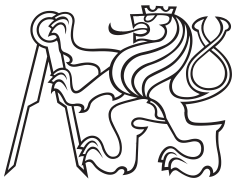
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Výukové materiály pro Node.js

Bc. Yulia Boronenko

Školitel: Ing. David Bernhauer, Ph.D.
Leden 2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Boronenko** Jméno: **Yulia** Osobní číslo: **457884**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Výukové materiály pro Node.js

Název diplomové práce anglicky:

Tutorials for Node.js

Pokyny pro vypracování:

Cílem práce je tvorba veřejně přístupných materiálů v českém jazyce, jež by představovaly strukturovaný tutoriál, který má zaškolit nové Node.js programátory pro český trh práce.
1. Provést výzkum znalostních a technických požadavků na pozici juniorního Node.js vývojáře.
2. Zaměřit se na komparativní průzkum používaných Node.js frameworků: Express, Koa a Nest.js.
3. Vypracovat rešerši (veřejných) existujících tutoriálů a výukových materiálů s podobným zaměřením.
4. Na základě předchozích bodů, najít vhodný způsob zprostředkování materiálů.
5. Analyzovat, navrhnout a vytvořit ukázkový projekt (backend aplikaci), který by na základě provedených analýz zohledňoval klíčové koncepty vybraných frameworků. Případně dodat frontendovou aplikaci s použitím technologií zvolených studentkou, aby výsledný projekt mohl být vhodně ilustrován.
6. Vytvořit a publikovat výukové materiály s podrobným popisem implementovaného projektu.
7. Konzultovat s alespoň 3 firmami materiály a zhodnotit splnění cíle.

Seznam doporučené literatury:

- TILKOV, Stefan; VINOSKI, Steve. Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, 2010, 14.6: 80-83.
- CASCIARO, Mario; LUCIANO Mammino. Node.js Design Patterns - Third Edition. [S.l.]: Packt Publishing, 2020.
- KODR, Marek. Výukové materiály pro nativní Android. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. David Bernhauer, Ph.D. katedra softwarového inženýrství FIT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **07.02.2022** Termín odevzdání diplomové práce: **10.01.2023**

Platnost zadání diplomové práce: **30.09.2023**

Ing. David Bernhauer, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ _____

Datum převzetí zadání Podpis studentky

Poděkování

Chtěla bych poděkovat vedoucímu této práce Ing. Davidu Bernhauerovi, Ph.D za cenné rady a bezmezné pochopení. Další projev vděku patří mému drahému manželovi za neuvěřitelnou mentální a fyzickou podporu, za schopnost dlouhé hodiny trávit čtením a opravováním nekonečné řady chyb, aniž by se z toho zbláznil, a také za to, že ho vedle sebe mám. Rovněž bych ráda poděkovala mému nejlepšímu kamarádovi Honzíkovi za účast ve všech důležitých fázích mého života nezávisle na tom, jestli to zahrnuje psaní diplomové práce nebo ne. Velký dík náleží všem účastníkům hodnocení těchto materiálů, bez vás by nic z toho nebylo. A samozřejmě v neposlední řadě bych chtěla poděkovat svým drahým rodičům za veškerou podporu, kterou jsem od nich obdržela během studijních let a mimo ně.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje.

V Praze, 9. ledna 2023

I declare that this work is my own work and I have cited all the sources I have used in the bibliography.

Prague, January 9, 2023

Abstrakt

Tato diplomová práce se zaměřuje na provedení průzkumu znalostních a technických požadavků na pozici juniorního Node.js vývojáře. Zároveň se věnuje detailnímu souhrnu odborných informací popisujících architekturu Node.js běhového prostředí a vysvětlujících, čím si Node získal tak velkou podporu a popularitu ve vývojářských kruzích. Klíčovou roli v práci představuje rešerše třech nodových frameworků – Express, Koa a Nest.js. Účelem této rešerše je vytipování specifik frameworků pro jejich následnou kompletní komparaci. Rovněž na základě předchozích popsaných bodů probíhá tvorba studijních materiálů v českém jazyce zohledňujících vlastnosti Expressu, Koa a Nestu prostřednictvím konstrukce univerzálního projektu ve třech technologiích. Na konci práce se popíše způsob reprezentace a publikace těchto materiálů a jejich hodnocení ze strany odborníků.

Klíčová slova: Node.js, Express, Koa, Nest.js, výukové materiály, REST API, vývoj backendových aplikací

Školitel: Ing. David Bernhauer, Ph.D. katedra softwarového inženýrství FIT ČVUT

Abstract

This thesis focuses on conducting research into the knowledge and technical requirements for the position of a junior Node.js developer. It also explores a detailed summary of technical information describing the architecture of the Node.js runtime environment and explaining what has made Node so popular in development circles. A key role in the work is played by the research of three Node frameworks – Express, Koa and Nest.js. The purpose of this research is to define the specifics of the frameworks for their subsequent complete comparison. Also, based on the previous points described above, study materials in Czech language are created taking into account the features of Express, Koa and Nest through the construction of a universal project in the three technologies. At the end of the work, the way of representation and publication of these materials and their evaluation by experts will be described.

Keywords: Node.js, Express, Koa, Nest.js, tutorials, REST API, backend applications development

Title translation: Tutorials for Node.js

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Moderní softwarový svět nabízí velkou plejádu programovacích jazyků, jejichž proměnlivá struktura a rozšiřující se seznam nabízených možností dokáže jak přilákat velké množství nových příznivců, tak i naopak odpudit a odradit svoje stálé stoupence. Pro začínajícího softwarového vývojáře je vyznat se v takovém nekonečném množství technologií, najít oblíbený(é) jazyk(y) a vyvíjet se správným směrem bez cizí pomoci nejen obtížným úkolem ale občas nereálným.

Jednou z možností jak vyřešit tento problém je samozřejmě projít vysokoškolským studiem, které vám poskytne pevný znalostní a teoretický základ práce s algoritmy a naučí vnímat a rozeznávat problémy. Rovněž Vám nabídne skvělé podklady pro práci s takovými programovacími giganty, jako je C/C++ nebo Java, ale pouze nakousne nebo prozradí existenci novějších technologií, příp. řekne, kde a v jakých oblastech by se mohly používat. Pokud zmíněné programovací jazyky nejsou Vaším šálkem čaje, pravděpodobně bez dostatečného samostudia a velké energie, které věnujete zvolené technologii mimo školu, nebudete mít šanci získat chtěnou pracovní pozici ani po dokončení studia. A je to naprosto normální a očekávaný stav, vždyť vysoká škola vás nemá naučit všechno (což, buďme upřímní, ani možné není), ale musí vás naučit, JAK SE MÁ správně UČIT. Nejednou jsem se potýkala s absolventy oboru softwarového inženýrství z různých českých a slovenských univerzit, kteří se zoufalstvím a frustrací tvrdili, že si odnášejí z vysoké školy smíšené pocity – umí všechno a nic zároveň, neboť v tom, čím se tak usilovně zabývali, pracovat skutečně nechtějí. A tak jsou ve výsledku nuceni pořídit si za poměrně velkou částku výukové kurzy a tutoriály v angličtině, němčině nebo jiných jazycích (kvůli nedostatku materiálů v tom jejich rodilém), aby prozkoumali oblasti a odvětví, ve kterých disponují po projití vysokoškolskými kurzy minimálními znalostmi.

Tu druhou možnost – praxi v IT – bychom mohli přirovnat k jednomu ze způsobů, jak se malé děti učí plavat: pár starostlivých rodičů sice preferuje začínat v malém bazénku, někteří jedinci radši hodí svoje dítko do mořské vody a čekají, než vyplave samo na povrch. Pokud se někomu zdá, že přirovnávám reálnou praxi v softwarové firmě k psychiku poškozujícímu dětskému zážitku, tak vás můžu ujistit, že se vám to nezdá. Neboť v obou případech, i když se jedná o přístup, jenž má rychlé, účinné a obdivuhodné výsledky, bude to pro

vás velmi stresující, pokud nemáte pevné znalostní základy a odpovídající dovednosti.

1.1 Osobní motivace pro věnování uvedené problematice

Stejně jako většina programátorů jsem našla svoji zálibu mimo předměty vyučované ve škole – v JavaScriptovém běhovém prostředí Node.js. A stejně jako většinu z nich mě hrozně mrzelo, že jsem se k tomu dostala dlouhou a trnitou cestou samostudia, prohrabáváním tuny suše napsaných dokumentací a samozřejmě tisícem vlastnoručně udělaných chyb. Ke kterým by, mimochodem, v případě, že bych měla k dispozici strukturované a rozumně vypracované materiály, ani vůbec nemuselo dojít.

I přesto, že Node.js nepatří k těm nejmladším technologiím, se kterými se na trhu potkáme (v roce 2023 oslaví svoje 14. narozeniny), po delším zkoumání mě dost ohromilo, že žádná z českých škol doteď nenabízí výuku ve formě detailnějšího zkoumání aspoň jednoho z velké sítě různých nodových frameworků. Zajímavým zjištěním zároveň bylo i to, že v českém jazyce k tomu tématu nejsou k dispozici kvalitní (fakticky žádné) materiály zdarma. A tím jsme se dostali k jádru tématu diplomové práce – návrhu veřejně přístupných materiálů, které by představovaly strukturovaný tutoriál pro juniorní programátory v Node.js.

Jako člověk, který na vlastní kůži zažil oba výukové přístupy popsané v předchozích odstavcích, jsem schopna identifikovat jejich klíčové klady a zápory, což by se dalo zohlednit při tvorbě podobných kurzů. Rovněž, co se týče obsahu budoucího kurzu samotného, považuji za podstatné nabídnout čtenářům přehled frameworků Express, Koa a Nest.js s popisem jejich stěžejních vlastností. Je osvědčeným výukovým přístupem vysvětlovat základy nových jazyků či používaných technologií při tvorbě menšího softwarového projektu. Proto budoucí materiály budou koncipovány stejným způsobem. Avšak podstatný rozdíl oproti většině anglických tutoriálů, které jsou volně či placeně k dipozici na internetu, bude spočívat právě v tom, že se stejný projekt postupně realizuje ve třech frameworcích. Tento přístup pomůže uživateli se nejen blíže seznámit s odlišnými strukturami a koncepty tvorby softwaru ve zvolených frameworcích, ale zároveň je vlastnoručně vyzkoušet, provést porovnání a najít pro sebe ten nejvíce vyhovující.

1.2 Převzetí výukového žezla

Zvolený koncept a zaměření práce nejsou běžné v softwarovém odvětví, nicméně nemluvíme o ojedinělém případu. Velkou inspiraci jsem čerpala z výsledků diplomové práce pana Ing. Marka Kodra [1], jež se zaměřovala na tvorbu výukových materiálů pro nativní Android[1]. Zhotovené materiály

[1] https://jduseucit.wixsite.com/android

měly pomoci začátečníkům a zájemcům na jejich osamělé vývojářské cestě s tvorbou mobilních aplikací, ponořit se do problematiky vývoje Androidu a naučit začínající programátory jeho základy. Pan inženýr se stejně jako většina z nás během studia potýkal s častým problémem neexistence hotových, veřejně dostupných materiálů v českém jazyce k technologiím, které si zvolil jako hlavní pracovní zaměření, a proto se rozhodl, že využije prostor nabízený vědeckou komunitou ke změně. Práce měla velký úspěch, a delší dobu byla nabízená jako podpůrný zdroj výuky předmětu BI-AND na Fakultě informačních technologií ČVUT.

V rámci své práce se mnohdy opírám o výzkum a výsledky uvedené ve zmíněné diplomové práce, a jako nástupce představený v jejím závěru se vydávám podobným směrem.

Kapitola 2
Analýza

Tvorba softwarového produktu je komplexním procesem zahrnujícím z pohledu vývojáře nekonečnou plejádu úkonů vyžadujících jak nesmírnou řadu znalostí ohledně neustále se rozšiřujícího a měnícího se balíčku technologií, tak i ohromné množství zkušeností a dovedností. Avšak pro začínajícího programátora vstupujícího do naprosto nového a pro něj neprozkoumaného odvětví je neskutečně obtížné se vyznat v základech, dokázat vyfiltrovat klíčové koncepty ve velkém informačním chaosu a zaměřit se na nezbytné kroky, jež má nový student v rámci své výuky podniknout.

Stejně jako většina vývojářů sdílím názor, že každý z nás dostává největší a nejcennější znalosti v praxi a že práce na rozmanitých projektech sama o sobě přináší ty nejbohatší zkušenosti, které v životě v našem oboru dokážeme nasbírat. Pokud Vás náhodou napadne otázka, jak v takovém případě má postupovat juniorní programátor, jenž svou pracovní cestu právě začíná, tak se rozhodně ptáte správně. V daném případě se potýkáme s několika protikladnými názory v českých softwarových společnostech. Jeden z nich zní, že postačující podmínkou pro přijetí na pozici juniorního vývojáře jsou základní programovací dovednosti a „schopnost analytického myšlení". Uvedené minimální požadavky se podle mého názoru dají vysvětlit zvyšující se poptávkou po softwarových vývojářích na trhu během posledních několika let a generováním většího počtu volných pracovních míst, než jsou aktuálně schopny zaplnit české vysoké školy. Tato nová tendence má ve výsledku několik pozitivních efektů, ke kterým, například, patří stěhování odborníků z jiných států do České republiky, tzv. příliv mozků. Rovněž nesmíme přehlížet i ten evidentní negativní důsledek, jako je výše zmíněné snížení požadavků u některých pracovních pozic, což ve výsledku vede k všeobecnému klesání úrovně pracovníků na pozicích juniorních vývojářů oproti jejich kolegům z jiných států. Jinými slovy jsou méně konkurenceschopní.

Popsaný argument považuji za klíčový, proč se domnívám, že na pozici juniorního programátora si člověk nevystačí s pouhým základem. Zároveň se opírám o svoje vlastní zkušenosti, tedy zkušenosti člověka, který na začátku své kariéry netušil, kde a jak má začít, čemu se má věnovat největší pozornost a proč je třeba o všem vědět něco a o něčem všechno.

2.1 Proč používat Node.js v roce 2023

Díky rostoucí oblibě JavaScriptu a dlouhodobě pozorované tendenci výskytu JS v seznamu preferovaných jazyků u mediorních a seniorních programátorů se tento jazyk v roce 2023 podle indexu PYPL (PopularitY of Programming Language Index) [2] stále drží v trojici nejpoužívanějších, spolu s Pythonem a Javou, a má pokračující narůstající trend oproti prvním dvěma. Dlouholetý odpor vůči JavaScriptu a jeho původní obrovská nepopularita ve vývojářské komunitě konečně začíná vypadat jako minulost. Současné statistiky naznačují existenci jiného, nadějného směru, jemuž nepochybně napomohlo jak zveřejnění ECMAScriptu7 (2015), tak i vznik prvního **javascriptového běhového prostředí** (často chybně označované za framework) pro psaní vysoce škálovatelných webových aplikací – Node.js – což zajistilo JavaScriptu dominantní pozici pro fullstackový vývoj aplikací. A to ani nemluvím o tom, jakou Node.js začal nabírat popularitu v IoT a tvorbě plnohodnotných desktopových aplikací v kombinaci s multiplatformním frameworkem Electron [8].

Avšak co se týče tvorby backendových aplikací, budeme muset uznat, že psaní v „čistém" Nodu se nedostává taková přízeň, jako použití předem stanovených a standardizovaných softwarových struktur, neboli frameworků.

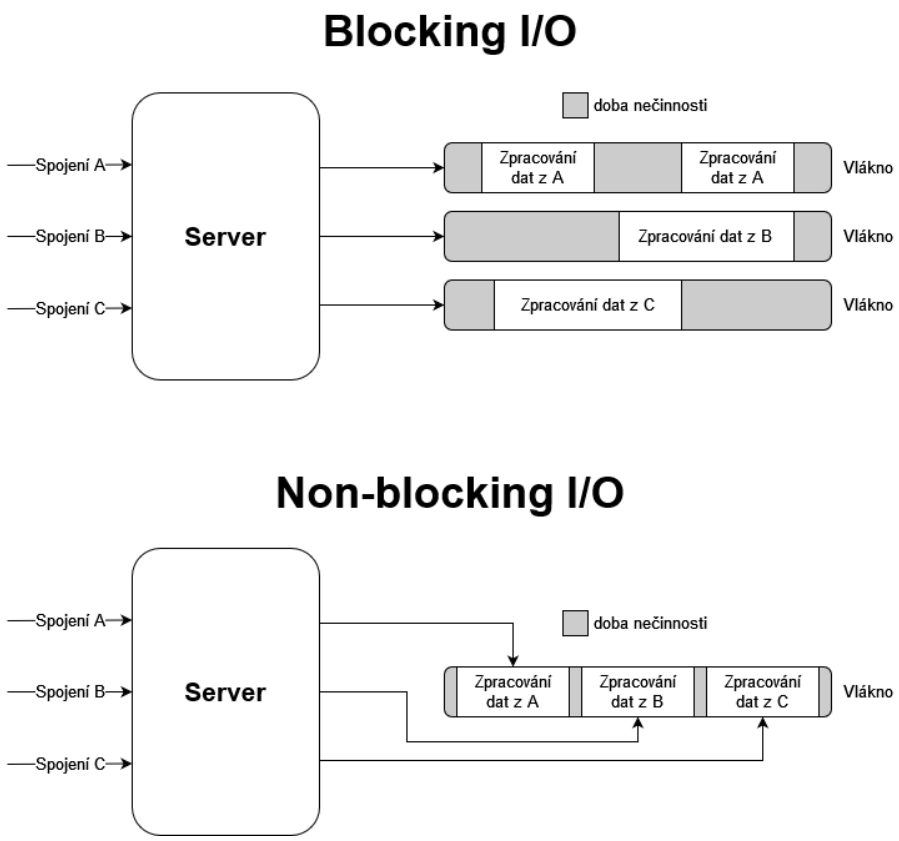
Předtím, než přejdeme k porovnání samotných frameworků, pojďme si říci, proč si Node získal takovou přízeň mezi vývojáři a v čem spočívá jeho největší kouzlo, kterým změnil způsob, jak se webové aplikace, které pracují v reálném čase, vytváří a škálují.

