

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Effio - webová aplikace pro vytváření testů



Matěj Kotrba

Moravskoslezský kraj

Opava, 2024

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Effio - webová aplikace pro vytváření testů

Effio - web application for creating tests

Jméno: Matěj Kotrba

Škola: Střední škola průmyslová a umělecká, Praskova 399/8,
Opava, 746 01

Kraj: Moravskoslezský kraj

Konzultant: Mgr. Marek Lučný

Opava, 2024

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Opavě dne 10. dubna 2024

Matěj Kotrba

Poděkování

Rád bych poděkoval Mgr. Marku Lučnému za poskytnutí konzultace ohledně tohoto projektu.

Anotace

Výsledkem projektu je funkční webová aplikace pro vytváření a vyplňování testů, které se skládají z různých možností otázek včetně programovací. Aplikace zahrnuje přihlášení přes Google a GitHub. Uživatel vytváří jednotlivé testy výběrem šablony, nebo importem z GIFT formátu, otázek, komentářů a následně upravuje detaily testu. Hotový test může sám zkusit z vlastní kolekce testů, a nebo z komunitního centra, kde se nacházejí komunitou vytvořené testy, po vyplnění testu se uživateli objeví výsledky a známka. Kromě tvorby a vyplňování testů aplikace obsahuje také skupiny, v níž mohou mezi sebou uživatelé např. komunikovat, sdílet testy, zobrazovat statistiky apod., na dříve vyplněné testy se mohou podívat v sekci testové historie. Přehled o svých aktivitách si uživatel může prohlédnout v dashboardu prostřednictvím vizuálních grafů. Dříve vytvořené testy se dají v části kolekce editovat, mazat a také exportovat do GIFT formátu pro použití například v Moodle. Aplikace disponuje zcela responzivním designem se světlým a tmavým režimem.

Klíčová slova

Webová aplikace; online testy; databáze; frontend; backend.

Annotation

The result of the project is a functional web application for creating and taking tests, which consist of various types of questions, including programming questions. The application supports login via Google and GitHub. Users can create individual tests by selecting a template or importing from the GIFT format, including questions, comments, and then customize the details of the test. The completed test can be tried by the user from their own collection of tests or from the community center, where tests created by the community are available. After completing the test, users will see the results and a grade. In addition to test creation and completion, the application also includes groups where users can communicate with each other, share tests and view statistics etc. Users can review previously taken tests in the test history section. An overview of user activity can be viewed in the dashboard through visual graphs. Previously created tests can be edited, deleted, and exported to the GIFT format in a text file for use, for example, in Moodle. The application features a fully responsive design with both light and dark modes.

Keywords

Web application; online tests; database; frontend; backend.

Obsah

Úvod	6
1 Architektura a datový model aplikace	7
1.1 Architektura webové aplikace	7
1.2 Typesafety	9
1.3 Datové modely	9
2 Backend	11
2.1 Založení a konfigurace projektu	11
2.2 Architektura backendu	11
2.3 Databáze	13
2.4 Autentifikace	14
2.5 Cloud hosting	15
2.6 Využité backendové technologie	15
3 Frontend	20
3.1 Design	20
3.2 Responzivita	20
3.3 Přístupnost	21
3.4 Využité frontendové technologie	22
4 Funkce aplikace	26
4.1 Přihlášení	26
4.2 Zobrazování testů	29
4.3 Skupiny	30
4.4 Administrátorská část	31
5 Závěr	33

ÚVOD

V dnešní době jsou webové aplikace běžně využívány pro vytváření testů a kvízů, které poté vyplňují ostatní uživatelé. Prostředí těchto aplikací však často působí neorganizovaně a tvorba testů může být pro běžného uživatele komplikovaná i časově náročná.

Příkladem mohou být testy vytvářené pedagogy v populárním e-learningovém systému Moodle, s nimiž se setkává mnoho středoškoláků i vysokoškolských studentů. Od svých učitelů vím, že uživatelské rozhraní pro tvorbu testů není v tomto systému zrovna přívětivé a zadávání jednotlivých úloh je stojí poměrně hodně času i nervů. Napadlo mě, že by bylo možné inspirovat se jedním ze standardních formátů elektronických testů a pokusit se vytvořit modernější, uživatelsky příjemnější prostředí.

