Mendelova univerzita v Brně
Provozně ekonomická fakulta

Systém pro hromadnou správu počítačů

Bakalářská práce

Vedoucí práce:
Ing. Jiří Lýsek, Ph.D.

Jan Vyhnalík

Brno 2016

Rád bych poděkoval vedoucímu bakalářské práce, panu Ing. Jiřímu Lýskovi, Ph.D. za jeho ochotu a cenné rady při řešení této práce. Také bych chtěl poděkovat panu Ing. Tomášovi Koubkovi za spolupráci při testování aplikace a dále také moji rodině za podporu při zpracování této práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Systém pro hromadnou správu počítačů** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně 22. prosince 2016 .....

Abstract

Vyhnalík, J. Creating system for mass computer administration. Bachelor thesis. Brno: Mendel University, 2016.

Bachelor thesis deals with implementing a system for employees of Mendel University, Faculty of Business and Economics. Application serves for managing computers and it's automation. Implementation of responsive web application is accomplished by PHP framework Laravel, JavaScript's library jQuery, AJAJ, HTML and CSS. Final solution improves speed and simplicity of daily operation in computer's management field.

Keywords

computer management, Morpheus, Laravel, web application

Abstrakt

Vyhnalík, J. Tvorba systému pro hromadnou správu počítačů. Bakalářská práce. Brno: Mendelova univerzita v Brně, 2016.

Tato bakalářská práce se zabývá vývojem aplikace pro zaměstnance Provozně ekonomické fakulty Mendelovy univerzity. Aplikace slouží ke správě počítačů a její automatizaci. Pro implementaci responzivní webové aplikace byl použit PHP framework Laravel, JavaScriptová knihovna jQuery, AJAJ, HTML a CSS. Vytvořené řešení umožňuje zjednodušit a urychlit každodenní úkony v oblasti správy počítačů.

Klíčová slova

správa počítačů, Morpheus, Laravel, webová aplikace

Obsah

1 Úvod a cíl práce

1.1 Úvod

Správa počítačů začíná být náročná se zvyšujícím se počtem klientských stanic, zvláště ve větších organizacích, které jsou závislé na každodenní práci s nimi. Jeden z hlavních cílů administrátora je snažit se zajistit maximální dostupnost stanic z důvodu rizika vzniku ušlého zisku při jejich nefunkčnosti, a také usiluje o zvýšení efektivity jejich provozu z ekonomického nebo uživatelského hlediska.

V dnešní době je dostupné nepřeberné množství různých technologií k dosažení těchto cílů. Základní platforma pro správu počítačů, jako je například Active Directory nedokáže poskytnout řešení pro pokročilé požadavky administrátora nebo uživatelů. V tomto případě je třeba vytvořit vlastní individuální řešení na míru dané organizaci a její infrastruktuře.

Tato práce se bude zabývat tímto individuálním řešením ve školním prostředí. Díky automatizaci úloh, která je jedna z klíčových oblastí pro zvýšení efektivity umožní zjednodušit práci administrátorovi a zapojí vybrané učitele do základní správy počítačů. Například jim umožní vzdálené zapínání studentských počítačů ve svých učebnách v konkrétní čas začátku výuky. Tímto můžou zamezit zdržení začátku výuky a ušetřit drahocenný čas.

Dále se bude zabývat webovými technologiemi, databázemi a responzivním uživatelským rozhraním, protože responzivita je v dnešní době velmi diskutovaná a žádaná záležitost, která zajišťuje uživatelům vysoký komfort a flexibilitu.

1.2 Cíl práce

Cílem práce je navrhnout a implementovat aplikaci pro hromadnou správu počítačů dle požadavků ÚI PEF Mendelu s ohledem na responzivitu uživatelského rozhraní. Aplikace by měla umožnit zobrazit přehled spravovaných počítačů a zpřístupnit nad nimi funkce, jako je například zapínání, vypínání, přihlášení uživatele, atd. Dále by měla umět spravovat automatické provádění těchto funkcí a přihlašování uživatelů. Uživatelé aplikace budou získávat oprávnění dle uživatelské role.

2 Aktuální stav

2.1 Dosavadní řešení

Na naší fakultě již jedno řešení existuje pod názvem Morpheus (náhled viz. přílohy) a je využíváno vybranou skupinou učitelů a správců pouze pro hromadné vypínání a zapínání počítačů. Je to jednoduchá webová aplikace založená na HTML a CSS, která odesílá HTTP požadavky na další server, který poskytuje SOAP[0] API[0]. Toto API následně spouští Bash skripty vytvořené administrátorem v prostředí operačního systému Linux CentOS. Shromažďuje informace o počítačích umístěných v jedenácti učebnách fakulty a ukládá je do databáze PostgreSQL. Funkce tohoto API jsou dostupné na serveru školy, kde poskytují velkou škálu možností a informací, které budou klíčové pro moje nové řešení a je potřeba si je kompletně prostudovat.

Tato aplikace je již nevyhovující a má velmi mnoho nedostatků, kterým se budu dále věnovat.

2.1.1 Uživatelské rozhraní

První problém je její uživatelská přívětivost. Rozložení rozhraní nepůsobí přehledně, výběrové prvky jsou malé a navíc blízko u sebe. To nutí uživatele trávit delší dobu správným výběrem požadovaných počítačů a hledáním informací. Uspořádání tlačítek s volbami je nelogické a jejich význam není zcela zřejmý. Kliknutí na tlačítko neposkytuje žádnou zpětnou vazbu a uživatel nemá žádnou informaci o tom, zda se mu podařilo kliknout správně, jestli se akce provedla a s jakým výsledkem.

V případě této aplikace je téměř nemožné ovládat aplikaci z mobilního zařízení, nebo tabletu. Chybí zde přímočaré a rychlé ovládání, na které je potřeba se zaměřit obzvlášť.

2.1.2 Komunikace aplikace

Druhý problém je v komunikaci a výměně dat mezi aplikací a API. Předávají si mezi sebou neefektivním způsobem velké množství zbytečných dat a to znatelně zpomaluje práci, nebo vede k nekonečnému čekání. Navíc má negativní dopad na rychlost školní sítě.

2.1.3 Management uživatelů

Aplikace je přístupná aktuálně pouze pod jediným loginem a heslem. To je velmi nepraktické z důvodu netransparentnosti chování uživatelů po přihlášení, protože nikdo neví, jestli je někdo zrovna přihlášený a co tam dělá. Není tam žádný administrátor, který by měl nad tím dohled a všichni mají neomezený přístup ke všem funkcím

[0]Formát pro posílání a přijímání zpráv založený na XML, který je nazávislý na platformě (W3schools, 2016)

[0]Abstraktní rozhraní definující funkce pro interakci se systémem (Wikipedia, 2016)

a počítačům. To znamená, že učitelé mají přístup k ovládání počítačů v učebnách, kde ani nepůsobí a nemají právo na tyto akce.

2.1.4 Nabídka funkcí

Další problém je v omezené funkčnosti aplikace a nevyužití potenciálu API, které nabízí rozsáhlou škálu funkcí a informací ze stanic. Z nabídky funkcí bych zmínil zejména přihlašování uživatelů do systému a provolání libovolného příkazu z nabídky Windows Management Instrumentation. Z informací o stanicích se hodí zejména stav zapnutí a nabootovaný systém, název nebo přihlášený uživatel.

Všechny dostupné funkcionality aplikace se v praxi často opakují, je nutné se každodenně přihlašovat a ručně je spouštět. To je pro uživatele značně nekomfortní a také zbytečně zatěžují webový server. Chybí tu nějaká možnost tyto funkcionality automatizovat a spravovat.

2.2 Přehled prací

Komunitní web pro hledače pokladů navrhoval Jiří Vyhnálek (Vyhnálek, 2014). Velkou část práce věnoval analýze požadavků lidí z jeho cílové skupiny a následnému zhodnocení uživatelských ohlasů. Tímto postupem se dopracoval k aplikaci, která je u uživatelů kladně hodnocena a je jim vytvořena na míru jejich potřebám. Dále představuje možné platformy použitelné pro vývoj. Srovnává výhody a nevýhody PHP frameworků a šablonovacích systémů. Jeho výběr frameworku Laravel a jeho argumentace je logická, lze s ní souhlasit. Zdůrazňuje, že český nástroj Nette, který je u nás zatím oblíbený a hodně využívaný už začíná být zastaralý. Jako výhodu jeho řešení vidím v použití moderních přístupů, jako je možnost přihlašování přes Facebook, responzivní design nebo užití mapových podkladů pro zaznamenání a přehled všech možných uživatelských nálezů s detaily a fotkami. Další zajímavá technologie je RESTful controller (API), která je vhodná pro další rozšíření jiných aplikací.

Návrhem rezervačního systému učeben se zabýval Petr Vybíral (Vybíral, 2012) ve své bakalářské práci. Autor využil k řešení framework firmy Microsoft – ASP.NET. Popisuje a implementuje webovou službu v C# s využitím MVC modelu. Toto řešení je obecné a umožňuje využití pro další možné aplikace, které budou využívat její data. Jako nevýhody tohoto řešení jsem shledal nedořešené grafické rozhraní a jeho responzivitu, která chybí úplně. Dále absence ošetření uživatelských vstupů a flexibilita tohoto řešení, které v případě přechodu na jinou platformu webového serveru bude obtížné migrovat.

Webové rozhraní pro správu uživatelských účtů navrhoval ve své bakalářské práci Filip Nešpor (Nešpor, 2007). Řešil problém jednotného ovládání a modifikování uživatelských účtů v Active Directory z důvodu přístupu více administrátorů a omezení počtu připojení. Jako řešení využil webovou aplikaci na serveru, která spouštěla jeho skripty v prostředí Windows Management Instrumention pro interpretaci příkazů a modifikaci databáze Active Directory. Tato aplikace je přístupná pouze pro

úzký okruh lidí, takže autor zvolil jednodušší a obyčejné rozhraní a zaměřil se hlavně na zabezpečení – nastavení Access Control List, SSL šifrování, zaheslování, kontrolu vstupů a také logování událostí. Nevýhoda je, že web umístil přímo na hlavní server a tím zvyšuje jeho zranitelnost a porušuje pravidla uspořádání služeb, které by mělo být odděleně na samostatných strojích.

Bakalářská práce pana Révaye (Révay, 2016) se zabývá návrhem a implementací informačního systému pro klub karate. Její cíl je vytvořit aplikaci, která by uměla spravovat členy klubu, jejich role oprávnění, plánování tréninků, místností a soutěže. Autor pracoval na webové aplikaci s pomocí frameworku Nette, alternativním technologiím se moc nevěnoval, stejně jako grafické rozhraní vytvořil kompletně v Bootstrapu bez dalšího zkoumání. V závěru nakonec aplikaci vyvinul, ale už nezmiňuje, jestli proběhlo nějaké testování nebo uživatelské hodnocení.

3 Dostupné technologie

3.1 HTML 5

HyperText Markup Language - neboli značkovací jazyk, který slouží pro vytvoření struktury webové stránky. Tato struktura je tvořena párovými a nepárovými HTML tagy. Je reprezentována stromovým rozložením, kterému se také říká Document Object Model (DOM). Verze 5 byla doporučena konsorciem W3C v roce 2014 a dnes je vnímána jako standard. Oproti verzi 4 obsahuje například nové sémantické značky *<header>,<nav>,<article>* atd. pro lepší přehlednost a optimalizaci vyhledávání. Další vylepšení je přehrávání videí a multimediálního obsahu tagem *<video>*, takže v prohlížeči není dále potřeba instalace flash pluginu a ostatních. Ještě bych zmínil například vykreslování grafiky Canvas za pomocí JavaScriptu, a podpora vektorové grafiky formátu SVG (Castro, 2012), (W3schools, 2016).

3.2 CSS 3

Cascading Style Sheet - neboli list kaskádových stylů je nezbytný pro definici vzhledu HTML prvků. Prvky DOMu jsou vybrány selektorem a na ty aplikuje definici stylu. Princip je založený na překrývání a dědičnosti stylů. Vývoj CSS probíhá zároveň s HTML a dnes jsou tyto dvě technologie vzájemně nerozlučné. CSS 3 nabízí mimo základní grafické styly a pozicování také pokročilé transformace, animace a stíny. Další užitečná věc, kterou CSS nabízí jsou dotazy na média, tzn. schopnost vytvoření selektoru závislého například na šířce obrazovky klienta (Castro, 2012).

Používání frameworků může urychlit vývoj nového front-endu. Kvalitní frameworky pomohou zlepšit celkovou funkčnost a výkon celého webu či aplikace. Použití vhodného frameworku může být přínos pro celkovou user experience webové stránky (Interval, 2015).

3.2.1 Bootstrap

Nejznámější a zároveň nejkomplexnější zástupce CSS frameworků je Bootstrap. Nabízí mnoho předpřipravených prvků pro rychlé a efektivní vytvoření layoutu i s podporou responzivity. Toho dosahuje rozdělením stránky do mřížky dvanácti pomyslných vertikálních sloupců. Každá šířka sloupce layoutu je definována počtem z těchto dvanácti sloupců. Šířky sloupců layoutu se tak se změnou okna prohlížeče mění v závislosti na těchto poměrech. Mimo jiné je zde velké množství hotových grafických prvků a JavaScriptových efektů (Bootstrap, 2016).

3.2.2 Pure

Další z řad CSS frameworků je Pure, jeho předností je minimální velikost souboru importovaného stylu a rychlejší načítání. Také podporuje vytváření responzivního layoutu, ale je omezený na pěti, nebo dvacetičtyř sloupcový layout. Nevýhoda je,

že neobsahuje moc grafických prvků, konkrétně postrádá například modální okna (Purecss, 2016).

3.2.3 Preprocesory

Pro větší projekty se používají preprocesory (nástavby), které pomáhají zjednodušit a zpřehlednit zápis velkého množství CSS stylů. Jako jejich zástupce můžeme označit např. LESS a SASS. Jejich nespornou výhodou je možnost zápisu stylu zanořením do jiného.

LESS rozšiřuje CSS o dynamické prvky jako jsou proměnné, mixiny, výpočty a funkce. Běží jak na klientské straně (prohlížeče), tak na straně serveru, s Node.js a Rhino (Lesscss, 2016). Funkcionalita obou preprocesorů je prakticky stejná, záleží zejména na osobních preferencích.

3.3 JavaScript

JavaScript je interpretovaný a objektový programovací jazyk, který se spouští na klientských prohlížečích. Umožňuje provádět manipulaci s prvky webové stránky (v DOMu), jako je například přidávání elementů, zobrazování zpráv a přesměrování adresy. Lze jej využít i pro HTTP komunikaci pomocí AJAX[0]/AJAJ. Je tedy schopný načítat data na pozadí ve formátu XML/JSON bez načtení celé stránky, to umožňuje plynulejší zobrazování obsahu webu.

JavaScript má mnoho vzorů pro vytváření objektů a nabízí obvykle více než jednu cestu k dosažení stejné věci. Lze definovat své vlastní typy nebo vlastní obecné objekty kdekoliv chceme. Ačkoliv JavaScript nemá koncept tříd, tak stále využívá dva druhy typů: primitivní a referenci. Primitivní typy jsou uloženy jako klasické datové typy. Reference jsou uloženy jako objekty, které jsou jen jako odkazy do místa v paměti. (Zakas, 2014)

Samotný jazyk má však i své nevýhody, jeho syntaxe je zdlouhavá a některé funkce se chovají v každém prohlížeči odlišně. Proto nad tímto jazykem vznikly knihovny a frameworky.

3.3.1 jQuery

jQuery je nejrozšířenější knihovnou, jeho výhody jsou v zjednodušené práci s událostmi, AJAXem a animacemi. Pracuje s jQuery objekty, které se volají pomocí znaku *$_* (JQuery, 2016). V praxi má jednoduchou syntaxi, ale jeho nadměrná variace vstupů v některých voláních není zrovna příjemná.

