VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

KONFIGURÁTOR A REZERVAČNÍ SYSTÉM KONCERTNÍCH SÍNÍ

CONFIGURATION AND RESERVATION SYSTEM FOR CONCERT HALLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VÍT SIKORA**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. JAROSLAV ZENDULKA, CSc.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/20232/2016/xsikor15

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Sikora Vít**
Obor: Informační technologie
Téma: **Konfigurátor a rezervační systém koncertních síní**
Configuration and Reservation System for Concert Halls
Kategorie: Databáze

Pokyny:

1. Seznamte se s požadavky kladenými na front-end aplikaci běžící ve webovém prohlížeči, která bude umožňovat konfiguraci koncertních síní pro jednotlivé akce a rezervaci míst návštěvníků.
2. Analyzujte požadavky na tuto aplikaci včetně požadavků na perzistentní data ukládaná do databáze. Pro analýzu využijte modelovacích technik jazyka UML.
3. Seznamte se s rámci pro JavaScript vhodnými pro implementaci aplikace, porovnejte je a zdůvodněte výběr rámce pro implementaci.
4. Navrhněte a implementujte aplikaci a serverovou část s databází.
5. Funkčnost aplikace ověřte na vhodně zvoleném vzorku testovacích dat.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího pokračování projektu.

Literatura:

- Grässle, P., Baumann, H., Baumann, P.: UML 2.0 in Action: A Project Based Tutorial. Packt Publishing. 2005. 229 s. ISBN 1-904811-55-8.
- JavaSript framework resources. Dostupné na https://github.com/vhf/free-programming-books/blob/master/javascript-frameworks-resources.md.

Pro udělení zápočtu za první semestr je požadováno:

- Body č. 1, 2, 3

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zendulka Jaroslav, doc. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce popisuje realizaci webové aplikace umožňující konfiguraci událostí v koncertních síních a definici sekcí a míst v těchto síních. Aplikace dále umožňuje běh v režimu pro koncové uživatele, v rámci něhož lze do vydefinovaných síní provádět rezervace míst. Je kladen důraz na vestavnost a přizpůsobitelnost tak, aby bylo jednoduché aplikaci integrovat do stávajícího systému potenciálního zákazníka. Implementace je rozdělena do serverové části (REST API) uskutečněné za pomoci PHP 7 s frameworkem Restler a klientské části, která je postavená jako jednostránková aplikace s pomocí moderního Javascriptu (ECMAScript 2016) s frameworkem React a překladem do běžného Javascriptu a HTML za pomocí nástrojů Webpack a Babel.

Abstract

This paper describes the implementation of a web application that enables configuring events in concert halls and defining sections and seats in those halls. Furthermore, application can run in a mode for end users that enables placing reservations in previously defined halls. Certain emphasis is placed on customizability so that it would be easy to integrate this application into an existing system. Implementation is divided into server part (REST API) realized using PHP 7 with Restler framework and to client part built as a single page application using modern Javascript (ECMAScript 2016) with React framework and transpiled to common Javascript and HTML using Webpack and Babel.

Klíčová slova

Rezervační systém, rezervace, síň, sál, hala, událost, jednostránková aplikace, Node.js, Javascript, ECMAScript, JSX, React, Redux, PHP, Restler, REST API, vkládání závislostí

Keywords

Reservation system, reservations, hall, auditorium, event, single page application, SPA, Node.js, Javascript, ECMAScript, JSX, React, Redux, PHP, Restler, REST API, dependency injection

Citace

SIKORA, Vít. *Konfigurátor a rezervační systém koncertních síní.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zendulka Jaroslav.

Konfigurátor a rezervační systém koncertních síní

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Jaroslava Zendulky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vít Sikora
16. května 2017

Poděkování

Chtěl bych poděkovat svému vedoucímu panu doc. Ing. Jaroslavu Zendulkovi, CSc. za odborné vedení mé práce a za jeho věcné a velmi užitečné připomínky. Dále bych chtěl poděkovat společnosti SOVA NET, s.r.o. za možnost vyvíjet a umístit serverovou část aplikace na její výkonné servery s vysokou konektivitou a za poskytnutí dalšího užitečného technického vybavení.

Obsah

Kapitola 1

Úvod

Rezervační systémy obecně nemohou být z principu zcela univerzální. Jeden systém má zpravidla jedno, výjimečně několik rozhraní, a je tedy schopný kvalitně pokrýt pouze omezenou množinu rezervovatelných prvků (těmi nyní myslím například sedadla, časová okna, ordinace, tribuny, stoly v restauracích aj.). Z uvedeného plyne, že čím specifičtější požadavky na rezervační systém subjekt má, tím problematičtější může být využití obecného rezervačního systému.

Dalším častým zdrojem komplikací je samotná integrace do vlastního systému. Potenciálních překážek se objevuje hned několik. Když už se vše podaří po technické stránce a rezervační systém funguje, neznamená to, že zcela zapadne i po stránce estetické a ve stávající prezentaci nebude působit jako ona „pěst na oko“.

Výše zmíněné problémy může řešit systém postavený na míru, ale ten s sebou často přináší jinou podstatnou nevýhodu, a to náklady na jeho vytvoření.

Tato bakalářská práce si klade za cíl vytvořit částečně obecný rezervační systém, který všechny již zmíněné problémy do určité míry řeší. Aplikace je koncipována jako rezervační systém pro koncertní síně, nicméně bude realizována způsobem, který bude umožňovat snadnou transformaci na obecný rezervační systém pro rezervaci míst (více v sekci 3.1.2). Dalším předpokladem je jednoduchost vložení do již existujícího systému (viz 3.1.3).

Práce je členěna celkem do 8 kapitol. Po úvodu následuje kapitola věnovaná základním pojmům a specifikaci zadání, na níž navazuje třetí kapitola, věnující se detailní analýze požadavků. Následuje kapitola 4, v níž je rozebrána volba vhodných technologií a rámců pro front-endovou část aplikace. V páté kapitole je popsána celá etapa návrhu a vývoje. Šestá kapitola popisuje testování aplikace a sedmá se věnuje dalšímu vývoji. V závěrečné kapitole jsou shrnuty dosažené výsledky a vlastní přínos.

Výše zmíněné cíle vychází ze skutečných klientských požadavků.

Kapitola 2

Určení nároků a specifik

V této kapitole se budu věnovat samotnému určení nároků na aplikaci a definování její stručné specifikace. Budu vycházet i ze specifikace zákazníka, která byla stručná, nicméně se postupně mírně měnila. Zákazníkovy požadavky byly získávány prostřednictvím osobních schůzek. Nejprve se jednalo o pouze ústní specifikaci, později to byly nákresy a diskuse nad prototypem aplikace.

2.1 Vymezení základních pojmů

Seznam pojmů, které se budou objevovat v této a v dalších kapitolách práce.

Aplikace — je výsledný systém, jehož vývoj tato zpráva popisuje, celým názvem Konfigurátor a rezervační systém koncertních síní, zkráceně také KORES.

Událost — je jedno konkrétní představení v dané datum, které se koná v síni. Má název, datum a čas konání.

Síň — je fyzický prostor obsahující místa k rezervaci, v němž se může konat více událostí. Má název a adresu a je tvořena sekcemi míst.

Událost v síni — je událost a síň spojena v jednu entitu. Nová událost ve stejné síni je novou položkou a na tu původní nemá žádnou vazbu.

Sekce — je skupina míst uvnitř síně uspořádaných do tvaru obdélníku. Má vlastní název, cenu, stav a typ míst a může se nacházet v libovolném patře síně.

Typ místa — je jeho fyzická dispozice, tedy například sedadlo, lavice, pódium, sloup aj. Sekce může tedy vyjadřovat například pódium, nebo nerezervovatelnou část síně.

Stav místa — je jeho logickou rolí, tedy například online předprodej (rezervovatelné skrze KORES), prodej na místě, rezervováno, neprodejné atp.

2.2 Specifikace zákazníka

Jak již zaznělo v úvodu, požadavek na částečně obecný systém vznikl kompromisem, a to mezi cenou řešení a vhodností implementace. Znovupoužitelné řešení snižuje potenciální výdaje prostředků, přičemž zachovává plné přizpůsobení požadavkům a specifikům koncertních síní.

Kromě vytvoření znovupoužitelného rezervačního systému klade specifikace nároky zejména na jednoduchost ovládání (cílová skupina uživatelů — potenciální návštěvníci koncertů — je častěji tvořena staršími a více umělecky než technicky zaměřenými lidmi) a časovou nenáročnost úprav v administraci.

Z výše uvedeného lze určit obecné požadavky, které musí aplikace splnit. Výčet následuje:

- Jednoduchá upravitelnost řešení pro odlišné požadavky,

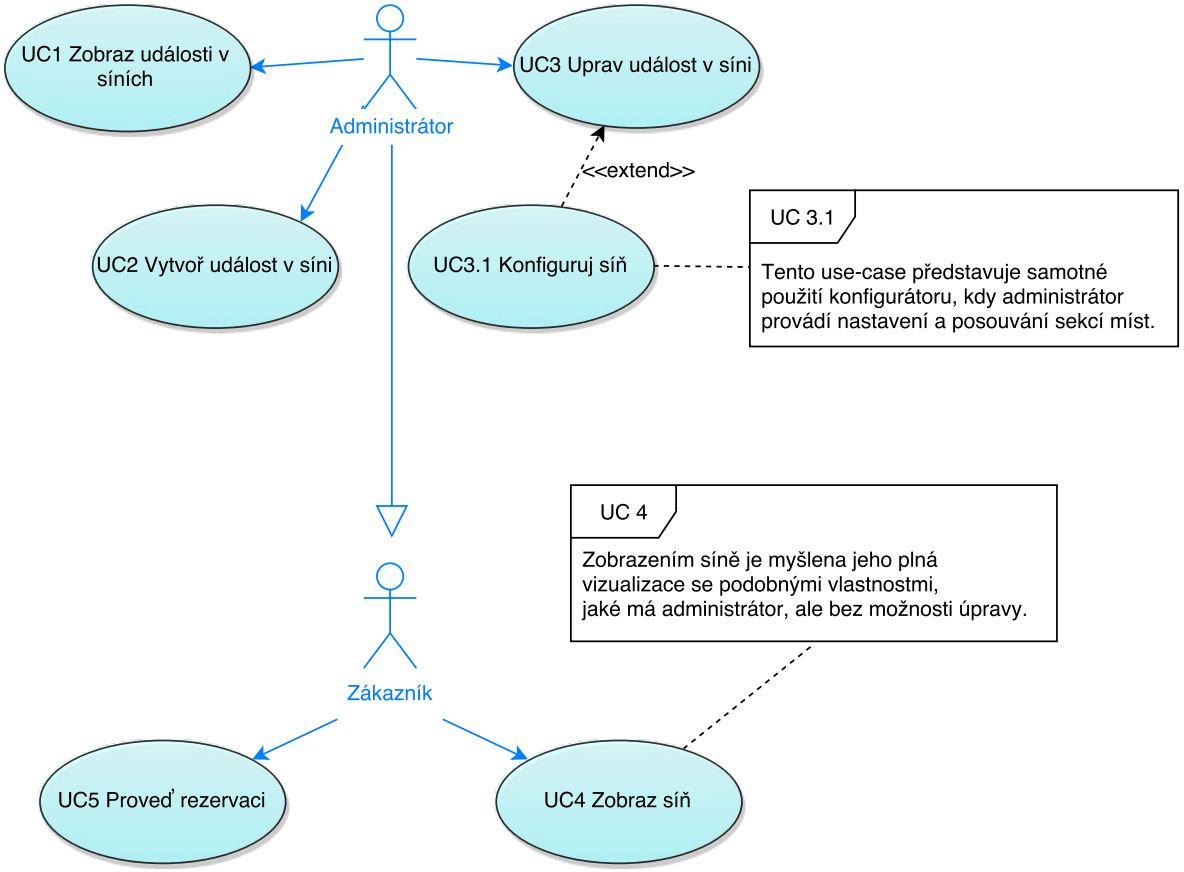
- možnost jednoduše nasadit do existující webové prezentace,
- snadno ovladatelné uživatelské rozhraní,
- efektivní administrace (odbourání nadbytečných operací).

2.3 Požadavky kladené na aplikaci

Respektováním zadání práce a specifikace zákazníka je možné vytvořit tento seznam požadavků na výslednou podobu aplikace:

- Možnost vytvářet, upravovat, mazat a klonovat události v síních vč. jejich atributů.
  - Dle specifikace se místo událostí a síní používá pouze událost v síni (dále už jen **událost**). Toto je z důvodu velmi častých dílčích úprav síně za určité období (zjištěno přímo zákazníkem z vlastní zkušenosti) a častých úprav stavu a cen sekcí. Redundance záznamů bude tedy v důsledku minimální.
  - Klonování celé události v síni vč. vnitřních sekcí má za cíl urychlit práci při vykonávání nejobvyklejšího případu užití, kterým je bezpochyby vytvoření nové události (v případě, že je podobná jiné, již vytvořené).
- Možnost vytvářet, upravovat a mazat sekce sedadel vč. jejich atributů.
- Možnost umisťovat sekce na mřížku a vytvořit tak reprezentaci síně blížící se realitě.
- Pro koncové uživatele umožnit rezervace do takto vydefinovaných síní.
  - Rezervace musí být plně grafická a zákazník musí volit místa taktéž v grafické reprezentaci obdobné té, v jaké byla síň vytvořena.
- Všechny úpravy prováděné v administraci se musí ukládat do databáze na základě příslušné akce uživatele.
- Aplikace musí fungovat ve všech aktuálně používaných internetových prohlížečích.
- Práce v aplikaci musí být dostatečně plynulá a uživatelské akce musí mít rozumnou odezvu.
- Aplikace musí být adekvátně zabezpečená.
- Aplikace se musí umět zotavit z dočasného výpadku internetového připojení.

2.4 Model případů použití



Obrázek 2.1: Model případů použití.

Podrobný popis jednotlivých případů na obrázku 2.1:

UC1 Zobraz události v síních — tento případ užití se týká administračního rozhraní a představuje situaci, kdy správce systému chce vidět veškeré existující události v síních. Od uživatele není očekáván žádný vstup a mělo by se jednat ideálně o hlavní pohled při spuštění aplikace. Pohled musí obsahovat přehledný seznam všech událostí, obsahující jejich název a celkovou kapacitu. Musí být možné přejít na konfiguraci síně, upravit událost nebo vytvořit její kopii. Dále také musí existovat možnost vytvoření nové události.

UC2 Vytvoř událost v síni — tento případ užití má dvě varianty — tou první je vytvoření úplně nové události, které následuje po vyvolání akce vytvoření. Druhou variantou je pak vytvoření zkopírováním existující události, které nastává po vyvolání akce vytvoření kopie. Obě varianty vyvolají okno pro nastavení všech parametrů nové události v síni a jediný rozdíl je v tom, že v případě druhé varianty jsou hodnoty předvyplněné dle kopírované události. Po potvrzení vytvoření následuje přechod na **UC 3.1** pro nově vytvořenou síň.

UC3 Uprav událost v síni — tento use-case představuje úpravu dat konkrétní události v síni. Kvůli širokosti tohoto pojmu jsem izoloval konfiguraci síně do samostatného use-case **UC3.1** a v tomto jsem ponechal úpravu parametrů události. Očekávanou uživatelskou akcí je vyvolání akce úpravy u příslušné události v pohledu **UC1**. Následuje zobrazení

obdobného okna, jaké je použito při **UC2**, ale po potvrzení změn se uživatel navrací na hlavní pohled **UC1**.

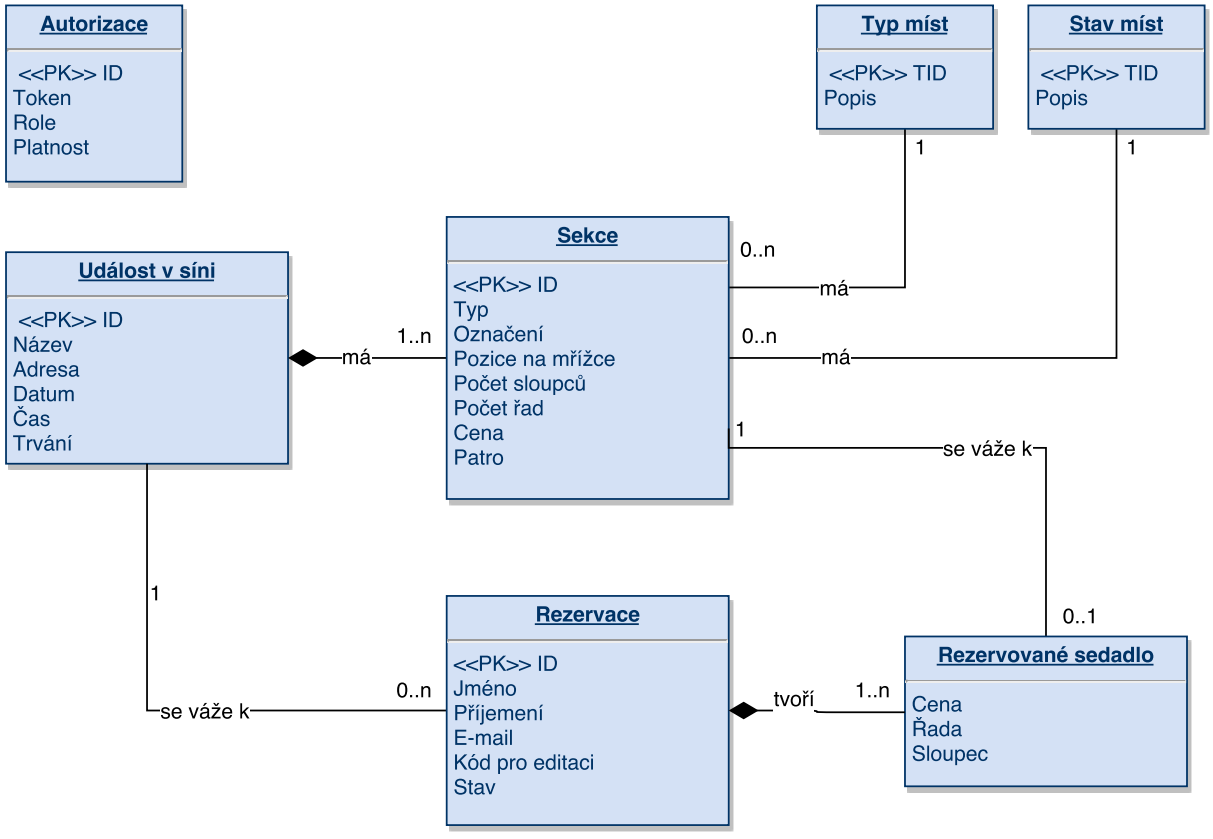
UC3.1 Konfiguruj síň — tento případ použití popisuje úpravu síně uživatelem a shrnuje všechny operace nad sekcemi, umisťovanými na mřížku, tedy jejich vytváření, mazání, posunování, změnu velikosti a dalších parametrů. Uživateli jsou tyto operace umožněny grafickým rozhraním pohledu na síň. Případ použití je dokončen vyvoláním akce uložení, čímž jsou všechny změny v síni uloženy do databáze.

