

**Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky**

**Komunikácia, architektúra a ďalšie aspekty
systému pre neurorehabilitáciu vo virtuálnej
realite**

Diplomová práca

**Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky**

**Komunikácia, architektúra a ďalšie aspekty
systému pre neurorehabilitáciu vo virtuálnej
realite**

Diplomová práca

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

Košice 2025

Bc. Damian Chmura

Abstrakt v SJ

L-NeRVEn je kolaboratívny systém pre motorickú neurorehabilitáciu založený na virtuálnej realite. Spája spoločné virtuálne prostredie pre interakciu terapeuta a pacienta s integrovanou EEG/BCI funkcionalitou používanou pri rehabilitačných cvičeniach. Vývoj systému prebieha v spolupráci so Slovenskou akadémiou vied (SAV), využíva platformu Unity 2022 a tiež sa aktívne používa pri výskumných rehabilitačných sedeniach s pacientmi.

Táto práca sa zaoberá technickým dlhom nahromadeným v prvej verzii systému L-NeRVEn, ktorý znižoval prevádzkovú spoľahlivosť a blokoval prípravu nadväzujúcej verzie. Namiesto úplného refaktoringu prechodného systému sa sústreďuje na zásahy, ktoré stabilizujú existujúcu verziu pre pokračujúce sedenia SAV a zároveň prinášajú znovupoužiteľné komponenty a prevádzkové postupy pre verziu V2.

Hlavnými výstupmi práce sú prepracovaná WebSocketová komunikačná vrstva a systém logovania aplikácie, súbor prevádzkových a používateľských zlepšení v EEG/BCI integrácii, nasadzovaní systému a priebehu sedení. Tieto zmeny boli implementované počas vývoja a následne overované pri debugovaní a aj pri sedeniach SAV, kde prispeli k stabilnejšej prevádzke a vytvorili prenositeľný základ pre verziu V2.

Kľúčové slová v SJ

virtuálna realita, neurorehabilitácia, rozhranie mozog-počítač, WebSocket, softvérová architektúra, Unity, Meta Quest

Abstrakt v AJ

L-NeRVEn is a collaborative virtual-reality-based system for motor neurorehabilitation. It combines a shared virtual environment for therapist-patient interaction with integrated EEG/BCI functionality used during rehabilitation exercises. The system is being developed in cooperation with the Slovak Academy of Sciences (SAV), uses Unity 2022, and is actively used in research rehabilitation sessions with patients.

This thesis addresses the technical debt accumulated in the initial version of L-NeRVEn, which reduced operational reliability and blocked preparation of a subsequent version. Rather than attempting a full refactor of the transitional system, the work focuses on changes that stabilise the existing version for ongoing

SAV sessions while providing reusable components and operational practices for V2.

The main outputs are a redesigned WebSocket communication layer and application logging system, together with a set of operational and usability improvements in EEG/BCI integration, system deployment, and session workflow. These changes were implemented during development and subsequently validated in debugging and SAV sessions, where they contributed to more stable operation and established a portable basis for V2.

Kľúčové slová v AJ

virtual reality, neurorehabilitation, brain-computer interface, WebSocket, software architecture, Unity, Meta Quest

Bibliografická citácia

CHMURA, Damian. *Komunikácia, architektúra a ďalšie aspekty systému pre neurorehabilitáciu vo virtuálnej realite*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2025. 82s. Vedúci práce: Ing. Štefan Korečko PhD.

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

**ZADANIE
DIPLOMOVEJ PRÁCE**

Študijný odbor: **Informatika**

Študijný program: **Informatika**

Názov práce:

**Komunikácia, architektúra a ďalšie aspekty systému pre neurorehabilitáciu vo
virtuálnej realite**

**Communication, Architecture, And Other Aspects of Virtual Reality
Neurorehabilitation System**

Študent: **Bc. Damian Chmura**

Školiteľ: **doc. 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 systému L-NeRVEn pre kolaboratívnu neurorehabilitáciu vo virtuálnej realite (VR) a podobnými riešeniami.
2. Identifikovať problémy a potrebné rozšírenia systému, najmä v oblasti údajovej komunikácie a zaznamenávania činnosti systému.
3. Na základe analýzy existujúcich riešení navrhnúť a implementovať riešenia vybraných problémov v systéme L-NeRVEn.
4. Navrhnúť inovácie architektúry systému s dôrazom na ľahké prispôsobenie novým scenárom rehabilitácie a dlhodobú udržateľnosť.
5. Návrh a implementáciu koordinovať s ďalšími riešiteľmi prác na systéme L-NeRVEn.
6. Implementované riešenie overiť testovaním na relevantnom VR hardvéri a vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: **slovenský**

Termín pre odovzdanie práce: **24.04.2026**

Dátum zadania diplomovej práce: **31.10.2025**



N. Z. Puklerová
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry, ako aj s využitím nástrojov generatívnej umelej inteligencie v súlade s Metodickým pokynom o záverečných a kvalifikačných prácach a Etickým kódexom študenta Technickej univerzity v Košiciach. Umelá inteligencia bola použitá ako podporný nástroj bez porušenia zásad akademickej etiky a autorskej integrity.

Podakovanie

Rád by som poďakoval svojmu vedúcemu práce, Ing. Štefanovi Korečkovi PhD., za odborné vedenie a cenné rady počas riešenia tejto práce. Ďakujem tiež kolegom z Technickej Univerzity v Košiciach a Slovenskej akadémie vied za spoluprácu na tomto projekte. V neposlednom rade ďakujem svojej rodine za ich trpezlivosť a podporu počas celého štúdia.

Obsah

1	Úvod	1
1.1	Systém L-NeRVEn	1
1.2	Vymedzenie problému	2
1.3	Ciele práce a hlavné príspevky	3
1.3.1	Riešené problémy	3
1.4	Kontext spolupráce	4
1.5	Štruktúra práce	4
2	Analýza systému L-NeRVEn a kontextualizácia práce	5
2.1	Virtuálna realita v neurorehabilitácii	5
2.2	Integrácia BCI a virtuálnej reality	6
2.2.1	Porovnanie vybraných VR-BCI systémov	6
2.3	Prehľad systému L-NeRVEn vo verzii „V1“	7
2.4	Identifikované problémy	8
2.4.1	Komunikácia a integrácia BCI	8
2.4.2	Architektúra a udržiavateľnosť	11
2.4.3	Použiteľnosť	12
2.4.4	Nasadzovanie a prevádzka	13
2.4.5	Stabilita servera	13
2.5	Architektonické rozhodnutia odvodené z identifikovaných problémov	14
2.6	Ovodené požiadavky	15
2.6.1	Funkčné požiadavky	15
2.6.2	Nefunkčné požiadavky	15
2.7	Rozhodnutie o rozsahu práce	16
3	WebSocketová komunikačná vrstva	18
3.1	Analýza pôvodnej implementácie	18
3.1.1	Architektúra a použité technológie	18

3.2	Analýza chyby náhodného odpájania	20
3.2.1	Úvodná izolácia problému	20
3.2.2	Vlastný Python klient na reprodukciu chyby	20
3.2.3	Testovanie s OpenViBE a Wiresharkom	20
3.2.4	Príčina zlyhania a oprava	21
3.3	Nedostatky pôvodnej implementácie	21
3.3.1	God object a nízka udržiavateľnosť	21
3.3.2	Thread-safety a práca s Unity aplikačným programovacím rozhraním (API)	22
3.3.3	Krehký protokol správ	22
3.3.4	Riziko zahltenia frontu a latencie	23
3.4	Návrh novej komunikačnej vrstvy	23
3.4.1	Požiadavky na nové riešenie	23
3.4.2	Hexagonálna architektúra	23
3.4.3	Návrh formátu správ	25
3.4.4	Pravidlá životného cyklu spojenia	25
3.4.5	Zoznam udalostí protokolu	26
3.4.6	Návrh súbežnosti a riadenia prevádzky	27
3.5	Implementácia refaktORIZOVANEJ architektúry	28
3.5.1	Transportná vrstva: <code>FleckNetworkTransport</code>	28
3.5.2	Výstupný port: <code>IOutboundMessenger</code>	28
3.5.3	Kompozičný koreň: <code>WebSocketManager</code>	29
3.5.4	Smerovanie správ: <code>MessageRouter</code>	30
3.5.5	Dvojfázová deserializácia a <code>TypedMessageHandler</code>	30
3.5.6	Typová bezpečnosť protokolu	31
3.5.7	Kompilačná hranica a interoperabilita s legacy kódom	31
3.5.8	Katalóg implementovaných handlerov	32
3.5.9	Spracovanie chýb a ochrana zdrojov	32
3.6	Výsledok a prenositeľnosť do „V2“	33
4	Systém logovania	34
4.1	Analýza existujúceho loggeru	34
4.1.1	Architektúra a väzba na životný cyklus	34
4.1.2	Chyby v perzistencii a logike zápisu	35
4.1.3	Synchronizácia a bezpečnosť vlákien	37
4.1.4	Nedostatky v observability	37
4.1.5	Dopad na diagnostiku WebSocketovej chyby	38
4.2	Návrh	38

4.2.1	Existujúce logovacie frameworky a zdôvodnenie vlastného riešenia	39
4.2.2	Požiadavky na nové riešenie	40
4.2.3	Architektonické rozhodnutia	40
4.2.4	Návrh prenositeľnosti do „V2“	41
4.3	Implementácia	42
4.3.1	Rozhranie ILogger a LogChannel	42
4.3.2	Fasáda GameLog	43
4.3.3	FileLogger: súborový zápis a flush stratégia	44
4.3.4	UnityLogger: konzolový výstup	45
4.3.5	CompositeLogger: kombinácia výstupov	45
4.3.6	Integrácia naprieč aplikáciou	46
4.4	Výsledok a prenositeľnosť do „V2“	47
4.4.1	Validácia na cieľovej platforme	47
4.4.2	Prevádzkové zlepšenia	48
5	Používateľské vylepšenia systému	50
5.1	EEG/BCI integrácia	50
5.1.1	Od jednosmernej komunikácie k obojsmernému protokolu	50
5.1.2	Zjednotenie terminológie motorickej imaginácie	51
5.1.3	Dynamické nastavenie threshold hodnoty	51
5.1.4	Používateľské rozhranie pre threshold hodnotu	52
5.1.5	Fatigue meter	52
5.1.6	Príkaz move v novom protokole	52
5.1.7	Synchronizácia stavu pre novoprihlásených klientov (Mirror SyncVar)	53
5.1.8	Výsledok oblasti EEG/BCI integrácie	54
5.2	Systém avatarov a inverznej kinematiky	55
5.2.1	Analýza chyby: kamera desktopového klienta pod úrovňou zeme	55
5.2.2	Prepracovanie škálovania avatara	55
5.2.3	Prepracovanie umiestňovania ruky pacienta	56
5.2.4	Oprava synchronizácie polohy avatara vo VR	56
5.2.5	Aktualizácia FinalIK z verzie 2.3 na 2.4	56
5.2.6	Ďalšie opravy avatarov	56
5.2.7	Výsledok oblasti avatarov a inverznej kinematiky (IK)	57
5.3	Stavový automat cvičenia	57
5.3.1	Pridanie stavu Prepare	57

5.3.2	Opravy prechodov medzi stavmi	57
5.3.3	Automatický režim cvičenia ako otvorený problém	57
5.4	Ďalšie používateľské a platformové zlepšenia	58
5.4.1	Platformové a build úpravy	58
5.4.2	Interakcia a používateľský zážitok (UX)	58
5.4.3	Vizuálne a lokalizačné úpravy	59
5.5	Zhrnutie kapitoly	59
6	Nasadzovanie a release stratégia	60
6.1	Stratégia release kanálov	60
6.1.1	Návrhové východiská	60
6.1.2	Definície kanálov	61
6.1.3	Schéma verziovania	61
6.1.4	Politika build promotion	62
6.1.5	Vizuálna identita	62
6.2	Perzistencia balíkového a produktového mena	63
6.3	Nasadzovací proces cez Meta Quest Developer Hub	64
6.4	Prevádzkové poznatky z platformy Meta Quest	65
6.4.1	24-hodinový lockout účtu	65
6.4.2	Cooldown po zmazaní aplikácie	65
6.4.3	SMS verifikácia pri každom uploade	65
6.4.4	Väzba na Android keystore	66
6.5	Výsledok kapitoly	67
7	Hodnotenie	68
7.1	Metodika overovania	68
7.2	P1 — WebSocketová vrstva	69
7.2.1	Odstránenie odpájania	69
7.2.2	Architektonická kvalita	69
7.2.3	Prenositelnosť do „V2“	70
7.3	P2 — Diagnostikovateľnosť a nasaditeľnosť	70
7.3.1	Logovací systém	70
7.3.2	Nasadzovacia a release stratégia	71
7.4	P3 — Používateľské úpravy	71
7.4.1	BCI integrácia: obojsmerný dátový tok	72
7.4.2	Synchronizácia stavu pre novoprihlásených klientov	72
7.4.3	Systém avatarov a inverznej kinematiky	73
7.4.4	Stavový automat cvičenia	73

7.4.5	Ďalšie používateľské vylepšenia	73
7.5	Splnenie odvodených požiadaviek	73
7.6	Diskusia k validite výsledkov	74
7.7	Obmedzenia a otvorené problémy	75
7.7.1	Problémy mimo rozsahu práce	75
7.7.2	Známe technické obmedzenia	75
7.7.3	Chýbajúce evaluačné dáta	76
8	Záver	77
8.1	Súhrn príspevkov	77
8.2	Strategický výsledok	78
8.3	Budúca práca	78
8.4	Záverečné poznámky	79
	Zoznam použitej literatúry	80
	Zoznam skratiek	83
	Slovník	84
	Zoznam príloh	88
A	Používateľská príručka	89
A.1	Inštalácia softvéru	89
A.1.1	Osobný počítač	89
A.1.2	Oculus Quest 2	89
A.2	Použitie a ukážka aplikácie	91
A.2.1	Kroky pre správne využitie teleportačného systému	92
A.2.2	Správa serverov	92
A.2.3	Hlasová komunikácia	93
A.3	Rola pacienta	94
A.4	Rola terapeuta	94
A.4.1	Nastavenie MI threshold hodnoty	95
A.4.2	Ukazovateľ únavy pacienta	97
A.4.3	Ukladanie nastavení pacienta	97
A.5	Ukážka systému	99
A.5.1	Vzhľad prostredia	99
A.5.2	Ukážka animácií	100

B	Systémová príručka	103
B.1	Importovanie projektu	103
B.2	Preklad aplikácie	104
B.2.1	Preklad pre server	104
B.2.2	Preklad pre stolový počítač	105
B.2.3	Preklad pre Oculus Quest 2	105
B.3	Nasadenie servera	107
B.3.1	Konfigurácia zoznamu serverov	107
B.4	Databáza	107
B.5	Nasadenie release buildov cez Meta Quest Developer Hub	108
B.5.1	Predpoklady	108
B.5.2	Identifikátory kanála	108
B.5.3	Nastavenia buildu v Unity	109
B.5.4	Pre-upload checklist	109
B.5.5	Upload cez Meta Quest Developer Hub	110
B.5.6	Verifikácia na zariadení	110
B.5.7	Build promotion (RC → STABLE)	110
B.5.8	Riešenie problémov	111
B.5.9	Správa keystore	111
B.6	REST API dokumentácia	111
B.7	Referenčná dokumentácia WebSocket protokolu	113
B.7.1	Pripojenie a životný cyklus	114
B.7.2	Formát správ	115
B.7.3	Relácie a autorizačné pravidlá	115
B.7.4	Katalóg udalostí	116
B.7.5	Spracovanie chýb	118
B.7.6	Limity a výkonnosť	118
B.7.7	Príklady správ	118
C	Implementačné detaily logovacieho systému	121
C.1	Formátovanie a publikovanie záznamu	121
C.2	Thread-safe zápis do súborového loggeru	122
C.3	Flush mechanizmus	122

Zoznam obrázkov

1.1	Prehľad hlavných komponentov systému L-NeRVEn a ich väzieb.	2
2.1	Zjednodušená architektúra systému L-NeRVEn „V1“ so zvýraznením dvoch komunikačných rovín: Mirror (VR relácia) a WebSocket (EEG/BCI integrácia).	8
3.1	Pôvodná architektúra WebSocketovej komunikácie	19
3.2	Hexagonálna architektúra WebSocketovej vrstvy s portami INetworkTransport a IOutboundMessenger	24
4.1	Architektúra pôvodného logovacieho systému	35
4.2	Kauzálny reťazec zlyhania perzistencie logov na Meta Quest	36
4.3	Tok zápisu, flushu a správy úložiska v triede FileLogger	49
5.1	Nastavenie MI threshold hodnoty v terapeutickom rozhraní	53
5.2	Ukazovateľ únavy pacienta v terapeutickom rozhraní	54
6.1	Trojkanálová release stratégia systému NeuroRehabVR s oddelenými app lines	61
6.2	Ikony a logá použité pre deployment varianty systému	63
A.1	Inštalácia ovládača Oculus Go	90
A.2	Úvodné menu	92
A.3	Správa serverov so zatvoreným panelom	93
A.4	Správa serverov s otvoreným panelom	93
A.5	Menu nastavení rehabilitácie	95
A.6	Menu s nastaveniami pozícií	95
A.7	Nastavenie MI threshold hodnoty v terapeutickom rozhraní	96
A.8	Ukazovateľ únavy pacienta v terapeutickom rozhraní	98
A.9	Tab Adjustments s uložením nastavení pacienta	98
A.10	Tab Training s uložením nastavení pacienta	99

A.11 Tlačidlo pre reset uloženej mierky avatara	99
A.12 Prostredie s pohľadom na stôl	100
A.13 Zvyšok miestnosti	100
A.14 Ukážka rôznych typov animácií	101
A.15 Dosah ruky pacienta	101
A.16 Animácia z pohľadu pacienta	102
B.1 Nastavenia XR Plug-in management pre server	105
B.2 Nastavenia na preklad pre Android	106
B.3 API dokumentácia - /training/move	112
B.4 API dokumentácia - /move	112
B.5 API dokumentácia - /rest	113
B.6 API dokumentácia - /spawn	113

Zoznam tabuliek

2.1	Porovnanie vybraných VR-BCI systémov — dokumentácia komunikačnej vrstvy je systematicky nedostatočná.	7
2.2	Obmedzenia pôvodného komunikačného protokolu medzi Unity a OpenViBE.	10
2.3	Tabuľka priorít problémových oblastí použitá pri určovaní rozsahu práce.	16
3.1	Porovnanie pôvodného a nového komunikačného protokolu	27
4.1	Prehľad identifikovaných nedostatkov pôvodného loggeru	38
4.2	Porovnanie kľúčových vlastností pôvodného a nového loggeru . . .	46
6.1	Definícia release kanálov systému NeuroRehabVR	61
6.2	Prevádzkové poznatky o platforme Meta Quest	66
7.1	Porovnanie diagnostickej kvality pôvodného a nového logovacieho systému.	71
7.2	Zavedené prevádzkové mechanizmy a spôsoby ich overenia.	72
B.1	Konfigurácia podľa release kanála	108
B.2	Prevádzkové limity WebSocket servera	118

Zoznam výpisov

3.1	Správa <code>client.checkin</code>	26
3.2	Odpoveď <code>client.checkin.ack</code>	26
3.3	Rozhranie <code>IOutboundMessenger</code>	29
3.4	Enum <code>ClientType</code> s mapovaním hodnôt protokolu	31
4.1	Rozhranie <code>ILogger</code> definujúce kontrakt logovacieho systému	42
4.2	Enum <code>LogChannel</code> s doménovo špecifickými kanálmi	43
4.3	Automatická inicializácia loggeru podľa typu buildu	43
4.4	Porovnanie pôvodného a nového spôsobu logovania	46
4.5	Príklad štruktúrovaného záznamu nového loggeru	47
B.1	Formát konfiguračného súboru <code>servers.json</code>	107
B.2	Štruktúra obálky WebSocket správy	115
B.3	Check-in — požiadavka	118
B.4	Check-in — potvrdenie	119
B.5	EEG/MI stream	119
B.6	Spustenie kroku cvičenia (prázdny payload)	119
B.7	Serverová notifikácia o začiatku odpočinku	119
B.8	Chybová odpoveď	120
C.1	Formátovanie a publikovanie záznamu vo fasáde <code>GameLog</code>	121
C.2	Thread-safe zápis do bufferu <code>FileLogger</code>	122
C.3	Bezpečný flush mechanizmus s ochranou prázdneho bufferu	123

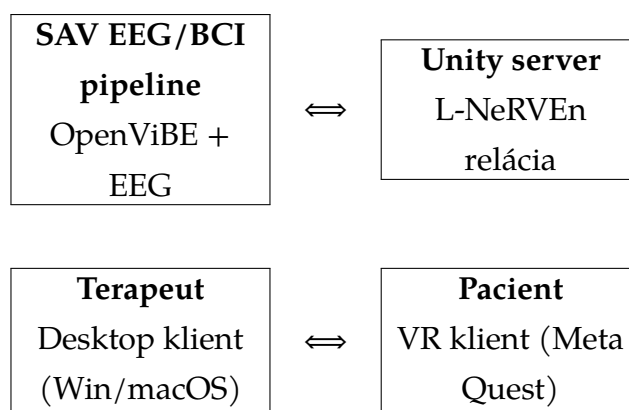
1 Úvod

Cievna mozgová príhoda patrí medzi hlavné príčiny dlhodobého motorického postihnutia a obmedzuje schopnosť pacienta zvládať bežné denné činnosti. Pri rehabilitácii horných končatín je najdôležitejšie dlhodobé opakovanie cvičení, motivácia a priebežná spätná väzba, či už od terapeutov, alebo od samotných pacientov. Virtuálna realita (VR) tento proces podporuje tým, že umožňuje opakovateľné cvičebné scenáre s merateľnými parametrami; aktuálne prehľady zároveň potvrdzujú jej klinický potenciál [1, 2].

V kombinácii s rozhraním mozog-počítač (BCI), konkrétne s paradigmou **Motor imagery**, možno do terapie aktívne zapojiť aj pacientov so závažnejšími limitáciami pohybu [3, 4]. Systém L-NeRVEn stavia na týchto princípoch: spája VR, kolaboratívnu reláciu pacient-terapeut a spätnú väzbu z elektroencefalografie (EEG) a BCI v reálnom čase. Podrobnejší odborný kontext VR a BCI je rozpracovaný v kapitole 2; úvod sa sústreďí na vymedzenie problému a výskumných cieľov.

1.1 Systém L-NeRVEn

L-NeRVEn je kolaboratívny systém virtuálnej reality vyvíjaný na Katedre počítačov a informatiky Technickej univerzity v Košiciach v spolupráci so Slovenskou akadémiou vied. Je navrhnutý tak, aby pacient a terapeut mohli súčasne zdieľať jedno virtuálne prostredie, pričom terapeut riadi priebeh cvičenia a pacient vykonáva úlohy založené na koncepte **Motor imagery**. Externý EEG/BCI pipeline je založený na OpenViBE, implementovaný Slovenskou akadémiou vied v Bratislave, ktorá systém aktívne využíva pri výskumných sedeniach. Unity aplikácia je postavená na Unity 2022 a podporuje obe role – pacienta aj terapeuta – v **buildoch** pre Meta Quest a aj ako desktop buildy pre Windows a macOS. Na desktope slúži rola pacienta primárne na testovacie účely; simulátor rozšírenej reality (XR) simuluje skutočné Quest zariadenie. Ďalej serverová inštancia zabezpečujúca spoločný priebeh relácie.



Obrázok 1.1: Prehľad hlavných komponentov systému L-NeRVEn a ich väzieb.

Z pohľadu členov projektu ide o systém, v ktorom sa stretávajú odlišné požiadavky. Vývojový tím TUKE rieši Unity architektúru a nasadenie aplikácie, Slovenská akadémia vied zabezpečuje EEG/BCI pipeline a ako hlavný používateľ udáva požiadavky, rola terapeuta systém obsluhuje počas sedení a pacienti predstavujú používateľov s výrazne obmedzenou motorikou. Systému L-NeRVEn bola venovaná aj bakalárska práca autora, v rámci ktorej bola implementovaná vrstva inverznej kinematiky pre avatara pacienta.

1.2 Vymedzenie problému

Prvotná verzia L-NeRVEn vznikla ako funkčný a použiteľný výskumný prototyp v rámci diplomovej práce. Vzhľadom na rozsah funkcionality a časové obmedzenia pôvodného vývoja bola priorita prirodzene zameraná najmä na overenie konceptu a praktickú použiteľnosť systému, nie na dlhodobé architektonické riešenia všetkých jeho častí. Po nasadení v spolupráci so SAV sa preto začali prejavovať obmedzenia typické pre výskumný prototyp, najmä v komunikačnej vrstve, diagnostikovateľnosti a oddelení doménovej logiky od infraštruktúry.

Počas ďalšej spolupráce so SAV boli opakovane hlásené výpadky komunikačnej vrstvy, ktoré narúšali výskumné relácie a znižovali dôveru v systém počas sedení s pacientmi. Údržba systému bola zároveň náročnejšia, keďže viaceré opravy zasahovali do centrálnych tried bez dostatočne jasných hraníc medzi jednotlivými zodpovednosťami. Tieto problémy komplikovali aj prípravu verzie „V2“, ktorá potrebovala vychádzať zo stabilnejších a lepšie prenositeľných riešení.

Systém bolo potrebné udržať v prevádzke pre pokračujúce sedenia SAV, no refaktorizácia celého prechodného systému nebola účelná, keďže paralelne vznikala jeho nástupnícka verzia. Práca preto vychádza z predpokladu, že najužitečnejšie sú zásahy, ktoré súčasne riešia prevádzkové problémy „V1“ a vytvárajú

prenositeľnú infraštruktúru pre „V2“. Podrobný opis identifikovaných problémov je uvedený v kapitole 2.

1.3 Ciele práce a hlavné príspevky

Hlavný zámer práce možno rozdeliť do dvoch navzájom prepojených cieľov. Prvým cieľom bola stabilizácia systému L-NeRVEn „V1“ pre pokračujúce výskumné sedenia realizované so Slovenskou akadémiou vied. To zahŕňalo odstránenie kritických problémov, hlavne v spoľahlivosti, zlepšenie integrácie medzi Unity aplikáciou a EEG pipeline a implementáciu takých klinických a používateľských vylepšení, ktoré priamo ovplyvnili priebeh reálnych sedení. Druhým cieľom bol návrh a implementácia infraštruktúrnych riešení používaných už vo verzii „V1“, ktoré boli zároveň navrhnuté tak, aby boli prenositeľné aj do verzie „V2“. Sem patrí najmä kompletný architektonický prepis WebSocketovej komunikačnej vrstvy, prepis a pilotná integrácia systému logovania a návrh nasadzovacej a release stratégie použiteľnej aj v ďalšom vývoji.

1.3.1 Riešené problémy

Na základe vymedzeného problému sú identifikované tri kľúčové problémy:

1. **P1:** Prepracovanie a rozšírenie WebSocketovej vrstvy medzi Unity a OpenViBE tak, aby boli odstránené zistené nedostatky a vrstva bola prenositeľná do verzie „V2“.
2. **P2:** Zabezpečenie diagnostikovateľnosti a stabilného nasadenia systému L-NeRVEn „V1“ v klinickom prostredí prostredníctvom architektonických a prevádzkových zásahov.
3. **P3:** Identifikácia a realizácia klinických a používateľských úprav s najvyšším praktickým prínosom pre prebiehajúce sedenia SAV pri rešpektovaní obmedzení prechodnej verzie „V1“.

Práca prináša tieto príspevky:

1. Nová WebSocketová komunikačná vrstva s rozšíreným obojsmerným dátovým tokom medzi Unity a OpenViBE.
2. Prepracovaný logovací systém zlepšujúci diagnostikovateľnosť v lokálnom aj vzdialenom nasadení.

3. Súbor klinických vylepšení v oblasti **BCI** integrácie, stavového automatu cvičenia, avatarov a používateľského rozhrania.
4. **Release** stratégia pre zariadenia Meta Quest vrátane prevádzkových pravidiel použiteľných aj vo „V2“.
5. Dokumentácia rozhodnutí o rozsahu zásahov do „V1“ a návrhový základ pre komponenty implementované vo verzii „V1“ a navrhnuté na prenos do „V2“.

Problémy mimo zvoleného rozsahu sú v práci identifikované, zdokumentované a odložené na ďalší vývoj, prípadne riešené spoluriešiteľmi.

1.4 Kontext spolupráce

V rámci spolupráce medzi TUKE a SAV bol autor tejto práce zodpovedný za stabilizáciu verzie „V1“ a za komponenty implementované pre túto verziu, ktoré boli zároveň navrhnuté s ohľadom na ich neskorší prenos do „V2“. Niektoré riešenia boli navrhnuté a špecifikované, zatiaľ čo ich plná implementácia ostáva súčasťou pokračujúceho vývoja verzie „V2“.

1.5 Štruktúra práce

Kapitola 2 analyzuje existujúci systém L-NeRVEn — identifikuje nedostatky v komunikácii, architektúre a praktickej použiteľnosti a zdôvodňuje rozsah vykonanej práce. Nasledujúce dve kapitoly sa venujú komponentom implementovaným vo verzii „V1“, ktorých architektúra zároveň umožňuje ich následný prenos do „V2“: kapitola 3 dokumentuje prepis WebSocketovej komunikačnej vrstvy od analýzy chyby a nedostatkov po výslednú implementáciu a kapitola 4 opisuje nový logovací systém. Klinické a používateľské vylepšenia vyžiadané SAV sú popísané v kapitole 5, nasadzovací proces a **release** stratégia v kapitole 6. Kapitola 7 hodnotí dosiahnuté výsledky a záver (kapitola 8) formuluje odporúčania pre ďalší vývoj.

2 Analýza systému L-NeRVEn a kontextualizácia práce

L-NeRVEn vo verzii „V1“ predstavoval funkčný, no v podstatných oblastiach krehký systém. Táto kapitola ho najprv zasadzuje do kontextu virtuálnej reality, neurorehabilitácie a integrácie rozhraní mozog-počítač, a následne analyzuje jeho technický stav, aby zdôvodnila výber cieľov vypracovaných v nasledujúcich kapitolách.

2.1 Virtuálna realita v neurorehabilitácii

Virtuálna realita (VR) umožňuje vytvárať kontrolované, opakovateľné a merateľné terapeutické scenáre s potenciálom zvýšiť imerziu a angažovanosť tréningu. Meta-review od Khan a kol. [1] potvrdzuje pozitívny vplyv VR na motorické schopnosti; Soleimani a kol. [2] dopĺňajú, že účinnosť závisí od konkrétneho návrhu systému, dĺžky tréningu a kvality implementácie. Samotná prítomnosť virtuálneho prostredia teda nestačí; o výsledku rozhoduje architektúra, kvalita spätnej väzby a vhodnosť interakcií.

Veľká časť publikovaných systémov napriek tomu zostáva na úrovni izolovaných prototypov zameraných na jedného používateľa a obmedzený scenár cvičenia; robustnej integrácii viacerých subsystémov sa venuje podstatne menej pozornosti.

Pre neurorehabilitáciu je relevantná aj kvalita virtuálnych avatarov, ktorá ovplyvňuje ilúziu vlastníctva tela (**body ownership**) [5] — tento aspekt sa v L-NeRVEn prejavil pri problémoch s modelmi Ready Player Me (sekcia 2.4.3).

L-NeRVEn sa od väčšiny experimentálnych **VR** rehabilitačných systémov odlišuje tým, že riešený problém nespočíva iba v terapeutickom scenári, ale v návrhu spoľahlivého systému, v ktorom je **VR** prepojená vrstvou rozhrania mozog-počítač (**BCI**) založenou na elektroencefalografii (**EEG**) a s terapeutickým ovládaním v reálnom čase.

2.2 Integrácia BCI a virtuálnej reality

Pre neurorehabilitáciu je zvlášť zaujímavá paradigma **motor imagery**, pri ktorej si pacient vedome predstavuje pohyb a EEG signál zachytí sprievodnú mozgovú aktivitu. Systém na základe toho poskytne vizuálnu spätnú väzbu a vytvorí uzavretú terapeutickú slučku — tá je veľmi dôležitá najmä pre pacientov so závažnými motorickými deficitmi, ktorí sa do terapie môžu zapojiť jedine prostredníctvom mozgovej aktivity.

Účinnosť tohto prístupu rastie v spojení s virtuálnou realitou. Leeb a Pérez-Marcos [3] ukazujú, že VR umožňuje prepojiť mozgovú aktivitu s virtuálnym avatarom a vytvoriť multisenzorický, motivačný priebeh terapie; Ma a kol. [4] ďalej potvrdzujú, že takáto rehabilitácia zlepšuje funkciu hornej končatiny a koreluje so zmenami kortikálnej aktivácie. V systéme L-NeRVEn zabezpečuje BCI vrstvu platforma OpenViBE — otvorený nástroj na spracovanie EEG signálu a konštrukciu experimentálnych scenárov [6]. V existujúcej výskumnej **pipeline** systému L-NeRVEn plní OpenViBE úlohu EEG/BCI vrstvy.

Unity aplikácia a EEG/BCI **pipeline** v OpenViBE spravidla bežia v odlišných **runtime** prostrediach, majú odlišné nároky na životný cyklus procesov a spracovanie chýb, a napriek tomu medzi nimi musí prúdiť časovo citlivý dátový tok. Používané riešenia siahajú od vlastných protokolov cez WebSockety po streamovacie rámce ako Lab Streaming Layer [7] — no literatúra venuje podstatne viac priestoru terapeutickým scenárom a signálovému spracovaniu než samotnej komunikačnej vrstve — tá býva zdokumentovaná nedostatočne, čo v praxi vedie ku krehkým, ťažko reprodukovateľným a udržiavateľným integráciám. V L-NeRVEn sa tento nedostatok prejavil konkrétne: komunikačná vrstva mala jednoducho definované aplikačné programovacie rozhranie (**API**), správy medzi **EEG** pipeline a **VR** aplikáciou mali jednoduchý formát ktorý síce bol v danej chvíli postačujúci, no nové požiadavky rýchlo odhalili jeho limity.

2.2.1 Porovnanie vybraných VR-BCI systémov

Tabuľka 2.1 sumarizuje vybrané VR-BCI systémy z hľadiska inžinierskeho zamerania. Posledný stĺpec ukazuje, do akej miery jednotlivé štúdie dokumentujú komunikačnú vrstvu medzi VR a EEG **pipeline**. Opakujúca sa absencia tejto dokumentácie potvrdzuje medzeru, ktorú táto práca adresuje.

Tabuľka 2.1: Porovnanie vybraných VR-BCI systémov — dokumentácia komunikačnej vrstvy je systematicky nedostatočná.

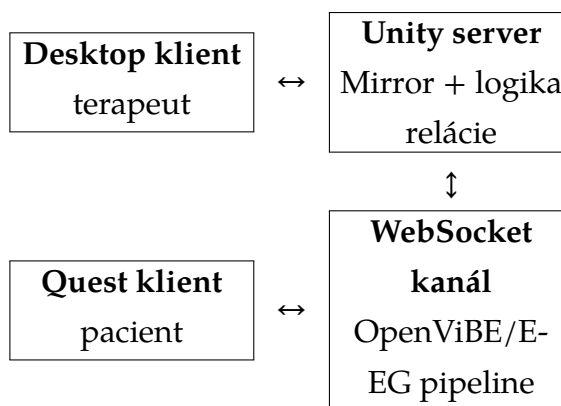
Systém / štúdia	VR platforma	BCI paradigma	Typ overenia	Komunikačná vrstva
NeuRow [8]	Imersívna VR	Motor imagery	Konferenčný prototyp	Integrácia popísaná na vysokej úrovni
VR neurofeedback pilot [9]	VR neurofeedback	Motor imagery	Pilotná klinická štúdia	Bez popisu transportnej vrstvy
Dual-task motorická imaginácia (MI) + VR [10]	VR + dual-task	Motor imagery	Randomizovaná pilotná štúdia	Fokus na klinický efekt
Immersive VR-BCI predictors [11]	Imersívna VR-BCI	Motor imagery	Klinická štúdia s EEG prediktormi	Dôraz na biomarkery, nie architektúru
L-NeRVEn [12] + táto práca	Kolaboratívna VR	Motor imagery (OpenViBE)	Výskumné a klinické sedenia SAV	Stabilný oboj-smerný protokol, diagnostikovateľnosť

2.3 Prehľad systému L-NeRVEn vo verzii „V1“

L-NeRVEn „V1“ je kolaboratívny neurorehabilitačný systém postavený na Unity 2022. Prepája viacero klientov a externý EEG/BCI pipeline do jedného terapeutického scenára. Architektúru systému možno zjednodušene opísať nasledovne:

- **Server build pre desktop:** autoritatívna inštancia scény a sieťovej logiky.
- **Desktop klient pre terapeuta:** ovládanie relácie, nastavení a dohľad nad priebehom cvičenia.
- **Quest klient pre pacienta:** samotné imerzívne VR prostredie používané počas cvičenia.

