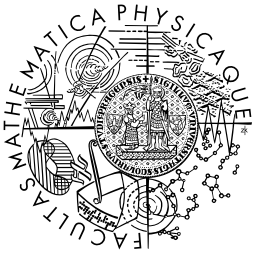


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Matěj Hrbáček

Řízení osvětlovacího systému divadla

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D.
Studijní program: Informatika
Studijní obor: Programování a softwarové systémy

Praha 2020

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Chtěl bych poděkovat svému vedoucímu Mgr. Pavlu Ježkovi, Ph.D. za jeho čas, rady a hlavně pomoc, kterou mi poskytoval při vypracování této práce. Zároveň bych chtěl poděkovat hlavnímu autorovi osvětlení mému bratrovi Zdeňkovi Hrbáčkovi a i celé rodině a všem, kteří mě při studiu podporovali.

Název práce: Řízení osvětlovacího systému divadla

Autor: Matěj Hrbáček

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: V divadle „Ochotnický soubor Lípa“ je potřeba ovládat na míru vyrobené osvětlení komunikující přes UDP datagramy. Cílem práce je vytvořit systém, ve kterém bude možné tato světla ovládat přes počítačovou aplikaci.

Knihovna ObjectsForLights, vzniklá v rámci systému, je přenositelná a slouží jako logický základ pro uživatelské aplikace. Knihovna obsahuje moduly klienta a server. Systém je vybudován centralizovaně, kdy server je společným bodem pro komunikaci jak s klienty, tak i se zařízeními. Moduly server a klient se starají o udržování dat pro zařízení, komunikaci mezi sebou a server i o ovládání osvětlení.

Nad moduly klient a server vznikly počítačové aplikace, které umožňují ovládat jednotlivá zařízení. Aplikace poskytují pro jednotlivá zařízení grafické ovládací prvky odpovídající druhu ovládaného zařízení, možnost přiřazovat jednotlivým zařízením názvy a vytvářet si z ovládaných zařízení skupiny ovládající více zařízení a komunikovat mezi moduly. Data pro zařízení je pak možné ukládat a načítat ze souborů.

Klíčová slova: systém ovládání osvětlení aplikace UDP server klient

Title: Theater lighting system control

Author: Matěj Hrbáček

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Ježek, Ph.D., Department of Distributed and Dependable Systems

Abstract: In the "Ochotnický soubor Lípa" theater, it is necessary to control custom-made lighting communicating via UDP datagrams. The goal of this theses is to create a system in which it will be possible to control these lights through a computer application.

The ObjectsForLights library, created within the system, is portable and serves as a logical basis for user applications. The library contains client and server modules. The system is built centrally, where the server is a common point for communication with both clients and devices. The server and client modules take care of maintaining data for devices, communication with each other. The server is also responsible for direct communication with controlled lights themselves.

Computer applications were created on top of the client and server modules, and enable user control of individual devices. Applications provide graphical controls for individual devices corresponding to the type of controlled device, provide ability to assign names to individual devices and create groups of controlled devices. Data for the device can then be saved and retrieved from files.

Keywords: system control lighting aplication UDP server client

Obsah

Úvod

Nezbytnou součástí moderního divadla je osvětlení, které divákovi pomáhá s orientací v pomyslném prostoru jeviště a ději divadelní hry. Osvětlovač se snaží pomocí světel vést divákovu pozornost správným směrem, ať už se jedná o vytvoření iluze různých míst a nebo se snaží před divákem skrýt technické detaily jako je například přestavba kulis. Nejlepším příkladem bude se podívat na jednu scénu nasvícenou třemi různými způsoby. Jak můžeme vidět na obrázku 1, je pomocí světel osvícené celé jeviště. Na obrázku 2 je nasvícena pouze zadní část jeviště a na obrázku 3 je nasvícena pouze malá část pro zvýraznění herce.

Obrázek 1: Celé jeviště.

Obrázek 2: Zdůraznění zadní části jeviště.

Obrázek 3: Zdůraznění malé části jeviště.

K tomu je potřeba velké škály světelných zařízení. Od základních světel, které svítí jednou barvou až po světla, která se pohybují a mohou měnit barvy. Aby osvětlovač mohl tato zařízení dostatečně flexibilně ovládat, potřebuje k tomu přirozený ovládací prostředek. Může se jednat o hardwarovou konzoli nebo počítačový program.

Právě problém s řízením osvětlení řeší ochotnické divadlo „Ochotnický soubor Lípa“ obnovené v roce 2013. Jelikož toto divadlo začínalo od nuly a nemělo finance na nákladné osvětlení a patřičný ovládací systém, muselo si vystačit s obyčejnými neprofesionálními zařízeními. Zpočátku divadlo používalo pouze halogenová světla ovládána pomocí jističů, kdy světla byla buď zapnuta nebo vypnuta. S postupným rozvojem divadla se rozvíjelo i technické zázemí a jeden z členů technického týmu začal pro divadlo vyrábět a dodávat levná světla a zařízení, jak můžeme vidět na obrázku 4.

Obrázek 4: Používaná zařízení

S těmito zařízeními je potřeba komunikovat tak, aby jim bylo možné sdělit

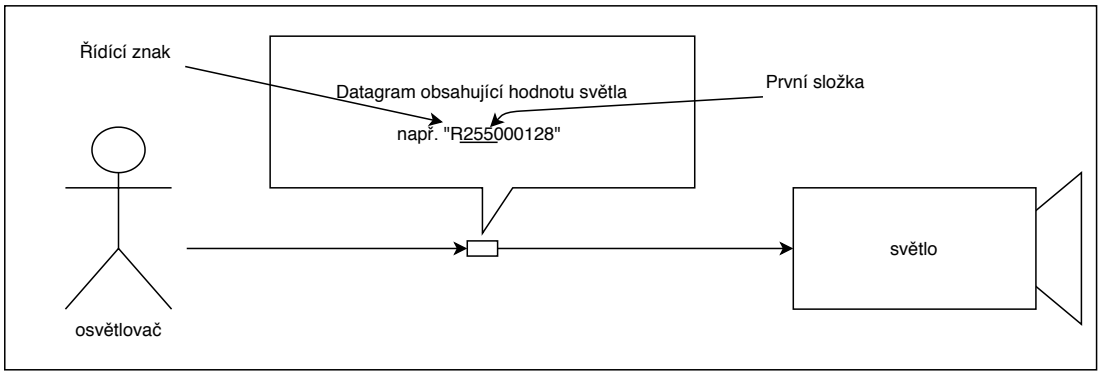
informace o tom, co právě mají udělat. Například jakou intenzitou mají svítit. Z toho důvodu má každé zařízení ethernetovou zásuvku a podporuje přijímání UDP datagramů.

Ve světě osvětlovací techniky již existují protokoly pro přenos dat přes UDP, například ART-NET [12], a existují i systémy pro ovládání osvětlení pracující nad těmito protokoly. Autor osvětlení se však rozhodl vytvořit si vlastní protokol pro komunikaci s jeho zařízeními a právě proto není možné použít pro jejich ovládání nějaký již existující systém.

Formát zpráv zařízení divadla Lípa Přenášené zprávy v systému ochotnického souboru Lípa jsou ve formátu:

- 0. byte obsahuje řídící znak, který určí jak mají být data prezentována.
- Byty 1.-n. představují data, která jsou v textové podobě. To znamená, že každý byte (hodnoty 0-255) přenášené informace je při přenosu reprezentován třemi byty, kde první byte představuje stovky, druhý desítky a třetí jednotky.

Například bytová hodnota 12 je přenášena jako textový řetězec "012", jak můžeme vidět na obrázku 5. Autora zařízení k tomuto kroku vedly jednak technické důvody, ale také jednoduché posílání a kontrola dat při vývoji.



Obrázek 5: Schéma komunikace.

Zařízení po příjmu datagramu ověří řídící znak a zjistí, jestli je schopné reprezentovat jeho data. Pokud není, ignoruje ho. Pokud je, uloží si obdržené data a reprezentuje je. Tato zařízení zatím neumí vysílat zprávy, přijímají zprávy na pevně daném portu a pouze od první ovládací konzole, která jim zašle data. To je dáno tím, že zařízení si při první přijaté zprávě vytvoří síťový socket s IP adresou odesílatele zprávy a nadále komunikuje pouze přes tento socket. Pro změnu řídící jednotky je nutné zařízení restartovat. Toto jsou omezení zavedená autorem světel, která vznikly z použitého hardwaru, proto se jimi budeme muset řídit.

Každé zařízení má v sobě uložený svůj stav. *Stav zařízení* je pole bytů, podle kterých se světlo chová. Tyto byty budeme nazývat *složky zařízení*. Nejlepším vysvětlením bude představení nejdůležitějších zařízení v současném systému.

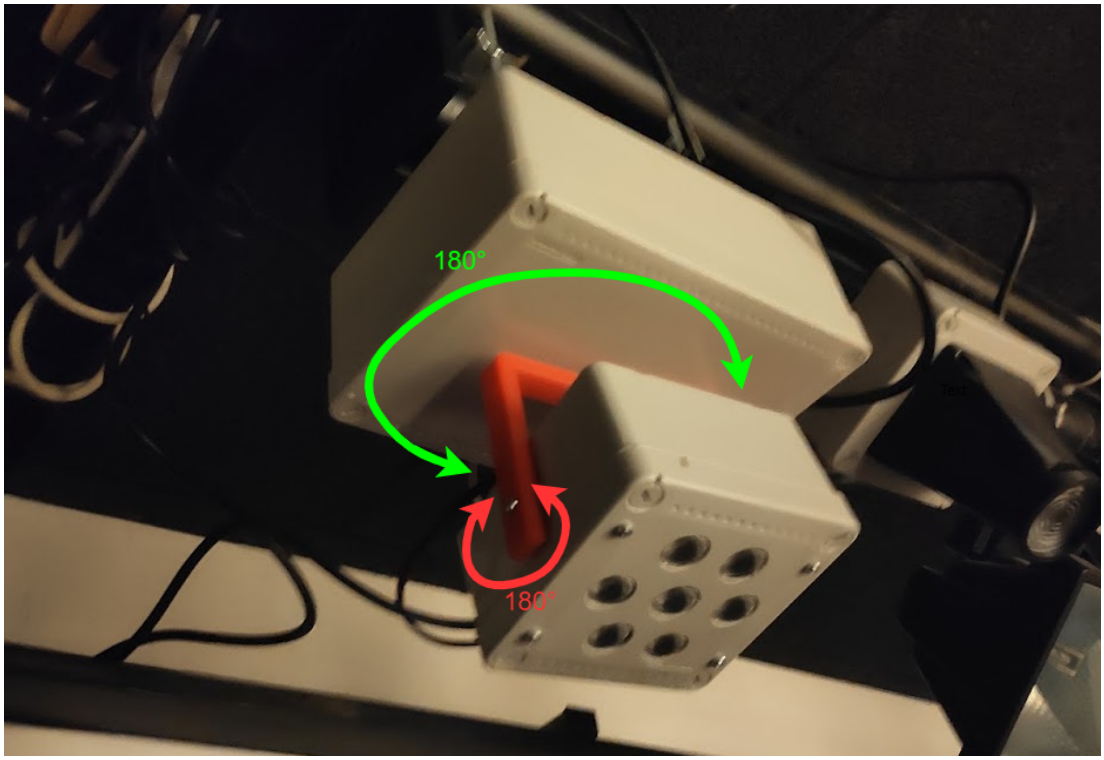
- Prvním vyrobeným zařízením je *stmívač* sloužící k ovládání intenzity světel. Jedná se o zařízení, které reguluje napájení do připojených zařízení a tak

umožňuje ovládat jejich výkon (intenzitu světla). Složka zařízení představuje jeden z jeho výstupů a udává jeho výkon. Na obrázku 6 můžeme vidět zapojený stmívač v rozvodné skříni. Současný stmívač regulující 12 výstupů se tedy skládá ze 12 složek a v ovládací zprávě očekává 12 hodnot k nastavení.

Obrázek 6: Stmívač.

- Druhým vyrobeným zařízením je *RGB světlo* složené z RGB led-diod. Světlo si můžeme prohlédnout na obrázku 8. Složky reprezentují intenzitu červené, zelené a modré barvy. V ovládací zprávě očekává 3 hodnoty k nastavení. Díky kombinacím barev je schopné svítit téměř jakoukoliv barvou.

- Třetím výrobkem je *pohyblivé světlo*. Jedná se o rozšíření RGB světla o možnost rotace ve dvou v osách tak, aby bylo schopné osvítit celou polokouli, jak je vidět na obrázku 7. Složky zařízení reprezentují intenzitu červené, zelené, modré barvy a stupeň otočení v osách. Stupně jsou omezeny na hodnoty 0-180, kdy základní poloha je 90. Zařízení v ovládací zprávě očekává 5 hodnot k nastavení, tři pro barvy a dvě pro osy.



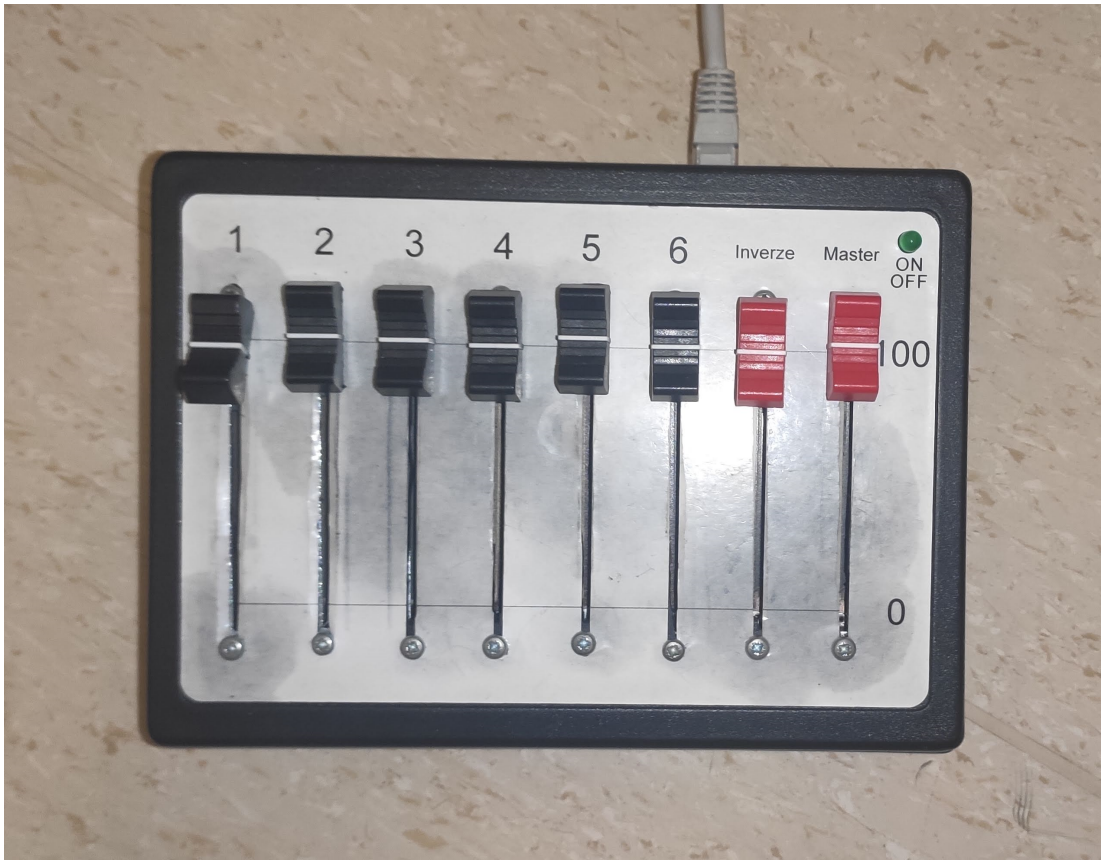
Obrázek 7: Pohyblivé světlo.

- Ostatní zařízení jsou zajímavá spíše po konstrukční stránce a jsou řízena stmívačem. Jedná se například o bodová světla, která pomocí optiky směřují celý svůj výkon na určitou plochu, jak můžeme vidět na obrázku 8.

Obrázek 8: Používaná zařízení

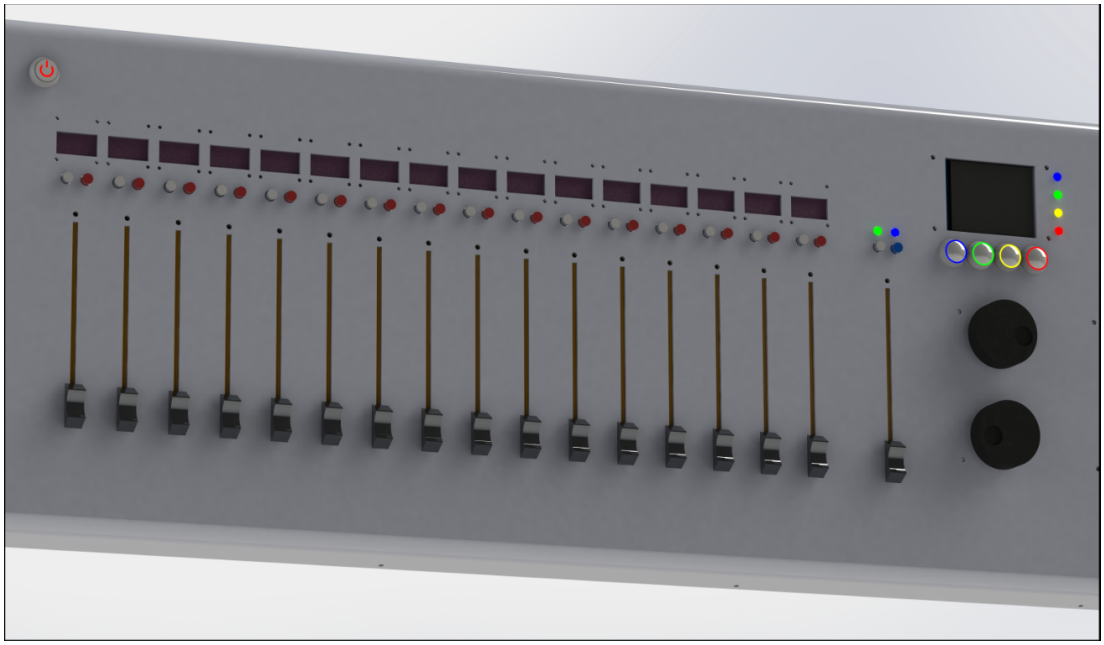
V současné době jsou tato zařízení řízena hardwarovou konzolí, kterou můžeme vidět na obrázku 9. Tato konzole umožňuje nastavovat pomocí fejdrů maximálně 6 hodnot. Tyto hodnoty jsou pak posílány zařízením, která jsou určena ve zdrojovém kódu, který vykonává konzole. Konzole neumí přijímat data, proto

musí být hlavní a jedinou konzolí v systému. Pro změnu ovládaných zařízení je nutné konzoli odpojit ze systému a přehrát zdrojový kód konzole, ve kterém jsou napevno nastavené data o zařízeních. To znamená, že při každé změně divadelní hry nebo změně zařízení je nutné, aby autor systému konzoli odpojil a přehrál její zdrojový kód.



Obrázek 9: Současná řídící konzole.

Plánovaným rozšířením je hardwarová konzole vyrobená autorem současných zařízení. Návrh vzhledu plánované konzole můžeme vidět na obrázku 10. Od současné konzole se má lišit tím, že bude podporovat přijímání a odesílání zpráv přes UDP. Tudíž se bude moci zapojit do komunikace více ovladačů a bude možné jednoduše měnit řízená zařízení, či případně přenést veškerou logiku jinam a konzoli ponechat jen jako zařízení pro nastavování hodnot.



Obrázek 10: Model plánované řídící konzole.

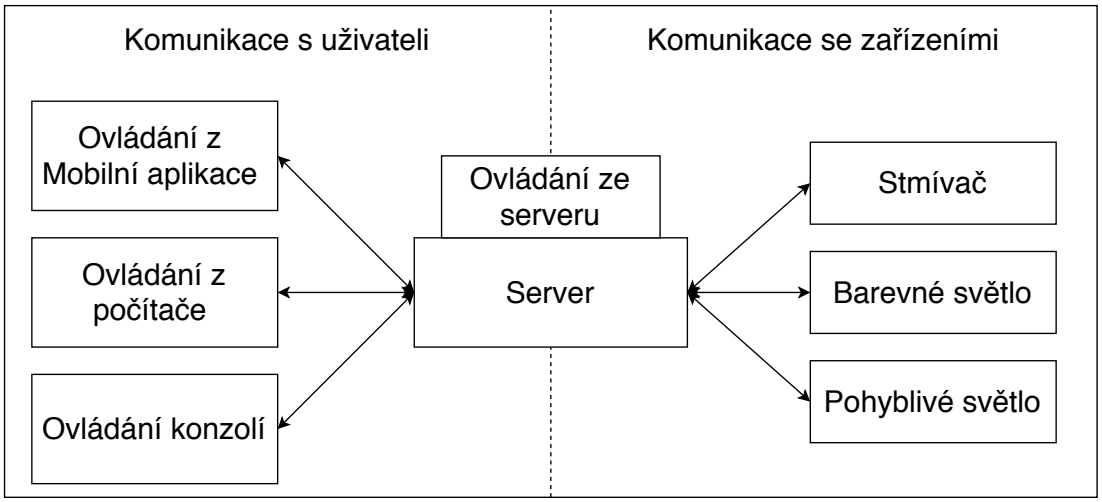
Hlavním cílem této práce je vytvořit počítačový systém na ovládání osvětlení, který bude podporovat ovládání z více konzolí. Oproti stávajícímu ovládání bude ulehčovat přidávání zařízení a ovládacích prvků.

1. Požadavky na systém

Naším cílem je vytvořit systém, který umožní technickým pracovníkům divadla co nejjednodušší ovládání osvětlení. Proto je důležité si uvědomit, jak by měl takový systém vypadat, pro koho je určen a co by měl podporovat. K tomu nám nejlépe pomůže předvést si požadavky se současnými zařízeními a v současném technickém stavu divadla.

1.1 Server ovládající zařízení

Jako první se musíme rozhodnout jak bude vypadat rozdělení systému a kdo s kým bude komunikovat. Světla jsou schopna komunikovat pouze s jedním zařízením, proto vytvoříme centralizovaný systém, kde veškerou komunikaci se světly ale i s ovladači bude obstarávat server. Server bude tvořit jediný spoj mezi uživatelskou částí a částí zařízení. Díky tomu server bude moc udržovat validní data pro zařízení a řešit konflikty mezi ovladači. Komunikace mezi serverem a světly bude realizována pomocí ethernetového kabelu spojujícího technickou místnost a switch, který je jen pro komunikaci v rámci tohoto systému. Komunikace mezi uživatelem a serverem bude realizována v rámci stejné sítě jako komunikace se světly, ale navíc bude možné použít soukromou divadelní wifi.



Obrázek 1.1: Představa rozložení systému.

Realizace systému

Prvním nápadem na realizaci serveru bylo zakoupit minipočítač, který by byl neustále zapojený v systému. To se po důkladnější úvaze zavrhlo a domluvilo se, že server bude realizován jako počítačový program s jednoduchým grafickým rozhraním a hlavně s možností ovládání osvětlení přímo z něj. Kromě toho, že počítač je uživatelsky přívětivější a výkonnější, hlavním důvodem byla úvaha, že k ovládání osvětlení vždy bude potřeba alespoň jeden počítač a pro osvětlovače bude jednodušší spustit program na svém počítači. Navíc uživatel s počítačem představující server bude mít plnou kontrolu nad systémem.

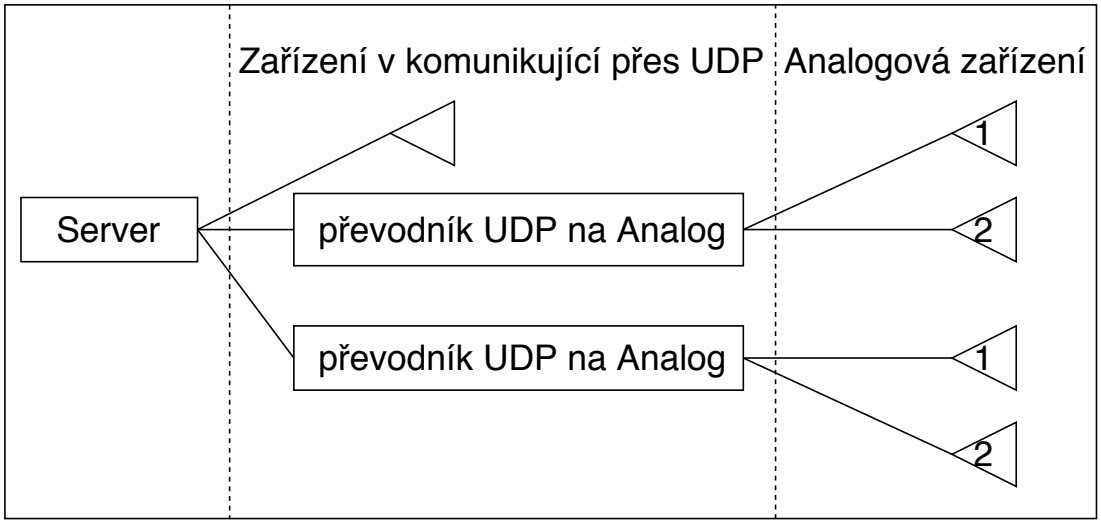
Požadavky na komunikaci

Komunikace serveru s uživateli by měla probíhat tak, že se uživatel přihlásí k serveru. Uživatel přihlášený k serveru automaticky dostane seznam zařízení a to tak, aby by jim mohl aktivně nastavovat hodnoty. Změna hodnot jednoho uživatele by se měla okamžitě projevit u ostatních uživatelů a nastavením dat u světel. Uživatelé by měli mít možnost navázat mezi sebou textovou komunikaci pro případné upozornění na technické či organizační problémy.

V současné situaci zařízení neodesílají žádné zprávy. To znamená, že serveru bude stačit pouze odesílat zařízením jejich příslušné hodnoty. V budoucnu by mohla přibýt možnost zpětné vazby, proto by na ni měl server být připraven reagovat.

Identifikace

V divadle se nachází více zařízení a je tedy nutné je během komunikace od sebe jednoznačně odlišit. Doposud stačilo jednotlivá zařízení odlišovat jen pomocí jejich IP adres. To se ale může stát časem nedostatečné. Pokud bychom se například v divadle rozhodli využívat převodník z UDP zpráv na analogové signály, který by dokázal přijmou UDP zprávu a dále ji distribuovat jednomu ze svých analogových zařízení, vznikla by potřeba na straně serveru jednoznačně určit toto analogové zařízení. Jak můžeme vidět na obrázku 1.2. To by však nešlo jen za pomoci IP adresy. Proto musí být v novém systému zavedena jemnější identifikace.



Obrázek 1.2: Komunikace s analogovými zařízeními přes převodník.

Zabezpečení

V systému by měla být i určitá forma zabezpečení. Ochrana proti promyšlenému útoku by byla příliš složitá. Případnému narušiteli nelze zabránit ve fyzickém útoku, například kdyby vyhodil pojistky pro celé divadlo či areál. Připojení do ethernetové sítě, ve které jsou zapojena světla, je značně omezené přístupovými body, které jsou přístupné pouze z jeviště a nebo v technické místnosti. Pro připojení k wifi síti je nutné znát heslo.

Ochrana by měla spíše být směřována proti lehkovážnému členu souboru nebo zvídavému návštěvníkovi vlastnící aplikaci. Osoba, která se dostane do wifi sítě nebo do ethernetové sítě, by měla být členem divadla, nebo blízká osoba, proto by mohla zjistit i heslo do systému. Zabezpečení bude probíhat pouze formou hesla v textové podobě. Heslo bude vyžadováno pouze při přihlašování do systému. Při změně hesla budou kontrolováni jen noví uživatelé. Uživatelé, kteří jsou již přihlášení, znovu kontrolováni nebudou.

1.2 Ovládání zařízení v reálném čase

V tomto systému je důležitou částí ovládání osvětlení v reálném čase. To znamená poskytnout osvětlovači rozhraní, přes které může dostatečně flexibilně nastavovat hodnoty jednotlivým zařízením. Jelikož se jedná o počítačovou aplikaci a ne o hardwarové zařízení, je nutné zvážit vhodnou reprezentaci ovládání jednotlivých prvků.

Na hardwarových konzolích lze přirozeně pohybovat více fejdry najednou a tím nastavovat hodnoty pro více jednotlivých zařízení. To ale na počítači není možné ani s dotykovou obrazovkou, protože uživatelům chybí fyzický odpor. Na druhou stranu v počítačové aplikaci není problém k ovládání používat ovladače, které lépe reprezentují požadovaný efekt.

Například pro RGB světlo musí hardwarová konzole reprezentovat každou složku jedním fejdrem. To v reálné situaci znamená, že pro jedno zařízení je využito více fejdrů, které jsou na hardwarové konzoli z technických důvodů značně omezená, ale hlavně to znemožňuje osvětlovači nastavit plynule požadovanou barvu. Ale na počítači by neměl být problém vygenerovat specifické ovládací prvky jako je například paleta barev pro RGB světlo.

Konflikty více uživatelů

V původním systému není žádná podpora pro ovládání zařízení z více konzolí, proto není jasné, jaké z toho mohou vznikat problémy. Stále se očekává že po celou dobu hry bude pouze jeden osvětlovač. Případným použitím dalšího ovládání by mohlo být například ovládání pro kulisáky, aby si v případě nutnosti přisvítili na jeviště při přestavbě kulis, či pro případné upravení světel, které není možné vidět ze stanoviště osvětlovače. Očekávaným problémem je konflikt hodnot, který nastane, pokud se více osvětlovačů bude snažit nastavit různé hodnoty jednomu zařízení. To znamená, že server obdrží od více konzolí pro jedno světlo různé hodnoty, ty bude světlu posílat a světlo bude blikat.

Proto je nutné zavést určitou hierarchii pro určení, čí hodnoty mají přednost. Každý ovládač bude mít pro každé zařízení nastavenou důležitost, nějaké malé číslo, se kterou se pokusí světlu změnit hodnotu. Pokud server obdrží data pro jedno světlo od více ovladačů, použije data s největší důležitostí. Tento princip bude fungovat na tom, že osvětlovači jsou kolegové a nebudou si tedy záměrně škodit, ale naopak, sami se budou snažit, aby systém fungoval co nejlépe.

1.3 Data v systému

V tomto systému je důležitá práce s daty. Nejdůležitějšími daty pro běh systému jsou informace o zařízeních. Pro správnou komunikaci mezi serverem a zařízeními je nutné, aby server znal pro každé zařízení protokol, kterým je ho možné ovládat. Proto bude potřeba, aby systém uměl načíst konfigurační soubor, ve kterém bude seznam světel a typ jejich komunikačního protokolu.

Díky tomu, že seznam světel bude v nezávislém konfiguračním souboru, bude možné jednoduše přidávat, odebírat zařízení a měnit jejich protokoly.

Skupina zařízení

Při divadelním představení je často důležité ovládat více světel najednou. Typickým příkladem je nasvícení jeviště červeným světlem. Na to, aby bylo celé jeviště nasvíceno červeně, jedno barevné světlo nestačí. Pokud je použito více barevných světel, je důležité, aby všechna barevná světla svítila stejnou barvou současně. Z toho důvodu je potřeba v systému vytvářet skupiny stejných světel. To znamená, že na virtuální konzoli přibude ovladač, který bude nastavovat hodnoty všem členům skupiny.

Skupina složek zařízení

V některých případech je nutné použít místo skupiny zařízení jen skupinu složek zařízení. Například je-li potřeba osvítit pouze malou část jeviště, chtěli bychom si vytvořit skupinku halogenových světel, jenže ta jsou řízena stmívačem.Pokud bychom nastavili všechny jeho složky, pak by byly osvíceny i nechtěné části jeviště. Použít pro každé halogenové světlo vlastní stmívač je prakticky nemožné. Proto je nutné dát osvětlovači možnost vybrat si ze všech možných zařízení jen jejich složky a vytvářet z nich skupiny. Jelikož se jedná o skupiny složek, není možné jim vytvořit lepší reprezentaci než obyčejný fejdr.

Inverze

U skupin složek dává smysl nastavit, že některé složky budou mít inverzní hodnotu vůči ovladači skupiny. Toho se například využívá, je-li potřeba vytvořit světelný efekt plynulého přechodu z jedné části jeviště na druhou, nebo při imitaci policejní sirény, kdy se plynule střídá červená a modrá barva.

Levelování

V některých případech je důležité, aby světla svítila vůči sobě v určitém poměru. Například chce-li osvětlovač vytvořit dojem, že z určité části jeviště vychází světlo. Pro tento efekt nasvítí onu část na maximální výkon, vedlejší část na poloviční a tak pokračuje až do ztracena. K tomu bude potřeba, aby šlo nadefinovat to, že složky ve skupině budou svítit vůči ovladači v určitém poměru.

Pojmenování

Při přípravě vystoupení je nutné zavést jednotnou terminologii pro jednotlivé části jeviště. Tento název se pak používá ve scénáři, na zkouškách, při technických

poradách. Například ve hře „Eva tropí hlouposti“ je jeviště pomyslně rozděleno na zámek a dům, a tak je i nasvíceno. Proto i jednotlivá světla, skupinky světel či skupiny složek světel by měly mít možnost dostat jednoduchý název pro snadnější orientaci.

1.4 Uživatelské rozhraní

Pro prezentaci dat uživateli a pro umožnění nastavování dat bude nutné vytvořit grafické rozhraní. Pro systém by měly vzniknout dvě aplikace, kdy jedna bude představovat konzoli připojenou k serveru a druhá server samotný. Obě dvě aplikace by měly umožňovat ovládat zařízení stejně jako konzole.

