Středoškolská technika 2025

Setkání a prezentace prací středoškolských studentů na ČVUT

VÝVOJ KOMPLEXNÍHO OBJEDNÁVKOVÉHO SOFTWARU

Pavel Viktora, Tomáš Vacík, Jan Durda

Gymnázium, Praha 9, Chodovická 2250
Chodovická 2250/36, 193 00, Praha 9 - Horní Počernice

Poděkování

V úvodu bychom rádi vyjádřili svou upřímnou vděčnost všem, kteří nám pomohli při zpracování této práce.

Děkujeme Richardu Vackovi za nápad, podporu, zadání pro program a příspěvek na koupi tiskárny.

Naše hluboká vděčnost patří také paní Ing. Daně Siegelové za podporu a umožnění práce při hodinách informatiky.

Velké díky patří rovněž našemu konzultantovi Ing. Davidu Tesaříkovi za vedení a cenné rady.

Anotace

V této práci se věnujeme návrhu a vývoji objednávkového softwaru, který má za cíl zefektivnit proces správy objednávek. Práce se soustředí na analýzu uživatelských požadavků, návrh architektury systému a implementaci softwarového řešení. Použité metody zahrnují systematický přístup k modelování dat a programování výsledného produktu. Získané poznatky přispívají k lepšímu pochopení problematiky vývoje softwaru a mohou sloužit jako podklad pro budoucí projekty v oblasti informačních technologií.

Klíčová slova

Programování, webová stránka, full-stack

Annotation

In this thesis, we are designing and developing ordering software to streamline the order management process. The work focuses on user requirements analysis, system architecture design and implementation of the software solution. The methods used include a systematic approach to data modelling and programming of the final product. The knowledge gained contributes to a better understanding of software development issues and can serve as a basis for future projects in the field of information technology.

Keywords

Coding, website, full stack

Obsah

1 Úvod

V době, kdy informační technologie hrají klíčovou roli v každodenním životě, se stále častěji setkáváme s rostoucí potřebou vlastních softwarových řešení. Profesionální softwarové balíky často vyžadují vysoké náklady, a proto vzniká nutnost vývoje cenově dostupných, přizpůsobitelných a intuitivních nástrojů. Tato práce vznikla z praktické potřeby – byl vyžadován software vhodný pro organizaci školní párty, který by umožnil snadné nastavení a ovládání dle aktuálních potřeb organizátorů.

Hlavním cílem práce je vytvořit funkční, přehledný a uživatelsky přívětivý software, který bude snadno ovladatelný, intuitivní a flexibilní. K dosažení tohoto cíle byla nejprve provedena podrobná rešerše zaměřená na tvorbu full-stack softwaru, jejímž účelem bylo získat přehled o moderních postupech, technologiích a nástrojích využívaných při vývoji komplexních softwarových řešení. Na základě těchto informací byl stanoven postup vývoje, který zahrnoval analýzu potřeb, návrh architektury systému, samotnou implementaci a následné testování výsledného produktu.

Struktura práce je rozdělena do tří hlavních částí. První část představuje základy full-stack vývoje a poskytuje čtenáři teoretický kontext potřebný k pochopení problematiky. Druhá část se věnuje detailnímu popisu použitých technologií a metod, zatímco třetí část obsahuje komplexní dokumentaci softwaru včetně popisu implementačního procesu a ukázek zdrojového kódu. Tento strukturovaný přístup zajišťuje, že práce nejen demonstruje realizaci praktického řešení, ale také přispívá k hlubšímu porozumění procesu vývoje komplexního softwaru.

2 FULL-STACK VÝVOJ

Pojem full-stack je celkem obecný pojem, který se používá v oblasti vývoje softwaru a značí schopnost pracovat ve všech vrstvách aplikace – frontendu a backendu, přičemž těmto pojmům se budeme více věnovat v další části. Full-stack vývojáři mají velmi široké znalosti v obou těchto vrstvách, což jim umožňuje spravovat kompletně celé aplikace od uživatelského rozhraní až po databázi a serverovou infrastrukturu.[1]

Co ale vlastně full-stack v programování znamená a jaký je jeho význam? Tradičnější rozdělení vývojářů bývalo na dvě hlavní skupiny – frontend vývojáře a backend vývojáře. Frontend vývojáři se zaměřují hlavně na rozhraní pro uživatele a vizuální stránku aplikací, kdežto backend vývojáři mají na práci serverovou logiku, databáze a API.[2]

Full-stack vývojář ale kombinuje obě tyto role najednou a zvládá tak celý proces vývoje. Tento způsob umožňuje velmi efektivní spolupráci mezi jednotlivými prvky aplikace a jednoznačně snižuje závislost na více lidech. Full-stack vývojář se samozřejmě nemůže spokojit pouze se zvládáním frontendu a backendu, ale musí disponovat komplexními znalostmi v oblasti softwaru, zabezpečení, správy serverů a optimalizace výkonu.[3]

Dalo by se tedy říct, že full-stack vývojáři dokážou pracovat efektivněji a jednodušeji. Je pravda, že tento způsob je rozhodně flexibilnější, efektivnější, nachází se zde možnost čistě samostatné práce a vývojář má tedy příležitost vidět celou aplikaci jako celek, což se může později ukázat, jako velmi důležité při řešení problémů. Na druhou stranu vývojáři, kteří se zabývají pouze užší částí programování jsou v daných oblastech specializovanější, tudíž u větších projektů se setkáme spíše se specializovanými vývojáři.[4]

2.1 Jak funguje full-stack aplikace?

Jedná se o software, který kombinuje frontend a backend, aby se vytvořil plně funkční software, schopný běžet na webu, v mobilu, nebo jako desktopová aplikace. Fungování celé aplikace začíná tím, že klient si vyžádá frontendovou stránku od serveru. Frontend může být vytvořen kombinací mnoha technologií – HTML, CSS nebo Javascript. Když jdeme poté například na stránku Google a zadáme adresu do prohlížeče, začíná interakce s backendem, který následně

[1] *Full Stack Development Explained*. Online. MongoDB. Dostupné z: https://www.mongodb.com/resources/basics/full-stack-development. [cit. 2025-02-13].
[2] SIMMONS, Liz. *Front-End vs. Back-End: What's the Difference?* Online. ComputerScience.org. August 19, 2024. Dostupné z: https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/. [cit. 2025-02-13].
[3] *Full Stack Development Explained*. Online. MongoDB. Dostupné z: https://www.mongodb.com/resources/basics/full-stack-development. [cit. 2025-02-13].
[4] SZABO, Pavel. *Full Stack Developer*. Online. Pavel Szabo. Dostupné z: https://www.pavelszabo.cz/co-je-full-stack-developer/. [cit. 2025-02-13].

vyšle stránku *google.com*, což je viditelný frontend. Pro fungování a komunikaci mezi frontendem a backendem se využívá rozhraní API.[5]

API znamená Application Programming Interface a jedná se o rozhraní, které zajišťuje komunikaci mezi jednotlivými softwarovými aplikacemi. API definuje jednotlivá pravidla a způsoby, jak mezi sebou aplikace mohou vyměňovat data, spolupracovat nebo celkově provádět jakékoli operace bez nutnosti toho, aby zasáhl uživatel.[6]

V ten moment, kdy backend přijme tento požadavek, ho zpracuje a pokud jsou zapotřebí data uložená v databázi, provede na ni dotaz. Znovu je zde velké množství jazyků, ve kterých může být backend napsaný. Mezi nejznámější patří například Python, Java, PHP nebo C#. Právě backend je tedy část zodpovědná za logiku aplikace. Data se nejčastěji nacházejí v databázi a v okamžiku zisku odpovědí odesílá backend tuto odpověď zpět na frontend, kde už dochází k zobrazení uživateli.

Aby se celý tento proces mohl odehrát a aplikace mohla fungovat online, musí být hostována na serveru. Full-stack vývoj zahrnuje naprosto všechno od tvorby vizuální části aplikace přes správu dát až po nasazení aplikace do produkce.[7]

Proces načtení a fungování webové stránky zahrnuje několik kroků, při kterých úzce spolupracují prohlížeč, server a databáze, aby došlo k zobrazení požadovaného obsahu uživateli. Jako první uživatel zadá URL do prohlížeče, kdy napíše webovou adresu do adresního řádku v prohlížeči a stiskne ENTER. Prohlížeč ověří, zdali tato stránka není v mezipaměti, pokud ano, otevře ji ihned, jinak zahájí proces získání stránky ze serveru. Ihned potom pošle prohlížeč HTTP GET požadavek na server, kdy daný požadavek obsahuje tyto informace: URL adresu stránky, metodu požadavku (GET), informaci o prohlížeči (hlavička User-Agent) a cookies, pokud jsou k dispozici. Požadavek pak může vypadat jako na obrázku 1.[8]

```http
GET / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
```

Obrázek 1: Prvotní požadavek webového prohlížeče

[5] *What is Full Stack Development ?* Online. Geeks For Geeks. 07 Aug, 2024. Dostupné z: https://www.geeksforgeeks.org/what-is-full-stack-development/. [cit. 2025-02-13].
[6] *What is an API (Application Programming Interface)?* Online. AWS Amazon Q. Dostupné z: https://aws.amazon.com/what-is/api/. [cit. 2025-02-13].
[7] *What is Full Stack Development ?* Online. Geeks For Geeks. 07 Aug, 2024. Dostupné z: https://www.geeksforgeeks.org/what-is-full-stack-development/. [cit. 2025-02-13].
[8] , RedGrittyBrick. *Journey of a Web Request*. Online. StackExchange. 2013. Dostupné z: https://superuser.com/questions/528001/journey-of-a-web-request. [cit. 2025-02-13].

Před odesláním požadavku prohlížeč zjistí IP adresu serveru za pomoci DNS – Domain Name systému.[9] Při dalším kroku webový server odpovídá frontendovou stránkou tímto způsobem: server přijme GET požadavek a zjistí, jaký obsah má poslat – statickou HTML stránku nebo dynamický obsah z backendu. Následně vrací odpověď, která obsahuje HTML kód. V okamžiku, kdy server obdrží HTML kód, zažádá o CSS soubory a Javascript soubory. Existuje situace, při které dochází k odeslání CSS a Javascript společně s HTML, ale to pouze tehdy, pokud jsou tyto přímo zahrnuté v HTML souboru. Taková odpověď pak může vypadat jako na obrázku 2.[10] Poté ještě následuje HTML kód stránky.

Obrázek 2: Odpověď serveru na prvotní požadavek

Následně prohlížeč vykreslí stránku – tzv. „rendering“. Tento proces probíhá v několika krocích. Jako první takzvané „parsování“ HTML, kdy prohlížeč přečte HTML a vytvoří DOM – Document Object Model. Následně dojde ke stažení, což ale probíhá jako další GET požadavek na server a aplikace CSS, kdy stránka získává svůj vzhled, potom stahuje a spouští Javascript, který umožňuje dynamickou interakci, a nakonec se stránka zobrazí uživateli.[11]

Dále dochází k dynamickému načítání dat, kdy Javascript začne komunikovat se serverem bez nutnosti toho, aby se celá stránka načítala znova, a to se děje přes Ajax, Fetch API nebo WebSockety. AJAX načítá data na pozadí, Fetch API nebo XHR (XMLHttpRequest) umožňuje získání a odesílání dat a WebSockety umožňují komunikaci v reálném čase, která je obousměrná. WebSockety fungují tak, že klient odesílá WebSocket „handshake“ (uvítací zprávu) přes HTTP, kdy následně server odpovídá a spojení se přepne na WebSocket. Od tohoto momentu mohou obě strany odesílat zprávy bez nutnosti neustálé aktualizace stránky.[12] WebSockety se nejčastěji používají v aplikacích určených ke komunikaci – např. WhatsApp Web nebo online videohry. Největší výhodou oproti běžnému HTTP je vysoká rychlost a nízká latence, neboť nenavazuje pokaždé nové spojení.

Příklad takové komunikace může být, když kód na obrázku 3 požádá server o seznam produktů, který pak posílá do konzole.[13]

---

[9] *What is DNS? | How DNS works*. Online. CloudFlare. Dostupné z: https://www.cloudflare.com/learning/dns/what-is-dns/. [cit. 2025-02-13].
[10] , RedGrittyBrick. *Journey of a Web Request*. Online. StackExchange. 2013. Dostupné z: https://superuser.com/questions/528001/journey-of-a-web-request. [cit. 2025-02-13].
[11] , RedGrittyBrick. *Journey of a Web Request*. Online. StackExchange. 2013. Dostupné z: https://superuser.com/questions/528001/journey-of-a-web-request. [cit. 2025-02-13].
[12] *WebSocket*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 9. 11. 2022. Dostupné z: https://cs.wikipedia.org/wiki/WebSocket. [cit. 2025-02-13].
[13] E RABBI, Fazal. *How the Web Works: A Deep Dive into What Happens When You Type a URL, DNS Lookup, and HTTP*. Online. Medium. 2024. Dostupné z: https://fazalerabbi.medium.com/how-the-web-works-a-deep-dive-into-what-happens-when-you-type-a-url-dns-lookup-and-http-f05af4be36d8. [cit. 2025-02-13].

Obrázek 3: Kód pro zažádání dat od serveru

V dalším kroku backend server zpracovává požadavek na data a k tomu dochází, pokud stránka potřebuje další data a udělá to tímto způsobem: přijme požadavek, data získá z databáze, data vrátí ve formátu JSON nebo XML a odpověď serveru vypadá například jako na obrázku 4.[14]

Obrázek 4: Příklad dat z odpovědi od serveru

Jako poslední prohlížeč aktualizuje stránku s novými daty. V moment, kdy Javascript získá odpověď ze serveru, aktualizuje obsah stránky a není nutné ji znovu načítat.

2.2 Frontend

Frontend je část aplikace na webu nebo webové stránky, která sama zodpovídá za to, co vidí uživatel a s čím může nějakým způsobem interagovat. Stará se o to, jak veškeré stránky vypadají a jak se spojují aplikace s uživateli. Frontend je tedy část, na kterou se vždy uživatel pohybuje, komunikuje s ním a zároveň mu prezentuje obsah zpracovaný ze serveru. Vývoj frontendu se zabývá tvorbou vizuálních prvků, do této skupiny spadá jednoduše vše, co může uživatel vidět – texty, obrázky, lišty, formuláře atd… Cílem frontendu je, aby byla stránka jednoduše pěkná, přehledná a hlavně funkční.[15]

Technologií využívaných pro vývoj frontendu je mnoho a několik z nich je zcela klíčových – HTML, CSS, Javascript, frameworky (např. React, Vue, Svelte apod.) a knihovny nebo tzv. „responsive design“.

HTML (Hyper Text Markup Language) je základní stavební jednotka všech webových stránek, která umožňuje vytvářet strukturu stránky, hlavně nadpisy, odstavce, obrázky, nebo jakékoli texty.

[14] , RedGrittyBrick. *Journey of a Web Request*. Online. StackExchange. 2013. Dostupné z: https://superuser.com/questions/528001/journey-of-a-web-request. [cit. 2025-02-13].
[15] BENNET, Lucas. *Kdo je front-end vývojář? Kompletní průvodce*. Online. GURU99. Srpen 13, 2024. Dostupné z: https://www.guru99.com/cs/front-end-developer.html. [cit. 2025-02-13].

CSS (Cascading Style Sheets) je jazyk pro editaci a vytváření vzhledu webových stránek, pomocí CSS se definují barvy, fonty a celkově všechny vizuální aspekty prvků HTML. CSS umožňuje oddělit obsah od jeho vzhledu.

Javascript slouží pro interaktivitu na webových stránkách – dynamické prvky, načítání nových dat bez znovunačtení stránky, nebo jakékoli dynamické funkce.

Frameworky a knihovny slouží k usnadnění vývoje a zvýšení efektivity a responsive design znamená, že webový stránka by měla být optimalizovaná pro různé velikosti obrazovek, kdy díky CSS technologiím lze upravit vzhled pro mobily, tablety, nebo desktopy.

Jakou roli zde plní vývojář? Tato osoba je plně odpovědná, jak již bylo zmíněno, za vše, co vidí uživatel. Hlavní náplní takové osoby je tedy především kódování HTML, CSS a Javascriptu, zajištění kompatibility vzhledem k rozdílnosti prohlížečů, zajištění pro všechny uživatele, aby byl web přístupný a optimalizaci výkonu.[16]

2.3 Backend

Backend se dá považovat za mozek celé aplikace. Zpracovává požadavky z frontendu, provádí výpočty, spravuje autentizaci uživatelů a komunikuje s databází.

Technologií, které se užívají v backendu, je několik. Jako hlavní jsou programovací jazyky, které existuje obrovské množství, dále webové frameworky a API, které umožňují komunikaci mezi frontendem a backendem. Backend často pracuje s mnoha databázemi, kde jsou uložena data z aplikace.[17]

2.3.1 HTTP

Pro komunikaci s WWW servery se využívá internetový protokol HTTP – Hyper Text Transfer Protocol. Slouží pro přenos hypertextových dokumentů ve formátu HTML, XML, i jiných typů souborů.[18] Existuje řada jednotlivých HTTP metod, které určují typ operace, kterou může uživatel na serveru provést. Jejich seznam najdete v tabulce 1.

[16] BENNET, Lucas. *Kdo je front-end vývojář? Kompletní průvodce*. Online. GURU99. Srpen 13, 2024. Dostupné z: https://www.guru99.com/cs/front-end-developer.html. [cit. 2025-02-13].
[17] *What's the Difference Between Frontend and Backend in Application Development?* Online. AWS Amazon Q. Dostupné z: https://aws.amazon.com/compare/the-difference-between-frontend-and-backend/. [cit. 2025-02-13].
[18] *Hypertext Transfer Protocol*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 19.1.2025. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [cit. 2025-02-13].

Metoda	**Význam**
GET	načtení dat ze serveru, nedochází k žádné změně
POST	odeslání dat na server, vznik nového záznamu
PUT	aktualizace nebo tvorba zdroje na serveru
DELETE	odstranění zdroje
PATCH	částečná aktualizace zdroje
HEAD	podobné jako GET, vrací pouze hlavičku, ne celý obsah
OPTIONS	zjištění, které metody server podporuje
CONNECT	tunelové spojení – HTTPS přes proxy
TRACE	test a diagnostika sítě

Tabulka 1: Seznam vybraných HTTP metod

Ohledně HTTP je důležité vysvětlit pojem HTTP hlavička, což patří mezi nedílnou součást komunikace klienta a serveru. Poskytují informace o požadavku nebo odpovědi a umožňují kontrolu ohledně přenosu dat. Typů HTTP hlaviček je hned několik a dají se rozdělit do několika kategorií jako v tabulce 2.

Kategorie	**Popis**
Hlavičky požadavku	požadavek klienta
Hlavičky odpovědi	odpověď ze serveru
Obecné hlavičky	požadavky a odpovědi
Hlavičky entit	popis obsahu přenášených dat
Bezpečnostní hlavičky	omezení rizika útoku

Tabulka 2: Seznam kategorií HTTP hlaviček

Pod jednotlivé kategorie následně vždy spadá velké množství jednotlivých hlaviček. Například u hlaviček požadavku najdeme hlavičky – Host, User-Agent, Accept…[19]

V téhle kategorii jsou tedy hlavičky odesílány klientem při jakémkoli požadavku na server. Opakem je kategorie HTTP odpovědi, kdy dochází k odpovědi serveru, obsahující informace o obsahu a způsobu zpracování. Mezi HTTP odpovědi patří například – Server, Date, Content-Type nebo Location.

2.3.2 Cookies

S cookies se denně setkává každý uživatel internetového světa a skoro vždy pouze klikneme na „souhlasím“, aniž bychom si úplně uvědomovali, co tento krok znamená. Cookies jsou malé textové soubory, které webová stránka ukládá do prohlížeče pro daného uživatele. Jednotlivých využití je mnoho, ale tyto patří mezi ty hlavní – udržování relace, ukládání preferencí, sledování klienta. Cookies pro udržení relace slouží například k přihlášení uživatele, preferenční pro nastavení jazyka a sledovací pro reklamní účely. Jak ale samotné cookies fungují? Jako první požádá klient server o stránku a v okamžik odpovědi serveru, posílá i soubor cookie v hlavičce Set-Cookie. Prohlížeč následně cookie ukládá a při každém další požadavku ji pošle zpět.[20]

2.3.3 Endpointy

Endpointy (koncové body) jsou specifické URL adresy, které umožňují klientovi komunikovat se serverem prostřednictvím API. Zároveň jsou naprosto klíčovou součástí REST API, GraphQL, API, WebSocketů. V REST API každý endpoint odpovídá určitému zdroji (/users, /products), v GraphQL API je nejčastěji pouze jeden endpoint, přes který se mohou posílat odkazy a WebSocket endpointy umožňují právě již zmíněnou obousměrnou komunikaci v reálném čase.

[19] *HTTP headers*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers. [cit. 2025-02-13].
[20] *What are cookies? | Cookies definition*. Online. CloudFlare. Dostupné z: https://www.cloudflare.com/learning/privacy/what-are-cookies/. [cit. 2025-02-13].

3 DESIGN APLIKACÍ

3.1 Klíčové principy

Design aplikací není jen o hezkém vzhledu, ale hlavně o tom, aby aplikace byla intuitivní, rychlá a funkční. Každá aplikace by měla stát na pevných základech: konzistence a efektivní komunikace mezi uživatelem a systémem, příjemný design pro oči. Design by měl být předem naplánován, logicky vystaven a strukturalizován. Stejně tak musí být promyšlen každý detail, aby výsledný produkt byl nejen pěkný na pohled, ale i pohodlný na používání.[21] [22]

3.1.1 Základní pojmy

UI (User Interface) je vizuální vrstva aplikace, kterou vidíte na první pohled. V UI se mluví o tlačítkách, ikonách, formulářích a barevných schématech. Vlastně o všem, co umožňuje, aby uživatel ihned pochopil, co má dělat.[23]

UX (User Experience) se zaměřuje na to, jaký celkový dojem má aplikace na uživatele. Od prvního kontaktu až po poslední kliknutí. UX zajišťuje, že vše poběží hladce a přirozeně, bez zbytečných překážek. [24]

Interaction design řeší, jak se jednotlivé prvky navzájem propojují a jak reagují na uživatelské akce. Každé kliknutí nebo gesto by mělo mít okamžitou a logickou odezvu.

3.1.2 Metody designu

Každá aplikace má jasná pravidla a standardy. Při vytváření aplikací se můžeme opřít o osvědčené postupy, které zaručují kvalitu výsledného řešení.[25]

Design thinking je metoda založená na hlubokém pochopení potřeb uživatelů. V podstatě jde o to, že se začne s definicí problému, poté se přejde k brainstormingu, tvorbě prototypů, testování a následně k dolaďování.[26]

---

[21] *Aplikace je firma ve firmě*. Online. Think Easy. 2024. Dostupné z: https://thinkeasy.cz/aplikace-je-firma-ve-firme/. [cit. 2025-02-13].
[22] KOTT, Petr. *5 principů, jak na tvorbu grafiky u mobilních aplikací*. Online. Peko studio. 07/06/2021. Dostupné z: https://peko-studio.cz/5-principu-jak-na-tvorbu-grafiky-u-mobilnich-aplikaci/. [cit. 2025-02-13].
[23] *Co je UX/UI design (webů a aplikací)?* Online. UX/UI design. Dostupné z: https://www.cojeuxui.cz. [cit. 2025-02-13].
[24] *Co je UX/UI design (webů a aplikací)?* Online. UX/UI design. Dostupné z: https://www.cojeuxui.cz. [cit. 2025-02-13].
[25] *Zásady při stavbě aplikace*. Online. Pixelmate. Dostupné z: https://pixelmate.cz/blog/zasady-pri-stavbe-aplikace. [cit. 2025-02-13].
[26] *Co je Design Thinking?* Online. Skillmea. 2022. Dostupné z: https://skillmea.cz/blog/co-je-design-thinking. [cit. 2025-02-13].

V agilním přístupu se klade důraz na flexibilitu a rychlé reakce na zpětnou vazbu. Vytváří se velké množství prototypů, které se podrobují pravidelné revizi. Zároveň je důležitá úzká spolupráce mezi designéry a vývojáři. Každá malá změna zvyšuje funkčnost a přehlednost finálního produktu.[27]

Atomic design rozkládá celý design na základní, opakovaně použitelné, komponenty, které lze sestavit dohromady jako stavebnice. Vše má své přesně definované místo a funkci.[28]

Lean UX se soustředí na odstranění zbytečné byrokracie a rychlé testování nápadů prostřednictvím prototypů. Je to jako rychle napsat kus kódu, hned ho otestovat a v případě potřeby upravit. Lze aplikovat jednoduše u menších projektů, ovšem u větších se může rychle zvrtnout.[29]

3.2 HTML

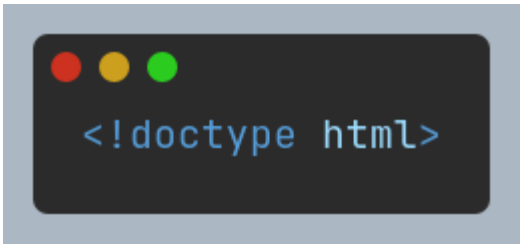
HTML, známé také jako Hyper Text Markup Language, je značkovací jazyk, který se v průběhu let stal základním stavebním kamenem každé webové stránky. Hlavním úkolem tohoto jazyka je strukturalizace konkrétní webové stránky pomocí tagů, což se děje pomocí jednotlivých prvků jako nadpisy (elementy h1, h6…), odstavce (element p), obrázky (element img). Obsah tohoto jazyka je interpretován webovými prohlížeči do podoby webové stránky, se kterou se uživatelé internetu setkávají dennodenně. HTML je jeden z nejzákladnějších a nejuniverzálnějších jazyků, za kterým stojí loajální a široká komunita vývojářů zajišťující neustálý přísun zdrojů pro studium a další rozvoj tohoto jazyka. Jedna z dalších výhod HTML souvisí s jeho kompatibilitou s ostatními jazyky jako je například CSS (Cascading Style Sheets), který se stará o vizuální úpravu stránek. Dále je HTML vysoce kompatibilní s Javascriptem, který se stará o funkci stránek.[30] [31]

3.2.1 Hlavička dokumentu

Ze začátku se musí deklarovat typ dokumentu neboli DTD. Lze to také brát jako směrnici, která webovému prohlížeči sděluje, že HTML je jazyk, ve kterém bude stránka napsána. To pomáhá webovým prohlížečům rychleji interpretovat kód.[32] DTD můžete vidět na obrázku 5.

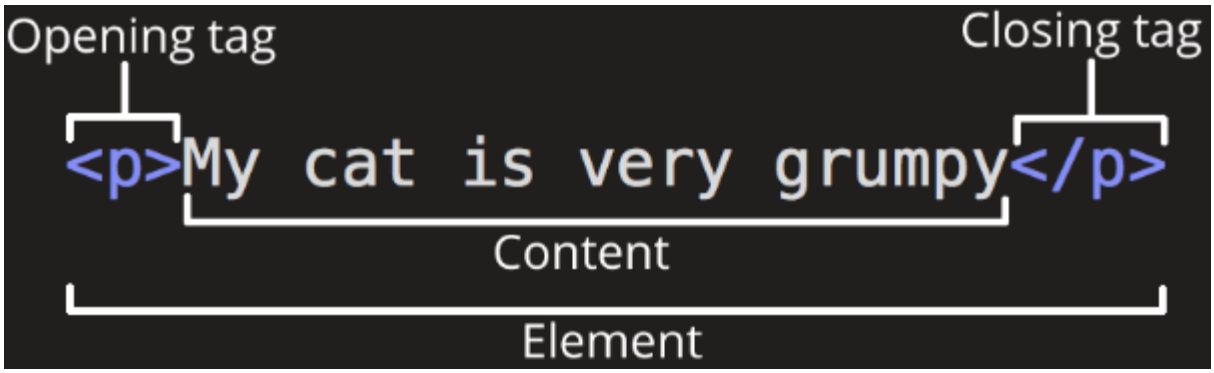
---

[27] FARD, Adam. *What is Agile Design Methodology and how to apply it?* Online. Adamfard. Dostupné z: https://adamfard.com/blog/agile-design. [cit. 2025-02-13].
[28] FROST, Brad. *Atomic Design*. Online. Bradfrost. Dostupné z: https://bradfrost.com/blog/post/atomic-web-design/. [cit. 2025-02-13].
[29] *Lean UX*. Online. SAFe STUDIO. Dostupné z: https://framework.scaledagile.com/lean-ux. [cit. 2025-02-13].
[30] *Co je Design Thinking?* Online. Skillmea. 2022. Dostupné z: https://skillmea.cz/blog/co-je-design-thinking. [cit. 2025-02-13].
[31] *Hypertext Markup Language*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Markup_Language. [cit. 2025-02-13].
[32] *Hlavička HTML dokumentu - Český HTML 5 manuál*. Online. Itnetwork.cz. Dostupné z: https://www.itnetwork.cz/html-css/html5/html-manual/html-css-html-manual-struktura/html-hlavicka-head-cesky-manual. [cit. 2025-02-13].



Obrázek 5: DTD pro HTML dokument

Každé jednotlivé hodnotě uzavřené pomocí tagů se říká element. Jeho struktura je zobrazena na obrázku 6.

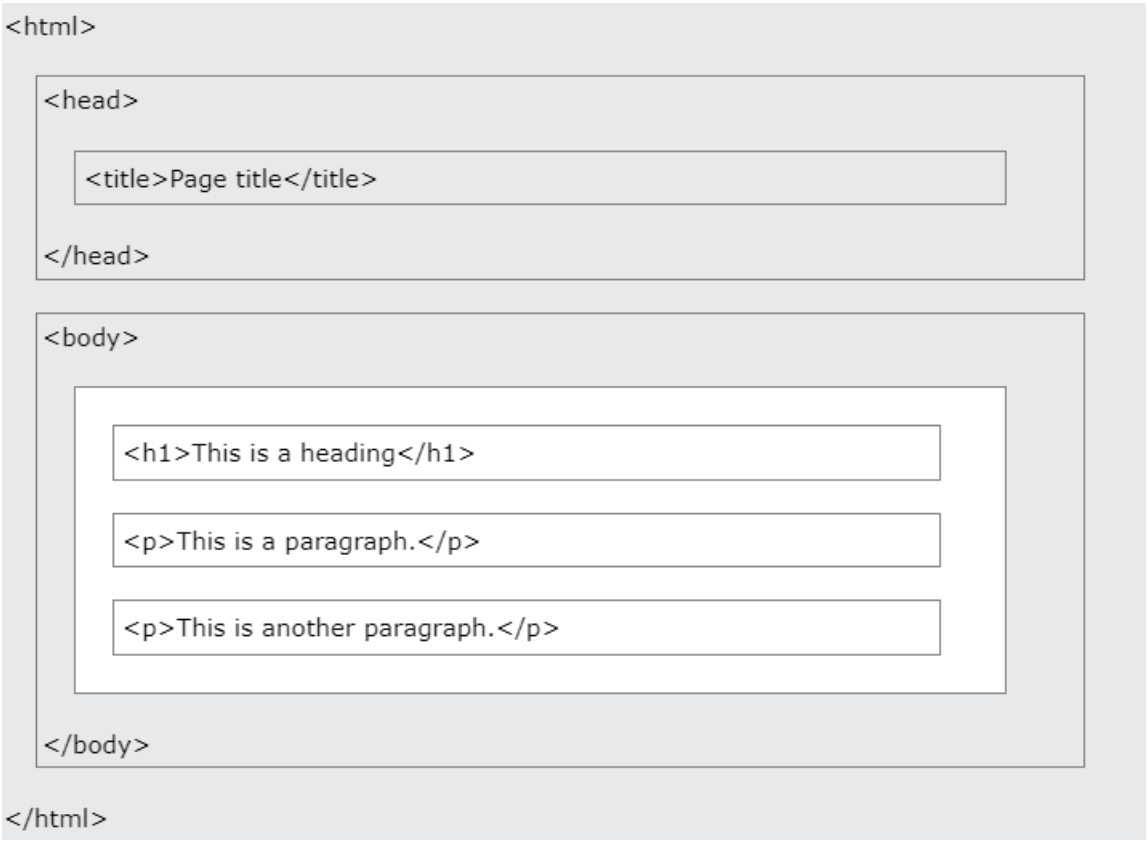


Obrázek 6: Struktura HTML elementu

Opening tag neboli úvodní značka je v tomto případě <p> a je ohraničená v znamínkách <>. Označuje místo, ve kterém prvek začíná. Zároveň se do opening tagu také přidávají atributy, které specifikují různé funkce tagů. *Content* neboli obsah, zobrazuje data, se kterými se dále pracuje. V tomto případě se tento text vypíše do odstavce, jelikož tag <p> označuje odstavec. *Closing tag* neboli uzavírací značka poté označuje konec tohoto elementu. Před název tagu by se mělo psát lomítko.[33]

Každý HTML dokument má podobnou strukturu, která se skládá ze začátku a konce HTML kódu, což je označeno pomocí <html>, </html>. Poté se zde nachází head element neboli hlavička dokumentu. Tato část se při načtení stránky nezobrazí ve webovém prohlížeči a je označena <head>, </head>. Obsahuje například informace o metadatech, nadpis stránky a odkazy na CSS knihovny.[34] Tuto strukturu můžete vidět na obrázku 7.

[33] *Basic HTML syntax*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Structuring_content/Basic_HTML_syntax. [cit. 2025-02-13].
[34] *What's in the head? Webpage metadata*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Structuring_content/Webpage_metadata. [cit. 2025-02-13].