2.1.1 Pod pokličkou Nodu

Jedním ze zmíněných a očividných důvodů popularity Nodu je samozřejmě JavaScript. Nezávisle na vašem vztahu k tomuto jazyku není možné popírat, že je mnohem jednodušší k naučení než většina ze známé nabídky, jako je např. C++, Prolog nebo Scala. Pravděpodobně další otázka, která nějakou delší chvíli tkvěla u vás v hlavě během čtení, zní „Jak je možné, že Node dokáže spouštět javaccriptový kód mimo prohlížeč?" Ve skutečnosti Node nespouští JS kód mimo prohlížeč, ale jeho „alternativní vazbu" v C++.

Node je javacriptový projekt, jenž má interně obrovskou kolekci závislostí, ke kterým patří i ty dvě nejzásadnější (viz [9]): **V8** a **libuv**. Každý webový prohlížeč, ať už Chrome, Firefox, Safari nebo Internet Explorer, má svůj javascriptový engine, jenž poskytuje běhové prostředí pro JS, a V8 je zrovna jedním z nich. V8 je otevřeným softwarem, "motorem" Chromu, jehož klíčovou úlohou je spouštění javascriptového kódu. Jedna věc, kterou bychom měli mít na paměti, je, že se v podstatě nejedná o nic jiného, než o C++ knihovnu. Prozkoumáme proces jejího fungování.

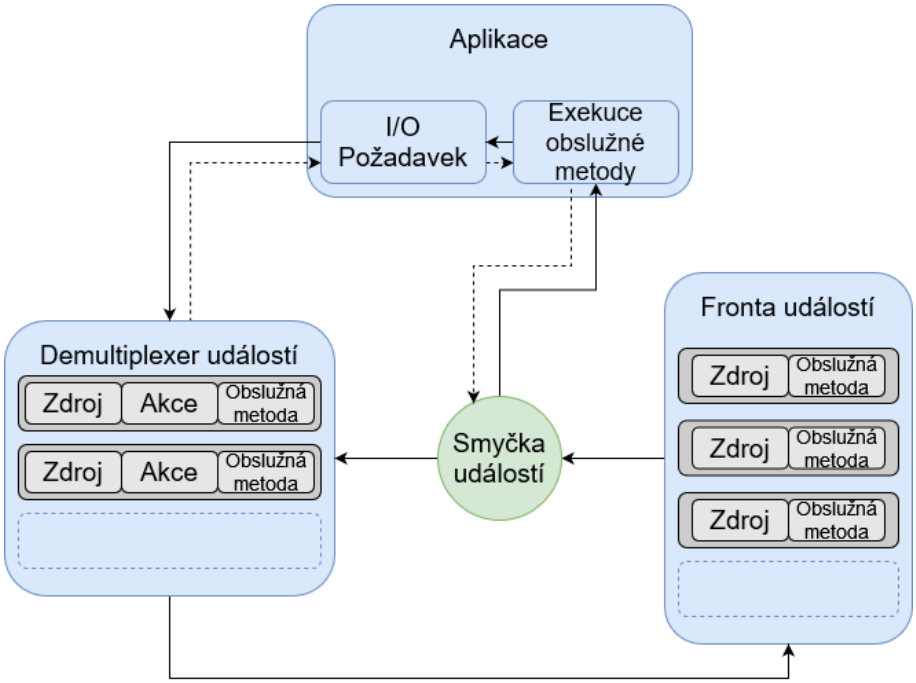
Ten primární důvod, proč je Node.js v takové oblibě, je jeho asynchronní chování. Dost dobře víme, že v tradičních programovacích praktikách většina blokujících operací (síťový požadavek, databázové operace, vstupní/výstupní operace na disku apod.) probíhá synchronně. Když se podíváme na to, co



Obrázek 2.1: Rozdíl mezi blocking a non-blocking I/O

se skutečně odehrává na pozadí, tak si všimneme, že provedení jakékoliv podobné operace zablokuje celé hlavní vlákno aplikace. V důsledku toho vlákno nedokáže přijmout žádný další požadavek, dokud se nezpracuje ten aktuální. Popsaný jev označujeme v odborné literatuře termínem blocking I/O. Node přichází s myšlenkou **non-blocking I/O**, kde jedno vlákno obsluhuje několik souběžných spojení (viz 2.1), přitom nečeká na dokončení konkrétní zpracovávané úlohy a přechází na další. Tato jednoduchá a přesto brilantní myšlenka se implementovala v podobě návrhového vzoru **Reaktor**, který vzhledem k jeho komplexitě představíme podrobněji.

Zásadní koncepce Reaktoru je přiřazení odpovídající **obslužné metody** (event handler) ke každé I/O operaci. Obslužná metoda v Nodu je reprezentována funkcemi zpětného volání (callbacky), což jsou funkce, které se předávají do jiných funkcí jako vstupní argumenty a mohou být jimi vyvolány. Celý proces mechanismu fungování Node aplikace využívající vzor Reactor je ilustrován na obrázku 2.2. Na začátku aplikace vygeneruje několik I/O operací, které následně přepraví na synchronní **demultiplexor** událostí. Pod termínem multiplexing si představte metodu, pomocí které se více signálů spojí do jednoho tak, aby se jednoduše přenášely přes médium s omezenou kapacitou. Demultiplexing je termín popisující obrácenou operaci, během



Obrázek 2.2: Implementace návrhového vzoru Reaktor [6]

které se signál rozděluje na původní komponenty. Odeslání nového požadavku do multiplexoru je neblokujícím voláním, díky čemuž se řízení okamžitě vrací zpátky aplikaci. Jak popisuje kniha [6], synchronní demultiplexor událostí sleduje více zdrojů a vrací novou událost (příp. sadu událostí), když se dokončí operace čtení nebo zápisu přes jeden z těchto zdrojů. Po dokončení sady I/O operací demultiplexor vloží vytvořené události do **fronty událostí** (event queue). V tomto okamžiku **smyčka událostí** (event loop) iteruje přes všechny položky fronty událostí. Pro každou událost je zavolána odpovídající obslužná metoda (callback). Obslužné metody mohou být jak blokujícího typu (sync), tak i neblokujícího (async) – v závislosti na implementaci. Po dokončení práce funkce zpětného volání předá řízení zpátky frontě událostí. Zároveň je třeba dodat, že během svého volání obslužná metoda může vyžádat nové asynchronní operace, což následně způsobí přidání dalších položek do synchronního multiplexoru událostí. Když se zpracují všechny položky z fronty událostí, fronta se zablokuje na demultiplexoru, který spustí novou iteraci, jakmile se objeví nová událost.

Tři klíčové programovací principy Node.js architektury:

- existence sady funkcí pro zpracování událostí (callbacky)
- tvorba vazby mezi registrovanými funkcemi a událostmi (viz práce demultiplexoru)
- dotazování se na nové události a volání obslužných metod smyčkou událostí po přijetí každé registrované události

Na základě zmíněných vlastností a hlubšího nahlédnutí do nitra Node.js skrze zkoumání jeho architektury můžeme vydedukovat, že jeho jádro je jednovláknové, a tím pádem si zodpovíme na výchozí otázku kteréhokoliv pohovoru na pozici Node.js vývojáře.

Node jako nesmírně výkonné a efektivní běhové prostředí díky realizaci non-blocking I/O umožňuje vývojářům bezproblémové škálování aplikací – jak horizontálně pomocí přidání nutných zdrojů existujícímu uzlu, tak i vertikálně pomocí přidání nových uzlů dle článku [7]. Je známo, že Node zvyšuje vývojářskou produktivitu – konečně se programátor nemusí "přepínat" mezi jazyky během psaní frontendové a backendové částí aplikace. Jedna z jeho dalších nespočetných výhod spočívá v tom, že dodržuje základy "lightweight" metody vývoje softwaru, která není komplexní, má omezený počet využívaných postupů a pravidel, jež se snadno dodržují. Nesmíme ani opomenout, že jak Node, tak i jeho bohatý ekosystém (NPM), patří do kategorie otevřeného softwaru (open-source).

Je rovněž mojí povinností poukázat i na jiné vhodné využití Nodu v dalších typech softwarových projektů, ke kterým patří např. nástroje pro interakci v reálném čase (Websockety), streamovací webové aplikace (které se realizují pomocí přenášení dat sekvenčně, v blocích, skrze vestavěný modul streamů) a aplikace založené na architektuře mikroslužeb.

2.2 Analýza a definice potenciální cílové skupiny

Předtím, než zahájíme tvorbu jakýchkoliv výukových materiálů, je třeba přesně specifikovat persony, neboli studenty, kteří vytváří poptávku po těchto kurzech. Následně by se měl na základě stanovených typů určit exaktní styl výuky a preferovaný způsob reprezentace hotových materiálů.

- Frontendový developer, jehož primárním účelem je přejít na Full-stackový vývoj.
  Posledních deset let se Lukáš Polák zabývá frontendovým vývojem aplikací v Reactu a Angularu. Mezi svými kolegy ve firmě je považován za seniorního programátora, zvládá práci pod tlakem a jeho názor je dost často rozhodující při řešení podstatných problémů týkajících se vývoje různých projektů. Vzhledem k tomu, že firma, které věnoval velkou část svého života, patří k tzv. „Korporátům" a má podle názoru Lukáše nespravedlivé podmínky pro kariérní postup (lepší finanční ohodnocení rychleji obdrží člověk s minimálně bakalářským vzděláním), přemýšlí náš protagonista o změně pracovní pozice. Jeho celoživotní přesvědčení ohledně nepotřebného vysokoškolského vzdělání v IT „na stará kolena" měnit neplánuje, a proto se rozhodl, že se naučí vytvářet backendové aplikace v JavaScriptu na profesionální úrovni. Proto hledá kurzy s širokou nabídkou, které dokáží pokrýt široké aspekty backendového vývoje a ponořit se více do struktury Nodu.

- Zkušený backendový vývojář, který je ochotný se naučit nové technologie.
  Vojtěch Filip je mediorní programátor, jenž má za sebou jak práci ve

vládním a soukromém sektoru, tak i inženýrský titul z Fakulty informačních technologií v Brně, který mu vzal spoustu energie a úsilí. Za devět let odborné praxe si Vojtěch uvědomil, že nezávislé na použitých frameworcích, programování v PHP, které ho celý dospělý život živilo, ho už omrzelo. Začal mu vadit synchronní „zpomalenější" základ PHP a komplikovaný proces škálování aplikací. Delší dobu Vojtěch uvažoval, jestli by ho více lákal Node nebo Python. Přiklonil se však k Nodu a rozhodl se, že by rád vyzkoušel více JS frameworků najednou, provedl jejich porovnání a následně by si díky tomu zvolil ten jemu nejpříjemnější pro následný pracovní rozvoj.

- Vysokoškolský student používající Node v jednom ze svých projektů.
  Jan Svoboda je studentem magisterského studia Otevřené informatiky Fakulty elektrotechnické ČVUT v Praze, kde studuje obor Softwarové inženýrství. Má za sebou titul Bc. a velké zkušenosti s vývojem Enterprise aplikací v Javě, dobře se orientuje v C++ a praxe v menší softwarové firmě v Dejvicích mu pomohla vyvinout na začátku chybějící manažerské dovednosti. Aktuálně se zaměřuje na programování backendu ve Flasku. V rámci školního oborového předmětu „Softwarové architektury" se však potýká s projekty, které jsou napsány v Nodu. Rovněž jeden z jeho kolegů, se kterým má v plánu vytvořit finální semestrální projekt, mu doporučil psát mikroslužby v Expressu nebo Koa. Základy JS syntaxe se díky svým rozsáhlým znalostem Jan zvládl naučit do několika dnů. Vzhledem k tomu, že pan Svoboda nemá talent na jazyky, preferoval by tutoriál jednoduše a srozumitelně napsaný v češtině.

- Programátor-začátečník, který má za sebou základy JavaScriptu.
  Ondřej Jabko je mladý středoškolský student z Ostravy, který chodí na všeobecné gymnázium, a stejně jako každý kluk jeho věku se trápí volbou správné vysoké školy. Ondra vždycky věděl, že má blízko k oboru IT, ale nikdy předtím se neodvážil najít odpověď na otázku „Kde a hlavně jak má začít?" Avšak před dvěma týdny ho oslovila reklama Udemy kurzu „JavaScript programming: JavaScript for beginners", který si pořídil a s velkým nadšením absolvoval. Aktuálně prochází fází volby mezi frontendovým a backendovým vývojem, chtěl by vyzkoušet oba směry a najít si ten, který by ho nejvíce naplňoval. Proto by si rád vyzkoušel kurzy v Nodu (preferoval by je v češtině, jelikož se zatím necítí dost sebejistě při čtení anglických textů), avšak Ondra nemá žádnou brigádu, a proto jsou pro něj dokonce i zlevněné kurzy Udemy příliš drahé.

- Node.js vývojář, který by se rád naučil pracovat s jinou sadou frameworků.
  Dvacetiletý Oleksandr Shevchenko je zahraniční student z Ukrajiny, který se přestěhoval do České republiky za účelem studia. Aktuálně je ve třetím ročníku bakalářského studia na ČVUT FEL a díky dobře zvládnuté práci s časem zároveň stíhá poloviční úvazek na pozici juniorního Node.js vývojáře ve firmě Accenture, kde převážně vyvíjí několik mikroslužeb v

Expressu. Ani JavaScript nebo TypeScript Oleksandrovi nedělají problémy, ale během dvou let studia se mu prozatím nenaskytla příležitost seznámit se s jiným nodovým frameworkem, než je Express. Po půl roce úspěšně zvládnuté zkušební doby je zapojen do práce na dalších projektech, z nichž jeden je napsán v Koa a druhý v Nest.js, které by se měl naučit v rámci samostudia. Angličtinu a češtinu Oleksandr zvládá přibližně na stejné úrovni, proto pro něj jazyk není rozhodujícím faktorem při volbě kurzů. Bylo by však vhodnější si najít či pořídit výukové materiály, jež by do sebe zahrnovaly oba zmíněné frameworky. Kdyby zároveň byly zdarma, tak by si ukrajinský student ušetřil nějaký finanční obnos na platbu za kolej.

Za negativní personu můžeme považovat takového zájemce o kurzy, který nedisponuje nutnými znalostmi JavaScriptu, TypeScriptu nebo osnov práce s relačními databázemi. Tímto tématem se zabývá nespočetné množství volně dostupných kurzů jak v češtině, tak i angličtině. Proto nepovažuji za nutnost se tomu v rámci plánovaných kurzů věnovat.

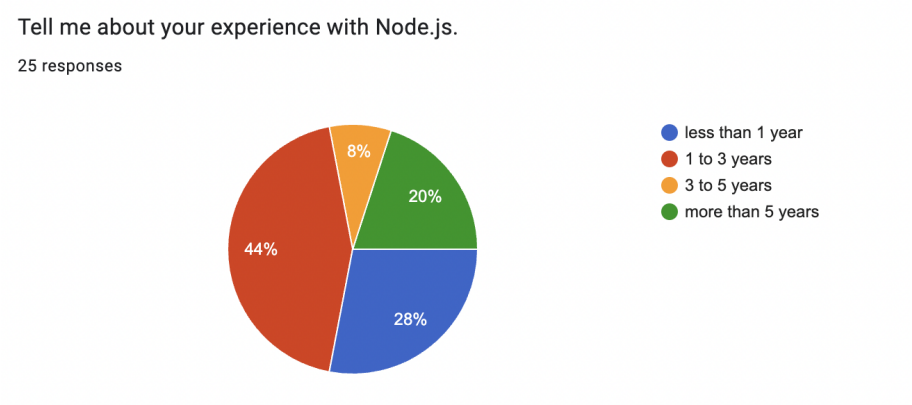
2.3 Výzkum znalostních a technických požadavků na pozici juniorního Node.js vývojáře

Node.js nabírá svoji popularitu v globálním měřítku – obrovský boom v USA, kde ho podle statistik na rok 2021 používá 6,3 milionu webových aplikací [3], ke kterým patří ti největší hráči, jako jsou Amazon, Netflix, eBay nebo PayPal. Česká republika nezaostává ve svém vývoji, a jen za poslední 4 roky počty nabídek a inzerátů na pozice Node.js vývojáře několikanásobně vzrostly (viz [4] a [5] – reporty Indeed.com, LinkedIn, prace.cz nebo jobs.cz).

Na základě velkého objemu přečtených inzerátů (60+) z výše uvedených zdrojů a shromážděných dat byly vytipovány směry a pilíře, na kterých by se měly zakládat znalosti juniorního Node.js softwarového vývojáře a kterými by měl disponovat před nástupem do práce. K nim podle daného výzkumu patří ovládání několika nodových frameworků, umění vytvářet API, chápání principů SQL a NoSQL databází a jejich použití, zkušenosti s cloudovými službami, dockerizací, CI/CD, klíčové znalosti softwarového inženýrství a umění otestovat svůj vlastní kód.