UC4 Zobraz síň — tento případ užití se už týká rozhraní pro koncové uživatele a představuje situaci bezprostředně po spuštění aplikace v tomto režimu — tedy pohled na síň, pro kterou je rezervace spuštěna. Tento pohled se podobá tomu v **UC3.1**, ale je uživatelsky přívětivější a místo sekcí zobrazuje jednotlivá místa.

UC5 Proveď rezervaci — tento případ použití je 3krokovým procesem. Nejprve je po uživateli vyžadována volba míst v pohledu **UC4**, poté po potvrzení je vyžadováno zadání povinných údajů. Po dalším pokračování je uživateli zobrazen výpis jím zvolených míst a tlačítko pro potvrzení rezervace, po jehož stisku je use-case završen uložením rezervace do databáze.

2.5 Datový model

Na základě uživatelských požadavků na aplikaci byl navržen ER diagram reprezentující vztahy mezi jednotlivými entitami v aplikaci. Podoba diagramu byla několikrát konzultována s vedoucím práce i zákazníkem, až došla do finální podoby, která se nachází na obrázku 2.2.



Obrázek 2.2: Entity-relationship diagram.

Na diagramu si lze povšimnout dvou vztahů kompozice, kterými jsem chtěl znázornit vysokou míru závislosti dílčích prvků na celku a explicitně vyjádřit, že celek je těmito částmi tvořen a nemůže bez nich existovat. Tato situace se týká Události v síni, která je tvořena Sekcemi a Rezervace, která je tvořena Rezervovanými sedadly.

Nachází se zde i entita, která nemá žádné vztahy k okolí — Autorizace. U této entity je zamýšleno využití externím systémem, který se bude starat o autentizaci, kdežto tato aplikace pouze o autorizaci přístupů.

Kapitola 3

Analýza požadavků na webovou aplikaci

V této kapitole se budu věnovat důkladné analýze požadavků z kapitoly minulé. Začnu obecnými požadavky, které mají přesah do celého projektu, a budu pokračovat k těm více specifickým.

3.1 Obecné aplikační požadavky

3.1.1 Snadno ovladatelné uživatelské rozhraní

Nejprve bych se chtěl věnovat obecnějším požadavkům, které mají vliv na realizaci celé části, mezi něž grafické uživatelské rozhraní (GUI) zcela jistě patří.

Uživatelské rozhraní musí být jednotné, uživatelsky přívětivé a především intuitivní. Jednotnost znamená, že všechny grafické prvky budou následovat stejná pravidla pro zobrazení, chování a interakci s uživatelem. Tato jednotnost dokáže významně přispět k intuitivnímu ovládání tak, že uživatel po pochopení funkce jednoho prvku už při pohledu na jiný bude dopředu správně odhadovat jeho funkcionalitu. Příkladem může být zbarvení akčního tlačítka ve chvíli, kdy jsou splněny podmínky pro vykonání akce. Jakmile se uživatel tuto funkci naučí, i u jiného akčního prvku bude očekávat stejné chování, a pokud jej uvidí neaktivní, bude vědět, že nebyly splněny podmínky pro spuštění akce.

Uživatelskou přívětivostí je myšleno využití technik, které uživateli pomáhají lépe pochopit funkcionalitu prvků a dávají mu zpětnou vazbu při jejich vykonávání. Příkladem může být využití animací a přechodů, které v uživateli podporují přirozený vjem toho, že k nějaké akci došlo postupně, a nikoliv „zničehonic“. Je mnohem lepší, když se například vysunovací boční lišta postupně vysune, namísto toho, aby se najednou objevila. Vyjetím na sebe upozorní, uživatel si jí lépe všimne a zároveň mu zůstává podvědomý pocit, že má chování aplikace pod kontrolou. Podobně je možné pracovat třeba i se stínem u tlačítek, který v uživateli vyvolává zpětnou vazbu, že tlačítko stisknul.

3.1.2 Upravitelnost

Upravitelnost je míra, do jaké lze systém modifikovat pro odlišné požadavky. Pro kvalitní systém by nemělo být problémem, pokud se drobně změní některé rozhraní, vznikne požadavek na přidání položky do některého z existujících výčtů, nebo se objeví potřeba nového pohledu. Naopak jedním z hlavních rysů nekvalitního systému je, že nelze snadno vyřešit drobnou úpravu požadavků — například proto, že implementace nějaké součásti je psaná přesně podle aktuální potřeby a na nějakou rozšiřitelnost není ani pomyšleno.

S upravitelností blízce souvisí pojem modularita, který je ve své definici trochu specifičtější a určuje míru, jakou je systém dělen do samostatných celků a jak jsou tyto celky nahraditelné jinou implementací. Nahraditelnost jinou implementací velmi závisí na těsnosti a explicitnosti závislostí na ostatních celcích.

Systém, který je modulární, má velmi dobře vykročeno k tomu, aby byl i upravitelný.

3.1.3 Vložitelnost do existujícího systému

Problematika vložitelnosti aplikace do cizího systému má několik aspektů. Prvním je technická část vkládání, které musí být řešeno tak, aby aplikace byla schopná vložení do vícera různých typů systémů. Nekvalitním řešením této problematiky by například bylo, kdyby aplikace šla vložit pouze do neresponzivního webu nepoužívajícího Javascript. Lze tedy konstatovat, že vložitelnost musí být do jisté míry univerzální.

Druhým aspektem problematiky je stránka estetická. Systém, který je snadno vložitelný, musí být také snadno upravitelný po vzhledové stránce, aby jej bylo možné graficky přizpůsobit jakémukoli okolí. Základem je alespoň přizpůsobení po stránce palety barev, vyšším stupněm potom i po stránce stylu použitých prvků a písem.

Dalším důležitým aspektem je absence vzájemného rušení systémů. Tímto mám na mysli například křížení jmenných prostorů stylů nebo funkcí, které může jeden ze systému paralyzovat.

V neposlední řadě je také podstatné, aby proces vložení byl jednoduchý a nevyžadoval vysokou odbornost. Kupříkladu by mělo být plně dostačující vložit zkopírovaný kus kódu a poté například do určeného místa vložit nějaký API klíč, ale rozhodně ne psát třeba skript pro vložení.

3.1.4 Efektivní administrace

Požadavek zní, že by obsluha administrace měla být rychlá a časté úkony by měly vyžadovat co nejméně kroků k dokončení. Těmito kroky je myšleno například zbytečné klikání navíc, když je často používaný akční prvek skryt v podnabídce.

Tato problematika má mírný přesah do intuitivnosti grafického rozhraní. Je totiž nutné, aby všechny akce byly efektivní a dostupné přes co nejméně kroků, ale zároveň musí dojít k zachování intuitivnosti.

Celou problematiku lze tedy shrnout jako optimalizaci často používaných prvků spolu se současným hledáním kompromisu s intuitivností prostředí.

3.1.5 Podpora v rozšířených internetových prohlížečích

Skutečným požadavkem je, aby bylo možné aplikaci spustit a provozovat na většině uživatelských zařízení. Slovem provozovat je v tomto kontextu myšleno bezproblémové používání. Problémem může být, pokud daný prohlížeč nepodporuje nějakou událost nebo funkci. Částečným splněním požadavku může být, když nepodporovaná funkcionalita pouze negativně ovlivňuje uživatelský komfort či výkon aplikace, ale pokud dojde k selhání (například zamrznutí) aplikace, není požadavek splněn.

3.1.6 Požadavky na výkon a efektivitu

Je samozřejmě žádoucí, aby aplikace pracovala plynule a s rozumnou odezvou. Opak způsobuje značný uživatelský diskomfort. Plynulostí se rozumí, že jednoduché operace, jako jsou například aktivace prvku, psaní textu nebo rolování obsahu, jsou vykonány instantně, bez znatelného zpoždění, a složité operace, jako například změna pohledu nebo stažení dat, mají nízkou a pochopitelnou odezvu, která je nejlépe graficky indikována. Grafická indikace čekání na nějakou

operaci (např. stažení dat) informuje uživatele o tom, že se něco děje a že má čekat, nikoliv přemýšlet o tom, zda tlačítko stisknul správně. Tato indikace znatelně prodlužuje interval, do jakého je zpoždění považováno za rozumné a přijatelné.

Požadavkem na výkon je samozřejmě myšlen požadavek na efektivitu a optimalizaci stěžejních algoritmů, jelikož výkon klientského zařízení nelze nijak ovlivnit.

3.1.7 Požadavky na zabezpečení

Zabezpečení je velmi široký pojem a v tomto kontextu jej lze konkretizovat jako zajištění ochrany proti cizímu vniknutí do systému nebo krádeži či znehodnocení dat aplikace. Aplikace například nesmí obsahovat citlivé informace jako přístupová hesla či klíče, nebo data jiných uživatelů, a to ani v paměti nebo zdrojových kódech, jelikož tyto informace jsou technicky zdatným uživatelům snadno přístupné.

3.1.8 Požadavek na zotavení

Požadavkem na zotavení je myšlena schopnost aplikace fungovat krátkodobě bez potřeby internetového připojení. Přesněji by mělo jít vykonat veškeré akce, které nutně nevyžadují stažení nových dat, které aplikace nemá ještě k dispozici. Pokud je ale stažení nutné, musí se aplikace z takové situace zotavit (a neskončit například s výjimkou) a dovolit uživateli pokusit se o stejnou akci kdykoliv znovu, například když bude připojení k dispozici.

3.2 Funkční požadavky

3.2.1 Správa událostí v síních

Správa událostí v síních sestává ze čtyř základních operací:

- vytváření,
- upravování,
- klonování,
- mazání.

V souvislosti s požadavkem na perzistenci dat je nutné, aby se všechny tyto operace ihned ukládaly do databáze. Vytváření i klonování události v síni musí umět pracovat se všemi atributy modelu události v síni a v případě klonování musí být předvyplněny hodnoty klonované události.

Klonování události nejenže vytvoří kopii události v síni, ale je nutné udělat kopii také všech sekcí, které se v síni nacházejí. Při této kopii však už nesmí být kopírovány další vazby, např. rezervovaná místa.

V případě mazání události nesmí být dovolen výmaz, existují-li do síně rezervace.

Mazání by měla být událost s potvrzením, aby se eliminovala případná uživatelská chyba.

3.2.2 Konfigurace síní

Tento požadavek je nejspíše tím nejnáročnějším. Stanovuje potřebu administrovat sekce míst ve dvourozměrném prostoru, představujícím fyzický prostor síně. Je požadováno, aby bylo možné volit aktuální patro a z hlediska uživatelského komfortu (viz 3.1.1) je vhodné, aby šlo prostor přibližovat a oddalovat pro pohodlnější práci s většími síněmi. U vytvořených sekcí míst musí být možnost pohodlně upravovat všechny její atributy, jako je například cena, typ, stav, název

či rozměry. Některé tyto atributy by se měly promítat i do vizuální reprezentace sekce, aby bylo celé zobrazení síně přehlednější.

V souvislosti s požadavky na efektivní administraci (viz 3.1.4) je vhodné grafickou správu sekcí realizovat uživatelsky co nejjednodušším způsobem tak, aby bylo vytváření a přesouvání sekcí co nejrychlejší.

3.2.3 Rezervační systém

Úkon provedení uživatelské rezervace sestává z několika dílčích kroků:

1. výběr míst,
2. vyplnění povinných údajů,
3. potvrzení rezervace.

Výběr míst — musí být realizován na adekvátní grafické reprezentaci síně, která by měla připomínat síň skutečnou. V této reprezentaci musí být graficky znázorněno, která místa si uživatel může rezervovat a která jsou již zarezervovaná někým jiným, případně je nelze zarezervovat vůbec. Přechodu na další krok by vždy měla předcházet validace kroku aktuálního — v tomto případě kontrola dokončení výběru míst.

Vyplnění povinných údajů — musí obsahovat formulář se všemi údaji, které jsou nezbytné pro úspěšné uložení rezervace do databáze a jejich vyplnění také ověřovat.

Potvrzení rezervace — je posledním krokem a musí obsahovat shrnutí a detailní popis všech rezervovaných míst. Po odeslání potvrzení se uživateli zobrazí informace o stavu výsledku, tedy zda rezervace uspěla, nebo ne. V případě neúspěchu je nutné sdělit i příčinu chyby.

Pro přehlednost a uživatelskou přívětivost (viz 3.1.1) musí všechny kroky obsahovat navigaci obsahující ostatní kroky a indikaci aktuálního kroku.

Kapitola 4

Výběr vhodného frameworku pro front-end

Tato kapitola popisuje průběh výběrů vhodného frameworku pro implementaci front-endu. Nároky na framework vychází z nároků na aplikaci a obecných nároků na udržitelnost webové aplikace. Celý proces se skládá z výběru počátečních kandidátů na základě prvotních podmínek, jejich následného detailního srovnání a konečného výsledku, který zdůvodňuji v závěru kapitoly.

4.1 Výběr počátečních kandidátů

Nejprve je potřeba stanovit si počáteční podmínky, které budou tvořit hrubé síto, díky kterému zvolím 3-4 vhodné kandidáty pro náročnější podrobné srovnání. Na internetu lze v současné době nalézt nepřeberné množství Javascriptových frameworků, nicméně spousta z nich trpí zásadními nedostatky.

Jako úplně hlavní požadavek, který je velmi důležitý pro udržitelnost projektu, vidím solidní zázemí, nejlépe velkou firmu nebo projekt, který za frameworkem stojí. Jistota, že vývoj použité nadstavby nezanikne do roka od dokončení aplikace, je zcela žádoucí. Podpora projektu, jehož framework není vyvíjen a ani záplatován proti bezpečnostním dírám, je noční můrou snad každého vývojáře. Z tohoto důvodu budu volit pouze frameworky, za nimiž stojí větší softwarová firma nebo velmi početná komunita a zároveň aktuálně na frameworku běží minimálně dva projekty velkého rozsahu.

Dalším zcela nutným požadavkem je volná licence typu MIT nebo BSD. Bez ní nemohu framework vůbec použít.

V neposlední řadě musí být výsledný framework vhodný pro tvorbu uživatelských rozhraní. Téměř v každém z nich bude nejspíše tvorba UI možná, mně jde ale primárně o to, aby byla i snadná. Velmi pomůže i jednoduchá rozšiřitelnost o další balíčky, abych mohl například zvolit knihovnu pro komunikaci s API.

A nakonec chci volit nadstavbu, která má kvalitní dokumentaci a aktivní fórum nebo komunitu na službě StackOverflow, aby nebyl vývoj zbytečně brzděn.

Všechny požadavky pro úvodní volbu by se daly shrnout takto:

1. Vývojářem je velká firma nebo velmi početná komunita.
2. Na frameworku aktuálně běží minimálně dva velké projekty.
3. Licence MIT nebo BSD.

4. Vhodný pro tvorbu UI.
5. Snadno rozšiřitelný balíčky třetích stran.
6. Kvalitní dokumentace a fórum nebo komunita na StackOverflow.

Prošel jsem značný počet používaných frameworků [2] [4] [7], profiltroval je výše zmíněnými požadavky a nakonec se omezil na první 3 kandidáty, kteří podmínkami prošli nejlépe. Výběr úvodních kandidátů vypadá následovně:

- **AngularJS**[1]
- **React**[2]
- **Ember**[3]

4.2 Popis zvolených frameworků

4.2.1 AngularJS

AngularJS je framework od společnosti Google na vytváření dynamických webových stránek. Je to nejznámější a nejspíše i nejpoužívanější [4] Javascriptový framework na tvorbu SPA. Mezi jeho hlavní výhody patří vysoká rychlost vytvoření a nasazení kódu (využívá *two-way data binding*[4]), vysoká přizpůsobitelnost vlastním potřebám (nejsou striktní pravidla jakou použít architekturu — někdy se jeho návrhovému vzoru říká MVW, čímž je myšleno model-view-whatever [4]).

Příklady projektů většího rozsahu, které využívají AngularJS [13]:

- Youtube[5],
- weather.com[6],
- freelancer.com[7].

4.2.2 React

React je komponentní framework pro tvorbu uživatelských rozhraní od společnosti Facebook. Je založen na návrhu jednosměrného toku dat (*unidirectional data flow*[8]), což znamená, že vzhled každé komponenty v každém okamžiku odpovídá jejímu stavu (datové struktuře) [7]. Změny stavu jsou vytvářeny událostmi a komponenty na změnu stavu reagují automaticky.

Doporučuje se spolu s frameworkem React využívat i stavový manažer, např. **Flux** nebo **Redux**. Stavový manažer dokáže přetvořit React z frameworku pro tvorbu UI na univerzální framework pro front-end tak, že povýší stav komponent na globální stav aplikace. Protože komponenty Reactu (a tedy celé UI) jsou ve skutečnosti *funkcí stavu*, dává tímto stavový manažer frameworku velkou sílu [6]. Příklady projektů většího rozsahu, které využívají React [5]:

[1] angularjs.org
[2] facebook.github.io/react
[3] www.emberjs.com
[4] Přístup, kdy se aktualizace modelu okamžitě promítá do UI a naopak změny UI elementů uživatelem se opět propagují zpět do modelu.
[5] www.youtube.com
[6] www.weather.com
[7] www.freelancer.com
[8] Přístup, kdy se aktualizace modelu okamžitě promítá do prvků UI, které ji dále propagují potomkům. Existuje jediný „zdroj pravdy“, kterým jsou data modelu.