Obrázek 7: Struktura HTML dokumentu[35]

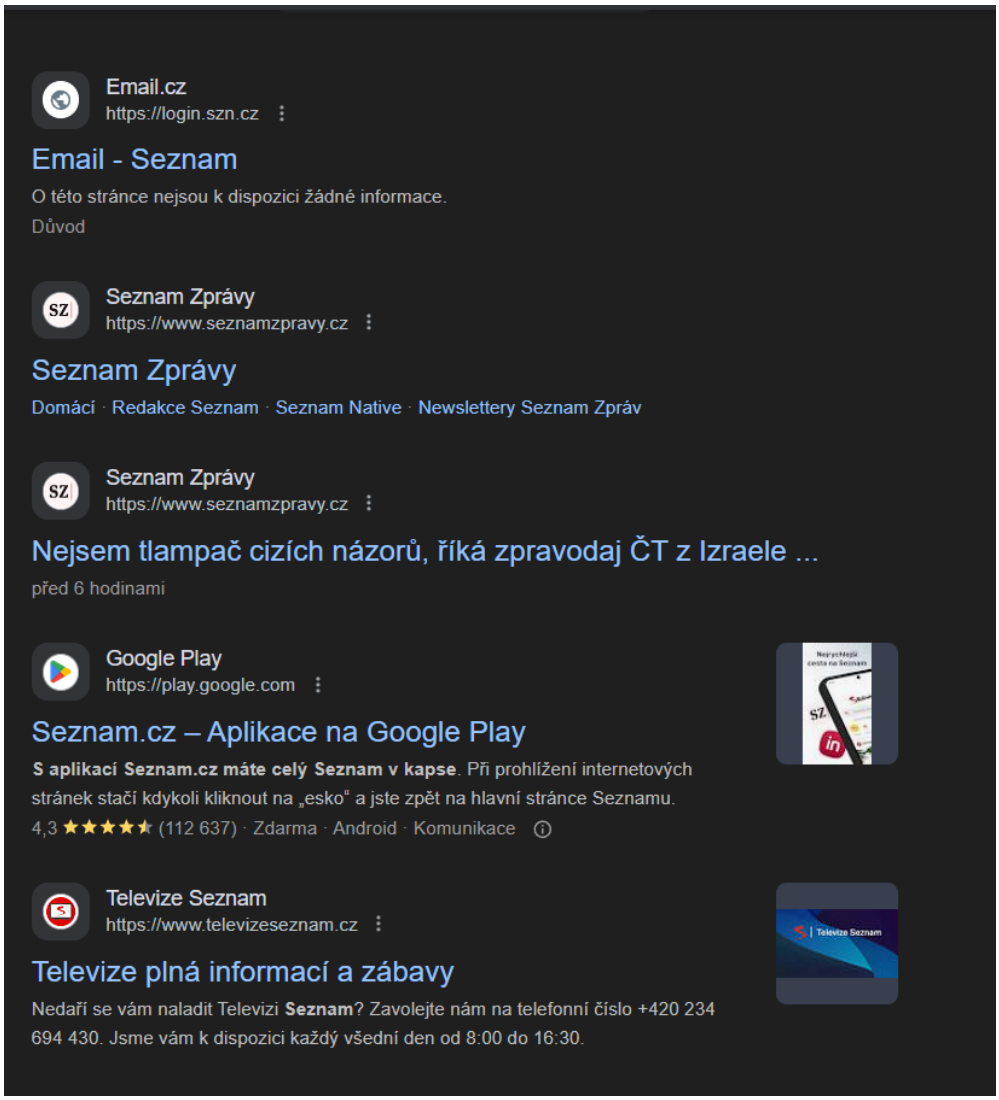
V hlavičce se také určuje element <meta>, který se stará o kódování znaků dokumentu a určuje, kterou znakovou sadu bude dokument využívat. UTF-8 je základní univerzální znaková sada zahrnující skoro všechny známé a zobrazitelné znaky.[36] Na obrázku 8 můžete vidět určení znakové sady.

Obrázek 8: Deklarace znakové sady UTF-8 pomocí elementu meta

Dále se v hlavičce určuje, co lze vidět ve webovém prohlížeči při vyhledávání obsahu přes prohlížeč. O název stránky se stará element <title>, který dokument pojmenuje. Toto jméno je

[35] SMITHA, P. S. *HTML*. Online. Slideshare.net. 2022. Dostupné z: https://www.slideshare.net/slideshow/html-1-251294332/251294332#1. [cit. 2025-02-14].
[36] *Hlavička HTML dokumentu - Český HTML 5 manuál*. Online. Itnetwork.cz. Dostupné z: https://www.itnetwork.cz/html-css/html5/html-manual/html-css-html-manual-struktura/html-hlavicka-head-cesky-manual. [cit. 2025-02-13].

zobrazeno ve webovém prohlížeči a také na jednotlivých kartách stránky.[37] Na obrázku č. 9 můžete vidět jednotlivé odkazy na stránky ve webovém prohlížeči.



Obrázek 9: Odkazy na stránky v prohlížeči s daty z hlavičky HTML dokumentu

V hlavičce se můžeme setkat s dalšími elementy, jako například <link>, pomocí kterého se dokument prováže s externím souborem.[38] Z pravidla se používá k připojení CSS. Nemá obsah, ale pouze atributy:

1. href – tento atribut se využívá k specifikaci umístění připojovaného dokumentu
2. rel – tento atribut se využívá ke specifikaci vztahu mezi jednotlivými dokumenty

Na obrázku 10 si můžete prohlédnout připojení stylu pomocí link elementu.

[37] <title>: The Document Title element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title. [cit. 2025-02-13].
[38] <link>: The External Resource Link element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link. [cit. 2025-02-13].

Obrázek 10: Připojení CSS stylu v hlavičce dokumentu

3.2.2 Nejpoužívanější HTML elementy

<a>

Tomuto elementu se přezdívá „anchor“, neboli kotva, a společně s atributem href vytváří hypertextové odkazy, které odkazují na jiné webové stránky a na různé části dokumentu.[39]

<p>

Element <p> představuje v HTML odstavec, což je blok textu oddělený od dalších bloků textu pomocí mezery, která pomáhá strukturovat text.[40]

Tento element představuje řádkový textový kontejner, který je používán k obalení textu. Lehce se upravuje pomocí CSS a umožňuje efektivní manipulaci také s využitím jazyka Javascript. Je to nejpoužívanější element, co se textu týče.[41]

Element je používán jako blokový kontejner, který většinou kompletně nahrazuje většinu elementů jako <main>, <article> a <section>. Stejně jako je jednoduše nastylován. Je to jeden z nejpoužívanějších elementů v HTML.[42]

[39] <a>: The Anchor element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a. [cit. 2025-02-13].
[40] <p>: The Paragraph element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p. [cit. 2025-02-13].
[41] : The Content Span element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span. [cit. 2025-02-13].
[42] : The Content Division element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div. [cit. 2025-02-13].

Tento element se používá ke vkládání obrázku do dokumentu HTML. Pomocí jeho atributů lze změnit jeho výšku a šířku. Zároveň se musí pomocí atributu src připojit jeho zdroj.[43]

3.3 CSS

CSS se stará o vzhled – odděluje obsah od designu. Díky tomu je práce s dokumentem mnohem jednodušší a stránky vypadají mnohem atraktivněji. CSS upravuje estetickou stránku webu. Od výběru barev, písem či okrajů, až po komplexní layouty a dynamické efekty, které dodávají stránkám moderní vzhled, atraktivnost a funkčnost.[44] [45]

3.3.1 CSS frameworky

Frameworky jsou předem připravené kolekce funkcí, mixinů, proměnných a dalších nástrojů, které umožňují pomocí předpřipravených stylů zrychlit psaní kódu. Obsahují sadu opakovaně využitelných pravidel pro řešení běžných úloh při tvorbě CSS.[46]

Bootstrap framework má v sobě zabudovaných několik hotových komponentů jako jsou tlačítka a formuláře. Díky těmto předdefinovaným funkcím lze rychle vystavět web bez hlubších znalostí CSS.[47]

Tailwind framework nemá předdefinované hotové komponenty, ale předdefinované třídy s konkrétními účely. Tyto třídy lze dále kombinovat. To zaručí, že se vždy lze dostat ke svému vysněnému designu.[48]

3.3.2 Selektory

Selektory jsou pravidla, která můžete volně využívat v celém dokumentu. Hlavně se zaměřují na práci s třídami, ID a HTML elementy. Třídy se definují pomocí tečky (například .nadpis) a do HTML se vkládají pomocí atributu class. Díky selektorům tak můžete jednoduše a konzistentně upravovat styly na stránce, aniž by se musel psát kód znovu a znovu.

V CSS se využívá celá řada selektorů, které vám umožní cílit přímo na konkrétní HTML elementy a aplikovat na ně požadované styly. Mezi základní typy selektorů patří selektory podle tagu, podle třídy a podle ID:

---

[43] <img>: The Image Embed element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img. [cit. 2025-02-13].
[44] *Kaskádové styly*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 17. 9. 2024. Dostupné z: https://cs.wikipedia.org/wiki/Kaskádové_styly. [cit. 2025-02-13].
[45] *CSS: Cascading Style Sheets*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS. [cit. 2025-02-13].
[46] ŠTRÁFELDA, Jan. *Kaskádové styly (CSS)*. Online. Jan Štráfelda. Dostupné z: https://www.strafelda.cz/kaskadove-styly. [cit. 2025-02-13].
[47] *Bootstrap*. Online. Dostupné z: https://getbootstrap.com/. [cit. 2025-02-13].
[48] *Tailwind*. Online. Dostupné z: https://tailwindcss.com/. [cit. 2025-02-13].

Selektor podle tagu se definuje pouze názvem tagu. Například selektor p vybírá všechny odstavce, jak lze vidět na obrázku č. 11.

Obrázek 11: Výběr všech odstavců <p> pomocí CSS pravidla

U selektoru podle třídy se používá tečka – například .nadpis aplikuje styly na všechny elementy, které mají atribut class="nadpis". Ukázku můžete vidět na obrázku 12.

Obrázek 12: Ukázka selektoru podle třídy

Selektor podle id používá symbol hashtagu – například #header na obrázku 13 cílí na element s id="header".

Obrázek 13: Ukázka selektoru podle ID

3.3.3 Kombinátory, pseudotřídy a pseudoelementy

Kombinátory definují vztahy mezi HTML elementy. Vybírá všechny elementy, které se nacházejí uvnitř určitého rodičovského elementu, bez ohledu na úroveň jejich zanoření.[49] V ukázce na obrázku č. 14 vybere všechny odstavce uvnitř elementu s třídou content.

Obrázek 14: Ukázka kombinačního pravidla CSS

Pseudotřídy (např. :hover) určují specifické stavy elementů a dynamicky mění jejich vzhled na základě interakcí uživatele.[50] V ukázce na obrázku č. 15 se při najetí myší na odkaz změní jeho barva na červenou.

Obrázek 15: Ukázka použití pseudotřídy

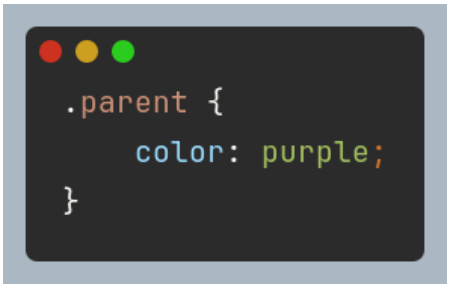
Pseudoelementy (např. ::first-letter) pak umožňují cílit na konkrétní části HTML elementů a aplikovat na ně specifické styly, třeba první písmeno odstavce.[51] V ukázce na obrázku č. 16 se první písmeno každého odstavce zvětší a zčervená.

[49] *CSS selectors and combinators*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_selectors/Selectors_and_combinators. [cit. 2025-02-13].
[50] *Pseudo-classes*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes. [cit. 2025-02-13].
[51] *Pseudo-elements*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements. [cit. 2025-02-13].

Obrázek 16: Ukázka použití pseudoelementu

3.3.4 Dědičnost

Dědičnost zase zajišťuje, že některé vlastnosti se automaticky přenášejí z rodičovských elementů na jejich potomky.[52] Všechny potomky této třídy zdědí barvu textu, což lze vidět na obrázku č. 17.



Obrázek 17: Ukázka rodičovského pravidla

3.3.5 Box model

Každý HTML element je vnímán jako obdélníkový box, který se skládá z několika vrstev:[53]

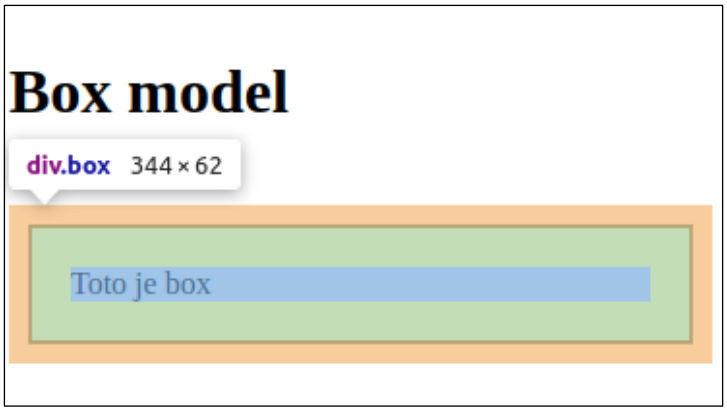
1. Obsah (content): Obsahuje data, jako je text nebo obrázky.
2. „Padding“: Vnitřní odsazení mezi obsahem a hranicí boxu.
3. „Border“: Rámeček, který obklopuje obsah a padding.
4. „Margin“: Vnější okraj, který odděluje box od okolních elementů.

Na obrázku 18 můžete vidět styl pro box na obrázku 19. Tam jsou barevně vyznačeny zmíněné části.

[52] MICHÁLEK, Martin. *Dědičnost v CSS: Co to je a kterých vlastností se týká?* Online. Vzhůru dolů. 2019. Dostupné z: https://www.vzhurudolu.cz/prirucka/css-dedicnost. [cit. 2025-02-13].
[53] *The box model*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Box_model. [cit. 2025-02-13].

```css
.box {
    width: 300px;
    padding: 20px;
    border: 2px solid #333;
    margin: 10px;
}
```

Obrázek 18: Kód pro element ukazující pravidla box modelu



Obrázek 19: Ukázka box modelu

3.3.6 Typografie a práce s barvami

CSS poskytuje nástroje pro detailní nastavení typografie:

1. Nastavení písma: font-family
2. Velikost písma: font-size
3. Řádkování: line-height
4. Zarovnání textu: text-align

Práce s barvami je také na vysoké úrovni. Lze je definovat pomocí hexadecimálních kódů, RGB či HSL hodnot. Na obrázku 20 je zobrazena ukázka. Když se kurzorem myši přejede přes tlačítko tak text zezelená.[54]

[54] *Typography in CSS*. Online. Cssreference.io. Dostupné z: https://cssreference.io/typography/. [cit. 2025-02-13].

```css
.button {
    background-color: #ff7e5f;
    color: #fff;
    transition: background-color 0.3s ease;
}

.button:hover {
    background-color: #06f957;
}
```

Obrázek 20: Ukázka typografie v CSS

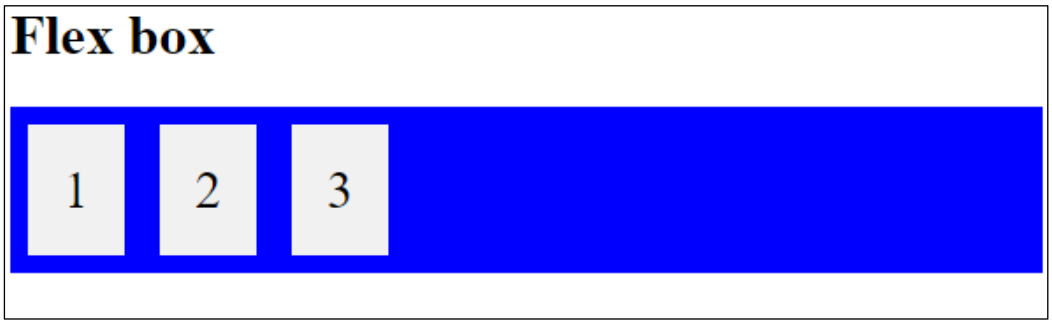
3.3.7 Flexbox

Flexbox je moderní způsob dynamického rozvržení prvků. Flexbox (Flexible Box Layout) funguje jako jednorozměrný systém, který umožňuje uspořádání prvků buď v řadě (horizontálně), nebo ve sloupci (vertikálně). Aby bylo možné využít výhod flexboxu, nastaví se kontejner pomocí vlastnosti display: flex, čímž se všechny jeho přímé potomky automaticky stanou flexibilními položkami. Dále lze nastavovat další vlastnosti flex-direction určuje směr uspořádání prvků, flex-wrap povoluje nebo zakazuje zalamování prvků na další řádky či sloupce, justify-content řídí horizontální rozmístění položek uvnitř kontejneru a umožňuje jejich centrování nebo rovnoměrné rozložení, dále align-items definuje vertikální zarovnání jednotlivých položek a align-content upravuje rozložení více řádků, pokud se položky zalamují.[55] Ukázku použití flexboxu najdete na obrázcích 21 a 21a.

```html
<!doctype html>
<html>
<head>
    <style>
        .flex-box {
            display: flex;
            background-color: blue;
        }

        .flex-box > div {
            background-color: #f1f1f1;
            margin: 10px;
            padding: 20px;
            font-size: 30px;
        }
    </style>
</head>
<body>

<h1>Flex box</h1>

<div class="flex-box">
    <div>1</div>
    <div>2</div>
    <div>3</div>
</div>

</body>
</html>
```

Obrázek 21a: Použití flexboxu v dokumentu

[55] *Flexbox*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Flexbox. [cit. 2025-02-13].



Obrázek 21: Výsledek kódu z obrázku 21a

3.3.8 Media Queries

Media queries se vyvolávají skrze direktivu @media. Používají se k tvorbě responzivních stránek a aplikací. Podle daných pravidel dokážou přizpůsobit styl a vzhled aplikace a webové stránky. Přizpůsobují aplikaci vašemu rozlišení a velikosti obrazovky. V ukázce na obrázku č. 22 se pro obrazovky s menší šířkou než 800px změní pozadí stránky a písmo odstavce.[56]

```
@media only screen and (max-width: 800px) {
    body {
        background-color: #ffeedd;
    }
    p {
        font-size: 20px;
    }
}
```

Obrázek 22: Ukázka responzivního stylu pomocí @media direktivy

3.3.9 Animace

@keyframes mění vlastnosti HTML dokumentu v průběhu času pomocí snímků (keyframes), ve kterých je specifikován stav animace. Tento způsob je velmi často využíván, jelikož se lze vyvarovat Javascriptu. Zadává se základní stav from a konečný stav to.[57] V ukázce na obrázku č. 23 se definuje animace, která mění průsvitnost.

[56] *Using media queries*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries/Using_media_queries. [cit. 2025-02-13].
[57] *@keyframes*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/@keyframes. [cit. 2025-02-13].

```css
@keyframes fadeIn {
    from {
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}

.element {
    animation: fadeIn 2s ease-in-out;
}
```

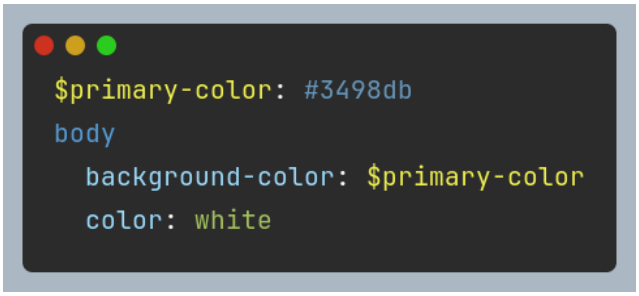
Obrázek 23: Ukázka animace v CSS

3.4 SASS

SASS, známý také jako Syntactically Awesome Stylesheets, je rozšíření jazyka CSS o programátorské funkce jako jsou mixiny, proměnné a cykly. Psaní kódu je díky těmto funkcím jednodušší a přehlednější. Je to takzvaný preprocesor, což znamená, že kód napsaný v SASSu se musí dále kompilovat (přeložit) do CSS kódu.[58]

3.4.1 Základní funkce

Mezi jednu z nejlepších funkcí patří proměnné, které umožňují zapamatování hodnot, jako jsou barvy, fonty či rozměry. SASS používá značku $ pro jejich označení. Na obrázku 24 například proměnná definuje barvu pozadí.[59]



```
$primary-color: #3498db
body
  background-color: $primary-color
  color: white
```

Obrázek 24: Proměnné v SASS

[58] *SASS*. Online. Dostupné z: https://sass-lang.com. [cit. 2025-02-13].
[59] *Variables*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/variables/. [cit. 2025-02-13].

Mixiny definují bloky stylů, které lze opakovaně využít. Zapisují se pomocí @mixin. Ukázku vidíte na obrázku 25. Mixinu radius-stinu lze poskytnout dva parametry – $radius a $stin. Následně lze použít mixin na třídu karta.[60]

```
@mixin radius-stinu($radius, $stin)
  border-radius: $radius
  box-shadow: $stin

.karta
  background-color: white
  padding: 20px
  @include radius-stinu(10px, 0 2px 5px rgba(0, 0, 0, 0.2))
```

Obrázek 25: Příklad mixinu v SASS

Dědičnost zajišťuje, že jedna třída bude mít stejné prvky jako druhá. Vyjadřuje se pomocí @extend. Podporuje princip DRY (Don't Repeat Yourself) a eliminuje nadbytečné opakování kódu, při kterém může jednoduše vzniknout chyba.[61]

```
%btn_zaklad
  padding: 10px 15px
  border: none
  border-radius: 3px
  font-size: 16px
  cursor: pointer

.btn_c1
  @extend %btn_zaklad
  background-color: $primary_color
  color: white
```

Obrázek 26: Příklad dědičnosti v SASS

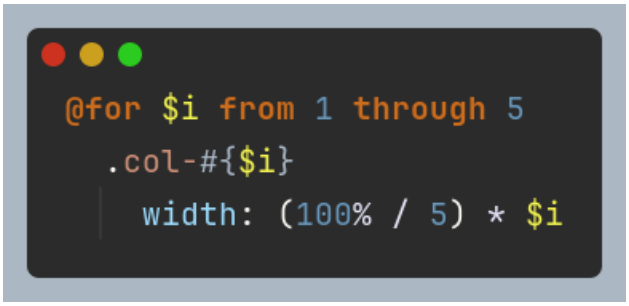
Jedny z nejdůležitějších funkcí SASSu, které značně usnadňují práci s kódem jsou určitě cykly. Umožňují psaní jednoduchých algoritmů.

[60] *Mixin Values*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/values/mixins/. [cit. 2025-02-13].
[61] *@extend*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/extend/. [cit. 2025-02-13].

@if/else direktiva definuje, za jakých podmínek se blok kódu použije. Parametr vrátí jednu z hodnot true nebo false. @else se používá pro vykonání kódu v případě, když se vrátí hodnota false.[62] Ukázku podmínky najdete na obrázku 27.

Obrázek 27: Ukázka direktivy @if

@for cyklus se opakuje několikrát podle zadaných hodnot a pokaždé se můžou výsledné hodnoty změnit. V ukázce na obrázku č. 28 se v CSS se vytvoří třídy od col1 do col5, přičemž každá bude mít jiné atributy.[63]

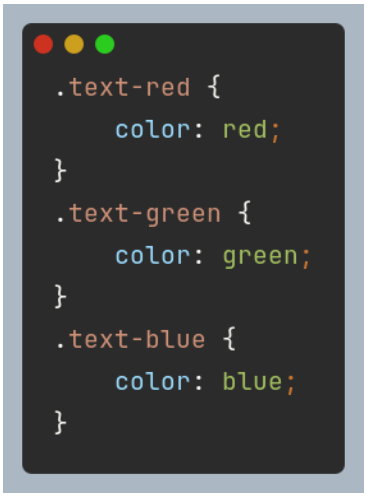


Obrázek 28: Ukázka for cyklu v SASS

@each cyklus umožňuje opakovat a střídat hodnoty v seznamech. V následujícím příkladu na obrázku 29 cyklus prochází seznam barev a pro každou generuje třídu s odpovídající barvou textu. Na obrázku 29a můžete vidět vygenerované CSS.[64]

[62] *@if and @else*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/control/if/. [cit. 2025-02-13].
[63] *@for*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/control/for/. [cit. 2025-02-13].
[64] *@each*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/control/each/. [cit. 2025-02-13].

Obrázek 29: Each cyklus v SASS



Obrázek 29a: Vygenerovaný CSS

@while cyklus pokračuje do doby, dokud výraz v podmínce nevrátí false. Vyjadřuje se jako @while <výraz/podmínky> {výsledek}. Cyklus z obrázku 30 vygeneruje tři třídy s rostoucí výškou.[65]

Obrázek 30: While cyklus v SASS

[65] *@while*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/control/while/. [cit. 2025-02-13].

4 VLASTNÍ TVORBA

4.1 Požadavky na program

Cílem naší praktické části bylo vytvořit objednávkový systém, podobný tomu v McDonald's, který by zjednodušil správu jídla a pití na studentské párty. Nejdůležitější částí měla být vyvolávací obrazovka s čísly objednávek, aby si každý mohl vyzvednout svou objednávku i ve všudypřítomném hluku. Dále byla potřeba obrazovka u kasy, kde může pokladník jednoduše vytvořit novou objednávku, a menší obrazovka pro zákazníka, zobrazující průběh tvorby objednávky, aby si mohl být jistý, že pokladník zadal vše správně. Samozřejmě nechyběla obrazovka v kuchyni a u výdeje, kde personál vidí, co je potřeba připravit, a zároveň může hotové objednávky „odkliknout“ (označit jako připravené k výdeji). Mezi další důležité funkce patřila možnost zobrazit a stáhnout zprávu o celkové tržbě, spolu se seznamem všech objednávek a nejprodávanějších produktů. Nejtěžším úkolem však bylo připojení tiskárny na účtenky, aby zákazník měl číslo své objednávky stále u sebe.

Po prvním testovacím provozu bylo potřeba přidat ještě malou obrazovku pouze na vracení kelímků (které byly zálohované, a tudíž bylo nutné vrácení zaznamenat pro správný výpočet výdělku), dále možnost snadné správy konfigurace produktů a speciální jednorázovou akci „Souboj budov“ (viz níže). Také byla provedena úprava, která umožnila neomezený počet obrazovek u kasy a malých obrazovek pro zákazníky.

4.2 Přístup k tvorbě

Jaký je tedy postup při tvorbě takto komplikované aplikace? Nejprve jsme se museli rozhodnout, o jaký typ programu půjde. Na výběr byly dvě možnosti – buď naprogramujeme počítačovou aplikaci, která bude obsahovat veškeré obrazovky, a server, který bude zajišťovat veškerou logiku a propojení mezi nimi, nebo vytvoříme webový server, kde jednotlivé obrazovky budou představovat webové stránky. Jelikož jsme měli více zkušeností s tvorbou webových stránek, rozhodli jsme se pro druhou možnost. Hlavním prvkem je tedy webový server, který si můžete představit jako aplikaci umožňující jiným zařízením navštívit určité webové stránky, jež server zpřístupňuje. Tento přístup se nazývá „full-stack vývoj“ a jeho principy jsou popsány výše.

4.3 Výběr programovacích jazyků

Po výběru typu programu následuje otázka „Jaké programovací jazyky využít?“. Full-stack je totiž velmi komplikovaná záležitost a backend je většinou napsaný jinak než frontend.

Pro tvorbu hlavního programu – webového serveru – bylo zvoleno využití programovacího jazyka Kotlin[66], jelikož s ním již máme nějaké zkušenosti. Kotlin je v podstatě novější a jednodušší verze Javy. Pokud máte s programováním nějaké zkušenosti, jistě víte, že Java je jazyk velmi užitečný díky schopnosti běžet na téměř jakémkoli operačním systému. Také obsahuje knihovny (soubory předem napsaných kódů pro jednodušší použití) skoro na cokoliv. Bohužel je ale také poměrně zastaralá a psaní kódu může být velmi repetitivní a vyčerpávající. Naštěstí firma JetBrains vytvořila programovací jazyk Kotlin, který se překládá do stejného „low-level" programovacího jazyka jako Java, a tím pádem dědí všechny její pozitivní vlastnosti. Je však mnohem jednodušší, přehlednější a obsahuje více funkcí. Navíc lze využít veškeré knihovny vyvinuté pro Javu i v prostředí Kotlinu. Na obrázku 31 můžete vidět ukázku kódu v Kotlinu.

```kotlin
fun main() {
    println("Hello, world!");

    val line = readln();
    println("You entered: $line");

    val (one, two) = readln().split(" ").map(String::toInt);
    println(Computer(one).add(two));
}

class Computer(val one: Int) {
    fun add(two: Int) = one + two;
}
```

Obrázek 31: Ukázka programovacího jazyka Kotlin

Dále potřebuje backend způsob, jak uchovávat data, například aktuální nastavení produktů pro objednání nebo samotné objednávky čekající na výdej. To lze vyřešit dvěma způsoby – pomocí databázového serveru (např. MySQL nebo PostgreSQL) nebo ukládáním dat do souboru. Rozhodli jsme se pro druhou možnost, protože je mnohem jednodušší. Data se v souborech obvykle uchovávají ve formátu JSON[67] (Javascript Object Notation), což je jednoduchý a přehledný způsob zápisu dat. Na obrázku 32 můžete vidět ukázku dat ve formátu JSON.

[66] https://kotlinlang.org/
[67] https://www.json.org/

Obrázek 32: Ukázka jazyka JSON

Výběr programovacích jazyků pro frontend byl velmi jednoduchý. Veškeré webové stránky totiž využívají tyto tři – HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets) a Javascript. Jak již bylo řečeno, HTML zpravidla definuje strukturu stránky, CSS vzhled a Javascript funkcionalitu. V rámci frontendu byla zároveň provedena drobná úprava – namísto tradičního CSS byl využit SASS.[68] (Syntactically Awesome Style Sheets), což, jak je také popsáno výše, je vylepšená verze CSS, která se do něj překládá. Na obrázcích 33, 34, 35 a 36 můžete vidět ukázky všech čtyř zmíněných jazyků.

[68] https://sass-lang.com/

```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>My first page</title>
</head>
<body>
<h1>Hello, world!</h1>
<p>What an exciting day!</p>
<div>
    <input type="text" placeholder="Enter your name">
    <button>Enter</button>
</div>
</body>
</html>
```

Obrázek 33: Ukázka HTML kódu

```css
.btn:hover {
    background-color: #9adcbe;
    transform: scale(0.9);
}

p {
    color: red;
    display: flex;
    font-family: Ani, Ubuntu, serif;
}

@keyframes wow {
    from {
        margin: 10rem;
        transform: translate(15px);
    }
    to {
        margin: 10px;
        transform: translate(-15px);
    }
}
```

Obrázek 34:Ukázka CSS kódu

```
let b :number = 8;
const numbers :any[] = [];

console.log("Hello, world!");
b += [];
compute(b, two: "test");

for (let i :number = 0; i < 15; i++) {
    numbers.push(i);
}

Show usages
function compute(one, two) {
    const sum = one + two;
    return sum - 2 / 3;
}
```

Obrázek 35: Ukázka Javascript kódu

```
@import "resources/css/sass/main"

$col: #54ab51
$var: 5
$list: 1 2 3 4 5 6 7 8 9

p
  color: $col
  @if($var > 2)
    background: red

.cls
  @extend .btn-outline
  border: none
  .top
    color: #4b989b

@each $i in $list
  .c#{$i}
    margin: #{$i}px
```

Obrázek 36: Ukázka SASS kódu

Na závěr stojí za zmínku šablonový „jazyk“ JTE[69] (Java Template Engine), který představuje jednoduché řešení pro propojení statických webových stránek a serveru. Funguje tak, že programátor místo čistého HTML kódu napíše tzv. šablonu (ukázka na obrázku 37), ve které

použije data ze serveru. Když pak server odesílá danou stránku klientovi, dosadí do ní aktuální hodnoty.

```
@import codes.rorak.maturprogram.models.Item
@import java.util.List

@param List<Item> items
@param boolean truthy

<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Item list</title>
</head>
<body>
@if(truthy)
    <h1>Hello, world!</h1>
@endif
<div id="items">
    @for(Item i : items)
        <div id="${i.getId()}">
            <span>${i.getName()}</span>
            <span>Price: ${i.getPrice()}</span>
        </div>
    @endfor
</div>
</body>
</html>
```

Obrázek 37: Ukázka JTE šablony

4.4 Další použité nástroje

Při programování nejsou důležité pouze jazyky, ale i prostředí, ve kterém je program vyvíjen. Bylo zvoleno využití IntelliJ IDEA Ultimate[70] od firmy JetBrains, která, jak již bylo zmíněno, stojí i za programovacím jazykem Kotlin. IntelliJ nabízí širokou škálu funkcí usnadňujících vývoj – od chytrého doplňování kódu až po nástroje pro práci s databázemi.

Za zmínku stojí také nástroj pro sestavení spustitelného programu Apache Maven[71], který pomocí konfiguračního souboru nejen instaluje potřebné knihovny, ale zároveň sestavuje program do konečné podoby. Do finálního JAR souboru přidá veškeré potřebné obrázky, soubory a knihovny, aby byl program kompletní a připravený k běhu.

[69] https://jte.gg/
[70] https://www.jetbrains.com/idea/
[71] https://maven.apache.org/

Nakonec uvedu některé použité knihovny. Nejdůležitější byl určitě framework Ktor[72], který zajišťuje veškeré funkce pro vytvoření a správu webového serveru. Jeho autorem je přímo JetBrains, stejná firma, která vytvořila Kotlin, což zaručuje vysokou spolehlivost a stabilitu kódu. Dále byly použity knihovny Jsoup[73] a Flying Saucer PDF[74] pro generování PDF souborů (například zprávy o výdělku) a knihovny JPOS a Epson pro komunikaci s tiskárnou.

4.5 Vlastní knihovna BetterKtor

Předem hotové a stažitelné knihovny však nebyly jediné, které jsme potřebovali. Nevýhodou frameworku Ktor je nepřehlednost jeho „endpointů“ (konkrétních URL adres pro komunikaci s webovou službou). Každý endpoint se totiž musí nejprve zaregistrovat v jednom konkrétním bloku, což může být nepřehledné. Mnohem přehlednější je tzv. file-based routing systém, kde stačí vytvořit soubor a jeho URL se automaticky odvíjí od jeho názvu a názvů složek, ve kterých se nachází. Tuto funkci jsme proto naprogramovali jako plugin (doplněk, který přidává nové funkce do programu) pro Ktor a publikovali ji jako vlastní Maven knihovnu *BetterKtor*.

4.5.1 Jak funguje

Při inicializaci webového serveru programátor použije funkci `install()`, čímž automaticky zaregistruje veškeré endpointy v určité složce ve zdrojovém kódu. V konfiguraci pluginu lze tuto složku specifikovat spolu s dalším nastavením, například způsobem převodu názvu souboru na URL nebo možností deaktivace websocketů (viz výše). Knihovna dále definuje rozhraní, která stačí v cílovém endpointu pouze implementovat. Metody endpointu pak určují buď HTTP metodu, nebo další „sub-endpointy“. K dispozici jsou rozhraní `BKRoute`, `BKWebsocket` a `BKErrorHandler`, která určují, zda je endpoint standardní, websocketový, nebo slouží jako záchyt pro chyby. Díky anotaci `@BKMulti` může jeden soubor implementovat více těchto rozhraní zároveň. Mimo to má programátor také mnoho dostupných anotací pro úpravu HTTP metod nebo URL cest. Na obrázku 38 můžete vidět ukázku jednoho takového endpointu.