Hlavná Unity klient-server komunikácia je realizovaná knižnicou Mirror. Popri nej existuje samostatný WebSocketový kanál na porte 7778, určený pre integráciu OpenViBE a EEG nástrojov bežiacich na zariadeniach Slovenskej akadémie vied. Systém avatarov využíva modely Ready Player Me a middleware FinalIK vo verzii 2.3. Z pohľadu aplikačnej logiky je cvičenie riadené jednoduchým stavovým automatom so stavmi Pause, Rest a Move.



Obrázok 2.1: Zjednodušená architektúra systému L-NeRVEn „V1“ so zvýraznením dvoch komunikačných rovín: Mirror (VR relácia) a WebSocket (EEG/BCI integrácia).

Z pohľadu zainteresovaných strán ide o multiaktérový systém. Vývojový tím na TUKE KPI je zodpovedný za Unity časť aplikácie, Slovenská akadémia vied zabezpečuje EEG/BCI pipeline a výskumné nasadenie. Používatelia systému sú Terapeuti, ktorí systém obsluhujú počas sedení a pacienti, typicky po cievej mozgovej príhode, predstavujú koncových používateľov s výrazne obmedzenými motorickými schopnosťami. Chyba v komunikácii, nevhodné ovládanie alebo zdĺhavá inicializácia pacienta majú okamžitý dopad na reálne použitie; podobné vrstvy technického dlhu no ale aj hodnotné funkčné základy boli ďalej identifikované a zdokumentované.

2.4 Identifikované problémy

Nasledujúce problémy boli identifikované systematickým postupom v štyroch krokoch: (1) statická analýza zdrojových kódov a architektúry „V1“, (2) reprodukčný debugging hlásených nedostatkov, (3) korelácia s prevádzkovými logmi a pozorovaniami z testovacích behov a (4) validácia dopadov počas výskumných a klinických sedení realizovaných so SAV. Pre potreby tejto práce boli zistenia priebežne konsolidované do zoznamu problémov, ktorý slúžil ako podklad na prioritizáciu rozsahu v sekcii 2.7. Pri každom probléme je uvedený jeho klinický dôsledok.

2.4.1 Komunikácia a integrácia BCI

Hlavné problémy sa nachádzali v oblasti komunikácie medzi Unity serverom a externou EEG/BCI pipeline; opakovane sa vyskytovali náhodné odpájania Web-

Socket klienta používaného v OpenViBE. Pracovníkmi SAV bol tento problém hlásený počas viacerých mesiacov a priamo prerušoval výskumné sedenia. Navyše sa nedal spoľahlivo reprodukovat' na požiadanie, čím sa výrazne sťažovala jeho diagnostika. Jadro problému súviselo s KeepAlive timeoutom a blokovaním Python klienta počas výpočtu baseline (kapitola 3).

Ďalším problémom nebola absencia protokolu ako takého, ale jeho obmedzený rozsah medzi EEG nástrojom a Unity aplikáciou. Pôvodná komunikácia pracovala s jednoduchým tvarom správ typu {op, roomid}, ktorý bol pre vtedy definované potreby stabilný a postačujúci. S rastúcimi požiadavkami však začala chýbať jasne definovaná obálka a verzionovanie, pričom potvrdenie registrácie klienta bolo iba čiastočné. Ak sa pri parsovaní správ vo formáte **JavaScript Object Notation (JSON)** objavila chyba, mohla viesť až k `NullPointerException` a pádu spojenia. Dôsledkom bolo, že SAV nemala k dispozícii robustný a dobre zdokumentovaný integračný bod, voči ktorému by bolo možné dlhodobo vyvíjať.

Rozsah tohto obmedzenia najlepšie ilustruje analýza pôvodného protokolu, zdokumentovaného v spoločnej publikácii výskumného tímu [12]. Pôvodný protokol poznal iba dve klientské správy (checkin a move) a šesť serverových príkazov riadiacich prechody cvičenia. Tok klinických dát bol prakticky jednosmerný: Unity mohol oznámiť OpenViBE zmenu fázy cvičenia, OpenViBE mohol potvrdiť úspešnú **motor imagery**, no žiadna klinicky relevantná hodnota sa medzi oboma stranami nezdieľala. Tento jednosmerný tok dát obmedzoval hodnotu systému a možnosti použitia, ako je podrobnejšie analyzované v kapitole 3. Rozsah obmedzení pôvodného protokolu sumarizuje tabuľka 2.2.

Pôvodná architektúra komunikácie je zobrazená na obrázku 3.1 v kapitole 3.

Praktický dopad týchto obmedzení ilustruje zmena threshold hodnoty **Motor imagery**: v pôvodnom systéme musel terapeut zastaviť cvičenie, verbálne koordinovať zmenu s EEG operátorom, ktorý reštartoval OpenViBE — celý proces trval rádovo minúty a priamo narúšal priebeh terapeutického sedenia.

Systém neposkytoval ani synchronizáciu stavu pre novoprihlásených klientov. Informácie o aktuálnom stave cvičenia alebo stave EEG zariadení boli odovzdávané iba klientom, ktorí už boli pripojení v čase zmeny. Ak sa pacient alebo iný klient pripojil neskôr, nezískal automaticky úplný aktuálny stav relácie. Terapeut musel manuálne obnoviť kontext relácie a používateľ vnímal nekonzistentné správanie systému. Technickou príčinou boli stavové premenné relácie, ktoré nevyužívali mechanizmus `[SyncVar]` knižnice `Mirror`. `[SyncVar]` zabezpečuje automatickú synchronizáciu aktuálnych hodnôt pri pripojení nového klienta. Namiesto toho boli zmeny stavu propagované iba jednorazovými udalosťami, nový

Tabuľka 2.2: Obmedzenia pôvodného komunikačného protokolu medzi Unity a OpenViBE.

Aspekt	Pôvodný protokol [12]	Dôsledok / Obmedzenie
Klientské udalosti	checkin, move	Iba 2 správy; chýba možnosť streamovania klinických dát od klienta
Serverové udalosti	start/stopEegEvaluation, start/stopRestPeriod, unknown, failedCheckin	Bez podpory dynamickej konfigurácie počas sedenia
Smer klinických dát	Jednosmerný	Chýba obojsmerný tok klinických dát
Prahová hodnota MI	Nezdieľaná	Zmena vyžaduje manuálny zásah a reštart OpenViBE
EEG/MI dáta pre terapeuta	Nedostupné v Unity	Terapeut nemá prístup k EEG/MI hodnotám počas sedenia
Chybový protokol	Iba failedCheckin	Žiadny štruktúrovaný chybový protokol
Obálka správ	{op, roomid}	Chýba verzionovanie, časové značky a štruktúrovaný payload

klient nebol spätne informovaný o aktuálnom stave relácie.

Osobitný problém predstavovala chýbajúca automatizácia pripojenia a inicializácie pacienta. Jednoduchý postup predpokladal, že sa pacient zvládne postaviť na kalibráciu výšky, vybrať si rolu a zvoliť miestnosť. Pre pacientov po cievnej mozgovej príhode s obmedzenou motorikou bol tento predpoklad nereálny. V praxi to znamenalo zvýšenú záťaž pre rolu terapeuta, zdĺhavé spúšťanie sedenia a vyššiu pravdepodobnosť chýb pri manuálnom nastavovaní.

Navrhované riešenie počítalo s rozdelením Unity aplikácie na dva režimy: režim terapeuta a režim pacienta. V režime pacienta by aplikácia neposkytovala konfiguračné prvky. Po spustení by sa prihlásila na server a oznámila, že konkrétny headset s priradeným profilom je pripravený na pripojenie do relácie. Terapeut by vo svojom rozhraní vybral preset, dostupný headset/profil a cieľovú miestnosť. Po potvrdení by server odoslal vybranému headsetu príkaz na pripojenie do určenej miestnosti. Cieľom návrhu bolo odstrániť potrebu, aby pacient pri štarte aplikácie manuálne vyberal rolu, miestnosť alebo ďalšie parametre.

2.4.2 Architektúra a udržiavateľnosť

V technickom jadre systému sa nahromadilo viacero architektonických rozhodnutí, ktoré umožnili rýchly počiatkový vývoj za cenu technického dlhu. Najviditeľnejším problémom bol monolitický `WebSocketHandler` resp. súvisiaci manažér `WebSocket`ovej komunikácie, ktorý spájal transport, deserializáciu **JSON** správ, smerovanie, zásahy do hernej logiky a v niektorých častiach aj modifikáciu používateľského rozhrania. Takýto **god object** porušoval princíp jednej zodpovednosti a prakticky znemožňoval izolované testovanie. Každá úprava jedného typu správy tak niesla riziko poškodenia inej, zdanlivo nesúvisiacej funkcionality.

S tým úzko súviseli **thread-safety** problémy. Zdieľané dátové štruktúry, najmä slovník pripojení, používalo hlavné vlákno Unity aj vlákna knižnice `websocket-sharp` bez akejkoľvek synchronizácie, čím sa zaviedlo riziko nedefinovaného správania a nepredvídateľných `InvalidOperationException` pádov. V systéme, kde musí byť spracovanie udalostí stabilné, hlavne počas aktívneho cvičenia s reálnym pacientom, išlo o architektonické zlyhanie. No tiež treba podotknúť, že dátový tok vo verzii „V1“ bol pomerne nízky a teda je otázne, koľko problémov tento nedostatok v praxi spôsobil.

Ďalším problémom bolo používanie knižnice `websocket-sharp`, ktorá už nie je aktívne udržiavaná. Neponúka aktuálne bezpečnostné opravy a jej kompatibilita s novšími verziami Unity bola neistá. Takáto knižnica znamená, že aj bez okamžitej chyby v systéme existuje latentné riziko, že budúce aktualizácie Unity alebo zmeny v sieti môžu znefunkčniť komunikačnú vrstvu.

Významnou, aj keď na prvý pohľad menej nápadnou problémovou vrstvou bolo logovanie. Pôvodný logger trpel roztrúsenou inicializáciou, absenciou **dependency injection**, nedostatočne rozlíšenými úrovňami a nejasnou thread-safety pri zápise. Ku štrukturálnym nedostatkom sa pridával chýbajúci prístup k logom zo vzdialene nasadených zariadení. Diagnostika tým bola sťažená, ako ukázala skúsenosť popísaná v sekcii 2.4.1: absencia štruktúrovaného logovania veľmi predlžovala čas potrebný na nájdenie príčiny problému.

Problém sa netýkal iba komunikácie. Stavový automat cvičenia bol pevne zakódovaný do stavov `Pause`, `Rest` a `Move`, pričom manuálny a automatický režim mali takmer duplicitné definície. Architektúra nebola ľahko prenositeľná medzi scénami ani rehabilitačnými scenármi. Každé rozšírenie cvičebného scenára tak vyžadovalo hlbší zásah do existujúcej logiky namiesto jednoduchého konfiguračného rozšírenia.

Napokon, výrazným rizikom bola aj samotná stratégia spracovania správ. Pôvodná implementácia spracovávala iba jednu správu na volanie `Update()` a pou-

žívala neobmedzený front. Pri prípadnej vyššej frekvencii EEG udalostí alebo pri spustení vlastných Python skriptov v OpenViBE, ktoré na niekoľko sekúnd blokovali spracovanie ďalších rámcov vrátane spätnej „Pong“ odpovede, tak potenciálne dochádzalo k rastu latencie, spotreby pamäte a v krajnom prípade k ohrozeniu stability celej aplikácie.

2.4.3 Použitelnosť

V systéme sa vyskytovalo viacero chýb priamo ovplyvňujúcich použitelnosť. Kamera desktop klienta sa v jednej z verzií pri prvom spustení objavila pod úrovňou zeme. Príčinou bola absencia predvolenej hodnoty výšky a nesprávne inicializovaný `avatarOffset`. Desktop klient terapeuta bol tým po prvom spustení prakticky nepoužiteľný.

Na desktop klientovi sa prejavovalo aj nesprávne škálovanie avatara. Kým v režime rozšírenej reality (XR) sa škálovanie správalo relatívne korektne, desktop pohľad terapeuta bol nekonzistentný a líšil sa aj podľa typu použitého avatara. V repozitári bol tento problém evidovaný ako samostatný problém a jeho dôsledkom nebol iba vizuálny nesúlad, ale aj nesprávne vyhodnocovanie polohy rúk a správania pacientovej inverznej kinematiky (IK).

Desktop klient sa navyše spúšťal s predvolenou schémou ovládania určenou pre XR simulátor namiesto klasickej kombinácie myš a klávesnica. XR simulátor primárne slúži ako vývojová pomôcka – pre nevývojára je ťažko použiteľný. Terapeutický klient bol teda bez manuálneho zásahu pri štarte nepoužiteľný.

Ďalšia skupina problémov súvisela s používanými XR knižnicami. Verzia XR Interaction Toolkit v systéme nepodporovala moderné XR Hands funkcionality, t. j. sledovanie rúk používateľa kamerami headsetu (Meta Quest Hand Tracking) bez potreby fyzických ovládačov. Zároveň mala nevyhovujúce mapovanie ovládania — uchopenie objektu a teleportácia používali rovnaké trigger tlačidlo. Pri testovaní to viedlo k tomu, že používatelia namiesto zdvihnutia objektu neúmyselne teleportovali. Hoci nešlo o pád systému, kvalita interakcie a dôveryhodnosť rehabilitačného scenára tým výrazne klesli. Podrobnejšia analýza ukázala, že v použitej verzii bolo spracovanie vstupu pevne naviazané na rovnakú **trigger** udalosť pre obe akcie. Novšie verzie XR Interaction Toolkit od verzie 3 tento problém riešia prepracovaným input systémom, ich nasadenie by však vyžadovalo rozsiahlejší zásah do XR integračnej vrstvy a systému avatarov. Preto bolo riešenie tohto problému zaradené medzi úlohy pre verziu „V2“.

S tým súvisel aj problém použitých Ready Player Me modelov, ktorý sa týkal skôr ďalšieho vývoja interakcie než aktuálnej funkcionality. Hoci boli dostatočné

pre základné VR scenáre, neponúkali kostrovú ani vizuálnu kvalitu potrebnú pre plnohodnotnú podporu XR Hands; modely mali nízky detail rúk a nedostatočný počet kostí na robustnú implementáciu detailnejšej hand-tracking interakcie.

2.4.4 Nasadzovanie a prevádzka

Prevádzková vrstva systému bola výrazne poddimenzovaná. Neexistovala jednotná **release** stratégia, verzovanie **buildov** ani jasné oddelenie stabilných a experimentálnych verzií. Tester aj výskumní pracovníci často nevedeli presne identifikovať, ktorú verziu aplikácie používajú, a pri objavení chyby neexistoval rýchly mechanizmus návratu k poslednému stabilnému build-u.

Problém sa zhoršoval aj neexistenciou zdokumentovaného nasadzovacieho procesu pre Meta Quest zariadenia. Hodnoty ako package name a product name boli uložené v súboroch projektu, tie sa pri git merge konfliktoch ľahko prepísali. V tíme tak chýbal nielen proces vydávania, ale aj základné technické pravidlá chrániace proces vytvárania build-ov.

Problémy logovacieho subsystému popísané v sekcii 2.4.2 sa v prevádzkovom kontexte prejavovali obzvlášť citelne: ak sa chyba objavila na nasadenom Quest zariadení, jediným spôsobom získania diagnostických informácií bol fyzický prístup k zariadeniu.

2.4.5 Stabilita servera

Okrem problémov, ktoré boli v tejto práci riešené, boli identifikované aj ďalšie nedostatky týkajúce sa stability serverovej časti: (1) pri ukončení servera dochádzalo k pádom, najmä pri prerušení procesu cez `Ctrl+C`; išlo pravdepodobne o problém v nižšej runtime alebo knižničnej vrstve, nie o priamu chybu v spravovanom kóde aplikácie, (2) načítanie zoznamu miestností cez komunikáciu typu **Representational State Transfer (REST)** bolo v niektorých prípadoch extrémne pomalé a odozva mohla presiahnuť jednu minútu a (3) objavovali sa ťažko reprodukovateľné situácie, keď rôzni klienti videli odlišný stav cvičenia alebo boli v nekonzistentných režimoch relácie.

Ďalšou vrstvou otvorených problémov bola nekonzistentná hlasová komunikácia a neistota, ako sa Mirror knižnica správa pri externých útokoch alebo neštandardných sieťových vstupoch. Tieto problémy boli dôležité z hľadiska celkového technického obrazu systému, avšak ich riešenie by si vyžadovalo hlbší zásah do samotnej Mirror vrstvy a celkovej serverovej architektúry. Ich riešenie bolo predmetom samostatnej diplomovej práce [13]; po aktualizácii Unity a sú-

visiacich knižníc sa shutdown problém prestal prejavovať.

Riešenia identifikovaných problémov sú rozpracované v nasledujúcich kapitolách: komunikácia a protokol v kapitole 3, logovanie v kapitole 4, klinické a užívateľské vylepšenia v kapitole 5 a prevádzkové otázky v kapitole 6.

2.5 Architektonické rozhodnutia odvodené z identifikovaných problémov

Identifikované problémy (sekcia 2.4) ukázali, že pre L-NeRVEn nestačí „fungujúci prototyp“. Systém musí súčasne zvládať: (1) robustnú komunikáciu s externou EEG pipeline, (2) bezpečný prechod medzi vláknami mimo Unity a hlavným vláknom enginu, (3) dlhodobú diagnostikovateľnosť v nasadení a (4) prenositeľnosť komponentov do verzie „V2“. Z týchto obmedzení vyplynuli konkrétne architektonické rozhodnutia.

Hlavným rozhodnutím bolo oddeliť doménovú logiku od technológie prenosu. Preto je ako hlavný rámec použitá hexagonálna architektúra (*Ports and Adapters*), ktorú Cockburn [14] definuje ako model komunikácie jadra cez porty s napojením konkrétnych technológií cez adaptéry. V prípade L-NeRVEn to znamená, že logika cvičebného scenára nepracuje priamo s konkrétnou WebSocket knižnicou, ale cez rozhrania pre transport a odosielanie správ. Oddelenie slúži pre jednoduchú výmenu knižníc a pri migrácii do „V2“, ako bolo overené pri náhrade neudržiavanej knižnice websocket-sharp za Fleck (sekcia 3.4).

Na nižšej úrovni konkrétne obmedzenia určili výber návrhových vzorov: smerovanie správ podľa typu si vyžiadalo vzor Strategy; opakujúca sa kostra validácia→deserializácia→doménová akcia viedla k Template Method; a asynchrónny príjem správ voči jednovláknovému modelu Unity vyžadoval explicitné oddelenie príjmu správ na vláknach na pozadí od ich spracovania v hlavnom vlákne pomocou fronty správ. Posledný bod je obzvlášť dôležitý: Unity API je bezpečne modifikovateľné iba z hlavného vlákna, no sieťové knižnice prijímajú udalosti na vláknach na pozadí. Bez explicitného riešenia vzniká riziko race conditions a pádov — preto je v takomto systéme nevyhnutná samostatná transportná vrstva a dispečer hlavného vlákna.

Dôležitou, no často zanedbávanou súčasťou architektúry je logovanie. Diagnostické informácie nesmú byť vedľajším efektom náhodných Debug výpisov, ale súčasťou navrhutej infraštruktúry [15]. Skúsenosť s chybou náhodného odpájania WebSocket klienta (sekcia 2.4.1) potvrdila, že bez štruktúrovaných logov sa

diagnostika incidentov neúmerne predlžuje.

2.6 Odvodené požiadavky

Na základe identifikovaných problémov sú formulované požiadavky slúžiace ako akceptačné kritériá pre implementačné kapitoly a následné hodnotenie.

2.6.1 Funkčné požiadavky

- **FR1:** Komunikačná vrstva musí podporovať stabilnú obojsmernú výmenu správ medzi Unity serverom a OpenViBE **pipeline** bez prerušovania relácie pri bežnom priebehu cvičenia.
- **FR2:** Protokol musí používať formálnu správovú obálku so zrozumiteľnou typizáciou udalostí, timestampom a chybovým kanálom.
- **FR3:** Novoprijatý klient musí po pripojení získať konzistentný stav relácie (stav cvičenia, relevantné parametre a kontext miestnosti).
- **FR4:** Systém musí umožniť **runtime** zmenu relevantných parametrov (napr. MI threshold) z terapeutického rozhrania bez nutnosti reštartovať OpenViBE alebo Unity reláciu.
- **FR5:** Logovací systém musí poskytovať konzistentné logovacie úrovne a čitateľné diagnostické záznamy pre lokálne aj vzdialené nasadenie.

2.6.2 Nefunkčné požiadavky

- **NFR1 (Spôľahlivosť):** Riešenie musí minimalizovať náhodné odpájania počas relácie a poskytovať deterministické správanie pri chybových stavoch.
- **NFR2 (Udržiavateľnosť):** Architektúra musí oddelovať doménovú logiku od infraštruktúry tak, aby bolo možné meniť transportnú technológiu bez zásahu do jadra systému.
- **NFR3 (Thread-safety):** Prenos správ z transportných vlákien do Unity musí byť explicitne synchronizovaný a bezpečný voči **race conditions**.
- **NFR4 (Diagnostikovateľnosť):** Pri zlyhaní musí byť možné spätne určiť príčinu incidentu zo štruktúrovaných logov bez nutnosti sieťovej analýzy.
- **NFR5 (Prenositel'nosť):** Kľúčové komponenty navrhnuté v tejto práci musia byť použiteľné ako základ pre migráciu do verzie „V2“.

2.7 Rozhodnutie o rozsahu práce

Kľúčovým rozhodnutím tejto práce bolo určiť, ktoré problémy systému „V1“ má zmysel riešiť v existujúcej code-base a ktoré by už predstavovali neefektívnu investíciu do prechodného systému. L-NeRVEn „V1“ v čase začiatku tejto práce nepredstavoval stabilnú finálnu platformu, ale systém, ktorý už slúžil vo výskumnom a klinickom kontexte a zároveň bol zvažovaný jeho nástupca – „V2“. Plná oprava všetkých nedostatkov „V1“ by preto bola z inžinierskeho hľadiska nesprávnym rozhodnutím.

Výber konkrétnych príspevkov bol riadený dvoma kritériami. Prvým bol priamy klinický dopad, teda to, či problém bezprostredne škodil výskumným sedeniam SAV a či ho bolo možné rozumne odstrániť ešte v rámci „V1“. Druhým bola prenositeľnosť do „V2“, teda to, či navrhnuté riešenie môže existovať ako samostatný komponent alebo overený postup, ktorý bude mať hodnotu aj po ukončení životného cyklu verzie „V1“.

Tabuľka 2.3: Tabuľka priorít problémových oblastí použitá pri určovaní rozsahu práce.

Oblasť	Klinický dopad	Prenositeľnosť do „V2“	Priorita
WebSocket komunikácia	Vysoký	Vysoká	Vysoká
Logovanie a diagnostika	Stredný až vysoký	Vysoká	Vysoká
BCI integračné vylepšenia	Vysoký	Stredná	Vysoká
Usability chyby desktop/ XR	Stredný	Nízka až stredná	Stredná
XR toolkit a asset pipeline	Stredný	Vysoká („V2“)	Odložené
Mirror server stabilita	Vysoký	Vysoká (hlboký zásah)	Odložené

Do implementačného rozsahu boli zaradené oblasti s vysokou prioritou: prepracovanie WebSocketovej vrstvy, prepis logging systému, BCI integračné vylepšenia, opravy avatarov/**IK**/stavového automatu a **release** stratégia. Automatizácia pripojenia pacienta bola v tejto práci spracovaná ako návrh pre verziu „V2“, nie ako implementovaná funkcionálna. Čiastočne nadväzujúca časť, databáza presetov a nastavení pre pacienta, bola riešená v samostatnej práci [13].

Naopak, upgrade XR knižníc, náhrada avatar modelov a hlbšie serverové problémy naviazané na Mirror boli odložené na neskoršiu verziu „V2“. Synchronizácia stavu pre novoprihlásených klientov bola čiastočne vyriešená mechanizmom [SyncVar] (kapitola 5).

Nasledujúce kapitoly postupne prezentujú riešenia oblastí zaradených do rozsahu práce: kapitola 3 sa venuje komunikačnej vrstve a protokolu, kapitola 4 logovaniu, kapitola 5 klinickým a užívateľským vylepšeniam a kapitola 6 prevádzkovým otázkam. Splnenie odvodených požiadaviek FR1–FR5 a NFR1–NFR5 je následne vyhodnotené v kapitole 7.

3 WebSocketová komunikačná vrstva

Nestabilita WebSocketovej komunikácie identifikovaná v kapitole 2 predstavovala najzávažnejší technický problém systému L-NeRVEn. Pri zlyhaní tejto vrstvy dochádzalo priamo k prerušeniu terapeutických a výskumných sedení. Cieľom kapitoly je zdokumentovať postup riešenia: od analýzy konkrétnej chyby, cez návrh novej architektúry, až po implementáciu prenositeľnú do pripravovanej verzie „V2“.

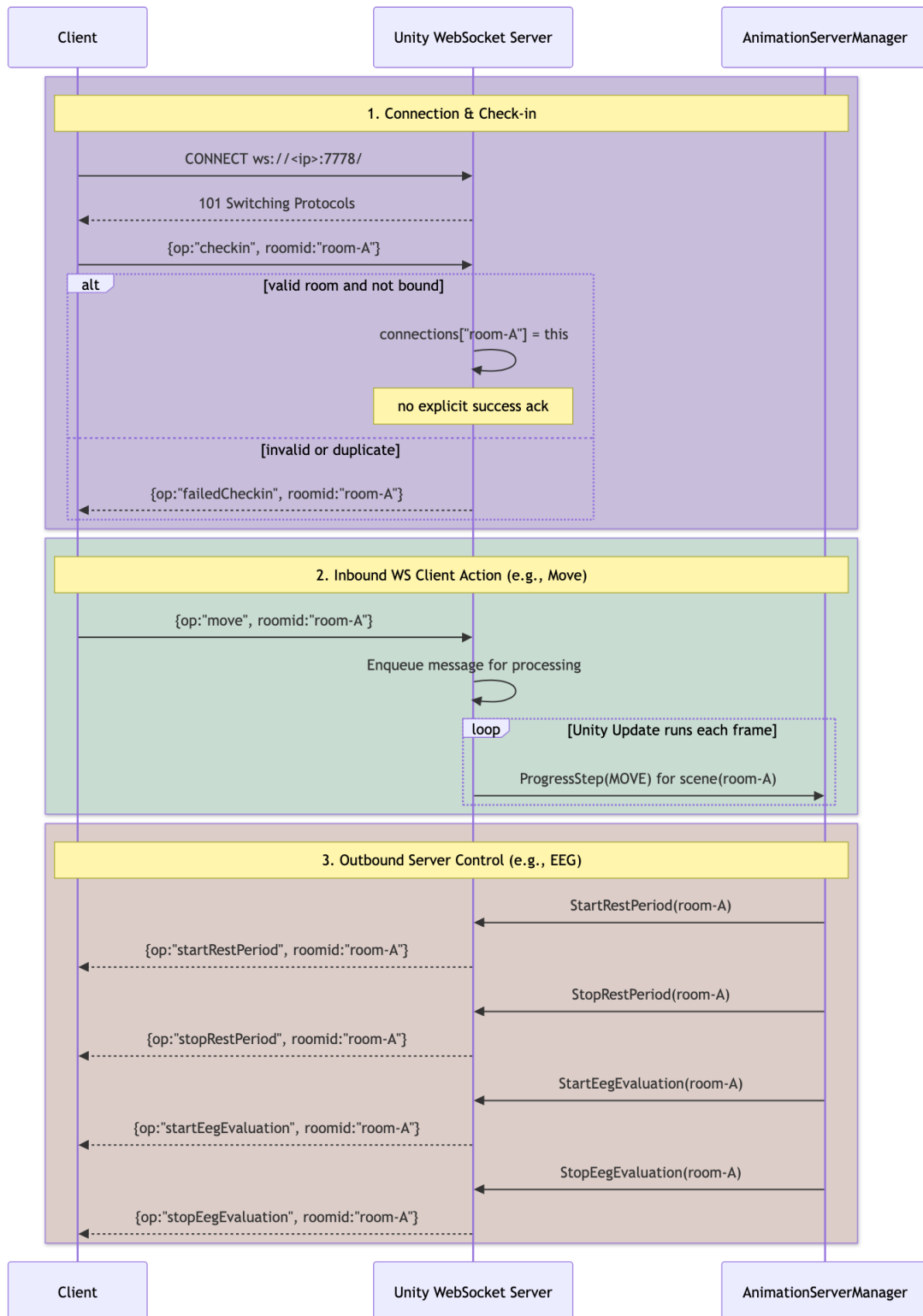
3.1 Analýza pôvodnej implementácie

Pôvodná WebSocketová vrstva slúžila ako dedikovaný kanál medzi serverovým **buildom** Unity aplikácie a externými procesmi, predovšetkým **pipeline** elektroencefalografie (**EEG**) pre rozhranie mozog-počítač (**BCI**) bežiacim v OpenViBE. Tento kanál bol oddelený od hlavnej klient-server komunikácie realizovanej knižnicou Mirror. Pôvodný protokol, zdokumentovaný aj v publikácii [12], podporoval iba obmedzený súbor riadiacich príkazov bez obojsmerného zdieľania klinických dát.

3.1.1 Architektúra a použité technológie

Serverová časť bola integrovaná priamo do Unity server **buildu** a používala knižnicu `websocket-sharp` vo verzii `1.0.3-rc11`. Server počúval na rovnakom zariadení ako herný server a typicky využíval samostatný port vedľa Mirror transportu. Jadro implementácie sa nachádzalo v monolitickej triede `WebSocketHandler.cs`. Tá naraz zabezpečovala inicializáciu servera, správu pripojení, **parsovanie** správ, smerovanie logiky a vybrané zásahy do stavu aplikácie.

Prichádzajúce správy boli ukladané do frontu a spracovávané v hlavnom vlákne Unity v metóde `Update()`, pričom za jeden snímok sa spracovala najviac jedna správa. Každé pripojenie bolo naviazané na konkrétnu miestnosť identifikovanú atribútom `roomId` a server udržiaval mapovanie medzi miestnosťou a aktívnym spojením.



Obrázok 3.1: Pôvodná architektúra WebSocketovej komunikácie

3.2 Analýza chyby náhodného odpájania

Najväčším problémom pôvodnej komunikačnej vrstvy bolo na prvý pohľad náhodné odpájanie WebSocket klienta používaného na strane OpenViBE. Problém bol so SAV hlásený počas viacerých mesiacov a prejavoval sa nekonzistentne — v niektorých prípadoch sa spojenie prerušilo počas cvičenia, v iných zlyhalo aj opätovné pripojenie.

3.2.1 Úvodná izolácia problému

Prvým krokom bolo oddelenie problému komunikačnej vrstvy od špecifického správania OpenViBE klienta. Ako náhrada za OpenViBE bol použitý nástroj Postman. S ním bolo simulovaný základný priebeh komunikácie. Testy s Postmanom však žiadne odpojenie nevyvolali. Výsledok testovania naznačil, že chyba súvisí so špecifickým správaním OpenViBE klienta v reálnom výskumnom scenári.

3.2.2 Vlastný Python klient na reprodukciu chyby

Všeobecné testovanie pomocou Postmanu neviedlo k reprodukcii problému, a tak bol vytvorený minimálny testovací klient `ws_test_client.py` (dostupný v repozitári projektu). Išlo o jednoduchý REPL nástroj postavený na knižnici `websocket-client`, ktorý umožňoval udržiavať jedno spojenie otvorené a interaktívne spúšťať jednotlivé typy udalostí. Pomocou tohto klienta boli systematicky otestované všetky hraničné scenáre ukončenia spojenia — korektné zatvorenie, prerušenie **Transmission Control Protocol (TCP)** spojenia bez Close rámca, vynútený reset aj simulovaný pád procesu. Ani v jednom z týchto prípadov sa nepodarilo vyvolať rovnaké správanie ako pri OpenViBE klientovi.

3.2.3 Testovanie s OpenViBE a Wiresharkom

Rozhodujúci krok nastal až pri spoločnom debugovacím sedení s výskumníkom zo Slovenskej akadémie vied, pri ktorom bol použitý reálny OpenViBE klient napojený na skutočnú EEG/BCI zostavu. V tomto prostredí sa problém podarilo zreprodukovať spoľahlivo. Analýza WebSocket komunikácie bola doplnená zachytávaním paketov pomocou nástroja Wireshark. Spojenie sa uzatváralo zdanlivo bez zjavnej príčiny a následne sa objavovala chyba s kódom 1002, ktorá v kontexte WebSocket protokolu označuje problém protokolovej vrstvy, často interpretovaný ako neplatný UTF-8 obsah v textovom rámci.

Keďže existujúce logovanie neposkytovalo dôvod uzatvorenia spojenia, bola vykonaná hlbšia analýza sieťovej komunikácie pomocou nástroja Wireshark. Až na tejto úrovni sa ukázalo, že skutočným dôvodom ukončenia nebol neplatný obsah správy, ale **KeepAlive timeout**. Server očakával odpoveď typu Pong v časovom limite jednej sekundy, no OpenViBE klient ju nestihol odoslať.

3.2.4 Príčina zlyhania a oprava

K zlyhaniu dochádzalo pri správe typu `rest`, po ktorej OpenViBE Python časť spúšťala vstupno-výstupné operácie a výpočet základnej EEG **baseline** používanej neskôr ako `threshold` hodnota pri vyhodnocovaní **Motor imagery**. Všetky operácie prebiehali na jednom vlákne a počas približne 6 až 7 sekúnd blokovali spracovanie ostatných rámcov vrátane Pong odpovede – čiže došlo k vypršaniu **KeepAlive** timeoutu a uzatvoreniu spojenia serverom.

Výsledná oprava pozostávala z dvoch krokov. Prvým bolo zvýšenie `KeepAlive` timeoutu z 1 sekundy na 10 sekúnd spolu s doplnením explicitného logu pri uzatvorení spojenia z tohto dôvodu. Druhým krokom bolo presunutie výpočtov `baseline` v OpenViBE časti na samostatné vlákno, aby hlavné komunikačné vlákno zostalo schopné odpovedať na Pong správy aj počas náročnejších výpočtov. Po nasadení oboch zmien sa odpojenia v nasledujúcich sedeniach prestali objavovať.

Analýza chyby zároveň ukázala, že problémom nebola iba jedna zle nastavená hodnota timeoutu, ale celková nízka diagnostikovateľnosť pôvodnej komunikačnej vrstvy. Slabé logovanie a monolitická implementácia značne predlžovali čas potrebný na analýzu. Z tohto dôvodu bola dočasná oprava chyby vnímaná iba ako stabilizačný krok a následne sa pristúpilo k úplnému architektonickému prepracovaniu vrstvy.

Predtým ako bolo možné pristúpiť k návrhu novej architektúry, bolo potrebné systematicky zmapovať štrukturálne nedostatky pôvodnej implementácie.

3.3 Nedostatky pôvodnej implementácie

3.3.1 **God object** a nízka udržiavateľnosť

Najvýraznejším architektonickým problémom bola koncentrácia príliš veľkého počtu zodpovedností do jednej triedy. Pôvodný `WebSocketHandler` v sebe spájal:

- správu **TCP**/WebSocket spojení,
- deserializáciu správ vo formáte **JavaScript Object Notation (JSON)**,

- rozhodovanie, ktorý typ správy sa má spracovať,
- väzbu na stav cvičenia a ďalšie objekty Unity.

