;

color:inherit;

}

</style>

export default {

    data: function () {

      return {

        state:{

            client:{

                name:'',

                code:null

            }

        }

      }

    },

    computed:{

        self:function(){

            return this.state.client.name;

        }

    },

    methods:{

        ready:function () {

            this.state.client.ready = true;

            this.socket.emit('begin', {'client': this.state.client});

        }

    }

}

import gsap from "gsap";

export default {

  data: function () {

    return {

        config:{

          boardWidth:600,

          boardHeight:650,

          initialized:false,

          playersInitialized:false,

          drawPos:{x:150,y:240},

          discardPos:{x:300,y:210},

          scoresPos:{x:370,y:210},

          players:{},

          timeline:null,

          specialCard:false,

          opponentCardScale:0.5,

          screenWidth:null,

          screenHeight:null,

          opponentPosOffset:{

            1:[20],

            2:[20,20],

            3:[110,20,110],

            4:[110,20,110,20]

          }

      }

    }

  },

  methods:{

    namePosition:function(clientName){

      if(typeof this.config.players[clientName] !== 'undefined'){

          let pos = this.config.players[clientName];

          if(clientName === this.self){

              return {

                  x:pos.x,

                  y:pos.y + 170

              };

          }else{

              return {

                  x:pos.x,

                  y:pos.y + 120

              };

          }

      }

    },

    initClientsConfig:function(clients){

      this.config.players = {};

      let count = clients.length - 1;

      let marg = this.config.boardWidth / count;

      let index = 0;

      let offsets = this.config.opponentPosOffset[count];

      for(let i=0; i<clients.length; i++){

          if(clients[i].name == this.self){

              this.config.players[clients[i].name] = {

                  x:(this.config.boardWidth \* 0.5),

                  y:this.config.boardHeight - 220,

                  scale:1.2

              };

          }else{

             this.config.players[clients[i].name] = {

                  x:((marg / 2) + (index \* marg)),

                  y:offsets[index],

                  scale: this.config.opponentCardScale

              };

              index++;

          }

      }

    },

  },

  mounted:function(){

    this.config.timeline = gsap.timeline();

    this.config.screenWidth = Math.max(document.documentElement.clientWidth || 0, window.innerWidth || 0);

    this.config.screenHeigh = Math.max(document.documentElement.clientHeight || 0, window.innerHeight || 0);

    this.config.boardWidth = Math.min(600, this.config.screenWidth);

    this.config.boardHeight = 800;

    this.config.drawPos.y = (this.config.boardHeight \* 0.5) - 60;

    this.config.discardPos.y = (this.config.boardHeight \* 0.5) - 80;

    this.config.scoresPos.y = (this.config.boardHeight \* 0.5) - 60;

  }

}

**<template>**

**<div class="scores" :style="{left:position.x+'px', top:position.y+'px'}">**

**<div class="column" v-for="(client,index) in clients" :key="index">**

**<div class="cell">{{client.name}}</div>**

**<div class="cell" v-for="(score,ind2) in client.score" :key="ind2"><i class='far' :class="scoreClass(score, index)"></i>{{scoreVal(score)}}</div>**