[72] https://ktor.io/
[73] https://jsoup.org/
[74] https://github.com/flyingsaucerproject/flyingsaucer

```
@BKMulti
class User: BKRoute, BKWebsocket {
    override suspend fun get(call: ApplicationCall) {
        call.respondText("Hello, world!");
    }

    override suspend fun handle(session: DefaultWebSocketServerSession) {
        session.sendSerialized("Hello, world!");
        session.close();
    }

    @BKMethod(BKHttpMethod.PUT)
    suspend fun new(call: ApplicationCall) {
        val user = call.receive<User>();
        users.add(user);

        call.respond(HttpStatusCode.OK);
    }
}
```

Obrázek 38: Ukázkový endpoint pro BetterKtor

4.6 Backend – mozek programu

Jak již bylo zmíněno, nejdůležitější součástí programu je kód, který běží na serveru – backend. Ten zajišťuje provoz webového serveru, posílá klientovi webové stránky k zobrazení, řídí funkcionalitu jednotlivých stránek, ukládá a zpracovává data a komunikuje s externími zařízeními – v našem případě s tiskárnou. Celý program si ale můžete představit i jednodušeji: backend je vlastně jediný soubor, který spustíte na serveru. Frontend jsou v tomto scénáři pouze soubory, jako třeba obrázky, uložené na serveru. Program nejprve spustí webový server a zpřístupní ho veřejnosti v rámci LAN sítě. Jakmile klientův webový prohlížeč načte stránku identifikovanou IP adresou serveru, program následně na základě URL odešle příslušnou frontendovou stránku klientovi. Ta pak dále komunikuje se serverem pomocí skriptů, které jí server poskytl, a posílá mu další „requesty“ (požadavky).

4.6.1 Inicializace serveru, pluginy

První věc, kterou náš server udělá po nastavení tiskárny a načtení databází, je samozřejmě zapnutí serveru. Toho dosáhne pomocí Ktor funkce `embeddedServer()`, která přijme serverový port, „HTTP engine“ (v našem případě `Netty`) a hlavní modul pro aplikaci – funkci, která se spustí, jakmile se server nastaví a zapne. Toto můžete vidět na obrázku 39. V hlavním modulu nejprve pomocí funkce `install()` řekneme Ktoru, které pluginy chceme použít. V tabulce 3 si můžete prohlédnout jejich seznam. Pro každý plugin můžeme také specifikovat další konfiguraci.

```
embeddedServer(
    factory = Netty,
    port = args.getOrNull(0)?.toIntOrNull() ?: 8080,
    module = Application::main
).start(wait = true);
```

Obrázek 39: Zapnutí Ktor serveru

Název pluginu	**Funkce**
WebSockets	Využívání websocketů – obousměrná komunikace
BKPlugin	BetterKtor – viz výše
ContentNegotiation	Konverze datových tříd do JSON formátu
PartialContent	Odesílání souborů klientovi po částech
DoubleReceive	Možnost přijmout v kódu data od klienta vícekrát
AutoHeadResponse	Automatické odesílání hlaviček při odesílání souborů
Jte	JTE podpora

Tabulka 3: Seznam instalovaných pluginů

Dále v hlavním modulu nastavíme tzv. „static resources“. Jedná se o soubory, ke kterým může klient přistupovat bez nutnosti registrace endpointu, přímo přes URL podle složky a názvu. Toho docílíme pomocí funkcí `staticResources()` a `staticFiles()` v bloku `routing{}`. Nejprve specifikujeme URL, pomocí kterého budou soubory dostupné, a následně složku, ve které se soubory nachází. Metoda `staticResources()` používá soubory v interním úložišti programu, zatímco `staticFiles()` soubory v běžném úložišti serveru. Nakonec instalujeme vlastní plugin pro nastavení statických JTE šablon. Proces instalace pluginů si můžete prohlédnout na obrázku 40 a nastavení statických souborů na obrázku 41.

Obrázek 40: Kód pro instalaci pluginů

Obrázek 41: Kód pro nastavení statických souborů

4.6.2 Konfigurace

Je samozřejmé, že pokud chce správce programu upravit dostupné položky v menu, nebude upravovat kód programu a poté ho celý znovu sestavovat. Musí existovat jiná možnost. Tou je konfigurační soubor ve formátu JSON (viz výše). Při prvním spuštění programu se vytvoří složka `files`, která obsahuje prázdný soubor `items.json`. Uživatel je vyzván k úpravě tohoto souboru. Veškeré náhledové obrázky pro položky musí být nahrány do složky `img`.

Nahrávání konfigurace do paměti programu probíhá při spuštění a obstarává to funkce `loadItems()` v souboru `Items.kt`. Tento soubor obsahuje konstantu `items`, která je inicializována právě pomocí hlavní funkce při startu (viz obrázek 42). Funkce po kontrole existence souboru pomocí objektu `json` (vytvořeného s funkcí `Json` v souboru `Util.kt` – obrázek 43) a metody `File#readText()` kompiluje JSON formát do objektu třídy `Items`, definované dříve v souboru (ukázka: obrázek 44). Nakonec konfiguraci zkontroluje, aby

neobsahovala duplicitní produkty, neplatné obrázky apod. Tuto konfiguraci může program následně využívat kdykoli je třeba. Metodu `loadItems()` a potřebné konstanty můžete vidět na obrázku 45. Ukázku konfiguračního souboru na obrázku 46.

```
fun main(args: Array<String>) {
    Printer.init("noprint" !in args);

    Config.loadCurrent();

    items; // <- Zde
    orderHistory;

    embeddedServer(
        factory = Netty,
        port = args.getOrNull(0)?.toIntOrNull() ?: 8080,
        module = Application::main
    ).start(wait = true);
}
```

Obrázek 42: Inicializace databáze s položkami v hlavní metodě

```
val json = Json { this:JsonBuilder
    prettyPrint = true;
    encodeDefaults = true;
    ignoreUnknownKeys = true;
};
```

Obrázek 43: Konstrukce objektu json

```
@Serializable
data class Item(
    val id: String,
    var name: String,
    var price: Int,
    var image: String,
    @JsonNames("properties")
    val propertyIds: MutableList<String> = mutableListOf(),
    @JsonNames("not-in-order")
    var notInOrder: Boolean = false
) {
    val properties get() = propertyIds.mapNotNull { id -> items.properties.find { it.id == id } };
};

@Serializable
data class Property(
    val id: String,
    var name: String,
    var price: Int = 0
);
```

Obrázek 44: Definice datové třídy Item

```
val itemsFile = File(filesDir, "items.json");
val items = loadItems();

private fun loadItems(): Items {
    itemsFile.parentFile.mkdirs();
    imagesDir.mkdirs();

    if (!itemsFile.exists()) {
        copyResource("/${itemsFile.name}", itemsFile);
        println("items.json file created. Please fill it with your items.");
        println("img/ directory created. Please fill it with your images.");
        println("You can use './config' or '.\\config.bat' file to fill it with interface.");
        exitProcess(0);
    }

    val items: Items = json.decodeFromString(itemsFile.readText());
    validateItems(items);

    return items;
}
```

Obrázek 45: Metoda pro načítání produktů ze souboru

```
"id": "food",
"name": "Jídlo",
"image": "food.png",
"items": [
  {
    "id": "toasts",
    "name": "Tousty",
    "image": "toasts.png",
    "price": 45,
    "properties": [
      "ketchup",
      "mustard"
    ]
  },
  {
    "id": "fries",
    "name": "Hranolky",
    "image": "fries.png",
    "price": 55,
    "properties": [
      "ketchup",
      "mustard"
    ]
  },
```

Obrázek 46: Ukázka z databáze produktů

4.6.3 JSON databáze

Na podobném principu funguje uchovávání dat, která musí vydržet po restartu serveru. Je potřeba uchovávat celkem tři věci – aktuální probíhající objednávky, aktuální objednávky na přípravu a výdej a historii všech objednávek. Po restartu serveru stačí uchovávat pouze poslední dvě. Za tímto účelem používá server soubory `orders.json` a `order-history.json`.

O načítání a ukládání se stará soubor `Orders.kt`, který, podobně jako `Items.kt`, obsahuje konstanty pro obě databáze a ukládací a načítací funkce – `loadOrder()`, `saveOrders()`, `saveOrderHistory()` a `loadOrderHistory()`. U aktuálních objednávek je to jednoduché – stačí stejně jako u konfigurace přečíst soubor pomocí objektu `json` a načíst ho do objektu definované datové třídy `OrderSaves`. Metoda pro uložení tento proces pouze obrátí a použije `json.encodeToString()` místo `json.decodeFromString()`. Ukázku kódu z operací s databází objednávek můžete vidět na obrázku 47.

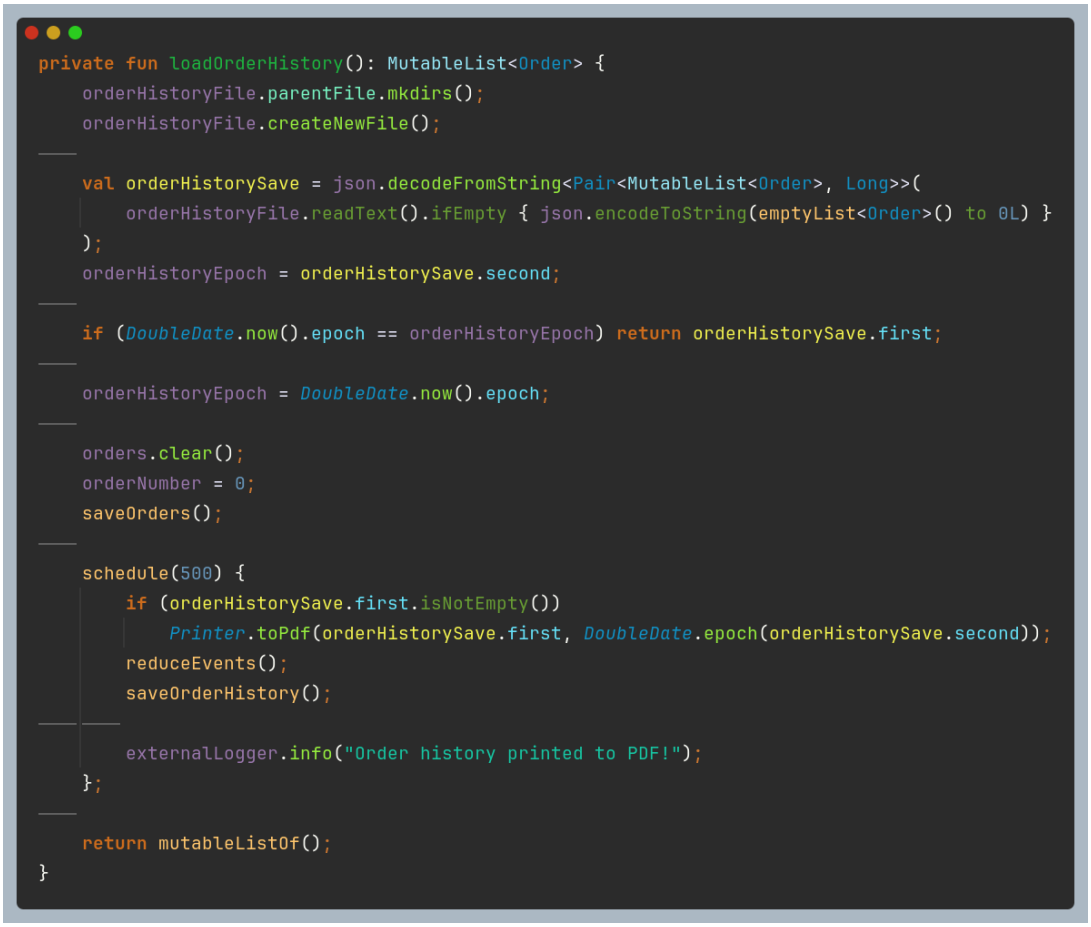
```
fun saveOrders() {
    orderFile.writeText(json.encodeToString(OrderSaves(orders, orderNumber)));
}

private fun loadOrders(): MutableList<Order> {
    orderFile.parentFile.mkdirs();
    orderFile.createNewFile();

    val orderSave: OrderSaves = json.decodeFromString(
        orderFile.readText().ifEmpty { Json.encodeToString(OrderSaves()); });

    orderNumber = orderSave.orderNum;

    return orderSave.orders.validate();
}
```

Obrázek 47: Metody pro uložení a načtení databáze s objednávkami

Historie objednávek je o něco složitější. Jelikož je třeba uložit pouze objednávky z aktuálního dne (tím se myslí od 12:00 do 12:00, jelikož párty se koná v noci), tak je třeba, aby se každý nový den staré objednávky nejprve uložily a následně vymazaly. Proto `order-history.json` obsahuje i položku s datem, pro které objednávky jsou. Funkce `saveOrderHistory()` i `loadOrderHistory()` ji obě porovnávají s aktuálním datem a na přelomu poledne pomocí pomocné třídy `Printer` uloží objednávky do PDF, které se dá později stáhnout. Ukázku kódu z obou funkcí můžete vidět na obrázcích 48 a 49.

```
fun saveOrderHistory() {
    if (DoubleDate.now().epoch == orderHistoryEpoch) {
        orderHistoryFile.writeText(json.encodeToString(orderHistory to orderHistoryEpoch));
        return;
    }

    if (orderHistory.isNotEmpty())
        Printer.toPdf(orderHistory, DoubleDate.epoch(orderHistoryEpoch));

    orders.clear();
    orderNumber = 0;
    saveOrders();

    orderHistory.clear();
    orderHistoryEpoch = DoubleDate.now().epoch;
    saveOrderHistory();
}
```

Obrázek 48: Metoda pro uložení historie objednávek



```
private fun loadOrderHistory(): MutableList<Order> {
    orderHistoryFile.parentFile.mkdirs();
    orderHistoryFile.createNewFile();

    val orderHistorySave = json.decodeFromString<Pair<MutableList<Order>, Long>>(
        orderHistoryFile.readText().ifEmpty { json.encodeToString(emptyList<Order>() to 0L) }
    );
    orderHistoryEpoch = orderHistorySave.second;

    if (DoubleDate.now().epoch == orderHistoryEpoch) return orderHistorySave.first;

    orderHistoryEpoch = DoubleDate.now().epoch;

    orders.clear();
    orderNumber = 0;
    saveOrders();

    schedule(500) {
        if (orderHistorySave.first.isNotEmpty())
            Printer.toPdf(orderHistorySave.first, DoubleDate.epoch(orderHistorySave.second));
        reduceEvents();
        saveOrderHistory();

        externalLogger.info("Order history printed to PDF!");
    };

    return mutableListOf();
}
```

Obrázek 49: Metoda pro načtení historie objednávek ze souboru

4.6.4 Systém jednorázových událostí

Další z klíčových bodů backendu je tzv. „event systém“. Po požadavku na jednorázovou akci „Souboj budov“, která spočívala v soutěži studentů ze dvou pracovišť školy o to, kdo utratí nejvíce, jsme se rozhodli vytvořit univerzální systém pro programování jednoduchých jednorázových akcí. Vše obsahuje soubor `Events.kt`. Nejdůležitější je abstraktní třída `Event`, kterou zdědí třída pro každou z akcí. Její implementaci můžete vidět na obrázku 50 a ukázku z objektu `BuildingCompetitionEvent`, který ji dědí na obrázku 51.

```
sealed class Event(val id: String, val name: String) {
    var enabled: Boolean
        get() {
            val value = events.events[id];
            if (value != null) return value;

            events.events[id] = false;
            saveEvents();
            return false;
        }
        set(value) {
            events.events[id] = value;
            saveEvents();
        };

    companion object {
        val all by lazy {
            Event::class.sealedSubclasses.mapNotNull { it.objectInstance }
        };
    }
}
```

Obrázek 50: Hlavní třída pro jednorázové události

```
data object BuildingCompetitionEvent: Event("build-competition", "Souboj budov") {
    suspend fun completeReceiveSchool(call: ApplicationCall): School {
        if (!enabled) return School.NONE;

        return runCatching { call.receive<Map<String, Int>>()["school"] }.getOrElse { it: Throwable
            throw MaturException("Invalid object received!");
        }?.school ?: School.NONE;
    }

    val Item.gamePointCount get() = eventItems.itemById(id).gamePointCount;
    val Order.school get() = events.orders.find { it.id == id }?.school ?: School.NONE;

    enum class School(val fullName: String) {
        HP("Horní Počernice"), CM("Černý Most"), NONE("Žádná");
    };
    val Int.school: School
        get() {
            return when (this) {...};
        };
    // ...
}
```

Obrázek 51: Zjednodušený kód třídy pro událost Souboj budov

Tento systém má také vlastní JSON soubor `events.json`, kde uchovává nejen informace o akcích (která je spuštěná a která ne), ale zároveň potřebná data pro akce. Například akce Souboj budov ukládá objednávky a jejich přiřazenou budovu. Všechny akce se dají ovládat pomocí `Events` endpointu (viz níže).

4.6.5 Endpointy, odchytávání chyb

A nakonec – nejdůležitější část backendu. Díky pluginu `BetterKtor` jsou všechny endpointy uloženy v balíčku (skupina souvisejících tříd a rozhraní v jedné složce) `endpoints`. Jejich detailní popis a rozdělení najdete níže.

Pozastavil bych se však u „endpointu“ `ErrorHandler`. Je to soubor, který implementuje rozhraní `BKErrorHandler`. Pokud je v jakémkoli jiném endpointu vyvolána jakákoliv chyba, je zavolána metoda `onError()`. V té se dle původu chyby buď vypne server, nebo je hlášení odesláno klientovi. Speciální chyby nastanou, pokud je neplatná cookie pro číslo objednávací obrazovky u kasy. V tom případě je tato cookie jednoduše resetována. Kód třídy `ErrorHandler` najdete na obrázku 52.

```
class ErrorHandler: BKErrorHandler {
    override suspend fun onError(call: ApplicationCall, cause: Throwable) {
        if (cause !is MaturException) {
            call.respond(HttpStatusCode.InternalServerError);
            throw cause;
        }

        val message = cause.msg;
        val code = cause.errorCode;

        if (cause is WrongCookieException)
            call.response.cookies.append(REGISTER_COOKIE_NAME, "", expires = GMTDate.START);

        if (call.request.httpMethod == HttpMethod.Get)
            call.respondTemplate("error.jte", "message" to message, responseCode = code);
        else call.respond(code, mapOf("message" to message));
    }
}
```

Obrázek 52: Třída pro odchytávání chyb z endpointů

4.7 Frontend – tvář programu

Avšak program by byl bezvýznamný, pokud by uživatel neměl možnost s ním interagovat. Pro komunikaci se serverem je totiž potřeba frontend; v našem programu ho představuje webová stránka. Hlavními cíli frontendu jsou jednoduchá vizualizace všech dat a intuitivní způsob manipulace s nimi. Dále udržuje spojení se serverem – odesílá změny (např. nové objednávky), ale také přijímá změny. Mozkem frontendu je programovací jazyk Javascript, který umožňuje

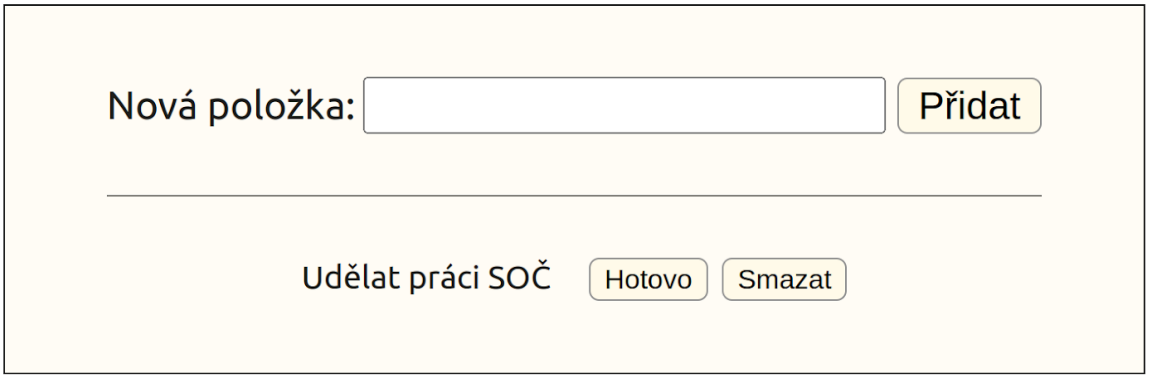
přímou komunikaci s uživatelským rozhraním – úpravu textu, stylů apod. – a zároveň pomocí metody `fetch()` odesílá data.

4.7.1 Zobrazování a aktualizace stránky

Představte si toto – tvoříte stránku pro správu vašeho seznamu úkolů. Musí obsahovat možnost tvorby nové položky, ale zároveň musí zobrazovat všechny aktuální položky. Pro každou z nich musí také nabízet možnost ji smazat nebo označit jako dokončenou. Strukturu stránky – HTML kód – si lze představit jako na obrázku 53. `<input>` element je místo, kam může uživatel napsat novou položku, a sousedícím `<button>` elementem ji může přidat na seznam. `<div>` element s `id="items"` je kontejner pro položky. Díky atributu `id` ho může Javascript jednoduše vyhledat a použít. V něm můžete vidět ukázkový produkt – další `<span>` element, který obsahuje tlačítko `<button>` pro smazání a pro označení. Z důvodu přehlednosti v rámci tohoto příkladu mají obě tlačítka identickou funkci. Pokud by byl ke stránce připojen i nějaký CSS soubor se stylem, mohla by vypadat třeba jako na obrázku 54.

```html
<label for="new">Nová položka:</label>
<input type="text" id="new">
<button>Přidat</button>
<div id="items">
    <span>Udělat práci SOČ <button>Hotovo</button><button>Smazat</button></span>
</div>
```

Obrázek 53: HTML kód pro „Todo“ aplikaci



Obrázek 54: Výsledná nastylovaná stránka

Následující ukázka demonstruje, jak lze stránce přidat potřebnou funkčnost. Nejprve vymažme ukázkový produkt a následně přidejme `onclick` atribut tlačítku pro přidání nové položky. V hodnotě tohoto atributu můžeme zavolat Javascriptovou funkci `add()`, kterou vytvoříme. Toto můžete vidět na obrázku 55. Samotný skript by vypadal asi podobně jako na obrázku 56. Pomocí funkce `document.querySelector()` vybere skript kontejner s položkami a

pomocí metody `createElement()` a `appendChild()` vytvoří a přidá nový element na stránku. Všimněte si nastavení `onclick` atributu, který jednoduše pomocí metody `remove()` element vymaže.

```html
<button onclick="add()">Přidat</button>
```

Obrázek 55: Použití atributu pro zavolání funkce ze skriptu

```javascript
function add() {
    const input = document.querySelector('#new');
    const items = document.querySelector('#items');

    const item = input.value;
    const element = document.createElement("span");
    const done = document.createElement("button");
    const remove = document.createElement("button");

    element.innerHTML = item;
    done.innerHTML = "Hotovo";
    remove.innerHTML = "Smazat";
    done.onclick = () => { element.remove(); };
    remove.onclick = () => { element.remove(); };

    element.appendChild(done);
    element.appendChild(remove);
    items.appendChild(element);

    input.value = "";
}
```

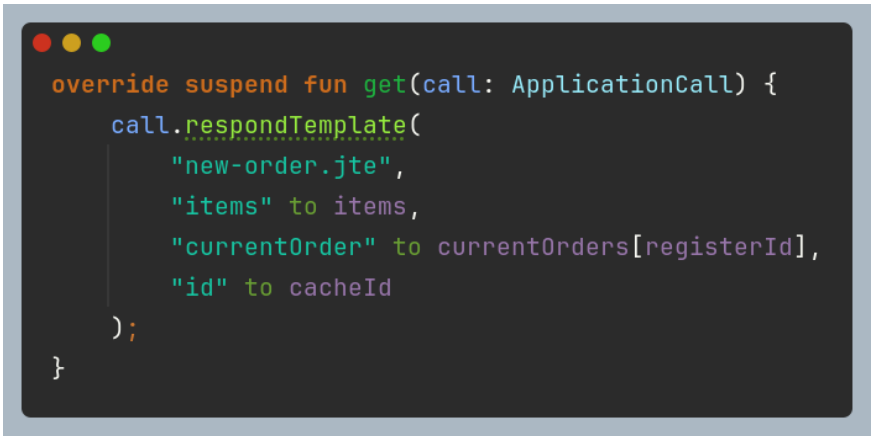
Obrázek 56: Finální skript pro „Todo“ stránku

Přesně takovou funkcionalitu má i náš program – nicméně výrazně komplexnější. Častá změna stylů a barev, zobrazování dialogů apod. tvoří většinu všech skriptů pro frontend. Určitě stojí za zmínku, že toto je všechno mnohem jednodušší za použití Javascriptových frontend frameworků (např. Svelte, Vue, React). V realitě málokdy narazíte na přístup, který používáme my.

4.7.2 JTE – šablony od serveru

Jak jsem již zmínil, JTE umožňuje odeslat webovou stránku přímo s aktuálními daty od serveru klientovi. Bez tohoto by klient musel o data požádat a až poté by stránka byla upravena. Tento

princip využíváme například při vykreslení obrazovky u kasy – server načte všechny produkty uvedené v konfiguraci do statického HTML souboru. Pomocí speciálních JTE elementů jako `@for` nebo `@if` lze do dokumentu jednoduše implementovat i logiku. Na začátku každého souboru definujeme pomocí elementu `@param` parametry – data od serveru, která může šablona využívat. Ukázku šablony jste mohli již vidět na obrázku 37 výše. Všimněte si, že data od serveru jsou uzavřena v `${}`. Ukázku kódu serveru, kde se odesílá šablona spolu s daty, můžete vidět na obrázku 57.

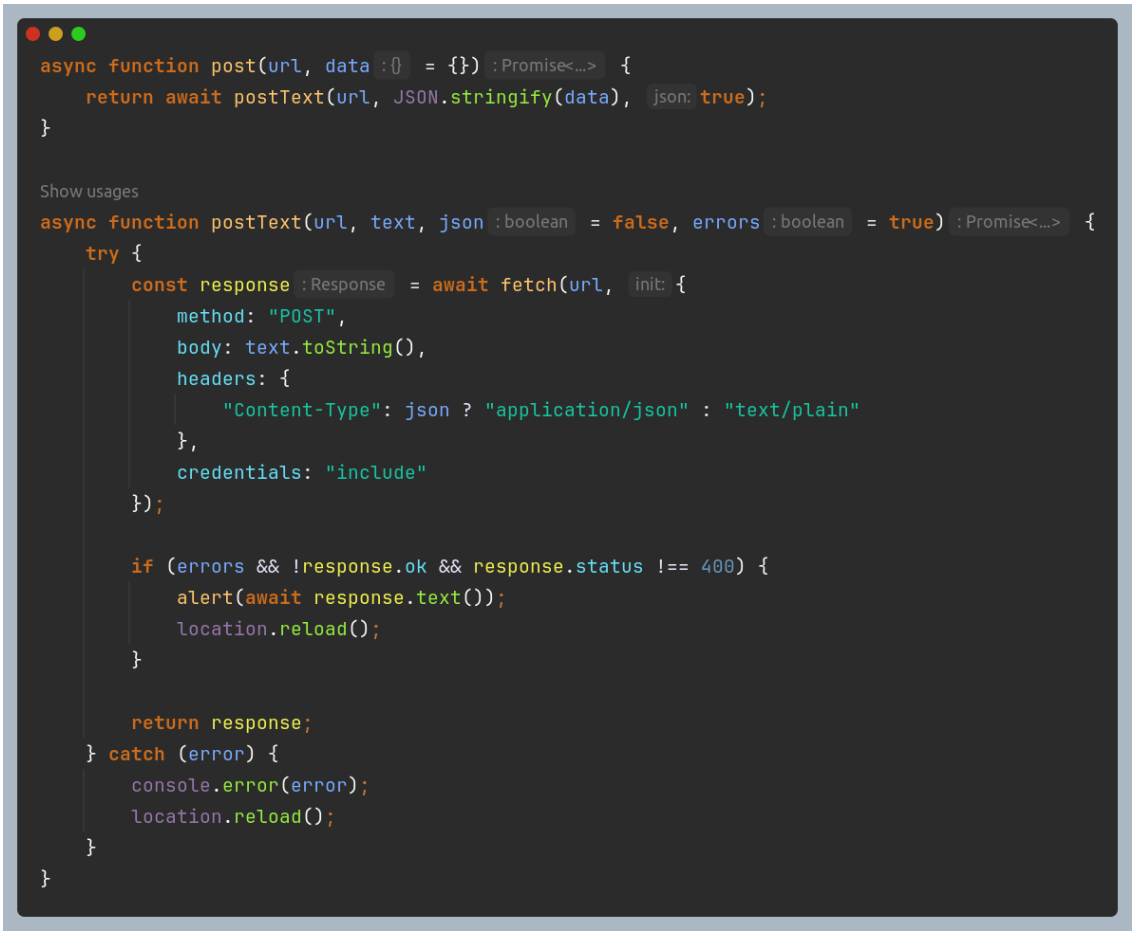


```
override suspend fun get(call: ApplicationCall) {
    call.respondTemplate(
        "new-order.jte",
        "items" to items,
        "currentOrder" to currentOrders[registerId],
        "id" to cacheId
    );
}
```

Obrázek 57: Odeslání JTE šablony

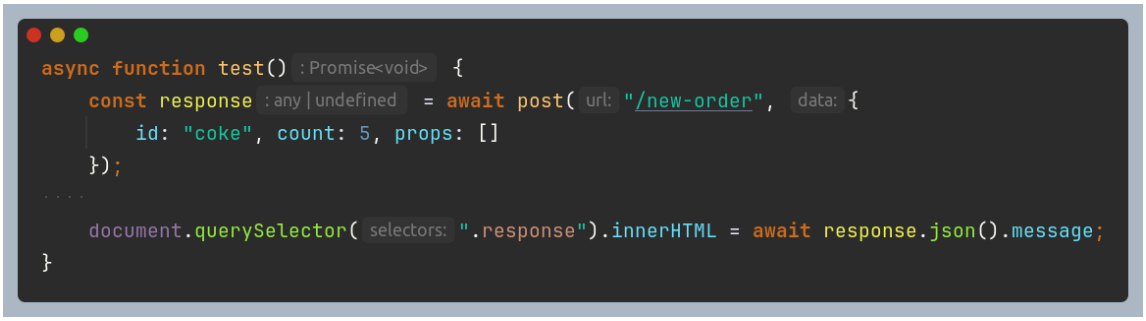
4.7.3 Komunikace se serverem

Hlavní způsob výměny dat se serverem je určitě pomocí funkce `fetch()`. Jedná se o asynchronní (pracuje na pozadí bez blokování programu) funkci, která pošle, defaultně `GET`, request na uvedené URL. Jelikož je třeba však nastavit spoustu různých parametrů, bylo jednodušší implementovat vlastní funkce `post()` a `postText()`, které tuto metodu využívají. Kód obou funkcí najdete na obrázku 58. Tyto funkce automaticky nastaví odesílaná data, hlavičku `Content-Type`, automaticky přidají cookie soubory a odchytí případné chyby. Všechny stránky jich využívají podobně, jako je ukázáno na obrázku 59. Odpověď serveru je následně uložena do proměnné typu `Response`. Ta obsahuje přijatá data, hlavičky, cookie soubory apod.



```
async function post(url, data :{}  = {}) :Promise<…>  {
    return await postText(url, JSON.stringify(data),  json: true);
}

Show usages
async function postText(url, text, json :boolean  = false, errors :boolean  = true) :Promise<…>  {
    try {
        const response :Response  = await fetch(url,  init: {
            method: "POST",
            body: text.toString(),
            headers: {
                "Content-Type": json ? "application/json" : "text/plain"
            },
            credentials: "include"
        });

        if (errors && !response.ok && response.status !== 400) {
            alert(await response.text());
            location.reload();
        }

        return response;
    } catch (error) {
        console.error(error);
        location.reload();
    }
}
```

Obrázek 58: Definice vlastních metod pro komunikaci se serverem



```
async function test() :Promise<void>  {
    const response :any | undefined  = await post( url: "/new-order",  data: {
        id: "coke", count: 5, props: []
    });

    document.querySelector( selectors: ".response").innerHTML = await response.json().message;
}
```

Obrázek 59: Použití vlastních metod pro komunikaci se serverem

4.7.4 Websockety

Funkce `fetch()` bohužel umožňuje pouze jednostrannou komunikaci – klient odesílá data a server odpovídá. V našem programu je však v mnoha případech potřeba úplný opak – například stránka s čísly objednávek potřebuje při každém přidání nebo odebrání objednávky obdržet od serveru zprávu, aby se mohla aktualizovat. Pro tento účel slouží websockety. Klient naváže připojení se serverem a následně čeká na zprávu. Můžete si to představit jako chatovací aplikaci: server může kdykoli něco napsat a klient též.

V Javascriptu jsou websockety používány pomocí třídy `WebSocket`. Ta má mimo metody `send()` pro odeslání dat také tzv. „event handlery“ `onmessage`, `onclose` a `onopen`, které se zavolají pokaždé, když se něco stane. Ukázku použití můžete vidět na obrázku 60. Podobně jako u funkce `fetch()` je třída `WebSocket` pro naší aplikaci nedostatečná. Proto jsme implementovali třídu `WS`, která umožňuje přidat více obslužných funkcí pro jeden událost a zároveň se automaticky snaží znovu připojit, pokud spojení selže. Ukázku kódu z `WS` můžete vidět na obrázku 61 a její použití na obrázku 62.

```
const socket :WebSocket = new WebSocket( url: "/new-order");
socket.onmessage = (e) :void => {
    element.innerHTML = e.data;
    ws.send( data: "OK!");
};
```

Obrázek 60: Ukázka použití Javascriptové websocket knihovny

Obrázek 61: Velmi zjednodušená definice vlastní websocketové třídy

Obrázek 62: Použití vlastní websocketové knihovny

4.7.5 Design – vlastní knihovna SASS tříd

Výše již byla zmíněna CSS knihovna Bootstrap, která definuje sadu předpřipravených tříd, jež usnadňují design stránek. Pro náš program jsme si podobnou knihovnu vytvořili sami – v SASS.