Takáto štruktúra porušovala princíp jednej zodpovednosti a zároveň spôsobovala, že akákoľvek zmena v jednom type správy mohla neúmyselne ovplyvniť iné časti systému. Rozširovanie komunikačného protokolu si vyžadovalo priame zásahy do centrálnej triedy, tým sa zvyšovalo riziko regresí a prakticky sa znemožňovalo izolované testovanie logiky.

3.3.2 Thread-safety a práca s Unity aplikačným programovacím rozhraním (API)

Ďalším dôležitým problémom bolo miešanie sieťového spracovania s jednovláknovým modelom Unity. WebSocket knižnica prijímala udalosti na vláknach bežiacich na pozadí, zatiaľ čo Unity aplikačné programovacie rozhranie (API) je bezpečné iba v hlavnom vlákne. V pôvodnej implementácii nebola hranica medzi týmito dvoma vrstvami rozdelená.

Prístup k zdieľaným dátovým štruktúram, najmä ku slovníku, ktorý obsahoval aktívne pripojenia, nebol systematicky synchronizovaný. Pri súbežnom prístupe z vlákien knižnice websocket-sharp a hlavného vlákna Unity tak vznikalo riziko `InvalidOperationException`, nekonzistentného stavu relácií a nepredvídateľných pádov. Čistenie záznamov pri odpojení klienta bolo síce implementované, ale realizovalo sa manuálnou iteráciou cez celý slovník pripojení, bez väzby na správu relácií. Pri neštandardnom ukončení spojenia tak mohli v slovníku zostať neplatné záznamy odkazujúce na neexistujúce pripojenia.

3.3.3 Krehký protokol správ

Pôvodný protokol používal minimálnu štruktúru s poľami `op` a `roomId`. Tento návrh bol síce jednoduchý a z počiatku dostačujúci, no z dlhodobého hľadiska príliš rigidný. Chýbalo formálne potvrdenie úspešnej registrácie klienta, správy nemali jednotnú obálku pre doplnkové dáta a chyba pri parsovaní JSON mohla potenciálne viesť až k pádu spojenia, napríklad prostredníctvom `NullReferenceException`. Pre tím SAV to znamenalo, že vývoj voči systému bol znáročný — správanie pri chybových stavoch nebolo jednoznačne definované.

3.3.4 Riziko zahltenia frontu a latencie

Spracovanie iba jednej správy za snímok v kombinácii s neobmedzeným frontom predstavovalo ďalšie potenciálne riziko. Pri vysokej frekvencii prichádzajúcich MI správ by sa fronta mohla postupne zaplňať, čím by narastala latencia medzi skutočným stavom EEG klienta a reakciou v prostredí virtuálnej reality (VR). V extrémnom prípade by takáto situácia mohla viesť aj k prudkému rastu spotreby pamäte a nepredvídateľnému správaniu aplikácie, hlavne ak berieme do úvahy OpenViBE skripty blokujúce hlavné vlákno a prijímanie správ.

3.4 Návrh novej komunikačnej vrstvy

Nasledujúci návrh vychádzal z dvoch hlavných kritérií: komunikačná vrstva musela zostať stabilná počas výskumných sedení SAV vo verzii „V1“ a zároveň byť dostatočne izolovaná na priamy presun do pripravovanej verzie „V2“.

3.4.1 Požiadavky na nové riešenie

Z analýzy vyplynuli nasledujúce požiadavky:

- oddeliť transportnú infraštruktúru od aplikačnej a doménovej logiky,
- zabezpečiť bezpečný prechod správ z vlákien na pozadí do hlavného vlákna Unity,
- nahradiť neudržiavanú knižnicu modernou a spoľahlivou alternatívou,
- zaviesť jednotný a rozšíriteľný formát správ,
- štandardizovať spracovanie chýb a odpovedí klientovi,
- obmedziť vplyv prívalu správ na snímkovú frekvenciu VR aplikácie,
- vytvoriť architektúru vhodnú na migráciu do „V2“ bez väzby na zvyšok pôvodného kódu.

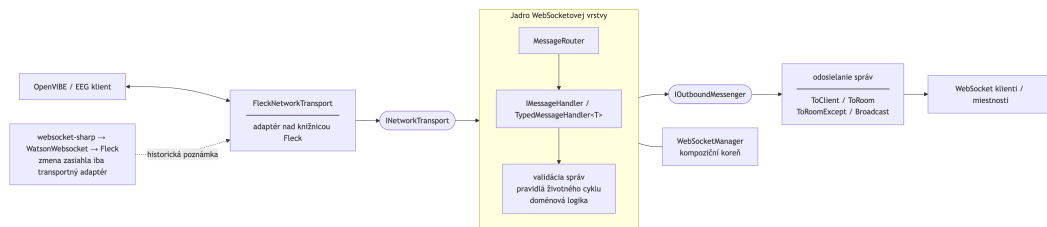
3.4.2 Hexagonálna architektúra

Navrhnuté riešenie využíva princípy **hexagonálnej architektúry**, známej aj ako *Ports and Adapters* [14]. Hlavná myšlienka spočíva v oddelení jadra aplikácie od

konkrétnej technológii prenosu. Doménová logika teda nepracuje priamo s konkrétnou WebSocket knižnicou, ale so súborom rozhraní, ktoré tvoria porty systému.

Vstupný port reprezentuje rozhranie `INetworkTransport`, ktorého úlohou je prijímať surové správy z transportnej vrstvy. Výstupný port reprezentuje rozhranie `IOutboundMessenger`, ktoré abstrahuje odosielanie odpovedí konkrétnym klientom, do miestností alebo všetkým pripojeným účastníkom. Medzi týmito portmi sa nachádza smerovacia vrstva a vrstva handlerov vykonávajúcich doménovú logiku.

Týmto prístupom sa odstránil pôvodný **god object** a zaviedli sa explicitné hranice medzi prenosom, smerovaním a spracovaním správ. Výslednú štruktúru portov, adaptérov a jadra subsystému ilustruje obrázok 3.2.



Obrázok 3.2: Hexagonálna architektúra WebSocketovej vrstvy s portami `INetworkTransport` a `IOutboundMessenger`

Migrácia WebSocket vrstvy nebola jednorazovým rozhodnutím, ale postupným spresňovaním výberu na základe reálnych obmedzení prostredia Unity.

Prvým východiskovým bodom bola knižnica `websocket-sharp`, od ktorej bolo potrebné ustúpiť pre neudržiavanosť a architektonické obmedzenia. Nasledujúcim kandidátom bola knižnica `WatsonWebSocket`. Javila sa ako vhodná náhrada najmä preto, že poskytovala serverovú implementáciu, moderný asynchrónny model a **API** relatívne blízke pôvodnému riešeniu.

Počas implementácie sa však ukázalo, že `WatsonWebSocket` interne využíva triedu `System.Net.HttpListener`, čo je problematické v **runtime** prostredí Unity Mono. Server odmietal niektoré korektné pokusy o pripojenie od štandardných klientov, napríklad Postman alebo Python skriptov, najmä pri spracovaní hlavičiek **HyperText Transfer Protocol (HTTP)** ako `permessage-deflate`. Zvolená knižnica teda nebola spoľahlivá v cieľovom prostredí, hoci mimo Unity pôsobila korektne.

Konečné riešenie využilo knižnicu `Fleck`. Tá je založená na `System.Net`.

Sockets. Znamená to, že pracuje priamo nad **TCP** socketmi namiesto vyššej **HTTP** abstrakcie používanej v predchádzajúcom riešení. Pre výsledný systém to znamená menšiu závislosť od problematickej časti **HTTP** zásobníka v prostredí Unity Mono, predvídateľnejšie spracovanie WebSocket **handshaku** a lepšiu interoperabilitu so štandardnými klientmi. Je treba zdôrazniť, že vďaka zavedeniu transportného portu zostal zvyšok architektúry od tejto zmeny izolovaný a zmena bola realizovaná bez potreby zásahu do doménovej logiky — bola potvrdená správnosť hexagonálneho návrhu.

3.4.3 Návrh formátu správ

S cieľom odstrániť rigiditu pôvodného protokolu bol navrhnutý jednotný formát správy pozostávajúci z troch hlavných častí:

- `eventType`: identifikátor typu správy v tvare `zdroj.príkaz`,
- `roomId`: identifikátor relácie alebo miestnosti,
- `payload`: flexibilný objekt nesúci konkrétne dáta správy.

Notácia `zdroj.príkaz` v atribúte `eventType` umožňuje čitateľné a rozšíriteľné smerovanie na oboch stranách komunikácie. Súčasťou obálky je aj atribút `timestamp`, ktorý nesie časovú pečiatku v unixových milisekundách. Pri prichádzajúcich správach ju nastavuje klient a server ju pri odchádzajúcich správach generuje sám. Server túto hodnotu nevaliduje; je potrebná na usporiadanie a analýzu latencie MI správ a na ich koreláciu so serverovými udalosťami cvičenia, napríklad pri sledovaní oneskorenia medzi odoslaním `exercise.action.move` a vykonaním príslušného kroku cvičenia na strane servera.

3.4.4 Pravidlá životného cyklu spojenia

Protokol definuje presné pravidlá autorizácie a postupnosti správ. Prvá správa odoslaná klientom musí byť vždy `client.checkin`. Ak klient odošle ľubovoľný iný typ správy pred úspešným check-inom, server odpovie správou `server.error` a požiadavku zahodí. Po úspešnom check-ine sa spojenie naviaže na jedinú `roomId`; ak klient neskôr odošle správu s inou `roomId`, ide o **cross-room** porušenie a server ju rovnako odmietne.

Pri opätovnom check-ine toho istého spojenia server nerozlišuje nový logický objekt, ale správa sa ako pri *reconnect*. Táto skutočnosť sa explicitne vracia v potvrdení `client.checkin.ack` pomocou stavu `connected/reconnected`.

Príklad páru požiadavka–odpoveď ukazujú výpisy 3.1 a 3.2. Úplný katalóg všetkých typov správ vrátane dátových streamov a chybových odpovedí je v systémovej príručke, sekcii B.7.

Výpis 3.1: Správa

```
client.checkin
```

```
"roomId": "room-123",
"eventType": "client.checkin",
"timestamp": 1700000000000,
"payload": {
  "clientType": "eeg"
}
```

Výpis 3.2: Odpoveď

```
client.checkin.ack
```

```
"roomId": "room-123",
"eventType": "client.checkin.ack",
"timestamp": 1700000000100,
"payload": {
  "clientType": "eeg",
  "status": "connected"
}
```

3.4.5 Zoznam udalostí protokolu

Implementovaný protokol podporuje nasledujúci zoznam udalostí. Na strane klienta:

- `client.checkin`,
- `mi.stream.fatigue`,
- `exercise.action.move`.

Na strane servera:

- `client.checkin.ack`,
- `mi.threshold.set`,
- `server.error`,
- `exercise.rest.start`, `exercise.rest.stop`,
- `exercise.eval.start`, `exercise.eval.stop`.

V porovnaní s pôvodným protokolom [12], ktorý poznal iba dve klient-ské správy (`checkin`, `move`) a šesť serverových príkazov, nový protokol zavádza obojsmerný tok klinických dát. Správy `mi.stream.fatigue` a `mi.threshold.set` umožňujú zdieľanie EEG/MI hodnôt medzi OpenViBE a terapeutickým rozhraním Unity. Pôvodný protokol túto funkciu nepodporoval. Porovnanie oboch protokolov je uvedené v tabuľke 3.1.

Tabuľka 3.1: Porovnanie pôvodného a nového komunikačného protokolu

Vlastnosť	Pôvodný protokol	Nový protokol
Klientské správy	2	3
Serverové správy	6	8
Obojsmerné zdieľanie EEG/MI dát	nie	áno
Potvrdenie registrácie	nie	áno
Štandardizované chyby	nie	áno
Formát obálky	op + roomId	eventType + roomId + payload

Výsledná dokumentácia vrstvy slúži zároveň ako integračná špecifikácia pre externých konzumentov, najmä EEG klienta.

S definovaným protokolom a zoznamom udalostí zostávalo vyriešiť, akým spôsobom budú tieto správy prúdiť cez jednovláknový vykonávací model Unity bez ohrozenia plynulosti VR aplikácie.

3.4.6 Návrh súbežnosti a riadenia prevádzky

Návrh bolo potrebné doplniť aj o spôsob, akým sa asynchrónny príjem sieťových správ oddelí od ich spracovania v hlavnom vlákne Unity. Riešením bolo vytvorenie dispečera hlavného vlákna založený na vzore **Producer-Consumer**: transportná vrstva prijíma správy na pozadí a vkladá ich do synchronizovaného frontu, zatiaľ čo hlavné vlákno Unity ich spracúva počas `update()` (implementačné detaily v 3.5).

Súčasťou návrhu bolo aj riadenie spracovania. Dispečer v každom snímku vyprázdňuje front do vyčerpania jedného z dvoch limitov: maximálneho počtu spracovaných akcií alebo pevného časového rozpočtu 5 ms meraného pomocou `Stopwatch`. Ak je niektorý limit dosiahnutý skôr, zostávajúce akcie sa odložia do nasledujúceho snímku. Súčasne sú definované ďalšie bezpečnostné limity (konkrétne hodnoty sú uvedené v 3.5). Rámce väčšie ako povolený limit sú zahodené bez odpovede. Rovnaké správanie sa uplatňuje aj pri neplatnom **JSON**-e, kde transportná vrstva nevytvára chybovú odpoveď, pretože správa ešte neprešla do aplikačnej úrovne.

3.5 Implementácia refaktORIZOVANEJ architektúry

Výsledná implementácia bola rozdelená do niekoľkých samostatných vrstiev a tried tvoriacich jednosmerný spracovateľský reťazec. Dáta prichádzajú z transportnej vrstvy, následne sú presunuté do hlavného vlákna Unity, overené, smerované ku konkrétnemu handleru a až potom aplikované na stav systému.

3.5.1 Transportná vrstva: **FleckNetworkTransport**

Trieda `FleckNetworkTransport` predstavuje adaptér nad knižnicou `Fleck` a implementuje vstupný port `INetworkTransport`. Jej úlohou je:

- inicializovať a spúšťať WebSocket server,
- prijímať rámce na vláknach spravovaných knižnicou `Fleck`,
- presúvať spracovanie správ do hlavného vlákna Unity,
- vynucovať prevádzkové limity, napríklad `MaxMessageBytes`.

Pri prijatí správy adaptér nevykonáva hernú logiku okamžite. Namiesto toho zapuzdří spracovanie do delegáta `System.Action` a vloží ho do synchronizovaného frontu `ConcurrentQueue<Action>`. S týmto mechanizmom sa odstránilo riziko medzivláknových prístupov a zároveň sa vytvorilo miesto, v ktorom je možné centrálne implementovať výkonnostné limity.

Transportná vrstva zároveň spresnila aj sieťové správanie servera. WebSocket endpoint počúva na adrese:

```
ws://{bindAddress}:{port}
```

Port je odvodený z `NetworkSettings.WebsocketPort` a predvolená hodnota je 7778; prepísať ju možno parametrom príkazového riadku `-socketport`. Bind adresa sa taktiež odvodzuje z konfigurácie: prázdna hodnota sa mapuje na `0.0.0.0`, hodnota `localhost` na `127.0.0.1` a inak sa použije explicitne nakonfigurovaná IP adresa.

3.5.2 Výstupný port: **IOutboundMessenger**

Odchádzajúca komunikácia je oddelená cez rozhranie `IOutboundMessenger`. Rozhranie abstrahuje detaily smerovania správ ku konkrétnemu klientovi alebo skupine klientov. Handler vie, že chce poslať odpoveď klientovi alebo do konkrétnej

miestnosti, ale nevie nič o tom, ako je správa serializovaná, cez ktorú knižnicu odchádza ani akým spôsobom sa riešia pripojenia. Kontrakt rozhrania je uvedený vo výpise 3.3.

Výpis 3.3: Rozhranie IOutboundMessenger

```
using NeuroRehab.Networking.Message;

namespace NeuroRehab.Networking.Transport
{
    public interface IOutboundMessenger
    {
        void ToClient(string clientId, WebSocketMessage message);
        void ToRoom(string roomId, WebSocketMessage message);
        void ToRoomExcept(string roomId,
            string exceptClientId,
            WebSocketMessage message);
        void Broadcast(WebSocketMessage message);
    }
}
```

3.5.3 Kompozičný koreň: WebSocketManager

Trieda WebSocketManager slúži ako vstupný bod celého subsystému. Predstavuje fasádu prepájajúcu transportnú vrstvu, správu relácií a smerovanie správ. Pri inicializácii registruje odber udalostí z transportnej vrstvy, overuje stav relácie klienta a pri odpojení zabezpečuje čistenie logických relácií, aby po výpadku nezostávali v systéme takzvané zombie vstupy.

Táto trieda zároveň vystupuje ako kompozičný koreň (*composition root*) v tom zmysle, že práve tu sa spája nová architektúra s pôvodným projektom. Zvyšok systému nemusí poznať detailnú štruktúru handlerov, routera ani transportu; komunikuje iba s manažérom, ktorý subsystém zastrešuje.

Server štartuje v CustomNetworkManager.OnStartServer(); táto metóda volá WebSocketManager.StartServer(). Pre zabránenie prenosu starého logického stavu medzi behmi servera sa pri štarte čistia aktívne session objekty. Zastavenie prebieha v CustomNetworkManager.OnStopServer() alebo pri zničení objektu v WebSocketManager.OnDestroy().

3.5.4 Smerovanie správ: `MessageRouter`

Samotné smerovanie je implementované triedou `MessageRouter` využívajúc vzor **Strategy**. Router udržiava mapovanie medzi hodnotou `eventType` a konkrétnou implementáciou `IMessageHandler`. Po prijatí obálky správy teda nevykonáva dlhý `switch/case`, ale na základe reťazca vyhľadá príslušnú stratégiu a deleguje jej spracovanie.

Týmto prístupom sa dosiahli dve dôležité vlastnosti. Po prvé, pridanie nového typu správy si vyžaduje iba vytvorenie nového handlera a jeho registráciu bez modifikácie existujúceho kódu. Po druhé, prípadná chyba v jednom handleri neohrozuje celú smerovaciu logiku.

Router vyhodnocuje chybové stavy na základe pravidiel životného cyklu definovaných v sekcii 3.4.4. Pri ich porušení vracia klientovi štandardizovanú odpoveď `server.error` s popisom konkrétneho zlyhania.

3.5.5 Dvojfázová deserializácia a `TypedMessageHandler`

Spracovanie správ bolo štandardizované pomocou dvojfázovej deserializácie. V prvej fáze sa z prichádzajúceho **JSON** reťazca vytvorí iba obálka `WebSocketMessage` — tá obsahuje hlavičku `eventType`, `roomId` a surový objekt `payload`. Až v druhej fáze konkrétny handler deserializuje `payload` do silne typovaného DTO objektu.

Tento prístup prináša dve výhody. Router môže rozhodnúť o smere správy bez nákladnej hlbkej deserializácie a zároveň chyba v jednej štruktúre `payload` neohrozuje samotné smerovanie. Na odstránenie opakovaného kódu bola väčšina handlerov postavená na abstraktnej bázeovej triede `TypedMessageHandler<T>`. Tá vynucuje jednotný postup:

1. kontrola existencie `payload`,
2. deserializácia na typ `T`,
3. logická validácia povinných polí,
4. vykonanie konkrétnej doménovej logiky.

Konkrétne handlers sa tak môžu sústrediť na vlastnú zodpovednosť namiesto opakovaného riešenia rovnakých validačných a chybových scenárov.

3.5.6 Typová bezpečnosť protokolu

Na hranici medzi **JSON** protokolom a doménovým modelom sa používa explicitné mapovanie reťazcových hodnôt na interné enumy. Tým sa zabraňuje šíreniu **magických reťazcov** (*magic strings*) do jadra aplikácie. Príklad je uvedený vo výpise 3.4.

Výpis 3.4: Enum `ClientType` s mapovaním hodnôt protokolu

```
namespace NeuroRehab.Networking.Protocol
{
    [JsonConverter(typeof(StringEnumConverter))]
    public enum ClientType
    {
        Unknown = 0,
        [EnumMember(Value = "patient")]
        Patient,
        [EnumMember(Value = "therapist")]
        Therapist,
        [EnumMember(Value = "eeg")]
        Eeg
    }
}
```

Využitie tejto formy mapovania zvyšuje čitateľnosť kódu a znižuje pravdepodobnosť chýb pri porovnávaní reťazcov a vytvára prirodzené miesto, kde sa protokol prekladá do interného doménového modelu.

3.5.7 Kompilačná hranica a interoperabilita s legacy kódom

Sieťový subsystém bol izolovaný vo vlastnej definícii zostavenia `NeuroRehab.Networking`. Výhoda kompilačnej hranice je praktická: zabraňuje náhodným závislostiam na zvyšku legacy projektu a skracaje čas rekompilácie.

Zároveň bolo potrebné zachovať interoperabilitu s pôvodnými konfiguračnými objektmi založenými na `ScriptableObject`. Zvolený na tento účel bol vzor *Adapter* pomocou čiastočných tried, ktorým sa pôvodný objekt `NetworkSettings` rozšíril tak, aby implementoval nové doménovo užšie rozhrania, napríklad `IWebSocketServerConfig`. Použitie *partial* triedy namiesto samostatného obalovacieho adaptéra bolo zvolené preto, že `NetworkSettings` je serializovaný `ScriptableObject` referencovaný desiatkami `MonoBehaviour` komponentov cez `Inspector`. Zavedenie nového typu by vyžadovalo úpravu všetkých týchto re-

ferencií a ich opätovné priradenie v editore, zatiaľ čo `partial` trieda zachovala existujúci `asset` aj všetky väzby bez zmeny - pôvodný kód tak mohol zostať nedotknutý a zároveň sa získala možnosť implementovať nové rozhrania potrebné pre WebSocketový subsystém a čítať legacy dáta bez priamej previazanosti novej architektúry na celý pôvodný dátový model.

3.5.8 Katalóg implementovaných handlerov

V aktuálnej verzii sú v `MessageRouter` registrované tri základné handlers. Register je úmyselne malý; nový typ správy si vyžaduje iba vytvorenie handlera a jeho registráciu.

client.checkin. Správu spracúva `CheckInHandler`, ktorý deserializuje `payload` do `CheckInPayload`. Handler validuje identifikátor miestnosti a typ klienta, naviaže spojenie na konkrétnu reláciu a pri úspechu vráti `client.checkin.ack`.

mi.stream.fatigue. Správa reprezentuje priebežný stream motor imagery (**MI**). Handler očakáva `payload` s atribútom `value`; túto EEG/**MI** hodnotu odovzdá serverovej logike miestnosti.

exercise.action.move. Správa spúšťa jednorazový krok cvičenia v danej miestnosti. Handler nevyžaduje žiadne doménové dáta; ak `payload` existuje, musí byť prázdny objektom.

Okrem typed handlerov je dôležité spomenúť aj sadu serverom generovaných odchádzajúcich udalostí. Pri úspešnom check-ine EEG klienta server bezprostredne po `client.checkin.ack` odošle aj `mi.threshold.set`. Klient tak dostane aktuálnu `threshold` hodnotu miestnosti. Táto informácia nie je súčasťou `ack` `payloadu`, ale samostatnou správou.

3.5.9 Spracovanie chýb a ochrana zdrojov

Spôľahlivosť bola posilnená viacvrstvovou stratégiou. Na úrovni domény boli definované špecifické typy výnimiek, napríklad `HandlerValidationException` pre neplatné vstupy a `HandlerOperationException` pre požiadavky, ktoré nemožno vykonať v aktuálnom stave systému. `MessageRouter` obalí vykonávanie handlerov do štandardizovaného bloku `try-catch` a pri zachytení výnimky vygeneruje konzistentnú chybovú odpoveď pre klienta.

Na úrovni transportu boli zavedené explicitné limity:

- `MaxMessageBytes` = 1 MB na obmedzenie veľkosti prichádzajúcich správ,

- `MaxQueueActionsPerFrame` = 500 na limit spracovania v jednom snímku,
- `QueueWarnThreshold` = 2000 ako signalizácia rastúceho zahltenia frontu.

Kombinácia týchto mechanizmov chránila systém pred poškodenými vstupmi, náhodnými špičkami v prevádzke aj pred scenármi, ktoré by mohli ohroziť plynulosť VR aplikácie. Aplikačné chyby, pri ktorých je možné identifikovať `eventType` a kontext správy, vracajú štandardizovanú odpoveď `server.error`. Transportné chyby (neplatný **JSON**, príliš veľká správa) odpoveď negenerujú — ich spracovanie končí pred vytvorením obálky `WebSocketMessage`.

3.6 Výsledok a prenositeľnosť do „V2“

Refaktoring WebSocketovej vrstvy priniesol dve skupiny výsledkov. Prvou bola okamžitá stabilizácia prevádzky v L-NeRVEn „V1“. Po odstránení chyby **KeepAlive** timeoutu a po nasadení novej architektúry prestali byť náhodné odpojenia opakujúcim sa problémom pri klinických sedeniach so SAV. Zároveň vznikol zrozumiteľnejší a lepšie diagnostikovateľný komunikačný kanál, voči ktorému bolo možné vyvíjať ďalšie BCI funkcionality s nižším rizikom regresíí.

Druhou skupinou výsledkov bola strategická hodnota pre pripravovanú verziu „V2“. Nová vrstva už nebola previazaná s konkrétnou implementáciou WebSocket knižnice ani s monolitickým legacy kódom. Vďaka samostatnej definícii zostavenia, rozhraniam portov a oddeleniu handlerov vznikol podsystém, ktorý je možné do „V2“ preniesť ako samostatný základ a napojiť na novú správu relácií a aplikačné služby.

Aktuálna implementácia má známe obmedzenia. Komunikácia prebieha výhradne cez nešifrovaný kanál `ws://` bez podpory **Transport Layer Security (TLS)** (`wss://`). V súčasnom nasadení, kde server aj klienti pracujú v rámci lokálnej siete, to nepredstavuje bezprostredné riziko; pri budúcom nasadení cez verejnú sieť bude však potrebné doplniť šifrovanie. Protokol takisto neobsahuje autentifikáciu klienta — overenie identity sa spolieha na správnosť `roomId` a `clientType` zaslaných v správe `client.checkin`.

4 Systém logovania

Nedostatky logovacieho systému identifikované v kapitole 2 sa ukázali ako jeden z faktorov ktorý podstatne spomalil diagnostiku chyby WebSocketového odpájania (sekcia 3.2), keďže pôvodný logger nedokázal spoľahlivo uchovávať logy na zariadeniach Meta Quest. Kapitola postupne pokrýva analýzu pôvodného loggeru, návrh a implementáciu nového riešenia.

4.1 Analýza existujúceho loggeru

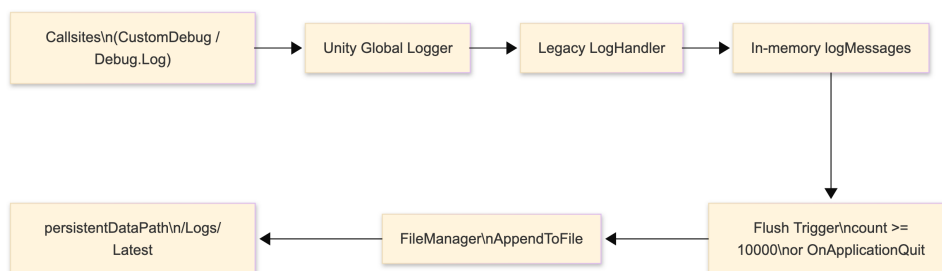
Pôvodný logovací systém pozostával z dvoch hlavných komponentov: (1) trieda `LogHandler`, implementácia rozhrania `ILogHandler`, ktorá zabezpečovala zachytávanie a ukladanie logov do súborov a (2) trieda `CustomDebug`, *wrapper* nad `Debug.Log*` volaniami Unity, poskytujúci farebné formátovanie a filtrovanie podľa logovacích úrovní. Oba komponenty boli navrhnuté nezávisle a neboli prepojené spoločným rozhraním ani injekciou závislostí.

Analýza zdrojového kódu repozitára odhalila výrazne nekonzistentné používanie logovacích volaní: 161 výskytov `customDebug.*`, 89 priamych volaní `Debug.Log*` v aplikačnom kóde. Nekonzistentné používanie rôznych rozhraní naznačovalo nielen technický dlh, ale aj absenciu jednotnej stratégie logovania naprieč projektom.

Obrázok 4.1 ilustruje *pipeline* pôvodného logovacieho systému od volacieho kódu až po zápis na disk.

4.1.1 Architektúra a väzba na životný cyklus

Trieda `LogHandler` bola globálne inicializovaná v metóde `Awake()` triedy `SettingsManager` prostredníctvom nahradenia existujúceho predvoleného volania `Debug.unityLogger.logHandler`. Zdrojový kód na tomto mieste obsahoval komentár *fast solution, would be better to rework it to use custom logger* — priamy dôkaz toho, že autor pôvodnej implementácie si bol vedomý architektonického kompromisu. Globálne nahradenie handlera malo dva zásadné dôsledky.



Obrázok 4.1: Architektúra pôvodného logovacieho systému

Dostupnosť loggeru závisela od životného cyklu `SettingsManager` a od poradia inicializácie scén v Unity — ak sa `SettingsManager` nenačítal ako prvý, logy z ostatných komponentov neboli zachytávané. Navyše, všetky logy Unity engine, knižníc tretích strán aj aplikačného kódu boli smerované cez jedinú implementáciu. Akákoľvek chyba v `LogHandler` tak priamo ovplyvňovala celý systém.

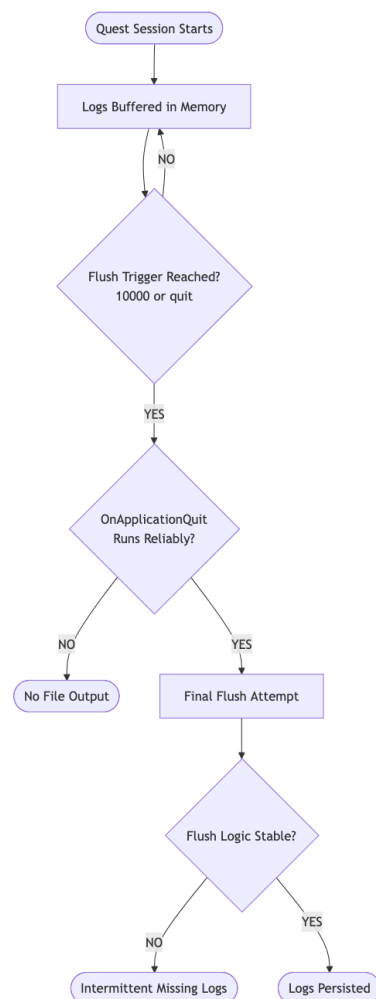
Ďalším problémom bolo správanie pri ukončení aplikácie — metóda `OnApplicationQuit()` v `SettingsManager` po uložení nastavení a zapísaní logov volala `Process.GetCurrentProcess().Kill()`. Proces sa tým okamžite ukončil bez štandardného cleanup postupu — prístup nekompatibilný s životným cyklom mobilných platforiem, najmä Meta Quest, kde systém Android riadi ukončovanie aplikácií vlastnými mechanizmami [16]. Násilné ukončenie cez `kill()` zabráňovalo dokončeniu prebiehajúcich operácií.

4.1.2 Chyby v perzistencii a logike zápisu

Najzávažnejšou slabinou pôvodného loggeru bola jeho stratégia perzistencie. Trieda `LogHandler` ukladala prichádzajúce správy do `List<string>` v pamäti a na disk ich zapisovala iba v dvoch prípadoch: keď počet správ dosiahol konfigurovateľný limit, alebo pri explicitnom zavolaní metódy `WriteCurrentLogs()` z `OnApplicationQuit()`. V aktuálnej konfigurácii bol limit nastavený na 10 000 správ — hodnota, ktorú bežné klinické sedenia na zariadení Meta Quest nedosiahli. Na platforme Android navyše nebol callback `OnApplicationQuit()` spoľahlivo volaný pri všetkých spôsoboch ukončenia aplikácie [17, 16]. Výsledkom bolo, že počas bežnej prevádzky na Quest neboli logy vôbec zapísané na disk. Logy sa strácali.

Problém nespočíval v ceste k súborom — pôvodná implementácia správne používala `Application.persistentDataPath` cez triedu `FileManager`, platný za-

pisovateľný adresár na zariadeniach s Android. Príčinou zlyhania bol návrh odloženého zápisu závislý na životnom cykle aplikácie [16, 17]. Obrázok 4.2 vizualizuje reťazec vedúci k intermitentnej absencii logov na zariadeniach Quest.



Obrázok 4.2: Kauzálny reťazec zlyhania perzistencie logov na Meta Quest

Samotná metóda `writeCurrentLogs()` navyše obsahovala chybu v indexovej aritmetike. Pôvodná implementácia v triede `LogHandler` počítala počet dávok výrazom $\text{ceil}(\text{count} / \text{logsPerWrite}) - 1$. Pri prázdnom bufferi ($\text{count} = 0$) tak vznikla hodnota -1 , ktorá sa následne použila pri výpočte rozsahu pre `GetRange`. Výsledkom bolo vyvolanie `ArgumentOutOfRangeException` práve v okamihu finálneho zápisu logov.

Ďalším závažným problémom bol **reentrant feedback loop** v zápise. Metóda `FileManager.AppendToFile()` volala `Debug.Log()` ešte pred samotným volaním

`File.AppendAllText()`. Keďže globálny `Debug.unityLogger.logHandler` bol nahradený triedou `LogHandler`, táto diagnostická správa sa vracala späť do rovnakého logovacieho mechanizmu a mohla rekurzívne spúšťať ďalší zápis. Pod záťažou tak feedback loop neúmerne zväčšoval objem logov a narúšal správanie pri zápise.

Navyše, výsledok operácie `AppendToFile()` nebol v `LogHandler` nijako kontrolovaný. Buffer `logMessages` bol po každom pokuse o zápis bezpodmienečne vymazaný volaním `Clear()`, a v prípade zlyhania zápisu tak dochádzalo k tichej strate dát.

4.1.3 Synchronizácia a bezpečnosť vlákien

Interný buffer `logMessages` typu `List<string>` nebol chránený žiadnym synchronizačným mechanizmom; metóda `LogMessage()`, ktorá pridávala záznamy do zoznamu, a metóda `WriteCurrentLogs()`, ktorá z neho čítala a mazala záznamy, mohli byť volané z rôznych vlákien. V Unity prostredí, kde sieťové knižnice typicky generujú udalosti na vláknach na pozadí, bol nesynchronizovaný buffer zdrojom **race conditions** a nekonzistentného stavu.

Konštruktor triedy `LogHandler` vykonával synchronnú operáciu `ZipFile.CreateFromDirectory()` na kompresiu starých logov bez akéhokoľvek ošetrovania výnimiek, kompresia blokovala inicializáciu aplikácie a v prípade zamknutého adresára spôsobovala zlyhanie.

4.1.4 Nedostatky v **observability**

Z hľadiska diagnostickej kvality logov vykazoval pôvodný systém viaceré štrukturálne nedostatky. Metóda `LogException()` zachytávala iba `exception.Message` bez stack trace, a tým výrazne znižovala diagnostickú hodnotu záznamov. Záznamy ukladané do súborov boli jednoduché reťazce bez štruktúrovaných polí pre úroveň závažnosti, kanál, modul alebo kontext volajúceho, čo je v rozpore s princípmi štruktúrovaného logovania [18].

Enum `LogLevel` definovaný v triede `_Enums.cs` obsahoval iba štyri hodnoty: `NONE`, `ERROR`, `WARNING` a `ALL`. Chýbala úroveň `Info` aj `Debug`, a jemné filtrovanie medzi informačnými a ladiacimi správami nebolo možné. Trieda `CustomDebug` síce poskytovala filtrovanie na základe tohto enumu, no súbežné používanie priamych `Debug.Log*` volaní v aplikačnom kóde túto politiku obchádzalo. Pôvodný systém taktiež neimplementoval žiadnu politiku správy úložiska, ako napríklad automatickú rotáciu alebo mazanie starých log súborov.