- Facebook[9],
- Instagram[10],
- Kiwi.com[11],
- Airbnb[12].

4.2.3 Ember

Ember je framework, za nímž stojí opravdu velká komunita a dříve byl považován za nejlepší framework pro tvorbu Javascriptových aplikací [4]. Je mu vlastní rovněž *two-way data binding* a návrhový vzor MVVM (model-view-viewmodel). Je velmi často využíván na tvorbu velkých a komplexních řešení. Příklady projektů většího rozsahu, které využívají Ember [1]:

- LinkedIn[13],
- Apple Music[14],
- Twitch[15].

4.3 Vzájemné porovnání

4.3.1 Kritéria

Všechny tři vybrané frameworky jsou použitelné a vhodné pro tvorbu projektu, jaký popisuje tato práce. Úkolem je však vybrat ten nejvhodnější a k tomu účelu je zapotřebí definovat kritéria, vycházející z nároků na aplikaci.

Pro zajištění modularity je nutné využívat izolovaný komponentní návrh. Ještě vyšším stupněm potom je, pokud umí framework pracovat s třídami představenými v normě ECMAScript 2015[16]. Vývoj objektově-orientované aplikace přináší další výhody pro izolaci a snadnou nahraditelnost tříd.

Důležitým kritériem bude také rychlost a možnost optimalizace způsobů renderování frameworku, aby byl zajištěn požadavek na aplikaci týkající se výkonu a odezvy.

Protože se bude nakonec jednat o aplikaci většího rozsahu, která bude dále rozvíjena, je velice důležitá udržovatelnost a další rozšiřitelnost celého projektu. Tyto vlastnosti u front-endové aplikace značně zvyšuje použití principu jednosměrného toku dat, neboli *unidirectional data flow*, který na rozdíl od přístupu *two-way data binding* zvyšuje přehlednost, pochopitelnost a explicitnost kódu [12].

V neposlední řadě je pro mě důležitá určitá volnost frameworku. Je podstatné, aby způsob vývoje nebyl zcela podsunutý, ale byla zde určitá míra přizpůsobitelnosti potřebám projektu — například možnost používat další knihovny spolu s frameworkem pro dosažení lepších výsledků v určité oblasti (např. výkon, nebo podpora).

Následuje výčet kritérii shrnutý do jednoho seznamu:

[9] www.facebook.com
[10] www.instagram.com
[11] www.kiwi.com
[12] www.airbnb.com
[13] www.linkedin.com
[14] www.apple.com/music
[15] www.twitch.com
[16] developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

- komponentní návrh,
- objektově-orientovaný vývoj,
- využití nejnovějších verzí Javascriptu,
- výkon a možnosti jeho optimalizace,
- jednosměrný tok dat,
- přizpůsobitelnost.

4.3.2 Výsledky

Vycházením z dokumentací k jednotlivým frameworkům a čerpáním z internetových zdrojů [2] [8] [14] jsem jim jednotlivě přidělil bodové ohodnocení v rozsahu 0–5 bodů pro každé dílčí kritérium. Tyto výsledky jsem pro přehlednost zapracoval do tabulky 4.1 níže. U některých hodnocení jsem přidal i vysvětlivku, protože by nemusel být zcela zřejmý důvod právě takového počtu bodů.

Kritérium	AngularJS	React	Ember
Komponentní návrh	2[17]	5	3
Objektově-orientovaný vývoj	3[18]	5	0
Využití ECMAScriptu 2016	5	5	3[19]
Výkon a optimalizace	3[20]	5[21]	4
Jednosměrný tok dat	3[22]	5	0
Přizpůsobitelnost	3	4	1[23]
Výsledné skóre	19	**29**	11

Tabulka 4.1: Srovnání kandidátů na front-endový framework dle zvolených kritérií

4.4 Závěr

Jednoznačným vítězem bodového srovnání se stal React. AngularJS velmi trpěl tím, že existuje markantní rozdíl mezi jeho první, více používanou verzí, a verzí 2.0, kde byly představeny velké změny, které někdy i rozdělovaly komunitu vývojářů [8]. Dle mého názoru měl být raději vytvořen nový framework, namísto jen nové verze, protože to vnáší zmatek i do dokumentace a vláken na StackOverflow, kde často není ani jasné, které verze Angularu se pomoc týká. Ember se nakonec ukázal jako ne moc vhodný kandidát — vstupními požadavky prošel velmi snadno, ale s detailními kritérii bylo více problémů, zejména chybějící využití tříd a nedokonalý komponentní návrh.

[17] Komponenty mají velmi komplexní a fixní strukturu a nejsou příliš znovupoužitelné [14].
[18] Nedá se vhodně využít pro view, spíše pro controllery a modely.
[19] Chybějící možnost vyvíjet součásti jako třídy.
[20] Two-way data binding a nutnost používat objekty pro sledování stavu UI komponent (tzv. watchery) je velkou výkonovou nevýhodou u větších projektů. Nicméně tento problém adresují v dalších verzích. [14].
[21] Orientace na vysoký výkon, používání virutální kopie DOMu a promítání pouze reálných změn [8], možnost ovlivňovat, kdy se má komponenta renderovat — metoda shouldComponentUpdate.
[22] Podporováno až v dalších verzích a jen volitelně.
[23] Velmi silná názorovost, jakou architekturu použít [8].

Na základě výsledků výběru jsem se rozhodl pro aplikaci použít knihovnu React. Paradoxně se ani nejedná o framework[24], ale doplněním o vhodné nástroje (zejména doporučovaný Redux [7]) se v podstatě frameworkem stává.

[24] Na vlastních stránkách se prezentují jako Javascriptová knihovna.

Kapitola 5

Návrh a implementace

Tato kapitola popisuje celou etapu vývoje projektu, rozdělenou na návrh řešení a popisy realizace klientské a serverové části a jejich zabezpečení.

5.1 Návrh řešení

Kvůli obecným požadavkům na jednoduchost a snadnou vložitelnost do cizího systému jsem se rozhodl aplikaci rozdělit do dvou nezávislých funkčních celků:

- Front-endová samostatná Javascriptová aplikace.
- Back-end REST API.

Toto logické rozdělení usnadní realizaci většiny zmíněných požadavků. Obě dvě části spolu budou komunikovat, ale nebudou na sobě závislé (klientská část může např. začít používat jiné API).

Následující podkapitoly rozebírají nezávisle vývoj obou těchto součástí. Na závěr se budu věnovat nově vzniknuvšímu požadavku na zabezpečení komunikace obou částí.

5.2 Klientská část

5.2.1 Volba nástrojů pro vývoj

Určitý čas jsem věnoval přípravě vhodného prostředí pro vývoj aplikace v Reactu. Věděl jsem, že investovaný čas se mi vrátí a nadále budu pracovat efektivněji. Prošel jsem mnoho internetových zdrojů (nebudu citovat, jednalo se hlavně o diskuzní fóra, ať už oficiální na stránkách Reactu nebo Quoru či Reddit) a nakonec jsem si zvolil plně modifikovatelný a v Javascriptu vytvořený editor Atom[1].

Tento editor lze vylepšovat o vlastní či komunitní balíčky a pro vývoj v Reactu a Reduxu je jich spousta. Dále se dá připojit i linter, což je nástroj, kontrolující syntaxi i konvence psaní kódu — umožňuje definovat pravidla, která se musí dodržovat[2]. Při sestavování sady pravidel jsem vycházel z respektované existující sady StandardJS[3]. Jelikož jsou syntaktická pravidla jazyka Javascriptu velmi volná (např. není většinou povinné ani psaní středníků [11]), dodržování určitých standardů povede k vyšší jednotnosti kódu a vyšší univerzalitě použití (bude dobře čitelný i pro další vývojáře).

[1] atom.io

[2] například, že mezi klíčovým slovem a následnou otevírací závorkou musí být mezera, apod.

[3] standardjs.com

Později jsem si editor vylepšil o vlastní mapování kláves, na které jsem zvyklý z jiných IDE, a doplnil pár vlastních maker, specifických pro vývoj React+Redux aplikace.

Krom nástroje pro editaci kódu je nutné zvolit i nástroj pro jeho transpilaci[4] a případně i vyhotovení produkčního sestavení. Na tomto poli se nástrojů pohybuje více, ale mám dobré zkušenosti s nástrojem Webpack[5], proto jsem jej zvolil znovu. Tento nástroj navíc podporuje technologii Hot Module Replacement[6], která značně urychluje vývoj aplikace v Reactu.

Celé vývojové prostředí vyžaduje instalování mnoha balíčků. Tento úkol jsem si zjednodušil použitím správce balíčků npm[7], což je výchozí správce balíčků pro aplikace běžící v prostředí Node.js, kterou tato je. Tento konzolový nástroj kromě zjednodušení samotného procesu instalace modulů dále automaticky řeší všechny závislosti, které mezi sebou balíčky mají — např. dva různé balíčky vyžadují rozdílnou verzi třetího. Všechny závislosti jsou zapsány v souboru **package.json**, jehož obsah se nachází v příloze A.

5.2.2 Grafické uživatelské rozhraní

Jak již bylo zmíněno v analýze požadavku na ovladatelnost uživatelského rozhraní (viz sekce 3.1.1), výsledné rozhraní musí být *jednotné*, *uživatelsky přívětivé* a *intuitivní*.

Jednotnost celého prostředí pomůže zaručit použití grafické knihovny. Z hlediska intuitivnosti je potom lepší volit takovou knihovnu, která je mezi lidmi dobře známá. Mezi další nároky na případnou knihovnu řadím pěkný vzhled, dostupné hotové komponenty a přípravu jejich uživatelské interakce, kvalitnost a dostupnost dokumentace a také snadnost integrace. Jako nadějné kandidáty jsem zvolil Bootstrap[8], Material UI[9] a Elemental UI[10].

Nakonec jsem se rozhodl zvolit knihovnu **Material UI**, která mi přišla nejlépe zpracovaná a s nejlépe připravenými hotovými komponentami. Dále je mi velmi sympatická sada grafických pravidel Material Designu[11], která má kladný vliv na uživatelskou přívětivost a zapojení [15].

Kromě využití knihovny, která zajistí grafickou jednotnost a podpoří uživatelskou přívětivost, je nutné věnovat určité úsilí návrhu rozhraní tak, aby bylo logické a intuitivní. Toho bude docíleno důkladným návrhem, konzultací s uživateli a hlavně uživatelským testováním.

5.2.3 Architektura aplikace

Aplikace je napsána v aktuálně nejnovější verzi jazyka Javascript (verze ECMAScript 2016[12]), využívá tedy objektově-orientovaný návrh a každá komponenta je třídou dědící od React.Component. Tento přístup přináší mnoho výhod hlavně z hlediska udržitelnosti projektu, je však nutné výsledný kód transpilovat do prohlížeči podporovaného Javascriptu (ECMAScript 5.1) pomocí překladače, například Babelu[13]. Tento překlad lze plně automatizovat ve výše zmíněném nástroji Webpack, a dokonce zachovat i podporu pro Hot Module Replacement (HMR). Návod na zprovoznění vývojového prostředí je obsažen v souboru readme.md, který je přidružen ke zdrojovým souborům.

[4] teamtreehouse.com/library/what-is-transpiling
[5] webpack.js.org
[6] webpack.github.io/docs/hot-module-replacement.html
[7] www.npmjs.com
[8] react-bootstrap.github.io
[9] www.material-ui.com
[10] elemental-ui.com
[11] material.io/guidelines/material-design/introduction.html
[12] cs.wikipedia.org/wiki/ECMAScript
[13] babeljs.io

Aplikační část zdrojových souborů je rozdělena do dvou hlavních adresářů **src** a **styles**. Adresář **src** obsahuje veškeré Javascriptové zdrojové kódy aplikace, které jsou rozděleny logicky do souborů podadresářů s následující strukturou:

- **actions/** — obsahuje soubory definující všechny akce, jaké může aplikace odbavit. Odbavení (dispatch) je jediný způsob, jak se může změnit globální stav aplikace na frameworku React.
- **api/** — obsahuje soubory definující adresy koncových bodů API a jejich rozhraní.
- **components/** — obsahuje všechny komponenty UI v hierarchické logické struktuře.
- **containers/** — obsahuje všechny kontejnery ke komponentám v úplně identické hierarchické struktuře jako komponenty. Kontejner je pouze obálka pro komponentu, která jí zpřístupňuje část globálního stavu a akce, které může odbavit. Tímto je zajištěna znovupoužitelnost komponent napříč systémy.
- **lib/** — obsahuje vlastní třídy se sdílenou aplikační logikou.
- **reducers/** — obsahuje tzv. reducery, neboli funkce, které na základě odbavované akce provádějí změnu stavu aplikace. Jsou součásti Reduxu.
- **storages/** — obsahuje datové repozitáře — in-memory databáze. Jsou plněny stahovanými daty z API a fungují jako datové modely.
- **themes/** — obsahuje pouze definice barevných témat pro administrační a rezervační část. Je takto umožněna snadná úprava barevnosti aplikace.
- **App.js** — je kořenová komponenta celé aplikace.
- **config.js** — je soubor obsahující základní konfiguraci jako velikost mřížky, konstanty pro její interakci apod.
- **definitions.js** — je soubor obsahující výčtové typy pro použití v aplikaci, např. seznam typů nějakého pohledu.
- **index.js** — je hlavní spouštěcí soubor, který vnoří kořenovou komponentu do stránky a předá jí parametry jako režim funkce a autorizační token.

Adresář **styles** pak obsahuje veškeré grafické styly pro aplikaci v soubrech formátu LESS[14]. Tyto soubory jsou také za pomoci webpacku a jeho balíčků transpilovány do běžného CSS.

Mimo tyto adresáře se zde nachází už jen složka **deploy**, do které je vkládáno výsledné produkční sestavení. Složka však obsahuje některé výchozí statické soubory, kterými jsou:

- **.htaccess** — je konfigurační soubor pro server Apache pro nastavení GZIP komprese zdrojů a zamezení přístupu k jiným souborům, než je index.html (pouze pro demonstrační nasazení, není součástí vnořované aplikace).
- **favicon.ico** — je ikonka pro stránku s demonstrací aplikace.

[14] lesscss.org

- **proxy.php** — je soubor pro přesměrovávání požadavků na skutečný server API. Při dotazech na jiný server přímo z HTML aplikace dochází k problémům na některých starých proxy serverech, které nerespektují nastavení direktiv Cross-origin resource sharing[15]. Pro zajištění funkčnosti všem klientům je proto potřeba protáhnout veškerou komunikaci skrze proxy skript na stejné doméně. Tento skript bude MUSET být přítomen na kořenovém adresáři webu, kde bude aplikace nasazena.

V hlavním adresáři aplikace se dále nachází tyto soubory:

- **.babelrc** — nastavení nástroje Babel pro transpilaci kódu, obsahuje seznam použitých balíčků a direktiv.
- **.eslintrc** — nastavení nástroje ESLint pro dodržování konvencí psaní kódu.
- **build.cmd** — skript pro vytvoření produkčního sestavení aplikace.
- **index.html** — šablona demo aplikace s nasazením KORESu.
- **index.tpl.html** — šablona demo aplikace s nasazením KORESu pro produkční sestavení.
- **package.json** — soubor pro správce balíčku npm s definicí používaných balíčků (jeho obsah se nachází také v příloze A).
- **README.md** — stručný popis aplikace a návod pro zprovoznění vývojového režimu a vytvoření produkčního sestavení.
- **server.js** — skript pro spuštění vývojového serveru na localhostu.
- **webpack.config.js** — konfigurace vývojového nástroje Webpack.
- **webpack.config.prod.js** — konfigurace nástroje Webpack pro produkční sestavení.

5.2.4 Modularita

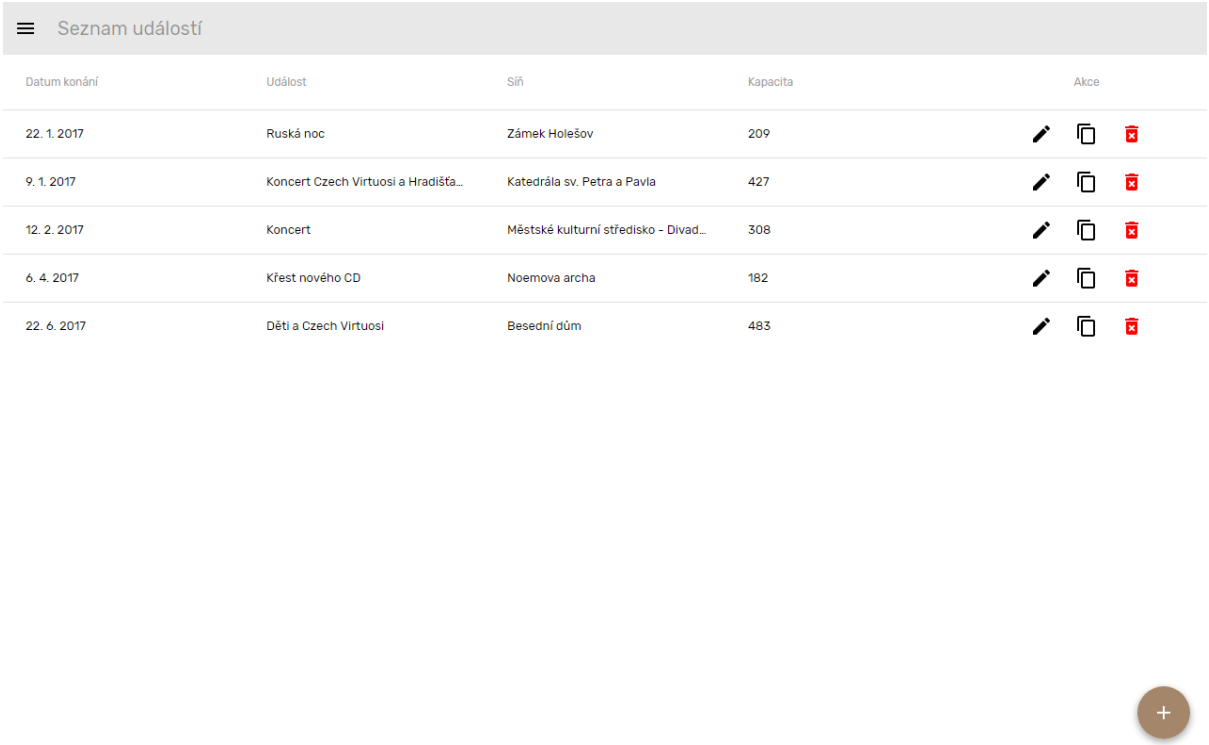
Aby systém disponoval určitou mírou upravitelnosti, musí být částečně modulární — výměnou jednoho modulu pak lze uskutečnit neinvazivní úpravu, která neovlivní zbytek systému (viz 3.1.2 Upravitelnost).

