FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Automatizace procesů objednávek
Student:	Leoš Tobolka
Vedoucí:	Ing. Josef Kejzlar
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je návrh a implementace automatizace procesů objednávek pro e-shop a jejich uživatelská konfigurace.

Automatizací procesů se rozumí, že systém bude schopen na základě stavů a dat objednávek automaticky přepínat jejich stavy a reagovat (odesílání e-mailů zákazníkům, upozornění administrátorů apod.).
Uživatelé e-shopu budou moci chování automatizace konfigurovat přes grafické rozhraní v administraci systému.

Požadavky na technologické prvky:
- backend bude implementován v PHP frameworku Symfony
- pro frontend proveďte analýzu technologií a vyberte vhodného kandidáta
- implementované rozšíření bude obsahovat automatizované testy

---

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 29. listopadu 2021 v Praze.

Bakalářská práce

AUTOMATIZACE PROCESŮ OBJEDNÁVEK

Leoš Tobolka

Fakulta informačních technologií
Katedra teoretické informatiky
Vedoucí: Ing. Josef kejzlar
12. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Chtěl bych poděkovat především svému vedoucímu mé bakalářské práce Ing. Josefu Kejzlarovi za jeho rady a ochotu v průběhu psaní této práce. Obdivuji jeho zkušenosti, které denně s ochotou předává mnoha svým kolegům. Také chci poděkovat všem, kteří se podíleli na testování aplikace a poskytli mi tím zpětnou vazbu. Nakonec chci poděkovat své rodině, která mi poskytuje podporu v mém studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 12. května 2022 .....

Abstrakt

Tato bakalářská práce si klade za cíl vytvořit pro internetový obchod rozšíření na automatizaci procesů objednávek, které administrátorům obchodu umožní prostřednictvím grafického rozhraní automatizaci objednávek konfigurovat. Práce obsahuje proces tvorby celého rozšíření. Jedná se o analýzu, návrh a implementaci. Výsledkem je uživatelsky přívětivá aplikace implementovaná ve frameworku Symfony a JavaScript knihovně React.

Klíčová slova Symfony framework, automatizace objednávek, React, návrh a implementace webové aplikace, správa objednávek, internetový obchod

Abstract

This bachelor thesis aims to create an extension for an online store to automate the processes of orders, allowing store administrators to configure the automation through a graphical interface. The thesis includes the process of creating the whole extension. It involves analysis, design and implementation. The result is a user-friendly application implemented in Symfony framework and React JavaScript library.

Keywords Symfony framework, orders automation, React, design and implementation of web application, orders management, e-shop

Seznam zkratek

JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
PHP	PHP Hypertext Preprocessor
JS	JavaScript
UUID	Universally unique identifier
GUI	Grafické rozhraní
UML	Unified Modeling Language
SaaS	Software jako služba
HTTP	Hypertext Transfer Protocol
MVC	Model-view-controller
DOM	Document Object Model

Úvod

Jednou ze základních funkcionalit internetových obchodů je přijímat objednávky od zákazníků. Přijaté objednávky jsou uloženy a poté interně zpracovávány zaměstnanci daných obchodů. Každá objednávka musí projít svým životním cyklem, než se dostane k zákazníkovi, proto je v praxi zvykem objednávky pečlivě organizovat, aby při zpracování nedošlo k chybě a každý životní cyklus zpracovávané objednávky byl správný. Organizace a zpracování vyžadují velké množství času, obzvláště při velkém množství objednávek. Proto bývá snahou tyto dvě věci usnadnit automatizací, aby se minimalizoval lidský faktor a předešlo se chybám. Kvůli různorodým požadavkům od provozovatelů obchodů na automatizaci nelze jednoduše vytvořit jednu společnou automatizaci, která by pro všechny obchody fungovala dle potřeb. Automatizace vyžaduje být zavedena individuálně pro jednotlivé obchody dle předem domluvených požadavků.

Nepříjemnost může být, pokud internetový obchod často přidává nové funkce. Takové okolnosti mohou zapříčinit, že budou chtít změnit způsob automatizace svých objednávek. Pokud ale na úpravu automatizace nemají grafický nástroj, neumí programovat nebo nemají přístup ke zdrojovým kódům svého internetového obchodu, tak automatizaci jednoduše upravit nemohou.

V této práci se zaměřím na vytvoření automatizačního nástroje, ve kterém administrátoři internetových obchodů mohou vlastnoručně v grafickém rozhraní nakonfigurovat automatizaci procesů objednávek dle svých potřeb a nebudou muset zatěžovat programátory, aby to udělali za ně.

Rešeršní část je věnována popisu existujících řešení a použitých technologií. Následuje návrh aplikace, kde jsou uvedeny požadavky, případy užití, diagram komponent a je vybráno grafické rozhraní. Kapitola implementace popisuje způsob realizace zadání. Ukázka aplikace je v kapitole Testování a ukázky.

Cíle práce

Hlavním cílem této práce je vytvořit nástroj, který bude moct být integrován do existujícího systému, ve kterém poté bude možné pro administrátory internetového obchodu nakonfigurovat vlastní automatizaci objednávek. V rešeršní části mají být vidět existující řešení, která se na trhu vyskytují. Další z cílů je podle požadavků na aplikaci navrhnout přívětivé grafické rozhraní, které budou moci používat i uživatelé neznalí programování. Dále, podle návrhu implementovat grafické rozhraní a implementovat serverovou část, která řeší celkovou logiku vyhodnocování uživatelských konfigurací automatizace. Serverová část je implementována ve frameworku Symfony jako balíček, který lze integrovat do jiných existujících Symfony projektů. Administrátoři výsledného systému mohou vytvářet a spravovat jednotlivé konfigurace automatizace. Kód vytvořeného programu zároveň obsahuje rozhraní pro možnost přidání dalších stavebních prvků konfigurace.

Kapitola 1
Rešeršní část

V této kapitole je popis existujících řešení, diskuze jejich výhod, nevýhod a důvody proč jsou tyto řešení nevyhovující pro dané požadavky. Dále je tu popis použitých technologií, doplněný důvodem jejich použití.

1.1 Existující řešení

Nejvýraznější řešení, které se mi podařilo najít při prohledávání této problematiky.

1.1.1 Brani

Existující řešení, se kterým se s největší pravděpodobností setkáte při průzkumu této problematiky je řešení od firmy Brani.cz.

Zmiňovaná firma se zaměřuje na automatizace internetových obchodů a jeden z jejich produktů je právě doplněk na automatizaci objednávek.

Doplněk umožňuje automaticky, na základě vámi nastavených podmínek, přepínat jednotlivé stavy u zadaných objednávek. Umožní vám třídit objednávky podle zadaných kritérií – například na základě platby, dopravy, proplacení objednávky, kódu a názvu produktu či dostupnosti na skladě. [1]

Výhoda této služby je provozování automatizace na serverech Brani.cz. Nemusí se řešit výpadek serverů, jejich údržba a podobně. [2] Stačí pouze napojit vaši službu na jejich a poté nakonfigurovat automatizaci v jejich rozhraní.

Hlavní nevýhodou je cena za služby. Měsíční provoz většiny nástrojů se počítá podle využití (počtu zpracovaných objednávek, počtu produktů a podobně) [3]. Pro velké obchody, které přijmou denně stovky objednávek se cena služby pohybuje v tisících korun.

Vzhledem k tomu že se procesy spojené s automatizací provádí na jejich serverech, může být složité si některé procesy přizpůsobit na míru.

1.1.2 Shopify

Shopify je platforma pro elektronické obchodování, která pomáhá podnikům provozovat internetový obchod. Funguje na modelu Software jako služba (SaaS) prostřednictvím měsíčního předplatného. Jedná se o cloudovou službu, takže není potřeba se starat o údržbu software či webové servery. [4]

Tato služba poskytuje rozšíření pro automatizaci. Stejně jako předchozí řešení, umí automatizovat objednávky na základě různých jejich vlastností.

Grafické rozhraní konfigurátoru se skládá z bloků dělící se na spouštěče, podmínky a akce. Tyto bloky lze mezi sebou spojovat a tím tvořit požadovanou logiku automatizace. [5]

Pro využití této služby je potřeba měsíčně platit za pronajmutí jejich služby. Vzhledem k tomu že se jedná o jeden z největších řešení na světovém trhu, cena je vysoká, jedná se o cenu přibližně 2 000 dolarů měsíčně [6].

1.2 Použité technologie

Technologie které jsem se v této práci rozhodl využít. Jejich stručný popis a odůvodnění výběru.

1.2.1 PHP

PHP (rekurzivní zkratka PHP: Hypertext Preprocessor, česky „PHP: Hypertextový preprocesor“, původně Personal Home Page) je skriptovací programovací jazyk. Je určený především pro programování dynamických internetových stránek a webových aplikací například ve formátu HTML, XHTML či WML. [7]

Při použití PHP pro dynamické stránky jsou skripty prováděny na straně serveru – k uživateli je přenášen až výsledek jejich činnosti. Interpret PHP skriptu je možné volat pomocí příkazového řádku, dotazovacích metod HTTP nebo pomocí webových služeb. Syntaxe jazyka je inspirována

několika programovacími jazyky (Perl, C, Pascal a Java). Jazyk PHP je nezávislý na platformě, rozdíly v různých operačních systémech se omezují na několik systémově závislých funkcí a skripty lze většinou mezi operačními systémy přenášet bez jakýchkoli úprav. [7]

Tento programovací jazyk je zvolen z důvodu, že výsledné řešení bude integrováno do již existující aplikace, napsané právě v PHP.

1.2.2 Symfony framework

Symfony je PHP framework jenž staví na principech MVC architektury. Jeho cílem je urychlit tvorbu a údržbu webových aplikací a nahradit opakující se úlohy kódování.

Symfony lze použít pro vytváření robustních aplikací v podnikovém prostředí, protože pomáhá vývojářům testovat, ladit a dokumentovat projekty a poskytuje jim plnou kontrolu nad konfigurací a přizpůsobením – od adresářové struktury až po cizí knihovny.

Symfony je jeden z nejvíce světově rozšířených PHP frameworků. [8][9]

Symfony framework jsem zvolil opět z důvodu technologických požadavků. Výsledné řešení práce bude integrováno jako Symfony balíček do již existujícího systému. Symfony balíček představuje strukturu zdrojových souborů, které dohromady implementují nějakou funkcionalitu.

1.2.3 Javascript

JavaScript je interpretovaný programovací jazyk, který umožňuje oživit elektronické dokumenty v jazyce (X)HTML o interaktivní prvky. Jedná se o dynamicky typovaný jazyk s prvky funkcionálního a objektově orientovaného programování, který původně vznikl jako jednoduchá alternativa k webovým appletům v jazyce Java. JavaScript se Javě podobá syntaxí, názvy vestavěných funkcí a datových typů, ale koncepčně se jedná o zcela odlišné jazyky. (…)

K elektronickému dokumentu lze z JavaScriptu přistupovat skrz rozhraní Document Object Model (DOM). DOM reprezentuje dokument jako strom, ve kterém je možné se pohybovat, přistupovat k jeho uzlům a reagovat na události, jako je kliknutí myši nebo stisk klávesy. [10]

JavaScript je nejpoužívanější programovací jazyk na světě, jen stěží si lze představit moderní webovou aplikaci která by ho na straně klienta nepoužívala, jeho využití je téměř povinnost.

1.2.4 TypeScript

TypeScript je ve vztahu s jazykem JavaScript. TypeScript nabízí všechny funkce JavaScriptu a k nim další vrstvu: typový systém.

JavaScript například poskytuje jazykové primitivy jako řetězec a číslo, ale nekontroluje, zda jste je důsledně přiřadili. TypeScript to dělá. [11]

Samotný kód psaný v TypeScriptu se kompiluje do JavaScriptu. Jelikož je TypeScript nadstavbou nad JavaScriptem, je každý JavaScriptový kód automaticky validním TypeScript kódem. [12]

TypeScript jsem se rozhodl použít hlavně z důvodu zvýšení přehlednosti kódu. Je vysoká pravděpodobnost že aplikace bude v budoucnu rozšířena o více funkcionalit, proto udržitelnost a přehlednost je jednou z hlavních priorit.