S touto myšlenkou jsem se rozhodl vytvořit webovou aplikaci, pro jejíž název jsem zvolil latinské slovo Effio, která by kombinovala typické funkce programů pro elektronické testování s přehledným moderním rozhraním a dalšími užitečnými prvky. Aplikace tak kromě tvorby testů a kvízů obsahuje i některé prvky typické pro sociální síť, tedy možnost vytváření komunit uživatelů a vzájemného sdílení testů.

Jedním mých z cílů bylo rovněž v praxi si ověřit a využít některé nové technologické postupy, které se nyní prosazují v oblasti tvorby webu. Výsledkem je tzv. „typesafe“ a „serverless“ aplikace, jež nevyžaduje vlastní server, čerpá z možností současných cloudových technologií a zároveň je plně responzivní pro uživatele na jakémkoli zařízení.

V dokumentaci aspoň stručně popisují klíčové využité technologie, postupy a jednotlivé funkcionality celé aplikace. První kapitola se zaměřuje na samotnou architekturu systému a zdůvodňuje zvolený přístup k řešení. V dalších kapitolách se snažím přiblížit technické řešení backendové i frontendové části aplikace a ta poslední popisuje základní funkce Effia. S ohledem na šíři celé práce, množství použitých knihoven i počet řádků kódu nemůže dokumentace v rozsahu doporučovaném pro SOČ ani zdaleka pojmut všechny aspekty projektu, dovoluji si proto odkázat na veřejný repozitář na Githubu: <https://github.com/matej-kotrba/effio>.

Samotná aplikace je provozována na doméně <https://effio.vercel.app/>

1 ARCHITEKTURA A DATOVÝ MODEL APLIKACE

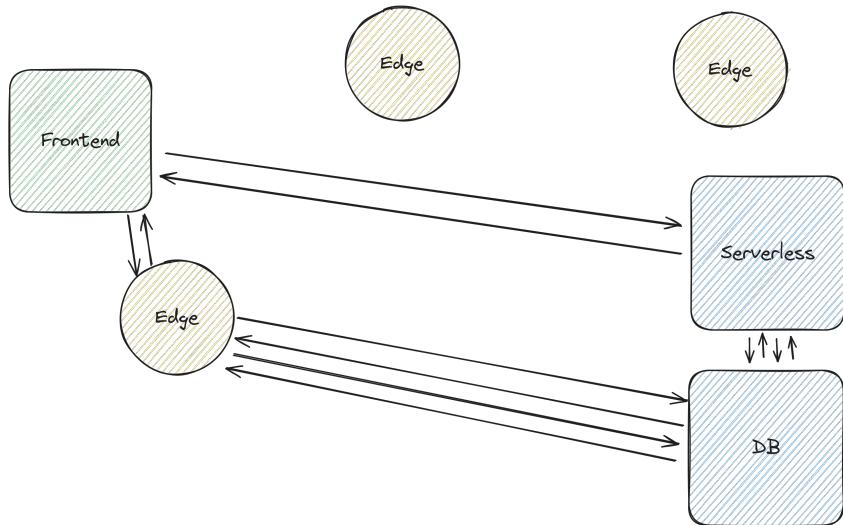
1.1 ARCHITEKTURA WEBOVÉ APLIKACE

1.1.1 Koncepty webu, jejich přednosti a nevýhody

K vytvoření komplexní webové aplikace, jakou je i Effio, je možné využít celou řadu konceptuálních řešení. Předem je proto nutné analyzovat požadavky a potřeby aplikace a tomu přizpůsobit volbu vhodné architektury i technologie. V následujících odstavcích zmiňují aspoň ty přístupy, o nichž jsem v souvislosti s tvorbou webu pro online testování uvažoval a analyzoval jsem jejich možné přínosy i nevýhody.