4.1.5 Dopad na diagnostiku WebSocketovej chyby

Identifikované nedostatky mali priamy dopad na diagnostiku chyby opísanej v sekcii 3.2. S funkčným štruktúrovaným logovaním by záznam typu [WARN] [Network] Connection closed: KeepAlive timeout okamžite identifikoval príčinu odpojenia bez potreby zachytávania sieťových paketov nástrojom Wireshark.

Tabuľka 4.1 sumarizuje identifikované nedostatky zoradené podľa priority.

Tabuľka 4.1: Prehľad identifikovaných nedostatkov pôvodného loggeru

Priorita	Oblasť	Nedostatky
P0	Perzistencia a korektnosť	Odložený flush závislý na životnom cykle (limit 10 000 alebo quit), chyba indexovania pri prázdnom bufferi, reentrant feedback loop v zápise
P1	Spoľahlivosť a bezpečnosť	Tichá strata dát po zlyhaní zápisu, ne-synchronizovaný zdieľaný buffer, násilné <code>Process.Kill()</code> pri ukončení, skrátené výnimky bez stack trace, nemaskované citlivé dáta v logoch
P2	Architektúra a správa	Logger viazaný na <code>SettingsManager</code> , globálny override s <i>side effects</i> , synchronná ZIP operácia pri štarte, žiadny úložiskový limit, hrubá <code>LogLevel</code> politika (4 hodnoty), nekonzistentné API naprieč kódom, nepoužívaný nový prototyp loggeru, chýbajúca ochrana dispatch pipeline

4.2 Návrh

Analýza odhalila nedostatky v troch kategóriách: spoľahlivosť perzistencie (odložený **flush**, chyby korektnosti, reentrancia), kvalita **observability** (chýbajúce metadáta, skrátené výnimky, hrubé filtrovanie) a architektonická väzba (globálny override, zmiešané API, závislosť na životnom cykle). Návrh nového riešenia preto musel pokryť všetky tri kategórie súčasne a zároveň zachovať prenositeľnosť do pripravovanej verzie „V2“.

4.2.1 Existujúce logovacie frameworky a zdôvodnenie vlastného riešenia

Voľba logovacieho riešenia pre L-NeRVEn bola obmedzená tromi praktickými faktormi cieľového prostredia. Jadrom problému nebola absencia logovacieho frameworku, skôr absencia spoľahlivej perzistencie na platforme Android/Quest. Kľúčová požiadavka — deterministický **flush** viazaný na životný cyklus aplikácie na Quest — vyžadovala priamu kontrolu nad časovaním zápisu a registráciu platformovo špecifických mechanizmov ukončenia. K tomu sa pridávajú obmedzenia **runtime**: Unity používa Mono resp. backend **Intermediate Language to C++ (IL2CPP)**, ten nie je plne kompatibilný so všetkými .NET závislosťami a integrácia externých frameworkov si vyžaduje overenie kompatibility s **IL2CPP** kompiláciou pre platformu Meta Quest. Klinické nasadenie systému v spolupráci so SAV navyše vyžadovalo minimalizáciu externých závislostí z dôvodu jednoduchšej údržby a nižšieho rizika tranzitívnych problémov pri aktualizáciách.

Ekosystém platformy .NET napriek tomu ponúka niekoľko etablovaných logovacích frameworkov. Serilog [19] je postavený na princípe štruktúrovaného logovania, kde každý záznam je typovaná udalosť s pomenovanými vlastnosťami namiesto простého textového reťazca. Štruktúrované logovanie, označované aj ako sémantické logovanie [18], umožňuje efektívne strojové spracovanie logov. NLog [20] poskytuje konfigurovateľnú **pipeline** s veľkou sadou výstupných cieľov a filtrovacích pravidiel. Apache log4net [21] je portom frameworku log4j do prostredia .NET s hierarchickým modelom loggerov. Všetky tri frameworky sú produkčne overené a ponúkajú rozsiahly ekosystém rozšírení. Avšak kvôli vyššie uvedeným obmedzeniam cieľovej platformy nebolo možné tieto priemyselné štandardy priamo nasadiť; ich kompatibilita s Android, Meta Quest životným cyklom je otázna.

Pre L-NeRVEn bolo preto zvolené vlastné riešenie. Implementácia priamo zameraná na systém a jeho obmedzenia s približne 420 riadkami kódu pokryla požadovanú funkcionálnosť bez zavedenia externých závislostí. Z architektonického hľadiska toto rozhodnutie nebráni budúcej integrácii. Rozhranie `ILogger` je dostatočne generické na to, aby adaptér nad Serilogom alebo NLogom mohol byť implementovaný ako alternatívna `ILogger` implementácia bez zmeny volajúceho kódu.

4.2.2 Požiadavky na nové riešenie

Požiadavky boli odvodené z nálezov v tabuľke 4.1 a zoradené od perzistencie (P0) cez spoľahlivosť (P1/P2) po prenositeľnosť.

- priebežný bufferovaný zápis obmedzujúci riziko straty logov pri ukončení aplikácie,
- prahový a explicitný **flush** odstraňujúci závislosť iba na `OnApplicationQuit()`,
- synchronizovaný buffer s explicitným zámkom,
- štruktúrovaný formát záznamu s časovou pečiatkou, úrovňou, kanálom a kontextom volajúceho,
- jednotná fasáda nahrádzajúca zmiešané logovacie vzory,
- bezpečné publikovanie logovacích udalostí pre používateľské rozhranie,
- nezávislosť od `SettingsManager` a globálneho override,
- mechanizmy riadeného ukončenia kompatibilné so životným cyklom Android,
- prenositeľnosť do „V2“: jednoduché jadrové rozhranie s izolovanými väzbami na Unity.

4.2.3 Architektonické rozhodnutia

Nový logovací systém bol postavený na niekoľkých kľúčových architektonických rozhodnutiach.

Závislosť na `SettingsManager` odstránil návrhový vzor **Facade** [22]. Statická trieda `GameLog` poskytuje jednotný vstupný bod pre logovacie volania. Inicializácia je **thread-safe** vďaka použitiu lock mechanizmu a prebieha automaticky ešte pred načítaním scény pomocou atribútu `RuntimeInitializeOnLoadMethod`. Ak volajúci kód neposkytne vlastnú implementáciu cez `Initialize()`, fasáda vytvorí predvolený logger podľa typu **buildu**: v editore `UnityLogger`, mimo editora `CompositeLogger` kombinujúci `FileLogger` a `UnityLogger`. Player buildy tak zapisujú perzistentný súbor a zároveň ponechávajú výstup dostupný aj cez Unity konzolu.

Perzistenciu v player buildoch zabezpečuje `FileLogger`. Implementácia zámerne používa jednoduchý synchronný model: záznamy sa ukladajú do `List<string>` chránenej zámkom a pri **flushi** sa naraz zapíšu cez

`File.AppendAllText()`. Bežné informačné správy sa zapisujú po prekročení prahu 20 záznamov, zatiaľ čo varovania a chyby vyvolajú okamžitý zápis. Na platforme Android a v batch režime sa `FileLogger` prepína do režimu zápisu pri každej správe, čím sa minimalizuje strata logov pri neočakávanom ukončení procesu. Konštruktor umožňuje prepísať cestu k súboru, flush prah, režim zápisu pri každej správe aj maximálnu veľkosť log adresára. Predvolený limit úložiska je 64 MB; po úspešnom zápise `FileLogger` odstráni najstaršie *.log súbory okrem aktuálne otvoreného súboru.

Návrh zároveň oddelil závažnosť logu od doménového kontextu. Závažnosť je vyjadrená voľbou metódy (`Info`, `Warning`, `Error` a `Exception`) a doménový kontext parametrom `LogChannel` s hodnotami `General`, `System`, `Network`, `UI` a `Audio`. Pôvodný enum `LogLevel` zostal zachovaný ako runtime filter úrovne logovania; nový `LogChannel` ho teda nenahradil, ale doplnil o rozmer subsystému.

Zvážené architektonické alternatívy

Pri návrhu boli zvážené aj alternatívne prístupy. Asynchrónny logger s dedikovaným vláknom a periodickým **flushom** by znižoval blokovanie hlavného vlákna, no zároveň by zavádzal ďalší životný cyklus, ukončovanie vlákna a riziko nedokončeného zápisu pri ukončení aplikácie. Pre aktuálny rozsah bol preto zvolený jednoduchší synchronný model s malým bufferom, okamžitým zápisom kritických udalostí a explicitnými lifecycle hookmi. Statická fasáda `GameLog` bola vybraná pre nízky migračný *overhead* existujúcich volaní, pričom verejná metóda `Initialize()` zachováva možnosť nahradiť implementáciu.

4.2.4 Návrh prenositeľnosti do „V2“

Rovnako ako pri WebSocketovej vrstve (kapitola 3) bola prenositeľnosť do „V2“ jedným z návrhových kritérií [15]. Rozhranie `ILogger` a enum `LogChannel` sú definované v *namespace* `Core.Diagnostics`. Rozhranie `ILogger` aj `LogChannel` používajú voliteľný parameter `UnityEngine.Object` pre kontext, takže nie sú úplne nezávislé od Unity, ale ich väzba je úzka a explicitná. Trieda `FileLogger` používa `Application.persistentDataPath`, `Application.platform` a metadáta aplikácie pri vytvorení cesty a úvodných riadkov log súboru.

Pre migráciu do „V2“ je najľahšie prenositeľný kontrakt: `ILogger`, `IFlushableLogger`, `LogChannel` a verejné API fasády `GameLog`. Platformovo špecifickú náhradu by vyžadovali `UnityLogger`, lifecycle hooky nad `MonoBehaviour` a časti `FileLogger` závislé od triedy `Application`.

4.3 Implementácia

Architektonické rozhodnutia z predchádzajúcej sekcie určili štruktúru implementácie. Vzor **Facade** sa premietol do triedy `GameLog`, oddelené výstupné adaptéry do tried `FileLogger` a `UnityLogger` a kontrakt systému do rozhraní `ILogger` a `IFlushableLogger`. Implementácia nového logovacieho systému tak pozostáva z týchto komponentov: rozhranie `ILogger` a enum `LogChannel` definujúce kontrakt, statická fasáda `GameLog` riadiaca inicializáciu, formátovanie, runtime filtrovanie a **flush**, `FileLogger` zabezpečujúci súborovú perzistenciu, `UnityLogger` poskytujúci farebne odlíšený konzolový výstup, `CompositeLogger` kombinujúci viac výstupov a `GameLogLifecycleHooks` prepájajúci logger so životným cyklom Unity.

4.3.1 Rozhranie `ILogger` a `LogChannel`

Kontrakt logovacieho systému definuje rozhranie `ILogger` s tromi metódami: `Log`, `Warning` a `Error`. Každá metóda prijíma `LogChannel` určujúci doménovú oblasť, textovú správu a voliteľný `Object` kontext umožňujúci identifikáciu zdrojového `GameObject` v Unity editore. Voliteľné rozhranie `IFlushableLogger` označuje implementácie, ktoré vedú explicitne zapísať alebo dokončiť vypísanie čakajúcich záznamov. Kontrakt rozhrania je uvedený vo výpise 4.1.

Výpis 4.1: Rozhranie `ILogger` definujúce kontrakt logovacieho systému

```
using UnityEngine;
```

```
namespace Core.Diagnostics
```

```
{
```

```
    public interface ILogger {
```

```
        void Log(Diagnostics.LogChannel channel, string message,
               Object context = null);
```

```
        void Warning(Diagnostics.LogChannel channel, string message,
                    Object context = null);
```

```
        void Error(Diagnostics.LogChannel channel, string message,
                  Object context = null);
```

```
    }
```

```
    public interface IFlushableLogger {
```

```
        void Flush();
```

```
    }
```

```
}
```

Enum `LogChannel` definuje päť doménovo špecifických kanálov. V porovnaní s pôvodným `LogLevel` s jeho štyrmi hodnotami, ktoré reprezentovali iba filtrovaciu politiku, nový systém pridáva samostatný doménový rozmer. `LogLevel` zostáva použitý na runtime filtrovanie cez `GameLog.SetRuntimeLogLevel()`, `LogChannel` označuje podsystém. Definícia kanálov je uvedená vo výpise 4.2.

Výpis 4.2: Enum `LogChannel` s doménovo špecifickými kanálmi

```
namespace Core.Diagnostics
{
    public enum LogChannel
    {
        General,
        System,
        Network,
        UI,
        Audio
    }
}
```

4.3.2 Fasáda `GameLog`

Aby volajúci kód nemusel poznať konkrétnu implementáciu loggera, `GameLog` dodáva statickú fasádu ([Facade](#) [22]) s verejnými metódami `Info()`, `Warning()`, `Error()`, `Exception()` a `Flush()`. Logovacie metódy formátujú správu do tvaru `[ClassName.MemberName] message` — atribút `[CallerMemberName]` automaticky zaznamená meno volajúcej metódy bez réžie reflexie za behu. Ak nie je dostupný `Unity Object` kontext, trieda sa odvodí z `CallerFilePath`.

Inicializácia je chránená zámkom `SyncRoot`. Prebieha automaticky pred načítaním scény a zároveň lenivo pri prvom použití. Ak nebol logger explicitne nastavený volaním `Initialize()`, vytvorí sa predvolená implementácia podľa typu [buildu](#). Príklad tejto vetvy je uvedený vo výpise 4.3.

Výpis 4.3: Automatická inicializácia loggera podľa typu buildu

```
private static ILogger CreateDefaultLogger() {
    #if UNITY_EDITOR
        return new UnityLogger();
    #else
        return new CompositeLogger(new FileLogger(), new UnityLogger());
    }
```

```
#endif
}
```

V editore sa používa `UnityLogger` — počas vývoja najužitočnejší výstup v Unity konzole. V player buildoch sa používa `CompositeLogger` zložený z `FileLoggera` a `UnityLoggera`, aby boli logy perzistentne dostupné na zariadení a súčasne viditeľné v konzole pri PC alebo serverovom behu. `GameLog` si zároveň uchováva referenciu na implementáciu `IFlushableLogger`, ak ju konkrétny logger podporuje.

Dôležitou súčasťou fasády je aj runtime filtrovanie. Informačné logy sú označené atribútom `Conditional`: v release player buildoch sa skompilujú iba vtedy, ak je definovaný symbol `ENABLE_LOGS`; vždy sú dostupné v editore, development buildoch a server buildoch. Varovania a chyby zostávajú bežnými metódami, pričom všetky tri úrovne ešte kontrolujú aktuálny `LogLevel`.

`GameLog` publikuje každý vytvorený záznam cez udalosť `EntryEmitted`. Tú využíva napríklad používateľské rozhranie `CustomLog`. Volanie subscriberov je obalené do `try-catch`, aby chyba v UI konzole neporušila samotné logovanie.

Samotné odoslanie záznamu do aktívnej implementácie je tiež chránené. Ak aktuálny logger vyhodí výnimku, `GameLog` ju zachytí a zapíše diagnostickú správu cez priamy fallback na `Debug.Log*`, prípadne `Console.Error`. Zlyhanie logovacieho backendu sa tak nemá ako propagovať do hernej logiky.

4.3.3 FileLogger: súborový zápis a flush stratégia

Spomedzi všetkých komponentov nového loggeru rieši `FileLogger` najkritickejší problém: spoľahlivý zápis na disk na zariadeniach Quest. Implementuje `ILogger` aj `IFlushableLogger`. V konštruktore vytvorí adresár `Logs/GameLog` pod `Application.persistentDataPath`, založí nový súbor s časovou pečiatkou a platformou v názve a zapíše úvodné riadky relácie (`productName`, platforma a verzia Unity).

Metóda `Write()` pridáva záznam do `List<string>` chránenej zámkom. Každý riadok obsahuje časovú pečiatku vo formáte ISO 8601, úroveň závažnosti, kanál, identifikátor vlákna a správu už obohatenú fasádou `GameLog` o triedu a metódu volajúceho. Prahový **flush** sa spustí po 20 čakajúcich záznamoch. Varovania a chyby volajú `Flush()` okamžite a na Androide alebo v batch režime sa flushuje každý zápis. Tým sa návrh vzdáva asynchrónneho I/O v prospech jednoduchosti a nízkeho rizika straty kritických udalostí.

Samotný `Flush()` skontroluje, či buffer nie je prázdny, spojí čakajúce riadky

a vykoná `File.AppendAllText()`. Buffer sa vymaže až po úspešnom zápise. Pri zlyhaní zápisu sa chyba odošle do Unity konzoly alebo na `Console.Error`; čakajúce záznamy v pamäti zostávajú zachované pre ďalší pokus.

Po úspešnom zápise `FileLogger` vynucuje limit veľkosti log adresára. Predvolená hodnota je 64 MB a hodnota menšia alebo rovná nule kontrolu vypína. Implementácia sčíta veľkosť všetkých *.log súborov v adresári aktuálneho loggera a pri prekročení limitu maže najstaršie súbory. Aktuálny súbor relácie sa zámerné nemaže, aby rotácia neodstránila práve zapisovaný cieľ.

Životný cyklus rieši `GameLog`, nie samotný `FileLogger`. Fasáda vytvorí skrytý `GameObject` s komponentom `GameLogLifecycleHooks` a registruje flush pri `Application.quitting`, `Application.lowMemory`, `AppDomain.ProcessExit`, `OnApplicationPause`, `OnApplicationFocus`, `OnApplicationQuit`, `OnDisable` a `OnDestroy`. Násilné ukončenie procesu nemusí zavolať žiadny callback; preto je dôležitý aj režim okamžitého zápisu na Androide. Tok zápisu, flushu a správy úložiska sumarizuje obrázok 4.3.

Tabuľka 4.2 porovnáva kľúčové vlastnosti pôvodného a nového systému.

4.3.4 UnityLogger: konzolový výstup

`UnityLogger` je tenký, bezstavový adaptér nad natívnym API herného enginu.

Trieda `UnityLogger` implementuje rozhranie `ILogger` a deleguje volania na `Debug.Log`, `Debug.LogWarning` a `Debug.LogError`. Pred odoslaním správy do konzoly ju obohatí o farebné značky pomocou Unity Rich Text syntaxe. Každý `LogChannel` je priradený k odlišnej farbe. Propagácia parametra `Object context` do Unity API umožňuje po kliknutí na log v konzole zvýrazniť príslušný `GameObject` v hierarchii scény. `UnityLogger` je predvolenou implementáciou v editore; v player buildoch sa namiesto neho predvolene používa ako jeden z výstupov `CompositeLogger`.

4.3.5 CompositeLogger: kombinácia výstupov

`CompositeLogger` implementuje rovnaké rozhranie ako ostatné adaptéry a deleguje každé volanie na zoznam podriadených loggerov. V player buildoch tým spája `FileLogger` a `UnityLogger`. Trieda implementuje aj `IFlushableLogger`; pri `Flush()` zavolá iba tie podriadené implementácie, ktoré toto rozhranie podporujú.

Každý podriadený logger je volaný samostatne v bloku try-catch. Zlyhanie jedného výstupu preto nezastaví zápis do ďalšieho výstupu. Diagnostika zlyha-

Tabuľka 4.2: Porovnanie kľúčových vlastností pôvodného a nového loggera

Vlastnosť	Pôvodný LogHandler	Nový GameLog + FileLogger
Flush stratégia	Limit 10 000 správ alebo OnApplicationQuit	Prah 20 správ, okamžitý zápis varovaní/chýb, každý zápis na Androide, explicitné lifecycle flush hooky
Thread-safety	Žiadna synchronizácia	lock + List<string>
Formát záznamu	Jednoduchý reťazec	ISO 8601 timestamp, level, kanál, thread id, volajúca trieda a metóda
Úložiskový limit	Žiadny	64 MB predvolený limit, mazanie najstarších *.log súborov okrem aktuálneho
Ochrana pred zlyhaním	Žiadna	Zachovanie bufferu pri zlyhaní zápisu, guarded dispatch, guarded publikovanie UI udalostí
Reentrancia	Debug.Log vo write path	Priamy zápis cez File.AppendAllText()
Inicializácia	Závislá na SettingsManager	Automatická inicializácia pred scénou a lazy fallback
API konzistencia	Zmiešané (161 + 89 callsites)	Jednotná fasáda GameLog

nia sa posiela priamo do Unity konzoly alebo `Console.Error`, aby sa nevracala späť do tej istej logovacej pipeline.

4.3.6 Integrácia naprieč aplikáciou

Prechod na nové API ilustruje nasledujúci príklad z WebSocketovej vrstvy (kapitola 3), prvého subsystému plne integrovaného s novým logovaním. Ukážka je uvedená vo výpise 4.4:

Výpis 4.4: Porovnanie pôvodného a nového spôsobu logovania

```
// Pred (CustomDebug):
customDebug.LogWarning("Connection failed");

// Po (GameLog):
GameLog.Warning(LogChannel.Network,
    "Connection failed", this);
```

Nový prístup pridáva tri kvalitatívne zmeny oproti pôvodnému API:

- doménový kanál Network umožňuje filtrovanie logov podľa subsystému,
- parameter `this` automaticky zachytí názov triedy a vďaka atribútu `[CallerMemberName]` aj meno metódy,
- výsledný záznam v súbore má štruktúrovanú podobu:

Výpis 4.5: Príklad štruktúrovaného záznamu nového loggeru

```
2025-01-15T14:23:01.456+01:00 [WARNING] [Network] [Thread:1]
[WebSocketManager.OnClientDisconnected] Connection closed: KeepAlive timeout
```

Migrácia pôvodných logovacích volaní bola v aplikačnom kóde dokončená: `CustomDebug` a `LogHandler` boli odstránené a používanie sa zjednotilo na fasáde `GameLog`. Priame volania `Debug.Log*` zostávajú iba vo vnútri logger adaptéra a v ochranných chybových cestách, kde slúžia ako posledná diagnostická možnosť.

4.4 Výsledok a prenositeľnosť do „V2“

Prepracovanie logovacieho systému prinieslo dve skupiny výsledkov. Prvou bolo okamžité prevádzkové zlepšenie, overené na reálnom nasadení.

4.4.1 Validácia na cieľovej platforme

Najdôležitejším praktickým prínosom bolo obnovenie dostupnosti diagnostických dát na zariadeniach Meta Quest. Logovanie na Quest bolo v pôvodnom systéme nefunkčné (sekcia 4.1), čo vynútilo diagnostiku štýlom pokus a omyl.

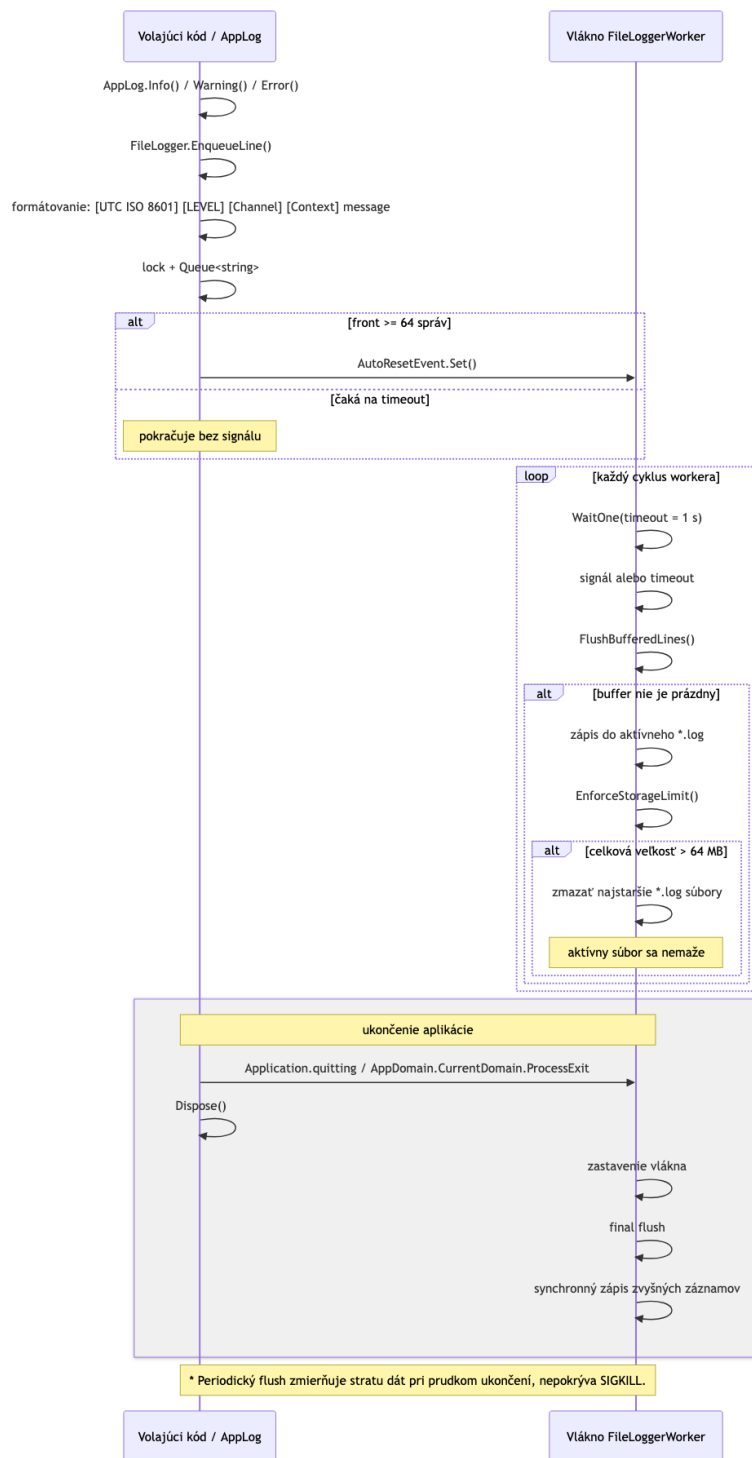
Po nasadení nového logovacieho systému produkovali zariadenia Quest log súbory pri každom zaznamenanom testovanom nasadení počas spolupráce so SAV, oproti nulovému výstupu pôvodného systému na rovnakej platforme. Pri nahlásení problému je teraz možné pripojiť zariadenie a extrahovať logy. Štruktúrované záznamy s časovými pečiatkami, kanálmi a kontextom volajúceho sú navrhnuté tak, aby umožňovali identifikáciu príčiny priamo zo záznamu, ako ilustruje príklad vo výpise 4.5. Prahový **flush**, okamžitý zápis varovaní a chýb a zápis každej správy na Androide zabezpečujú, že záznamy sa dostávajú na disk priebežne počas sedenia, nie až pri ukončení aplikácie.

4.4.2 Prevádzkové zlepšenia

Okrem obnovenia dostupnosti logov nový systém prináša ďalšie prevádzkové zlepšenia. Výnimky sa zaznamenávajú cez `exception.ToString()`, takže sa ne stráca stack trace. `EntryEmitted` umožňuje zobrazovať logy aj v používateľskom rozhraní bez toho, aby UI komponenty priamo zapisovali na disk. Lifecycle hooky rozširujú finálny zápis za hranicu samotného `OnApplicationQuit()` a zachytávajú aj pauzu aplikácie, stratu focusu, nízku pamäť a ukončenie procesu.

Druhou skupinou výsledkov je strategická hodnota pre verziu „V2“. Najviac prenositeľná je verejná štruktúra riešenia: fasáda, kontrakt `ILogger`, `IFlushableLogger` a doménové kanály. Samotné implementácie sú však stále previazané s Unity cez `UnityEngine.Object`, `Application.persistentDataPath` a lifecycle callbacks, preto by pri prenose vyžadovali platformovo špecifické adaptéry.

Riešenie má aj obmedzenia. Politika maskovania citlivých dát zostáva na strane volajúceho kódu a nie je vynucovaná jadrom loggeru. Výstupný formát je štruktúrovaný text, nie plnohodnotná schéma **JavaScript Object Notation (JSON)**. Súborový zápis je synchronný; pri veľmi častom logovaní preto môže stále predstavovať I/O réžiu. Retenčná politika pracuje s celkovým limitom adresára, ale nerotuje aktuálny súbor podľa veľkosti počas aktívnej relácie. Okrem toho `SettingsManager` po explicitnom `GameLog.Flush()` stále ukončuje player build volaním `Process.GetCurrentProcess().Kill()`, takže táto časť životného cyklu ostáva samostatným technickým dlhom.


Obrázok 4.3: Tok zápisu, flushu a správy úložiska v triede `FileLogger`

5 Používateľské vylepšenia systému

Predchádzajúce dve kapitoly opísali prácu na infraštruktúrnych vrstvách systému — WebSocketovej komunikácii a logovacím systéme. Pre pacienta a terapeuta zostali tieto zmeny neviditeľné; vytvorili však priestor pre úpravy, ktoré sa ich dotýkajú priamo. Práve také zásahy zoskupuje táto kapitola: integráciu rozhrania mozog-počítač (BCI) a elektroencefalografie (EEG), systém avatarov a inverznej kinematiky, stavový automat cvičenia a sériu menších, no používateľsky relevantných používateľských zlepšení. Väčšina z nich vznikala priebežne ako reakcia na požiadavky SAV a spätnú väzbu zo sedení.

5.1 EEG/BCI integrácia

Po úprave WebSocketovej vrstvy opísanej v kapitole 3 bolo možné doplniť nadväzujúce BCI funkcie. V pôvodnom stave projektu síce existovalo základné prepojenie medzi Unity a OpenViBE, no v praxi podporovalo iba obmedzený prenos správ a bolo náchylné na zlyhania. Požiadavky výskumníkov SAV sa pritom netýkali iba samotného prenosu správ, ale aj toho, aby sa dôležité EEG/MI hodnoty a stavové informácie dostali priamo do terapeutického rozhrania aplikácie Unity.

5.1.1 Od jednosmernej komunikácie k obojsmernému protokolu

Obmedzenia pôvodného jednosmerného protokolu sú analyzované v sekcii 2.4.1 a sumarizované v tabuľke 2.2. Výsledkom práce bolo rozšírenie komunikácie medzi Unity a OpenViBE o dva nové dátové toky:

- **Unity → OpenViBE:** nastavovanie threshold hodnoty motor imagery v reálnom čase cez správu `mi.threshold.set`, bez potreby reštartu OpenViBE.
- **OpenViBE → Unity:** priebežné posielanie hodnoty vypočítanej z EEG a motorickej imaginácie (MI) do terapeutického rozhrania cez správu `mi.stream.fatigue`.

Pôvodné WebSocketové príkazy zároveň dostali v novom aplikačnom programovacom rozhraní (API) jednoznačné názvy udalostí. Napríklad pôvodný príkaz `move` bol premenovaný na `exercise.action.move`. Význam príkazu sa tým nezmenil, no nový názov jasnejšie vyjadruje jeho úlohu v protokole.

Po tejto zmene Unity a OpenViBE neposielajú iba jednosmerné príkazy. Vymieňajú si aj hodnoty potrebné na sledovanie stavu relácie. Porovnanie pôvodného a nového protokolu je uvedené v tabuľke 2.2 v analytickej kapitole. Nasledujúce podsekcie opisujú jednotlivé časti tejto zmeny.

5.1.2 Zjednotenie terminológie motorickej imaginácie

Pôvodný kód používal označenie `bci` ako všeobecný názov pre správy, handle-ry aj dátové toky, hoci výskumníci SAV pracovali konkrétne s **motor imagery**. Preto boli názvy zmenené tak, aby používali skratku `mi`. EEG klient má vo finálnom protokole typ `eeg`; streamy a threshold hodnoty sú pomenované podľa **motor imagery**. Tým sa odstránil rozdiel medzi technickými názvami v kóde a terminológiou výskumníkov SAV.

5.1.3 Dynamické nastavenie threshold hodnoty

Threshold hodnota **motor imagery** — hranica, pri ktorej sa neurálna aktivita počas fázy `Move` vyhodnotí ako pokus o pohyb — existovala v pôvodnom riešení iba na strane OpenViBE. Terapeut ju nemohol upraviť priamo z Unity, preto zmena hodnoty počas sedenia vyžadovala zastavenie aktuálneho cvičenia, manuálnu úpravu v OpenViBE, reštart OpenViBE a často aj opätovné nastavenie cvičenia v Unity. V praxi mohol tento postup trvať minúty — prerušovala sa plynulosť terapeutického sedenia.

Po úprave môže terapeut nastavovať hodnotu priamo v terapeutickom rozhraní. Posuvník v Unity odošle zvolenú hodnotu cez WebSocketovú správu `mi.threshold.set` a OpenViBE ju použije bez reštartu. Rovnaká správa sa používa aj pri prihlásení EEG klienta do miestnosti: server mu po pripojení odošle aktuálnu threshold hodnotu relácie.

Aktuálna hodnota je zároveň uložená v databáze, ktorú do systému doplnila súbežná práca ďalšieho študenta [13]. Nastavená threshold hodnota tak zostáva dostupná aj po reštarte servera.

5.1.4 Používateľské rozhranie pre threshold hodnotu

Do terapeutického rozhrania v aplikácii Unity bol pridaný ovládací prvok na nastavenie **MI** threshold hodnoty. Terapeut môže citlivosť doladovať počas relácie bez presúvania do externého nástroja. Rozhranie rozlišuje medzi hodnotou vybranou posuvníkom a hodnotou, ktorá sa už používa v OpenViBE. Posunutie posuvníka neodošle novú hodnotu EEG klientovi — hodnota sa odošle až po stlačení tlačidla *Send*, takže EEG zariadenie nie je zapažované pri každej zmene hodnoty a terapeut môže pred odoslaním skontrolovať, akú hodnotu posiela.

Hodnota sa zobrazuje v rozsahu od 0.00 do 1.00. Pri nižšej hodnote systém vyhodnotí prah ako dosiahnutý aj pri slabšej EEG aktivite. Vyššia hodnota vyžaduje výraznejšiu EEG aktivitu. Obrázok 5.1 ukazuje ovládací prvok priamo v terapeutickom rozhraní.

5.1.5 **Fatigue meter**

Výskumníci SAV požadovali zobrazenie priebežných EEG/MI hodnôt priamo v terapeutickom rozhraní. Časť týchto dát sa spracovávala na strane OpenViBE, no terapeut k nim nemal prístup.

Do terapeutického rozhrania bol doplnený ukazovateľ fatigue hodnoty pacienta. Zobrazuje priebežnú EEG/**MI** hodnotu prijatú v správe `mi.stream.fatigue`. Terapeut tak vidí aktuálny odhad únavy pacienta a môže rozhodnúť, či treba upraviť dĺžku odpočinku alebo citlivosť systému.