3.3.2 AngularJS

AngularJS od společnosti Google můžeme zařadit mezi frameworky. Už od verze číslo jedna přichází se zajímavými koncepty, jako nejužitečnější se uvádí Two Way

[0]Asynchronous JavaScript and XML

Data-Binding, implementace Dependency Injection, testovatelnost, direktivy a znovupoužitelnost komponent (AngularJS, 2012). Osobně se mi ale moc nelíbí jeho složitost s velkou dávkou "magie".

3.4 PHP

Zkratka PHP je rekurzivní akronym pro Hypertext Preprocesor. PHP je celosvětově rozšířený a oblíbený open source skriptovací jazyk, který je vhodný zejména pro programování na webu a může být vnořen do HTML (Leiss, 2010).

Tento programovací jazyk se spouští na straně serveru, v praxi to znamená, že webový prohlížeč pošle požadavek na server, kde se spustí náš vytvořený skript, který dynamicky vygeneruje HTML stránku a odešle ji zpět jako odpověď. Skript už z principu nemůže běžet neomezenou dobu, ale je omezen na určitý čas a poté se ukončí.

Objektově orientované programování stojí u PHP jasně v popředí, následuje tím trend, který se projevil také u jiných programovacích jazyků (Leiss, 2010). Verze 5, která je nyní nejrozšířenější a má největší podporu mezi poskytovateli webhostingu upevnila svou pozici i jako objektový jazyk. Využívá klasických přístupů, jako je zapouzdření, dědičnost a polymorfismus. Rok 2015 přinesl novou verzi s číslem 7, jeho klíčové přednosti jsou až dvakrát vyšší výkon při využití méně operační paměti, nové operátory, lepší ošetřování vyjímek a další (Hongkiat, 2015).

3.4.1 MVC a frameworky

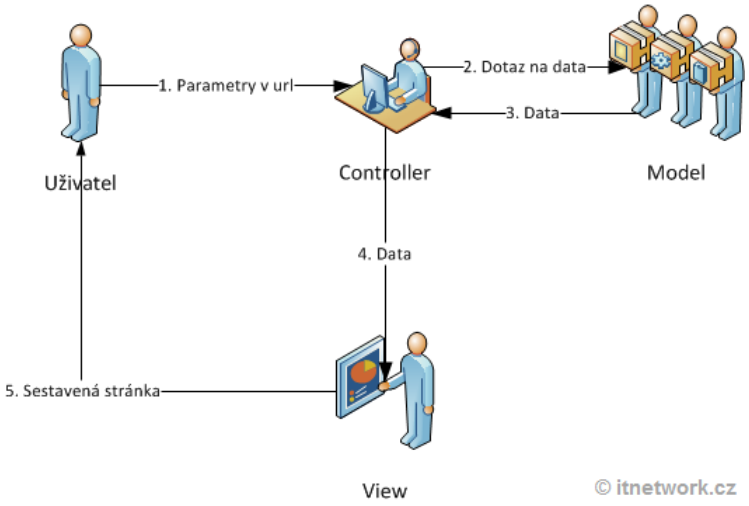
Některé aplikace začínají být v dnešní době velmi rozsáhlé, už nestačí psát v jednom souboru logiku aplikace zároveň s vykreslováním grafiky (typicky v PHP pomocí funkce echo a HTML tagů) a zápis databázových dotazů. Tento přístup je dlouhodobě neudržitelný a také nepraktický. Problém je zejména v přehlednosti kódu a jeho pozdější úpravě. Každý programátor má do jisté míry vlastní postup pro tvorbu aplikace, jeho kolegové tak mají při společných projektech ztíženou práci. MVC architektura je základ pro každou firmu zabývající se vývojem aplikací, jejich projekty se bez ní již těžko můžou obejít.

Architektura obecně určuje strukturu a vztahy, konkrétně typ MVC rozděluje aplikaci na komponenty Model, View, Controller a popisuje jejich vazby (Itnetwork, 2013).

Model je určen pro manipulaci s daty, nejčastěji komunikuje s databází. Jeho úkol je pouze ukládat a doručovat správná data v případě volání jeho metod (Itnetwork, 2013).

View neboli pohled je pověřen zobrazováním výsledků operací. To obvykle znamená použití šablonovacího systému, který přidává vlastní značky do HTML struktury (Itnetwork, 2013).

Controller je řídící komponenta, zastupuje business logiku aplikace s podporou modelů. Výstup je zobrazen ve View (Itnetwork, 2013).

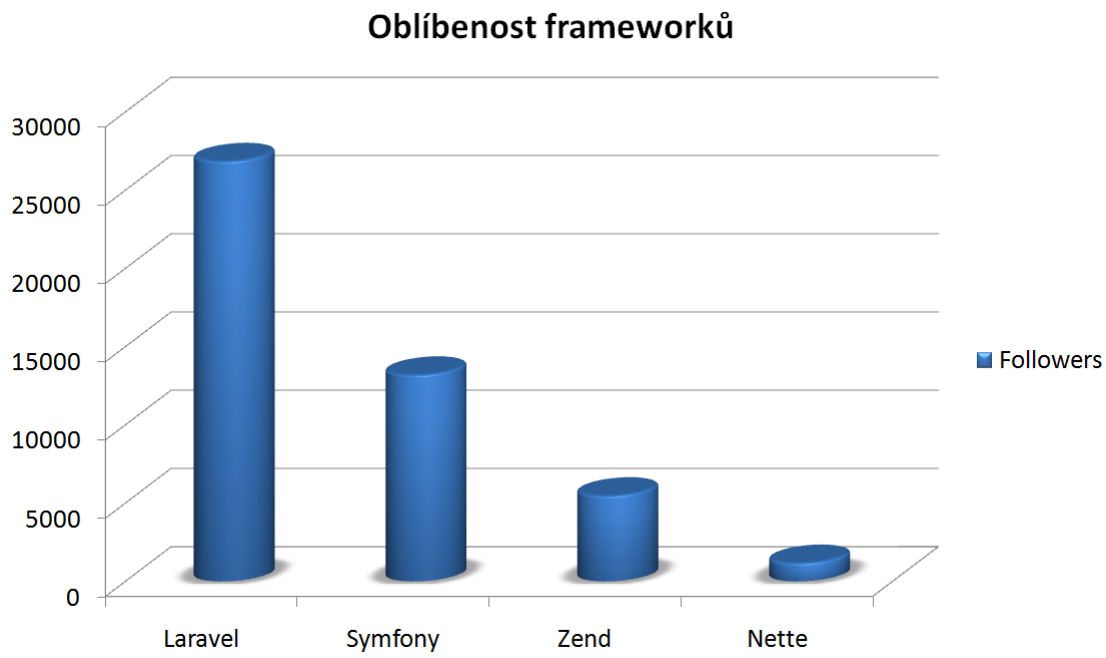


Obrázek 1: Rozdělení MVC architektury (Itnetwork, 2013)

MVC je velmi oblíbený architektonický vzor, který se uchytil zejména na webu, ačkoli původně vznikl na desktopech. Je součástí populárních webových frameworků, jakými jsou např. Zend nebo Nette pro PHP, Ruby On Rail pro Ruby nebo MVC pro ASP .NET (Itnetwork, 2013). PHP frameworky přichází s vlastní implementací MVC a přináší další hotové funkcionality. Jejich cíl je implementovat elementární funkce, které programátoři musí opakovaně programovat, ztrácet s nimi čas. Z toho vyplývá, že programátor se může více soustředit na svůj úkol, než samotnou implementaci.

1. Laravel - ve světě nejpoužívanější, poměrně nový, podporovaný velkou komunitou a nabízí množství tutoriálů nebo šablonový systém Blade. Má Eloquent ORM[0] model (Hongkiat, 2016).
2. Symfony - největší množství funkcí v základu, jeho části používá i Laravel, velká komunita, ORM Doctrine, šablonový systém Twig (Hongkiat, 2016).
3. Nette - český produkt s poměrně malou komunitou, dobrá dokumentace v češtině, šablonovací systém Latte. Není známý ve světě a nemá ORM (Nette, 2016).
4. Zend - komplexní řešení určené spíše pro větší projekty, přichází s pokročilými nástroji, jako je např. kódovací kryptografické nástroje, editor drag & drop a Unit testování (Hongkiat, 2016).

[0]je programovací technika, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. (Wikipedia, 2016)



Obrázek 2: Popularita PHP frameworků 2016 dle Github (Github, 2016)

4 Metodika

Metodika je důležitá pro úspěšné dokončení každého softwarového projektu, představuje osnovu a doporučení postupu vývoje aplikace.

Na začátku budu sbírat požadavky od zainteresovaných osob, které mi pomohou vytyčit cíle a vytvořit si celkovou představu o požadované aplikaci.

Druhá navazující fáze bude pokračovat analýzou požadavků a celkové problémové domény. Analýza mi pomůže lépe pochopit řešený problém, zachytit všechny aspekty a úskalí aplikace.

Další fáze spočívá v navrhování uživatelského rozhraní, které zahrnuje tvorbu drátových modelů potřebných stránek a postupnou konzultaci návrhů s uživateli. Po odsouhlasení modelů přichází na řadu návrh back-endu, tedy samotná architektura aplikace a klíčové funkční principy. Je potřeba také zvážit vhodné technologie pro implementaci aplikace.

Vyřešení těchto otázek mi umožní přejít k implementaci vybranými technologiemi a k vytvoření samotné aplikace.

Nikdy si však nemůžeme být jisti správností všech algoritmů na první pohled, proto přichází na řadu testování funkčnosti, opravy a optimalizace aplikace.

5 Požadavky

V této kapitole se budu snažit shrnout všechny důležité požadavky, které vznikly po komunikaci s kolektivem ÚI PEF Mendelu.

5.1 Grafické rozhraní

Uživatelské rozhraní aplikace musí mít funkční a minimalistický design bez zbytečných grafických prvků. Měl by být responzivní na všech typech běžně používaných zařízení, tzn. mobilní telefony, tablety a monitory.

Aplikace v základním náhledu zobrazuje přehled počítačů ve vybrané počítačové učebně a seznam skupin počítačů, který je rozdělen podle příslušnosti do učeben. Rozmístění počítačů v aplikaci musí odpovídat fyzickému rozmístění v konkrétní místnosti a pořadí je vždy takové, aby počítač označený číslem 01 (tedy učitelský) byl v horní části stránky. Výjimkou je počítačová studovna PEF, která bude orientována opačně a tedy počítač 01 bude v dolní části stránky. Pravidla rozmístění a pořadí však není nutné dodržet v mobilním zobrazení.

Každý počítač je barevně odlišen stavem, v jakém se počítač nachází. Seznam možných stavů je dán výčtem:

1. Zapnutý a dostupný
2. Vypnutý nebo nedostupný
3. Chybový stav - je třeba zásah administrátora kvůli nestandardním situacím
4. Přidán do výběru - očekává se, že s ním bude provedena nějaká akce

Počítač musí vždy zobrazovat svoje číslo, v případě zapnutí nabootovaný systém, a pokud je přihlášen nějaký uživatel, tak se zobrazuje jeho login. Pokud je přihlášen superuživatel (jejich seznam se bude editovat v aplikaci), tak jeho jméno bude zvýrazněno grafickým prvkem. Další typ zvýraznění bude přiřazen zvláštnímu uživateli *test* (specifický uživatel přihlašovaný v režimu přijímačky).

V rozhraní je důležitý prvek vybírání počítačů, ten je třeba provádět napříč učebnami (tedy např. 4 počítače ze skupiny Q04 a 1 počítač ze skupiny Q12).

5.2 Komunikace s API

Vstupní data pro tuto aplikaci budou získávány ze školního SOAP API. To je dostupné z jistého webového serveru, jehož přístup je omezen pravidly firewallu - pouze pro interní potřeby školy. Služba poskytuje vrstvu mezi shellovými skripty v prostředí UNIX a webovým rozhraním, které poskytuje funkce, jejichž parametry a návratové hodnoty jsou striktně definovány. Nachází se zde například skript pro zjištění stavu počítače, ten na základě IP adresy prověří dostupnost počítače s pomocí příkazu ping. Jako další skript lze uvést zapínání počítačů vzdáleně přes síť s využitím odesílání tzv. "magic packetu". Výsledek činnosti skriptů se ukládá do místní PostgreSQL

databáze, ze které se dále čerpají data pro příchozí požadavky různých aplikací. Demonstrativní přehled klíčových funkcí API:

- getAllClassrooms() - vrací seznam všech učeben v DB -> Array ()
- getPC(string ucebna) - vrací seznam počítačů v učebně -> Array ()
- getPosition(array počítače) - vrací pozici x,y počítačů učebně -> Array()
- getAvailability(array počítače) - vrací stav počítače -> [0(on) | 1(off) | 2(err)]
- wakeUp(array počítače) - zapíná počítače
- logMeIn(array počítače, string uzivatel, string heslo) - přihlásí uživatele

5.3 Funkcionalita

Systém umožňuje s jednotlivými počítači (nebo celými skupinami najednou) provádět správu počítačů, např. dálkově je zapínat, vypínat, spouštět na nich libovolné příkazy, přihlašovat vzdáleně uživatele.

Umožňuje správu provádět jak manuálně, tak automaticky pomocí pravidel. Pravidlo obsahuje několik povinných prvků - název, čas provedení, akci, která se má vykonat a specifikaci dnů v týdnu, ve kterých se pravidlo má provádět. Pravidla jsou v seznamu řezena podle času provedení. Je nutné, aby poskytoval jednoduchou správu a autorizaci uživatelů, kteří dostanou přístup. Proto budou v systému dostupné uživatelské oprávnění (role). Administrátor aplikace bude tyto role přidělovat, včetně omezení přístupu do jednotlivých učeben.

Autentizace uživatelů bude probíhat v režii školního LDAP serveru. V této části jsou jistá úskalí, jako je například otázka vstupu do aplikace v případě výpadku adresářového serveru, v tomto případě by ani admin nemohl provádět administraci. Tato problematika bude probírána dále.

Aplikace musí ukládat informace o tom, který uživatel provedl jakou akci v syslog kompatibilním tvaru kvůli zpracování a analyzování logů administrátorem.

5.4 Další nezbytnosti

Aplikační kód by měl být v souladu s dobrými programátorskými zvyky, jako je například vhodné pojmenování proměnných a funkcí, dobře strukturovaný a přehledný kód. Samozřejmě jsou vyžadovány komentáře popisující části programu a funkce. Celková přehlednost je podstatná pro pozdější rozšiřování nebo úpravu funkcionalit.

Jeden z požadavků je sepsání administrátorského manuálu, tedy dokumentu s vysvětlením postupu instalace a prvotního nastavení na serveru. Dokument bude dále obsahovat podrobný postup přidání dalších prvků pro rozšíření funkčnosti.

6 Analýza

6.1 Tenký a tlustý klient

Na začátku je třeba učinit rozhodnutí, jaký bude poměr mezi implementací v JavaScriptu a implementací v PHP. Jinými slovy lze říci, jak velkou část výpočetní zátěže přesuneme na klienta a jaká zůstane na serveru. Toto rozhodnutí následně vymezí typ klienta (tenký nebo tlustý) a kladené nároky na jeho zařízení. Rozhodnul jsem se jít střední cestou, která zachová dobrou úroveň interaktivity aplikace a nízké nároky na hardware. S tímto úkolem mi pomůže nejlépe jednoduchá JavaScriptová knihovna jQuery, v úvahu proto nepřipadá žádné komplexnější řešení typu AngularJS, atd.