- **Tradiční web server** představuje nejběžnější způsob vytváření webových stránek. Jednotlivé stránky jsou generovány na serveru a poté odeslány klientovi. Poslaný kód může zahrnovat i JavaScript pro frontendovou funkci. Pro backendovou část je možné využít jazyk podle výběru. Pro veškerá přesměrování a další akce je nezbytné komunikovat se serverem. [1]
- **Single Page Application (SPA)**. Toto řešení v podstatě odstraňuje nutnost serveru a nechází veškerou zodpovědnost frontendovému frameworku, jako jsou například React, Svelte nebo Solid. Nemá ale k dispozici žádný způsob jak spouštět kód, který na klientovi nemůžeme použít, například provádění SQL dotazů. Další nevýhodou je, že stránka je generována až na klientovi pomocí JavaScriptu, což znamená, že vyhledávače nejsou schopny detekovat obsah stránky, což vede k mnohem horším výsledkům ve vyhledávání (SEO). [1]
- **Serverless**. Jedná se o architekturu, kde vývojář využívá server poskytovatele, o který se nemusí starat a je škálován podle potřeby. Tento koncept však má i své nevýhody, jako jsou omezená doba relace odpovědi, menší úložný prostor pro načítání knihoven a nemožnost spravovat vlastní server. [2]
- **Edge runtime**. Tato technologie je podobná serverless architektuře; v tomto případě serverový kód neběží v jedné lokalitě, nýbrž na jednotlivých CDN. Funkce se spouštějí ne skrze Node, ale přes Edge runtime. A to má své nevýhody: Edge není Node, a proto

nemůžeme používat Node moduly, jako je například „fs“. Další nevýhodou je omezená kapacita paměti, která je pro dostupnou instanci k dispozici. Výhodou je naopak velmi nízká cena spuštění takové funkce a bezkonkurenční rychlosť odpovědi. Tato výhoda je největší u serverových úkolů, jako jsou přesměrování, práce s cookies nebo geograficky založená data. V případě několikanásobných dotazů do databáze se ale cesta potřebná k získání dat prodlužuje a výhoda rychlosti postupně mizí. [2]



Obrázek 1.1: Rozdíl mezi serverless a edge.

- **Metaframework** je technologie, která kombinuje výhody „single page application“ a tradičního web serveru. Disponuje možností běhu kódu na serveru, přesměrováním na klientské části a dalšími přednostmi. Takových technologií existuje celá řada, například populární NextJS. Významným rysem této architektury je hostování v cloudu. Nejlepší variantou většinou bývá hostování přes providery, příkladem mohou být Vercel nebo Netlify. V takovém případě se tato technologie primárně aplikuje jako „serverless“. [3]

1.1.2 Zvolené řešení

Pro svůj projekt zvolil SvelteKit, což je výkonný metaframework postavený na Svelte, umožňující vytvářet robustní a vysoce výkonné webové aplikace.

Při volbě vhodného řešení jsem vycházel z předpokladu hostování Effia na cloudových službách, proto jsem si vybíral hlavně mezi technologiemi „serverless“ a „edge computing“. Výhodou providera, kterého jsem si vybral (Vercel), je, že kombinace obou technologií je velice snadná. Základní variantou je „serverless“ s jednoduchým přepnutím dané cesty na „edge“. Ve spojení s konceptem metaframeworku nabízí velmi flexibilní, rychlou a příjemnou cestu směrem k zamýšlenému cíli.

1.2 TYPESAFETY

Webové aplikace standardně využívají JavaScript, který ale přináší signifikantní nevýhodu v podobě nemožnosti „otypovat“ kód. To způsobuje obtížnou orientaci v kódu, vysoké množství produkčních chyb a také mnoho času stráveného pochopením dříve napsaného kódu.

Pro Effio jsem se tedy rozhodl využít moderní technologie a vytvořit téměř plně „typesafe“ (otypovanou) aplikaci. TypeScript v Effiu nahrazuje JavaScript a do tohoto jazyka přináší typovou strukturu. To však nestačí, protože API endpointy a databázové dotazy stále nemohou být otypované. Proto jsem se rozhodl využít také knihovny tRPC a Prisma.

1.3 DATOVÉ MODELY

1.3.1 Formát GIFT

Pro import a export testů v rámci platformy Moodle, je využíván formát GIFT, jenž obsahuje speciální syntaxi v rámci textového souboru pro záznam jednotlivých otázek. K zajištění konverze tohoto formátu do JSON, se kterým se dá dobře pracovat, jsem využil knihovnu gift-pegjs. Pokud ale šlo o konverzi opačnou, tedy z JSON do GIFT, zvolit jsem vlastní řešení a naprogramoval jsem malou knihovnu pro generaci daného souboru gift-format-generator.