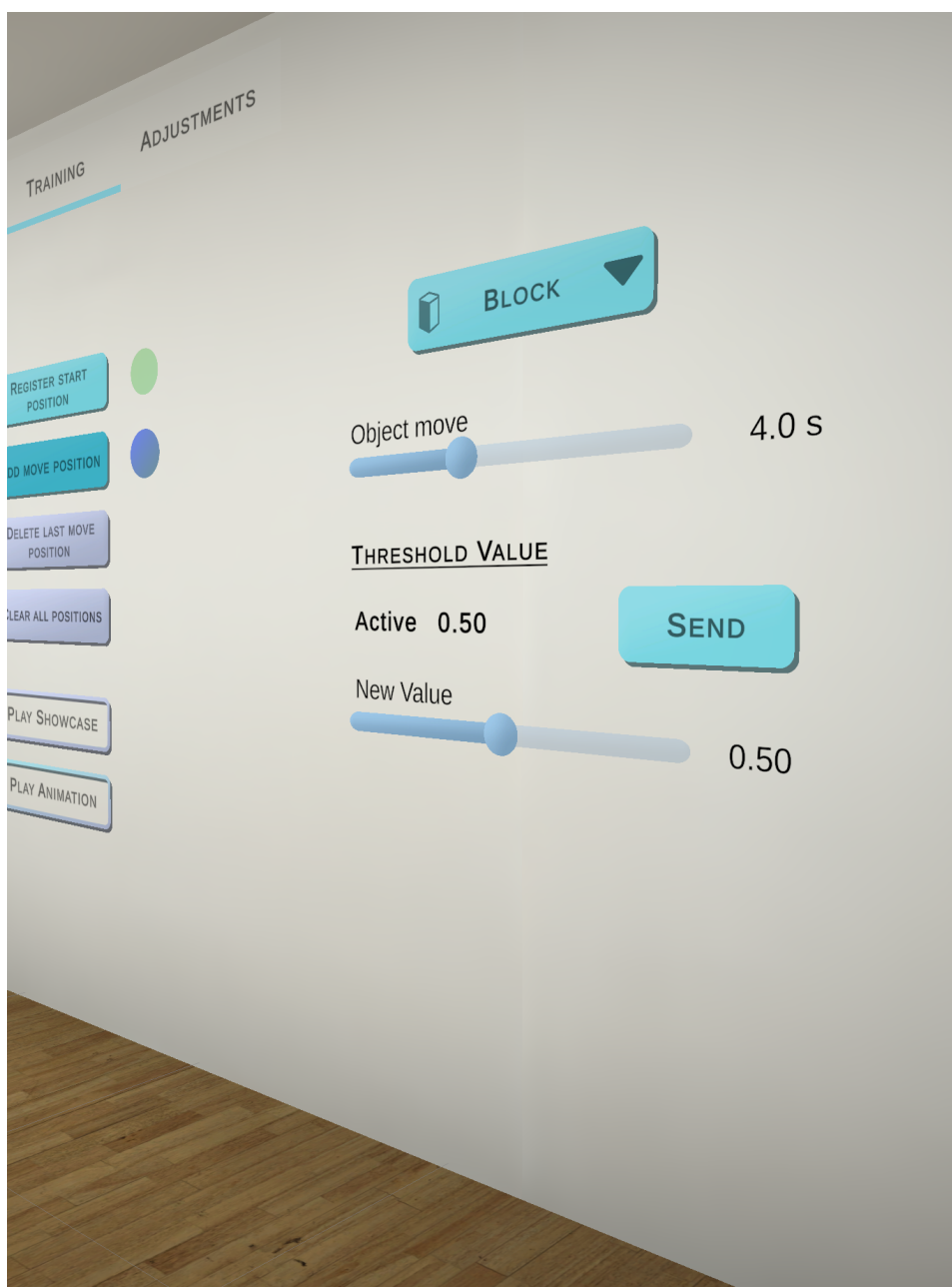
Rozhranie zobrazuje hodnotu v rozsahu od 0.00 do 1.00. Zvislý ukazovateľ sa vyplňa podľa aktuálnej hodnoty. Pri nízkych hodnotách je zelený, pri vysokých červený. Ukazovateľ zobrazuje aj posledné prijaté hodnoty, aby terapeut videl krátkodobý trend. Obrázok 5.2 ukazuje nízku a vysokú hodnotu únavy.

Po tejto úprave sa interné EEG/MI údaje v OpenViBE stali plnohodnotnou súčasťou celého systému. Terapeut pri cvičení nemusí prechádzať z **VR** do OpenViBE prostredia na inom desktopovom PC.

OpenViBE zostáva prostredím na prípravu a spúšťanie EEG **pipeline**; počas sedenia už slúži ako ovládacie rozhranie Unity.

5.1.6 Príkaz **move** v novom protokole

Pôvodný príkaz **move** má v novom protokole názov `exercise.action.move`. EEG/BCI strana ho po vyhodnotení úspešného **motor imagery** pošle aplikácii Unity a tým spustí krok cvičenia. Význam príkazu sa nezmenil; zmenil sa jeho názov a miesto v štruktúre protokolu.

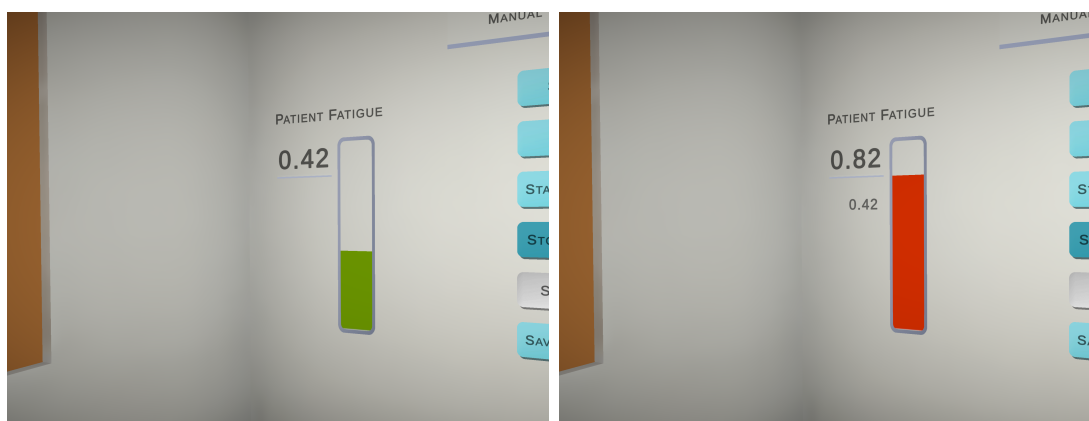


Obrázok 5.1: Nastavenie MI threshold hodnoty v terapeutickom rozhraní

Takáto zmena mala zmysel až po stabilizácii WebSocketovej vrstvy. Na pôvodnom nestabilnom kanáli by rovnaká funkcia pridala ďalšiu zložitosť bez istoty, že sa správa doručí spoľahlivo.

5.1.7 Synchronizácia stavu pre novoprihlásených klientov (Mirror **SyncVar**)

Okrem komunikácie medzi Unity a OpenViBE bolo potrebné opraviť aj synchronizáciu stavu medzi Unity klientom a serverom cez knižnicu Mirror. Ako opisuje



(a) Nízka hodnota únavy

(b) Vysoká hodnota únavy

Obrázok 5.2: Ukazovateľ únavy pacienta v terapeutickom rozhraní

analytická kapitola (sekcia 2.4.1), terapeut sa po pripojení do už prebiehajúcej relácie dostal k predvoleným hodnotám namiesto aktuálneho stavu sedenia.

Príčinou bolo nevyužitie mechanizmu [SyncVar] knižnice Mirror. Atribút [SyncVar] na poli triedy NetworkBehaviour hovorí Mirroru, aby hodnotu po pripojení nového klienta automaticky zosynchronizoval. Pôvodné stavové premenné relácie tento atribút nemali. Zmeny stavu sa posielali jednorazovými udalosťami, ktoré dostali iba klienti pripojení v čase zmeny.

Stavové premenné, ktoré musí vidieť aj neskôr pripojený klient, dostali atribút [SyncVar] a hook metódy aktualizujúce UI pri zmene hodnoty. Na strane WebSocketovej komunikácie server po úspešnom `client.checkin.ack` automaticky odosiela `mi.threshold.set` s aktuálnou threshold hodnotou miestnosti (kapitola 3). Vďaka kombinácii [SyncVar] vo vrstve Mirror a `replay` správ vo WebSocketovej vrstve dostane neskoro pripojený klient aktuálny stav relácie.

Terapeut sa mohol pripojiť v ľubovoľnom bode relácie a okamžite vidieť správny stav cvičebnej miestnosti — threshold hodnotu aj informáciu o EEG pripojení.

5.1.8 Výsledok oblasti EEG/BCI integrácie

Obojsmerný dátový tok medzi Unity a OpenViBE nahradil pôvodný jednosmerný protokol [12]. Terapeut získal prístup k EEG/MI hodnotám, možnosť meniť threshold hodnotu v reálnom čase a konzistentný stav pri pripojení do prebiehajúcej relácie. RefaktORIZOVANÁ WebSocketová vrstva nebola cieľom sama o sebe, ale bola potrebná na spoľahlivé doručenie týchto funkcií.

5.2 Systém avatarov a inverznej kinematiky

Analýza ukázala dve rôzne skupiny problémov v systéme avatarov. Prvou boli chyby, ktoré sa dali opraviť vo verzii „V1“: nesprávna výška kamery na desktope, nekonzistentné škálovanie avatara, umiestňovanie ruky pacienta a synchronizácia polohy. Druhou bolo obmedzenie použitých Ready Player Me modelov. Ich ruky nemali dostatočný detail ani počet kostí pre plnohodnotnú hand-tracking interakciu cez XR Hands. Výmena modelov presahovala rozsah tejto práce, preto nasledujúce podsekcie opisujú opravy realizované v rámci existujúceho systému avatarov a inverznej kinematiky.

5.2.1 Analýza chyby: kamera desktopového klienta pod úrovňou zeme

Pri prvom spustení desktopového klienta sa kamera objavila pod úrovňou zeme — scéna sa zobrazovala z nesprávnej výšky a bola prakticky nepoužiteľná.

Príčinou bola spoločná trieda AvatarController pre XR aj desktopový režim. Kód v nej počítal s tým, že klient má headset a vie z neho získať výšku používateľa. Desktop taký údaj nemá, preto chýbala predvolená výška a avatarOffset nebol správne inicializovaný.

Refactoring oddelil spoločnú časť do triedy AvatarControllerBase. Pre XR slúži trieda AvatarControllerXR.

Pre desktopový režim slúži trieda AvatarControllerDesktop.

Podobne bol rozdelený aj CharacterManager. Spoločná časť ostala v základnej triede. XR časť je v triede XRCharacterManager. Nové škálovanie používa VRIK z novej verzie FinalIK. Desktop už po úprave nepoužíva nastavenie určené pre headset; XR a desktopový režim sa nastavujú samostatne.

5.2.2 Prepracovanie škálovania avatara

Škálovanie avatara, evidované aj ako samostatná úloha, fungovalo prijateľne v XR režime, no na desktope bolo nekonzistentné. Ženské avatary mali navyše odlišné proporcie než mužské. Prepracovanie zahŕňalo:

- korektné fungovanie škálovania na desktopovom klientovi,
- zjednotenie proporcií ženských a mužských avatarov,
- úpravu polohy kamery desktopového klienta,

- inicializáciu hodnoty `Scale` na platný počiatočný stav,
- automatické nastavenie škály na XR zariadeniach,
- pridanie inkrementálneho škálovania cez tlačidlá,
- možnosť ručne prepísať škálovanie v offline scéne.

Po úprave terapeut vidí avatara v konzistentnejších proporciách a desktopový klient už nepoužíva XR inicializáciu.

5.2.3 Prepracovanie umiestňovania ruky pacienta

Pôvodný kód umiestňovania ruky pacienta na stôl kontroloval polohu ruky v každom snímku v metóde `Update()`. Stále zisťoval, či sa ruka nachádza v príslušnej zóne. Zbytočný **polling** a logika nebola priamo naviazaná na vstup ruky do zóny.

Nový kód používa **event-driven** prístup s **trigger** zónami. Ruka sa umiestni vtedy, keď avatar vstúpi do konkrétnej zóny. Zmena odstránila kontrolu v každom snímku a naviazala kód na konkrétnu udalosť v scéne.

5.2.4 Oprava synchronizácie polohy avatara vo **VR**

Ostatní účastníci relácie videli VR avatara pacienta v nesprávnej pozícii. Príčinou bola nesprávna poloha objektu `Offset`, referenčného bodu pri výpočte výslednej polohy. Po oprave sa poloha medzi klientmi synchronizuje spoľahlivejšie a terapeut vidí pacienta v zodpovedajúcej pozícii.

5.2.5 Aktualizácia FinalIK z verzie 2.3 na 2.4

FinalIK bol aktualizovaný z verzie 2.3 na 2.4. Novšia verzia obsahuje VRIK škálovanie použité pri oprave kamery desktopového klienta aj pri širšom prepracovaní škálovania. Aktualizácia zároveň znížila technický dlh, pretože projekt prestal používať staršiu verziu tejto knižnice.

5.2.6 Ďalšie opravy avatarov

Bolo opravené aj chybné prepínanie fixovanej ruky pacienta v scenári, keď mala zostať jedna ruka blokována. Táto chyba priamo narúšala vierohodnosť reprezentácie tela počas relácie.

5.2.7 Výsledok oblasti avatarov a inverznej kinematiky (IK)

Po úpravách funguje systém avatarov spoľahlivejšie na oboch platformách. Desktopový klient už nezlyháva pri prvom štarte, škálovanie je konzistentné a poloha avatara sa synchronizuje spoľahlivejšie. Keďže telo pacienta je vo VR rehabilitácii hlavnou časťou interakcie, tieto opravy priamo zlepšili priebeh relácie.

5.3 Stavový automat cvičenia

Ďalšou oblasťou zmien bol stavový automat cvičenia. Ako bolo uvedené už v analytickej kapitole, pôvodná verzia systému používala pevne danú štruktúru stavov. Aj keď táto štruktúra postačovala na základnú formu cvičenia, v rozvíjajúcom prostredí sa ukázalo, že niektoré prechody sú nedostatočne definované alebo chýbajú úplne.

5.3.1 Pridanie stavu Prepare

Do stavového automatu bol zavedený nový stav Prepare. Pôvodný tok Pause -> Rest -> Move -> Pause sa rozšíril na Pause -> Prepare -> Rest -> Move -> Pause.

Terapeut často potrebuje čas na potvrdenie pripravenosti pacienta, korekciu polohy alebo inštrukcie pred začiatkom merania — pacienta nemôžeme okamžite dať do fázy zberu **baseline** dať. Stav Prepare vytvoril samostatný krok medzi pripojením klienta a začiatkom rehabilitačného cyklu. Implementácia zahŕňala aj lokalizačné refazce a úpravy UI.

5.3.2 Opravy prechodov medzi stavmi

Opravený bol aj prechod Pause -> Rest, ktorý predtým nefungoval spoľahlivo. Kód už nedovolí prechod Pause -> Move, pretože pacient sa musí pred aktívnou fázou cvičenia dostať cez Pause a Rest fázu.

Súčasne bola opravená pozícia vzniku presúvateľného objektu. Po resete alebo novom cykle sa objekt konzistentne zjavuje na začiatkovej pozícii — opakované cvičebné cykly sú teraz konzistentné a predvídateľné.

5.3.3 Automatický režim cvičenia ako otvorený problém

Hoci aplikácia obsahovala manuálny aj automatický režim cvičenia, automatický režim v praxi iba preberal nastavenia manuálneho režimu a opakoval ich vo via-

cerých cykloch. Pre ďalej plánované rozšírené použitie to nestačí; implementácia neumožňuje vytvárať zmysluplné variabilné cvičebné plány.

Tento problém nebol v práci vyriešený zámerne. Na úplnú úpravu bolo potrebné najprv s terapeutmi určiť, aké automatické plány majú klinický zmysel, aké typy plánov majú existovať a ktoré parametre sa majú nastavovať. Bez tejto špecifikácie by ďalšia implementácia pridávala najmä kód, nie jasné zlepšenie. Z tohto dôvodu bola úplná revízia automatického režimu odložená do verzie „V2“.

5.4 Ďalšie používateľské a platformové zlepšenia

Popri veľkých oblastiach vznikli aj menšie úpravy. Každá riešila úzky problém, spolu však uľahčili každodenné používanie systému. Pre prehľadnosť sú rozdelené do troch skupín: platformové a **build** zmeny, úpravy interakcie a vizuálno-lokalizačné opravy.

5.4.1 Platformové a **build** úpravy

Predvolená schéma ovládania desktopového klienta sa pôvodne spúšťala s nastavením určeným pre XR simulátor namiesto štandardného ovládania myšou a klávesnicou. Po oprave sa desktopový klient správa ako bežná desktopová aplikácia bez potreby manuálnych zásahov po inštalácii.

Pribudla podpora macOS pre XR platformu a balíky boli upravené tak, aby sa XR automaticky nespúšťalo v desktopových a serverových buildoch. Pre používateľa bol priamy dopad menší, no na vývoj a testovanie bolo možné používať viac zariadení a mimo XR buildov ubudli problémy s nepotrebnou XR inicializáciou.

Do Meta Store buildu bolo pridané povolenie pre mikrofón potrebné pre hlasovú komunikáciu. Malá zmena, no bez nej hlasový kanál na cieľovej platforme nefungoval.

5.4.2 Interakcia a používateľský zážitok (**UX**)

Raycasty boli vypnuté na avataroch terapeutov. Túto úpravu vyžiadala SAV, pretože počas sedení bolo spätnou väzbou zistené, že raycasty terapeutov sú pre pacienta rušivé. Po oprave zostal pacientov vizuálny priestor čistejší a menej zaťažovaný prvkami, ktoré nesúviseli s jeho vlastnou interakciou.

Pacientovi pribudol vizuálny bod na sústredenie pohľadu a po spätnej väzbe pacienta zo zorného poľa zmizli rušivé prvky — okná, poličky s náhodnými ob-

jektmi. Pacient tým získal menší vizuálny šum počas úlohy kedy je maximálne sústredenie prioritou.

Opravené bolo aj načítanie zoznamu miestností, ktoré bolo v pôvodnej verzii pomalé a občas zlyhalo. Výber miestnosti patrí medzi prvé úkony pri štarte relácie; preto mala zmena priamy dopad na plynulosť spustenia systému.

5.4.3 Vizualne a lokalizačné úpravy

Posledná skupina zahŕňala opravy fontových assetov, vrstiev používateľského rozhrania, vyskakovacích okien a doplnenie prekladov pre nové prvky — stav Prepare, nastavenie threshold hodnoty, *fatigue meter*. Samostatne sú tieto zmeny malé, no spolu znížili počet rušivých nesúládov, ako nám bolo potvrdené zo strany SAV, s ktorými sa terapeut počas relácie stretával.

5.5 Zhrnutie kapitoly

Zmeny opísané v tejto kapitole ukazujú, že stabilizácia L-NeRVEn „V1“ nebola iba otázkou infraštruktúry. Popri WebSocketovej vrstve a logovacím systéme bolo potrebné riešiť aj tie časti systému, ktoré boli priamo viditeľné pre používateľov. BCI úpravy sprístupnili dôležité EEG/MI informácie priamo v terapeutickom rozhraní. Systém avatarov a *IK* odstránil chyby s priamym dopadom na použiteľnosť relácie. Stavový automat získal vhodnejší tok pre klinické použitie a menšie používateľské úpravy znížili počet každodenných prevádzkových prekážok. Spoločne tieto zásahy pomohli naplniť hlavný cieľ práce: udržať verziu „V1“ v dostatočne spoľahlivom stave pre pokračujúce výskumné využitie so SAV.

6 Nasadzovanie a **release** stratégia

Systém „V1“ nemal release stratégiu — **buildy** neboli verzionované, testerí nevedeli rozlíšiť, ktorú verziu aplikácie spustili, a pri nestabilnom builde neexistoval rýchly návrat k overenej verzii. Táto kapitola opisuje návrh trojkanálovej release stratégie, manuálny nasadzovací postup cez Meta Quest Developer Hub a prevádzkové poznatky získané z opakovaných nasadzovacích cyklov.

6.1 Stratégia **release** kanálov

6.1.1 Návrhové východiská

Pôvodný systém nemal systematickú stratégiu prevádzkového používania **buildov**. V klinickom prostredí nestačí mať „najnovší build“ — terapeut musí vedieť, ktorá verzia je stabilná, ktorá slúži na testovanie a ku ktorej sa možno okamžite vrátiť, ak sa objaví kritická chyba tesne pred sedením.

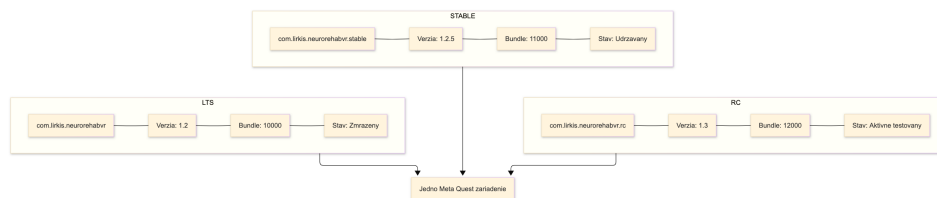
Preto bola navrhnutá trojkanálová **release** stratégia pozostávajúca z kanálov Long-Term Support (**LTS**), **STABLE** a **RC**. Kanál **LTS** predstavuje dlhodobu overenú záložnú verziu. Kanál **STABLE** slúži ako aktuálne udržiavaná stabilná vetva určená na bežné používanie. Kanál **RC** (*Release Candidate*) slúži na overovanie najnovších zmien — ich úplnú stabilitu ešte nemožno garantovať.

Kľúčovým návrhovým rozhodnutím bolo, že jednotlivé kanály používajú odlišné Android package name. Toto rozhodnutie malo priamy prevádzkový význam:

- jednotlivé varianty bolo možné nainštalovať súčasne na to isté zariadenie,
- testerí mohli medzi buildmi prepínať bez preinštalovania aplikácie,
- pri zlyhaní **RC** bolo možné okamžite prejsť na **STABLE**.

Návrh adresoval problém identifikovaný v analytickej kapitole 2: pri nestabilnej verzii neexistoval jasný **rollback** mechanizmus. Po zavedení kanálov sa

z buildov stali jasne rozlíšené varianty. Vzťah medzi jednotlivými kanálmi a ich samostatnými *app lines* sumarizuje obrázok 6.1.



Obrázok 6.1: Trojkanálová *release* stratégia systému NeuroRehabVR s oddelenými *app lines*

6.1.2 Definície kanálov

V čase finalizácie tejto práce bol stav *release* kanálov definovaný podľa tabuľky 6.1. Používané boli kanály Long-Term Support (*LTS*), *STABLE* a *RC*.

Tabuľka 6.1: Definícia release kanálov systému NeuroRehabVR

Kanál	Package Name	Verzia	Bundle Code	Stav
LTS	com.lirkis.neurorehabvr	1.2	10000	Zmrazený
STABLE	com.lirkis.neurorehabvr.stable	1.2.5	11000	Udržiavany
RC	com.lirkis.neurorehabvr.rc	1.3	12000	Aktívne testovaný

Popri package name bolo potrebné aj jednoznačné používateľsky viditeľné označenie *buildu*. Preto boli do názvu produktu zavedené prípony [*LTS*], [*STABLE*] a [*RC*]. Na zariadení sa buildy zobrazujú napríklad ako NeuroRehabVR [*STABLE*]. Pre klinickú prevádzku ide o podstatný detail: tester alebo terapeut okamžite vidí, ktorú verziu aplikácie spustil, a nevzniká priestor na omyl spôsobený tým, že viaceré buildy vyzerajú navonok identicky.

6.1.3 Schéma verziovania

Build má dve verzie. Prvou je číslo verzie viditeľné pre používateľa, napríklad 1.2.5 alebo 1.3. Druhou je interný Android *Bundle Version Code*, teda interné

rastúce číslo, podľa ktorého systém rozhoduje, či je nový build aktualizáciou danej aplikácie.

Pre dané package name musí Bundle Version Code vždy rásť. Kanály STABLE a RC používajú odlišné package name, preto sú ich sekvencie nezávislé. Prakticky to znamená, že pre kanál STABLE sa používa rada 11000, 11001, 11002, ... a pre RC rada 12000, 12001, 12002, ...

Výnimku tvorí kanál LTS. Ten je v súčasnosti zmrazený a nemožno ho nanovo zostaviť. Jeho verzia 1.2 a Bundle Code 10000 preto predstavujú archivovaný bod obnovy, nie živú release vetvu. Výnimka priamo súvisí so závažným praktickým zistením tejto kapitoly: stratou pôvodného podpisovacieho kľúča, ku ktorej sa viac píše v sekcii 6.4.

6.1.4 Politika *build promotion*

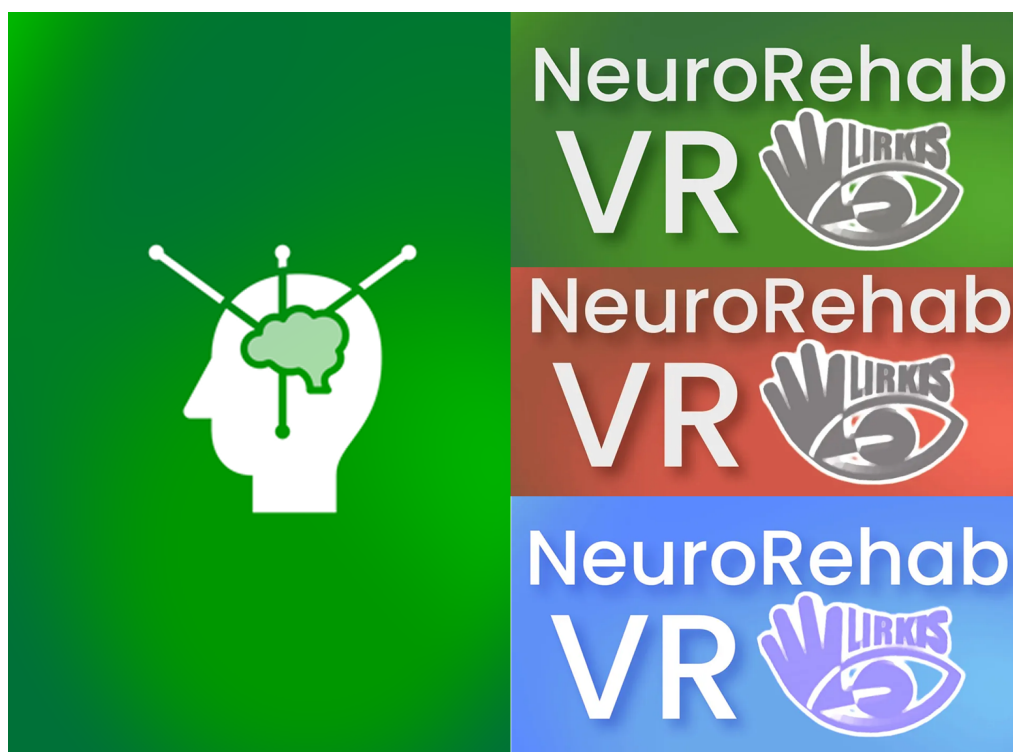
Samotná existencia kanálov by nemala veľký význam bez definovaného postupu, ktorý uvádza ako sa *build* presúva z testovacej vetvy do stabilnej. Praktická politika build promotion bola preto definovaná nasledovne:

1. nový build vzniká v kanáli RC,
2. build sa otestuje na cieľových zariadeniach,
3. ak sa objavia kritické chyby, vytvorí sa nový RC build s vyšším Bundle Code,
4. po úspešnom overení sa rovnaký stav kódu pripraví ako STABLE build s vlastným package name, product name a zvýšeným Bundle Code,
5. build sa nahrá do príslušného *release* kanála a doplnia sa release notes.

Kanál LTS sa v tomto procese nepoužíva ako bežná cieľová vetva, ale iba ako núdzová záloha.

6.1.5 Vizúálna identita

Prípony v názve produktu samé osebe nestačili na to, aby tester a terapeuti na prvý pohľad rozlíšili, s ktorým kanálom pracujú. Preto popri technickej časti *release* prípravy bola pripravená aj odlišená vizuálna identita aplikácie: každý kanál dostal vlastnú ikonu — spoločný grafický motív s farebne odlišeným pozadím, zeleným pre LTS, modrým pre STABLE a červeným pre RC. Pre systém prezentovaný terapeutom, pacientom a externému výskumnému partnerovi bolo toto vizuálne odlíšenie praktickou pomôckou pri každodennej prevádzke. Používané ikony a logá pre jednotlivé deployment varianty sú zobrazené na obr. 6.2.



Obrázok 6.2: Ikony a logá použité pre deployment varianty systému

6.2 Perzistencia balíkového a produktového mena

Zmena Android package name alebo product name môže pôsobiť ako drobná konfigurácia, no pri nasadzovaní na Meta Quest určuje identitu celej aplikácie. Ak sa package name nastaví nesprávne, nestačí build iba znovu nahráť. Celá aplikácia sa musí zostaviť znova so správnymi nastaveniami a v praxi to často znamenalo pripraviť aj nový desktopový a serverový build.

Riziko vyplýva zo spôsobu, akým Unity ukladá product name a Android package name do textových súborov ProjectSettings. V repozitári s viacerými vývojármi sú tieto súbory bežne predmetom zmien v merge request. Predvolené správanie systému Git sa na ne uplatňuje prístup *last-write-wins* (prevážia posledné zapísané zmeny). Pri mergovaní vetiev mohlo dôjsť k tichej strate správneho package name alebo product name.

Riešenie zaviedlo merge stratégie na úrovni `.gitattributes`, ktoré chránia kľúčové konfiguračné hodnoty pred nechceným prepísaním pri mergovaní. Použitá konfigurácia označila Unity súbor obsahujúci deployment identifikátory tak, aby sa pri integrácii vetiev zachovala lokálna verzia:

```
ProjectSettings/ProjectSettings.asset merge=ours
```

Cieľom nebolo blokovať merge ako taký, ale chrániť hodnoty priamo určujúce

identitu release kanála.

Úprava mala širší význam než zjednodušenie vývoja. Ak je nasadzovacia stratégia postavená na troch paralelných app líniach, potom perzistencia package name a product name nie je voliteľná, ale nevyhnutná podmienka správneho fungovania celej stratégie. Bez nej by bolo možné relatívne jednoducho vytvoriť build navonok vyzerajúci správne, no technicky patriaci do nesprávneho kanála.

6.3 Nasadzovací proces cez Meta Quest Developer Hub

Nasadzovanie *buildov* na Meta Quest zariadenia prebieha manuálne cez Meta Quest Developer Hub (*MQDH*). Dôležité bolo previesť tento postup do jasného a opakovateľného postupu, tak, aby build mohol pripraviť a nahráť ktokoľvek bez závislosti na vedomostiach niekoho iného.

Pred každým uploadom je potrebné overiť niekoľko predpokladov:

- správnu príponu product name ([LTS], [STABLE], [RC]),
- správne Android package name,
- správne viditeľné číslo verzie,
- inkrementovaný Bundle Version Code pre príslušnú package líniu,
- správny podpis buildu príslušným *keystore* pre daný kanál.

Samotný manuálny upload prebieha nasledovne:

1. otvoriť *MQDH* a prihlásiť sa Meta účtom s prístupom do správnej developer organizácie,
2. prejsť do sekcie *App Distribution* a zvoliť správnu organizáciu a aplikáciu,
3. vybrať cieľový *release* kanál,
4. kliknúť na *Upload* a zvoliť príslušný súbor .apk alebo .aab,
5. pred potvrdením overiť všetky identifikačné prvky definované v zozname vyššie,
6. dokončiť upload a počkať na spracovanie buildu,

7. overiť, že build sa objavil v správnom kanáli a že je možné ho nainštalovať alebo aktualizovať na zariadení.

Ak **MQDH** build odmietne kvôli zhodnému Bundle Version Code, postup je vždy rovnaký: treba zvýšiť Bundle Version Code, build znovu zostaviť a upload zopakovať. Tento krok síce znie triviálne, no v praxi patril medzi opakujúce sa príčiny zlyhania uploadu a bol dôvodom, prečo bolo verziovanie zahrnuté do každého pre-release checklistu.

6.4 Prevádzkové poznatky z platformy Meta Quest

Počas opakovaných nasadzovacích cyklov bolo objavených niekoľko nezdokumentovaných správaní platformy Meta Quest, ktoré prakticky ovplyvňujú vytváranie nových verzií aplikácie. Všetky zistenia boli zapísané ako dôsledok reálnych problémov počas nasadzovania a ich správanie ovplyvnilo schopnosť promptne pripraviť **build** na testovanie alebo na rýchlu nápravu zistenej chyby.

6.4.1 24-hodinový lockout účtu

Počas testovania sa ukázalo, že chyba *too many requests* sa môže objaviť aj pri bežnom prvom prihlásení, nie iba ako reakcia na akcie aktérov so zlým úmyslom. Nové nasadenie je preto potrebné plánovať s dostatočnou rezervou.

6.4.2 Cooldown po zmazaní aplikácie

Ďalším zistením bolo, že zmazanie aplikácie z developer organizácie v praxi neznamená okamžité uvoľnenie package name na opätovné použitie. Po zmazaní nasleduje obdobie, počas ktorého nie je možné danú identitu znovu použiť na nový **app line**. V praxi je zmazanie aplikácie prakticky ireverzibilný úkon s one-skoreným dopadom.

Zistenie zásadne ovplyvnilo návrh **release** stratégie. Namiesto prístupu „vymazať starú verziu a nahradiť ju novou“ je bezpečnejšie uchovávať historické app lines a pracovať so zmrazenou záložnou vetvou. Preto bol kanál LTS navrhnutý ako trvalá záloha komplimentujúca viac meniace sa kanály STABLE a RC.

6.4.3 SMS verifikácia pri každom uploade

Každý upload **buildu** môže vyžadovať SMS verifikačný kód — Vzniká závislosť na okamžitom prístupe k telefónnemu číslu člena tímu a predstavuje to aj prí-

Tabuľka 6.2: Prevádzkové poznatky o platforme Meta Quest

Poznatok	Príčina	Dopad	Odporúčaný postup
24-hodinový lockout účtu	chyba <i>too many requests</i> aj po jedinom úkone	úplné zablokovanie nasadzovania na 24 hodín	vyčkať, neeskalovať uploady rýchlymi opakovaniami
Cooldown po zmazaní aplikácie	odstránenie <i>app line</i> z developer organizácie	nemožnosť znovu použiť rovnaké package name niekoľko dní	aplikácie nemažte, ponechajte ako zmrazený <i>fallback</i>
SMS verifikácia pri každom uploade	každý upload buildu	spomalenie postupu, nutnosť mať okamžitý prístup k telefónu	plánovať čas na verifikáciu
Väzba na Android <i>keystore</i>	podpisovací kľúč je viazaný s package name	strata kľúča znamená nemožnosť aktualizovať daný <i>app line</i>	povinné bezpečné uloženie a dokumentácia <i>keystore</i> údajov

padnú prekážku potenciálnej budúcej automatizácie cez kontinuálnu integráciu a kontinuálne nasadzovanie (*CI/CD*).

6.4.4 Väzba na Android *keystore*

Android aplikácia je kryptograficky viazaná na podpisovací *keystore*: bez správneho kľúča nemožno publikovať update pre daný package name. Práve strata pôvodného *keystore* — dôsledok chýbajúcej dokumentácie a nezabezpečeného uloženia — zmrazila kanál LTS (sekcia 6.1). Od tohto momentu sa správa *keystore* stala formálnou politikou: súbory, aliasy a heslá musia byť evidované v bezpečnom úložisku spolu s informáciou, ku ktorému *app line* patria. Pre budúcu verziu „V2“ ide o jedno z hlavných procesných odporúčaní — riadna správa kľúčov musí existovať od prvého dňa vývoja.

6.5 Výsledok kapitoly

Trojkanálová *release* stratégia, dokumentovaný *MQDH* postup a prevádzkové pravidlá zmenili nasadzovanie z improvizovaného kroku na riadený proces. Testeri identifikujú verziu podľa názvu aplikácie, *rollback* je jednoduchý — stačí spustiť funkčne overenú STABLE verziu aplikácie a kanál LTS zostáva k dispozícii ako núdzová záloha. Opísaná politika správy release kanálov, *keystore*, je priamo prenositeľná do verzie „V2“.

7 Hodnotenie

Predchádzajúce kapitoly prezentovali návrh a implementáciu jednotlivých príspevkov. Táto kapitola sa vracia k riešeným problémom z úvodu (sekcia 1.3.1) a k odvodeným požiadavkám (sekcia 2.6) a konfrontuje ich so získanými výsledkami. Hodnotenie sa opiera o dva hlavné ciele práce: udržanie prevádzkovej stability „V1“ pre sedenia SAV a dodanie prenositeľnej infraštruktúry pre „V2“. Záverečné sekcie diskutujú obmedzenia zvoleného spôsobu overovania a otvorené problémy.

7.1 Metodika overovania

Overovanie prebiehalo na troch úrovniach s rastúcou mierou validity:

1. **Vývojárske testovanie** — manuálne overenie na cieľových platformách (Meta Quest, desktop Windows, macOS). WebSocketová vrstva bola testovaná vlastným Python klientom simulujúcim OpenViBE správy a nástrojom Postman.
2. **Spoločné debugovacie sedenia so SAV** — integračné overenie s reálnym OpenViBE klientom a **pipeline** elektroencefalografie (**EEG**). Pri týchto sedeniach sa podarilo reprodukovat' a opraviť chybu odpájania (sekcia 3.2).
3. **Výskumné sedenia SAV** — nasadenie v reálnej praxi s pacientmi. Ide o najsilnejšiu formu validácie; autor však nebol priamo prítomný pri sedeniach.