Kód je obsažen v souboru `style.sass`, kde jsou definovány veškeré třídy pro práci s flexboxem, mezerami, velikostí apod. V souboru `main.sass` se pak nachází definice barev a více specifických tříd určených přímo pro náš program. Na obrázku 63 můžete vidět ukázku kódu, obrázek 64 naopak ukazuje příklad jeho použití.

```sass
// margins
@each $unit in $units
  .m#{nth($unit, 1)}
    margin: nth($unit, 2)
  .mt#{nth($unit, 1)}
    margin-top: nth($unit, 2)
  .mr#{nth($unit, 1)}
    margin-right: nth($unit, 2)
  .mb#{nth($unit, 1)}
    margin-bottom: nth($unit, 2)
  .ml#{nth($unit, 1)}
    margin-left: nth($unit, 2)
  .mh#{nth($unit, 1)}
    margin-right: nth($unit, 2)
    margin-left: nth($unit, 2)
  .mv#{nth($unit, 1)}
    margin-top: nth($unit, 2)
    margin-bottom: nth($unit, 2)
```

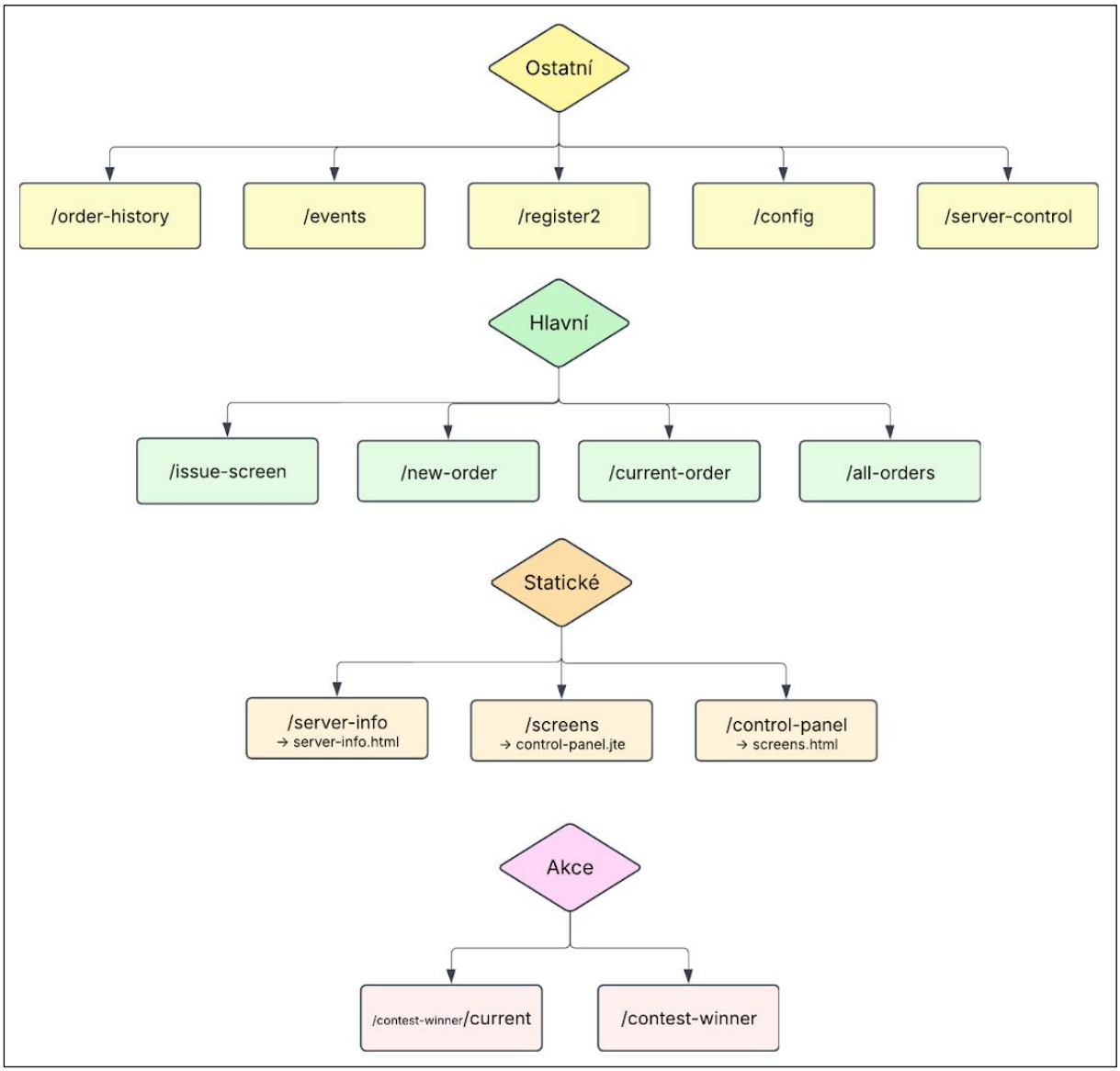
Obrázek 63: Ukázka kódu z vlastní SASS knihovny – kód pro vnější mezery objektů

```html
<div class="ccc mx4 px2 b">
    <span class="fsx5 r tac">Hello, world!</span>
    <button class="btn-outline btn-red-d fsx2">Click me</button>
</div>
```

Obrázek 64: Použití vlastní SASS knihovny

4.8 Jednotlivé stránky

Na obrázku 65 si můžete prohlédnout mapu všech jednotlivých endpointů/stránek. V tabulce 4 je u každého endpointu jednoduše popsána jeho funkce. Pro URL `/` je nastaveno automatické přesměrování na `/screens`.



Obrázek 65: Schéma endpointů programu

URL	Endpoint	Soubor webové stránky	Popis
`/new-order`	`NewOrder.kt`	`new-order.jte`	U kasy, obrazovka, kam pokladník zadává zákazníkovy objednané produkty.
`/current-order`	`CurrentOrder.kt`	`current-order.html`	Malá obrazovka u kasy, kde zákazník vidí svou objednávku.
`/all-orders`	`AllOrders.kt`	`all-orders.html`	Obrazovky v kuchyni a u výdeje pro personál.
`/issue-screen`	`IssueScreen.kt`	`issue-screen.html`	Velká obrazovka s čísly objednávek, dělená na hotové a nedokončené.
`/register2`	`Register2.kt`	`register2.jte`	Obrazovka pouze pro vracení kelímků (kvůli evidenci).
`/config`	`Config.kt`	`config.jte`	Obrazovka pro správu konfigurací.
`/order-history`	`OrderHistory.kt`	`order-history.jte`	Obrazovka pro zobrazení a stáhnutí denních zpráv o výdělku.
`/events`	`Events.kt`	`events.jte`	Obrazovka pro správu jednorázových akcí.
`/server-control`	`Settings.kt`	`server-control.html`	Obrazovka pro správu serverových databází a serveru jako takového.
`/contest-winner`	*Statický*	`contest-winner.jte`	Obrazovka s vyhlášením aktuálního vítěze soutěže.
`├─ /current`	*Statický*	`contest-winner/current.jte`	Obrazovka s průběžným skóre obou soutěžících budov.
`/server-info`	*Statický*	`server-info.html`	Obrazovka pro kontrolu, zdali server běží.
`/screens`	*Statický*	`screens.html`	Hlavní směrník obrazovek, obsahuje odkazy na hlavní obrazovky.

`/control-panel`	*Statický*	`control-panel.jte`	Hlavní směrník obrazovek pro administrátory.

Tabulka 4: Seznam endpointů programu

4.8.1 Rozcestník obrazovek

Stránka `/screens` je úvodní stránkou programu. Je definována jako statický HTML soubor ve složce `static` pomocí metody `staticResources()`, jak je již zmíněno výše. Obsahuje odkazy konkrétně na tyto stránky: `/new-order`, `/register2`, `/current-order`, `/all-orders`, `/issue-screen` a `/server-info`.

Frontend

HTML struktura stránky obsahuje pouze nadpis a následně šest tlačítek (`<button>` elementů) uzavřených v `<div>` kontejneru. Skript je velmi jednoduchý – jedná se o jedinou funkci přímo v HTML souboru ve `<script>` elementu. Ta je potřeba pouze při výběru nové objednávky nebo zobrazení aktuální objednávky. Zeptá se uživatele na ID – jelikož je možné mít více kas, je třeba je nějak odlišit, aby se správná objednávková obrazovka dala propojit se správnou obrazovkou pro zobrazení. Přesměrování probíhá pro všechny obrazovky pomocí `location.replace()`. Tato funkce je použitá proto, aby se například pokladník nemohl omylem kliknutím na šipku zpět v prohlížeči vrátit na rozcestník a něco pokazit. HTML kód můžete vidět na obrázku 66 a skript na obrázku 67.

```
<span class="tb pac mb2 fsx7">Výběr obrazovky</span>
<div class="c tac">
    <button onclick="idChooseRedirect('/new-order')" class="btn-outline btn-green-d mb1 fsx2">Nová objednávka</button>
    <button onclick="location.replace('/register2')" class="btn-outline btn-cyan-d mb1 fsx2">Mini kasa</button>
    <button onclick="idChooseRedirect('/current-order')" class="btn-outline btn-red-d mb1 fsx2">Zobrazení aktuální
        objednávky
    </button>
    <button onclick="location.replace('/all-orders')" class="btn-outline btn-orange-d mb1 fsx2">Objednávky k připravení
    </button>
    <button onclick="location.replace('/issue-screen')" class="btn-outline btn-purple-d mb1 fsx2">Vyvolávání objednávek
    </button>
    <button onclick="location.replace('/server-info')" class="btn-outline btn-grey-d fsx2">Stav serveru</button>
</div>
```

Obrázek 66: Hlavní část kódu pro rozcestník obrazovek

```
function idChooseRedirect(path) {
    const id = prompt("Zadej id pro kasu, nech prázdné pro minulé id.");
    if (!id) {
        location.replace(path);
        return;
    }

    if (!id.match(/^1?\d$/)) {
        alert("Id kasy může být pouze číslo od 0 do 19!");
        location.reload();
        return;
    }

    location.replace(`${path}?registerId=${id}`);
}
```

Obrázek 67: Funkce pro změnu URL použitá v rozcestníku

Design

Pro tuto stránku jsme se rozhodli použít velmi jednoduchý a hravý design bez složitých vlastních stylů – pouze jsme využili naší výše zmíněnou předprogramovanou knihovnu. Použili jsme duhové barevné kombinace pro tlačítka a světle modré pozadí s růžovým textem nadpisu. Výsledek můžete vidět na obrázku 68.

Obrázek 68: Rozcestník obrazovek

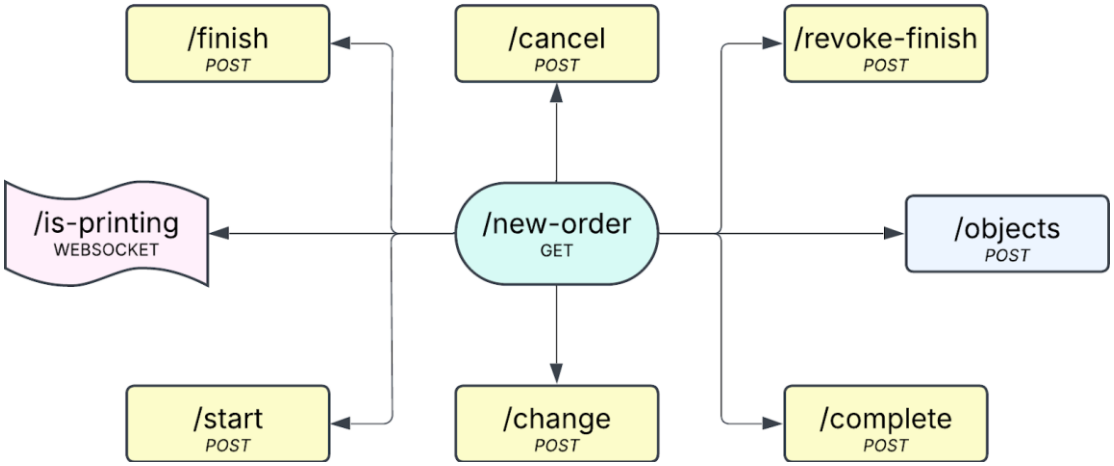
4.8.2 Tvorba nové objednávky

Nejdůležitější endpointem celého programu je bezesporu endpoint `/new-order`. Jak bylo již zmíněno, každá tato obrazovka má své identifikační číslo, aby se dala jednoduše propojit s

obrazovkou pro zákazníka. Při spuštění se nejprve zobrazí pokyn „Klikni pro začátek nové objednávky“, načež se po kliknutí zobrazí seznam kategorií a položek, ze kterých pokladník dle zákazníkova přání vybírá a přidává je „do košíku“. Samozřejmě je možné objednávku libovolně upravovat, přidávat maličkosti navíc (například kečup k hranolkům) apod. Po dokončení objednávky může pokladník zadat procentuální slevu a po potvrzení se zákazníkovi zobrazí pokyn k zaplacení. Po zaplacení server vytiskne účtenku a objednávku pošle dál do systému.

Backend

Veškerou logiku pro novou objednávku zajišťuje endpointová třída `NewOrder.kt`. Mapu rozvržení backendových funkcí můžete vidět na obrázku 69.



Obrázek 69: Rozvržení endpointů stránky pro tvorbu nové objednávky

Při načtení stránky webovým prohlížečem je poslán `GET` request s query parametrem `registerId` obsahujícím identifikační číslo kasy. Podle čísla se server rozhodne mezi třemi možnostmi:

- Pokud je ID neplatné, přesměruje klienta na obrazovku `/screens`.
- Pokud na dané kase ještě není započatá objednávka, odešle klientovi stránku `new-order-start.html` s výzvou k zahájení objednávky.
- Pokud objednávka na dané kase již běží, načte stránku `new-order.jte` s produkty v objednávce a jejím stavem.

Zároveň také uloží ID kasy do cookie souboru, který odešle spolu se stránkou. Kód této metody můžete vidět na obrázku 70. Při zkušebním provozu byla stránka velmi pomalá, jelikož se všechny obrázky načítaly po každém obnovení stránky znovu. Tomu jsme jednoduše zabránili vygenerováním nového ID pro obrázky po každém spuštění serveru a použitím `CacheControl.MaxAge()`, které nastaví trvanlivost obrázků v interní paměti prohlížeče. Jednoduše řečeno – obrázky budou načteny pouze poprvé po spuštění serveru.

```
override suspend fun get(call: ApplicationCall) = mutex.withLock {
    val registerId = call.request.queryParameters["registerId"] ?: call.registerCookie;

    if (registerId == null) {
        call.respondRedirect("/screens");
        return;
    }

    call.response.cookies.append(REGISTER_COOKIE_NAME, registerId, maxAge = 86400L);

    if (currentOrders[registerId] == null) call.respondPage("new-order-start.html");
    else call.respondTemplate(
        "new-order.jte",
        "items" to items,
        "currentOrder" to currentOrders[registerId],
        "id" to cacheId
    );
}
```

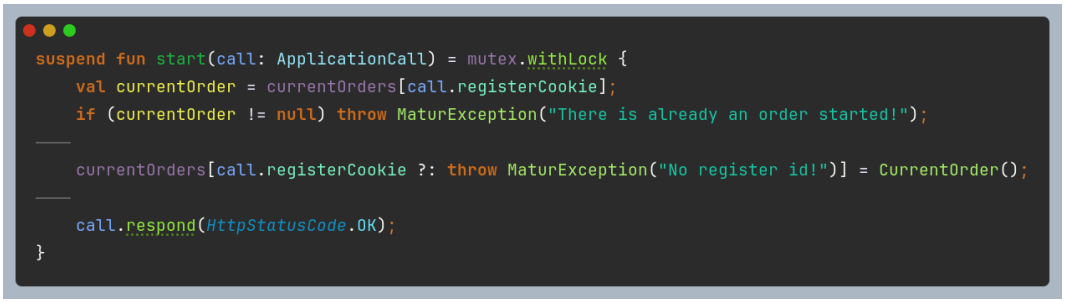
Obrázek 70: Kód pro metodu GET endpointu /new-order

Šablony JTE bohužel načítají data pouze do struktury stránky, není možné je předat Javascriptu. K tomu slouží endpoint `/new-order/objects`, který Javascript použije ihned po spuštění stránky a pomocí něj si upraví interní paměť. Kód je velmi jednoduchý a najdete ho na obrázku 71.

```
suspend fun objects(call: ApplicationCall) = mutex.withLock {
    val currentOrder = currentOrders[call.registerCookie] ?: throw MaturException("Current order not found!");

    call.respond(currentOrder to items);
}
```

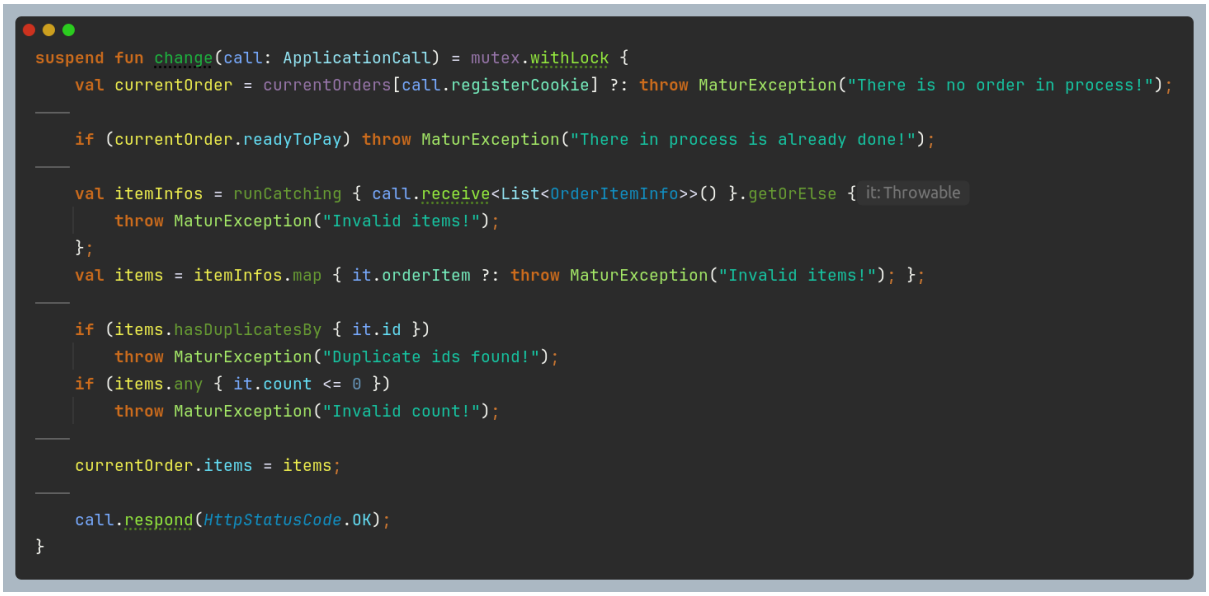
Obrázek 71: Kód endpointu pro získání aktuální objednávky z databáze

Pokud není objednávka spuštěná a klient ji spustí kliknutím na výzvu, odešle se POST request na `/new-order/start`. Server vytvoří novou objednávku na kase s daným ID a upraví interní databázi. Klienta však nepřesměruje – ten se totiž obnoví sám, což způsobí nový GET request, tentokrát s existující objednávkou. Kód je na obrázku 72.



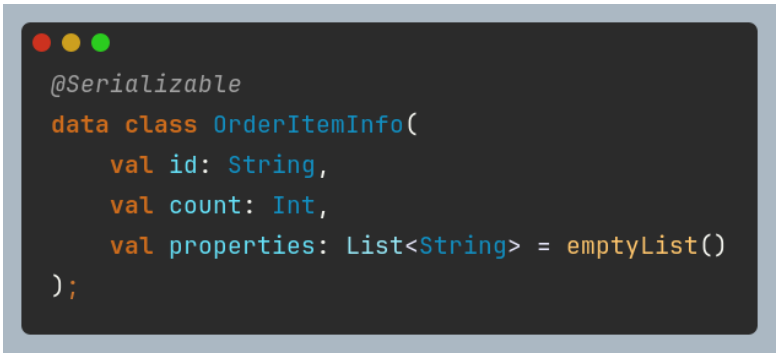
Obrázek 72: Kód endpointu pro započetí nové objednávky

Pokaždé, když se něco v objednávce změní, pošle klient `POST` request na `/new-order/change` se všemi produkty v aktuálním košíku. Server nejprve všechny produkty zkontroluje a následně upraví svou databázi. Datová třída pro reprezentaci produktu v košíku je definována v souboru `Orders.kt` jako `OrderItemInfo`. Kód metody `change()` můžete vidět na obrázku 73 a definici `OrderItemInfo` na obrázku 74.



```
suspend fun change(call: ApplicationCall) = mutex.withLock {
    val currentOrder = currentOrders[call.registerCookie] ?: throw MaturException("There is no order in process!");

    if (currentOrder.readyToPay) throw MaturException("There in process is already done!");

    val itemInfos = runCatching { call.receive<List<OrderItemInfo>>() }.getOrElse { it: Throwable
        throw MaturException("Invalid items!");
    };
    val items = itemInfos.map { it.orderItem ?: throw MaturException("Invalid items!"); };

    if (items.hasDuplicatesBy { it.id })
        throw MaturException("Duplicate ids found!");
    if (items.any { it.count <= 0 })
        throw MaturException("Invalid count!");

    currentOrder.items = items;

    call.respond(HttpStatusCode.OK);
}
```

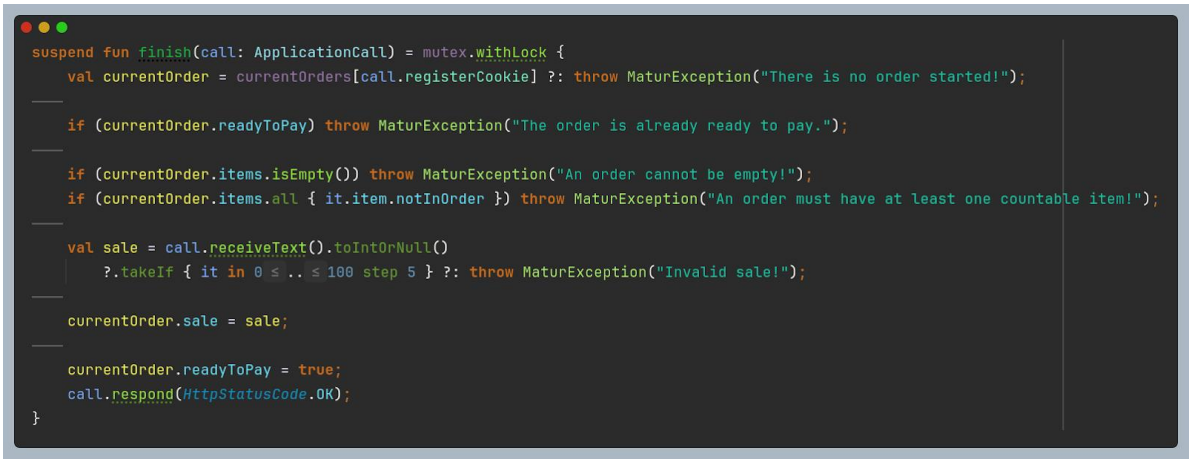
Obrázek 73: Kód funkce pro endpoint zařizující změnu položek v aktuální objednávce



```
@Serializable
data class OrderItemInfo(
    val id: String,
    val count: Int,
    val properties: List<String> = emptyList()
);
```

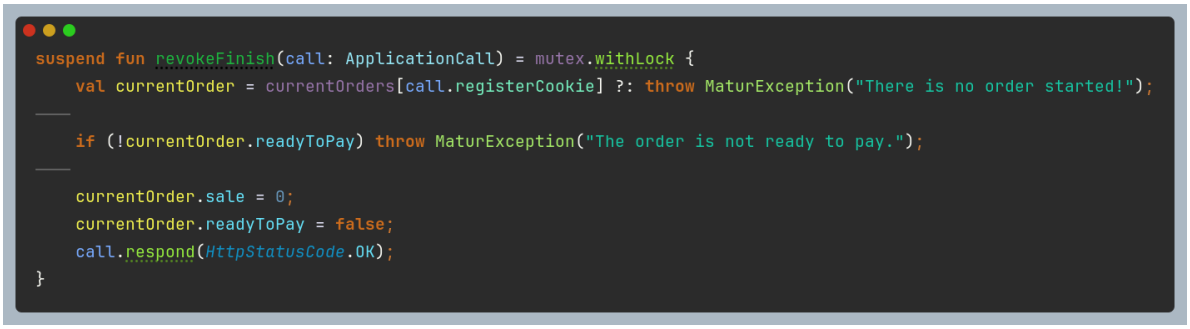
Obrázek 74: Definice datové třídy pro výměnu položek mezi serverem a klientem

Jakmile má zákazník vybráno, dokončí pokladník objednávku. To odešle `POST` request na `/new-order/finish` spolu s vybranou slevou. Server nejprve zkontroluje slevu a košík, poté nastaví objednávku jako připravenou k zaplacení. Kód této metody můžete vidět na obrázku 75. Pokud si však zákazník něco rozmyslí, je zde možnost objednávku upravit i po dokončení. K tomu slouží endpoint `/new-order/revoke-finish`, který jednoduše objednávku vrátí do původního stavu. Kód této metody najdete na obrázku 76.



```
suspend fun finish(call: ApplicationCall) = mutex.withLock {
    val currentOrder = currentOrders[call.registerCookie] ?: throw MaturException("There is no order started!");

    if (currentOrder.readyToPay) throw MaturException("The order is already ready to pay.");

    if (currentOrder.items.isEmpty()) throw MaturException("An order cannot be empty!");
    if (currentOrder.items.all { it.item.notInOrder }) throw MaturException("An order must have at least one countable item!");

    val sale = call.receiveText().toIntOrNull()
        ?.takeIf { it in 0 ≤ .. ≤ 100 step 5 } ?: throw MaturException("Invalid sale!");

    currentOrder.sale = sale;

    currentOrder.readyToPay = true;
    call.respond(HttpStatusCode.OK);
}
```

Obrázek 75: Kód endpointu pro ukončení nové objednávky



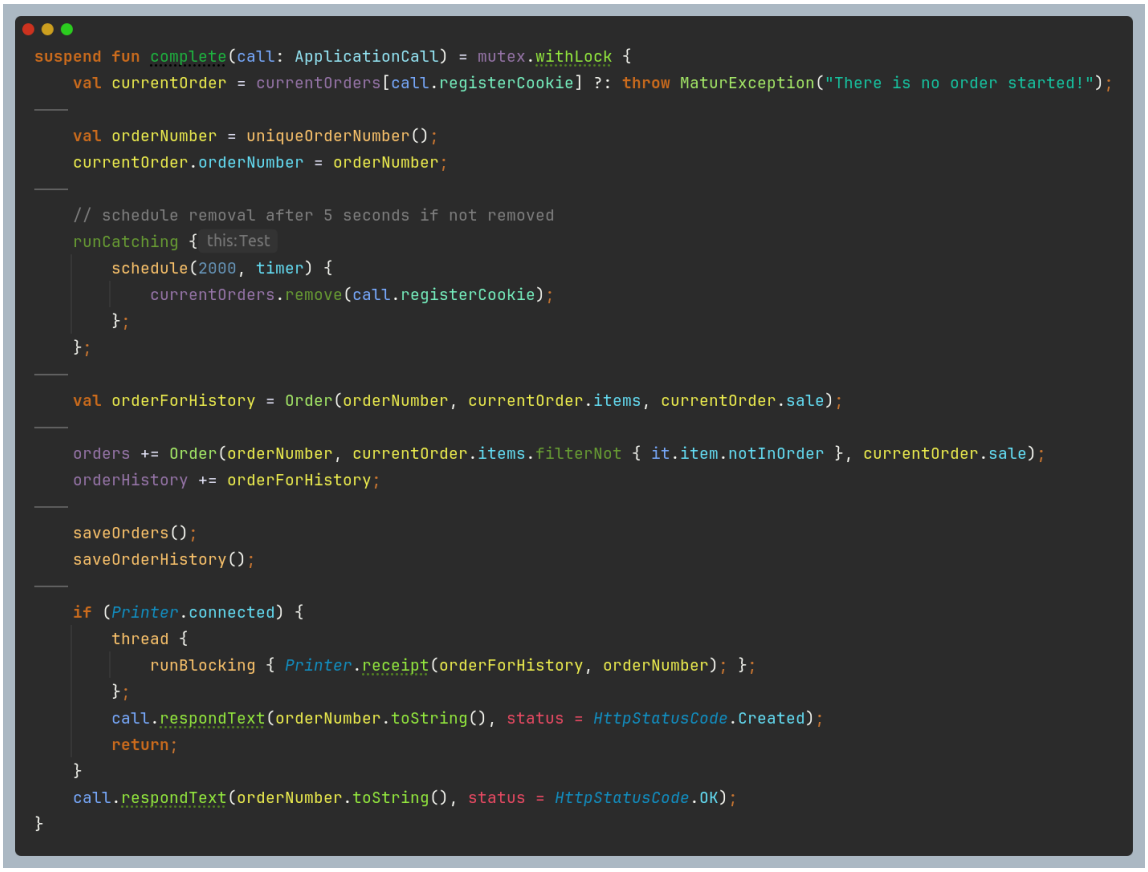
```
suspend fun revokeFinish(call: ApplicationCall) = mutex.withLock {
    val currentOrder = currentOrders[call.registerCookie] ?: throw MaturException("There is no order started!");

    if (!currentOrder.readyToPay) throw MaturException("The order is not ready to pay.");

    currentOrder.sale = 0;
    currentOrder.readyToPay = false;
    call.respond(HttpStatusCode.OK);
}
```

Obrázek 76: Kód endpointu pro vrácení nové objednávky do neukončeného stavu

Po zaplacení zbývá už jen objednávku uzavřít. Jakmile klient pošle `POST` request na `/new-order/complete`, stane se několik věcí:

- Objednávce je vytvořeno číslo, které je poté odesláno klientovi i obrazovce pro zákazníka.
- Objednávka je převedena na objekt, který lze uložit do databáze a následně uložena.
- Pokud je připojena tiskárna, vytiskne se účtenka. Pokud není, klient obdrží speciální odpověď, která pokladníkovi umožní ručně napsat číslo pro zákazníka.

Aktuální objednávka je odebrána z databáze probíhajících objednávek až poté, co si obrazovka `CurrentOrder` vyžádá číslo. Pokud se tak nestane, objednávka je smazána po pěti vteřinách. Zjednodušený kód můžete vidět na obrázku 77.



```
suspend fun complete(call: ApplicationCall) = mutex.withLock {
    val currentOrder = currentOrders[call.registerCookie] ?: throw MaturException("There is no order started!");

    val orderNumber = uniqueOrderNumber();
    currentOrder.orderNumber = orderNumber;

    // schedule removal after 5 seconds if not removed
    runCatching { this:Test
        schedule(2000, timer) {
            currentOrders.remove(call.registerCookie);
        };
    };

    val orderForHistory = Order(orderNumber, currentOrder.items, currentOrder.sale);

    orders += Order(orderNumber, currentOrder.items.filterNot { it.item.notInOrder }, currentOrder.sale);
    orderHistory += orderForHistory;

    saveOrders();
    saveOrderHistory();

    if (Printer.connected) {
        thread {
            runBlocking { Printer.receipt(orderForHistory, orderNumber); };
        };
        call.respondText(orderNumber.toString(), status = HttpStatusCode.Created);
        return;
    }
    call.respondText(orderNumber.toString(), status = HttpStatusCode.OK);
}
```

Obrázek 77: Zjednodušený kód metody pro zkompletování objednávky

Pokud je objednávka v jakékoli fázi zrušena, zavolá se metoda `cancel()`, která objednávku po zkontrolování dat smaže. Kód najdete na obrázku 78.

```
suspend fun cancel(call: ApplicationCall) = mutex.withLock {
    val currentOrder =
        currentOrders[call.registerCookie] ?: throw MaturException("Cannot cancel a non-existent order!");

    if (currentOrder.orderNumber != null) throw MaturException("Cannot cancel an already completed order!");

    currentOrders.remove(call.registerCookie);
    call.respond(HttpStatusCode.OK);
}
```

Obrázek 78: Kód metody obsluhující endpoint pro zrušení nové objednávky

Posledním a nejzajímavějším endpointem je websocket `/new-order/is-printing`. Na něj se klient připojí po uzavření objednávky a čeká na zprávu od serveru o dokončení tisku. Kód najdete na obrázku 79. Konstanty jako `Printer.printNums` budou probrány později v sekci o tiskárně.

```
@BKMultiFor(BKRouteType.WEBSOCKET)
suspend fun isPrinting(session: DefaultWebSocketServerSession, call: ApplicationCall) {
    val printNum = call.request.cookies["print-num"] ?: session.close().also { return; };

    while (true) {
        delay(200);
        if (Printer.printNums.contains(printNum)) continue;
        session.send("close");
        session.close();
        break;
    }
}
```

Obrázek 79: Kód websocketové „handle“ funkce pro zjištění stavu tiskárny

Frontend

Šablona stránky `new-order.jte` je velmi komplikovaná. Jako parametry přijímá objekt typu `CurrentOrder` obsahující informace a položky aktuální objednávky, objekt typu `Items` obsahující konfiguraci dostupných položek a textový řetězec s identifikačním číslem obrázků. Stránka se následně dělí do tří hlavních sekcí – výběr kategorií (`<div id="category-chooser">`), produkty v dané kategorii (více `<div>` elementů, viditelný pouze jeden) a košík spolu s akcemi (potvrzení a zrušení objednávky). Poté následuje spousta `<dialog>` elementů pro různé akce. Ke stránce je také připojen styl `new-order.sass` a skript `new-order.js`.