**</div>**

**</div>**

**</template>**

**<script>**

**export default {**

**name: "Popup",**

**props:['clients', 'position'],**

**methods:{**

**scoreClass:function(score, index){**

**let smileys = [**

**"fa-smile",**

**"fa-grin-squint",**

**"fa-grin-beam",**

**"fa-laugh-squint",**

**"fa-smile-beam",**

**"fa-laugh",**

**"fa-laugh-beam"**

**];**

**if(score === '-'){**

**return smileys[index];**

**}**

**return null;**

**},**

**scoreVal:function(score){**

**if(score === '-')return '';**

**return score;**

**}**

**}**

**}**

**</script>**

**<style scoped>**

**.scores{**

**position: absolute;**

**width:250px;**

**height:150px;**

**right:20px;**

**top:220px;**

**transform: rotateZ(3deg);**

**}**

**.column{**

**position:relative;**

**width:55px;**

**height:100%;**

**float:left;**

**border-left:solid 1px white;**

**}**

**.cell{**

**position:relative;**

**width:100%;**

**height:25px;**

**color: white;**

**text-align: center;**

**line-height: 25px;**

**font-size: 16px;**

**text-shadow: 0px 1px 2px black;**

**}**

**.cell:nth-child(1){**

**border-bottom:solid 1px white;**

**}**

**.column:first-child{**

**border-left:none;**

**}**

**</style>**

**<template>**

**<Popup>**

**<template v-slot:topRow>**

**<div class="won-message">{{winner}} won!</div>**

**</template>**

**<template v-slot:bottomRow v-if="showButton">**

**<Button faIcon="paper-plane" :clickHandler="buttonHandler">Next round</Button>**

**</template>**

**</Popup>**

**</template>**

**<script>**

**import Popup from "./Popup"**

**import Button from "./Button"**

**export default {**

**name: "PopupWon",**

**components: {Popup, Button},**

**props:['buttonHandler', 'winner', 'showButton']**

**}**

**</script>**

**<style scoped>**

**.won-message{**

**text-align: center;**

**font-family: "Trebuchet MS", Helvetica, sans-serif;**

**letter-spacing: 0px;**

**word-spacing: 0.2px;**

**font-weight: 700;**

**filter: drop-shadow(0px 4px 4px black);**

**height:50px;**

**line-height:50px;**

**font-size: 58px;**

**background: linear-gradient(to bottom, rgba(248,80,50,1) 0%, rgba(241,111,92,1) 50%, rgba(246,41,12,1) 51%, rgba(240,47,23,1) 71%, rgba(231,56,39,1) 100%);**