V případě této aplikace je prvním stupněm modularity rozdělení na serverovou a klientskou část. Obě tyto části by však také měly být jistou měrou modulární i samy o sobě. Použití komponentního návrhu frameworku React umožňuje řešit případné modifikace například obměnou jedné či více komponent, což je izolovaná úprava, která nijak neovlivní funkčnost zbytku systému.

5.2.5 Správa událostí v síních

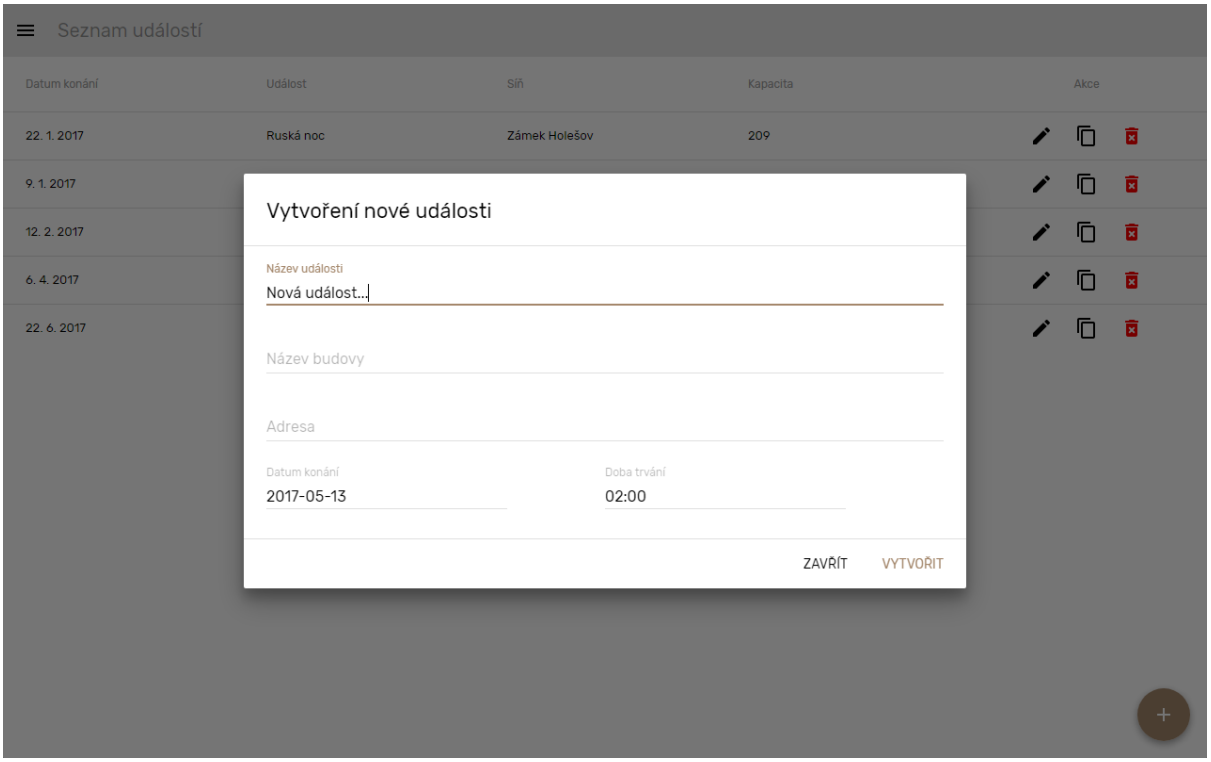
Hlavní a výchozí obrazovkou editačního rozhraní aplikace je seznam událostí v síních, implementovaný jako tabulka (viz obrázek 5.1). Operace nad událostmi (úprava, klonování, mazání) jsou řešeny akčními tlačítky v každém řádku tabulky a jedním hlavním tlačítkem mimo tabulku pro vytvoření nové události.

[15] cs.wikipedia.org/wiki/CORS

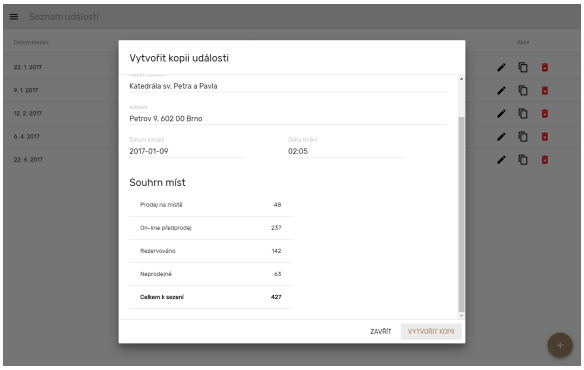


Obrázek 5.1: Seznam událostí v síních.

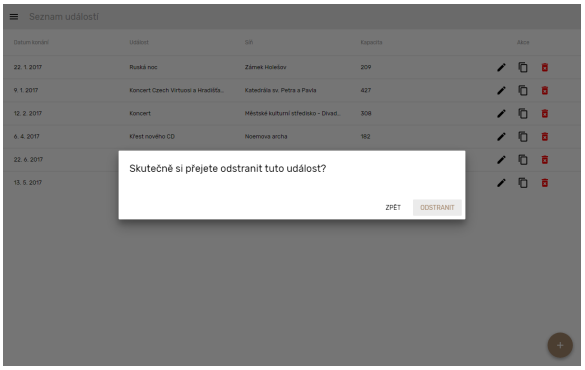
Úprava, klonování a vytváření nové sekce vyvolá pohled detailu s políčky pro úpravu atributů (viz obrázek 5.2). Při klonování a úpravě jsou tyto hodnoty předvyplněny dle aktuální události v síni (obrázek 5.3) a navíc jsou zobrazeny statistiky míst. Při pokusu o odstranění sekce je z důvodu ochrany před nechtěným kliknutím ještě zobrazen dotaz s potvrzením akce (obrázek 5.4).



Obrázek 5.2: Vytvoření nové události.



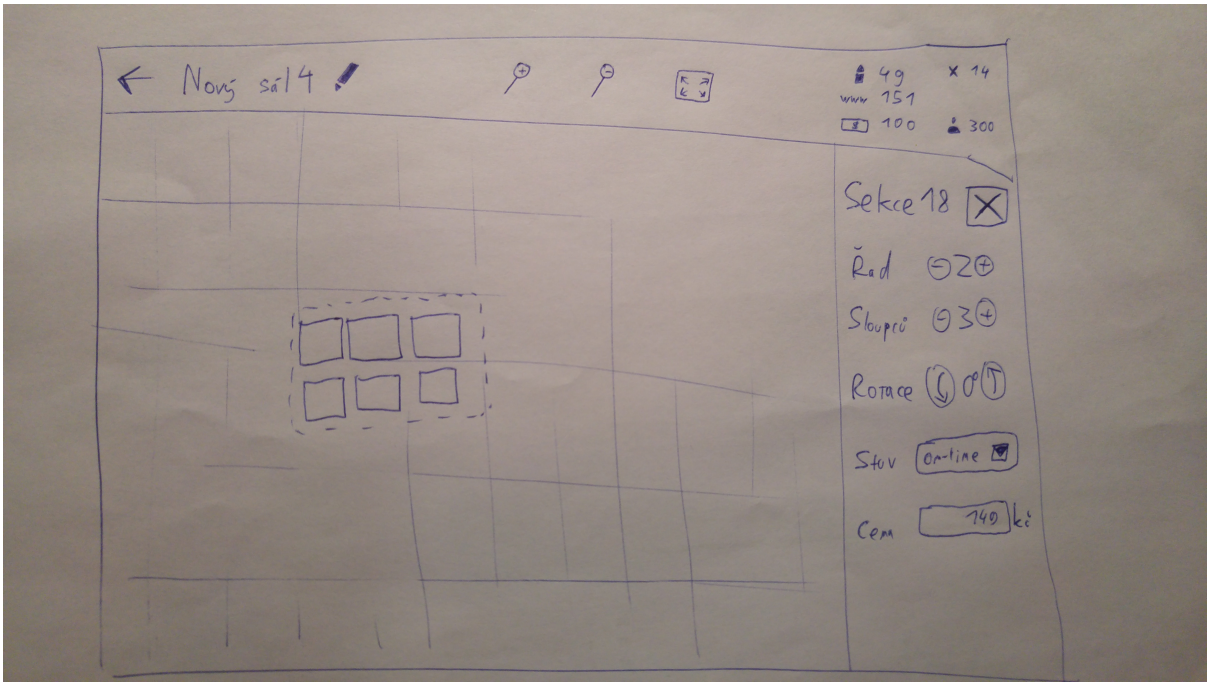
Obrázek 5.3: Vytvoření kopie události.



Obrázek 5.4: Odstranění události.

5.2.6 Konfigurace síní

Celou konfiguraci jsem rozhodl řešit jediným pohledem, ve kterém se nachází všudypřítomná nástrojová lišta a v pravé části panel s detailem vybrané sekce, který bude zobrazen, pouze pokud je některá sekce aktivní. Zbytek plochy je vyčleněn pro zobrazení mřížky a sekcí.



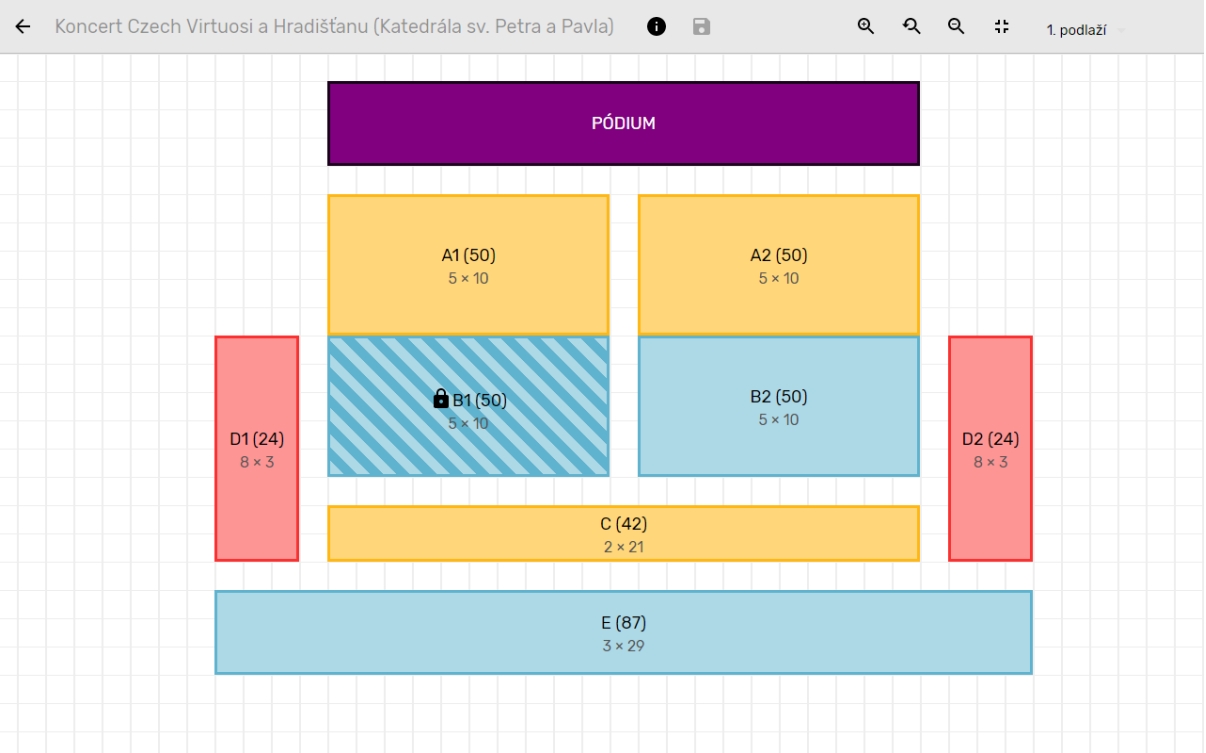
Obrázek 5.5: Mockup uživatelského rozhraní konfigurace síně.

Tuto představu jsem zhmotnil do podoby mockupu rozhraní, které je zobrazeno na obrázku 5.5. V nástrojové liště jsem zamýšlel (zleva) tlačítko pro návrat na seznam událostí v síních, název události, tlačítko pro úpravu detailu události, tlačítka pro přiblížení/oddálení pohledu na mřížku, tlačítko pro přechod do celoobrazovkového režimu (editace bude vyžadovat mnoho prostoru na obrazovce a může se stát, že hostitel aplikace jí v základu neposkytne ani celou šířku stránky) a nakonec statistiky jednotlivých míst (počty dle typu a stavu).

V pravé části je vidět panel detailu sekce, kde zamýšlím možnost ruční doúpravy velikosti, možnost nastavení názvu sekce, ceny sedadel a již zmíněného typu a stavu.

Množinu hodnot stavů i typů míst si bude moci stanovit klient.

Od zamýšleného grafického návrhu jsem se nakonec tolik nevzdálil, jak je vidět na snímcích obrazovky výsledného rozhraní (obrázky 5.6 a 5.7).



Obrázek 5.6: Výsledné rozhraní konfigurace síně.

Na obrázku 5.6 je možné vidět výchozí stav konfiguračního rozhraní bezprostředně po zahájení editace síně pomocí tlačítka s ikonkou tužky. Nástrojová lišta vychází z velké částí z návrhu, ale několik položek je zde navíc. Jmenovitě se jedná o:

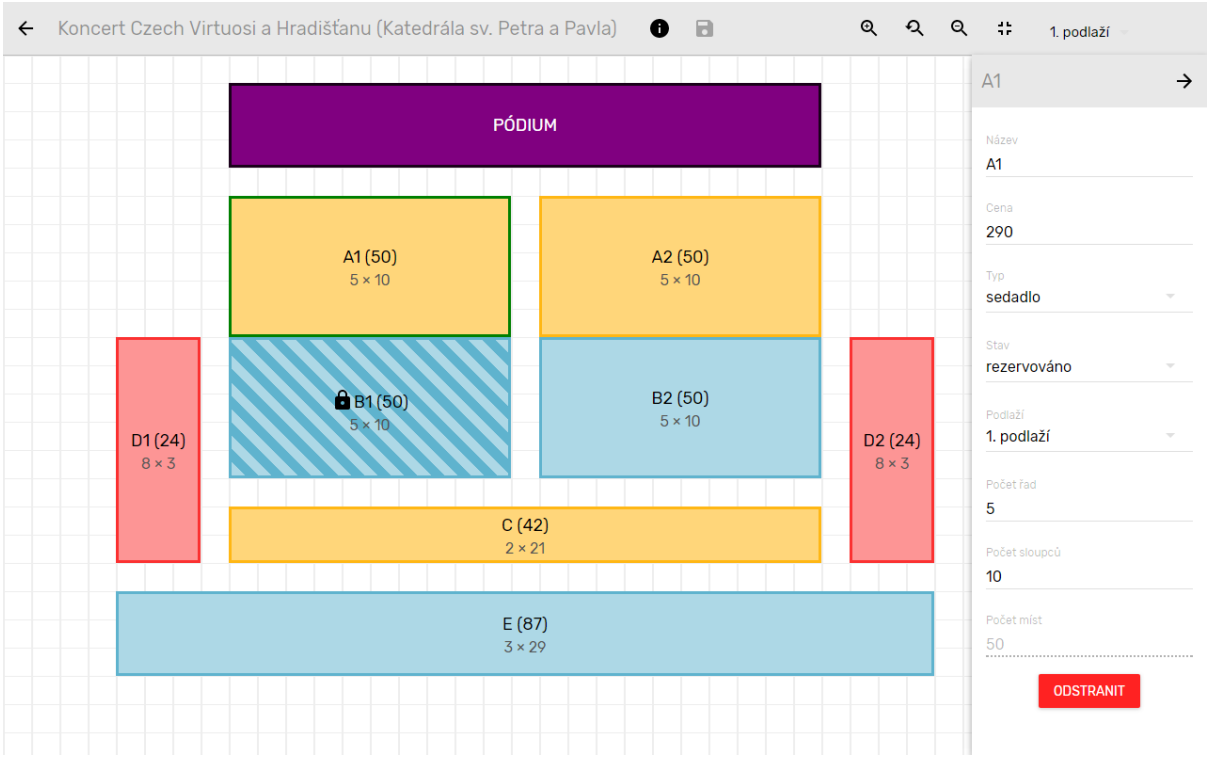
- Tlačítko ⓘ, které nahrazuje ikonku tužky — spustí zobrazení detailu události v síni (stejné, jaké se používá v seznamu událostí pro úpravu události a jaké se nachází na obrázku 5.3). V tomto detailu se nachází také statistiky míst, které jsem z nástrojové lišty naopak vypustil.
- Tlačítko 🖫, které uloží všechny aktuálně provedené změny v síni. Tlačítko není aktivní, pokud k žádným změnám nedošlo, nebo pokud všechny změny vedly k návratu na původní stav. Pokud je tlačítko aktivní a uživatel se pokusí opustit síň (např. návratem na seznam událostí v síních), je mu zobrazeno upozornění o neuložených změnách (viz obrázek 5.11).
- Tlačítko 🔍, které resetuje veškeré nastavení přiblížení. Uznal jsem za vhodné jej přidat, jelikož není vždy jasné, jaké je aktuální přiblížení.
- Políčko pro volbu aktuálního patra. Volbou patra se přepíná zobrazení sekcí — každá sekce může být pouze v jednom patře[16]. Ukázku přepnutí patra je možné vidět na obrázku 5.9.