Na základě toho byl vytvořen dotazník s použitím nástroje Google Forms[1], jehož účelem mělo být oslovení co největšího počtu Node.js vývojářů, kterých se ptal, jaké technologie by měl podle jejich názorů juniorní backendový programátor ovládat. I přesto, že mým primárním zaměřením měla být Česká republika, zajímalo mě, jakým způsobem se budou lišit odpovědi vývojářů z jiných zemí. Proto jsem vytvořený dotazník poskytla na stránky mezinárodních fór. Největším zádrhelem a mojí hlavní obavou byla malá odezva na prosbu o vyplnění formuláře, což značně záleželo na samotné distribuci dotazníku a

[1] https://docs.google.com/forms/d/e/1FAIpQLSem0bGvVJbXcMc6sKleDu2Xq-ujd12aHTrbPokDtMxQeLC7Rw/viewform?usp=sf_link

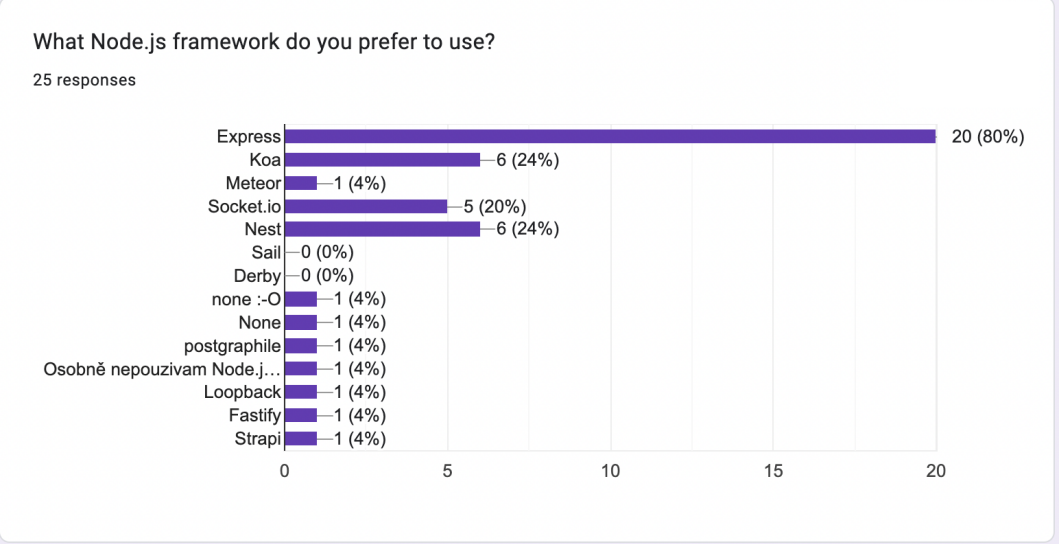


Obrázek 2.3: Zkušenost výzkumné skupiny s Node.js běhovým prostředím

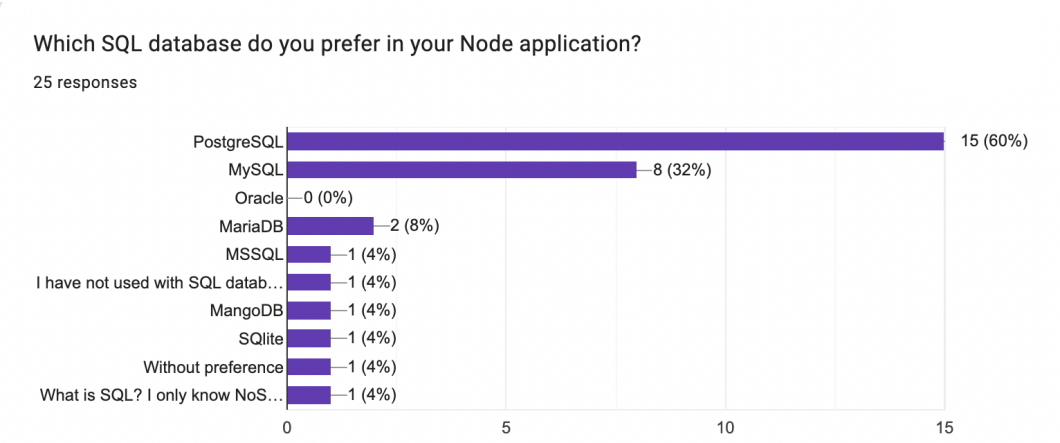
nalezení korektního způsobu formulování otázek, jejich souladnosti a stručnosti celku (vzhledem k tomu, jak málo času průměrně věnuje uživatel jedné konkrétní otázce). Naštěstí se moje obavy nesplnily a povedlo se nashromáždit dostatečné množství odpovědí. Kolem 76 % respondentů pocházelo z Česka a Slovenska, zbytek nepřekvapivě rozdělily mezi sebou Indie a Pákistán. Moji pozornost právě upoutal ten fakt, že při procházení jednotlivých odpovědí nebyl vidět tak zásadní rozdíl mezi výsledky poskytnutými dotazovanými z České republiky a zahraničí. Kolem 44 % respondentů mělo s vývojem v Node.js zkušenosti od 1 do 3 roků a 28 % lidí v daném běhovém prostředí vyvíjelo déle než 3 roky. Podrobnější výsledky jsou k dispozici na obrázku 2.3.

První z položených otázek (viz obrázek dole) se týkala vyjmenování často používaných frameworků, kde s převahou vyhrál Express, což respondenti následně vysvětlují jednoduchostí a rychlostí vývoje aplikací, flexibilitou, výsledným čitelným kódem, dobře propracovanou mohutnou dokumentací a existujícími online tutoriály. Socket.io jako knihovna, která zajišťuje oboustrannou komunikaci v reálném čase mezi prohlížečem a serverem, má odlišný účel, než ostatní uvedené frameworky pro vývoj škálovatelných server-side aplikací, a obsadila druhé místo. Nicméně je třeba poznamenat, že se jednalo o chyták, neboť socket.io nepatří k frameworkům, ale ke knihovnám. Což naznačuje, že bych měla v rámci výukových materiálů vymezit rozdíly mezi knihovnami a frameworky. Zároveň je z výsledků 2.4 patrné, že se projevil narůstající trend frameworku Nest.js, jenž používá TypeScript, a frameworku Koa.js, který oslovená část Node.js komunity považuje za vylepšenou verzi Expressu. Tři respondenti poukázali na to, že žádné frameworky zatím nepoužili, ale podle dotazníku patří ke skupině, jež se věnuje programování v Nodu méně než jeden rok. Tímto je za žádných okolností nechci diskreditovat, ale dlouholeté zkušenosti jsou to, čemu bychom v rámci výzkumu měli přidávat tu největší hodnotu.

Následně jsem se zabývala použitím databázových systémů, které by respondenti doporučili umět ovládat začínajícím Node.js vývojářům ještě před podáním žádosti o pracovní nabídku. Mezi SQL databázemi obdrželo prven-



Obrázek 2.4: Průzkum nejpoužívanějších nodových frameworků

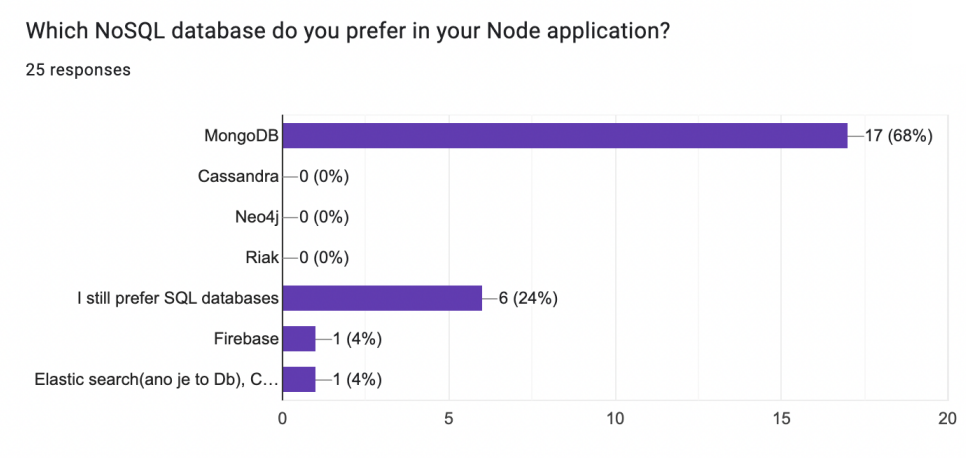


Obrázek 2.5: Nejvíce preferované relační databáze výzkumné skupiny

ství PostgreSQL, za kterým následuje MySQL (viz 2.5). Co se týče NoSQL databází, z výsledku 2.6 beze sporu vyhrává MongoDB.

Je třeba neopomenout testování vlastního kódu alespoň jednotkovými testy (což považují za přirozený požadavek 99 % pracovních inzerátů), s čímž souhlasí i většina dotazovaných a volí testovací framework Jest jako jeden z nejspolehlivějších ze seznamu těch aktuálně nejpopulárnějších. Osobně mě příjemně překvapilo, že se významné množství backendových vývojářů věnuje psaní integračních a akceptačních UI testů (na co poukazuje, např., volba Cypressu v 2.7).

Ve většině pracovních nabídek byla zmíněna nutnost znalosti principů CI a CD. Majorita dotazovaných rovněž uznala, že si juniorní vývojář vystačí s chápáním základního konceptu, ale jen 18 % vyjádřilo názor, že tato dovednost je stěžejní a měla by být znázorněná ve výukových materiálech (viz 2.8).



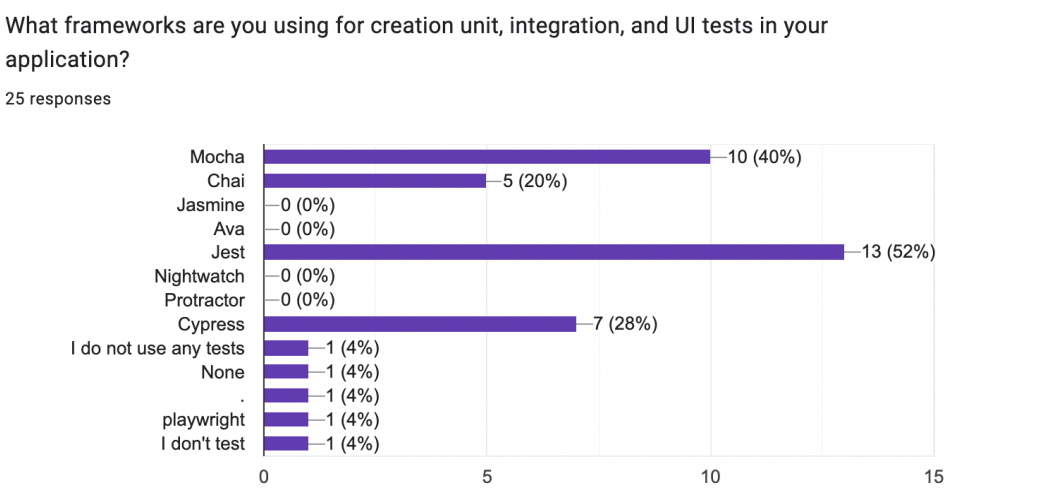
Obrázek 2.6: Upřednostňované NoSQL databáze výzkumné skupiny

Rovněž považuji za velkou a příjemnou změnu narůstající přízeň vůči TypeScriptu. Vycházím z toho, že 54 % lidí ve svých odpovědích zmínilo, že pravidelně používá jak Javascript, tak i Typescript, a 32 % Typescript dokonce preferuje.

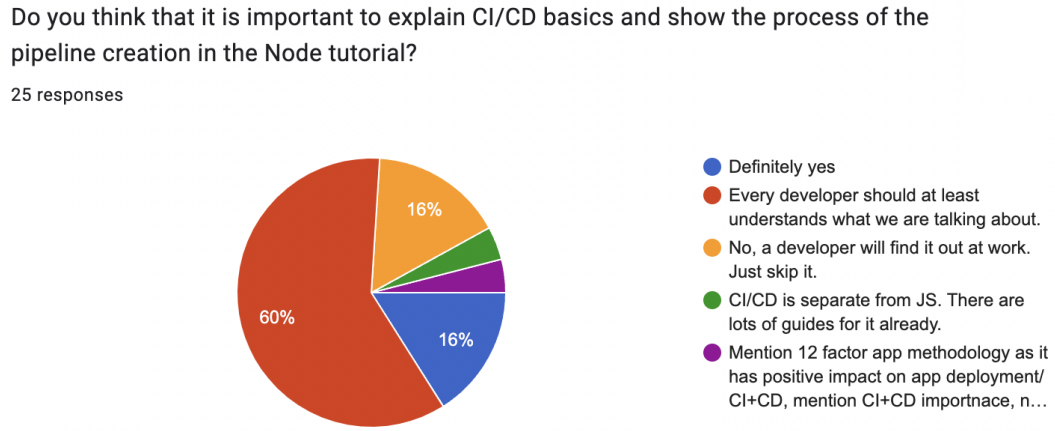
Významnou roli v pracovních nabídkách měla otázka dockerizace. Několikrát jsem narazila v inzerátech na požadavek znalosti zacházení s Kubernetes, který mě osobně do jisté míry překvapil, vzhledem k tomu, že pochopení principů dockerizace je z mých osobních zkušeností nutné až pro mediorního vývojáře. Rovněž podle mého názoru tyto dovednosti patří k těm, které se vývojář jednoduše naučí v pracovním prostředí, když to po něm bude vyžadováno. Stejný postoj, který je ilustrován na obrázku 2.9) se mnou zaujala i dotazovaná skupina, kde jen 13,6 % odpovědělo, že se jedná o nutnou znalost, jež by zájemce o juniorní pracovní pozici měl disponovat.

Následně se v dotazníku zjišťovalo, které cloudové služby (AWS, GSP, Azure) participanti nejčastěji používají. Největší oblibu mezi respondenty však získal Amazon Web Services (kolem 52,4 %). Druhé místo obdržel Azure. Pozoruhodné ale je, že procentuální stejná část dotazovaných odpověděla, že upřednostňuje oproti hotovým vlastní řešení.

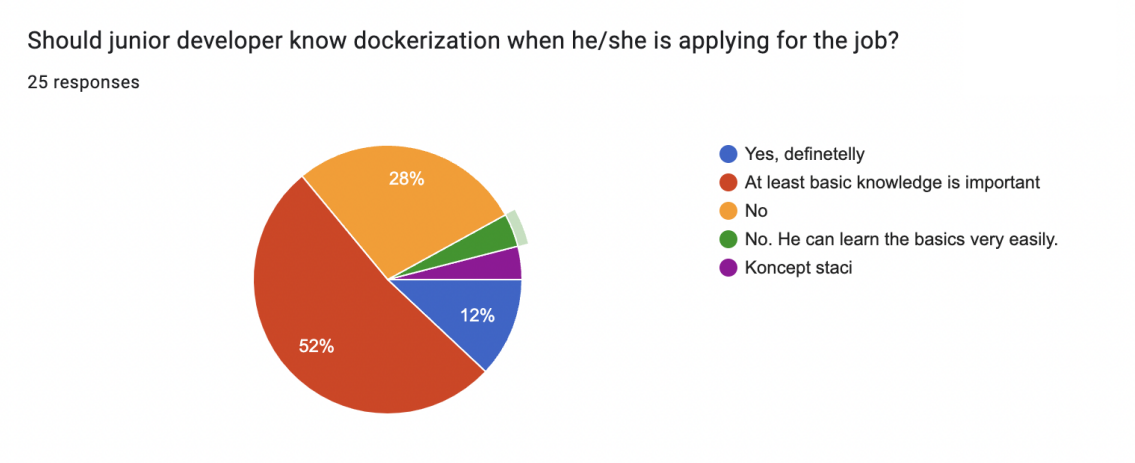
Na konci dotazníku byla položena poměrně zavádějící otázka na návrhové vzory, které sice nezmiňuje každý pracovní inzerát, avšak jsou neoddělitelnou součástí softwarového inženýrství stejně jako je např. Chápání a použití SOLID principů. Vývojáři nejvíce doporučuji ovládat Flow Patterns (Promisy, async/await), následně se soustředit na Creational návrhové vzory (Factory, Singleton, Revealing constructor) a architektonické vzory.



Obrázek 2.7: Testovací frameworky používané dotazovanou skupinou



Obrázek 2.8: Nutnost znalosti CI/CD pro juniorního vývojáře podle participantů



Obrázek 2.9: Nutnost znalosti dockerizace pro juniorního vývojáře podle participantů

Kapitola 3

Komparativní průzkum Node.js frameworků

Podle aktuálních zdrojů čtyři z pěti nodových vývojářů aktivně využívají „předpřipravenou" softwarovou strukturu, neboli frameworky, jejichž aktuální počet se blíží k několika desítkám. Jinými slovy si programátor dokáže vybrat na základě osobních preferencí a požadavků na konkrétní aplikaci, o jaký přesně MVC, Full-Stackový MVC nebo REST API nodový framework se bude jednat. My se v rámci této práce soustředíme a porovnáme tři nejzákladnější a podle předchozího výzkumu i nejpoužívanější frameworky, se kterými se s velkou pravděpodobností jakýkoliv Node.js vývojář potká v praxi: Express, Koa a Nest.js

3.1 Express

Express.js je open-source webový framework pro Node.js, který se vyznačuje svojí rychlou a snadnou tvorbou webových aplikací [10]. Jedná se o jeden z nejpopulárnějších a nejstarších webových frameworků založených na Nodu (hlavně na nodovém *http* modulu) zajišťující tvorbu serverových aplikací. Jeho hlavní využití je tvorba restového API, nicméně Express žádným způsobem neomezuje plejádu možností poskytnutých samotným Nodem [8] (IoT, mobilní aplikace atd.). Pomocí Expressu jsme schopni výkonně „produkovat" jak jednostránkové (SPA), tak i vícestránkové (MPA) či hybridní aplikace v krátkém čase.

Express patří k lightweight frameworkům a sám o sobě hodně vylepšuje základní funkcionality Node.js. Jeho kouzlo spočívá v poskytování jednoduchého routování požadavků ze strany klientů. V roce 2023 je Express nejvyspělejším a nejpoužívanějším Node frameworkem s gigantickou komunitou, která se exponenciálně rozrůstá s každým novým dnem. Díky tomu považuji seznámení s Expressem za první nejjednodušší a nejzásadnější krok, který by měl udělat jakýkoliv programátor začínající v Nodu. S více než +59 500 hvězdami na GitHubu a přes 20 milionů npm stažení týdně patří Express k těm nejpopulárnějším nodovým frameworkům. Jeho jednoduchost spočívá v minimalismu a flexibilitě, které ho zařazují do kategorie **názorově nevyhraněný** (un-opinionated) framework (čímž je hodně blízký PHP nebo .NET), neboť rozvazuje ruce vývojářům, dává jim větší kontrolu tím, že nediktuje a nespecifikuje, jak mají vypadat architektura a osvědčené konvence.

Zdůrazníme, že Express je silně závislý na knihovnách middlewarů, proto se mu často říká **middleware framework**.

3.2 Koa

Koa je brán jako čerstvý nástupce Expressu, kterého přivedl na svět stejný tým vývojářů pět let po vzniku jeho „předka". Primárním účelem daného otevřeného softwaru (open-source) frameworku je být menším, robustnějším a lepším základem pro tvorbu webových aplikací a API, než jeho předchůdce. Ovšem Koa je poměrně velkým odklonem od Expressu, jehož design je v podstatě hodně odlišný. Oproti zmíněnému Expressu Koa umožňuje eliminovat zpětná volání (callbacky) a zřetelně zlepšuje zpracování chyb díky použití asynchronních funkcí. Jeho pilířovou funkcionalitou je aplikace generátoru ES6, která zajišťuje psaní mnohem jednoduššího a čistšího kódu. Zkušeného uživatele Nodu příjemně překvapí použití kontextového objektu, který nejenom do sebe zapouzdřuje čisté nodové objekty požadavků a odpovědí, ale vytváří nad nimi nadstavbu. Koa je více modulární než Express, díky čemuž je „lehčí" (chybí např. routování nebo šablony) – měli bychom zahrnout pouze moduly, které potřebujeme. Oproti ostatním frameworkům má Koa patrně menší rozměry (footprint), jen pár řádků kódu, což podle tutoriálu [31] pomáhá vývojářům psát lepší a tenčí middleware. I přesto, že Koa využívá middleware, který není kompatibilní s většinou zbylých frameworků Node.js [11] (taky kvůli zmíněnému kontextovému objektu), +33 500 hvězd na GitHubu a přes milion stažení z výchozího správce balíčku týdně mluví o rostoucí oblibě menšího bratra Expressu mezi zkušený uživateli. Vzhledem k tomu, že framework je zařazen do třídy „vysoce názorově nevyhraněný" (highly un-opiniated), Koa poskytuje vývojářům větší možnost sebevyjádření tím, že je vybízí k vývoji jakéhokoliv potřebného middlewaru (i když mají možnost rozšířit systém připojením omezené řady existujících modulů) nebo tím, že nediktuje, jakým způsobem má vypadat projektová struktura a nevnucuje jmenné konvence.

