Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Správa používateľov a zdieľaných entít pre vizualizáciu údajov v zdieľanej rozšírenej realite

Diplomová práca

2023 **Bc. Martin Frajkor**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Správa používateľov a zdieľaných entít pre vizualizáciu údajov v zdieľanej rozšírenej realite

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko, PhD.

Košice 2023 **Bc. Martin Frajkor**

Abstrakt v SJ

Táto práca sa zaoberá návrhom a implementáciou virtuálnej reality (VR) aplikácie využívajúcej rámec *A-Frame* a *WebSocket* protokol pre zdieľanie entít medzi viacerými používateľmi v projekte *NDMVR*. Cieľom práce je umožniť používateľom interagovať s rovnakými virtuálnymi objektmi v reálnom čase a vytvárať kolaboratívne prostredie.

V práci sme navrhli a implementovali klient-server architektúru, ktorá umožňuje viacerým používateľom v rôznych miestnostiach pripojiť sa a spolupracovať vo virtuálnej scéne. Popísali sme rôzne metódy zdieľania entít a navrhli sme spôsob synchronizácie entít medzi používateľmi.

Výsledkom práce je plne funkčná aplikácia, ktorá umožňuje viacerým používateľom interagovať s rovnakými virtuálnymi objektmi v reálnom čase. Naša implementácia ukazuje, že použitie WebSocket protokolu pre zdieľanie entít vo VR scéne je efektívny spôsob interakcie medzi používateľmi.

Kľúčové slová v SJ

virtuálna realita, zdieľané entity, vizualizácia údajov, kolaborácia

Abstrakt v AJ

This thesis deals with the design and implementation of a virtual reality (VR) application using the A-Frame framework and WebSocket protocol for sharing entities between multiple users in the NDMVR project. The goal of the work is to enable users to interact with the same virtual objects in real-time and create a collaborative environment.

We designed and implemented a client-server architecture that allows multiple users in different rooms to join and collaborate in a virtual scene. We described various methods for sharing entities and proposed a way to synchronize entities between users.

The result of the work is a fully functional application that allows multiple users to interact with the same virtual objects in real-time. Our implementation demonstrates that using the WebSocket protocol for sharing entities in a VR scene is an effective way of user interaction.

Kľúčové slová v AJ

virtual reality, shared entities, data visualisation, collaboration

Bibliografická citácia

FRAJKOR, Martin. *Správa používateľov a zdieľaných entít pre vizualizáciu údajov v zdieľanej rozšírenej realite*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2023. 58s. Vedúci práce: Ing. Štefan Korečko, PhD.

68094

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

Správa používateľov a zdieľaných entít pre vizualizáciu údajov v zdieľanej rozšírenej realite

User and Shared Entities Management for Data Visualization in Extended Reality

Študent: **Bc. Martin Frajkor**
Školiteľ: **Ing. Štefan Korečko, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Oboznámiť sa so súčasným stavom webového komponentu NDMVR pre vizualizáciu údajov z fyzikálnych experimentov v rozšírenej realite.
2. Analyzovať aktuálne prístupy k realizácii zdieľaných virtuálnych prostredí z hľadiska ich použiteľnosti pre komponent NDMVR.
3. Na základe analýzy navrhnúť a implementovať rozšírenie komponentu NDMVR, umožňujúce zdieľanú vizualizáciu vo virtuálnom 3D prostredí.
4. V rámci možností overiť implementované riešenie za použitia údajov z reálnych experimentov.
5. Návrh a implementáciu koordinovať s ďalšími riešiteľmi komponentu NDMVR.
6. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 21.04.2023
Dátum zadania diplomovej práce: 31.10.2022

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -12-

.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 27.4.2023 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som sa rád poďakoval všetkým, ktorí mi pomohli pri vypracovaní mojej diplomovej práce. Najprv by som sa chcel poďakovať svojmu školiteľovi a konzultantovi práce za nápomoc a podporu počas celej práce, za ich vynikajúce rady a inšpirácie, ktoré mi pomohli zlepšiť moju prácu.

Ďalej by som sa chcel poďakovať všetkým členom mojej rodiny za ich neustálu podporu a motiváciu.

Rád by som tiež poďakoval všetkým mojim spolužiakom a priateľom za to, že som sa mohol podeliť o svoje myšlienky a získal od nich cenné názory a pripomienky.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam zdrojových kódov

Úvod

Vytvárenie realistických prostredí vo virtuálnej realite za účelom poskytnutia autentického pocitu zo vzdelávania, práce a zábavy vzhľadom na pandemickú situáciu bolo výrazne podnietené nedostatkom sociálneho kontaktu. Práve tieto oblasti umožnili účastníkom vo virtuálnom svete navzájom spolupracovať na spoločných problémoch a ťažkostiach, pričom rozdielom nie je ani oblasť dátovej analytiky.

Vizualizácia dát v dnešnom svete, kde organizácie generujú obrovské množstvo dát prakticky každý deň, sa stala nevyhnutnou potrebou pre lepšie porozumenie získaných dát a rozpoznanie vzorov. Grafická reprezentácia dát zobrazená efektívnym spôsobom dáva lepšiu predstavu a umožňuje pohodlnejšie zorientovanie sa v zobrazenom probléme. Rovnako grafická reprezentácia ponúka vynikajúci prístup pri prezentovaní výsledkov a konzultovaní s recenzentmi a kolegami. Virtuálna realita poskytuje ideálny spôsob pre získavanie nových pohľadov na dáta, ktorý spája bežné zaužívané praktiky v reálnom svete a vzájomnú komunikáciu medzi zainteresovanými osobami - *zdieľaná virtuálna realita.*

Základné jadro pre túto diplomovú prácu predstavuje implementovaný projekt *NDMVR*, ktorého cieľom je práve vizualizácia experimentálnych dát vo virtuálnej realite. Keďže zdieľanie priestoru vo virtuálnej realite pre viacerých používateľov sa stalo oborom veľkého množstva štúdií a tento problém prešiel viacerými fázami inovácie počas desiatok rokov, jedným z cieľov tejto práce je analyzovať dostupné riešenia a známe postupy a aplikovať ich pri implementácií zdieľanej virtuálnej reality pre projekt *NDMVR*.

Táto práca skúma použitie *Websocket*, ľahkého a flexibilného protokolu pre obojsmernú komunikáciu cez internet, s cieľom uľahčiť zdieľanie virtuálnych realitných entít v rámci *A-Frame*, populárneho open-source frameworku na tvorbu VR zážitkov na webe.

Formulácia úlohy

Cieľom tejto práce je priblížiť problematiku vytvárania zdieľaných entít a ukladania ich stavu v prostrediach VR, analyzovať už dostupné nástroje pre následne využitie vhodnej technológie pre projekt *NDMVR* za účelom vytvorenia zdieľaného prostredia pre kolaboratívnu dátovú analytiku.

Zahrnuté ciele je možné rozdeliť do nasledujúcich bodov:

1. Predstavenie projektu *NDMVR* - časť 1.4.
2. Analýza problematiky zdieľaného prostredia a samotných entít vrámci prostredia VR projektu *NDMVR* určeného pre dátovú analytiku - časti 1.1.2 a 1.5.
3. Vybratie vhodného riešenia pre implementáciu samotnej zdieľanej virtuálnej reality - vrámci časti 1.5.
4. Implementácia zdieľaného prostredia prostredníctvom vybraného protokolu - časť 2.1.
5. Vyhodnotenie implementovaného riešenia - v časti 3, opísanie možnosti pre ďalšie rozšírenia projektu *NDMVR* - v časti 3.5.

1 Analytická časť

Neustály technologický vývoj hardvérových nástrojov podnietil aj zmeny vo sfére virtuálnej a rozšírenej reality. Počas tohto procesu bola technológia VR/AR s podporou zdieľania priestoru často popísaná nekonzistentnými spôsobmi. To podľa [1] viedlo k zbytočným zmätkom aj v technickej literatúre.

1.1 Virtuálna realita

Virtuálna realita (ďalej len VR) predstavuje rozsiahle rozhranie medzi počítačom a človekom, ktorého cieľom je simulácia realistického prostredia [2], pričom pri chápaní VR ako možného komunikačného média, získavame technológiu, ktorá ostane relevantná bezohľadu na to, ako sa vyvíjajú ostatné technológie [3]. Ako uviedol *J.Zheng, et al.* [2], prostredie VR umožňuje používateľom „pohybovať sa vo virtuálnom svete, pozorovať ho z rôznych uhlov, dotýkať sa a meniť ho“, a to takým spôsobom, ktorý je interaktívny a intuitívny. Rovnako VR popisuje aj *F.P.Brooks* [4], ktorý toto prostredie definuje ako „zážitok, do ktorého je používateľ úplne ponorený,“ a v ktorom má kontrolu nad pohľadom.

Napriek tomu, že základná myšlienka VR, a to princíp interaktivity a ponorenosti sa do prostredia [2], počas rokov ostala rovnaká, architektúra a samotné možnosti prevádzkovania VR sa vyvíjali ruka v ruke so zvyšujúcim sa výkonom, ktorý poskytujú stále lepšie a lepšie hardvérové komponenty. Rýchlejšie a spoľahlivejšie zariadenia umožnili prejsť od zariadení nevyhnutných pre beh VR, napr. senzory pre pohyb a otáčanie hlavy, stereoskopické displeje, k bežným simuláciam programov v prostredí osobných počítačov bez použitia uvedených zariadení [5]. Z tohto dôvodu môžeme uvažovať nad hardvérovo nezávislými riešeniami, s ktorými ľudia prídu do kontaktu v bežnom živote častejšie.

1.1.1 Využitie virtuálnej reality v bežnom živote

Menšie požiadavky pre prvotný vstup do sféry VR umožnili postupné rozširovanie sa tejto technológie aj do bežného sveta. Čoraz častejšie je VR používané

ako marketingový ťah pre prilákanie nových používateľov či sponzorov. Takmer bežným je použitie VR ako prostriedku pre vzdelávanie a odbornú štúdiu.

Aktuálny stav VR zahŕňa vyspelé hardvérové zariadenia, ako sú okuliare a rukavice, ktoré umožňujú presnejšiu interakciu s virtuálnym svetom. K dispozícii sú tiež softvérové nástroje a platformy na vytváranie a publikovanie VR obsahu. Medzi dôležité oblasti využitia prostredí VR patria tréningové simulačné systémy, trenažéry, ktoré sú prispôsobené samotným oborom, na ktoré boli určené. Simulované VR prostredia poskytujú základné poznatky a prvotnú skúsenosť v sférach ako je letectvo, doprava, agrokultúra a armáda [6].

Medzi hlavné metódy využívania VR patria:

1. **Simulácie**: VR sa často používa na simulácie reálnych situácií a prostredí, ako sú letecké simulácie, výcvikové programy pre záchranné tímy a výukové programy pre medicínske profesie.

2. **Zábava**: VR je tiež využívaná na vytváranie zábavných hier a zážitkov, ktoré ponúkajú interaktívne prostredie pre používateľa.

3. **Terapia**: VR sa tiež používa v terapeutických situáciách na liečbu rôznych duševných a fyzických problémov, ako sú fóbie a bolesti.

4. **Prezentácie**: VR sa čoraz častejšie používa na prezentácie produktov a služieb, čo umožňuje používateľom interaktívne sa oboznámiť s produktom alebo službou.

Štúdie týkajúce sa výskumu fóbií [7], kontroly bolesti [8] či všeobecné využitie v psychológií [9] bežne využívajú prostredie virtuálnej aj rozšírenej reality pre lepšie pochopenie problému.

V súčasnosti sa VR stále vyvíja a rozširuje sa do ďalších odvetví, ako sú vzdelávanie, zdravotníctvo a pracovné prostredie. S rozvojom technológie sa očakáva, že VR bude mať stále väčší vplyv na spôsob, akým ľudia pracujú, učia sa a zábavne trávia čas.

1.1.2 VR a vizualizácia dát

Práve lepšie pochopenie a pohľad na problémy z iného uhla pohľadu sú jednými z kľúčových faktorov, prečo sa VR stáva populárnejšie aj v iných odvetviach. Keďže platformy VR sú graficky orientované, spolu so široko dostupným výkonným hardvérom za určitú cenu poskytujú možnosť lepšieho objavovania v doménach, ktoré vyžadujú priestorové rozmery [10]. Namiesto prezerania množstva

tabuľkových dát je zobrazenie pomocou grafických komponentov v trojrozmernom priestore ideálnejšie a pre mnohých komfortnejšie. Podobne, ako uviedol *C.Donalek, et al.* v [10], z hľadiska veľkých datasetov mnohí výskumníci siahajú po dátovej vizualizácii, čo sa preukázalo byť efektívne vo viacerých doménach.

VR ponúka mnoho možností pre vizualizáciu dát, čo umožňuje lepšie pochopiť a zobraziť komplexné informácie.

1. **3D modely a interaktívne vizualizácie**: VR umožňuje vytvoriť interaktívne modely a vizualizácie, ktoré umožňujú používateľom prehliadať a skúmať dáta v trojrozmernom priestore.

2. **Infografiky a grafy**: VR umožňuje vytvoriť interaktívne infografiky a grafy, ktoré umožňujú používateľom analyzovať a porovnávať dáta vizuálne.

3. **Mapy a geografické dáta**: VR umožňuje vytvoriť interaktívne mapy a vizualizácie geografických dát, čo umožňuje používateľom prezerať a analyzovať geografické informácie v trojrozmernom priestore.

4. **Virtuálne prezentácie**: VR umožňuje vytvoriť virtuálne prezentácie, kde môžu používatelia prezerať a interagovať s dátami v reálnom čase.

5. **Dátové modely**: VR umožňuje vytvoriť interaktívne modely dát, ktoré umožňujú používateľom analyzovať a vizualizovať komplexné dátové vzťahy.

Prepojením vizualizície dát a prostredia VR hovoríme o doméne *pohlcujucej analytiky* (z angl. *Immersive Analytics*(*IA*))[11], ktorej cieľom je „vybudovať kolaboratívny, interaktívny systém“, ktorý dokáže čo najviac pohltiť používateľa do ich dát [12]. Existuje množstvo „pohlcujúcich“ nástrojov pre kolaboratívnu prácu na projekte, avšak hlavný problém IA predstavujú nedostatočné skúsenosti používateľov s technológiami tohto typu [13]. Základom pre riešenie tohto problému je poskytnutie dostatočného ponaučenia pre vykonanie rozdielnych úkonov vrámci IA a spôsobu ako komunikovať s inými používateľmi zdieľajúcimi rovnaké prostredie [13]. Nevyhnutné je taktiež oboznámiť používateľov s obmedzeniami, ktoré prináša VR ako je odozva alebo sledovanie pohybových senzorov a ich premapovanie prostredia.

Čo sa týka dátovej analytiky, *E. Barret, B.Bach, et al.* [13] rozdelili spôsob komunikácie o dátach, na dva typy:

1. **voľná forma** - dátoví analytici sa snažia získať zmysel dát spoločne,

2. **formálna komunikácia** - odborník prezentuje dáta publiku.

Na základe tohto rozdelenia je dôležité vo VR technológiach pre IA umožniť obidve prístupy pre zdieľanie dát za účelom vytvorenia plnohodnotného ponorenia sa do systému.

1.2 Webový rámec A-Frame

Webový rámec *A-Frame*[1] je open-source projekt umožňujúci budovanie prostredí vo VR. Poskytuje entitno-komponentový rámec založený na jazyku *HTML* a štruktúrovo podobný knižnici *Three.js*. *A-Frame* podporuje väčšinu VR headset-ov, pričom podporuje aj rozšírenú realitu, z tohto dôvodu predstavuje ideálneho kandidáta pre budovanie cross-platform VR prostredí. Bol vyvinutý spoločnosťou Mozilla a umožňuje vývojárom vytvárať VR aplikácie s použitím *HTML*.