Každá sekce je zbarvena dle jejího určeného stavu. Například sekce pro online rezervace jsou vyznačeny modře. Sekce, do kterých už byly provedeny uživatelské rezervace, jsou vyobrazeny se šrafováním v barvě okraje a s ikonou zámku (lze vidět u sekce B1). Takovéto sekce lze pouze posouvat, nelze měnit jejich velikost ani jiné parametry (viz obrázek 5.8).

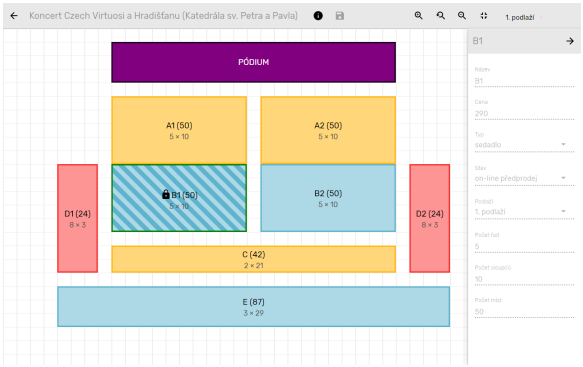
[16] Výjimkou je sekce typu *pódium*, která se zobrazuje napříč všemi patry.

Všechny sekce vyjma pódia mají popisek, který se skládá z označení sekce, celkové kapacity míst (v závorce), a počtu řad a sloupců (šedě ve druhém řádku). Tento popisek se automaticky redukuje, pokud je sekce velmi malá.

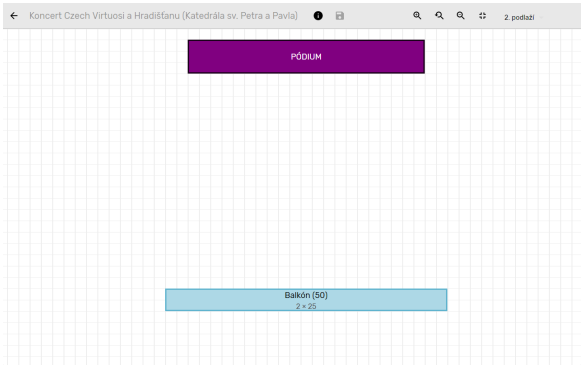
Kliknutím na sekci dojde k její aktivaci a je umožněno sekci upravovat. Tuto situaci lze vidět na obrázku 5.7 níže.



Obrázek 5.7: Konfigurace síně s aktivní sekcí.

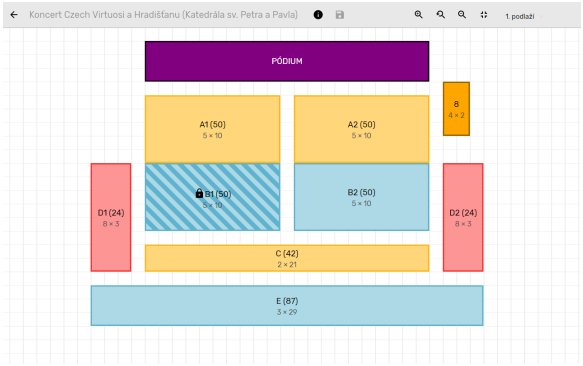


Obrázek 5.8: Konfigurace síně s aktivní sekcí s rezervací.

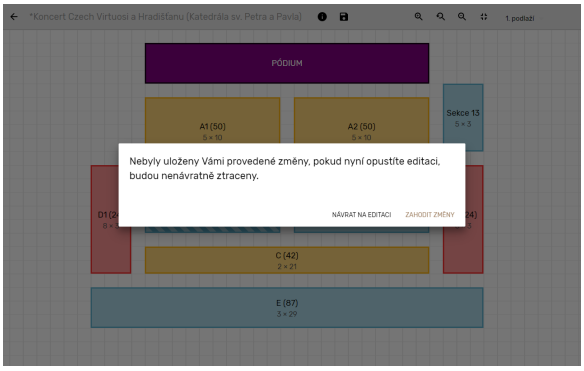


Obrázek 5.9: Konfigurace síně se zobrazením jiného patra.

Tažením myší po volné ploše je vytvářena nová sekce. V jejím popisku se v reálném čase ukazuje aktuální velikost, aby autor přesně věděl, kdy je to akorát. Tuto situaci lze vidět v pravém horním rohu obrázku 5.10.



Obrázek 5.10: Vytváření nové sekce v konfiguraci síně.



Obrázek 5.11: Upozornění o neuložených změnách v síni.

Aby byly splněny požadavky na efektivitu administrace (viz 3.1.4) týkající se minimalizace náročnosti provádění nejčastějších operací, bylo nutné správně využít všechny uživatelské akce myší a vyřešit i jejich vzájemné provázání, protože se někdy překrývají (např. zvolení sekce kliknutím a tažení za roh).

Tento krok by nebyl vhodný, pokud by měla administraci používat široká veřejnost, jelikož ovládání pak není zcela intuitivní a je potřeba se naučit, co které tlačítko myši znamená. Nicméně to není tento případ, administraci bude využívat pouze správce systému, který si ovládání rychle osvojí a bude pracovat efektivně.

Toto ovládání jsem nakonec realizoval následovně: tažení levým tlačítkem – tvorba nové sekce, tažení pravým tlačítkem – posun po ploše, pohyb kolečkem myši – oddálení/přiblížení. Existující sekce lze tažením posouvat po ploše, případně měnit jejich velikost tažením za pravý spodní roh.

5.2.7 Rezervační systém

Aplikace umožňuje běh v režimu rezervačního systému. Tento režim je určen pro nasazení na veřejné části webu, a proto pro jeho používání není vyžadována žádná zvláštní autorizace.

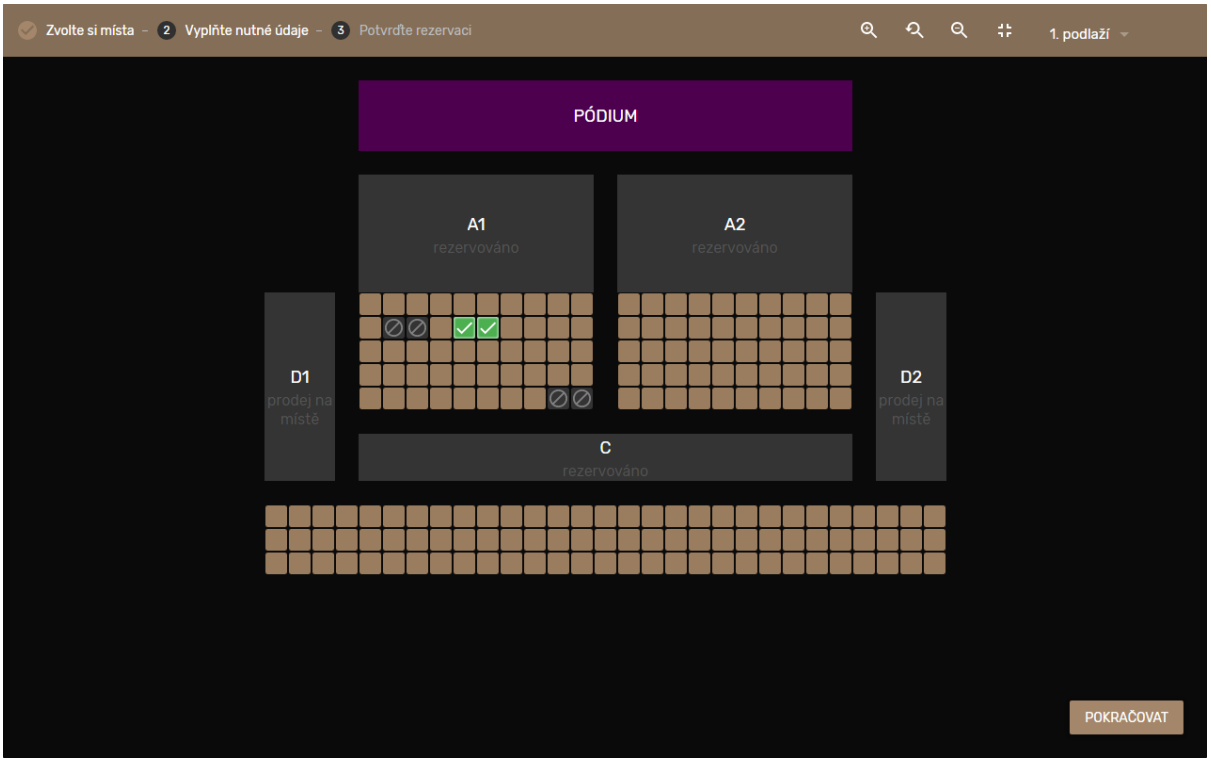
Celý proces rezervace je vícekrokový a je realizovaný více pohledy, které však sdílí hlavní horní lištu, na níž se nachází indikace aktuálního kroku, případně další operace, pokud to daný krok vyžaduje.

První krokem, který lze vidět na obrázku 5.12, je provedení volby míst v grafické reprezentaci síně. Volit lze pouze místa v sekcích, které jsou určeny pro online rezervaci a zároveň takové, které nejsou zarezervované někým jiným (ikona zákazu).

Aktuální volba je indikována ikonou fajfky a zelenou barvou. Bez výběru alespoň jednoho místa není možné v rezervaci pokračovat.

Pohled na sekce je automaticky přizpůsoben oknu — kolem existujících sekcí je utvořena minimální možná obálka a tato tvoří plochu, ve které se lze při rezervaci pohybovat. Není tedy možné odscrollovat pohled na síň mimo zorné pole. Na rozdíl od editačního pohledu zde není možné používat kolečko myši pro přibližování nebo oddalování — to lze provést už pouze tlačítky na liště a kolečko myši plní svoji klasickou funkci posouvání. Tento způsob byl zvolen, protože je pro uživatele obvyklejší, a nepředpokládá se taková nutnost používat při rezervaci přiblížení. Nadále je možné přepínat patra a vstoupit do celoobrazovkového režimu.

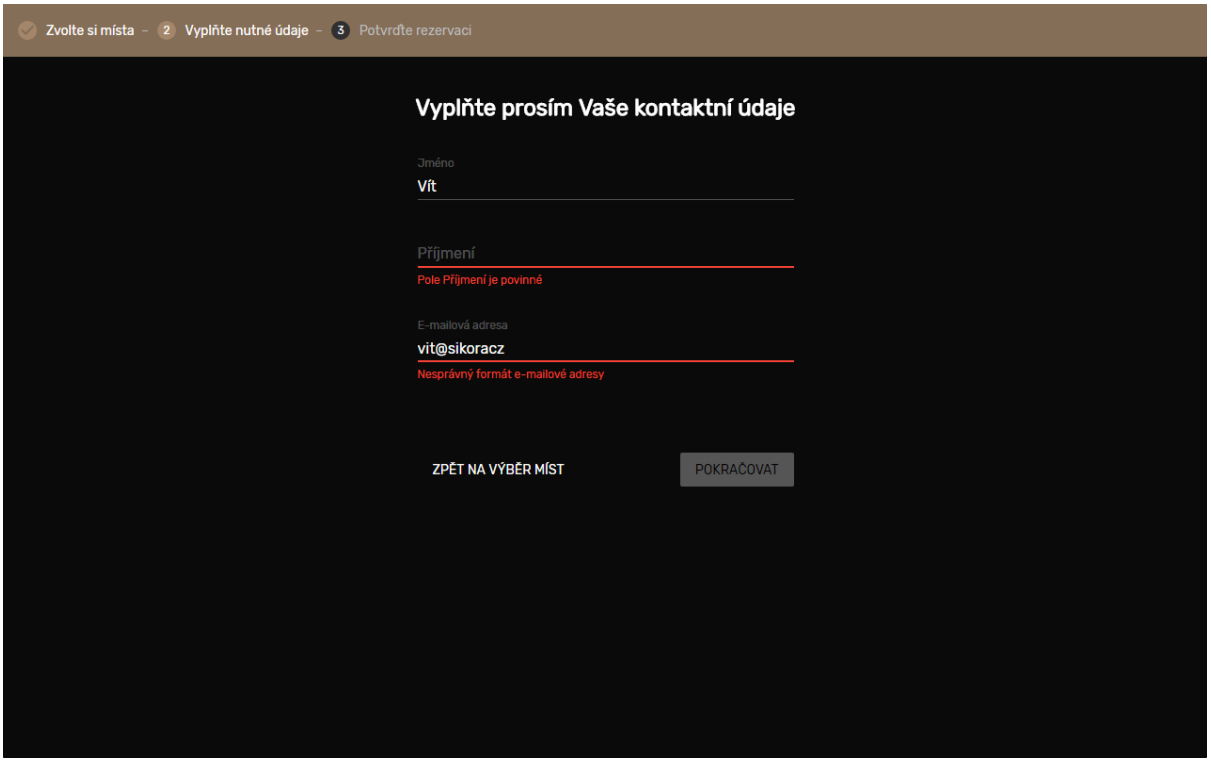
Přesun mezi kroky rezervace je možný nejen akčními tlačítky ve spodní části pohledů, ale i přímo kliknutím na navigaci (pokud je přechod z hlediska ověření dokončení kroku možný).



Obrázek 5.12: Rezervace (krok 1/3) — volba míst.

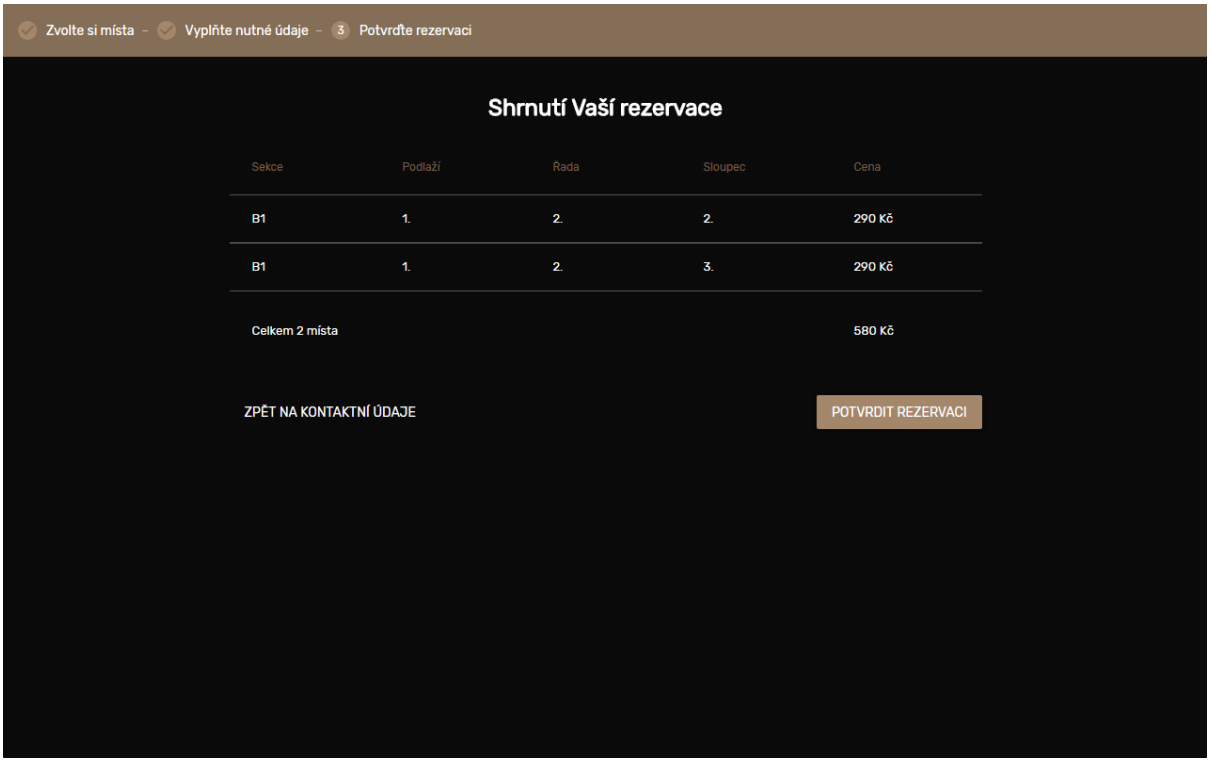
Druhým krokem provedení rezervace je vyplnění povinných údajů, bez kterých nelze rezervaci přijmout. Mezi tyto patří celé jméno a e-mailová adresa objednatele. Vyplnění formuláře je validováno nejen na přítomnost hodnot, ale v případě e-mailové adresy i na její formát — toto může odhalit některé typy překlepu v adrese. Bez splnění této validace není možné v rezervaci pokračovat.

Pohled na formulář pro vyplnění údajů se nachází na obrázku 5.13.