3.3 Nest.js

Pokud nejste moc velkým fanouškem čistého JavaScriptu a preferujete jazyky zajišťující typovou bezpečnost, rovněž si dokážete vybrat ze široké nabídky nodových frameworků, a s velkou pravděpodobností vaše volba skončí u Nest.js (použití čistého JavaScriptu není v Nestu explicitně zakázáno, ale podle dokumentace je „prozatím povoleno"). Nest.js je jeden z nejrychleji rostoucích MVC typescriptových „názorově vyhraněných" frameworků pro tvorbu škálovatelných, robustních a efektivních serverových aplikací za pomoci Node.js [12].

Nest.js nepřináší jenom základní výhody obyčejného typescriptového frameworku (defaultně založeného na Express, nicméně Nest zajišťuje kompatibilitu i s dalším frameworkem – Fastify) – jeho silný základ angularové filosofie proklamující dědičnost, dynamické moduly a služby a pokročilé vkládání

závislostí můžeme považovat za obrovskou výhodu při tvorbě podnikových aplikací. Nest.js není první a rozhodně ani poslední JS framework vybudovaný na základě jiného frameworku, jehož účelem je zlepšení stávajících funkcionalit a přidání nové úrovně abstrakce.

Pokud jste dlouholetým frontendovým vývojářem se zkušenostmi v Angularu, Reactu nebo Vue, Nest může být vaší nejlepší volbou pro začátek programování v Nodu. Zajisté Nest přijde dost povědomý svou architekturou vývojářům v .NETu nebo javovských aplikací. Aplikace konstrukcí na vyšší úrovni (např., interceptory, roury, filtry, guardy) nebo aplikace některých návrhových vzorů, ke kterým patří injekce závislostí nebo moduly, sice příznivcům Expressu nebo Koa mohou připadat na začátku dost matoucí a komplikované, avšak po delší chvíli strávené s frameworkem jeho uživatel dokáže ocenit výhody zmíněných struktur a vzorů a zamilovat se do nich (o čemž svědčí i počet hvězd na GitHubu – 53 400). Díky tomu, že Nest.js je nadstavbou nad Nodem, je samozřejmostí, že může využívat všechny jeho moduly.

3.4 Porovnání charakteristik a vlastností frameworků

V rámci přípravy komparativního průzkumu všech zmíněných frameworků, jejichž výsledky jsou k nahlédnutí v podkapitolách, byla přečtena jak oficiální dokumentace Koa, Nestu a Expressu, tak i řada odborných článků, vyzkoušena tvorba několika projektů a provedeno hlubší zkoumání zdrojových kódu samotných frameworků. Jejich výsledné hodnocení probíhalo na základě několika kritérií: jednoduchost základní instalace, zajištění routovacího mechanismu, křivka učení, middleware, architektura, údržba zdrojového kódu a rychlost vývoje, způsob zpracování chyb a nakonec rozsah komunity a dokumentace.

3.4.1 Jednoduchost základní instalace

Express
Lehký a srozumitelný návod, stačí pár řádků pro rozběhnutí serveru. Pokud máme Node, instalace Expressu trvá pár vteřin (jednořádkový příkaz).
Koa
Lehký a srozumitelný návod, stačí pár řádků pro rozběhnutí serveru, avšak vyžaduje Node v7.6.0 nebo vyšší pro podporu ES2015.
Nest.js
Lehký a srozumitelný návod, stačí pár řádků pro rozběhnutí serveru. Instalace trvá o něco déle oproti ostatním frameworkům kvůli většímu počtů závislostí. Navíc se doporučuje instalace nástroje pro příkazovou řádku Nest CLI jako pomocný nástroj pro inicializaci, vývoj a údržbu Nest.js projektů.

3.4.2 Zajištění routovacího/směrovacího mechanismu

Routování, nebo směrování, je mechanismus, určující jaká metoda obsluhy (handler) obdrží specifický HTTP požadavek od klienta a naservíruje vyžadovaný obsah.

Express
Vysoce pokročilý mechanismus směrování, avšak je náročnější na práci, jelikož vyžaduje ruční vytváření všech koncových bodů, výsledkem čehož je komplikovanější debugování a větší riziko chybovosti.

Koa
Vyžaduje instalaci balíčku na routování navíc (*koa-router*). Vysoce pokročilý mechanismus směrování připomínající Express.

Nest.js
Směrovaní v aplikacích obstará jednoduché a robustní využití kombinace řadičů (controllers) a dekorátorů – cesta pro obslužnou metodu (handler) se definuje kombinací nepovinného prefixu řadiče a cesty dekorátoru metody.

3.4.3 Křivka učení

Express
Jednoduchá křivka učení, člověk se základy JS a chápáním funkcionálního programování na přijatelné úrovni, internetu jako globální sítě, HTTP protokolu, DNS, fungování prohlížečů a klient-server modelu může bez problémů začít s tvorbou REST API. Doporučením je rovněž orientace v principech objektově orientovaného programování v JS.

Koa
Jednoduchá křivka učení, vyžaduje od studenta znalosti na podobné elementární úrovni jako Express. Vzhledem k tomu, že se jedná o novější projekt vývojářského týmu Expressu, pro lepší porozumění technologie osobně doporučuji se na Koa zaměřit až po studiu Expressu.

Nest.js
Středně těžká křivka učení, hlavně pro juniorní vývojáře. Rozsáhlejší dovednostní požadavky na začínajícího programátora – ke znalostem moderního JS se přidávají zkušenosti s TypeScriptem (využívání čistého JS se nedoporučuje). Nest.js vyžaduje více času na pochopení složitějších konstrukcí (roury, guardy, interceptory, atd.) a celkové filosofie Angularu.

3.4.4 Middleware

Middleware je způsob zapouzdření různorodých funkcionalit, jež se zaměřují na zpracování příchozího požadavku a jeho přípravu/úpravu pro funkci obsluhy, případně předčasné ukončení odpověď-požadavek cyklu.

Express
Middlewarové funkce jsou v Expressu uspořádány do řetězce a volány postupně, v přímém pořadí. Práce probíhá se třemi hlavními argumenty middlewaru – objekt požadavku, objekt odpovědi a funkcí *next()*.

Koa
Dle vlastní dokumentace a porovnání [17] Koa do svého jádra z principu nezabaluje žádný middleware, a raději místo něj nabízí sadu uhlazených a jednoduchých metod. Pokud se middleware v Koa aplikaci použije, tak není kompatibilní s ostatními frameworky, zčásti i kvůli využívání kontextového objektu *ctx*, který „nahrazuje“ první dva standardní argumenty Expressu *req* a *res*.
Pořadí, ve kterém jsou middlewarové funkce Koa aplikace řazené v souboru, je pořadím, ve kterém jsou vykonané. Styl, ve kterém jsou volané, se jmenuje **kaskádovým** a připomíná chování datové struktury zásobník.
Nest.js
V defaultním nastavení se middleware Nestu neliší od middlewaru Expressu [12]. Stejným způsobem se přistupuje k objektům *Request* a *Response* a volání dalšího middlewaru se zajišťuje pomocí funkce *next()*. Jeho vykonávání je s Expressem obdobné.

3.4.5 Architektura

Express
Není předem dána, vývojář si sám určuje strukturu projektu. Patří do kategorie názorově nevyhraněných frameworků, čímž zajišťuje určitou svobodu vývojářům, ale zároveň předává na jejich ramena obrovskou zodpovědnost za kvalitu výsledného produktu.
Koa
Není předem dána, stejně jako v případě Expressu se jedná o vysoce názorově nevyhraněný framework. Koa poskytuje vývojářům větší možnost sebevyjádření tím, že je vybízí k vývoji jakéhokoliv potřebného midddlewaru nebo tím, že nediktuje, jakým způsobem má vypadat projektová struktura a nevnucuje jmenné konvence.
Nest.js
Jasná architektura s několika jednoduchými komponentami. Nest se zakládá na třech hlavních komponentách, což jsou řadiče (controllers), providery a moduly. Modulární struktura, kterou Nest „dědí“ od Angularu, pomáhá organizovat aplikaci do jednotných funkčních bloků. Jsou přesně stanoveny struktury složek a použité jmenné konvence.

3.4.6 Zpracování chyb

Zpracování chyb definuje, jak aplikace reaguje na chyby, které se v ní vyskytnou. Jedná se o důležitou součást vývoje softwaru, vzhledem k tomu, že jakákoliv uživateli zobrazená chyba synchronního nebo asynchronního původu kazí uživatelskou zkušenost a přímo ovlivňuje fungování vašeho businessu.
Express
Express přichází s defaultní vestavěnou obsluhou chyb (error handler), aby se zabránilo pádu aplikace (viz dokumentace Expressu [10]). Znamená to, že ji již není třeba implementovat pro počáteční rozběhnutí aplikace. Jedná se o middleware vložený úplně na konec zásobníku middlewarů námi vytvořeného

kódu (s **error** argumentem na začátku). Defaultní obsluha chyb dokáže zpracovat pouze synchronní chyby, realizace odchycení asynchronních výjimek je zcela odpovědností programátora.

Koa

Aby se zabránilo pádu aplikace, Koa přichází s defaultní vestavěnou obsluhou chyb, která je o hodně jednodušší, než to, co jsme pozorovali u Expressu a zároveň ve výchozím nastavení zpracovává všechny druhy chyb nezávislé na tom, jestli se jedná o chyby synchronního nebo asynchronní charakteru. Vlastní obsluha chyb (rovněž middleware) se má implementovat na začátku řetězce kvůli kaskádovému volání middlewarových funkcí.

Nest.js

Nest přichází s hotovou zabudovanou vrstvou výjimek, jejímž účelem je trasování a zpracování všech neošetřených výjimek v celé aplikaci (synchronní a asynchronní). Mechanismu, který odchytává výjimky a je provolán po handleru řadiče, se říká **filtr** na výjimky.

3.4.7 Komunita a dokumentace

Express

Nabízí rozsáhlou dokumentaci a má vynikající komunitní podporu díky své popularitě a „vyspělosti“. Jedná se o nejstarší nodový framework, proto počet dostupných materiálů můžeme ohodnotit jako nadměrně uspokojivý.

Koa

Open source komunita je mnohem menší ve srovnání s Expressem. Výrazně limitované množství volně dostupných výukových materiálů oproti jiným prozkoumaným frameworkům.

Nest.js

Nejlepší z vybraných frameworků, co se týče stavu dokumentace, nabízí přehledný postup pro tvorbu komplexnějších aplikací srozumitelný i pro začátečníky. Omezené množství dostupných tutoriálů vzhledem k poměrně krátkému životu samotného frameworku a jeho živému vývoji. Největší rozrůstající se TS Node.js komunita.

3.4.8 Údržba zdrojového kódu a rychlost vývoje

Express

Poslední hlavní (major) verze Expressu `5.x` se vyvíjí několik dlouhých let (podle Githubu od roku 2015 [1]) a zatím to nevypadá, že by se v nějaké rozumné době dočkala oficiálního přechodu do stavu RTM (release to marketing).

Koa

Údržba projektu probíhá na pravidelném základu, ovšem release nové hlavní verze v blízké budoucnosti není plánován.

Nest.js

Nejživější repozitář ze všech prozkoumaných. Pravidelná údržba a kolaborace vývojářů, rychlá odezva na reportované problémy a systematické vypouštění

[1] https://github.com/expressjs/express

hlavních releasů s frekvencí jednou za 8 až 12 měsíců zřejmě poukazuje na to, jakým směrem se aktuálně Node.js komunita ubírá.

Kapitola 4

Rešerše existujících tutoriálů

Pro tvorbu rozumně postavené konstrukce výukových materiálů si nikdy nevystačíme s pouhým čtením oficiální dokumentace frameworků, je to pouze nutná podmínka. I přesto, že se opíráme o výsledky provedeného výzkumu znalostních požadavků na juniorního Node.js vývojáře, měli bychom zhodnotit obsah existujících konkurenčních produktů, jejich potenciál a schopnost aspoň částečně pokrýt potřeby námi definované cílové skupiny. Eventuálně se můžeme inspirovat způsobem výkladu konkrétních konceptů.

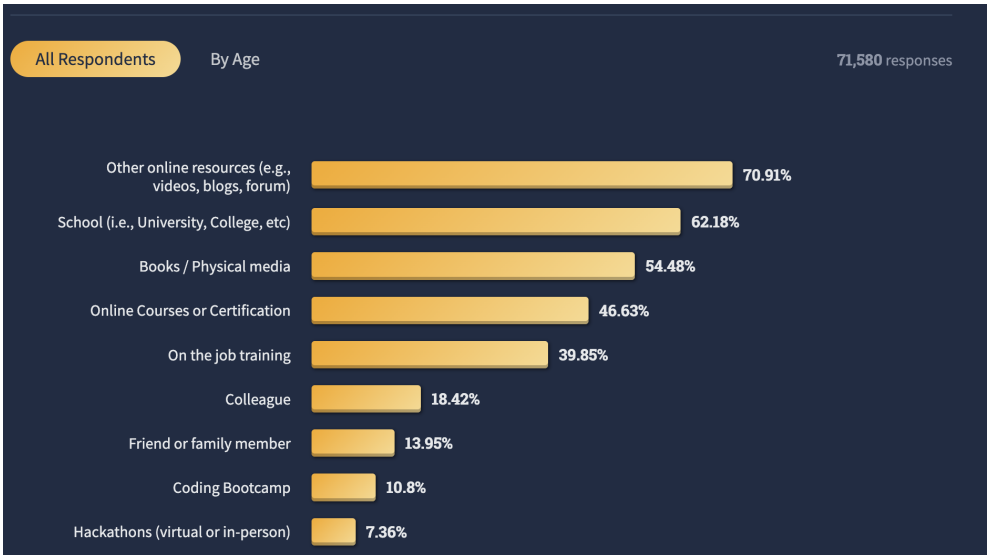
4.1 Formulace požadavků a zvolení formy reprezentace

Požadavky na výukové materiály se vyvíjí z potřeb označené cílové skupiny, které bych ráda shrnula do následujícího seznamu:

- pokrytí Express, Koa a Nest.js frameworků, jejich komparace
- detailnější rozbor specifik frameworků odpovídající potřebám juniorních programátorů za pomoci praktického příkladu
- využití českého jazyka
- dostupnost ve veřejném prostoru

Online forma výukových materiálů dostala přednost před papírovou podobou hned z několika důvodů, jako je, kupříkladu, snadná dostupnost materiálů, možnost zajistit výuku zadarmo bez prvotních nákladů na zajištění tisku u vydavatele nebo proveditelnost pravidelných úprav a aktualizací zdrojů, což je klíčovým nárokem na popis technických projektů, vzhledem ke stále trvajícímu vývoji jak zvolených technologií, tak i programovacích jazyků. Nicméně jednou z mála výjimek, kterou bych ráda ocenila, je kniha „Express in Action“ pana Evana Hanha [8], jehož snahou je frekventovaná modernizace obsahu kapitol, o niž jsem se opírala při tvorbě vlastních tutoriálů.

Validitu vybraného formátu potvrzují i výzkumy společnosti Stack Overflow z let 2021 a 2022, které měly více než 70 000 respondentů (viz 4.1). Až 70 % participantů se naučilo programovat právě z online zdrojů. Mladší



Obrázek 4.1: Učení se programátorskému řemeslu – výzkum společností Stack Overflow z roku 2022 [30]

respondenti (pod 45 let) mají podle průzkumu tendenci se učit z online kurzů a dalších online zdrojů, když starší dotázaní preferují tradiční média jako knihy nebo školní výuku, která v případě výuky Node.js frameworků, jak už bylo poznamenáno, není v České republice na vysokých školách zajištěna.

Dalším pozitivním zjištěním ze stejného zdroje byl fakt, že během posledního roku se učení kódování a programování z online zdrojů zvýšilo o 10 %, což taktéž hraje ve prospěch zvolené formy reprezentace výsledných materiálů.

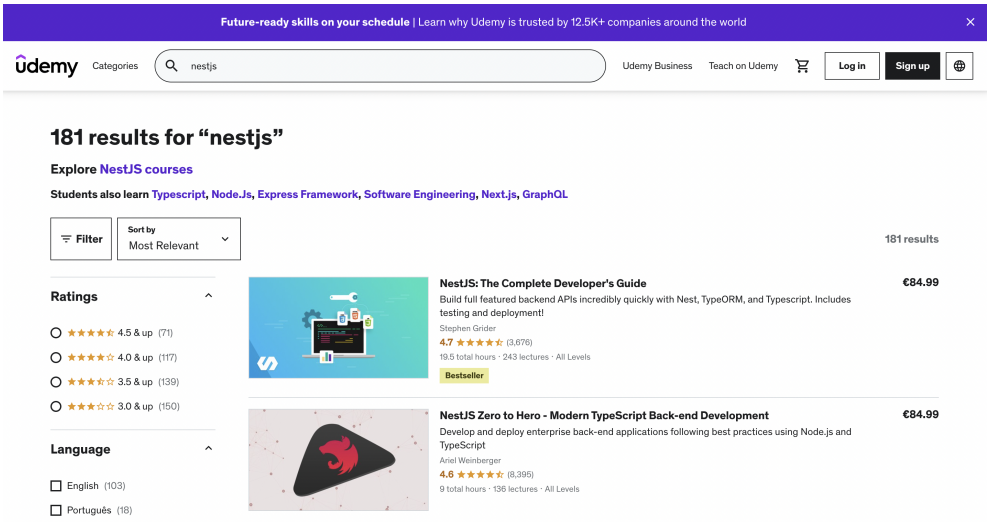
4.2 Hodnocení konkurence

Zkoumání knižních ekvivalentů na trhu nedopadlo úspěšně. I přes bohatou, přehlcenou nabídku, žádná z nich neodpovídala postavenému zadání – obsah byl primárně založen na jednom backendovém frameworku, jen málo z nich bylo přeloženo do češtiny a žádná z publikací nenabízela uživateli možnost detailního porovnání třech zvolených frameworků.