Sekce výběru kategorie je ta nejméně komplikovaná. Pomocí `@for` cyklu program pouze projde dostupné kategorie a každé z nich vytvoří element se jménem, fotkou a akcí při kliknutí. Samotnou akci změny kategorií zařizuje metoda `onCategoryClick()`, jejíž kód najdete na obrázku 80. Nejprve nastaví všechny `<div>` elementy s produkty jako neviditelné a poté zviditelní pouze ten vybraný.

```
function onCategoryClick(categoryId, element) : void {
    const currentCategoryDiv : Element = document.querySelector( selectors: `#${categoryId}-category`);

    categoriesDivs.forEach( callbackfn: d : Element => d.hidden = true);
    currentCategoryDiv.hidden = false;

    selectorDivs.forEach( callbackfn: d : Element => d.classList.remove( tokens: "active"));
    element.classList.add("active");
}
```

Obrázek 80: Kód funkce pro změnu kategorií

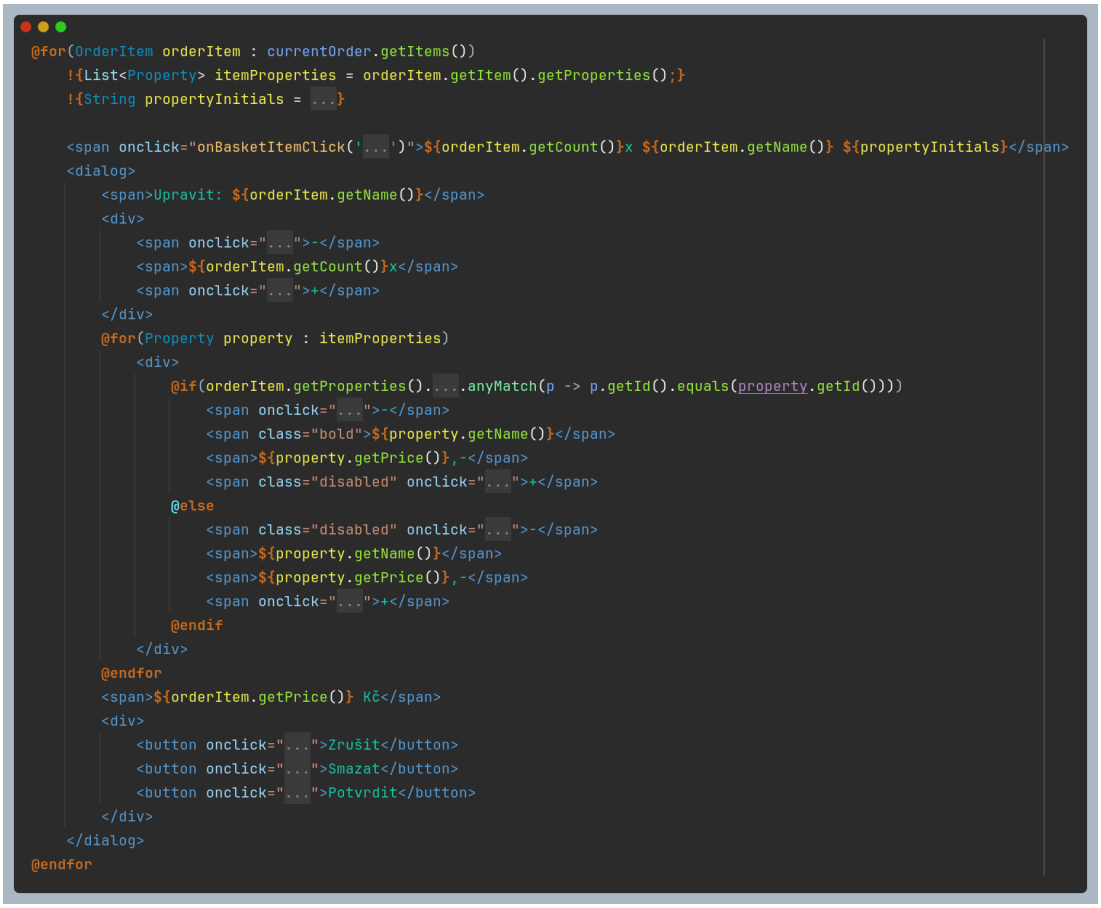
Sekce s produkty samotnými nejprve vytvoří `<div>` element pro každou kategorii a následně ho naplní produkty z dané kategorie. Zároveň každému z nich vytvoří `<dialog>` element, který se zobrazí při zvolení produktu. Zjednodušenou verzi kódu si můžete prohlédnout na obrázku 81. O aktualizaci obrazovky podle přidaných produktů se starají funkce `onItemClick()`, `onItemCountChange()`, `onPropertyAdd()`, `onPropertyRemove()`, `onItemCancel()` a nejdůležitější – `onItemAdd()`. Její zjednodušenou implementaci najdete na obrázku 82.

```
@for(Category category : items.getCategories())
    <div id="${category.getId()}" hidden="${items.getCategories().indexOf(category) != 0}">
        @for(Item item : category.getItems())
            <div id="${item.getId()}" onclick="...">
                <span>${item.getName()}</span>
                <span>${item.getPrice()},-</span>
                <img src="/resources/img/${item.getImage()}?id=${id}">
                <dialog>
                    <span>${item.getName()}</span>
                    <div>
                        <span onclick="...">-</span>
                        <span>1x</span>
                        <span onclick="...">+</span>
                    </div>
                    @for(Property property : item.getProperties())
                        <div class="${property.getId()}">
                            <span onclick="...">-</span>
                            <span>${property.getName()}</span>
                            <span>${property.getPrice()}</span>
                            <span onclick="...">+</span>
                        </div>
                    @endfor
                    <span>${item.getPrice()} Kč</span>
                    <div>
                        <button onclick="...">Zrušit</button>
                        <button onclick="onItemAdd('...')">Hotovo</button>
                    </div>
                </dialog>
            </div>
        @endfor
    </div>
@endfor
```

Obrázek 81: Zjednodušený kód šablony pro produktovou část stránky `/new-order`

Obrázek 82: Zjednodušený kód funkce pro přidání produktu do košíku

Sekce košíku je při načtení stránky naplněna produkty z aktuální objednávky. Pro každý z nich je vytvořen dialog pro úpravu nebo smazání. Kód je velmi podobný kódu pro přidání produktu, proto se jím dále nebudu zabývat. Vedle košíku je také `<div>` element `#summary`, který obsahuje dvě akce: zrušit objednávku (`onOrderCancel()`) a dokončit objednávku (`onOrderComplete()`). Po dokončení objednávky je zobrazen nejprve `confirm-dialog`, který nabídne pokladníkovi možnost zadat slevu, a po potvrzení `pay-dialog` s výpisem produktů a tlačítkem o potvrzení k zaplacení (akci na serveru vyvolá pouze `pay-dialog`, `confirm-dialog` je jen ze strany klienta). Nějaké ukázky kódu můžete najít na obrázcích 83, 84 a 85.

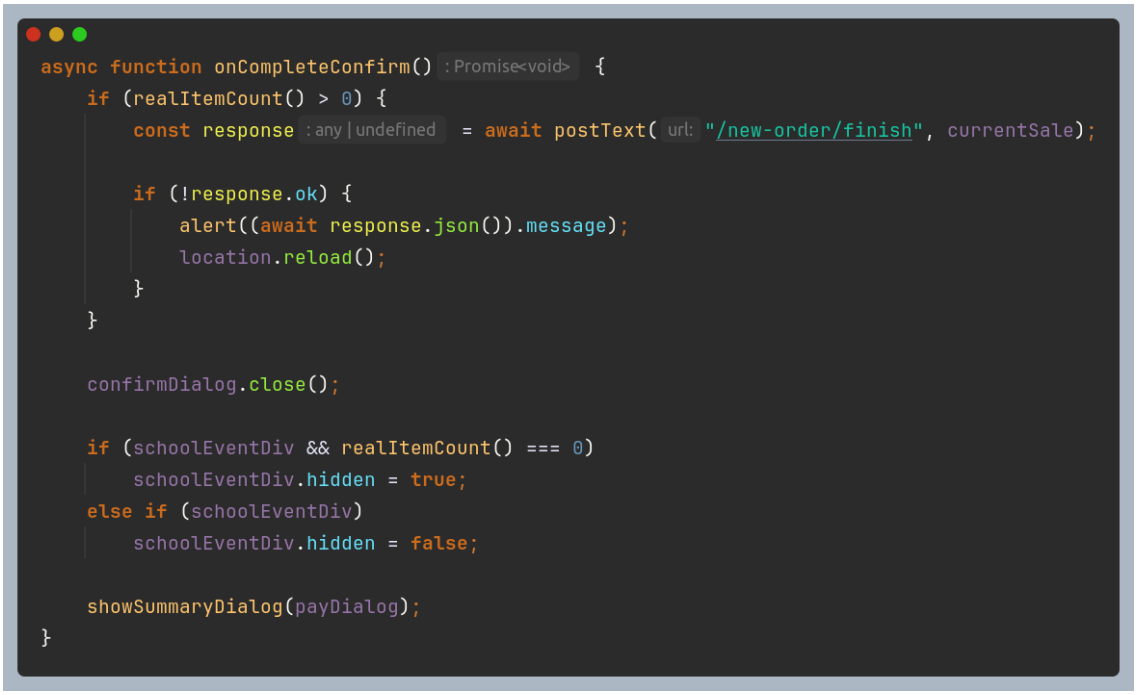


```
@for(OrderItem orderItem : currentOrder.getItems())
    !{List<Property> itemProperties = orderItem.getItem().getProperties();}
    !{String propertyInitials = ...}

    <span onclick="onBasketItemClick('...')">${orderItem.getCount()}x ${orderItem.getName()} ${propertyInitials}</span>
    <dialog>
        <span>Upravit: ${orderItem.getName()}</span>
        <div>
            <span onclick="...">-</span>
            <span>${orderItem.getCount()}x</span>
            <span onclick="...">+</span>
        </div>
        @for(Property property : itemProperties)
            <div>
                @if(orderItem.getProperties()....anyMatch(p -> p.getId().equals(property.getId())))
                    <span onclick="...">-</span>
                    <span class="bold">${property.getName()}</span>
                    <span>${property.getPrice()},-</span>
                    <span class="disabled" onclick="...">+</span>
                @else
                    <span class="disabled" onclick="...">-</span>
                    <span>${property.getName()}</span>
                    <span>${property.getPrice()},-</span>
                    <span onclick="...">+</span>
                @endif
            </div>
        @endfor
        <span>${orderItem.getPrice()} Kč</span>
        <div>
            <button onclick="...">Zrušit</button>
            <button onclick="...">Smazat</button>
            <button onclick="...">Potvrdit</button>
        </div>
    </dialog>
@endfor
```

Obrázek 83: Košíková část šablony pro stránku s tvorbou nové objednávky

```
async function onOrderComplete() : Promise<void> {
    if (currentOrder.items.length === 0) {
        zeroItemsDialog.showModal();
        return;
    }

    saleBtn.hidden = realItemCount() === 0;

    showSummaryDialog(confirmDialog);
}
```

Obrázek 84: Funkce pro zobrazení dialogu s potvrzením pro dokončení objednávky



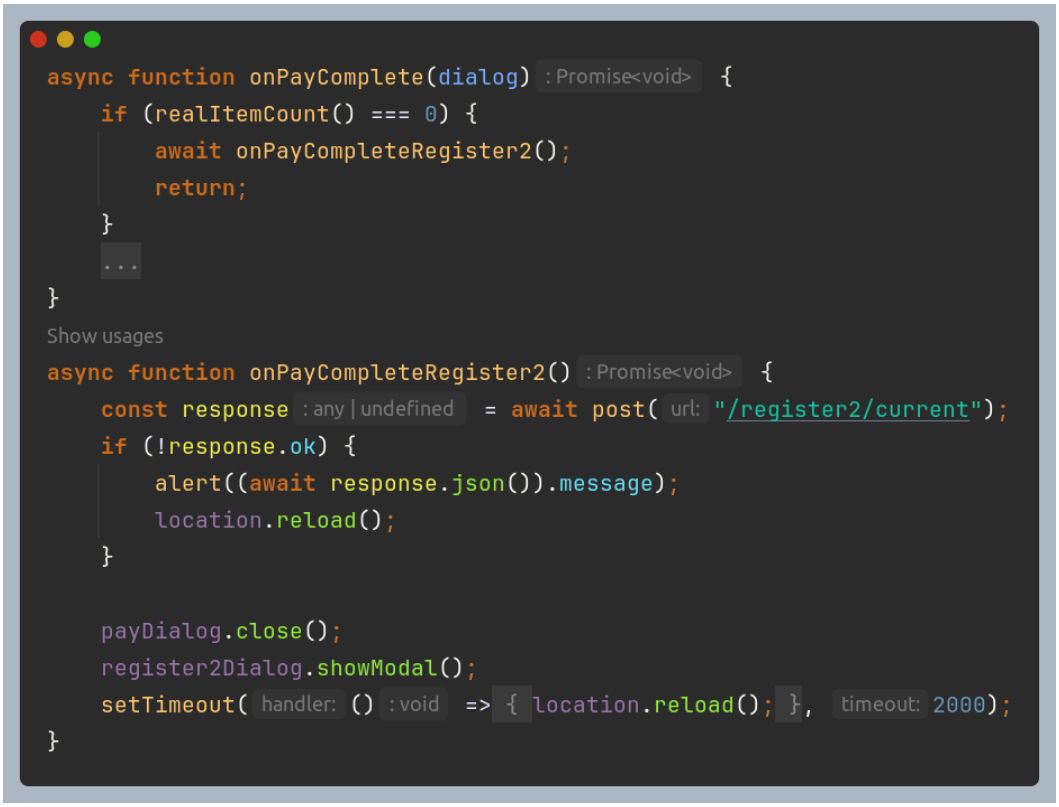
```
async function onCompleteConfirm() : Promise<void> {
    if (realItemCount() > 0) {
        const response : any | undefined = await postText( url: "/new-order/finish", currentSale);

        if (!response.ok) {
            alert((await response.json()).message);
            location.reload();
        }
    }

    confirmDialog.close();

    if (schoolEventDiv && realItemCount() === 0)
        schoolEventDiv.hidden = true;
    else if (schoolEventDiv)
        schoolEventDiv.hidden = false;

    showSummaryDialog(payDialog);
}
```

Obrázek 85: Funkce pro dokončení objednávky

Hlavní funkcí skriptu je `onLoad()`, která si nejprve vyžádá objekty od endpointu `/new-order/objects` a následně dle nich zobrazí aktivní dialogy. Ta je na obrázku 86. Za zmínku také stojí výjimečný případ, kdy objednávka obsahuje pouze kelímky na vrácení. V takovém případě se stránka zachová stejně jako `Register2` – a na něj pošle i request. Toto můžete v kódu vidět na obrázku 87.

```
async function onLoad() : Promise<void> {
    const response : any | undefined = await post( url: "/new-order/objects");
    if (!response.ok) {
        alert((await response.json()).message);
        location.reload();
        return;
    }

    const obj = await response.json();

    currentOrder = obj.first;
    items = obj.second;

    if (currentOrder.orderNumber) {
        infoDialog.classList.add("purple-bg");
        infoDialog.querySelector( selectors: ".processing-order").hidden = false;
        infoDialog.showModal();
        setTimeout( handler: () : void => location.reload(), timeout: 2000);
    } else if (currentOrder.readyToPay) {
        currentSale = currentOrder.sale;
        showSummaryDialog(payDialog);
    }
}
```

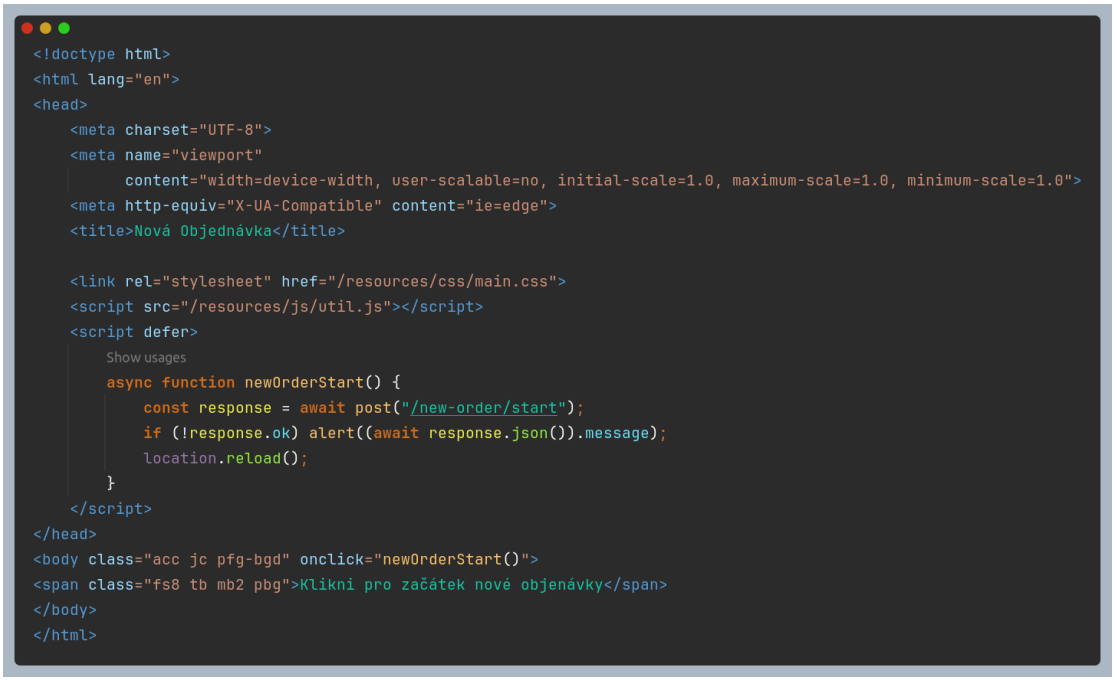
Obrázek 86: Hlavní funkce stránky /new-order spuštěná při jejím načtení



```
async function onPayComplete(dialog) : Promise<void> {
    if (realItemCount() === 0) {
        await onPayCompleteRegister2();
        return;
    }
    ...
}
Show usages
async function onPayCompleteRegister2() : Promise<void> {
    const response : any | undefined = await post( url: "/register2/current");
    if (!response.ok) {
        alert((await response.json()).message);
        location.reload();
    }

    payDialog.close();
    register2Dialog.showModal();
    setTimeout( handler: () : void => { location.reload(); }, timeout: 2000);
}
```

Obrázek 87: Funkce zobrazující schopnost stránky chovat se jako stránka /register2 při určitých podmínkách

A nakonec jen pár slov ke stránce `new-order-start.html`. Je to úvodní stránka s výzvou k vytvoření nové objednávky. Je velmi jednoduchá a celý její kód si můžete prohlédnout na obrázku 88.



```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Nová Objednávka</title>

    <link rel="stylesheet" href="/resources/css/main.css">
    <script src="/resources/js/util.js"></script>
    <script defer>
        Show usages
        async function newOrderStart() {
            const response = await post("/new-order/start");
            if (!response.ok) alert((await response.json()).message);
            location.reload();
        }
    </script>
</head>
<body class="acc jc pfg-bgd" onclick="newOrderStart()">
<span class="fs8 tb mb2 pbg">Klikni pro začátek nové objenávky</span>
</body>
</html>
```

Obrázek 88: Kód úvodní stránky při tvorbě nové objednávky

Design

Nejdůležitější při volbě designu byla samozřejmě intuitivnost uživatelského rozhraní – aby se v tom vyznal úplně každý. Rozhodli jsme se nakombinovat paletu žluté, růžové a modré pro tvorbu velmi výrazné, ale zároveň přehledné stránky. A aby elementy nesplývaly k sobě, rozhodli jsme se použít střídavé barvení pomocí CSS selektoru `nth-child()`.

Produkty jsou zobrazeny v řádcích uprostřed obrazovky a pro jednoduchost obsahuje jejich displej pouze jméno, cenu a obrázek. Při kliknutí se zobrazí dialog s možnostmi, jako je počet nebo přídavné produkty. Košík je tvořen seznamem názvů produktů a jejich množství. Po kliknutí na text se zobrazí dialog podobný tomu pro přidání, kde může pokladník provést libovolné akce. Při potvrzení objednávky se zobrazí dialog s přehledným shrnutím produktů a akcemi pro dokončení a zrušení. Obrázky 89, 90 a 91 zobrazují různé stavy finální obrazovky.

Obrázek 89: Obrazovka pro novou objednávku

Obrázek 90: Dialog pro zkompletování objednávky obrazovky /new-order

Obrázek 91: Dialog s informací o tisku a číslem objednávky stránky /new-order

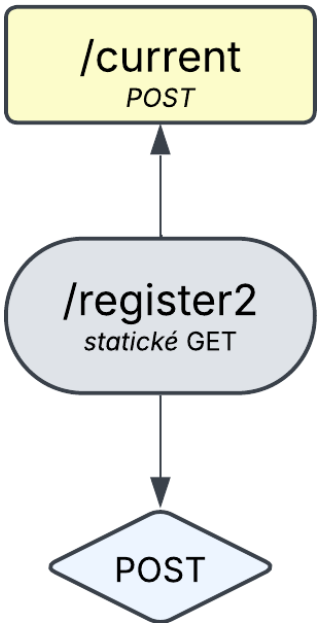
4.8.3 Obrazovka pro vracení kelímků

Celkem novým přírůstkem programu je druhá kasa pro vracení kelímků – endpoint `/register2`. Je velmi jednoduchá – jelikož není potřeba ani ukládat aktuální objednávku, ani ji dále odesílat, nemusí řešit podobné věci jako kasa normální. Jediné, na co je potřeba, je evidence vrácených kelímků do historie objednávek. Těchto kas může být neomezené množství

a ani nepotřebují ID, jelikož nemohou být propojeny s obrazovkou pro zobrazení aktuální objednávky. Samozřejmě lze v konfiguraci nastavit, které produkty jsou v ní dostupné.

Backend

Třída `Register2.kt` obsahuje pouze dvě metody, čímž dělá tento endpoint jedním z nejjednodušších. I přesto si můžete prohlédnout mapu rozvržení na obrázku 92. Můžete si jistě všimnout, že třída nemá žádnou `GET` metodu – stránka je totiž načtená staticky pomocí vlastního pluginu pro statické šablony (viz výše).



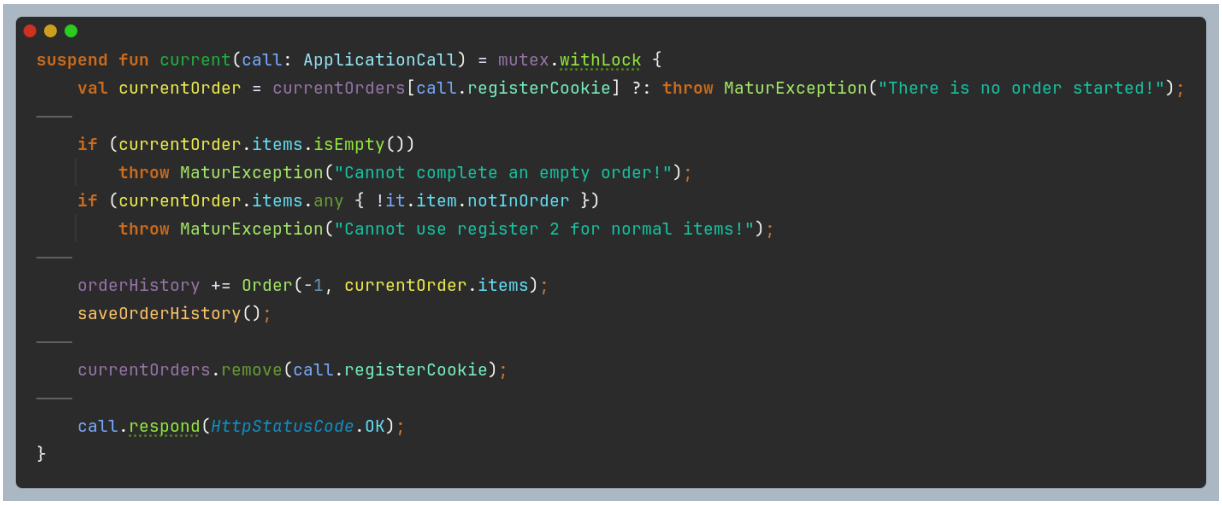
Obrázek 92: Schéma endpointu pro druhou kasu

Při dokončení a potvrzení „objednávky“ odešle klient `POST` request na adresu `/register2`. Metoda `post()` poté objednávku zkontroluje a přidá do databáze s historií objednávek. Tyto speciální objednávky se poznají tak, že mají číslo `-1`. Kód této metody je na obrázku 93.

```
override suspend fun post(call: ApplicationCall): Unit = mutex.withLock {
    val itemInfos = runCatching { call.receive<List<OrderItemInfo>>() }.getOrElse { it: Throwable
        throw MaturException("Invalid items!");
    };
    val items = itemInfos.map { it.orderItem ?: throw MaturException("Invalid items!"); };

    if (items.isEmpty())
        throw MaturException("No items found!");
    if (items.any { !it.item.notInOrder })
        throw MaturException("Invalid items!");
    if (items.hasDuplicatesBy { it.id })
        throw MaturException("Duplicate ids found!");
    if (items.any { it.count <= 0 })
        throw MaturException("Invalid count!");

    orderHistory += Order(-1, items);
    saveOrderHistory();

    call.respond(HttpStatusCode.OK);
}
```

Obrázek 93: Kód pro metodu POST od endpointu pro vracení kelímků

Pokud je však objednávka dokončena z normální kasy, jak je již zmíněno výše, je třeba aktuální rozdělanou objednávku i smazat. K tomu slouží endpoint `/register2/current`, který navíc i upraví databázi aktuálních objednávek. Implementaci této metody najdete na obrázku 94.



```
suspend fun current(call: ApplicationCall) = mutex.withLock {
    val currentOrder = currentOrders[call.registerCookie] ?: throw MaturException("There is no order started!");

    if (currentOrder.items.isEmpty())
        throw MaturException("Cannot complete an empty order!");
    if (currentOrder.items.any { !it.item.notInOrder })
        throw MaturException("Cannot use register 2 for normal items!");

    orderHistory += Order(-1, currentOrder.items);
    saveOrderHistory();

    currentOrders.remove(call.registerCookie);

    call.respond(HttpStatusCode.OK);
}
```

Obrázek 94: Kód metody pro zpracování aktuální objednávky jako objednávky pouze pro vracení kelímků

Frontend

Šablona `register2.jte` je velmi jednoduchá. Má pouze dva parametry – objekt typu `List<Item>`, který představuje kolekci dostupných produktů, a ID pro obrázky (viz výše). HTML kód obsahuje pouze jeden `<div>` element se všemi dostupnými produkty

(inicializovanými pomocí `@for` cyklu) a jeden `<div>` element s košíkem a akcemi pro zrušení nebo dokončení objednávky. Ke stránce je připojen styl `register2.sass` a skript `register2.js`. Hlavní část kódu najdete na obrázku 95.

```
<div>
    @for(Item item : items)
        <div onclick="onItemAdd('${item.getId()}', '${item.getName()}', ${item.getPrice()})">
            <span>${item.getName()}</span>
            <span>${item.getPrice()},-</span>
            <img src="/resources/img/${item.getImage()}?id=${id}">
        </div>
    @endfor
</div>
<div>
    <div id="basket">

    </div>
    <div class="summary">
        <span id="return-label">Na vrácení:</span>
        <span hidden id="pay-label">Na zaplacení:</span>
        <span id="total">0 Kč</span>
        <div class="button-chooser">
            <button onclick="onRemoveAll()">Vynulovat</button>
            <button onclick="onSubmit()">Hotovo</button>
        </div>
    </div>
</div>
```

Obrázek 95: Zjednodušená šablona pro stránku druhé kasy

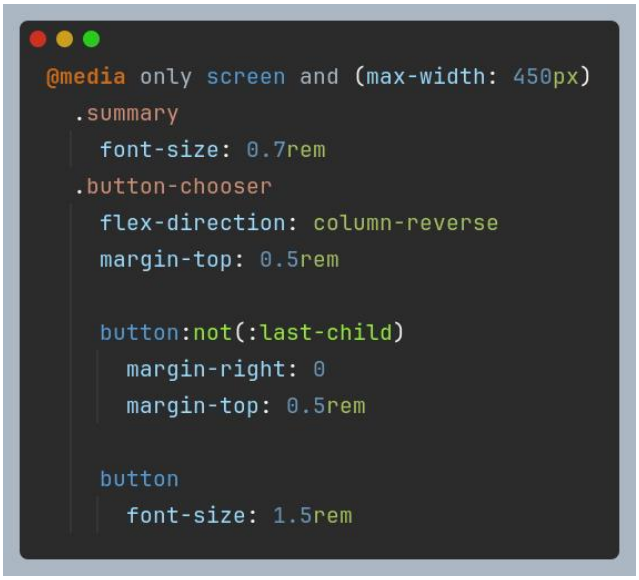
Nejdůležitější funkcí skriptu je `onItemAdd()`, která se zavolá po kliknutí na produkt a přidá produkt do košíku. Mezi další funkce patří `onItemRemove()`, která je zavolána po stisknutí tlačítka odebrání u produktu v košíku, nebo funkce `onSubmit()`, která se zavolá po potvrzení objednávky. Aktualizaci rozhraní zajišťuje funkce `reloadBasket()`, kterou mimo jiné volá `onItemAdd()` i `onItemRemove()`. Její kód si můžete prohlédnout na obrázku 96.

```
function reloadBasket() :void {
    basketDiv.innerHTML = "";
    returnLabel.hidden = true;
    payLabel.hidden = true;

    currentOrder.forEach((item :{…} ) :void => {
        const itemDiv :HTMLElement = cloneTemplate(basketItemTemplate);
        itemDiv.querySelector( selectors: ".basket-name").innerHTML = `${item.count}x ${item.name}`;
        itemDiv.querySelector( selectors: ".minus-btn").onclick = () :void => { onItemRemove(item.id); };
        basketDiv.appendChild(itemDiv);
    });

    const price = currentPrice();

    totalSpan.innerHTML = Math.abs(price) + " Kč";
    if (price < 0) returnLabel.hidden = false;
    else payLabel.hidden = false;
}
```

Obrázek 96: Metoda pro znovunačtení produktů v košíku u druhé kasy

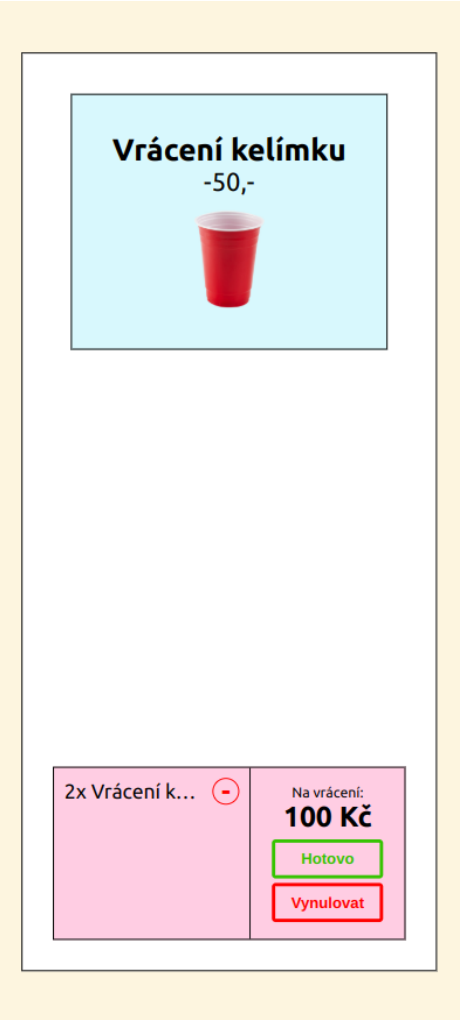
Design

Pro tuto kasu jsme se zaměřili na responzivní design, aby byla stránka dobře přizpůsobena jak pro tablety, tak pro mobilní zařízení. Toho jsme dosáhli použitím `@media` direktivy v SASS, která upravuje styly na základě velikosti obrazovky. Tento přístup můžete vidět na obrázku 97. Pro zbytek designu jsme zvolili podobný přístup jako při tvorbě objednávací obrazovky – střídavé barvy a intuitivní design. Na obrázku 98 si můžete prohlédnout konečný vzhled stránky.



```
@media only screen and (max-width: 450px)
  .summary
    font-size: 0.7rem
  .button-chooser
    flex-direction: column-reverse
    margin-top: 0.5rem

    button:not(:last-child)
      margin-right: 0
      margin-top: 0.5rem

    button
      font-size: 1.5rem
```

Obrázek 97: Responzivní stylování stránky pro vracení kelímků



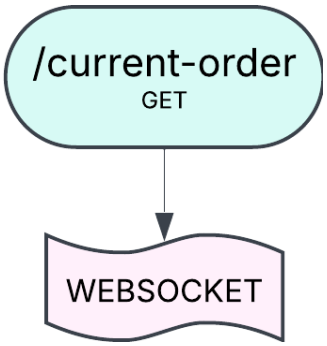
Obrázek 98: Stránka pro vracení kelímků

4.8.4 Zobrazení aktuální objednávky

Stránka s cestou `/current-order` je navržena především pro malé displeje, jako jsou mobilní telefony nebo specifické obrazovky u pokladen. Na této obrazovce zákazník může v reálném čase sledovat tvorbu své objednávky, včetně přidávání položek a změn ceny. Kromě toho zobrazuje aktuální cenu objednávky a na konci, při dokončení, také pokyn pro zaplacení. Na pár sekund se zákazníkovi zobrazí i číslo jeho objednávky, aby mohl vidět, že jeho objednávka byla úspěšně zaregistrována. Tato obrazovka vyžaduje ID objednávací obrazovky, které je připojeno, aby bylo možné sledovat a aktualizovat správnou objednávku.

Backend

Veškerou funkcionalitu tohoto endpointu zařizuje třída `CurrentOrder.kt`. Ta obsahuje pouze dvě metody – `get()` metodu pro načtení stránky a `handle()` metodu pro websocket připojení. Bylo to totiž jediné logické řešení pro implementaci požadovaných funkcí – ten, kdo odesílá data, je totiž server, ne klient. Mapu rozvržení tohoto jednoduchého endpointu vidíte na obrázku 99.