1.3.2 Datový model testu v Effiu

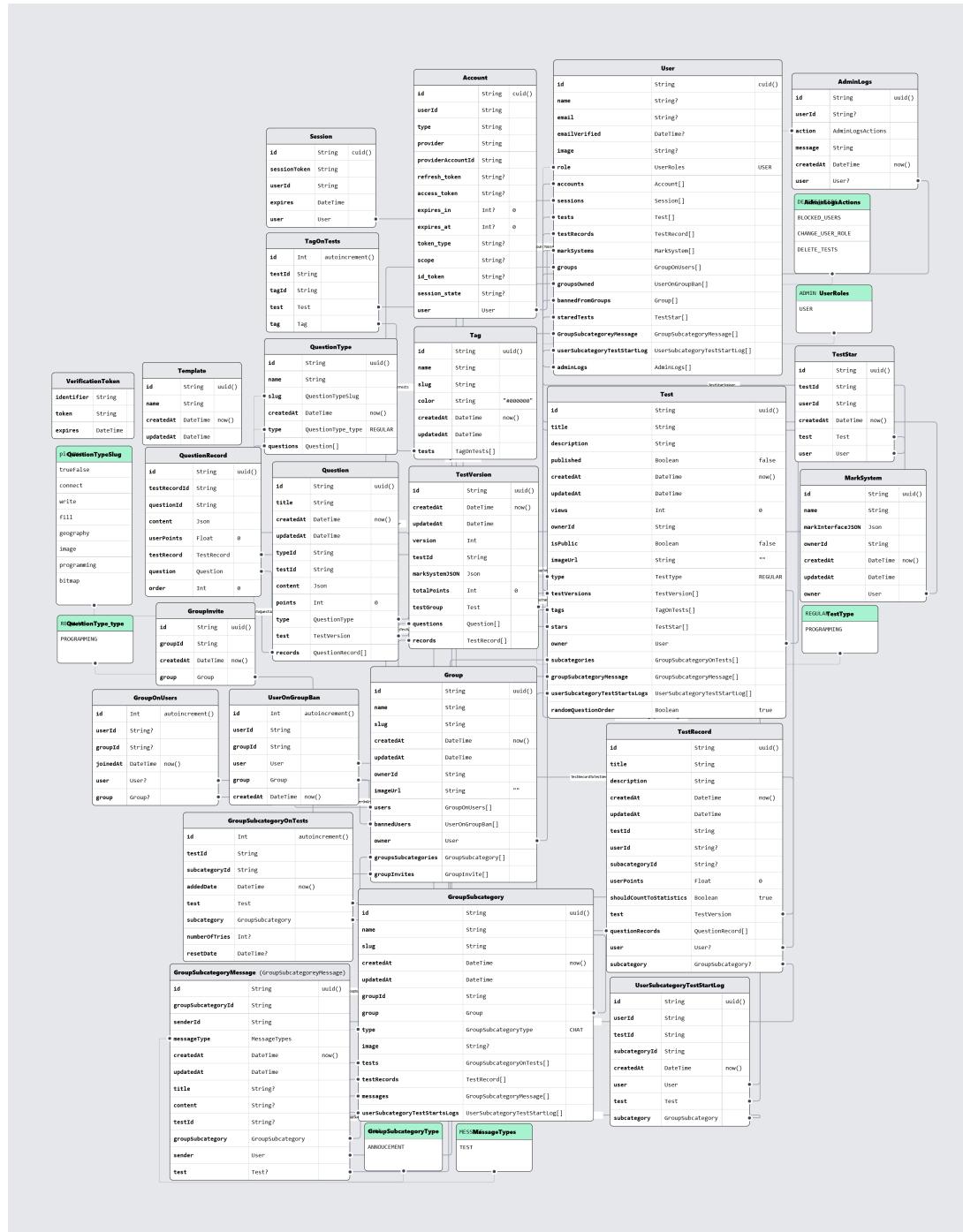
Pro práci s testem v rámci Effia jsem se rozhodl pro vytvoření vlastního formátu, který umožňuje uchovávat test ve „store“ (objektu) až do jeho uložení do databáze.

```
1 type ClientTest = {
2   title: string;
3   description: string;
4   questions: QuestionClient[]; // Otázky mají poté dalších spoustu atributů
5   errors: {
6     title?: string;
7     description?: string;
8     markSystem?: {
9       [Key in keyof MarkSystemJSON as Key extends "marks" ? never : Key]?: string;
10      & { marks: {[Key in keyof MarkSystemJSON["marks"]][number]?: string;}[] };
11    };
12    tagIds?: string[];
13  };
14}
```