Dalším krokem bylo pátrání po existujících tutoriálech/článcích schopných aspoň částečně pokrýt vybrané požadavky, proto do rešerše byly zahrnuty jak komerční, tak i volně dostupné produkty.

4.2.1 Udemy

Udemy nabízí nesmírně bohatou a širokou nabídku kurzů pro úplné začátečníky až mírně pokročilé. Jsou kvalitně a promyšleně zpracovány, soustředí se na ty nejpodstatnější a klíčové koncepty každého z frameworků, vysvětlují ty nejzásadnější detaily. Osobně jsem s těmito kurzy začínala a považuji je za skvělý start pro všechny budoucí Node programátory. Bohužel musím uznat, že každý kurz je zaměřen na konkrétně jeden framework, což neodpovídá



Obrázek 4.2: Ukázka Udemy výukových kurzů [23]

základní myšlence této práce. I přes veškeré pokusy o zaměření Udemy platformy na mezinárodní trh, žádný z nich se nenabízí v českém jazyce. Kurzy jsou k dispozici na adrese: `https://www.udemy.com`

Částka za takové kurzy se pohybuje mezi 13 až 100 Eury. Platforma Udemy neumožňuje si pořídit předplatné, které by se v daném případě studentům dost vyplatilo.

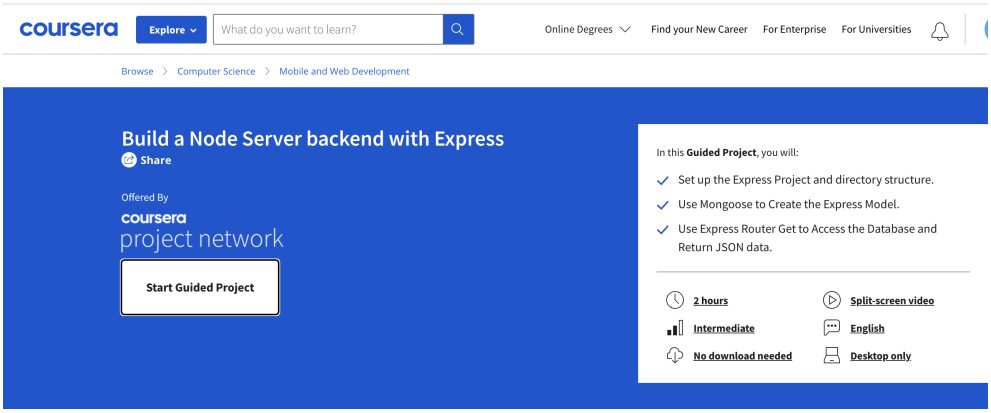
4.2.2 Coursera

Coursera je platforma nabízející rozmanitou sadu kurzů, jež se liší jak obsahově, tak i typem výuky (předem připravená videa, pravidelné online schůze) nebo formou nabízených materiálů. Cenová politika je tu rovněž odlišná. Některé vybrané kurzy uživatel obdrží zdarma, jako je například "Server-side Development with NodeJS, Express and MongoDB", které zaštiťuje Hongkongská univerzita vědy a techniky.

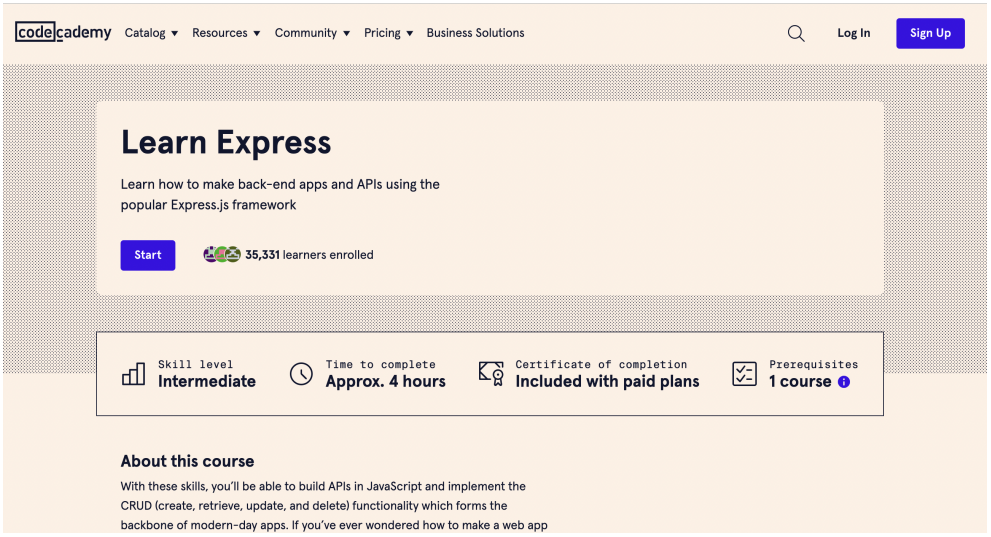
V případě ostatních kurzů se cena může vyšplhat na hodnotu 1000 dolarů, kterou považuji za nepochopitelnou, jelikož jejich obsah se nijak výrazně neliší od nabízeného obsahu na Udemy. Překvapilo mě, že i přes velkou popularitu Coursery je nabízený počet kurzů věnovaných výuce Node.js minimální, zároveň žádný z nich nepokrývá Koa a kurz obsahující Nest.js framework s popisem fundamentálních osnov je k dispozici pouze ve španělštině.
Materiály od společnosti Coursera jsou k dispozici na stránkách:
`https://www.coursera.org`

4.2.3 Codecademy

Kurzy nabízené Codecademy jsou podle business plánu rozděleny na dvě kategorie: základní (Basic) a Pro verze, která funguje na myšlence pořizování měsíčního/ročního předplatného (cena se pohybuje v rozmezí 20 až 39 dolarů v závislosti na typu placení). Bohužel všechny kurzy, které jsou zaměřeny na



Obrázek 4.3: Ukázka Coursera výukového kurzu [24]

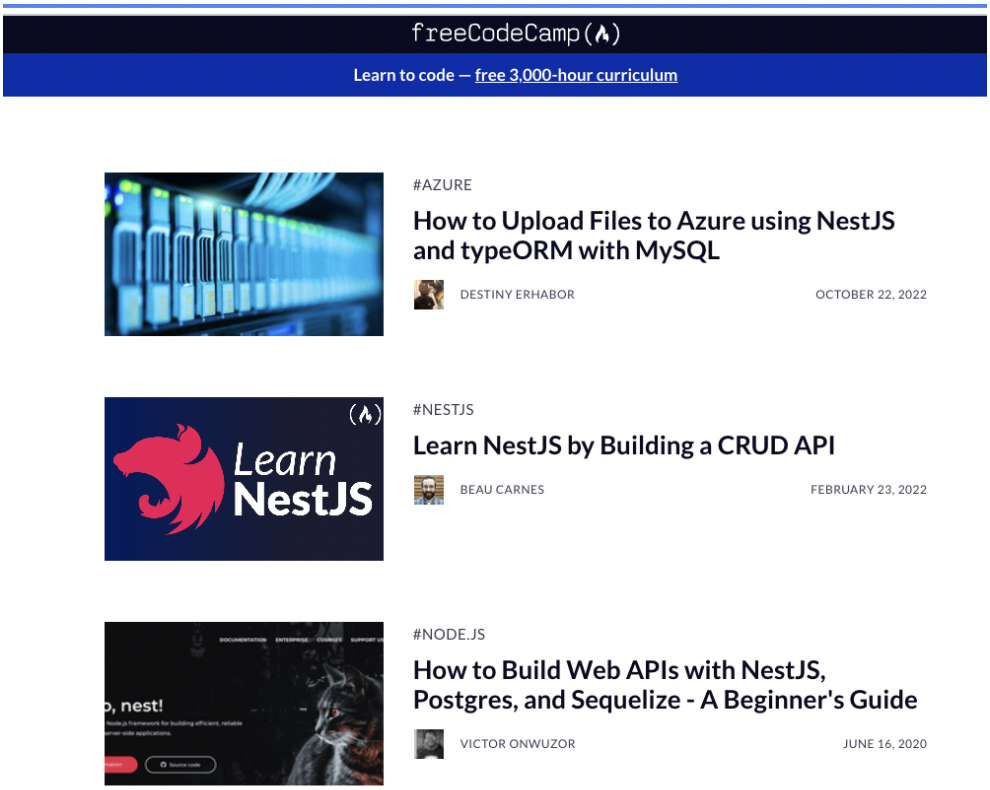


Obrázek 4.4: Ukázka kurzu Expressu od společnosti Codecademy [25]

výuku Nodu, využívají pouze Express framework. Současně bych ráda upozornila, že komerční verze kurzů Codecademy jsou nesmírně podrobně a kvalitně zpracované, jejich autoři si dali záležet nejen na obsahové stránce popisu Express frameworku, ale i na dalších okolních dovednostech backendového vývojáře, jako je virtualizace/kontejnerizace nebo verzování kódu. Rovněž je třeba zdůraznit, že žádný z neplacených kurzů neobsahuje detailnější přehled žádného ze zmíněných frameworků. Materiály od společnosti Codecademy jsou k dispozici na adrese: `https://www.codecademy.com/`

4.2.4 FreeCodeCamp

Samotný název platformy naznačuje, že se jedná o volně přístupné textové materiály, články, online kurzy a interaktivní lekce programování. Mluvíme o neziskové organizaci, jíž fandím celým svým srdcem, podporuji a souhlasím s jejich primárním cílem udělat programování více dostupnější pro všechny



Obrázek 4.5: Ukázka FreeCodeCamp nabídky kurzů/tutoriálů Nest.js [22]

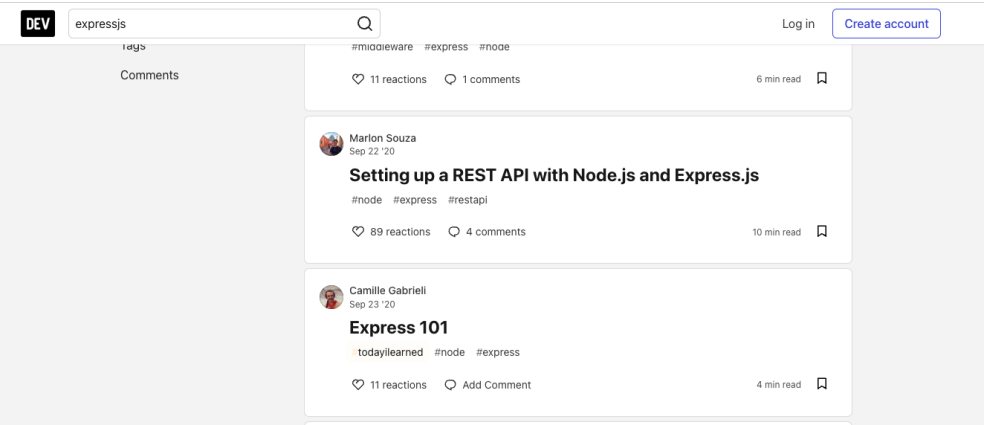
zájemce nezávisle na finančních možnostech (což je částečný důvod vzniku této práce). FreeCodeCamp nabízí poměrně aktuální podrobné tutoriály jak pro Node.js Express framework, tak i pro typescriptový Nest.js. Kratší návod na tvorbu menší aplikace v Nestu pouze nakousne část funkcionality, ale nepokryje všechny její aspekty a stěžejní struktury.

Materiály od společnosti FreeCodeCamp jsou k dispozici na adrese: `https://www.freecodecamp.org`

4.2.5 Medium a DEV

Medium a DEV jsou servery pro publikování článků a tutoriálů. Medium se zaměřuje na široké spektrum témat, naproti tomu DEV výhradně na softwarové inženýrství. Zdroje jsou předloženy z velké části v textové podobě v kombinaci se zdrojovými kódy (pokud se jedná o články zaměřené na softwarové inženýrství), odkazy na videa nebo dokonce audio verze článků, jež značně zlepšuje přístup k informacím lidem se zrakovým postižením nebo dyslexií (k dispozici jsou články na stránce [21]). Navíc DEV je komunitou sdružující navzájem si pomáhajících vývojářů softwaru (viz [20]) a fungující na stejném principu jako komunita FreeCodeCamp. Obě bychom mohli zařadit do skupiny „Od vývojáře vývojářům“.

Při zaměření se na články týkající se běhového prostředí Node a nodových



Obrázek 4.6: Nabídka článků věnujících se Expressu na DEV platformě [20]

frameworků, vidíme, že každý článek je věnován drobnějším částem problematiky. Rozdíl oproti ostatním zdrojům spočívá v tom, že servery nenabízí přesný postup pro studium zvolené tematiky v porovnání s Courserou nebo Udemy, zájemce musí přesně vědět, co hledá. Čtenář si zde může dohledat mnoho článků od úvodních, seznamujících čtenáře s platformou, až po experimentální návrhy a hlubší analýzy komponent, které jsou ve vývoji. Pro zkušenější programátory tyto platformy začínají plnit roli hlavního zdroje nápomocných informací, čímž postupně zastiňuje všeobecně známý Stack Overflow, hlavně použitím jiného svěžího konceptu, než časem ověřený (a fungující) přístup *otázka-odpověď*.

Medium je k dispozici na adrese: `https://medium.com/` Odkaz na DEV: `https://dev.to/`

4.2.6 OKškolení, NICOM.CZ, GOPAS

Ve všech třech případech se jedná buď o interaktivní nebo skupinové odborné školení, jež jsou k nalezení na českém trhu, které v nejlepším případě pokrývají rozsah a elementární základy pouze Express frameworku. Často jsou k dispozici v kombinací s dalším výkladem, např., zásad moderního JavaScriptu, nebo jsou součásti fullstackového dvoudenního kurzů. Osobně nedokážu ocenit zmíněný způsob výuky vzhledem k patrným problémům vyplývajícím z nedostatků flexibility, časovým restrikcím nebo omezeností využití materiálů poskytovaných na podobných školeních vzhledem k jejich často komerční povaze. Bohužel se jedná o nejpopulárnější kurzy na českém trhu, které svým obsahem alespoň zčásti pokrývají obsah, jenž plánuji v tutoriálech nabídnout.

4.2.7 Porovnání aktuální konkurence

Z porovnání s konkurencí znázorněnou v tabulce 4.1 lze jasně poznat, že výukové kurzy jsou primárně business, a že mnou naplánovaný kurz v češtině by neměl konkurenci. Pokud se tedy někdo z dříve zmíněných person rozhodl,

Obrázek 4.7: Skupinové GOPAS kurzy

společnost	jazyk	forma	cena	zohlednění všech frameworků
Udemy	angličtina	VZ	12 až 100 Eur	ano, při nákupu třech kurzů
Coursera	angličtina	VZM	zdarma až 1 000 Eur	ne, (2/3)
Medium	angličtina	MZ	zdarma	ano
Codecademy	angličtina	VZM	20 až 39 dollarů	ne, (1/3)
DEV	angličtina	ZMP	zdarma	ano
GOPAS	čeština	MZO	17 600 Kč	ne (1/3)
NICON.CZ	čeština	MZO	Od 7190 Kč	ne (1/3)
OKškolení	čeština	MZO	10 000 Kč	ne (1/3)

Tabulka 4.1: Porovnání aktuální konkurence na rok 2023
(M je zkratkou pro textové materiály, V - pro videa, Z - zdrojové kódy, P - podcasty a O - osobní konzultace)

že by chtěl absolvovat ucelený kurz Node.js frameworku v češtině, musel by zaplatit poměrně vysokou částku. Prozatím žádná z českých technických univerzit nenabízí tyto kurzy v potřebném rozsahu, proto studium na univerzitě nemůže sloužit jako východisko pro žádnou personu, na kterou by kurz byl orientován.

Kapitola 5

Návrh výukového ukázkového projektu

Konstrukce výukových materiálů Express, Koa a Nest.js frameworků by se měla zakládat na co největším zdůraznění stěžejních charakteristik použitých technologií, než na použité doméně. Proto jedním ze zásadních rozhodnutí při jejich tvorbě bylo využití stejné oblasti / stejného návrhu domény, který by po seznámení a dočtení sekce zaměřené na jeden z frameworků poskytl studentům možnost se snadno ponořit do další frameworkové sekce a zaměřit se hlavně na komparativní rozdíly.

Do seznamu nároků na výukový projekt, podle kterého budou vyučovány v plánovaných materiálech Node.js frameworky, jsem zařadila dva požadavky – unikátnost a využitelnost v reálném životě. Veřejný prostor je plný jednoduchých open-source projektů, jako jsou ToDo List aplikace, internetové obchody, aplikace na předpověď počasí apod. Mým cílem bylo se všemi možnými způsoby vyhnout všedním projektům a navrhnout smysluplnou aplikaci, která by mohla napomoci menším podnikům, případně by plnila charitativní účely. Všimla jsem si, že většina menších zoologických zahrad v republice nemá připravené webové stránky a napadlo mě, že by to mohl být ten správný prostor pro realizaci.

Vzhledem k tomu, že v rámci nodových výukových materiálů budeme logicky chtít se zaměřit na backendový vývoj (serverovou stranu client-server modelu), jehož neoddělitelnou částí je tvorba API [18] a [29], dodržíme známou a osvědčenou strategii separace frontendových a backendových aplikací, kterou v rámci vytvořených materiálů chceme propagovat. Proto i přesto, že všechny pro rešerši zvolené Node.js frameworky podporují šablonovací systémy (template engines), jejich využití v tutoriálech zamezíme. Z jedné strany pro nás toto řešení otevře značnou sadu výhod, které čtenář dokáže do budoucna ocenit, jako jsou, např., jednoduchá škálovatelnost aplikace, využití výhod modularity nebo rychlejší produkční nasazení. Z druhé strany, frontend a backend očekává od vývojářů odlišnou sadu dovedností, tudíž jsem měla konstantní obavy z „přesycení“ informacemi juniorních zájemců o zmíněnou disciplínu. Posledním argumentem, proč jsem se zaměřila pouze na tvorbu API a těsnou práci s databází, je hlavní orientace naše cílové skupiny, která je „obrněná“ základy frontendu na aspoň minimální úrovni a od kurzů nodových frameworků má jiná očekávání, než tvorbu šablon či implementaci logiky na straně klienta.