1.2.5 React

React je knihovna, která pomáhá vývojářům vytvářet uživatelská rozhraní jako strom malých částí nazývaných komponenty. Komponenta je směs HTML a JavaScriptu, která zachycuje veškerou logiku potřebnou k zobrazení malé části většího uživatelského rozhraní. Každou z těchto komponent lze sestavit do postupně složitějších částí aplikace. [13]

Kapitola 2

Návrh aplikace

V této kapitole jsou definovány funkční a nefunkční požadavky pro požadovanou aplikaci. Návrhy a výběr konceptu grafického rozhraní, základní případy užití, diagram komponent a návrh finální podoby grafického rozhraní.

2.1 Požadavky

Primární účel aplikace je umožnění vytváření automatizace objednávek. Aplikace má přívětivé GUI a je dostatečně srozumitelná, aby v ní mohl pracovat i uživatel bez znalostí programovaní.

Příklady které musí aplikace umožňovat:

- Nakonfigurovat odeslání e-mailů zákazníkovi na základě dat z objednávky
- Nakonfigurovat změnu příznaků objednávky na základě dat z objednávky (stav zaplacení, zařazení do příslušného stavu atp.)
- Zobrazit záznam provedených akcí (pro zpětné dohledání chování automatizace)
- Upozornění administrátora – aplikace umožňuje upozornit administrátory e-shopu (např. dlouho nevyřízená objednávka, chybné údaje v objednávce atp.)
- Testovací režim (pro otestování konfigurace automatizace bez změny dat)

2.1.1 Funkční požadavky

F1 – Podmínky a akce

V aplikaci lze nakonfigurovat podmínky které objednávka musí splňovat a akce které se při splnění podmínek provedou. Různé typy podmínek a akcí mohou mít své detailnější nastavení, které uživatel může nastavit dle svého uvážení.

F2 – Spouštěče

Existují různé spouštěče a možnost zvolit si jejich typ. Příklady spouštěčů jsou: Periodické spouštění, spuštění při nějaké události týkající se objednávky (vytvoření objednávky, změna stavu) atp. Spouštěče určují na jakých objednávkách se budou podmínky a akce aplikovat.

F3 – Testovací režim

Testovací režim je vlastnost, která umožňuje uživatelům otestovat vlastní konfiguraci bez vlivu na produkční data.

F4 – Záznam aktivit

Možnost zaznamenávat aktivity aplikace, které dopomáhají uživatelům zpětně dohledat akce provedené na objednávkách.

F5 – Seznam všech konfigurací

Možnost vypsat seznam všech vytvořených konfigurací automatizace.

F6 – Název a popis konfigurace

Každá konfigurace automatizace má svůj vlastní název a popis, který si uživatel může zvolit. Slouží k lepší orientaci mezi konfiguracemi.

F7 – Možnost vypnutí konfigurace

Konfigurace má možnost být vypnuta. V tomto stavu je neaktivní a neprovádí se její akce.

2.1.2 Nefunkční požadavky

NF1 – Funkčnost na velkém počtu objednávek

Aplikace funguje i pro velký počet objednávek. Uživatel má možnost jím vytvořenou konfiguraci otestovat, zda se aplikuje na správné objednávky a provádí správné akce.

NF2 – Implementace vlastních podmínek a akcí

Programátor má možnost jednoduše implementovat vlastní podmínky a akce.

NF3 – Přívětivé grafické rozhraní

GUI je pochopitelné i pro běžné uživatele. Nevyžaduje znalost programování.

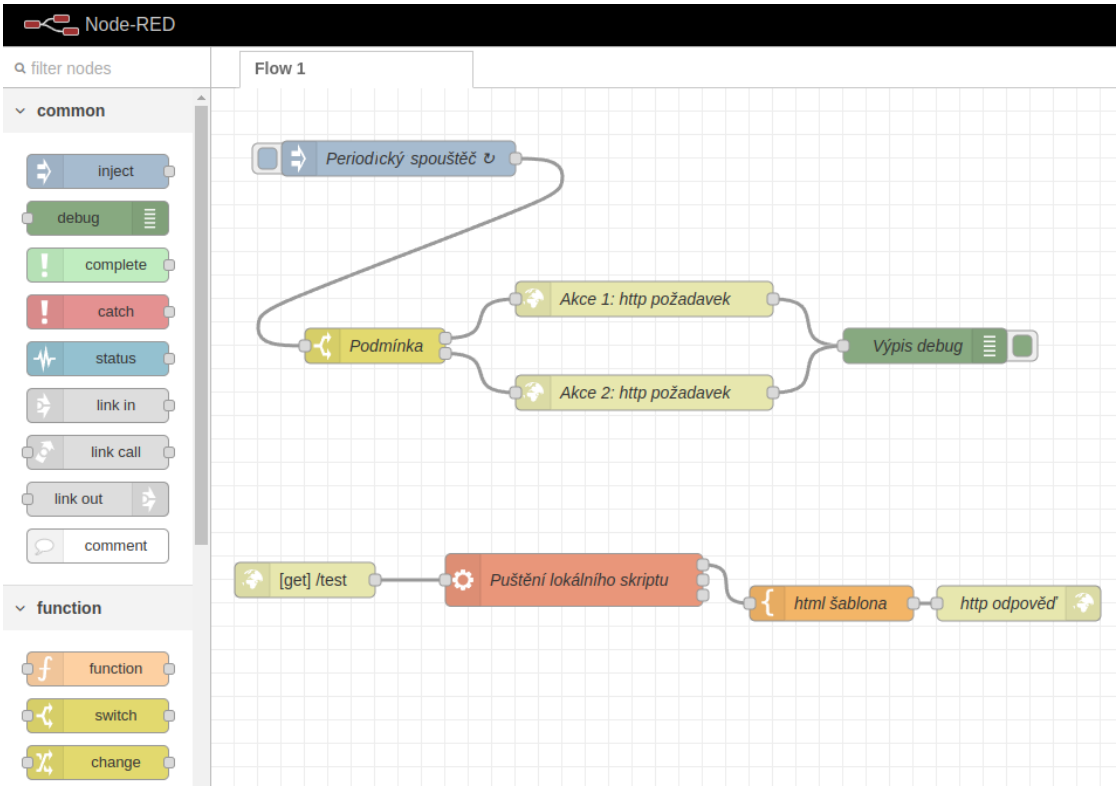
2.2 Výběr konceptu uživatelského rozhraní

V průběhu analýzy a návrhu GUI vznikly dva rozdílné koncepty. V této kapitole je představím a odůvodním svůj výběr.

2.2.1 Rozhraní založené na toku informací mezi uzly

První návrh, který vznikl je inspirovaný grafickým rozhraním aplikace Node-RED. Jedná se o jednoduchý nástroj na vizuální programovaní založený na konceptu toku informací mezi uzly. Lze si představit, že každý uzel funguje jako spouštěč, podmínka, modifikátor nebo akce. Jednotlivé uzly mohou mít vstupy a výstupy. Spojováním uzlů mezi sebou, jakoby provázkem, se tvoří program. Uzly které slouží jako spouštěče čekají na událost, která když nastane, pošle data do dalšího uzlu se kterým je spojená. Postupnému předávání informací mezi uzly vniká tok. Každý validní tok má počátek a konec, začíná spouštěčem a končí akcí. Po cestě toku může být libovolný počet uzlů, kde každý může mít různou roli.

Na obrázku 2.1 je příklad jednoduchého příkladu z popisované aplikace. V příkladu je nakonfigurované periodické odesílání http požadavku a jeho následné vypsání do konzole a pod tím spouštění skriptu při navštívení url „/test“ a vrácení jeho výsledku.

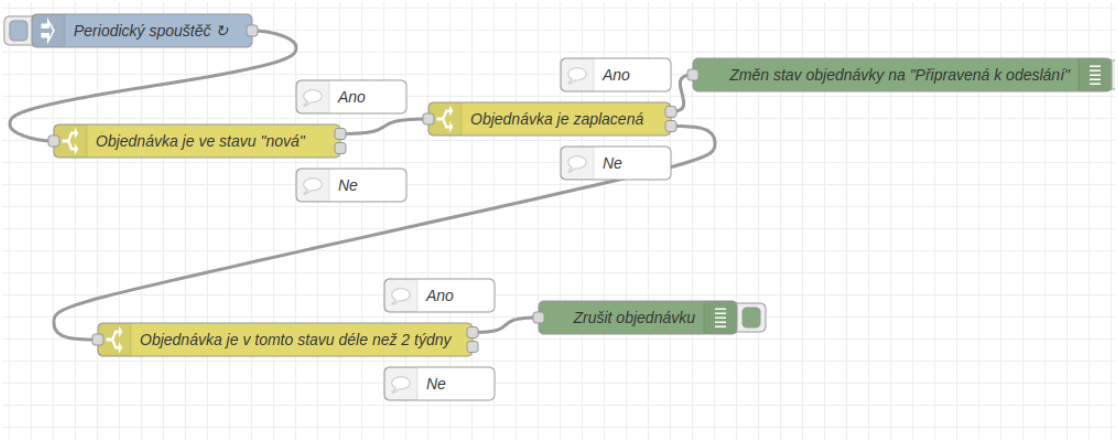


Obrázek 2.1 Ukázka programu Node-RED

Node-RED se nejčastěji používá na IoT (Internet věcí). Lze v něm vytvořit velké množství projektů a vyčnívá svojí jednoduchostí a flexibilitou.

Právě tento program mě inspiroval k návrhu, který je konceptuálně velmi podobný tomuto nástroji. Vezmeme-li zmiňované uzly spouštěčů, podmínek a akcí, lze si představit nástroj, který jde aplikovat i na problematiku automatizace objednávek. Uzly představující spouštěče mohou naplňovat funkční požadavek F2 a uzly představující podmínky či akce mohou naplňovat funkční požadavek F1. Node-RED samotný využít na problematiku automatizace a její uživatelské konfigurace přímo nejde, ale graficky je velmi podobný a slouží jako užitečný příklad, na jehož základě je možnost vytvořit funkční aplikaci na potřebnou problematiku.

Obrázek 2.2 vyjadřuje, jak by podobný nástroj mohl být využitý na nastavení automatizace na změnu stavu objednávky na „připravená k odeslání“ pokud je objednávka ve stavu „nová“ a zároveň má příznak „zaplacená“. Nebo opačně v případě že objednávka je ve stavu „Nová“ déle než 2 týdny objednávku zrušit.



Obrázek 2.2 Ukázka využití konceptu Node-RED na problematiku automatizace objednávek

Výhody:

Znovupoužitelnost uzlů Možnost spojení jednoho uzlu s více dalšími. Lze skládat podmínky paralelně či sériově mezi sebou. To umožňuje využít jednu podmínku i pro dvě naprosto rozdílné události. Pro pokročilé konfigurace tato vlastnost pomáhá přehlednosti a efektivitě práce.

Flexibilita Skládáním podmínek paralelně či sériově umožňuje vytvářet komplexní konfigurace.

Přehlednost Možnost posouvání uzlů tahem myši po pracovní ploše. Lze uspořádat uzly dle potřeby uživatele.

Nevýhody:

Složitost Pro nezkušené uživatele je tento návrh matoucí a musí investovat více času k pochopení celého konceptu.

Implementace Realizace tohoto návrhu by vyžadovala hodně času.