Kód 1.1: Ukázka z +page.server.ts

1.3.3 Databázový model

Databázový model celého systému tvoří mnoho vzájemně propojených tabulek. Původní model, který zachycuje níže uvedené schéma, se dále rozrůstá a vznikají nové tabulky.



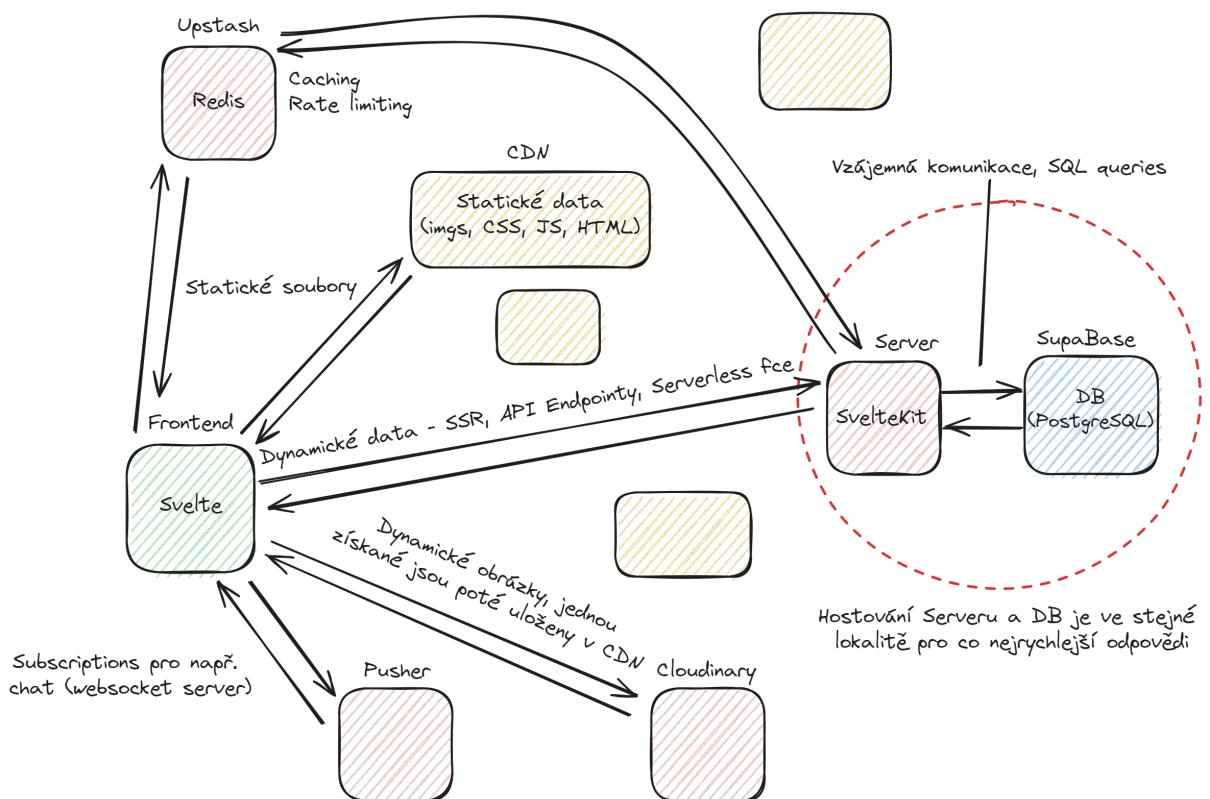
Obrázek 1.2: Databázový model Effia. Vytvořeno pomocí Prismaliser