Na webovej stránke projektu *A-Frame*[2] autori poskytujú jednoduchý fragment zdrojového kódu, ktorý predstavuje základné štruktúry pre vytvorenie VR prostredia (viď. zdrojový kód 1.1). Vzhľadom na to, že rámec A-frame, ako už bolo spomenuté vyššie, je založený na jazyku HTML, fragment zdrojového kódu 1.1 je po umiestnení do <*body*> tagov a nainštalovaní *A-Frame* knižnice (*npm install aframe* alebo dodanie zdroja ako skriptu) plne funkčný a zobrazujúci 3D scénu vo webovom prostredí, ako je zobrazené na obrázku. Tento základný útržok kódu (viď. zdrojový kód 1.1) bude slúžiť ako východiskový bod pre experimentálne účely tejto práce pre vyskúšanie rôznych knižníc umožňujúcich zdieľanie vo VR a ich vzájomné porovnanie.

```
1 <a-scene>
2   <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
3   <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
4   <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5"
    ↪  color="#FFC65D"></a-cylinder>
5   <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4"
    ↪  color="#7BC8A4"></a-plane>
6   <a-sky color="#ECECEC"></a-sky>
7 </a-scene>
```

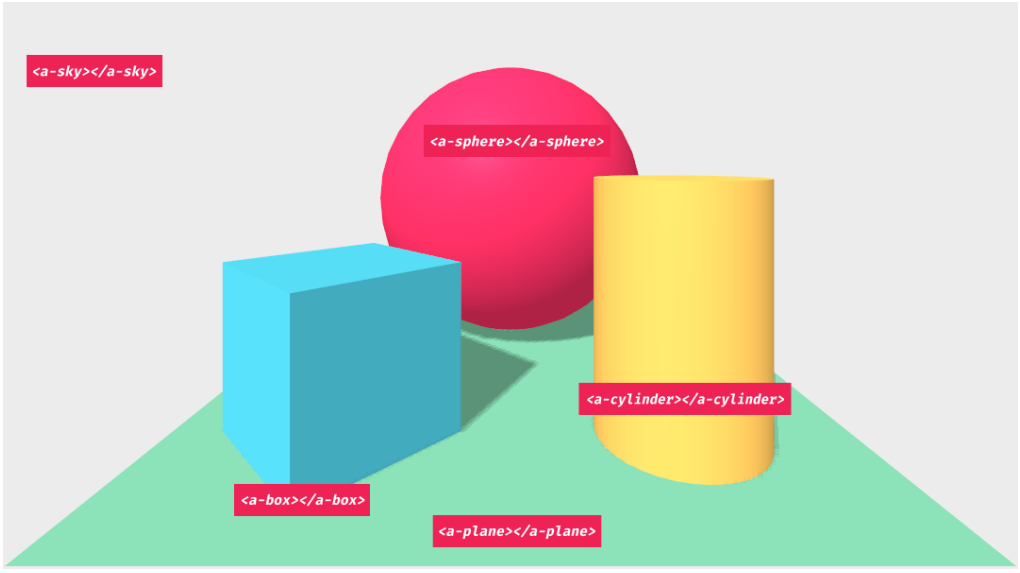
Zdrojový kód 1.1: Základný A-Frame príklad

Fragment kódu 1.2 vysvetľuje základný princíp štruktúrovania pri vytváraní

[1] https://aframe.io/

[2] https://aframe.io/docs/1.3.0/introduction/

objektov, ktoré sú prístupné vrámci *A-Frame* scény, čomu korešponduje tag <*a-scene*>. Rámec *A-Frame* poskytuje primitíva pre vytváranie objektov typu box, sféra, cylinder (tagy <*a-box*>, <*a-sphere*>, <*a-cylinder*> uvedené v tom poradí, čo zobrazuje aj obrázok 1.1) a mnoho ďalších, ktoré sú dostupné v dokumentácií projektu. Tieto primitíva je možné spájať do väčších komponentov tzv. entít, ktorým zodpovedá tag <*a-entity*> a umožňuje jednoduchšiu manipuláciu vrámci scény VR a ich identifikáciu. Ako zobrazuje zdrojový kód 1.2 samotným aframeovské primitívam a entitám je potrebné nastaviť atribúty s možnosťou výberu zo širokej škály ako sú napr. pozícia vrámci scény, rotácia, farba a rôzne iné.



Obr. 1.1: Vyrenderovaný základný A-Frame príklad

A-Frame je založený na koncepte entit (entities), ktoré sú HTML elementy, ktoré predstavujú objekty a funkcie v VR priestore. Tieto entity sú prispôsobiteľné pomocou atribútov a CSS štýlov a umožňujú vývojárom vytvoriť komplexné a interaktívne VR scény. *A-Frame* tiež podporuje rôzne funkcie VR, ako sú pohybové ovládanie, interakcie s objektmi a fyzikálne simulácie. Framework tiež umožňuje jednoduchú integráciu s inými webovými technológiami, ako je *WebGL* a *Three.js*.

Ďalšou výhodou *A-Frame* rámca je, že jeho aplikácie sú kompatibilné s väčšinou VR zariadení, vrátane PC VR headsetov, mobilných VR headsetov a webových VR zariadení. To umožňuje vývojárom vytvárať a distribuovať VR aplikácie širokej verejnosti.

Hlavnou výhodou rámca *A-Frame* je jednoduchosť z hľadiska polohovania jednotlivých entít v scéne. Tento rámec umožňuje vnorenie tagov do iných, čo z implementačného hľadiska predstavuje reláciu rodič - potomok, pričom inicializovaná poloha potomka je presne v strede rodiča, t.j. (0,0,0) v 3D priestore u

potomka znamená presný stred v priestore entity rodiča. Táto vlastnosť rámca *A-Frame* však nie je v výhodná vo všetkých prípadoch, konkrétne v tých kedy potrebujeme získať absolútnu pozíciu entity v scéne. Rámec *A-Frame* neposkytuje prístupový bod k získaniu tejto informácie a preto pri množstve vnorených entít je táto stratégia komplikovaná nakoľko vyžaduje dopočítavanie pozície v scéne vzhľadom na rodiča danej entity. Avšak, v takomto prípade dokážeme využiť fakt, že *A-Frame* poskytuje porovnateľnú štrukúru s knižnicou *Three.js*, na ktorom je založený a danú informáciu získať odtiaľ.

1.3 React

React (alebo *React.js*) je open-source javascriptovská knižnica pre tvorbu frontend používateľských rozhraní založených na komponentoch [3]. *React* využíva virtuálny *DOM* ako riešenie pre problémy s obojsmerným viazaním z hľadiska uchovávania stavu. „Virtuálny DOM je rýchla reprezentácia skutočnej DOM uložená v pamäti, pričom táto abstrakcia umožňuje brať Javascript a DOM akokeby boli reaktívne“ [14].

React tiež poskytuje funkcie pre vývoj pomocou prístupu komponentového programovania, čo umožňuje vytvárať aplikácie z jednotlivých, ľahko testovateľných a opätovne použiteľných častí. Framework taktiež podporuje server-side rendering a umožňuje vývoj aplikácií pre rôzne platformy, ako sú web, mobilné zariadenia a desktopy.

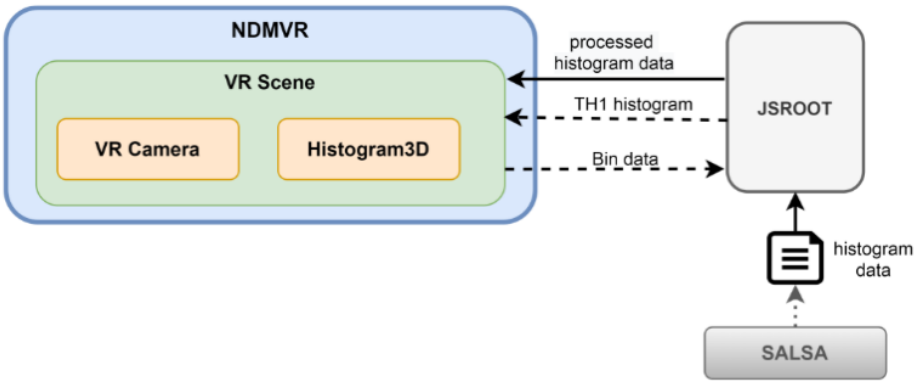
React 18.0 prichádza s vylepšenými funkciami pre prácu s komponentami, ako je napríklad zlepšený algoritmus pre spracovanie stavu a vylepšený systém pre zdroje. Framework tiež obsahuje nové nástroje pre optimalizáciu a diagnostiku aplikácií, čo umožňuje vývojárom vytvárať rýchle a efektívne aplikácie.

Pre účely tohto projektu bol *React* zvolený hlavne z dôvodu, že dáta, ktoré prichádzajú zo servera sú dynamické a vzhľadom na čas sa menia. Práve „reaktívnosť“ tejto knižnice zabezpečuje, že získané dáta sa aktualizujú v takmer reálnom čase (podľa rýchlosti vykresľovania scény). Keďže hlavnou myšlienkou je len preposielať dáta zo servera s čo najmenším ukladaním stavu (časť 1.6) je tento prístup vhodný pre prepojenie s projektom *NDMVR*1.4 pre dosiahnutie reaktívnej a pohlcujúcej skúsenosti vo VR. Veria *React* využitá v tomto projekte: v18.0.

[3] `https://reactjs.org/`

1.4 NDMVR

Softvérový komponent NDMVR (N-dimenzionálna VR)[15] postavený na webovom rámci A-frame1.2 a React1.3 knižnici umožnuje vizualizáciu dvoch a troch dimenzionálnych histogramov. Cieľom *NDMVR* komponentu je byť súčasťou väčšieho ekosystému pre dátovú analytiku, pričom dáta získané pre tento komponent sú poskytnuté ďalším SALSA komponentom (jednoduchý dávkovací systém výpočtových úloh) [16].



Obr. 1.2: Architektúra NDMVR [15].

Obrázok 1.2 predstavuje softvérovú architektúru NDMVR komponentu, pričom ako bolo uvedené vyššie získané dáta pochádzajú z rozdielneho komponentu SALSA. Táto architektúra predstavuje východiskový bod pre implementáciu zdieľanej reality a vo fáze implementácie zdieľaných entít bude kľúčom pre vyriešenie problému.

V tejto práci je spomenutá aj knižnica *React-NDMSPC-Core* [4], ktorá obsahuje pomocné funckie pre distribúciu dát a vytváranie samotných histogramov pre scénu zobrazovanú v rámci *A-Frame*, a projekt *React-NDMSPC* [5], ktorý predstavuje spojenie tejto knižnice a projektu *NDMVR* a ich vzájomné prepojenie pre vizualizáciu dát.

1.5 Zdieľaná virtuálna realita

Problém vytvorenia zdieľanej reality, vrámci ktorej sa používatelia navzájom vidia a interagujú s prostredím nie je jedinečný a počas minulých desaťročí došlo k

[4] `https://www.npmjs.com/package/@ndmspc/react-ndmspc-core`
[5] `https://www.npmjs.com/package/@ndmspc/react-ndmspc`

implementácií viacerých knižníc, ktoré umožňujú riešienie tohto problému. Riešenia týkajúce sa A-Frame rámca predstavujú open-source projekty ako *aframe-sharedspace-component* 1.5.1 či *Networked A-Frame* 1.5.2. Avšak vzhľadom na softvérovú architektúru *NDMVR* viď. časť 1.2 je potrebné analyzovať aj všeobecné riešenia využitím štandardných komunikačných protokolov ako je *WebSocket*.

1.5.1 aframe-sharedspace-component

A-Frame shared-space[6] komponent predstavuje open source riešenie pre rámec *A-Frame* s cieľom zabezpečiť zdieľanie entít vrámci VR scény. Tento komponent, ako uvádzajú jeho autori, „poskytuje jednoduchý účastnícky model“, vrámci ktorého sa môžu používatelia prihlásovať a odhlásovať z vytvorenej scény a navzájom posielať správy ostatným účastníkom a publikovať zvukové nahrávky.

S cieľom poskytnúť minimálnu signalizáciu v infraštruktúre medzi „rovesníkmi“ (z angl. „peer“), tento komponent beží na aplikačno-programovom rozhraní *WebRTC*[7], ktorý poskytuje podporu pre telefónne hovory a videochat spustiteľné z webových prehliadačov a mobilných aplikácií. Ako bolo spomenuté, technológia *WebRTC* je založená na peer-to-peer komunikácií, pričom API umožňuje priamu komunikáciu dvoch peerov, z prehliadača do prehliadača[17].

Autori na stránke tohto projektu poskytujú aj minimálnu časť kódu, ktorá je potrebná pre spustenie tohto projektu a integráciu do *A-Frame* scény, čo je zobrazené na fragmente zdrojového kódu1.2. Tento príklad je však spustiteľný až po patričnom nainštalovaní tejto knižnice (teda *npm install aframe-sharedspace-component*). Ako je možné vidieť na príkladovom kóde 1.2, tento komponent umožňuje zadefinovanie entity avatara v tagu <*template*>, ktorý bude predstavovať pripojeného používateľa z prehliadača priamo vo VR scéne.

```
1  <a-scene>
2    <a-entity sharedspace="audio: true" avatars>
3    </a-entity>
4  </a-scene>
5  <template>
6    <a-sphere radius="0.1"></a-sphere>
7  </template>
```

Zdrojový kód 1.2: Kód pre spustenie shared-space-komponent

[6] https://github.com/delapuente/aframe-sharedspace-component
[7] https://www.w3.org/TR/webrtc/

Toto open source riešenie pre rámec *A-Frame* nepredstavuje vhodného kandidáta pre riešenie vrámci nášho projektu, vzhľadom na to, že sa jedná o zdieľanie statického prostredia. To znamená, že zdieľať je možné len avatarov používateľov a ich pozíciu vrámci VR scény, nie samotné dynamické prostredie a zachytávanie zmien v ňom. Keďže cieľom tejto práce je zdieľanie entít a dát vo VR, ktoré sú dynamické, teda meniace sa s časom alebo iným faktorom z pohľadu dátovej analytiky, využitie tohto komponentu vrámci implementácie riešenia neprichádza do úvahy.

1.5.2 Networked A-Frame

Networked A-Frame[8] (ďalej len NAF) je open-source rámec pre písanie multi používateľských VR aplikácií v jazyku HTML a Javascript. Tento rámec je postavený nad *A-Frame* rámcom a zabezpečuje podporu pre *WebRTC* a resp. alebo *WebSocket* pripojenia, pričom je možné využiť charakterisitcké vlastnosti tohto projektu, medzi ktoré patria:

- Podpora streamovania zvuku pre komunikáciu používateľov v aplikácii,
- Podpora video-hovorov,
- Multi-platformové riešenia (dostupné pre webové prehlidače, Oculus Rift, HTC Vive...),
- Senzitivita z hľadiska bandwidth-u (upozornenia len ak sa niečo zmení).

Rovnako ako komponent shared-space 1.5.1 aj NAF umožňuje zadefinovanie používateľských avatarov pre používateľov pomocou tagov <*a-assets*> a <*template*> a relatívne jednoduchú inicializáciu pre zabezpečenie zdieľania vrámci VR scény. Pre inicializáciu zdieľania je potrebné ľahko pozmeniť kód, čo zobrazuje zdrojový kód 1.3. VR scénu, na ktorej má zdieľanie prebehnúť je nutné označiť kľúčovým slovom *networked-scene*, a entitu, ktorá má byť zdieľaná vrámci danej scény slovom *networked*, pričom je možné zadať zoznam parametrov, ktoré sú dostupné v dokumentácii projektu. Zároveň, nevyhnutnosťou pre spustenie projektu je inicializácia servera (WebRTC alebo WebSocket), ktorého príklad je rovnako dostupný na stránke projektu. Tento server sa stará o uchovávanie informácií o zdieľaných entitách a zmenách, ktoré v nich nastali.

[8] `https://github.com/networked-aframe/networked-aframe`

```
<a-scene networked-scene>
  <a-assets>
    <template id="avatar-template">
      <a-sphere></a-sphere>
    </template>
  </a-assets>
  <a-entity id="player"
  ↪ networked="template:#avatar-template;attachTemplateToLocal:false;"
  ↪ camera wasd-controls look-controls>
  </a-entity>
</a-scene>
```

Zdrojový kód 1.3: Networked A-Frame príklad

Na prvý pohľad sa zdá, že NAF je ideálnym riešením pre implementáciu zdieľanej reality a dátovej analytiky vo VR, avšak kvôli implementačným rozhodnutiam projektov *NDMVR*1.4 a *React-NDMSPC* pri experimentoch s týmto nástrojom sme narazili na viacero problémov, resp. nedostatkov.

Ukladanie celého stavu entít

NAF ako nástroj pre zdieľanie entít vo VR scénach rámca *A-Frame* využíva a pre správne správanie sa, vyžaduje server, na ktorý ukladá informácie o zdieľaných entitách, teda tagy označené kľučovým slovom *networked*. Keďže sú na serveri uložené všetky dôležité informácie o danej entite z pohľadu vykresľovania do scény a projekt *NDMVR* využíva knižnicu React na manažment stavov, ukladanie týchto informácií predstavuje zbytočnú duplicitu a možné zahltenie siete pri nespočetnom množstve entít a informácií o nich.

Rovnako z toho dôvodu, že projekt *NDMVR* obsahuje svoj vlastný server odkiaľ získava len čisté dáta pre zobrazenie môžeme využiť funkcionalitu knižnice *React* a vytvoriť vlastný server, ktorý by uchovával len potrebné informácie. Nejedná sa teda, o ukladanie celých entít a informácií o nich, ale čisto dáta, ktoré sa vrámci týchto entít majú meniť dynamicky.