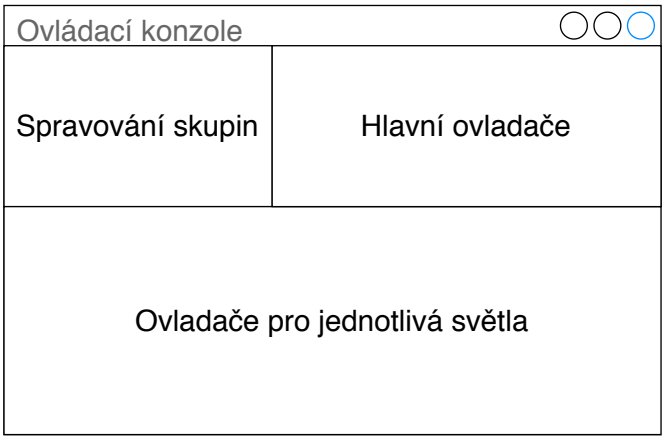
1.4.1 Virtuální ovládací konzole

Důležitou částí grafického rozhraní bude vytvořit virtuální ovládací konzoli sloužící k ovládání zařízení stejně tak jako hardwarová konzole. Virtuální konzole, jak můžeme vidět na obrázku 1.3, bude rozdělena na tři podčásti:

- *Ovládání pro všechna zařízení.*
  Mělo by se jednat o lištu ovládacích prvků, pomocí kterých se budou nastavovat hodnoty složkám zařízení a tak je ovládat v reálném čase.

- *Ovladače pro všechna zařízení určitého druhu.*
  Mělo by se jednat o lištu ovládacích prvků, kde každý ovladač nastaví hodnoty všem zařízením, které svými vlastnostmi odpovídají ovladači. Například osvětlovač pomocí ovladače pro všechna barevná světla nastaví hodnotu nejen všem barevným světlům, ale také barevné hodnoty všem pohyblivým světlům, protože pohyblivá světla svítí barevně.

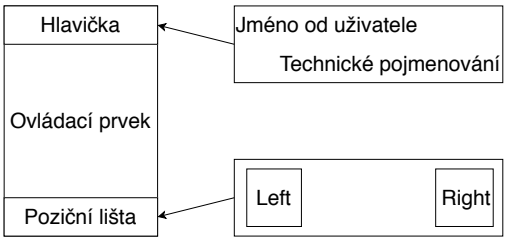
- *Podporu pro vytváření a ovládání skupin zařízení.*
  Mělo by se jednat o ovladač umožňující uživateli vytvořit si nebo nahrát již dříve vytvořené skupiny zařízení, či skupiny složek zařízení a přidat si je do lišty s ovládacími prvky. Ovladač skupiny nebo skupiny složek bude nastavovat hodnoty členům skupiny.



Obrázek 1.3: Náčrt ovládací konzole.

Kostra ovládacího prvku

Každý ovládací prvek bude mít hlavičku a poziční lištu. Hlavička se bude skládat z pojmenování zařízení a z technického jména. Pojmenování zařízení bude zvoleno uživatelem, pokud pojmenování nebude zvoleno, bude použit identifikátor. Technické jméno u ovladače pro jedno zařízení bude tvořit identifikátor zařízení případně alespoň jeho část, u skupiny bude technické jméno tvořit výčet identifikátorů členů skupiny. Poziční lišta se bude skládat z tlačítek doleva a doprava. Při kliknutí na tlačítko se ovládací prvek posune zvoleným směrem. Mezi hlavičkou a pozicní lištou bude ovladač sloužící k nastavování hodnot. Návrh kostry ovládacího prvku můžeme vidět na obrázku 1.4.



Obrázek 1.4: Kostra ovládacího prvku.

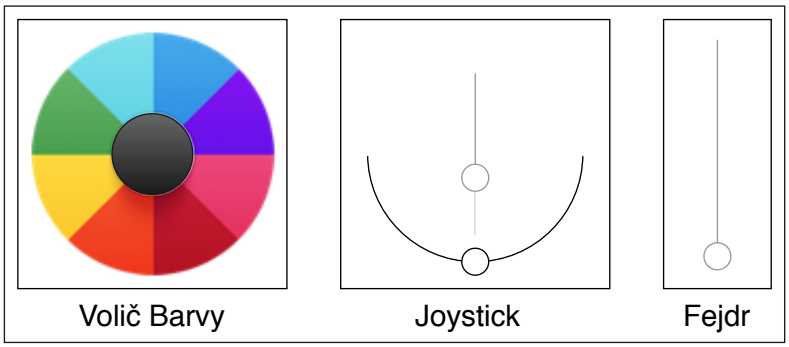
Ovládací prvek

Pro ovládání jednotlivých zařízení nebudou existovat ovládací prvky v jednom kuse, to by bylo příliš nepraktické. Například pro pohyblivé světlo bude lepší vygenerovat dva samostatné ovládací prvky, kdy jeden nastavuje barvu a druhý směr svícení. V případě, kdy osvětlovači nepotřebují měnit směr světla, ale jen barvu světla, bude stačit, aby si ovládací prvek směru odsunuli bokem a ten nadále nepřekážel.

Potřebné ovladače pro současný systém:

- Pro nastavování barvy bude potřeba mít *volič barev*, na kterém bude možné nastavit barvu.
- Pro nastavení polohy bude v systému *joystick* umožňující zvolit stupně otočení světla.
- Pro nastavení intenzity světla bude v systému fejdr. Ten bude umožňovat uživateli nastavit hodnotu z daném rozmezí.

Návrh vzhledu ovladačů je možné vidět na obrázku 1.5.

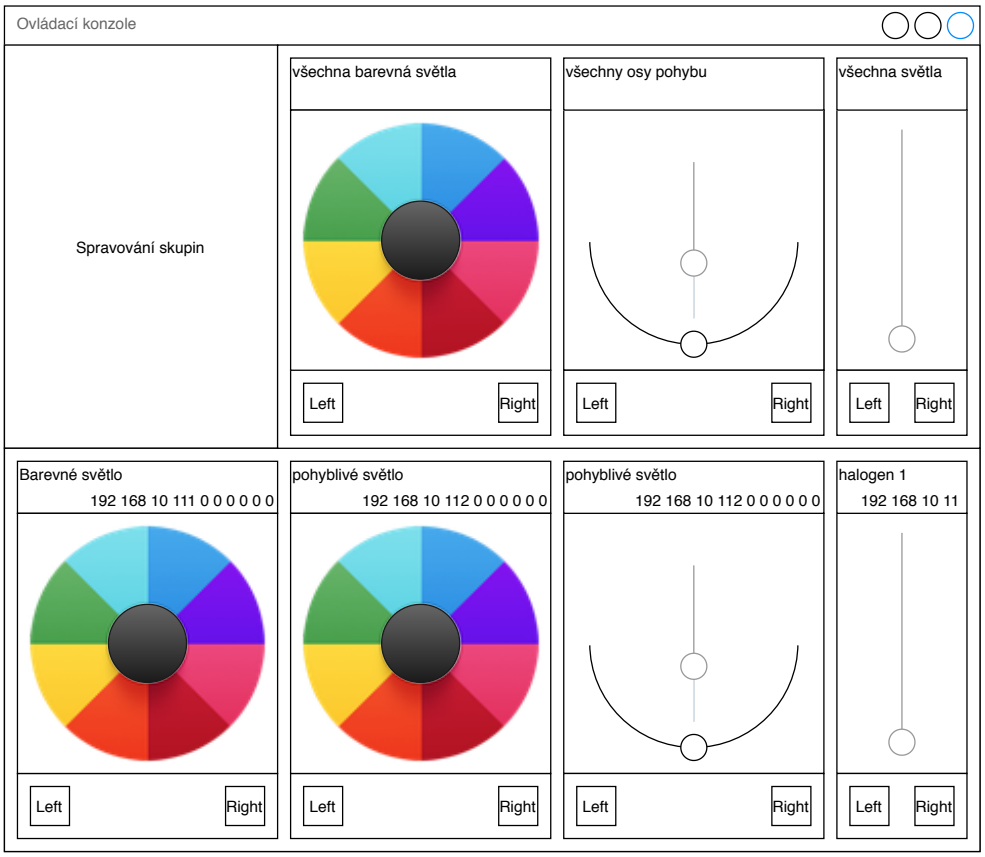


Obrázek 1.5: Náčrt ovládacích prvků.

S pomocí výše uvedených ovládacích prvků budeme reprezentovat jednotlivé druhy zařízení.

- K ovládání barevného světla bude plně dostačovat použít volič barev.
- K ovládání pohyblivého světla budeme pro výběr barvy používat volič barev a pro směr svícení joystick.
- Složky stmívače nejsou nijak propojeny, proto pro každou použijeme fejdr. Druhům zařízení, pro které není nijak jinak uvedeno, může být vždy vygenerován pro každou složku fejdr.

Spojením navržených prvků do sebe nám vznikne ucelená ovládací konzole, jak můžeme vidět na obrázku 1.6



Obrázek 1.6: Náčrt vzniklé konzole.

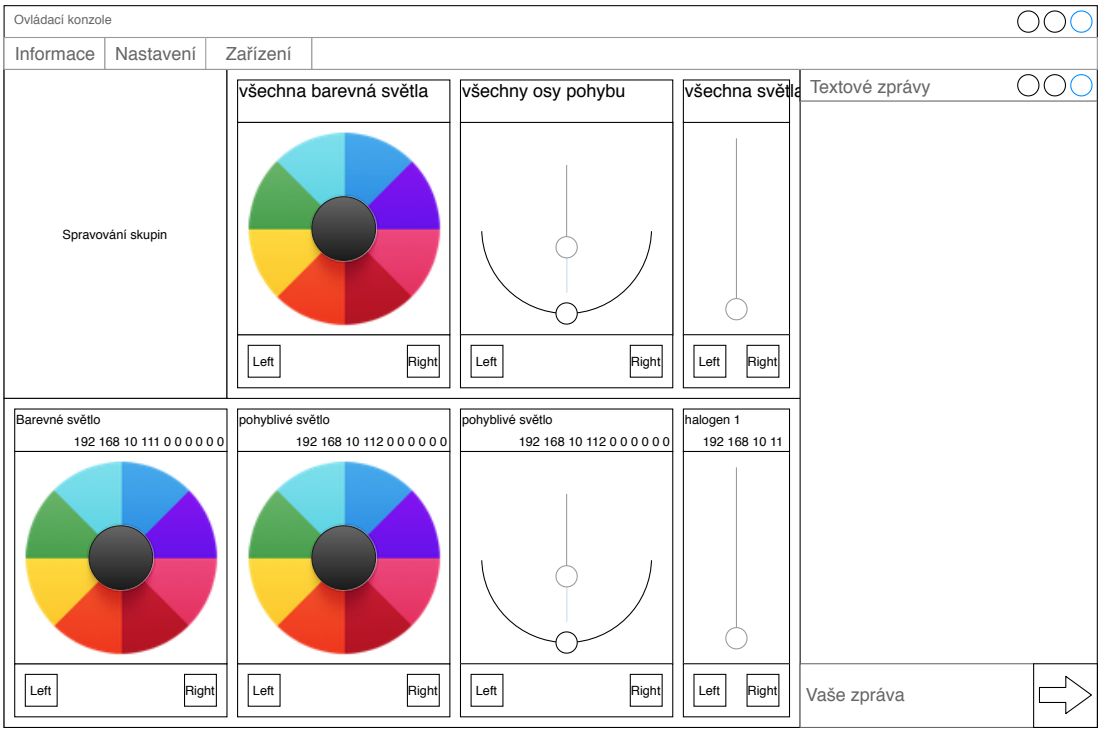
1.4.2 Počítačová verze rozhraní pro uživatele

Pro uživatele bude existovat počítačová aplikace, která bude představovat uživatelskou konzoli. Konzole bude uživateli poskytovat jednoduché rozhraní, kde bude:

- Pomocí virtuální ovládací konzole nastavovat data zařízením.
- Číst si zprávy textové komunikace a psát je.
- Načíst a spravovat data, jako jsou například jména přiřazená jednotlivým zařízením.

Pro tyto akce budeme chtít, aby okno bylo rozděleno na více nezávislých podoken, se kterými budeme moci manipulovat.

Přibližný vzhled výsledné aplikace si můžeme prohlédnout na obrázku 1.7.



Obrázek 1.7: Náčrt uživatelské aplikace představující konzoli.

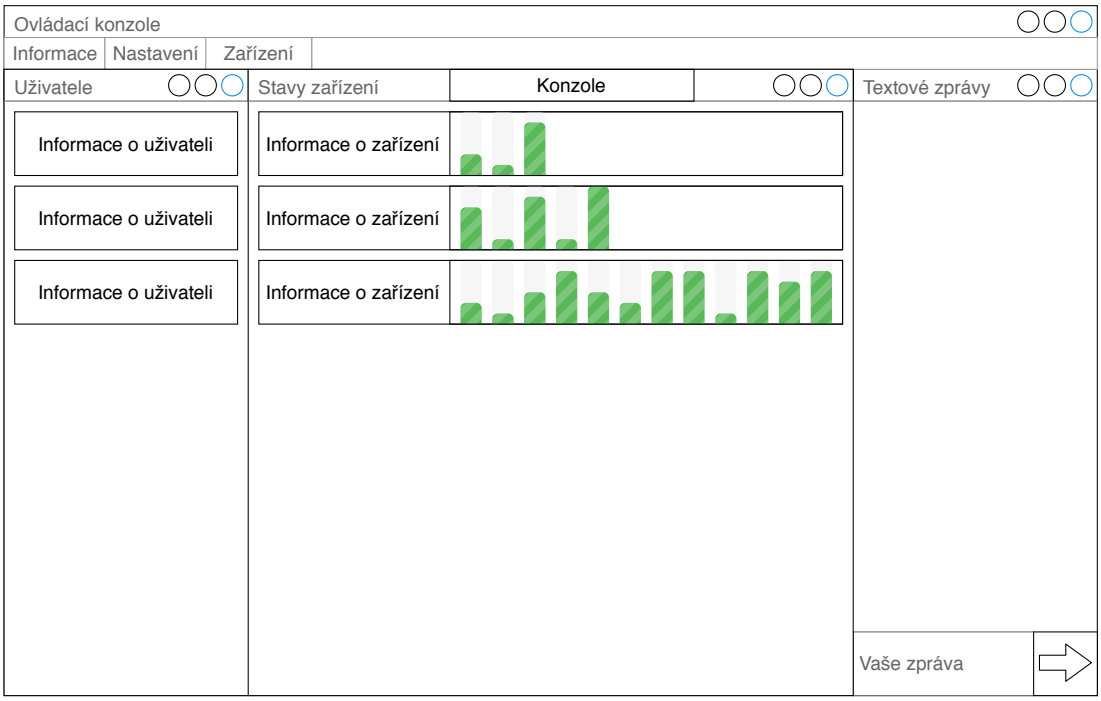
1.4.3 Počítačová verze rozhraní pro server

Pro hlavního osvětlovače bude existovat počítačová aplikace, která bude představovat server.

Z grafického rozhraní nad serverovou třídou bude možné:

- Sledovat data nastavená na zařízeních a v případě potřeby provádět nad nimi drobné změny.
- Poskytovat možnost vytvořit si virtuální ovládací konzoli a nastavovat data zařízením.
- Číst si zprávy textové komunikace a psát je.
- Zobrazit si přihlášené uživatele a případně je odhlásit.
- Načíst a spravovat data jako například protokoly pro jednotlivá zařízení.

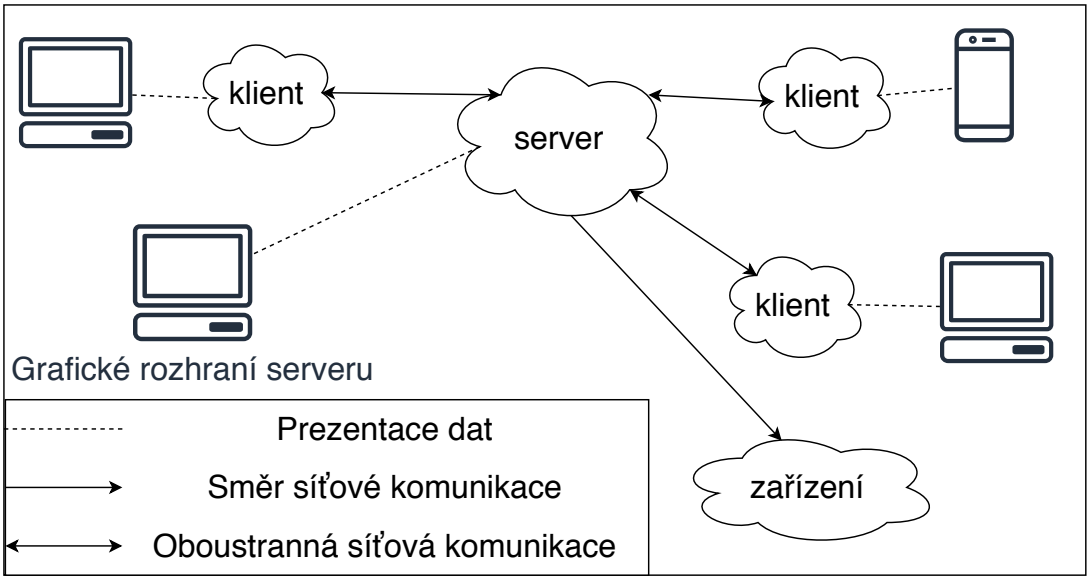
Pro tyto akce budeme chtít, aby okno bylo rozděleno na více nezávislých podoken, se kterými budeme moci manipulovat. Přibližný vzhled výsledné aplikace si můžeme prohlédnout na obrázku 1.8.



Obrázek 1.8: Náčrt uživatelské aplikace představující server.

2. Analýza Projektu

V této části se pokusíme analyzovat možná řešení, tak abychom splnili požadavky na systém z kapitoly 1. Při vyvíjení systému se nejdříve zaměříme na to, jak by měl být rozdělený a za co budou jednotlivé části zodpovědné. V představě systému máme určeno, že bychom chtěli mít server ovládající zařízení, na který budou připojeni klienti. Pro server i klienty budeme chtít grafické rozhraní, pomocí kterého je budeme ovládat, jak můžeme vidět na obrázku 2.1.



Obrázek 2.1: Představa systému.

Pokud se zamyslíme nad realizací grafické prezentace, dojdeme k závěru, že rozhodně nebude pro všechny druhy zařízení stejná. Například grafické rozhraní pro klienta bude pro mobilní telefon úplně jiné než pro počítač, ne kvůli technické stránce, ale protože budeme-li chtít co nejpříjemnější ovládání, nemůžeme vytvořit jednotnou verzi. Hlavním důvodem je velikost obrazovky zařízení. Například pro počítačovou aplikaci bude možné na displej zobrazit více ovladačů a k nim potřebné informace a při tom zachovat pro uživatele příjemné ovládání, ale v mobilní verzi s rostoucím počtem zobrazených ovladačů klesne jejich ovladatelnost. Dalším důvodem je, že mobilní aplikace má naprosto jiný účel než počítačová. Pomocí počítačové aplikace bude možné řídit osvětlení po dobu celého představení, ale mobilní aplikace by byla příliš nevhodná. Projekt si proto rozdělíme, a to na Backend, kde vznikne pomyslný server a klient, a na prezentační část, kde budeme data ze serveru a klienta prezentovat uživateli. Představu rozdělení si můžeme prohlédnout na obrázku 2.2.

Backend

Nyní je potřeba zvážit, jestli, případně jak, rozdělit backend. Na základě předchozích informací si můžeme navrhnout naše požadavky na serverový a klientský modul, které nám budou plnit funkci serveru a klienta.

Serverový modul

- Serverový modul si bude udržovat data pro všechna zařízení a zajišťovat, aby byla validní.
- Serverový modul bude zasílat data zařízením.
- Serverový modul bude komunikovat s klientským modulem, přijímat od něj data pro zařízení i ostatní data potřebná ke správnému chodu systému.
- Serverový modul bude podporovat textovou komunikaci, proto si serverový modul bude muset udržovat textové zprávy.
- Serverový modul si bude udržovat seznam přihlášených klientů a ke každému klientovi uživatelské jméno. *Uživatelské jméno* je označení, které si uživatel nad klientským modulem zvolí při přihlašování k serverovému modulu. Důvod, aby k identifikátorům mohlo být přiřazeno uživatelské jméno je ten, aby až budou data ze serverového modulu prezentována uživateli, byla pro uživatele zjednodušená orientace v připojených klientech.
- Nad serverovým modulem bude možné vytvořit uživatelské grafické rozhraní.

Klientský modul

- Klientský modul si bude držet data pro všechna zařízení. Pokud dojde ke změně dat, odešle je serverovému modulu.
- Klientský modul bude přijímat a zpracovávat zprávy od serverového modulu.
- Klientský modul si bude držet textovou komunikaci.
- Nad klientským modulem bude možné vytvořit uživatelské grafické rozhraní.

Představa modulových závislostí

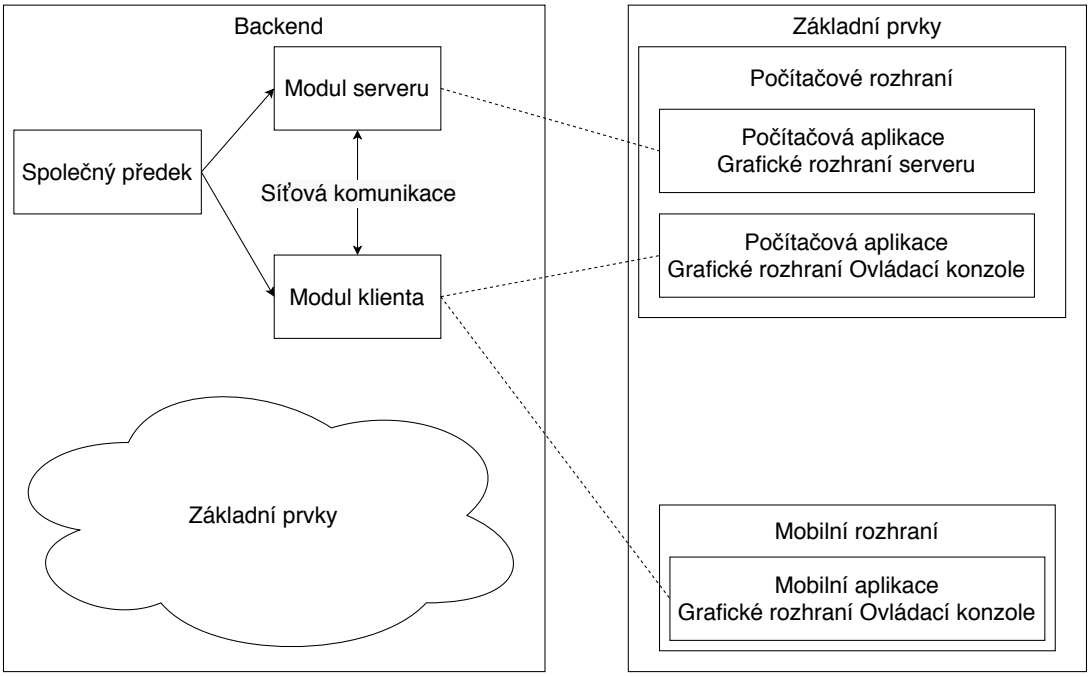
Jak je vidět z výčtů požadavků na serverový a klientský modul, jsou si celkem podobné. Oba moduly budou podporovat komunikaci mezi sebou, udržovat si validní informace pro zařízení, udržovat si textovou komunikaci a umožňovat nad sebou vybudovat grafickou reprezentaci pro uživatele, proto by bylo vhodné, aby oba moduly měly společného předka a zůstaly spolu.

Základní prvky

Klientský i serverový modul budou potřebovat nějaké základní prvky, nad kterými budou pracovat. Jedná se například o formát identifikace jednotlivých zařízení, nebo jak budou reprezentovány stavy jednotlivých zařízení. Proto je důležité, abychom do backendu přidali tyto základní komponenty a nadefinovali jejich vlastnosti a informace, které budou obsahovat.

Prezentační část

Backend nám poskytne knihovnu, ze které získáme klientský či serverový modul. Našim úkolem v prezentační části bude nad těmito moduly vybudovat aplikaci s grafickým rozhraním, kterou uživatel spustí, nastaví data pro spojení a nadále ji jen používá k ovládání světel a s tím spojené činnosti. Jelikož není možné vytvořit aplikaci, která by byla přenositelná mezi počítači, tablety a telefony, kvůli naprosto různým formám ovládání, bude potřeba si pro každé zařízení vyrobit vlastní projekt. Serverový i klientský modul jsou si svým chováním a vlastnostmi dostatečně podobné, takže bude docela dobře možné, vyrobit si jeden projekt, ve kterém bude vytvořena aplikace představující klienta i aplikace představující server.



Obrázek 2.2: Teoretický návrh členění systému.

2.1 Způsob síťové komunikace

Při budování představy výsledného systému jsme si systém rozdělili na dvě části a pro každou část jsou jiné požadavky, které musí splňovat.

Síťová komunikace serverového modulu se zařízeními Síťová komunikace serverového modulu se zařízeními je dána vlastnostmi zařízení a formátem dat, která jsou schopna přijímat. To znamená, že v této části systému musí komunikace probíhat přes UDP a data jsou přenášena ve formátu, který vyžadují zařízení, jak jsme se dozvěděli v části Formát zpráv zařízení divadla Lípa.

Síťová komunikace serverového modulu s klientským modulem Síťová komunikace mezi serverovým modulem a klientským modulem není pevně určena, proto se musíme na základě našich požadavků rozhodnout jak bude vypadat, co se bude přenášet a pomocí jakých technologií.

2.1.1 TCP versus UDP

Hlavní funkcí komunikace mezi serverem a konzolemi bude přenášení dat o stavu ovládaných zařízeních, proto tomu musíme podřídit způsob spojení.

TCP

Mezi klientem a serverem bychom mohli navázat spojovanou komunikaci pomocí TCP. Díky spojované komunikaci by se zprávy doručovaly postupně a v pořadí, v jakém byly odeslány, ale za cenu toho, že zprávy by byly doručovány se zpožděním, vzniklým opakovaným zasíláním nedoručených zpráv [6]. To je v našem případě nežádoucí. Pro nás je vhodnější dostávat zprávy co nejdříve i za cenu toho, že některé přeskočíme.

V systému, jaký je v divadle teď, kdy je pro komunikaci nachystaná vlastní linka, ztráty zpráv příliš nehrozí, ale v budoucnu by se mohlo stát, že se síť značně zkomplikuje, případně se v síti objeví nová zařízení, která by tyto ztráty mohla zapříčinit, například by se mohlo jednat o nové světlo aktivně se snažící komunikovat nebo, jak se v současnosti děje, by do naší sítě mohly být připojeny sítové kamery, sloužící hercům jako pomůcka, aby věděli, kdy mají jít na scénu.

Představme si situaci, kdy se osvětlovač bude snažit nastavit zhasnutá světla na maximální výkon. Při případných ztrátách by sice nastavování probíhalo plynule mezi serverovým a klientským modulem, ale tato plynulost by byla opět ztracena mezi serverovým modulem a zařízeními, kde je UDP komunikace se zařízeními nutná. Navíc by docházelo k tomu, že data na serveru by byla opožděna oproti stavu na klientském modulu a kazilo by to dojem z nastavování v reálném čase.

UDP

Lepším způsobem pro přenášení dat mezi konzolemi a serverem bude použít nespojované spojení. To znamená, že u zpráv není zajištěné, že jsou doručeny a ani jejich pořadí [7].

To, že se některé zprávy ztratí, nám příliš nevadí, protože ze serverového modulu můžeme data neustále posílat jak zařízením tak i klientům, tím případné ztracené zprávy překrýt.

Problém bude s pořadím zpráv, které je v systému důležité. Mohlo by se například stát, že při přenosu se jedna zpráva ztratí a dojde někdy později, kdy už není aktuální. Tento problém bude nutné řešit alespoň u zpráv, u kterých záleží na pořadí. K vyřešení tohoto problému použijeme *časové razítko*. Časové razítko bude představovat informaci, která nám u zpráv dokáže určit pořadí, v jakém přišly. Na základě těchto důvodů a úvah se data pro zařízení budou mezi uživatelským a serverovým modulem přenášet pomocí UDP.

UDP společně s TCP

To, že pro přenos hlavních zpráv používáme UDP, ale neznamená, že tak musíme přenášet i ostatní zprávy, jako jsou například zprávy textové komunikace nebo zprávy pro přihlášení a odhlášení ze serveru. U těchto zpráv není potřeba, aby dorazily okamžitě, ale spíše aby dorazily. Pro tyto zprávy bychom si mohli

vytvořit nezávislý spojovaný kanál. Výhodou by bylo, že bychom nemuseli řešit jestli zprávy přišly a případně je posílat vícekrát. Na druhou stranu bychom tím ztratili možnost zařadit do systému plánovanou hardwarovou konzoli, z obrázku 10, komunikující jen přes UDP. Proto veškerou komunikaci povedeme jen přes UDP.

2.1.2 Přenášení změny dat

V systému bude potřeba přenášet data mezi serverovým a klientským modulem, určená pro ovládání zařízení. V tuto chvíli nastává otázka, jak bychom měli změny dat přenášet. Jestli budeme chtít posílat řídící příkazy, rozdíly mezi složkami nebo rovnou konkrétní hodnoty.

Řídící příkaz

Jedním z možných způsobů komunikace by bylo posílat řídící příkazy. Představa je taková, že při změně dat na klientském modulu bychom poslali serverovému modulu zprávu s identifikací zařízení, pro které je změna určena a popis prováděné změny. Například popis prováděné změny by mohl vypadat následovně: „Snížení hodnoty z maxima na minimum za 5 vteřin.“ Tento princip by byl velice úsporný na počet přenášených zpráv,ale nepodporoval by změnu dat v reálném čase, protože by musel čekat až osvětlovač akci dokončí, nebo by příslušné operace musely být předem nadefinovány, což by bylo velice pracné a nepraktické. Varianta, že by klient odesílal řídící příkazy při každé změně dat, je sice možná, ale ztratila by se tím hlavní výhoda odesílání malého množství zpráv.

Zaslání rozdílu

Dalším způsobem by bylo posílat hodnoty určující rozdíl mezi současným stavem a stavem, kterého chceme docílit. Nevýhoda tohoto způsobu spočívá v tom, že je příliš náchylný na nekonzistenci dat, rozdílová hodnota musí znaménková a tedy alespoň o bit větší a nepřináší na oplátku žádnou výraznou výhodu oproti přímému posílání aktuálních hodnot.

Posílat aktuální hodnoty

Asi nejlepším řešením bude rovnou posílat hodnoty, které si přejeme nastavit. Tím si jednak serverový modul s klientským budou neustále aktualizovat data, ale hlavně se vyhneme riziku, že dojde ke špatné interpretaci příkazu.

2.2 Základní prvky

V systému bude potřeba mít spoustu základních prvků, které budou plnit své role při síťové komunikaci i při komunikaci v rámci backendu a prezentační vrstvy. Je proto důležité nadefinovat si, jaké budou mít funkce v systému a jaké informace budou obsahovat.

2.2.1 Časové razítko

Časové razítko má být informace, která nám dokáže ze dvou razítek určit, které je novější. Jak jsme uvedli výše, hlavním smyslem časového razítka bude zabránit, aby zpráva odeslaná od modulu, která se někdy zdrží, nebyla interpretována jako novější. To znamená, že nám je ve skutečnosti jedno, jak je razítko reprezentováno, jestli se jedná o čas nebo pouze číselnou hodnotu. Hlavním požadavkem je, aby šlo pomocí dvou razítek od stejného modulu určit, která z těchto razítek je novější a razítka od různých modulů můžeme prozatím považovat za neporovnatelné.

Čas nebo hodnota

Pro razítka máme teď dvě základní možnosti jak je reprezentovat. První možností by bylo, zprávy si pouze číslovat a tedy jako razítko použít číselnou hodnotu představující pořadí, v jakém byla zpráva odeslána. Nevýhodou této varianty je, že by bylo potřeba udržovat si číslování zpráv a kdyby v budoucnu byla potřeba porovnávat pořadí těchto zpráv nejen v rámci zpráv jednoho modulu, ale i v rámci všech obdržených zpráv, bylo by nemožné razítka k tomu jednoduše rozšířit. Druhou možností je jako časové razítko použít čas ze zařízení. Tato varianta nezávisí na počtu odeslaných zpráv jako varianta s číselnou hodnotou a navíc, v případě potřeby je ji možné i rozšířit. Při případném rozšiřování by pak bylo zapotřebí zajistit, aby všechna zařízení účastnící se komunikace měla nastavená jednotný čas. Proto použijeme variantu kde hodnota časové razítko představuje skutečný čas.

Obsah razítka

Nyní je potřeba rozhodnout, jakou přesnost razítka budeme požadovat. Pokud bychom použili přesnost pouze na hodiny, minuty a sekundy, budeme mít problém rozhodnout, jak zacházet se zprávami odeslanými ve stejné sekundě, proto budeme vyžadovat zjemnění, alespoň na milisekundy. V případě, že razítka jsou stejná až na milisekundy, budeme je brát, jako že vznikla ve stejný okamžik. Na opačné straně musíme vyřešit problém s tím, jak chceme postupovat, když komunikace bude trvat déle než hodinu, případně když komunikace překročí nejvyšší limit. Z toho důvodu budeme požadovat, aby časové razítko obsahovalo i dny, měsíce a roky.

2.2.2 Identifikace

Při představování systému jsme se rozhodli, že identifikaci jednotlivých zařízení bude potřeba zjemnit. Pro základní jednoznačnou identifikaci všech zařízení, i s podporou analogových zařízení, která budou do systému připojena přes analogový převodník, je pro potřeba IP adresa zařízení případně IP adresa převodníku a číslo zařízení v podsystému. To je dohromady 6 bytů, ale není zde žádná rezerva pro případná rozšíření. Například kdyby zařízení vyžadovala komunikaci na různých portech, vyžadovalo by to alespoň další 2 byty. Nyní nastává otázka, jak velkou rezervu budeme požadovat. Jednou z možností by bylo, připravit se na přechod na IPv6 adresy a očekávat, že IP adresy budou dlouhé 16 bytů. To

by znamenalo, že identifikace bude dlouhá 16 bytů pro IPv6, 2 byty pro analogovou adresu a 2 byty pro případné rozšíření. To by bylo ale příliš přehnané. Za cenu toho, že místo opravdu potřebných 6 bytů bychom používali 20 bytů, získali bychom připravenost na IPv6. Při případné potřebě přechodu na IPv6 adresy, což je velice nepravděpodobné, protože zařízení nepodporují IPv6 a komunikace probíhá v lokální síti, by bylo výhodnější si identifikátor znovu zanalyzovat a případně upravit. Výhodnější bude připravit si jen malou rezervu tak, abychom pokryli očekávané potřeby a měli rezervu menší než je aktuálně potřebný počet bytů. Naše varianta, kterou použijeme, bude vypadat tak, že si zarovnáme naši identifikaci na 10 bytů, kdy 4 byty zůstanou volné pro pozdější případné rozšíření. Identifikace tak bude ve tvaru 4 byty IP adresa, 4 volné byty a 2 byty analogové adresy, jak je možné vidět v tabulce 2.1.