2 BACKEND

2.1 ZALOŽENÍ A KONFIGURACE PROJEKTU

Prvním krokem bylo založení projektu a stažení potřebných knihoven, které jsem plánoval využít. Kombinace mnou vybraných technologií nebyla kompletně konvenční, a proto jsem se v některých případech musel obrátit na komunitou vytvořené adaptéry. Příkladem je například knihovna `trpc-sveltekit`, která propojuje SvelteKit a tRPC. Toto propojení je vytvořeno s ohledem na skutečnost, že tRPC je primárně navrženo buď jako samostatný server, nebo jako implementace do Next.js.

2.2 ARCHITEKTURA BACKENDU



Obrázek 2.1: Jednoduchý přehled backendové části Effia.

- CDN, neboli Content Delivery Network, je síť serverů, která je charakterizována zejména tím, že je rozmístěna po celém světě a ve velkém množství. Tato síť se stará o distribuci statického obsahu, čímž umožňuje klientům získávat data ze serverů, které jsou většinou mnohem blíže. Dále se CDN stará o cachování dat, což opět zkracuje dobu, po kterou uživatelé zobrazují obsah.
- Termín „Server“ neoznačuje server jako takový ale spíše místo kde se spouštějí instance serverových funkcí, což jsou funkce, které se vytváří podle potřeby na reálných serverech, které ale spravuje provider této služby, v méém případě Vercel, respektive Cloudflare.

Programátora nemusí tyto servery vůbec zajímat, instance se samy spouštějí, škálují a obecně jsou pro vývojáře velice příjemným řešením.

Tento server je v Effiu využíván hlavně pro první zobrazení stránky metodou SSR („server side rendering“) a také pro získávání dat, které nemohou být získány na klientu (např. SQL dotazy), pro jednotlivé stránky, formulářové akce nebo API endpointy.

- DB - PostgreSQL databáze hostovaná přes službu Supabase. Ta nabízí mnoho nástrojů pro vývoj webových aplikací včetně databáze, je open source, má „free tier“ pro cloud hosting a poskytuje také „self hosting“.
- Pusher, Cloudinary, Upstash - jedná se o clouдовé služby, sloužící účelům, které s touto architekturou nejsem schopný zařídit.
 - Pusher se stará o webové sockety, konkrétně o trvalé spojení, což není možné s „serverless“ spojením. V Effiu sloužil pro chat v kanálech skupin pro aktualizaci zpráv všech uživatelů, když někdo odešle zprávu.
 - Cloudinary slouží pro ukládání obrázků a jejich distribuci do CDN.
 - Upstash je služba, která nabízí Redis instanci, kterou využívám pro caching a také „rate limiting“ (to znamená, že kontroluji počet requestů na jednotlivé endpointy). Redis je „inmemory“ databáze a hlavní výhodou je rychlost zpracování dat.

2.3 DATABÁZE

Jedná se o PostgreSQL databázi, která je hostovaná přes clouдовou platformu Supabase. Komunikace s aplikací probíhá skrze ORM (Object-Relational Mapping), přesněji Prismu. Databázový model je umístěn jako příloha obrázku 1.2.

2.3.1 Stručný popis účelů jednotlivých tabulek

Autorizace a autentifikace

- User - tabulka s údaji o uživateli.
- Account - účet uživatele, typ provozera přes kterého je přihlášen a data k relaci.
- Verification Token - ověřovací identifikátor provozera.
- Session - relace, do jisté míry propojuje jednotlivé tabulky.

Otázky

- Question - otázka jako taková, obsahuje název, popis, její propojení s testy atd.
- QuestionType - typ otázky, rozhoduje o jejím chování na frontendu.
- QuestionRecord - záznam otázky, po vyplnění testu se vytvoří ke každé otázce jeden nový záznam s výsledky, je sdružován Test Recordem.

Testy

- Test - obsahuje název testu, jeho popis a sdružuje veškeré další podrobnosti testu, jako jsou Tagy, Stars, obsahuje množství TestVersion, což je vlastně určitá verze testu.
- TestVersion - verze testu, uchovává odpovědi, body a Mark System, verze se aktualizují při každé změně testu.
- TestRecord - záznam vyplněného testu, obsahuje také Question Records.