Nekompatibilita s knižnicou React

Pri prevádzaní prvotných experimentov s NAF a React-om 1.3 nad základnou scénou1.1 bola spozorovaná chyba, ktorá prispela k rozhodnutiu o nevyužití tejto knižnice pre implementačné účely. Nakoľko *React* pracuje s virtuálnou DOM [18]

a *A-Frame* je postavený hlavne na jazyku *HTML*, ktorá má vlastnú štruktúru zodpovedajúcu DOM webového prehliadača, dochádza ku konfliktu pri ideológií týchto technológii. Čo sa týka odpozorovanej chyby, išlo o znefunkčnenie *A-Frame* kamery pri zadefinovaní vlastnej (custom) kamery, pričom entita defaultne nastavenej kamery nebola správne reinicializovaná a došlo ku konfliktu dvoch kamier na jednej scéne. Tento jav bol odsledovaný len vrámci *NAF* projektu, pričom popísaná chyba v projekte bola v danom čase v stave „otvorená" a vedelo sa o nej. Samotnú knižnicu *A-Frame* to však neovplyvnilo.

Udržiavatelia tejto knižnice sa prostredníctom *A-frame Slack* kanálu vyjadrili, že spájanie *A-Frame* a *React* knižníc dokopy za účelom jednoduchšieho spravovania stavu nie je ideálne a rázne radia zdržať sa tohto kroku. Na základe tohto faktu, si myslíme, že vyskytnutá chyba má spojenie práve s týmto prístupom nakoľko *A-Frame* a *React* využívajú dve rozdielne štruktúry pre získavanie a úpravu entít resp. komponentov.

1.5.3 Komunikačný protokol Websocket

Komunikačný protokol WebSocket[9] [19] v princípe predstavuje mechanizmus pre odovzdávanie správ [20]. Umožňuje dvojsmernú komunikáciu medzi klientom a vzdialeným hostom, ktorý povolil komunikáciu takýmto mechanizmom [19]. Z hľadiska bezpečnosti využíva tzv. origin-based bezpečnostný model, ktorý je zakompovaný vo väčšine dnešných webových prehliadačov [10]. Ako je uvedené v IETH štandarde pre WebSocket [19], protokol sa skladá z inicializačného podania rúk (z angl. handshake) a následovného rámcovania správ rozvrstvovaných prostredníctvom štandardu TCP.

Komunikácia pomocou*WebSocket protokolu* je relatívne jednoduchá a poskytuje možnost spracovať textové a binárne dáta [21]. Keďže Javascript umožňuje prevedenie objektov na formát JSON, teda text, je využitie tohto protokolu čoraz bežnejšie v správovo založených systémoch.

Nakoľko je pre implementáciu zdieľania dát v projekte *NDMVR* potrebné len posielanie potrebných dynamicky meniacich sa dát, ako bolo opísané v 1.5.2, je tento postup vhodný pre splnenie cieľa tejto práce a teda zabezpečenia zdieľania entít vrámci scény VR. Napriek tomu, že prvotné experimenty a návrh servera pre ukladanie dát bol implementovaný knižnicou *Socket.io*1.5.4, pre splnenie cieľov tejto práce bude WebSocket predstavovať východískový protokol pre komunikáciu medzi klientom a serverom.

[9] `https://github.com/websockets/ws`

[10] `https://crossbar.io/docs/Browser-Support/`

1.5.4 Knižnica Socket.io

Knižnica *Socket.io*[11][22] umožňuje nízko-latečnú, dvojsmernú a na udalostiach založenú komunikáciu medzi klientom a serverom. Napriek tomu, že *Socket.io* nepredstavuje implementáciu *Websocketu* ako takú, využíva *Websocket* kde je možné. Avšak ako bonus pridáva doplňujúce metadáta ku každému paketu a poskytuje out-of-the-box funkcie, ktoré je pri WebSockete potrebné dodefinovať osobitne.

Socket.io automaticky vyberie najlepšiu dostupnú metódu pre real-time komunikáciu, či už to bude WebSockets, long-polling alebo iný protokol, a zabezpečuje prehľadné a jednotné API pre komunikáciu na strane klienta aj servera. Knižnica tiež umožňuje vytvárať kanály alebo miestnosti pre komunikáciu, čo umožňuje lepšiu organizáciu a kontrolu komunikácie.

Vrámci tejto práce bol využitý pre zorientovanie sa v problematike a pre prvotný návrh a implementáciu projektu. Ako bolo spomenuté, táto knižnica nebude vrámci finálnej implementácie súčasťou syntetickej časti, keďže iné projekty vrámci *react-ndmspc* využívajú *Websocket* ako hlavnú komunikačnú metódu a implementácia zdieľania entít bola prispôsobená adekvátne.

1.6 Ukladanie stavu vs. relaying

Ako bolo uvedené v kapitole 1.5 hlavným úmyslom tejto práce je vytvorenie VR prostredia, v ktorom používatelia pracujú s rovnakými synchronizovanými dátami a navzájom spoločne v prostredí interagujú. Vzajomná interakcia predstavuje:

- možnosť vidieť ostatných používateľov na scéne,
- práca so scénou, pričom sa scéna mení rovnako pre všetkých,
- analýza zobrazených dát.

Na základe týchto požiadaviek vieme rozdeliť funkcionalitu projektu z pohľadu spracovania dát na dva hlavné typy a to: **dáta, ktorých stav potrebujeme uložiť** a **dáta, ktoré posunieme ďalej** tzv. (*relay*).

Keďže dáta, ktoré sa majú zobraziť v komponente *NDMVR* 1.4 prichádzajú zo servera v určitých intervaloch, tento stav nie je potrebné ukladať a preto ho posunieme ďalej do *React* rámca, ktorý zabezpečí znovu vykreslenie prijatých dát a teda reaktívnosť a dynamickosť tohto komponentu. Na druhej strane, pre zabezpečenie prostredia zdieľanej VR musí každý klient poslať na server informácie o

[11] `https://socket.io/docs/v4/`

svojom aktuálnom stave, ktoré je potrebné rozposlať ostatným klientom. Tieto informácie obsahujú 3 dôležité polia a to: identifikátor, súradnice pozície a rotácie v priestore scény. Pre jednoduchosť implementácie, sme sa prvotnú časť rozhodli navrhnúť tak, aby server, ktorý rozposiela dáta ostatným klientom, tieto informácie o stave ukladal, čo však v neskorších testovacích fázach môže viesť k spomaleniu celého systému alebo jeho kompletnej nefunkčnosti. Preto toto riešenie považujeme za dočasné, kým sa nevyrieši hlavná časť a to zdieľaná vizualizácia v priestore.

Podobne ako zdieľanie pozície bude riešená aj zmena a synchronizácia prostredia v scéne. Keďže jeden klient bude predstavovať „mastra“, ktorý bude interagovať s prostredím a ostatní klienti „listeneri“, akcie, ktoré „master“ vykoná bude potrebné ukladať vo forme inštrukcií, ktoré sa majú vykonať na serveri. Uloženie týchto informácií je dôležité práve preto, aby sa dosiahol konzistentný stav aj v prípadoch, kedy sa klient napojí do systému o niečo neskôr, a aby jeho prostredie bolo zosynchronizované s ostatnými klientmi a tak mohol analyzovať rovnaké dáta.

V prípade, že sa rozhodneme využiť spôsob, že dáta nie sú zdieľané, ale sú zdieľané len entity klientov a ich pohyb, môžeme využiť základný projekt *NDMVR* 1.4 a komponent *NDMVRShared* implementovaný v syntetickej časti tejto práce, ktorý zabezpečí zdieľanie entít na scéne.

1.6.1 Lokálne uložisko na serveri

Dáta prichádzajúce zo strany klienta, teda informácie o identifikátore (konkrétny pripojený *Websocket* klient), pozícii a rotácii sme sa vrámci prvotnej implementácie rozhodli ukladať na serveri, čo je bližšie popísané v kapitolách 1.7.2 a 2.1 syntetickej časti tejto práce. Tento spôsob zabezpečuje, že informácie o každom klientovi sú dostupné kedykoľvek z ktoréhokoľvek pripojeného klienta a teda *React* knižnica zabezpečí vykreslenie ostatných klientov na základe uložených informácií.

Hlavným problémom tohto implementačného rozhodnutia je prípad, pri ktorom je na server napojené väčšie množstvo klientov, pričom môže dôjsť k oneskoreniu pri spracovaní požiadaviek na získanie stavu (a teda oneskorenie celého systému) alebo absolútnom znefukčnení servera, ktorý sa stará o distribúciu stavov a dát. Tento bod je hlavne prepojený na technické detaily hardvéru, na ktorom server beží (teda CPU, pamäť atď) a čiastočne sa tomuto problému dá vyhnúť lepším a výkonnejším hardvérom (samozrejme do určitej miery).

1.6.2 React.useState problém

Pri prvotnej implementácii zdieľanej virtuálnej reality pomocou *Socket.io* (viď. časti 1.5.4 a 1.7.2) bol stav z klienta uložený lokálne na serveri a pre zobrazenie ostatných entít klientov na virtuálnej scéne bol získaný a uložený pomocou *useState* hook-u. Teda vždy, ak bol stav jedného klienta aktualizovaný, aktualizoval sa aj zoznam o ostatných klientoch s patričnými hodnotami a React zabezpečil prekreslenie scény s danými aktualizáciami. Problém však nastal pri zmene požiadaviek a využitia *WebSocket* protokolu pre komunikáciu so serverom a zachovania "jednotnosti"kódu vrámci *React-NDMSPC* projektu.

Pri využití *Websocket* protokolu po jednoduchých zmenách v implementácii *Socket.io* servera *React useState* hook nestíhal aktualizovať prichádzajúce zmeny o stave klientov zo strany servera a následné prekresľovanie entít neprebehlo. Z tohto dôvodu sme sa rozhodli využiť alternatívu pre uchovávanie stavu mimo samotnej *React* knižnice - *Redux*.

1.6.3 Redux

Redux[12] predstavuje javascriptovskú knižnicu pre riadenie a centralizovanie stavu aplikácie. Primárnym cieľom je riadenie aplikácie s veľkým globálnym stavom a uľahčenie zdieľania dát medzi komponentmi. Globálny stav aplikácie sa nachádza v „store“. Všetky komponenty aplikácie majú prístup k tomuto stavovému úložisku a môžu ho čítať a upravovať prostredníctvom akcií.

Redux využíva funkcie pre redukčné funkcie, ktoré sú funkcie, ktoré prijímajú aktuálny stav a akciu a vracajú nový stav. Tieto redukčné funkcie sú jednoduché a ľahko testovateľné a slúžia na prevedenie zmien v stave aplikácie na základe akcií. Keď používateľ vykoná akciu, tá prechádza cez redukčnú funkciu tzv. „*reducer*“, ktorý vezme predchádzajúci globálny stav a prichádzajúcu akciu a vráti ďalší pozmenený globálny stav [23]. Akonáhle sa aktualizuje „store“ cez redukčnú funckiu spôsobí prekreslenie v DOM, čo je v podstate len zmena v používateľskom rozhraní.

Redux tiež umožňuje prenos stavu medzi komponentami bez nutnosti ručnej implementácie prenosu stavu prostredníctvom *props*. Pomocou konceptu „providera“ a „connectoru“ môžu komponenty priamo komunikovať so stavovým úložiskom bez nutnosti prenosu stavu cez nadradené komponenty.

Využitie tohto prístupu pre tento projekt je vhodné vzhľadom na to, že nie je potrebné ukladať stav aj na strane klienta v *React* rámci, ale stav získame rovno zo

[12] `https://redux.js.org/`

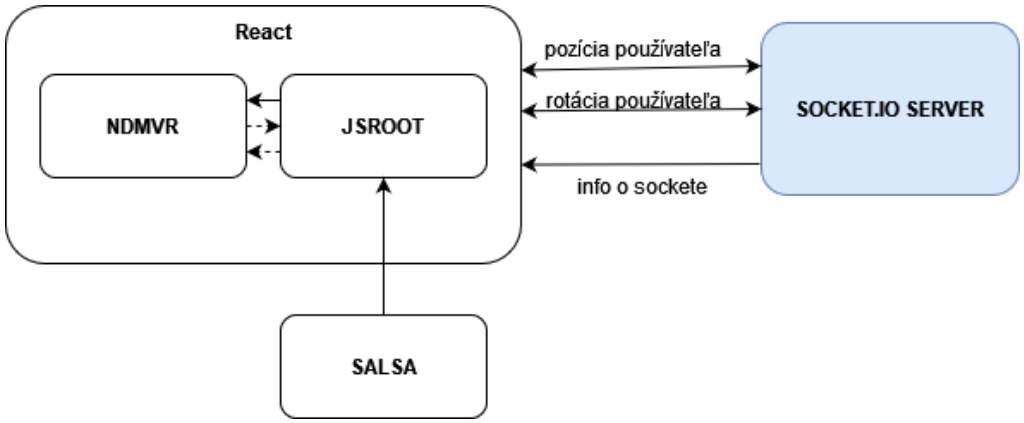
„storu“, či už na strane servera alebo klienta. Druhou výhodou pri zavedení tejto knižnice je to, že opätovne zjednotíme zaužívané prístupy vrámci *React-NDMSPC* projektu a ich implementácie len obohatíme o potrebnú funkcionalitu.

1.7 Prvé experimenty

Počas prvotných experimentov s využívaním knižnice *Socket.io* neboli do úvahy brané všetky dáta, ktoré boli potrebné pre správnu implementáciu projektu. Vrámci experimentov sme sa rozhodli zamerať hlavne na pozíciu a rotáciu používateľa na scéne a na odosielanie základných informácii o serveri (názov, verzie, identifikátor socketu) pripojenému klientovi.

1.7.1 Prvotný návrh pre zobrazenie používateľov

Po prvých konzultáciach vyzerala implementácia zdieľania polohy a rotácie v prostredí NDMVR rovnako ako zobrazuje obrázok 1.3.



Obr. 1.3: Prvotná implementácia zdieľanej VR.

Modrá enita na obrázku 1.3 reprezentuje vytvorený server, ktorý prijíma a odosiela aktuálne informácie o pozícii a rotácii pripojeného používateľa a jednorázovo po pripojení odosiela informácie o novo vytvorenom sockete. *Socket.io* server bol patrične zmenený na *WebSocket* server.

1.7.2 Socket.io implementácia

Implementácia serverovej časti

Z hľadiska serverovej časti pre zdieľanie entít vo virtuálnej scéne je potrebné vyriešiť tri hlavné problémy. Prvým z nich je pripojenie klienta na server, čo je vyriešené fragmentom zdrojového kódu 1.4. Po prihlásení sa inicializuje stav klienta

s jeho pozíciou, rotáciou a farbou vykreslenej enity. Akonáhle sa klient pripojí, je potrebné odoslať všetkým pripojeným klientom správu, že došlo k zmene v počte klientov, aby túto zmenu odzrkadlili na svojej strane, teda vykreslili danú klientsku entitu. Samotný server predstavuje *Node.js*[13] server inicializovaný prostredníctvom rámca *Express.js*[14] s intuitívnym API pre rýchle vytváranie serverov.

```
const clients = {}
ioServer.on('connection', (client) => {
  clients[client.id] = {
    position: [0, 0, 0], rotation: [0, 0, 0, 0],
    color: '#' + Math.floor(Math.random() * 16777215).toString(16)
  }
  ioServer.sockets.emit('move', clients)})
```

Zdrojový kód 1.4: Socket.io-server pripojenie

Keďže server čaká na správu zo strany klienta o aktualizácií vlastnej pozície, bolo potrebné implementovať časť kódu, ktorá by po prijatí správy o pohybe aktualizovala uložené údaje pre klienta a následne rozoslala zmeny ostatným klientom. Túto implementáciu zobrazuje zdrojový kód 1.5 a daná akcia bola pomenovaná ako „move“ (t.j. pohyb).

```
client.on('move', ({ id, position, rotation }) => {
  if (id !== undefined) {
    clients[id].position = position
    clients[id].rotation = rotation
    ioServer.sockets.emit('move', clients)}})
```

Zdrojový kód 1.5: Socket.io-server akcia pohybu

Aby vykreslené entity vo virtálnej scéne rámcom *A-Frame* po odhlasení klienta zo systému neostali vykreslené, je nutné po odhlásení klienta zo servera záznam o jeho existencii zmazať a tieto informácie rovnako poslať ostatným, čo zobrazuje fragemnt kódu 1.6.

[13] https://nodejs.org/en
[14] https://expressjs.com/

```
client.on('disconnect', () => {
  delete clients[client.id]
  ioServer.sockets.emit('move', clients)
})
```

Zdrojový kód 1.6: Socket.io-server odhlásenie

1.7.3 Implementácia klientskej časti

Na strane klienta bola implementácia relatívne jednoduchá nakoľko ide len o to, aby v daných časových intervaloch klient o sebe posielal informácie na server. Teda bol vytvorený riadiaci komponent, ktorý získava informácie zo scény o pozícii a rotácii a po pripojení na server v časových intervaloch odosiela tieto údaje.

Takýto komponent by v *React* rámci vyzeral podobne ako popisuje fragment zdrojového kódu 1.7, pričom hlavné využitie predstavuje *useEffect* hook pre nastavenie socketu a pri odoslaní správy na server po zmene pozície alebo rotácie entity klienta.