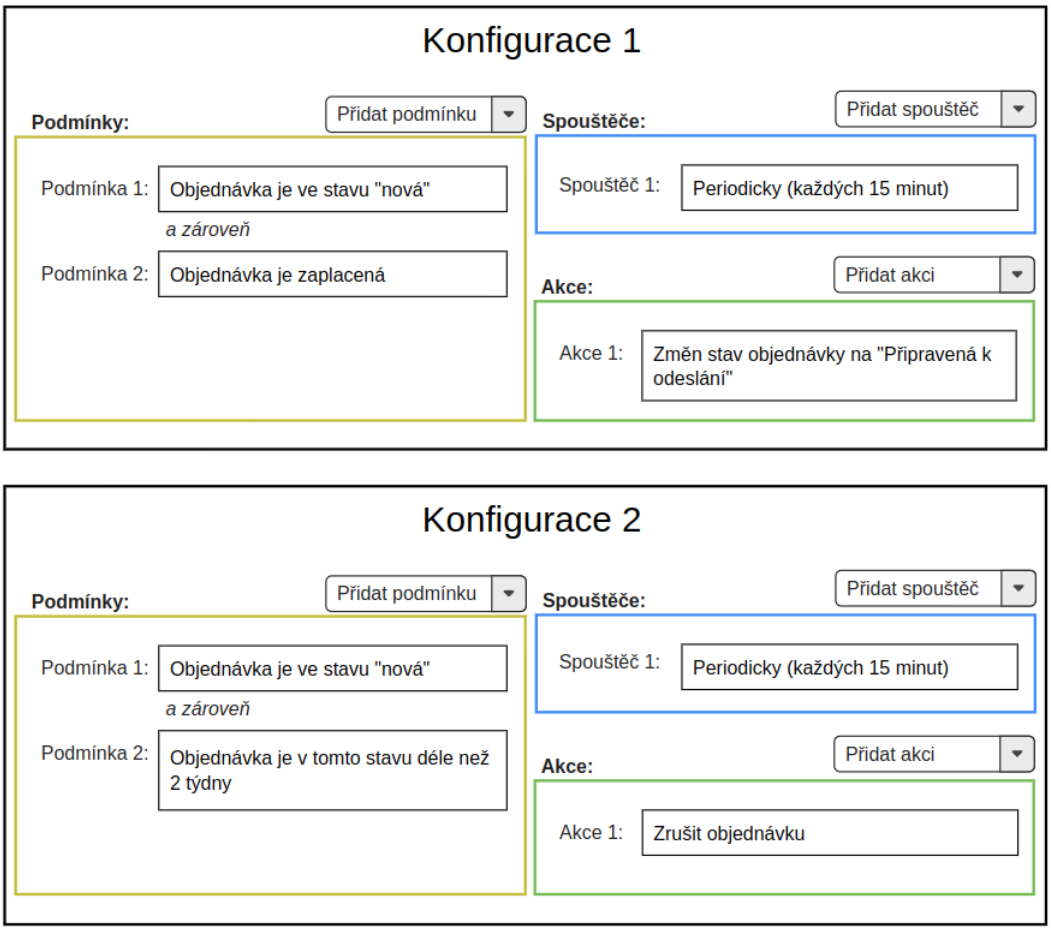
Závěr: Tento koncept předchází vzniku velmi flexibilního nástroje na automatizaci. Uživatel má velký prostor k vytvoření vlastní konfigurace. Vhodné pro zkušenější uživatele.

2.2.2 Rozdělení na bloky spouštěčů, podmínek a akcí

Druhý návrh je založen na rozdělení spouštěčů, podmínek a akcí na oddělené bloky. Vznikl jako reakce na nedostatky prvního návrhu po konzultaci s vedoucím projektu. Zaměřuje se více na přívětivější rozhraní pro nezkušené uživatele a vychází z dlouholetých zkušeností prací s klienty.

Rozdělením stěžejních pojmů na bloky pomáhá nezaměňovat účel spouštěčů, podmínek a akcí. Blok spouštěčů je vázaný přímo na blok podmínek a blok podmínek je přímo vázaný na blok akcí. Spouštěče opět určují, při jakých událostech se mají podmínky na jednotlivých objednávkách kontrolovat. Pokud jsou podmínky pro objednávku splněny, provede se pro objednávku blok akcí.

Přibližný návrh tohoto konceptu je vidět na obrázku 2.3, který řeší stejný příklad jako příklad z předchozího návrhu 2.2. Na rozdíl od předchozího návrhu je nutno tento příklad v tomto návrhu realizovat pomocí dvou rozdílných konfigurací, protože je potřeba aplikovat dvě různé akce pro jinou skupinu podmínek. Bloky spouštěčů, podmínek a akcí jsou barevně odděleny.



Obrázek 2.3 Ukázka využití konceptu rozdělení na bloky

Výhody:

Jednoduchost návrhu Návrh je jednodušší na vysvětlení. Přívětivější pro běžné uživatele.

Implementace Návrh je jednodušší k realizaci, tudíž i poté k údržbě. Oddělení spouštěčů, podmínek a akcí do samostatných bloků zjednodušuje komplexitu.

Nevýhody:

Flexibilita Podmínky nelze sériově či paralelně skládat. Nutnost vytvoření více oddělených konfigurací, pokud se liší akce pro jinou sadu podmínek.

Závěr: Vzhledem k cílové skupině uživatelů, jednoduchost návrhu hraje velkou roli ve výběru konceptu, proto jsem zvolil tento návrh. Dále v textu se předpokládá použití tohoto návrhu.

2.3 Případy užití

Diagramy případů užití se obvykle označují jako diagramy chování, které se používají k popisu souboru činností (případů užití), které by měl nebo může provádět nějaký systém ve spolupráci s jedním nebo více externími uživateli systému (aktéry). Každý případ užití by měl poskytovat nějaký pozorovatelný a hodnotný výsledek pro aktéry nebo jiné zúčastněné strany systému. [14]

2.3.1 Aktéři případů užití

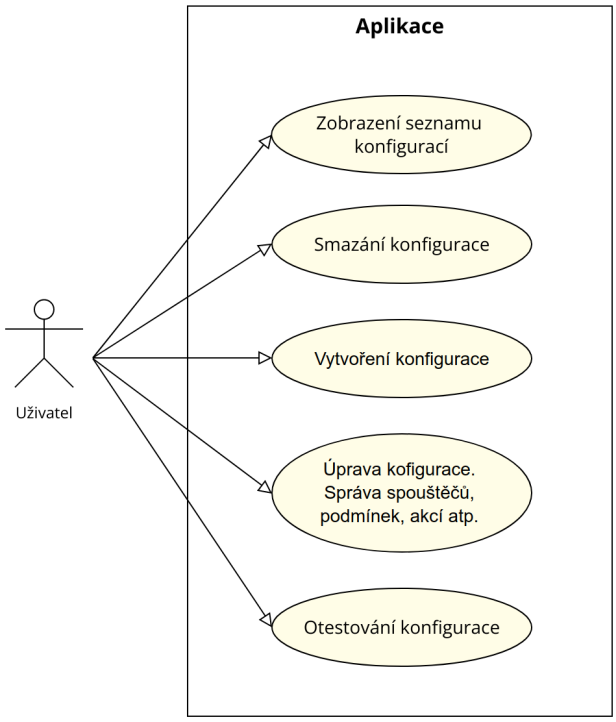
Popis aktérů vyskytujících se v následujících případech užití.

Uživatel Jedná se o uživatele aplikace který pracuje v GUI.

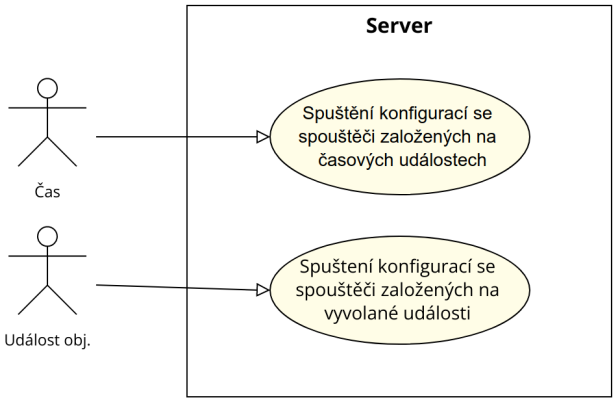
Čas Reprezentuje spouštěče v konfiguraci objednávek založených na časových událostech.

Událost objednávky Pro každou objednávku může nastat událost, na kterou spouštěč může reagovat. Například změna stavu objednávky nebo vytvoření objednávky.

2.3.2 Diagramy případů užití



Obrázek 2.4 Případy užití uživatele



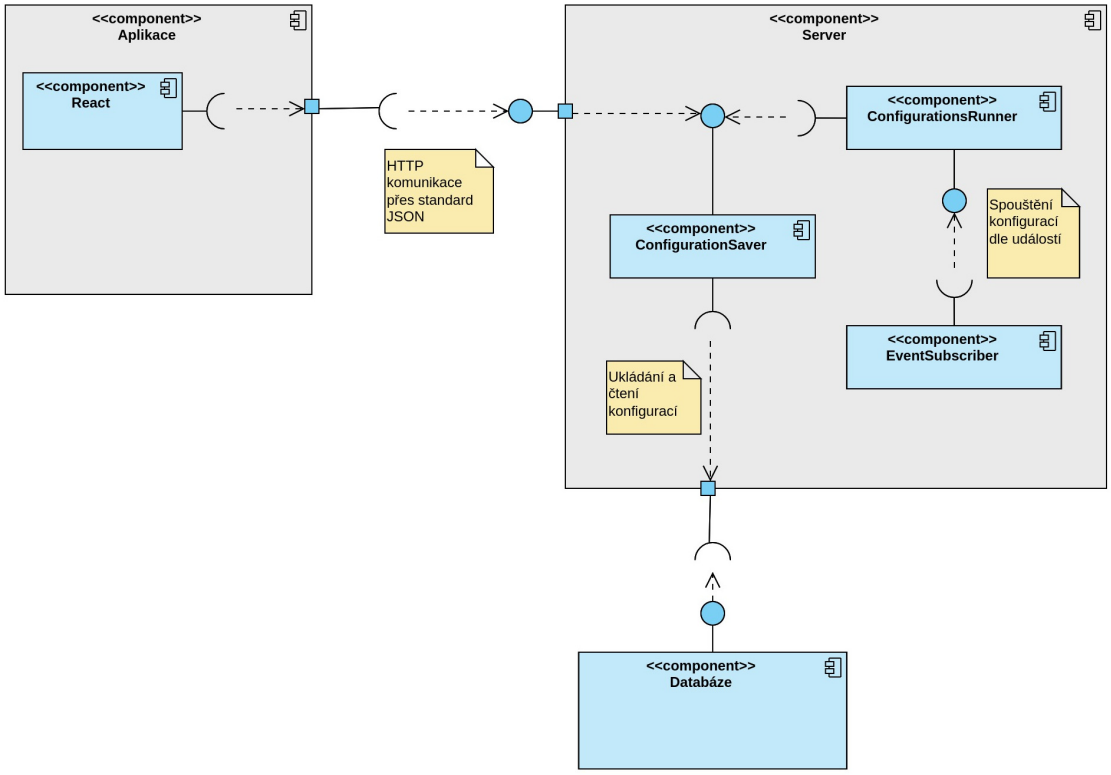
Obrázek 2.5 Případy užití událostí

2.4 Diagram komponent

Diagramy komponent se kreslí, aby pomohly modelovat detaily implementace a kontrolovat, zda jsou všechny aspekty požadovaných funkcí systému pokryty plánovaným vývojem. [15]

Komponenta je modulární část systému, která zapouzdřuje svůj obsah a jejíž projev je nahraditelný (tj. komponenty, poskytující ekvivalentní funkcionalitu založenou na kompatibilitě jejich rozhraní, mohou být libovolně zaměňovány, a to buď už v čase tvorby designu, nebo až za běhu cílového systému) [16]