Tabulka 2.1: Návrh identifikace vzniklý z analýzy.

	pořadí	**délka**
IP adresa	0-3	4
volné byty	4-7	4
analogová adresa	8-9	2

2.2.3 Druh zařízení

V systému můžeme očekávat, že se v budoucnu objeví zařízení, která jsou si svými vlastnostmi velmi podobná, ale liší se například formátem dat, kterým se s nimi komunikuje.

Například v současném systému jsou dvě různá barevná světla. Nová barevná světla používají formát představený v úvodní části Formát zpráv zařízení divadla Lípa a pro každou barevnou složku umí nastavit 256 hodnot. Starší barevné světlo používá pro komunikaci také již dříve představený formát, ale bez řídícího znaku a pro červenou a zelenou barevnou složku má 256 hodnot, ale pro modrou barevnou složku má jen 128 hodnot.

Nyní nastává otázka, jak s těmito zařízeními v systému pracovat. Jedním z řešení by bylo říct, že se jedná o různá zařízení a nadále se tak k nim i chovat. Pro každé zařízení mít v prezentační části vlastní ovladač, pro každé zařízení vytvořit vlastní komponenty pro uchovávání hodnot a tak dále. To by vedlo k velkému množství kódu. Proto tento způsob nepoužijeme.

Druhým možným přístupem je všechna zařízení si rozdělit podle vlastností na určité druhy. Tím bychom získali to, že by nám stačilo v systému rozlišovat jen pár druhů zařízení a teprve až ve chvíli, kdy serverový modul bude posílat data konkrétním zařízením, by musel určit o jaký typ zařízení se jedná a podle toho zvolit formát komunikace s tímto zařízením.

Například by v budoucnu mohla nastat situace, kdy budeme mít druh barevných zařízení, který pokrývá současná barevná světla. Druh barevných zařízení bude očekávat, že složky všech zařízení tohoto druhu budou představovat červenou, zelenou a modrou barvu. V systému by se ale objeví světlo, které bude požadovat ve svých složkách HSV hodnoty místo hodnot RGB. Jelikož by se jednalo o barevné světlo, bylo by zařazeno do druhu barevných světel. Celou dobu by se s ním jednalo jako se světlem, které je ovládané pomocí složek RGB a až při

odesílání dat konkrétnímu zařízení by serverový modul podle typu zařízení rozhodl jaký formát dat očekává. V tomto případě by to znamenalo, že by z barvy určené RGB hodnotami serverový modul získal HSV hodnoty a v příslušném formátu je zařízení odeslal.

Z těchto důvodů budeme používat druhý navrhovaný systém.

Současná světla můžeme rozdělit na tři druhy:

- Barevná světla, která mají 3 složky, kde složky reprezentují barvu v RGB.
- Pohyblivá světla, která mají 5 složek, kde tři složky reprezentují barvu v RGB a dvě složky směr svícení.
- Stmívač, který má 12 složek, kde složky nejsou nijak spojeny.

2.2.4 Stav zařízení

Abychom mohli zařízení ovládat, bude důležité určit si, jaká data k tomu potřebujeme a spojit si je do celku, který budeme nazývat *stav zařízení*. Tento stav zařízení budeme potřebovat nejen pro udržování dat v serverových a klientských modulech, ale také pro přenášení dat v komunikaci mezi serverovým a klientským modulem. Ke zjištění dat, která potřebujeme, si zkusíme projít očekávaný průběh nastavovaní dat a jejich odesílání zařízením.

Mějme barevné světlo, které chceme řídit pomocí našeho systému. První co bychom chtěli ze stavu zjistit, je to, k jakému zařízení patří, proto by měl obsahovat identifikátor zařízení, pro které obsahuje data. Abychom měli vůbec nějaká data k odeslání pro světlo, budeme si ve stavu zařízení držet hodnoty pro jednotlivé složky tohoto zařízení. To je plně dostačující k tomu, abychom mohli zařízení ovládat na straně komunikace serverového modulu se zařízeními.

Na straně komunikace serverového modulu s klientským modulem to ovšem stačit nebude. Představme si situaci, kdy dva klientské moduly odešlou současně stav zařízení, který chtějí nastavit. V tuto chvíli musí serverový modul řešit konflikt dat. Při představě systému jsme určili, že každý uživatel bude mít pro každé zařízení určenu důležitost, s jakou mu hodnoty nastavuje. Proto by si měl každý stav zařízení udržovat hodnotu důležitosti, s jakou byl nastaven a v případě jako je tento, serverový modul bude moc použít tuto hodnotu k tomu, aby určila jaká data použít.

V systému ale může dojít i ke konfliktu na úrovni složek. Představme si situaci, že dva klientské moduly odešlou současně stav zařízení se stejnou důležitostí, který chtějí nastavit, ale každá chce nastavit jednu složku a o ostatní složky se vůbec nezajímá. Tato situace nastane zejména u zařízení, jako je stmívač, protože ten ovládá spoustu světel, které nemusí mít nic společného. Při současném obsahu hodnot, stav zařízení neposkytuje serverovému modulu žádné informace, jaké data použít. Jedním z možných řešení by bylo neposílat vždy všechny složky, ale jen ty, které chceme nastavit. To by znamenalo přidat do stavu informaci, o které složky přesně se jedná, navíc bychom tak už částečně odebírali serverovému modulu informace, které by mohl využít při validaci dat. Lepším způsobem řešení tohoto problému bude držet si pro každou složku pravdivostní hodnotu o tom, jestli byla nastavena záměrně, či nikoliv.

2.3 Přenášené informace a jejich zpracování

Při komunikaci mezi serverovým a klientským modulem bude potřeba přenášet více druhů informací než jen pouze informace o zařízeních. Pro každý druh informace si tedy vyrobíme zprávu, která nám ho přenese.

Skoro každá zpráva, kterou budeme chtít přenášet, by měla obsahovat alespoň základní informace jako jsou identifikace odesílatele zprávy, identifikace příjemce zprávy a typ zprávy samotné. Těmto informacím budeme říkat *hlavička zprávy*. Důvodem, proč potřebujeme, aby hlavička nesla identifikaci odesílatel a příjemce je ten, kdyby se někdy v budoucnu využívalo hardwarových konzolí, které by byly připojeny do systému přes analogový převodník, obdobně jako tomu bylo s analogovými zařízeními, jak jsme mohli vidět na obrázku 1.2. Nejlepším způsobem jak zjistit, jaké informace budeme chtít přenášet a jaké zprávy na to budou potřeba, bude popsat si očekávaný průběh komunikace mezi serverovým a klientským modulem.

Představení se serveru

Pro spojení klientského modulu se serverovým bude potřeba, aby klientský modul znal identifikaci serverového modulu a port, na kterém budou komunikovat. Můžeme předpokládat, že komunikace klientského modulu a serverového modulu bude vždy probíhat na stejném portu, případně při vytváření klientského modulu bude port pro komunikaci znám. Ovšem nemusí platit to, že serverový modul bude vždy mít stejnou identifikaci, protože jak jsme již dříve řešili, server bude moci být spuštěn na jakémkoliv počítači. Z toho důvodu budeme chtít, aby v systému byla možnost zeptat se na identifikaci serverového modulu.

Asi nejjednodušším a nejpraktičtějším způsobem, jak tento dotaz realizovat, bude tak, že klientský modul odešle dotaz na identifikaci serverového modulu všem do sítě a pouze serverové moduly odpoví.

Představující se zprávy Pro zjištění identifikace serverového modulu bude muset klientský modul dostat zprávu nazpátek, proto bude muset v dotazu uvést svou identifikaci, aby bylo jasné, komu má serverový modul odpověď odeslat. K představení tedy bude existovat *klientská představující zpráva* obsahující identifikaci odesílatele této zprávy. Serverový modul bude muset jako odpověď odeslat svou identifikaci, k tomu bude *serverová představující zpráva* obsahující identifikaci serverového modulu.

Reakce modulů na představující se zprávy Nyní si musíme promyslet, jak by měly být představující se zprávy posílány a jak na ně jednotlivé moduly budou reagovat. Odesílatel klientské představující se zprávy se chce dozvědět o serverových modulech v síti, proto tuto zprávu odešle všem zařízením v síti a bude očekávat, že mu na ni serverový modul odpoví. Jelikož zpráva byla odeslána všem, obdrží ji i ostatní klientské moduly. Pokud klientský modul obdrží klientskou představující se zprávu, měla by ji ignorovat. Pokud serverový modul obdrží klientskou představující zprávu, odešle nazpět serverovou představující zprávu.

Serverová představující zpráva je odeslána na identifikace z klientské představující zprávy a proto jediný, kdo přijme serverovou představující zprávu, je

klientský modul, který o ni požádal, proto si klientské moduly získají z příchozích serverových představujících zpráv identifikaci serveru a uloží si ji.

Přihlášení na server

Ve chvíli, kdy klientský modul zná identifikaci serverového modulu a port, na kterém komunikuje s klienty, bude se smět pokusit o přihlášení.

Aby serverový modul věděl, kdo se k němu přihlašuje, je potřeba, aby mu klientský modul sdělil svou identifikaci, a jak již bylo řečeno dříve, klienti by měli potvrdit své právo na přihlášení pomocí textového hesla a uvést uživatelské jméno, pod kterým pak budou vedeni. Pro přenos těchto informací zavedeme *uživatelskou přihlašovací zprávu*, která bude obsahovat identifikaci odesílatele, jméno uživatele a heslo.

Přihlášení klientského modulu na serverový by mělo probíhat tak, že klientský modul odešle serverovému modulu uživatelskou přihlašovací zprávu. Serverový modul ověří platnost hesla a v případě, že odesílatel zprávy uvedl správné heslo, zařadí si obdrženou identifikaci na seznam přihlášených klientů a přiřadí si k němu uživatelské jméno. V případě, že heslo nebylo v pořádku nebo zpráva nebyla doručena, klientský modul, který odeslal přihlašovací zprávu, by měl dostat najevo, jestli se přihlášení podařilo, či nikoliv. Proto v případě, že přihlášení proběhlo, odešle serverový modul klientskému modulu potvrzení přihlášení o přihlášení. Proto v systému bude *serverová potvrzovací zpráva o přihlášení*.

Sdílení dat zařízeních

Pro udržování si aktuálních hodnot jednotlivých zařízení bude potřeba, aby si serverový a klientský modul mohli vzájemně posílat zprávy k tomu určené. V předchozích částech jsme se na základě analýzy rozhodli, že budeme posílat aktuální hodnoty a navrhli jsme si proto pro udržování dat jednoho zařízení Stav zařízení.

Nyní se musíme rozhodnout, jaké informace budeme potřebovat přenášet při sdílení dat pro zařízení. Asi je jasné, že budeme chtít přenášet stavy jednotlivých zařízení a identifikaci odesílatele této zprávy. Důležité je si uvědomit, že tohle je jedna ze zpráv, kde je velmi důležité hlídat si pořadí zpráv, aby se nestalo to, že ztracené zprávy dorazí později s již neaktuálními daty. Proto bude důležité, aby tato zpráva obsahovala časové razítko. Pro tuto výměnu dat si tedy navrhneme *scénovou zprávu*, která bude obsahovat identifikaci pro příjemce, stavy zařízení, které chceme sdílet a časové razítko určující čas odeslání.

Zpracování scénové zprávy U scénové zprávy bude potřeba rozmyslet si, jak má být jednotlivými moduly zpracována.

Scénová zpráva pro klientský modul V případě, že klientský modul obdrží scénovou zprávu od serverového modulu, musíme vědět, jak ji bude zpracovávat. Vzhledem k tomu, že systém je budován centralizovaně a serverový modul je odpovědný za udržování validních dat, můžeme příchozí data považovat za validní. Otázkou ale zůstává, jestli jsou právě příchozí data nejnovější. Z toho důvodu by si klientský modul měl pamatovat časovou známku poslední přijaté zprávy a ověřit, že nová přijatá data mají novější časové razítko než poslední přijatá data.

V případě, že přijatá zpráva nemá novější časové razítko, může být zahozena, protože je pro klientský modul již zastaralá. V případě, že právě přijatá zpráva má nejnovější časové razítko, budou její data nastavena jako aktuální. Nyní je důležité uvědomit si, co budeme očekávat od nastavení dat u klientského modulu. Stavy pro zařízení, které klientský modul nemá, může přijmout tak, jak jsou. To znamená, že klientský modul si vloží na seznam všech stavů zařízení tento nový stav zařízení. Stavy, které již klientský modul má, by měly být aktualizovány. Je důležité rozhodnout, jak by měly být stavy aktualizovány. Pokud bychom nahradili celé staré stavy pro zařízení novými, ztratili bychom informaci, jakou hodnotu důležitosti měly jednotlivé stavy pro tohoto klienta. Proto klientovým stavům ponecháme původní hodnotu důležitosti. U každé složky máme příznak, jestli ji chce klient nastavit či ne. Jelikož právě nastavujeme data ze serveru, můžeme označit všechny složky, kterým nastavujeme nové hodnoty, jako to, že nejsou nastavené záměrně.

Scénová zpráva pro serverový modul Návrh zpracování scénové zprávy serverovým modulem si můžeme prohlédnout na algoritmu 1. Pokud není klientský modul, který data odeslal na seznamu přihlášených klientských modulů, zpráva není zpracována, protože data mohou měnit pouze přihlášení uživatelé. Ověříme, jestli tento klientský modul již poslal scénovou zprávu a serverový modul má již od toho klientského modulu časové razítko. Pokud serverový modul ještě neobdržel scénovou zprávu s časovým razítkem od tohoto klientského modulu, uloží si jeho časové razítko a zpracuje tuto zprávu. Pokud serverový modul již obdržel scénovou zprávu s časovým razítkem od tohoto klientského modulu a příchozí zpráva obsahuje starší časové razítko, tato zpráva není zpracována. Pokud časové razítko přijaté zprávy je novější než u poslední přijaté zprávy od tohoto klientského modulu, serverový modul si uloží jeho časové razítko a zpracuje tuto zprávu.

Zpracování zprávy: Serverový modul projde přijatou scénovou zprávu a pro každý její stav zařízení rozhodne, jak s ním naložit. Pro přijatý stav zařízení je potřeba si promyslet, jaké je asi očekávané jeho vyhodnocení a jak by měly být dané hodnoty nastaveny. Pokud serverový modul doposud nemá stav pro zařízení, pro který je přijatý stav zařízení, nemá hodnoty k porovnání a přidá si jej celý bez změny. Pokud serverový modul již má stav pro toto zařízení, musí zkontrolovat, jestli příchozí stav má stejnou, případně vyšší důležitost. Pokud příchozí stav má menší důležitost, není potřeba se s ním nadále zabývat a není použit. Pokud má vyšší nebo stejnou důležitost jako současný stav, je důležité podívat se na jednotlivé složky.

Pokud není současná složka nastavena záměrně a příchozí je nastavena záměrně, znamená to, použít hodnotu příchozí složky a označit ji jako nastavenou záměrně.

Pokud není příchozí hodnota složky označena jako nastavena záměrně a současná hodnota složky je označena jako nastavena záměrně, ponecháme současnou hodnotu. V tomto okamžiku může dojít k tomu, že přijatý stav zařízení má větší důležitost než stav zařízení, který je na serverovém modulu, ale i přes to je ponechán původní. Důvodem je, že pokud někdo posílá stav zařízení, ve kterém je složka označena, že není nastavena záměrně, je to bráno, jako že odesílateli na hodnotě u této složky nezáleží, ale musí ji poslat kvůli tomu, jak je stav zařízení

nadefinován v předchozí části jménem Stav zařízení. Stav zařízení musí obsahovat všechny složky pro případ, že tento stav je pro modul, který jej přijímá, prvním pro toto zařízení, tak aby věděl, jaké hodnoty mají mít jednotlivé složky.

Pokud jsou obě složky ve stejném stavu, ať už jsou obě nastavené záměrně nebo ne, je použita složka z příchozího stavu.

Po zpracování přijaté zpráv se změnila data, proto bude nutné poslat všem klientům aktuální data.

Algoritmus 1: zpracování scénových zpráv serverovým modulem.

```
Serverový modul ještě neobdržel scénovou zprávu od toho klientského
modulu? Uložit tomuto klientovi časové razítko přijaté zprávy. Zpracuj
zprávu. if Razítko z přijaté zprávy je novější než rázítko uložené u
tohoto klientského modulu then
| Uložit tomuto klientovi časové razítko přijaté zprávy. Zpracuj zprávu.
else
| Tato zpráva se nebude zpracovávat. return.
end
foreach Stavy z přijaté Zprávy do
| if Data serveru obsahují stav pro stejné zařízení jako stav ze zprávy?
|  then
|  | if Důležitost stavu ze zprávy >= Důležitost stavu ze serveru ?
|  |  then
|  |  | foreach složky z obou stavů do
|  |  |  | if Je složka ze zprávy nastavena a složka ze současného
|  |  |  |  stavu ne? then
|  |  |  |  | Použij hodnotu složky ze zprávy a označ ji jako
|  |  |  |  |  nastavenou.
|  |  |  | else
|  |  |  |  | if Mají obě složky stejný stav nastavení ? then
|  |  |  |  |  | Použij složku z příchozího stavu.
|  |  |  |  | end
|  |  |  | end
|  |  | end
|  | end
| else
|  | Přidej stav ze zprávy do současných dat.
| end
end
Po zpracování příchozí zprávy server odešle všem přihlášeným
 uživatelům, kromě odesílatele právě zpracované scény, aktuální stav
 všech zařízení.
```

Textová komunikace uživatelů.

V systému jsme si při představě určili, že budeme chtít podporovat i textovou komunikaci mezi serverovými a klientskými moduly. K tomu budeme potřebovat zprávu přenášející text. Proto v systému zavedeme *textové zprávy*, které bude možné posílat si mezi serverovými a klientskými moduly a budou schopné nést textové informace.

Klientský i serverový modul si bude držet seznam zpráv. Nyní je důležité

promyslet si, jak budou textové zprávy šířeny a zpracovávány.

Jednou z možností by bylo, aby je modul při odesílání zpráv poslal všem v síti a každý modul si do svých textových zpráv uložil všechny přijaté textové zprávy. Nevýhodou by bylo, že by danou zprávu přijali i nepřihlášení uživatelé a nad šířením by nebyla žádná kontrola.

Druhou variantou by bylo zachovat koncept centralizovaného systému, kdy by klientský modul textové zprávy odesílal jen serverovému modulu, který by přijaté zprávy poslal přihlášeným klientům.

Nyní ale musíme řešit problém, jak si moduly budou zprávy ukládat do svých seznamů zpráv.

První variantou by bylo, že klientské moduly si při odesílání zpráv uloží odesílanou zprávu do seznamu zpráv a serverový modul si uloží přijatou zprávu do svého seznamu a odešle ji všem přihlášeným klientům, kromě odesílatele této zprávy. U této varianty bude docházet k tomu, že jednotlivé moduly budou mít naprosto různé pořadí obdržených zpráv. Například v situaci, kdy všechny moduly odešlou zprávu současně, bude každá z nich mít ve svém seznamu svou zprávu na prvním místě.

Druhá varianta by byla, že serverový modul si do seznamu přijatých zpráv uloží všechny zprávy, které obdrží od klientů a i ty, které jsou vytvořeny na ní. Navíc serverový modul každou vytvořenou i přijatou zprávu pošle všem přihlášeným klientským modulům. Klientský modul si své vytvořené zprávy neukládá, pouze je pošle serverovému modulu, klientský modul si ukládá pouze přijaté zprávy. Tím dochází k tomu, že aby si klientský modul uložil sebou odeslanou zprávu, musí tato zpráva doputovat na serverový modul a ten mu ji musí poslat na zpět.

Druhá varianta je jednak jednodušší na implementaci, ale hlavně zmírňuje problém s pořadím zpráv. Proto ji použijeme i my.

Odhlášení ze serveru

Aby se klient odhlásil ze serveru, bude stačit aby poslal serverovému modulu informaci, že se chce odhlásit, k tomu bude *uživatelská odhlašovací zpráva*, která bude obsahovat alespoň odesílatele. Aby serverový modul dal vědět klientovi, že byl odhlášen, pošle klientskému modulu *serverovou potvrzovací zprávu o odhlášení*, čímž dá klientovi vědět, že byl odhlášen a serverový modul odesílatele uživatelské odhlašovací zprávy vyřadí ze seznamu přihlášených klientů.

Informační komunikace

Pro případ, že by v systému došlo k chybě, neobvyklé situaci, či jen nějaké akci, která by stála za zaznamenání, budeme chtít, aby byla v systému podpora pro přenos zpráv tohoto druhu. Například pokusí-li se klientský modul, který není do systému přihlášen zaslat zprávu, sloužící k nastavení hodnot pro zařízení, nebo se klientský modul pokusí přihlásit, ale s neplatným heslem. Tyto záznamy by měly technickému týmu sloužit k případnému dohledávání průběhu komunikace a k tomu, proč neprobíhá podle představ.

Příklad využití informační komunikace: Uživatel se bude snažit připojit do systému, ale to se mu nebude dařit. Na serverové straně pak půjde ze záznamu určit, že je to například kvůli tomu, že zadal špatné heslo.

Pro tyto informační zprávy budeme tedy mít *informační zprávu*, která bude přenášet textová data.

Shrnutí základních zpráv

Pro základní komunikaci budou potřeba tyto zprávy:

- Scénová zpráva
- Textové zprávy
- Chybové zprávy
- Uživatelská přihlašovací zpráva
- Uživatelská odhlašovací zpráva – stačí hlavička
- Uživatelská představující zpráva – stačí hlavička
- Serverová potvrzovací zpráva o odhlášení – stačí hlavička
- Serverová potvrzovací zpráva o přihlášení – stačí hlavička
- Serverová představující zpráva – stačí hlavička

Z výčtu potřebných zpráv je vidět, že ve většině zpráv je potřeba přenášet jen identifikátory komunikujících stran a typ zprávy.

2.3.1 Způsob serializace pro přenos

Další důležitou částí komunikace po síti je serializace a deserializace přenášených dat. Nejjednodušším způsobem by bylo použít nějaký rozšířený framework, který by nám potřebné zprávy zpracoval do nějakého rozšířeného formátu, jako je například JSON. Autor zařízení si však přál, aby byl pro systém vytvořen vlastní formát. Hlavními podmínkami pro formát bylo, aby byl úsporný na paměť, lehce deserializovatelný a aby bylo jednoduše možné určit, o jaký typ zprávy se jedná a kdo je její příjemce odesílatel. Protože výsledný formát by pak v budoucnu mohly přijímat a snažit se zpracovat velice jednoduchá zařízení s omezenými paměťovými možnostmi. Našim cílem u tohoto formátu není rozšířitelnost, ale co největší jednoduchost, aby data byla co nejstručnější a v základu obsahovala jen to, co je potřeba.

Navíc, i přes to, že použijeme vlastní formát, v systému zůstane možnost použít pro serializaci a deserializaci složitějších dat jakýkoliv jiný formát. V systému může být zpráva, která bude odpovídat vytvořenému formátu, ale bude přenášet pole bytů, do kterého si bude moc potencionální programátor uložit jakékoliv data a tak je přenést mezi zařízeními.

Navržený formát

Z předchozího výčtu zpráv je jasné, že nejdůležitější a společnou částí pro všechny zprávy je její hlavička. Z ní jde poznat, od koho a pro koho daná zpráva je. Další informace pak jen záleží na daném typu zprávy. První otázkou je, jak serializovat jednotlivé části. Pro identifikaci je to jednoduché, u té stačí vycházet z jejího návrhu. Kolik bytů je ale potřeba pro serializaci typu? V současném návrhu pro základní komunikaci předpokládáme asi 10 zpráv. Proto by nám měl stačit 1 byte. Návrh serializace zpráv můžeme vidět v tabulce 2.2.

Tabulka 2.2: Návrh serializace hlavičky zprávy do bytů.

	pořadí	**délka**
IP adresa příjemce	0-9	10
IP adresa odesílatele	10-19	10
typ zprávy	20	1
data podle typu	21 - ...	...

Ve chvíli, kdy bychom potřebovali více než 256 zpráv, nastal by problém, jak typ zprávy rozšířit. V tomto případě je asi nepravděpodobné, že by bylo potřeba tolik zpráv, ale pro všechny případy si můžeme určit, že zpráva s typem 255 bude obsahovat data s rozšiřujícím typem.

2.4 Prezentační část

Pro prezentaci dat uživateli vytvoříme grafické rozhraní, které bude prezentovat data z klientského a serverového modulu. Naší cílovou platformou, pro kterou budeme tyto grafické rozhraní vyvíjet, bude Windows. Pro vývoj těchto aplikací je důležité zvolit si knihovnu, nad kterou bude grafické rozhraní budováno. Představu návrhu rozhraní si můžeme prohlédnout v sekci Uživatelské rozhraní.

Z této představy si můžeme ujasnit, jaké požadavky budeme klást na knihovnu, nad kterou budeme tato rozhraní budovat.

Jednou z hlavních částí rozhraní, která bude společná pro aplikaci nad klientským i nad serverovým modulem, bude virtuální ovládací kozole. Pro virtuální ovládací kozoli bude potřeba vyrobit si vlastní ovládací prvky, například joystick, proto musí námi vybraná knihovna podporovat vytváření si nových prvků. Druhým hlavní požadavkem bude, aby v situaci, kdy aplikace poběží na zařízení s dotykovým displejem bylo možné ovládat více ovladačů najednou, tedy aby zde byla podpora multitouch.

Pro aplikace z návrhu bude hlavním požadavkem, aby šly rozdělit na více oken a tato okna šlo různě přemisťovat a stále udržet pěkný vzhled aplikace.

K tomuto účelu máme na výběr ze dvou nejrozšířenějších knihoven a to Windows Forms nebo WPF. Obě knihovny mají své pro a proti, proto bude důležité vyzdvihnout si jejich výhody pro řešení našeho problému.

Výhody Windows Forms

Windows Forms je starší, měl by tedy mít i širší podporu knihoven třetích stran. Na internetové stránce [8] je popsán volně dostupný nuget balíček pro Windows Form, který nám poskytuje systém pro rozložení oken v rámci systému tak, jak bychom si přáli. Windows Forms podporuje použití WPF prvků [2].

Výhody WPF

WPF je novější a lépe podporuje vytváření vlastních uživatelských prvků. Ve WPF narozdíl od Windows Forms je přímo možné vytvářet prvky, které představují 3D prostor [4] a lépe podporuje multitouch. WPF podporuje použít Windows Forms prvků [3].

Použité technologie

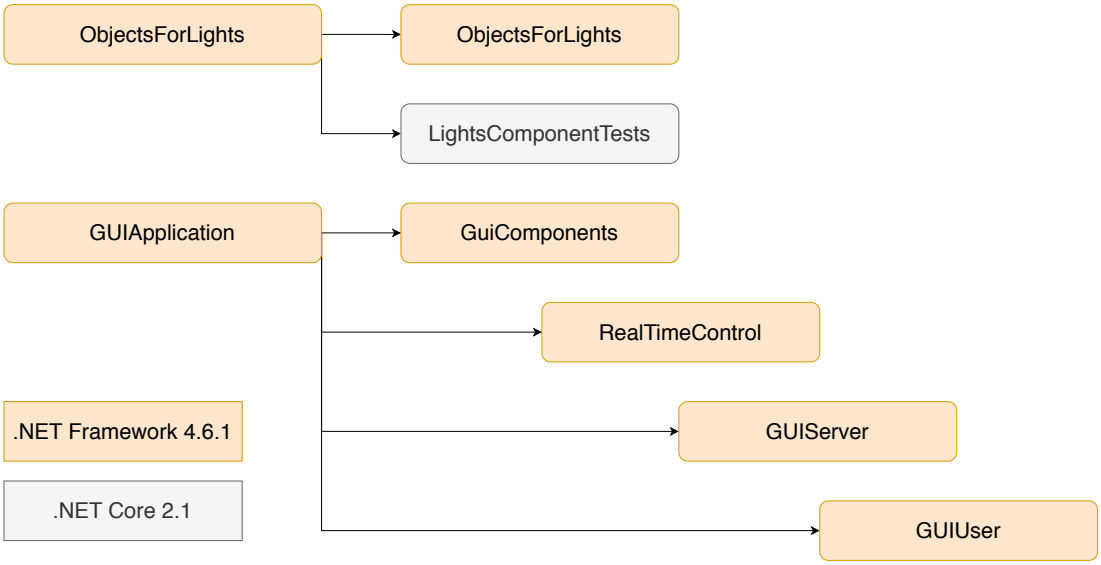
Z porovnání výhod vypadá, že Windows Forms je lepší volba pro budování aplikace jako celku, ale WPF bude lepší pro vývoj virtuální ovládací konzole. Vzhledem k tomu, že obě dvě knihovny se vzájemně podporují a jde je spojovat a vzájemně kombinovat, můžeme si dovolit využít obě dvě. Nosnou kostrou aplikací bude Windows Forms, ale v částech, jako je například virtuální ovládací konzole, kde bude potřeba použít složitějších ovládacích prvků nebo si vyrobit vlastní ovládací prvky s podporou multitouch, použijeme WPF.

3. Dokumentace implementace systému

Celý systém, jak můžeme vidět na obrázku 3.1, je rozdělený na dva hlavní Solution:

- Solution `ObjectsForLights`, který představuje backend pro tento systém. Tento solution obsahuje knihovnu `ObjectsForLights`, kde se nachází základní prvky a hlavně komunikační třídy představující klientský a serverový modul, dále pak tento solution obsahuje projekt s alespoň základními testy pro tuto knihovnu.
- Solution `GUIApplications`, který představuje implementaci prezentační vrstvy pro počítače. Solution `GUIApplications` je rozdělen na čtyři projekty:
  - Knihovna `GuiComponnents` poskytuje programátorovi prvky pro vývoj aplikace ve Windows Forms.
  - Knihovna `RealTimeControl` poskytuje programátorovi virtuální ovládací konzoli a WPF části, ze kterých je sestavena.
  - Projekt `GUIUser` představuje okenní aplikaci pro ovládání osvětlení jako klient.
  - Projekt `GUIServer` představuje okenní aplikaci, která je grafickým rozhraním pro server.

Vše má jako cílový framework .NET Framwork 4.6.1., až na projekt s testy, který má jako cílový framework .NET Core 2.1.



Obrázek 3.1: Rozložení projektu.

3.1 Architektura knihovny ObjectsForLights

Základním produktem celé knihovny `ObjectsForLights` jsou třídy `Client` a `Server`, které představují klientský a serverový modul tak, jak jsou popsány v kapitole 2 Analýza Projektu. Tyto třídy si po zkonstruování obstarávají veškeré udržování dat pro funkčnost systému, komunikaci mezi sebou a třída `Server` udržuje i komunikaci se zařízeními a jejich ovládání. Poskytují tak základ, nad kterým může být vybudováno grafické uživatelské rozhraní.

Udržování dat

Abychom splnili veškeré požadavky, které jsme si v předchozích částech zavedli, udržujeme ve třídách `Client` a `Server` data, která představují:

- Textovou komunikaci mezi moduly – `List<string> Conversation`
- Stavy zařízení – `Dictionary<IID,IState> Data`
- Informačních zprávy – `List<string> Errors`

Třída `Server` si navíc ještě udržuje informace o přihlášených klientských třídách s jejich uživatelskými jmény
– `Dictionary<IID,string> connected` a mapování zařízení s informacemi, jaký ovládací formát používají – `Dictionary<IID,IProtocol> map`.

`IID` je rozhraní pro identifikátory zařízení v systému. `IProtocol` je rozhraní pro komunikaci se zařízeními, jak je uvedeno v sekci 3.1Odesílání dat zařízením.

Aby nad těmito třídami a hlavně daty bylo možné zkonstruovat uživatelské rozhraní, ze kterého by bylo možné tato data zobrazovat a případně měnit, jsou tato data veřejná.

Data pro zařízení

K udržování dat pro zařízení je v systému implementována generická struktura `State<T>`, která představuje stav zařízení tak, jak jsme si ho navrhli v sekci 2.2.4 Stav zařízení. Stavy zařízení jsou závislé na tom, pro jaký konkrétní druh zařízení jsou určeny, protože jednotlivé druhy jsou ovládány pomocí různého počtu složek. Proto je nutné, aby u každého stavu bylo určeno, pro který druh zařízení je používán.

Kvůli určení druhu je v systému abstraktní třída `Kind<T>`, jejíž potomci představují jednotlivé druhy zařízení. Určují tak, o jaký druh zařízení se jedná a nesou informaci o počtu složek potřebných k ovládání tohoto druhu zařízení.

`State<T>` je tedy odvozené od `T`, které je potomkem třídy `Kind<T>` a představuje tedy druh zařízení.