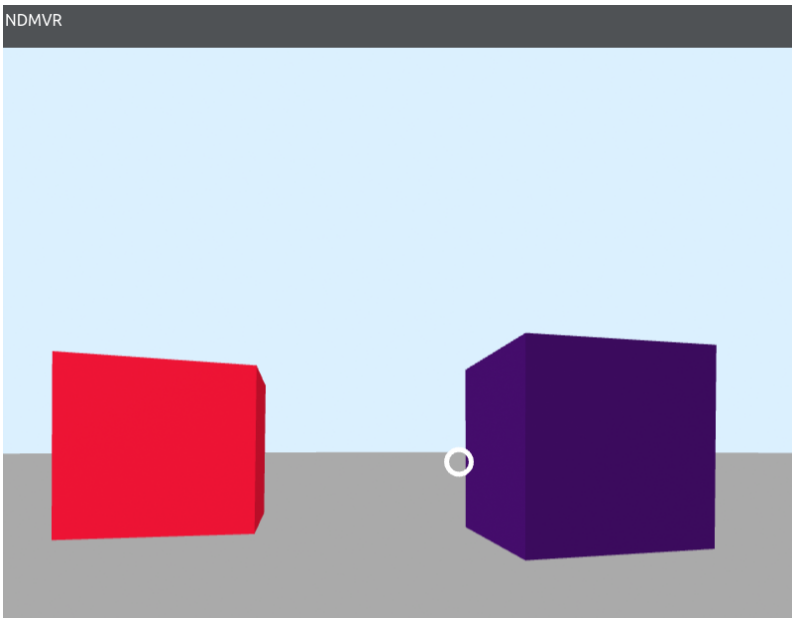
```
import io from 'socket.io-client';
const Controller = () => {
    useEffect(() => {
        const socket = io('http://localhost:5000');
        setSocket(socket);
    }, [])

    useEffect(() => {
        socket.emit('move', {position, rotation});
    }, [position, rotation])
    ...
}
```

Zdrojový kód 1.7: NDMVR klient - server komunikácia

Pre zobrazovanie vykreslených entít a odzrkadľovanie zmeny ich stavu bol vytvorený komponent *UserAvatar*2.3.3, ktorý predstavuje klientsku entitu. Návrh tohto komponentu a samotné zobrazenie vrámci scény je bližšie popísané v kapitole 2.3.3 o návrhu klientskych avatarov.

1.7.4 Zobrazenie v NDMVR



Obr. 1.4: Zdieľaný priestor v NDMVR

Obrázok 1.4 zobrazuje výslednú implementáciu pomocou *Socket.io* v zdieľanom prostredí *NDMVR*. Jednotlivé farebné kocky predstavujú pripojených klientov, ktorí sa navzájom vo virtuálnom prostredí pohybujú a vrámci ktorého sa vidia. Pre získanie tohto obrázka boli použití traja klienti, teda jeden v tomto prípade predstavuje kameru.

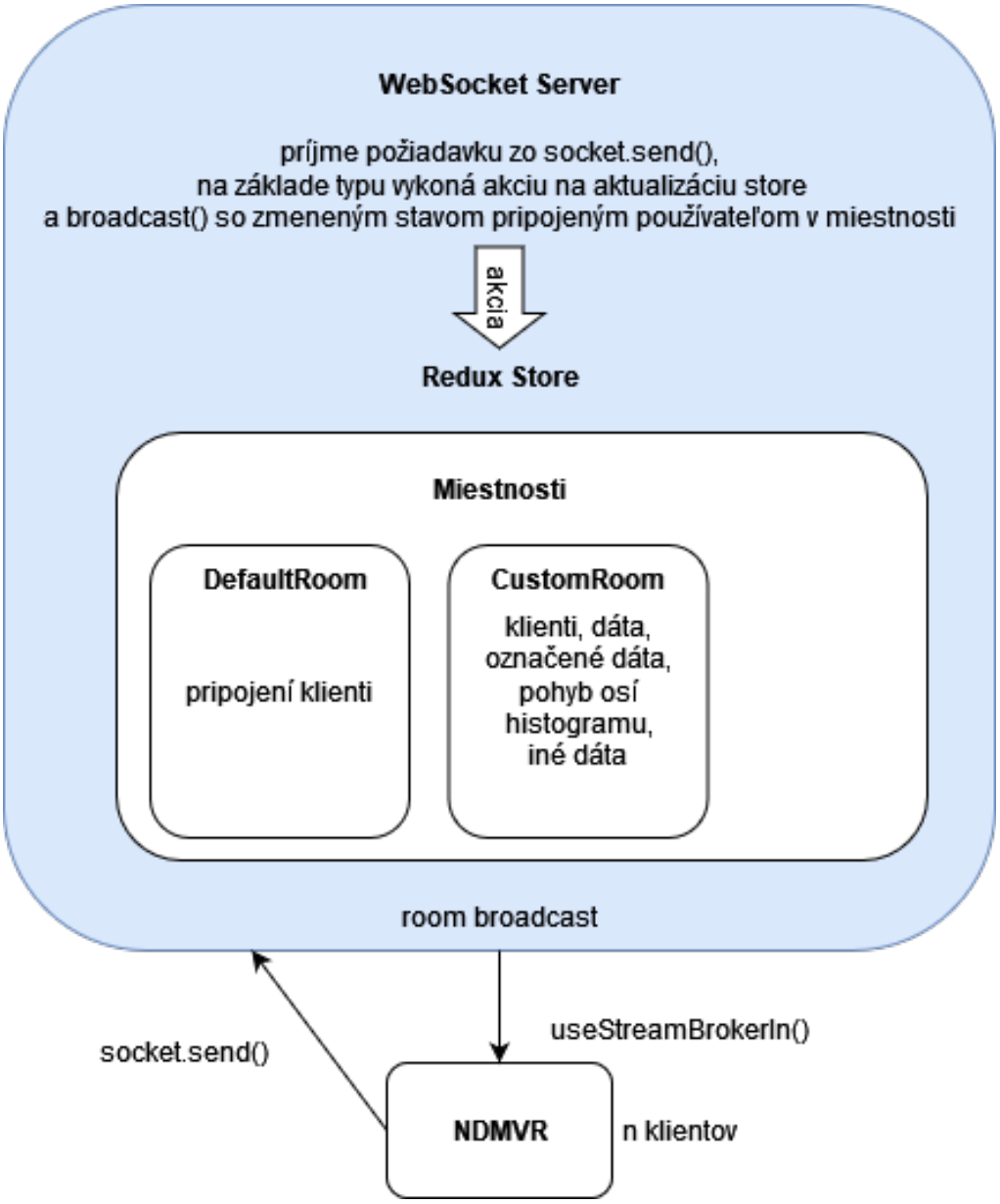
2 Syntetická časť

Hlavný návrh pre vytvorenie servera a zabezpečenia funkcionality na strane klienta v projekte *NDMVR* bol rozdelený do viacerých komponentov pre lepšie pochopenie a implementáciu vyžadovaného riešenia. Štruktúra súborov 2.1 zobrazuje usporiadanie súborov v projekte a odkazuje na jednotlivé kapitoly tejto práce, ktoré sa danými súbormi zaoberajú.

```
server/
├── store/
│   ├── reducers/
│   │   └── sharedReducer.js ..... definované akcie
│   └── store.js
├── index.js ..... prijímanie správ a vykonávanie akcií
├── Dockerfile ..... časť 2.28
└── package.json
⋮
NDMVR
├── ⋮
│   └── NDMVRShared/
│       ├── components/
│       │   ├── AdjustHistogramAxisController.jsx ..... časť 2.24
│       │   ├── AssignMasterController.jsx ..... časť 2.4.1
│       │   ├── Controller.jsx ..... časť 2.11
│       │   ├── SelectedBinsController.jsx ..... časť 2.18
│       │   └── UserAvatar.jsx ..... časť 2.3.3
│       ├── forms/
│       │   └── WebSocketConnectForm.jsx
│       └── NdmVrShared.jsx ..... wrapper pre komponenty
└── ⋮
```

Obr. 2.1: Štruktúra projektu

Po zmene protokolu na komunikačný protokol *Websocket* vyzeral návrh riešenia pre zdieľanú virtuálnu realitu projektu *NDMVR* s manažmentom miestností a ukladania stavu do *Redux store* podobne ako je možné vidieť na obrázku 2.2.



Obr. 2.2: Návrh servera pre zdieľanie dát

2.1 Implementácia WebSocket servera

Ako bolo spomenuté v analytickej časti, po zmene požiadaviek na túto prácu bol zmenený aj komunikačný protokol na *WebSocket* s cieľom dodržať zaužívané postupy projektu *React-NDMSPC* a integrovať riešenie po technologickej stránke. Keďže knižnica *Socket.io* nadväzuje na *WebSocket* riešenie, ale nie je jeho implementáciou, zavedené zmeny z hľadiska zdrojového kódu boli minimálne, čo odzrkadľujú fragmenty zdrojového kódu 2.1 a 2.2. Hlavným problémom v tomto prípade bolo uchovávanie stavu o zdieľaných entitách (viď. časť 1.6), ktorý bol vyriešený zavedením *Redux* knižnice a následným získavaním informácii o stave

z globálneho stavu aplikácie „store“.

```
const clients = {}
wss.on('connection', (ws) => {
    clients[client.id] = {
        position: [0, 0, 0],
        rotation: [0, 0, 0, 0],
        color: '#' + Math.floor(Math.random() * 16777215).toString(16)
    }
})
```

Zdrojový kód 2.1: Websocket pripojenie a vytvorenie záznamu o klientovi

Vyššie uvedený zdrojový kód 2.1 zobrazuje potrebné vykonané zmeny oproti *Socket.io* implementácii a zachovanie stavu o klientskej entite na serveri. Fragment kódu 2.2 predstavuje aktualizáciu stavu uloženého na serveri a následné rozoslanie týchto aktualizovaných informácií všetkým pripojeným klientom na *Websockete*. Samotné ukladanie stavu na serveri bolo neskôr nahradené globálnym „store“ využitím *Redux* popísané v kapitole 2.2.

```
case "move":
    if (packet.content.id !== undefined) {
        clients[id].position = packet.content.position
        clients[id].rotation = packet.content.rotation

        ws.send(
          JSON.stringify({
            type: 'move',
            content: clients
          })
        )
      }
    break;
```

Zdrojový kód 2.2: Pohyb klienta - server

2.2 Redux store implementácia

Na základe spätnej väzby z konzultácií a vzájomných konverzácií o ideálnom prevedení zdieľania polohových a rotačných dát o používateľoch vo VR priestore, boli projektové požiadavky opätovne pozmenené a časti 1.7.2 a 2.1 výrazne pozmenené. Komunikácia naďalej prebiehala cez Websocket server a klienta, avšak spôsob akým sa dáta preposielajú, resp. zdieľajú je následovný.

1. Každý klient uchováva informáciu o svojej pozícií a rotácii v objekte, ktorý je uložený na serveri.
2. Pomocou rôznych typov akcií - v tomto prípade MOVE, je možné zistiť, kedy sa daný klient pohol.
3. Pri obdržaní objektu s typom akcie MOVE, sa dáta uložia, ak došlo zmene.
4. Dáta sa odošlu všetkým pripojeným pre zobrazenie zmeny v prostredí.

Vyššie popísaný prístup však nezohľadňuje problém, ktorý predstavuje veľké množstvo posielaných dát v jednom čase. Ak sa pripojí veľké množstvo klientov, každé prerenderovanie scény znamená zmenu pozícií na scéne, a teda odoslanie dát. Tento spôsob je náchylný na preťaženie sieti a výrazné spomalanie celého systému. Z tohto dôvodu bola implementovaná optimalizácia, ktorá zabezpečuje, aby sa predišlo spomaleniu popísaná v kapitole 2.5.3.

Keďže prvotne bolo odosielanie dát naviazané na prerenderovanie scény a teda každý frame by znamenal jednu zmenu, zmeny sa odosielali po istých časových intervaloch, konkrétne každých 333ms. To však znamenalo, že používateľské entity vo VR scéne by boli nezosynchronizované s reálnymi pozíciami. Ako riešenie bola využitá implementácia vlastných hookov, ktoré umožňujú čakať na zmenu stavu a pri zmene (namiesto každého prekleslenia scény) ju odoslať na server. Tieto hooky predstavujú rovnakú implementáciu ako bolo popísané vo fragmente zdrojového kódu 1.7.

2.2.1 Konfigurácia Redux store

Prvotným krokom k dosiahnutiu cieľa zdieľania VR prostredia bolo nakonfigurovanie globálneho stavu *Redux store*, vrámci ktorého sa ukladá aktuálny stav scény, konkrétne informácie o entitách (klienti), zvýraznené časti grafu a mapovanie akcií pre zosynchronizovanie neskôr pripojených používateľov. Spoločne s inicializáciou dát o klientoch, konfiguráciu pre zdieľanie dát zobrazuje fragment zdrojového kódu 2.3 uvedený nižšie.

```
const initialState = {
  clients: [],
  master: null,
  selectedBins: [],
  axisMap: { X_UP: 0, X_DOWN: 0, Y_UP: 0, Y_DOWN: 0, Z_UP: 0, Z_DOWN: 0 },
  latestAxis: null,
  histogram: null,
  otherData: null
}
```

Zdrojový kód 2.3: Počiatočný stav globálneho stavu servera

Potrebné inicializované stavy vo fragmente zdrojového kódu 2.3 predstavujú pripojených klientov (identifikátor, aktuálna pozícia, rotácia a farba), rolu master (identifikátor), označené entity binov na scéne, pohyby osí histogramu, dáta zdieľaného histogramu a iné dáta (pre uľahčenie implementácie v budúcnosti).

Každej takejto inicializovanej hodnote prislúcha funkcia, ktorá aktualizuje jej stav na základe prijatých dát zo strany klienta a rozdistribuje tento stav pre všetkých klientov pripojených klientov z *NDMVR* scény. Jednotlivé funckie, ktoré pracujú s týmito stavmi sú popísané v ďalších častiach tejto práce, konkrétne kapitola 2.2.4, týkajúcich sa priradení *master* roly klientovi, zdieľaniu dát a prostredia naprieč všetkým pripojeným klientom.

2.2.2 Konfigurácia miestností pre klientov

Na základe požiadaviek a vychádzajúc z faktu, že inicializovaný stav popísaný v predchádzajúcej kapitole predstavuje globálny stav na serveri, teda umožňuje len jedno zdieľanie dát, jeden typ označených entít binov, jednu master rolu a podobne, rozhodli sme sa o vytvorenie tzv. miestností (z angl. room), ktoré ďalej rozvrstvia globálny stav aplikácie na viacero „pod-serverov“ (ale na jednom serveri), inicializácia globálneho stavu sa zmenila následovne, zobrazená fragmentom zdrojového kódu 2.4.

```
const initialRoomState = {
  clients: [],
  master: null,
  selectedBins: [],
  axisMap: { X_UP: 0, X_DOWN: 0, Y_UP: 0, Y_DOWN: 0, Z_UP: 0, Z_DOWN: 0 },
  latestAxis: null,
  histogram: null,
  otherData: null
  }

  const initialState = {
      defaultRoom: initialRoomState
  }
```

Zdrojový kód 2.4: Počiatočný stav globálneho stavu servera s miestnosťami

Táto zmena umožnila definíciu východiskového na oko pod-servera alebo *lobby*, kde sa po prvotnom pripojení na server nezobrazujú na klientskej strane žiadne zmeny, ale stavy o pripojených klientoch sa ukladajú.

2.2.3 Odosielanie a prijímanie dát

Na základe konzultácií a odporúčaní o tom, ako čo najefektívnejšie zdieľať dáta a hlavne aby zavedené princípy fungovali s už implementovanými funkciami v projektoch *NDMVR, React-NDMSPC-Core a React-NDMSPC* boli adaptované funckie na odosielanie a prijímanie dát následovne.

Pre prijímanie dát bola využitá funkcia z *React-NDMSPC-Core* knižnice s názvom *useStreamBrokerIn*, kde parameter tejto funkcie je pomenovanie typu dát odchádzajúcich broadcatst-om zo servera. Využíva kontext socketu, ktorý bol vytvorený formulárom pre prihlasovanie sa na server popísanom v kapitole 2.3.1. Fragment kóde 2.5 v časti prijímania dát popisuje ako presne využiť túto funkciu, pričom pre príklad bol uvedený typ „move“, pre zdieľanie informácií o pripojených klientoch. Po obdržaní správy zo servera sa nastaví stav *React* stav so zmenenými dátami a následne prebehne prerenderovanie *A-Frame* scény.