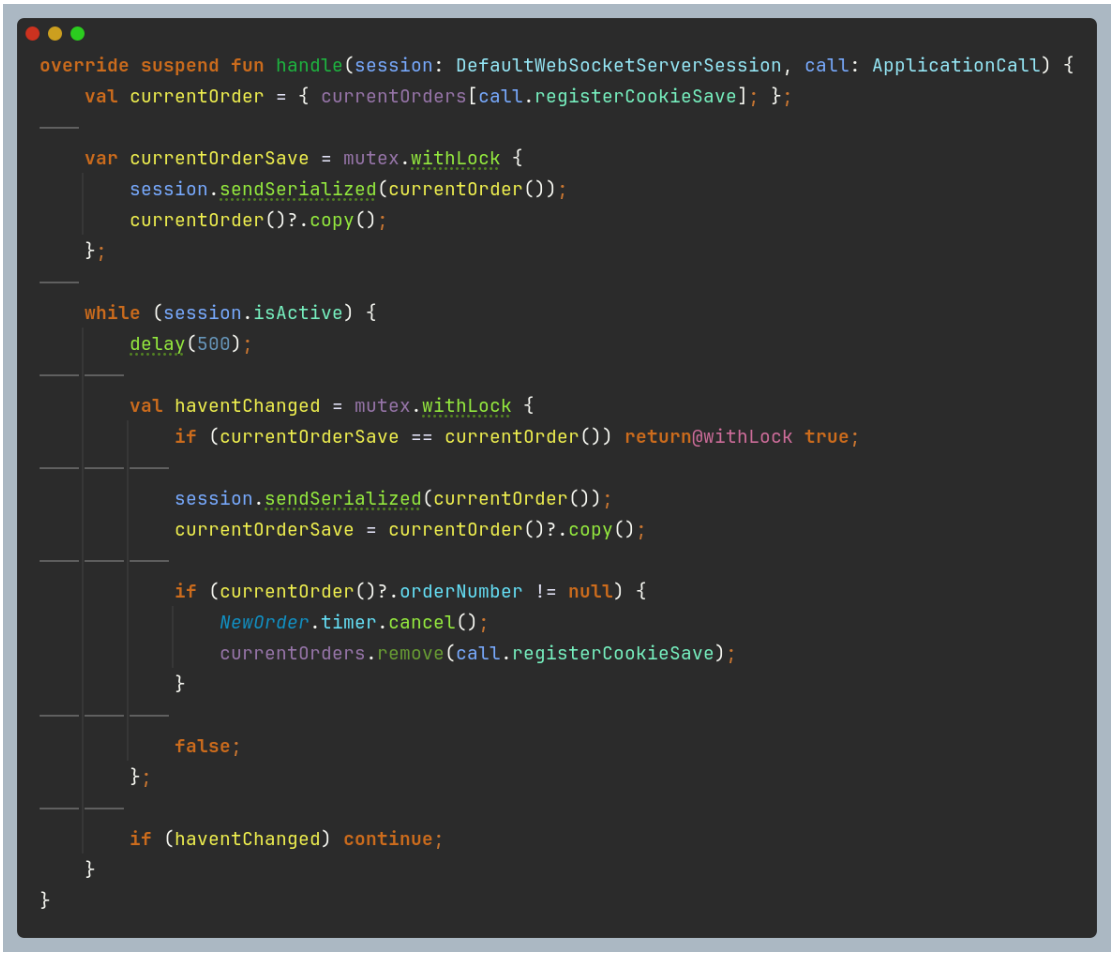
Obrázek 99: Schéma endpointu /current-order

Při načítání této stránky ve webovém prohlížeči pošle klient serveru `GET` request s query parametrem `registerId`, který obsahuje identifikační číslo kasy, k níž se má stránka připojit. Metoda `get()` nejprve zkontroluje toto ID, poté ho zkopíruje do cookie souboru a ten spolu se stránkou `current-order.html` odešle klientovi. Toto můžete vidět na obrázku 100.

```kotlin
override suspend fun get(call: ApplicationCall) {
    val registerId = call.request.queryParameters["registerId"] ?: call.registerCookie;

    if (registerId == null) {
        call.respondRedirect("/screens");
        return;
    }

    call.response.cookies.append(REGISTER_COOKIE_NAME, registerId, maxAge = 86400L);
    call.respondPage("current-order.html");
}
```

Obrázek 100: Metoda pro zpracování `GET` požadavku na stránku pro zobrazení aktuální objednávky

Ihned po spuštění stránky naváže klient se serverem websocket připojení. Server odešle aktuální objednávku ihned a poté pokaždé, když se změní. A jak bylo již zmíněno, pokud je objednávka již uzavřená a má číslo, smaže ji po odeslání z databáze aktuálních objednávek. Kód funkce `handle()` můžete vidět na obrázku 101.



```
override suspend fun handle(session: DefaultWebSocketServerSession, call: ApplicationCall) {
    val currentOrder = { currentOrders[call.registerCookieSave]; };

    var currentOrderSave = mutex.withLock {
        session.sendSerialized(currentOrder());
        currentOrder()?.copy();
    };

    while (session.isActive) {
        delay(500);

        val haventChanged = mutex.withLock {
            if (currentOrderSave == currentOrder()) return@withLock true;

            session.sendSerialized(currentOrder());
            currentOrderSave = currentOrder()?.copy();

            if (currentOrder()?.orderNumber != null) {
                NewOrder.timer.cancel();
                currentOrders.remove(call.registerCookieSave);
            }

            false;
        };

        if (haventChanged) continue;
    }
}
```

Obrázek 101: Funkce websocketového endpointu stránky /current-order

Frontend

Hlavním souborem pro frontend je `current-order.html`. Jedná se o velmi jednoduchou strukturu – nejprve obsahuje tři nadpisy, z nichž je vždy viditelný pouze jeden, ten, který je zrovna platný. Poté už je na stránce pouze `<div>` element s `id="item-container"`, kam přibývají nové produkty, `<div>` s informací o slevě, pokud je nějaká platná, a nakonec `<div>` element s celkovým součtem. Strukturu této stránky můžete vidět na obrázku 102. K souboru je poté připojen i styl `current-order.sass` a skript `current-order.js`.

Obrázek 102: Struktura stránky s aktuální objednávkou

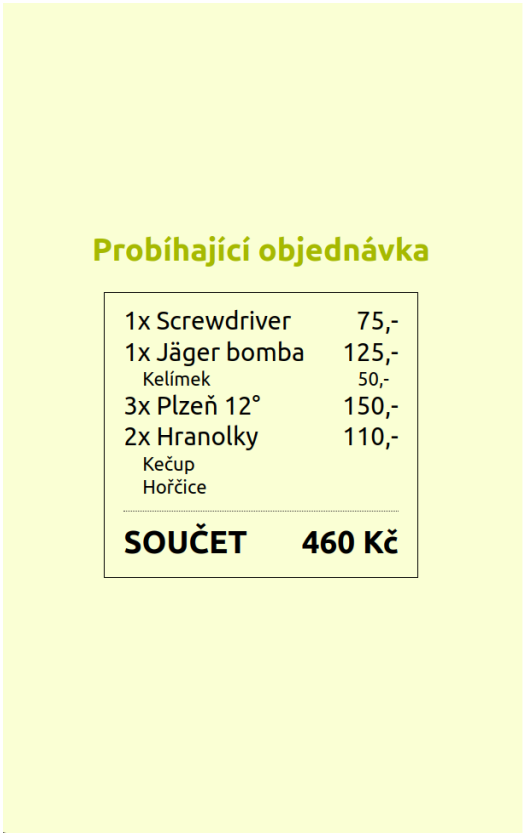
Jelikož na tomto endpointu nemusí klient vykonávat žádné akce, je Javascriptový kód velmi jednoduchý. Pomocí dříve zmíněné třídy `WS` je vytvořeno websocketové připojení a zavolána funkce `wsEvents()`, která nastaví tzv. handler funkci, jež se zavolá, jakmile obdrží prohlížeč od serveru zprávu. Ta je následně ve funkci `parseData()` zpracována a podle uchovaných dat je nastavena správná obrazovka. Funkce `orderUpdate()` se poté postará o obnovení uživatelského rozhraní. Ukázku kódu můžete vidět na obrázku 103.

```
function orderUpdate(data) :void {
    if (data.orderNumber) {
        orderNumber(data.orderNumber);
        return;
    } else if (data.readyToPay) {
        payOrderTitle.hidden = false;
        containerDiv.classList.add("pay-order");
    } else {
        orderTitle.hidden = false;
        containerDiv.classList.add("order");
    }

    itemContainer.innerHTML = "";
    data.items.forEach((item :T ) :void => {
        const itemDiv :Node = cloneTemplate(itemTemplate);
        itemDiv.querySelector( selectors: ".name").innerHTML = `${item.count}x ${item.name}`;
        itemDiv.querySelector( selectors: ".price").innerHTML = item.price + ",-";
        itemContainer.appendChild(itemDiv);

        item.properties.forEach(property :T => {...});
    });
    if (data.sale > 0) {
        saleDiv.hidden = false;
        noSalePrice.innerHTML = data.price + " Kč";
        salePercent.innerHTML = `-${data.sale}%`;
    } else saleDiv.hidden = true;

    summaryTotal.innerHTML = percent(data.price, data.sale) + " Kč";
}
```

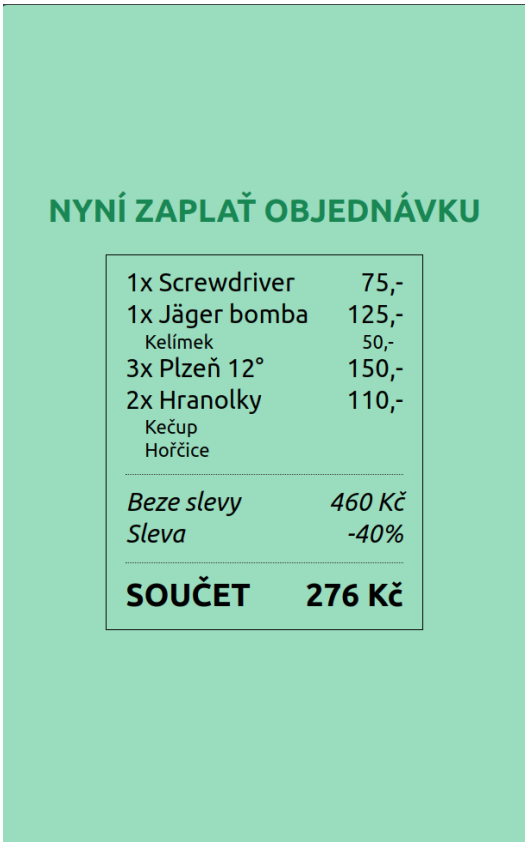
Obrázek 103: Ukázka z kódu funkce pro obnovení dat obrazovky s aktuální objednávkou

Design

Design této stránky je velmi jednoduchý. Barva pozadí indikuje stav objednávky – světle zelená značí rozdělanou objednávku, tmavě zelená objednávku připravenou k zaplacení a červená značí, že objednávka není ještě započata. Pro zobrazení produktů jsme zvolili minimalistický přístup – položky pod sebou a doplňující produkty odsazené a menším fontem. Pokud objednávka obsahuje slevu, je zobrazena italickým písmem. Ukázky můžete vidět na obrázcích 104 a 105.



Obrázek 104: Obrazovka s aktuální objednávkou



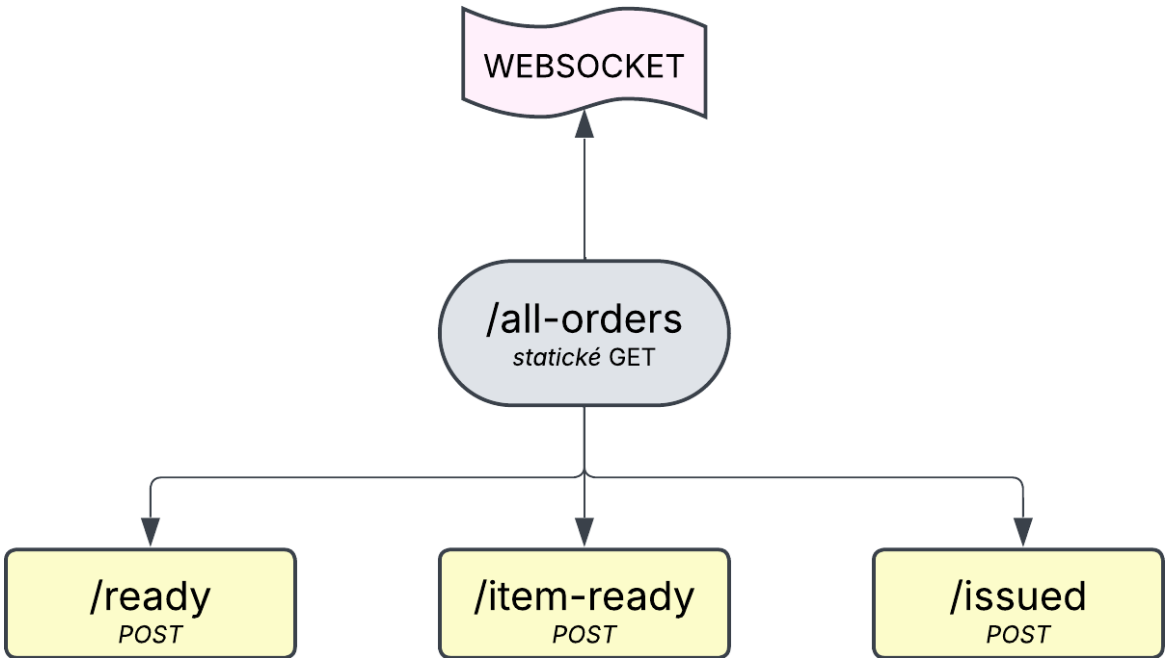
Obrázek 105: Obrazovka s aktuální objednávkou připravenou k zaplacení

4.8.5 Seznam všech objednávek

Mezi další velmi důležité obrazovky se jistě řadí i obrazovka s cestou `/all-orders`, která je určená do kuchyně a k výdeji. Jelikož je třeba aby stránka nejen přijímala data, konkrétně zprávy o nových objednávkách nebo změny v jejich stavu, ale také data odesílala. Každá objednávka, ale zároveň i každý dílčí produkt, lze označit jako hotový. To je také jeden z hlavních rozdílů mezi naším softwarem a ostatními objednávkovými systémy. Díky našemu nápadu mohou zákazníci vidět, jak na tom jejich objednávka je – jaká část položek je již hotová a jaká ještě zbývá.

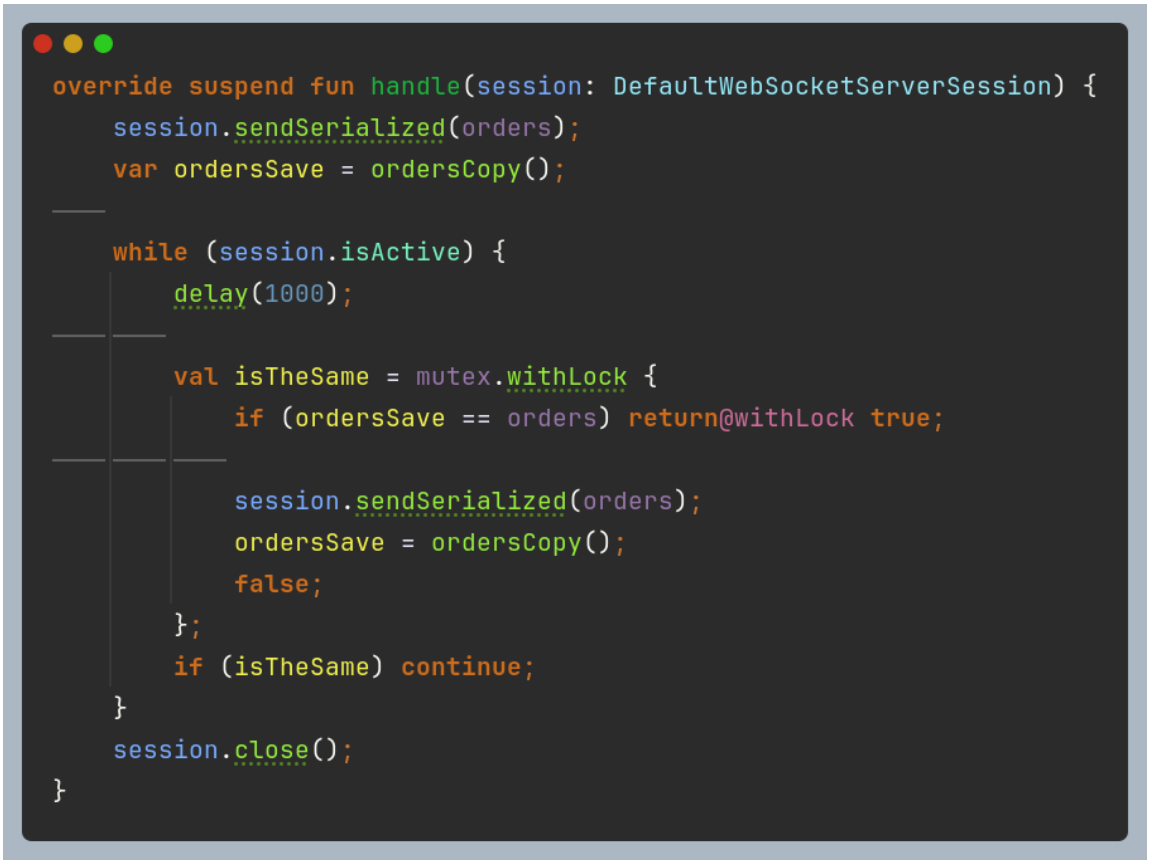
Backend

Hlavním souborem zařizujícím oboustrannou komunikaci serveru a klienta je `AllOrders.kt`. Dávalo by smysl domnívat se, že vše zařizuje jeden websocketový endpoint, avšak pro přehlednost kódu jsme se rozhodli využít websockety pouze pro odesílání dat, k přijímání slouží normální endpointy. Mapu rozvržení můžete vidět na obrázku 106. Endpoint také nedefinuje `GET` metodu, jelikož je stránka načtena jako statický soubor.



Obrázek 106: Schéma stránky se seznamem všech objednávek

Metoda `handle()` zařizuje websocketovou komunikaci. Je velmi jednoduchá a funguje podobně jako ta v `CurrentOrder.kt`. Do proměnné si udělá kopii aktuálního stavu a poté jednou za čas tuto kopii porovná s aktuálním stavem. Pokud si nejsou identické, odešle klientovi nový stav a znovu nastaví proměnnou s kopií. Kód najdete na obrázku 107.



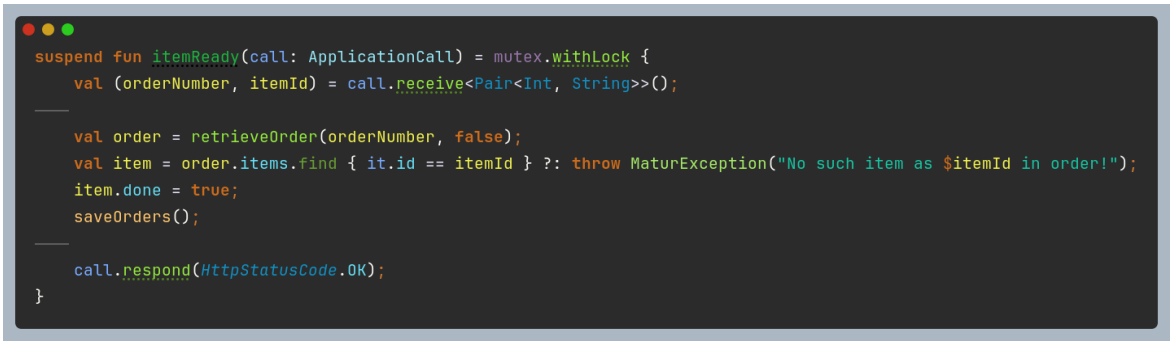
```
override suspend fun handle(session: DefaultWebSocketServerSession) {
    session.sendSerialized(orders);
    var ordersSave = ordersCopy();

    while (session.isActive) {
        delay(1000);

        val isTheSame = mutex.withLock {
            if (ordersSave == orders) return@withLock true;

            session.sendSerialized(orders);
            ordersSave = ordersCopy();
            false;
        };
        if (isTheSame) continue;
    }
    session.close();
}
```

Obrázek 107: Funkce websocketu pro odesílání změn v objednávkách obrazovce se seznamem objednávek

Metody `ready()`, `itemReady()` a `issued()` pak spravují změny ve stavu objednávek a produktů. `ready()` nastaví celou objednávku s přijatým číslem jako připravenou k vyzvednutí, `itemReady()` označí jednu konkrétní položku z objednávky jako hotovou a `issued()` vymaže objednávku z databáze pro objednávky, což značí její vyzvednutí. Kód těchto metod se nachází na obrázcích 108, 109 a 110.

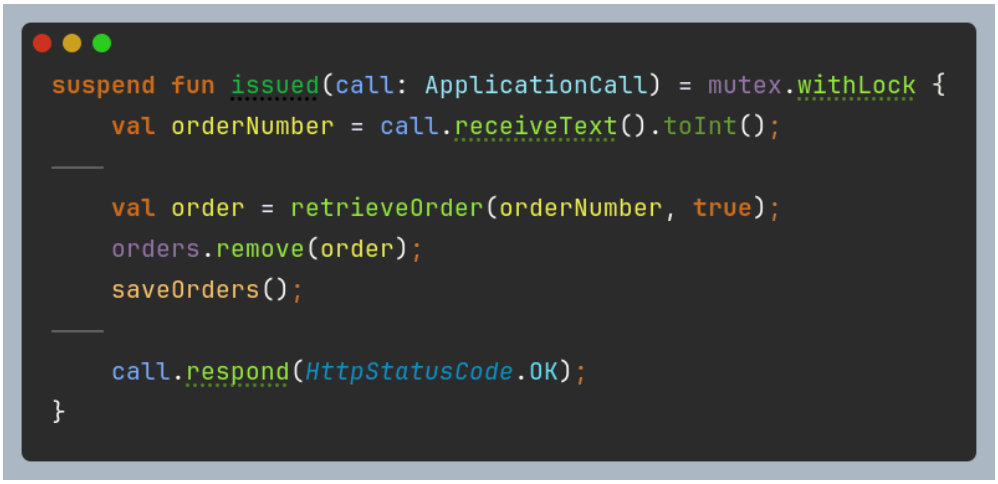
```
suspend fun ready(call: ApplicationCall) = mutex.withLock {
    val orderNumber = call.receiveText().toInt();

    val order = retrieveOrder(orderNumber, false);
    order.done = true;
    saveOrders();

    call.respond(HttpStatusCode.OK);
}
```

Obrázek 108: Endpoint obrazovky /all-orders pro označení objednávky jako připravené k výdeji



```
suspend fun itemReady(call: ApplicationCall) = mutex.withLock {
    val (orderNumber, itemId) = call.receive<Pair<Int, String>>();

    val order = retrieveOrder(orderNumber, false);
    val item = order.items.find { it.id == itemId } ?: throw MaturException("No such item as $itemId in order!");
    item.done = true;
    saveOrders();

    call.respond(HttpStatusCode.OK);
}
```

Obrázek 109: Metoda endpointu obrazovky se seznamem objednávek pro označení produktu jako hotového

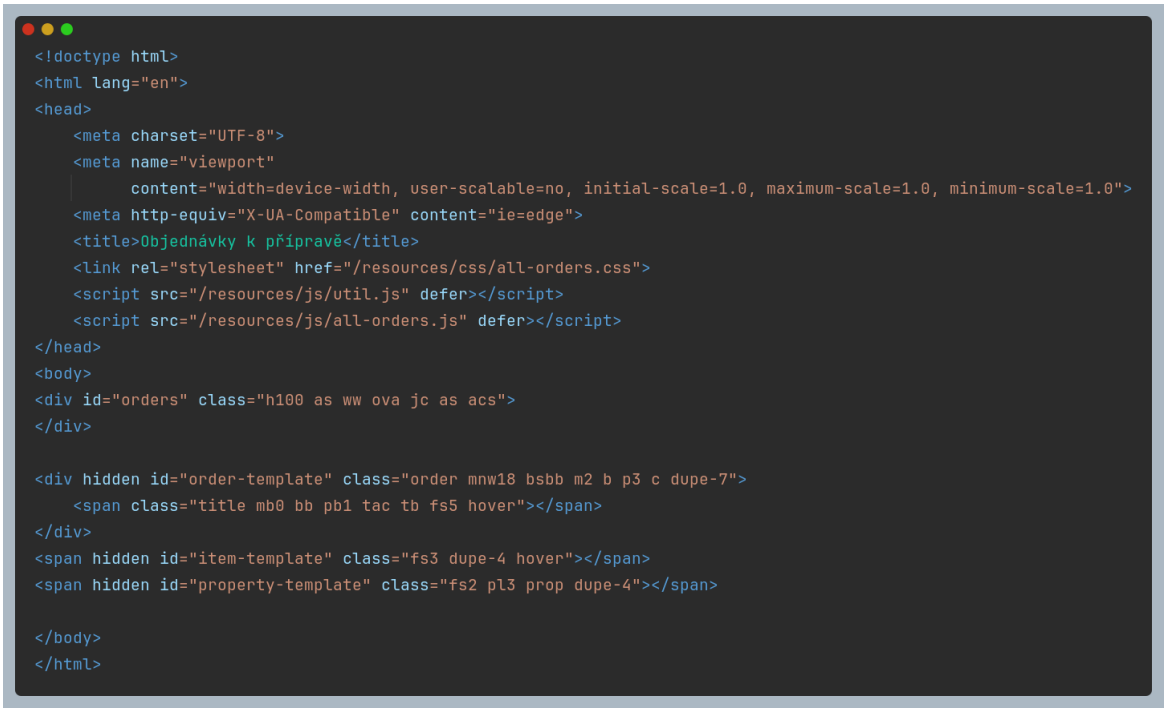


```
suspend fun issued(call: ApplicationCall) = mutex.withLock {
    val orderNumber = call.receiveText().toInt();

    val order = retrieveOrder(orderNumber, true);
    orders.remove(order);
    saveOrders();

    call.respond(HttpStatusCode.OK);
}
```

Obrázek 110: Metoda endpointu /all-orders/issued mazající objednávku

Frontend

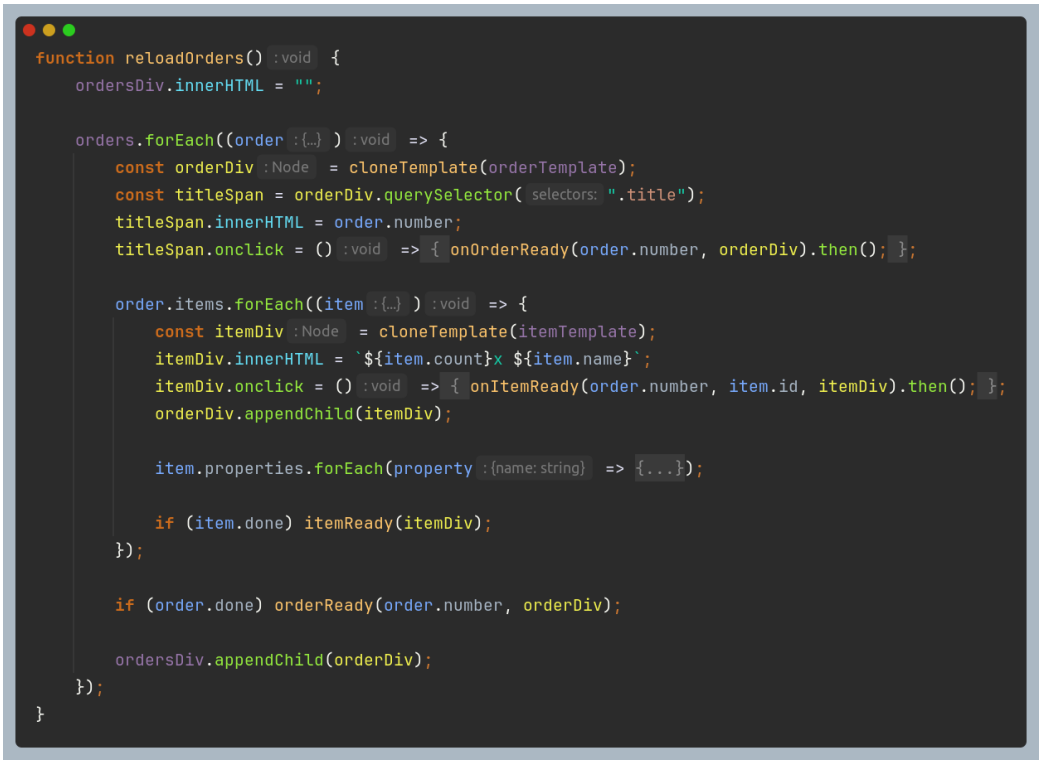
Celou stránku představuje soubor `all-orders.html`, který je velmi jednoduchý. Jedná se vlastně pouze o jediný `<div>` element s `id="orders"`, do kterého jsou ukládány objednávky. Poté obsahuje už jen tři `<div>` elementy – šablony pro objednávku, produkt a doplňující produkt, které skript kopíruje, upraví a poté duplikuje do hlavního kontejneru. Celý HTML kód najdete na obrázku 111. Ke stránce je poté přiložen styl `all-orders.sass` a skript `all-orders.js`.



```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Objednávky k přípravě</title>
    <link rel="stylesheet" href="/resources/css/all-orders.css">
    <script src="/resources/js/util.js" defer></script>
    <script src="/resources/js/all-orders.js" defer></script>
</head>
<body>
<div id="orders" class="h100 as ww ova jc as acs">
</div>

<div hidden id="order-template" class="order mnw18 bsbb m2 b p3 c dupe-7">
    <span class="title mb0 bb pb1 tac tb fs5 hover"></span>
</div>
<span hidden id="item-template" class="fs3 dupe-4 hover"></span>
<span hidden id="property-template" class="fs2 pl3 prop dupe-4"></span>

</body>
</html>
```

Obrázek 111: Kód struktury pro stránku se seznamem všech objednávek

Samotný skript připomíná skript pro endpoint `/current-order`. Nejprve ustanoví připojení k websocketu a poté nastaví metodu `reloadOrders()` jako funkci, která se zavolá při každé nové zprávě. V proměnné `orders` si uchová přijatá data a poté, pokud došlo ke změně, vykreslí celou obrazovku znovu. Zjednodušený kód této funkce najdete na obrázku 112. Dále soubor definuje tři funkce – `onItemReady()`, `onOrderReady()` a `onOrderIssued()`, které se zavolají po kliknutí na určitou část objednávky. Ty kontaktují daný endpoint a následně upraví jak obrazovku, tak interní proměnnou `orders`. Ukázku z funkce `onItemReady()` a její pomocné funkce `itemReady()` najdete na obrázku 113.



```
function reloadOrders() :void {
    ordersDiv.innerHTML = "";

    orders.forEach((order :{…} ) :void => {
        const orderDiv :Node = cloneTemplate(orderTemplate);
        const titleSpan = orderDiv.querySelector( selectors: ".title");
        titleSpan.innerHTML = order.number;
        titleSpan.onclick = () :void => { onOrderReady(order.number, orderDiv).then(); };

        order.items.forEach((item :{…} ) :void => {
            const itemDiv :Node = cloneTemplate(itemTemplate);
            itemDiv.innerHTML = `${item.count}x ${item.name}`;
            itemDiv.onclick = () :void => { onItemReady(order.number, item.id, itemDiv).then(); };
            orderDiv.appendChild(itemDiv);

            item.properties.forEach(property :{name: string} => {...});

            if (item.done) itemReady(itemDiv);
        });

        if (order.done) orderReady(order.number, orderDiv);

        ordersDiv.appendChild(orderDiv);
    });
}
```

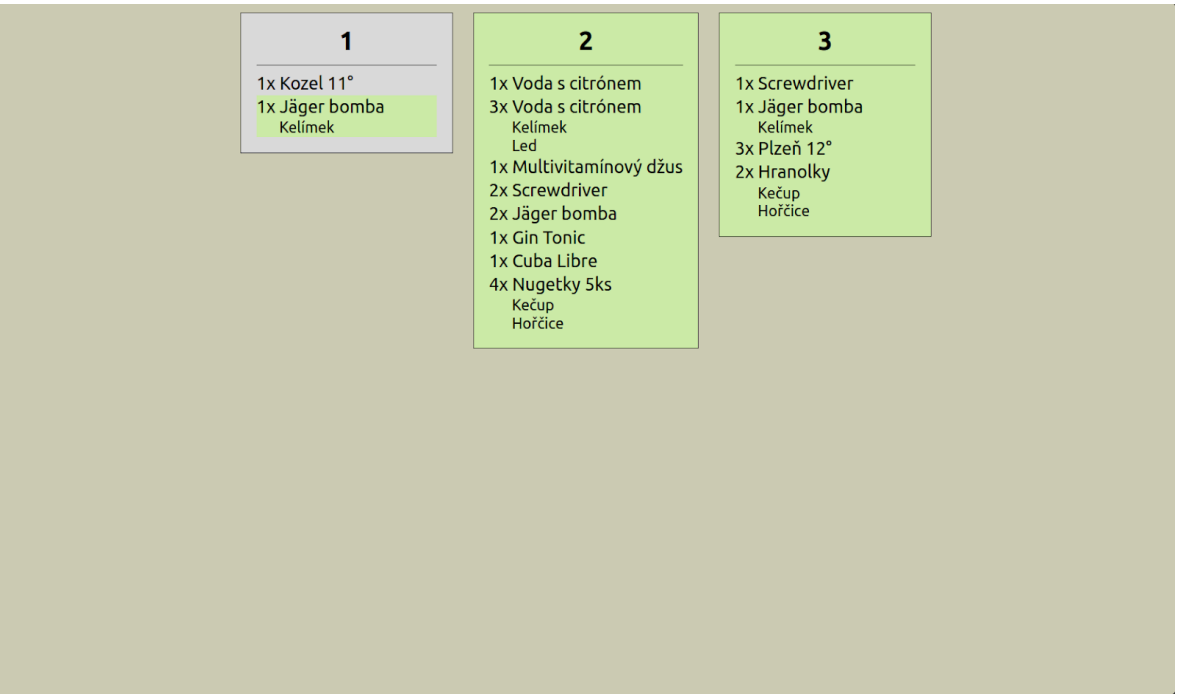
Obrázek 112: Funkce obrazovky se seznamem objednávek pro obnovení stránky

```
function itemReady(itemDiv) :void {
    itemDiv.classList.add("done");
    itemDiv.classList.remove( tokens: "hover");
    let element :Element = itemDiv.nextElementSibling;

    while (element && element.classList.contains("prop")) {
        element.classList.add("done");
        element = element.nextElementSibling;
    }

    itemDiv.onclick = null;
}
no usages
async function onItemReady(number, itemId, itemDiv) :Promise<void> {
    itemReady(itemDiv);

    const order :{…} = orders.find(i :{…} => i.number === number);
    const item :{…} = order.items.find(i :{…} => i.id === itemId);
    item.done = true;

    const response :Response | undefined = await post( url: "/all-orders/item-ready", data: {
        first: number,
        second: itemId
    });
    if (!response.ok) {
        alert((await response.json()).message);
        location.reload();
    }
}
```

Obrázek 113: Funkce obrazovky se seznamem objednávek pro označení produktu jako hotového

Design

Vzhled této stránky je velmi jednoduchý, možná i minimalistický. Na béžovém pozadí se objevují bílé objednávky s produkty v seznamu podobném souhrnu u `/new-order`. Nejprve je zobrazen počet, následně název. Doplňující položky jsou odsazené a zobrazené menším fontem. Produkt, na který uživatel klikne, se zbarví do světle zelené, stejně jako celá objednávka po kliknutí na její číslo, které je zobrazeno tučně a velkým fontem v hlavičce obdélníku. Jak toto vypadá, můžete vidět na obrázku 114.