**background-clip: text;**

**-webkit-text-fill-color: transparent;**

**margin-bottom:30px;**

**}**

**</style>**

**<template>**

**<Popup>**

**<template v-slot:topRow>**

**<div class="card-center">**

**<CardTemplate :type="card.type"></CardTemplate>**

**</div>**

**</template>**

**<template v-slot:bottomRow>**

**<div class="button-float">**

**<Button faIcon="arrow-circle-up" :clickHandler="leave">Place</Button>**

**</div>**

**<div class="button-float">**

**<Button faIcon="arrow-circle-down" :clickHandler="takeHandler">Take</Button>**

**</div>**

**</template>**

**</Popup>**

**</template>**

**<script>**

**import Popup from "./Popup"**

**import CardTemplate from "./CardTemplate"**

**import Button from "./Button"**

**export default {**

**name: "PopupTake",**

**props:['takeHandler','leaveHandler','card'],**

**components: {Popup, CardTemplate, Button},**

**methods:{**

**leave:function(){**

**this.leaveHandler(this.card.id);**

**}**

**}**

**}**

**</script>**

**<style scoped>**

**.card-center{**

**display: inline-block;**

**}**

**.button-float{**

**display: inline-block;**

**margin-left: 15px;**

**margin-right: 15px;**

**margin-top: 20px;**

**}**

**</style>**

**<template>**

**<Popup>**

**<template v-slot:topRow>**

**<div class="card-float">**

**<CardTemplate :type="'r'+getNumber()" :clickHandler="click" hover="true"></CardTemplate>**

**</div>**

**<div class="card-float">**

**<CardTemplate :type="'y'+getNumber()" :clickHandler="click" hover="true"></CardTemplate>**

**</div>**

**<div class="card-float">**

**<CardTemplate :type="'g'+getNumber()" :clickHandler="click" hover="true"></CardTemplate>**

**</div>**

**<div class="card-float">**

**<CardTemplate :type="'b'+getNumber()" :clickHandler="click" hover="true"></CardTemplate>**

**</div>**

**</template>**

**</Popup>**

**</template>**

**<script>**

**import Popup from "./Popup"**

**import CardTemplate from "../components/CardTemplate"**

**export default {**

**name: "PopupSpecial",**

**props:['clickHandler', 'card'],**

**components: {Popup, CardTemplate},**

**methods:{**

**getNumber:function(){**

**return this.card.type.charAt(1);**

**},**

**click:function(cardType){**

**this.clickHandler(this.card.id, cardType);**

**}**

**}**

**}**

**</script>**

**<style scoped>**

**.card-float{**

**display: inline-block;**

**margin-left: 5px;**

**margin-right: 5px;**

**}**

**</style>**

**<template>**

**<Popup>**

**<template v-slot:topRow>**

**<div class="wait-message">Waiting for game to start</div>**

**</template>**

**<template v-slot:bottomRow v-if="showButton">**

**<div class="spacer"></div>**

**<Button faIcon="paper-plane" :clickHandler="buttonHandler">Ready!</Button>**

**</template>**

**</Popup>**

**</template>**

**<script>**

**import Popup from "./Popup"**

**import Button from "./Button"**

**export default {**

**name: "PopupReady",**

**props:['buttonHandler', 'showButton'],**

**components: {Popup, Button}**

**}**

**</script>**

**<style scoped>**

**.wait-message{**

**text-align: center;**

**font-family: "Trebuchet MS", Helvetica, sans-serif;**

**letter-spacing: 0px;**

**word-spacing: 0.2px;**

**font-weight: 700;**

**filter: drop-shadow(0px 4px 4px black);**

**height:50px;**

**line-height:45px;**

**font-size: 45px;**

**background: linear-gradient(to bottom, rgba(164,179,87,1) 0%, rgba(117,137,12,1) 100%);**

**background-clip: text;**

**-webkit-text-fill-color: transparent;**

**}**

**.spacer{**

**margin-top:30px;**

**}**

**</style>**

**<template>**

**<div class="popup">**

**<div class="row">**

**<slot name="topRow"></slot>**

**</div>**

**<div class="row">**

**<slot name="bottomRow"></slot>**

**</div>**

**</div>**

**</template>**

**<script>**

**export default {**

**name: "Popup",**

**}**

**</script>**

**<style scoped>**

**.popup{**

**width:100%;**

**height:auto;**

**position:relative;**

**left:0;**

**margin:0;**

**padding:0;**

**top:410px;**

**background: rgba(0,0,0,0.6);**

**/\*border:solid 1px red;\*/**

**}**

**.popup:before{**

**content:"";**

**display: block;**

**width:100%;**

**height:20px;**

**top:-20px;**

**background: linear-gradient(to bottom, rgba(0,0,0,0) 0%, rgba(0,0,0,0.6) 100%);**

**position:relative;**

**}**

**.popup:after{**

**content:"";**

**display: block;**

**width:100%;**

**height:20px;**

**bottom:-20px;**

**background: linear-gradient(to top, rgba(0,0,0,0) 0%, rgba(0,0,0,0.6) 100%);**

**position:relative;**

**}**

**.popup .row{**

**display: block;**

**width:100%;**

**position: relative;**

**text-align: center;**

**/\*border:solid 1px blue;\*/**

**}**

**</style>**

**<template>**

**<div class="name" :style="transform">**

**<div class="client-name" :class="{active:client.turn}">**

**<span>{{client.name}}</span>**

**</div>**

**</div>**

**</template>**

**<script>**

**export default {**

**name: "NamePlate",**

**props: ['client', 'position'],**

**computed:{**

**transform:function(){**

**return {**

**transform:'translate3d('+this.position.x+'px, '+this.position.y+'px, 300px)'**

**};**

**}**

**}**

**}**

**</script>**

**<style scoped>**

**.name{**

**width:150px;**

**height:35px;**

**position:absolute;**

**left:0;**

**top:0;**

**}**

**.client-name{**

**width:150px;**

**height:35px;**

**line-height:35px;**

**position:relative;**

**left:-75px;**

**top:0;**

**text-align: center;**

**vertical-align: top;**

**font-family: "Trebuchet MS", Helvetica, sans-serif;**

**font-size: 28px;**

**letter-spacing: 0px;**

**word-spacing: 0.2px;**

**font-weight: 700;**

**filter: drop-shadow(0px 5px 3px black);**

**background: linear-gradient(to bottom, rgba(252,234,187,1) 0%, rgba(252,205,77,1) 61%, rgba(248,181,0,1) 62%, rgba(251,223,147,1) 100%);**

**background-clip: text;**

**-webkit-text-fill-color: transparent;**

**opacity: 0.6;**

**}**

**.client-name.active{**

**opacity: 1;**

**}**

**.client-name span{**

**line-height: 25px;**

**/\*border-bottom:solid 4px rgba(252,205,77,.3);\*/**

**padding-left: 40px;**

**padding-right: 40px;**

**border-bottom: none;**

**}**

**.hand-cards-transform{**

**position: absolute;**

**transform: translate(350px,580px);**

**}**

**.client-name.active span{**

**background: linear-gradient(to bottom, rgba(241,231,103,0) 0%, rgba(254,182,69,0.2) 100%);**

**border-bottom:solid 4px rgba(255, 251, 0, 0.8);**

**}**

**</style>**

**<template>**

**</template>**

**<script>**

**export default {**

**name: "ErrorMessage",**

**props: ['messages']**

**}**

**</script>**

**<style scoped>**

**.error-message{**

**display:block;**

**position: absolute;**

**width: 100%;**

**height:40px;**

**line-height: 40px;**

**text-align: center;**

**background-color: rgba(255,0,0,0.5);**

**color:yellow;**

**font-weight: bolder;**

**}**

**</style>**

**<template>**

**<div class="playing\_card" @click="clickHandler?clickHandler(type):null" :class="[type, hoverable]"><div></div><div></div><div></div></div>**