6.2 Služby frameworku

Pro implementaci PHP, tzn. stěžejní části aplikace bude nejlepší volba framework Laravel. Z praktického hlediska přináší hodně výhod, například ORM Eloquent, který přichází s mezivrstvou nad databází a s modely. Používá vlastní syntaxi pro přístup k datům a jejich tvorbu, spolu s migracemi umožňují před nasazením aplikace jedním kliknutím změnit typ použité databáze. V případě přechodu na jiný databázový systém je díky tomuto řešení možné pouze jednoduše změnit jeden parametr v nastavení bez jakéhokoliv zásahu do kódu.

V Laravelu najdeme také pokročilou manipulaci s Cronem, bude se hodit pro automatizaci pravidel. Nebude třeba skoro vůbec zasahovat do úloh cronu, jen počáteční záznam a potom už jen manipulace s konkrétními metodami.

6.3 Front-end

Front-end bude zobrazovat v hlavním okně maximálně deset počítačů na šířku v případě studovny, zároveň musí být zajištěna responzivita, tj. čitelnost i při zmenšení obrazovky. Tomuto účelu plně vyhovuje implementace mřížky z Bootstrapu, která má dvanáct sloupců.

Na první pohled by rozhraní aplikace mělo pomoci uživateli jasně rozeznat definované funkční oblasti. Rozdělení je důležité pro uživatele z důvodu rychlého rozhodování v navigaci webu. Také by web měl obsahovat co nejméně zbytečných slov, částí a obsahu, protože většina lidí tento obsah stejně ignoruje (Krug, 2006).

Bootstrap obsahuje mnoho grafických prvků i pro tvorbu layoutu, ale je lepší si vytvořit vlastní s vlastním barevným schématem už z důvodu tuctového vzhledu Bootstrapu, důležitá pro mě bude zejména responzivní mřížka. Pro množství vylepšení v CSS stylech použiji preprocesor LESS.

6.4 Autentizace a správa uživatelů

Autentizace administrátora a všech ostatních uživatelů je třeba provádět ověřením jejich loginu a hesla, tyto údaje vznikají při registraci zaměstnance nebo studenta a jsou uložené v adresářové struktuře LDAP. S tím však přichází řada problémů, tj. například určení administrátora krátce po instalaci aplikace. Nelze předem v aplikaci stanovit login (člověka), který bude mít práva admina, protože daný login při další instalaci v budoucnu už nemusí ani existovat, nebo aplikaci nebude využívat.

Jelikož přidání uživatelů, kteří budou používat aplikaci je jednorázová záležitost při první instalaci, nabízí se využít konzoli Artisan. Konzole je součást frameworku a je dostupná po instalaci aplikace, lze v ní definovat vlastní příkazy, takže je zde prostor pro přidávání a odebírání uživatelů příkazem s parametry.

Prvotní instalaci včetně přidání uživatelů tak lze automatizovat v podobě spuštění jediného Shell skriptu. Ukázkový skript:

```
#!/bin/sh
git clone https://www.gitserver.com/aplikace.git .
cp my_config .env #Konfigurační soubor Laravelu - připojení k DB a LDAP
composer install #Stažení frameworku
chmod -R 777 /storage /bootstrap/cache #Oprávnění souborů
cp my_crontab /etc/crontab #Přidání příkazu 'artisan schedule:run'
php artisan migrate #Vytvoření struktury databáze
php artisan db:seed #Naplnění databáze
php artisan user:add --admin xlogin #Vytvoření účtu admina
```

Tento přístup má i jinou výhodu, instalaci provádí jeden člověk (administrátor serveru) a v tomto případě pouze on má právo a přístup k přidávání a odebírání uživatelů. Z toho vyplývá zvýšený stupeň kontroly a snížení vzniku potencionálních bezpečnostních rizik v podobě neoprávněného přidělení přístupu nepovolaným osobám.

Problém výpadku nebo nefunkčnost LDAP serveru je také nutné brát v zřetel. Nemohl by se přihlásit žádný uživatel včetně admina, pro tento případ bude bezpodmíněčně třeba vytvořit speciální nouzový administrátorský účet v režimu off-line autentizace.

Framework samozřejmě nepodporuje autentizaci na LDAPu, ale dle dokumentace (Laravel, 2016) si lze vytvořit vlastní ovladač s jakoukoli implementací.

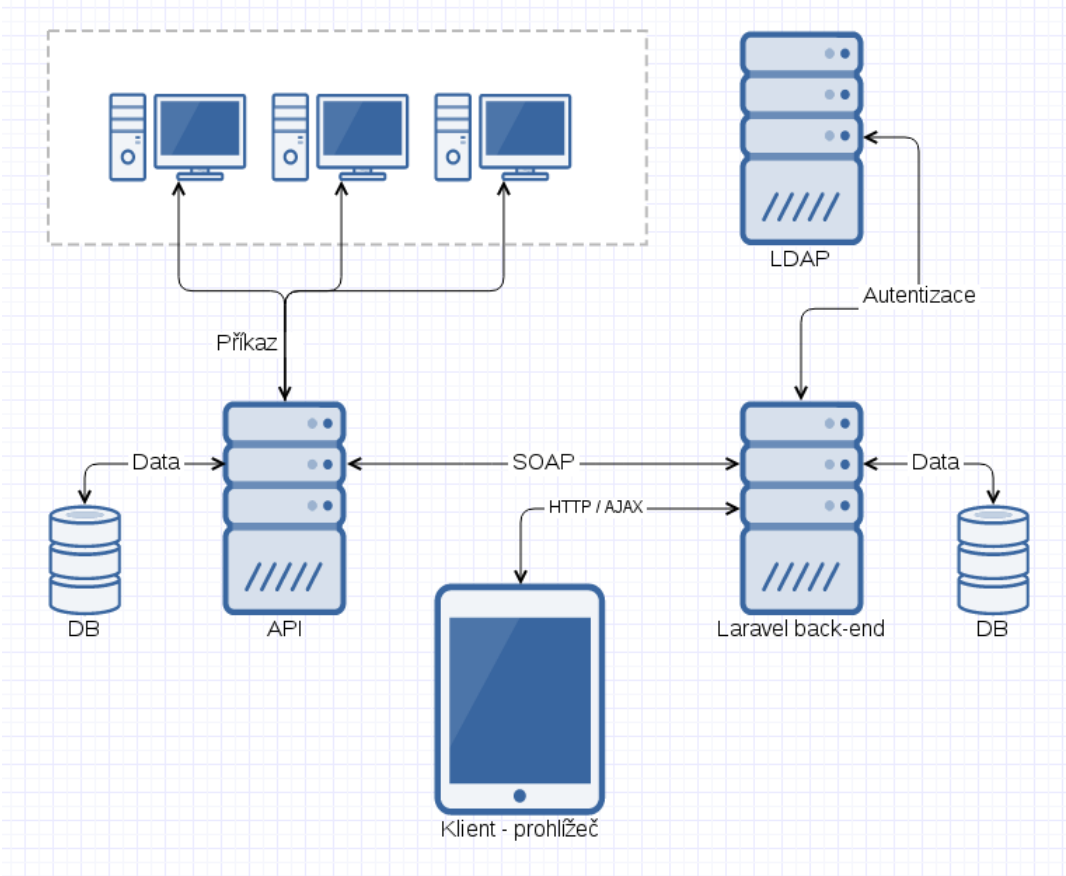
6.5 Závislost na API

Aplikace by se měla vždy přizpůsobovat výstupu z API. Zobrazit tento výstup není problém i při změnách typu odebrání učebny nebo počítačů. Problém nastává při přidávání pravidel do aplikace, v naší databázi tak musí být uloženy všechny počítače a učebny z API kvůli propojení pravidla s jeho počítači a učebnou. Aplikace musí proto kontrolovat aktuálnost lokální a vzdálené databáze, ideální kontrola bude při každém novém přihlášení uživatele. Po kontrole se musí v případě rozdílu

synchronizovat a provést další akce k zajištění integrity pravidel. Samozřejmě zde bude i ochrana před výpadkem API v podobě kontroly prázdnosti odpovědi.

Téměř každý požadavek klientského prohlížeče na tuto aplikaci je přímo závislý na výstupu z API. Nejčastější scénář komunikace je následující:

1. Klient odešle HTTP požadavek na server
2. Server ověří oprávnění požadavku
3. Server zpracuje požadavek a vytvoří nový, typu SOAP, který odešle na API
4. a) API odešle daný příkaz počítačům pomocí skriptu
   b) API aktualizuje databázi dle odezvy počítačů
   c) API vybere z databáze data požadována serverem
   d) API odešle odpověď na server
5. Server zpracuje odpověď a odešle přes HTTP novou, ve formátu JSON nebo HTML



Obrázek 3: Schéma komunikace aplikace

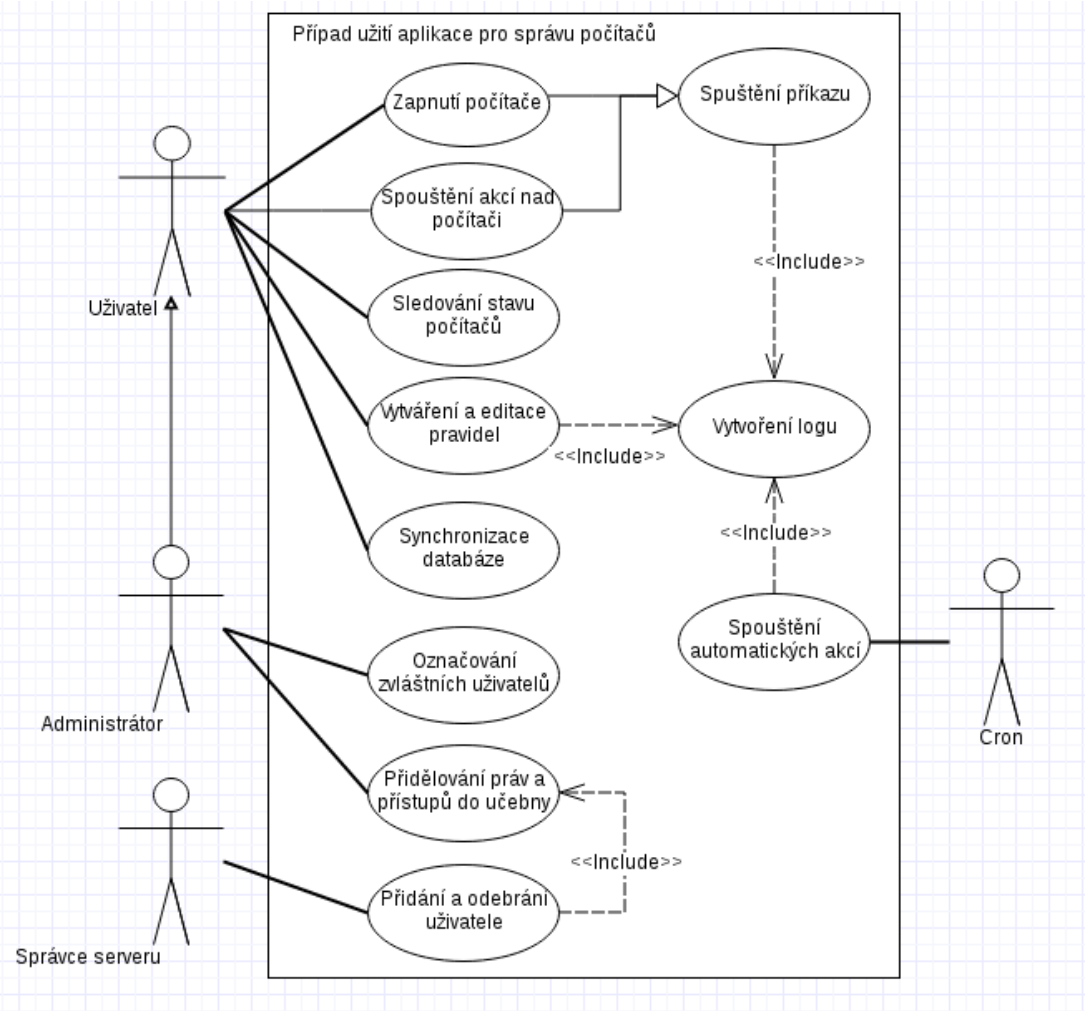
6.6 Use case

V systému se nachází celkem čtyři role:

1. Uživatel - Technik nebo učitel, jejich hlavní činností je spouštění akcí nad počítači, sledování jejich stavu a vytráření pravidel. Při přihlášení navíc synchronizují databázi.
2. Administrátor - Provádí stejné činnosti jako uživatel, jen má automaticky všechna práva a navíc může označovat barvou vybrané loginy přihlášené na počítačích a přidělovat oprávnění ostatním uživatelům aplikace.
3. Správce serveru - Specifická role zodpovědná za prvotní instalaci a správu uživatelů.
4. Cron - Automatizace v prostředí Unix, spouští pravidla.

6.6.1 Diagram

Uživatel s administrátorem mají k dispozici široké spektrum funkcí, tj. zapínání počítačů, provádění příkazů, přihlášení uživatele, atd. Tyto funkce představují různé případy užití, jelikož jich je mnoho s podobným průběhem, lze je shrnout do jedné obecné činnosti, tzv. spouštění příkazů.



Obrázek 4: Případ užití aplikace

6.6.2 Specifikace zapnutí počítače

Popis případu zapnutí vybraných počítačů uživatelem, podobně lze popsat i ostatní akce nad počítači.

Aktéři:

- Uživatel
- Systém

Podmínky spuštění:

- Login uživatele je přidaný správcem do databáze systému
- Uživatel zná svoje přihlašovací údaje
- Správce přidělí uživateli přístup do učebny

- Správce přidělí uživateli právo ke správě napájení
- LDAP a API je funkční

Základní tok:

1. Systém vyzve k přihlášení.
2. Uživatel zadá login a heslo.
3. Systém ověří, zda je login v seznamu povolených, následně autentizuje s LDAP, nakonec oprávnění učeben a pravomoce. Zobrazí seznam učeben, podle pravomocí také funkční tlačítka.
4. Uživatel provede výběr počítačů a stiskne tlačítko zapnout.
5. Systém odešle požadavek zapnutí s výběrem počítačů na API, pošle uživateli zprávu a vytvoří log obsahující jméno uživatele, akci a počítače.

Alternativní toky:
3.A - Login uživatele není ověřen. Admin ho musí přidat na požadavek uživatele.
3.B - Systém nezobrazí požadované učebny. Admin musí přidělí na požadavek.
3.C - Systém nenabídne tlačítko zapnutí. Admin musí přidělit práva.
3.D - LDAP je nefunkční, nejde se tedy přihlásit.
3.E - Nenačte počítače a učebny, přestože má uživ. práva. API je zřejmě nefunkční.
5.A - Systém odešle požadavek na API, ale není potvrzen, zobrazí chybovou zprávu

6.6.3 Specifikace přidělování práv a přístupů do učebny

Aktéři:

- Uživatel
- Systém

Podmínky spuštění:

- Login uživatele je přidaný správcem do databáze systému
- Správce označil uživatele jako admina aplikace (při vytvoření)
- Uživatel zná svoje přihlašovací údaje
- LDAP je funkční

Základní tok:

1. Systém vyzve k přihlášení.
2. Uživatel zadá login a heslo.

3. Systém ověří, zda je login v seznamu povolených, následně autentizuje s LDAP, ověří status admina. Zobrazí kompletní rozhraní a funkcionalitu.
4. Uživatel klikne na ikonu nastavení.
5. Systém zobrazí seznam všech uživatelů, učeben a oprávnění
6. Uživatel vybere konkrétního uživatele ze seznamu
7. Systém zobrazí přístupy a práva tohoto uživatele
8. Uživatel změní přístupy nebo oprávnění a klikne na "Uložit"
9. Systém uloží změny a zobrazuje informace o změně

Alternativní toky:
3.A - Login uživatele není ověřen. Správce ho musí přidat na požadavek uživatele.
3.B - Systém nenabídne ikonu nastavení. Správce musí přidělit práva.
3.C - LDAP je nefunkční, nejde se tedy přihlásit.
5.A - Systém neuloží změny, možná chyba na straně databáze, zobrazí chybu.