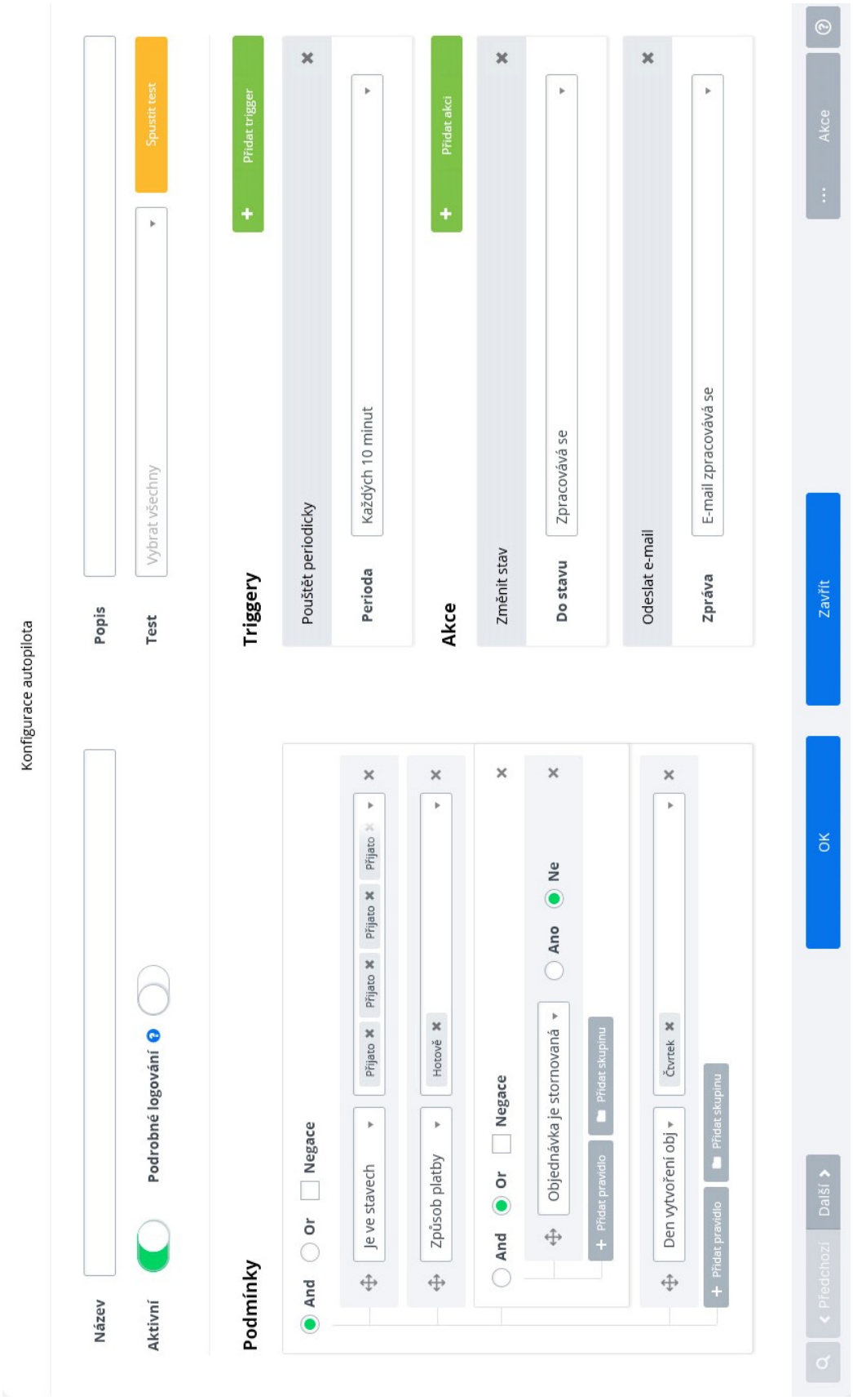
Na následujícím diagramu komponent jsou znázorněny vztahy mezi hlavními komponentami systému. Uživatel v aplikaci pomocí React knihovny vytváří a edituje konfigurace automatizace, které jsou posílány na server ve formátu JSON, kde je komponenta `ConfigurationSaver` uloží do databáze. Komponenta `EventSubscriber` reaguje na události a pošle je do komponenty `ConfigurationRunner`, ta přes `ConfigurationSaver` získá konfigurace a vybere všechny odpovídající konfigurace se spouštěčem čekajícím na tuto událost. Vybrané konfigurace spustí na objednávkách, kde se zkontrolují podmínky a v případě jejich splnění provede akce. Viz. obrázek 2.6.



Obrázek 2.6 Diagram komponent

2.5 Grafický návrh uživatelského rozhraní

Pro vybraný koncept GUI 2.2.2 vznikl pokročilý grafický návrh. Zavedení tohoto designu není součástí této práce.



Obrázek 2.7 Pokročilý grafický návrh GUI

Kapitola 3
Implementace

Kapitola popisuje realizaci aplikace z návrhu a je rozdělená na klientskou a serverovou část implementace.

3.1 Struktura řešení

Implementace se dělí na klientskou část a serverovou. Klientská část je zaměřená na GUI, kde uživatelé mají možnost nakonfigurovat požadovanou automatizaci objednávek. Serverová část řeší ukládání dat a spouštění uživatelem nakonfigurované logiky.

Jak jsem již popisoval v kapitole 2.4. Komunikace mezi těmito dvěma komponentami probíhá ve formátu JSON. Klientská část (klient) přijme ze serveru nastavení konfigurovatelných elementů. Uživatel libovolně nakonfiguruje nastavení automatizace a při uložení klient odešle na server konfiguraci automatizace, která se uloží do databáze. Poté se podle událostí vybírají uložené konfigurace a spouští se. Vybrané konfigurace jsou takové, které mají k událostem odpovídající spouštěč.

3.2 Klientská část

V této kapitole je popsána implementace klientské části aplikace. Popsány jsou využité knihovny a komponenty z kterých je aplikace poskládána.

Grafické rozhraní je implementováno podle návrhu rozdělení na bloky 2.2.2. Pro jeho tvorbu jsem využil JS knihovnu React. Hlavním cílem je vytvořit přívětivé GUI, ve kterém uživatel může nakonfigurovat požadovanou automatizaci a uložit ji.

Grafický návrh 2.7 je finální podoba GUI, která není součástí implementace a bude realizována v budoucnu.

3.2.1 Využité knihovny

V aplikaci jsem využil několik knihoven. Mezi ně patří například knihovny pro vylepšení formulářových prvků [1], knihovna na generování UUID (Pro lepší práci s interními strukturami) a nejdůležitější část – knihovna pro GUI na stavbu logických výrazů.

3.2.1.1 React awesome query builder

Uživatelsky přívětivá React komponenta pro sestavování dotazů nebo výrazů. Inspirovaná je knihovnou jQuery QueryBuilder[2]. [3] [18]

Poskytuje GUI, které umožňuje sestavit logický výraz. Lze vytvořit libovolný počet polí, ty je možné vkládat do podskupin, spojovat pomocí základních logických operátorů a negovat. Pole mohou být v kontextu objednávek například cena, kategorie, status platby atp.

Knihovna je vysoce konfigurovatelná, tudíž lze přidávat vlastní pole různých typů. Také podporuje více formátů ve kterém lze sestavený výraz zobrazit. Pro účely mé aplikace jsem použil formáty JS výrazu a JSON struktury, ty poté zpracuji v serverové části aplikace.

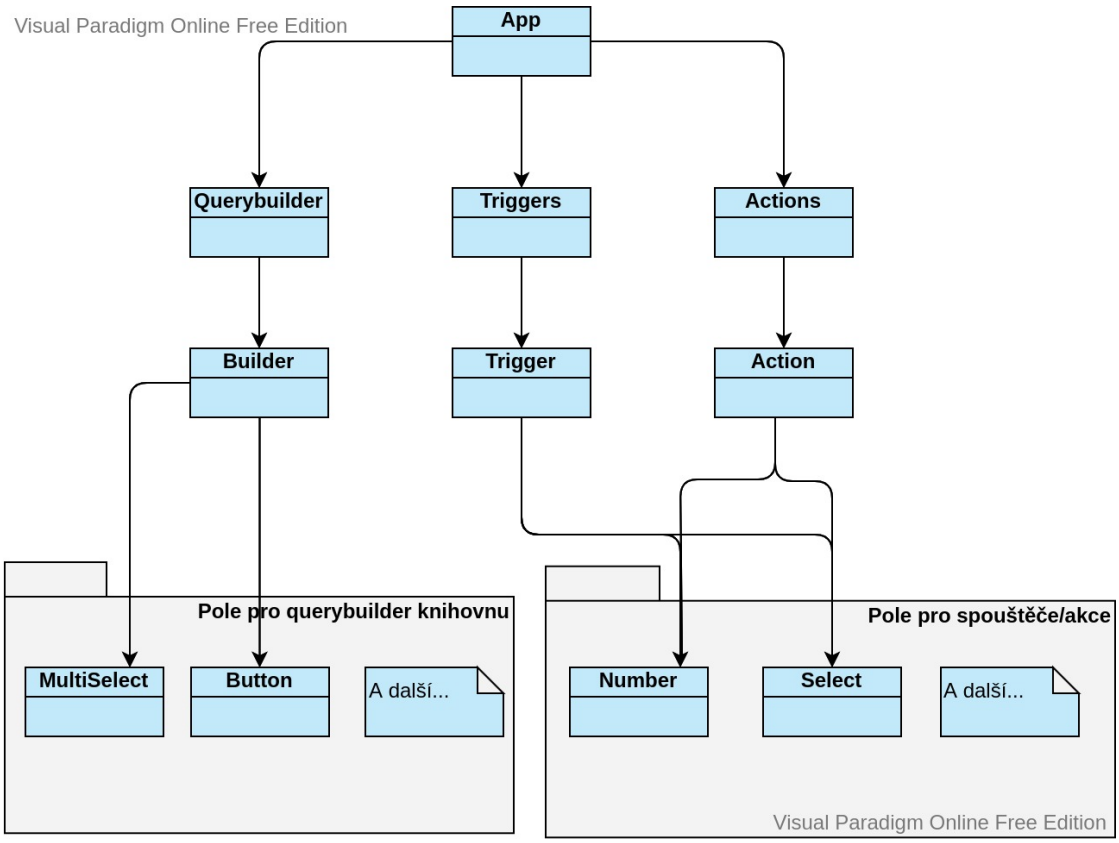
Toto je ideální nástroj na realizovaní podmínek, které objednávka musí splňovat, aby se provedli akce. Lze v ní vytvořit dostatečně pokročilý výraz, aby to pokrylo většinu požadavků od uživatele. Není potřeba implementovat podobný nástroj znovu.

3.2.2 Komponenty

React komponenty jsou nezávislé a opakovaně použitelné části kódu. Slouží ke stejnému účelu jako funkce JavaScriptu, ale pracují samostatně a vracejí HTML. [19]

[1] Vstupní pole, tlačítka a přepínače.
[2] `https://querybuilder.js.org/`
[3] jQuery je JS knihovna, ulehčující práci s HTML elementy, animacemi a asynchroním načítáním. [17]

Struktura popisované aplikace se dělí do několika komponent. Celý princip je založen na rozdělení spouštěčů, podmínek a akcí do na sobě nezávislých komponent. Jak je vidět na následujícím diagramu, struktura je skutečně rozdělena z komponenty `App` (kořene stromu) na tři větve, kde každá větev reprezentuje jednotlivé bloky z návrhu.



Obrázek 3.1 Struktura React komponent

3.2.2.1 Popis komponent

App je kořenová komponenta. Zpracovává JSON data ze serveru.

Triggers zaobaluje všechny elementy spouštěčů, vytváří jednotlivé spouštěče.

Trigger reprezentuje jednotlivý spouštěč. Každý spouštěč může obsahovat několik uživatelsky konfigurovatelných polí.

Pole pro spouštěče/akce jsou skupina komponent, které slouží jako typy konfigurovatelných polí pro spouštěče a akce. Každé konfigurovatelné pole má nějaký typ. Na diagramu jsou vidět např. komponenty `Number` a `Select`, ale může jich být mnohem více.

Actions podobně jako komponenta `Trigger` zaobaluje elementy akcí.

Action reprezentuje jednotlivou akci. Opět, jako spouštěče, může obsahovat několik uživatelsky konfigurovatelných polí.

QueryBuilder je komponenta reprezentující blok podmínek. Zaobaluje komponentu `Builder`, která je rozšířením z knihovny React awesome query builder.

Builder je komponenta z knihovny. V kapitole 3.2.1.1 je knihovna popsána.

3.2.3 Stavy aplikace

V Reactu má každá komponenta svůj stav (state). Občas je ale potřeba, aby na změnu stavu jedné komponenty reagovaly i ostatní komponenty. To může být vyřešené například předáváním callback funkce mezi komponentami, nicméně čím větší je aplikace, tím se přehlednost kódu zmenšuje a předáváním velkého množství stavů tímto způsobem je velmi nepřehledné.

K řešení tohoto problému jsem využil React komponentu Context, která poskytuje způsob, jak předávat data ve stromu komponent, aniž by bylo nutné předávat stavy na každé úrovni ručně. Slouží ke správě globálního stavu a poskytuje data komponentám bez ohledu na to, jak hluboko ve stromu komponent se nacházejí. [20]

3.3 Serverová část

V této kapitole je popsána implementace serverové části aplikace. Popsány jsou využité knihovny, způsob vyhodnocování podmínek automatizace, princip konfiguračních elementů, reprezentace uživatelské konfigurace, zaznamenávání aktivit, ukládání dat a automatizované.

Serverová část je implementována jako balíček v Symfony frameworku. Tato část se stará o ukládání konfigurací automatizace a jejich provádění.

3.3.1 Využité knihovny

Composer je nástroj pro správu závislostí v PHP. Umožňuje vám deklarovat knihovny, na kterých váš projekt závisí, a bude je spravovat (instalovat/aktualizovat) za vás.[21]

Symfony framework využívá právě tohoto nástroje ke správě knihoven.