Overenie nasadzovania prebehlo opakovanými cyklami vytvorenia a nahratia **buildov** cez Meta Quest Developer Hub (sekcia 6.3); pri každom cykle sa kontroloval package name, product name, verzia, Bundle Version Code a podpis príslušným **keystore**.

Ohraničenia metodiky. Práca neobsahuje formálnu používateľskú štúdiu s pacientmi ani terapeutmi, kvantitatívne výkonnostné benchmarky ani automatizo-

vanú regresiovú sadu testov. Nejde o klinickú štúdiu, ale o stabilizáciu výskumného prototypu — výsledky majú oporu v praktickom použití vybraných funkcií, spoločnom debugovaní so SAV a manuálnom overovaní autorom. Počas vývoja nebol vedený presný register počtu sedení na strane SAV, počtu nahratých buildov ani počtu relácií, v ktorých vznikli logy na zariadeniach Meta Quest. Hodnotenie preto neformuluje kvantitatívne tvrdenie o celkovej spoľahlivosti systému. Dôsledky tohto ohraničenia rozoberá sekcia 7.7.

7.2 P1 — WebSocketová vrstva

Prepracovanie WebSocketovej vrstvy medzi Unity a OpenViBE tak, aby boli odstránené kritické výpadky spojenia a vrstva zostala použiteľná aj vo „V2“.

7.2.1 Odstránenie odpájania

Príčina a oprava chyby náhodného odpájania sú popísané v sekcii 3.2.

Po nasadení opravy (zvýšenie KeepAlive timeoutu z 1 s na 10 s a presun baseline výpočtu na samostatné vlákno v OpenViBE) nebol autorovi znovu hlásený pôvodný typ náhodného odpájania WebSocket klienta. Presný počet následných sedení však nebol systematicky evidovaný, preto sa tento výsledok hodnotí ako praktické potvrdenie odstránenia reprodukovateľného problému, nie ako štatistické meranie dostupnosti komunikačnej vrstvy.

7.2.2 Architektonická kvalita

Kvalitu hexagonálneho návrhu potvrdili dve nezávislé udalosti:

1. **Migrácia knižnice.** Pôvodná websocket-sharp bola najprv nahradená knižnicou WatsonWebSocket, tá však zlyhávala v runtime prostredí Unity Mono. Konečná migrácia na Fleck prebehla výhradne na úrovni transportného adaptéra — zmena knižnice si nevyžiadala úpravu ani jedného handlera (sekcia 3.4).
2. **Rozšírenie protokolu o udalosti rozhrania mozog-počítač (BCI).** Nové typy správ (`mi.stream.fatigue`, `mi.threshold.set`, `exercise.action.move`) boli pridané vytvorením handlerov a ich registráciou v MessageRouter bez modifikácie existujúceho smerovacieho kódu (kapitola 5).

Bezpečný prechod medzi vláknami rieši vzor **Producer–Consumer**: transportná vrstva vkladá akcie do `ConcurrentQueue<Action>`, hlavné vlákno Unity ich vyberá v `Update()` s rozpočtom 5 ms a limitom 500 akcií na snímok (sekcia 3.5).

7.2.3 Prenositelnosť do „V2“

Celý sieťový podsystém existuje vo vlastnej definícii zostavenia `NeuroRehab.Networking` a kompiluje sa nezávisle od zvyšku projektu. Do „V2“ bol prenesený výhradne zmenou *namespace*. Rozhrania `INetworkTransport` a `IOutboundMessenger` umožňujú mock testovanie aj bez Unity enginu.

7.3 P2 — Diagnostikovateľnosť a nasaditeľnosť

Zabezpečenie diagnostikovateľnosti a stabilného nasadenia systému L-NeRVEn „V1“ v klinickom prostredí prostredníctvom architektonických a prevádzkových zásahov.

7.3.1 Logovací systém

Dostupnosť logov. Pôvodný logger na zariadeniach Meta Quest systematicky strácal záznamy (sekcia 4.1). Nový `FileLogger` zapisuje na disk pri naplnení bufferu 20 správ, okamžite pri varovaniach a chybách a na Androide alebo v batch režime pri každej správe. Logger bol manuálne overovaný autorom a ďalším študentom v opakovaných reláciách s rolami terapeut a pacient. Pri týchto overeniach zariadenia Quest produkovali log súbory, zatiaľ čo pôvodný systém na rovnakej platforme systematicky zlyhával. Nešlo však o dlhodobé monitorovanie všetkých zariadení v produkčnej prevádzke.

Kvalita diagnostických záznamov. Rozdiel medzi pôvodným a novým logovaním zachytáva tabuľka 7.1.

Retrospektíva: diagnostika chyby odpájania. Diagnostika chyby odpájania (sekcia 3.2) s pôvodným loggerom vyžadovala zachytávanie sieťových paketov nástrojom Wireshark počas spoločného debugovacieho sedenia. S novým loggerom by záznam `[WARN] [Network] Connection closed: KeepAlive timeout` identifikoval príčinu priamo — bez analýzy na paketovej úrovni.

Tabuľka 7.1: Porovnanie diagnostickej kvality pôvodného a nového logovacieho systému.

Vlastnosť	Pôvodný logger	Nový logger
Časová pečiatka (ISO 8601)	nie	áno
Úrovně závažnosti	4 (NONE, ERROR, WARNING, ALL)	3 (LOG, WARN, ERROR)
Doménové kanály	nie	5 (General, System, Network, používateľské rozhranie (UI), Audio)
Kontext volajúceho (trieda, metóda)	nie	áno
Spoločný flush na Quest	nie	áno (20 správ, warning/error okamžite, Android každý zápis)
Rotácia logov	nie	áno (64 MB limit)

Pokrytie a prenositeľnosť. Aplikačný kód používa jednotnú fasádu GameLog; pôvodné volania `customDebug.*` a `LogHandler` už v aktuálnom zdrojovom kóde nezostávajú. Priame volania `Debug.Log*` zostávajú iba vo vnútri logger adaptéra a ochranných fallback ciest. Jadrové komponenty loggeru prešli do „V2“ len so zmenou *namespace*; ide o architektonické overenie prenositeľnosti, nie o plné prevádzkové overenie loggeru vo verzii „V2“.

7.3.2 Nasadzovacia a release stratégia

Pred zavedením stratégie chýbalo verzionovanie **buildov** — tester nevedeli zistiť, ktorú verziu používajú, a pri nestabilnom builde nemal nikto ako sa vrátiť k predchádzajúcej (sekcia 2.4.4). Zavedené mechanizmy a spôsob ich overenia zachytáva tabuľka 7.2.

Všetky tri mechanizmy — stratégia, ochrana identifikátorov aj politika keystore — sú priamo použiteľné aj vo verzii „V2“.

7.4 P3 — Používateľské úpravy

Identifikácia a realizácia používateľských úprav s najvyšším praktickým prínosom pre prebiehajúce sedenia SAV pri rešpektovaní obmedzení prechodnej verzie „V1“.

Tabuľka 7.2: Zavedené prevádzkové mechanizmy a spôsob ich overenia.

Mechanizmus	Problém, ktorý rieši	Overenie
Trojkanálová stratégia (LTS , STABLE, RC)	Chýbajúce verzie a roll-back	Súčasná inštalácia na Quest overená opakovanými cyklami
Ochrana identifikátorov cez <code>.gitattributes</code>	Tiché prepísanie package name pri merge	Lokálne deployment identifikátory zachované pravidlom <code>merge=ours</code>
Politika keystore	Strata podpisového kľúča zmrazila LTS	Kľúče evidované v bezpečnom úložisku
Zdokumentovaný MQDH postup	Improvizácia pri nasadzovaní	Postup použitý pri manuálnych nariadeniach

7.4.1 BCI integrácia: obojsmerný dátový tok

Dátové toky zavedené v rámci tejto práce popisuje sekcia 5.1.1. Pre sedenia SAV priniesli najmä:

- **Zmena threshold hodnoty motor imagery (MI) v reálnom čase** — terapeut nastavuje threshold hodnotu priamo z Unity **UI** a zmena sa okamžite propaguje cez `mi.threshold.set` bez reštartu OpenViBE (sekcia 5.1). Predtým bolo nutné cvičenie zastaviť a zmenu verbálne koordinovať s EEG operátorom.
- **Fatigue meter** — priebežne zobrazuje EEG/MI hodnotu zo správy `mi.stream.fatigue`; doteraz bol tento údaj viditeľný len na strane OpenViBE.

7.4.2 Synchronizácia stavu pre novoprihlásených klientov

Pôvodný problém nekonzistentného stavu po pripojení terapeuta do bežiacей relácie (sekcia 2.4.1) rieši kombinácia `[SyncVar]` s hook metódami (sekcia 5.1.7)

a automatického odoslania `mi.threshold.set` po `client.checkin.ack`. Terapeut sa tak pripojí kedykoľvek počas relácie a okamžite vidí aktuálny stav cvičenia, `threshold` hodnotu aj informáciu o EEG pripojení.

7.4.3 Systém avatarov a inverznej kinematiky

- Kamera desktop klienta pod úrovňou zeme pri prvom spustení bola odstránená refactoringom hierarchie tried (sekcia 5.2).
- Škálovanie avatarov zjednotené na oboch platformách (desktop a režim rozšírenej reality (XR)).
- Umiestňovanie ruky pacienta prešlo na **event-driven** model s **trigger** zónami; zbytočný per-frame **polling** odpadol.
- FinalIK aktualizovaný z verzie 2.3 na 2.4.

7.4.4 Stavový automat cvičenia

Stav `Prepare` rozšíril stavový automat (sekcia 5.3). Prechod `Pause` → `Move` bol explicitne zakázaný ako klinicky neplatný. Oba prechody boli overené počas sedení.

7.4.5 Ďalšie používateľské vylepšenia

Predvolenú schému ovládania desktop klienta nahradilo štandardné ovládanie myšou a klávesnicou. Pre pacienta pribudol vizuálny fokusový bod; rušivé prvky v zornom poli boli odstránené. Vypnutie **raycastov** na ne-lokálnych avataroch zamedzilo chybným interakciám v zdieľanom priestore. Lokalizačné reťazce pokrývajú všetky nové prvky.

7.5 Splnenie odvodených požiadaviek

Požiadavky FR1 a NFR1 boli naplnené v rozsahu reprodukovateľného problému náhodného odpájania WebSocket klienta. Chyba bola počas spoločného debugovania so SAV reprodukováná, identifikovaná a opravená úpravou `KeepAlive` timeoutu a presunom blokujúceho výpočtu na samostatné vlákno. Po tejto úprave nebol autorovi znovu hlásený rovnaký typ incidentu. Keďže počet následných sedení nebol systematicky evidovaný, nejde o kvantitatívny dôkaz dostupnosti systému, ale o praktické overenie odstránenia konkrétnej príčiny výpadkov.

Požiadavky FR2, NFR2 a NFR3 boli overené architektonicky. Nová správa používa formálnu obálku s typom udalosti, časovou pečiatkou a štruktúrovaným payloadom (tabuľka 3.1). Výmena WebSocket knižnice zasiahla iba transportný adaptér, nie doménové handlery. Spracovanie správ medzi vláknami je riešené cez `ConcurrentQueue<Action>` a obmedzený Producer–Consumer dispečer v hlavnom vlákne Unity.

Požiadavky FR3 a FR4 boli naplnené implementačne a manuálne overené. Nový klient po pripojení získava aktuálny stav relácie kombináciou `Mirror [SyncVar]` mechanizmu a odoslania aktuálnej **MI** threshold hodnoty po `client.checkin.ack`. Terapeut môže meniť threshold hodnotu z Unity rozhrania bez reštartu OpenViBE alebo Unity relácie.

Požiadavky FR5 a NFR4 boli naplnené čiastočne. Nový logger poskytuje konzistentné úrovne, časové pečiatky, doménové kanály, kontext volajúceho a priebežný zápis na disk. Aplikačné logovanie bolo zjednotené na `GameLog API`. Logger bol overený manuálne v opakovaných reláciách s rolami terapeut a pacient, nie dlhodobým zberom prevádzkových logov zo všetkých zariadení.

Požiadavka NFR5 bola splnená architektonicky. WebSocketová vrstva a jadrové komponenty loggeru boli prenesené do „V2“ po zmene *namespace* a kompilovali sa v samostatných zostavách. Toto overenie potvrdzuje prenositeľnosť komponentov, nie úplné produkčné nasadenie všetkých funkcionalít vo verzii „V2“.

7.6 Diskusia k validite výsledkov

Dôkazy v tejto práci majú dva charaktery: **prevádzkový** (vybrané funkcionality boli použité v praxi a pôvodný hlásený incident sa autorovi znovu nevrátil) a **architektonický** (rozhrania dodržané, knižnica vymenená bez zásahu do domény). Pre stabilizáciu výskumného prototypu sú oba typy primerane výpovedné, no oba majú aj svoje hranice.

Vnútoraná validita. Bez automatizovaných testov sa správnosť opiera výlučne o manuálne testovanie a reálnu prevádzku. Regresné chyby tak nemusia byť zachytené okamžite.

Vonkajšia validita. Výsledky sa vzťahujú na konkrétnu kombináciu L-NeRVEn + OpenViBE + Unity 2022. Architektonické princípy (hexagonálny návrh, Producer–Consumer dispečer) sú prenositeľné do iných prostredí; konkrétna implementácia nie.

Ekologická validita. Väčšia časť overovania prebehla v reálnom prostredí SAV, nie iba v simulovaných podmienkach — ekologická validita je priamo zvýšená, no zároveň sa znížila kontrola nad podmienkami overovania. Bez presného záznamu počtu sedení, dĺžky používania jednotlivých funkcií a počtu nasadení nie je možné doložiť štatisticky podloženú mieru spoľahlivosti.

7.7 Obmedzenia a otvorené problémy

7.7.1 Problémy mimo rozsahu práce

Nasledujúce problémy boli zdokumentované, no ich riešenie presahovalo stanovený rozsah:

- **Upgrade XR knižníc** — aktuálna verzia XR Interaction Toolkit nepodporuje XR Hands a má nevyhovujúce mapovanie ovládania. Najmä kolízia uchopenia objektu a teleportácie na rovnakom trigger tlačidle by vyžadovala prechod na novší input systém XR Interaction Toolkit 3+ a zásah do systému avatarov. Plánované pre „V2“.
- **Náhrada avatar modelov** — modely Ready Player Me nemajú dostatočnú kostrovú kvalitu pre hand tracking. Viazané na XR upgrade.
- **Stabilita Mirror servera** — pády pri ukončení servera, nekonzistentné stavy relácií a súvisiace problémy nižšej runtime alebo knižničnej vrstvy vyžadujú hlbší zásah. Riešenie bolo predmetom samostatnej diplomovej práce [13].
- **Automatizácia pripojenia pacienta** — v tejto práci bola navrhnutá ako mechanizmus pre „V2“; implementovaná nebola. Databáza presetov a nastavení pre pacienta vznikla čiastočne v súbežnej práci [13].

7.7.2 Známe technické obmedzenia

- **Absencia Transport Layer Security (TLS)** — komunikácia prebieha nešifrovaným kanálom `ws://`. V lokálnej sieti akceptovateľné; pri vzdialenom nasadení nutné doplniť `wss://`.
- **Chýbajúca autentifikácia** — overenie identity sa spolieha na `roomId` a `clientType` v správe `client.checkin`.

- **Obmedzená správa log súborov** — migrácia na jednotné GameLog **API** je dokončená a adresár má 64 MB retenčný limit. Logger však nerotuje aktuálny súbor počas relácie podľa jeho veľkosti.
- **Textový formát logov** — pre automatizovanú analýzu by bola vhodnejšia strojovo čitateľná schéma **JavaScript Object Notation (JSON)**.

7.7.3 Chýbajúce evaluačné dáta

Táto práca sa zameriavala na stabilizáciu systému a jeho použiteľnosť v praxi, preto neobsahuje formálnu používateľskú štúdiu, výkonnostné benchmarky (latencia správ, snímková frekvencia, priepustnosť WebSocket kanála) ani automatizovanú regresnú sadu testov. Sekcia 7.6 preto špecifikuje, ako tieto skutočnosti limitujú interpretáciu dosiahnutých výsledkov.

8 Záver

8.1 Súhrn príspevkov

Práca sledovala dva ciele: stabilizovať systém L-NeRVEn „V1“ pre pokračujúce výskumné sedenia SAV a dodať prenositeľnú infraštruktúru pre verziu „V2“.

Náhodné odpájanie WebSocket klienta opakovane prerušovalo výskumné relácie. Príčinou bol KeepAlive timeout nastavený na jednu sekundu; po jeho zvýšení a presune **baseline** výpočtov na samostatné vlákno na strane SAV bola odstránená reprodukovaná príčina výpadkov (sekcia 3.2). Jednosmerný protokol [12] nahradil obojsmerný dátový tok medzi Unity a OpenViBE. Terapeut tak získal možnosť meniť threshold hodnotu motor imagery (**MI**) priamo za behu a sledovať priebežnú EEG/MI hodnotu v Unity (sekcia 5.1). Novoprihlásení klienti dostávajú konzistentný stav relácie vďaka [SyncVar] a replay mechanizmu na WebSocketovej vrstve (sekcia 5.1.7). V systéme avatarov a stavovom automate cvičenia boli opravené chyby a pribudol stav Prepare (kapitola 5). Neformálny nasadzovací proces nahradila trojkanálová **release** stratégia s ochranou deployment identifikátorov a politikou **keystore** (kapitola 6). Po nasadení opravy nebol autorovi znovu hlásený pôvodný typ náhodného odpájania WebSocket klienta; presný počet následných sedení však nebol systematicky evidovaný.

WebSocketová vrstva pre „V2“ stojí na hexagonálnej architektúre v samostatnej zostave NeuroRehab.Networking (kapitola 3). Izolácia sa overila neúmyselne — výmena WebSocket knižnice (websocket-sharp → WatsonWebSocket → Fleck) zasiahla iba transportnú vrstvu. Doménová logika zostala nedotknutá. Logovací systém prešiel kompletným prepisom; nový FileLogger produkuje štruktúrované záznamy s ISO 8601 pečiatkou a doménovým kontextom (kapitola 4). Oba komponenty fungujú vo „V2“ po zmene *namespace*. **Release** stratégia a prevádzkové poznatky o platforme Meta Quest sú rovnako prenositeľné.

Splnenie požiadaviek FR1–FR5 a NFR1–NFR5 je preto potrebné čítať v kontexte dostupných dôkazov. Komunikačná vrstva a používateľské zmeny boli overené praktickým použitím a manuálnym testovaním; architektonická prenositeľ-

nosť bola overená kompiláciou komponentov vo „V2“. Logovací systém zostáva dokončený v rozsahu tejto práce: aplikačné volania boli zjednotené na nové aplikačné programovacie rozhranie (API) a súborový logger dopĺňa retenčný limit adresára.

8.2 Strategický výsledok

L-NeRVEn „V1“ dosiahol hranice praktického rozširovania. Opravy prechádzali cez krehké centrálné triedy bez hraníc medzi doménovou logikou a infraštruktúrou. „V2“ nemohla začať, kým „V1“ zostávala nestabilná. Refaktorizácia celého prechodného systému nebola účelná (sekcia 2.7).

Zvolené boli preto zásahy s najvyššou kombinovanou hodnotou — klinickým prínosom pre prebiehajúce sedenia a architektonickou prenositeľnosťou do „V2“. Výsledkom nie je kompletne opravený systém. Sú to dva produkčné komponenty (WebSocketová vrstva a logovací systém), zdokumentovaná nasadzovacia stratégia a súbor klinických vylepšení overených v reálnej prevádzke. Výskum SAV pokračoval bez prerušenia. Vývojový tím „V2“ preberá funkčné komponenty a prevádzkové znalosti — nie prázdny základ.

8.3 Budúca práca

Nasledujúce položky boli v analýze identifikované a zdokumentované. Ich riešenie bolo vedome odložené (sekcia 2.7, sekcia 7.7).

Infraštruktúra „V2“

Migrácia WebSocketovej vrstvy a logera do „V2“ je pripravená. Zostavy kompilujú nezávisle a boli overené zmenou *namespace*. Pre automatizáciu pripojenia pacienta bol pripravený návrh postupu: pasívny režim pacienta, evidencia dostupných headsetov/profilov a pripojenie headsetu do miestnosti na pokyn terapeuta. Implementácia tohto mechanizmu zostáva úlohou pokračujúceho vývoja; databáza presetov a nastavení pre pacienta bola čiastočne riešená v práci [13].

Klinický systém

Hand tracking a moderné funkcionality rozšírenej reality (XR) vyžadujú upgrade XR Interaction Toolkit a prechod na novšiu verziu Unity. Náhrada avatar

modelov Ready Player Me za modely s vyššou kostrovou kvalitou je na tento upgrade viazaná. Automatický režim cvičenia vyžaduje revíziu formátu cvičebných plánov spolu s terapeutmi.

Architektúra a prevádzka

Mirror server trpí pádmi pri ukončení a nekonzistentnými stavmi relácií — oprava vyžaduje hlbší zásah do serverovej architektúry. Automatizovaná regresová sada testov pre WebSocketovú vrstvu a výkonnostné benchmarky (latencia správ, priepustnosť pod záťažou) by doplnili manuálne overovanie. Kontinuálna integrácia a kontinuálne nasadzovanie (CI/CD) pre Meta Quest buildy by nahradili manuálne nasadzovanie cez Meta Quest Developer Hub (MQDH). Logovanie je zjednotené na GameLog API; za zváženie zostáva rotácia aktuálneho log súboru počas relácie a prechod na strojovo čitateľný formát logov.

8.4 Záverečné poznámky

Práca vznikala v kontexte reálnej klinickej spolupráce s výskumníkmi Slovenskej akadémie vied. Ich výskum závisel od spoľahlivosti systému. Príspevky neboli akademickými cvičeniami — boli inžinierskymi rozhodnutiami pod obmedzením, že sedenia nemožno rušiť a systém musí fungovať. Výsledkom je spoľahlivejší a udržiavateľnejší základ pre pokračujúci vývoj systému L-NeRVEn.

Zoznam použitej literatúry

1. KHAN, Ammar; IMAM, Yahia Z.; MUNEEER, Mohamed et al. Virtual reality in stroke recovery: a meta-review of systematic reviews. *Bioelectronic Medicine*. 2024, roč. 10, č. 23. Dostupné z DOI: [10.1186/s42234-024-00150-9](https://doi.org/10.1186/s42234-024-00150-9).
2. SOLEIMANI, Mohsen; GHAZISAEEDI, Marjan; HEYDARI, Soroush. The efficacy of virtual reality for upper limb rehabilitation in stroke patients: a systematic review and meta-analysis. *BMC Medical Informatics and Decision Making*. 2024, roč. 24, č. 135. Dostupné z DOI: [10.1186/s12911-024-02534-y](https://doi.org/10.1186/s12911-024-02534-y).
3. LEEB, Robert; PÉREZ-MARCOS, Daniel. Brain-computer interfaces and virtual reality for neurorehabilitation. *Handbook of Clinical Neurology*. 2020, roč. 168, s. 183–197. Dostupné z DOI: [10.1016/B978-0-444-63934-9.00014-7](https://doi.org/10.1016/B978-0-444-63934-9.00014-7).
4. MA, Zhen-Zhen; WU, Jia-Jia; CAO, Zhi et al. Motor imagery-based brain-computer interface rehabilitation programs enhance upper extremity performance and cortical activation in stroke patients. *Journal of NeuroEngineering and Rehabilitation*. 2024, roč. 21, č. 91. Dostupné z DOI: [10.1186/s12984-024-01387-w](https://doi.org/10.1186/s12984-024-01387-w).
5. GONZALEZ-FRANCO, Mar; OFEK, Eyal; PAN, Ye et al. The Rocketbox Library and the Utility of Freely Available Rigged Avatars. *Frontiers in Virtual Reality*. 2020, roč. 1, s. 561558. Dostupné z DOI: [10.3389/frvir.2020.561558](https://doi.org/10.3389/frvir.2020.561558).
6. RENARD, Yann; LOTTE, Fabien; GIBERT, Guillaume; CONGEDO, Marco; MABY, Emmanuel; DELANNOY, Vincent; BERTRAND, Olivier; LÉCUYER, Anatole. OpenViBE: An Open-Source Software Platform to Design, Test, and Use Brain-Computer Interfaces in Real and Virtual Environments. *Presence: Teleoperators and Virtual Environments*. 2010, roč. 19, č. 1, s. 35–53. Dostupné z DOI: [10.1162/pres.19.1.35](https://doi.org/10.1162/pres.19.1.35).
7. KOTHE, Christian; SHIRAZI, Seyed Yahya; STENNER, Tristan; MEDINE, David; BOULAY, Chadwick; GRIVICH, Matthew I.; ARTONI, Fiorenzo; MULLEN, Tim; DELORME, Arnaud; MAKEIG, Scott. The Lab Streaming

- Layer for Synchronized Multimodal Recording. *Imaging Neuroscience*. 2025, roč. 3, IMAG.a.136. Dostupné z DOI: [10.1162/IMAG.a.136](https://doi.org/10.1162/IMAG.a.136).
8. VOURVOPOULOS, Athanasios; FERREIRA, André; BERMÚDEZ I BADIA, Sergi. NeuRow: An immersive VR environment for motor-imagery training with the use of brain-computer interfaces and vibrotactile feedback. In: *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies*. 2016, s. 43–53. Dostupné z DOI: [10.5220/0005939400430053](https://doi.org/10.5220/0005939400430053).
9. VOURVOPOULOS, Athanasios; MARIN PARDO, Octavio; LEFEBVRE, Stéphanie; NEUREITHER, Meghan; SALDANA, David; JAHNG, Esther; LIEW, Sook-Lei. Effects of a Brain-Computer Interface With Virtual Reality (VR) Neurofeedback: A Pilot Study in Chronic Stroke Patients. *Frontiers in Human Neuroscience*. 2019, roč. 13, s. 210. Dostupné z DOI: [10.3389/fnhum.2019.00210](https://doi.org/10.3389/fnhum.2019.00210).
10. WAN, Chunli; ZHANG, Qiyuan; QIU, Yu; ZHANG, Wenting; NIE, Yao; ZENG, Shuyi; WANG, Jian; SHEN, Xiaowen; YU, Cui; WU, Xixi; ZHANG, Yuting; LI, Yongqiang. Effects of dual-task mode brain-computer interface based on motor imagery and virtual reality on balance and attention in patients with stroke: a randomized controlled pilot trial. *Journal of NeuroEngineering and Rehabilitation*. 2025, roč. 22, č. 1, s. 187. Dostupné z DOI: [10.1186/s12984-025-01730-9](https://doi.org/10.1186/s12984-025-01730-9).
11. VALENTE, Madalena; BRANCO, Diogo; BERMÚDEZ I BADIA, Sergi; FERNANDES, Jean-Claude; FIGUEIREDO, Patrícia et al. EEG-based predictors of motor recovery during immersive VR-BCI rehabilitation. *Scientific Reports*. 2026, roč. 16, s. 7870. Dostupné z DOI: [10.1038/s41598-026-39106-1](https://doi.org/10.1038/s41598-026-39106-1).
12. KOREČKO, Štefan; MATHES, Jaroslav; SOBOTA, Branislav; CHMURA, Damián; ROSIPAL, Roman; VANKÓ, Martin. L-NeRVEn virtual environment for neurorehabilitation: architecture, functionality, and evaluation. *Open Computer Science*. 2026, roč. 16, č. 1, s. 20250049. Dostupné z DOI: [10.1515/comp-2025-0049](https://doi.org/10.1515/comp-2025-0049).
13. DZEMIDOVICH, Andrei. *Technické vylepšenia a rozšírenie funkcionality systému pre neurorehabilitáciu vo virtuálnej realite*. 2026. Unpublished.
14. COCKBURN, Alistair. *Hexagonal architecture the original 2005 article* [online]. 2005. [cit. 2026-03-22]. Dostupné z: <https://alistair.cockburn.us/hexagonal-architecture>.

15. MARTIN, Robert C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, 2017. ISBN 9780134494166.
16. ANDROID DEVELOPERS. *Processes and app lifecycle* [online]. [cit. 2026-03-25]. Dostupné z: <https://developer.android.com/guide/components/activities/process-lifecycle>.
17. UNITY TECHNOLOGIES. *MonoBehaviour.OnApplicationQuit()* [online]. [cit. 2026-03-25]. Dostupné z: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnApplicationQuit.html>.
18. OPENTELEMETRY AUTHORS. *OpenTelemetry Logging Specification* [online]. [cit. 2026-03-25]. Dostupné z: <https://opentelemetry.io/docs/specs/otel/logs/>.
19. SERILOG CONTRIBUTORS. *Serilog — Simple .NET logging with fully-structured events* [online]. [cit. 2026-03-25]. Dostupné z: <https://serilog.net>.
20. NLOG PROJECT. *NLog — Flexible & free open-source logging for .NET* [online]. [cit. 2026-03-25]. Dostupné z: <https://nlog-project.org>.
21. APACHE SOFTWARE FOUNDATION. *Apache log4net — A port of the excellent Apache log4j framework to the Microsoft .NET runtime* [online]. [cit. 2026-03-25]. Dostupné z: <https://logging.apache.org/log4net/>.
22. GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN 9780201633610.

Zoznam skratiek

API aplikačné programovacie rozhranie.

BCI rozhranie mozog-počítač.

CI/CD kontinuálna integrácia a kontinuálne nasadzovanie.

EEG elektroencefalografia.

HTTP HyperText Transfer Protocol.

IK inverzná kinematika.

IL2CPP Intermediate Language to C++.

JSON JavaScript Object Notation.

LTS Long-Term Support.

MI motorická imaginácia.

MQDH Meta Quest Developer Hub.

REST Representational State Transfer.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UI používateľské rozhranie.

UX používateľská skúsenosť.

VR virtuálna realita.

XR rozšírená realita.

Slovník

App line distribučná línia aplikácie jednoznačne identifikovaná kombináciou Android package name a podpisovacieho kľúča (keystore). Každý release kanál (LTS, STABLE, RC) predstavuje samostatný app line.

Baseline referenčné meranie vykonané pred začiatkom aktívnej úlohy, slúžiace ako východisková hodnota na porovnanie s následnou aktivitou.

Body ownership ilúzia vlastníctva virtuálneho tela — neurokognitívny fenomén, pri ktorom používateľ vníma virtuálny avatar ako vlastné telo vďaka synchronizovanému vizuálno-motorickému feedbacku.

Build výsledný skompilovaný a zostavený spustiteľný artefakt aplikácie určený na nasadenie, v kontexte Unity označuje konkrétnu platformovú verziu projektu (napr. serverový build, klientský build).

Build promotion postup povýšenia buildu z nižšieho distribučného kanála (RC) na vyšší (STABLE, LTS) po overení stability.

Composition root miesto v aplikácii, kde sa zostavujú a prepájajú závislosti objektov a odkiaľ sa inicializuje výsledný objektový graf.

Cross-room porušenie protokolového pravidla, pri ktorom klient odošle správu s identifikátorom miestnosti odlišným od miestnosti, na ktorú je jeho relácia naviazaná.

Dependency injection návrhový vzor, pri ktorom sa závislosti objektu dodávajú zvonka namiesto ich vytvárania vnútri triedy, čím sa zvyšuje testovateľnosť a modularita.

Dispatch mechanizmus smerovania požiadavky na príslušný spracovateľ. V kontexte logovacieho systému označuje delegovanie logovacieho volania z fasády na konkrétnu implementáciu rozhrania ILogger.

Event-driven architektonický prístup, pri ktorom tok programu riadia udalosti (signály, správy) namiesto sekvenčného vykonávania alebo pravidelného dotazovania.

Facade návrhový vzor poskytujúci zjednodušené rozhranie k množine tried alebo subsystému, čím skrýva vnútornú zložitosť.

Fallback záložný mechanizmus alebo záložná verzia, na ktorú sa systém alebo používateľ presunie v prípade zlyhania primárnej varianty. V kontexte nasadzovania označuje historicky stabilný build kanála LTS.

Fatigue meter komponent používateľského rozhrania zobrazujúci aktuálnu hodnotu neurologickej únavy pacienta získanú z EEG/BCI pipeline v reálnom čase.

Feedback loop cyklická závislosť, pri ktorej výstup operácie vyvoláva opätovné spustenie tej istej alebo nadradenej operácie. V kontexte logovania označuje situáciu, keď zápis logu generuje nový log záznam, čím vzniká rekurzívna slučka.

Flush operácia vyprázdnenia vyrovnávacej pamäte (bufferu), pri ktorej sa nahromadené dáta zapíšu na cieľové úložisko (disk, sieť). V kontexte logovania označuje moment, keď sa záznamy z pamäťového bufferu fyzicky zapíšu do log súboru.

God object anti-vzor v objektovo orientovanom návrhu, pri ktorom jediná trieda preberá príliš veľa zodpovedností a porušuje princíp jednej zodpovednosti.

Handshake úvodná výmena správ medzi dvoma stranami na nadviazanie spojenia a dohodnutie komunikačných parametrov.

Hexagonal architecture architektonický štýl (tiež Ports and Adapters), pri ktorom doménová logika komunikuje s vonkajším svetom výlučne cez definované porty a adaptéry, čím sa oddeľuje od infraštruktúry.

KeepAlive mechanizmus periodického zasielania kontrolných správ na udržanie aktívneho spojenia a detekciu jeho výpadku.

Keystore súbor obsahujúci kryptografický podpisovací kľúč, ktorým sa Android build identifikuje ako legitímna aktualizácia konkrétnej aplikácie. Väzba medzi keystore a package name je trvalá — strata kľúča znemožňuje publikovanie aktualizácií pre daný app line.

Magic strings reťazcové literály s implicitným významom rozptýlené v kóde namiesto použitia typovo bezpečných konštánt, enumov alebo centralizovaného mapovania.

Motor imagery mentálna simulácia pohybu bez jeho skutočného vykonania, využívaná v neurorehabilitácii ako paradigma na aktiváciu motorických oblastí mozgu prostredníctvom rozhrania mozog-počítač.

Neurofeedback metóda biologickej spätnej väzby, pri ktorej sa pacientovi v reálnom čase zobrazuje jeho mozgová aktivita s cieľom naučiť sa ju vedome regulovať.

Observability schopnosť systému poskytovať dostatočné informácie o svojom vnútornom stave prostredníctvom logov, metrík a trasovaní na diagnostiku problémov.

Parsovanie syntaktická analýza vstupných dát, pri ktorej sa surový textový reťazec (napr. JSON) rozloží na štruktúrované objekty podľa definovanej gramatiky.

Pipeline postup spracovania dát v sérii na seba nadväzujúcich krokoch; v kontexte EEG/BCI označuje reťazec modulov OpenViBE spracúvajúcich signál od akvizície po klasifikáciu.

Polling technika opakovaného dotazovania na stav prostriedku v pravidelných intervaloch namiesto čakania na notifikáciu o zmene.

Producer-Consumer vzor súbežného spracovania, pri ktorom producent generuje dáta do zdieľanej fronty a konzument ich asynchrónne odoberá a spracúva.

Race condition chybový stav vznikajúci pri súbežnom prístupe viacerých vlákien k zdieľanému prostriedku bez synchronizácie, pri ktorom výsledok závisí od poradia vykonania.

Raycast technika vrhania lúča z bodu v priestore na detekciu priesečníkov s objektmi scény, používaná na výber, interakciu a kolízne dotazy.

Reentrant vlastnosť funkcie alebo kódu, ktorý môže byť bezpečne volaný opakovane (vrátane rekurzívne) pred dokončením predchádzajúceho volania.

Release vydanie konkrétnej verzie softvéru pre koncových používateľov; v kontexte nasadzovacej stratégie označuje prechod buildu z vývojového prostredia do distribučného kanála.