**</template>**

**<script>**

**export default {**

**name: "CardTemplate",**

**props: ['type', 'clickHandler', 'hover'],**

**computed:{**

**hoverable:function(){return (this.hover === 'true') ? 'card-hoverable' : '';}**

**}**

**}**

**</script>**

**<style scoped>**

**.playing\_card{**

**display:block;**

**position:relative;**

**/\*border:solid 1px #555555;\*/**

**border-radius:8px;**

**width:93px;**

**height:140px;**

**box-shadow:-3px -4px 3px rgba(0,0,0,0.3);**

**}**

**.card-hoverable:hover{**

**box-shadow: 0px 0px 1px 5px rgba(255,234,0,1);**

**}**

**.r0,.r1,.r2,.r3,.r4,.r5,.r6,.r7,.r8,.r9,.rp,.rn,.rr,.rc,.rg{**

**background:url('../../public/img/outl.png') no-repeat, url('../../public/img/grad.png') repeat-x 0 6px;**

**}**

**.g0,.g1,.g2,.g3,.g4,.g5,.g6,.g7,.g8,.g9,.gp,.gn,.gr,.gc,.gg{**

**background:url('../../public/img/outl.png') no-repeat, url('../../public/img/grad.png') repeat-x 0 -122px;**

**}**

**.b0,.b1,.b2,.b3,.b4,.b5,.b6,.b7,.b8,.b9,.bp,.bn,.br,.bc,.bg{**

**background:url('../../public/img/outl.png') no-repeat, url('../../public/img/grad.png') repeat-x 0 -250px;**

**}**

**.y0,.y1,.y2,.y3,.y4,.y5,.y6,.y7,.y8,.y9,.yp,.yn,.yr,.yc,.yg{**

**background: url('../../public/img/outl.png') no-repeat, url('../../public/img/grad.png') repeat-x 0 -378px;**

**}**

**.kg,.kc{**

**background: url('../../public/img/outl.png') no-repeat, url('../../public/img/grad.png') repeat-x 0 -506px;**

**}**

**.playing\_card>div:nth-child(1){**

**position:absolute;**

**top:9px;**

**left:9px;**

**}**

**.playing\_card>div:nth-child(2){**

**position:absolute;**

**bottom:8px;**

**right:9px;**

**}**

**.playing\_card>div:nth-child(3){**

**position:absolute;**

**top:35px;**

**left:11px;**

**width:69px;**

**height:74px;**

**}**

**.r0>div:nth-child(1),.r0>div:nth-child(2),**

**.g0>div:nth-child(1),.g0>div:nth-child(2),**

**.b0>div:nth-child(1),.b0>div:nth-child(2),**

**.y0>div:nth-child(1),.y0>div:nth-child(2){**

**width:13px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat;**

**}**

**.r0>div:nth-child(3),**

**.g0>div:nth-child(3),**

**.b0>div:nth-child(3),**

**.y0>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat;**

**}**

**.r1>div:nth-child(1),.r1>div:nth-child(2),**

**.g1>div:nth-child(1),.g1>div:nth-child(2),**

**.b1>div:nth-child(1),.b1>div:nth-child(2),**

**.y1>div:nth-child(1),.y1>div:nth-child(2){**

**width:11px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -13px 0;**

**}**

**.r1>div:nth-child(3),**

**.g1>div:nth-child(3),**

**.b1>div:nth-child(3),**

**.y1>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -69px 0;**

**}**

**.r2>div:nth-child(1),.r2>div:nth-child(2),**

**.g2>div:nth-child(1),.g2>div:nth-child(2),**

**.b2>div:nth-child(1),.b2>div:nth-child(2),**

**.y2>div:nth-child(1),.y2>div:nth-child(2){**

**width:13px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -24px 0;**

**}**

**.r2>div:nth-child(3),**

**.g2>div:nth-child(3),**

**.b2>div:nth-child(3),**

**.y2>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -138px 0;**

**}**

**.r3>div:nth-child(1),.r3>div:nth-child(2),**

**.g3>div:nth-child(1),.g3>div:nth-child(2),**

**.b3>div:nth-child(1),.b3>div:nth-child(2),**

**.y3>div:nth-child(1),.y3>div:nth-child(2){**

**width:13px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -37px 0;**

**}**

**.r3>div:nth-child(3),**

**.g3>div:nth-child(3),**

**.b3>div:nth-child(3),**

**.y3>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -207px 0;**

**}**

**.r4>div:nth-child(1),.r4>div:nth-child(2),**

**.g4>div:nth-child(1),.g4>div:nth-child(2),**

**.b4>div:nth-child(1),.b4>div:nth-child(2),**