Ostatní tabulky týkající se otázek

- TestStar - ohodnocení testu hvězdičkou, každý uživatel mimo majitele může test takto ohodnotit, ekvivalent "like" na sociálních sítích
- MarkSystem - známkovací systém daného testu, uživatel si ho může sám upravit a při uložení testu je uchován právě v této tabulce.

- Tag/TagOnTest - uchovává štítky vybrané majitelem, jež se týkají tématu testu.

Skupiny

- Group - skupina pro uživatele, obsahuje základní vlastnosti skupiny jako jméno, slug apod.
- GroupSubcategory - jednotlivý kanál skupiny, každý tento kanál má i samostatné zprávy, chat apod.
- GroupSubcategoryMessage - Zpráva v kanálu, vztahuje se k ní také její odesilatel, název, obsah atd. Obsahuje také MessageType, což je druh zprávy, kterou uživatel poslal.

Ostatní

- Template - šablona testu

2.4 AUTENTIFIKACE

2.4.1 Auth.js

Auth.js je knihovna sloužící pro autentifikaci, poskytuje možnost JWT autentifikace i „session based“ autentifikace, kterou využívá Effio.

Knihovna podporuje také OAuth s mnohými providery, v tomto projektu je využit GitHub a Google s jednoduchou možností přidat další.

Výhodou knihovny je, že data si vývojář spravuje sám, neboli jsou ukládána do jeho vlastní databáze v podobě tabulek (které si také může sám upravit): Account, Session, User a Verification Token, které poskytují naprostou kontrolu nad ověřením uživatelů.

2.4.2 Proces přihlášení

Uživatel, který se rozhodne přihlásit pomocí Google nebo GitHub účtu na podstránce /login, je následně přesměrován na stránky těchto providerů. Zde potvrdí přístup k informacím o svém účtu a následně je vrácen zpět do Effia. V databázi jsou vytvořeny tabulky o tomto uživateli, včetně nové relace (Session), díky které je schopen přihlášení. Tento proces probíhá automaticky po návratu uživatele zpět od providera.

2.5 CLOUD HOSTING

Jak již bylo zmíněno v úvodu, měla by se tato aplikace obejít bez vlastního serveru. Nejde ale jenom o databázi ale také například o ukládání obrázků nebo hostování aplikace. To je prováděno přes „Cloud hosting“ Vercel, který vedle hostování stránky řeší i rozesílání statických dat do CDN a poskytuje serverless lambda funkce, Supabase pro hostování mé PostgreSQL databáze, Cloudinary, který slouží jako „bucket“ pro obrázky a také umožňuje jejich editaci přes URL parametry, Pusher, který slouží jako web socket server například pro chat, nebo Upstash, na kterém běží instance Redisu sloužícího pro caching a „rate limiting“.

2.6 VYUŽITÉ BACKENDOVÉ TECHNOLOGIE

2.6.1 SvelteKit

SvelteKit je metaframework postavený nad Svelte, podobně jako ostatní metaframeworky slouží k backendovým potřebám aplikace. Jeho hlavní výhody zahrnují:

- Rychlosť. Svelte vytváří velmi rychlé aplikace, a ve spojení s Vitem (bundler) dosahuje také výborných výsledků v rychlosti build procesu a „hot module replacementu“. Rychlosť vývoje je také podpořena minimálním množstvím „boilerplate“ kódu díky přístupu SvelteKitu.
- Flexibilita. SvelteKit umožňuje jednoduchou konfiguraci jednotlivých stránek nebo cest pro různé způsoby vykreslování, jako jsou SPA, SSR, SSG nebo MPA. Dále umožňuje variabilní kombinaci kódu běžícího na serveru a na klientovi, poskytuje rozsáhlou kontrolu nad jejich chováním a umožňuje snadné úpravy.
- Přehlednost. SvelteKit využívá „file-based routing“, což znamená, že cesty aplikace jsou generovány podle složek vytvořených vývojářem. Soubory jsou následně pojmenovány konzistentně, například `+page.svelte` pro stránky a `+layout.svelte` pro layouty. Dalším prvkem přispívajícím k přehlednosti je podobnost s JavaScriptem, neboli to, že framework se snaží udržovat syntaxi velice podobnou čistému JavaScriptu.