Replay opakované prehratie uložených správ alebo udalostí; v kontexte sieťovej vrstvy mechanizmus zaistenia doručenia stavu klientom pripojeným po začiatku relácie.

Rollback návrat na predchádzajúcu stabilnú verziu aplikácie v prípade, že aktuálny build vykazuje kritické chyby. V kontexte trojkanálovej release stratégie označuje prechod z kanála RC na STABLE alebo LTS.

Runtime prostredie, v ktorom sa program vykonáva, zahŕňajúce správu pamäte, vlákien a ďalšie služby potrebné na beh aplikácie.

Strategy návrhový vzor, pri ktorom sa rodina algoritmov zapuzdrí do samostatných tried a za behu sa vyberá konkrétna implementácia.

SyncVar atribút knižnice Mirror označujúci premennú, ktorej hodnota sa automaticky synchronizuje zo servera na všetkých pripojených klientov.

Template Method návrhový vzor, pri ktorom nadtrieda definuje kostru algoritmu a podtriedy implementujú jednotlivé kroky bez zmeny celkovej štruktúry.

Thread-safety vlastnosť kódu umožňujúca jeho bezpečné súbežné vykonávanie viacerými vláknami bez vzniku dátových pretekov alebo nekonzistentného stavu.

Trigger v kontexte Unity kolízna zóna, ktorá deteguje vstup herného objektu bez fyzickej kolízie a vyvoláva príslušnú udalosť.

Zoznam príloh

Príloha A Používateľská príručka

Príloha B Systémová príručka

Príloha C Implementačné detaily logovacieho systému

Príloha D CD médium — záverečná práca v elektronickej podobe,

Príloha E Repozitár výsledného projektu je dostupný na adrese

<https://git.kpi.fei.tuke.sk/skor-lirkis-projects/vrarm-unity>

A Používateľská príručka

Táto používateľská príručka nadväzuje na pôvodnú projektovú príručku. Bola aktualizovaná a rozšírená novými časťami, ktorých obsahom je dokumentácia novej funkcionality. Dôvodom pre tento krok je, že sa veľká časť funkcionality nezmenila a obsahovo je stále relevantná.

A.1 Inštalácia softvéru

Aplikácie aktuálne nie je na žiadnej platforme, bežne používanej na šírenie softvéru. Je preto potrebné zabezpečiť prístup k potrebným súborom cez priame spojenie. Kontaktné adresy: `stefan.korecko@tuke.sk`, `damian.chmura@student.tuke.sk`, `andrei.dzemidovich@student.tuke.sk`. Pre fungovanie je potrebné mať zabezpečený funkčný server a správne nastavené klientske inštancie (IP adresa servera). Aktuálne server nie je verejne dostupný.

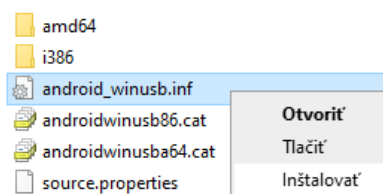
A.1.1 Osobný počítač

Aplikáciu nie je potrebné inštalovať. Spúšťa sa otvorením `NeuroRehabVR.exe`, ktorý je súčasťou priečinka so všetkými súborami.

A.1.2 Oculus Quest 2

Aktuálne nie je možné spúšťať a inštalovať aplikáciu cez platformu na šírenie softvéru (ako napríklad Oculus Store). Je potrebné ju ručne nainštalovať. Postup inštalácie obsahuje niekoľko krokov, na inštaláciu softvéru je potrebné použitie stolového počítača. Postup je nasledovný:

1. **USB debugging** - na inštaláciu aplikácií je potrebné zapnúť developer mode. Je možné ho zapnúť dvoma spôsobmi: použitím aplikácie v smartfóne alebo zmenou nastavení priamo v zariadení Oculus. Pri použití aplikácie Meta Quest na smartfóne je potrebné spárovať zariadenia, následne vybrať zariadenie a prejsť do nastavení zariadenia, kde je možné povoliť developer



Obrázok A.1: Inštalácia ovládača Oculus Go

mode. V prípade nastavenia na headsete priamo, je postup nasledovný: je potrebné prejsť do nastavení “Settings”, následne otvoriť kartu “System”. Ako posledný krok je potrebné prejsť do karty “Developer” na ľavej strane. V tejto karte je potrebné povoliť možnosť “USB Connection Dialog”. Následne sa po pripojení zariadenia do počítača na headsete objaví dialóg “Allow USB debugging?”. Je potrebné povoliť túto možnosť. Ak plánujete používať počítač v spojení s helmou častejšie, odporúčame povoliť možnosť “Always allow from this computer”,

2. **inštalácia ovládačov** - na stolovom počítači je potrebné nainštalovať Oculus Go ovládač¹. Po stiahnutí adresára, je potrebné ho rozbaľiť. Po rozbalení je možné nainštalovať ovládač. Pravým tlačidlom myši na `android_winusb.inf` sa zobrazí možnosť “Inštalovať” (obr. A.1),
3. **Android Debug Bridge** - skrátene ADB, slúži na komunikácia s android zariadeniami, pripojenými k počítaču. Keďže Oculus Quest využíva android ako operačný systém, je možné tento nástroj použiť na inštaláciu softvéru. ADB je možné stiahnuť z oficiálnej stránky android². Po stiahnutí a rozbalení priečinka je potrebné prejsť do cieľového priečinka, kde sú potrebné súbory rozbalené (odporúčame použiť iný priečinok než ten z predošlého kroku),
4. **inštalácia aplikácie** - inštaláciu je potrebné uskutočniť cez príkazový riadok. Prejdite do priečinka, kde ste rozbalili ADB nástroje v predošlom kroku. Podržaním tlačidla Shift a stlačením pravého tlačidla myši otvoríte dialógové menu. Zvoľte možnosť “Otvoriť tu okno prostredia PowerShell”, prípadne “Otvoriť tu príkazový riadok”. Môže sa to líšiť v závislosti od zariadenia. Následne je potrebné zadať príkaz:

```
.\adb.exe install -r .\Build.apk
```

“.\Build.apk” je potrebné nahradiť celou cestou k súboru aplikácie. Príkaz

¹<https://developer.oculus.com/downloads/package/oculus-go-adb-drivers/>

²<https://developer.android.com/studio/releases/platform-tools>

s celou cestou by mohol vyzeráť nasledovne:

```
.\adb.exe install -r D:\Unity\NeuroRehabVR\Oculus\Build.apk
```

Pre jednoduchosť je preto možné uložiť inštalačný súbor “Build.apk” do rovnakého priečinka, v ktorom sú uložené ADB nástroje. Následne je možné použiť relatívnu cestu, ako je ukázané v prvom príklade príkazu,

5. **spustenie aplikácie** - po inštalácii sa aplikácia objaví medzi nainštalovaným softvérom. Je možné ju nájsť v priečinku “Unknown sources”.

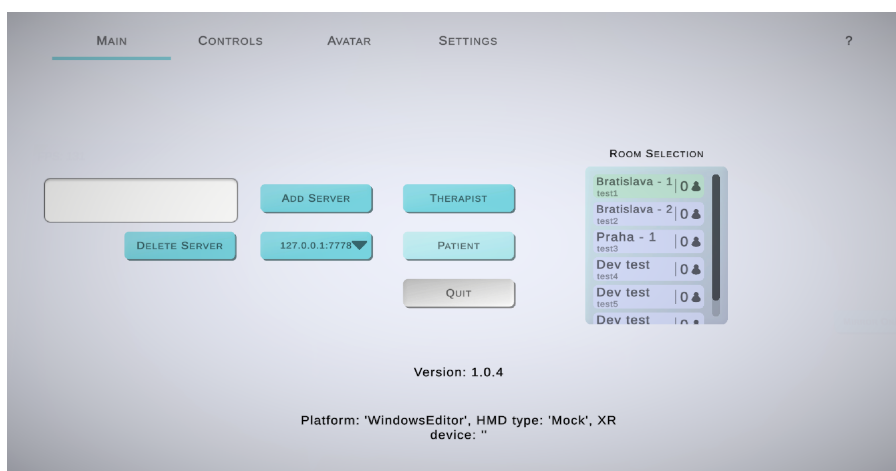
A.2 Použitie a ukážka aplikácie

Po spustení aplikácie sa používateľ nachádza v offline lobby scéne. Používateľ môže pridať novú IP adresu servera na pripojenie, meniť nastavenia, prispôbiť vzhľad avatara, ako aj resetovať výšku avatara. Pohyb v prostredí závisí od platformy, na ktorej sa nachádzame.

V prípade **osobného počítača** je možné ovládať pohyb pomocou klávesov “WASD” alebo kurzorových šípok, otáčanie funguje pomocou myši, interakcia s objektmi (UI aj 3D objekty) pomocou ľavého tlačidla myši. Menu je možné ovládať priamo stlačením tlačidiel v 3D priestore, alebo je možné zobrazíť 2D menu na vrchu obrazu. Po stlačení skratky na zobrazenie menu (klávesa “M”) sa odomkne myš a ovládanie menu je možné štandardným spôsobom. Presúvanie objektu je implementované jeho presúvaním myšou a zároveň držaním ľavého tlačidla myši. Presúvať objekt je možné v bežnom móde pohybu, stlačením klávesy “G” je možné uzamknúť rotáciu kamery. Zároveň sa odomkne kurzor myši. Pred pripojením má používateľ možnosť voľby roly, v ktorej bude vystupovať počas rehabilitácie. Vzhľad úvodného menu je na obr. [A.2](#).

Počas používania **VR helmy** je možný pohyb pomocou chôdze, teleportácie alebo použitím páčky ľavého ovládača. Otáčanie je možné pomocou páčky praveho ovládača, ale vzhľadom na pocit nevoľnosti, ktorý spôsobuje odporúčame otáčať sa len celým telom. Interakcia s menu elementami je možná stlačením hlavného tlačidla ovládača, alebo použitím “Trigger” tlačidla. Interakcia funguje obooma ovládačmi. Presun objektov je možný zamierením ovládača na objekt, alebo dotknutia sa objektu a držaním tlačidla “Grip” ovládača (V prípade Oculus Quest ovládačov to sú tlačidlá na stranách). Zobrazenie menu nad ovládačom je možné pomocou tlačidla menu na ľavom ovládači. Podržaním tohto tlačidla sa objaví menu pred používateľom, ktoré zostane prilepené na danej pozícii.

Systém teleportácie umožňuje rýchly a jednoduchý pohyb v rámci virtuálneho prostredia. Tento spôsob pohybu je určený hlavne pre používateľov, ktorým



Obrázok A.2: Úvodné menu

klasický pohyb pomocou joysticka spôsobuje nevoľnosti.

A.2.1 Kroky pre správne využitie teleportačného systému

1. Aktivácia Teleportácie:

- Uchopte ľavý ovládač VR systému.

2. Výber Teleportačného Miesta:

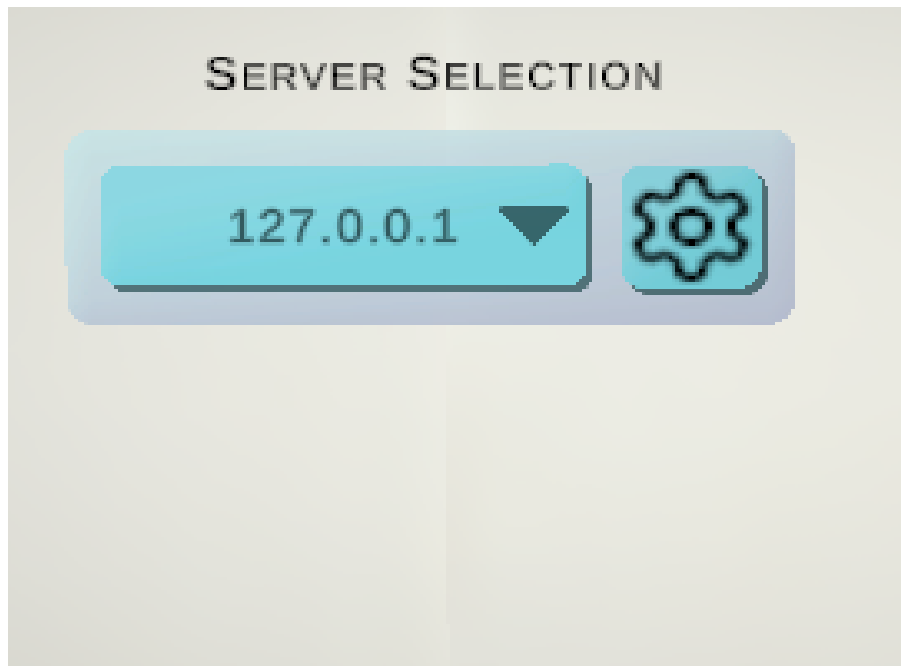
- Nasmerujte ľavý ovládač na miesto, kam sa chcete teleportovať. Farba ukazovateľa sa zmení na žltú, ak sa je na dané miesto možné presunúť.

3. Teleportácia:

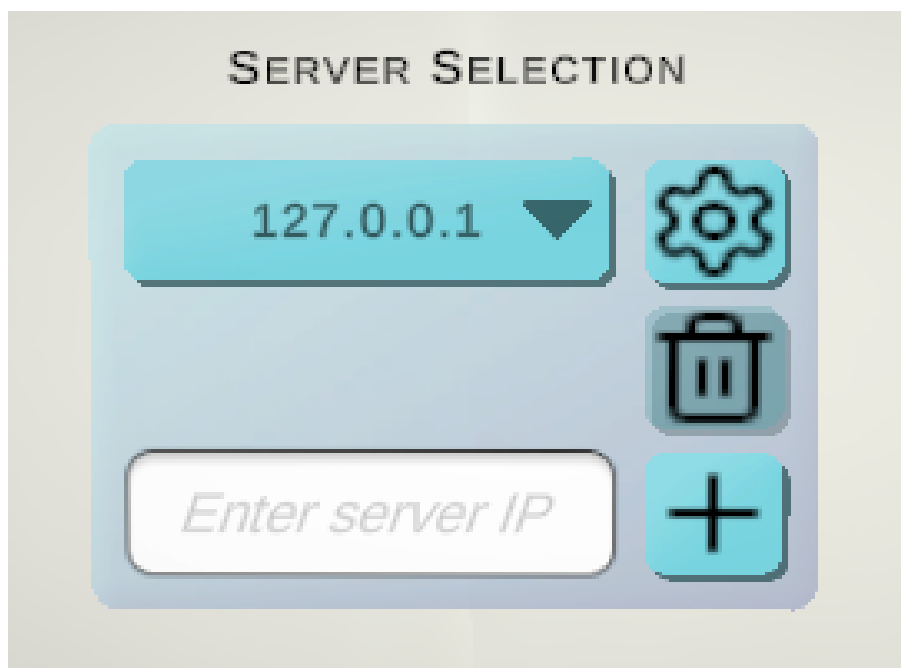
- Stlačte tlačidlo uchopenia na ľavom ovládači, keď je laserový ukazovateľ nasmerovaný na platnú teleportačnú oblasť.
- Po stlačení tlačidla sa okamžite presuniete na zvolené teleportačné miesto.

A.2.2 Správa serverov

Zoznam serverov je predvolene načítaný z konfiguračného súboru. Správa serverov je skrytá v popup paneli, ktorý sa zobrazí po stlačení príslušného tlačidla (obr. A.3 a A.4). V paneli je možné pridať nový server zadáním jeho IP adresy. Servery načítané z konfiguračného súboru sú chránené a nie je možné ich odstrániť. Po výbere servera v rozbaľovacom zozname sa aktualizuje zoznam dostupných miestností.



Obrázok A.3: Správa serverov so zatvoreným panelom



Obrázok A.4: Správa serverov s otvoreným panelom

A.2.3 Hlasová komunikácia

Po úspešnom pripojení k serveru je hlasová komunikácia automaticky inicializovaná, avšak mikrofón aj reproduktory sú predvolene stlmené. Hlasovú komunikáciu je možné aktivovať manuálne prostredníctvom prepínača v používateľskom rozhraní.

A.3 Rola pacienta

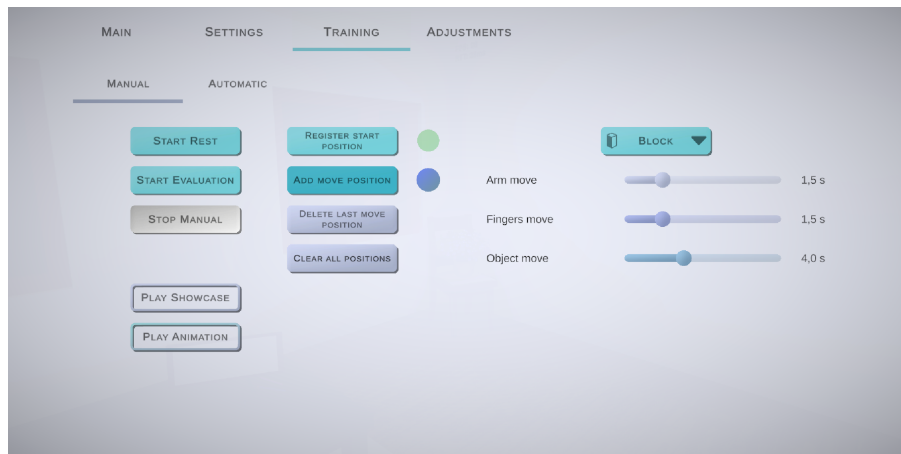
Pacient má počas cvičenia značne obmedzené možnosti. Môže sa hýbať v prostredí, no jeho prácou je len počkať na pokyny terapeuta. Po pripojení je automaticky zvolená ruka, ktorou sa vykonáva cvičenie pravá. Toto nastavenie môže meniť terapeut po pripojení do online prostredia. Terapeut môže pacientovi zafixovať zdravú ruku. Pacient to môže urobiť sám stlačením "N" na klávesnici alebo "Y" a "B" na ovládači.

Ak sú pre danú miestnosť uložené nastavenia pacienta z predchádzajúceho sedenia, server ich automaticky aplikuje po pripojení pacienta bez nutnosti manuálneho zásahu.

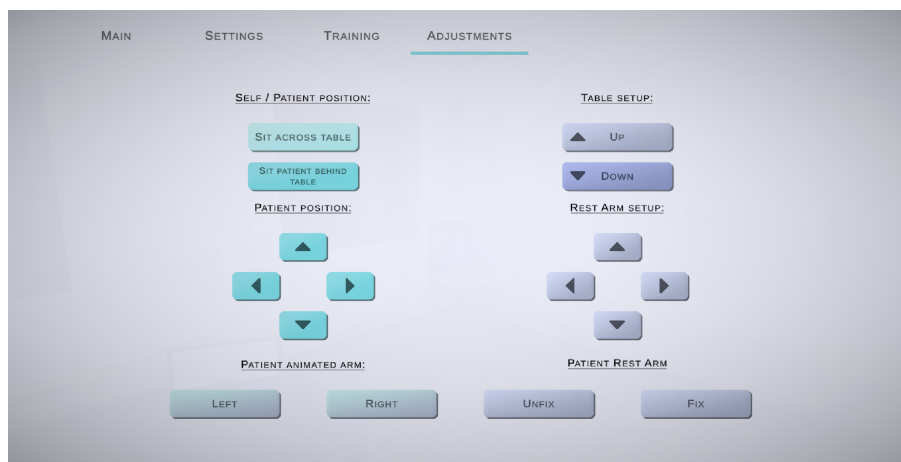
A.4 Rola terapeuta

Terapeut má za úlohu viesť celú rehabilitáciu. V porovnaní s rolou pacienta má tiež po pripojení svoje vlastné menu, kde sa nachádzajú nastavenie týkajúce sa rehabilitácie, ako aj pozícií počas cvičenia. Terapeut môže zmeniť aktívnu ruku pacienta, ktorá je rehabilitovaná alebo zafixovať zdravú. Na obr. A.5 je možné vidieť nastavenia, ktoré sú dostupné počas cvičenia terapeutovi. Môže meniť typ pohybu, ktorý je aktuálne používaný. Na výber je zo štyroch rôznych pohybov. Každý pohyb je špecifický predmetom, ktorý sa pri pohybe používa. Terapeut je taktiež schopný presúvať predmet a pridávať ďalšie pohyby pre rehabilitáciu. Dôležité je aj nastavenie časovania pre animácie, čas čakania alebo aj počet opakovaní. Tieto nastavenia je možné meniť v pravej časti menu. Pred spustením rehabilitácie môže terapeut tiež aj zobraziť ukážkovú animáciu. Po spustení rehabilitácie bude následne server čakať na udalosť z prostredia OpenVibe. Po prijatí správy sa spustí animácia na každom klientovi. Ak sa nepošle animácia v rozmedzí čakania, celé aktuálne cvičenie sa preruší a je potrebné ho znova spustiť.

Podstatnú časť nastavení ešte pred rehabilitáciou tvorí prispôsobenie správnych pozícií. Na obr. A.6 je možné vidieť možnosti, ktoré terapeut má. Môže presunúť svoju postavu bez nutnosti pohybu za stôl. Taktiež dokáže presunúť za stôl aj pacienta. To znamená, že pacient a terapeut sa nemusia fyzicky presúvať. Nie je potrebné sa presúvať ani pomocou ovládačov. Terapeut taktiež dokáže precíznejšie meniť pozíciu pacienta jeho posúvaním vpred, vzad a do strán. Systém taktiež ponúka aj možnosť kalibrácie výšky stola a zmenu oddychovej polohy ruky pacienta.



Obrázok A.5: Menu nastavení rehabilitácie



Obrázok A.6: Menu s nastaveniami pozícií

A.4.1 Nastavenie MI threshold hodnoty

Terapeut môže počas prípravy alebo priebehu cvičenia nastaviť threshold hodnotu pre EEG/MI vyhodnocovanie. Táto hodnota určuje hranicu, voči ktorej sa porovnáva prijímaný EEG signál. Nižšia hodnota znamená, že systém bude prah považovať za dosiahnutý pri slabšej EEG aktivite. Vyššia hodnota vyžaduje výraznejšiu EEG aktivitu.

Postup zmeny threshold hodnoty:

1. Otvorte terapeutické menu s nastaveniami cvičenia.
2. Nájdite ovládací prvok pre threshold hodnotu.
3. Posuňte slider na požadovanú hodnotu.
4. Skontrolujte zobrazenú novú hodnotu.



Obrázok A.7: Nastavenie MI threshold hodnoty v terapeutickom rozhraní

5. Stlačte tlačidlo Send, aby sa hodnota použila v aktuálnej relácii.

Hodnota je zobrazená v normalizovanom rozsahu od 0.00 do 1.00. Napríklad hodnota 0.55 zodpovedá 55% normalizovanej škály. Rozhranie rozlišuje medzi hodnotou vybranou sliderom a aktuálne aplikovanou hodnotou. Samotné posunutie slidera ešte nemeňte aktívne nastavenie; hodnota sa aplikuje až po stlačení tlačidla Send.

Ak sa do miestnosti neskôr pripojí ďalší klient alebo sa znovu pripojí EEG klient, systém mu poskytne aktuálne nastavenú threshold hodnotu automaticky.

A.4.2 Ukazovateľ únavy pacienta

Ukazovateľ únavy pacienta zobrazuje priebežne prijímanú EEG/MI hodnotu. Slúži na sledovanie aktuálneho odhadu únavy pacienta počas cvičenia bez toho, aby terapeut musel opustiť terapeutické rozhranie.

Ukazovateľ zobrazuje normalizovanú hodnotu od 0.00 do 1.00:

- 0.00 znamená najnižšiu prijatú úroveň únavy,
- 1.00 znamená najvyššiu prijatú úroveň únavy,
- hodnoty medzi nimi predstavujú pomernú úroveň na rovnakej škále.

Zvislý ukazovateľ sa vyplnía zdola nahor podľa rastúcej hodnoty. Farba hodnoty sa mení od zelenej pri nízkych hodnotách cez stredné odtiene až po červenú pri vysokých hodnotách. Aktuálna hodnota je zároveň zobrazená aj číselne.

Pri prijatí novej hodnoty sa ukazovateľ aktualizuje a predchádzajúca hodnota sa presunie do krátkej histórie. História zobrazuje posledných päť predchádzajúcich hodnôt, pričom najnovšia predchádzajúca hodnota je uvedená navrchu. Terapeut tak môže sledovať, či sa únava v posledných krokoch zvyšuje, znižuje alebo zostáva približne stabilná.

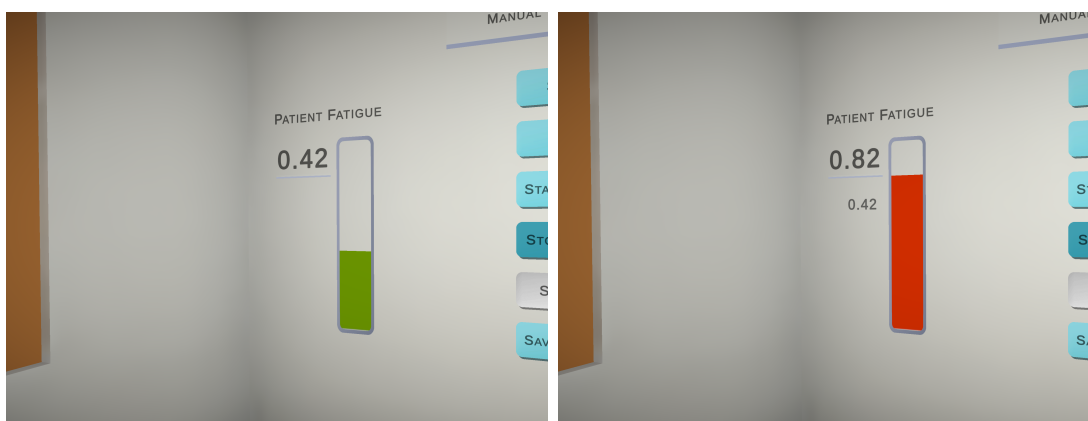
Terapeut môže ukazovateľ únavy použiť najmä na:

- sledovanie nárastu únavy počas cvičenia,
- porovnanie aktuálnej hodnoty s poslednými prijatými hodnotami,
- rozhodnutie, či pacient potrebuje prestávku alebo zníženie intenzity,
- overenie, že systém prijíma EEG údaje počas relácie.

Ak sa ukazovateľ únavy neaktualizuje, je potrebné overiť, či je EEG klient pripojený, či je pripojený do správnej miestnosti a či počas relácie odosiela údaje.

A.4.3 Ukladanie nastavení pacienta

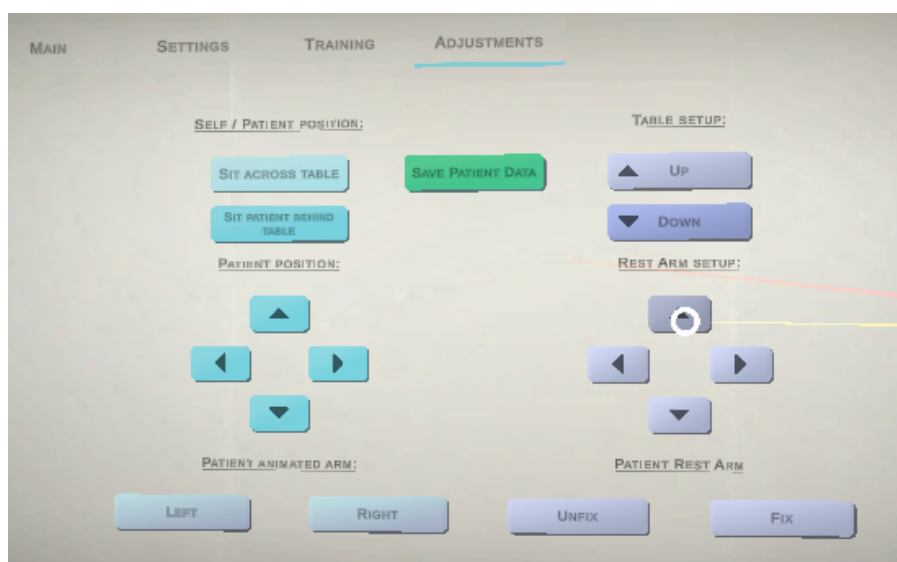
V taboch "Adjustments" a "Training" je k dispozícii tlačidlo "Save Patient Data" (obr. A.9 a A.10). Stlačením tlačidla sa aktuálne nastavenia uložia do databázy a pri ďalšom pripojení pacienta do tej istej miestnosti sa automaticky obnovia. Tab "Adjustments" ukladá výšku stola, pozíciu a rotáciu pacienta, polohu odpočívajúcej ruky a mierku avatara. Tab "Training" ukladá tréningové parametre: trvanie



(a) Nízka hodnota únavy

(b) Vysoká hodnota únavy

Obrázok A.8: Ukazovateľ únavy pacienta v terapeutickom rozhraní

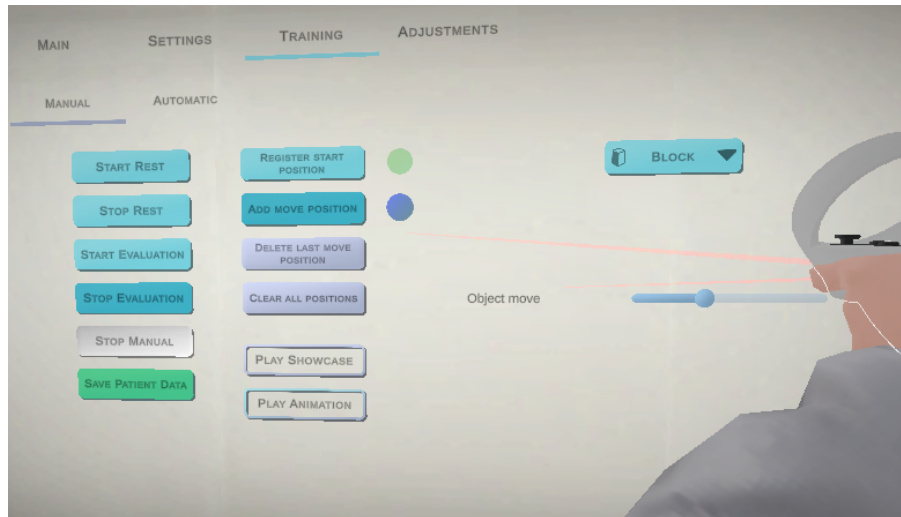


Obrázok A.9: Tab Adjustments s uložením nastavení pacienta

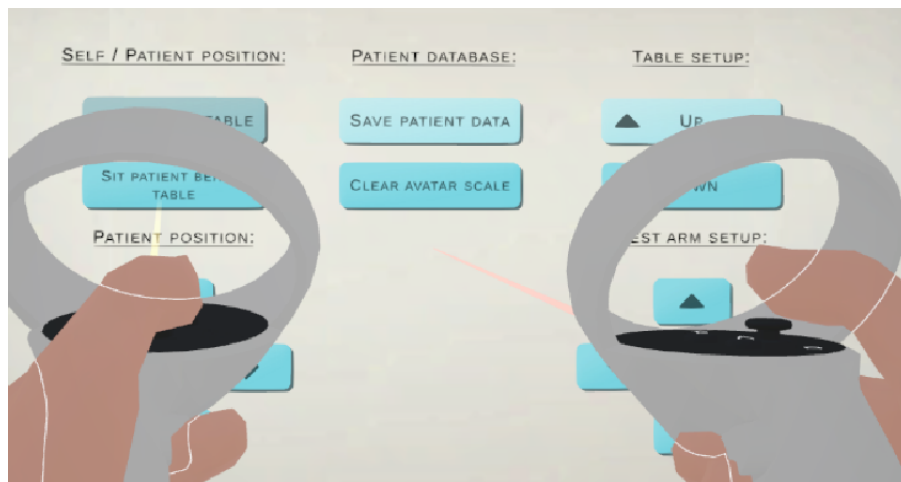
periódy čakania a odpočinku, trvanie pohybu, počet opakovaní, typ animácie a pozície objektov pre jednotlivé typy cvičení.

Ak je mierka avatara uložená v databáze a je potrebné ju zmeniť, terapeut môže použiť tlačidlo pre reset mierky avatara v tabe “Adjustments” (obr. A.11). Po jeho stlačení sa uložená hodnota vymaže a pri ďalšom pripojení pacienta sa použije hodnota zvolená v offline scéne.

Terapeut je schopný nastaviť viacero pozícií pre pohyb počas animácie. Systém ale má nastavených niekoľko obmedzení. Maximálny počet pozícií pre pohyb je 10, animácia potrebuje mať nastavenú aspoň jednu pozíciu (začiatočnú) aby ju bolo možné spustiť, pozícia sa použije len v prípade, že je v dosahu ruky pacienta. To znamená, ak sa pacient pohne počas rehabilitácie a pozícia už nie je v jeho dosahu, pozícia sa preskočí. Animácia s kľúčom a zámkom má trochu iné nastave-



Obrázok A.10: Tab Training s uložením nastavení pacienta



Obrázok A.11: Tlačidlo pre reset uloženej mierky avatara

nia. Animácia musí mať vždy dve pozície, aby ju bolo možné spustiť začiatočnú pozíciu a cieľovú pozíciu zámku.

A.5 Ukážka systému

V tejto časti ukážeme zopár obrázkov, kde je možné vidieť systém a niektoré vlastnosti, ktorými disponuje.

A.5.1 Vzhľad prostredia

Na obr. [A.12](#) a [A.13](#) je vyobrazené prostredie, v ktorom prebieha rehabilitácia. Na stole je možné vidieť objekt, ktorý sa aktuálne používa počas rehabilitácie. Pri zmene typu animácie sa nový objekt zobrazí na stole.



Obrázok A.12: Prostredie s pohľadom na stôl



Obrázok A.13: Zvyšok miestnosti

A.5.2 Ukážka animácií

Na pod-obrázkoch obr. A.14 je možné vidieť rozdielne pohyby počas vykonávania animácií. Na pod-obrázku (d) sú taktiež vidieť ukazovatele, ktoré vyznačujú pozície pre pohyby. Zelená guľa predstavuje štartovaciu pozíciu, modrá ostatné pozície. Tieto značky vidí len terapeut. Ako bolo spomenuté v predošlej kapitole, pozície animácií sú obmedzené dosahom ruky pacienta. Na obr. A.15 je možné vidieť ukazovateľ dosahu. Tento ukazovateľ je viditeľný len terapeutom počas presúvania objektu. Na obr. A.16 je vidieť animáciu z pohľadu pacienta.



(a) Uchopenie a presun kvádra



(b) Uchopenie kocky



(c) Zdvihnutie pohára



(d) Presunutie kľúča do zámky

Obrázok A.14: Ukážka rôznych typov animácií



Obrázok A.15: Dosah ruky pacienta



Obrázok A.16: Animácia z pohľadu pacienta

B Systémová príručka

Táto systémová príručka je určená pre vývojárov a správcov systému NeuroRehabVR. Nadväzuje na pôvodnú projektovú príručku a dopĺňa ju o postupy pre importovanie projektu, preklad aplikácie, nasadzovanie servera, správu release kanálov a integračné rozhrania.

B.1 Importovanie projektu

Projekt je potrebné možné stiahnuť a importovať do Unity, napríklad použitím aplikácie Unity Hub. Aplikácie aktuálne používa verziu Unity editora 2022.3.62f3³, ktorú je možné stiahnuť z oficiálnych stránok Unity. Moduly pre platformy, ktoré je potrebné stiahnuť pri inštalácii sú:

- Android Build Support,
- Windows Build Support,
- Linux Dedicated Server,
- Windows Dedicated Server.