7 Návrh řešení

7.1 Uživatelské rozhraní

Grafické rozhraní bude rozděleno do třech logických celků, kterými jsou správa počítačů, správa pravidel a administrace. Výchozí stránka bude pro uživatele v pohledu na správu počítačů.

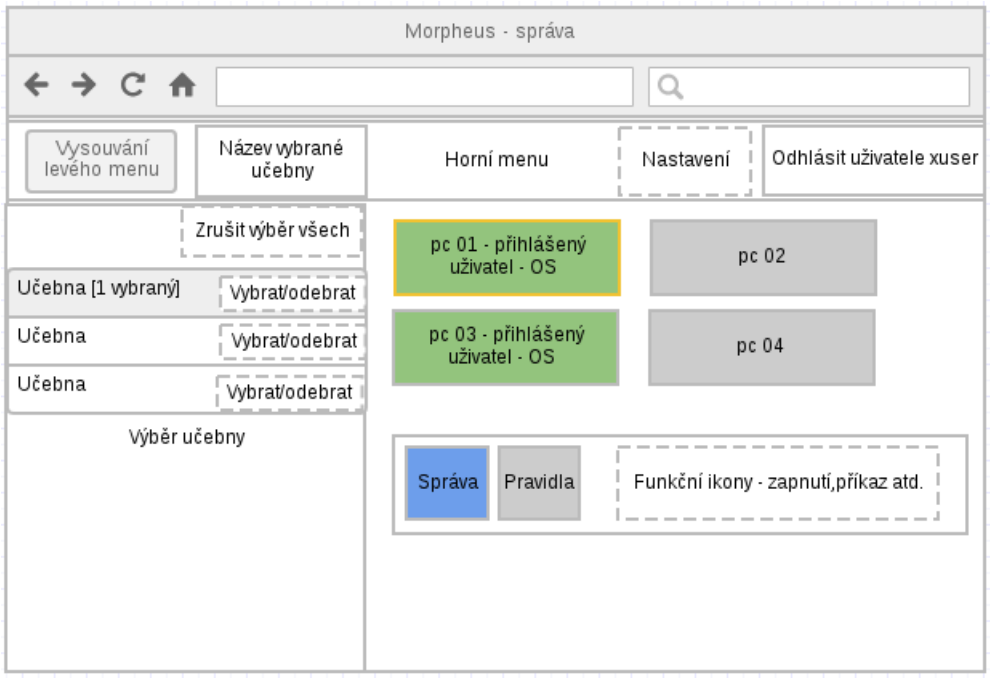
7.1.1 Layout pro správu počítačů

Layout pro správu zahrnuje tyto bloky:

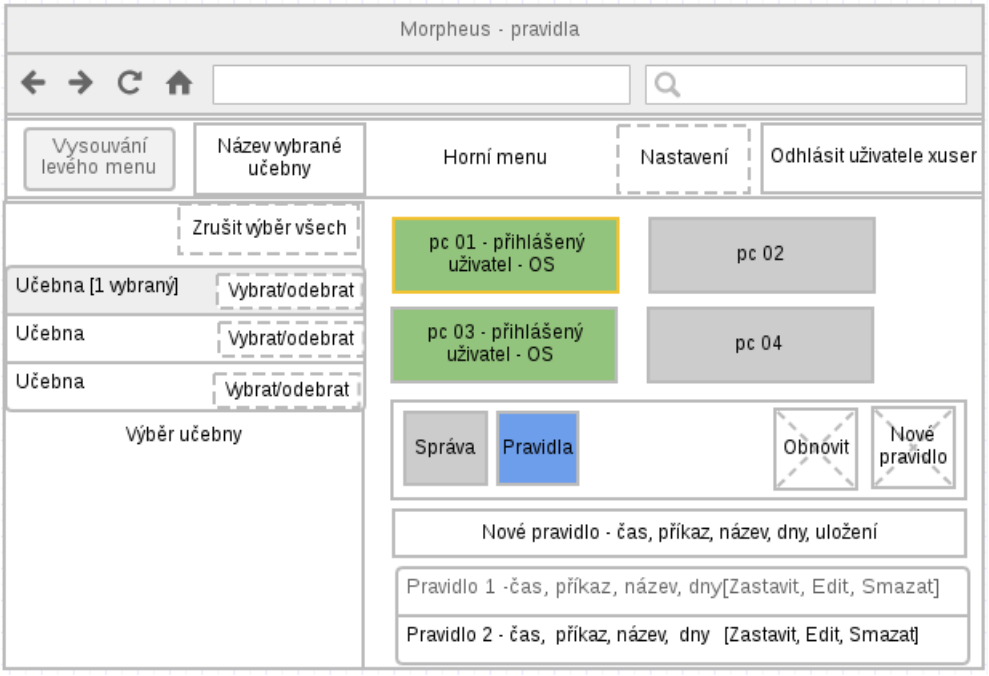
1. Horní menu - obsahuje tlačítko pro skytí/zobrazení panelu pro výběr učeben, název aktuálně vybrané učebny, odkaz do administrace a odhlášení uživatele.
2. Levý panel výběru učeben - tvoří ho seznam skupin počítačů (učeben). Kliknutí na učebnu zobrazí počítače učebny v hlavním okně. Panel lze tlačítkem skrýt, nebo zobrazit, avšak jeho skrytí je ovlivněno i automaticky při zmenšení okna pod jistou hodnotu, tj. typicky mobilní zařízení nebo malé monitory, kde ušetří místo na obrazovce. Sekundární účel panelu je označení/odznačení počítačů učebny, u každé je ovládací prvek pro výběr všech počítačů a tlačítko pro zrušení výběru všech počítačů. Název učebny zároveň signalizuje graficky a číslem, zda jsou některé počítače (a kolik) v učebně označeny. V případě označení počítačů z více učeben (výběry i mezi učebnami jsou trvale označeny až do provedení akce) lze všechen výběr jednoduše zrušit prvkem nahoře v tomto panelu.
3. Hlavní okno s výčtem počítačů - jednotlivé počítače zde budou mít tvar obdelníku, jehož šířka bude záviset na velikosti okna prohlížeče. K tomu poslouží CSS Bootstrapu se systémem mřížky, na desktopu bude kompletní reálné rozložení podle souřadnic a v mobilním zařízení pouze dva sloupce bez rozložení. Okno počítače bude obsahovat číslo, přihlášeného uživatele a ikonu nabootovaného systému. Kliknutím na obdelník dojde ke zvýraznění/odznačení obdelníku a přidání/odebrání výběru. Barva obdelníků se mění dle aktuálního stavu (zapnuto, vypnuto, vybráno, chyba).
4. Ovládací panel - nachází se pod výčtem počítačů, jeho šiřka se vždy přizpůsobuje počtu počítačů. V první části se nachází přepínač pohledů (kontextů). Druhá část mění svůj obsah dle kontextu, nachází se zde příslušná funkční tlačítka.

7.1.2 Přepínání kontextu

Přepínač v ovládacím panelu slouží k přepínání rozhraní na kontext správy počítačů a kontext správy pravidel. V podstatě umožní dívat se na stejnou věc (počítače a učebny) z jiných pohledů (spouštění funkcí ručně a automaticky). Výhoda je v tom, že skoro všechny prvky zastávají pořád stejnou funkci beze změn, včetně výběru počítačů.



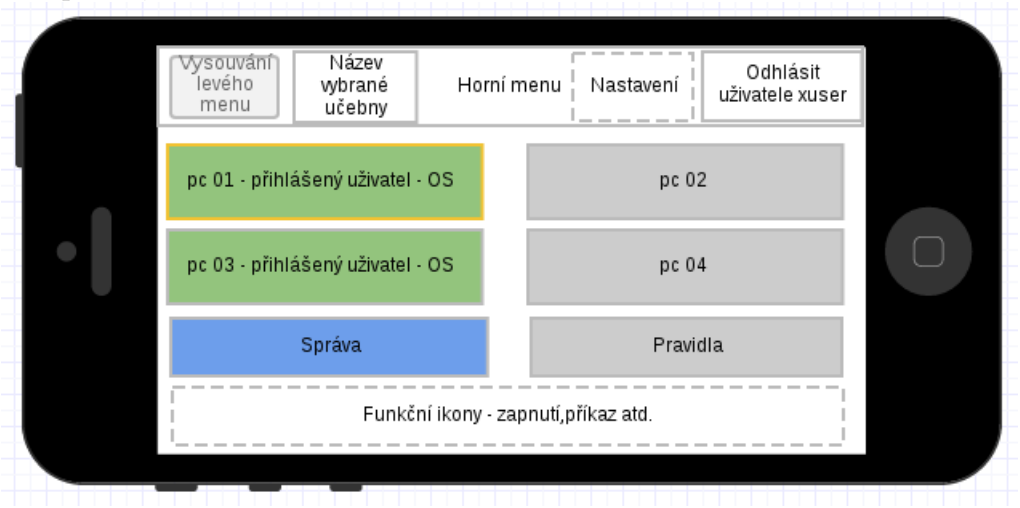
Obrázek 5: Drátěný model kontextu správy počítačů



Obrázek 6: Drátěný model kontextu správy pravidel

Kontext správy počítačů slouží k ručnímu spouštětí akcí nad vybranými počítači. V ovládacím panelu se proto nachází tlačítka s ikonou vypnout, zapnout, poslat příkaz, přihlásit uživatele, atd.

Kontext správy pravidel zobrazuje v ovládacím panelu ikony obnovit pravidla a nové pravidlo. Po kliknutí na nové pravidlo se zobrazí panel pro vložení, kde se nachází textbox pro zadání názvu, input pro výběr specifického času, dropdown list s výběrem příslušné akce, pole ikon pro výběr dne (příp. dnů v týdnu), ve kterém se má dané pravidlo provádět a tlačítko Uložit. Po jeho stisku se pravidlo přidá do seznamu vytvořených pravidel pod ovládací panel. Daná akce se bude provádět na těch počítačích, jejichž obdelníky byly při stisku tlačítka uložit pravidlo zařazené do výběru. Vedle přidaného pravidla se pak objeví několik dalších ovládacích prvků - tlačítka pauza, smazat a editovat.



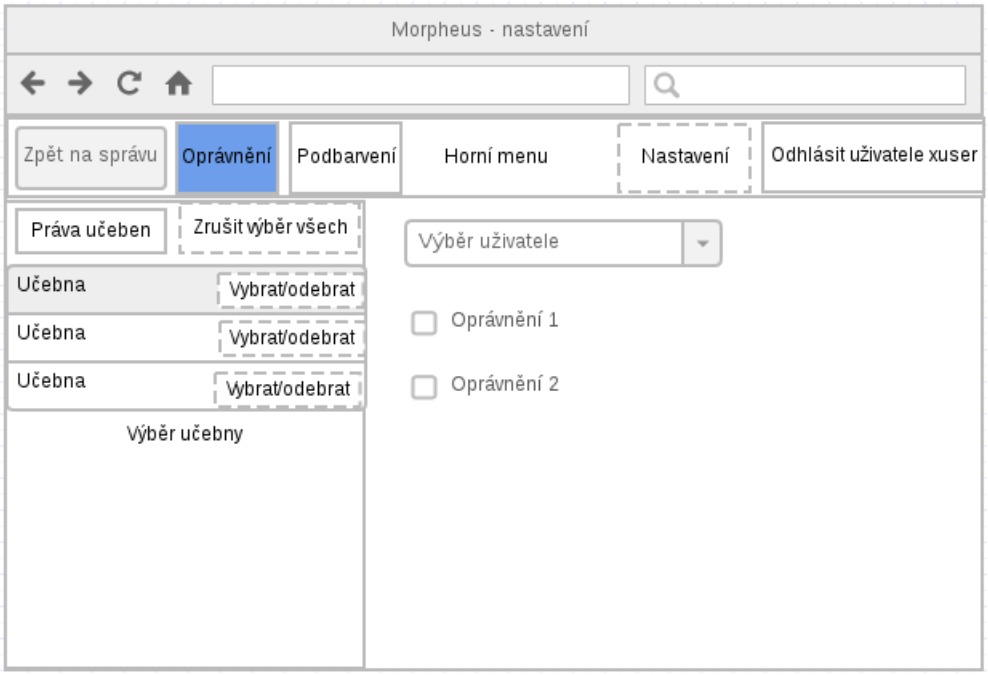
Obrázek 7: Drátěný model mobilního layoutu

7.1.3 Layout administrace

Administrační rozhraní se skládá ze sekcí management oprávnění a obarvování specifických uživatelů, tato nabídka je obsažena v horním menu.

Sekce oprávnění bude nabízet v hlavním okně výběr uživatele, po výběru se pod ním zobrazí v checkboxech jeho konkrétní oprávění. Opět se zde využije levý panel výběru učeben, který je rozebírán výše, jen s menší úpravou bude sloužit pro výběr učeben (bez ukazatele počtu počítačů), do kterých má konkrétní uživatel právo vstupu. Poslední prvek je tlačítko pro potvrzení změn a uložení nastavení aktuálního uživatele.

Obarvování uživatelů má v podstatě jednoduchý layout, pouze horní menu a hlavní okno s barevnými boxy pro vkládání a odebírání loginů.



Obrázek 8: Drátěný model managementu oprávnění

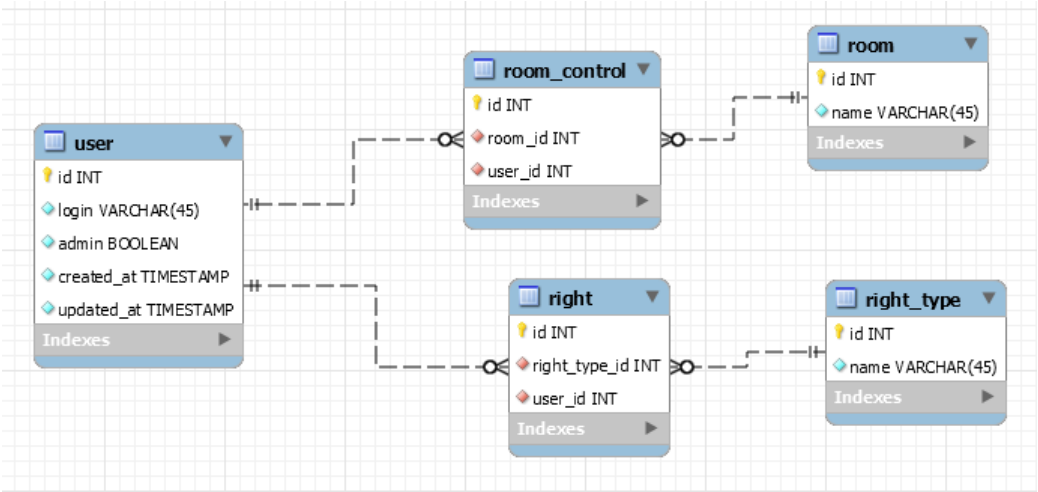
7.2 Rozvržení back-endu

Správa počítačů a pravidel bude prováděna na jedné fyzické stránce, tedy prvotní načtení stránky vykreslí učebny a počítače na jejich správných místech (pouze s názvem a číslem). Poté se načte JavaScriptový soubor, který AJAXovým požadavkem na server zajistí stažení a zobrazení stavu počítačů té učebny, ve které se zrovna uživatel nachází. Nadále bude JavaScript přepínat učebny, kontext aplikace zmíněný výše a poslouží i při tvorbě výběrů, kliknutí na funkční tlačítko i editaci pravidel.