Obrázek 114: Obrazovka se seznamem všech objednávek

4.8.6 Výdej objednávek

Poslední důležitou obrazovkou je samozřejmě i ta s cestou `/issue-screen`, kterou vidí zákazníci. Je vytvořena pro hlavní displej u výdeje. Podobně jako například v McDonald's nebo KFC zobrazuje čísla objednávek ve dvou kategoriích – „Připravujeme“ a „Hotovo“. Inovací je však indikátor, který umožňuje zákazníkovi vidět, z jaké části je objednávka hotová. O tom se více dozvíte v oddíle o designu.

Backend

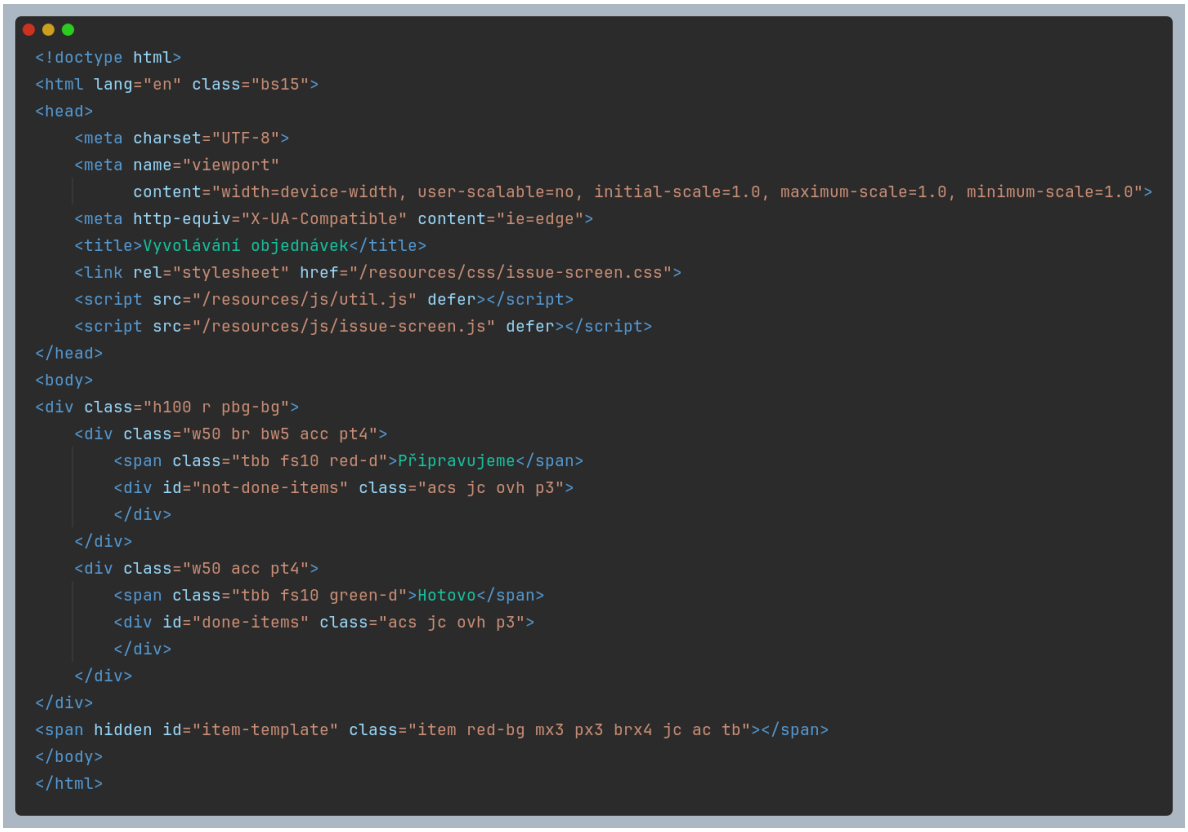
Co se backendu týče, jedná se pouze o jednu `handle()` metodu pro websocket připojení v souboru `IssueScreen.kt`. Načítání stránky je řešeno staticky. Kód funguje stejně jako v `AllOrders.kt` pomocí úschovy kopie a porovnávání dat, avšak místo odesílání všech objednávek odešle backend pouze objekt typu `OrderScreenData`. Jeho implementaci spolu s kódem metody najdete na obrázku 115. Mapa rozvržení v tomto případě potřeba není.

```
@BKMulti
class IssueScreen: BKRoute, BKWebsocket {
    override suspend fun handle(session: DefaultWebSocketServerSession) {
        mutex.withLock { session.sendSerialized(orders.screenData); };

        var save = orders.screenData;
        while (session.isActive) {
            delay(500);
            val isTheSame = mutex.withLock {
                if (save == orders.screenData) return@withLock true;

                session.sendSerialized(orders.screenData);
                save = orders.screenData;

                false;
            };

            if (isTheSame) continue;
        }
    }
}
```

Obrázek 115: Kód endpointu pro obrazovku u výdeje

Frontend

Podobně jako ostatní statické soubory je i `issue-screen.html` velmi jednoduchý. Obsahuje pouze dva `<div>` kontejnery s nadpisem kategorie a dalším kontejnerem pro čísla objednávek. Na konci souboru se nachází šablona pro takové číslo. HTML kód můžete vidět na obrázku 116. Součástí je samozřejmě i stejnojmenný styl a skript.



```
<!doctype html>
<html lang="en" class="bs15">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Vyvolávání objednávek</title>
    <link rel="stylesheet" href="/resources/css/issue-screen.css">
    <script src="/resources/js/util.js" defer></script>
    <script src="/resources/js/issue-screen.js" defer></script>
</head>
<body>
<div class="h100 r pbg-bg">
    <div class="w50 br bw5 acc pt4">
        <span class="tbb fs10 red-d">Připravujeme</span>
        <div id="not-done-items" class="acs jc ovh p3">
        </div>
    </div>
    <div class="w50 acc pt4">
        <span class="tbb fs10 green-d">Hotovo</span>
        <div id="done-items" class="acs jc ovh p3">
        </div>
    </div>
</div>
<span hidden id="item-template" class="item red-bg mx3 px3 brx4 jc ac tb"></span>
</body>
</html>
```

Obrázek 116: Struktura výdejové obrazovky

`issue-screen.js` je také velmi podobný skriptu pro aktuální objednávku. Hlavní funkcí je v tomto případě `parseScreen()`, která je mimo funkci po spuštění jedinou funkcí programu. Její jediný úkol je vykreslení objednávek na obrazovce podle přijatých dat od websocketu. Zjednodušený kód najdete na obrázku 117.

```
function parseScreen(data) :void {
    notDoneItems.innerHTML = "";
    doneItems.innerHTML = "";

    data.done.forEach((item :T ) :void => {
        const element :Node = cloneTemplate(itemTemplate);
        element.classList.add("done");
        element.innerHTML = item;
        doneItems.appendChild(element);
    });
    data.notDone.forEach((item :{first: number, second: number} ) :void => {
        const element :Node = cloneTemplate(itemTemplate);
        ...
        element.innerHTML = item.first;
        notDoneItems.appendChild(element);
    });
}
```

Obrázek 117: Funkce pro obnovu výdejové obrazovky dle dat přijatých od serveru

Design

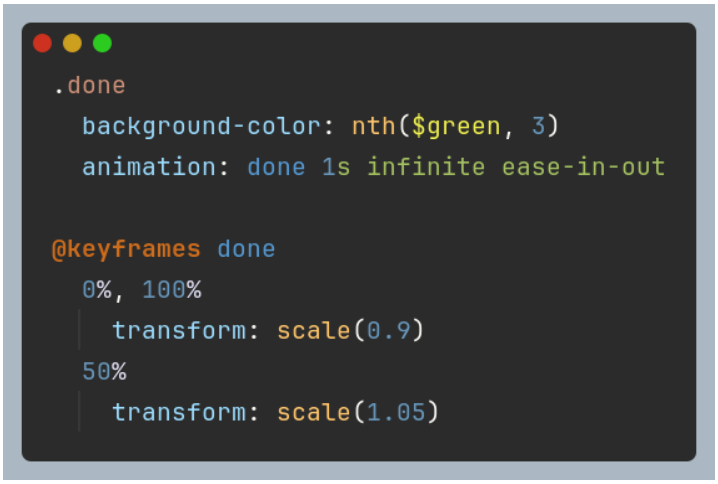
Nejzajímavější částí této obrazovky je jistě její designová stránka. Jak bylo již řečeno, dělí se na dvě sekce – připravené objednávky (zelené) a nepřipravené objednávky (červené). Pro zobrazení průběhu objednávky je použita CSS funkce `linear-gradient()`, ve které Javascriptový soubor mění proměnnou `--percent` podle toho, z kolika procent je objednávka hotová. Toto můžete vidět na obrázcích 118.

```
.item.percent
  --percent: 0%
  background-image: linear-gradient(nth($red, 2) calc(100% - var(--percent) - 5%), lightgreen calc(100% - var(--percent) + 5%))
```

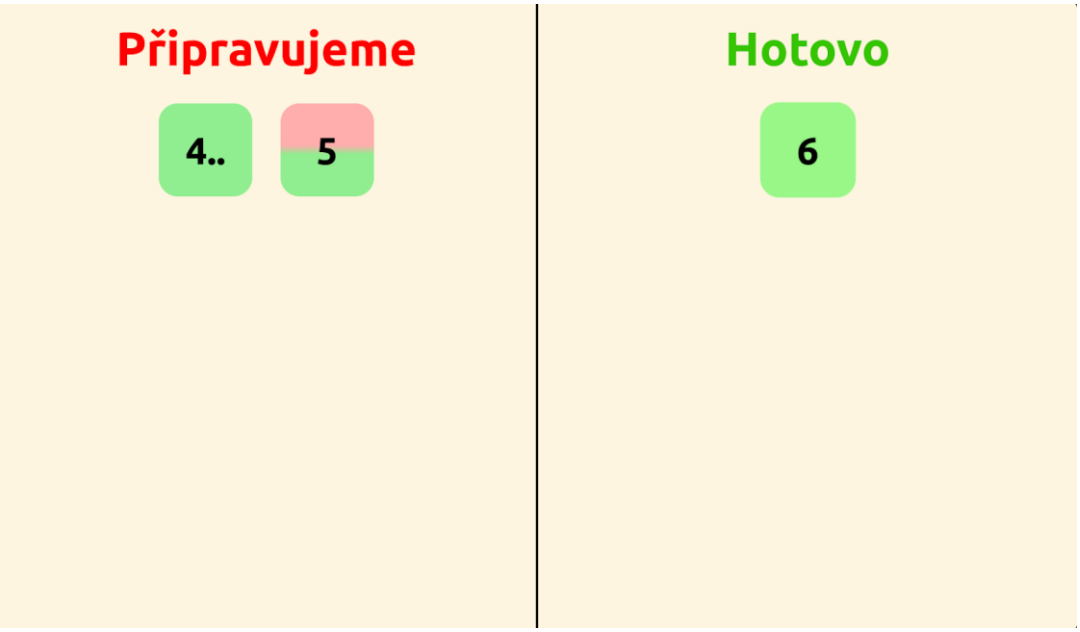
Obrázek 118: CSS kód pro zobrazení „progress baru“ u objednávek na výdejové obrazovce

Další zajímavostí jsou animace. První animace je spuštěna v případě, kdy jsou označeny za hotové všechny produkty, ale objednávka je ještě na straně „Připravujeme“. Za číslem se postupně vykreslují tečky – od žádné až po tři. To je pouze Javascriptová animace a její kód je na obrázku 119. Další animace je však už v CSS. Jakmile je objednávka hotová, její číslo „pulzuje“ – element se zvětší a následně zmenší. Kód můžete vidět na obrázku 120 a obrazovku samotnou na obrázku 121.

Obrázek 119: Javascriptová animace pro skoro dokončenou objednávku na výdejové obrazovce



Obrázek 120: CSS animace pro hotovou objednávku na výdejové obrazovce



Obrázek 121: Výdejová obrazovka

4.8.7 Informace o stavu serveru

Poslední obrazovkou dostupnou personálu je `/server-info`, která je načtená staticky ze souboru `server-info.html`. Jedná se o nejjednodušší stránku z celého programu. Každou vteřinu pošle GET request na adresu `/is-running` a pokud server neodpoví, změní text ze „Server běží“ na „Server neběží“. Celý kód je na obrázku 122 a obrazovka samotná na obrázku 123.

```
    <script>
        let runningDiv;
        let stopDiv;

        Show usages
        function serverInfoLoad() {
            runningDiv = document.querySelector("#running");
            stopDiv = document.querySelector("#stop");

            setInterval(() => {
                fetch("/is-running").then(() => {
                    runningDiv.hidden = false;
                    stopDiv.hidden = true;
                }, () => {
                    runningDiv.hidden = true;
                    stopDiv.hidden = false;
                });
            }, 1000);
        }
    </script>
</head>
<body onload="serverInfoLoad()">
<div id="running" class="sr h100 cc green-bg">
    <span class="fs8 tbb green-d">Server běží</span>
</div>
<div hidden id="stop" class="sr h100 cc red-bg">
    <span class="fs8 tbb red-d">Server neběží</span>
</div>
</body>
```

Obrázek 122: Kód obrazovky s informací o stavu serveru

Obrázek 123: Obrazovka s informací o stavu serveru

4.8.8 Rozcestník pro administrátory

Podobně jako `/screens` je `/control-panel` pouze rozcestník stránek – tentokrát však pro administrátory. Jedná se o staticky načtenou JTE šablonu, která podle spuštěných událostí zobrazí nebo skryje odkazy s ní spojené. Ukázku této funkcionality najdete na obrázku 124 a ukázku designu na obrázku 125.

```
@if(BuildingCompetitionEvent.INSTANCE.getEnabled())
    <a href="/events/contest-winner">Vyhlášení soutěže</a>
    <a href="/events/contest-winner/current">Aktuální výherce soutěže</a>
@endif
```

Obrázek 124: Ukázka z šablony rozcestníku pro administrátory s reaktivním zobrazením odkazů na jednorázové akce

Obrázek 125: Rozcestník pro administrátory

4.8.9 Historie objednávek

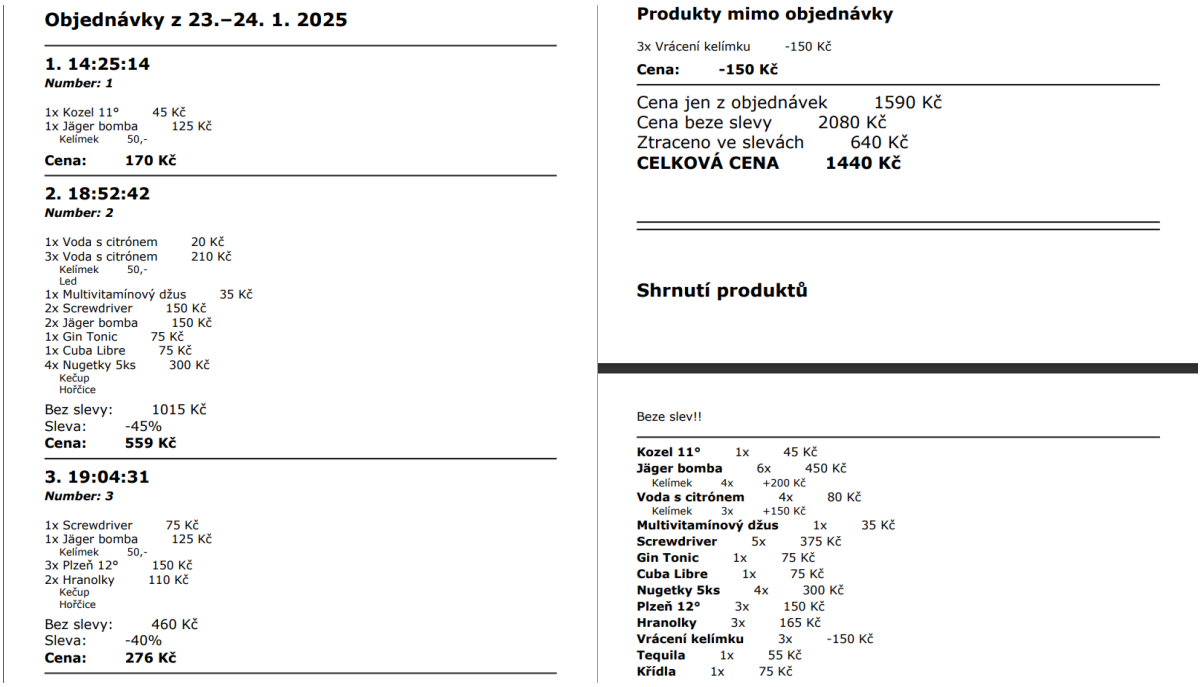
Z hlediska kódu je velmi zajímavou stránkou endpoint `/order-history`. Umožňuje uživateli zobrazit a stáhnout PDF soubor se souhrnem dne – seznam objednávek, celkovou tržbu, oblíbené produkty atd. Zároveň umožňuje i zobrazení nebo vymazání starších

automaticky vytvořených souhrnů. Šablona `order-history.jte` slouží k zobrazení veškerých uložených souborů a `order-history-file.jte` k vygenerování PDF se souhrnem. K tomu napomáhá také třída `Printer` a její metoda `toPdf()`. Její ukázku najdete na obrázku 126, samotnou stránku na obrázku 127 a ukázku z PDF souboru na obrázcích 128 a 128a.

```
fun toPdf(orders: List<Order>, date: DoubleDate = DoubleDate.now(), eventValues: Boolean = true): File {
    val dateFormat = date.string(PDF_NAME_TEMPLATE);
    val file = File(historyDir, dateFormat + PDF_NAME_SUFFIX);
    file.createNewFile();

    val renderOutput = StringOutput();
    engine.render(
        PDF_TEMPLATE,
        mapOf("orders" to orders, "date" to date, "eventValues" to eventValues),
        renderOutput
    );
    val htmlContent = Jsoup.parse(renderOutput.toString());
    htmlContent.outputSettings().syntax(Document.OutputSettings.Syntax.xml);
    htmlContent.outputSettings().charset(Charsets.UTF_8);

    val renderer = ITextRenderer();

    FONT_FILENAMES.forEach { it: String
        renderer.fontResolver.addFont(File(fontsDir, it).path, BaseFont.IDENTITY_H, BaseFont.EMBEDDED);
    };

    renderer.setDocumentFromString(htmlContent.html());
    renderer.layout();
    file.outputStream().use { renderer.createPDF(it); };

    return file;
}
```

Obrázek 126: Metoda pro převedení objednávek na PDF s historií objednávek

Obrázek 127: Stránka pro správu historie objednávek



Objednávky z 23.–24. 1. 2025

1. 14:25:14
Number: 1

1x Kozel 11° 45 Kč
1x Jäger bomba 125 Kč
Kelímek 50,-
Cena: 170 Kč

2. 18:52:42
Number: 2

1x Voda s citrónem 20 Kč
3x Voda s citrónem 210 Kč
Kelímek 50,-
Led
1x Multivitamínový džus 35 Kč
2x Screwdriver 150 Kč
2x Jäger bomba 150 Kč
1x Gin Tonic 75 Kč
1x Cuba Libre 75 Kč
4x Nugetky 5ks 300 Kč
Kečup
Hořčice
Bez slevy: 1015 Kč
Sleva: -45%
Cena: 559 Kč

3. 19:04:31
Number: 3

1x Screwdriver 75 Kč
1x Jäger bomba 125 Kč
Kelímek 50,-
3x Plzeň 12° 150 Kč
2x Hranolky 110 Kč
Kečup
Hořčice
Bez slevy: 460 Kč
Sleva: -40%
Cena: 276 Kč

Produkty mimo objednávky

3x Vrácení kelímku -150 Kč
Cena: -150 Kč

Cena jen z objednávek 1590 Kč
Cena beze slevy 2080 Kč
Ztraceno ve slevách 640 Kč
CELKOVÁ CENA 1440 Kč

Shrnutí produktů

Beze slev!!

Kozel 11° 1x 45 Kč
Jäger bomba 6x 450 Kč
Kelímek 4x +200 Kč
Voda s citrónem 4x 80 Kč
Kelímek 3x +150 Kč
Multivitamínový džus 1x 35 Kč
Screwdriver 5x 375 Kč
Gin Tonic 1x 75 Kč
Cuba Libre 1x 75 Kč
Nugetky 5ks 4x 300 Kč
Plzeň 12° 3x 150 Kč
Hranolky 3x 165 Kč
Vrácení kelímku 3x -150 Kč
Tequila 1x 55 Kč
Křídla 1x 75 Kč

Obrázek 128 a 128a: Ukázka z PDF s historií produktů

4.8.10 Konfigurace programu

Endpoint `/config` v souboru `Config.kt` slouží ke správě uložených konfigurací dostupných produktů. Šablona `config.jte` načte seznam všech konfigurací, které lze následně přejmenovat, smazat, ale hlavně i nastavit jako aktuální. Také je možné uložit aktuální konfiguraci jako novou. O to se starají dílčí endpointy `/load`, `/remove`, `/rename` a `/save`. Konfigurace je bohužel načtena pouze při startu serveru, takže po každé změně ji funkce automaticky restartuje. O vše se stará pomocná třída `Config`. Ukázku z jejího kódu najdete na obrázku 129 a design stránky na obrázku 130.

```
object Config {
    private val configDir = File(filesDir, "config");
    private val editingImagesDir = File(configDir, "images");
    private const val CONFIG_EXTENSION = "conf";
    ...
    val configs: List<String>
        get() = configDir.listFiles()?.map(File::nameWithoutExtension)!!;
    val nameRegex = Regex("[a-zA-Z0-9_\\- ]{4,15}");
    ...
    fun loadCurrent() {...}
    private fun File.findConfigFile() = listFiles { file -> file.extension == CONFIG_EXTENSION }?.firstOrNull();
    private fun load(file: File) {
        imagesDir.deleteRecursively();
        unzip(file, filesDir);
    }
    fun load(name: String) = load(File(configDir, name.configName));
    fun remove(name: String) = File(configDir, name.configName).delete();
    fun rename(name: String, newName: String) {
        File(configDir, name.configName).copyTo(File(configDir, newName.configName), true);
        remove(name);
    }
    fun save(name: String) {
        zip(filesDir, File(configDir, name.configName)) { it: File
            it.name in setOf(itemsFile.name, imagesDir.name) || it.extension in IMAGE_EXTENSIONS
        };
    }
    ...
    private fun zip(source: File, destination: File, filter: (File) -> Boolean = { true }): File {
        ZipOutputStream(destination.outputStream()).use {...};
        return destination;
    }
    private fun unzip(source: File, destination: File): File {
        destination.mkdirs();
        ZipInputStream(FileInputStream(source)).use {...};
        return destination;
    }
    ...
    private val String.configName: String get() = "$this.$CONFIG_EXTENSION";
    fun init() {...}
}
```

Obrázek 129: Zjednodušený kód třídy Config, zprostředkovávající manipulaci s konfigurací

Obrázek 130: Obrazovka pro správu konfigurací

4.8.11 Správa jednorázových událostí

Endpoint `/events` slouží ke správě již zmíněného systému jednorázových akcí. Třída `Events.kt` obsahuje pouze POST metodu, která aktivuje nebo deaktivuje událost, kterou klient specifikuje. Staticky načtená šablona `events.jte` tyto události zobrazí se správným stavem. Kód z backendu můžete vidět na obrázku 131 a design na obrázku 132.

```
@BKPath("events/{id}")
class Events: BKRoute {
    override suspend fun post(call: ApplicationCall) = mutex.withLock {
        val id = call.parameters["id"];
        val event = Event.all.find { it.id == id } ?: throw MaturException("No event with id $id!");

        event.enabled = !event.enabled;

        call.respond(HttpStatusCode.OK);
    }
}
```

Obrázek 131: Kód endpointu pro správu jednorázových akcí

Obrázek 132: Obrazovka správy jednorázových akcí

4.8.12 Správa serveru

Posledním endpointem je stránka `/server-control` – staticky načtený soubor `server-control.html`, který designem napodobuje `control-panel.jte`. Jedná se o pět tlačítek, které odesílají POST requesty na tyto metody třídy `Settings`:

- `restart()` – restart serveru
- `stop()` – ukončení serveru
- `resetOrders()` – resetování databáze objednávek
- `resetOrderHistory()` – resetování databáze s historií objednávek
- `resetBoth()` – kombinace posledních dvou

Všechny tyto metody nejprve počkají na konec tisku a poté provedou danou akci. Všechny metody kromě `stop()` také volají funkci `Server.restart()`, která je popsána v sekci o instalaci, jelikož s ní přímo souvisí. Kód vybraný z metod najdete na obrázku 133 a design stránky na obrázku 134.

```
suspend fun resetBoth(call: ApplicationCall): Unit = mutex.withLock {
    waitForPrint();

    orders.clear();
    orderHistory.clear();
    saveOrders();
    saveOrderHistory();

    Server.restart();
}
```

Obrázek 133: Funkce endpointu pro reset databází serveru

Obrázek 134: Obrazovka pro správu serveru

4.9 Připojení tiskárny

Nejnáročnějším úkolem bylo bezpochyby připojení tiskárny na účtenky k celému systému. První pokus jsme dělali s velmi levnou tiskárnou značky Xprinter. To byl však nemožný úkol – nejen že tiskárna nenabízela žádnou knihovnu pro Javu (vlastně vůbec žádnou knihovnu), takže bychom museli naprogramovat driver sami, ale zároveň ani pořádně nefungovala – zasekávala se apod. Byli jsme proto nuceni hledat další alternativu. Mezi nejdůvěryhodnější

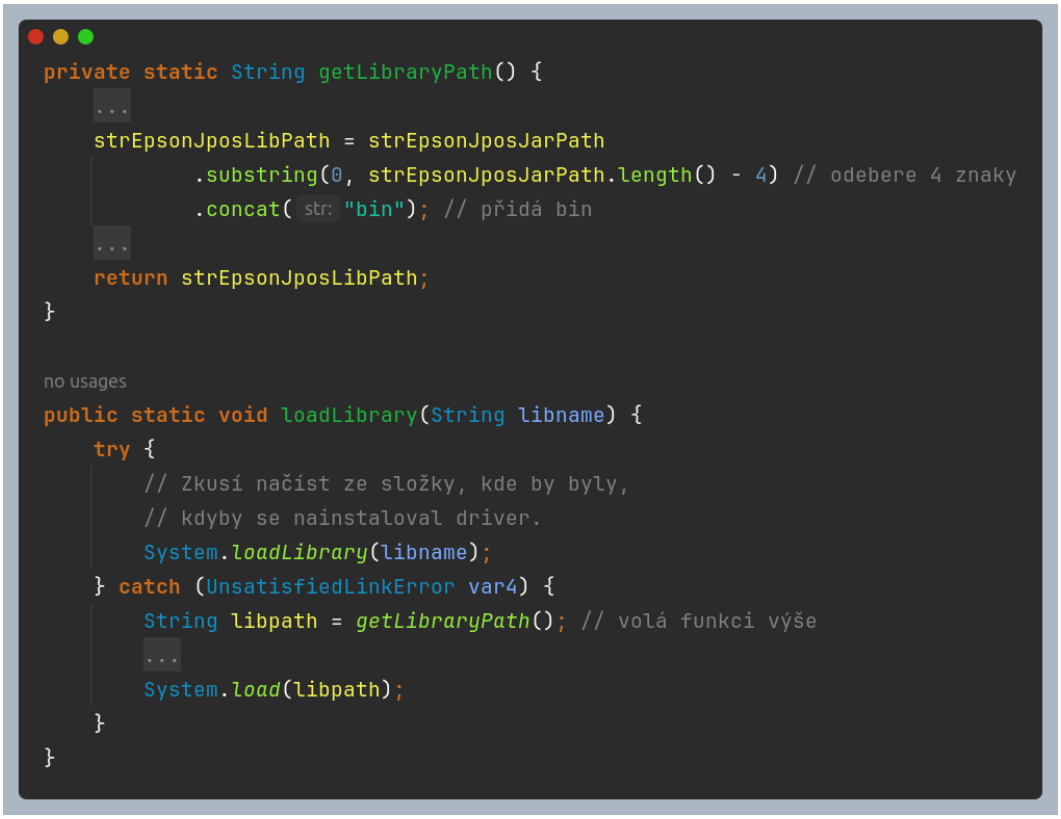
prodejce účtenkových tiskáren patří Epson, vybrali jsme od nich tedy tiskárnu s modelem T20II a dali jsme se do práce.[75]

4.9.1 Driver, problém s instalací a řešení

Pro použití tiskárny byly potřeba dvě externí knihovny – `usbio31.dll` (pro Linux `libusbio31.so`) a `epsonjpos.dll` (pro Linux `libepsonjpos.so`). Tyto knihovny jsou specifické pro platformu, což znamená, že jiný operační systém je nemůže použít. Knihovna `usbio` slouží jako komunikátor s USB portem, zatímco `epsonjpos` komunikuje přímo s tiskárnou (v podstatě driver). Jelikož je Java (a tedy i Kotlin) multiplatformní jazyk, je potřeba použít i Java knihovny (`jpos.jar` a `epson.jar`), které propojí tyto externí knihovny pomocí nástroje JNI (Java Native Interface). Při integraci těchto knihoven se vyskytly menší problémy, například že tyto knihovny nejsou součástí Maven repositáře, takže bylo nutné je přidat lokálně, což bylo složitější, než jsme čekali. To však nebylo to největší úskalí.

Hlavní problém nastal s načítáním knihoven. JNI knihovny musí být načteny pomocí funkce `System.load()`. To by měla zajistit knihovna `epson.jar`, ale problém je v tom, že standardní postup instalace tiskárny vyžaduje další spustitelný soubor, který nainstaluje potřebné externí knihovny do složky. Tu prohledává funkce `System.loadLibrary()`. V našem případě by však měl být program pouze jeden soubor, který by fungoval bez jakýchkoli externích souborů. Jak to tedy řeší `epson.jar`? Od cesty ke spustitelnému souboru (v našem případě `maturprogram.jar`) odebere čtyři znaky, přidá „bin” a odtud čerpá potřebné knihovny. Kód přímo z `JposEnv.java` si můžete prohlédnout na obrázku 135. Bohužel knihovny jsou nainstalovány ještě před spuštěním hlavní metody programu a tím pádem je nelze nainstalovat ručně. Jak tedy zařídit, aby mohl náš program tiskárnu použít bez nutnosti stažení externího driveru?

[75] KRATOCHVÍL, Adam. *Top tiskárny účtenek*. Online. RecenzeS. 2023. Dostupné z: https://www.recenzes.cz/top-tiskarny-uctenek. [cit. 2025-02-13].



```
private static String getLibraryPath() {
    ...
    strEpsonJposLibPath = strEpsonJposJarPath
            .substring(0, strEpsonJposJarPath.length() - 4) // odebere 4 znaky
            .concat( str: "bin"); // přidá bin
    ...
    return strEpsonJposLibPath;
}

no usages
public static void loadLibrary(String libname) {
    try {
        // Zkusí načíst ze složky, kde by byly,
        // kdyby se nainstaloval driver.
        System.loadLibrary(libname);
    } catch (UnsatisfiedLinkError var4) {
        String libpath = getLibraryPath(); // volá funkci výše
        ...
        System.load(libpath);
    }
}
```

Obrázek 135: Zjednodušený kód knihovny epsonjpos

Tento problém měl pouze jedno řešení – vytvořit tzv. installer. Jedná se o program, který při spuštění vytvoří složku, do které vloží potřebné soubory, a následně i spustitelný cílový program. Tento program vytvoří složku se třemi znaky (+ jeden pro znak `/`), kam umístí cílový JAR soubor, a dále složku `bin`, do které vloží potřebné externí knihovny. Pokud vám to dosud nedošlo, tento problém by vůbec neměl nastat. Způsob načítání externích knihoven Epson byl nedostatečně navržen a pro programátory, jako jsme my, to působí velmi špatně. Lepší řešení by bylo například zavést proměnnou, která by umožnila programátorovi nastavit cestu pro externí knihovny.