Odesílání dat zařízením

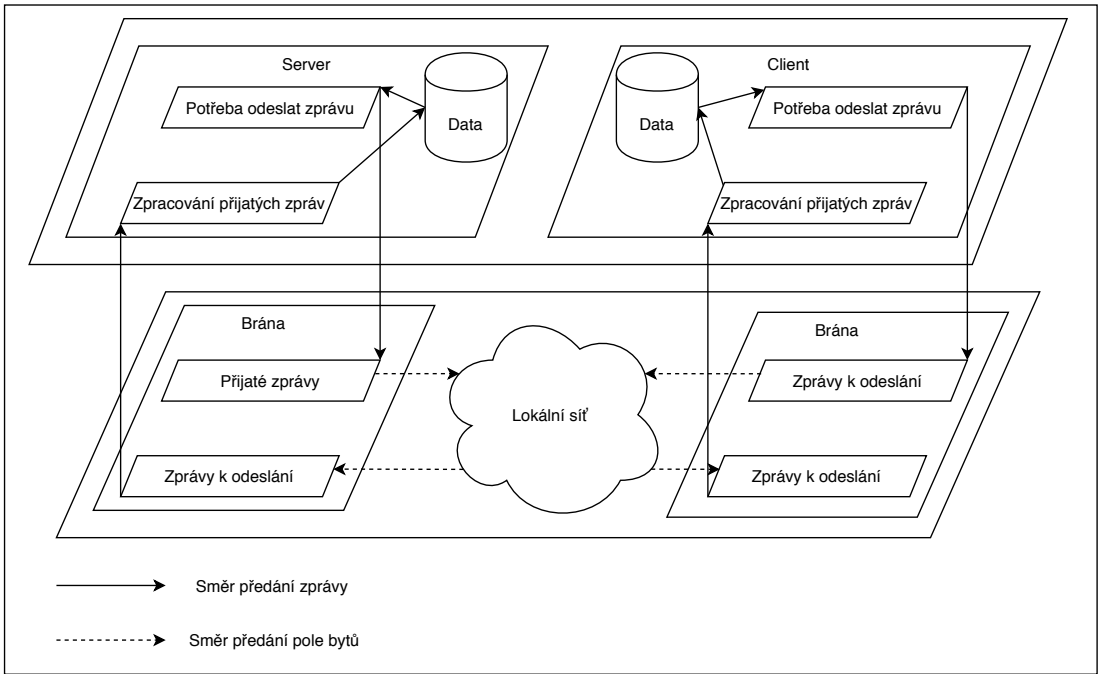
Aby třída `Server` mohla odesílat data jednotlivým zařízením, je potřeba, aby znala formát, ve kterém tato zařízení přijímají data, a uměla ze svých hodnot o stavech zařízení vygenerovat data v tomto formátu. Za tímto účelem jsou v knihovně třídy implementující rozhraní `IProtocol`. Rozhraní `IProtocol` je zde jako šablona a třídy, které ho implementují, pak poskytují metody, pomocí které

lze ze Stavu zařízení (`State<T>`) vygenerovat data pro zařízení ve formátu kterému rozumí. Serverová třída si proto ke každému zařízení udržuje třídu splňující rozhraní `IProtocol`. Při potřebě odeslat data zařízením pak nad touto třídou serverová třída zavolá funkci `byte[] ToBytes(IState data)`. Vzniklé pole bytů odpovídá formátu, kterému zařízení rozumí a tak jsou i schopna tato data zpracovat.

Komunikace

Aby se třídy `Client` a `Server` nemusely zabývat tím, jak komunikovat po síti a řešit serializaci a deserializaci dat, používají proto potomky abstraktní třídy `Gate<T>`. Jak můžeme vidět na obrázku 3.2, který představuje komunikaci mezi třídami `Client` a `Server`. Tito potomci, nadále jen brány, pak slouží k odesílání a přijímání dat typu `T` po lokální síti nad daným portem. Brány tak umožňují svým vlastníkům nestarat se o síťovou komunikaci. Brány pak přijímají data k odeslání a dále se o vše s tím spojené sami postarají a umožňují odebírat již přijaté a zpracované zprávy. Třídy `Client` a `Server` pak mají každá svou instanci brány pro komunikaci mezi sebou. Pro komunikaci mezi sebou používají tyto třídy zprávy typu `Message`, které představují zprávy nadefinované v sekci 2.3 Přenášené informace a jejich zpracování. Brány jsou pak typu `Gate<Message>`.

Třída `Server` má druhou bránu, pomocí které komunikuje se zařízeními. Pro tuto komunikaci `Server` používá zprávy typu `ByteMessage`, které obsahují pouze data k odeslání a cílovou adresu.



Obrázek 3.2: Komunikace pomocí bran tříd Client a Server.

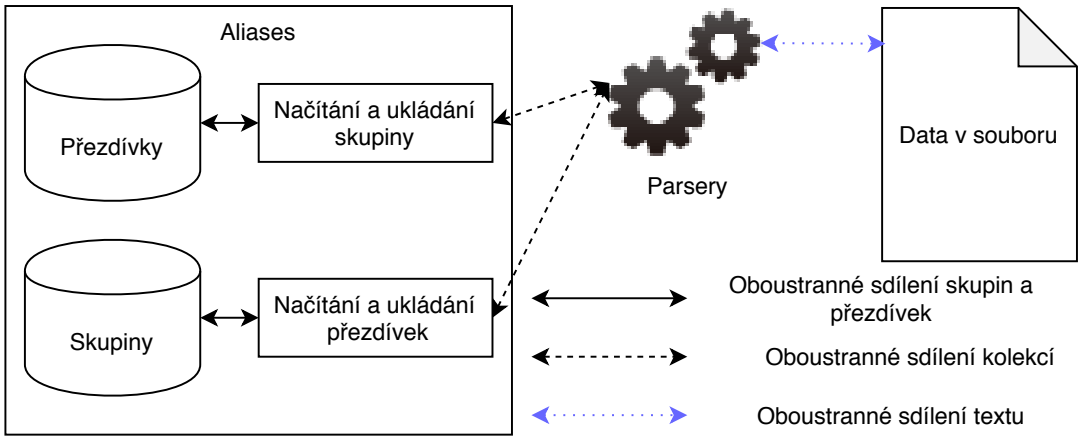
Pojmenování a skupiny zařízení

Pro splnění představy systému v sekci 1.3 Data v systému je v knihovně implementována třída `Aliases` poskytující seznamy přezdívek a skupin, kterou

obsahují obě třídy `Client` a `Server`. Tato třída slouží pouze pro podporu uživatelského rozhraní. Její data jsou rozdělena na dvě skupiny, a to sice:

- Data na pro spravování jmen jednotlivých zařízení a jmen jejich složek.
- Data o skupinách zařízení a skupinách složek zařízení.

K tomu, aby data mohla být použita opakovaně, je potřeba je ukládat, a proto jsou v systému parsovací třídy jako `ParsersJsonDictionary<K,V>` a `ParsersJsonList<T>`. Použití tříd pak můžeme vidět na obrázku 3.3. Při vytváření grafického rozhraní jsou tato data přístupná a grafické rozhraní je může použít, aby uživateli usnadnilo orientaci v zařízeních a jejich ovládání.



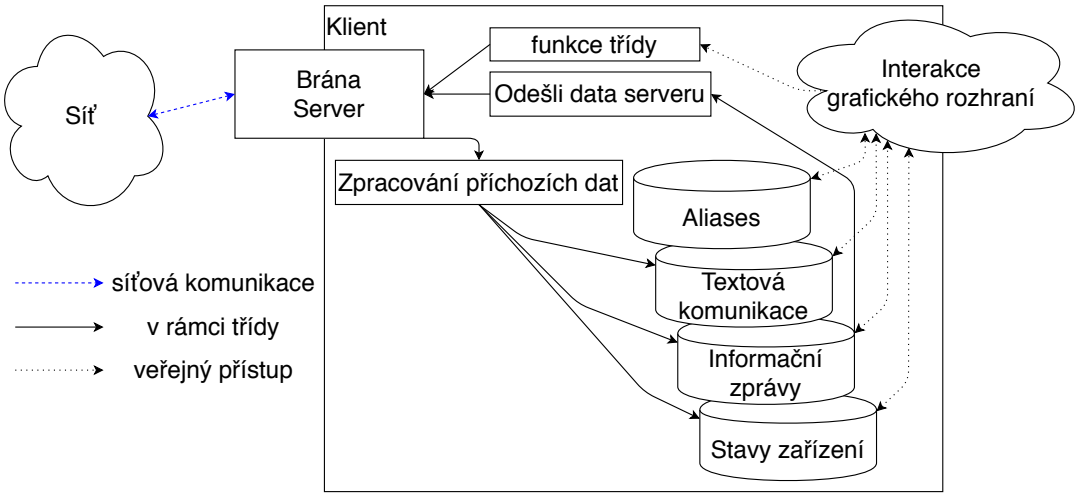
Obrázek 3.3: Komunikace Aliases s textovými soubory.

Client

Výsledná komunikace v rámci třídy `Client`, jak můžeme vidět na obrázku 3.4, je pak tvořena dvěma vlákny:

- Odešli data serveru – `UpdateServer`, které při změně stavů zařízení odešle aktuální stavy zařízení serveru.
- Zpracování příchozích zpráv – `processor`, které zpracuje příchozí zprávy a uloží je do patřičných dat.

Dále pak třída poskytuje funkce, skrze které je i z grafického prostředí možné zasílat zprávy a aktualizační informace. A k veškerým datům je povolen přístup pro případné grafické rozhraní.



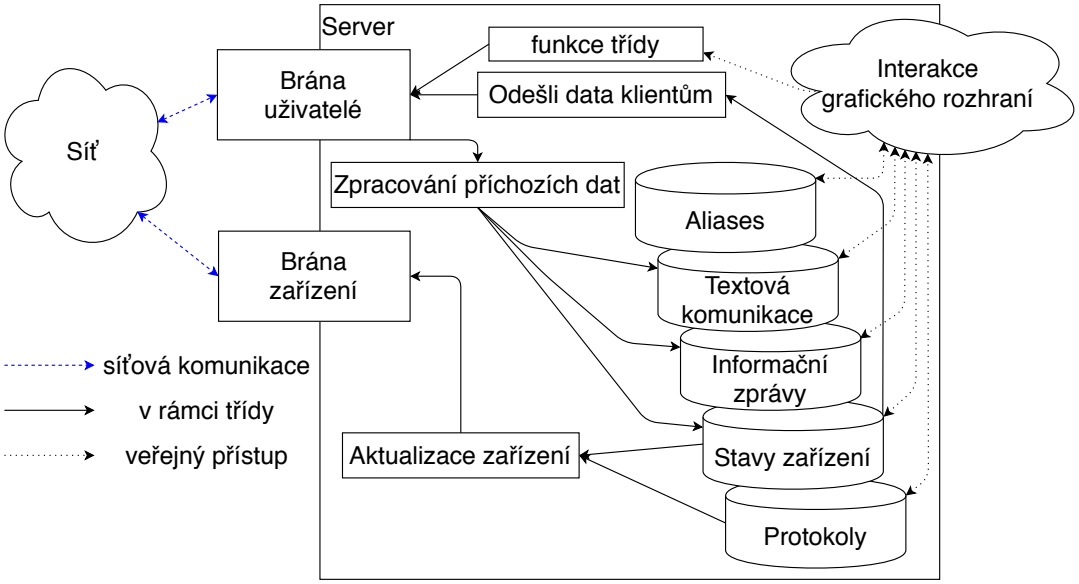
Obrázek 3.4: Propojení v rámci třídy Client.

Server

Výsledná komunikace v rámci třídy `Server`, jak můžeme vidět na obrázku 3.4, je pak tvořena třemi vlákny:

- Data klientům – `clientUpdator`, které při změně stavů zařízení odešle aktuální stavy zařízení klientům.
- Zpracování příchozích zpráv – `processor`, které zpracuje příchozí zprávy a uloží je do patřičných dat.
- Aktualizace zařízení – `sceneMaker`, které pravidelně zasílá aktuální stavy připojeným zařízením.

A k veškerým datům je povolen přístup pro případné grafické rozhraní.



Obrázek 3.5: Propojení v rámci třídy Server.

3.2 Knihovna ObjectsForLights

Knihovna `ObjectsForLights`, která nám představuje backend z části 2 Backend je rozdělena do několika oblastí jmenných prostorů:

- `EquipmentsKinds` – Druhy, na které jsou tříděny jednotlivé zařízení.
- `Transport` – Prvky připravené na přenos v rámci systému.
- `Message` – Zprávy pro přenos v rámci systému.
- `Protocols` – Třídy obsluhující formátování dat pro zařízení.
- `Gate` – Třídy sloužící pro odesílání a přijímaní UDP zpráv.
- `Comunication` – Třídy vytvářející serverovou a komunikační třídu.
- `Parsers`– Třídy a nástroje sloužící k ukládání a načítání dat z textové podoby. Třídy sloužící pro udržování a vytváření potřebných dat pro skupiny složek zařízení a skupin zařízení.

3.2.1 EquipmentsKinds – Druhy zařízení

V tomto jmenném prostoru jsou řešeny druhy zařízení tak, jak bylo navrženo v části 2.2.3 Druh zařízení.

ImplementedKinds – druhy zařízení

Pro udržování druhů zařízení je zde výčtový typ `enum ImplementedKinds`, ve kterém jsou uvedeny jednotlivé druhy zařízení, které náš systém podporuje. V současnou chvíli jsou v systému pouze tři druhy zařízení:

- Druh barevných světel `RGBLight = 3`
- Druh pohyblivých barevných světel `SpyderLight = 5`
- Stmívač `DIMMLight = 12`

Čísla u jednotlivých položek výčtového typu nejsou důležitá. V těchto případech jsou pro přehlednost zvolena tak, aby odpovídaly počtu složek jednotlivých zařízení, ale není to nutné.

Kind<T> – Druh zařízení

Jelikož pro každý druh zařízení bude potřeba vytvářet kontejner a druhy zařízení mají různé počty složek, bude v systému zaveden systém tříd, podle kterých se dále budou určovat počty složek těchto zařízení. Předlohou pro všechny druhy zařízení je abstraktní generická třída `Kind<T> where T : Kind<T>`, která obsahuje hodnoty `internal static byte TypeNumber` sloužící k určení druhu zařízení a `internal static byte Size`, která slouží k určení počtu složek zařízení.

Druhy zařízení

V současnou chvíli jsou v systému tři druhy zařízení:

- Druh barevných světel, který reprezentuje třída `RGBLight`
  s hodnotami `TypeNumber=ImplementedKinds.RGBLight` a `Size=3`
- Druh pohyblivých barevných světel, který reprezentuje třída `SpyderLight`
  s hodnotami `TypeNumber=ImplementedKinds.SpyderLight` a `Size=5`
- Stmívač, který reprezentuje třída `DIMMLight`
  s hodnotami `TypeNumber=ImplementedKinds.DIMMLight` a `Size=12`

KindSerialization – podpora druhů zařízení

Statická třída `KindSerialization` slouží pro práci s druhy zařízení, pro jejich serializaci a deserializaci do a z textové podoby a pro vytváření instancí jednotlivých druhů.

Tato třída podporuje následující funkce:

- `string Serialize(IState state)`
  Funkce, která ze stavu zařízení serializuje do textu pouze jeho druh.
- `IState Deserialize(string state)`
  Funkce, která ze serializovaného
  druhu vytvoří Stav pro tento druh.
- `IState GetState(ImplementedKinds state,ID id=default(ID))`
  Funkce, která podle výčtového typu vytvoří příslušný stav.
- `Type TypeNumberToKind(byte typeNumber)`
  Funkce, která podle `TypeNumber` určí `Type` druhu.

Zavedení nového druhu do systému

Pro zavedení nového druhu zařízení do systému je potřeba do výčtového typu `enum ImplementedKinds` přidat nový druh, naimplementovat třídu pro tento druh, která je potomkem třídy `Kind<T> where T : Kind<T>` a v jejím konstruktoru nastavit hodnoty `TypeNumber` na odpovídající hodnotu z
`enum ImplementedKinds` a `Size` na počet složek, který má toto zařízení mít. Ve třídě `KindSerialization` rozšířit metody tak, aby podporovaly nový druh.

3.2.2 Transport

Ve jmenném prostoru `Transport` se nachází implementace základních struktur potřebných pro udržování dat a k pozdější komunikaci po síti. Jelikož tyto struktury slouží pro komunikaci po síti, musí být všechny serializovatelné.

ID – identifikátor

Struktura `ID` implementující `interface IID`, představuje identifikátor zařízení v systému tak, jak je popsán v sekci Identifikace. Jedná se pouze o pole bytů a proto i při serializaci a deserializaci, je takto přenášeno.

Component<T> – Pole složek

Struktura `Component<T> where T : Kind<T>`, udržuje podle druhu zařízení potřebný počet složek k jeho ovládání.

Pro možnost přenosu této struktury je serializovatelná do a z pole bytu. Formát této serializace a deserializace je:

- 0. byte určuje hodnotu n, která představuje počet složek tohoto zařízení.
- 1.-n. byte, kde n je počet složek, pak představují hodnoty jednotlivých složek.

State<T> – stav zařízení

Generická struktura `State<T> where T : Kind<T>` představuje stav zařízení jak byl popsán v sekci Stav zařízení.

`State<T>` je generický, protože je závislý na druhu zařízení `T` pro který slouží, protože ten určuje, kolik složek má druh a tedy kolik hodnot si `State<T>` musí držet.

- `Id` – ID zařízení, pro které je tento stav určen.
- `Component<T> Parts` – Hodnoty pro jednotlivé složky.
- `Importance` – Hodnota představující důležitost, s jakou byl tento stav nastaven.
- `BitArray PartImportance` – Příznaky, jestli byla daná složka nastavena záměrně či nikoliv.

Serializace dat do pole bytů

Jak můžeme vidět v tabulce, stav zařízení je do bytů serializován tak, že první byte představuje druh zařízení, pak následuje identifikátor zařízení. Po identifikátoru je byte důležitost s jakou je stav nastaven, hodnoty jednotlivých složek zařízení a nakonec jsou připojeny příznaky nastavení. Příznak nastavení je reprezentován bity, proto velikost všech příznaků bytech je (počet složek)/8.

Tabulka 3.1: Návrh serializace hlavičky zprávy do bytů.

	pořadí	**délka**
druhy zařízení – TypeNumber	0	1
ID zařízení	1-10	10
důležitost stavů	11	1
data složky zařízení	12-n	počtu složek
příznaky nastavení – PartImportance	n-k	(počet složek)/8

TimeStamp – časové razítko

Pro určení času je zde struktura `TimeStamp`, obsahující hodnoty určující čas. První hodnotou, které určuje čas v rozsahu YY-MM-DD-hh-mm-ss, je `Int64 DateTimeStamp`. Druhou hodnotou, která určuje milisekundy, je `Int32 MilisecondsStamp`.

Tato struktura nadále poskytuje funkce:

- `static Int32 GetTimeStampDateTimed()` vracející aktuální čas v rozsahu YY-MM-DD-hh-mm-ss.
- `static Int32 GetTimeStampMilisecond()` vracející aktuální čas v rozsahu milisekund.
- `bool NewerOrTheSame(TimeStamp stamp)`, která určí, která známka je novější.

3.2.3 Messages – zprávy pro přenos dat

Ve jmenném prostoru `Messages` jsou nadefinovány zprávy, které je potřeba přenášet v rámci komunikace mezi klientským a serverovým modulem, ale i serverovým modulem a zařízeními.

Message – komunikace na straně klientů a serveru

Pro komunikaci mezi serverovým a klientským modulem jsou v systému zavedeny zprávy, které jsme si nadefinovali v sekci Přenášené informace a jejich zpracování. Tyto zprávy jsou potomkem třídy `Message` a implementují rozhraní `IMessage`. Třída message obsahuje identifikaci příjemce `IID Recipient`, odesílatele `IID Sender` a bytovou hodnotu `internal byte Type`, která určuje o jakou zprávu se jedná. Tato zpráva představuje hlavičkovou zprávu a je serializována, tak jak bylo popsáno při analýze.

Jednotlivé třídy odpovídají zprávám z Přenášené informace a jejich zpracování:

- `Chat` – Textová zpráva
  Zpráva přenášející textový řetězec představující zprávu textové komunikace mezi třídami.
- `Error` – Chybové zprávy
  Zpráva přenášející textový řetězec představující informační zprávu.
- `LogIn` – Uživatelská přihlašovací zpráva
  Zpráva přenášející textové řetězce představující jméno uživatele a heslo.
- `LogInConfirm` – Serverová potvrzovací zpráva o přihlášení
  Zpráva potvrzující úspěšné přihlášení na server.
- `LogOut` – Uživatelská odhlašovací zpráva
  Zpráva oznamující odhlášení ze serveru.
- `LogOutConfirm` – Serverová potvrzovací zpráva o odhlášení
  Zpráva potvrzující úspěšné odhlášení na server.

- `IntroducingServer` – Serverová představující zpráva
  Zpráva obsahující identifikátor serveru pro navázání komunikace.
- `IntroducingClient` – Uživatelská představující zpráva
  Zpráva obsahující identifikátor klienta pro navázání komunikace.
- `Scene` – Scénová zpráva
  Zpráva přenášející data o stavech jednotlivých zařízení.

Header – serializace zpráv

Třída `Message` je potomkem třídy `Header`. Třídě `Header` obsahuje výčtový typ `ImplementedMessages`, který obsahuje všechny typy implementovaných zpráv a funkci `void createCorrectMessage(byte[] data, out Message msg, IPAddress IPA)`, která z pole bytů `data` vyrobí podle typu zprávy správného potomka třídy `Message` a vrátí ho parametrem `out Message msg`.

Serializace zpráv

Zprávy, které nepřenášejí žádná přidaná data, pouze svůj typ a identifikátor příjemce a odesílatele, nastavují v konstruktoru `Type` a jsou pak serializovány stejně, jako hlavička, jak je vidět v tabulce 3.2.

Tabulka 3.2: Serializace hlavičky zpráv do bytů.

	pořadí	**délka**
IP adresa příjemce	0-9	10
IP adresa odesílatele	10-19	10
typ zprávy	20	1

Textové zprávy

Pro zprávy, které přenášejí textové pole, je zde abstraktní třída `TextMessages:Message`, která poskytuje metody pro serializaci a deserializaci více textových polí. Pokud zpráva přenáší pouze jeden textový řetězec, stačí ho serializovat a připojit za hlavičku. V případě, že zpráva přenáší více textových řetězců, jsou zde metody, které tyto řetězce spojí do jednoho a následně je i rozpojí. Jako oddělovač slouží znak $. V případě, že přenášený text obsahuje znak $, je tento znak odescapován pomocí symbolu \. Výsledná serializace je pak vidět v tabulce 3.3

Tabulka 3.3: Serializace textových zpráv do bytů.

	pořadí	**délka**
IP adresa příjemce	0-9	10
IP adresa odesílatele	10-19	10
typ zprávy	20	1
text	21-n	délka textu v bytech

Scene – serializace scénové zprávy

Scénová zpráva, jak můžeme vidět v tabulce 3.4, je serializována tak, že za hlavičku jsou připojeny jednotlivé části časové známky, první datumová a pak milisekundová, za těmito daty jsou pak jen postupně připojeny všechny serializované stavy, jaké tato zpráva obsahovala.

Tabulka 3.4: Serializace scénové zprávy do bytů.

	pořadí	**délka**
IP adresa příjemce	0-9	10
IP adresa odesílatele	10-19	10
typ zprávy	20	1
YY-MM-DD-hh-mm-ss	21-28	8
Milisekundy	29-32	4
Stavy zařízení	33-?	?

Přidání nových zpráv

Pro přidání nové zprávy k současným zprávám je nutné, aby nová zpráva byla potomkem `Message`. Ve třídě `Header` je pak nutné rozšířit výčtový typ `ImplementedMessages` o typ nové zprávy a funkci `createCorrectMessage` tak, aby vracela požadovanou třídu.

Samotné implementování nových zpráv je pak možná rozdělit na případy:

Zpráva bez přidaných dat Při implementování zprávy bez přidaných dat stačí podědit od třídy `Message` a nastavit v konstruktoru správný typ.

Zprávy s přidanými daty Při implementování zprávy, která obsahuje přidaná data, stačí podědit od třídy `Message`, nastavit v konstruktoru správný typ, vytvořit konstruktor s parametrem pole bytů, který vyplní obsah třídy a přepsat metody `byte[] ToBytes()` tak, aby ze serializovaného pole bytů bylo možné konstruktorem vytvořit obsah třídy tak jak, byl před serializací.

Textové zprávy Při implementování zprávy, která obsahuje textové řetězce, je možné použít třídu `TextMessages:Message`, která poskytuje metody sloužící pro podporu serializace a deserializace textových řetězců.

ByteMessage – komunikace na straně serveru a zařízení

`ByteMessage` je struktura, která drží pouze IP adresu, na kterou je potřeba zprávu zaslat a pole bytů, ve formátu, který je zařízení, kterému je tato zpráva zasílána, schopné zpracovat.

3.2.4 Gate – UDP brány

V tomto jmenném prostoru `Gate` se řeší odesílání a přijímání zpráv. Jelikož v systému jsou dvě části, které vyžadují různé formáty zpráv, ať už se jedná o naši

navrženou komunikaci ze sekce Přenášené informace a jejich zpracování, nebo komunikaci se zařízeními z odstavce Formát zpráv zařízení divadla Lípa.

Gate<T> – abstraktní generická UDP brána

Hlavní třídou představující UDP bránu je abstraktní generická třída `Gate<T>`, která slouží k tomu, že přijímá a odesílá zprávy daného typu `T`. Aby pak byla v programu co nejjednodušeji použitelná a potenciální programátor této třídě jen dával zprávy typu `T` k odeslání a odebíral zprávy požadovaného typu `T`, stará se sama o serializaci a deserializaci zpráv. Při použití potomků této třídy pak stačí, aby implementovaly nasledující tři funkce:

- `GetIpAddress`, která z odesílané zprávy získá IP adresu příjemce.
- `ToByteArray`, které odesílanou zprávu serializuje do pole bytů.
- `MessageMaker`, která z pole bytů vytvoří zprávu očekávaného typu.

Generická brána se pak sama stará o odesílání a přijímání dat. Pro zkonstruování brány je potřeba zadat port, na kterém má tato brána komunikovat. Při konstruování brány se nastaví flag `isActive` jako pravdivý a vytvoří se dvě prázdné fronty zpráv:

- `Queue<T> toSend`, do které jsou ukládány přijaté zprávy.
- `Queue<T> recieved`, do které jsou ukládány odchozí zprávy.

Dále se spustí dvě vlákna:

- `Thread InComing`, ve kterém po dobu aktivity této brány běží procedura `void inComing`, která se pokouší od kohokoliv přijmout pole bytů. Pokud přijme pole bytů, pomocí metody `MessageMaker` z něj vytvoří očekávaný typ a zařadí jej do do fronty `recieved`. Pokud tato procedura ve svém běhu nepřijme pole bytů, vzdá se svého běhu.
- `Thread OutGoing`, ve kterém po dobu aktivity této brány běží procedura `void outGoing`, která pokud je ve frontě `toSend` položka k odeslání, vyzvedne ji. Metodou `GetIpAddress` získá z položky IP adresu příjemce. Metodou `ToByteArray` tuto položku serializuje do pole bytů a odešle ji příjemci. Pokud ve frontě zpráv k odeslání není žádná zpráva, vlákno se vzdá svého běhu.

Pro ukončení brány je zde funkce `ShutDown`, která nastaví flaf `isActive` na nepravdivý a zastaví vlákna `InComing` a `Outgoing`.

GateByteMessage – komunikace se zařízeními

Pro komunikaci serverové třídy se zařízeními je v prostoru specifikována třída `GateByteMessage`, která je potomkem třídy `Gate<T>` a implementuje ji pro strukturu `ByteMessage`.

GateIMessage – komunikace serverové třídy s klientskými třídami

Pro komunikaci serverové třídy s klientskými třídami je v prostoru specifikována třída `GateIMessage`, která je potomkem třídy `Gate<T>` a implementuje ji pro rozhraní `IMessage`.

GateMessage<T> – obecná brána

Pro pozdější potřebu je zde vytvořena brána pracující s prvky implementující rozhraní `IGateSendable` a `new()`.

Rozhraní `IGateSendable` vynucuje metody:

- `IPAddress GetIpAddressOfRecipient()`, která získá z prvku implementující toto rozhraní IP adresu příjemce této položky.
- `byte[] MessageToByteArray()`, která prvek implementující toto rozhraní serializuje do pole bytů.
- `void FillMessageData(byte[] data, IPAddress IPA)`, která naplní prvek implementující toto rozhraní přijatými daty.

Nad těmito metodami jsou pak implementovány metody `GetIpAddress`, `ToByteArray` a `MessageMaker`.

3.2.5 Protocols

Ve jmenném prostoru `Protocols` se řeší transformace dat ze stavu zařízení `State<T>` na pole bytů pro příslušná zařízení.

Rozhraní IProtocol

Rozhraní `IProtocol` udává základní funkce, které každý Protokol musí implementovat. Jedná se o :

- `byte[] ToBytes(IState data)` Tato funkce ze Stavu zařízení `IState` vygeneruje pole bytů, kterému bude zařízení řízené tímto protokolem rozumět a bude jej správně interpretovat.
- `ImplementedKinds KindOfEquipment { get; }` Tato funkce vrátí druh zařízení, pro který je tento protokol určen.
- `byte[] DefaultMinValue { get; }` Pro získání maximálních hodnot zařízení.
- `byte[] DefaultMaxValue { get; }` Pro získání minimálních hodnot zařízení.

Poznámka: Maximální a minimální hodnoty se mohou lišit od očekávání, například maximální i minimální hodnoty pro Pohyblivé světlo řízené protokolem `SpyderLights` obsahují v obou případech složky pro osy v základní pozici.

Implementace pro současný stav divadla

Dále se v tomto jmenném prostoru nachází implementace rozhraní `IProtocol` pro současné zařízení a jejich formáty. Základní skupinkou jsou protokoly sloužící pro komunikaci se současnými zařízeními `AsciiRGB`, `AsciiSpyder` a `AsciiDIMM` s formátem dat, jak byl uveden v sekci Formát zpráv zařízení divadla Lípa. K těmto protokolům je implementován také `AsciiRGBWrong` protokol pro ovládání starých barevných světel, které nepoužívají řídící znak, jak bylo zmíněno v sekci Druh zařízení.

ProtocolFactory – podpora protokolů

Nachází se zde statická třída `ProtocolFactory`, poskytující výčtový typ implementovachých protokolů `enum ImplementedProtocols` a následující metody pro práci s protokoly:

- `IProtocol ChooseProtocol(string protocolTXT, ID id = default(ID), Dictionary<IID, IState> defaultValues = null)`
  Ze jména v textovém řetězci určí, o jaký protokol se jedná, a v případě že je zadáno `id` a kolekce `defaultValues`, vygeneruje do slovníku základní stav zařízení pro tento protokol.
- `string ProtocolToString(IProtocol protocol)`
  Ze třídy protokolu vygeneruje textový řetězec.
- `IProtocol GetProtocol(ImplementedProtocols protocol)`
  Pro hodnoty z výčtového typu `enum ImplementedProtocols` vrací příslušné instance protokolů.

Zavedení nového protokolu pro komunikaci se zařízeními do systému

Pro zavedení nového protokolu do systému je potřeba do výčtového typu `enum ImplementedProtocols` přidat nový protokol, naimplementovat třídu protokolu, která implementuje rozhraní `IProtocol`. Ve třídě `ProtocolFactory` rozšířit metody, tak aby podporovaly nový protokol.

3.2.6 Communication – CommunicationClass

Hlavním úkolem celé knihovny je poskytnout serverový a klientský modul. V tomto jmenném prostoru jsou vytvořeny třídy `Client` a `Server`, které slouží jako klientský a serverový modul. Jelikož tyto třídy mají hodně společných funkcí a vlastností, obě jsou potomkem třídy `CommunicationClass`.

CommunicationClass

`CommunicationClass` je abstraktní třída, která obsahuje společné metody a datové struktury sloužící oběma třídám. Hlavními datovými strukturami jsou kolekce

- `List<string> Conversation`, která slouží pro uchovávání zpráv textové komunikace mezi serverem a klientskými třídami.

- `List<string> Errors`, která slouží pro uchovávání informačních zpráv.
- `Dictionary<IID,IState> Data`, která slouží pro uchovávání stavů jednotlivých zařízení.

Jelikož chceme nad těmito třídami vybudovat grafické rozhraní, bude potřeba sdělovat grafickému rozhraní, že došlo ke změně. V systému poběží více vláken a různě budou provádět změny nad daty, které jsou graficky reprezentovány. Proto, aby bylo možné dát rozhraní vědět, že došlo ke změně, budou v systému pro jednotlivé kolekce příznaky změny. Tyto příznaky jsou plně v režii prezentační části, klientská a serverová třída má za úkol jen to, že pokud nějaká změní, tak nastaví patřičný příznak na pravdivý.

- `public bool ConversationCFO` – Příznak, že došlo ke změně na datech textové komunikace.
- `public bool ErrorsCFO` – Příznak, že došlo ke změně na datech informačních zpráv.
- `public bool DataCFO` – Příznak, že došlo ke změně na datech pro uchovávání stavů jednotlivých zařízení.

Obecně pravdivostní hodnoty ve klientské a serverové třídě končící `CFO` signalizují změnu na datech.

Problém s výběrem adresy

Zařízení obyčejně přijímá UPD datagramy z více rozhraní. To nastává i v našem případě. Aplikace představující server bude připojena jak do ethernetové sítě, tak i do wifi sítě. Pro každé rozhraní má pak zařízení různou IP adresu. Pro to, aby při odesílání zpráv třída použila správnou IP adresu, je v systému funkce `IID GetIDForCommunication(IID Aim,int port)`, která podle identifikace příjemce určí, jaká identifikace odesílatele má být použita při odesílání zprávy.

3.2.7 CommunicationClass – Server – serverový modul

Hlavní třídou této knihovny je třída `Server`, která splňuje požadavky na serverový modul.

Pro použití třídy `Server` je potřeba ji zkonstruovat s parametry představující porty pro komunikaci s klientskou třídou a zařízeními. Tyto porty musí být různé. Při konstruování Serverové třídy se kromě inicializace potřebných kolekcí pro udržování dat a nastavení příznaku `isActive`, který určuje, jestli je server stále aktivní, jako pravdivý, vytvoří dvě brány, kdy brána `GateClients` je zkonstruována s portem pro komunikaci s klienty a brána `GateEquipments` je zkonstruována s portem pro komunikaci se zařízeními. Od zkonstruování bran se předpokládá, že pracují jak je uvedeno v sekci Gate – UDP brány.

Po zkonstruování bran jsou spuštěna hlavní tři vlákna `processor`,`sceneMaker` a `clientUpdator`, kdy tato vlákna už nadále sama zodpovídají za chod serveru.

Zpracovávání příchozích zpráv – vlákno processor

Vlákno `processor` je zodpovědné za zpracovávání příchozích zpráv, v tomto vlákně je spuštěna funkce `processor`, která, dokud je server aktivní, vybírá od `GateClients` přijaté zprávy. Pokud není žádná zpráva ke zpracování, vlákno se vzdá svého běhu. Pokud je přijatá zpráva ke zpracování, pak je metodou `void callCorrectAction(Message msg)` podle jejich typu rozhodnuto, jak má být zpracována a je zpracována funkcí `processMsg`.