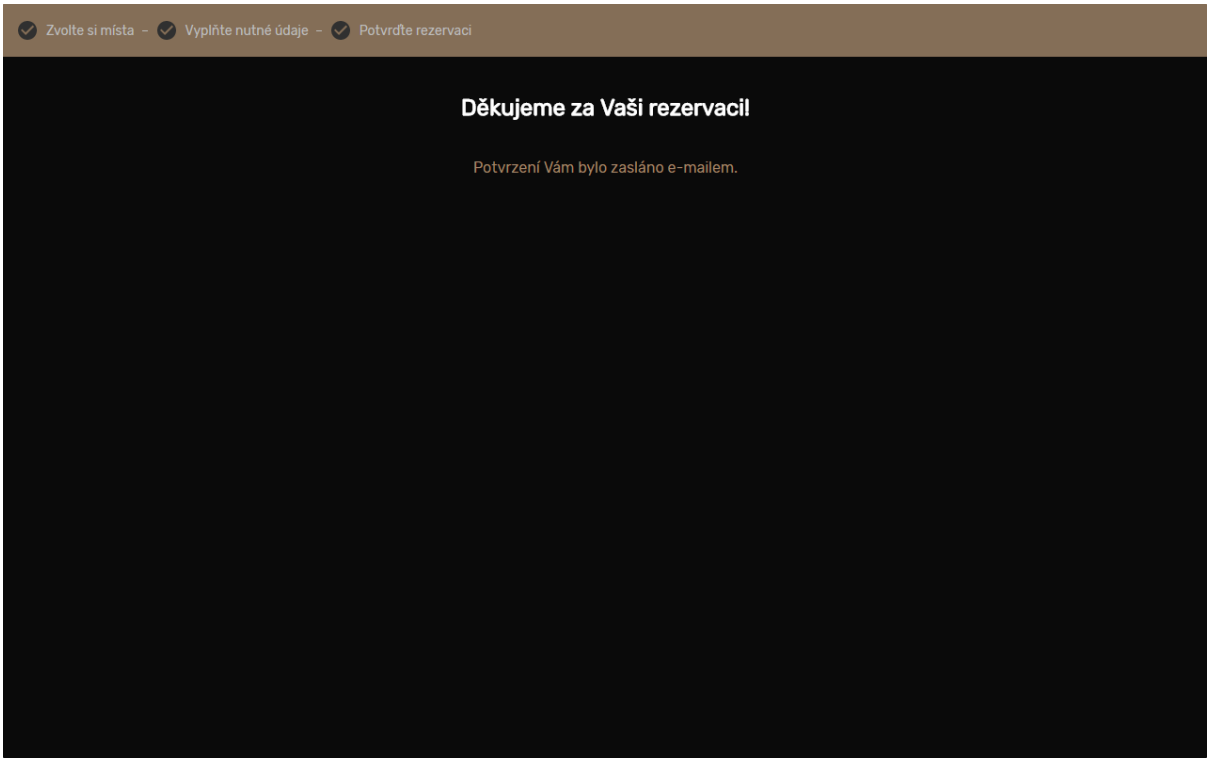
Obrázek 5.13: Rezervace (krok 2/3) — vyplnění údajů.

Třetím krokem rezervace je pohled na její souhrn, sloužící pro kontrolu objednaných míst a potvrzení celé akce (viz obrázek 5.14). Souhrn je řešen tabulkou s důležitými údaji včetně ceny. Nechybí ani součet všech míst a celková cena.



Obrázek 5.14: Rezervace (krok 3/3) — potvrzení.

V případě úspěšného vytvoření rezervace se uživateli zobrazí děkovná hláška (obrázek 5.15) a tím je použití aplikace ukončeno. V případě neúspěchu je uživateli zobrazena hláška obsahující důvod neúspěchu (například, že si dané místo stihl zarezervovat v mezičase někdo jiný) a uživatel má možnost upravit jakékoli údaje a zkusit provést rezervaci znovu.



Obrázek 5.15: Rezervace dokončena.

5.2.8 Vložitelnost do cizího systému

Pro nejlepší integraci do webové stránky vnějšího systému a využití plného potenciálu možné vzájemné interakce je nutné přímé vložení do DOM[17], nikoli např. přes iFrame. Mezi jednoznačné přínosy aplikace běžící přímo v hlavním dokumentu, patří:

- možnost interakce s ostatními prvky v dokumentu (vč. hlavičky dokumentu),
- vyšší rychlost načtení (nemusí se načítat separátní zdroj),
- lepší možnosti responzivního návrhu (souvisí s prvním bodem),
- vyšší bezpečnost webu (v případě načítání z cizího zdroje),
- možnost ovlivnit vzhled prvků aplikace (přetížením CSS pravidel).

Mezi nevýhody pak patří riziko křížení CSS selektorů, ale tento problém je adresován použitím lokálních stylů, kdy jsou názvy tříd a identifikátorů automaticky transformovány tak, aby byly unikátní (např. selektor **#boardGrid** může být převeden při překladu na **#boardGrid__8ae45fc1**). Tuto automatickou transformaci provádí nástroj Webpack s pomocí balíčku **css-loader**.

Aplikace je vytvořena tak, aby její vložení bylo pouze otázkou vytvoření cílového kontejneru s identifikátorem *#kores* a připojení skriptu a stylu KORESu.

[17] Document Object Model — cs.wikipedia.org/wiki/Document_Object_Model

5.2.9 Optimalizace

Respektováním požadavků na výkonnost a efektivitu aplikace jsem věnoval pozornost úsekům výkonného kódu a dbal na jeho patřičnou optimalizaci. Ne však v začátcích implementace, abych se vyhnul častému anti-patternu[18] **premature optimization**, neboli předčasné optimalizaci.

Pozornost o výkon zpracování si žádaly hlavně algoritmy týkající se konfigurace síní, zejména pak ty pro řešení jejich kolizí. Jelikož je řešení kolizí problém se složitostí $\mathcal{O}(n^2)$, je jeho optimalizace velmi vítaná. Např. řešení kolize ořízntím (používá se při tvorbě a změně velikosti sekce) využívá mírnou heuristiku a ve velmi okrajových případech nefunguje zcela korektně. Ale považuji to za vhodný kompromis pro plynulost zmíněných operací. Algoritmy se nacházejí v souboru */src/lib/tools.js*.

5.2.10 Podpora v rozšířených internetových prohlížečích

Vzhledem k použití moderních webových technologií je řešení tohoto problému jednodušší než u tradičních webových aplikací. Právě díky využití překladu a automatické tvorby výsledného kódu je podpora starších prohlížečů pouze otázkou použití balíčků a přepínačů pro transpiler. Např. balíček nastavení **babel-preset-es2015-ie** aktivuje podporu pro IE >= 9.

Rovněž u stylopisů dochází ke transpilingu, a to z jazyka LESS do CSS. Přepínače však někdy neřeší úplně vše a některé grafické prvky bylo nutné testovat v prohlížečích individuálně.

5.2.11 Zotavení při výpadku internetového připojení

Aplikace je schopná pracovat i bez internetového připojení. Od prvního načtení aplikace disponuje všemi nutnými statickými zdroji a pouze v případě potřeby si stahuje další data. Pokud se stažení dat, které je nutné pro vykonání nějaké operace (např. konfigurace síně), nezdaří, aplikace se neukončí s chybou, ale pouze informuje uživatele, že stažení dat selhalo a ať zkusí akci opakovat později. V takové situaci je nadále možné s aplikací pracovat a používat data, která byla stažena dříve a aplikace je má uchované v své in-memory databázi.

5.3 Serverová část

5.3.1 Volba nástrojů pro vývoj a IDE

Před zahájením vývoje back-endové části aplikace bylo nutné učinit rozhodnutí, jaké softwarové vybavení bude k tomuto účelu použito. Jakožto IDE[19] jsem si zvolil NetBeans[20], jelikož je to dle mého názoru nejlepší bezplatné IDE pro vývoj v PHP 7.

Protože si vývoj serverové části vyžádá použití knihoven třetích stran, rozhodl jsem pro využití správce balíčků Composer[21], což je nejznámější správce pro balíčky v jazyce PHP a používá ho takřka každá knihovna (minimálně všechny, o kterých jsem uvažoval).

Composer si udržuje seznam všech nainstalovaných balíčků (závislostí) v souboru composer.json (obdobně jako npm s package.json).

Použity byly tyto knihovny:

[18] Návrhový antivzor (angl. Anti-pattern) v softwarovém inženýrství představuje obecný postup při řešení opakujících se problémů při návrhu počítačových programů, které jsou všeobecně vnímány jako nesprávné nebo neefektivní. [3]

[19] Integrated Development Environment

[20] netbeans.org

[21] getcomposer.org

```
"luracast/restler": "*",
"nette/database": "^2.4",
"nette/di": "^2.4",
"tracy/tracy": "^2.4",
"phpmailer/phpmailer": "^5.2",
"latte/latte": "^2.4"
```

Výpis 5.16: Závislosti v souboru composer.json

5.3.2 Architektura REST API

Po serverové části je požadováno, aby pracovala jako webové aplikační rozhraní. Tím je myšleno, že server nebude mít žádný HTML výstup a klient nebude nikdy přímo přistupovat na jeho URL adresu. Namísto toho bude mít k dispozici různé rozhraní pro práci s konkrétními zdroji a s těmito pracuje na pozadí právě front-endová aplikace.

Pro dosažení jednotnosti a standardnosti takového řešení jsem se rozhodl použít architekturu REST, neboli Representational State Transfer [10], která pro webové API definuje jednotné rozhraní pro práci se zdroji a nad každým určuje 4 základní operace:

1. **Create** – vytvoření nové položky.
2. **Read** – načtení existujících dat.
3. **Update** – aktualizace dat.
4. **Delete** – trvalé odstranění dat.

Souhrnně se tyto operace označují zkratkou **CRUD**. Pro většinu entit (jakou je například sekce) jsou definovány všechny operace, ale není to nutnou podmínkou — může se stát, že nad některou z entit nebude určitá operace implementována a v takovém případě server vrátí odpověď **501 Not Implemented**.

Každý zdroj (entita) má vlastní identifikátor URI (tedy např. kores-api.sovanet.cz/section), na kterých lze nad nimi provádět operace. Typ operace je určen metodou HTTP požadavku: GET odpovídá Read, POST odpovídá Create, PUT značí Update a DELETE je stejnojmenný. Server vždy odpovídá zprávami ve formátu JSON, protože pro Javascript je to přirozený formát, který umí automaticky přeložit do nativních typů (objekty a pole).

Stavbu REST API v PHP může značně urychluje využití frameworku, určeného primárně na vytváření API. Po počátečním hledání jsem volil mezi dvěma kandidáty, a to mikroframeworky **Slim**[22] a **Restler**[23]. Nakonec jsem zvolil Restler pro jeho hlavní orientaci na REST API a přímočarost definice rozhraní, kdy vždy jedna třída odpovídá přesně jednomu zdroji.

Seznam všech zdrojů (koncových bodů) REST API a operací nad nimi vypadá takto:

/hall-event{/:id} [GET, POST, PUT, DELETE]

/section{/:id} [GET, POST, PUT, DELETE]

/seat [GET]

/reservation [POST]

[22] www.slimframework.com
[23] restler3.luracast.com

/reserved-seat [GET]

/save-event-sections/:id [PUT]

Koncový bod *hall-event* se stará o úpravu událostí v síních a podporuje všechny CRUD akce, nicméně kromě akce Read vyžadují všechny administrátorské oprávnění. Vytváření kopií je řešeno jako akce Create, tedy metodou POST, ale k adrese je přidán ještě parametr **?duplicate**.

Bod *section* funguje velmi podobně jako hall-event, i nastavení práv je totožné.

Bod *seat* slouží pouze k získání seznamu všech možných typů a stavů míst a jejich povolených kombinací. K adrese se nepřidává žádný identifikátor a k získání seznamu stačí oprávnění uživatelské úrovně.

V případě bodu *reservation* je možná pouze operace Create a ta je využívána právě při zakládání nové rezervace. Opět stačí pouze uživatelské oprávnění.

Koncový bod *reserved-seat* slouží k získání seznamu všech zarezervovaných míst pro danou událost, aby mohly být v rezervačním systému vizualizovány a mohlo být zamezeno jejich výběru. Opět pouze uživatelské oprávnění.

Koncový bod *save-event-sections* slouží pouze k uložení konfigurace celé síně (uložení všech provedených změn) a má definovanou pouze operaci Update, tedy metodu PUT. Je nutné specifikovat přímo v adrese identifikátor cílové události v síni a v těle požadavku se musí nacházet JSON struktura se všemi sekcemi. Operace lze vykonat pouze s oprávněním administrátora.

Co se týče souborové struktury API, jsou v hlavním adresáři 3 důležité složky: **public**, **src** a **vendor**.

Složka **public** je veřejná a její obsah je přístupný z webového serveru. Obsahuje pouze soubory **.htaccess** (primárně pro nastavení překladu adres) a **index.php**, který zavádí systém (načítá soubor **src/bootstrap.php**).

Složka **src** obsahuje autorské zdrojové kódy. Nachází se v ní soubory **bootstrap.php** pro zavedení systému, **config.neon** pro nastavení serveru (primárně definice vkládání závislostí), a **utils.php**, což je knihovna obecných užitečných funkcí. Obsažené adresáře jsou **Exception** (třídy výjimek), **Model** (třídy modelů), **Presenter** (třídy koncových bodů API) a **Templates** (HTML šablony; pouze pro emailové notifikace). Složka **vendor** pak obsahuje zdrojové kódy třetích stran (balíčky nainstalované přes Composer), jakými jsou například framework Restler, či části frameworku Nette. Tyto nejsou součástí zdrojových kódu aplikace.

V hlavním adresáři API se pak také nachází soubor **composer.json**, jehož význam už byl popsán v sekci 5.3.1.

5.3.3 Modularita

Obecným požadavkem na serverou část byla určitá míra modularity, kterou lze dosáhnout pomocí techniky vkládání závislostí. Rozhodl jsem se použít kvalitní nástroj Dependency injection z frameworku Nette[24], se kterým mám dobré zkušenosti. Je sice nutné předem v konfiguračním souboru zahrnout všechny třídy se kterými má pracovat, dále je ale zcela automatický a objekty instanciuje přesně v takovém pořadí, aby nedošlo ke kolizi závislostí (což nevylučuje případné cyklické závislosti, na které však okamžitě upozorní výjimkou).

5.3.4 Realizace databáze

Nezbytností serverové části je bezpochyby připojení k databázi. Volba vhodné databázové technologie byla velmi přímočará — zvolil jsem technologii MySQL (konkrétně její výkonnější odnož

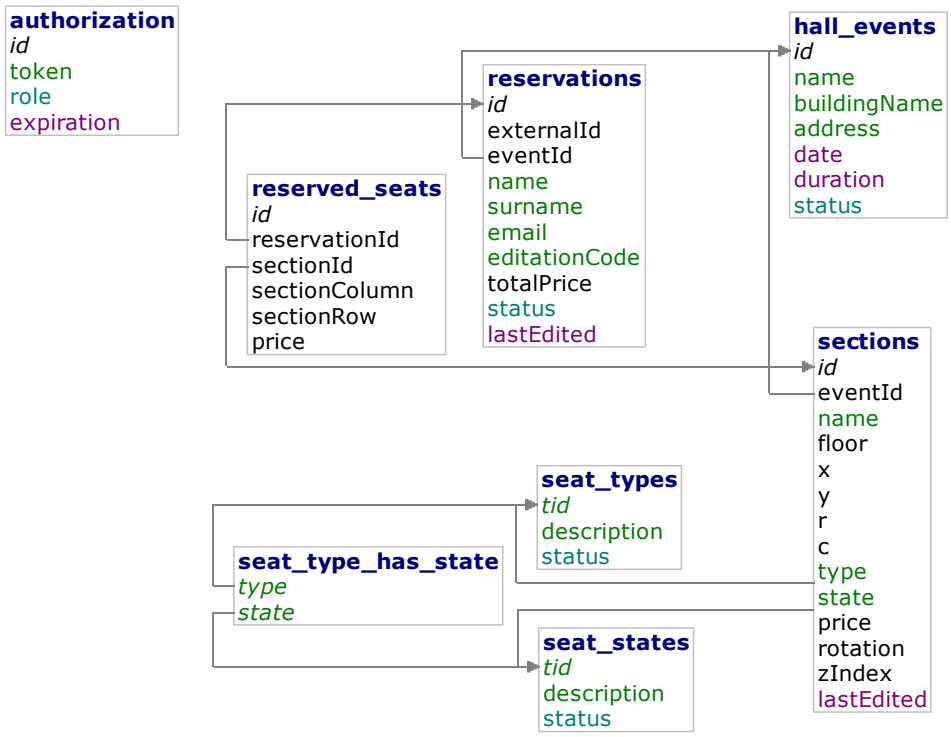
[24] nette.org/cs

MariaDB) pro její jednoduchost a snadnost propojení s PHP aplikací, ale také pro dlouholeté zkušenosti, které s ní mám.

Dále bylo nutné určit, jaký nástroj použít pro práci s databází přímo v PHP. Mám zkušenosti jak s vestavěnými balíčky (mysqli, PDO), tak i rozšířeními třetích stran (např. **dibi**[25]), ale žádný z nich mi z různých důvodů úplně nevyhovoval. Pro tento projekt jsem se rozhodl zamířit do nových vod, a to použitím **Nette Database Table**[26] [27] založenému na knihovně **NotORM**[28] od Jakuba Vrány[29]. Od stejného autora je také skvělá a rychlá aplikace **Adminer**[30], kterou jsem používal pro návrh databáze.

Návrh samotné databáze probíhal nejprve zhmotněním požadavků na uložení dat do ER diagramu (viz obrázek 2.2) a poté jeho převodem na databázové schéma (obrázek 5.17 níže). Toto odpovídá finálnímu návrhu realizace databáze.

V tomto diagramu je u sloupců tabulek použito několik stylů textu: zelený text značí datový typ řetězec, fialový datum, tyrkysový je výčtový typ a černou jsou přirozená čísla. Primární klíče jsou psané kurzívou, šedé šipky jsou zobrazením cizích klíčů a vedou vždy od cizího klíče k primárnímu.



Obrázek 5.17: Schéma databáze.

Lze si povšimnout několika drobných rozdílů oproti zmiňovanému ER diagramu. Tyto jsou většinou příčinou dodatečných úprav specifikace zákazníka. Příkladem je tabulka **seat_type_**

[25] dibiphp.com
[26] doc.nette.org/cs/2.4/database-table
[27] phpfashion.com/dibi-vs-nette-database-story
[28] www.notorm.com
[29] www.vrana.cz
[30] www.adminer.org/cs

has_state, jejímž úkolem je definovat, které stavy míst mohou být použity v kombinaci s typy míst (například pódium je vždy neprodejné, stejně tak třeba sloup apod.). Kvůli tomu, že tyto typy i stavy definuje zákazník, je nutné ponechat ke snadné editaci i vzájemné vztahy těchto entit.