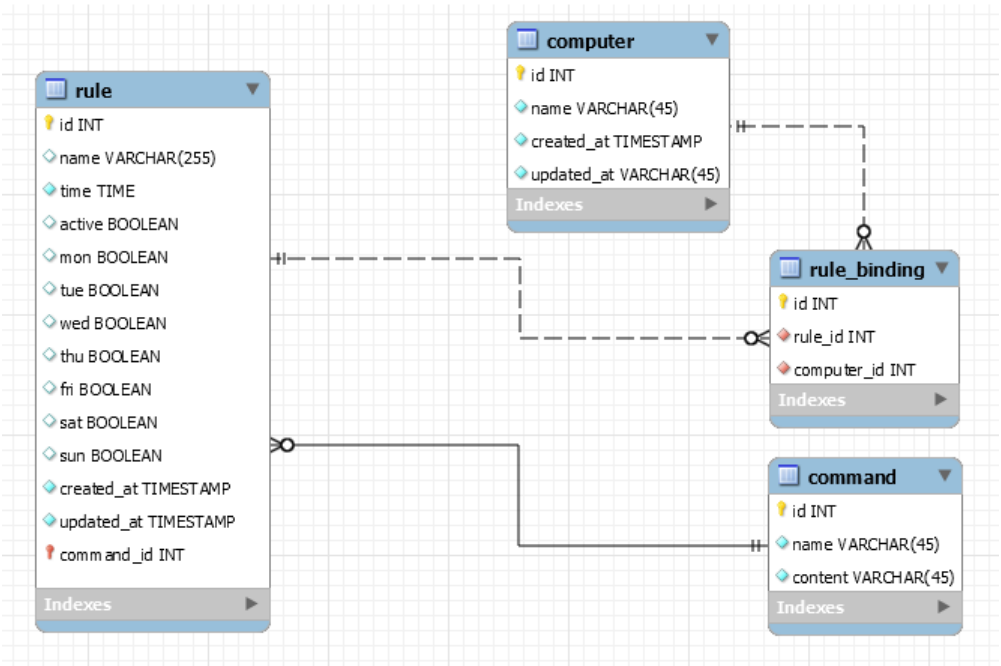
Rozvržení kontrolerů bude následující:

1. Login - implementace autentizace, synchronizace databáze
2. Grid - generuje počítače a učebny dle oprávnění uživatele (prvotní načtení)
3. Computer - generuje stav počítače a obsluhuje funkční tlačítka
4. Rule - generuje, přidává, upravuje automatické akce
5. Admin - spravuje editaci oprávnění a přístupu, obarvení speciálních uživatelů

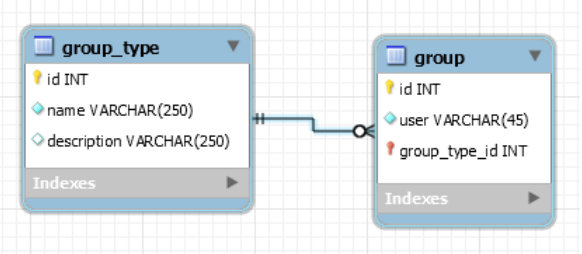
7.3 ER diagram



Obrázek 9: Model oprávění a přístupu uživatele



Obrázek 10: Model automatických pravidel



Obrázek 11: Model rozlišení speciálních uživatelů

8 Implementace

8.1 Responzivní layout

Layout je vytvořený pro maximální kompatibilitu na všech typech zařízení, využívá zejména CSS media query a pozicování, Bootstrapovou mřížku a v menší míře i JavaScript. Rozložení se mění v závislosti na šířce okna prohlížeče v bodech, náhledy aplikace, viz. přílohy. Body zalomení jsou následující:

1. od 1300px - jsou zobrazeny všechny grafické prvky
2. od 990px do 1300px - skryje se levý panel pro výběr učeben
3. od 770px do 990px - mění se rozložení počítačů na dvousloupcové

Horní panel má nastaven CSS parametr position na *fixed*, takže při mobilním prohlížení se drží stále nahoře. Další zajímavý prvek je levý panel s možností vysouvání, který je založen na pokročilém Bootstrapovém prvku (Startbootstrap, 2016) s množstvím dodatečných úprav.

Důležitá je implementace mřížky, jejich počet se odvíjí od počtu učeben. Při generování Blade šablony vytvoříme iterací daný počet mřížek s jejich konkrétními počítači. Mřížky mají na počátku nastavený parametr display na *none*, který ji zneviditelní. Ukázka viditelné a skryté mřížky:

```
<div id="Q04" data-width="5" data-nazev="Q04" class="row">
   <div class="col-md-1 col-xs-6 hidden-xs hidden-sm ">
      <div class="styl-vypnuteho-pc"></div>
   </div>
   <div class="col-md-1 col-xs-6">
      <div class="styl-zapnuteho-pc">...</div>
   </div>
</div>
<div id="Q05" data-width="5" data-nazev="Q05" class="row"
style="display: none;">
   <div class="col-md-1 col-xs-6">
      <div class="pocitac-run" data-computer="Q05"
      data-nazev="prof11" data-target="11">
         <h4>11<i class="fa fa-linux"></i></h4>
         <div class="user-any"></div>
      </div>
   </div>
   <div class="col-md-1 col-xs-6">...</div>
</div>
```

První mřížka *#Q04* je jako jediná viditelná (došlo tedy ke kliknutí na učebnu v levém panelu). Mřížka obsahuje dva prvky (políčka), první je pouze prázdná bílá plocha, druhý je už počítač. Oba mají Bootstrapové třídy pro určení počtu zobrazených

dílků (z celkových dvanácti) na každém typu zařízení. Na obrazovce typu medium a výš bude stále počítač jeden dílek, na obrazovce extra small a small už je to šest (polovina obrazovky), navíc zmizí prázdná políčka.

Druhá mřížka učebny Q05 je skryta a obsahuje dvě políčka s počítači, první z nich prezentuje konkrétní implementaci zapnutého počítače. Prvek třídy *.pocitac-run* má další potřebné data atributy pro identifikaci po kliknutí myší, dále obsahuje nadpis vlastnící číslo a ikonu a další prvek pro zobrazení uživatele.

8.2 Vytvoření databáze

Implementace vytvoření struktury databáze je nezávislá na nasazené technologii databáze. Je toho dosaženo díky systému migrací, který nastavením jediného parametru *DB_CONNECTION* v souboru *.env* umí převést obecný předpis struktury do konkrétní vybrané technologie. V našem případě je nastavena hodnota na *pgsql*, tedy PostgreSQL.

Definice struktury se nachází v tzv. migracích, to jsou PHP třídy se speciální notací danou frameworkem. Každá tabulka databáze má tedy svoji migraci. Příklad vytvoření migrace pro tabulku Rule:

```
class CreateRuleTable extends Migration
{
    public function up()
    {
        Schema::create('rule',function(Blueprint $table){
            $table->increments('id');
            $table->string('name');
            $table->time('time');
            $table->integer('command_id');
            $table->foreign('command_id')->references('id')
            ->on('command')->onDelete('cascade');
            $table->boolean('active');
            $table->boolean('Mon');
            $table->boolean('Tue');
            $table->boolean('Wed');
            $table->boolean('Thu');
            $table->boolean('Fri');
            $table->boolean('Sat');
            $table->boolean('Sun');
            $table->timestamps();
        });
    }
}
```

Jestliže máme vytvořené migrace pro všechny tabulky ve správném pořadí (kvůli integritě klíčů), stačí pouze příkazem *php artisan migrate* spustit proces migrování do nastaveného databázového systému.

Nyní máme strukturu hotovou, ale často je potřeba doplnit defaultní záznamy do některých tabulek, jako například přidání výchozího off-line administrátora. Pro tento účel slouží tzv. seedování databáze, seed je také PHP třída s vlastní notací. Ukázka vypadá nasledovně:

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(GroupTableSeeder::class);
        $this->call(RightTypeSeeder::class);
        $this->call(UserTableSeed::class);
        $this->call(CommandSeeder::class);
    }
}

class UserTableSeed extends Seeder
{
    public function run()
    {
        DB::table('user')->insert([
            'login' => 'off-line-admin',
            'admin' => true,
            ]);
    }
}
```

První třída zajišťuje volání vybraných seedů, druhá třída už je konkrétní implementace seedu. Spuštění seedování následuje po vytvoření struktury, a to příkazem *php artisan db:seed*.

8.3 Vykreslení počítačů dle souřadnic

Pro splnění požadavku vykreslení dle fyzického rozložení v učebně je třeba vhodně zpracovat získané souřadnice a další parametry, jako je název a přezdívka učebny nebo maximální počet počítačů na šířku. Multidimenzionální pole je ideální volba pro uchování těchto parametrů v podobě vícerozměrné struktury, kterou po vygenerování v PHP rozbalíme v Blade šabloně. Všechna tato data využijeme pro naplnění levého seznamu učeben a Bootstrapové struktury mřížky počítačů.

Implementace vytvoření pole s počítači učeben v pseudokódu:

```
IF uživatel == admin THEN
    učebny = stáhni seznam všech učeben
ELSE
    učebny = povolené učebny uživatele z databáze
ENDIF
FOR učebny
    počítače = stáhni seznam počítačů učebny
    parametry = stáhni parametry počítačů
    IF NOT EMPTY parametry THEN
        přezdívka_učebny = podřetězec prvního počítače
        FOR parametry
            názevPC = parametr[nazev]
            pozicePC = ((parametr[poz_y] * 12) + parametr[poz_x]) -13
            mřížka[pozicePC][čísloPC] = podřetězec(názevPC)
            mřížka[pozicePC][učebna] = učebna
            mřížka[pozicePC][název] = názevPC
            IF parametr[poz_x] > maxřádek THEN
                maxřádek = parametr[poz_x]
            ENDIF
        ENDFOR
        PUSH(mřížka,maxřádek,název_učebny,přezdívka_učebny) TO mřížky
    ENDIF
ENDFOR
RETURN mřížky
```

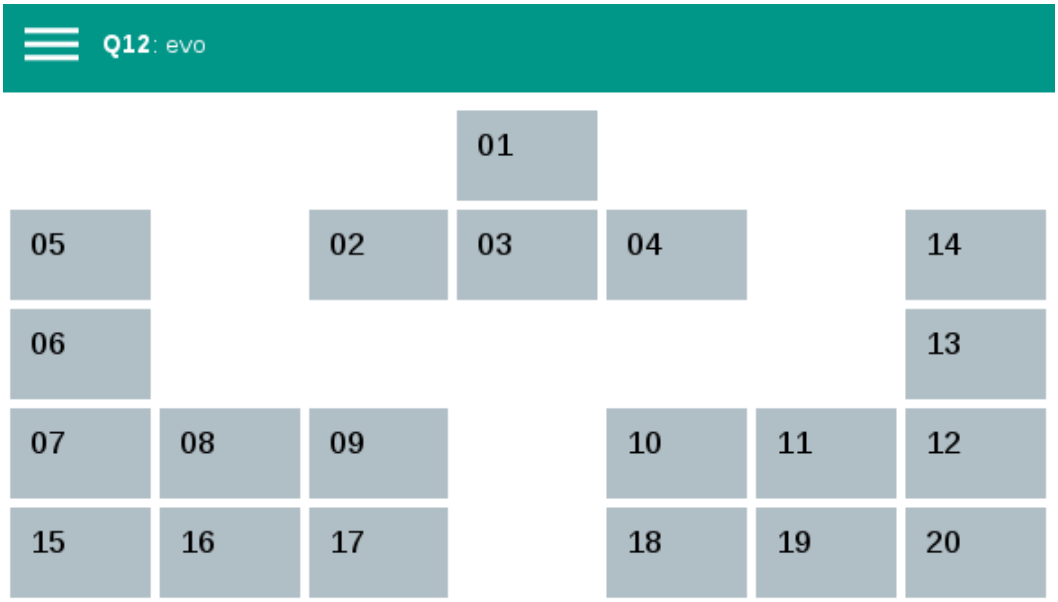
Na začátku zjistíme, zda je přihlášený admin nebo uživatel. V případě admina načteme všechny názvy učeben, uživatel načítá z databáze pouze svoje přístupné. Vznikne seznam učeben, se kterými pracujeme v dalších krocích.

Pro každou učebnu načítáme seznam počítačů a dále zpracujeme parametry počítačů. Každý počítač zařadíme na přesně dané místo do vektoru pomocí transformace souřadnic. Transformací je myšleno převést souřadnice počítačů, tedy z 2D prostoru do klasického jednorozměrného vektoru.

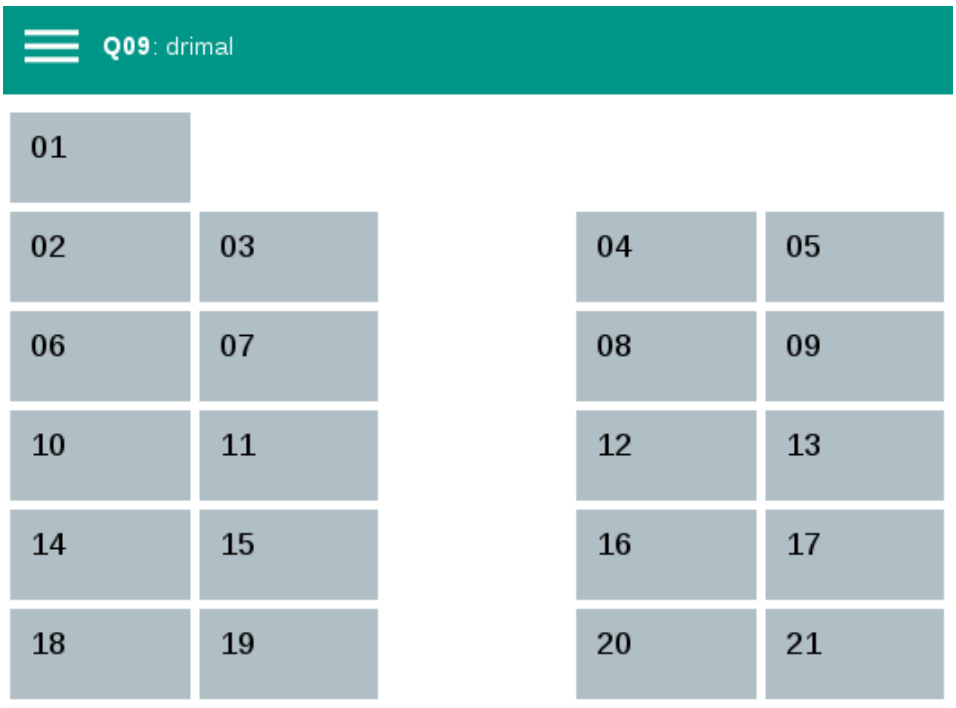
Výsledný vektor je rozdělený na pomyslné subvektory o délce dvanáct prvků, takže ho můžu dopředu inicializovat, každý dvanáctý prvek bude mít příznak (klíč v poli) zalomení řádku. Pokud známe šířku subvektorů, neboli pomyslnou šířku matice, lze souřadnice převést dle vztahu: $S = poz\_y * 12 + poz\_x$ - *13*. Buňka v poli na výsledné souřadnici S bude obsahovat název, číslo počítače a nadřazenou učebnu. Mimojiné v každé učebně zjišťujeme maximální souřadnici X pro zarovnání ovládácího panelu grafiky.

Na straně šablony se v každé učebně vykreslí okna (počítače) CSS stylem dle klíčů pole, a to i včetně prázdných oken, přiložený příznak dvanáctého prvku vloží

třídu *clearfix* do HTML, kde vznikne zalomení řádku a prakticky dodrží konvenci počtu sloupců Bootstrapu.



Obrázek 12: Ukázka výstupu pro učebnu Q12



Obrázek 13: Ukázka výstupu pro učebnu Q09

8.4 Zobrazení stavu počítače

Druhá fáze, která nastává ihned po sestavení stránky s počítači je zobrazení informací u každého stroje, tj. stav napájení, nabootovaný systém, přihlášený uživatel, skupina uživatele, chybový stav. Při kliknutí na učebnu v levém panelu se zavolá metoda *pullInfo(učebna)*, která AJAXovým požadavkem na back-end získá tato data ve formátu JSON a pomocí JavaScriptu je zobrazí na daných počítačích. Na úrovni back-endu je důležité znát výstupy API funkcí, konkrétně hlavně nabootovaný systém a stav napájení. Možné stavy jsou následující:

Nabootovaný systém:

- 1 = windows
- 2 = unix
- 3 = amt (speciální případ)

Napájení systému:

- 0 = zapnuto
- 1 = vypnuto
- 2 = error

Vyjímka z těchto pravidel:

- KDYŽ napájení = 0 A systém = 3 TAK napájení = 1

Pseudokód implementace metody computerInfo v kontroleru ComputerController - vytoření struktury pro předání informací o počítačích:

```
počítače = stáhni seznam počítačů učebny
parametry = stáhni parametry počítačů
FOR parametry # var i- iterace
   dostupnost = parametr[i][dostupnost]
   system = parametr[i][system]
  IF (NOT (dostupnost AND system ==3) OR dostupnost OR dostupnost = 2) THEN
      pocitace[i][dostupnost] = vypnuto
      pocitace[i][system] = zadny
      pocitace[i][prihlaseny] = nikdo
   ELSE IF dostupnost == 0 THEN
      pocitace[i][dostupnost] = zapnuto
      IF system == 1 THEN
         pocitace[i][system] = windows
      ELSE IF system == 2 THEN
         pocitace[i][system] = linux
      ELSE
```

```
         pocitace[i][system] = zadny
      ENDIF
      IF (NOT NULL parametr[i][prihlaseny] AND LENGTH < 12) THEN
         pocitace[i][prihlaseny] = parametr[i][prihlaseny]
         pocitace[i][skupina_uživatele] = zjisti označení uživatele
      ELSE
         pocitace[i][prihlaseny] = nikdo
      ENDIF
   ENDIF
   pocitace[i][nazev] = parametry[i][nazev]
   IF (zkontroluj vybrané parametry == TRUE) THEN
      pocitace[i][dostupnost] = error
   ENDIF
ENDFOR
RETURN pocitace
```

Algoritmus opět využívá multidimenzionální pole a klíče hodnot. Na začátku se stáhne seznam počítačů a jejich podrobnosti, cyklem procházíme všechny počítače a postupně zpracujeme potřebné hodnoty z klíčů. Větvení je rozděleno na dvě hlavní části, tj. dostupnost je negativníb nebo pozitivní. V případě dostupnosti zjistíme nabootovaný systém, dále zda je přihlášený uživatel a pomocnou funkcí jeho označení barvou (dotazem na databázi). Poslední záležitost je volání funkce na kontrolu chybového stavu.

Pro indikaci chybového stavu některého z parametrů počítače slouží v grafickém rozhraní vykreslení oranžovou barvou (CSS styl *pocitac-error*). Metoda pro kontrolu tohoto stavu má defaultně dva parametry - autologin počítače a přihlášený uživatel. V této funkci vyhodnocujeme tyto vstupy, zda jsou korektní a pokud nejsou, tak vrací true a počítač se označí oranžově. Je zde prostor pro kontrolu dalších parametrů, stačí přidat formální parametr a při volání funkce dosadit skutečný ve tvaru: *$result[$i]["parametr počítače"]*. V implementaci funkce už zbývá jen definovat chybové hodnoty a návratovou hodnotu true.

Obrázek 14: Ukázka výstupu pro učebnu Q05

8.5 Tvorba výběrů

Výběry provádí uživatel klikáním na počítače a to lze i napříč učebnami. Aktivní počítač je znázorněn tmavší barvou, všechny výběry jsou v levém panelu zvýrazněny číslem, které uvádí počet vybraných počítačů u každé učebny. Tato funkcionalita je plně v kompetenci JavaScriptu. Základem je pole, které se definuje prázdné po načtení stránky a při kliknutí na počítač se do něj přidávájí a odebírají objekty s názvem a učebnou počítače. Implementace pomocí jQuery je následující:

```
var pc_selected;
$(document).ready(function () {
pc_selected = [];
$("div[data-computer]").click(function () {
    if ($(this).hasClass("selected")) {
        $(this).removeClass("selected");
        unSelectPc($(this));
    } else {
        $(this).addClass("selected");
        selectPc($(this));
    }
    updatePocet();
    });
}
```

```
function selectPc(pc) {
    var computer = {};
    computer.name = pc.data('name');
    computer.room = pc.data('room');
    pc_selected.push(computer);
}
function unSelectPc(pc) {
    var last_index = pc_selected.length - 1;
    var computer = pc.data('name');
    for (var i= 0; i< pc_selected.length; i++) {
        if (computer === pc_selected[i].name) {
            pc_selected[i] = pc_selected[last_index];
            pc_selected.pop();
            break;
        }
    }
}
```

Po kliknutí na HTML prvek *div* s parametrem *data-computer* (tedy počítač) se zjistí přítomnost CSS třídy *selected*. V případě nalezení se třída odebere a zavolá funkci pro odebrání z vektoru. V opačném případě při nenalezení třídy *selected* se tato třída přidá a zavolá se metoda pro přidání do vektoru. Oba případy získávají informace z data atributů tohoto HTML prvku, tj. jeho jméno a učebnu.

Důležité je také umístění deklarace proměnné *pc_selected* hned na začátku souboru před načtením dokumentu, protože tato proměnná je globální a lze k ní přistoupit i z jiných .JS souborů. Tato vlastnost je využita při implementaci pravidel, kdy z jiného souboru přistupuji k této proměnné při přiřazení počítačů k pravidlu.

Po přidání nebo odebrání je nakonec volána funkce pro aktualizaci signalizace v levém panelu, její implementace je následující:

```
function updatePocet() {
    $(".sidebar-nav li div").each(function () {
       var class = $(this).data('nick');
      var count = $(".selected[data-computer=" + class + "]").length;
       $(this).children().text(count);
       if (count > 0) {
           $(this).css("font-weight", "bold");
       } else
       {
           $(this).css("font-weight", "normal");
       }
    });
}
```

Je založena na průchodu všech učeben v levém panelu (HTML prvky *li*), každá iterace určí počet prvků s data atributem dané učebny a aktualizuje tak čítač a tučnost zvýraznění textu panelu.

Výběr není nutné tvořit klikáním na jednotlivé počítače, při větších selekcích lze rychle vybrat počítače učeben, při najetí myší na příslušnou položku menu se objeví ikony pro označení a odznačení počítačů v příslušné učebně. Další možnost je kliknutí na ikonu v horní části levého menu, která slouží pro označení a odznačení úplně všech počítačů. Tyto dva postupy se můžou mezi sebou kombinovat a tak rychle vytvořit požadovaný výběr. Implementace těchto rychlých výběrů je založena na výše uvedených principech.

Obrázek 15: Ukázka multivýběru počítačů

8.6 Odeslání příkazu zapnutí

Volba provedení akce nad počítači se nachází v ovládacím panelu ve formě odpovídajících ikon. Akce jsou uspořádány následovně: zapnout, vypnout, obnovit stav, režim přijímačky, přihlášení uživatele, poslat příkaz, smazat dočasná data.

Obrázek 16: Ovládací panel s akcemi

Princip implementace je u všech v zásadě stejný, proto si uvedeme příklad řešení odeslání příkazu zapnutí.

První krok je přidání ikony do ovládacího panelu s CSS identifikátorem, který určuje typ akce, tj. *#power_on*. Následuje svázání jQuery eventu po kliknutí na ikonu:

```
$('#power_on').click(function () {
    changePowerState($(this).attr('id'));
});
```

Volaná funkce vytvoří AJAXový požadavek na back-end, obsahuje příkaz (*power_off*, nebo *power_on*), počítače a jejich učebnu. Počítače a učebny se získají z již výše uvedeného pole *pc_selected* v podobě dvou oddělených vektorů. Informace o učebně potřebujeme zejména kvůli vytvoření logu. V případě úspěchu získání dat se zajistí smazání aktuálního výběru a vykreslí se zpráva uživateli o provedené akci. Implementace této funkce:

```
function changePowerState(command) {
    var computers = divideArray(pc_selected, 'name');
    var classes = divideArray(pc_selected, 'room');
    $.ajax({
        dataType: "json",
        url: "change_power_state",
        method: "POST",
        data: {
            computers: computers,
            classes: classes,
            command: command
        },
        success: function (data) {
            changeActiveAll('', 'unselect_all');
            flashSuccess(data.msg);
        },
        error: function (data) {
            flashError(data);
        }
    });
}
```

Implementace back-endu v pseudokódu:

```
IF uživatel má právo na změnu napájení THEN
   IF NOT EMPTY computers AND command == power_on THEN
      odešli požadavek zapnutí s počítači na API
      vytvoř log (počítače,učebna,akce,uživatel)
      RETURN JSON stroje se zapínají, status 200
```

```
   ELSE IF NOT EMPTY computers AND command == power_off THEN
      odešli požadavek vypnutí s počítači na API
      vytvoř log (počítače,učebna,akce,uživatel)
      RETURN JSON stroje se vypínají, status 200
   ELSE IF EMPTY computers THEN
      RETURN JSON nejsou vybrané stroje, status 400
   ELSE
      RETURN JSON někde je chyba, status 400
   ENDIF
ELSE RETURN JSON nemáte oprávnění, status 403
ENDIF
```

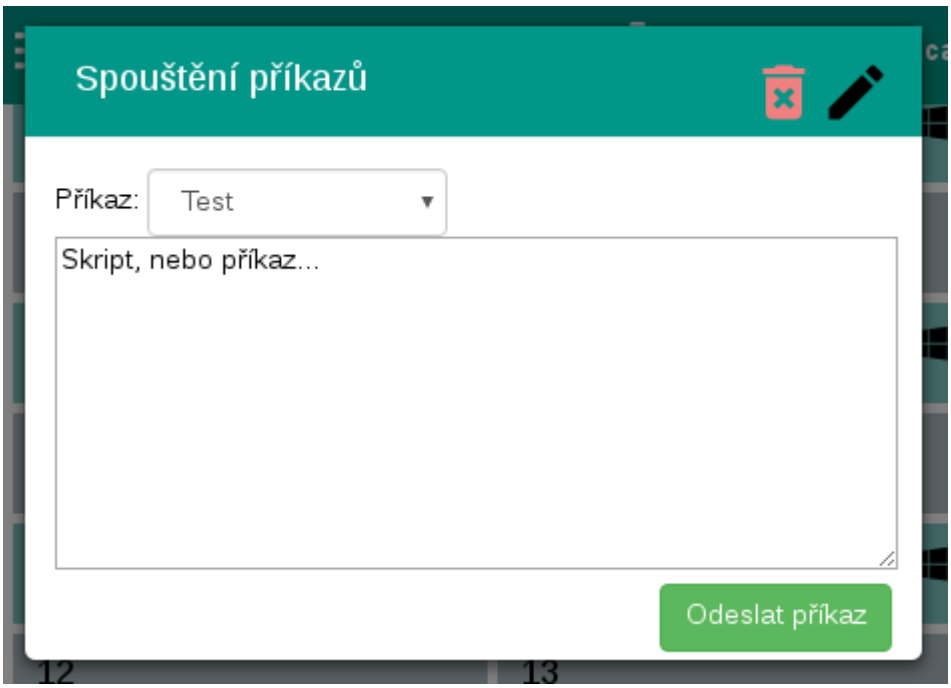
Každé volání akce nad počítači podléhá kontrole oprávnění uživatele. Návratovou hodnotou této funkce je HTTP odpověd obsahující JSON a zadaný status kód.

8.7 Automatické spouštění akcí

V defaultním stavu jsou k dispozici pouze akce vypnout a zapnout počítače, ale lze si definovat i libovolný příkaz v tzv. rozšířeném režimu. Rozšířený režim znamená, že máme k dispozici trvalý seznam příkazů, který si může pověřená osoba editovat dle svého uvážení. Seznam slouží pro opakovatelné využití uložených příkazů, které jsou reprezentovány svými výstižnými názvy. Příkazy se uplatní jak v manuálním režimu, tak v automatickém.

8.7.1 Vytvoření vlastního příkazu

Seznam příkazů se nachází v kontextu správy počítačů pod ikonou poslat příkaz. Kliknutím zobrazíme modální okno se seznamem uložených příkazů a editačním oknem. V pravém horním rohu se nachází tlačítko pro přidání nového příkazu. Pro odeslání vybraného, nebo vytvořeného příkazu manuálním způsobem můžeme kliknout na odeslat příkaz, pro automatický způsob pouze uložíme a přesuneme se do části vytvoření akce, viz. další podkapitolu.

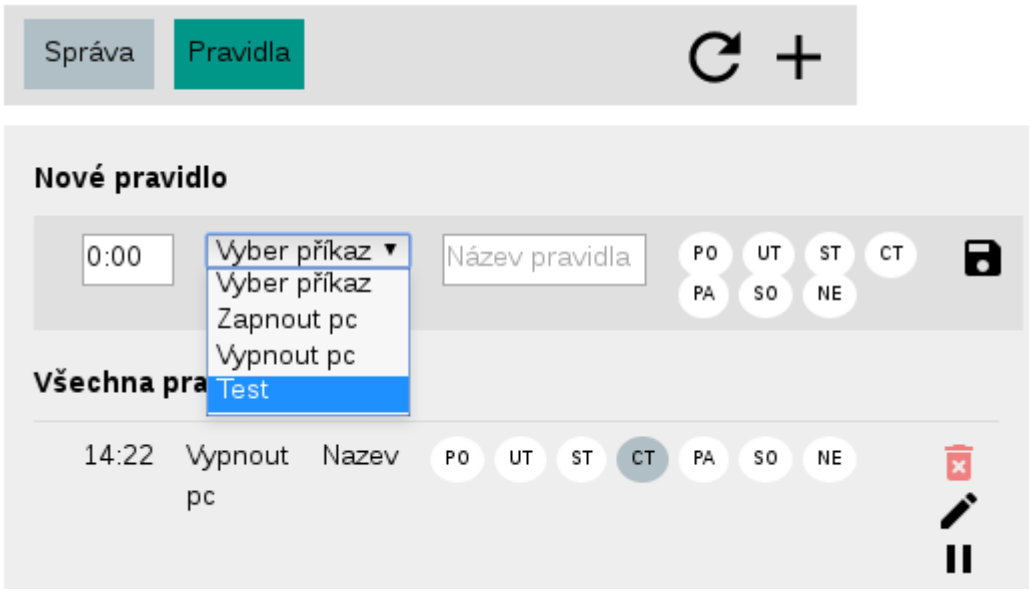


Obrázek 17: Modální okno s editací příkazu

Implementace této části je v podstatě jednoduchá, po kliknutí na ikonu zobrazíme pomocí Bootstrapu modální okno a AJAXem stáhneme z back-endu seznam příkazů, jímž naplníme HTML prvek výběru, stejně tak vytvoření pouze odešle požadavek na přidání záznamu nového příkazu do databáze.

8.7.2 Vytvoření akce

Jestliže máme vytvořený požadovaný příkaz nebo chceme využít ty defaultní, tak můžeme přistoupit k vytvoření akce. Seznam s akcemi najdeme v kontextu správa pravidel, ovládací panel v tuto chvíli obsahuje tlačítka pro aktualizaci stavu pravidel a vytvoření nového pravidla. Kliknutí na tlačítko nové pravidlo aktivuje jQuery event, který zviditelní HTML prvek s formulářem pro vyplnění údajů, mezi kterými je i seznam s našimi vytvořenými příkazy z minulého kroku.



Obrázek 18: Přidání nové akce

Implementace uložení nové akce je založena opět na AJAXovém požadavku, jeho parametry jsou hodnoty získané z HTML prvků formuláře. Vytvoření požadavku vypadá následovně:

```
$('#save_pravidlo').click(function () {
    addRule($(this));
});

function addRule(rule) {
    var parent = rule.parent().parent();
    var time = parent.find('input[name=kdy]').val();
    var command = parent.find('select[name=command] option:selected')
    .attr('data-id');
    var name = parent.find('input[name=name]').val();
    var days = [];
    parent.find('div[data-date]').each(function () {
        if ($(this).hasClass('date_active')) {
            days.push(1);
        } else {
            days.push(0);
        }
    });
    if (command > 0) {
        computers = divideArray(pc_selected, 'name');
        classes = divideArray(pc_selected, 'room');
        $.ajax({
            dataType: 'json',
```

```
            url: 'rules/addrule',
            method: 'POST',
            data: {
                time: time,
                command_id: command,
                name: name,
                days: days,
                computers: computers,
                classes: classes
            },
            success: function (rules) {
                makeRule(rules.rules[0]);
                $('.nove_pravidlo').hide();
                changeActiveAll('', 'unselect_all');
            },
            error: function (xhr, msg) {
                flashError(msg);
            }
        });
    }
}
```

Hodnoty z HTML prvků získáme traverzováním nad DOM strukturou, stejně také dny v týdnu mají data atributy *data-date*, které všechny iterujeme a dle přítomnosti třídy značící aktivní výběr vytvoříme pole *days* s logickými hodnotami. Následuje ověření vybraného příkazu, pokud je platný, tak dojde k získání dvou vektorů s vybranými počítači a učebnami z již zmíněné globální proměnné *pc_selected*. V případě úspěšnému uložení do databáze vrací back-end pro ověření data o této akci, která zobrazíme v seznamu dostupných pravidel pomocí JS funkce. Nakonec dojde ke skrytí formuláře a ke zrušení výběru.

Pseudokód zpracování požadavku v back-endu:

```
IF uživatel má právo na změnu pravidel THEN
   vytvoř pravidlo(name,time,active=true,command_id,
   mon=days[0],tue=days[1],wed...)
   IF NOT EMPTY computers THEN
      error = FALSE
      FOREACH computers
         pc_id = computer id z databáze
         vytvoř záznam v tab. RuleBinding(id_pravidla,pc_id)
         IF záznam neuložen THEN
            error = TRUE
         ENDIF
      ENDFOR
```

```
   ENDIF
   IF pravidlo vytvořeno A error == FALSE THEN
      vytvoř log(počítače,učebny,pravidlo,uživatel)
      pravidlo = vyber pravidlo znovu z databáze
      #přidej pravidlu informace pro vykreslení
      IF pravidlo[active] = TRUE THEN
         pravidlo[tlačítko_pozastavit] = pozastavit
         pravidlo[stav] = pozastaveno
      ELSE
         pravidlo[tlačítko_pozastavit] = aktivovat
         pravidlo[stav] = běží
      ENDIF
      pravidlo[příkaz] = název příkazu z databáze
      FOR pravidlo[mon..sun]
         IF pravidlo[mon] == TRUE THEN
            pravidlo[mon] = aktivní datum
         ENDIF
      ENDFOR
      RETURN JSON pravidlo a jeho počítače, status 200
   ELSE
      RETURN JSON vyskytla se chyba, status 400
   ENDIF
ENDIF
```

Nejprve uložíme pravidlo a vytvoříme záznam spojení každého počítače s tímto pravidlem v databázi. Pokud vše proběhne v pořádku, vytvoří se log a pravidlo se opět načte z databáze, následuje jeho úprava v podobě přidání potřebných příznaků (CSS třídy a další) z důvodu vykreslování prvků v grafice. Úprava příznaků se používá i při načítání celého seznamu pravidel, ve skutečnosti je implementována funkcí. Na závěr se vrací multidimenzionální pole s daty o pravidle plus názvy připojených počítačů.

8.7.3 Automatické spouštění

Jestliže máme pravidla uložena v databázi, zbývá už jen zajistit jejich automatické spuštění v požadovaný čas. Laravel poskytuje speciální rozhraní pro spouštění úloh, stačí jen vložit záznam do Cronu serveru a můžeme začít s implementací. Záznam vypadá následovně:

```
* * * * * php /path/to/artisan schedule:run >> /dev/null 2>\&1
```

Implementace je provedena v souboru */App/Console/Kernel.php*, kde se nachází úloha s minutovým cyklem kontroly času pravidel. Princip v Pseudokódu je následující:

```
day_of_week = den v týdnu
```

```
time = čas ve formátu h:m

tasks = pravidlo z databáze, kde ((day_of_week == TRUE)
   A (active == TRUE) A (time == time))
IF NOT EMPTY tasks THEN
   připoj se k API
   FOR tasks
      computers = počítače pravidla z databáze
      command = obsah příkazu z databáze dle id příkazu
      vytvoř log o spuštěné akci
      IF command == ZAP THEN
         pošli příkaz zapnutí na API (computers)
      ELSE IF command == VYP THEN
         pošli příkaz vypnutí na API (computers)
      ELSE
         pošli vlastní příkaz na API (computers,command)
      ENDIF
   ENDFOR
ENDIF
```

Zde je důležitá správná synchronizace času a časového pásma, jinak by docházelo k nesprávnému spuštění úlohy. Klíčový je dotaz na databázi, kde hledáme aktivní pravidla v aktuální den a čas každou minutu. Při shodě zjistíme počítače pravidla a dle příkazu s nimi voláme danou akci.

8.8 Správa uživatelů

Uživatele spravuje admin serveru pomocí terminálu a konzole Artisan. K dispozici má trojici příkazů, jejichž seznam nebo syntaxi si může prohlédnout příkazem *php artisan help [příkaz]*. Příkazy jsou také popsány v manuálu aplikace, zde je jejich popis:

- *php artisan user:add [–admin] xlogin* - přidání uživatele do aplikace, přepínač *admin* přidělí práva admina
- *php artisan user:remove xlogin* - odebrání uživatele aplikace
- *php artisan user:list* - přehled uživatelů a jejich skupin

Implementace konzolového příkazu - vypsání uživatelů:

```
class Kernel extends ConsoleKernel {
    protected $commands = [
        Commands\AddUser::class,
        Commands\RemoveUser::class,
        Commands\ListUser::class,
```

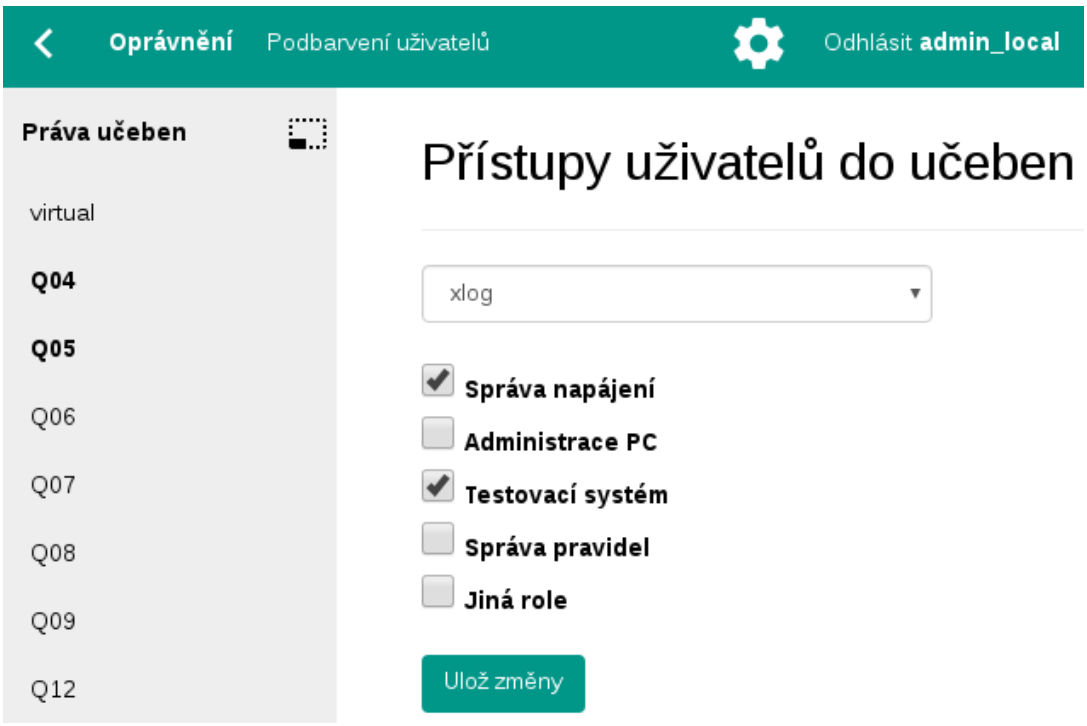
```
    ];
}

class ListUser extends Command
{
    protected $signature = 'user:list';
    protected $description = 'Vypsání všech uživatelů, včetně adminů';
    public function handle()
    {
        $admins = User::where('admin',true)->get();
        $this->info('Admini');
        foreach($admins as $admin){
            $this->info('-'.$admin->login);
        }
        $users = User::where('admin',false)->get();
        $this->info('Uživatelé');
        foreach($users as $user){
            $this->info('-'.$user->login);
        }
    }
}
```

První třída registruje jednotlivé příkazy, druhá třída definuje samotný příkaz vypsání, kde metoda *$this->info()* posílá text do konzole.

8.9 Správa oprávnění

Oprávnění přiděluje uživatelům administrátor aplikace v sekci nastavení, jedná se o přístup k učebnám a jednotlivým funkcím aplikace. Výběr přístupu do učeben je vyřešen pomocí již zmíněného levého panelu v rozhraní. V této části došlo ke znovupoužití komponenty s menší modifikací v podobě odstranění nepotřebného ukazatele počtu počítačů. Obsah stránky je tvořen výběrem, který nabízí seznam uživatelů a zaškrtávacími políčky symbolizujícími funkce aplikace. Náhled do nastavení:



Obrázek 19: Editace oprávnění

Vybráním uživatele dojde k vytvoření AJAXového požadavku na back-end, který načte práva uživatele. Poté je s pomocí JavaScriptu zobrazíme v rozhraní. V případě změně práv a stisku ukládacího tlačítka dojde opět k vytvoření požadavku na pozadí a server vytvoří požadovaný záznam v databázi, konkrétně v asociačních tabulkách *right* a *room_control*.

8.10 Přihlašování

Přihlášení do aplikace je nutno přepracovat ze standardní implementace Laravelu, která počítá pouze s ověřením loginu a hesla uživatelů v jeho databázi. Potřebujeme tedy zajistit, aby naše databáze uživatelů obsahovala pouze loginy uživatelů, kteří dostali přístup do aplikace a v druhé fázi odeslat požadavek ověření zadaného hesla na LDAP server, jehož výsledek určí povolení, či zamítnutí autentizace.

Pro kompletní přepsání autentizace frameworku je třeba dle George Buckinghama (Buckingham, 2016) postupovat těmito kroky:

1. Registrovat nový user provider v */app/Providers/AuthServiceProvider.php*
2. Změnit v configu aplikace položku *user driver*
3. Vytvořit novou třídu (model) uživatele v */App/Models/User*
4. Vytvořit nový user provider */App/Providers/UserProvider.php*

Po splnění těchto bodů a implementaci všech potřebných metod ve zmíněných třídách se dostaneme v našem user provideru ke klíčovým metodám pro ověřování loginu a hesla, tj. *validateCredentials* a *retrieveByCredentials*.

Poslední ze jmenovaných metod má na vstupu login, její úkol je vyhledat tento login v databázi. V případě úspěchu vrací instanci daného uživatele, jinak prázdnou instanci.

Metoda *validateCredentials* přejímá instanci uživatele a heslo, tato metoda slouží k vlastnímu ověření hesla. Výstup je logická hodnota, která rozhoduje, zda konkrétní uživatel bude přihlášen, či ne. Implementace metody v Pseudokódu je následující:

```
IF NOT EMPTY instance uživatele THEN
   IF login == 'lokální admin' && heslo == 'heslo' THEN
      RETURN TRUE
   ENDIF
   spojení = vytvoř spojení s~LDAP serverem
   IF spojení == TRUE THEN
      ověření = vytvoř požadavek na ověření loginu a~hesla
      IF ověření == TRUE THEN
         RETURN TRUE
      ELSE RETURN FALSE
      ENDIF
   ENDIF
ELSE RETURN FALSE
ENDIF
```

Před ověřením hesla na LDAPu (druhý řádek kódu) je zajištěno manuální ošetření případu, kdy se přihlašuje lokální off-line admin. Pokud údaje odpovídají, k požadavku na LDAP už nedojde.

Poslední záležitost, která se týká přihlášení, je synchronizace naší databáze s databází na API. Proběhne pouze v případě řádné autentizace do systému, a pokud API nezasílá prázdné odpovědi. Jedná se zde o synchronizaci učeben a počítačů z důvodu dodržení integrity DB, navíc díky tomuto mechanismu lze po první instalaci aplikace se pouze přihlásit a databáze se sama naplní bez potřeby práce admina.

Princip je v podstatě jednoduchý a je stejný pro počítače i učebny. Spočívá v porovnání dvou množin, první množina má prvky naší databáze a druhá prvky z databáze API. Porovnání jednotlivých prvků zajistí odebrání toho, který by neměl existovat, nebo přidání toho, který by měl existovat a nemáme ho.

9 Závěr

Tato práce se zabývala vytvořením aplikace pro hromadnou správu počítačů, která má nahradit zastaralé řešení ve školním prostředí.

Na začátku jsem provedl analýzu původního řešení, uvedl podstatné nedostatky, které byly pro novou práci zásadní. Dále jsem postupoval dle metodiky od shromáždění požadavků k analýze, návrhu a nakonec k implementaci a testování.

Po celou dobu vývoje aplikace byl postup konzultován s kolektivem ÚI PEF pro nalezení nejlepšího řešení vymezených cílů a technického zajištění spolupráce všech potřebných serverů a technologií.

Byla nasazena zkušební a téměř kompletní verze aplikace v rámci testovacího režimu pro několik vybraných správců, díky kterým jsem postupně mohl opravovat nalezené chyby. Ty se týkaly například špatného zobrazení a přetečení prvků rozhraní na některých typech zařízení nebo výskytu neočekávaných hodnot v některých funkcích, který způsoboval pád aplikace. Také jsme narazili na problém při odesílání požadavku z klienta, jednalo se o zapnutí velkého množství stanic, které skončilo chybovým stavem. Chyba byla v použití požadavku typu GET, jehož délka je omezena určitým počtem znaků, právě zde docházelo k přetečení. Oprava byla zajištěna změnou typu požadavku na POST, která prostor spolehlivě navýšila. Po dobu testování bylo také implementováno různé vylepšení dle poznámek zúčastněných osob.

Nyní je aplikace v provozu na produkčním serveru, kde slouží správcům a učitelům pro ulehčení jejich práce. Bylo dosaženo všech vytyčených cílů a uživatelé jsou s novou aplikací spokojeni.

Byla také vytvořena příručka administrátora, která obsahuje pokyny pro kompletní uvedení aplikace do provozu, včetně připojení potřebných služeb, obsluhy a přidání dodatečných funkcí.

Tato aplikace má velký potencionál pro možné rozšiřování, zajímavé by bylo například umožnit správcům z rozhraní aplikace převzít vzdálenou plochu nad vybraným strojem nebo integrovat do seznamu příkazů kompletní nabídku PowerShellu.

10 Reference

API. API - Wikipedia [online]. 2016 [cit. 2016-04-5]. Dostupné z: <https://cs.wikipedia.org/wiki/API>.

BUCKINGHAM, George. *Laravel 5.2 Authentication Custom User Providers* [online]. 2016 [cit. 2016-11-10]. Dostupné z: <https://blog.georgebuckingham.com/laravel-52-auth-custom-user-providers-drivers/>.

CASTRO, Elizabeth a Bruce HYSLOP. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. 1. vyd. Brno: Computer Press, 2012. ISBN 978-80-251-3733-8.

ČÁPKA, David. *MVC architektura* [online]. 2013 [cit. 2016-04-26]. Dostupné z: <http://www.itnetwork.cz/navrhove-vzory/mvc-architektura-navrhovy-vzor>.

JQUERY FOUNDATION. *JQuery* [online]. 2006 [cit. 2016-04-25]. Dostupné z: <http://jquery.com/>.

KOUBEK, Tomáš. *Dokumentace k API Morpheus*. Mendelova Univerzita Brno, 2016.

KRUG, Steve. *Webdesign: Nenuťte uživatele přemýšlet!*. 2. vyd. Brno: Computer Press, 2006. ISBN 80-251-1291-8.

LEISS, Oliver a Jasmin SCHMIDT. *PHP v praxi*. 1. vyd. Praha: Grada Publishing a.s, 2010. ISBN 978-80-247-3060-8.

MACHAČ, Marek. *11 nejlepších HTML a CSS frameworků pro webdesignéry* [online]. 2015 [cit. 2016-04-26]. Dostupné z: <https://www.interval.cz/clanky/11-nejlepsich-html-a-css-frameworku-pro-webove-designery/>.

MILLER, David. *Simple Sidebar* [online]. 2016 [cit. 2016-12-03]. Dostupné z: <https://startbootstrap.com/template-overviews/simple-sidebar/>.

MONUS, Anna. *PHP 7: 10 Things You Need to Know* [online]. 2015 [cit. 2016-04-26]. Dostupné z: <http://www.hongkiat.com/blog/php7/>.

MONUS, Anna. *10 PHP Frameworks For Developers – Best Of* [online]. 2016 [cit. 2016-04-26]. Dostupné z: <http://www.hongkiat.com/blog/best-php- frameworks/>.

MROZEK, Jakub. *Začínáme s AngularJS* [online]. 2012 [cit. 2016-04-26]. Dostupné z: <https://www.zdrojak.cz/clanky/zaciname-s-angularjs/>

NEŠPOR, Filip. *Webové rozhraní pro správu uživatelských účtů*. Brno, 2007. Dostupné z: <https://is.muni.cz/th/139515/fi_b/BakPrace.pdf>. Bakalářská

práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce RNDr. Radek Ošlejšek, Ph.D..

NETTE FOUNDATION. *Nette framework* [online]. 2016 [cit. 2016-04-26]. Dostupné z: <https://nette.org/cs/>.

OBJEKTOVĚ RELAČNÍ MAPOVÁNÍ. Objektově relační mapování - Wikipedia [online]. 2016 [cit. 2016-04-27]. Dostupné z: <https://cs.wikipedia.org/wiki/Objektově_relační_mapování>.

OTTO, Mark a Jacob THORTON. *Bootstrap* [online]. 2016 [cit. 2016-04-27]. Dostupné z: <http://getbootstrap.com/css/>.

OTTO, Mark a Jacob THORTON. *Github* [online]. 2016 [cit. 2016-05-02]. Dostupné z: <https://github.com/>.

OTWELL, Taylor. *Laravel* [online]. 2016 [cit. 2016-11-10]. Dostupné z: <https://laravel.com/docs/5.3/authentication>.

REFSNES DATA FOUNDATION. *XML Soap* [online]. 2016 [cit. 2016-10-12]. Dostupné z: <http://www.w3schools.com/xml/xml_soap.asp>.

REFSNES DATA FOUNDATION. *HTML 5* [online]. 2016 [cit. 2016-04-26]. Dostupné z: <http://www.w3schools.com/html/html5_intro.asp>.

RÉVAY, Petr. *Systém pro správu sportovního klubu karate*. Brno, 2016. Dostupné z: <https://theses.cz/id/x4uqwl>. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce RNDr. Jaromír Plhák, Ph.D..

SELLIER, Alexis. *LESS* [online]. 2016 [cit. 2016-04-26]. Dostupné z: <http://www.lesscss.cz/>.

VYBÍRAL, Petr. *Webové aplikace a webové služby v C#*. Brno, 2012. Dostupné z: <https://dspace.vutbr.cz/handle/11012/9645>. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Lattenberg, Ivo.

VYHNÁLEK, Jiří. *Komunitní web pro hledače pokladů*. Plzeň, 2014. Dostupné z: <https://otik.uk.zcu.cz/handle/11025/13498>. Bakalářská práce. Západočeská Univerzita v Plzni, Fakulta Aplikovaných Věd, Katedra Informatiky a výpočetní techniky. Vedoucí práce Fiala, Dalibor.

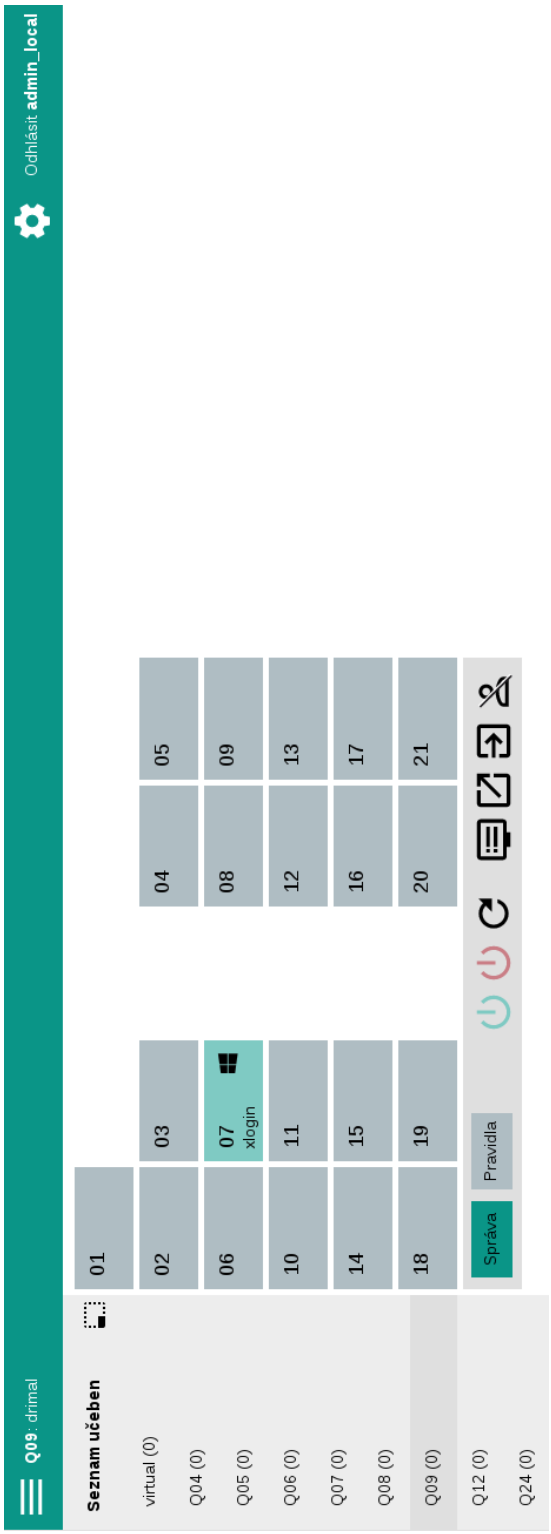
YAHOO FOUNDATION. *PureCss* [online]. 2016 [cit. 2016-04-26]. Dostupné z: <http://purecss.io/>.

ZAKAS, C. Nicholas. *The Principles of Object-Oriented JavaScript*. 1. vyd. San Francisco: No Starch Press, Inc., 2014. ISBN 978-1-59327-540-2.

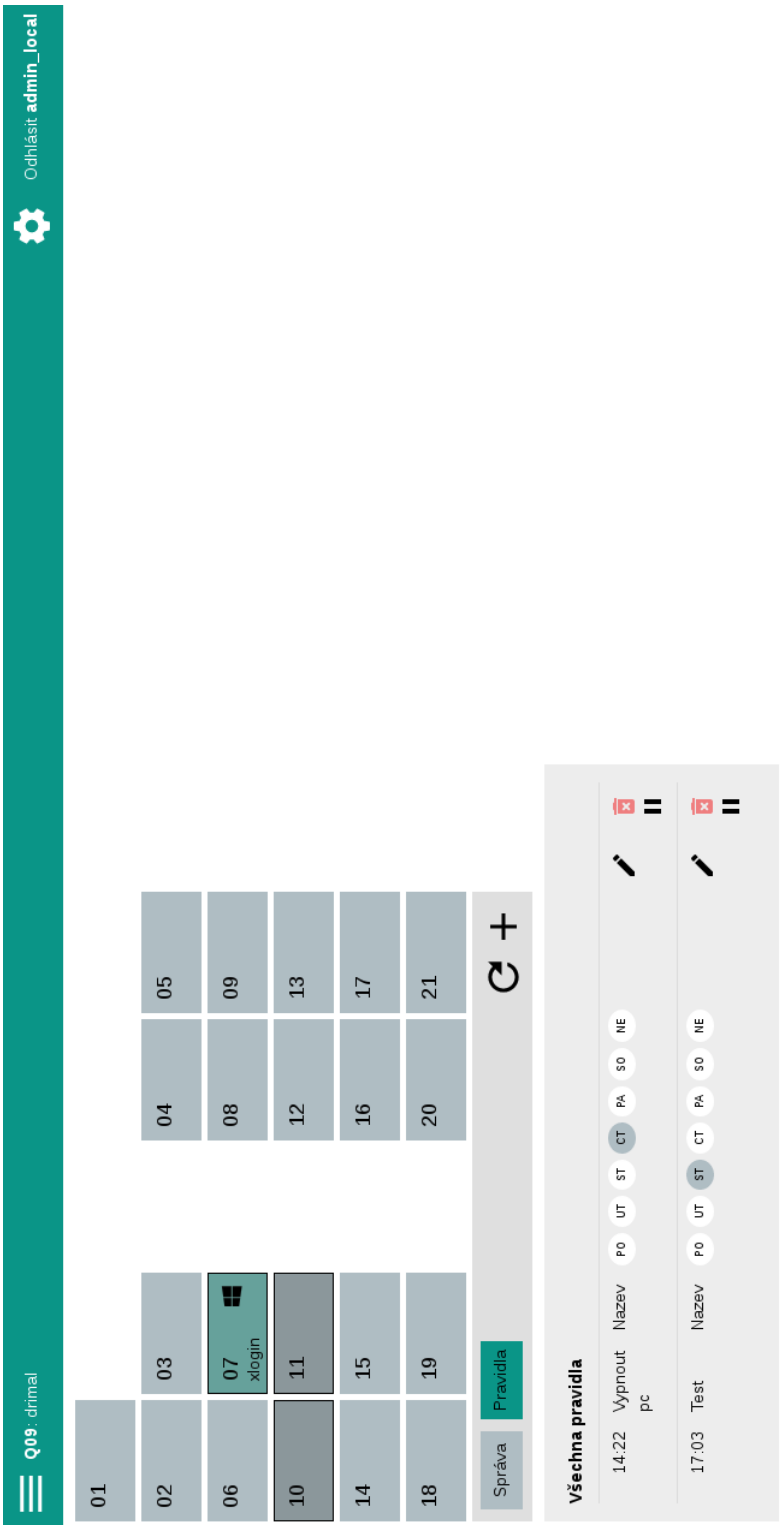
Přílohy

A Původní řešení

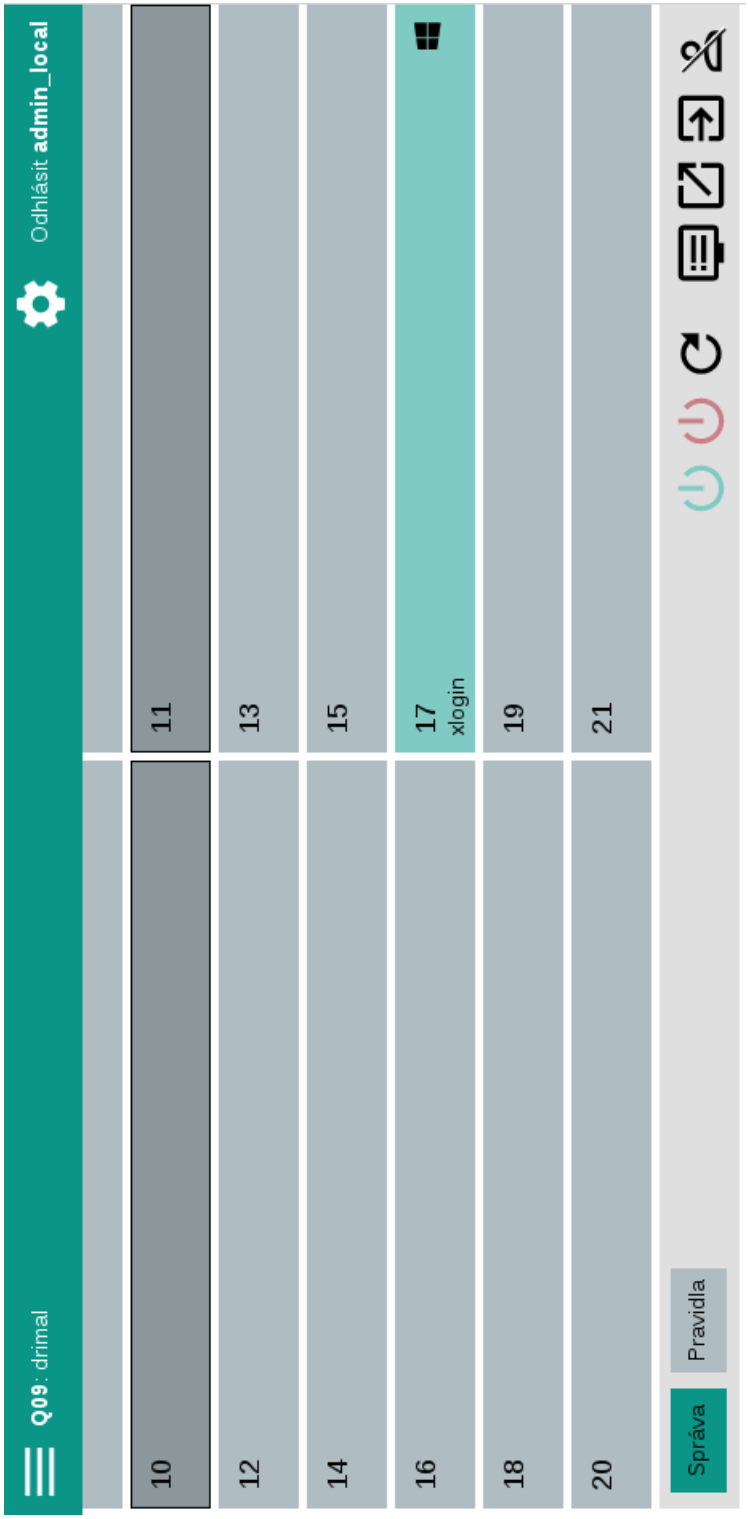
B Náhled po přihlášení na desktopu



C Náhled v kontextu pravidel na menší obrazovce



D Náhled po přihlášení na tabletu



E Náhled po přihlášení na telefonu