Seznam funkcí `processMsg` a jak zpracovávají jednotlivé zprávy:

- `processMsg(Chat msg)`
  Zařadí zprávu textové komunikace do svého seznamu zpráv `List<string> Conversation` a pošle ji přihlášeným klientským třídám.
- `processMsg(LogIn msg)`
  Zpracuje požadavek klienta na přihlášení.
- `processMsg(LogOut msg)`
  Zpracuje požadavek klienta na odhlášení.
- `processMsg(IntroducingClient msg)`
  Odpoví klientovi svou `IntroducingServer`.
- `processMsg(IntroducingServer msg)`
  Nijak nereaguje.
- `processMsg(Error msg)`
  Zařadí text zprávy do svého seznamu zpráv `List<string> Errors`.
- `processMsg(Message msg)`
  Zařadí informaci o přijaté neznámé zprávě do svého seznamu zpráv `List<string> Errors`.

Důležité je uvědomit si, že všechny přijaté zprávy jsou zpracovávány v jednom vlákně. To znamená, že v případě, že se mnoho klientů bude pokoušet současně měnit hodnoty pro zařízení, může docházet k tomu, že server bude zahlcený. Na druhou stranu je nutné vzít v potaz, že jediné zprávy, které jsou relativně dlouhé na zpracování, jsou zprávy se stavem scény. A zprávy se stavem scény jsou jen velmi obtížně paralelizovatelné kvůli velkému množství parametrů, které rozhodují, jestli budou daná data použita, případně jestli bude použita alespoň jejich část. Navíc hlavním požadavkem není, aby server byl připraven na to, že na něj bude současně spousta uživatelů posílat velké množství dat, ale aby mu občas poslalo data více uživatelů. Jak již bylo dříve zmíněno, typickým použitím je, že veškerá světla stále ovládá jeden osvětlovač a jen v okrajových situacích se mu do toho někdo snaží zasáhnout.

Odesílání dat zařízením – vlákno sceneMaker

Vlákno `sceneMaker` je zodpovědné za odesílání aktuálních dat na serveru zařízením, pro které má server data. V tomto vlákně je spuštěna funkce `void DataSender()`, která dokud je server aktivní, projde data pro zařízení a zařízením, pro které má protokol, pomocí protokolu nechá vyrobit byte[], které pak přes EquipmentsGate odešle zařízením.

Odesílání zpráv o stavu zařízení klientům klientům – clientUpdator

Vlákno `clientUpdator` je zodpovědné za odesílání aktuálních dat na serveru připojeným klientům. V tomto vlákně je spuštěna funkce
`void UpdateConnectedThread`, která dokud je server aktivní, odesílá data pro zařízením všem klientům. Pošle je vždy, když dojde na datech ke změně, případně po určitém intervalu.

Ukončení serveru

Pro ukončení činnosti serveru poskytuje třída funkci `void ShutDown()`, která nastaví příznak serveru `isActive` na nepravdivý, zavolá `void ShutDown()` na `GateClients` i na `GateEquipments`, a ukončí všechna svoje vlákna.

3.2.8 CommunicationClass – Client – klientský modul

Pro použití třídy `Client` je potřeba ji zkonstruovat s parametrem představujícím port pro komunikace se serverovým modulem. Při konstruování třídy `Client` se kromě inicializace potřebných kolekcí pro udržování dat a nastavení příznaku `isActive`, který určuje, jestli je klient stále aktivní, jako pravdivý, vytvoří brána `GateServer`, která je zkonstruována s portem pro komunikaci se serverem. Od zkonstruování brány se předpokládá, že pracuje, jak je uvedeno v sekci Gate – UDP brány. Po zkonstruování brány jsou spuštěna dvě vlákna `processor`, `UpdateServer`, kdy tato vlákna nadále už zodpovídají za chod klienta.

Zpracovávání příchozích zpráv – vlákno processor

Vlákno `processor` je zodpovědné za zpracovávání příchozích zpráv. V tomto vlákně je spuštěna funkce `processor`, která dokud je klient aktivní, vybírá od `GateServer` přijaté zprávy. Pokud není zpráva ke zpracování, vlákno se vzdá svého běhu, pokud je přijatá zpráva ke zpracování, pak je metodou
`void callCorrectAction(Message msg)` podle jejího typu rozhodnuto, jak má být zpracována a je zpracována funkcí `processMsg`.

Seznam funkcí `processMsg` a jak zpracovávají jednotlivé zprávy.

- `processMsg(Chat msg)` Zařadí zprávu textové komunikace do svého seznamu zpráv `List<string> Conversation` a pošle ji přihlášeným klientským třídám.
- `processMsg(LogInConfirm msg)` Nastaví příznak `ConnectedToServer` jako pravdivý a do informačních zpráv zanese zprávu o úspěšném přihlášení.
- `processMsg(LogOutConfirm msg)` Nastaví příznak `ConnectedToServer` jako nepravdivý a do informačních zpráv zanese zprávu o úspěšném odhlášení.
- `processMsg(IntroducingClient msg)` Nijak nereaguje.
- `processMsg(IntroducingServer msg)` Do seznamu serveru `List<IID> Servers` si zařadí ID odesílatele této zprávy.

- `processMsg(Error msg)` Zařadí text zprávy do svého seznamu zpráv `List<string> Errors`.
- `processMsg(Message msg)` Zařadí informaci o přijaté neznámé zprávě do svého seznamu zpráv `List<string> Errors`. Tato metoda je pro zprávy, které nejsou implementovány ke zpracování.

Odesílání zpráv o stavu zařízení serveru – UpdateServer

Vlákno `UpdateServer` je zodpovědné za odesílání aktuálních dat na klientu serveru. V tomto vlákně je spuštěna funkce `void updateServer()` která, dokud je klient aktivní, odesílá data pro zařízením serveru vždy, když dojde na datech ke změně.

Ukončení klienta

Pro ukončení činnosti klienta poskytuje třída funkci `void ShutDown()`, která nastaví příznak `isActive` na nepravdivý, zavolá `void ShutDown()` na `GateServer` a ukončí všechna svoje vlákna.

Získání adresy serveru

Při analyzování navazování komunikace klientského modulu se serverovým modulem v sekci 2.3Přenášené informace a jejich zpracování, jsme se rozhodli, že za účelem získání IP adresy serveru bude v systému zpráva k tomu sloužící. Tuto komunikaci je potřeba provést ještě než je vytvořena klientská třída, aby klientská třída mohla být zkonstruována s IP adresou serveru. Za tímto účelem Klientská třída poskytuje statickou funkci `IID GetServerID(int port, int time)`, která se v případě, že v síti je server, o tuto výměnu dat sama postará. Této funkci je potřeba zadat port, na kterém je komunikace očekávána a čas, po který má čekat na případnou odpověď serveru.

3.2.9 Parsers

Ve jmenném prostoru `Parsers` se nachází třídy sloužící pro ukládání a načítání dat z textové podoby a pro udržování dat, které jsou požadovány v sekciData v systému.

Ukládání dat

V systému je potřeba ukládat a načítat data z textové podoby. Pro obecná data se zde nacházejí generické statické třídy `ParsersJsonList<T>` a `ParsersJsonDictionary<K,V>`, které používají nuget balíček Newtonsoft.Json [11]. Statická generická třída `ParsersJsonList<T>` slouží k serializování a deserializování dat, která jsou uložena v kolekci `List<T>`. Statická generická třída `ParsersJsonDictionary<K,V>` slouží k serializování a deserializování dat, která jsou uložena v kolekci `Dictionary<K,V>`.

Aliases – skupiny a přezdívky

V sekci Data v systému je určeno, že bude potřeba vytvářet skupiny zařízení, skupiny složek zařízení a poskytovat uživatelům možnost přiřazovat jednotlivým zařízením či složkám zařízení pojmenování. Z tohoto důvodu je zde třída `Aliases`, která se stará právě o tyto funkce.

Tato třída v první řadě poskytuje funkce:

- `LoadNickNames`, která slouží k načítání dat ze souborů, které obsahují jména pro zařízení či složky zařízení.
- `SaveNickNames`, která slouží k ukládání dat z této třídy do souboru.
- `LoadGroups`, která slouží k načítání dat ze souborů, které obsahují skupiny.
- `SaveGroups`, která slouží k ukládání dat z této třídy do souboru.

A dále pak slouží, aby nad ní bylo možné tyto data zpracovávat z grafického rozhraní pro uživatele.

3.2.10 LightsComponentsTests – testování serializace

Pro knihovnu je napsána i sada NUnit testů sloužící pro ověření základní serializace a deserializace zpráv a struktur, které budou sloužit pro síťovou komunikaci [1].

3.3 Architektura aplikací

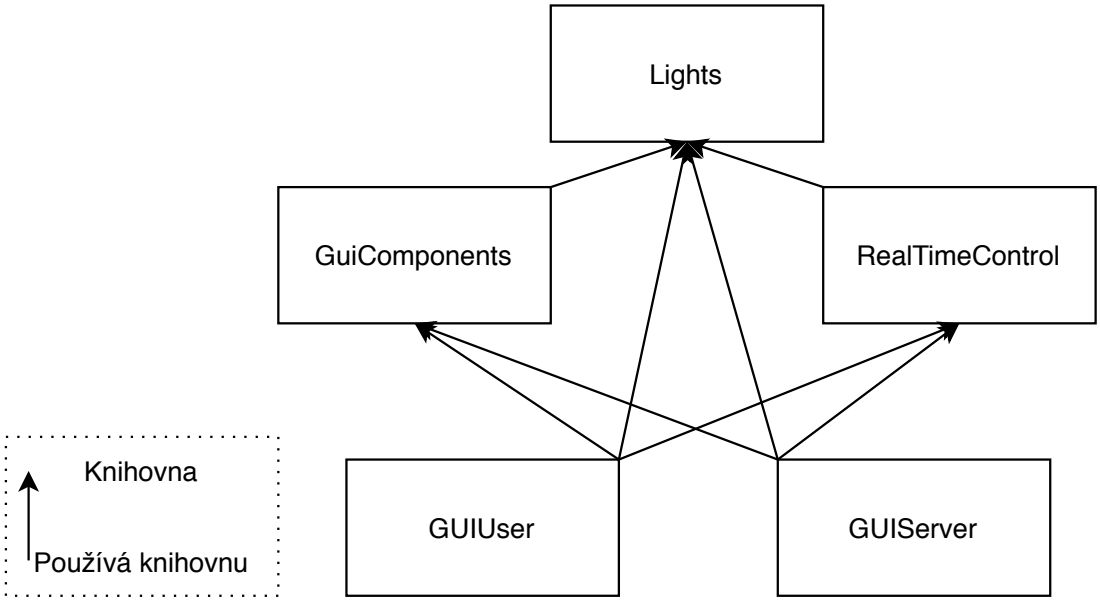
Jak již bylo dříve řečeno, v solution GUIAplications se nachází dva spustitelné projekty:

- `GUIUser` je uživatelská aplikace, která představuje klienta připojeného na server.
- `GUIServer` je uživatelská aplikace, která představuje server.

a dvě knihovny obsahují grafické prvky:

- `GuiComponents` poskytuje základní potřebné Winforms prvky.
- `RealTimeControl` obsahuje virtuální ovládací konzoli a WPF prvky na její konstrukci.

Jak můžeme vidět na obrázku 3.6, všechny tyto projekty pak používají knihovny `ObjectsForLights` a spustitelné projekty pak používají knihovny `GuiComponents` a `RealTimeControl`.



Obrázek 3.6: Použití knihoven.

3.3.1 Rozhraní klienta – GUIUser

Hlavním oknem celé aplikace je formulář `UserHome`, který jako jediný konstrukční parametr přijímá instanci třídy `Client`, nad kterou se pak zkonstruuje.

Login

Aby mohl vzniknout formulář `UserHome`, je potřeba mít třídu `Client`. K jejímu vytvoření je potřeba znát základní údaje jako je port, na kterém probíhá komunikace se serverem, IP adresu serveru, uživatelské jméno a heslo.

Proto první věcí, která se při spuštění aplikace zobrazí, je přihlašovací formulář `LoginForm`, který slouží pro vyplnění informací potřebných pro vytvoření třídy `Client` a pro přihlášení na `Server`. Po přijmutí dat aplikace vytvoří třídu `Client` a pokusí se přihlásit na `Server`. V případě neúspěšného přihlášení `Server` je uživateli tato skutečnost oznámena a uživatel dostane možnost používat nepřihlášenou aplikaci, případně se aplikace ukončí.

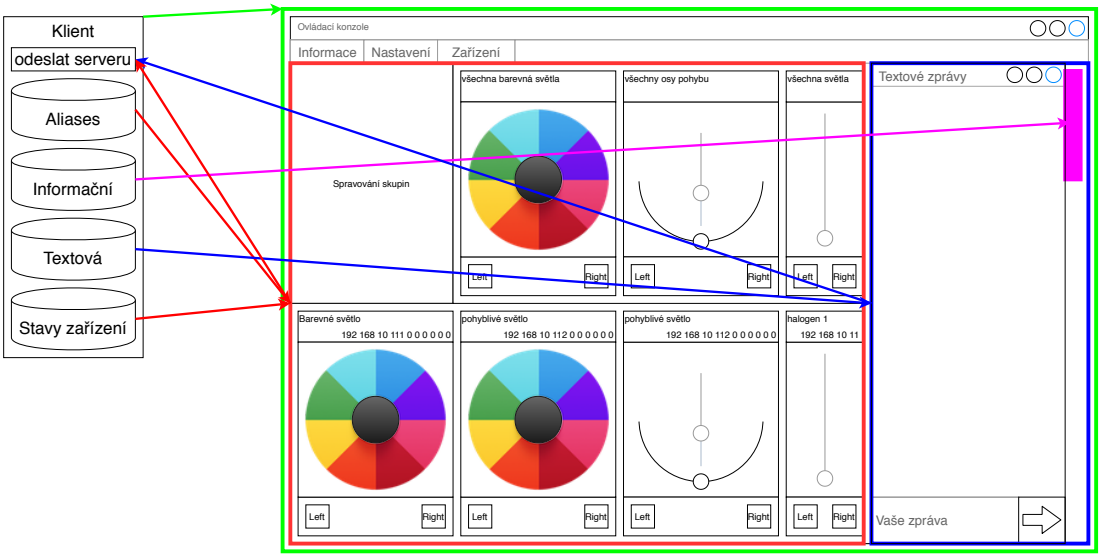
Hlavní okno

Formulář `UserHome` podporuje dockování formulářů do sebe a sám obsahuje pouze jednoduché menu, které slouží k nahrávání a ukládání dat do souborů. Při konstruování formuláře `UserHome` jsou pak nad daty z třídy `Client` zkonstruovány a zadockovány formuláře, tak jak byly naznačeny v části 1.4.2 Počítačová verze rozhraní pro uživatele.

Nad daty textové konverzace z `Conversation` je vytvořen formulář `Messenger chat`. Nad daty informačních zpráv `Errors` je vytvořen formulář `Messenger errors`. A nad daty `Aliasses` a `Data` je vytvořena virtuální ovládací konzole `RealTimeSetting RealTimeSetter`.

Pro aktualizování zobrazení je zde funkce `UpdateTimeLayout`, která je zavolána po daném časovém intervalu a jednotlivým oknům jsou zde volány funkce pro

aktualizaci vzhledu. Náčrt komunikace aplikace s jednotlivými položkami třídy `User` si pak můžeme prohlédnout na obrázku 3.7.



Obrázek 3.7: Komunikace klientské třídy a aplikace.

3.3.2 Rozhraní serveru – GUIServer

Hlavním oknem celé aplikace je formulář `ServerHome`, který jako jediný konstrukční parametr přijímá instanci třídy `Server`, nad kterou se pak zkonstruuje.

Nastavení

Aby mohl vzniknout formulář `ServerHome`, je potřeba mít třídu `Server`. K jejímu vytvoření je potřeba znát základní údaje jako je port, na kterém probíhá komunikace s klienty, port na kterém probíhá komunikace se zařízeními a heslo, které musí uživatelé zadat pro přihlášení do komunikace.

StartSetting

Proto první věcí, co se při spuštění aplikace zobrazí, je nastavovací formulář `StartSetting`, který slouží pro vyplnění informací potřebných pro vytvoření třídy `Server`. Po přijmutí dat, aplikace vytvoří třídu `Server` a vytvoří hlavní okno `ServerHome`

Hlavní okno

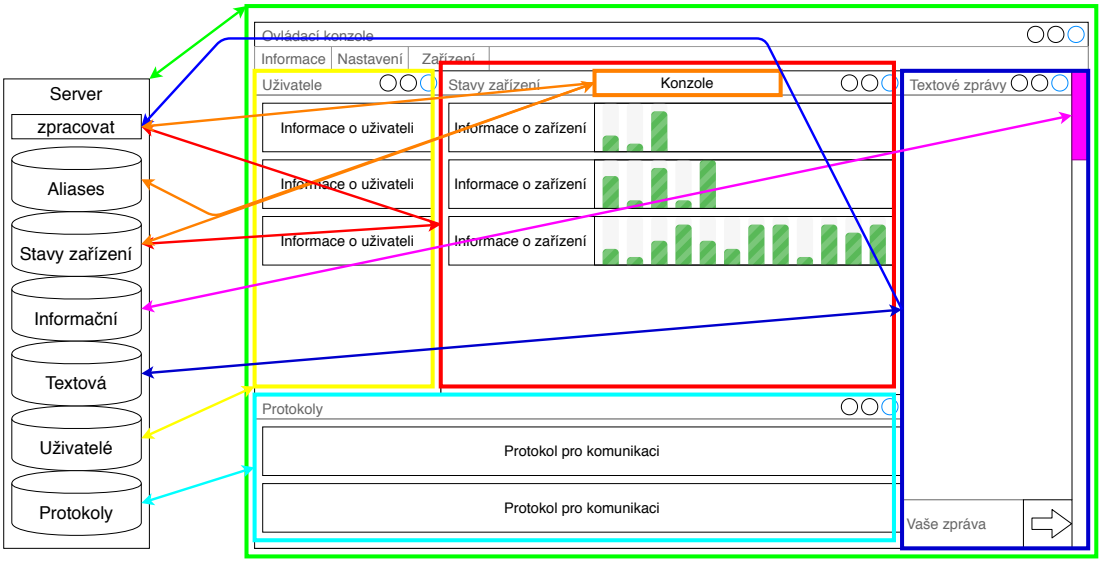
Formulář `ServerHome` podporuje dockování formulářu do sebe a sám obsahuje pouze jednoduché menu, které slouží k nahrávání a ukládání dat do souborů. Při konstruování formuláře `ServerHome` jsou pak nad daty z třídy `Server` zkonstruovány a zadockovány formuláře, tak jak byli naznačeny v návrhu 1.4.3 Počítačová verze rozhraní pro server.

Nad daty textové konverzace z `Conversation` je vytvořen formulář `Messenger chat`. Nad daty informačních zpráv `Errors` je vytvořen formulář `Messenger errors`.

Nad daty o přihlášených uživatelích `Connected` je vytvořen formulář `Data<IID,string> connected` a nad daty určujícími jednotlivým zařízením protokoly `Map` je vytvořen formulář `Data<IID,IProtocol> map`.

Nad stavy zařízení `Data` je vytvořen formulář `EquipmentsData` `DataToLights`, který zobrazuje data zařízení a umožňuje nad nimi základní nastavování hodnot. V případě potřeby, že je toto nastavování dat zařízením nedostatečné, lze si zde otevřít virtuální ovládací konzole `RealTimeSetting` `RealTimeSetter`. Náčrt komunikace aplikace s jednotlivými položkami třídy `Server` si pak můžeme prohlédnout na obrázku 3.8.

Pro aktualizování zobrazení je zde funkce `UpdateTimeLayout`, která je zavolána po daném časovém intervalu a jednotlivým oknům jsou zde volány funkce pro aktualizaci vzhledu.



Obrázek 3.8: Komunikace serverové třídy a aplikace.

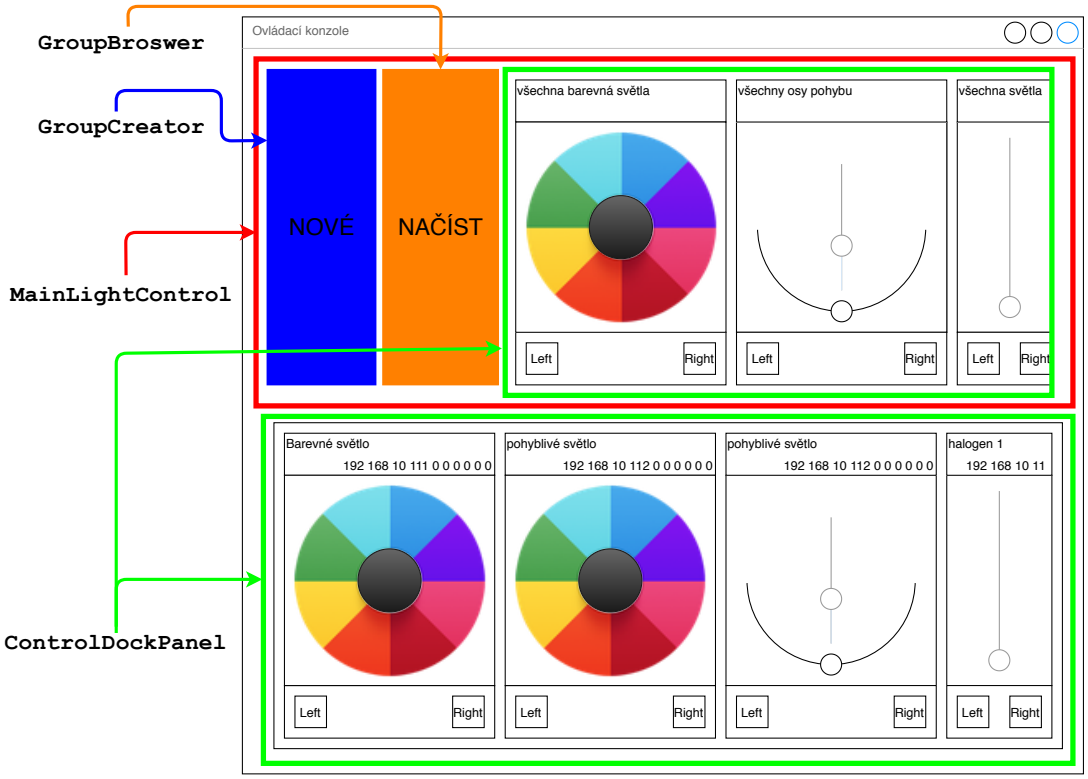
3.3.3 Virtuální ovládací konzole – RealTimeControl

Hlavním produktem knihovny `RealTimeControl` je virtuální ovládací konzole `RealTimeSetting`. Přestože je skoro celá konzole vytvořena ve WPF, aby byla lépe použitelná z uživatelských aplikací, které jsou ve Windows Forms, tak je umístěna do Windows Forms okna.

Jak můžeme vidět na obrázku 3.9, konzole je pak rozdělena na dvě části, a to vrchní část s obsluhou skupin a ovladači ovládající všechna zařízení – `MainLightControl` a dolní část, která slouží k zobrazování ovládacích prvků pro skupiny a jednotlivá zařízení – `ControlDockPanel`.
Lišta pro ovladače `ControlDockPanel` pak slouží pro udržování ovladačů a jejich velikosti. Horní část `MainLightControl` je pak rozdělena na ovladače pro práci se skupinami a lištou ovladačů pro všechna zařízení. Lišta pro všechna zařízení je tvořena `ControlDockPanel`.

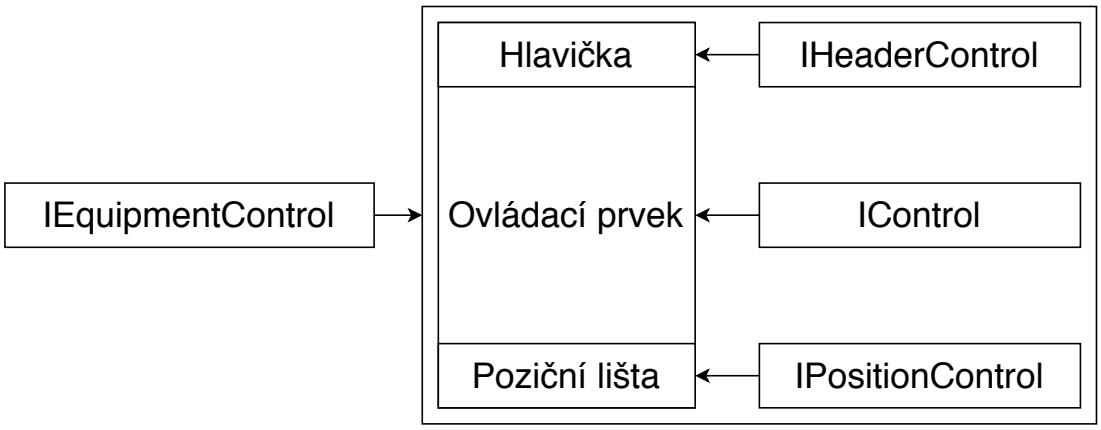
Pro práci se skupinami se v konzoli nachází dva ovladače.

- `GroupCreator`, který je odpovědný za vytváření nových skupin.
- `GroupBrowser`, který se stará o načítání a ukládání skupin do souborů a umožňuje vybrané skupiny používat.



Obrázek 3.9: Rozdělení konzolové aplikace.

Jak můžeme vidět na obrázku 3.10, jednotlivé ovladače pak implementují rozhraní `IEquipmentControl` a jsou tvořeny třemi částmi tak, jak bylo uvedeno při představě projektu v sekci 1.4.1 Kostra ovládacího prvku. Vrchní část představuje hlavičku a implementuje rozhraní `IHeaderControl`. Prostřední část představuje ovladač a implementuje rozhraní `IControl`. Spodní část představuje poziční lištu a implementuje rozhraní `IPositionControl`.



Obrázek 3.10: Spojení rozhraní s návrhem.

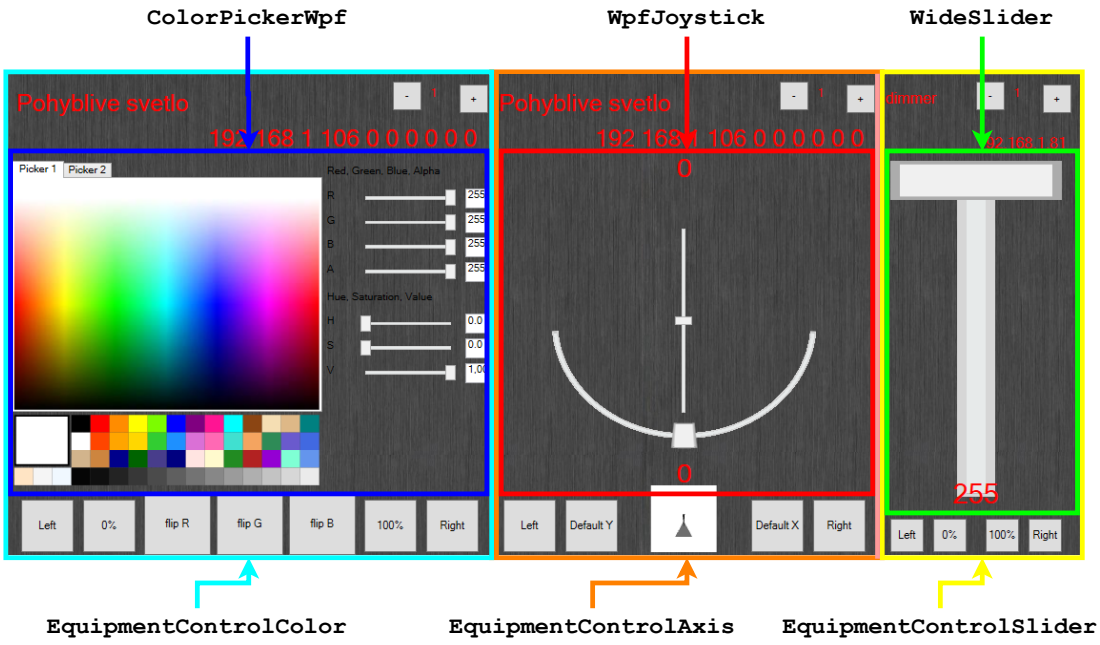
Rozhraní `IControl` pak splňují ovládací prvky:

- `WpfJoystick` sloužící pro nastavování os.
- `ColorControl` a `ColorPickerWpf` sloužící pro nastavování barev.
- `WideSlider` sloužící pro nastavování hodnoty.

Tyto prvky odpovídají ovládacím prvkům ze sekce 1.4.1 Ovládací prvek a nad nimi jsou pak zkonstruované celé ovladače splňující rozhraní `IEquipmentControl`:

- `EquipmentControlAxis`
  – Ovladač pro nastavování hodnot složek os zařízení.
- `EquipmentControlColor`
  – Ovladač pro nastavování hodnot barevných složek zařízení.
- `EquipmentControlSlider`
  – Ovladač pro nastavování hodnoty složce zařízení.

Výslednou podobu ovládacích prvků a celých ovladačů si pak můžeme prohlédnout na obrázku 3.11.



Obrázek 3.11: Implementace ovladačů a ovládacích prvků.

Pro generování ovladačů je v systému třída `RepresentStateFactory` , která na základě druhu Stavu zařízení, který dostane zadaný, generuje do `ControlDockPanel` ovladače.

3.4 Solution GUIAplications

V tomto solutionu jsou knihovny obsahující grafické prvky a aplikace představující serverové a klientské rozhraní. Solution je rozdělený na základní čtyři projekty, kde:

- Knihovna `GuiComponnents` poskytuje programátorovi prvky pro vývoj aplikace ve Windows Forms.
- Knihovna `RealTimeControl` poskytuje programátorovi virtuální ovládací konzoli a části, ze kterých je sestavena.
- Projekt `GUIUser` představuje okenní aplikaci pro ovládání osvětlení jako klient.
- Projekt `GUIServer` představuje okenní aplikaci, která je grafickým rozhraním pro server.

3.4.1 DockPanelSuite – dockovací systém

Při představě systému bylo určeno, že bychom chtěli, aby s některými částmi naší aplikace bylo možné zacházet jako se samostatnými okny, ale také aby je bylo možné uchytit na jednom místě, takzvaně *dockovat*. K dosažení tohoto chování je v aplikaci použit nuget balíček `DockPanelSuite` [8], který poskytuje požadované chování a umožňuje pro dosažení lepšího vzhledu nastavit i téma [9].

Při implementaci oken, které později chceme dockovat je nutné, aby byly potomky okna `DockContent`.

3.4.2 GuiComponnents

Knihovna `GuiComponnents` obsahuje základní okenní formuláře, které jsou zapotřebí pro ovládání uživatelských aplikací. Přehled základní oken v implementaci a jejich účel:

- `Messenger` je formulář pro udržování a zasahování do textových dat. Toto okno je určeno pro dokování a konstruuje se s parametry:
  - `List<string> data`, data, která budou zobrazována.
  - `ActionOfButton act`, uživatelem definovaná funkce přijímající textový řetězec.

  Okno obsahuje textové pole pro data, kde se vypisují data z kolekce, textové pole pro zápis, kde je možné psát text, a tlačítko, které vymaže textové pole pro text a vykoná akci definovanou v konstruktoru. Jako parametr této akce je použita hodnota z textového pole pro psaní před vymazáním.

- `Data` je okenní formulář sloužící pro zobrazování dat z `Dictionary<K,T>` a konstruuje se s parametry:
  - `Dictionary<K, T> data` představuje data, která budou zobrazována.
  - `Line<K,T>.ActionOfButton act` je uživatelem definovaná funkce přijímající `K` a `T`.
  - `TextFormater t` funkce generující z `K` a `T` textový popis.

  Okno vykresluje pro jednotlivé dvojice ze slovníku `data` tlačítko provádějící uživatelskou akci a data v textové podobě.

- `StringSetterForm` je okenní formulář pro získání textového řetězce od uživatele.

- `AddNewEquipment` je okenní formulář sloužící k vytvoření nového zařízení v systému, případně nastavení přezdívky pro zařízení či složku zařízení.

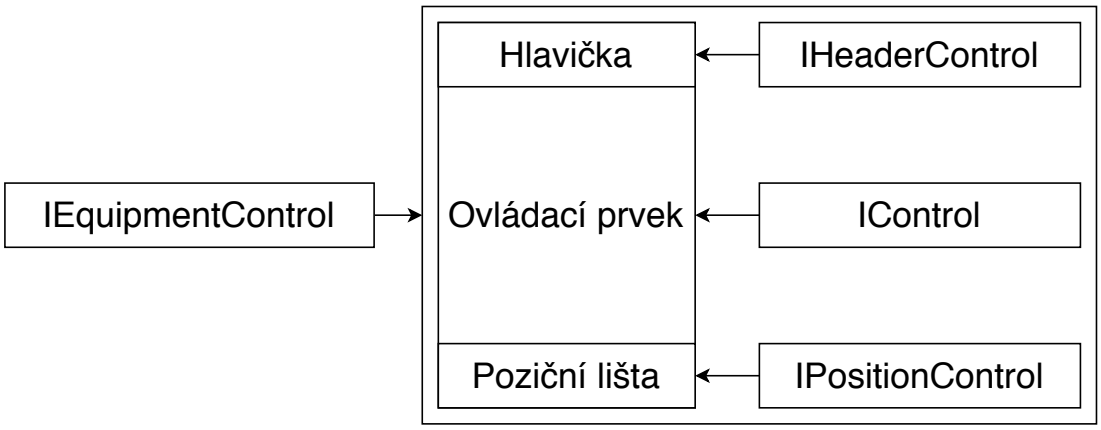
- `CreateNewContex` je okenní formulář umožnující vytvářet soubory, které se odkazuje na soubory s daty pro zařízení jako jsou přezdívky či skupiny a umožňuje tak uspořádávat si data do celku pro jednodušší načítaní dat.

- `FileManager` je nástavba nad existujícími prvky pro získání validních dat pro práci se soubory.

- `MenuFunctions` třída sloužící pro třídy Client a Server pro načítání dat ze souborů. Zobrazuje souborový vyhledávač a načítá potřebná data.

3.4.3 RealTimeControl

Knihovna `RealTimeControl` poskytuje programátorovi virtuální ovládací konzoli, nadále jen *konzole* a části, ze kterých je možné ji sestavit.



Obrázek 3.12: Spojení rozhraní s návrhem.