3.3.1.1 js2php

Knihovna [4] slouží k parsování [5] JavaScriptu, který následně co nejpřesněji překládá do kódu PHP. [23] Knihovna využívá konceptu PEG (Parsing expression grammar), což je typ analytické formální gramatiky, neboli popisuje formální jazyk pomocí souboru pravidel pro rozpoznávání řetězců v jazyce. [24]

Ve své práci využívám tuto knihovnu pro překládání JS výrazů do PHP kódu, který na straně serveru mohu spustit. JS výrazy generuje knihovna React awesome query builder 3.2.1.1.

3.3.2 Vyhodnocování podmínek

Jedna z hlavních funkcí, kterou systém musí splňovat, je z uživatelsky nakonfigurovaných podmínek určit, jaké objednávky je splňují, aby se akce provedli jen pro správné objednávky. Proto pro nakonfigurované podmínky se ukládají 2 typy objektů – JS výraz a JSON struktura reprezentující logiku a složení podmínek. Oba tyto objekty jsou generovány na straně klienta a odesílají se na server, kde se uloží do databáze k pozdějšímu vyhodnocení.

JS výraz se využívá k filtrování na aplikační vrstvě serveru. Filtrují se objekty reprezentující objednávky. K tomuto účelu slouží, jak už bylo naznačeno, knihovna js2php (viz. 3.3.1.1), která

[4] https://github.com/jakubkulhan/js2php

[5] Jde o proces, při kterém se kompilují a interpretují počítačové programy [22]

překládá JS výraz na PHP kód. Vzhledem k tomu, že se toto provádí na aplikační vrstvě, může tato operace při velkém množství kontrolovaných objednávek být velmi časově náročná.

K vyvážení náročnosti prvního způsobu filtrování používám zmiňovanou JSON strukturu, kterou lze jednoduše rozložit na jednotlivé podmínky a pomocí toho vyfiltrovat velké množství objednávek už při dotazovaní do databáze. Databázové servery umožňují rychle vyfiltrovat a vybrat data. Např. relační databáze mohou výrazně zvýšit rychlost dotazování pomocí indexů. Na druhou stranu, u komplexních podmínek může být obtížné složit databázový dotaz takový, aby přesně odpovídal nakonfigurovaným podmínkám, obzvlášť když výraz podmínek obsahuje závorky, různé operátory, negace nebo podmínka závisí na datech mimo databázi.

Oba způsoby mají své výhody a nevýhody. Zkombinování těchto dvou způsobů však tvoří optimální řešení, stačí pouze aby každý element podmínky měl dvě následující metody (funkce). První metoda je taková, která se vykoná před spuštěním dotazu do databáze a dotaz upraví. Tímto způsobem lze vyfiltrovat objednávky už při dotazování do databáze. Druhá metoda vrací data do proměnné pro v JS výrazu. Metody se volají při vyhodnocování výrazu.

Pro vysvětlení, jak tyto dva způsoby jsou zkombinovány, si představme nakonfigurovaný výraz podmínek jako strom, kde každý list představuje jednotlivou podmínku a vnitřní vrcholy jsou operátory. V první úrovni stromu (pod kořenem) se pro každý list (podmínku) zavolá první metoda pro upravení dotazu. Po vybrání objednávek z databáze se pro každý objekt představující objednávku spustí JS výraz přeložený do PHP a zkontroluje se zda výraz splňuje.

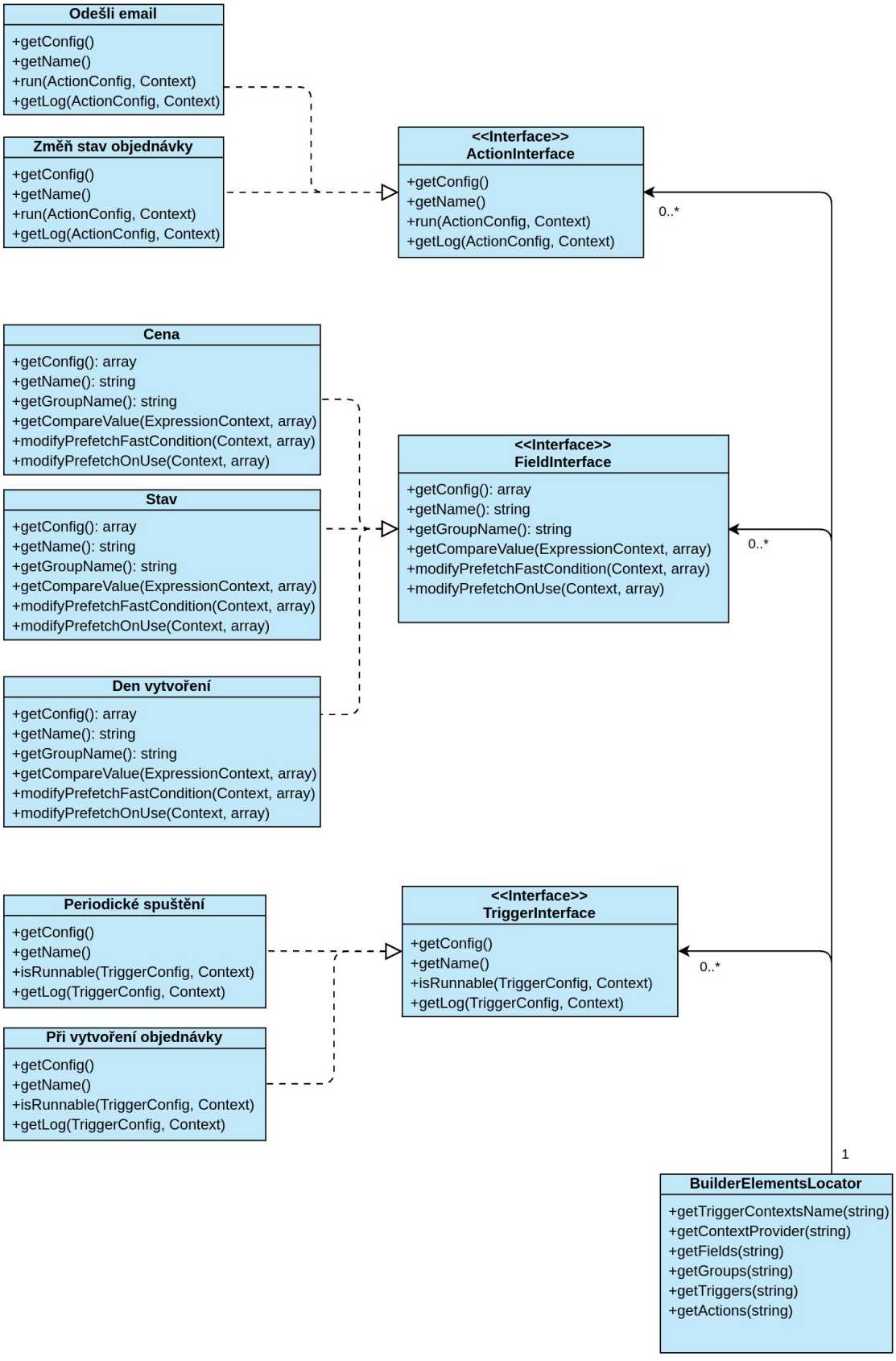
Důvod, že první metoda elementů podmínek se volá pouze pro první úroveň listů ve stromu, je takový, že při implementaci této metody není potřeba přemýšlet nad závorkováním a různými operátory celého výrazu.

Další výhodou tohoto způsobu vyhodnocování je umožnění vytvoření pokročilého podmínkového bloku obsahující libovolný JavaScript výraz.

3.3.3 Konfigurační elementy

Konfigurační elementy jsou stavebními prvky konfigurací automatizace. Dělí se na tři typy – spouštěče, podmínky a akce. Každý typ má svoje vlastní rozhraní. Každý element musí implementovat jedno z rozhraních, aby se zařadil mezi správný typy elementů. Spouštěče musí implementovat `TriggerInterface`, podmínky `FieldInterface` a akce `ActionInterface`. Třída `BuilderElementsLocator` dokáže implementované elementy najít a vrátit jejich instance.

Na dalším obrázku 3.2 je vidět diagram tříd. Příklady implementovaných elementů jsou nalevo a rozšiřují vždy svůj typ rozhraní. Dále jsou popsány jednotlivé metody rozhraních.



Obrázek 3.2 Diagram tříd stavebních elementů

3.3.3.1 TriggerInterface

getConfig vrací konfiguraci, která specifikuje typ a chování elementu v GUI.

getName určuje unikátní jméno. Díky této hodnotě se jednotlivé elementy rozeznávají mezi sebou.

isRunnable při spuštění události se stejným jménem rozhoduje, zda je spouštěč spustitelný. To závisí na jeho uživatelsky nastavitelných polích a jejich hodnotách. První parametr typu `TriggerConfig` obsahuje uživatelem nastavené hodnoty a druhý parametr `TriggerEvent` obsahuje kontext, se kterým je spouštěč momentálně zavolán. Výsledkem funkce je boolean.

modifyPrefetchContext má možnost zúžit množinu načtených objednávek. Např. vybrat objednávku podle identifikátoru.

getLogDescription vrací řetězec, který bude zapsán do logu.

3.3.3.2 FieldInterface

getGroupName určuje do jaké skupiny pole patří. V GUI se pole zařadí do skupiny s tímto jménem. Slouží pouze k organizaci polí.

getCompareValue se volá při vyhodnocování JS výrazu pro každou proměnnou či funkci kterou řetězec obsahuje. První parametr obsahuje konkrétní objekt objednávky na které je JS výraz spuštěný. V případě funkce druhý parametr obsahuje argumenty volané funkce z JS výrazu.

modifyPrefetchFastCondition slouží, jak je popsáno v kapitole 3.3.2, k přednačítání pro podmínky v první úrovni stromu podmínek. První parametr obsahuje dotaz který lze před spuštěním upravit a druhý parametr obsahuje část z JSON struktury týkající se této podmínky.

modifyPrefetchOnUse funguje podobně jako `modifyPrefetchFastCondition`. Opět slouží k modifikaci dotazu načítající objednávky. Jen s tím rozdílem, že se metoda volá pouze jednou pro každý typ podmínky použité ve výrazu. Například může být využito k připojení tabulky k dotazu.

3.3.3.3 GroupInterface

getConfig vrací pole s názvem identifikující skupinu a zobrazitelný název.

3.3.3.4 ActionInterface

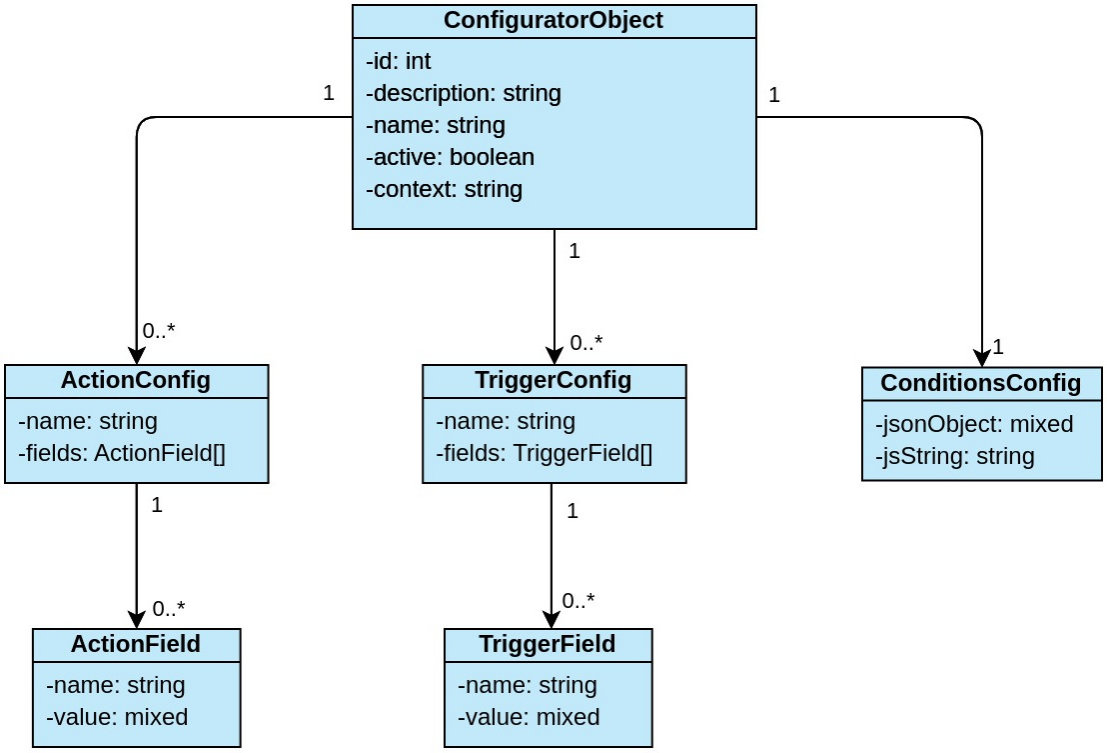
run obsahuje logiku která se provádí při spuštění akce.

getLog vrací řetězec který se bude zapsán do logu.