Odosielanie dát, rovnako ako ich prijímanie, využíva kontext socketu, pričom sa pomocou funkcie *send* odošlú všetky potrebné dáta s daným typom. Na základe tohto typu implementovaný server rozozná druh správy a následne prevedie korešpondujúce akcie na uloženie stavu a odoslanie zmenených dát všetkým používateľom vrámci jednej miestnosti. Časť odosielania dát vo fragmente

zdrojového kódu 2.5 zabrazuje získanie inštancie vytvoreného socketu a následne pomocou funckie *send*() odoslanie potrebných informácií na server.

```
// Prijímanie dát
const move = useStreamBrokerIn('move', wssb)

const [clients, setClients] = useState([])

  useEffect(() => {
    if (move.payload !== undefined) {
      move.payload.clients.length > 0
        ? setClients(move.payload.clients)
        : setClients([])
    }
  }, [move.payload])

// Odosielanie dát
const sb = useContext(NdmSpcContext)[wssb]
sb.send({
    type: 'move',
    payload: {
      position: absolutePosition.toArray(),
      rotation: rotationQuaternion.toArray()
    }
  })
```

Zdrojový kód 2.5: Prijímanie a odosielanie dát - príklad

2.2.4 Definovanie akcií pre zmenu stavu

Akcie pre zmenu globálneho stavu „store“ popisuje tabuľka 2.1, ktorá priradzuje názvu globálneho stavu konkrétny typ dát aké sa budú na serveri ukladať a definovanú akciu, ktorá tento stav zmení. Ako bolo popísané v kapitole 2.2.2, rozhodli sme sa vytvoriť možnosť priradenia klienta do miestnosti, pričom každá miestnosť je definovaná iniciálnym stavom zobrazenom v 2.4.

AxisMap použitá v tabuľke 2.1 predstavuje posun osí histogramu a pričom sa zároveň nastaví aj stav *latestAxis* na poslednú zmenenú hodnotu. Na základe celej mapy dokážeme synchronizovať klienta po pripojení na server, následne nám stačí posledná zmena vykonaná master klientom.

Stav histogramu je ukladaný ako *string* nakoľko objekt predstavujúci histo-

gram na scéne predstavuje veľké množstvo dát a teda bol transformovaný pomocou funkcií *JSRoot - parse, toJSON* a *JSON.parse, JSON.stringify*. Iné dáta v stave *otherData* ukladáme podobne ako rozparsovaný objekt na strane servera.

Globálny stav	Typ stavu	Priradené akcie zmeny
InitialState	initialState	CREATE_ROOM
Clients	Pole [Id, Pozícia, Rotácia, Farba]	ADD_USER, REMOVE_USER, UPDATE_USER
Master	[Id]	ASSIGN_MASTER, DEASSIGN_MASTER
AxisMap	[X_UP: 0, X_DOWN: 0, Y_UP: 0, Y_DOWN: 0, Z_UP: 0, Z_DOWN: 0]	AXIS_SCALING
Histogram	serializovaný objekt do *string*-u	SET_HISTOGRAM
OtherData	objekt iných dát	SET_OTHER_DATA

Tabuľka 2.1: Priradené akcie stavom

Na základe implementácie miestností je potrebné každú definovanú akciu zásobovať jej potrebnými parametrami, ktoré stav zmenia a názvom miestnosti, pre ktorú sa má daná zmena udiať. Dodatočne teda, každá akcia zmeny získava aj názov miestnosti, do ktorej je klient aktuálne priradený. Pri prvotnom pripojení na server predstavuje názov miestnosti string „defaultRoom“, ktorý korešponduje s názvom atribútu iniciálneho stavu „store“.

2.3 Návrh zdieľania používateľských entít a manažment miestností

Po zmene inciálneho stavu Redux „store“, ako bolo popísané v kapitole 2.2.2, boli nadefinované tri funckie, ktoré reprezentujú vytvorenie samotných miestností: *createRoom*(), *joinRoom*() a *leaveRoom*(). Rovnako je potrebné zabezpečiť spôsob pripojenia sa na server a do miestnosť.

Hlavnou prioritou, aby bola splnená požiadavka zdieľaného prostredia, bol korektný návrh pre zdieľanie informácií o pripojených klientoch v hlavnej scéne projektu. Aby sme zaistili plytký pohyb používateľov po scéne a zabezpečili odlíšenie jednotlivých používateľov, potrebujeme primárne štyri vlastnosti: identifikátor (generovaný serverom po pripojení), aktuálna pozícia a rotácia, a farba (rovnako generovaná po pripojení na server).

2.3.1 Implementácia používateľkých miestností

Ako bolo uvedené vyššie, hlavnú funkcionalitu tejto časti predstavujú tri funckie a to: *createRoom*(), *joinRoom*() a *leaveRoom*(). Funkcia *createRoom*() vyvtorí novú miestnosť pre používateľov, ak miestnosť s daným názvom neexisuje, a po vytvorení zapíše klienta, ktorý ju vytvoril do zoznamu klientov a presunie klienta

zo základnej miestnosti do novo vytvorenej. Funkcia *joinRoom*() predstavuje pripojenie klienta do miestnosti, ktorá už existuje a funckia *leaveRoom*() zabezpečí odstránenie klienta z aktuálnej miestnosti a premiestni ho do základnej miestnosti „defaultRoom“. Fragmenty kódu 2.6, 2.7, 2.8 popisujú práve funkcionalitu týchto troch funkcií.

```
1  const createRoom = (roomName, client) => {
2  store.dispatch({
3      type: 'CREATE_ROOM',
4      payload: { roomName: roomName }
5    })
6    store.dispatch({
7      type: 'REMOVE_USER',
8      payload: { id: client.id, roomName: 'defaultRoom' }
9    })
10   store.dispatch({
11     type: 'ADD_USER',
12     payload: { id: client.id, roomName: roomName }
13   })}}
```

Zdrojový kód 2.6: Funkcia createRoom()

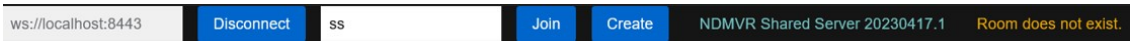
```
1  const joinRoom = (roomName, client) => {
2    store.dispatch({
3      type: 'REMOVE_USER',
4      payload: { id: client.id, roomName: 'defaultRoom' }
5    })
6    store.dispatch({
7      type: 'ADD_USER',
8      payload: { id: client.id, roomName: roomName }
9    })
10 }
```

Zdrojový kód 2.7: Funkcia joinRoom()

```
1  const leaveRoom = (roomName, client) => {
2    store.dispatch({
3      type: 'REMOVE_USER',
4      payload: { id: client.id, roomName: roomName }
5    })
6    store.dispatch({
7      type: 'ADD_USER',
8      payload: { id: client.id, roomName: 'defaultRoom' }
9    })
10 }
```

Zdrojový kód 2.8: Funkcia leaveRoom()

Na klientskej strane, teda v *React* aplikácii *NDMVR* projektu, bol vytvorený formulár, ktorý zabezpečuje pripojenie klienta na server a do danej miestnosti. Vhľadom na to, že server môže byť na rôznej IP adrese, je potrebné, aby používateľ mohol nadefinovať IP adresu serveru kam sa prihlási (viď. časť 2.28). Keďže je viacero obmedzení, kedy sa do miestnosti nie je možné pripojiť alebo ju nie je možné vytvoriť (popísané vyššie), bolo potrebné klientovi odoslať chybovú hlášku, aby nedošlo k zmätkom. Samotný formulár a chybovú hlášku popisuje obrázok 2.3.



Obr. 2.3: Formulár pre pripojenie klienta na server

2.3.2 Implementácia zdieľania používateľských entít

Fragment kódu 2.2 využívajúci *Socket.io* bol pozmenený, aby využíval *Redux store* akcie s názvom: *ADD_USER* pri prvotnom pripojení sa na server a následne akciu *UPDATE_USER* pre ukladanie stavu entít používateľov. Tieto dve funkcie popisujú fragmenty zdrojového kódu 2.9 a 2.10 uvedené nižšie.

```
case 'ADD_USER':
  roomName = action.payload.roomName
  userId = action.payload.id
  return {
    ...state,
    [roomName]: {
      ...state[roomName],
      clients: [
        ...state[roomName].clients,
        {
          id: userId,
          position: [0, 0, 0],
          rotation: [0, 0, 0, 0],
          color: '#FFFFFF'}]}}
```

Zdrojový kód 2.9: Akcia pre pridanie stavu klienta

Obidve tieto akcie pre zmenu stavu vyžadujú parameter s aktuálnym identifikátorom klienta, ktorý bol do miestnosti pridaný alebo sa pohol vrámci scény a názov miestnosti, vrámci ktorej bol pohyb na scéne zaznamenaný.

```
roomName = action.payload.roomName
  userId = action.payload.id
  return {
    ...state,
    [roomName]: {
      ...state[roomName],
      clients: state[roomName]?.clients?.map((client) =>
        client.id === userId
          ? {
              ...client,
              position: action.payload.position,
              rotation: action.payload.rotation,
              color: action.payload.color
            }
          : client )}}
```

Zdrojový kód 2.10: Akcia pre obnovenie stavu klienta

Z hľadiska klientskej strany aplikácie *NDMVR* bol vytvorený komponent *Use-*

rAvatar, ktorého atribúty sú identifikátor, pozícia, rotácia, identifikátor aktuálneho master používateľa a farba kienta, ktorá mu bola priradená.

Výpočet rotácie je uvedený v následujecej kapitole 2.3.3, keďže vyžadoval prepočet, avšak pri zisťovaní pozícií bolo rovnako potrebné využiť funckionalitu *Three.js* rámca, ktorá nám umožní získať absolútnu pozíciu a rotáciu entity klienta na scéne. Bolo potrebné získať referenciu pre kameru používateľa a následne podľa jej pozície získať absolútne súradnice na *A-Frame* scéne. Pri tomto procese je potrebné súradnice premeniť na 3D vector pre pozíciu a *Quaternion* pre rotáciu. Tieto získané informácie sme potom premenili na pole troch súradníc pre pozíciu a štyroch súradníc pre rotáciu. Túto funkcionalitu zobrazuje zdrojový kód 2.11, na ktorom je uvedený funckionálny *React* komponent s názvom *Controller*. Tento komponent odosiela informácie o aktuálnej pozícii a rotácii klienta každých 100ms pomocou *useInterval* hooku z *React* rámca.

```
1  const Controller = ({ wssb = 'shared' }) => {
2      const [time, setTime] = useState(0)
3      const sb = useContext(NdmSpcContext)[wssb]
4  useEffect(() => {
5      const camera = document.getElementById('camera')
6      const absolutePosition = new THREE.Vector3()
7      const rotationQuaternion = new THREE.Quaternion()
8      camera.object3D.getWorldPosition(absolutePosition)
9  const interval = setInterval(() => {
10   camera.object3D.getWorldPosition(absolutePosition)
11   camera.object3D.getWorldQuaternion(rotationQuaternion)
12   sb.send({
13     type: 'move',
14     payload: {
15       position: absolutePosition.toArray(),
16       rotation: rotationQuaternion.toArray()}})
17   setTime((prevTime) => prevTime + 1)
18 }, 100)
19 return () => clearInterval(interval)
20 }, [time])
21 return (<div></div>)}
```

Zdrojový kód 2.11: Zisťovanie aktuálnej pozície a rotácie klienta

2.3.3 Návrh entít avatarov pre klientov

Pre zobrazovanie klientov vo virtuálnej scéne a predchádzaniu zmätku v tom, čo používatelia na obrazovke vidia je potrebný správny návrh entít klientov.

Zdrojový kód pre *A-Frame* entitu klienta 2.14 na riadkoch 1 až 15, predstavuje hlavu avatara klientskej entity s očami(riadok 5-8, 10-15) a nosom na riadku 16 až 17. Riadok 18 predstavuje torso avatara, riadky 19-25 jeho ruky a riadky 27 až 34 jeho nohy. Takúto jednoduchú implementáciu umožnil fakt popísaný v kapitole 1.2 týkajúci sa relatívneho umiestňovania vnorených entít vzhľadom na rodičovské tagy.

Avšak, nakoľko je tento prístup vnorenia tagov jednoduchý pri implementácii sme sa stretli s problémom získavanie aktuálnej pozície a rotácie klientskych entít. Keďže rámec *A-Frame* poskytuje funkciu *getPosition*, ktorá vracia absolútnu pozíciu entity na scéne, obsahuje len funkciu *getRotation*, ktorá vracia len relatívnu rotáciu na rodičovskú entitu. Z tohto dôvodu bolo potrebné získať údaje o rotácii z rámca *Three.js*, na ktorom bol *A-Frame* postavený a prekonvertovať cez potrebné transformácie, ktoré zobrazuje zdrojový kód 2.12.

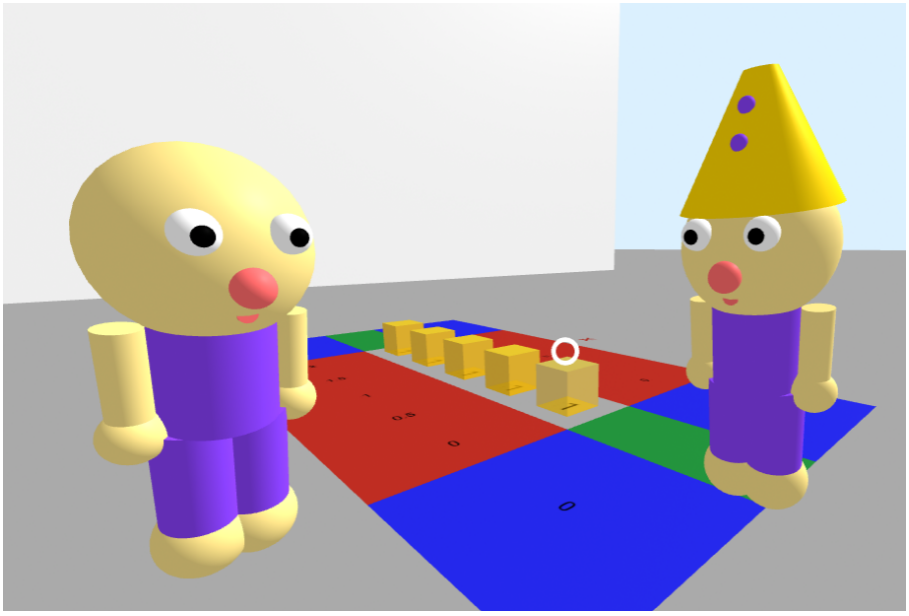
```
1  useEffect(() => {
2      const user = document.getElementById(id)
3      const quaternionToApply = new THREE.Quaternion().fromArray(rotation)
4      const threeEuler = new
       ↪  THREE.Euler().setFromQuaternion(quaternionToApply, 'YXZ')
5      const aframeRotation = {
6          x: THREE.MathUtils.radToDeg(threeEuler.x),
7          y: THREE.MathUtils.radToDeg(threeEuler.y),
8          z: -THREE.MathUtils.radToDeg(threeEuler.z)}
9      user.setAttribute('rotation', aframeRotation)
10 }, [rotation])
```

Zdrojový kód 2.12: Transformácia relatívnej rotácie na absolútnu

Výslednú entitu zobrazuje obrázok 2.4, na ktorom je možné vidieť 2 klientov z pohľadu tretieho. Jeden z nich má aj farebne odlíšenú čiapočku, ktorú generuje funkcia *renderTopHat* 2.13 a reprezentuje priradenie master roly, pre možnosť zdieľania prostredia, čo je bližšie popísané v časti 2.4.1.

```
const renderTopHat = () => {
    return (
        <a-entity position='0 0.25 0'>
            <a-cone
            color='gold'
            radius-bottom='0.5'
            radius-top='0.1'
            height='0.8'
            position='0 0.4 0'
            ></a-cone>
    <a-sphere color={color} radius='0.05' position='0 0.4 -0.3'></a-sphere>
    <a-sphere color={color} radius='0.05' position='0 0.6 -0.2'></a-sphere>
  </a-entity>)}
```

Zdrojový kód 2.13: Čiapočka pre master klienta



Obr. 2.4: Návrh avatara klientskej entity

```
<a-entity position={position} id={id}>
  <a-sphere radius='0.5' color='#F8DF87'>
    {/* TOP HAT */} {masterId === id ? renderTopHat() : <></>}
  </a-sphere>
  <a-entity position='0.2 0.15 -0.45'>
    <a-sphere radius='0.1' color='#FFFFFF'>
      <a-sphere radius='0.05' color='#000000' position='0 0
      ↪  -0.07'></a-sphere>
    </a-sphere>
  </a-entity>
  <a-entity position='-0.2 0.15 -0.45'>
    <a-sphere radius='0.1' color='#FFFFFF'>
      <a-sphere color='#000000' radius='0.05' position='0 0 -0.07'
      ></a-sphere>
    </a-sphere>
  </a-entity>
  <a-cylinder height='0.05' radius='0.05' color='#FF6B6B' position='0 -0.25
  ↪  -0.4' rotation='0 90 90'></a-cylinder>
  <a-sphere radius='0.1' color='#FF6B6B' position='0 -0.1 -0.5'></a-sphere>
  <a-cylinder height='0.8' radius='0.3' color={color} position='0 -0.6
  ↪  0'></a-cylinder>
  <a-entity position='-0.4 -0.4 0'>
    <a-cylinder height='0.6' radius='0.1' color='#F8DF87' position='0 -0.3
    ↪  0'></a-cylinder>
    <a-sphere radius='0.15' color='#F8DF87' position='0 -0.6 0'></a-sphere>
  </a-entity>
  <a-entity position='0.4 -0.4 0'>
    <a-cylinder height='0.6' radius='0.1' color='#F8DF87' position='0 -0.3
    ↪  0'></a-cylinder>
    <a-sphere radius='0.15' color='#F8DF87' position='0 -0.6 0'></a-sphere>
  </a-entity>
  <a-entity position='-0.15 -1 0'>
    <a-cylinder height='0.8' radius='0.2' color={color} position='0 -0.4
    ↪  0'></a-cylinder>
    <a-sphere radius='0.25' color='#F8DF87' position='0 -0.8 0'></a-sphere>
  </a-entity>
  <a-entity position='0.15 -1 0'>
    <a-cylinder height='0.8' radius='0.2' color={color} position='0 -0.4
    ↪  0'></a-cylinder>
    <a-sphere radius='0.25' color='#F8DF87' position='0 -0.8 0'></a-sphere>
  </a-entity>
</a-entity>
```

Zdrojový kód 2.14: Entita klienta

2.4 Návrh pre zdieľanie dát

Pre zdieľanie experimentálnych dát, ktoré sa majú zobrazovať, sme sa rozhodli využiť React-Redux (viď. časť 1.6.3) knižnicu, ktorá umožňuje ukladať aktuálny stav dát na strane klienta (teda toho, kto dáta zdieľa (*MASTER*)), takým štýlom, aby všetci ostatní boli schopní zobraziť dané zdieľané dáta rovnako. Tento prístup bol zvolený hlavne z dôvodu, že využitie tejto knižnice pri implementácii zobrazovania používateľov vo VR prostredí sa osvedčilo, nielen z hľadiska správnosti implementácie, ale aj časovej odozvy pri preposielaní dát.