**.y4>div:nth-child(1),.y4>div:nth-child(2){**

**width:13px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -50px 0;**

**}**

**.r4>div:nth-child(3),**

**.g4>div:nth-child(3),**

**.b4>div:nth-child(3),**

**.y4>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -276px 0;**

**}**

**.r5>div:nth-child(1),.r5>div:nth-child(2),**

**.g5>div:nth-child(1),.g5>div:nth-child(2),**

**.b5>div:nth-child(1),.b5>div:nth-child(2),**

**.y5>div:nth-child(1),.y5>div:nth-child(2){**

**width:13px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -63px 0;**

**}**

**.r5>div:nth-child(3),**

**.g5>div:nth-child(3),**

**.b5>div:nth-child(3),**

**.y5>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -345px 0;**

**}**

**.r6>div:nth-child(1),.r6>div:nth-child(2),**

**.g6>div:nth-child(1),.g6>div:nth-child(2),**

**.b6>div:nth-child(1),.b6>div:nth-child(2),**

**.y6>div:nth-child(1),.y6>div:nth-child(2){**

**width:14px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -76px 0;**

**}**

**.r6>div:nth-child(3),**

**.g6>div:nth-child(3),**

**.b6>div:nth-child(3),**

**.y6>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -414px 0;**

**}**

**.r7>div:nth-child(1),.r7>div:nth-child(2),**

**.g7>div:nth-child(1),.g7>div:nth-child(2),**

**.b7>div:nth-child(1),.b7>div:nth-child(2),**

**.y7>div:nth-child(1),.y7>div:nth-child(2){**

**width:11px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -90px 0;**

**}**

**.r7>div:nth-child(3),**

**.g7>div:nth-child(3),**

**.b7>div:nth-child(3),**

**.y7>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -483px 0;**

**}**

**.r8>div:nth-child(1),.r8>div:nth-child(2),**

**.g8>div:nth-child(1),.g8>div:nth-child(2),**

**.b8>div:nth-child(1),.b8>div:nth-child(2),**

**.y8>div:nth-child(1),.y8>div:nth-child(2){**

**width:13px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -101px 0;**

**}**

**.r8>div:nth-child(3),**

**.g8>div:nth-child(3),**

**.b8>div:nth-child(3),**

**.y8>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -552px 0;**

**}**

**.r9>div:nth-child(1),.r9>div:nth-child(2),**

**.g9>div:nth-child(1),.g9>div:nth-child(2),**

**.b9>div:nth-child(1),.b9>div:nth-child(2),**

**.y9>div:nth-child(1),.y9>div:nth-child(2){**

**width:13px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -114px 0;**

**}**

**.r9>div:nth-child(3),**

**.g9>div:nth-child(3),**

**.b9>div:nth-child(3),**

**.y9>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -621px 0;**

**}**

**.rp>div:nth-child(1),.rp>div:nth-child(2),**

**.gp>div:nth-child(1),.gp>div:nth-child(2),**

**.bp>div:nth-child(1),.bp>div:nth-child(2),**

**.yp>div:nth-child(1),.yp>div:nth-child(2){**

**width:24px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -127px 0;**

**}**

**.rp>div:nth-child(3),**

**.gp>div:nth-child(3),**

**.bp>div:nth-child(3),**

**.yp>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -828px 0;**

**}**

**.rr>div:nth-child(1),.rr>div:nth-child(2),**

**.gr>div:nth-child(1),.gr>div:nth-child(2),**

**.br>div:nth-child(1),.br>div:nth-child(2),**

**.yr>div:nth-child(1),.yr>div:nth-child(2){**

**width:13px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -193px 0;**

**}**

**.rr>div:nth-child(3),**

**.gr>div:nth-child(3),**

**.br>div:nth-child(3),**

**.yr>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -759px 0;**

**}**

**.rn>div:nth-child(1),.rn>div:nth-child(2),**

**.gn>div:nth-child(1),.gn>div:nth-child(2),**

**.bn>div:nth-child(1),.bn>div:nth-child(2),**

**.yn>div:nth-child(1),.yn>div:nth-child(2){**

**width:18px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -175px 0;**

**}**

**.rn>div:nth-child(3),**

**.gn>div:nth-child(3),**