Dalším viditelným rozdílem jsou atributy sekce, kde se navíc objevuje **rotace** a **zIndex**. Tyto atributy se zde nacházející díky jednomu z návrhů, že by se sekcemi dalo otáčet a mohly by se i překrývat. Nicméně realizace tohoto požadavku byla vyhodnocena jako náročná a zákazník zatím není rozhodnutý, zda si tyto úpravy přeje. Tyto atributy jsou zde nyní bezvýznamné a až dojde k finálnímu rozhodnutí, budou buď využity, nebo odstraněny.

Rovněž se objevily nové sloupce u tabulky rezervací, kde se nově objevuje i editační kód a celková cena rezervace. Editační kód je generovaný unikátní kontrolní součet, který má v budoucnu sloužit pro možnost operace s rezervací přímo z odkazu v těle e-mailu, který tento kód ponese, prozatím však není využíván. Celková cena je potom údaj sloužící především pro kontrolu a pohodlnost, jelikož lze cena rezervace vypočítat i součtem cen všech zarezervovaných sedadel.

Vztahy kompozice z ER diagramu jsou v databázi realizovány cizím klíčem s vlastností kaskádování změn. Smaže-li se celek, tedy rezervace nebo událost v síni, jsou smazány rovněž i všechny jejich části, tedy rezervovaná sedadla, resp. sekce.

Často se objevující atribut **status** je výčtový typ s možnými hodnotami **A** (active) nebo **D** (deleted), případně u rezervací ještě (prozatím nevyužívaný) **S** (suspended) a jehož smyslem je mít možnost položky některé entit nemazat permanentně, ale pouze nastavovat příznak smazání. Tento přístup zvýší bezpečnost při nechtěných výmazech dat (lze je snadno vrátit zpět a nejsou přerušeny vazby dat), přičemž není nutné hledat zálohu a mít riziko ztráty části dat.

5.3.5 Realizace aplikační logiky

Jak již bylo zmíněno v sekci o architektuře (5.3.2), aplikace sestává především z presenterů, což jsou třídy koncových bodů API a modelů, což jsou třídy konkrétních entit.

Všechny presentery dědí od základní třídy **BasePresenter**, která pro všechny existující operace vrací výjimku s kódem **501 Not Implemented**. Tím pádem všechny v presenterech neobsažené operace se automaticky stávají neimplementovanými. Všechny výjimky jsou typu **ApiException**, což je vlastní výjimka, která umožňuje specifikovat HTTP kód, který se má vrátit.

Standardně presentery pouze kontrolují a ověřují vstup — výkonnou logiku a databázové transakce ponechávají na modelech.

Všechny modely dědí od základní třídy **BaseModel**, která definuje základní operace nad databází (jako např. metody **getById**, **getAll**, atp.). Každý model má dále atribut **$table**, pomocí něhož určuje název tabulky, jejíž entitu modeluje. Základní databázové operace pracují s hodnotou tohoto atributu.

Všechny třídy využívají vkládání závislostí, a proto všechny musí jako parametr v konstruktoru uvádět ostatní třídy, na kterých závisí (viz sekce 5.3.3).

5.3.6 Perzistence a integrita dat

Bylo nutné zajistit, aby v případě paralelní práce s API (více požadavků od více klientů naráz) nenastal problém při ukládání dat, nebo nebyla operace ukládání přerušena v půli (např. je vytvořen záznam o rezervaci, ale nastala výjimka při pokusu o vytvoření záznamů o rezervovaných místech) a databáze tak zůstala v nekonzistentním stavu.

Tento problém dokáže zcela vyřešit využití databázových transakcí. Například provádění rezervací je řešeno jako jediná transakce a v případě problému se provede operace **rollback**,

tedy navrácení databáze do stavu, v jakém byla před zahájením celé transakce. Uživateli je poté zobrazena hláška, že rezervace selhala a ať to zkusí znovu, často i doplněná odůvodněním. Například, že si někdo jiný stihl ve stejný okamžik zarezervovat stejná místa a ať si vybere jiná.

Tento přístup zajistí konzistenci databáze v uvedených případech, ale navíc bylo nutné přidat i další integritní omezení. Ukázkovým případem je omezení u vazby tabulky rezervovaných míst na tabulku sekcí (cizí klíč), kde je zamezeno smazat sekci, do které již existuje nějaká rezervace.

Toto omezení se přidává k definici cizího klíče a má syntaxi **ON DELETE RESTRICT** a zamezí odstranění položky, na kterou v databázi existuje alespoň jedna vazba. Opačným přístupem je použití **ON DELETE CASCADE**, které v případě smazání položky naopak odstraní i všechny ty, které se k ní vážou. Tento postup je použitý například u vazby sekcí na událost v síni. Při pokusu o smazání celé události v síni tak dojde i k pokusu o smazání všech sekcí a pokud do některé ze sekcí existuje rezervace, je celá operace zrušena.

5.3.7 Rezervace

Rezervace lze v aplikaci KORES pouze vytvářet, nikoliv spravovat. O jejím vytvoření je e-mailovou notifikací zpraven jak zákazník, tak administrátor, nicméně toto je jediný případ, kdy aplikace informuje správce o detailech rezervace. Předpokládá se nasazení aplikace uvnitř jiného systému, který bude s vytvořenými rezervacemi pracovat. K tomuto účelu bude externí systém číst tabulku rezervací, případně jim nastavovat příznak smazání.

Tato část aplikace však může být v budoucnu dopracována (viz kapitola 7).

5.4 Zabezpečení

5.4.1 Zabezpečení komunikace a autorizace přístupů

Rozdělení aplikace do dvou celků má nad řadu výhod i své problémy a tím nejdůležitějším je nutnost komunikaci zabezpečit. Bez příslušného zabezpečení by mohl kdokoliv „uhodnout“ URI identifikátor některého zdroje a zkoušet posílat požadavky, dokud by neuspěl a nenapáchal v systému škody. Právě díky tomu, že se jedná o REST API, které má standardní architekturu, je uhodnutí příslušné operace až triviální (např. poslat HTTP požadavky DELETE na /section/{id} a zkoušet jen různá ID).

Naštěstí má tento problém i jednoduché řešení — všechny přístupy lze kontrolovat na přítomnost autorizačního tokenu, vygenerovaného při autentizaci. Autorizační token je dlouhý řetězec většinou složený z hexadecimálních číslic a je produktem silné hashovací funkce, např. SHA-256. Vygeneruje se při autentizaci do systému a má definovanou platnost (někdy i automatické prodlužování).

Veřejná část aplikace, tedy rezervační systém, ale žádnou autentizaci nemá. V tomto případě je nutné použít slabší autorizační token, který nemá takové pravomoci, jelikož jej případný útočník může získat inspekcí zdrojových kódů nebo síťové komunikace. Při použití tohoto tokenu musí API rozhraní dovolit operace jako je získání aktuálně zarezervovaných míst a vytvoření nové rezervace. Ostatní operace musí zamítnout jako neautorizované a to odpovědí **401 Unauthorized**.

Aplikace v konfiguračním režimu pak předpokládá nasazení v nějakém administračním systému, který autentizaci má, a v takovém případě je vygenerování tokenu po úspěšném přihlášení úlohou právě tohoto cizího systému. Ten poté tuto vygenerovanou hodnotu vloží do autorizační tabulky KORESu (**authorization**) a může si zvolit i délku platnosti a úroveň oprávnění (v tomto případě **admin**). Stejnou hodnotu pak tento systém předá i vložené front-end aplikaci, která bude moci fungovat v plném rozsahu.

Výhodou dříve zmíněného mikroframeworku Restler (viz 5.3.2 Architektura REST API) je vbudovaná podpora pro různé úrovně autorizace. Lze definovat autorizační třídu, jejíž metody se vykonávají při každém požadavku, a nemůže se tak stát, že se třeba u jednoho ze zdrojů na autorizaci zapomene. Framework dále rozpoznává uživatelské role a dovoluje je přidělovat jednotlivým CRUD operacím pomocí anotací (PHP komentářů), což velmi zjednodušuje a zpřehledňuje celé nastavení práv. Role se využívají pouze dvě, a to **user** a **admin**, pro veřejnou, resp. administrační část.

5.4.2 Ochrana proti Cross-site scripting

XSS neboli Cross-site scripting[31] je způsob narušení webové stránky vložením nebezpečného kódu, který se do stránky dostane neošetřeným vstupem. Příkladem může být, že teoretický záškodník provede rezervaci míst, ale místo svého jména napíše Javascriptový kód, který přesměrovává stránku na nebezpečný web. Ve chvíli, kdy si správce aplikace otevře seznam rezervací, ve kterém se nachází i jména, je okamžitě vykonán záškodníkův kód s správce je přesměrován na jeho útočný web.

Předcházení těmto útokům je naštěstí poměrně snadné, stačí ošetřovat všechny vstupy, nebo ještě lépe, všechny výstupy dat. V případě ošetřování pouze vstupů existuje šance, že se nějaký zapomene, kdežto ošetřování výstupů se dá nastavit automaticky. Pro framework React je ošetření všech výstupů výchozí a automatickou záležitostí.

[31] cs.wikipedia.org/wiki/Cross-site_scripting

Kapitola 6

Testování

Tato kapitola popisuje, jak byla aplikace během vývoje a po svém dokončení testována, aby byla zaručena správná funkčnost všech klíčových vlastností a intuitivnost prostředí. Nejprve je prezentována metodika a výsledky testování funkčnosti a následuje testování uživatelského rozhraní.

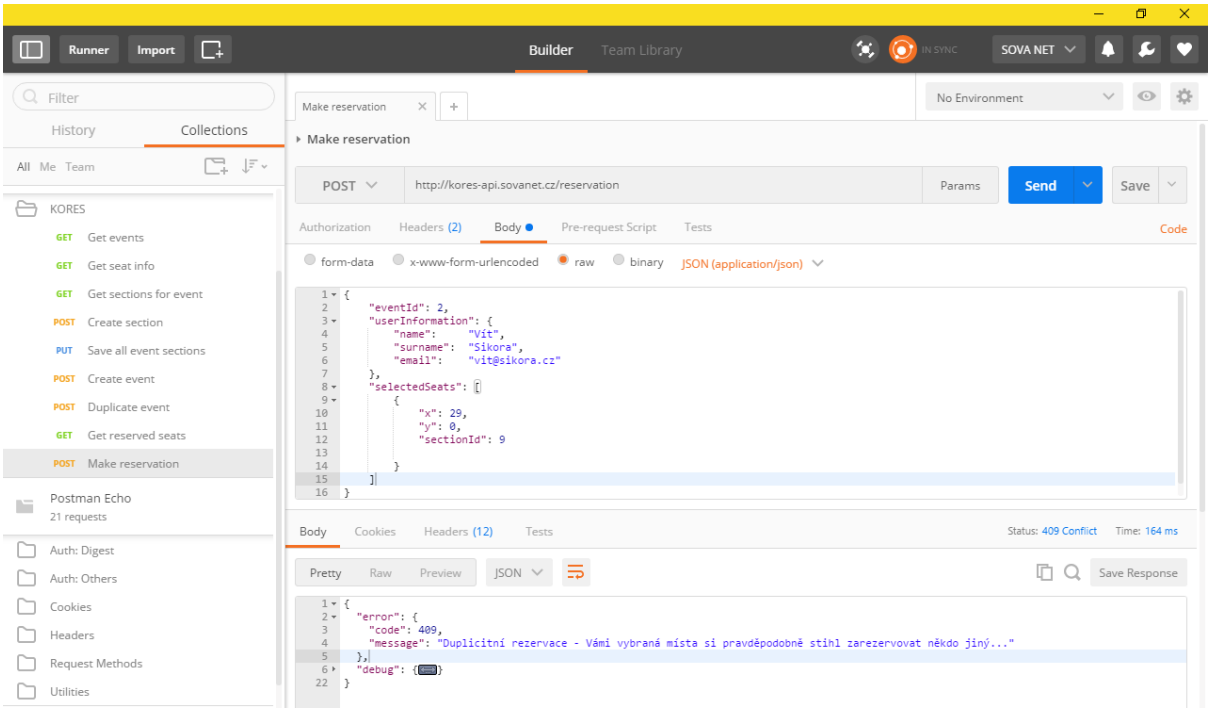
6.1 Testování funkčnosti

6.1.1 Testování koncových bodů API

Díky rozdělení aplikace na dvě samostatné části je také mnohem snazší testovat funkcionalitu back-endu, který je izolovaný a snadno dostupný přes koncové body zdrojů, popsaných v sekci 5.3.2 Architektura REST API.

Pro testování těchto bodů jsem používal nástroj Postman[1], což je aplikace přímo určená pro vývoj a testování API. Umožňuje ukládat si definice koncových bodů a požadavků, které mají být na API odeslány a zobrazuje jak veškeré HTTP hlavičky, tak i těla požadavků, kde lze přepínat podoba mezi syrovou (plain text), přeformátovanou (barevná syntaxe a odsazení, vhodné právě pro JSON) a webovou (pro HTML kód). Ukázku testování touto aplikací je možné vidět na obrázku 6.1.

[1] www.getpostman.com



Obrázek 6.1: Testování REST API nástrojem Postman.

Touto technikou byla testována sada základních operací a bylo tak zajištěno, že API funguje správně. Pokud byl poté při vývoji front-endu objeven nějaký problém, mohlo být snadno zjištěno, zda se jedná o chybu API, což izolaci její příčiny značně urychlilo.

Základní, pravidelně testované operace, byly následující:

- Získání seznamu událostí v síních,
- vytvoření nové a úprava existující události,
- získání sekcí pro konkrétní síň,
- uložení stavu sekcí pro konkrétní síň,
- získání seznamu rezervovaných sedadel,
- provedení rezervace.

Operace byly testovány a případně ihned opraveny vždy, když došlo k významnější úpravě API, nebo pokud byla hledána příčina chyby objevené na front-endu.

6.1.2 Testování funkčnosti celku

Pro testování funkčnosti celé aplikace byla zvolena metodika testování na skutečných datech a reálných případech použití. Ve spolupráci s klientem byl vytvořen návrh několika modelových síní, které zachycují typické rysy síní skutečných, pomocí nichž byla aplikace testována. Byl vytvořen soupis operací, které by se nad nimi měly vykonat. Vykonávání probíhalo přímo přes front-end aplikace.

Seznam těchto operací je následovný:

- Vytvoření nové události v síni dle předem připravené modelové síně.

- Vytvoření další modelové síně naklonováním té první.
- Dodatečné úprava detailů obou událostí a několika jejich náhodně zvolených sekcí.
- Vytvoření nové testovací události a její následné odstranění.
- Provedení dvou paralelních rezervací na stejná místa druhé modelové síně — první rezervace musí projít a druhá musí skončit s chybou, ale po volbě jiných míst a opětovném odeslání musí také projít.
- Nahodilé testování operací v administraci při krátkodobém odpojení od internetu a následný opětovný pokus po jeho zapojení.

Testové operace byly prováděny v posledních verzích prohlížečů **Google Chrome** (ve kterém probíhal i vývoj), **Mozilla Firefox**, **Microsoft Edge** a **Microsoft Internet Explorer**.

Nalezené chyby v průběhu vývoje byly vždy opravovány. Pro příklad, první takovou chybou byla nemožnost pohybovat se po mřížce síně tažením pravého tlačítka myši v prohlížečích od společnosti Microsoft. Na vině byla odlišná implementace atributu MouseEvent.button[2] v těchto prohlížečích od implementace v Google Chrome, ve kterém byla aplikace vyvíjena. Obě implementace nicméně splňují specifikaci[3], ve které je mj. napsáno, že pro události typu mousemove není hodnota tohoto atributu garantovaná. Nepřímo tedy nese vinu na chybě i nedostatečná přísnost specifikace. Problém byl nakonec vyřešen použitím atributu MouseEvent.buttons[4] (pozor na „s“ na konci), který má sice slabší podporu, ale jeho implementace je povinná i pro událost typu mousemove.

Při poslední iteraci testů po dokončení vývoje aplikace nebyly zjištěny žádné chyby.

6.2 Uživatelské testování

Pro zajištění toho, že je uživatelské rozhraní skutečně intuitivní, je nutné provést skutečné testování na reálných uživatelích a nelze se spoléhat pouze na vlastní představy o tom, jaké jsou potřeby uživatele, nikdy se totiž do něj nelze plně vcítit [9].

Z tohoto důvodu jsem se rozhodl prověřit své uživatelské rozhraní testováním ve dvou fázích.

6.2.1 Fáze 1: Osobní sledování

První fáze uživatelského testování byla přímá a kontaktní — uživateli jsem zadal, co má udělat, přičemž jsem mu o aplikaci řekl jen nutné minimum informací, a bez dalších pokynů jsem sledoval, co bude dělat a zkoušet, co se mu podaří na první pokus a u čeho bude například úplně zmaten. K této formě testování jsem si vybíral uživatele co nejblíže cílové skupiny lidí (viz sekce 2.2), tedy starší a méně technicky zdatné. Nakonec jsem takto testoval na celkem dvou uživatelích a odnesl jsem si cenné poznatky, z nichž některé jsem ihned realizoval.

Kupříkladu bylo takto velmi rychle zjištěno, že v pohledu pro výběr míst rezervace (obrázek 5.12) je nutné mít tlačítko pro pokračování v pravém spodním rohu (v původní, testované implementaci zde nebylo a přechod na další krok byl možný pouze přes horní krokovou navigaci).

Dalším vzniknuvším poznatkem k této obrazovce byla absence legendy, ale tu jsem se přesto rozhodl neimplementovat — sice možná mírně zvýší rychlost zorientování se v pohledu, ale sedadla mají jen dva stavy (volné, rezervované) a navíc by byla realizace technicky náročnější (bylo by nutné vyřešit překryvy se sekcemi).