Nie je nutné stiahnuť všetky moduly naraz, keďže je možné ich stiahnuť aj neskôr. Pri prvom otvorení projektu sa automaticky stiahnu všetky potrebné balíčky, ktoré nie sú súčasťou projektu. Po spustení editora je ešte potrebné počkať, kým sa importujú všetky potrebné knižnice a nastaví sa potrebné parametre prostredia. Ďalej je potrebné cez menu “Edit” > “Project Settings” > “Services” vybrať našu organizáciu NeuroRehabVR, aby sa zapol Unity Cloud. Pre spustenie aplikácie a otestovanie plnej funkcionality je potrebné použiť viacero inštancií (aspoň jeden server a jeden klient). Odporúčame použitie rozšírenia ParrelSync do Unity. Pomocou ParrelSyncu je možné vytvoriť si klony a tie používať ako ďalšie inštalácie aj na rovnakom počítači. Veľkou výhodou ParrelSyncu je to, že zmeny vykonané v hlavnej inštancii sa automaticky aktualizujú aj v klonoch. Pri spúšťaní aplikácie

³<https://unity.com/releases/editor/qa/lts-releases?version=2022.3>

v editore je potrebné dať si pozor na to, akú scénu máme aktuálne otvorenú. Je potrebné spúšťať aplikáciu so scénou “MenuScene” otvorenou. Server spustíte stlačením ServerOnly vľavo hore (v normálnom builde sa server spustí sám, ale v Editore tomu tak nie je). Klientov pripojiť k serveru je možné priamo použitím menu v scéne. Je možné použitie aj HUD vľavo hore, toto menu je ale len zamýšľané na použitie počas vývoja, nie počas prevádzky. Aplikácie funguje len na Windowse alebo Androide (Oculus). OpenXR nepodporuje DX11 na Linuxe z nejakého dôvodu. Pokusy o spustenie na Linuxe pomocou Protonu alebo Wine-u neboli úspešné. Jediný spôsob, ako poskytnúť funkcionality na Linuxe je zmena nastavení (nižšie opísané v sekcii **B.2**).

Ak by bol nejaký problém, napr. nefungujúce pripojenie klientov na server, je potrebné skontrolovať niekoľko parametrov, ktoré môžu pomôcť opraviť tento problém:

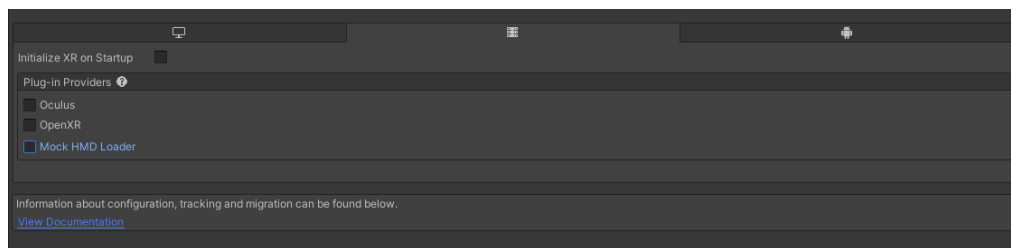
- správne IP adresy (je možné použiť localhost) - server aj klienti musia byť nastavený na rovnakú IP adresu,
- antivírusový softvér - niekedy antivírusový softvér blokuje porty (aktuálne server používa port 7777),

B.2 Preklad aplikácie

Preklad v Unity sa vykonáva cez menu “File” > “Build Settings”. Otvorí sa okno, kde je možné meniť nastavenia pre preklad. Je dôležité dať si pozor na to, aby boli zvolené všetky scény, ktoré sú v aplikácii používané. Pri preklade aplikácie je potrebné nastaviť XR Plugin, ktorý poskytuje funkcionality XR. V aplikácii používame dva pluginy: OpenXR a Mock HMD. OpenXR zabezpečuje fungovanie VR headsetov, Mock HMD Loader používame pri simulovaní XR na desktope.

B.2.1 Preklad pre server

Z ľavej časti je potrebné vybrať možnosť Dedicated Server. Následne je potrebné vybrať Target platform. Je možné použiť aj Linux aj Windows, aplikácia je otestovaná na Windowse primárne, Linux je otestovaný čiastočne. Pred tým, než môžeme vygenerovať potrebné súbory je potrebné zmeniť nastavenia projektu. Do nastavení môžeme prejsť dvoma spôsobmi, v okne určenom na build stlačením tlačidla “Player settings”, alebo cez menu “Edit” > “Project Settings”. Nastavenia kvality a iných špecifikácií už sú vytvorené. Najdôležitejšie nastavenie je ale v karte “XR Plug-in management”. V otvorenom menu je potrebné vybrať



Obrázok B.1: Nastavenia XR Plug-in management pre server

pod-kartu určenú pre dedikované servery (obr. B.1). V prípade prekladu pre servery je potrebné vypnúť loader-y. Ak ostanú loader-y povolené predlžuje to čas buildovania a aj veľkosť vygenerovaných súborov. OpenXR loader taktiež nie je kompatibilný s operačným systémom Linux. Tieto loadre nie sú vôbec potrebné pre fungovanie servera, keďže XR modul nikdy nezapíname. Následne je možné zmeniť platformu a spustiť generovanie buildu.

B.2.2 Preklad pre stolový počítač

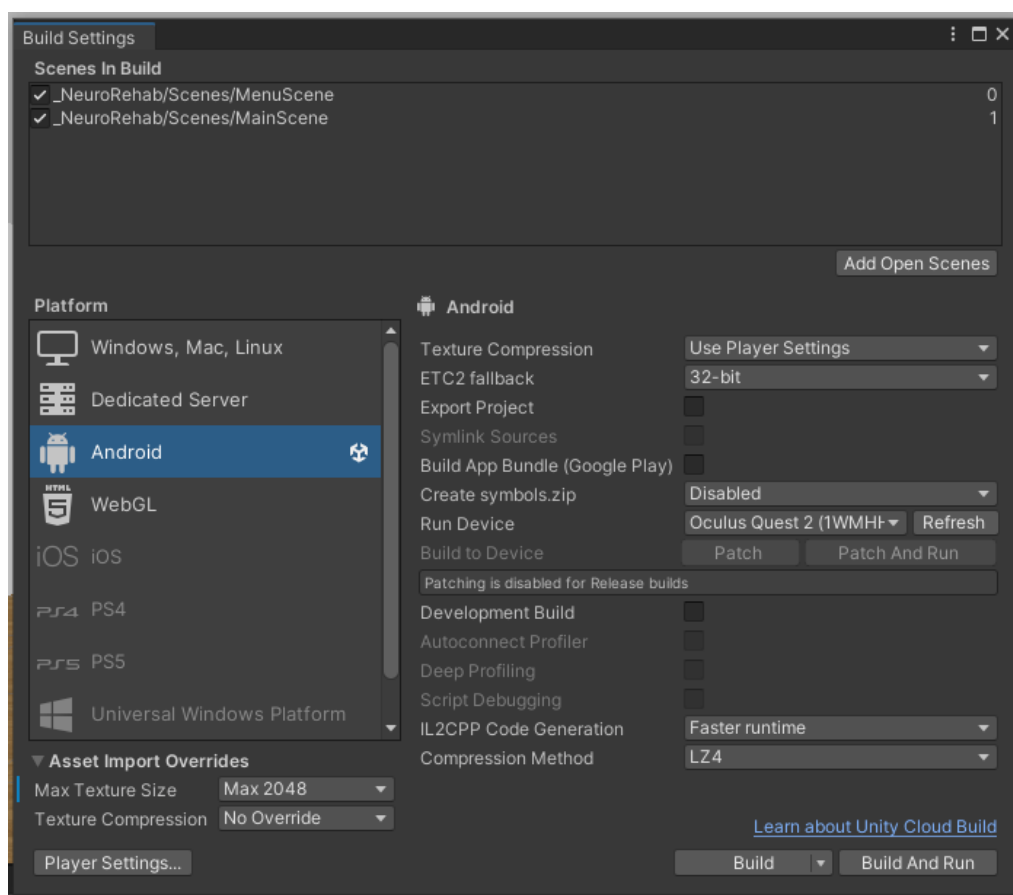
V okne “Build Settings” zvolíme možnosť pre “Windows, Mac, Linux”. Target platform zvolíme Windows. Ak sme pred tým menili nastavenia pre dedikovaný server je potrebné prejsť do nastavení projektu, konkrétne “XR Plug-in management” a zapnúť loadre nanovo. Dôvodom je, že pri zmene nastavení “XR Plug-in management” pre server sa zmenia aj pre desktop. Podľa informácií, ktoré sme našli na fóre je to takto “by design”. Možno sa to v budúcnosti zmení. OpenXR plugin sa použije v prípade, že používame VR headset pripojený k stolovému počítaču.

Linux

Aj napriek tomu, že OpenXR nefunguje na Linuxe, je možné preložiť aplikáciu bez tohto pluginu. V nastaveniach pre “XR Plug-in management” je potrebné vypnúť loader OpenXR. Problémom je potreba čiastočne prispôbiť kód. Bez pluginu OpenXR bude možné aplikáciu používať v desktop móde, prípadne použitie simulovaného VR (keďže táto funkcionlita je zabezpečená pluginom Mock HMD). Aplikácie ale nie je otestovaná na Linuxe, takže je potrebné tomu venovať pozornosť v prípade, že chceme ponúkať podporu pre Linux.

B.2.3 Preklad pre Oculus Quest 2

Vytvorenie apk súboru pre Oculus Quest 2 je možné aj bez pripojeného zariadenia. Ak je ale zariadenie aj pripojené je možné hneď nainštalovať aplikáciu priamo



Obrázok B.2: Nastavenia na preklad pre Android

na Oculus. Ohľadom loadrov, ktoré sme museli meniť v prípade prekladu na stolový počítač alebo server, to na android nie je potrebné. Nesmie ale byť povolený Mock HMD Loader, keďže jeho spustenie by znefunkčnilo OpenXR plugin. Nastavenia prekladu je možné do určitej miery upravovať. Jedným z nich je Run Device. Ak je Oculus Quest 2 pripojený k počítaču, je možné prepnúť na tento VR headset. Následne Stlačením tlačidla “Build nad Run” sa spustí preklad aplikácie. Nová verzia aplikácie sa zároveň aj nainštaluje na zariadenie. Ďalším nastavením je “IL2CPP Code Generation” . Je možné zvoliť z dvoch možností, možnosť pre rýchlejší beh aplikácie a možnosť pre rýchlejší preklad. Počas vývoja je možné používať druhú možnosť, keďže generuje menšie inštalčné súbory a taktiež aj preklad trvá kratšie. Ak chceme inštalovať priamo na Oculus je potrebné mať povolený developer mode. Rozdiel v trvaní prekladu ale nie je príliš významný. Na obr. B.2 je možné vidieť nastavenia, ktoré sme používali počas vývoja.

B.3 Nasadenie servera

Nasadenie servera je jednoduché, jediné čo je potrebné spraviť je spustiť správny build. Server umožňuje spustenie na konkrétnej IP adrese, takže nie je potrebné vytvárať nový build len kvôli zmene IP adresy (klientske inštalácie je potrebné vygenerovať nanovo, aktuálne neponúkajú možnosť konfigurácie). Spustenie servera so špecifickou IP adresou je možné prepínačom `serverip`. Celý príkaz by mohol vyzeráť nasledovne:

```
.\NeuroRehabVR.exe -serverip 192.168.1.210
```

Namiesto IP adresy je možné taktiež použiť kľúčové slovo "localhost".

B.3.1 Konfigurácia zoznamu serverov

Zoznam predvolených serverov pre klientske inštalácie je definovaný v konfiguračnom súbore `servers.json`, ktorý je súčasťou projektu. Súbor má nasledujúci formát:

Výpis B.1: Formát konfiguračného súboru `servers.json`

```
[  
  {  
    "serverIp": "147.232.205.23"  
  },  
  {  
    "serverIp": "127.0.0.1"  
  }  
]
```

Server na prvej pozícii v súbore sa pri prvom spustení aplikácie stane predvoleným. Servery definované v tomto súbore sú označené ako chránené, preto ich používatelia nemôžu odstrániť prostredníctvom používateľského rozhrania.

B.4 Databáza

Systém používa SQLite databázu na serveri pre ukladanie profilov pacientov. Databáza je inicializovaná automaticky pri prvom spustení servera na danom zariadení a naplní sa zoznamom terapeutických miestností z nakonfigurovaného objektu `NetworkSettings`. Databázový súbor sa nachádza v adresári:

```
%USERPROFILE%\AppData\LocalLow\<CompanyName>\<ProductName>\
```

Obsah databázy je možné prezerať a upravovať pomocou nástrojov pre SQLite, napríklad DB Browser for SQLite. Priama úprava databázy je vhodná v prípade potreby zmeny zoznamu miestností alebo opravy nesprávnych hodnôt. Pred priamou úpravou odporúčame zálohovať súbor databázy.

B.5 Nasadenie release buildov cez Meta Quest Developer Hub

Táto časť opisuje postup nasadzovania nových verzií aplikácie NeuroRehabVR do beta vetvy Meta Store. Postup vychádza z procesov popísaných v kapitole 6.

B.5.1 Predpoklady

Pred zostavením a nahraním buildu je potrebné overiť:

- prístup k Meta developer organizácii (účet s oprávneniami na upload),
- nainštalovaný Meta Quest Developer Hub (MQDH),
- prístup k správne Android keystore súboru (vrátane aliasu a hesla) pre cieľový kanál,
- telefón pre SMS verifikáciu pri uploade,
- Unity projekt s korektnou konfiguráciou.

B.5.2 Identifikátory kanála

Podľa cieľového release kanála je potrebné nastaviť správne hodnoty:

Tabuľka B.1: Konfigurácia podľa release kanála

Kanál	Package Name	Product Name	Bundle Code rada
LTS	com.lirkis.neurorehabvr	NeuroRehabVR [LTS]	10000+ (zmrazený)
STABLE	com.lirkis.neurorehabvr.stable	NeuroRehabVR [STABLE]	11000+
RC	com.lirkis.neurorehabvr.rc	NeuroRehabVR [RC]	12000+

B.5.3 Nastavenia buildu v Unity

1. Player Settings → Android:

- *Package Name* — nastaviť podľa tabuľky B.1,
- *Product Name* — nastaviť s príslušnou príponou kanála,
- *Version* — viditeľné číslo verzie (napr. 1.2.5),
- *Bundle Version Code* — inkrementovať v rámci rady kanála, musí byť vyšší ako predchádzajúci upload.

2. Build Settings:

- *Development Build* — **vypnúť**.

3. XR Plugin Management:

- overiť, že OpenXR plugin je aktívny pre platformu Android.

4. Keystore (Publishing Settings):

- vybrať správny .keystore súbor pre daný kanál,
- zadať alias a heslo.

5. Vizuálna identita:

- overiť, že ikona (logo) a banner zodpovedajú príslušnému kanálu.

Build sa zostaví cez *File → Build Settings → Android → Build*. Výstupom je súbor .apk alebo .aab.

B.5.4 Pre-upload checklist

Pred nahraním buildu do MQDH je nutné potvrdiť všetky nasledujúce body:

- ☐ Správne package name podľa cieľového kanála
- ☐ Správna prípona product name ([LTS], [STABLE], [RC])
- ☐ Správne viditeľné číslo verzie
- ☐ Bundle Version Code inkrementovaný oproti poslednému uploadu
- ☐ Build podpísaný správnym keystore pre daný kanál
- ☐ Development mode vypnutý
- ☐ OpenXR plugin aktívny

B.5.5 Upload cez Meta Quest Developer Hub

1. Otvoriť **MQDH** a prihlásiť sa Meta účtom s prístupom do správnej developer organizácie.
2. Prejsť do sekcie *App Distribution* a zvoliť správnu organizáciu a aplikáciu.
3. Zvoliť cieľový release kanál (beta branch).
4. Kliknúť na *Upload* a zvoliť príslušný súbor .apk alebo .aab.
5. Pred potvrdením overiť všetky body z checklistu (sekcia **B.5.4**).
6. Dokončiť SMS verifikáciu (vyžaduje prístup k telefónu).
7. Počkať na spracovanie buildu platformou.
8. Overiť, že build sa objavil v správnom kanáli.

B.5.6 Verifikácia na zariadení

Po úspešnom uploade je potrebné overiť build priamo na Meta Quest zariadení:

1. Nainštalovať alebo aktualizovať aplikáciu na zariadení.
2. Overiť, že názov aplikácie obsahuje správnu príponu kanála.
3. Overiť základnú funkčnosť aplikácie.

B.5.7 Build promotion (RC → STABLE)

Postup presunu buildu z testovacieho kanála do stabilného:

1. Otestovať RC build na cieľových zariadeniach.
2. Ak sa objavia kritické chyby, vytvoriť nový RC build so zvýšeným Bundle Code a testovanie zopakovať.
3. Po úspešnom overení pripraviť rovnaký stav kódu ako STABLE build:
 - zmeniť package name na `com.lirkis.neurorehabvr.stable`,
 - zmeniť product name na `NeuroRehabVR [STABLE]`,
 - inkrementovať Bundle Version Code v rade STABLE (11000+).
4. Nahrať STABLE build do príslušného release kanála podľa postupu v sekcii **B.5.5**.
5. Doplniť release notes.

B.5.8 Riešenie problémov

Bundle Version Code conflict — **MQDH** odmietne build so zhodným alebo nižším Bundle Code. Riešenie: zvýšiť kód, znovu zostaviť, upload zopakovať.

24-hodinový lockout účtu — chyba *too many requests*. Riešenie: počkať 24 hodín, neskúšať opakovane nahrávať.

Cooldown po zmazaní aplikácie — po zmazaní app line nie je možné znovu použiť rovnaké package name niekoľko dní. Riešenie: aplikácie nemazať, ponechať ako zmrazený *fallback*.

Strata keystore — bez správneho podpisovacieho kľúča nie je možné aktualizovať daný app line (poučenie z kanála **LTS**).

B.5.9 Správa keystore

Keystore súbory, aliasy a heslá nesmú byť uložené v repozitári. Musia byť evidované v bezpečnom úložisku spolu s informáciou, ku ktorému app line patria.

- Každý kanál má vlastný keystore.
- Keystore je zviazaný s package name — strata znamená nemožnosť aktualizovať daný app line.
- Súbor `.gitattributes` chráni deployment identifikátory (package name, product name) pred nechceným prepísaním pri mergovaní vetiev.
- Pri zakladaní nového app line alebo verzie „V2“ musí byť správa keystore zavedená od prvého dňa vývoja.

B.6 REST API dokumentácia

Systém obsahuje prístup pomocou REST API k serveru zvonku. API obsahuje niekoľko referencií. POST Request bude vždy úspešný, ak sa podarí kontaktovať server. Mení sa response body. Hodnota “result” je rovná “true” len v prípade, že sa naozaj podarí vykonať funkciu spustenú POST requestom. Na obr. **B.3** je vyobrazená referencia na spustenie animácie počas tréningu. Na obr. **B.4** je request pre obyčajný pohyb, tento je možné vykonať pokiaľ nebeží tréning. Request na obr. **B.5** zmení aktuálny typ animácie na “Off” a tým skryje objekty určené na animácie. Request `/spawn` slúži na nastavenie typu animácie, typ animácie sa prepne do posledného typu animácie. (obr. **B.6**).

POST **/training/move** Progress move during training

If training is running, will progress step and show animation

Responses

Code	Description	Links
200	Successful connection to server, result is true when move is successfully started. Otherwise it's false. Media type application/json Controls Accept header. Example Value Schema <pre>{ "result": true }</pre>	No links
400	Failed to connect	No links

Obrázok B.3: API dokumentácia - /training/move

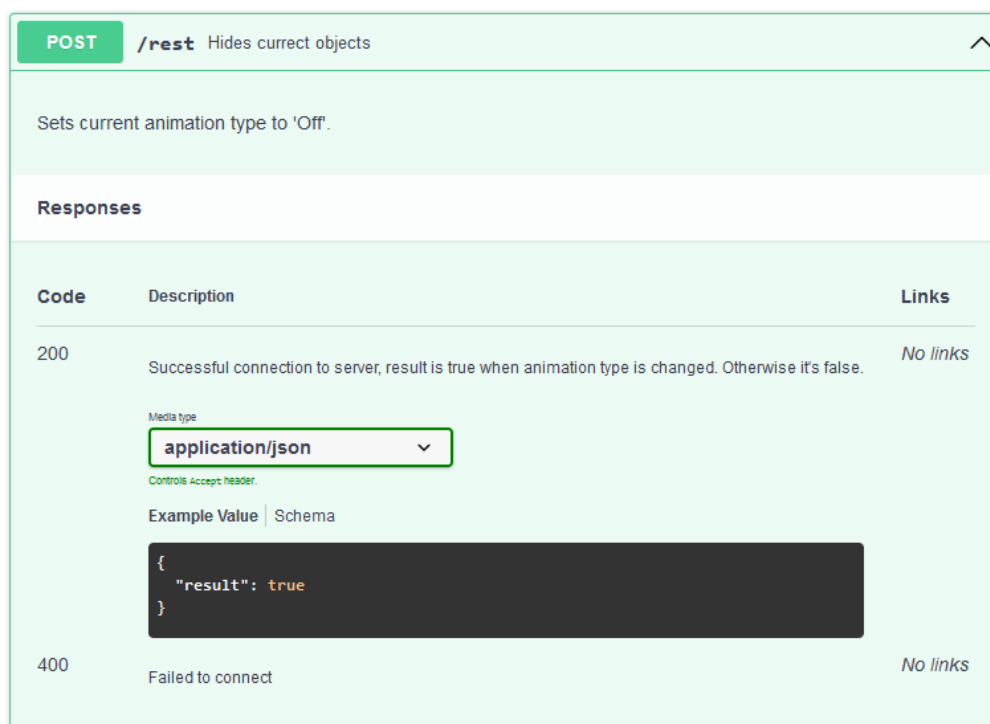
POST **/move** Show animation outside of training sequence

If training is not running, will show animation

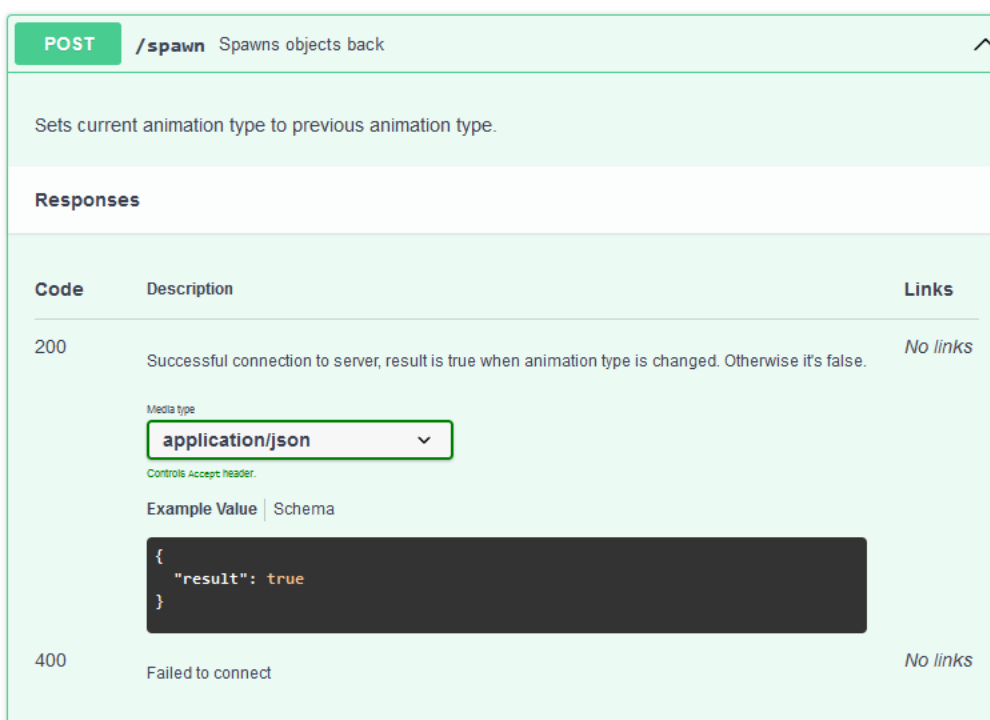
Responses

Code	Description	Links
200	Successful connection to server, result is true when move is successfully started. Otherwise it's false. Media type application/json Controls Accept header. Example Value Schema <pre>{ "result": true }</pre>	No links
400	Failed to connect	No links

Obrázok B.4: API dokumentácia - /move



Obrázok B.5: API dokumentácia - /rest



Obrázok B.6: API dokumentácia - /spawn

B.7 Referenčná dokumentácia WebSocket protokolu

Táto časť obsahuje referenčnú dokumentáciu WebSocket protokolu používaného serverom NeuroRehabVR. Slúži ako integračná špecifikácia pre externých kon-

zumentov, predovšetkým OpenViBE/EEG klienta. Architektúra, návrh a zdôvodnenie rozhodnutí sú popísané v kapitole 3.

B.7.1 Pripojenie a životný cyklus

Adresa servera

WebSocket server počúva na adrese:

```
ws://{bindAddress}:{port}
```

Hodnoty sú odvodené z konfiguračného objektu `NetworkSettings`:

- **Port** — `WebSocketPort`, predvolená hodnota 7778. Možno prepísať parametrom príkazového riadku `-socketport {port}`.
- **Bind adresa:**
 - prázdna alebo medzera → `0.0.0.0`,
 - `localhost` → `127.0.0.1`,
 - inak sa použije explicitne nakonfigurovaná IP adresa.

Šifrovaný kanál `wss://` nie je v aktuálnej verzii podporovaný (viď sekciu 3.6).

Životný cyklus servera

1. Server sa spúšťa v `CustomNetworkManager.OnStartServer()` volaním `WebSocketManager.StartServer()`.
2. Pri štarte sa čistia všetky aktívne session objekty.
3. Server sa zastavuje v `CustomNetworkManager.OnStopServer()` alebo pri zničení objektu v `WebSocketManager.OnDestroy()`.

Transportné správanie

Transportná vrstva je implementovaná pomocou knižnice `Fleck` (viď sekciu 3.5).

- Prichádzajúce správy sú parsované ako **JSON** do obálky `WebSocketMessage`.
- Správy presahujúce `MaxMessageBytes` sú zahodené bez odpovede.
- Neplatný **JSON** je zahodený bez odpovede.

B.7.2 Formát správ

Všetky správy v oboch smeroch používajú rovnakú **JSON** obálku:

Výpis B.2: Štruktúra obálky WebSocket správy

```
{
  "roomId":    "room-123",
  "eventType": "client.checkin",
  "timestamp": 1700000000000,
  "payload": {
    "clientType": "eeg"
  }
}
```

Polia obálky

roomId (*string*, povinné) — identifikátor miestnosti; slúži na smerovanie a autorizáciu.

eventType (*string*, povinné) — typ udalosti; rozlišuje veľké a malé písmená; musí zodpovedať jednej z podporovaných udalostí (sekcia **B.7.4**).

timestamp (*number*) — unixové milisekundy. Nastavuje odosielateľ; server generuje vlastnú hodnotu pri odchádzajúcich správach. Server túto hodnotu nevaliduje.

payload (*object* alebo *null*) — telo správy. Povinné pre handlersy vyžadujúce typovaný obsah (viď sekciu **B.7.4**). Ak je prítomné, musí byť objekt.

Dvojfázová deserializácia

Transportná vrstva vždy najprv deserializuje obálku. Pole `payload` sa ponechá ako surový **JSON** objekt a deserializuje sa na konkrétny typ až po výbere handlera podľa `eventType`.

B.7.3 Relácie a autorizačné pravidlá

Požiadavka na check-in

Prvá správa odoslaná klientom **musí** byť `client.checkin`. Ak klient odošle akúkoľvek inú správu pred úspešným check-inom, server odpovie `server.error` a správu zahodí.

Viazanosť na miestnosť

Po úspešnom check-ine sa spojenie naviaže na jednu roomId. Ak klient neskôr odošle správu s odlišnou roomId, server odpovie `server.error` a správu zahodí.

Opätovné pripojenie

Ak sa rovnaké identifikačné spojenie prihlási opätovne, server ho považuje za *reconnect* a vracia stav `status = "reconnected"` v správe `client.checkin.ack`.

B.7.4 Katalóg udalostí

Táto sekcia uvádza kompletný zoznam podporovaných udalostí. Udalosť neuvedená v tomto zozname sa považuje za nepodporovanú a server na ňu odpovie `server.error`.

Klient → Server (prichádzajúce)

client.checkin Registrácia klienta do miestnosti a priradenie typu zariadenia.

Pole payloadu	Popis
<code>clientType</code> (<i>string enum</i>)	"patient" "therapist" "eeg"

Odpoveď: `client.checkin.ack` pri úspechu; `server.error` pri zlyhaní validácie.
Validačné pravidlá:

- payload musí byť prítomný,
- `clientType` nesmie byť `Unknown`,
- `roomId` musí existovať na serveri.

mi.stream.fatigue Odoslanie priebežnej EEG/MI hodnoty na server.

Pole payloadu	Popis
<code>value</code> (<i>float</i>)	nameraná EEG/MI hodnota

Odpoveď: žiadna pri úspechu; `server.error` pri neplatnom vstupe alebo chýbajúcej autorizácii.

Validačné pravidlá:

- payload musí byť prítomný,
- `roomId` musí existovať na serveri.

exercise.action.move Spustenie jedného kroku cvičenia v miestnosti.

Pole payloadu	Popis
---------------	-------

<i>prázdny objekt alebo vynechaný</i>	
---------------------------------------	--

Odpoveď: žiadna pri úspechu; `server.error` pri neplatnom vstupe alebo chýbajúcej autorizácii.

Validačné pravidlá:

- `roomId` musí existovať na serveri,
- ak je `payload` prítomný, musí byť prázdny objekt (`{}`).

Server → Klient (odchádzajúce)

client.checkin.ack Potvrdenie registrácie klienta.

Pole payloadu	Popis
<code>clientType</code> (<i>string enum</i>)	"patient" "therapist" "eeg"
<code>status</code> (<i>string enum</i>)	"connected" "reconnected"

mi.threshold.set Nastavenie aktuálnej MI threshold hodnoty pre miestnosť. Server túto správu odosiela automaticky po úspešnom check-ine EEG klienta, bezprostredne po `client.checkin.ack`.

server.error Štandardizovaná chybová odpoveď.

Pole payloadu	Popis
<code>error</code> (<i>string enum</i>)	"generic"
<code>type</code> (<i>string</i> , voliteľné)	eventType, ktorý chybu spôsobil
<code>description</code> (<i>string</i>)	textový popis chyby

exercise.rest.start / exercise.rest.stop Notifikácia klientov v miestnosti o začiatku, resp. ukončení odpočinkovej fázy. Payload je prázdny objekt.

exercise.eval.start / exercise.eval.stop Notifikácia klientov v miestnosti o začiatku, resp. ukončení fázy EEG vyhodnocovania. Payload je prázdny objekt.

B.7.5 Spracovanie chýb

Server odpovedá správou `server.error` v nasledujúcich prípadoch:

- chýbajúci alebo nepodporovaný `eventType`,
- neplatný alebo chýbajúci `payload` pre typované handlers,
- neplatná alebo neexistujúca `roomId`,
- akákoľvek správa odoslaná pred úspešným check-inom,
- *cross-room* správa (nesúladi `roomId` so `session`),
- validačné alebo prevádzkové chyby handlera.

Transportné chyby **negenerujú** odpoveď:

- neplatný **JSON**,
- správa presahujúca `MaxMessageBytes`.

B.7.6 Limity a výkonnosť

Prevádzkové limity sú definované v `NetworkSettings.Adapter`:

Tabuľka B.2: Prevádzkové limity WebSocket servera

Parameter	Hodnota	Popis
<code>MaxMessageBytes</code>	1 MB	maximálna veľkosť jednej správy
<code>MaxQueueActionsPerFrame</code>	500	max. akcií spracovaných za snímok
<code>QueueWarnThreshold</code>	2 000	varovná hranica veľkosti frontu

Prichádzajúce správy sú dispečované v hlavnom vlákne Unity s časovým rozpočtom ~5 ms na snímok. Pri záťaži sa správy odkladajú do nasledujúcich snímkov.

B.7.7 Príklady správ

Výpis B.3: Check-in — požiadavka

```
{
  "roomId": "room-123",
  "eventType": "client.checkin",
```

```
"timestamp": 1700000000000,  
"payload": {  
  "clientType": "eeg"  
}  
}
```

Výpis B.4: Check-in — potvrdenie

```
{  
  "roomId": "room-123",  
  "eventType": "client.checkin.ack",  
  "timestamp": 1700000000100,  
  "payload": {  
    "clientType": "eeg",  
    "status": "connected"  
  }  
}
```

Výpis B.5: EEG/MI stream

```
{  
  "roomId": "room-123",  
  "eventType": "mi.stream.fatigue",  
  "timestamp": 1700000000200,  
  "payload": {  
    "value": 0.42  
  }  
}
```

Výpis B.6: Spustenie kroku cvičenia (prázdny payload)

```
{  
  "roomId": "room-123",  
  "eventType": "exercise.action.move",  
  "timestamp": 1700000000300,  
  "payload": {}  
}
```

Výpis B.7: Serverová notifikácia o začiatku odpočinku

```
{  
  "roomId": "room-123",  
  "eventType": "exercise.rest.start",  
  "timestamp": 1700000000400,
```

```
"payload": {}  
}
```

Výpis B.8: Chybová odpoveď

```
{  
  "roomId": "room-123",  
  "eventType": "server.error",  
  "timestamp": 1700000000500,  
  "payload": {  
    "error": "generic",  
    "type": "mi.stream.fatigue",  
    "description": "Check-in required before sending messages."  
  }  
}
```

C Implementačné detaily logovacieho systému

Táto príloha obsahuje zdrojové kódy kľúčových implementačných metód logovacieho systému opísaného v kapitole 4. Rozhranie `ILogger`, enumerácia `LogChannel` a príklady výstupu zostávajú v hlavnom texte kapitoly.

C.1 Formátovanie a publikovanie záznamu

Metóda `Emit()` vo fasáde `GameLog` vytvorí jednotný textový tvar správy, v chránenom bloku deleguje zápis na aktuálnu implementáciu `ILogger` a následne publikuje udalosť `EntryEmitted` pre používateľské rozhranie.

Výpis C.1: Formátovanie a publikovanie záznamu vo fasáde `GameLog`

```
private static void Emit(
    LogChannel channel,
    string severity,
    string message,
    Object context,
    string memberName,
    string filePath,
    Action<ILogger, LogChannel, string, Object> writer
) {
    string formattedMessage = Format(
        context, memberName, filePath, message);
    try {
        _impl?.Let(writer, channel, formattedMessage, context);
    } catch (Exception ex) {
        WriteFallback(
            severity, channel, formattedMessage, context, ex);
    }
    PublishEntry(new GameLogEntry(
```

```
        DateTimeOffset.Now,  
        severity,  
        channel,  
        formattedMessage,  
        context));  
}
```

C.2 Thread-safe zápis do súborového loggeru

Metóda `Write()` pridá záznam do synchronizovaného bufferu `FileLoggera`. Varovania, chyby, Android buildy a batch režim vyvolajú okamžitý **flush**; informácie sa zapisujú po prekročení konfigurovateľného prahu bufferu.

Výpis C.2: **Thread-safe** zápis do bufferu `FileLoggera`

```
private void Write(  
    string severity,  
    LogChannel channel,  
    string message,  
    bool flushImmediately = false  
) {  
    lock (_syncRoot) {  
        _pendingEntries.Add(  
            FormatEntry(severity, channel, message));  
  
        if (flushImmediately  
            || _flushOnEveryWrite  
            || _pendingEntries.Count >= _flushThreshold) {  
            FlushLocked();  
        }  
    }  
}
```

C.3 Flush mechanizmus

Metóda `FlushLocked()` kontroluje obsah bufferu pred zápisom, spája čakajúce riadky a zapisuje ich na disk cez `File.AppendAllText()`. Buffer sa vymaže až po úspešnom zápise, čím sa znižuje riziko straty dát pri zlyhaní jednej z mnoha súborovej operácie. Po úspešnom zápise sa kontroluje aj limit veľkosti log adresára.

Výpis C.3: Bezpečný flush mechanizmus s ochranou prázdneho bufferu

```
private void FlushLocked() {  
    if (_pendingEntries.Count == 0) {  
        return;  
    }  
  
    string content = string.Join(  
        Environment.NewLine, _pendingEntries)  
        + Environment.NewLine;  
  
    try {  
        File.AppendAllText(_filePath, content);  
        _pendingEntries.Clear();  
        EnforceStorageLimit();  
    } catch (Exception ex) {  
        ReportInternalFailure(  
            $"[GameLog] Failed to append logs "  
            + $"to '{_filePath}': {ex}");  
    }  
}
```