3.3.4 Uživatelská konfigurace

Každé uživatelské nastavení jednotlivých konfigurací automatizace je interně reprezentováno třídou `ConfiguratorObject`. Třída se skládá z množin tříd představujících nastavení jednotlivých elementů. Uživatelské nastavení každého spouštěče reprezentuje třída `TriggerConfig`. Nastavení každé akce obsahuje třída `ActionConfig`. JS výraz a JSON strukturu obsahuje třída

`ConditionsConfig`. Každý element spouštěče/akce může obsahovat libovolný počet konfigurovatelných polí. Proto třídy pro tyto dva typy elementů navíc obsahují navíc množinu tříd `TriggerField/ActionField`, které reprezentují hodnoty jednotlivých polí elementu. Celá tato struktura je vidět na následujícím UML diagramu tříd 3.3.

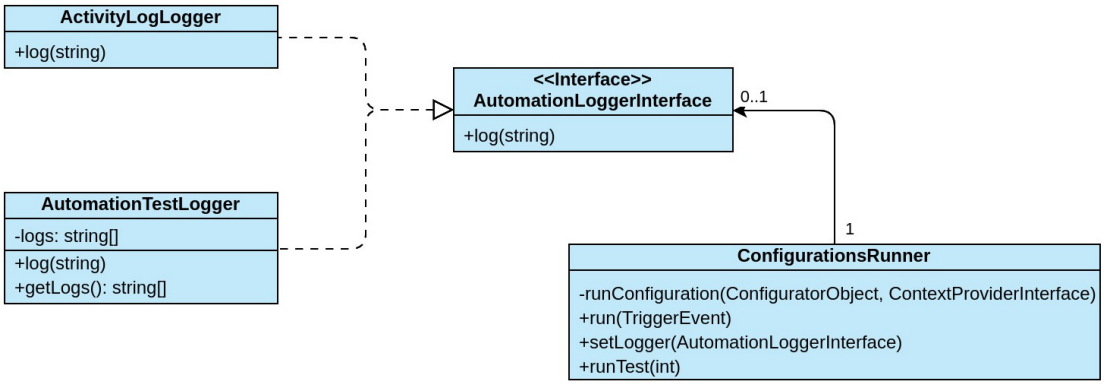


Obrázek 3.3 Diagram tříd znázorňující strukturu třídy uživatelské konfigurace

3.3.5 Záznam aktivit

Třída `ConfigurationsRunner`, zpracovávající a provádějící uložené konfigurace automatizace, využívá třídy implementující rozhraní `AutomationLoggerInterface`, které slouží k zaznamenávání provedených akcí a dalších operací. Rozhraní obsahuje metodu `log`, která přijímá řetězec. Třídy které rozhraní implementují mají možnost zpracovat předaný řetězec podle potřeby. Pomocí metody `setLogger` třídy `ConfigurationsRunner` lze předat libovolnou třídu rozšiřující zmiňované rozhraní. Díky tomu lze vytvořit vlastní třídu, která řetězec zpracovává, jak je potřeba.

Na následujícím obrázku 3.4 je vidět implementace dvou tříd – `ActivityLogLogger` a `AutomationTestLogger`. Třída `ActivityLogLogger` zaznamenává řetězce do databáze, ze které je lze poté využít například ke zpětnému dohledání chování automatizace. Druhá třída `AutomationTestLogger` se využívá při testování konfigurací, kde se akce neaplikují na produkční data, proto řetězce není žádoucí zaznamenávat do databáze a stačí pouze po skončení běhu automatizace řetězce získat metodou `getLogs` a vypsat je uživatelovi.



Obrázek 3.4 Diagram tříd znázorňující použití rozhraní `AutomationLoggerInterface`

3.3.6 Automatizované testy

V Symfony aplikaci se využívá automatizovaných testů k testování konfiguračních elementů. Je možnost, že v budoucnu bude potřeba některý z existujících elementů spouštěčů, podmínek a akcí upravit, ale aplikace bude již v produkci. V případě chybné úpravy by mohla automatizace i selhat a způsobit škody. Proto je nutné zavést automatizované testy, aby byla větší pravděpodobnost, že se chyba nalezne a opraví už před nasazením chybné aplikace.

Symfony využívá nezávislou knihovnou PHPUnit a tak poskytuje rozsáhlé testovací možnosti. [25]

Pro testování jsem použil integrační testy [6]. Podobně jako při běžném použití konfigurací automatizace, jen s tím rozdílem že se používá testovací databáze s testovacími daty, automatizovaný test vyvolá událost, na kterou zareagují konfigurace s odpovídajícím spouštěčem. Do služby `ConfigurationRunner` se nastaví instance `AutomationTestLogger` (viz. 3.3.5), tudíž zaznamenané aktivity lze po dokončení vyvolaných konfigurací automatizace jednoduše získat a lze ověřit, zda zaznamenané aktivity odpovídají předpokládaným aktivitám. Pro otestování konfiguračních elementů stačí tedy vytvořit několik konfigurací automatizace, které dohromady pokryjí všechny elementy potřebné k otestování.

[6] Integrační testování je definováno jako typ testování, při kterém jsou softwarové moduly kombinovány a testovány jako skupina. [26]

Kapitola 4
Ukázka aplikace

Kapitola obsahuje popis testovací aplikace, testovacích dat a ukázku funkčnosti celého řešení.

Pro otestování funkčnosti jsem vytvořil jednoduchou testovací aplikaci. Jedná se o Symfony projekt, tudíž lze rozšíření na automatizaci jednoduše do této testovací aplikace integrovat. Aby šlo co nejlépe vidět jak se konfigurace automatizace chová, lze v aplikaci vytvářet a spravovat objekty znázorňující objednávky, na kterých lze automatizaci vyzkoušet. Dále aplikace obsahuje pomocné funkce ulehčující testování, například manuální vyvolávání časových událostí nebo seznam pro zobrazení záznamu chování automatizace.

4.1 Testovací data

V testovací aplikaci jsou připravené testovací data skládající se z 10 objednávek a 2 konfigurací automatizace. Tyto data používají ukázky v dalších kapitolách. Ve zdrojových souborech lze nelézt skript `initData.sh`, který databázi vyprázdní a těmito daty poté naplní. Případně lze v aplikaci vytvořit vlastní data, či upravit stávající.

4.1.1 Struktura objednávky

Objekty objednávky, sloužící k testování funkčnosti rozšíření na automatizaci, v testovací aplikaci mají jednoduchou strukturu. Každá objednávka má 13 atributů, které objednávky mezi sebou rozlišují. Na základě těchto dat automatizace pracuje.

Tabulka 4.1 Struktura testovací objednávky

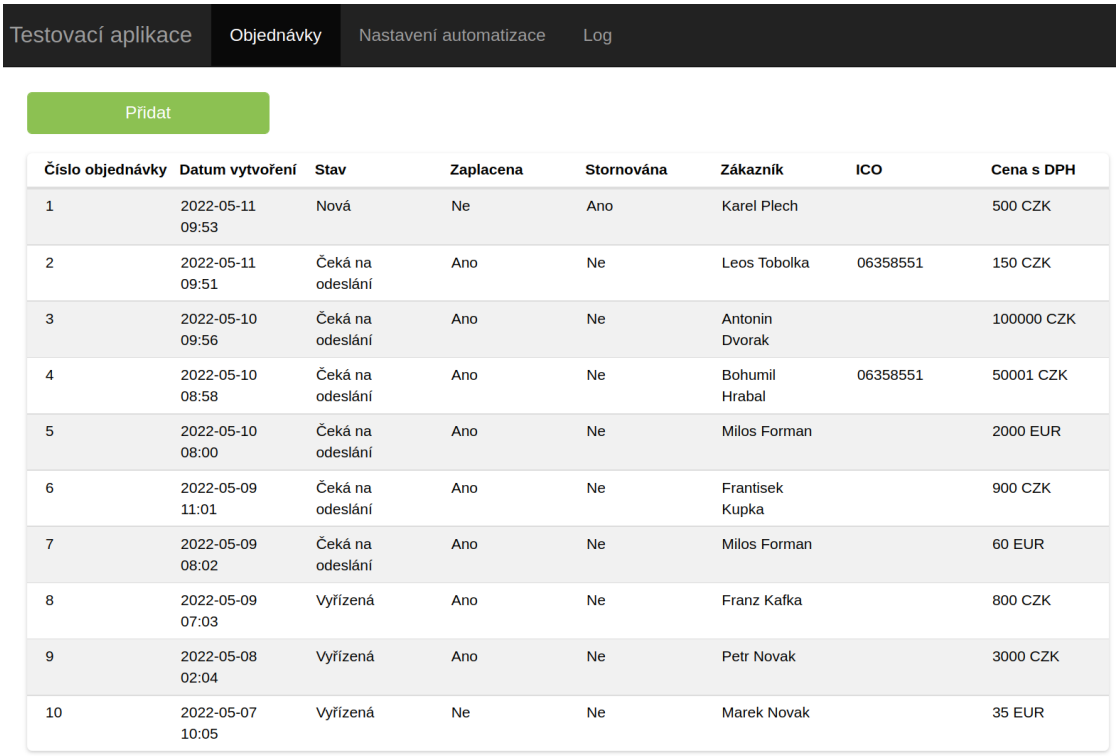
Název	Datový typ	Popis
Číslo objednávky	číslo	unikátní identifikátor
Jméno zákazníka	text	
Příjmení zákazníka	text	
Datum vytvoření	datum a čas	datum kdy objednávka byla vytvořena
Stav objednávky	číslo	stav ve kterém se objednávka nachází
Objednávka zaplacená	boolean	příznak zda je objednávka zaplacená
Objednávka stornována	boolean	
Cena objednávky	číslo (float)	cena objednávky s daní
Cena bez daně	číslo (float)	
IČO zákazníka	číslo	
PSČ	číslo	
Město	text	
Měna	text	měna objednávky

4.2 Popis aplikace

Aplikace obsahuje navigaci, pomocí které se lze dostat na úvodní stránku, seznam objednávek, seznam konfigurací automatizace a seznam záznamů chování automatizace. V podkapitolách dále lze vidět jednotlivé stránky detailněji.

4.2.1 Seznam objednávek

Seznam objednávek slouží pro zobrazení a úpravu objednávek. V seznamu lze vidět hlavní, ne však všechny, údaje objednávky. Jedná se o datum vytvoření objednávky, stav ve kterém se objednávka právě nachází, jestli je zaplacená či stornovaná, jméno zákazníka, identifikační číslo osoby (IČO) a celkovou cenu. Objednávky jsou seřazeny sestupně podle data vytvoření. Po kliknutí na objednávku se otevře její editace.