**.bn>div:nth-child(3),**

**.yn>div:nth-child(3){**

**background:url('../../public/img/symbols.png') no-repeat -690px 0;**

**}**

**.rg>div:nth-child(1),.rg>div:nth-child(2),**

**.gg>div:nth-child(1),.gg>div:nth-child(2),**

**.bg>div:nth-child(1),.bg>div:nth-child(2),**

**.yg>div:nth-child(1),.yg>div:nth-child(2),**

**.kg>div:nth-child(1),.kg>div:nth-child(2){**

**width:24px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -151px 0;**

**}**

**.rg>div:nth-child(3),**

**.gg>div:nth-child(3),**

**.bg>div:nth-child(3),**

**.yg>div:nth-child(3),**

**.kg>div:nth-child(3){**

**width:70px;**

**height:76px;**

**top:33px;**

**left:11px;**

**background:url('../../public/img/symbols\_s.png') no-repeat -74px 0;**

**}**

**.rc>div:nth-child(1),.rc>div:nth-child(2),**

**.gc>div:nth-child(1),.gc>div:nth-child(2),**

**.bc>div:nth-child(1),.bc>div:nth-child(2),**

**.yc>div:nth-child(1),.yc>div:nth-child(2),**

**.kc>div:nth-child(1),.kc>div:nth-child(2){**

**width:14px;**

**height:20px;**

**background:url('../../public/img/symbols\_sm.png') no-repeat -206px 0;**

**}**

**.rc>div:nth-child(3),**

**.gc>div:nth-child(3),**

**.bc>div:nth-child(3),**

**.yc>div:nth-child(3),**

**.kc>div:nth-child(3){**

**width:74px;**

**height:92px;**

**top:24px;**

**left:9px;**

**background:url('../../public/img/symbols\_s.png') no-repeat 0 0;**

**}**

**</style>**

**<template>**

**<div class="card" :class="hoverable" ref="animateCard">**

**<div class="inner" :class="innerClass" ref="animateFlip" @click="click">**

**<div class="card-front-face">**

**<CardTemplate :type="card.type"></CardTemplate>**

**<div v-if="showOverlay" class="overlay"></div>**

**</div>**

**<div class="card-back-face"></div>**

**</div>**

**</div>**

**</template>**

**<script>**

**import CardTemplate from "../components/CardTemplate"**

**export default {**

**name: "Card",**

**props: {**

**clickHandler:{ type:Function },**

**transitionFinishHandler:{ type:Function },**

**card: { type: Object },**

**timeline: { type: Object },**

**active: {type: Boolean}**

**},**

**components:{CardTemplate},**

**watch:{**

**'card.transform': {**

**handler: function (after, before) {**

**this.transformAnimate();**

**},**

**deep: true**

**}**

**},**

**computed:{**

**innerClass:function(){**

**return this.card.type?'':'hidden';**

**},**

**hoverable:function(){**

**return (this.card.nextMoveValid && this.active)?'card-hoverable':'';**

**},**

**showOverlay:function(){**

**return (!this.card.nextMoveValid && this.active && this.card.owner !== 'dsc');**

**}**

**},**

**methods:{**

**click:function(){**

**this.clickHandler(this.card);**

**},**

**transitionFinish:function(){**

**this.transitionFinishHandler(this.card);**

**},**

**transformAnimate:function(){**

**if(typeof this.card.transform !== 'undefined'){**

**let parent = this;**

**this.timeline.to(this.$refs.animateCard,**

**{**

**x: this.card.transform.x,**

**y: this.card.transform.y,**

**z: this.card.transform.z,**

**rotation:this.card.transform.angle,**

**scaleX: this.card.transform.scale,**

**scaleY: this.card.transform.scale,**

**duration: this.card.transform.d,**

**onComplete:parent.transitionFinish**

**});**

**}**

**}**

**},**

**mounted:function(){**

**this.transformAnimate();**

**}**

**}**

**</script>**

**<style scoped>**

**.card{**

**position:absolute;**

**width:93px;**

**height:140px;**

**background-color: transparent;**

**perspective: 1000px;**

**transform-origin: 0px 70px;**

**top:0;**

**left:0;**

**}**

**.card-hoverable .inner:hover{**

**box-shadow: 0px 0px 1px 5px rgba(255,234,0,1);**

**}**

**.overlay{**

**display:block;**