[2] developer.mozilla.org/en-US/docs/Web/API/MouseEvent/button
[3] www.w3.org/TR/DOM-Level-2-Events/events.html#Events-MouseEvent
[4] developer.mozilla.org/en-US/docs/Web/API/MouseEvent/buttons

6.2.2 Fáze 2: Vzdálené testování

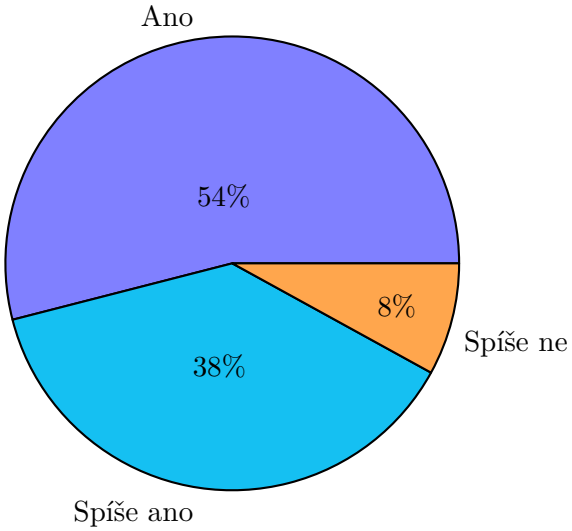
Ve druhé fázi, která probíhala společně s dolaďováním posledních detailů v aplikaci, jsem se rozhodl položit otázky týkající se rozhraní, na které bych chtěl znát odpověď. Otázky zněly:

- Je ovládání dostatečně intuitivní?
- Je prostředí uživatelsky přívětivé?
- Je uspořádání prvků logické a přehledné?

Testování jsem navrhl tak, že poskytnu vzdáleným uživatelům jednoduchý popis aplikace (veskrze pouze to, že se jedná o konfigurátor a rezervační systém pro koncertní síně) a zorientování se v rozhraní nechám plně na nich. Dále jsem doplnil dotazník s výše uvedenými otázkami a možnými odpovědmi v rozsahu: ano, spíše ano, spíše ne a ne. Připojil jsem i pole pro dodatečné poznámky a připomínky. Respondenty tohoto testování byli převážně studenti, a to díky jejich snadné dostupnosti a ochotě testovat ostatní aplikace.

Celkový počet testujících subjektů byl 13 a výsledky odpovědí na jednotlivé otázky byly následující:

Je ovládání dostatečně intuitivní?



Obrázek 6.2: Odpovědi na otázku „Je ovládání dostatečně intuitivní?“

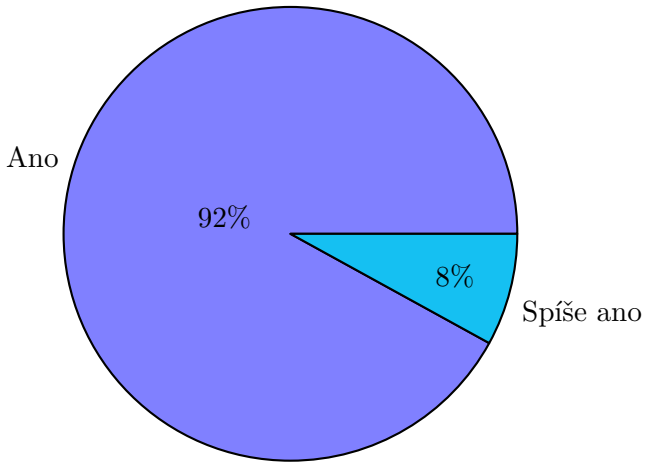
Tato otázka dopadla asi nejhůře ze všech, ale naštěstí se uživatelé často shodovali ve svých připomínkách. Nejčastějším poznatkem bylo, že u ikonových tlačítek chybí tooltip[5] a nebylo jasné, co tlačítko udělá — zejména u tlačítka pro vytvoření kopie události v síni. Tuto zpětnou vazbu jsem dostal nezávisle od pěti uživatelů. Rozhodl jsem se proto tuto funkcionalitu okamžitě doplnit.

Mezi další poznatky patřilo kupříkladu to, že u políčka pro výběr patra síně není zcela jasné, že je klikatelné nebo že chybí implementace klávesových zkratek.

Dále jsem dostával pozitivní zpětnou vazbu na to, že je uživatelské rozhraní systému jednoduché a neobsahuje zbytečné prvky, které by uživatel nevyužil a mátly jej.

[5] Popisek, který se zobrazí po najetí myši.

Je prostředí uživatelsky přívětivé?

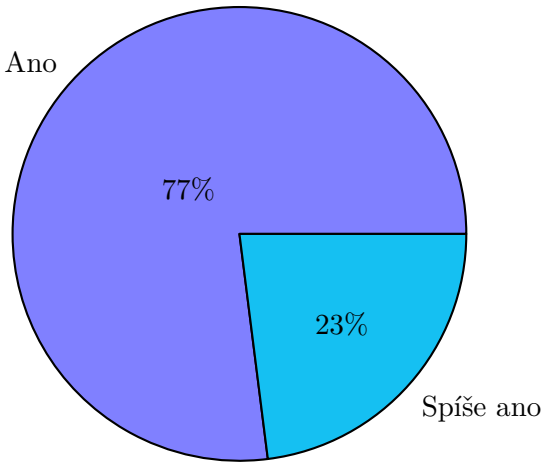


Obrázek 6.3: Odpovědi na otázku „Je prostředí uživatelsky přívětivé?“

Výsledky odpovědí na tuto otázku jsem byl mírně překvapen — nečekal jsem tak pozitivní reakci. Dle mého názoru je však za vysokou uživatelskou přívětivost velkým dílem zodpovědná právě použitá grafická knihovna Material UI, která je skutečně povedená.

Odpovědi jsou tedy především pozitivní zpětnou vazbou na volbu vhodné knihovny a na realizaci uživatelské interakce skrze její komponenty.

Je uspořádání prvků logické a přehledné?



Obrázek 6.4: Odpovědi na otázku „Je uspořádání prvků logické a přehledné?“

S výsledky této otázky jsem rovněž spokojen, ale k umístění prvků jsem jinou zpětnou vazbu nedostal, bez dalších testů tedy nevím, co by se v této situaci dalo ještě zlepšit.

Dále jsem dostal i poznatek týkající se funkční stránky aplikace: Jednomu respondentovi skončila e-mailová notifikace o rezervaci ve spamu. Aby se podobným případům zabránilo, je nutné na serveru nastavit podepisování zpráv DKIM[6], což už sice nestihnu vyřešit do odevzdání, ale jsem rád, že jsem uživatelským testováním tuto slabinu odhalil.

[6] www.hromadnaposta.cz/podepisovani-zprav-pomoci-dkim-signatury.html

Kapitola 7

Další vývoj

Protože vývoj projektu odevzdáním práce nekončí, rozhodl jsem se sepsat veškeré poznatky k vývoji, ať už se jedná o drobné chyby a nedodělky, nebo návrhy na možná rozšíření.

7.1 Nevyřešené problémy

Následuje seznam drobnějších problémů, které jsem nestihl zapracovat kvůli jejich příliš nízké prioritě, nebo příliš pozdnímu zjištění.

- Pokud při konfiguraci síně není zvolena žádná sekce a uživatel se pokusí o roztažení sekce, která se nachází u pravého okraje, je tato zvolena a překryta panelem detailu — uživatel tak nemůže změnu velikosti dokončit.
- V editoru není znázorněno, která konkrétní místa v sekci jsou zarezervována. Chce-li to zjistit, musí přejít do rezervačního systému.
- Validace e-mailové adresy ve formuláři rezervace je průběžná a informuje tak o špatném formátu adresy ještě dříve, než ji uživatel dopíše — toto lze řešit lépe, a to validováním až po skončení editace.

7.2 Návrhy na možná rozšíření

Následují seznamy všech mých nápadů na možná rozšíření, rozdělené dle režimu aplikace.

7.2.1 Editační režim

- Po smazání sekce by se mohla objevit dočasná spodní lišta (SnackBar v Material UI) s možností vrátit smazání.
- Optimalizace výkonu editační mřížky (nyní déle trvající načtení, kvůli vyššímu počtu prvků).
- „Nekonečná“ editační plocha (nyní omezena na 100×100 polí).
- Kontextové přibližování a oddalování plochy dle polohy kurzoru myši (nyní vždy směrem do/ze středu plochy).
- Realizace klávesových zkratek (např. Esc pro zrušení výběru, Delete pro smazání vybrané sekce atp.).

- Na editační ploše zobrazovat s nízkou průhledností i sekce v ostatních patrech (toto nejspíše ovládané přepínačem).
- Možnost exportu rezervací do Excelového souboru nebo CSV.
- Možnost správy rezervací.

7.2.2 Rezervační režim

- Po najetí myší na zarezervovatelné sedadlo by se namísto tooltipu mohlo zobrazit malé graficky zpracované modální okno s informacemi o sedadle a sekci.
- Možnost resetovat volbu sedadel.
- Volba pater pomocí tlačítek přímo na ploše, namísto SelectBoxu v liště.
- Možnost specifikovat jiné číslování sedadel, než je výchozí (obrácené pořadí řádků a/nebo sloupců, vynechávání uliček, vlastní číslování atp.).
- Vyhnout se konfliktům rezervací dočasnou blokací míst po dobu vyplňování rezervačních údajů.
- Po rezervaci míst vygenerovat lístky v PDF s čárovými kódy a poslat instrukce k platbě.

Kapitola 8

Závěr

Cílem celé této práce bylo vytvořit webovou aplikaci pro návrh koncertních síní a rezervační systém nad těmito síněmi. Aplikace měla být modulární, snadno upravitelná, uživatelsky přívětivá a vložitelná do externího systému.

Tento úkol jsem úspěšně splnil — aplikace má drobné nedostatky (seznam v sekci 7.1), ale požadavky na aplikaci byly splněny.

Funkčnost aplikace byla ověřena testováním nad zákazníkem zvolenými daty. Modularita a snadná upravitelnost byla dosažena pomocí několika technik, zejména použitím komponentního návrhu a vkládáním závislostí. Uživatelská přívětivost byla ověřena pomocí uživatelského testování a jednoduché vložitelnosti do externího systému se také podařilo dosáhnout.

Aplikace je aktuálně připravena na reálné produkční nasazení, nicméně o finálním zveřejnění rozhoduje zákazník a je na něm, zda nebude chtít nejprve doimplementovat některé z mnou nabízených rozšíření.

Při vývoji této aplikace jsem vycházel ze svých dosavadních zkušeností s vývojem Javascriptových i PHP aplikací (při volbě technologií jsem někdy zmiňoval, že mám s některými už zkušenosti), ale i tak jsem překonal mnoho dříve ani nevyzkoušených výzev. Tou největší byla asi delikátní práce myší s mřížkou konfigurátoru sekcí, u které jsem strávil desítky hodin optimalizací kódu, čtením specifikace W3C a testováním.

Literatura

[1] *Ambitious web applications built using Ember.js*. [Online; navštíveno 04.03.2017]. Dostupné z: <http://builtwithember.io/>

[2] *JavaSript framework resources*. [Online; navštíveno 10.05.2017]. Dostupné z: <https://github.com/vhf/free-programming-books/blob/master/javascript-frameworks-resources.md>

[3] *Návrhový antivzor*. [Online; navštíveno 06.03.2017]. Dostupné z: <https://cs.wikipedia.org/wiki/N%C3%A1vrhov%C3%BD_antivzor>

[4] *5 BEST JAVASCRIPT FRAMEWORKS IN 2017*. Leden 2017, [Online; navštíveno 01.03.2017]. Dostupné z: <https://da-14.com/blog/5-best-javascript-frameworks-2017>

[5] *Sites Using React*. Březen 2017, [Online; navštíveno 04.03.2017]. Dostupné z: <https://github.com/facebook/react/wiki/sites-using-react>

[6] ABRAMOV, D.: *Redux: Usage with React*. [Online; navštíveno 04.03.2017]. Dostupné z: <http://redux.js.org/docs/basics/UsageWithReact.html>

[7] ELLIOTT, E.: *Top JavaScript Frameworks & Topics to Learn in 2017*. Prosinec 2016, [Online; navštíveno 01.03.2017]. Dostupné z: <https://medium.com/javascript-scene/top-javascript-frameworks-topics-to-learn-in-2017-700a397b711>

[8] KOZLOV, S.: *Comparison of JS Frameworks Angular.js vs React.js vs Ember.js*. Listopad 2016, [Online; navštíveno 08.03.2017]. Dostupné z: <https://www.romexsoft.com/blog/js-frameworks-comparison/>

[9] KRUG, S.: *Nenuťte uživatele přemýšlet!: praktický průvodce testováním a opravou chyb použitelnost [sic] webu*. Computer Press, 2010, ISBN 978-80-251-2923-4.

[10] MALÝ, M.: *REST: architektura pro webové API*. Srpen 2009, [Online; navštíveno 25.02.2017]. Dostupné z: <https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>

[11] MAROHNIĆ, M.: *Semicolons in JavaScript are optional*. Květen 2010, [Online; navštíveno 11.04.2017]. Dostupné z: <https://mislav.net/2010/05/semicolons/>

[12] NEARY, A.: *Unidirectional Data Flow? Yes. Flux? I Am Not So Sure*. Březen 2015, [Online; navštíveno 07.03.2017].

Dostupné z: <https://medium.com/@AdamRNeary/unidirectional-data-flow-yes-flux-i-am-not-so-sure-b4acf988196c>

[13] SHAIKH, S.: *Top 15 websites and apps built with AngularJS*. Červen 2015, [Online; navštíveno 04.03.2017].
Dostupné z: <https://www.eduonix.com/blog/web-programming-tutorials/top-15-websites-and-apps-built-with-angularjs/>

[14] VLAD, V.; SVIATOSLAV, A.; FEDOR, L.: *React vs AngularJS - A Popular JavaScript Library and a Powerful JavaScript Framework*. [Online; navštíveno 07.03.2017].
Dostupné z: <https://rubygarage.org/blog/react-vs-angularjs>

[15] WILSON, M.: *Google's New, Improved Android Will Deliver A Unified Design Language*. Červen 2014, [Online; navštíveno 12.04.2017].
Dostupné z: <https://www.fastcodesign.com/3032378/googles-new-improved-android-will-deliver-a-unified-design-language>

Přílohy

Seznam příloh

Příloha A

Obsah souboru package.json

```
{
  "name": "kores",
  "version": "1.0.0",
  "description": "Configuration and Reservation System for Concert Halls",
  "scripts": {
    "start": "node server.js",
    "build": "build.cmd"
  },
  "license": "UNLICENSED",
  "keywords": [
    "react",
    "reactjs"
  ],
  "author": "Vit Sikora",
  "bugs": {},
  "homepage": "",
  "devDependencies": {
    "autoprefixer": "^6.7.5",
    "babel-cli": "^6.23.0",
    "babel-core": "^6.23.1",
    "babel-eslint": "^7.0.0",
    "babel-loader": "^6.4.1",
    "babel-plugin-syntax-decorators": "^6.13.0",
    "babel-plugin-transform-class-properties": "^6.23.0",
    "babel-plugin-transform-decorators-legacy": "^1.3.4",
    "babel-plugin-transform-es2015-classes": "^6.23.0",
    "babel-plugin-transform-es2015-modules-commonjs": "^6.23.0",
    "babel-plugin-transform-proto-to-assign": "^6.23.0",
    "babel-plugin-transform-runtime": "^6.23.0",
```

Výpis A.1: Obsah souboru package.json — část 1. / 3

```
    "babel-polyfill": "^6.23.0",
    "babel-preset-es2015": "^6.16.0",
    "babel-preset-es2015-ie": "^6.6.2",
    "babel-preset-react": "^6.23.0",
    "babel-preset-stage-0": "^6.16.0",
    "babel-runtime": "^6.23.0",
    "classes": "^0.3.0",
    "css-loader": "^0.25.0",
    "cssnext-loader": "^1.0.1",
    "dedupe": "^2.0.3",
    "ejs-loader": "^0.3.0",
    "eslint": "^3.16.1",
    "eslint-plugin-react": "^6.10.0",
    "eslint-plugin-standard": "^2.0.1",
    "extract-text-webpack-plugin": "^1.0.1",
    "file-loader": "^0.9.0",
    "html-loader": "^0.4.4",
    "html-webpack-plugin": "^2.28.0",
    "js-stylesheet": "0.0.1",
    "less": "^2.7.1",
    "less-loader": "^2.2.3",
    "node-libs-browser": "^1.0.0",
    "postcss-import": "^8.1.2",
    "postcss-loader": "^0.13.0",
    "precss": "^1.4.0",
    "react-hot-loader": "*",
    "style-loader": "^0.13.1",
    "url-loader": "^0.5.7",
    "webpack": "^1.13.2",
    "webpack-dev-server": "^1.16.1",
    "webpack-livereload-plugin": "^v0.8.2",
    "webpack-node-externals": "^1.5.4",
    "webpack-strip-block": "^0.1.1"
},
```

Výpis A.2: Obsah souboru package.json — část 2. / 3

```
  "dependencies": {
    "isomorphic-fetch": "^2.2.1",
    "lodash": "^4.16.2",
    "lokijs": "^1.4.3",
    "material-ui": "^0.17.4",
    "moment": "^2.18.1",
    "react": "^15.5.4",
    "react-custom-scrollbars": "^4.1.1",
    "react-dimensions": "^1.3.0",
    "react-dom": "^15.5.4",
    "react-redux": "^5.0.4",
    "react-s-alert": "^1.3.0",
    "react-tap-event-plugin": "^2.0.1",
    "redux": "^3.6.0",
    "redux-api": "^0.9.19",
    "redux-immutable": "^3.1.0",
    "redux-thunk": "^2.1.0",
    "screenfull": "https://registry.npmjs.org/screenfull/-/screenfull-3.2.0.tgz"
  }
}
```

Výpis A.3: Obsah souboru package.json — část 3. / 3

Příloha B

Obsah přiloženého DVD

Na DVD se v kořenovém adresáři nacházejí tyto soubory a adresáře:

kores/ — zdrojové kódy front-endové části aplikace.

kores-api/ — zdrojové kódy back-endové části aplikace.

db.sql — soubor pro vytvoření databáze.

readme.txt — návod na instalaci aplikace.

manual.txt — jednoduchá uživatelská příručka k ovládání aplikace.

technicka-zprava.pdf — textová část práce (tento soubor).