Rozhodnutí pro zvolení standardu pro tvorbu backendového API padlo poměrně kvapně – i přes veškerý obdiv vůči narůstající popularitě GraphQL[1], převládající dominance RESTových aplikací během posledních desetiletí a hlavně neustálá poptávka po RESTu jako znalostním minimu pro dokonce i juniorní vývojáře backendových aplikací ovlivnila směr, kterým jsem se rozhodla v ukázkovém projektu vydat. Zvoleným formátem pro uložení a přenos dat je JSON[2] vzhledem k jeho jednoduchosti, strukturovanosti a poměrně rychlému procesu serializace/deserializace.

Prvním základním krokem před zahájením tvorby návrhu jakéhokoliv API by měla být definice zdrojů a případů užití prostřednictvím provedení analýzy funkčních požadavků.

5.1 Definice funkčních požadavků

Následující seznam možných funkčních požadavků byl připraven na základě detailnějšího průzkumu většího množství existujících volně dostupných webových aplikací zoologických zahrad z celé České republiky.

F1 Navrhovaný systém poskytuje přehled chovaných zvířat v zoologické zahradě včetně základních informací o jejich umístění ve vědecké klasifikaci, výskytu, potravě a délce života.

F2 Ke každému zvířeti systém přiděluje jednoho osobního chovatele.

F3 Systém zajišťuje vyhledávání zvířat podle jejich stáří, klasifikačního zařazení, příbuzenského vztahu s jiným zvířetem (tj. podle rodiče) nebo existence fotografické podoby v profilu.

F4 Systém zajišťuje třídění specifikovaných výsledků vyhledávání zvířat podle data příjezdu do zoologické zahrady, případně data narození, pokud se narodilo v zajetí.

F5 Systém dovoluje klientovi zjistit taxonomické zařazení uvedených zvířat.

F6 Systém nabízí uživatelům přehled o různorodých expozicích.

F7 Systém poskytuje možnost provedení offline (rezervace) objednávky.

F8 Systém rozlišuje jeden druh zboží, které nabízí svým návštěvníkům k rezervaci, a to jsou vstupenky různých druhů (plnohodnotné nebo zlevněné pro speciální kategorie návštěvníků). Nicméně je systém připraven

[1] Grafový dotazovací jazyk na úrovni aplikační vrstvy zveřejněný společností Facebook v roce 2016, jehož účelem je vyhnout se / napravit stávající problémy RESTu (komplikovaná agregace dat, vysoká provázanost mezi frontendovým rozhraním a backendovými řadiči, rozsáhlý počet endpointů atd.) [32]

[2] Jak JSON, tak i XML mohou být využity při tvorbě RESTových rozhraní. XML však není nejlepší volbou pro tvorbu API. Jeho datový model není kompatibilní s tím, jak většina jazyků včetně JS data modeluje. Co se týče procesů serializace, je významně pomalejší než JSON.

pro pravděpodobné budoucí rozšíření nabídky poskytnutého zboží (fyzické výrobky, dárkové karty/poukazy, sponzoring ve tvaru adopce či patronace), který předloží nejenom prostřednictvím offline objednávky, ale i ve formě prodeje.

F9 Provedení objednávky není omezeno věkem uživatele aplikace.

F10 Systém nabízí uživatelům přehled o v zoologické zahradě plánovaných akcích (krmení, charitativní sbírky, komentované prohlídky).

F11 Systém rozeznává dva typy uživatelů: přihlášené a nepřihlášené.

F12 Přihlášený uživatel má právo jak na správu veřejně nabízeného obsahu systému, tak i na náhled do provedených objednávek či modifikaci jejich stavů.

Jsem si vědoma, že zvolené požadavky nemusí odpovídat veškerým potřebám menších zoologických zahrad, které by chtěly vyvinutý a již hotový projekt použít pro vlastní účely, avšak se snažím přizpůsobit výsledný produkt třem nejdůležitějším požadavkům – a to jsou univerzalita, jednoduchost a rozšiřitelnost. Rovněž bych ráda upozornila i na to, že představený seznam nepředstavuje popis toho, jak by měl vypadat finální funkční projekt v produkci podle představ reálného klienta. Účelem výukových materiálů je tvorba **MVP** (minimálního životaschopného produktu), což znamená, že se chci v tutoriálech detailně zaměřit pouze na klíčové a základní koncepty aplikace, jež by podle mě měly největší výukovou hodnotu.

Některé funkční požadavky, které do MVP zahrnuty nejsou, ale přitom byly vymezeny jako součást důkladné analýzy podobných softwarových produktů zoologických zahrad v rámci výzkumu, bych zde rovněž ráda zmínila. Jejich pokročilejší implementaci uvedu jako nadstavbu pro existující projekt v budoucí inkrementaci tutoriálu. Rovněž je to z mého hlediska nutno zmínit i pro ty netrpělivé čtenáře tutoriálů, kteří by se touto cestou preferovali vydat sami, ale netuší čím začít.

- Součástí projektu je rovněž systém pro správu webového rozhraní (CMS).
- CMS rozlišuje minimálně dvě základní role: administrátor a editor.
- Administrátor by měl mít k dispozici neomezená práva na tvorbu, editaci či případné mazání záznamů / obsahu stránek.
- Přístupová práva uživatelů ze skupiny editor by měla být řízena administrátorem.
- Systém umožňuje provést adopci konkrétního zvířete nebo celého druhu (finanční příspěvek na předem určené stanovené období), stát se jeho patronem (provedení jednorázové platby) nebo pozvat zvíře na oběd.
- Budoucí systém nabízí jak pro jednotlivce, tak celé skupiny, různorodé charitativní programy výukového či rekreačního charakteru, které by se daly pořídit online prostřednictvím nákupu či rezervace.

- Prodej rozmanitého množství fyzických výrobků je další funkcionalitou aplikace, jež by se měla aktivovat pouze administrátorem v systému pro správu rozhraní.
- U každého zboží je možné stanovit, zda je určeno pro online prodej, nebo je k dispozici pouze rezervační formou.
- Po provedení objednávky proběhne vygenerování faktury ve formátu PDF, kterou systém následně zašle uživateli na e-mail uvedený v objednávce.
- Je zajištěno automatické mazání objednávek, emailů a vygenerovaných faktur starších tří let.

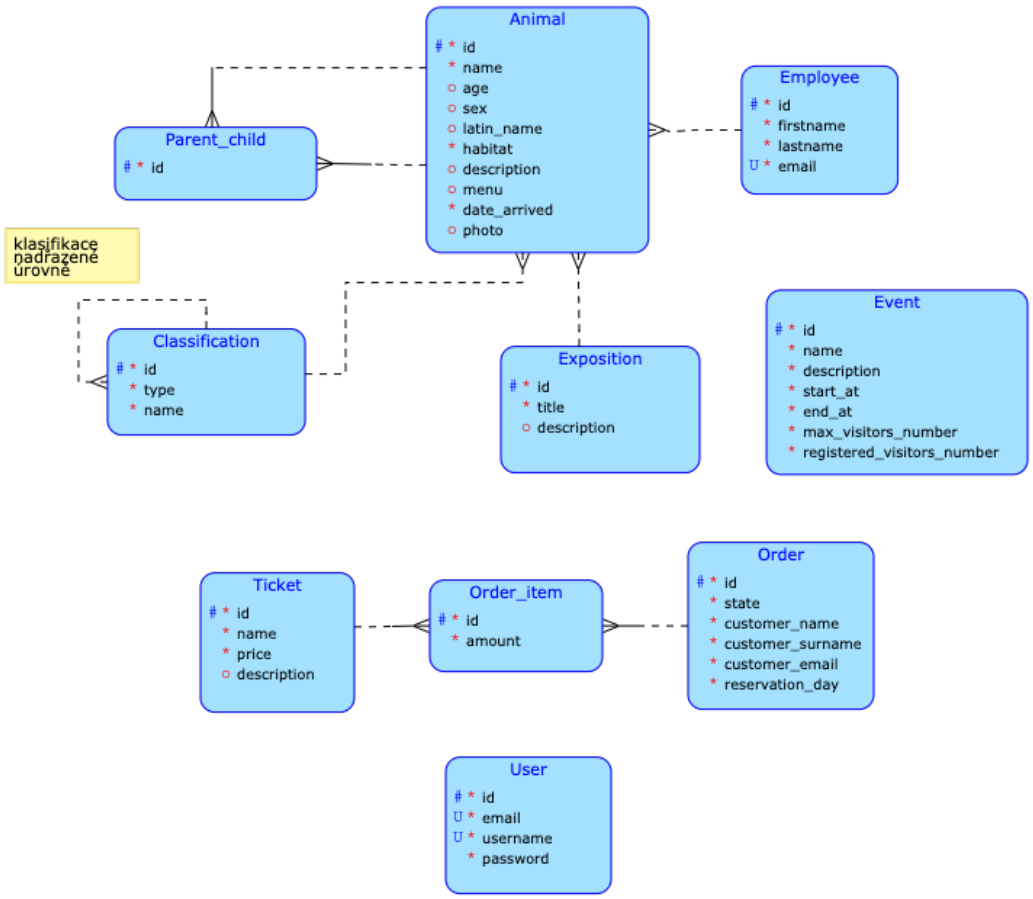
5.2 Tvorba konceptuálního databázového modelu

Vzhledem k větší komplexitě formulovaných funkčních požadavků na aplikaci oproti řadě existujících a volně dostupných tutoriálů a z toho vyplývajícího navrženého databázového modelu (viz obr. 5.1), považuji za nutnost tento model rozdělit na čtyři jádra logicky provázaných nebo naopak izolovaných struktur pro lepší reprezentaci budoucích zdrojů, kterým bude věnována zvláštní pozornost:

1. Pětice *Animal*, *Exposition*, *Eployee*, *Classification* a *Parent_child* poskytující detailnější přehled chovaného zvířete, jeho původu, klasifikace, místa výskytu, chovatele zodpovědného za zvíře a vztahu k představeným expozicím.
   V daném případě bychom mohli věnovat větší pozornost entitě *Classification*, neboť pozorujeme 1:N rekurzivní vztah, kde se jedná o nepovinnou účast determinantu. Entita *Parent_child* se na začátku může tvářit jako tradiční příklad dekompozice kardinality M:N, jejíž výsledkem však není slabá ale silná vztahová entita [3]. Zvolený případ je ale dostatečně „extravagantní“ i tím, že se jedná o M:N vztah rekurzivní relaci realizující myšlenku komplexních vztahů zvířat v zoologické zahradě – zvíře může mít jak několik rodičů, tak i několik dětí[4]. Představenou část modelu můžeme číst následovně: „Zvíře je vždy přiřazeno k jednomu chovateli. Zaměstnanec zoologické zahrady může, ale nemusí mít na starosti až N zvířat. Zvíře patří do maximálně jedné expozice.“
2. Prodej základního vstupného je obstarán spojením třech entit – *Ticket*, *Order* a *Order_item* (poslední je výsledkem provedené dekompozice).
   Kdybychom chtěli převést použitý model do lidské řeči, znělo by to následovně: „Objednávka může obsahovat několik položek různých typů vstupenek“. V situaci, že bychom měli v plánu rozšířit prodej produktů

[3] Slabá vztahová entita je částečně nebo plně identifikována vztahem k jiné entitě. Silná entita je naopak definovaná svými vlastními klíči.

[4] Omezení na počet rodičů zvířat nebo kontrolu na existenci záznamu se stejnými cizími klíči, ale opačnými vazbami ve vytvořené tabulce, by se povedlo implementovat na straně databázového serveru pomocí *ADD CONSTRAINT...CHECK* konstrukcí.



Obrázek 5.1: Konceptuální databázový model projektu, na němž se zakládají výukové materiály pro Nest.js, Koa a Express frameworky

o další položky jiného druhu, než vstupné (přidat charitativní výrobky, dárkové poukazy atd.), použili bychom v modelu mezi vztahovou entitou a entitami produktů ARC relaci odpovídající XOR operaci.

3. Osamocená entita *User* se sice nevyskytuje v žádné relaci, přitom hraje stěžejní roli v aplikaci. Registrace, přihlášení a celý autorizační mechanismus je postaven jen na záznamech v *users* tabulce (samozřejmě nesmíme opomenout zásluhu generátoru a verifikátoru JWT tokenů nebo algoritmů na hashování hesel).

4. Další ojedinělá entita *Event* je ztvárněním všech prošlých či plánovaných akcí konaných v areálu podniku. Jejím primárním záměrem je vedení statistik zájemců o tyto akce.

koncový bod (endpoint)	GET	POST	PUT	DELETE
/animals	✓	✓	–	–
/animals/*:id*	✓	–	✓	✓
/animals/*:id*/child/*:childId*	–	✓	–	✓
/animals/*:id*/parent/*:parentId*	–	✓	–	✓
/expositions	✓	✓	–	–
/expositions/*:id*	✓	–	✓	✓
/employees	✓	✓	–	–
/employees/*:id*	✓	–	✓	✓
/classifications	✓	✓	–	–
/classifications/*:id*	✓	–	✓	✓
/classifications/*:id*/tree	✓	–	—	–
/tickets	✓	✓	–	–
/tickets/*:id*	✓	–	✓	–
/orders	✓	✓	–	–
/orders/*:id*	✓	–	✓	–
/auth/register	–	✓	–	–
/auth/login	–	✓	–	–
/events	✓	✓	–	–
/events/*:id*	✓	–	✓	✓

Tabulka 5.1: RESTové rozhraní výukového materiálu

5.3 Návrh RESTového rozhraní

Přejdeme k popisu koncových bodů, přes které budeme přistupovat ke dříve zmíněným zdrojům 5.1. V rámci budování projektu nejenom zpřístupníme určité kolekce nebo jejich konkrétní prvky, ale některé z nich rovněž dokážeme filtrovat, vyhledávat nebo třídit na základě parametrů URI dotazu, či modifikovat odpověď pro případ, že potřebujeme přistoupit k dalším datům pro různé typy objektů.

Ukážeme si příklad aplikace heterogenních parametrů pro zajištění provedení filtrace a vyhledávání prvků kolekce *Animal* a následnou modifikaci odpovědi klientovi dostupné pod endpointem `/animals`, kde se navíc používají vnořené řetězce dotazů:
`/animals?age[gt]=1&age[lt]=10&sort=asc&parentsIncluded=true&childrenIncluded=true&parentId=2&classificationId=3&photo=false`
Zároveň se některé parametry dají aplikovat i na konkrétní prvky zmíněné kolekce:
`/animals/:id?classificationTree=true&parentsIncluded=true&childrenIncluded=true`, kde *id* je identifikátorem zdroje typu *zvíře*. Více o připravených endpointech s podrobnějšími informacemi o tom, jak vypadají požadavky a odpovědi, je k nalezení v připravené Postman kolekci v kořenovém adresáři u každého ze zdrojů výukových projektů dostupných na GitHubu: `https://github.com/just-node-it`.

5.4 Využité technologie

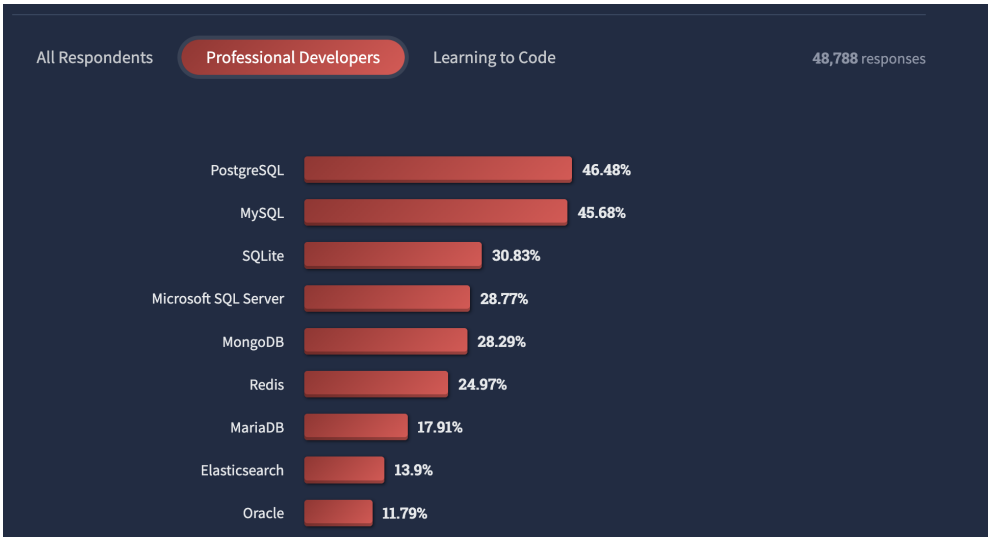
V této kapitole se práce věnuje všem druhům technologií, které se použily pro vývoj ukázkových projektů v třech Node.js frameworcích. Jak už jednou bylo zmíněno, hlavní myšlenkou tvorby výukových materiálů fungující na pozadí bylo upřednostnění demonstrace specifik samotných frameworků na reálných příkladech, než věnování se výuce odlišnosti podpůrných technologií pro tvorbu serverových aplikací. Proto na základě různých postupů – využití výsledků provedeného výzkumu, následování trendů a doporučení v odvětví nebo hledání knihoven/balíčků „zužitkovatelných“ jak v typescriptových, tak i javascriptových projektech – se vybral následující seznam.

5.4.1 Technologie užité pro backend

Node.js – Nest.js, Koa, Express
Uplatnění zmíněného běhového prostředí založeného na otevřeném softwaru V8, což je javascriptový engine, a trojici jeho frameworků by pro čtenáře nemělo být překvapující, nicméně je zde uvedeme pro konzistenci výpisu.

npm a yarn
Node **P**ackage **M**anager neboli NPM je oficiálně největším světovým registrem softwarů na světě (a vztahujícím se k němu metadat) [26]. Znovu vymýšlet kolo, když se nejlepší mozky planety soustředí na výrobu aut s nulovou uhlíkovou stopou, je stejně zbytečné, jako příprava vlastnoručně psaného kódu místo využití řady užitečných sdílených balíčků nabízených NPM databází (žádná komerční řešení nebudou využita v rámci tutoriálů). NPM je rovněž defaultní správce balíčků (stejnojmenný software, ale o zkratku se tentokrát nejedná), který je dodáván při instalaci Node.js. Má na starosti veškerou správu balíčků třetích stran. V rámci Express a Koa tutoriálů se použije **npm**, avšak při práci s Nestem byl ze studijních a ilustrativních důvodů využit alternativní správce – **yarn**, který je podle vlastní dokumentace a poznatků vývojářů (např., dle [16]) rychlejší pro sestavování aplikace než npm díky svému pokrokovějšímu algoritmu. Nicméně vytvořené ukázkové projekty nevynikají nadměrnou komplexitou, proto uvedený rychlostní rozdíl nebude k poznání.