Rozhraní pro ovladače

Pro různé druhy zařízení je potřeba různé ovladače, které by měly plnit stejné funkce. Za tímto účelem jsou v knihovně nadefinována rozhraní, která nám poskytují, která představují minimální požadavky na tyto ovladače.

Jak můžeme vidět na obrázku 3.12, tato rozhraní v podstatě představují jednotlivé části výsledného ovladače, jak bylo požadováno v sekci 1.4.1 Kostra ovládacího prvku.

Hlavička – IHeaderControl

Rozhraní `IHeaderControl` představuje hlavičku ovladače tak, jak byla navrhnuta v sekci 1.4.1 Kostra ovládacího prvku a zobrazuje uživateli základní informace o zařízení, které ovladač ovládá, jako je:

- `Priority` – Důležitost, s jakou ovladač zařízení nastavuje.
- `NameLabel` – Jméno od uživatele.
- `TechnicalLabel` – Technické jméno.

V knihovně pak toto rozhraní implementuje `HeaderControl` a poskytuje funkce a metody, pomocí nichž je možné jeho položky nastavovat a udržovat.

Poziční lišta – IPositionControl

Rozhraní `IPositionControl` představuje poziční lištu sloužící k tomu, aby na ní pomocí tlačítek šlo provést základní operace jako jsou:

- `UserAction MoveRight` – Posunout ovladač doprava.
- `UserAction MoveLeft` – Posunout ovladač doleva.
- `UserAction SetMax` – Nastavit maximální hodnotu ovladače, případně jinou nadefinovanou.

- `UserAction SetMin` – Nastavit minimální hodnotu ovladače, případně jinou nadefinovanou.

V knihovně pak toto rozhraní implementuje `PositionControl`.

Ovládací prvky – IControl

Rozhraní `IControl` představuje ovládací prvky určené pro ovládání jednotlivých zařízení, případně alespoň jejich částí. Aby byla pokryta potřeba představených ovladačů v sekci 1.4.1 Ovládací prvek, nad rozhraní `IControl` jsou v knihovně implementovány ovládací prvky:

- `ColorControl` – Představující Volič barev.
- `ColorPickerWPF` – Představující Volič barev.
- `WpfJoystick` – Představující joystick.
- `WideSlider` – Představující fejdr.

WpfJoystick Pro nastavování hodnot os je v systému vytvořený ovladač `WpfJoystick`. Tento ovladač slouží k nastavování 2 hodnot, které prezentuje jako osy pohybu. Pro dosažení lepší představy ovládání je fejdr pro složku představující rotaci vůči ose světla kolmé k zemi zakřivená do půlkruhu. Při vytváření tohoto prvku jsem vycházel z blogu Charlese Petzolda [5]. Ovladač podporuje multitouch.

ColorControl a ColorPickerWPF Pro nastavování barev jsou v systému přítomny hned dva ovládací prvky. Oba dva slouží k nastavování RGB složek zařízení, hlavní rozdíl je v jejich prezentaci dat uživateli.

Ovládací prvek `ColorControl` je založen na prvku `ColorCanvas` z nugetu balíčku Extended.Wpf.Toolkit [13].

Ovládací prvek `ColorPickerWPF` je založen na prvku ColorPickerControl z nugetu ColorPickerWPF, [10].

WideSlider Pro nastavování hodnot z rozmezí je v systému ovládací prvek `WideSlider`. Ovladač je pouze upravený prvek slider a podporuje multitouch.

Vytváření ovladačů pro zařízení – RepresentStateFactory

V knihovně je implementovaná třída `RepresentStateFactory`, která slouží pro vytváření ovladačů pro jednotlivá zařízení. Při představě ovladačů konzole v sekci 1.4.1 Ovládací prvek bylo určeno, že pro jednotlivé zařízení nebudou vznikat ovladače vcelku, ale rozdělené na jednotlivé části.
Proto je třídě `RepresentStateFactory` předán stav zařízení a lišta na ovladače – `ControlDockPanel`, do které mají být ovladače pro tento stav vygenerovány. Třída pak podle druhu zařízení vygeneruje patřičné ovladače 1.4.1 Ovládací prvek.

- Barevná světla – `State<RGBLight>`
  Pro barevná světla je vygenerován ovladač `EquipmentControlColor`.

- Pohyblivá světla– `State<SpyderLight>`
  Pro pohyblivá světla jsou vygenerovány ovladače `EquipmentControlColor` pro barevné složky a `EquipmentControlAxis` pro pohyblivé složky.

- Stmívač – `State<DIMMLight>`
  Pro každou složku je vygenerován ovladač `EquipmentControlSlider`.

- Nedefinované zařízení – `IState`
  Pro každou složku je vygenerován ovladač `EquipmentControlSlider`.

Lišta na ovladače – ControlDockPanel

Třída `ControlDockPanel` je nástavba nad WPF prvkem `StackPanel`, která slouží k udržování velikosti ovladačů a jejich pozici.

Hlavní ovladače – MainLightControl

Třída `MainLightControl` obsahuje lištu na ovladače `ControlDockPanel` a prvky `GroupCreator` a `GroupBrowser` na udržování skupin.

Generování globálních ovladačů pro zařízení Ve třídě `MainLightControl` dochází k vytváření globálních ovladačů, které slouží k nastavování hodnot všem zařízením. Jelikož definování, co konkrétně má daný ovladač ovládat, je nejednoznačné, jsou zde pro každý ovladač funkce, které k tomu slouží.

Příklad nejednoznačnosti: Ovladač pro všechna světla
– `EquipmentControlSlider AllLights` má za úkol nastavovat hodnoty všem světlům v systému, ale nenastavuje hodnoty pro osy pohyblivých světel.

Ovladač pro všechna barevná světla
– `EquipmentControlColor AllColorControl` má za úkol nastavovat hodnoty všem barevným světlům v systému, ale také všem pohyblivým barevným světlům.

GroupCreator – vytváření nových skupin Třída `GroupCreator` slouží pro vytváření nových skupin a skupin složek zařízení. Po vyrobení skupiny je tato skupina přidána do `Aliases`.

GroupBrowser – Načítání skupin Třída `GroupBrowser` zobrazuje skupiny uložené ve třídě `Aliases` a umožňuje jejich přidávání do lišty s ovladači jednotlivých zařízení.

Celá konzole

Jelikož zbytek systému je vytvořený ve Windows Forms, bylo by žádoucí, aby i výsledná konzole byla jednoduše použitelná ve Windows Forms. Z toho důvodu je výsledná konzole vložena do Windows Form formuláře podporující dokování. Pro zkonstruování okna je potřeba zadat :

- Kolekci `Dictionary<IID,Istate> data`, nad kterou je celá konzole udržována.

- Třídu `Aliases aliases`, odkud jsou brány skupiny zařízení a pojmenování.

- Funkci `UserAction userChanged`, která má být provedena, pokud konzole změní data.

Rozšíření grafické prezentace – ovladač

V případě, že se v systému objeví nový druh zařízení, je jeho ovládání v základu realizováno tak, že pro každou jeho složku je vytvořen fejdrový ovladač. V případě, že je potřeba pro nový druh zařízení vytvořit specifický ovladač, případně změnit stávající, je pak potřeba nadefinovat ovládací prvek, který bude splňovat rozhraní `IControl`. Spojit ho s prvky splňující rozhraní `IHeaderControl` a `IPositionControl` do ovladače splňující rozhraní `IEquipmentControl` a ve třídě `RepresentStateFactory` upravit funkce, které slouží pro generování ovladačů. To znamená nadefinovat v nich, pro jaké druhy zařízení a jaké složky je určen.

Rozšíření grafické prezentace – globální ovladač

V případě vytváření nového globálního ovladače pro tento nový druh je potřeba ve třídě `MainLight`nadefinovat funkce pro vytvoření a udržování tohoto globálního ovladače a zapsat její volání do funkce `UpdateMainControls`, která je zavolána při změně používaných zařízení. V případě, že nový druh se částečně překrývá s jinými ovladači, je potřeba upravit ostatní funkce definující globální ovladače.

Rozšíření grafické prezentace – skupiny

Pro rozšíření vytváření skupin je potřeba rozšířit funkci `string GetLabel(int i, Type type)`, která podle typu zařízení a pořadí složky vygeneruje název této složky. Přidat nový druh do výčtového typu `SupportedGroups` a nadefinovat tlačítko, sloužící pro výběr druhu zařízení.

Pro rozšíření přidávání skupin je potřeba ve funkci `CreateGroup` nadefinovat, jakým ovladačem má být skupina prezentována.

3.4.4 GUIServer

Projekt `GUIServer` vytváří spustitelnou okenní aplikaci sloužící jako rozhraní pro ovládání serveru, dále tuto aplikaci budeme nazývat jen jako *serverová aplikace*.

Pro vytvoření třídy `Server` je potřeba, aby byla zadána data, jako jsou port, na kterém bude probíhat komunikace mezi serverem a klienty, port, na kterém bude probíhat komunikace mezi serverem a zařízeními a heslo, které musí zadat klienti pro přihlášení do systému. Z toho důvodu při spuštění serverové aplikace je potřeba tato data získat od uživatele. Za tímto účelem aplikace obsahuje formulář `StartSetting`. Tento formulář obsahuje kolonky na výše uvedené informace a tlačítko s nápisem *SET*, které po stisknutí zkonstruuje se zadanými parametry třídu `Server` a nad ní zkonstruuje třídu `ServerHome`.

Serverová aplikace

Okno `ServerHome` tvoří kostru serverové aplikace. Toto okno podporuje dockování, tak se nad serverovou třídou sestrojí a zadockuje podle uživatelského ná-

kresu:

- `chat` – Instanci třídy `Messenger`, která prezentuje data z klientské kolekce `List<string> Conversation` a jako akci si uloží zprávu do textové komunikace a odešle ji všem klientům.
- `errors` – Instanci třídy `Messenger`, která prezentuje data z klientské kolekce `List<string> Errors` a jako akci si uloží do této kolekce zprávu.
- `connected` – Instanci třídy `Data<string>`, zobrazuje všechny přihlášené klienty i s uživatelskými jmény a umožňuje je odebírat ze seznamu přihlášených klientů a tak je odhlašovat.
- `map` – Instanci třídy `Data<IProtocol>`, zobrazuje protokoly, které jsou použity pro komunikaci s jednotlivými zařízeními a umožňuje je odebírat a tak přerušit komunikaci s daným zařízením.
- `DataToLights` – Instanci třídy `EquipmentsData`, která slouží pro základní reprezentaci stavů zařízení a nastavování jejich složek. V případě, že je tato prezentace nedostačující, je zde možnost otevřít si virtuální ovládací konzoly.

EquipmentsData – základní prezentace dat

Okno `EquipmentsData` slouží pro zobrazování stavů jednotlivých zařízení. Stavy zařízení jsou prezentovány pomocí jsou třídy `EquipmentGUI`.
Třída `EquipmentGUI` zobrazuje základní data jako je identifikátor zařízení, důležitost s jakou je nastaven tento stav a jestli jsou jednotlivé složky nastaveny záměrně. Pro reprezentování složek je použita třída `CreateParts`, implementující rozhraní `ICreateParts`. Tato třída pro jednotlivé druhy zařízení generuje různou prezentaci, tak aby lépe odpovídala tomu, co složky představují. Pro rozšíření je potřeba upravit třídu `CreateParts`, případně napsat novou třídu implementující rozhraní `ICreateParts`.

Virtuální ovládací konzole Virtuální ovládací konzole je náročná na konstrukci, proto pokud uživatel o ni z tohoto formuláře požádá, nastaví se u konzole vlastní funkce, co se má stát při zavírání a to tak, že místo ukončení okna je jen nastavena viditelnost, tak aby nebylo vidět, a je omezeno posílání událostí tomuto oknu. Při dalších pokusech o otevření okna je pak jenom znovu zviditelněno.

Reakce na změny

Pro aktualizování vzhledu aplikace je použit časovač `UpdateTimeLayout`, který po určitém časovém intervalu zavolá všechny zaregistrované funkce a ty mohou aktualizovat vzhled aplikace. Volané funkce si pak na základě dat, případně na základě příznaků `CFO`, ze sekce 3.2.6, sami rozhodnou, jestli se mají překreslit.

3.4.5 GUIUser

Projekt `GUIUser` vytváří spustitelnou okenní aplikaci sloužící jako rozhraní nad klientskou třídou, dále tuto aplikaci budeme nazývat jen jako *uživatelská aplikace*.

Pro vytvoření klientské třídy je potřeba, aby byla zadána data, jako jsou port, na kterém bude komunikace probíhat, IP adresu serveru, své uživatelské jméno a v případě potřeby i heslo. Z toho důvodu při spuštění uživatelské aplikace je potřeba tato data získat od uživatele. Za tímto účelem aplikace obsahuje formulář `LoginForm`.

Uživatelská aplikace

Kostra uživatelská aplikace je tvořena třídou `UserHome`. Toto okno podporuje dokování a tak si nad klientskou třídou sestrojí a zadokuje podle uživatelského nákresu:

- `chat` – Instanci třídy `Messenger`, která prezentuje data z klientské kolekce `List<string> Conversation` a jako akci odešle serveru zprávu představující zprávu z textové komunikace.
- `errors` – Instanci třídy `Messenger`, která prezentuje data z klientské kolekce `List<string> Errors` a jako akci uloží do této kolekce zprávu.
- `RealTimeSetter` – Instanci třídy `RealTimeSetting`, která nad klientskou třídou vytvoří virtuální grafickou konzoli.

Reakce na změny

Pro aktualizování vzhledu aplikace je použit časovač `UpdateTimeLayout`, který po určitém časovém intervalu zavolá všechny zaregistrované funkce a ty mohou aktualizovat vzhled aplikace. Volané funkce si pak na základě dat, případně na základě příznaků `CFO`, ze sekce 3.2.6, sami rozhodnou jestli se mají překreslit.

4. Uživatelská dokumentace

Podmínky

Aplikace je spustitelná na operačním systému Windows. Vyzkoušena byla na Windows 7 a Windows 10 (verze 1903). Jelikož aplikace komunikují v rámci lokální sítě, musí být zařízení připojeno do sítě. Na jednom počítači může být spuštěna pouze jedna ze aplikace v jednom čase. Při nedodržení těchto podmínek není chování aplikace zaručeno.

Zprovoznění

Pro používání aplikace stačí získat balíček `Lights.zip` obsahující:

- Složka Server, ve které je uložena spustitelná aplikace `Server.exe`
- Složka User, ve které uložena spustitelná aplikace `User.exe`
- Složka `Data`, ve které jsou již uložena připravená základní data pro systém.

4.1 Základní použití aplikací

Před spuštěním aplikace mějme lokální ethernetovou síť, do které jsou zapojena zařízení, která chceme ovládat. Tato zařízení jsou ve stejné podsíti, komunikují na stejném portu a my známe jejich IP adresy a port, na kterém komunikují.

Před spuštěním aplikace je potřeba ověřit, že náš počítač je zapojen do lokální sítě a má nastavenou IP adresu do stejné podsítě jako mají zařízení. Při nedodržení těchto podmínek není chování aplikace zaručeno.

4.1.1 Spuštění aplikací

Prvním krokem pro zprovoznění aplikací je extrahovat si balíček `Lights.zip`. Po extrahování balíčku pak stačí spouštět z balíčku aplikaci `Server\Server.exe`. Pokud je potřeba více ovládacích konzolí, je možné na jiném počítači spustit aplikaci `User\User.exe` a s ní se připojit na server.

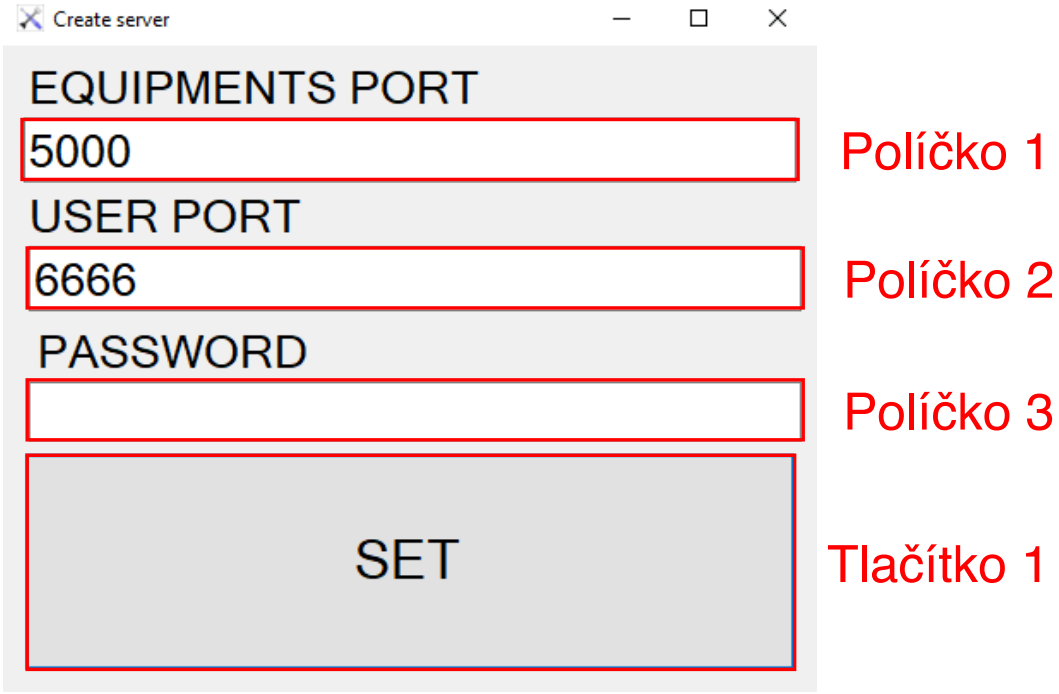
Pokud se aplikace odmítá spustit, případně se po spuštění ukončí, a jsou dodrženy výše uvedené podmínky, je to s největší pravděpodobností zapříčiněno antivirovou ochranou a je nutné si tuto aplikaci povolit.

4.1.2 Základní použití

Hlavní aplikací je aplikace `Server.exe`, která má sloužit jako hlavní ovládací pult, proto si na ní předvedeme spuštění celého systému a zavedení dat do něj. Aplikace `User.exe` poskytuje podobné rozhraní a hlavním rozdílem oproti aplikaci `Server.exe` je, že některé funkce nejsou dostupné. Více informací o použití aplikace `User.exe` je uvedeno v sekci 4.2.

4.1.3 Po spuštění aplikace Server.exe

Po spuštění aplikace `Server.exe` se zobrazí formulář *Create server* sloužící k nastavení dat pro server. Formulář můžeme vidět na obrázku 4.1.



Obrázek 4.1: Nastavení serveru.

Je potřeba vyplnit:

- Port pro komunikaci se zařízeními (políčko 1 z obrázku 4.1).
- Port pro komunikaci s klienty (políčko 2 z obrázku 4.1).
- Heslo, které budou uživatele muset zadat pro přihlášení do systému (políčko 3 z obrázku 4.1).

Přednastavené hodnoty odpovídají hodnotám pro divadlo „Ochotnický soubor Lípa“.

Po vyplnění hodnot je potřeba stisknout tlačítko *SET* (tlačítko 1 z obrázku 4.1). Po stisknutí tlačítka formulář zmizí a vytvoří se prázdný formulář Server.

4.1.4 Přidání zařízení do systému

Tato funkce není v Aplikaci `User.exe` podporována. Jak můžeme vidět na obrázku 4.2, pro zavedení nového zařízení do systému je potřeba:

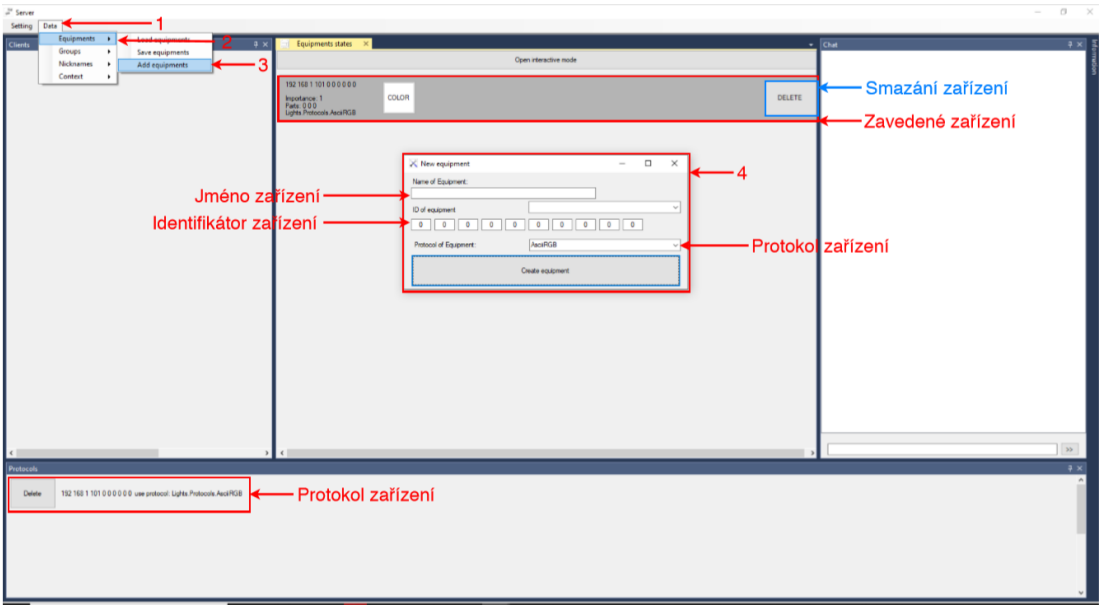
1. Otevřít kartu z menu *Data*.
2. Vybrat podkartu *Equipments*.
3. Vybrat podkartu *Add equipments*.

4. Po stisknutí *Add equipments* se zobrazí formulář pro přidávání zařízení *New equipment*.

Zobrazený formulář slouží pro zavádění nových zařízení a je v něm potřeba:

- Zvolit jméno zařízení, pod kterým bude vedeno v systému.
  – Povinné, musí být unikátní.
- Identifikátor zařízení, kde první čtyři hodnoty představují IP adresu a poslední dvě čísla adresu v analogové podsíti.
  – Povinná položka, musí být unikátní.
- Protokol zařízení, kterému zařízení rozumí.
  – Povinná položka, musí být vybrána ze seznamu.

Po vyplnění dat je potřeba stisknout tlačítko *Create equipment*. Pokud nejsou data vyplněna validně, jsou zahozena. Pokud jsou vyplněna validně, v systému vznikne nové zařízení.



Obrázek 4.2: Zavedení nového zařízení.

Zavedená zařízení

Zařízení v systému je možné sledovat ve formuláři *Create equipment*. V tomto formuláři je možné tato zařízení i smazat tlačítkem *DELETE*.

Protokoly pro zařízení

Ve formuláři *Protocols* jsou zobrazeny, komunikační protokoly, které jednotlivá zařízení používají.

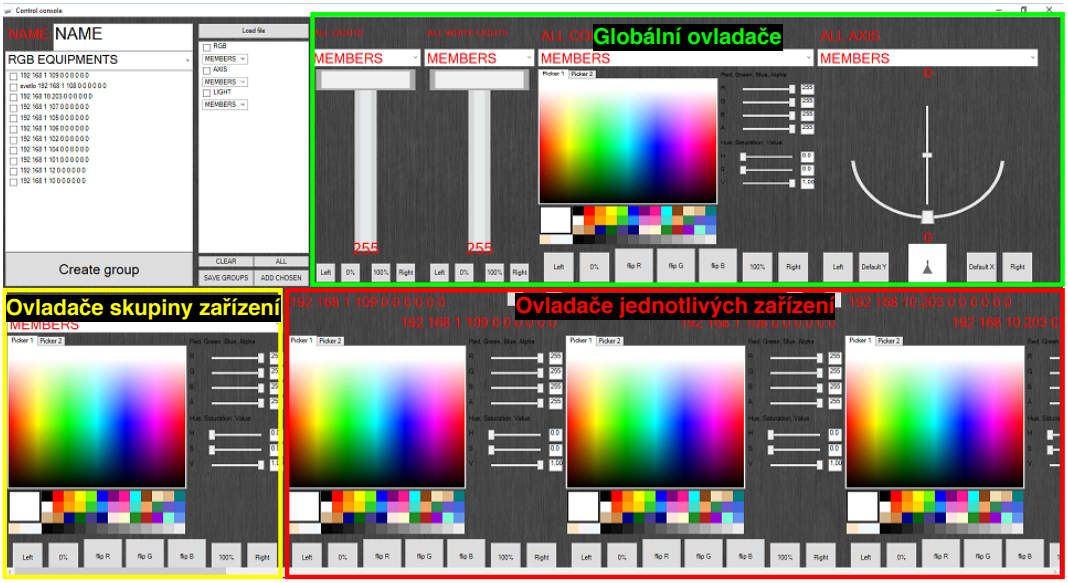
4.1.5 Virtuální grafická konzole

Pro ovládání světel je vhodné použít virtuální grafickou konzoli. Aplikace `User.exe` má zobrazenou virtuální grafickou konzoli hned po spuštění hlavního okna. V aplikace `Server.exe` stačí pro spuštění virtuální grafické konzole stisknout v hlavním okně tlačítko *Open interactive mode*.

Nastavování hodnot

Virtuální grafická konzole zobrazuje ovladače pro aktuální ovládaná zařízení Jak můžeme vidět na obrázku 4.3, ovladače jsou pak rozděleny na:

- Ovladače jednotlivých zařízení.
  Nastavují hodnoty jednomu zařízení či jeho části.
- Ovladače skupin zařízení.
  Nastavují hodnoty skupině zařízení definované uživatelem.
- Globální ovladače.
  Nastavují hodnoty všem zařízením určitého druhu.



Obrázek 4.3: Základní rozdělení ovladačů.

Použití ovladačů k nastavování hodnot by mělo být intuitivní. Více o ovladačích nalezneme v kapitole 4.6 Virtuální ovládací konzole

Skupiny zařízení

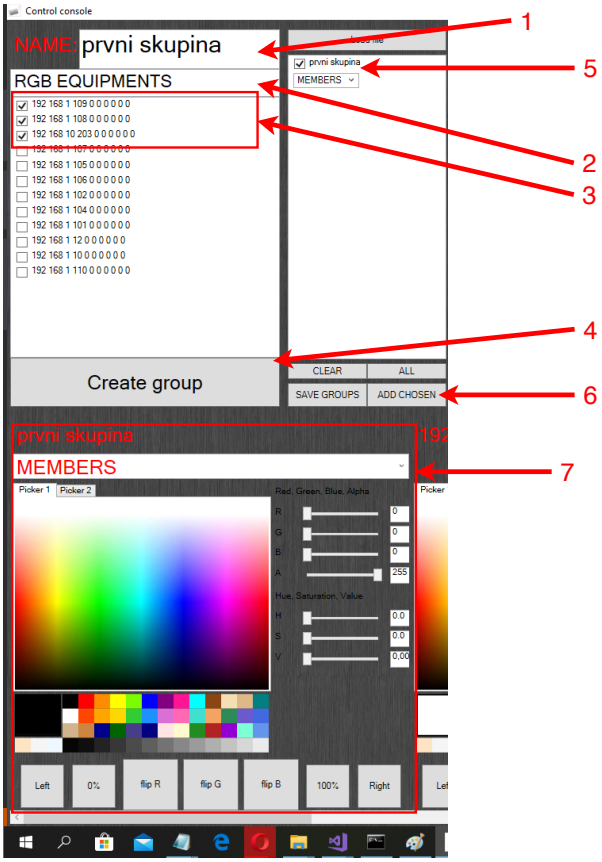
Virtuální ovládací konzole také slouží k vytváření a používání skupin zařízení.
Jak můžeme vidět na obrázku 4.4, pro vytvoření skupiny je zapotřebí provést následující kroky:

1. Zadat jméno vytvářené skupiny.

2. Vybrat ze seznamu druh skupiny. Druh skupiny určuje možné členy skupiny a jakým ovladačem bude skupina ovládána.
3. Označit členy skupiny.
4. Tlačítkem potvrdit vytvoření skupiny.

Skupina je vytvořena a zavedena do systému. Pro její použití je potřeba:

5. Označit skupiny, které chceme použít.
6. Tlačítkem potvrdit přidání skupin mezi ovladače.
7. Nově vzniklý ovladač skupiny.



Obrázek 4.4: Vytvoření a použití skupiny zařízení.

Více o vytváření skupin zařízení se můžeme dozvědět v sekcích 4.6.3 Vytváření skupin zařízení a 4.6.4 Ovladač na načítání skupin zařízení.

4.1.6 Ukládání a načítání dat

Veškerá data zavedená do systému je možné uložit a opět načíst do systému. V systému se dají uložit jednotlivé druhy dat:

- Zařízení s protokoly.

- Skupiny zařízení a skupiny složek zařízení.
- Jména jednotlivých zařízení či složek zařízení.

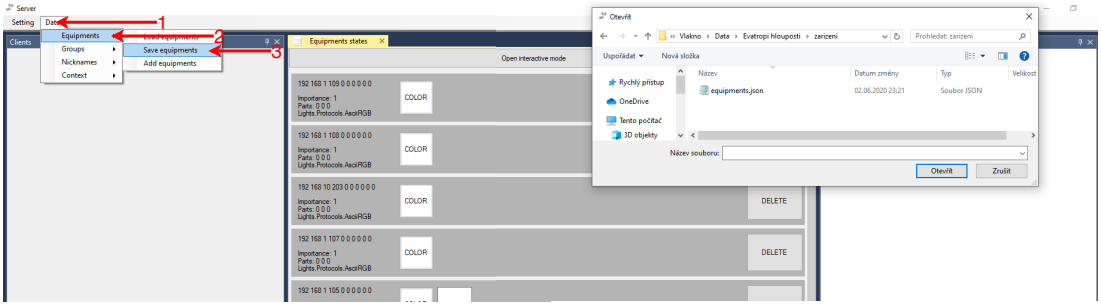
Postup ukládání a načítání dat je pro všechny druhy stejný, proto si ho předvedeme pouze na zařízeních.

Uložení dat

1. Otevřít kartu z menu *Data*.
2. Vybrat podkartu *Equipments*.
3. Vybrat podkartu *Save equipments*.
4. Po stisknutí *Save equipments* se zobrazí souborový prohlížeč.

V souborovém prohlížeči pak stačí vybrat soubor, do kterého mají být data uložena a potvrdit ho.

Ukládání dat si můžeme prohlédnout na obrázku 4.5.



Obrázek 4.5: Ukládání dat.

Načtení dat

1. Otevřít kartu z menu *Data*.
2. Vybrat podkartu *Equipments*.
3. Vybrat podkartu *Load equipments*.
4. Po stisknutí *Load equipments* se zobrazí souborový prohlížeč.

V souborovém prohlížeči pak stačí vybrat soubor, ze kterého mají být data načtena a potvrdit ho. Pokud soubor neobsahuje data, která mají být načtena, nic nenačte. Načítání dat je velmi podobné ukládání dat z obrázku 4.5.

Omezení aplikace User.exe

Aplikace User.exe nemůže ukládat data o zařízeních, protože pro ně nezná protokoly. Aplikace User.exe může načítat data o zařízeních, ale protokoly jsou ignorovány.

Úprava uložených dat

Data je možné modifikovat v souborech tak, jak jsou uložena. Pokud k tomu dojde a data nejsou nastavena validně, aplikace je může odmítnout. Nejedná se o očekávaný způsob použití.

Více souborová data

Pro načítání více dat najednou jsou v systému data *Context*. Více informací o nich se nachází v sekci 4.5.6 Context.

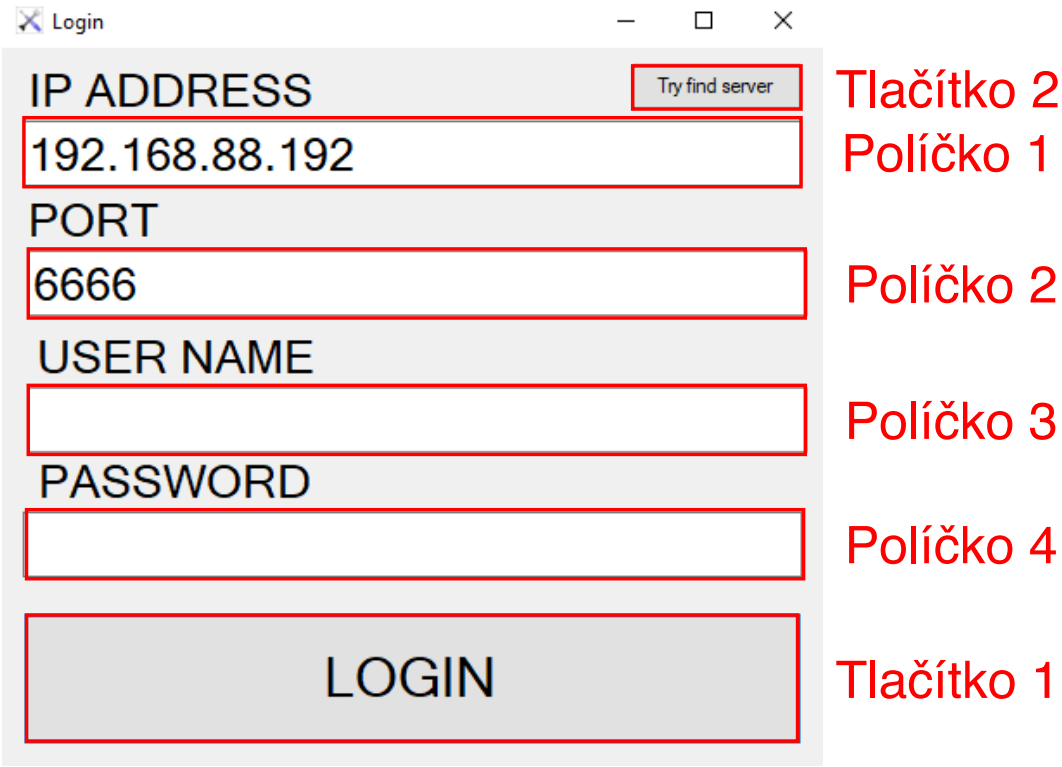
4.2 Popis aplikace User.exe

Po spuštění aplikace se zobrazí okno z obrázku 4.6, které slouží pro přihlášení do systému. Pro připojení do existujícího systému je potřeba vyplnit:

- IP adresu serveru (políčko 1 z obrázku 4.6).
- Port pro komunikaci se serverem (políčko 2 z obrázku 4.6).
- Zvolit si uživatelské jméno, pod kterým bude klient veden v systému (políčko 3 z obrázku 4.6).
- Heslo, které server vyžaduje pro přihlášení do systému (políčko 4 z obrázku 4.6).