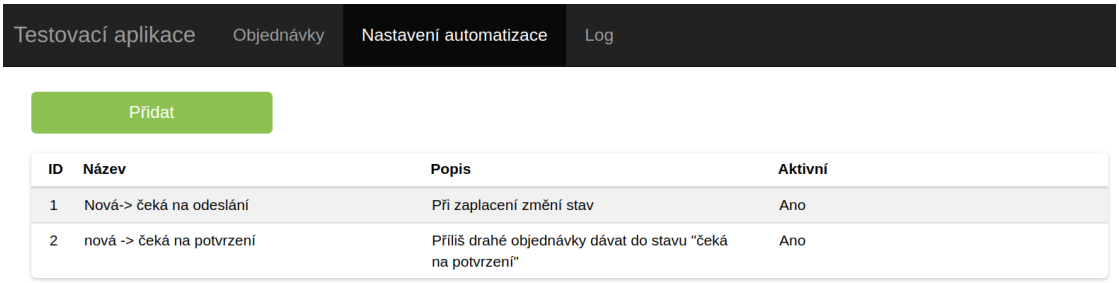


Číslo objednávky	Datum vytvoření	Stav	Zaplacena	Stornována	Zákazník	ICO	Cena s DPH
1	2022-05-11 09:53	Nová	Ne	Ano	Karel Plech		500 CZK
2	2022-05-11 09:51	Čeká na odeslání	Ano	Ne	Leos Tobolka	06358551	150 CZK
3	2022-05-10 09:56	Čeká na odeslání	Ano	Ne	Antonin Dvorak		100000 CZK
4	2022-05-10 08:58	Čeká na odeslání	Ano	Ne	Bohumil Hrabal	06358551	50001 CZK
5	2022-05-10 08:00	Čeká na odeslání	Ano	Ne	Milos Forman		2000 EUR
6	2022-05-09 11:01	Čeká na odeslání	Ano	Ne	Frantisek Kupka		900 CZK
7	2022-05-09 08:02	Čeká na odeslání	Ano	Ne	Milos Forman		60 EUR
8	2022-05-09 07:03	Vyřízená	Ano	Ne	Franz Kafka		800 CZK
9	2022-05-08 02:04	Vyřízená	Ano	Ne	Petr Novak		3000 CZK
10	2022-05-07 10:05	Vyřízená	Ne	Ne	Marek Novak		35 EUR

Obrázek 4.1 Seznam objednávek v testovací aplikaci

4.2.2 Seznam konfigurací automatizace

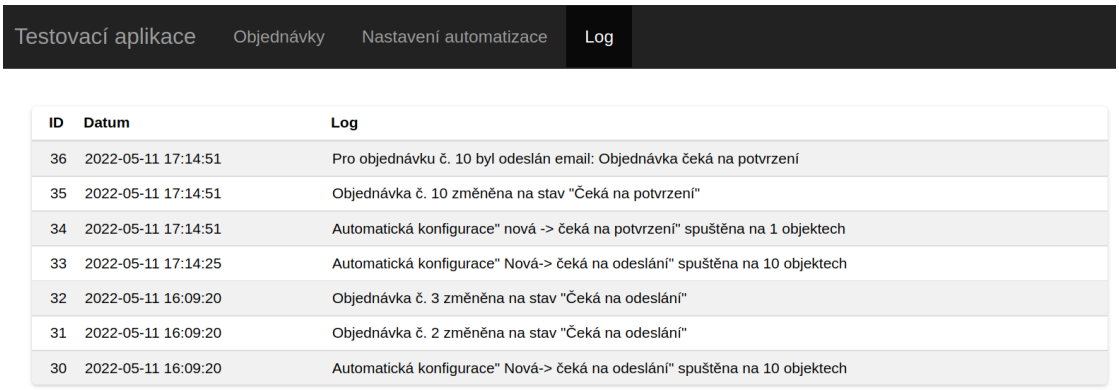
V tomto seznamu lze vidět všechny konfigurace automatizací. Seznam zobrazuje název, popis a zda je konfigurace aktivní, Po kliknutí na jednotlivé konfigurace se otevře konfigurátor, kde je podrobné nastavení elementů spouštěčů, podmínek a akcí.



Obrázek 4.2 Seznam konfigurací automatizace v testovací aplikaci

4.2.3 Log – záznam chování automatizace

Seznam obsahuje záznam operací provedených automatizací, případně další informace, které mohou pomoci při ladění automatizace. Například záznam číslo 30 na následujícím obrázku 4.3 s textem „Automatická konfigurace Nová-> čeká na odeslání spuštěna na 10 objektech“ značí že daná konfigurace s názvem „Nová-> čeká na odeslání“ vybrala 10 objednávek na kterých vyhodnocuje podmínky a vybírá pro které se mají provést akce.



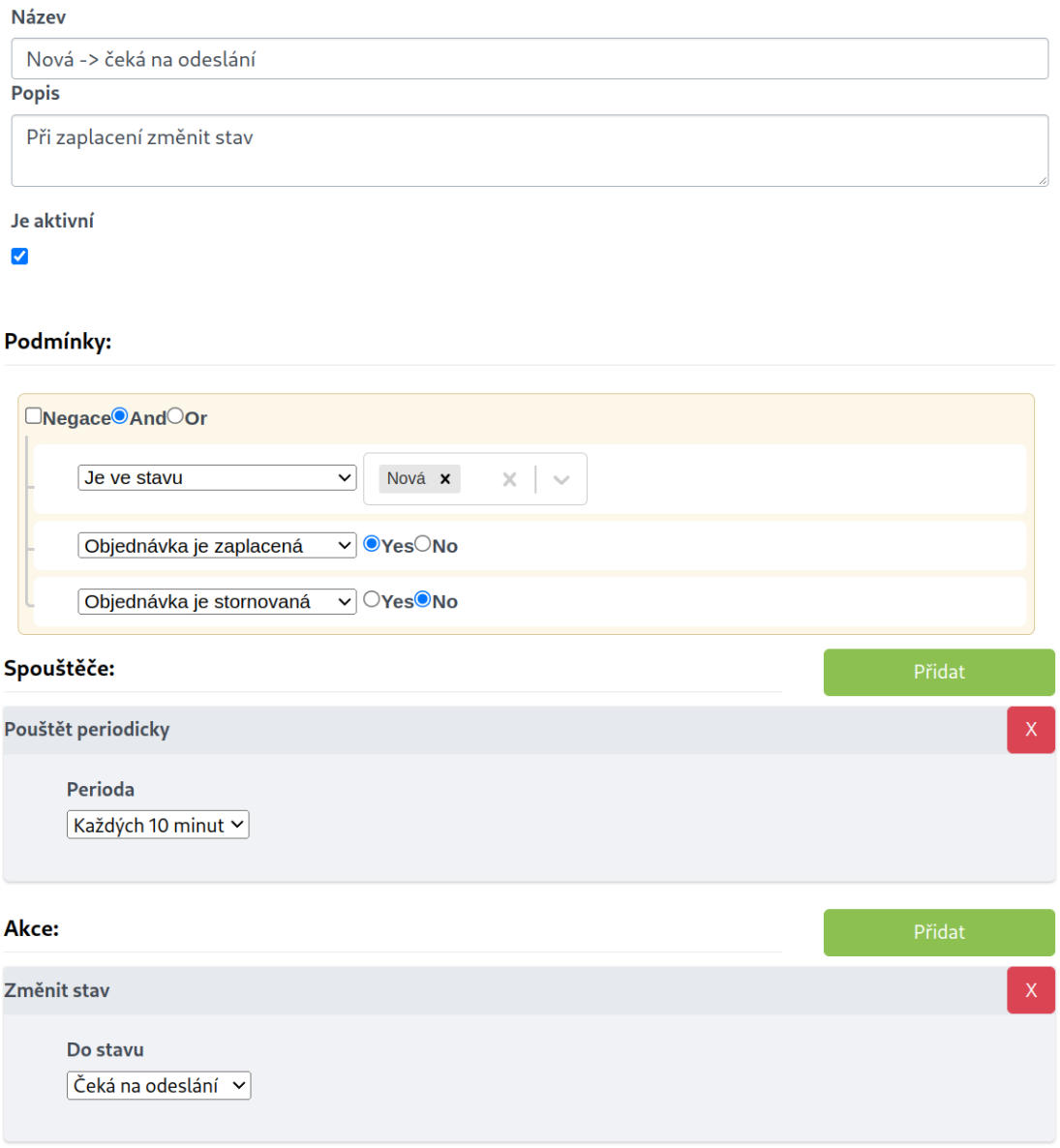
Obrázek 4.3 Záznam provedených akcí v testovací aplikaci

4.3 Ukázky

Předvedení vytvořeného konfigurátoru automatizace v testovací aplikaci.

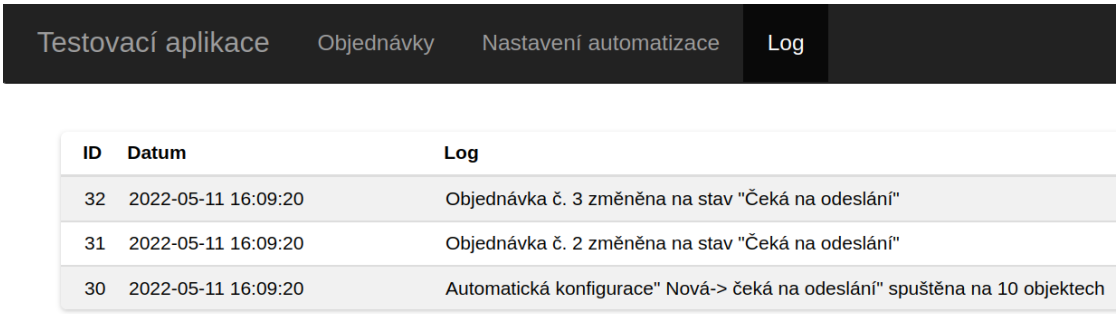
4.3.1 Ukázka 1

První ukázka konfigurace automatizace slouží pro přepínání stavu objednávky ze stavu „Nová“ do stavu „Čeká na odeslání“, pokud je objednávka zaplacená a zároveň není stornovaná. Využívá se periodický spouštěč, který se aktivuje každých 10 minut. Tento spouštěč je vhodný pro tuto konfiguraci z toho důvodu, že platby k objednávkám jsou v praxi přiřazovány většinou zpětně, tudíž je potřeba periodicky kontrolovat, zda už objednávka není zaplacená.



Obrázek 4.4 Konfigurace automatizace ukázky 1

Výsledný záznam chování po aplikování této konfigurace, jak je vidět na následujícím obrázku 4.5, může vypadat následovně:



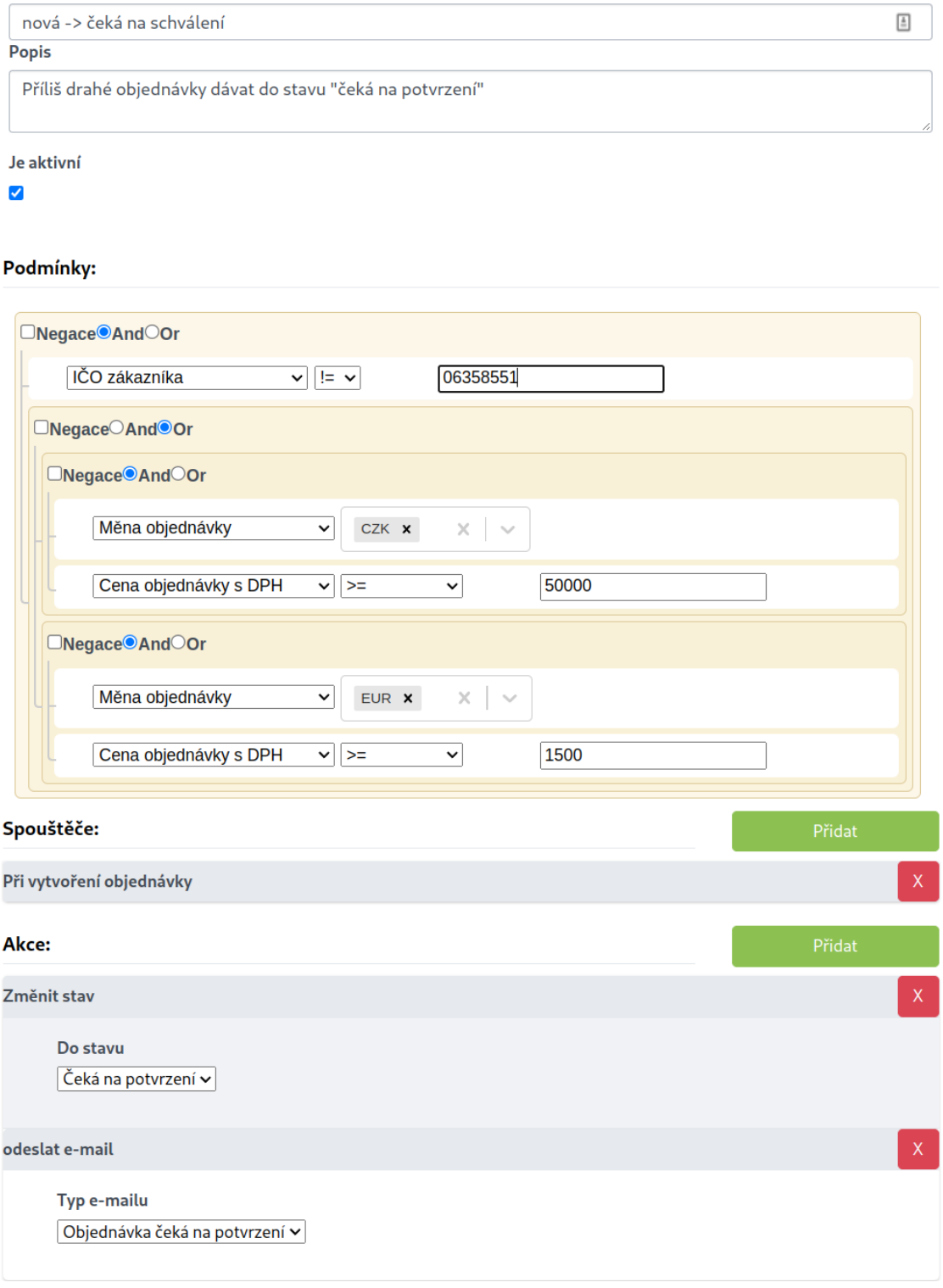
Obrázek 4.5 Záznam akcí – ukázka 1

4.3.2 Ukázka 2

Druhá ukázka obsahuje v bloku podmínek lehce složitější výraz, než tomu bylo v první ukázce. Zde je vidět už zanořování podmínek a využití různých operátorů mezi nimi. Pro jednu objednávku se při splnění podmínek provádí dvě akce.