**position:absolute;**

**top:0;**

**left:0;**

**width:100%;**

**height:100%;**

**border-radius:8px;**

**backface-visibility: hidden;**

**-webkit-backface-visibility: hidden;**

**background-color: rgba(0, 0, 0, 0.4);**

**}**

**.inner{**

**left:-46px;**

**position: relative;**

**width: 100%;**

**height: 100%;**

**transition: transform 1.0s;**

**transform-style: preserve-3d;**

**transform-origin: 0px 70px;**

**border-radius:8px;**

**}**

**.inner.hidden{**

**transform: rotateY(-180deg) translateX(-93px);**

**}**

**.card-front-face{**

**color: black;**

**display:block;**

**position:absolute;**

**width:93px;**

**height:140px;**

**border-radius:8px;**

**backface-visibility: hidden;**

**-webkit-backface-visibility: hidden;**

**}**

**.card-back-face{**

**color: white;**

**display:block;**

**position:absolute;**

**width:93px;**

**height:140px;**

**border-radius:8px;**

**background:url('../../public/img/back.png') no-repeat;**

**background-size:100% 100%;**

**backface-visibility: hidden;**

**-webkit-backface-visibility: hidden;**

**transform: rotateY(180deg);**

**}**

**</style>**

**<template>**

**<div class="button" @click="clickHandler"><i :class="'fa fa-'+faIcon"></i> <slot></slot></div>**

**</template>**

**<script>**

**export default {**

**name: "Button",**

**props: ['faIcon', 'clickHandler']**

**}**

**</script>**

**<style scoped>**

**.button {**

**position:relative;**

**font-family: 'Ubuntu', georgia;**

**text-align: center;**

**display: inline;**

**padding:10px 30px;**

**line-height: 61px;**

**border-radius: 35px;**

**box-sizing: border-box;**

**font-size: 35px;**

**text-decoration: none;**

**color: green;**

**font-weight: bold;**

**font-family: arial;**

**filter: drop-shadow(0px 5px 3px black);**

**box-shadow: 0 1px 0 rgba(255, 255, 255, 0.54) inset, 0 -2px 1px rgba(0, 0, 0, 0.31) inset;**

**background: #ffbf00;**

**background: linear-gradient(to bottom, #ffbf00 0%,#ff9700 100%);**

**box-shadow: 0px 10px 32px -6px rgba(0,0,0,0.75);**

**text-shadow: 0px 1px 1px white;**

**}**

**.button:after {**

**content: '';**

**border-radius: 38px;**

**position: absolute;**

**top: -6px;**

**left: -6px;**

**bottom: -6px;**

**right: -6px;**

**border:solid 6px rgba(255, 255, 255, 0.4);**

**}**

**.button:hover {**

**background: #ffdc00;**

**background: linear-gradient(to bottom, #ffdc00 0%,#ffa700 100%);**

**}**

**.button:active {**

**background: #ff9100;**

**background: linear-gradient(to bottom, #ff9100 0%,#ffb300 100%);**

**box-shadow: 0 1px 4px rgba(0, 0, 0, 0.32) inset, 0 -1px 0px white inset;**

**text-shadow: none;**

**}**

**</style>**

**<template>**

**<div class="table">**

**<div class="grad"></div>**

**<slot></slot>**

**</div>**

**</template>**

**<script>**

**export default {**

**name: "Board"**

**}**

**</script>**

**<style scoped>**

**.table{**

**position:absolute;**

**width:100%;**

**height:100%;**

**border-bottom:solid 1px #000;**

**border-top:solid 1px #000;**

**border-left:solid 1px #9a6515;**

**border-right:solid 1px #9a6515;**

**border-radius:12px;**

**background:url('../../public/img/table.jpg')no-repeat;**

**background-repeat: round;**

**box-shadow:0 6px 12px #000;**

**transform-style: preserve-3d;**

**/\*perspective: 1000px;\*/**

**}**

**.grad{**

**position:relative;**

**width:100%;**

**height:100px;**

**background:url('../../public/img/bg\_grad.png')no-repeat;**

**background-repeat: round;**

**}**

**</style>**

module.exports = class Client{

    constructor(name){

        this.name = name;

        this.code = Math.random();

        this.ready = false;

        this.socketId = null;