Keďže hlavnou myšlienkou bolo to, aby dáta zdieľal vždy len jeden používateľ (teda *MASTER*) a ostatní (*LISTENER*) len tieto zmeny dát adekvátne zobrazovali, bolo potrebné najprv vytvoriť rozlíšenie medzi klientmi. Pre prípad, že *MASTER* ešte nie je určený, prvý používateľ resp. pripojený klient, po stlačení klávesy M, odošle na server požiadavku o prevzatie *master* statusu pomocou ekvivalentnej akcie *ASSIGN_MASTER* . Serverová časť zabezpečí priradenie tohto statusu pre daný klienstký socket a umožní tak zdieľanie dát. Akonáhle sa pripojia iní klienti, server im priradí status *LISTENER* a aktualizuje zobrazené dáta získané od *MASTER*. Ak si *LISTENER* vyžiada rolu *MASTER*-a stlačením klávesy M, bude zamietnutá z dôvodu, že *MASTER* už je priradený. *LISTENER* môže teda získať status na zdieľanie dát, až po uvoľnení *MASTER* statusu rovnakou klávesou, alebo jeho úplným odpojením zo servera.

Keďže počet *MASTER* statusov v miestnosti je obmedzený na jedného klienta, dáta prechádzajú len jedným smerom a to od *MASTER*-a k *LISTENER*-om. Takýmto spôsobom bolo zaručené, že dáta, ktoré majú byť zdieľané zo strany *MASTER*-a na stranu pripojených *LISTENER*-ov boli konzistentné a synchronizované pre všetky pripojené sockety, a aby sa predišlo rozdielom v zobrazovaných dátach.

Avšak v neskoršej implementácii došlo k rozhodnutiu ponechania možnosti komunikácie každý s každým, a preto v prípade, ak *MASTER* pre danú miestnosť neexistuje, je povolená komunikácia každý s každým a v prípade ak sa niekto prihlási o rolu zdieľania dát, len dáta žiadajúceho klienta budú vyzdieľané. Tento fakt však neplatí pre zdieľanie dát histogramu, nakoľko vyžaduje priradenú rolu *MASTER* aspoň jednému používateľovi v danej miestnosti.

2.4.1 Implementácia priradenia master roly pre klienta

Na strane používateľa, teda klienta, bolo v prvom rade potrebné zabezpečiť, aby mohol požiadať o priradenie roly na serveri. Túto časť opisuje fragment zdrojo-

vého kódu 2.3, kde je definovaná funkcia pre poradenie si so stlačenou klávesou M, pričom sa odošle požiadavka na server. V tomto prípade bola po konzultácií vybraná klávesa *m* ktorému prislúcha hodnota 77, a akcia, ktorá reprezentuje túto požiadavku bola pomenovaná relevatne ako *assign master*. Cez socket (React-NDMSPC socket pre Redux store) je po stlačení klávesy následne odoslaná táto akcia s prázdnym objektom *payload* , na koľko nepotrebujeme dodefinovať žiadne ďalšie dáta.

```
1 const _handleKeyM = (event) => {
2   switch (event.keyCode) {
3     case M_KEY:
4       socket.send(
5         JSON.stringify({
6           type: 'assign_master',
7           payload: {}
8         }))
9       }}
```

Zdrojový kód 2.15: Priradenie master roly používateľovi - klient

Po prijatí požiadavky na server kód zobrazený na fragmente zdrojového kódu 2.16 rozpozná prijatú akciu, do *Redux Store* pošle akciu pre zmenu stavu s typom *ASSIGN MASTER* a objektom s údajom *id* socketa, na ktorom požiadavku prijal. Následne je zmena stavu spracovaná pomocou Redux store a výsledný stav (buď zmenený alebo nezmenený) je rozoslaný všetkým klientom (používateľom), aby mohli aktualizovať svoj stav.

```
const masterInfo = (data) => {
  return {
    type: 'master',
    payload: {master: data}}}

if (parsedMessage.type === 'assign_master') {
  store.dispatch({
    type: 'ASSIGN_MASTER',
    payload: { master: ws.id, roomName: currentRoom }})
  wss.broadcast(JSON.stringify(
    currentRoom,
    masterInfo(store.getState().shared[currentRoom].master))))}
```

Zdrojový kód 2.16: Priradenie master roly používateľovi - server broadcast

Fragment kódu 2.17 predstavuje práve logiku pre zmenu stavu v *Redux Store*. Ak rola master nie je priradená žiadnemu používateľovi, server ju priradí socketu, ktorý o to požiadal. Ak sa iný klient pokúsi získať rolu mastra, pričom táto rola je obsadená už iným mastrom zmena stavu neprebehne a všetkým klientom bude odoslaný aktuálny stav s nezmeneným používateľom, ktorý predstavuje mastra. Jedine klient, ktorý aktuálne drží rolu mastra je schopný požiadať o uvoľnenie, a to opätovným stlačením klávesy alebo odpojením sa zo servera, pričom po zmene stavu takýmto štýlom môže akýkoľvek iný klient vytvoriť požiadavku pre získanie master roly.

Kód bol pôvodne rozdelený do dvoch akcií pre zmenu stavu a to: akcia typu *ASSIGN_MASTER* a *DEASSIGN_MASTER*, avšak vzhľadom na to, že v každom momente môže byť priradená len jedna rola mastra pre danú miestnosť, rozhodli sme sa logiku spojiť a akciu *DEASSIGN_MASTER* využiť len pri odpojení klienta zo servera. Implementačne zodpovedá jedna druhej, z toho *DEASSIGN_MASTER* v tejto práci nebol uvedený.

```
case 'ASSIGN_MASTER':
  roomName = action.payload.roomName
  userId = action.payload.master
if (state[roomName].master) {
    if (state[roomName].master === userId) {
      return {...state, [roomName]: {
          ...state[roomName], master: null, selectedBins: [],
          latestAxis: null, axisMap: { X_UP: 0, X_DOWN: 0, Y_UP: 0, Y_DOWN:
          ↪  0, Z_UP: 0, Z_DOWN: 0 }, histogram: null, otherData: null}}
    } else { return state }
} else { return {...state, [roomName]: {...state[roomName], master:
↪  userId}}}
```

Zdrojový kód 2.17: Priradenie master roly používateľovi - server Redux action

Pre účely prehľadu implementovaného kódu akcie *ASSIGN_MASTER* bol fragment zdrojového kódu 2.17 ukrátený o pár riadkov týkajúcich sa logovania stavu aplikácie a odsadzovania kódu, pretože tie nezachytávajú gro akcie a rozhodli sme sa zachytiť implementovanú funkciu z hľadiska podstaty. Takýmto štýlom sú fragmenty zdrojových kódov akcií zobrazené aj v ďalších kapitolách.

2.4.2 Implementácia zdieľania dát o entitách

Implementácia zdieľania dát prebiehala rovnako ako v časti o priradení master roly, teda bolo potrebné zabezpečiť vytvorenie požiadavky na strane klienta po označení binu, následné spracovanie požiadavky na strane servera a rozposlanie aktualizovaného stavu všetkým klientom. Na základe konzultácií bolo rozhodnuté, že zdieľanie označených binov má fungovať aj keď nie je priradená rola master, tým pádom zdieľanie prebehne vrámci všetkých pripojených používateľov. Riešenie sa líší v tom, že potrebujeme ukladať id binov, ktoré sa majú zdieľať, teda bolo potrebné posielať dáta o binoch a na strane servera zabezpečiť rozdistribuovanie týchto informácii podľa nastavenia master roly, teda buď MASTER to LISTENERS alebo ALL TO ALL. Táto časť práce pracuje s poľom *selected bins* definovaným v konfigurácii „Redux storu“ v časti 2.3.

Fragment kódu 2.18 predstavuje implementáciu odoslania informácií zvolených binov na server, pričom reaguje na funkciu *onClick*, ktorá po kliku na daný biny prevedie označenie zmenou farby. Typ akcie bol pomenovaný ako *selected_bins*, ktorý predstavuje aktuálne zvolené biny pomocou poľa identifikátorov. Samotné id binov, o ktorých informácia sa má odoslať boli vybrané priamo zo scény pomo-

cou už implementovanej funkcie, ktorá po akomkoľvek kliku vráti informácie o entite, nad ktorou bola prevedená a následne tieto údaje boli premapované na ich konkrétne identifikátory, ako zobrazuje framgment kódu 2.18. Funkcia *sendSelectedBinsToServer* v tomto prípade zabezpečuje odoslanie požiadavky na server pre zmenu stavu podobne ako 2.15 pomocou funckie *socket.send*().

```
const [selectedBins, setSelectedBins] = useState([])
const handleClick = (data) => {
  setSelectedBins((previousState) => {
    let newState
    if (previousState.some((x) => x.id === data.binId)) {
      newState = previousState ? [...previousState.filter((x) => x.id !==
      ↪  data.binId)] : []
    } else {
      newState = previousState ? [...previousState, { id: data.binId }] : [{
      ↪  id: data.binId }]
    }
    sendSelectedBinsToServer(sb, newState)
    return newState })}
```

Zdrojový kód 2.18: Zdieľanie označených binov - klient

Redux logika akcie prislúchajúcej požiadavke na strane klienta, ktorá aktualizuje stav zvolených binov v globálnom store je zobrazená na fragmente zdrojového kódu nižšie.

```
case 'SET_SELECTED_BINS':
    roomName = action.payload.roomName
    if (
      state[action.payload.roomName].master &&
      state[action.payload.roomName].master !== action.payload.master
    ) { return state } else {
      return { ...state,
        [roomName]: {...state[roomName], selectedBins:
        ↪  action.payload.selectedBins}}}
```

Zdrojový kód 2.19: Zdieľanie označených binov - akcia

Po prijatí id označených binov na serveri je odoslaná akcia na zmenu stavu do *Redux Store* a následne daný stav odošle všetkým pripojeným klientom pomocou

funkcie *broadcast*. Túto časť implementácie zobrazuje fragment zdrojového kódu 2.20 s preddefinovaným typom odpovede zo servera na riadkoch 1 až 5. Riadky 7 až 12 predstavujú vykonanie akcie *SET_SELECTED_BINS* a odoslaním pozmeneného stavu všetkým pripojeným klientom v danej miestnosti.

```
1 const selectedBins = (data) => {
2   return {
3     type: 'selectedBins',
4     payload: {
5       selectedBins: data, master: ws.id, roomName: currentRoom}}}
6
7 if (parsedMessage.type === 'selected_bins') {
8     store.dispatch({
9       type: 'SET_SELECTED_BINS',
10      payload: { master: ws.id, selectedBins:
        ↪  parsedMessage.payload.selectedBins}})
11    wss.broadcast(
12      currentRoom,
13      JSON.stringify(
14          selectedBins(store.getState().shared[currentRoom].selectedBins)))}
```

Zdrojový kód 2.20: Zdieľanie označených binov - server

2.5 Návrh pre zdieľanie prostredia

NDMVR poskytuje možnosť upravovania osí 3D histogramu, na ktorom sa dáta zobrazujú (napr. zväčšenie a zníženie rozsahu zobrazovaných dát). Preto implementácia zdieľania dát bola riešená podobne ako v 2.4 pomocou *MASTER* a *LISTENER* statusov.

Hlavným problémom tejto časti je vyriešenie zosychronizovania scény pre *LISTENER*-ov, ktorí sa pripojili na server už po zmene scény *MASTER*-om. V tomto prípade je potrebné zachovávať všetky akcie, ktoré sa vykonali *MASTER*-om a následne všetkých pripojených klientov previesť danými akciami. Pre implementáciu bola zvolená dátová štruktúra *map* pre zachovanie zmien na scéne, ktorá následne po pripojení *LISTENER*-a na server bude preposlaná a samotný klient zosynchronizuje stav scény s *MASTER* statusom.

2.5.1 Implementácia zdieľania dát histogramu

Pre zdieľanie dát histogramu je potrebné, aby aspoň jeden používateľ v danej miestnosti mal priradenú rolu *MASTER*, nakoľko je táto akcia na strane klienta spojená s odoslaním dát o histograme, ktorého dáta sa menia v čase. Nastavenie dát histogramu a ostatných zdieľaných informácií prebieha v *binDistributor* funkcii definovanej v knižnici *React-NDMSPC-Core*, ktorá zabezpečí zosynchronizovanie dát v komponentoch projektu *NDMVR* pomocou kontextu a *providera* dát do komponentu.

```
1    case 'SET_HISTOGRAM':
2        if (
3          state[action.payload.roomName].master &&
4          state[action.payload.roomName].master !== action.payload.master
5        ) {
6          return state
7        } else {
8          return {
9            ...state,
10           [action.payload.roomName]: {
11             ...state[action.payload.roomName],
12             histogram: action.payload.histogram }}}
```

Zdrojový kód 2.21: Akcia zdieľania dát histogramu

Vytvorenú akciu pre zmenu stavu globálneho *Redux store* zobrazuje fragment zdrojového kódu 2.21, ktorý pre aktuálnu miestnosť, v ktorej sa používateľ nachádza, nastaví na dáta v podobe stringu, aby sme predišli zbytočnému rozosielaniu objektov po sieti, ktoré by zbytočne komunikáciu zahltili.

Po prijatí správy typu *histogram* na serveri, vyšleme požiadavku o zmene stavu do *Redux store* a dáta rozošleme všetkým používateľom v danej miestnosti. Túto fukcionalitu zobrazuje časť zdrojového kódu 2.22.

```
1    const histogram = (data) => {
2      return { type: 'histogram', payload: { histogram: data}}}
3    if (parsedMessage.type === 'selected_bins') {
4      store.dispatch({
5        type: 'SET_HISTOGRAM',
6        payload: {
7          master: ws.id,
8          roomName: currentRoom,
9          histogram: parsedMessage.payload}})
10     wss.broadcast(
11       currentRoom,
12       JSON.stringify(
13         histogram(store.getState().shared[currentRoom].histogram))
14     )}
```

Zdrojový kód 2.22: Akcia zdieľania dát histogramu

Na strane klienta bolo potrebné dáta pred odoslaním a prijatím adekvátne pripraviť, nakoľko objekt dát histogramu predstavujú typ z knižnice *JSRoot*. Táto knižnica poskytuje funckie *parse* a *toJSON*, ktorých výsledok aj tak nepredstavuje *JSON* string. Preto, výsledok týchto funckii bolo potrebné pretransformovať na text pomocou funkcie *JSON.stringify*() a *JSON.parse*() pre vytvorenie objektu po zmene aktuálneho histogramu. Klientsku časť kódu zobrazuje fragment zdrojového kódu 2.23, kde zdieľanie dát je napojené na priradenie role *MASTER*.

```
const sb = useContext(NdmSpcContext)[wssb]
const { histogram } = useContext(StoreContext).data
const _handleKeyM = (event) => {
  switch (event.keyCode) {
      //  ... master požiadavka
      sb.send({
        type: 'histogram',
        payload: JSON.stringify(toJSON(histogram))
      })}}
  ...

  // PRIJATIE DÁT HISTOGRAMU
  ...
useEffect(() => {
  if (histogramResponse.payload !== undefined) {
    setHistogram(parse(JSON.parse(histogramResponse.payload.histogram)))
  }
}, [histogramResponse.payload])
```

Zdrojový kód 2.23: Zdieľanie dát histogramu

2.5.2 Implementácia zdieľania prostredia

NDMVR využíva klávesové skratky pre prácu s nastavovaním osí grafu, ktorý je vrámci scény zobrazený. Tieto klávesy sú I, J, K, L, U a O pre zväčšovanie a znížovanie rozsahu osí X, Y a Z. Teda I posun X osi o jedno, J posun Y osi o jedno viac, U posun Z osi o jedno viac a opačne.