PostgreSQL
Volně dostupný objektově-relační systém pro správu databází (ORDBMS), jehož zrození se datuje do roku 1996 [15]. PostgreSQL rozšiřuje SQL jazyk a nabízí bohatou sadu funkcí a datových typů spolu s možností definovat svoje vlastní, za což je všestranně oblíbený uživateli. Jedná se o spolehlivé a robustní řešení, které našlo svoje uplatnění při vývoji podnikových a mobilních aplikací a start-upů. Podle statistik z ledna 2022 je PostgreSQL celosvětově čtvrtý nejpoužívanější RDBMS (viz 5.2), který je k dispozici vývojářům zcela zdarma, přitom jeho funkcionality nezaostávají za nejpopulárnějšími komerčními řešeními (Oracle, Microsoft SQL Server). Rovněž se v dotazníku



Obrázek 5.2: Žebříček celosvětově nejoblíbenějších RDBMS v roce 2022 podle statistik [30]

pro Node.js programátory z výzkumu provedeného v diplomové práci Postgres umístil na první pozici.

Na pravděpodobnou otázku, proč nebyla využitá MongoDB, již participanti označili za preferovatelnou NoSQL databázi, odpovím, že znalosti návrhu relačních databází a umění sestavovat efektivní SQL dotazy jsou pevným vědomostním základem pro určení toho, jaký typ databáze je nejvhodnější volbou pro budoucí softwarové projekty. Tyto klíčové koncepty, jež backendový vývojář-začátečník musí ovládat na dobré úrovni, jsou základem pro hlubší pochopení výhod NoSQL databází.

Prisma

Prisma je softwarem s otevřeným zdrojovým kódem (open source, OSS) považovaný za ORM nové generace (dle její vlastní dokumentace [14]), jež do sebe zahrnuje několik částí: *Prisma Client* (software poskytující možnost tvorby databázových dotazů přímo v aplikacích), *Prisma Migrate* (nástroj pro databázové migrace) a *Prisma Studio* (zabudované rozhraní pro vizualizaci a jednoduchou manipulaci s daty).

REST API

Datově orientovaná architektura, která je, jak popisuje Vinay Sahni [33], založená na několika zásadních pilířích: „všechno je zdroj“, každý prostředek je identifikovatelný pomocí jedinečného URI, používají se pouze standardní HTTP metody, komunikace probíhá bezstavově a zdroje mohou mít více reprezentací.

Docker

Docker je nástrojem, softwarovou platformou zaměřenou na tvorbu, manipulaci a nasazení aplikačních kontejnerů, jejichž primárním cílem je usnadnění

vývoje pomocí zabalení aplikací a jejich konfigurace do balíčku, který může být následně nasazen. V případě jednoduchého lokálního vývoje projektu zoologické zahrady se využije *postgres* obraz (image) databáze.

5.4.2 Technologie užité k vývoji serverových aplikací

VS Code
Zvolené vývojové prostředí, podle mého názoru jedna z mála povedených technologií Microsoftu nabírající popularitu a velkou ozvěnu ve vývojářském světě. Zdarma nabízený, kvalitně zpracovaný textový editor pro tři hlavní operační systémy a s rozmanitou nabídkou snadno konfigurovatelných rozšíření pro každý vkus. Je nejvíce doporučeným prostředím ve mnou provedeném výzkumu.

Git
Nástroj pro verzování existujícího kódu. Rovněž byl využit GitHub jako vzdálený repozitář pro uložení softwarového projektu. Veškerý zdrojový kód vyvinutých aplikací je veřejný a je k dispozici na adrese:
`https://github.com/just-node-it`

Postman
Tvorba API dotazů pro testování vytvořených koncových bodů byla zajištěna prostřednictvím platformy Postman (API Client). V rámci vytvořené kolekce byly využity proměnné prostředí a napsán kratší skript pro organizaci automatického přihlášení uživatele v případě dotazovaní se na zabezpečené endpointy.

Kapitola 6

Tvorba kurzu

Následující kapitola rozsáhle popisuje metodiku vývoje výukových materiálů včetně tvorby jejich obsahu a další distribuce. Výsledný projekt je k nalezení a přečtení na webových stránkách: `https://just-node-it.eu/`. Ráda bych poznamenala, že uvedená doména je registrovaná na moje jméno pomocí správce domén a webhostingové služby Webglobe. Zvolený název podle mého hlediska zajišťuje tutoriálům větší kredibilitu.

6.1 Připravené materiály

Odpovědi na mnoho zásadních otázek, které bylo třeba položit v rámci zvoleného zadání, měly určit výukový směr aplikovaný na všechny nodové tutoriály. Značně jsem při pátrání po odpovědích čerpala inspiraci z několika málo kvalitně zpracovaných a na dnešní dobu aktuálních materiálů, jako je, např., kniha „Practical Nest.js: Develop clean MVC web applications“ od Daniela Correa a Greg Lima nebo „Server Side development with Node.js and Koa.js Quick Start Guide“ od Olayinka Omole [13]. Adaptovala jsem jejich výukovou metodiku ke svým vlastním potřebám, což mi dovolilo se soustředit na ty nejdůležitější a nejtěžší aspekty práce na tutoriálech:

- stanovit obsah a rozsah nabízených materiálů
- nalézt a označit správné meze mezi nutnými juniorními znalostmi a dovednostmi přesahující tuto hranici
- vymezit principy, struktury, návrhové vzory a architektury, kterým by se měla věnovat co největší pozornost
- určit vlastnosti frameworků, které by se měly odrazit v kapitolách
- strukturovat nodovou aplikaci tak, aby bylo možné ukázat osobitost vybraných technologií, ale přitom se neodklonit od deklarované univerzality projektu

Ukážeme si společné rysy vytvořených projektů a tutoriálů a následně si je rozebereme dopodrobna.

Všechny mnou vytvořené projekty popsané v materiálech se staví na komunikaci s PostgreSQL databází rozběhnuté v docker kontejneru pomocí nástroje ORM Prisma. Snažím se nezakládat pouze na vysvětlení jednoduchých CRUD dotazů, kterými je sice třeba z pochopitelných důvodů začít, avšak s postupem času, jak se aplikace s každou novou tématickou sekcí rozrůstá a začíná se obalovat komplexnější business logikou, aplikujeme pokročilejší principy dotazování – vnořené dotazy, vnější spojení (*JOIN* operace), které rovněž implementuji pomocí zvoleného ORM nástroje. Ve výukových materiálech uvádím a vysvětluji vlastnosti databázových transakcí, které pomocí Prismy vykonáme několika způsoby. Představení práce s Prismou kulminuje tvorbou rekurzivního dotazu (na zjištění taxonomického zařazení zvířat), který je implementován s využitím tradiční PostgreSQL syntaxe.

Validaci a sanitaci/transformaci požadavků příchozích na server provádím pomocí nástrojů třetích stran – NPM balíčků Joi[1], class-validator[2] a express-validator[3]. Zacházení s nimi studentovi podrobně vysvětluji v každém tutoriálů tak, aby po dokončení celé výukové trojice měl začínající vývojář větší přehled o nabízených možnostech a vybral si validátor podle vlastních potřeb.

Nezbytným při tvorbě podobného druhů aplikací zůstává zajištění autentizace a autorizace uživatele, kterou studentovi ve všech výukových materiálech předvedeme za pomocí implementace generování a ověřování JWT tokenů. Zajištění těchto procesů pomocí session je rovněž klíčová znalost pro vývojáře, nicméně je pro začátečníky komplikovanější na pochopení, proto ji přidáme do kurzu až v rámci příští iterace. Rovněž se neobejdeme bez hashování hesel, ke kterému dodáme několik teoretických základů a vysvětlivek.

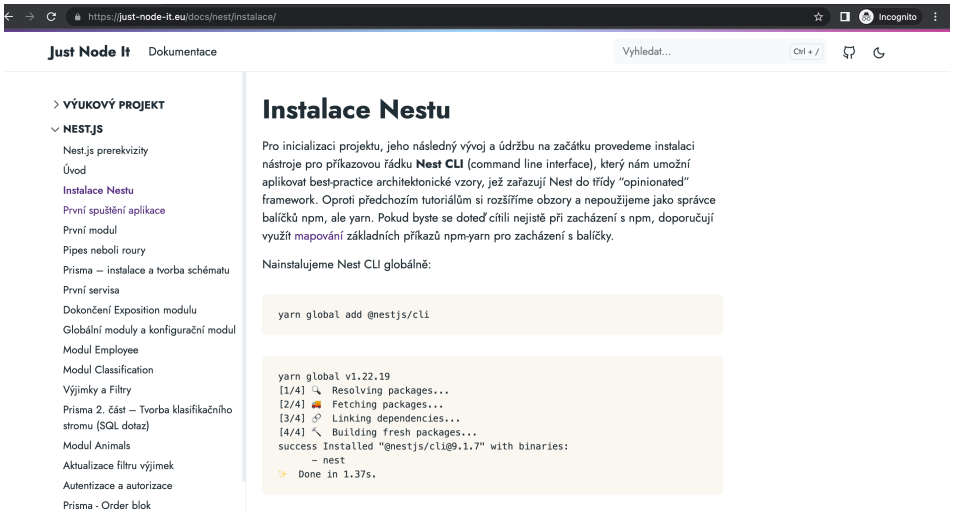
I přesto, že tutoriály jsou připravené pro čtení v jakémkoliv pořadí, v kapitole s detailním popisem projektu osobně studentům doporučuji jejich procházení začít Nestem, v případě, že svoje preference dávají využití TS oproti JS a mají aspoň minimální zkušenosti s vývojem nodových aplikací. V jiných případech radím se primárně zaměřit na Express kvůli jednoduššímu pochopení principu zřetězení middlewarů.

Ve všech projektech jsme použili moderní JavaScript s použitím Promis a async/await „syntaktického cukru“.

6.1.1 Express

V rámci práce na kurzu Expressu, který je tvořen primárně pro úplné začátečníky, jsou dodatečně probrány základy HTTP protokolu, URL, komunikace mezi klientem a serverem a fungování *odpověď-požadavek* cyklu. Zde se uživatel seznámí nejenom s budováním REST API, ale s nepostradatelným konceptem směrování (routování), stavby a řetězení middlewarových funkcí, zajištění vlastní obsluhy chyb a návrhu vrstvené (layered) architektury.

[1] https://joi.dev/
[2] https://www.npmjs.com/package/class-validator
[3] https://express-validator.github.io/docs/



Obrázek 6.1: Ukázka vytvořeného Nest.js kurzu

6.1.2 Koa

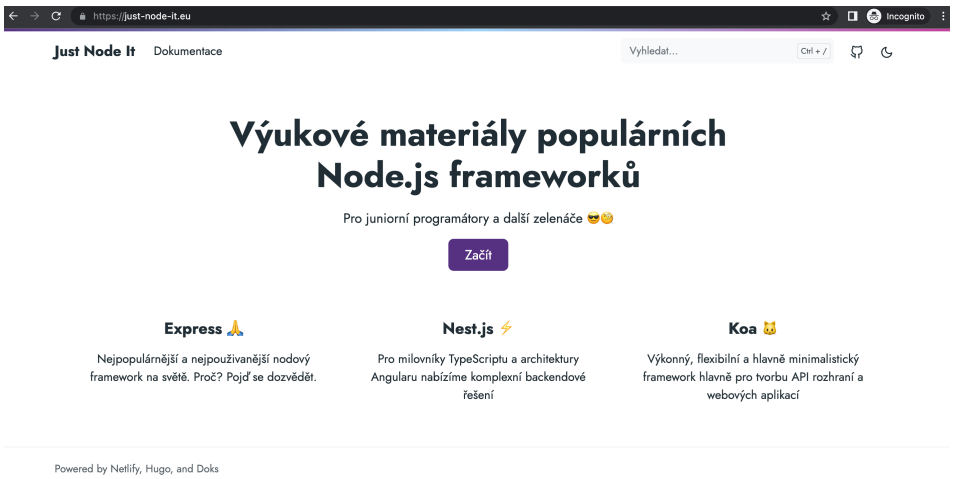
Struktura Koa projektu se nijak kardinálně neliší od struktury námi dříve založené Express aplikace. Osobně toto rozhodnutí pokládám za velkou výhodu poskytující váženému čtenáři možnost se co nejvíce zaměřit na podstatné rozdíly frameworků samotných, než na odlišnosti pro projekty zvolených architektur. V těchto tutoriálech během konstrukce restového API hlavně poukážeme na zásadní rozdíly fungování middlewarů a defaultní obsluhy chyb oproti Expressu a použijeme jiný validátor požadavků. Všechny zmíněné stěžejní body v Express kapitolách jsou taktéž popsané v Koa s provedenou modifikací a adaptací pro nový framework.

6.1.3 Nest.js

Výukové materiály pro Nest.js jsou odhadově dvakrát rozsáhlejší než předchozí dva, což se vysvětluje jak větší komplexitou a rozmanitostí Nestem nabízených struktur, tak i pravděpodobně mou osobní zálibou vůči danému frameworku. V Nestu studentovi představím MVC architekturu, seznámím ho s architekturou inspirovanou Angularem, vysvětlím koncepty providerů, řadičů, služeb, rour, filtrů a interceptorů a nakonec uvedu názornou ukázku implementace JWT strategie pomocí guardů. Samotná práce s Prisma ORM zde nabývá novou „zážitkovou úroveň“ díky typové bezpečnosti v kombinaci se zvoleným jazykem TS.

6.2 Platforma

Pro psaní dokumentačních souborů tutoriálů byl zvolen Markdown pro jeho široké celosvětové použití při profesionálním formátování technické dokumentace. Další klíčový důvod volby Markdownu spočíval v jeho využití na



Obrázek 6.2: Domovská stránka https://just-node-it.eu

GitHubu (jemuž se žádný juniorní či seniorní softwarový vývojář ve své profesní kariéře nevyhne), který ho automatický přerenderuje do HTML. Opírajíc o výsledky práce Marka Kodra [1] skončilo původní hledání programu pro konverzi vstupního Markdown souboru na HTML rovněž u open source projektu *Claat* [27], který od uživatelů vyžadoval pouhou instalaci jazyku Go. Nicméně se časem zjistilo, že Google Codelabs neodpovídají třem požadavkům, které by student na výukové materiály kladl – fulltextové vyhledávání v souborech, rozdělení do sekcí s názornou stromovou strukturou a větším zanořením (sekce, podsekce ...) a jednoduchou orientací v seznamu podkapitol. Bohužel se během delšího zkoumání zjistilo, že samotný projekt není vývojáři z Googlu v dostačující normě udržovaný, proto se od zvoleného řešení rozhodlo odstoupit.

Novým kandidátem na generátor statického obsahu, pro nějž se následně rozhodlo, byl Hugo[4] v kombinaci s Doks[5] tématem připraveným převážně pro tvorbu dokumentace (viz 6.2).

6.3 Hostování

Někteří registrátoři domén nabízí statický web k doméně zcela zdarma, a to i použité Webglobe. Existují i hostingy přímo specializované na statické weby, nicméně se rozhodlo pro vlastní řešení self-hostingu, kde v případě provozování statického webu je to opět ta „nejlevnější“ varianta. Hosting statických webů patří k nejméně náročným činnostem vzhledem k jejích jednodušší konfiguraci a správě.

Libovolný vlastní webový server se základním nastavením tuto funkci splní. Příkladem poslouží nginx, Apache, lighttpd nebo Tomcat. Následně stačí

[4]Webový framework, generátor statického obsahu (https://gohugo.io/), který má vynikající podporu pro Markdown [28]

[5]https://getdoks.org/

zajistit průchodnost portu 80 do širokého internetu, nastavit DNS záznamy zakoupené domény a vysněný `https://just-node-it.eu/` žije. Osobně si myslím, že se jedná o skvělý základ pro to si rozšířit znalosti v oboru sítí a administrace, které nejsou součástí této diplomové práce...ale fantazii se meze nekladou.

Kapitola 7

Hodnocení kurzu českými firmami

Jedním z požadavků na diplomovou práci bylo hodnocení vytvořených kurzů vývojářskou komunitou. Proto bych v této kapitole ráda předložila výsledné zprávy týkající se připravených výukových materiálů napsané mediorními a seniorními programátory z různých českých firem (nebo českých poboček mezinárodních firem), jako Quanti s.r.o., DEK a.s. a DoDo Czech s.r.o.

Jméno a příjmení: Martin Štefan
Aktuální pracoviště: Quanti s.r.o.
Hlavní pozice a náplň práce: Senior PHP Developer, taktéž zastává Tech Leadra a týmové DevOPS. V PHP profesně pracuje přes 10 let (5 let z toho samostudium). DevOPS se věnuje profesně přes 2 roky (samostudiem přes 5 let).

> Po přečtení materiálů, které popisují 3 JS frameworky, v pořadí Nest, Express a Koa, tak za stávajících a nových informací bych se přiklonil k práci s frameworkem Nest. Nest framework je dle tří tutoriálů pro full stack programátora, který se chce začít na FE věnovat novým věcem, nejideálnější cestou, protože dost věcí bude pro něj už hodně „známých". Hlavně základní koncepce MVC.
>
> Každý z tutoriálů byl psaný trochu jinou formou, kde se i následně spoléhalo, že čtenář už něco málo ví o základech JavaScriptu a TypeScriptu. Tutoriál se snažil vše vysvětlit a i podat na více místech vysvětlení, co přesně se tam vlastně děje, ale hlavně proč tomu tak je.
>
> Nest tutoriál, kterým jsem začínal, byl opravdu velice obsáhlý, z pohledu juniora je hodně detailně psaný, ale z pohledu seniora a můj osobní názor je, že tento tutoriál obsahoval až moc „omáčky“ okolo. Občas bylo potřeba pročíst se dlouhým odstavcem, kde se čtenář dozvěděl jen něco málo, co by stačilo v jedné větě. Ale jak píšu výše, pro juniora je lepší vysvětlit a trochu opsat, co se tam děje.
>
> Express a Koa tutoriál měly mnohem méně „omáčky“, ale zde se předpokládá, že čtenář už zpracoval tutoriál pro Nest, kde byly věci vysvětlené detailněji. Z pohledu seniora byly Express a Koa tutoriál

tak akorát pro potřeby se naučení a rozpoznání rozdílů mezi třemi prezentovanými frameworky.