Okno poskytuje možnost nad zadaným portem (v políčku 2 z obrázku 4.6) vyhledat v lokální síti IP adresu serveru. Pro pokus o vyhledání serveru je potřeba zmáčknout tlačítko *Try find server* (tlačítko 2 z obrázku 4.6). Po stisknutí tlačítka se aplikace po určitý krátký čas snaží získat IP adresu serveru. V případě, že v síti se vyskytuje server a odpoví na představující zprávu, je jeho IP adresa vyplněna do políčka pro IP adresu serveru (políčko 1 z obrázku 4.6). V případě, že se v lokální síti nevyskytuje server, obsah políčka pro IP adresu serveru (políčko 1 z obrázku 4.6) se nezmění a vyskočí oznámení o nenalezení serveru.

Po vyplnění všech potřebných údajů je nutné stisknout tlačítko *LOGIN* (tlačítko 1 z obrázku 4.6). V případě, že nejsou všechna data zapsána správně, je uživateli poskytnuta informace, která data nejsou zadána správně. V případě, že se uživatel snaží přihlásit na IP adresu, která není v daném kontextu validní, je mu tato informace sdělena.



Obrázek 4.6: Přihlášení k serveru.

Pokud je přihlášení na server úspěšné, spustí se hlavní okno aplikace. Pokud není přihlášení na server úspěšné, uživatel má možnost spustit si aplikaci bez připojení na server, jinak se aplikace ukončí.

Hlavní okno uživatelské aplikace.

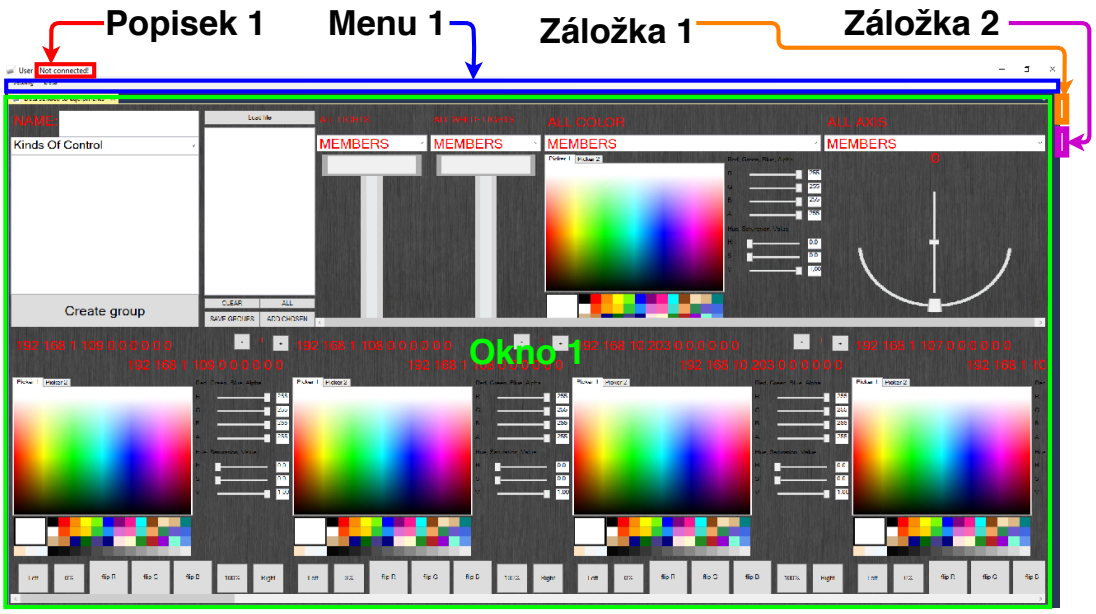
V hlavním okně uživatelské aplikace, které je zobrazeno na obrázku 4.7, se nachází zadockovaná virtuální grafická konzole. Použití virtuální grafické konzole je popsáno v části 4.6 Virtuální ovládací konzole.

Na pravé straně uživatelské aplikace jsou v liště umístěny dvě záložky. První záložka (záložka 1 z obrázku 4.7) představuje okno sloužící pro komunikaci uživatelů v rámci systému. Druhá záložka (záložka 2 z obrázku 4.7) představuje okno sloužící pro zobrazení informací o průběhu funkčnosti systému.

Použití těchto komunikačních oken nalezneme v sekci 4.4.

V horní části aplikace se nachází menu (menu 1 z obrázku 4.7), které slouží hlavně pro načítání a ukládání dat. Použití menu nalezneme v sekci 4.5.

V hlavní liště aplikace se vedle názvu okna nachází popisek (popisek 1 z obrázku 4.7), který signalizuje, jestli je aplikace přihlášena k serveru či ne.



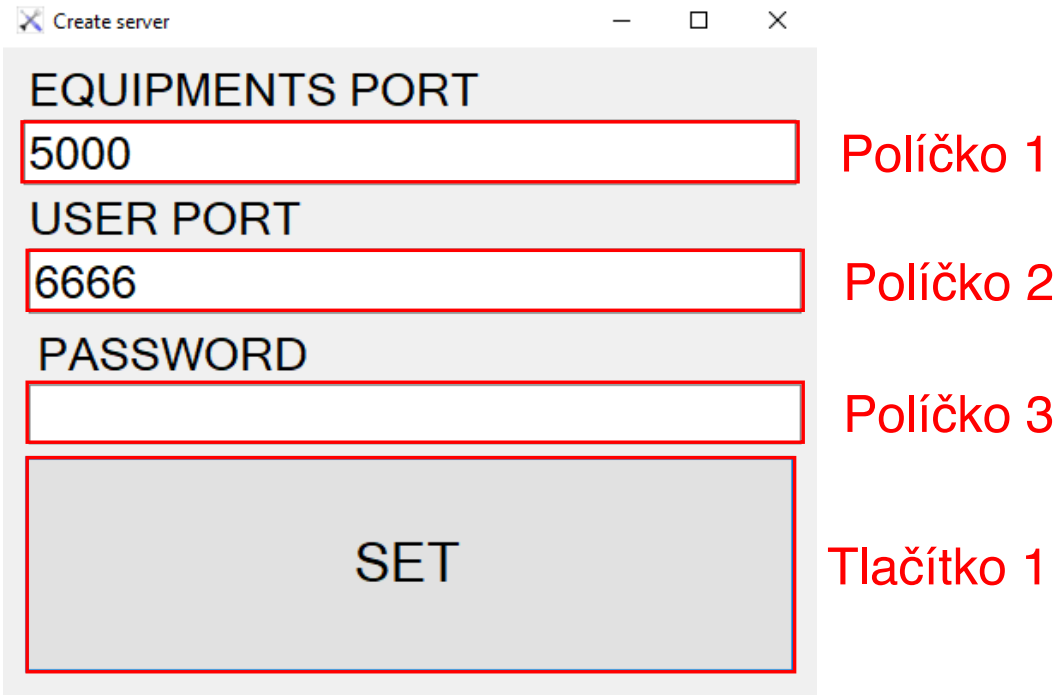
Obrázek 4.7: Hlavní okno uživatelské aplikace.

4.3 Popis aplikace Server.exe

Po spuštění aplikace se zobrazí okno z obrázku 4.8, které slouží pro nastavení základních dat serveru. Pro spuštění serveru je potřeba vyplnit:

- Port pro komunikaci se zařízeními (políčko 1 z obrázku 4.8).
- Port pro komunikaci s klienty (políčko 2 z obrázku 4.8).
- Heslo, které budou uživatele muset zadat pro přihlášení do systému (políčko 3 z obrázku 4.8).

Po vyplnění všech potřebných údajů je nutné stisknout tlačítko *SET* (tlačítko 1 z obrázku 4.8). Po stisknutí tlačítka se spustí server a otevře hlavní okno aplikace.



Obrázek 4.8: Nastavení serveru.

Hlavní okno serverové aplikace.

V hlavním okně serverové aplikace, které je zobrazeno na obrázku 4.13, se nachází:

Na pravé straně serverové aplikace je v liště umístěna záložka (záložka 1 z obrázku 4.13), představující okno, které slouží pro zobrazení informací o průběhu funkčnosti systému.

Vedle této záložky se nachází okno(okno 1 z obrázku 4.13), sloužící pro komunikaci uživatelů v rámci systému. Použití těchto komunikačních oken nalezneme v sekci 4.4.

V horní části aplikace se nachází menu (menu 1 z obrázku 4.13), které slouží pro načítání a ukládání dat. Použití menu nalezneme v sekci 4.5.

4.3.1 Klienti

V levé části hlavního okna se nachází okno (okno 2 z obrázku 4.13), které zobrazuije přihlášené klienty a jejich uživatelská jména. Klienty je možné odhlásit pomocí tlačítka *DELETE*.

4.3.2 Protokoly

V dolní části hlavního okna se nachází okno (okno 3 z obrázku 4.13), které zobrazuje jaké protokoly používají jednotlivá zařízení. Protokoly je možné smazat od zařízení pomocí tlačítka *DELETE*.

4.3.3 Stavy zařízení

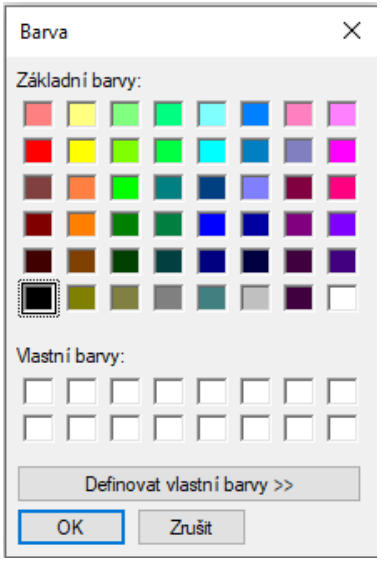
Uprostřed hlavního okna se nachází okno (okno 4 z obrázku 4.13), které zobrazuje stavy jednotlivých zařízení. V okně se stavy zařízení se nachází tlačítko *Open interactive mode* (tlačítko 1 z obrázku 4.13), které otevře okno s virtuální ovládací konzolí. Použití virtuální grafické konzole je popsáno v sekci 4.6 Virtuální ovládací konzole.

Jak můžeme vidět na obrázku 4.12, stav zařízení obsahuje informace:

- Identifikaci zařízení (textové pole 1 z obrázku 4.12).
- Důležitost, s jakou bylo zařízení nastaveno (textové pole 2 z obrázku 4.12).
- Používaný protokol pro komunikaci se zařízením (textové pole 3 z obrázku 4.12).

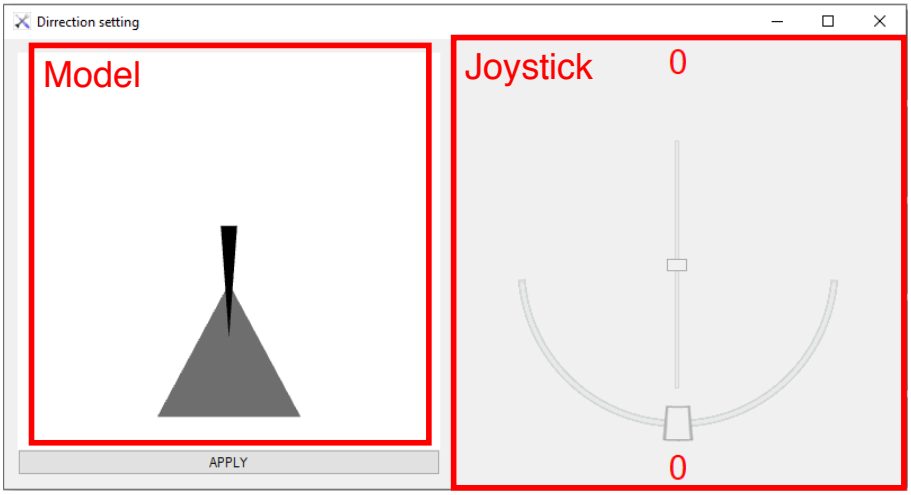
Dále pak stav zařízení zobrazuje hodnoty složek zařízení. Složky jsou pak prezentovány podle druhu zařízení a toho, co znázorňují.

Barevná světla Pro barevná světla či světla s barevnou částí je nastavená hodnota prezentována tlačítkem (tlačítko 2 z obrázku 4.12), které zobrazuje aktuální barvu. Pro změnu této hodnoty je možné kliknutím na tlačítko (tlačítko 2 z obrázku 4.12) otevřít okno pro výběr barvy. Jak můžeme vidět na obrázku 4.9, v okně lze kliknutím vybrat barvu a potvrdit ji tlačítkem *OK*.



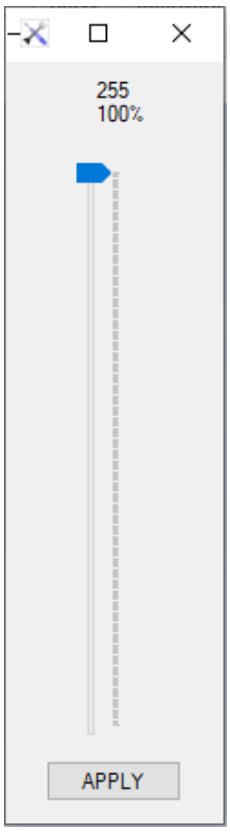
Obrázek 4.9: Nastavení barvy.

Pohyblivá světla Pro pohyblivá světla, či světla s pohyblivou částí je nastavená hodnota prezentována tlačítkem (tlačítko 3 z obrázku 4.12), které zobrazuje model světla ukazující směr zařízení. Pro změnu této hodnoty je možné kliknutím na tlačítko (tlačítko 3 z obrázku 4.12) otevřít okno pro nastavení směru zařízení. Jak můžeme vidět na obrázku 4.10, v okně lze pomocí joysticku (joystick z obrázku 4.10) nastavit směr zařízení, který je vidět na modulu (model z obrázku 4.10) a potvrdit ho tlačítkem *APPLY*.



Obrázek 4.10: Nastavení směru zařízení.

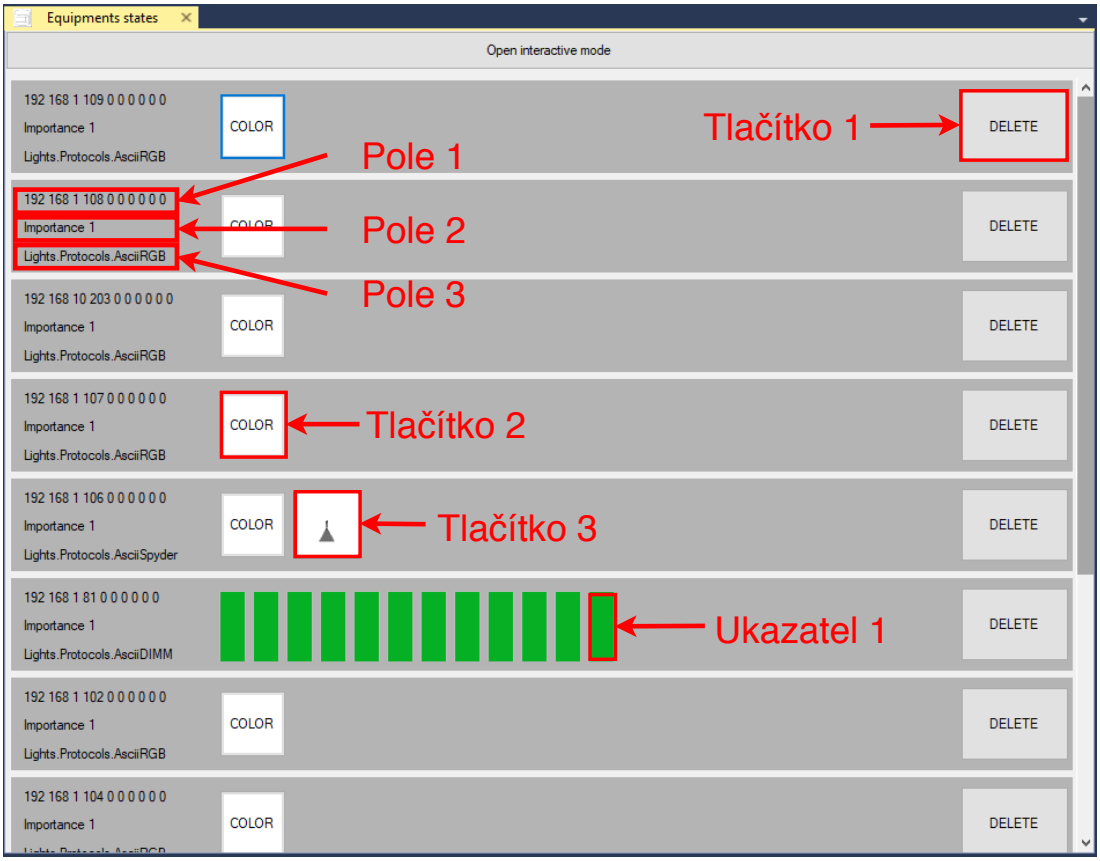
Stmívače Pro stmívače a světla či částí světel, u kterých složky nereprezentují celek, je nastavená hodnota prezentována ukazatelem stavu (ukazatel 1 z obrázku 4.12), zobrazující hodnotu složky. Pro změnu této hodnoty je možné kliknutím na ukazatel (ukazatel 1 z obrázku 4.12) otevřít okno pro nastavení hodnoty. Jak můžeme vidět na obrázku 4.11, v okně lze pomocí fejderu nastavit hodnotu zařízení a potvrdit ho tlačítkem *APPLY*.



Obrázek 4.11: Nastavení hodnoty složky.

Každý stav zařízení obsahuje tlačítko *DELETE* (tlačítko 1 z obrázku 4.12),

které k slouží k odstranění tohoto zařízení. Po odstranění již nejsou tomuto zařízení posílána data.

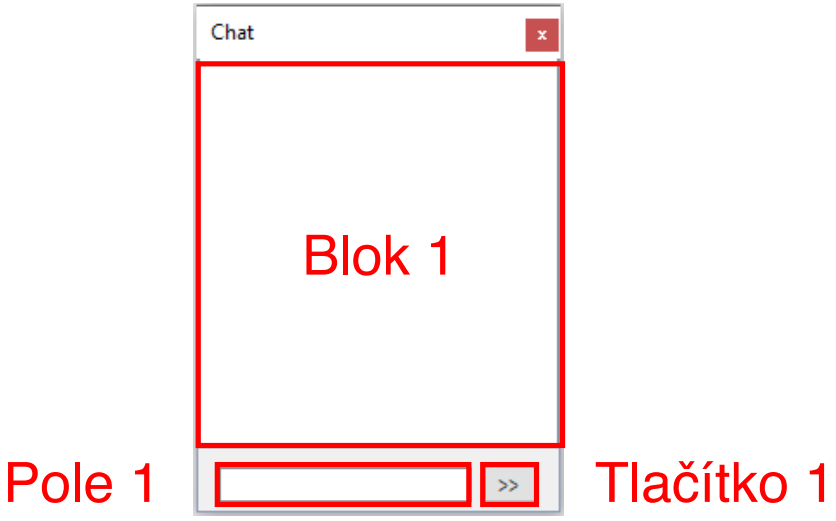


Obrázek 4.12: Stav zařízení.

Obrázek 4.13: Hlavní okno serverové aplikace.

4.4 Komunikační okna

Komunikační okna, jak můžeme vidět na obrázku 4.14, se skládají z textového bloku (blok 1 z obrázku 4.14), kde je vypisován text komunikace, textového pole (pole 1 z obrázku 4.14), které slouží pro záznam textové zprávy a tlačítka » (tlačítko 1 z obrázku 4.14), které slouží ke zpracování textu v textovém poli. Místo tlačítka lze použít i klávesu *ENTER*. V rámci systému není podporována diakritika, proto je vhodné ji nepoužívat.



Obrázek 4.14: Komunikační okno.

4.5 Menu

Menu je rozdělené na dvě karty:

- Karta *Setting*, kterou otevřenou můžeme vidět na obrázku 4.15.
- Karta *Data*, kterou otevřenou můžeme vidět na obrázku 4.16.

V případě, že karta je zašedlá, znamená to, že tuto možnost nejde v současném systému zvolit.

4.5.1 Setting

Karta *Setting* obsahuje podkarty sloužící k zobrazování a udržování dat spojených s komunikací v rámci systému.

Obrázek 4.15: Otevřená karta Setting.

Information

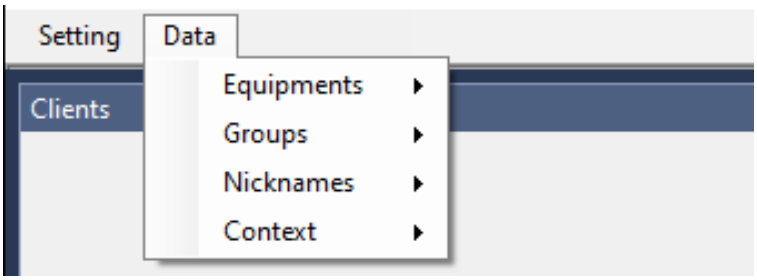
Po kliknutí na podkartu *Information* (podkarta 1 z obrázku 4.15) se zobrazí vyskakovací okno obsahující základní data spojená s komunikací v systému, jako je například port, na kterém komunikace probíhá, případně vlastní IP adresa.

Change password

Po kliknutí na podkartu *Change password* (podkarta 2 z obrázku 4.15) se zobrazí vyskakovací okénko sloužící pro zadání hesla. V případě aplikace `Server.exe` se změní heslo, které musí zadat klienti snažící se připojit na server. V případě aplikace `User.exe` se změní heslo, které aplikace používá při připojování server a pokusí se s tímto heslem připojit.

4.5.2 Data

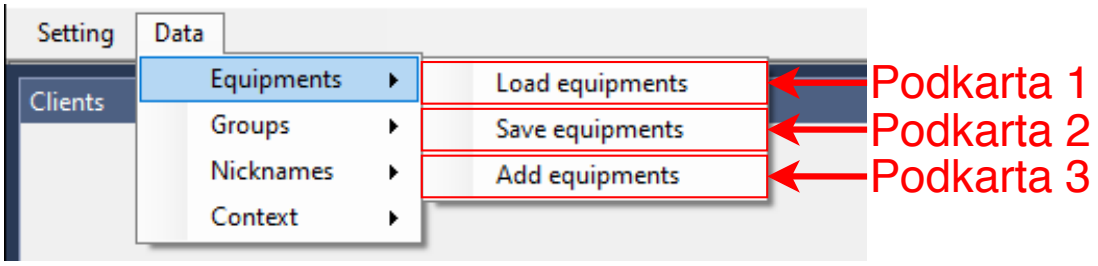
Karta *Data* obsahuje podkarty sloužící k načítání a ukládání dat sloužících k práci se zařízeními.



Obrázek 4.16: Otevřená karta Data.

4.5.3 Equipments

Podkarta *Equipments*, jak můžeme vidět na obrázku 4.17, obsahuje podkarty sloužící pro načítání a ukládání dat o zařízeních.



Obrázek 4.17: Otevřená podkarta Equipments.

Load equipments

Load equipments (podkarta 1 z obrázku 4.17). Slouží pro načtení dat o zařízeních. V případě aplikace `Server.exe` načte i protokoly pro komunikaci.

Save equipments

Save equipments (podkarta 2 z obrázku 4.17). Slouží pro uložení aktuálních dat o zařízeních. Dostupná pouze v aplikaci `Server.exe`.

Add equipments

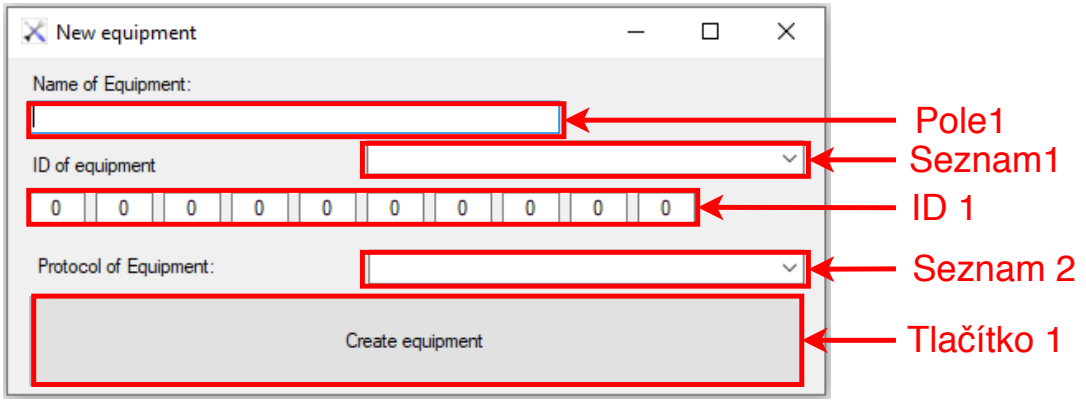
Add equipments (podkarta 3 z obrázku 4.17), otevře okno z obrázku 4.18, které slouží pro zavedení nového zařízení do systému. Funkce dostupná pouze v aplikaci `Server.exe`.

Zavedení nového zařízení. Pro zavedení nového zařízení do systému je potřeba validně vyplnit formulář z obrázku 4.18.

1. Vyplnit unikátní jméno zařízení (pole 1 z obrázku 4.18).
2. Vybrat protokol, kterým zařízení komunikuje (seznam 2 z obrázku 4.18).
3. Vyplnit unikátní identifikátor zařízení (ID 1 z obrázku 4.18).

Po vyplnění hodnot je potřeba stisknout tlačítko *Create equipment* (tlačítko 1 z obrázku 4.18). Pokud byla data vyplněna validně, je zavedeno do systému nové zařízení. Pokud data nejsou data vyplněna validně, je zobrazena chybová hláška.

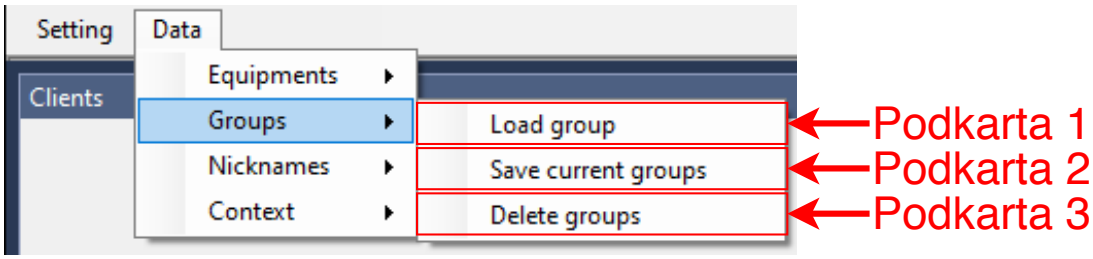
Formulář poskytuje seznam (seznam 1 z obrázku 4.18), který nabízí již dříve vyplněné identifikátory pro usnadnění práce.



Obrázek 4.18: Přidání nového zařízení.

4.5.4 Groups

Podkarta *Groups*, jak můžeme vidět na obrázku 4.19, obsahuje podkarty sloužící pro načítání a ukládání dat o skupinách zařízení.



Obrázek 4.19: Otevřená podkarta Groups.

Load groups

Load groups (podkarta 1 z obrázku 4.19) slouží pro načtení skupin zařízení.

Save groups

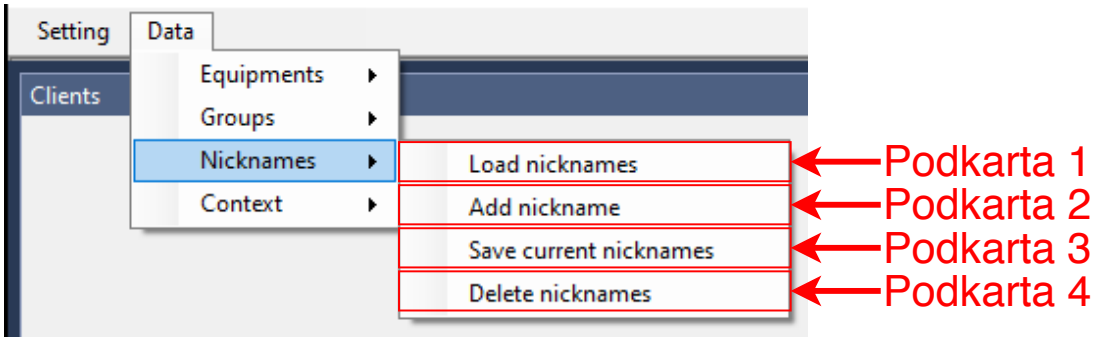
Save groups (podkarta 2 z obrázku 4.19) slouží pro uložení aktuálních dat skupin.

Delete groups

Delete groups (podkarta 3 z obrázku 4.19) otevře okno sloužící k mazání skupin.

4.5.5 Nicknames

Podkarta *Nicknames*, jak můžeme vidět na obrázku 4.20, obsahuje podkarty sloužící pro načítání a ukládání jmen zařízení.



Obrázek 4.20: Otevřená podkarta Nicknames.

Load nicknames

Load nicknames (podkarta 1 z obrázku 4.20) slouží pro načtení dat o zařízeních.

Save nicknames

Save nicknames (podkarta 2 z obrázku 4.20) slouží pro uložení aktuálních přezdívek zařízení.

Add nicknames

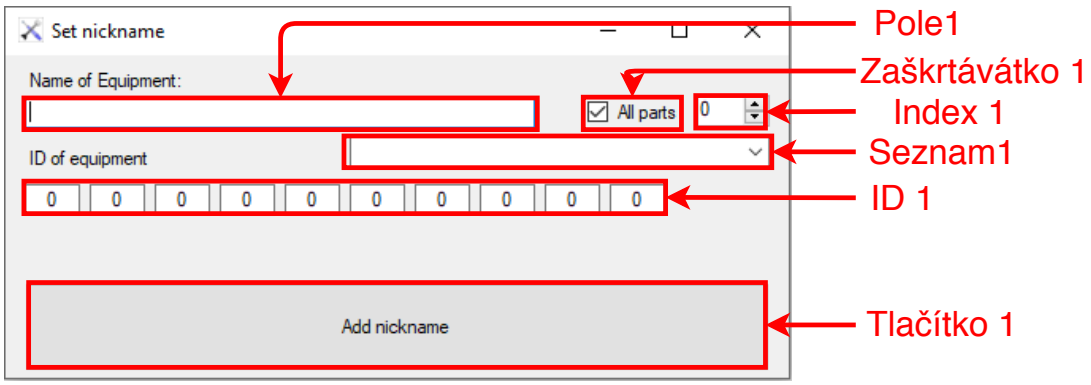
Add nicknames (podkarta 3 z obrázku 4.20) otevře okno z obrázku 4.21, které slouží pro zavedení nových jmen zařízení do systému.

Zavedení nového jména pro zařízení. Pro zavedení nového jména pro zařízení do systému je potřeba validně vyplnit formulář z obrázku 4.21.

- Vyplnit unikátní jméno (pole 1 z obrázku 4.21).
- Nastavit, jestli se jméno vztahuje na celé zařízení či na jednu složku (zaškrtávátko 1 z obrázku 4.21).
- Pokud se jméno vztahuje pouze na jednu složku, je potřeba uvést její index (index 1 z obrázku 4.21).
- Vyplnit unikátní identifikátor zařízení (ID 1 z obrázku 4.21).

Po vyplnění hodnot je potřeba stisknout tlačítko *Add nickname* (tlačítko 1 z obrázku 4.21). Pokud byla data vyplněna validně, je zavedeno do systému nové jméno pro zařízení. Pokud data nejsou vyplněna validně, je zobrazena chybová hláška.

Formulář poskytuje seznam (seznam 1 z obrázku 4.21), který nabízí již dříve vyplněné identifikátory pro usnadnění práce.



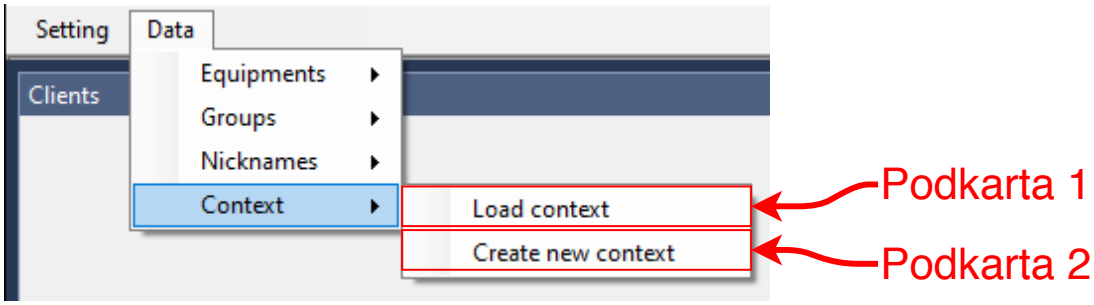
Obrázek 4.21: Přidání nového jména pro zařízení.

Delete nicknames

Delete nicknames (podkarta 4 z obrázku 4.20) otevře okno sloužící k mazání jmen zařízení.

4.5.6 Context

Podkarta *Context*, jak můžeme vidět na obrázku 4.22, obsahuje podkarty sloužící pro načítání a vytváření souborů, obsahující kontextová data. Tyto soubory obsahují názvy souborů s daty jako jsou zařízení, jména zařízení a skupiny zařízení.



Obrázek 4.22: Otevřená podkarta Context.