    }

    setName(name){

        this.name = name;

        return this;

    }

    getName(){

        return this.name;

    }

    setCode(code){

        this.code = code;

        return this;

    }

    getCode(){

        return this.code;

    }

    setReady(ready){

        this.ready = ready;

        return this;

    }

    getReady(){

        return this.ready;

    }

    setSocketId(socketId){

        this.socketId = socketId;

        return this;

    }

    getSocketId(){

        return this.socketId;

    }

};

let Client = require('./Client.js');

/\*\*

 \* Client storage class

 \* @type {module.ClientRepository}

 \*/

module.exports = class ClientRepository{

    constructor(){

        this.clients = [];

    }

    get(index){

        return this.clients[index];

    }

    findAll(){

        return this.clients;

    }

    findByName(name){

        return this.clients.find(function(elem){return elem.name === name;});

    }

    findBySocketId(socketId){

        return this.clients.find(function(elem){return elem.socketId === socketId;});

    }

    findIndexByName(name){

        return this.clients.findIndex(function(elem){return elem.name === name;});

    }

    findByCode(code){

        return this.clients.find(function(elem){return elem.code === code;});

    }

    findByReady(ready){

        return this.clients.find(function(elem){return elem.ready === ready;});

    }

    findByHasWon(hasWon){

        return this.clients.find(function(elem){return elem.hasWon === hasWon;});

    }

    findNext(client){

        let index = this.findIndexByName(client.getName());

        return this.get((index + 1)%this.count());

    }

    findPrevious(client){

        let index = this.findIndexByName(client.getName());

        let prev = (index + this.count() - 1)%this.count();

        return this.get(prev);

    }

    count(){

        return this.clients.length;

    }

    insert(client){

        this.clients.push(client);

        return client;

    }

};

/\*\*

 \* Base class for GameService user action handlers

 \* @type {module.GameActionHandler}

 \*/

module.exports = class GameActionHandler{

    constructor(gameService){

        this.gameService = gameService;

    }

    getGameService(){

        return this.gameService;

    }

    handleAction(data){

    }

};

module.exports = class GameService{

    constructor(id, messageRepository, clientRepository) {

        this.id = id;

        //Notifications sent to clients

        this.messagesRepository = messageRepository;

        //Client repository, logged into server

        this.clientRepository = clientRepository;

        //Action handling. Actions are messages received from Clients

        this.actionHandlers = {};

    }

    handleAction(socket, action, data){

        data.socketId = socket.id;

        //Should be replaced with some sort of access token

        let clientName = data.client.name;

        for(let key in this.actionHandlers){

            if(key === action){

                this.actionHandlers[key].handleAction(data);

                break;

            }

        }

    }

    getClientResponseData(socketId){

        return false;

    }

    getClientRepository(){

        return this.clientRepository;

    }

    getMessageRepository(){

        return this.messagesRepository;

    }

};

let GameService = require('./GameService.js');

let UNOGameService = require('./UNO/UNOGameService.js');

module.exports = class GameServiceFactory{

    constructor() {

    }

    create(name, channel){

        if(name === "UNO"){

            return new UNOGameService(channel);

        };

        return null;

    }

};

module.exports = class Message{

    constructor(type, message, socketId){

        this.type = type;

        this.socketId = socketId;

        this.message = message;

    }

};

<!DOCTYPE html>

<html lang="en">

<head>

    <title>UNO Node.js</title>

    <meta charset="UTF-8"

        name="viewport"

        content="width=device-width, initial-scale=1">

<link href="client/js/chunk-vendors.js" rel="preload" as="script"><link href="client/js/index.js" rel="preload" as="script"></head>

<body>

    <div id="clientApp"></div>

<script type="text/javascript" src="client/js/chunk-vendors.js"></script><script type="text/javascript" src="client/js/index.js"></script></body>

</html>

let Message = require('./Message.js');

module.exports = class MessageRepository{

    constructor(){

        this.messages = [];

    }

    clear(){

        this.messages = [];

    }

    insert(message){

        if(message instanceof Message){

            this.messages.push(message);

        }

    }

    findAll(){

        return this.messages;

    }

    findByClient(client){

        let socketId = client.getSocketId();

        return this.messages.find(function(elem){return elem.socketId === socketId;});

    }

};