Na strane klienta bola podobne namapované klávesy ako v konfigurácii *Redux Store* 2.3, teda jednotlivé klávesy predstavujú akcie I=X_UP, J=Y_UP, U=Z_UP, K=X_DOWN, L=Y_DOWN, O=Z_DOWN čo je zobrazené aj na fragmente zdrojového kódu 2.24, rovnako ako aj zabezpečenie odoslania akcie typu *axis_scaling*. V Javascripte majú klávesy I, J, K, L, U, O hodnoty 73, 74, 75, 76, 85, 79 v danom poradí.

```
const keyMap = new Map([[I_KEY, 'X_UP'], [K_KEY, 'X_DOWN'], [L_KEY, 'Y_UP'],
  ↪  [J_KEY, 'Y_DOWN'], [U_KEY, 'Z_UP'], [O_KEY, 'Z_DOWN']])

const dispatchAxisChange = (axisChange) => {
  const axisScalingLatest = useStreamBrokerIn('axisScalingLatest', wssb)
  switch (axisChange) {
      case 'X_UP':
          document.dispatchEvent(new KeyboardEvent('keydown', { key: 'I' }))
      break
      case 'X_DOWN':
          document.dispatchEvent(new KeyboardEvent('keydown', { key: 'K' }))
      break
    ...

  useEffect(() => {
      if (axisScalingLatest.payload !== undefined) {
      dispatchAxisChange(axisScalingLatest.payload.latestAxis)
  }
  }, [axisScalingLatest.payload])
```

Zdrojový kód 2.24: Zdieľanie prostredia - klient

Časť zdrojového kódu 2.24, konkrétne funckia *dispatchAxisChange* predstavuje posunutie osi u danú súradnicu, ktorá bola vložená ako parameter. Takýmto štýlom sme schopní všetky posunutia osí vykonať aj na strane *LISTENER* klientov, teda pohyb osi nastane na scéne priamo u klienta namiesto servera.

Vyriešenie problémového prípadu pri prvotnom pripojení klienta na server a zosynchronizovanie stavu s *MASTER* klientom bol vyriešený fragmentom zdrojového kódu 2.25, v ktorom sa iteruje po celej mape prijatých zmien osí, a tie sa následne vykonávajú na strane novo pripojeného klienta spustením klávesy bez jeho dotyku.

```
const axisScaling = useStreamBrokerIn('axisScaling', wssb)

useEffect(() => {
  if (axisScaling.payload !== undefined) {
    Object.keys(axisScaling.payload.axisMap).forEach((key) => {
      for (let i = 0; i < axisScaling.payload.axisMap[key]; i++) {
        dispatchAxisChange(key)
      }
    })
  }
}, [axisScaling.payload])
```

Zdrojový kód 2.25: Zdieľanie prostredia prvotné pripojenie - klient

Na fragmente zdrojového kódu uvedenom nižšie je zobrazená implementovaná logika redux akcie AXISSCALING, ktorá je zavolaná na strane servera po obdržaní akcie zo strany klienta s ciešom aktualizovať globálny Redux store stav.

```
case 'AXIS_SCALING':
    console.log('Setting axis' + action.payload.axis)
    if (state.master && state.master !== action.payload.master) {
      return state
    } else {
      let tempAxisMap = Object.assign({}, state.axisMap)
      tempAxisMap[action.payload.axis] =
        state.axisMap[action.payload.axis] + 1
      return { ...state, axisMap: tempAxisMap }
    }
  default:
    return state
```

Zdrojový kód 2.26: Zdieľanie prostredia - redux akcia

Server po rozpoznaní prijatej správy zo strany klienta s typom axisscaling vyšle akciu do redux store na aktualizáciu stavu a následne rozpošle nový stav všetkým pripojeným klientom.

```
    const axisScaling = (data) => {
      return {
        type: 'axisScaling',
        payload: {
          axisMap: data }}}
  if (parsedMessage.type === 'axis_scaling') {
    store.dispatch({
      type: 'AXIS_SCALING',
      payload: {
        master: ws.id,
        axis: parsedMessage.payload.axis}})
    wss.broadcast(
      JSON.stringify(axisScaling(store.getState().shared.axisMap)))}
```

Zdrojový kód 2.27: Zdieľanie prostredia

2.5.3 Integrácia s NDMSPC projektmi

Keďže väčšina projektov *NDMSPC* je nasadzovaná pomocou *Kubernetes* manažovacieho systému pre kontajnere, a *Docker* kontajnerami samotnými, bolo potrebné vytvoriť konfiguračný súbor pre vytvorenie obrazu servera, cez ktorý má komunikácia prechádzať. Nasledujúci fragment zdrojového kódu 2.28 predstavuje konfiguračný súbor pre vytvorenie docker obrazu, ktorý bol následne nasedený do kubernetes clustra pomocou *kind* nástroja.

```
FROM node:16
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8443
CMD [ "node", "index.js" ]
```

Zdrojový kód 2.28: Docker konfiguračný súbor

Tento prístup umožňuje jednoduchý manažment kontajnerov servera pomocou už spomínaného *Kubernetes* nástroja, vrámci ktorého je možné v konfiguračnom súbore nastaviť potrebný počet replík, ktoré majú byť k dispozícii pri nasadení systému. Tento konfiguračnž súbor umožní, že škálovanie systému je jedno-

duché a v spojení s implementovaným manažmentom miestností pre používateľov by nemalo dojsť k prehlteniu komunikácie bežiacej na sieti.

Rovnako podporuje aj zašifrovanú komunikáciu, ktorú je možné nastaviť priamo v konfiguračnom súbore pre zabezpečenie situácie, v prípade vyžadovania, aby zdieľané dáta neunikli von zo systému. V najhoršom prípade, sa vytvorí nová bežiaca inštancia servera, na ktorú sa používatelia budú schopní pripojiť.

Pri release novej verzie v *React-NDMSPC* projekte dochádza k automatickému spusteniu *CI/CD* pipeline, ktorá zabezpečí vytvorenie nového tagu s aktualizovaným obrazom aktuálneho servera. Pre správnu funkcionalitu bolo potrebné upraviť konfiguračný súbor *.gitlab-ci.yml* pre vytvorenie novej verzie obrazu servera, čo zobrazuje zdrojový kód 2.29. Po úspešnom merge requeste sa nový obraz vytvorí a nasadí namiesto predchádzajúcej verzie.

```
build-docker-image:
    stage: deploy
    script:
        - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
        ↪ $CI_REGISTRY
        - docker pull $CI_REGISTRY_IMAGE:latest || true
        - cd server
        - docker build --cache-from $CI_REGISTRY_IMAGE:latest --tag
        ↪ $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA --tag
        ↪ $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME --tag
        ↪ $CI_REGISTRY_IMAGE:latest .
        - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
        - if [ "$CI_COMMIT_REF_NAME" != "main" ]; then docker push
        ↪ $CI_REGISTRY_IMAGE:latest; fi
    allow_failure: true
    only:
    - tags
```

Zdrojový kód 2.29: Gitlab CI konfigurácia

Zdieľaná funkcionalita *NDMVR Shared* (zdieľanie VR prostredia) bola nasadená do *Kubernetes* klustra ako *k8s operator* v projekte *NDMSPC-Operator* [1]. Používatelia si môžu operátora nainštalovať lokálne z oficiálneho *k8s* repozitára s názvom NDMSPC-Operator[2].

[1] https://gitlab.com/ndmspc/ndmspc-operator

[2] artifacthub.io/packages/olm/community-operators/ndmspc-operator

3 Vyhodnotenie

3.1 Metódy vyhodnotenia

Pre určenie a overenie kvality implementačnej časti tejto práce prešiel projekt viacerými fázami pre vyhodnotenie aktuálneho stavu projektu, či už vzájomnými konzultáciami s nadradenými práce alebo počas riešenia problémov, ktoré sa počas tejto fázy vyskytli. Z hľadiska overenia kvality implementácie boli do úvahy brané viaceré časti, ktoré sú popísané v podkapitolách nižšie.

3.1.1 Testovanie pri vývoji

Počas vypracovania hlavného cieľa - teda zabezpečenia zdieľaného prostredia pre projekt *NDMVR* boli vyskytnuté problémy, riešené následovne. Vyskytnutý problém bol konzultovaný s nadradeným práce, po získaných odporúčaniach boli vybrané ideálne z nich a pre dané riešenie následne implementované.

Pri vývoji bol projekt testovaný pre zopár pripojených klientov a teda nebolo možné zistiť chyby, ako sú zaťaženie siete alebo nezosynchronizovaný používatelia na scéne. Týmto problémom sa venuje časť. 3.1.2.

3.1.2 Celková spokojnosť používateľov

Napriek prvotným úvaham uskotočnenia testovania na reálnych používateľoch, či už na webe alebo vo VR, v čase písania tejto práce nedošlo k obšírnemu testovaniu implementovaného riešenia pre širšiu verejnosť. Z tohto dôvodu sme sa rozhdli časť o celkovej spokojnosti používateľov z vyhodnotenia tejto práce vynechať a zaoberať sa len metódami použítými pri implementácii a realizácii projektu. Avšak, keďže implementácia zodpovedá požiadavkám a je súčasťou projektu *NDMVR* testovanie je voľne dostupné pre všetkých zainteresovaných používateľov s odkazom na konkrétnu stránku projektu. Keďže projekt je stále v priebehu vývoja, táto implementácia sa môže s postupom času výsledne meniť a môžu byť doimplementované viaceré možné vylepšenia, ktoré popisuje časť 3.5.

3.1.3 Splnenie požiadaviek

Hlavným bodom vyhodnotenia tejto práce je skutočnosť, či implementácia splnila stanovené ciele a požiadavky popísané na začiatku tejto práce. Požadované časti implementácie

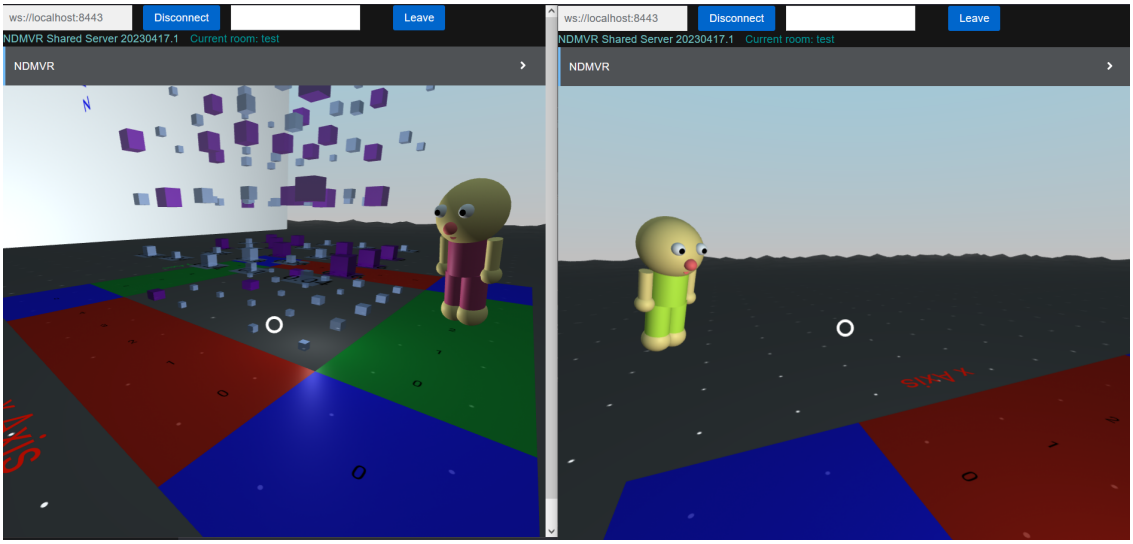
1. Vytvorenie miestností pre klientov - splnené v časti 3.2
2. Zdieľanie pozície a vzájomné vizualizácia entít na scéne - splnené v časti 3.2,
3. Zdieľanie zobrazovaných dát - splnené v časti 3.3,
4. Synchronizácia scény VR prostredia - splnené v časti 3.4

boli úspešne naimplementované a požiadavky na návrh spôsobu pre zdieľanú komunikáciu vo VR prostredí rámca *A-Frame* pre projekt *NDMVR* boli splnené.

Avšak, aj napriek úspešnej implementácii a realizícii riešenie existujú spôsoby ako vylepšiť už existujúci navrhnutý systém. Týmito vylepšeniami sa zaoberá kapitola 3.5, v ktorej boli identifikované určité nedostatky systému a možné riešenia pre zlepšenie celkovej komunikácie s cieľom predísť zahlteniu komunikačného kanála, ktorý bol vytvorený pomocou *Websocket* protokolu.

3.2 Vizualizácia klientov a manažment miestností

Na základe implementácie zdieľania používateľských entít v časti 2.11 a manažmentu miestností pre zdieľanie informácii v časti 2.4 je výsledok realizácie riešenia zobrazený na obrázku 3.1. Tento obrázok a ďalšie vo vyhodnocovacej časti tejto práce zobrazuje pripojenie dvoch klientov (ľavá strana - klient A a pravá strana - klient B).



Obr. 3.1: Zdieľanie používateľských entít

V hornej časti obrázka 3.1 je možné vidieť formulár pre pripojenie sa do miestnosti pre používateľov ako aj tlačidlo pre celkové pripojenie sa na server. Pravá a ľavá strana na obrázku 3.1 predstavuje dvoch pripojených klientov, ktorí zdieľajú prostredie vo VR, v tomto prípdate v miestnosti s názvom „test“. Entita avatarov popísaná v časti 2.3.3 je farebne odlíšená a umožňuje nechaotickú identifikáciu klientov na scéne.

3.3 Zdieľanie zobrazovaných dát histogramu

Implementácia zdieľania označených dát na scéne a celkových dát v histograme, ktorá bola popísaná v časti 2.4 je zobrazená na obrázku 3.2. Z obrázka je možné vidieť, že jeden z používateľov, konkrétne klient A na ľavo si priradil rolu *MASTER* (pomocou stlačenia tlačidla *M*, čo vo VR scéne predstavuje čiapočku implementovanú v časti 2.13.

Priradenie role *MASTER* automaticky spôsobilo, že dáta na strane klienta A boli vyzdieľané na server a druhý klient B (na pravo) tieto dáta prijal a histogram bol úspešne aktualizovaný. Zároveň priradenie role umožnilo prevádzanie akcií na scéne, ktorých výsledky sú rovnako zdieľané.

Obr. 3.2: Zdieľanie dát

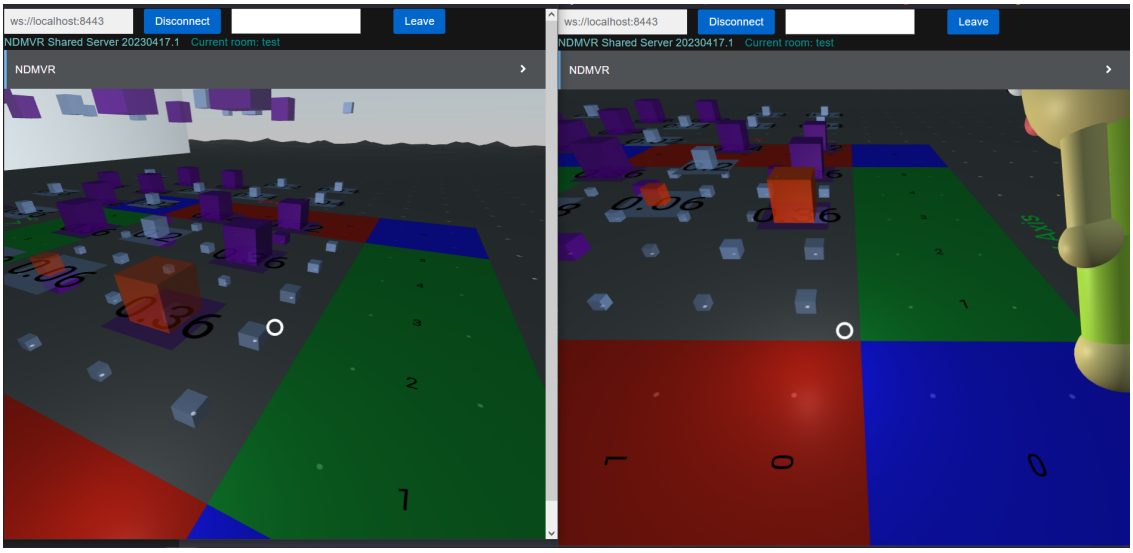
Zdieľanie označovania dát v histograme prebehlo úspešne, nakoľko z obrázka 3.2 je možné vidieť, že po kliknutí na niektorý z „binov“ prebehlo označenie oranžovou farbou a dáta boli vyzdieľané na server a prijaté klientom B. Z tohto faktu vyplýva, že bola splnená ďalšia požiadavke tejto práce a to zdieľanie dát vo VR scéne.