Load context

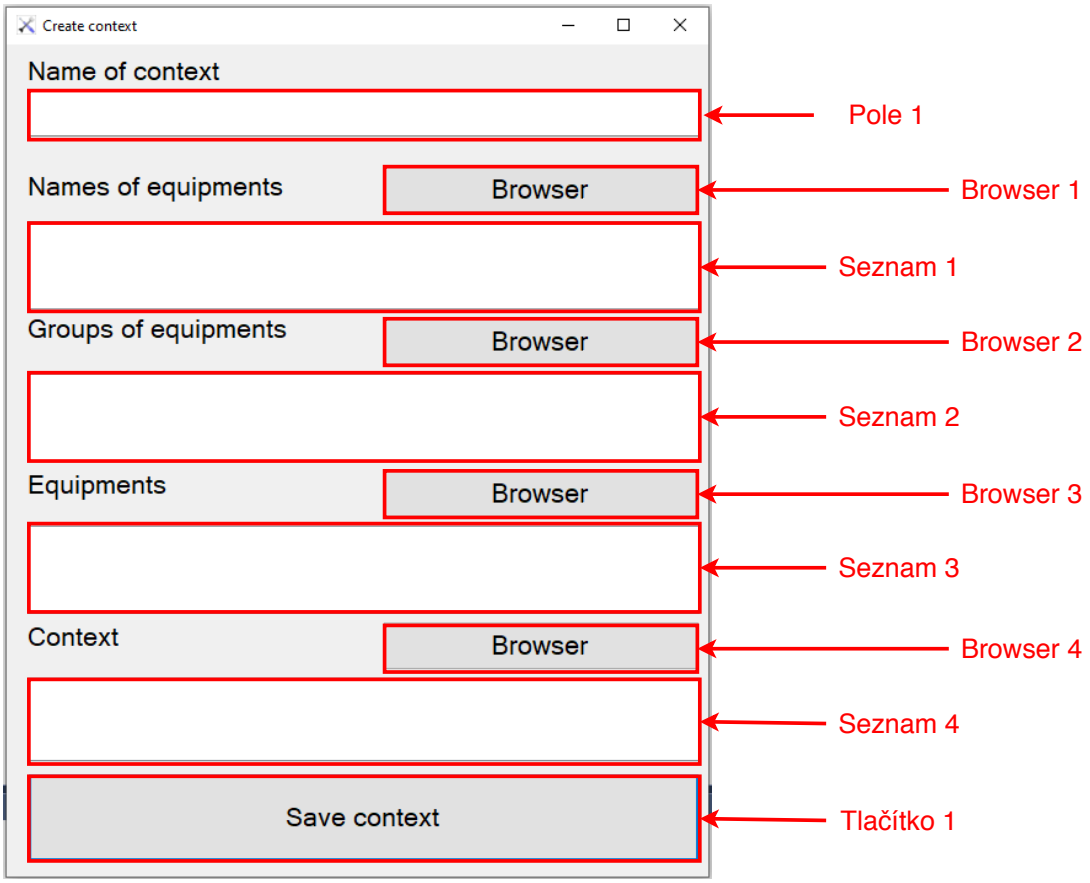
Load context (podkarta 1 z obrázku 4.22) slouží pro načtení souboru, který obsahuje názvy souborů, které má načíst. Soubory jsou pak podle druhu dat načteny.

Create new context

Create new context (podkarta 2 z obrázku 4.22) otevře okno z obrázku 4.23, které slouží k vytvoření souboru s názvy souborů.

Pro vytvoření kontextového souboru je potřeba:

- Zvolit jméno kontextu (pole 1 z obrázku 4.23).
- Načíst jména souborů se zařízeními (browser 1 z obrázku 4.23) a v seznamu (seznam 1 z obrázku 4.23) vybrat ty, které budou použity.
- Načíst jména souborů se skupinami zařízení(browser 2 z obrázku 4.23) a v seznamu (seznam 2 z obrázku 4.23) vybrat ty, které budou použity.
- Načíst jména souborů se jmény zařízení (browser 3 z obrázku 4.23) a v seznamu (seznam 3 z obrázku 4.23) vybrat ty, které budou použity.
- Načíst jiný kontextový soubor (browser 4 z obrázku 4.23), jeho soubory se zapíšou do ostatních seznamů a v seznamu (seznam 4 z obrázku 4.23) se objeví jeho kontextové jméno.



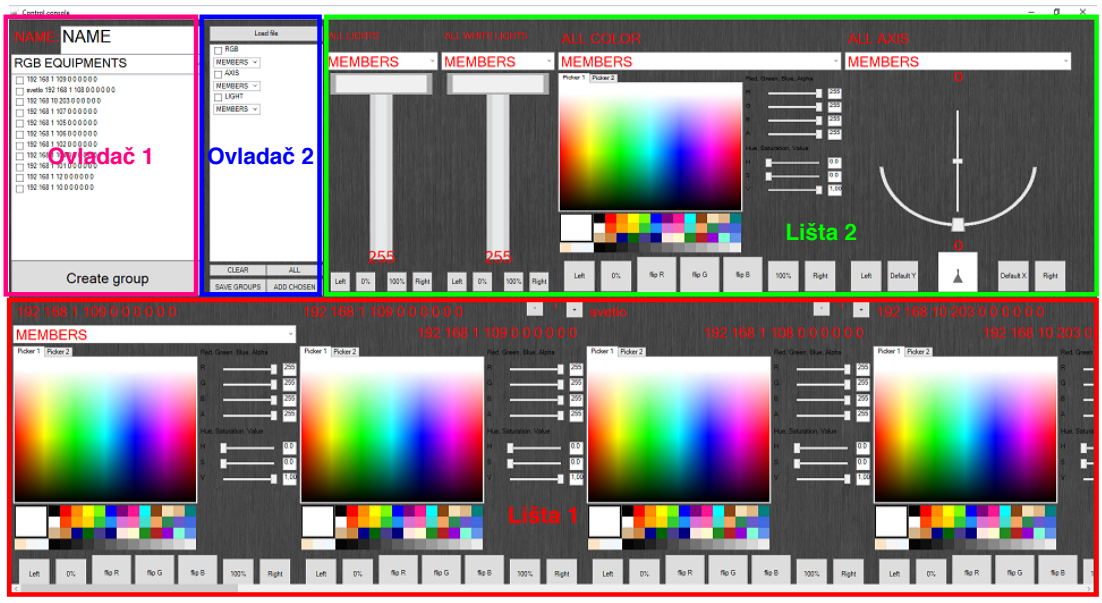
Obrázek 4.23: Vytvoření nového contextu.

Vygenerované soubory obsahují absolutní cestu, proto je nelze jednoduše přenášet.

4.6 Virtuální ovládací konzole

Virtuální ovládací konzole slouží pro ovládání zařízení. Jak můžeme vidět na obrázku 4.24, virtuální ovládací konzole se skládá z:

- Lišta ovladačů (lišta 1 z obrázku 4.24)
  Ovladače sloužící pro ovládání zařízení.
- Lišta globálních ovladačů (lišta 2 z obrázku 4.24)
  Ovladače sloužící pro ovládání všech zařízení určitého druhu.
- Ovladač na vytváření skupin zařízení(ovladač 1 z obrázku 4.24)
- Ovladač na načítání skupin zařízení (ovladač 2 z obrázku 4.24)
  Ovladač sloužící na načítání a ukládání skupin ze souborů a pro jejich přidávání na lištu ovladačů (lišta 1 z obrázku 4.24).



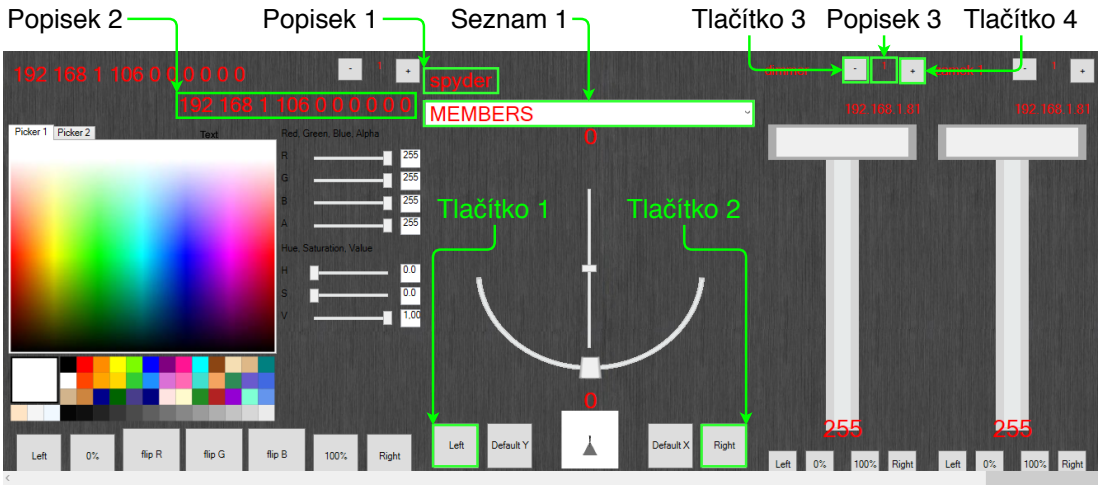
Obrázek 4.24: Virtuální ovládací konzole.

4.6.1 Lišta ovladačů

Lišta ovladačů obsahuje ovladače sloužící pro nastavování hodnot zařízením. Všechny ovladače pak obsahují:

- Tlačítko *Right* (tlačítko 1 z obrázku 4.25), které slouží k tomu, aby ovladač, pokud je to možné, byl posunut doprava.
- Tlačítko *Left* (tlačítko 2 z obrázku 4.25), které slouží k tomu, aby ovladač, pokud je to možné, byl posunut doleva.
- Jmenný popisek (popisek 1 z obrázku 4.25), který uvádí jméno zařízení, případně skupiny, je-li k dispozici.
- Technický popisek (popisek 2 z obrázku 4.25), který uvádí identifikaci zařízení nebo seznam zařízení (seznam 1 z obrázku 4.25), které jsou tímto ovladačem ovládané.
- Popisek důležitosti (popisek 3 z obrázku 4.25), který uvádí důležitost, s jakou je zařízení nastavované.

- Tlačítka + a - (tlačítka 3 a 4 z obrázku 4.25), které slouží k nastavení hodnoty důležitosti.



Obrázek 4.25: Ovladače.

Volič barvy

Pro výběr barvy slouží volič barvy, který je zobrazen na obrázku 4.26.

Tlačítko *0%* (tlačítko 1 z obrázku 4.26) nastaví všem složkám představující barvy hodnotu 0.

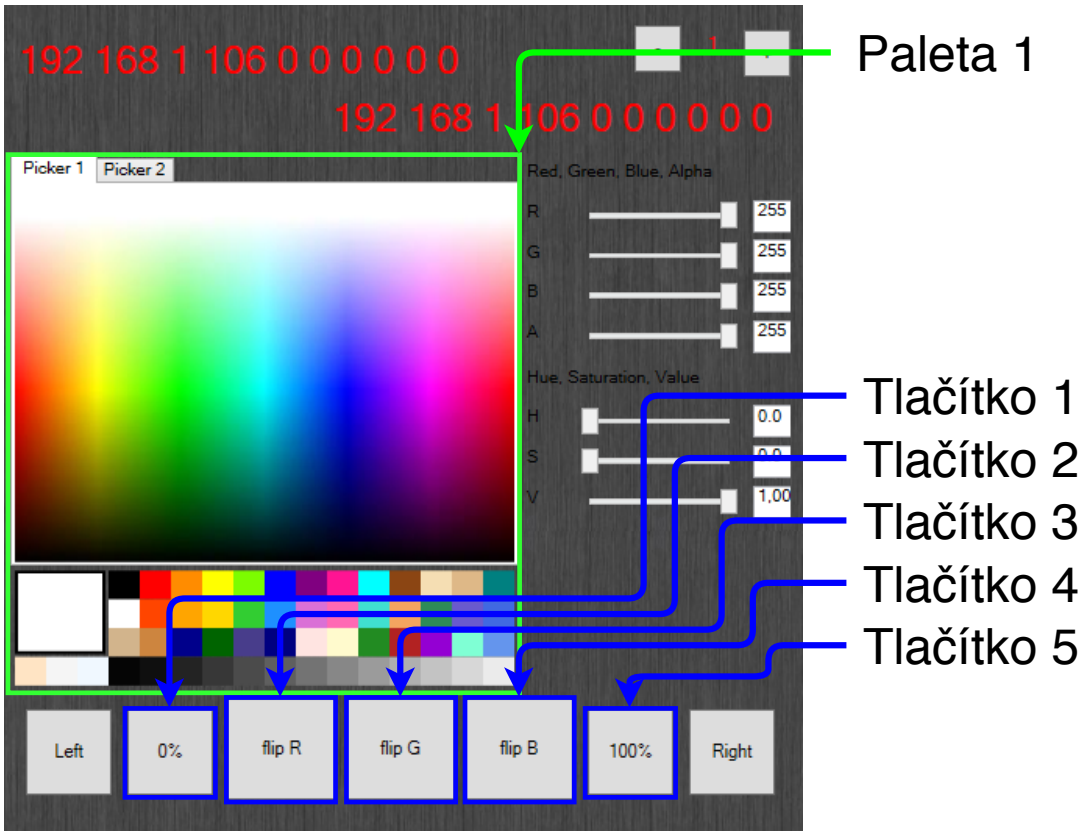
Tlačítko *flip R* (tlačítko 2 z obrázku 4.26) nastaví složce představující červenou barvu její doplňkovou hodnotu.

Tlačítko *flip G* (tlačítko 3 z obrázku 4.26) nastaví složce představující zelenou barvu její doplňkovou hodnotu.

Tlačítko *flip B* (tlačítko 4 z obrázku 4.26) nastaví složce představující modrou barvu její doplňkovou hodnotu.

Tlačítko *100%* (tlačítko 5 z obrázku 4.26) nastaví všem složkám představující barvy hodnotu 255.

Pro výběr barvy pak slouží barevná paleta (paleta 1 z obrázku 4.26), na které lze pomocí kurzoru, případně dotykem prstu, vybírat barvu.



Obrázek 4.26: Volič barev.

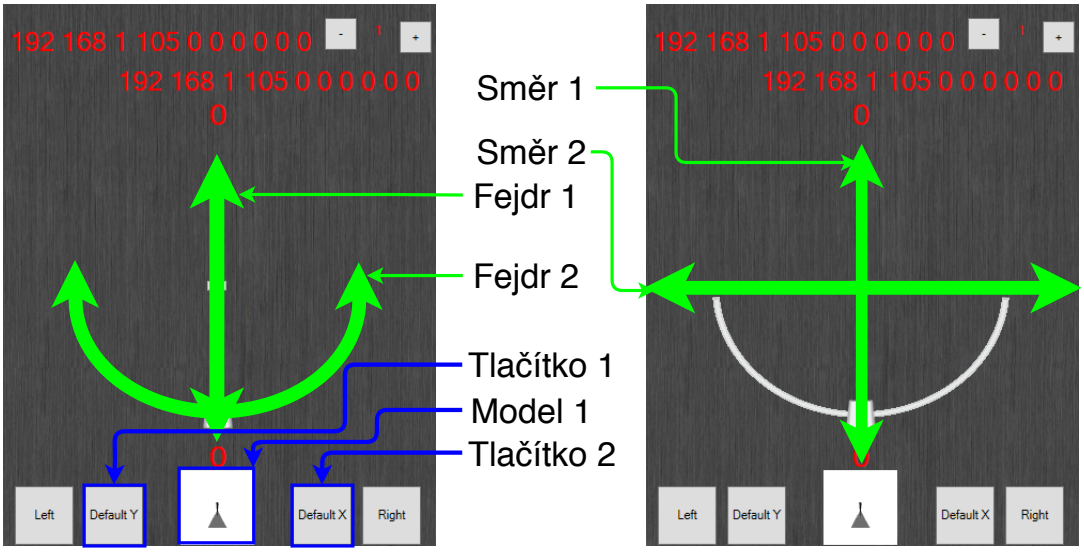
Joystick

Pro nastavení směru světla slouží ovladač joystick, který je zobrazen na obrázku 4.27.

Tlačítko *Default Y* (tlačítko 1 z obrázku 4.27), nastaví složce Y její defaultní hodnotu.

Tlačítko *Default X* (tlačítko 2 z obrázku 4.27), nastaví složce X její defaultní hodnotu.

Aktuální nastavení směru zařízení je možné si prohlédnout na obrázku modelu (model 1 z obrázku 4.27).



Obrázek 4.27: Joystick.

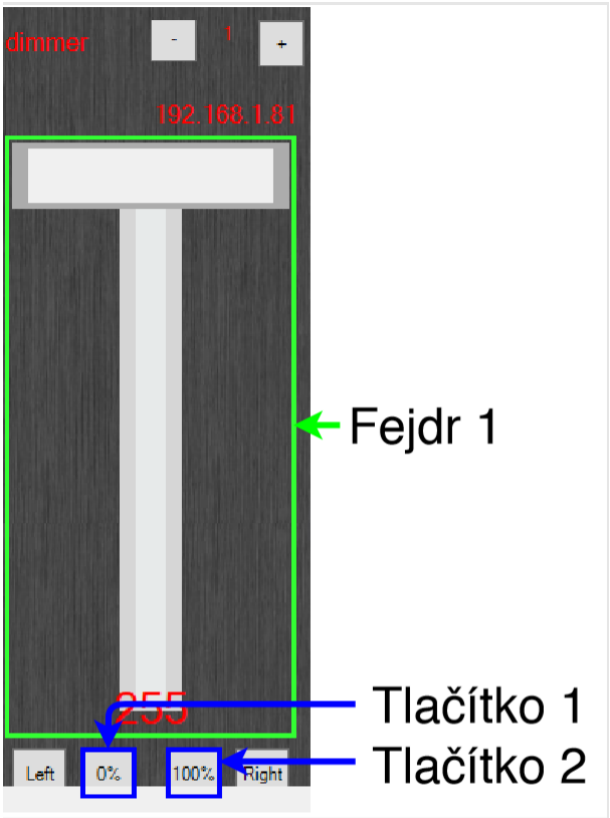
Fejdr

Pro nastavení hodnoty složce slouží ovladač fejdr, který je zobrazen na obrázku 4.28.

Tlačítko *0%* (tlačítko 1 z obrázku 4.28) nastaví složce složce hodnotu 0.

Tlačítko *100%* (tlačítko 2 z obrázku 4.28) nastaví složce složce hodnotu 255.

Pro nastavení hodnoty pak slouží fejdr (fejdr 1 z obrázku 4.28), který podporuje multitouch a lze s ním pohybovat pomocí kurzoru, případně dotykem prstu.



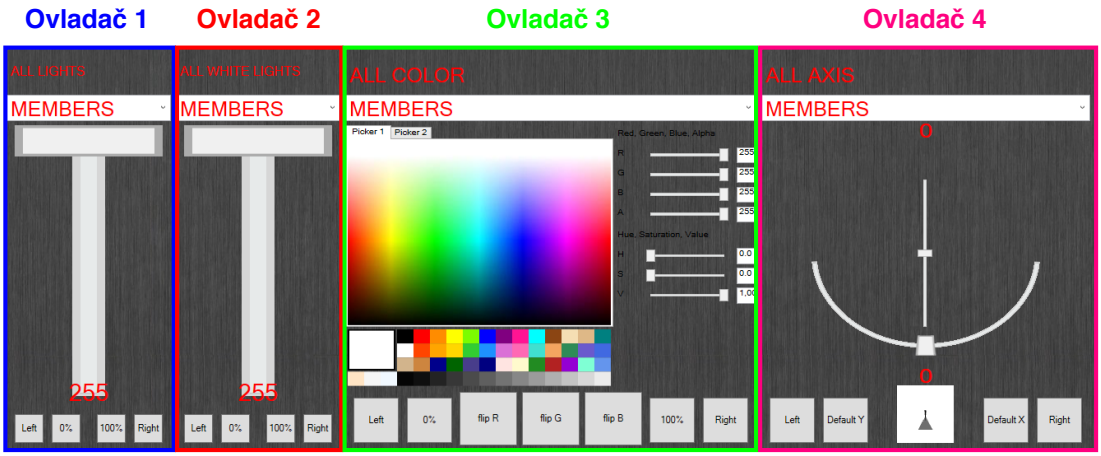
Obrázek 4.28: Fejdr.

4.6.2 Lišta globálních ovladačů

Jak můžete vidět na obrázku 4.29, virtuální ovládací konzole nabízí uživateli ovladače, které ovládají všechna zařízení daného druhu. Ovladače se používají stejně jako ovladače ze sekce 4.6.1 Lišta ovladačů.

Konzole pak poskytuje čtyři základní globální ovladače:

- Ovladač *ALL COLOR* (ovladač 1 z obrázku 4.29) ovládá všechna zařízení, která svítí barevně.
- Ovladač *ALL AXIS* (ovladač 2 z obrázku 4.29) ovládá všechna zařízení, která mohou měnit směr světla.
- Ovladač *ALL WHITE LIGHTS* (ovladač 3 z obrázku 4.29) ovládá všechna zařízení, která svítí jen bílým světlem.
- Ovladač *ALL LIGHTS* (ovladač 4 z obrázku 4.29) ovládá všechny složky zařízení, které představují světelnou hodnotu.



Obrázek 4.29: Globální ovladače.

4.6.3 Vytváření skupin zařízení

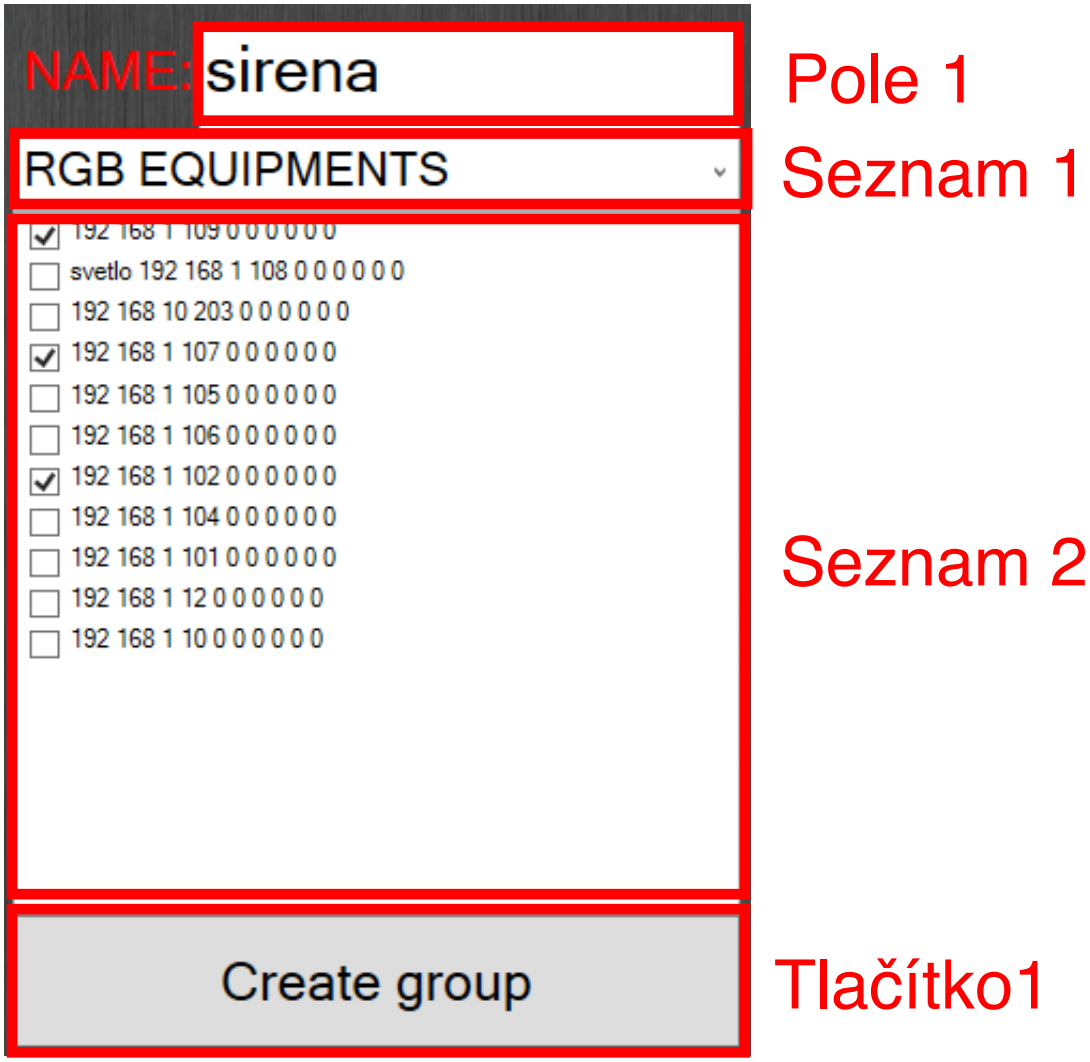
Při vytváření skupiny zařízení je nutné zadat do textového pole (pole 1 z obrázku 4.30) jméno vytvářené skupiny. V seznamu možných skupin (seznam 1 z obrázku 4.30) vybrat druh skupiny, který určí možné členy skupiny a ovladač pro skupinu. Na výběr jsou:

- Druh skupiny *RGB*, členy skupiny mohou být zařízení svítící barevným světlem. Ovladačem je pak Volič barev z obrázku 4.26.
- Druh skupiny *AXIS*, členy skupiny mohou být zařízení, která se mohou pohybovat po polokouli. Ovladačem je pak joystick z obrázku 4.27.
- Druh skupiny *ONE PART*, členy skupiny mohou být naprosto všechny složky zařízení. Ovladačem je pak fejdr z obrázku 4.28.

V seznamu (seznam 2 z obrázku 4.30) možných členů skupiny je pak nutné označit zařízení, která budou členy skupiny. Skupina nemůže být prázdná.

Po vybrání členů skupiny je pak nutné skupinu vytvořit tlačítkem *Create group* (tlačítko 1 z obrázku 4.30).

Po vytvoření je skupina uložena do seznamu možných skupin a použitelná z ovladače na načítání skupin.



Obrázek 4.30: Virtuální ovládací konzole.

ONE PART – škálovatelná skupina

Skupiny druhu *ONE PART* podporují, aby se jednotlivé složky vůči ovladači chovaly různými způsoby. K nastavení chování slouží lišty (lišty z obrázku 4.31), které je možné nastavit při výběru členů skupiny.

- Zaškrtávátko (zaškrtávátko 1 z obrázku 4.31) slouží k určení, jestli daná složka patří do skupiny.
- Fejdr (fejdr 1 z obrázku 4.31) slouží k určení poměru hodnoty na ovladači a hodnoty předávané složce zařízení.
- Zaškrtávátko (zaškrtávátko 2 z obrázku 4.31) určuje, jestli zařízení je předána skutečná hodnota nebo její rozdíl do maximální hodnoty.

Obrázek 4.31:

4.6.4 Ovladač na načítání skupin zařízení

Pro spravování skupin slouží ovladač z obrázku 4.32.

Pomocí tlačítka *Load file* (tlačítko 1 z obrázku 4.32) lze načíst soubor se skupinami světel.

Po načtení skupin světel je možné označit skupiny jednotlivě pomocí zaškrtávátek (zaškrtávátka z obrázku 4.32), případně pomocí tlačítka *ALL* (tlačítko 2 z obrázku 4.32) označit všechny skupiny. Pomocí tlačítka *CLEAR* (tlačítko 3 z obrázku 4.32) lze odznačit všechny skupiny.

Tlačítko *ADD CHOSEN* (tlačítko 4 z obrázku 4.32) označené skupiny vloží do lišty ovladačů.

Tlačítka *SAVE GROUPS* (tlačítko 5 z obrázku 4.32) otevře vyhledávací okno, ve kterém lze vybrat soubor, do kterého se všechny skupiny uloží.

Obrázek 4.32: Virtuální ovládací konzole.

Závěr

V závěru práce zhodnotíme splnění požadavků vznesených v kapitole 1 Požadavky na systém.

Vytvořit systém pro komunikaci virtuálních ovládacích rozhraní a připravit možnou spolupráci s hardwarovými ovládacími panely. Pro komunikaci virtuálních ovládačů jsme vytvořili systém zpráv a brány potřebné na jejich odesílání a přijímání. Dále jsme připravili centralizovaný systém se třídami `Client` a `Server`, které spolu komunikují. V rámci podpory hardwarových ovládacích panelů jsme dodrželi požadavky autora zařízení na formát serializace zpráv. Zprávy jsou ve striktním úsporném formátu tak, aby je bylo možné deserializovat na jednoduchých zařízeních.

Implementovat, vytvořit a případně poupravit současné komunikační protokoly s ohledem na jejich další rozšíření. Pro komunikaci se zařízeními jsme navrhli rozhraní `IProtocol`, které bude sloužit k transformaci dat do formátu, který jednotlivá zařízení používají. Pro zařízení používaná v současném systému jsme implementovali protokoly splňující rozhraní `IProtocol` a umožnující komunikaci se zařízeními. Při případném rozšiřování protokolů je zapotřebí pouze implementovat rozhraní `IProtocol` tak, aby odpovídal novému protokolu.

Vytvořit počítačový systém na ovládání osvětlení, který bude podporovat ovládání z více konzolí. V rámci projektu jsou implementovány spustitelné aplikace User.exe a Server.exe, které jsou schopné mezi sebou schopny komunikovat a ovládat zařízení připojená do systému.

Systém bude oproti stávajícímu ovládání ulehčovat přidávání zařízení a ovládacích prvků. Pro přidání zařízení do systému stačí načíst předem připravený soubor, případně přidat nové zařízení přes jednoduchý formulář. Ovládací prvky se automaticky generují pro všechna zařízení.

Grafické rozhraní pro nastavování dat. V rámci systému je implementována virtuální ovládací konzole.

Software má být schopen ovládat typické výkonové prvky s podporou komunikace UDP jako jsou stmívače, LED rampy a motorem řízené prvky. Aplikace Server.exe ovládá všechna zařízení v současném systému a je připravena na případné rozšíření.

Budoucí rozšíření systému

Již při vývoji současné verze světelného systému se začaly objevovat nápady na užitečná budoucí rozšíření systému. Hlavní směry budoucího rozšíření si probereme v následujících částech.

Scénář

Při osvětlování divadelní hry je často zapotřebí ovládat v jeden okamžik velký počet světel. V současnou chvíli je úkolem osvětlovače se s touto činností vypořádat.

Jedním z možných usnadnění práce osvětlovače by bylo poskytnout mu možnost vytvořit v systému scénář, který by řídil světla místo něho. Nešlo by o to,

aby počítač řídil celou hru, ale jen určité momenty.

Například je-li potřeba ve hře simulovat policejní sirény. V současné době se to dělá pomocí červeného a modrého světla. Na hardwarové konzoli je nutné, aby osvětlovač plynule nastavoval pomocí dvou fejdrů hodnoty tak, aby byly vůči sobě inverzní, což je velmi složitá činnost. V našem systému si lze vyrobit skupinu zařízení, která obsahuje právě červené a modré složky tak, aby byly vůči sobě inverzní. Vzniklým ovladačem pak stačí jezdit plynule od minima k maximu a dosahujeme stejného efektu. Rozšíření o tento scénář by nám pak poskytlo možnost pouze spustit požadovaný efekt a v případě potřeby ho vypnout.

Hudební světla

Divadlo neslouží jenom pro hraní divadelních her, ale i pro vystoupení kapel a pořádání jiných kulturních akcí. Při vystoupení kapely je normální, že koncert je doplněn o světelné efekty. Profesionální kapely, které vystupují v divadle, mají vlastní vybavení, které se o to postará, případně mají vlastního osvětlovače s příslušnou technikou. Při vystoupení začínajících kapel, případně při pouštění hudby ze záznamu, by také bylo vhodné doplnit hudbu světelnými efekty. Za tímto účelem by pak bylo vhodné do systému implementovat modul, který by se o ovládání světel při takových příležitostech staral. V ideálním případě by světla nastavoval podle hudby, případně by i stačilo, aby světla ovládal podle předem nachystaných scénářů.

Mobilní aplikace

Při chystání divadelního systému například v případech, kdy divadlo vystupuje v cizím divadle, ale používá vlastní světla, nebo při změně divadelních her, je potřeba světla fyzicky přesunout a správně nasměrovat. Tyto úpravy často probíhají ze žebříku a tudíž není možné vzít si sebou počítač či hardwarovou konzoli. Za tímto účelem by dalším rozšířením systému mohla být nad klientskou třídou implementace mobilní aplikace, která by v alespoň omezené míře umožňovala nastavování světel.

Seznam použité literatury

[1] Testování jednotek u kódu v jazyce c# s použitím nunit a .net core. online. URL https://docs.microsoft.com/cs-cz/dotnet/core/testing/unit-testing-with-nunit.

[2] Windows forms. online. URL https://docs.microsoft.com/cs-cz/dotnet/framework/winforms/.

[3] Windows presentation foundation. online. URL https://docs.microsoft.com/cs-cz/dotnet/framework/wpf/.

[4] Přehled 3d grafiky. online. URL https://docs.microsoft.com/cs-cz/dotnet/framework/wpf/graphics-multimedia/3-d-graphics-overview.

[5] Charles, P. Curved scrollbars using wpf 3d. online. URL http://www.charlespetzold.com/blog/2008/08/Curved-ScrollBars.html.

[6] Jon, P. Transmission control protocol. online. URL https://tools.ietf.org/html/rfc793.

[7] Jon, P. User datagram protocol. online. URL https://tools.ietf.org/pdf/rfc768.pdf.

[8] Lex, L. Dockpanelsuite. online. URL https://www.nuget.org/packages/DockPanelSuite.

[9] Li, L. Existing themes. online. URL https://www.nuget.org/packages/DockPanelSuite.ThemeVS2015/.

[10] MIT. Colorpickerwpf. online. URL https://www.nuget.org/packages/ColorPickerWPF/.

[11] Newtonsoft. Newtonsoft.json. online. URL https://www.nuget.org/packages/Newtonsoft.Json/.

[12] Wikipedia. Art-net. online. URL https://en.wikipedia.org/wiki/Art-Net.

[13] XCEED SOFTWARE, I. Extended wpf toolkit. online. URL https://www.nuget.org/packages/Extended.Wpf.Toolkit/.

Dostupnost těchto stránek byla ověřena ke dni 4.6.2020.

A. Přílohy

- /ObjectsForLights – Solution ObjectsForLights obsahující:
  - projekt knihovny ObjectsForLights
  - projekt testů LightsComponentsTests
- /GuiApplications – Solution GuiApplications obsahující:
  - projekt knihovny GuiComponents
  - projekt knihovny RealTimeControl
  - projekt aplikace GUIServer
  - projekt aplikace GUIUser
- /Server – Složku Server obsahující spustitelnou aplikaci Server.exe
- /User – Složku User obsahující spustitelnou aplikaci User.exe
- /Data – Složka obsahující základní data pro systém
- /prace.pdf – Soubour s touto prací