Konfigurace slouží pro oddělení příliš drahých objednávek do vlastního stavu „Objednávka čeká na potvrzení“. Tento stav může být pro administrátory být použit například tím způsobem, že takovéto objednávky musí projít manuální kontrolou před tím, než se skutečně schválí.

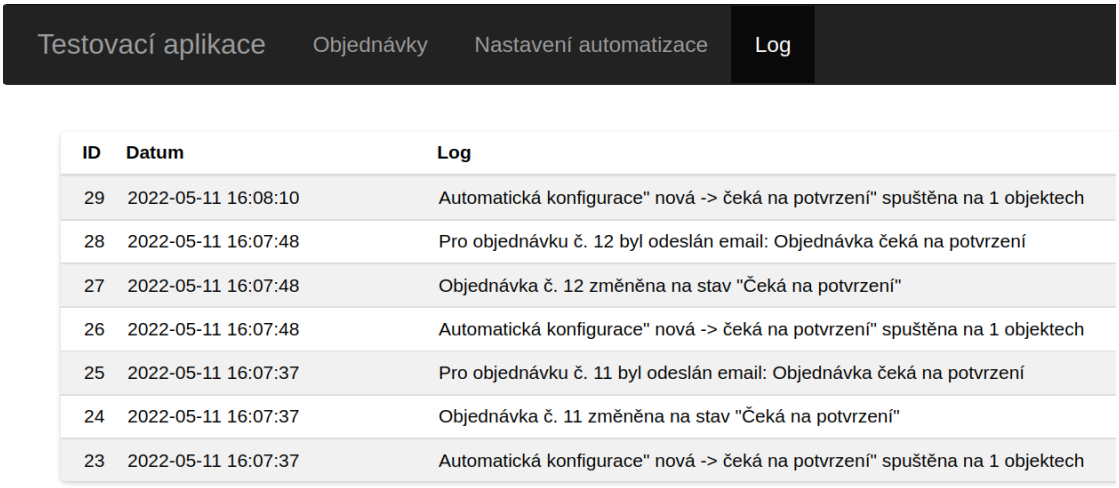
Vybrány jsou takové objednávky, které jsou dražší než 50 000 českých korun nebo dražší než 1 500 euro. Pokud ale zákazník má konkrétní IČO, tak se na něj tyto pravidla nevztahují. Akce které se provedou při splnění podmínek jsou – změna stavu a odeslání informačního e-mailu zákazníkovi. Spouštěč je nastavený na „Při vytvoření objednávky“, ten reaguje na událost přijetí objednávky od uživatele. V minulém příkladu byl použit periodický spouštěč z důvodu, že se muselo čekat na platbu a pravidelně kontrolovat, zda už nepřišla. V tomto příkladu ale všechny potřebné informace objednávky jsou neměnné, tudíž se tato konfigurace automatizace může provést hned při přijetí objednávky. Také, jedna z akcí je odesílání e-mailu zákazníkovi, což je věc která je vhodná doručit zákazníkovi co nejdříve. Proto je v tomto případu periodický spouštěč dokonce nevhodný.



Obrázek 4.6 Konfigurace automatizace ukázky 2

Po vytvoření této konfigurace jsem vytvořil 3 nové objednávky. První objednávka číslem 11 má cenu 55 000Kč a nemá vyplněné IČO. Druhá objednávka s číslem 12 má cenu 2 000 euro a také nemá vyplněné IČO. Třetí objednávka s číslem 13 má cenu 60 000Kč, ale má vyplněné IČO s hodnotou 06358551.

Jak je z následujícího záznamu chování 4.7 vidět, pro první dvě objednávky se akce provedly správně. Pro třetí objednávku se akce neprovedly, protože objednávka obsahovala právě IČO z podmínky.



Obrázek 4.7 Záznam akcí – ukázka 2

Kapitola 5

Závěr

Cílem této práce bylo prozkoumat existující řešení této problematiky a zhodnotit jejich výhody a nevýhody. Vytvořit návrh aplikace s přívětivým grafickým rozhraním, implementovat navržené grafické rozhraní a implementovat serverovou část, která má na starost vyhodnocování a aplikování konfigurací automatizace vytvořených uživateli.

V rešeršní části jsem vybral dvě známé existující řešení, které jsem prošel a zhodnotil jejich výhody a nevýhody. Navrhl jsem dva koncepty, podle kterých bylo řešení možné realizovat. Z navržených konceptů jsem vybral ten, který je uživatelsky přívětivější a zároveň jednodušší na implementaci – tj. koncept rozdělení elementů konfigurátoru na bloky spouštěčů, podmínek a akcí. Klientskou část a její grafické rozhraní jsem implementoval v JS knihovně React. Serverovou část jsem implementoval ve frameworku Symfony jako balíček. Kód této části obsahuje rozhraní pro přidávání nových elementů konfiguračního nástroje, tudíž v budoucnu lze jednoduše rozšiřovat možnosti, které pro automatizaci objednávek v nástroji konfigurovat. Serverová část také obsahuje automatizované testy, které slouží k snížení pravděpodobnosti vzniku chyb při budoucím vývoji.

Náplň této práce pokrývá všechny požadavky z definovaných cílů a ze zadání práce. Vznikl nástroj na automatizaci objednávek, který má grafické rozhraní pro tvorbu konfigurací automatizace a má serverovou část, která dokáže vytvořené konfigurace uchovávat a aplikovat.

Bibliografie

1. *Automatizace objednávek | Brani.cz* [online]. Brani.cz, [2020] [cit. 2022-04-24]. Dostupné z: https://www.brani.cz/nastroj/automatizace-objednavek.
2. *Podmínky spolupráce | Brani.cz* [online]. Brani.cz, [2020] [cit. 2022-04-24]. Dostupné z: https://www.brani.cz/podminky-spoluprace.
3. *Automatizace objednávek | Brani.cz* [online]. Brani.cz, [2020] [cit. 2022-04-24]. Dostupné z: https://www.brani.cz/cenik.
4. *What is Shopify, and How Does it Work? - Small Business Trends* [online]. Small Business Trends LLC, c2003 - 2022 [cit. 2022-05-12]. Dostupné z: https://smallbiztrends.com/2021/08/what-is-shopify.html.
5. *Ecommerce automation: A revolutionary way to scale and customize your business with Shopify Plus* [online]. Shopify Inc., [2006] [cit. 2022-05-12]. Dostupné z: https://www.shopify.com/plus/solutions/ecommerce-automation.
6. *Shopify Plus pricing* [online]. Shopify Inc., [2006] [cit. 2022-05-12]. Dostupné z: https://www.shopify.com/plus/pricing.
7. *PHP - Wikipedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-05-12]. Dostupné z: https://en.wikipedia.org/wiki/PHP.
8. *Symfony, High Performance PHP Framework for Web Development* [online]. Symfony SAS., c2006 - 2017 [cit. 2022-05-12]. Dostupné z: http:%20//symfony.com/.
9. *Symfony. PHP-based web development framework* [online]. ntchosting.com, c2002 - 2022 [cit. 2022-05-12]. Dostupné z: https://www.ntchosting.com/encyclopedia/frameworks/symfony/.
10. SOJKA, Petr; RŮŽIČKA, Michal; LUPTÁK, Dávid; NOVOTNÝ, Vít. *PB029: Jazyky CSS a JavaScript* [online]. Https://www.fi.muni.cz/, c2005–2021 [cit. 2022-05-12]. Dostupné z: https://www.fi.muni.cz/lemma/PB029/practices/css-a-javascript/.
11. *TypeScript: Documentation - TypeScript for JavaScript Programmers* [online]. Microsoft, c2012-2022 [cit. 2022-05-12]. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.
12. *TypeScript - Wikipedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-05-12]. Dostupné z: https://en.wikipedia.org/wiki/TypeScript.
13. *1. What Is React? - What React Is and Why It Matters [Book]* [online]. O'Reilly Media, Inc., c2022 [cit. 2022-05-12]. Dostupné z: https://www.oreilly.com/library/view/what-react-is/9781491996744/ch01.html.
14. *UML Use Case Diagrams* [online]. Uml-diagrams.org, c2009-2020 [cit. 2022-05-04]. Dostupné z: https://www.uml-diagrams.org/use-case-diagrams.html.

15. *Component Diagrams - See Examples, Learn What They Are* [online]. SmartDraw, LLC, c1994-2022 [cit. 2022-05-04]. Dostupné z: https://www.smartdraw.com/component-diagram/.

16. *UML: component diagram - diagram komponent* [online]. Mpavus.wz.cz, [2006] [cit. 2021-05-13]. Dostupné z: http://www.mpavus.wz.cz/uml/uml-s-component-3-3-2.php.

17. *JQuery* [online]. OpenJS Foundation, c2022 [cit. 2022-05-12]. Dostupné z: https://jquery.com/.

18. *Ukrbublik/react-awesome-query-builder: User-friendly query builder for React* [online]. 2020 [cit. 2022-05-12]. Dostupné z: https://github.com/ukrbublik/react-awesome-query-builder.

19. *React Components* [online]. Refsnes Data, c1999-2022 [cit. 2022-05-06]. Dostupné z: https://www.w3schools.com/react/react_components.asp.

20. *Context – React* [online]. Meta Platforms, c2022 [cit. 2022-05-09]. Dostupné z: https://reactjs.org/docs/context.html%5C#reactcreatecontext.

21. NILS ADERMANN, Jordi Boggiano a. *Introduction - Composer* [online]. Composer, [2013] [cit. 2022-05-08]. Dostupné z: https://getcomposer.org/doc/00-intro.md.

22. *Co je to Parsování? - IT Slovník* [online]. IT-Slovnik.cz team, c2008-2022 [cit. 2022-05-06]. Dostupné z: https://it-slovnik.cz/pojem/parsovani.

23. KULHAN, Jakub. *Jakubkulhan/js2php: Javascript parser, compiler and interpreter written in PHP* [online]. Jakub Kulhan [cit. 2022-05-06]. Dostupné z: https://github.com/jakubkulhan/js2php.

24. FORD, Bryan. Parsing expression grammars: a recognition-based syntactic foundation. In: *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 2004.

25. *Testing (Symfony 4.4 Docs)* [online]. Symfony SAS [cit. 2022-05-12]. Dostupné z: https://symfony.com/doc/4.4/testing.html.

26. HAMILTON, Thomas. *Integration Testing: What is, Types, Top Down & Bottom Up Example* [online]. Guru99, c2022 [cit. 2022-05-12]. Dostupné z: https://www.guru99.com/integration-testing.html.

Obsah přiloženého média

- `readme.txt` ..... instalační příručka
- `automatizace_procesu_objednavek.pdf` ..... Dokumentace
- `app`
  - `react` ..... adresář pro klientskou část implementace
    - `src` ..... adresář pro zdrojové kódy komponent
    - `package.json` ..... soubor popisující závislosti aplikace
    - …
  - `symfony` ..... adresář pro serverovou část implementace
    - `src` ..... adresář pro zdrojové kódy
    - `composer.json` ..... soubor popisující závislosti aplikace
    - `initData.sh` ..... skript pro inicializaci databáze
    - …
  - `docker-compose.yml`
- `latex` ..... adresář pro zdrojové kódy dokoumentace