Jméno a příjmení: Ing. Vít Neckář
Aktuální pracoviště: DEK a.s.
Vysokoškolské vzdělání: absolvent Otevřené informatiky na škole ČVUT FEL
Hlavní pozice a náplň práce: Profesně se softwarovým vývojem zabývá 5 let. Aktuálně pracuje jako ERP developer. Má zkušenosti převážně s Delphi a T-SQL, v poslední době se taktéž zabývá samostudiem jazyka C#.

Do čtení tutoriálů jsem šel jako člověk, který má zkušenosti převážně s objektově orientovanými jazyky, jako jsou C# nebo Delphi, a nemám velké zkušenosti s prací na backendových projektech v JavaScriptu a TypeScriptu. Tutoriály jsem tedy bral jako bránu k hlubšímu poznání tajů javascriptového vývoje.

Začal jsem čtením nejrozsáhlejšího tutoriálu, který se týkal frameworku Nest.js. Ze začátku bylo pro mě chápání TS kódu trochu náročnější, po čase se to ale zlepšilo. Oceňuji, že tutoriál nejen vysvětluje, jak se má něco psát, ale i proč se to tak má psát a jak věci uvnitř fungují, což je podle mě důležité pro to, aby čtenář správně pochopil pravidla a postupy vývoje softwaru, které poté může správně aplikovat ve své praxi. Následují tutoriály pro frameworky Express a Koa, které již nejsou tak rozsáhlé, jako tutoriál pro Nest, je to ale dáno převážně tím, že staví na znalostech získaných při studiu prvního frameworku. Zároveň, jak píše sama autorka, Koa je evolucí Express a některé postupy jsou zde velmi podobné, což taktéž zjednodušuje a zkracuje učení.

Po přečtení tutoriálů bych si osobně vybral práci v Nestu nebo Expressu, převážně kvůli velké komunitě obou frameworků. Nest bych asi před Expressem preferoval, vzhledem k tomu, že je typově bezpečný. Tutoriály mi přišly čitelné a, jak jsem psal výše u odstavce o Nestu, líbilo se mi, že autorka nechává čtenáře nahlédnout pod pokličku a vysvětluje, proč se věci dělají daným způsobem. Tutoriály taktéž popisují rozdíly mezi třemi zmiňovanými frameworky, což čtenáři pomůže ve výběru toho, ve kterém se mu bude nejlépe pracovat. Na druhou stranu bych ale ocenil omezení rozsáhlých souvětí, ve kterých se autorka vyžívá. Doporučuji se držet pravidla používání jednoduchých vět, případně souvětí o maximálně třech větách. Značně se tím zlepší čitelnost a zároveň to nijak neuškodí sděleným informacím. Taktéž mi v některých větách přišel zvláštní použitý slovosled, což je ale pochopitelné vzhledem k původu autorky. Většina těchto nedostatků ale byla později opravena.

Jméno a příjmení: Jan Polášek
Aktuální pracoviště: logistická společnost DoDo Czech s.r.o
Poznámka: Byly nabídnuté pouze Nest.js výukové materiály

Hlavní pozice a náplň práce: Profesně se zabývá frontendovým vývojem, primárně ve frameworku Angular s drobným přesahem do nativního vývoje pro platformu Android. 10 let zkušeností s JS, 5 let Angularem. Ve volném čase se pak učí Flutter a Swift.

> Na úvod bych zmínil, že jsem měl k dispozici pouze tutoriál na Nest.js a nikoliv ty další dva na Express a Koa, které jsou na mnoha místech referencované. Nicméně jak Express tak Koa jsem v minulosti používal a věděl jsem tedy, o čem je řeč. A hodnotit budu z pozice člověka, který má s JavasSriptem přes 10 let zkušeností, byť až 5 let zpátky jsem se jím začal zabývat profesně - konkrétně Angularem, se kterým Nest sdílí množství základních konceptů. Každopádně před přečtením jsem o Nestu samotném nic nevěděl.
>
> Na rozdíl od mnohých tutoriálů tento vysvětloval nejen „jak“, ale i „proč“, což je extrémně důležité. Bez alespoň částečné znalosti toho, jak daný framework/knihovna funguje, je snadné udělat chyby, které se jen těžko hledají a opravují. Díky tomu byl užitečný i pro mne, přestože je psán formou vhodnou spíše juniorům a zabýval se i některými úplnými základy.
>
> Jsem přesvědčený, že tento tutoriál obsahuje všechny základní informace, které by vývojář mohl do začátků potřebovat, a buduje pevný základ, na kterém je možné dále stavět a postupně objevovat další taje Nest.js. Za svůj život jsem si přečetl nesčetně tutoriálů a mnohé z nich vynechávají některé důležité informace, ale z tohoto jsem takový dojem neměl a nenarazil jsem na žádnou sekci, u které bych si říkal „počkat, počkat, a jak se sem vůbec dostanu?“
>
> Z čistě subjektivního úhlu pohledu bych vytknul používání české terminologie a to z toho důvodu, že mimo akademické prostředí, které je obecně odtržené od reality, se nepoužívá. Z vlastní zkušenosti vím, že jakákoliv technická debata ohledně programování naopak probíhá skoro až „angličtinou s českou gramatikou“, což je zase opačný extrém.
>
> A na závěr bych se v případě TypeScriptu obecně vyhýbal frázi „typově bezpečný“. Komukoliv, kdo s TypeScriptem nemá zkušenosti, to může dávat falešný pocit bezpečí, které ale reálně neexistuje, protože se nejedná o skutečné typy, pouze glorifikované anotace. Jakmile se v kódu objeví typ any, otevírají se brány mnohým Javascriptovým chybám jako např. snahy o sečtení čísla s undefined. A tradiční "Uncaught TypeError: Cannot read property 'x' of undefined"jistě také vykoukne.

Jméno a příjmení: Ing. Jan Nejtek
Pracovní zkušenosti: automotive průmysl, smluvní vývoj pro ŠKODA AUTO a.s., Porsche Engineering Services s.r.o. a Tesla a.s.
Vysokoškolské vzdělání: absolvent Otevřené informatiky na škole ČVUT FEL

Hlavní pozice a náplň práce: Profesně se zabývá backendovým vývojem, zejména pro účely sběru dat s přesahem do embedded a industrial prostředí. 7 let zkušeností s .NET frameworkem, propojení s Pythonem (včetně frameworku Flask) a nízkoúrovňovým vývojem.

> Naprostá většina projektů, do kterých jsem se profesně zapojil, vyžadovala komponentu pro uložení a zpracování zaznamenaných dat s dodatečnými nároky na vlastní vnitřní logiku nebo jinou „skrytou“ inteligenci. Samotné databázové programy či servery jednoduše nestačily a tak jsem se dostal tímto poměrně nezvyklým způsobem do oboru tvorby backendů. Ačkoliv má praxe příliš nesouvisí s vývojem webových stránek, rostoucí požadavky na výkon při sběru dat v průmyslovém či automobilovém prostředí mne postupně dovedly od začátků s PHP k produktům rodiny .NET s občasným zapojením Pythonu až k Node.js. Toto prostředí mě zaujalo nabídkou své vysoké datové prostupnosti díky asynchronní architektuře.
>
> Jelikož jsem byl po spoustu let, stejně jako mnoho mých kolegů z oboru, nakažen dogmatickou nenávistí k JavaScriptu, doposud jsem nezískal s Node.js větší zkušenost. Do čtení tutoriálů jsem tedy vstoupil jako člověk s poměrně obsáhlým povědomím ze světa IT a softwarového vývoje a architektury backendů obecně, ovšem jako JavaScriptový nováček. Začal jsem tutoriálem pro Express, jelikož o něm jsem z různých zdrojů zaslechl nejvíce.
>
> Ocenil jsem kapitolu se specifikací aplikace pro zoologickou zahradu, která je nezávislá na jednotlivých tutoriálech. V mé praxi jsou požadavky na různé projekty či experimenty definovány obdobně, takže jsem se se cítil jako doma. JavaScript ani TypeScript jsem neshledal nijak zvláště obtížný, ale to dost možná může být mojí relativně nadstandardní znalostí jiných, mnohdy obdobných skriptovacích či programovacích jazyků.
>
> Kdybych si jako teoretik mohl vybrat, nejradši bych ve své praxi používal Nest.js, líbí se mi jeho pocit stability a pravidelnosti pramenící z jeho názorové vyhraněnosti. Bohužel, tato vyhraněnost je evidentně postavená okolo tvorby backendů pro webové a mobilní aplikace, a v mé praxi by byla limitující. Čistě prakticky je pro mě asi nejvhodnější Koa díky velkým možnostem přizpůsobení, ať už co se týká business logiky, klientů, specifických možností zpracování požadavků, přístupů k nezvyklým datovým úložištím a podobně.
>
> Kdo ví, třeba časem vznikne něco jako Nest-Industrial, který bude vhodnější pro IoT a průmyslová měření. :) Závěrem bych rád autorce pochválil praktické využití ORM i v natolik komplexních případech, jako jsou rekurzivní relace. Tutoriály je vzhledem k nadstandardnímu grafickému zpracování radost číst a velice jsem ocenil i přítomnost funkčního vyhledávání.

Kapitola 8
Závěr

Hlavní záměr této práce se vyvíjel čtyřmi klíčovými proudy, jež na sebe navzájem navazovaly či se pravidelně proplétaly. K nim patří provedení hlubší analýzy a průzkumu znalostních a technických požadavků juniorního Node.js vývojáře, realizace rešeršní práce třech nejpopulárnějších Node.js frameworků pro provedení jejich kompletního komparativního výzkumu, tvorba univerzálního projektu ve všech zmíněných frameworcích a publikace popisující tvorbu podrobných tutoriálů pro zaučení nových juniorních programátorů. Co se týče prvního postaveného cíle, povedlo se mi sehnat relativně velkou dotazovanou skupinu respondentů pro to, abych považovala výsledky dotazníku za dostatečné a uspokojivé. Současně pokládám za velký úspěch uskutečnění porovnávání hlavních modulů, komponent a s nimi souvisejících struktur Expressu, Koa a Nestu, díky čemuž se mně osobně prohloubilo chápaní myšlenkového pochodu vývojářů a jejich způsobu uvažování nad definovanými problémy. Snažila jsem se v diplomové práci prostým a co nejvíce obyčejným způsobem odkázat na zjištěné zásadní rozdíly zvolených technologií. Výsledek této práce může hrát zásadní roli pro jakéhokoliv programátora, usnadnit jeho rozhodnutí při volbě „správného“ frameworku v závislosti na požadavcích, které si na software pokládá, což může za normálních okolností trvat až několik dní.

Tu nejzávažnější část práce, kterou jsem si osobně vzala moc blízko k srdci, vzhledem k tomu, jak velkou roli by hrál její význam v mém kariérním životě před několika lety, tvořila konstrukce samotných výukových materiálů. Musím se přiznat, že vzhledem k neexistenci konkurence mnou vyráběného produktu na českém trhu jsem se ocitla pod větším tlakem, než jsem původně očekávala. Vznikla potřeba nejen nabídnout správnou a rozumnou strukturu výukového projektu pro začátečníky, ale i najít vhodný způsob sdělení informací a znalostí obdržených v předchozích bodech diplomové práce. Usilovala jsem o nalezení zlatého středu mezi potřebou „Nevysvětlovat zbytečně moc, abych udržela pozornost čtenáře“ a „Vysvětlit části tutoriálů dostačujícím způsobem, aby uživatel nebyl nucen vyhledávat kompletní terminologii u jiných zdrojů“. Snažila jsem se horlivě držet myšlenky „Když to neumíš vysvětlit jednoduše, nerozumíš tomu“.

Na základě ohlasů zkušených vývojářů se domnívám, že výsledné tutoriály saturují cíle postavené zadáním diplomové práce. Nicméně bych rada zmínila

další požadavky, o jejichž plnění se dozvím až časem. Za prvé, napomoci světu zničit mylnou představu o extrémní komplikovanosti a náročnosti práce softwarového inženýra. A za druhé, pomoci připravit chybějící lidi na pracovní pozice vývojářů v České republice. Vím, že stejné cíle byly splněné v mojí bakalářské práci, proč by diplomová měla být výjimkou?

Budoucí vývoj platformy `{https://just-node-it.eu` jsem už nejednou naznačila v dřívějších odstavcích, stejně jako i můj nesmírný zájem pokračovat v dalších iteracích materiálů. Otevřený prostor pro pokračování se zdá být neomezený: postup nasazení aplikace, HTTPS a SSL protokoly, NoSQL databáze a jejich škálování...

Na konec bych ráda vyjádřila nesmírnou radost z objevení této koncepce diplomové práce stejně jako nekonečný smutek z toho, že neznám dalšího vývojáře, kterému bych výukové žezlo následně předala. Nicméně pevně věřím v to, že se jedná pouze o začátek nového odvětví.

Literatura

[1] Kodr, Marek. Výukové materiály pro nativní Android. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[2] PYPL PopularitY of Programming Language index. Page Redirection [online]. [03.01.2023]. Dostupné z: https://pypl.github.io/PYPL.html

[3] 64 Node JS Stats that Prove Its Awesomeness in 2022. WebTribunal [online]. [14.12.2022]. Dostupné z: https://webtribunal.net/blog/node-js-stats/#gref

[4] The Hottest Skills in Tech Job Searches. Indeed Hiring Lab [online]. [30.11.2022]. Dostupné z: https://www.hiringlab.org/2018/11/29/hottest-skills-tech-job-searches1/

[5] Business.linkedin.com [online] [30.11.2022]. Dostupné také z: https://business.linkedin.com/content/dam/me/business/en-us/talent-solutions/emerging-jobs-report/Emerging_Jobs_Report_U.S._FINAL.pdf

[6] Casciaro, M., & Mammino, L. (2020). Node.js Design Patterns: Design and implement production-grade node.js. applications using proven patterns and techniques. Packt Publishing Ltd.

[7] ATHAPATHU, Rukshani. Blocking I/O and non-blocking I/O. Medium [online]. 31.08.2017 [10.12.2022]. Dostupné z: https://medium.com/coderscorner/tale-of-client-server-and-socket-a6ef54a74763

[8] HOLOWAYCHUK, T. J., et al. Node. js in Action. Manning Publications Co. LLC, 2017.

[9] Documentation | Node.js. Node.js [online]. [cit. 07.12.2022]. Dostupné z: https://nodejs.org/en/docs/

[10] Express - Node.js web application framework. [online]. [cit. 07.12.2022]. Dostupné z: https://expressjs.com

[11] Koa - next generation web framework for node.js. [online]. [cit. 08.12.2022]. Dostupné z: https://koajs.com/

[12] Documentation | NestJS - A progressive Node.js framework. [online]. [cit. 08.12.2022]. Dostupné z: https://docs.nestjs.com/

[13] OMOLE, Olayinka. Server Side Development with Node. js and Koa. js Quick Start Guide: Build Robust and Scalable Web Applications with Modern JavaScript Techniques. Packt Publishing, Limited, 2018.

[14] Prisma | Next-generation ORM for Node.js & TypeScript. Prisma [online]. [cit. 18.12.2022]. Dostupné z: https://www.prisma.io/

[15] PostgreSQL. PostgreSQL [online]. [cit. 19.12.2022]. Dostupné z: https://www.postgresql.org/

[16] TELSAN, Nick. Node Package Managers in 2022 | Viget. https://www.viget.com [online]. 18.08.2022 [23.12.2022]. Dostupné z: https://www.viget.com/articles/node-package-managers-in-2022/

[17] koa/koa-vs-express.md at master · koajs/-koa. GitHub [online]. [23.12.2022]. Dostupné z: https://github.com/koajs/koa/blob/master/docs/koa-vs-express.md

[18] NANDAA, Anthony. Beginning API Development with Node.js: Build highly scalable, developer-friendly APIs for the modern web with JavaScript and Node.js. Packt Publishing, 2018.

[19] SUFIYAN, Taha. Top 50+ Node.js Interview Questions and Answers for 2023 | Simplilearn. Simplilearn.com [online]. 07.07.2020 [11.12.2022]. Dostupné z: https://www.simplilearn.com/tutorials/nodejs-tutorial/nodejs-interview-questions

[20] DEV Community. DEV Community [online]. [10.12.2022]. Dostupné z: https://dev.to/

[21] Medium – Where good ideas find you. [10.12.2022]. Medium. Dostupné z: https://medium.com/

[22] FreeCodeCamp [10.12.2022]. Dostupné z: https://www.freecodecamp.org/

[23] Online Courses – Learn Anything, On Your Schedule | Udemy. Online Courses – Learn Anything, On Your Schedule | Udemy [online]. Copyright © 2023 Udemy, Inc. [cit. 28.11.2022]. Dostupné z: https://www.udemy.com/

[24] Coursera | Online Courses & Credentials From Top Educators. Join for Free. Coursera [online]. [28.12.2022]. Dostupné z: https://www.coursera.org/

[25] Learn to Code - for Free | Codecademy. Codecademy [online]. [17.11.2022]. Dostupné z: https://www.codecademy.com

[26] npm. npm [online]. [no date] [06.09.2023]. Available from: https://www.npmjs.com/

[27] toolsclaat at main · googlecodelabs/tools. GitHub [online]. [28.12.2022]. Dostupné z: https://github.com/googlecodelabs/tools/tree/main/claat

[28] The world's fastest framework for building websites. The world's fastest framework for building websites | Hugo [online]. [09.01.2023]. Dostupné z: https://gohugo.io/

[29] Backend Developer Roadmap. roadmap.sh [online]. [14.11.2022]. Dostupné z: https://roadmap.sh/backend/

[30] Stack Overflow Developer Survey 2022. Stack Overflow [online]. [11.12.2023]. Dostupné z: https://survey.stackoverflow.co/2022/#developer-profile-learning-to-code

[31] Koa.js Tutorial. Online Tutorials Library [online]. [26.12.2022]. Dostupné z: https://www.tutorialspoint.com/koajs/index.htm

[32] GraphQL | A query language for your API. GraphQL | A query language for your API [online]. [14.11.2022]. Dostupné z: https://graphql.org/

[33] BOJINOV, Valentin. RESTful Web API Design with Node.js. Packt Publishing, 2015.

Kapitola 9

Seznam použitých zkratek

API Application programming interface

JSON JavaScript Object Notation

XML Extensible Markup Language

URI Uniform Resource Identifier

OOP Object-oriented programming

DNS Domain Name System

REST Representational state transfer

ORM Object–relational mapping

CMS Content management system

CI Continuous integration

CD Continuous delivery