4.9.2 Installer podrobně

Samotný kód installeru je naštěstí velmi jednoduchý. Program podle operačního systému vybere vhodné knihovny a poté vytvoří potřebné složky, do kterých nahraje soubory ze svého interního úložiště. Toto můžete vidět na obrázku 136. Finální struktura programu by měla vypadat přibližně jako na obrázku 137.

```
const val PROGRAM_NAME = "maturprogram";
val OS_NAME: String = System.getProperty("os.name").lowercase();

val IS_LINUX = OS_NAME.startsWith("linux");
val IS_WINDOWS = OS_NAME.startsWith("windows");
val RUNNER_EXTENSION = if (IS_LINUX) "" else ".bat";
val LIB_EXTENSION = if (IS_LINUX) ".so" else ".dll";
val LIB_PREFIX = if (IS_LINUX) "lib" else "";

val programDir = File("maturprogram");
val libDir = File(programDir, "bin");
val jarDir = File(programDir, "jar");

val runnerFile = File(programDir, PROGRAM_NAME + RUNNER_EXTENSION);
val jarFile = File(jarDir, "${PROGRAM_NAME}.jar");

val libraryFiles = listOf("epsonjpos", "usbio31")
    .map { File(libDir, "$LIB_PREFIX$it$LIB_EXTENSION"); }

fun main() {
    if (!IS_WINDOWS && !IS_LINUX) throw RuntimeException("Your OS '$OS_NAME' is not supported");

    libDir.mkdirs();
    jarDir.mkdirs();

    copyResource(jarFile);
    copyResource(runnerFile);

    libraryFiles.forEach(::copyResource);
    if (IS_LINUX) Runtime.getRuntime().exec("chmod +x ${runnerFile.absolutePath}");

    println("Maturprogram was successfully installed!");
}

fun copyResource(file: File) {
    val resource =
        {}::class.java.getResource("/${file.name}") ?: throw RuntimeException("Resource ${file.name} not found!");
    file.writeBytes(resource.readBytes());
}
```

Obrázek 136: Kód installeru programu

Obrázek 137: Finální hierarchie složek vytvořená installerem

V popisu instalačního programu ještě nebyl zmíněn soubor `maturprogram.bat` (případně `maturprogram` pro Linux). V závislosti na operačním systému jde buď o Batch soubor (Windows) nebo Shell Script (Linux). Uživatel tak nemusí hledat JAR soubor ve složce a spouštět ho pomocí Javy, protože tento skript to udělá za něj. Tímto zároveň získáváme i jednu výhodu — možnost restartovat program. To je totiž velmi obtížné, protože když se program vypne, nemůže sám sebe znovu spustit. Načasování procesu a jeho oddělení od programu je složité. Proto jsme použili tzv. „exit code“ programu. Každý program může při ukončení vrátit číselnou hodnotu. V našem případě, pokud program vrátí hodnotu `3`, skript ho automaticky spustí znovu. Pokud vrátí jakoukoli jinou hodnotu, proces skončí. Kód v Shell Scriptu (pro Linux) můžete vidět na obrázku 138. Na obrázku 139 je také slíbená implementace metody `Server.restart()`. Jednoduché, že?

```
#!/usr/bin/env bash

directory=$(dirname "${BASH_SOURCE[0]}")
cd "$directory"
while
  echo "Started maturprogram"
  java -jar jar/maturprogram.jar $@
  exitCode=$?
  [[ $exitCode -eq 3 ]]
do true; done
```

Obrázek 138: Shell script pro spuštění programu

```
fun restart(): Nothing {
    Runtime.getRuntime().halt(3);
    exitProcess(3);
}
```

Obrázek 139: Funkce pro restartování serveru

4.9.3 Operace s tiskárnou

Veškerou komunikaci s tiskárnou zařizuje objekt `Printer`. Ta využívá instance třídy `POSPrinter`, která vyjadřuje tiskárnu samotnou. Už při spuštění programu je zavolána metoda `Printer.init()`, která zkontroluje připojení tiskárny a nastaví potřebné hodnoty. Jejím vstupním parametrem je proměnná typu `Boolean` (ukládá hodnoty `true` = pravda, `false` = nepravda), která určuje, zda má program kontrolovat připojení k tiskárně . Dále tato metoda překopíruje důležité soubory z interního úložiště, jako jsou obrázky s čísly na hlavičku účtenek, konfigurace pro Jpos, nebo fonty potřebné pro tvorbu PDF souborů (třída `Printer` totiž také zařizuje tvorbu souhrnů prodeje). Na závěr metoda zkontroluje připojení k tiskárně. Pokud tiskárna není připojena a uživatel potvrdí její existenci, program změní hodnotu `MODE` v souboru `/lib/udev/rules.d/50-udev-default.rules`, což umožní komunikaci s tiskárnou bez administrátorských práv. Kód metody `init()` je na obrázku 140.

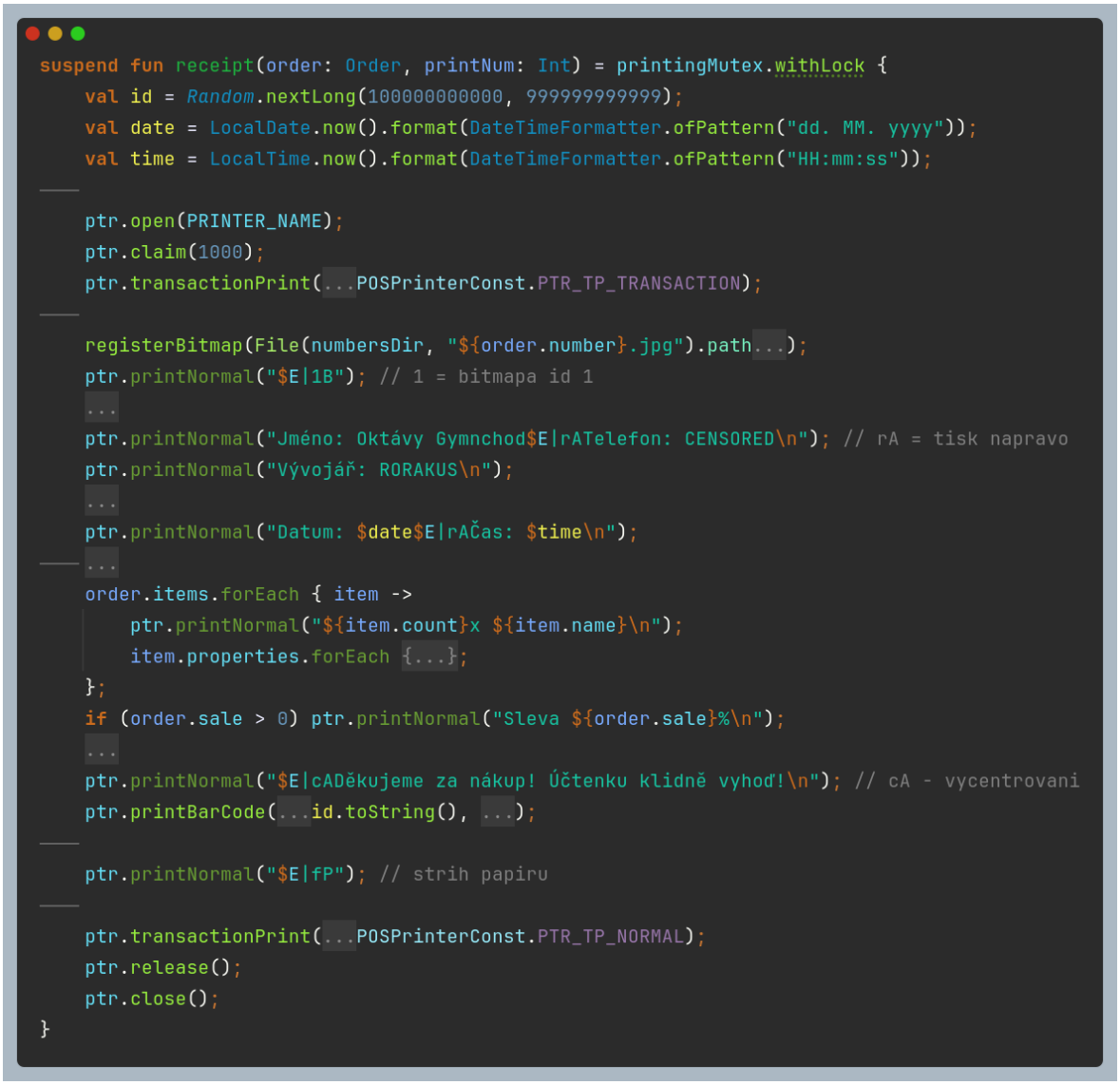
```
fun init(checkConnection: Boolean) {
    if (!isLinux && !isWindows) throw Error("Unsupported OS!");

    numbersDir.mkdirs();
    fontsDir.mkdirs();
    historyDir.mkdirs();

    if (!jposFile.exists())
        copyResource("/printer/${jposFile.name}", jposFile);

    for (i in 1 ≤ .. ≤ 100) {
        val file = File(numbersDir, "$i.jpg");
        if (file.exists()) continue;

        copyResource("/printer/numbers/$i.jpg", file);
    }

    FONT_FILENAMES.forEach { copyResource("/printer/fonts/$it", File(fontsDir, it)) };

    System.setProperty(JposPropertiesConst.JPOS_POPULATOR_FILE_PROP_NAME, jposFile.path);

    if (checkConnection && isLinux && !connected) linuxSudo();
}
```

Obrázek 140: Metoda pro inicializaci tiskárny

Dalším důležitým prvkem třídy je proměnná `connected`, která definuje tzv. „getter“ (funkci, která se zavolá při získání hodnoty této proměnné). V podstatě nejde ani o proměnnou, spíše o funkci. Tento getter naváže připojení s tiskárnou. Pokud připojení selže, vrátí `false`; pokud připojení proběhne úspěšně, odpojí se od tiskárny a vrátí `true`. Kód této funkce můžete vidět na obrázku 141.

Obrázek 141: „Property“ třídy tiskárny pro zjištění stavu připojení

Nejdůležitější je ale samozřejmě metoda `Printer.receipt(order: Order, printNum: Int)`, která vytiskne účtenku k dané objednávce. Parametr `printNum` určuje číslo objednávky, která se tiskne, z důvodu identifikace při mnohonásobném tisku, ale o tom bude řeč později. Funkce nejprve pomocí metod `POSPrinter#open()` a `POSPrinter#claim()` naváže spojení s tiskárnou. Poté pomocí `POSPrinter#transactionPrint()` nastaví jednotný tisk – nebude se tisknout jeden řádek po druhém, ale celá účtenka najednou. Následně registruje bitmapu (obrázek) čísla účtenky pomocí `POSPrinter#setBitmap()`, kterou rovnou vytiskne s `POSPrinter#printNormal()`. Díky této funkci pak řádek po řádku napíše celý text pro účtenku, vytiskne náhodný bar kód pomocí `POSPrinter#printBarcode()` a pomocí znaku `\u001b|fP` automaticky uřízne papír. Nakonec tiskárnu s `POSPrinter#release()` a `POSPrinter#close()` uzavře. Velmi zjednodušený kód metody najdete na obrázku 142. Proměnná `E` obsahuje znak `\u001b`, který se používá ke speciálním akcím tiskárny. Proměnná `ptr` obsahuje instanci třídy `POSPrinter`.



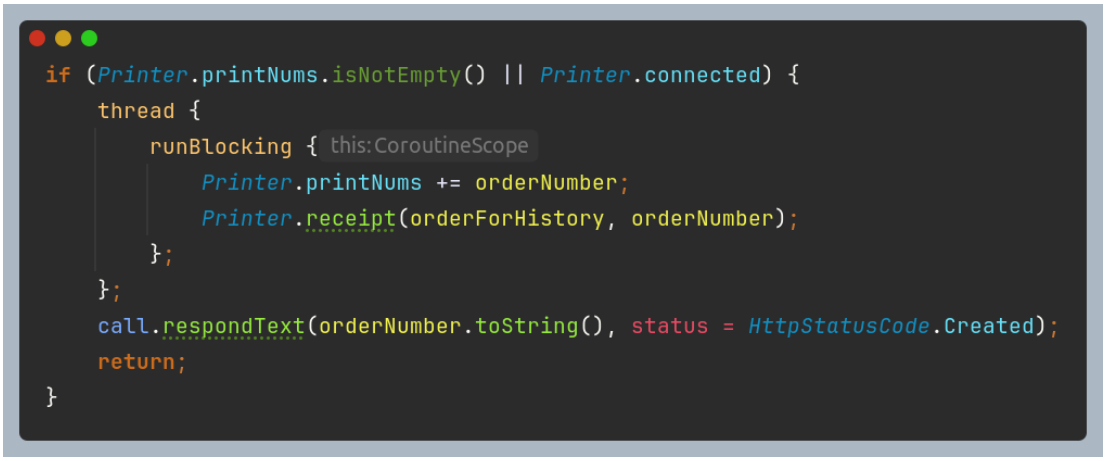
```
suspend fun receipt(order: Order, printNum: Int) = printingMutex.withLock {
    val id = Random.nextLong(100000000000, 999999999999);
    val date = LocalDate.now().format(DateTimeFormatter.ofPattern("dd. MM. yyyy"));
    val time = LocalTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss"));

    ptr.open(PRINTER_NAME);
    ptr.claim(1000);
    ptr.transactionPrint(...POSPrinterConst.PTR_TP_TRANSACTION);

    registerBitmap(File(numbersDir, "${order.number}.jpg").path...);
    ptr.printNormal("$E|1B"); // 1 = bitmapa id 1
    ...
    ptr.printNormal("Jméno: Oktávy Gymnchod$E|rATelefon: CENSORED\n"); // rA = tisk napravo
    ptr.printNormal("Vývojář: RORAKUS\n");
    ...
    ptr.printNormal("Datum: $date$E|rAČas: $time\n");
    ...
    order.items.forEach { item ->
        ptr.printNormal("${item.count}x ${item.name}\n");
        item.properties.forEach {...};
    };
    if (order.sale > 0) ptr.printNormal("Sleva ${order.sale}%\n");
    ...
    ptr.printNormal("$E|cADěkujeme za nákup! Účtenku klidně vyhoď!\n"); // cA - vycentrovani
    ptr.printBarCode(...id.toString(), ...);

    ptr.printNormal("$E|fP"); // strih papiru

    ptr.transactionPrint(...POSPrinterConst.PTR_TP_NORMAL);
    ptr.release();
    ptr.close();
}
```

Obrázek 142: Velmi zjednodušená metoda pro tisk účtenky

4.9.4 Tisk z více kas zároveň

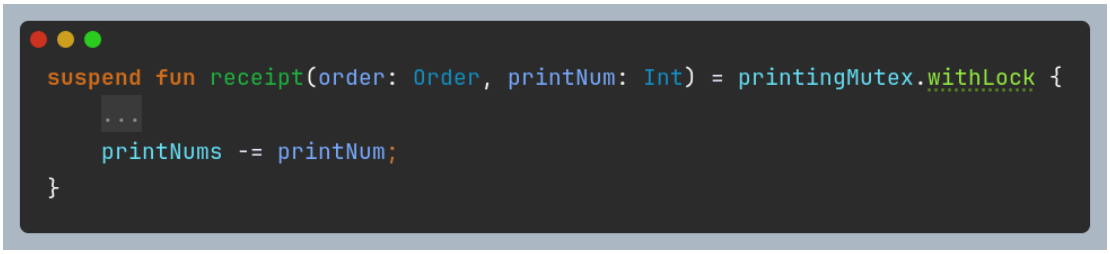
Poslední z problémů s tiskárnou nastal při implementaci neomezeného počtu obrazovek u kasy. Jak má pokladník poznat, která z vytisknutých účtenek patří jeho zákazníkovi? Lze vůbec tisknout více účtenek najednou? Jak bude fungovat endpoint `/is-printing` s více tisknutími zároveň? Naštěstí se našlo velmi jednoduché řešení. Proměnná `Printer.printNums` typu `MutableList<Int>`, která bude uchovávat ID všech objednávek, které zrovna tisknou. Metoda `NewOrder.complete()` před tiskem přidá číslo do tohoto listu a poté ho předá jako parametr metodě `Printer.receipt()`. Ta po dokončení tisku toto číslo z listu odebere. A metoda `/is-printing` ho jednoduše z cookie souboru získá a pokud je obsaženo v `printNums`, tak je tisk stále v průběhu. Metoda `complete()` také odešle toto číslo klientovi, tudíž pokladník uvidí, jaká účtenka zákazníkovi

patří. Konstanta `printingMutex` typu `Mutex` a její metoda `withLock()` zařídí, že se v jednu chvíli bude tisknout pouze jedna účtenka. Ukázku kódu najdete na obrázcích 143 a 144.



```
if (Printer.printNums.isNotEmpty() || Printer.connected) {
    thread {
        runBlocking { this:CoroutineScope
            Printer.printNums += orderNumber;
            Printer.receipt(orderForHistory, orderNumber);
        };
    };
    call.respondText(orderNumber.toString(), status = HttpStatusCode.Created);
    return;
}
```

Obrázek 143: Komunikace s tiskárnou v endpointu `/new-order/complete`



```
suspend fun receipt(order: Order, printNum: Int) = printingMutex.withLock {
    ...
    printNums -= printNum;
}
```

Obrázek 144: Odebrání čísla tisku ze seznamu aktuálně tisknoucích se účtenek na konci funkce `receipt()`

4.10 Program pro úpravu konfigurace

Vzhledem k tomu, že konfigurace musí být upravována v textovém editoru a ve formátu JSON, tak jsme se rozhodli vytvořit externí program, který bude sloužit jako jednoduchá možnost úpravy v grafickém rozhraní. Je to tedy další full-stack projekt - server v kotlinu, který dokáže generovat vytvořené a upravené konfigurace a klient v HTML, CSS a Javascriptu – rozhraní, ve kterém lze klientem konfigurace upravit.

4.10.1 Backend

Celý serverový kód se nachází v souboru `Main.kt`. Tento soubor, podobně jako hlavní program, inicializuje Ktor server a instaluje potřebné pluginy. Slouží k tomu, aby se z položek, které obdrží od klienta, mohly generovat JSON soubory. Hlavní proměnnou je `configItems`, která obsahuje položky konfigurace upravované v daný moment. Dále definuje sedm endpointů:

- `/` – indexový endpoint, který podle stavu úprav načte buď stránku `select.html` (pokud není upravován žádný soubor), nebo `edit.html`.
- `/items` – odešle klientovi všechny položky aktuální konfigurace.

- `/upload` – pokud chce uživatel upravit už existující konfiguraci na svém počítači, tento endpoint ji dokáže načíst a zahajuje úpravu.
- `/new` – umožní uživateli začít vytvářet úplně novou konfiguraci.
- `/save` – uloží položky aktuálně upravované konfigurace do proměnné `configItems`.
- `/close` – zruší úpravy konfigurace a vymaže položky z proměnné.
- `/download` – odešle klientovi konfiguraci, která je dostupná ke stažení.

Ukázku kódu můžete vidět na obrázku 145.

```
post("/save") { this: PipelineContext<Unit, ApplicationCall> }
    if (configItems == null) throw ConfigException("You have to start editing first!");

    val items = runCatching { call.receive<Items>(); }
        .getOrElse { throw ConfigException("Invalid JSON: ${it.message}."); };
    configItems = items;

    call.respond(HttpStatusCode.OK);
};

post("/close") { this: PipelineContext<Unit, ApplicationCall> }
    if (configItems == null) throw ConfigException("You have to start editing first!");

    configItems = null;
    configFileName = DEFAULT_CONFIG_NAME;

    call.respond(HttpStatusCode.OK);
};

post("/download") { this: PipelineContext<Unit, ApplicationCall> }
    if (configItems == null) throw ConfigException("You have to start editing first!");

    call.response.header(
        HttpHeaders.ContentDisposition, ContentDisposition.Attachment.withParameter(
            ContentDisposition.Parameters.FileName, configFileName
        ).toString()
    );

    call.respondBytes(json.encodeToString(configItems).toByteArray(), contentType = ContentType.Text.Plain);
}
```

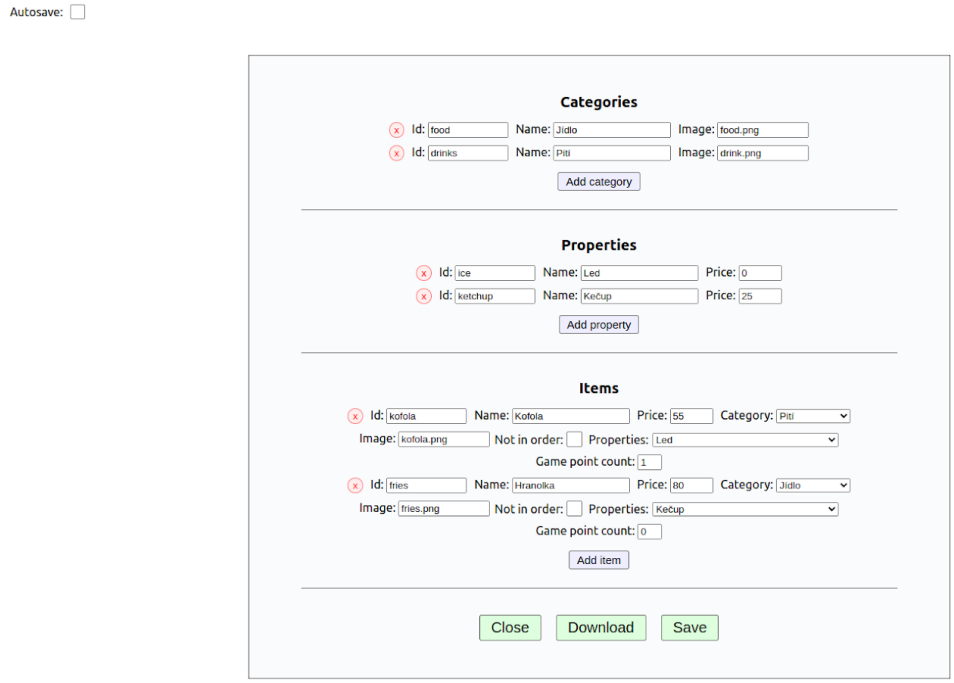
Obrázek 145: Ukázka kódu pro konfigurační program

4.10.2 Frontend

Frontend se skládá ze dvou již zmiňovaných stránek: `select.html` a `edit.html`. V této části se však budeme věnovat pouze stránce edit.html. HTML struktura této stránky je vcelku jednoduchá – obsahuje tři prázdné `<div>` kontejnery: jeden pro úpravy kategorií, další pro úpravy doplňkových položek a poslední pro úpravy samotných produktů. Dále stránka nabízí tři akce: zavřít, stáhnout a uložit. Při načtení stránky Javascript automaticky získá z endpointu `/items` všechny aktuální produkty, které následně pomocí funkce `renderDOM()` a mnoha dalších pomocných funkcí zobrazí uživateli.

Uživatel má poté u každé položky na výběr z několika akcí – může ji smazat, upravit jakoukoliv její vlastnost nebo vytvořit novou v té samé kategorii. Celý program je vysoce reaktivní, tudíž například přejmenování doplňkové položky okamžitě změní text výběru doplňkových položek u všech produktů.

Uživatel si může svou práci v průběhu ukládat, takže při obnovení stránky nepřijde o své úkony. Během práce je také možné konfiguraci kdykoli celou stáhnout a to ve formátu JSON prostřednictvím endpointu `/download`. Program zároveň obsahuje funkci „auto-save“, která spouští automatické ukládání po každé změně. Ukázku z rozhraní programu můžete vidět na obrázku 146.



Obrázek 146: Grafické rozhraní konfiguračního programu

5 ZÁVĚR

5.1 Náš program vs McDonald's

V textu jsme několikrát zmiňovali objednávkový systém v McDonald's. Ke konci praktické části bychom se rádi chvíli zabývali porovnáním mezi naším systémem a tím jejich. Největším rozdílem bude jistě to, že náš program byl vyvinut během dvou měsíců týmem tří lidí, zatímco na jejich programu se podílel celý tým specialistů. I přesto si myslím, že má náš program oproti jejich spoustu výhod. Například, zaměstnanci si často stěžují, že jejich systém je příliš komplikovaný, neintuitivní, design je nepřehledný a aplikace je velmi pomalá. Náš program oproti tomu nabízí velmi jednoduchý a intuitivní design spolu s efektivním kódem, což zajišťuje vysokou rychlost. Dalším rozdílem je například naše funkce – náš program nabízí zobrazení toho, z jaké části je objednávka hotová.

Samozřejmě má náš program ještě daleko do dokonalosti. Nemá tolik funkcí (např. vlastní objednávky) a pravděpodobně není tak přizpůsobitelný. Dále bychom řekli, že design některých obrazovek v McDonald's je mnohem lepší než design jakékoliv stránky v našem programu. V této oblasti bychom určitě viděli místo pro zlepšení.

5.2 Celkové zhodnocení projektu

Tato práce se zaměřila na vývoj vlastního objednávkového softwaru určeného pro organizaci školní párty. Všechny stanovené cíle byly naplněny, neboť výsledný software je funkční, přehledný a snadno ovladatelný. Projekt vznikl na základě reálné potřeby mít dostupný a přizpůsobitelný nástroj, který dokáže efektivně podporovat organizaci akcí, zejména v kontextu rostoucího významu informačních technologií a vysokých nákladů na profesionální softwary.

Během práce jsme provedli rozsáhlou rešerši zaměřenou na full-stack vývoj, což nám umožnilo osvojit si aktuální trendy a technologie. Vypracovaný postup byl efektivní, avšak v průběhu vývoje jsme identifikovali několik oblastí, kde je prostor pro zlepšení. Především by bylo vhodné dále vylepšit design softwaru a při dalším vývoji využít moderní Javascriptový framework, který by usnadnil správu a tvorbu uživatelského rozhraní.

Vývojový proces s sebou nesl určité technické výzvy. Při integraci tiskárny pro tisk účtenek a řešení drobných bugů jsme se setkali s komplikacemi, které však byly díky důkladnému testování a průběžným úpravám úspěšně překonány. Tyto zkušenosti nám poskytly cenné poznatky o řešení praktických problémů v oblasti softwarového vývoje.

Jako návrhy na další rozvoj doporučujeme implementaci možnosti úpravy konfigurace přímo v uživatelském rozhraní, čímž by se odstranila nutnost editace JSON souboru. Dalším perspektivním krokem by mohlo být umožnění objednávání a platby přímo z mobilních zařízení, což by celý proces ještě více zjednodušilo a zvýšilo jeho atraktivitu pro uživatele.

Celkově lze práci hodnotit jako úspěšnou, neboť vytvořený software nejenže prokázal svou funkčnost v praxi, ale zároveň představuje kvalitní základ pro budoucí inovace.

6 ZDROJE

6.1 Seznam zdrojů

1. Full Stack Development Explained. Online. MongoDB. Dostupné z: https://www.mongodb.com/resources/basics/full-stack-development. [cit. 2025-02-13].

2. SIMMONS, Liz. *Front-End vs. Back-End: What's the Difference?* Online. ComputerScience.org. August 19, 2024. Dostupné z: https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/. [cit. 2025-02-13].

3. *Full Stack Development Explained*. Online. MongoDB. Dostupné z: https://www.mongodb.com/resources/basics/full-stack-development. [cit. 2025-02-13].

4. SZABO, Pavel. *Full Stack Developer*. Online. Pavel Szabo. Dostupné z: https://www.pavelszabo.cz/co-je-full-stack-developer/. [cit. 2025-02-13].

5. *What is Full Stack Development ?* Online. Geeks For Geeks. 07 Aug, 2024. Dostupné z: https://www.geeksforgeeks.org/what-is-full-stack-development/. [cit. 2025-02-13].

6. *What is an API (Application Programming Interface)?* Online. AWS Amazon Q. Dostupné z: https://aws.amazon.com/what-is/api/. [cit. 2025-02-13].

7. *What is Full Stack Development ?* Online. Geeks For Geeks. 07 Aug, 2024. Dostupné z: https://www.geeksforgeeks.org/what-is-full-stack-development/. [cit. 2025-02-13].

8. RedGrittyBrick. *Journey of a Web Request*. Online. StackExchange. 2013. Dostupné z: https://superuser.com/questions/528001/journey-of-a-web-request. [cit. 2025-02-13].

9. What is DNS? | How DNS works. Online. CloudFlare. Dostupné z: https://www.cloudflare.com/learning/dns/what-is-dns/. [cit. 2025-02-13].

10. , RedGrittyBrick. *Journey of a Web Request*. Online. StackExchange. 2013. Dostupné z: https://superuser.com/questions/528001/journey-of-a-web-request. [cit. 2025-02-13].

11. , RedGrittyBrick. *Journey of a Web Request*. Online. StackExchange. 2013. Dostupné z: https://superuser.com/questions/528001/journey-of-a-web-request. [cit. 2025-02-13].

12. *WebSocket*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 9. 11. 2022. Dostupné z: https://cs.wikipedia.org/wiki/WebSocket. [cit. 2025-02-13].

13. E RABBI, Fazal. *How the Web Works: A Deep Dive into What Happens When You Type a URL, DNS Lookup, and HTTP*. Online. Medium. 2024. Dostupné z: https://fazalerabbi.medium.com/how-the-web-works-a-deep-dive-into-what-happens-when-you-type-a-url-dns-lookup-and-http-f05af4be36d8. [cit. 2025-02-13].

14. , RedGrittyBrick. *Journey of a Web Request*. Online. StackExchange. 2013. Dostupné z: https://superuser.com/questions/528001/journey-of-a-web-request. [cit. 2025-02-13].

15. BENNET, Lucas. *Kdo je front-end vývojář? Kompletní průvodce*. Online. GURU99. Srpen 13, 2024. Dostupné z: https://www.guru99.com/cs/front-end-developer.html. [cit. 2025-02-13].

16. BENNET, Lucas. *Kdo je front-end vývojář? Kompletní průvodce*. Online. GURU99. Srpen 13, 2024. Dostupné z: https://www.guru99.com/cs/front-end-developer.html. [cit. 2025-02-13].

17. *What's the Difference Between Frontend and Backend in Application Development?* Online. AWS Amazon Q. Dostupné z: https://aws.amazon.com/compare/the-difference-between-frontend-and-backend/. [cit. 2025-02-13].

18. *Hypertext Transfer Protocol*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 19.1.2025. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [cit. 2025-02-13].

19. *HTTP headers*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers. [cit. 2025-02-13].

20. *What are cookies? | Cookies definition*. Online. CloudFlare. Dostupné z: https://www.cloudflare.com/learning/privacy/what-are-cookies/. [cit. 2025-02-13].

21. *Aplikace je firma ve firmě*. Online. Think Easy. 2024. Dostupné z: https://thinkeasy.cz/aplikace-je-firma-ve-firme/. [cit. 2025-02-13].

22. KOTT, Petr. *5 principů, jak na tvorbu grafiky u mobilních aplikací*. Online. Peko studio. 07/06/2021. Dostupné z: https://peko-studio.cz/5-principu-jak-na-tvorbu-grafiky-u-mobilnich-aplikaci/. [cit. 2025-02-13].

23. *Co je UX/UI design (webů a aplikací)?* Online. UX/UI design. Dostupné z: https://www.cojeuxui.cz. [cit. 2025-02-13].

24. *Co je UX/UI design (webů a aplikací)?* Online. UX/UI design. Dostupné z: https://www.cojeuxui.cz. [cit. 2025-02-13].

25. *Zásady při stavbě aplikace*. Online. Pixelmate. Dostupné z: https://pixelmate.cz/blog/zasady-pri-stavbe-aplikace. [cit. 2025-02-13].

26. *Co je Design Thinking?* Online. Skillmea. 2022. Dostupné z: https://skillmea.cz/blog/co-je-design-thinking. [cit. 2025-02-13].

27. FARD, Adam. *What is Agile Design Methodology and how to apply it?* Online. Adamfard. Dostupné z: https://adamfard.com/blog/agile-design. [cit. 2025-02-13].

28. FROST, Brad. *Atomic Design*. Online. Bradfrost. Dostupné z: https://bradfrost.com/blog/post/atomic-web-design/. [cit. 2025-02-13].

29. *Lean UX*. Online. SAFe STUDIO. Dostupné z: https://framework.scaledagile.com/lean-ux. [cit. 2025-02-13].

30. *Co je Design Thinking?* Online. Skillmea. 2022. Dostupné z: https://skillmea.cz/blog/co-je-design-thinking. [cit. 2025-02-13].

31. *Hypertext Markup Language*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Markup_Language. [cit. 2025-02-13].

32. *Hlavička HTML dokumentu - Český HTML 5 manuál*. Online. Itnetwork.cz. Dostupné z: https://www.itnetwork.cz/html-css/html5/html-manual/html-css-html-manual-struktura/html-hlavicka-head-cesky-manual. [cit. 2025-02-13].

33. *Basic HTML syntax*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Structuring_content/Basic_HTML_syntax. [cit. 2025-02-13].

34. *What's in the head? Webpage metadata*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Structuring_content/Webpage_metadata. [cit. 2025-02-13].

35. SMITHA, P. S. *HTML*. Online. Slideshare.net. 2022. Dostupné z: https://www.slideshare.net/slideshow/html-1-251294332/251294332#1. [cit. 2025-02-14].

36. *Hlavička HTML dokumentu - Český HTML 5 manuál*. Online. Itnetwork.cz. Dostupné z: https://www.itnetwork.cz/html-css/html5/html-manual/html-css-html-manual-struktura/html-hlavicka-head-cesky-manual. [cit. 2025-02-13].

37. <title>: The Document Title element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title. [cit. 2025-02-13].

38. <link>: The External Resource Link element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link. [cit. 2025-02-13].

39. <a>: The Anchor element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a. [cit. 2025-02-13].

40. <p>: The Paragraph element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p. [cit. 2025-02-13].

41. : The Content Span element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span. [cit. 2025-02-13].

42. : The Content Division element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div. [cit. 2025-02-13].

43. <img>: The Image Embed element. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img. [cit. 2025-02-13].

44. *Kaskádové styly*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 17. 9. 2024. Dostupné z: https://cs.wikipedia.org/wiki/Kaskádové_styly. [cit. 2025-02-13].

45. *CSS: Cascading Style Sheets*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS. [cit. 2025-02-13].

46. ŠTRÁFELDA, Jan. *Kaskádové styly (CSS)*. Online. Jan Štráfelda. Dostupné z: https://www.strafelda.cz/kaskadove-styly. [cit. 2025-02-13].

47. *Bootstrap*. Online. Dostupné z: https://getbootstrap.com/. [cit. 2025-02-13].

48. *Tailwind*. Online. Dostupné z: https://tailwindcss.com/. [cit. 2025-02-13].

49. *CSS selectors and combinators*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_selectors/Selectors_and_combinators. [cit. 2025-02-13].

50. *Pseudo-classes*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes. [cit. 2025-02-13].

51. *Pseudo-elements*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements. [cit. 2025-02-13].

52. MICHÁLEK, Martin. *Dědičnost v CSS: Co to je a kterých vlastností se týká?* Online. Vzhůru dolů. 2019. Dostupné z: https://www.vzhurudolu.cz/prirucka/css-dedicnost. [cit. 2025-02-13].

53. *The box model*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Box_model. [cit. 2025-02-13].

54. *Typography in CSS*. Online. Cssreference.io. Dostupné z: https://cssreference.io/typography/. [cit. 2025-02-13].

55. *Flexbox*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Flexbox. [cit. 2025-02-13].

56. *Using media queries*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries/Using_media_queries. [cit. 2025-02-13].

57. *@keyframes*. Online. Mdn web docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/@keyframes. [cit. 2025-02-13].

58. *SASS*. Online. Dostupné z: https://sass-lang.com. [cit. 2025-02-13].

59. *Variables*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/variables/. [cit. 2025-02-13].

60. *Mixin Values*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/values/mixins/. [cit. 2025-02-13].

61. *@extend*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/extend/. [cit. 2025-02-13].

62. *@if and @else*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/control/if/. [cit. 2025-02-13].

63. *@for*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/control/for/. [cit. 2025-02-13].

64. *@each*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/control/each/. [cit. 2025-02-13].

65. *@while*. Online. SASS. Dostupné z: https://sass-lang.com/documentation/at-rules/control/while/. [cit. 2025-02-13].

75. KRATOCHVÍL, Adam. *Top tiskárny účtenek*. Online. RecenzeS. 2023. Dostupné z: https://www.recenzes.cz/top-tiskarny-uctenek. [cit. 2025-02-13].

6.2 Seznam obrázků a tabulek