3.4 Synchronizácia scény VR prostredia

Synchronizácia dát a zdieľanie prostredia vo VR prostredí rámca *A-Frame*, ktorá bola implementovaná v časti 2.5.2 práce prebehla rovnako úspešne, keďže z obrázka 3.3 je možné vidieť, že po prevedení akcie posunu osi X o 1 na strane klienta A bol úspešne zosynchronizovaný aj klient B na pravej strane.

Posun osí, ako bolo spomenuté v implementačnej časti 2.5.2 bol prevedený pomocou vykonanie rovnakej akcie ako u klienta s rolou *MASTER*. Teda, došlo k stlačeniu „virtuálneho“ tlačidla na strane klienta B s rovnakým typom a posun bol vo VR prostredí zaregistrovaný.

Hlavným faktorom pri vyhodnotení bola synchronizácia klientov, ktorý sa pripojili až po niekoľkých akciách prevedených klientom s rolou *MASTER*. Po pripojení sa na server, klient bez role príjme dáta s aktuálnym stavom posunov osí histogramu a akcie sa vykonajú za radom podľa počtu prevedených akcii na strane klienta s *MASTER* rolou. Takýmto štýlom bolo zabezpečené, že používateľ, ktorý sa do miestnosti prihlási neskôr je taktiež zosynchronizovaný s aktuálnym zdieľaným stavom prostredia.



Obr. 3.3: Zdieľanie prostredia

3.5 Vylepšenia do budúcnosti

Ako bolo viackrát spomenuté, projekt *NDMVR* zo skupiny projektov *NDMSPC* je neustále v priebehu vývoja, preto je možné pozerať do budúcnosti tohto projektu a zadefinovať možné vylepšenie k už existujúcemu implementovanému riešeniu. Medzi tieto vylepšienia patria:

1. **Minimalizácia množstva prenášaných dát** - čím menej dát sa prenáša, tým rýchlejšia bude komunikácia. Preto je vhodné optimalizovať dáta pre komunikáciu na minimum. Môžete použiť napríklad binárne kódovanie alebo kompresiu dát.

2. **Prispôsobenie veľkosti fragmentov dát** - *WebSocket* umožňuje prenos dát v častiach (fragmentoch). Ak je veľkosť fragmentov prispôsobená veľkosti dát, ktoré sa prenášajú, môže to výrazne zlepšiť odozvu.

3. **Optimalizácia architektúry servera** - vytvorenie efektívneho servera pre prenos dát cez WebSocket. To zahŕňa použitie vysoko výkonného servera s nízkou prodlevou, ktorý umožňuje prenášať a spracovávať dáta rýchlejšie.

4. **Využitie asynchrónnych operácií** - *WebSocket* poskytuje asynchrónne API, čo umožňuje vykonávať operácie nezávisle na sebe, bez toho, aby blokovali vlákno. To umožňuje prenášať a spracovávať dáta rýchlejšie.

5. **Celkový stav histogramu** - Narozdiel od zdieľania jednotlivých dát napr. označené „biny“, dáta histogramu, pohyb osí je možné vytvoriť dátovú štruk-

túru, ktorá obsahuje všetky informácie o zdieľanom histograme. Týmto by sa predišlo tvorbe nespočetného množstva funkcii a zdieľal by sa len jeden stav, avšak preťaženie komunikácie by bolo o niečo väčšie, nakoľko za každým razom aj pri malej zmene musíme odosielať celkový stav histogramu.

6. **Animácie pre pohyb klientskych entít** - Rámec *A-Frame* poskytuje možnosť vytvorenia animácii, ktoré by zjemnili pohyb klientov po scéne, keďže v aktuálnej implementácii sa dáta odosielajú po zmene súradníc alebo každých 100ms, čo môže pre používateľov pôsobiť „sekavo“.

4 Záver

Implementácia zdieľania entít vo VR scéne pomocou *WebSocket* protokolu pre rámec *A-Frame* a projekt *NDMVR* bola úspešná. Tento spôsob komunikácie umožňuje rýchle a efektívne zdieľanie informácií medzi viacerými používateľmi a zvyšuje interakciu a spoluprácu medzi nimi v prostredí VR.

Zdieľanie používateľských entít, dát a synchronizícia VR scény bola prevedená pomocou miestností, vrámci ktorých je možné odčleniť aktuálne zdieľané dáta a tak zabezpečiť väčšie množstvo pripojených používateľov, ktorí môžu dáta zdieľať. Tento princíp sa osvedčil, nakoľko každá miestnosť pre zdieľanie dát predstavuje samostatný izolovaný stav VR scény.

V práci sme identifikovali oblasti, v ktorých by bolo možné zlepšiť komunikáciu cez *WebSocket* protokol. Navrhli sme niekoľko postupov na zlepšenie odozvy, ako je minimalizácia množstva prenášaných dát, prispôsobenie veľkosti fragmentov dát, využitie asynchrónnych operácií, zvýšenie šírky pásma a optimalizácia architektúry servera. Tieto postupy by mohli zvýšiť efektivitu komunikácie a vylepšiť užívateľský zážitok.

Celkovo sme dosiahli cieľ práce a ukázali sme, že použitie *WebSocket* protokolu pre zdieľanie entít vo VR scéne je efektívny spôsob interakcie medzi používateľmi. Budúce výskumy by sa mohli zamerať na ďalšie vylepšenia v oblasti komunikácie cez *WebSocket* protokol pre rámec A-Frame.

Literatúra

1. BURDEA, Grigore C; COIFFET, Philippe. *Virtual reality technology*. John Wiley & Sons, 2003.
2. ZHENG, JM; CHAN, KW; GIBSON, Ian. Virtual reality. *Ieee Potentials*. 1998, roč. 17, č. 2, s. 20–23.
3. SHERMAN, William R; CRAIG, Alan B. Understanding virtual reality. *San Francisco, CA: Morgan Kauffman*. 2003.
4. BROOKS, Frederick P. What's real about virtual reality? *IEEE Computer graphics and applications*. 1999, roč. 19, č. 6, s. 16–27.
5. PAPE, Dave. A hardware-independent virtual reality development system. *IEEE Computer Graphics and Applications*. 1996, roč. 16, č. 4, s. 44–47.
6. BERG, Leif P; VANCE, Judy M. Industry use of virtual reality in product design and manufacturing: a survey. *Virtual reality*. 2017, roč. 21, č. 1, s. 1–17.
7. GARCIA-PALACIOS, Azucena; HOFFMAN, Hunter; CARLIN, Albert; FURNESS III, Thomas A; BOTELLA, Cristina. Virtual reality in the treatment of spider phobia: a controlled study. *Behaviour research and therapy*. 2002, roč. 40, č. 9, s. 983–993.
8. MAHRER, Nicole E; GOLD, Jeffrey I. The use of virtual reality for pain control: A review. *Current pain and headache reports*. 2009, roč. 13, č. 2, s. 100–109.
9. WILSON, Christopher J; SORANZO, Alessandro. The use of virtual reality in psychology: A case study in visual perception. *Computational and mathematical methods in medicine*. 2015, roč. 2015.
10. DONALEK, Ciro; DJORGOVSKI, S. G.; CIOC, Alex; WANG, Anwell; ZHANG, Jerry; LAWLER, Elizabeth; YEH, Stacy; MAHABAL, Ashish; GRAHAM, Matthew; DRAKE, Andrew; DAVIDOFF, Scott; NORRIS, Jeffrey S.; LONGO, Giuseppe. Immersive and collaborative data visualization using virtual reality

platforms. In: *2014 IEEE International Conference on Big Data (Big Data)*. 2014, s. 609–614. Dostupné z DOI: 10.1109/BigData.2014.7004282.

11. MARRIOTT, Kim; SCHREIBER, Falk; DWYER, Tim; KLEIN, Karsten; RICHE, Nathalie Henry; ITOH, Takayuki; STUERZLINGER, Wolfgang; THOMAS, Bruce H. *Immersive analytics*. Zv. 11190. Springer, 2018.
12. BUTCHER, Peter WS; RITSOS, Panagiotis D. Building immersive data visualizations for the web. In: *2017 international conference on cyberworlds (CW)*. IEEE, 2017, s. 142–145.
13. ENS, Barrett; BACH, Benjamin; CORDEIL, Maxime; ENGELKE, Ulrich; SERRANO, Marcos; WILLETT, Wesley; PROUZEAU, Arnaud; ANTHES, Christoph; BÜSCHEL, Wolfgang; DUNNE, Cody; DWYER, Tim; GRUBERT, Jens; HAGA, Jason H.; KIRSHENBAUM, Nurit; KOBAYASHI, Dylan; LIN, Tica; OLAOSEBIKAN, Monsurat; POINTECKER, Fabian; SAFFO, David; SAQUIB, Nazmus; SCHMALSTIEG, Dieter; SZAFIR, Danielle Albers; WHITLOCK, Matt; YANG, Yalong. Grand Challenges in Immersive Analytics. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Yokohama, Japan: Association for Computing Machinery, 2021. CHI '21. ISBN 9781450380966. Dostupné z DOI: 10.1145/3411764.3446866.
14. FEDOSEJEV, Artemij. *React. js essentials*. Packt Publishing Ltd, 2015.
15. KOREČKO, Š; VAL'A, M; FEKETE, M. VISUALIZATION OF EXPERIMENTAL DATA IN WEB-BASED VIRTUAL REALITY. 2021.
16. MATIS, Dominik. *Riadenie získavania experimentálnych údajov na účely vizualizácie*. 2022. Dipl. pr. Technická unierzita v Košiciach. diplomová práca.
17. SREDOJEV, Branislav; SAMARDZIJA, Dragan; POSARAC, Dragan. WebRTC technology overview and signaling solution design and implementation. In: *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*. IEEE, 2015, s. 1006–1009.
18. GACKENHEIMER, Cory; PAUL, Akshat. *Introduction to React*. Zv. 52. Springer, 2015.
19. FETTE, Ian; MELNIKOV, Alexey. *The websocket protocol*. 2011. Tech. spr.
20. LOMBARDI, Andrew. *WebSocket: lightweight client-server communications*. " O'Reilly Media, Inc.", 2015.
21. FURUKAWA, Y. Web-based control application using WebSocket. *ICALEPCS2011*. 2011, s. 673–675.

22. RAI, Rohit. *Socket. IO Real-time Web Application Development*. Packt Publishing Ltd, 2013.

23. KUDIABOR, Dominic Travis. State management with React-Redux. 2020.

Zoznam príloh

A Systémová príručka

Táto príručka predstavuje prehľad konkrétnych „payload“ a „response“ objektov, ktoré *NDMVR Shared Server* očakáva resp. odosiela pre správnu funkcionalitu celého projektu.

Typy prijatých dát

Odosielanie dát na server prebieha pomocou funckie *socket.send*().

Používateľské miestnosti

- **Vytvorenie miestnosti**

```
{ type: 'createRoom', payload: roomName }
```

  payload : string

- **Pripojenie sa do miestnosti**

```
{ type: 'joinRoom', payload: roomName }
```

  payload : string

- **Opustenie miestnosti**

```
{ type: 'leaveRoom', payload: roomName }
```

  payload : string

- **Zmazanie miestnosti**

```
{ type: 'deleteRoom', payload: roomName }
```

  payload : string

Manažment používateľov

- **Pohyb používateľov po scéne**

```
{
    type: 'move',
    payload: {
        position: absolutePosition.toArray(),
        rotation: rotationQuaternion.toArray()
    }
}
```

payload : objekt s atribútmi position a rotation,

position : pole 3 súradníc X,Y,Z

rotation : pole 4 Eulerových parametrov rotácie - e0, e1, e2 a e3

- **Priradenie MASTER role**

```
{
    type: 'assign_master',
    payload: {}
}
```

payload : prázdny objekt

- **Odobratie MASTER role**

```
{
    type: 'deassign_master',
    payload: {}
}
```

payload : prázdny objekt

Zdieľanie dát

- **Zdieľanie označených binov**

```
{
    type: 'selected_bins',
    payload: [ { id: 1 } ]
}
```

payload : pole objektov s atribútom id,

id : integer

- **Zdieľanie dát histogramu**

```
{
    type: 'histogram',
    payload: { JSON.stringify(toJSON(histogram)) }
}
```

payload : JSON string z toJSON funkcie knižnice jsRoot

- **Zdieľanie iných dát**

```
{
    type: 'otherData',
    payload: { otherData: {} }
}
```

payload : objekt s atribútom otherData,

otherData: akýkoľvek objekt

Zdieľanie prostredia

- **Zdieľanie osí**

```
{
    type: 'axis_scaling_latest',
    payload: { axis: latestAxisChange }
}
```

payload : objekt s atribútom axis,

axis: string X_UP, X_DOWN, Y_UP, Y_DOWN, Z_UP, Z_DOWN

Typy odoslaných dát

Server odosiela dáta všetkým používateľom vrámci jednej miestnosti pomocou funkcie *broadcast*(). V projekte *NDMVR* je možné tieto dáta odchytávať pomocou *useStreamBrokerIn* funckie, kde parameter je typ odoslaných dát zo servera, ktoré je potrebné odchytiť.

Informácie o sockete

Typ správy *ws* odošle používateľovi identifikátor socketu generovaný priamo na serveri.

```
{
    type: 'ws',
    payload: { id: ws.id }
}
```

Typ správy *app* odošle používateľovi základné informácie o servere a aktuálnu verziu z *package.json* súboru.

```
{
    type: 'app',
    payload: {
        version: packageJson.version,
        name: 'NDMVR Shared Server'
    }
}
```

Odosielanie dát

Tabuľka A.1 zobrazuje typ odoslaných dát ako string (napr. "move") a typ dát, ktoré sa odošlú používateľom vrámci jednej izby pomocou funkcie broadcast().

Type (string)	**Payload**
move	{ clients: [id, position, rotation, color] }
master	{ master: id }
selectedBins	{ selectedBins: [{id: 1}, {id: 2}, ...] }
axisScaling	{ axisMap: mapa }
axisScalingLatest	{ latestAxis: string }
histogram	{ histogram: string }
otherData	{ otherData: objekt }

Tabuľka A.1: Typ odoslaných dát a samotné dáta

B Používateľská príručka

Táto príručka predstavuje pomôcku pre *NDMVR Shared Server* projekt. Obsahuje informácie o tom, ako projekt spojazdniť a využívať so všetkými jeho vlastnosťami.

Spustenie projektu

Existuje mnoho spôsobov ako projekt *NDMVR* a *NDMVR Shared Server* spojazdniť, avšak využívajú sa hlavne nižšie uvedené možnosti.

Stiahnutie z repozitára

Pre lokálne spustenie je potrebné stiahnuť aktuálnu verziu projektu. Buď ako package do projektu z npm registra[1] pomocou príkazu:

```
npm install @ndmspc/ndmvr
```

alebo pomocou git clone príkazu z GitLab stránky projektu:

```
git clone https://gitlab.com/ndmspc/ndmvr
```

Pri tomto postupe je potrebné nainštalovať závislosti projektu príkazom:

```
npm install
```

a následne aj v server adresári:

```
cd server
npm install
node .
```

[1] https://www.npmjs.com/package/@ndmspc/ndmvr

To zaručí, že rovnako ako aj klient, tak aj server budú mať potrebné závislosti. Posledný príkaz „node .“ spôsobí spustenie servera, ktorý bude čakať na požiadavky z klientskej aplikácie.

Klientsku časť aplikácie *NDMVR* je možné spustiť pomocou príkazu

```
npm run dev
```

v najvyššom adresári projektu. To spôsobí spustenie príkladu v prehliadači, ktorý obsahuje aj príklad pre zdieľanú scénu pomocou *NDMVR Shared Server*, na ktorý sa stačí pripojiť pomocou formulára.

V prípade, že je potrebné využiť komponent na klientskej časti systému *NDMVR* jediné, čo je potrebné pri vytváraní scény, je nastaviť useShared na hodnotu true a aktuálny socketId a masterId. Tým sa zabezpečí render celého komponentu *NdmVrShared*, ktorý komunikuje so serverom.

```
return (
    ...
    <NdmVrScene
      useShared={true}
      masterId={masterId}
      socketId={socketId}
    ...
)
```

NDMSPC Operator

Projekt *NDMVR* je taktiež možné spustiť pomocou *NDMSPC Operator*[2], ktorý zabezpečí spustenie a manažment aplikácie v *Kubernetes* klustroch.

Prerekvizita pre zvolenie tohto prístupu je nainštalovanie *Docker* nástroja, *kubectl* nástroja a *kind* nástroja. Výhodou tohto prístupu je, že server, na ktorom bude *NdmVrShared* je konfigurovateľný a testovateľný na jemu priradenej IP adrese.

[2] `https://operatorhub.io/operator/ndmspc-operator`