V mé aplikaci SvelteKit zastává naprosto zásadní roli, od routování, přes serverové operace (jako jsou load funkce a API endpointy), konfiguraci bundleru až po přesměrování. Ve zkratce se jedná o základní stavební blok, bez kterého by aplikace nebyla funkční.

Tento kód ukazuje získání dat z databáze při načtení stránky, ale před zobrazením uživateli (load funkce), a poté také actions, což jsou formulářové akce vytvářené pro specifickou cestu, které poté můžeme využívat; zde je vidět mazání uživatele ze skupiny.

```

1 export const load: ServerLoad = async (event) => {
2   // Mohou se vracet i Promisy, SvelteKit je sám resolvne, mění se ve SvelteKit 2.0
3   const users = prisma.user.findMany({ where: { name: event.params.name } })
4   return users
5 }
6 export const actions: Actions = {
7   deleteUsers: async (event) => {
8     const formData = await event.request.formData()
9     const users: string[] = []
10    formData.forEach((value) => { users.push(value.toString()) })
11    try {
12      await (await trpcServer(event)).groups.kickUsersFromGroup({
13        groupSlug: event.params.name as string,
14        userIds: users,
15      })
16      return { success: true }
17    }
18    catch (e) {
19      if (e instanceof TRPCError) {
20        return fail(getHTTPStatusCodeFromError(e), { message: e.message })
21      }
22      else { return fail(500, { message: "Something went wrong." }) }
23    }
24  }
25}

```

Kód 2.1: Ukázka z +page.server.ts

2.6.2 tRPC

V minulé kapitole jsem se zmínil o problémech s otypováním API endpointů. tRPC (Type-script Remote Procedure Call) tento problém řeší tím, že vytváří dynamické typy pro jednotlivé endpointy, podle toho, jak si je sami nařeší. Ty se potom dají volat pomocí funkcí bez přímého použití fetcha nebo třeba axiosu (tyto funkce ve skutečnosti „fetch“ requesty vykonávají ale programátor se o ně nemusí starat přímo), tyto funkce jsou dokonale otypované a v kódu tím pádem fungují jako jakákoli jiná funkce vytvořená programátorem. Další výhodou je také to, že jednotlivé „procedury“, což je vlastně API endpoint, mají k dispozici metody pro kontrolu vstupu nebo třeba middleware, který například může zjišťovat stav přihlášení uživatele. Jako zbytek knihovny jsou i tyto metody perfektně otypované.

V Effiu tRPC využívám hlavně jako možnost komunikace klienta s databází. Proces spočívá v tom, že tRPC v build time vytvoří API endpointy, které poté klient přes zmíněné funkce volá. V těchto endpointech se většinou nachází operace s databází - ty na klientu provádět nemohu.

Výhodou je, že tyto funkce mohu využívat i na serveru s trochu odlišnou implementací.

Ukázka kódu zobrazuje vytvoření procedury, neboli endpointu, který se poté dá volat. V druhé části je zobrazena právě zmíněná funkce, díky které získáme potřebné data.

```
1 getTestId: procedure.input(z.object({
2   id: z.string(),
3   includeGroupSubcategories: z.boolean().optional()
4 })) .query(async ({ ctx, input }) => {
5   const test = await ctx.prisma.test.findUnique({
6     where: { id: input.id },
7     include: {
8       subcategories: input.includeGroupSubcategories || false,
9       owner: true,
10      tags: { include: { tag: true } },
11      testVersions: {
12        include: {
13          questions: { include: { type: true } }
14        },
15        orderBy: { version: "desc" },
16        take: 1
17      }
18    },
19  })
20
21  if (!test) return null
22  return test
23}),
```

Kód 2.2: Endpoint generovaný pomocí tRPC

```
1 const imageUrlToDeleteTest = await trpc($page).getTestId.query({
2   id: props.data.id,
3 })
```

Kód 2.3: Volání funkce pomocí tRPC klienta s metodou getTestId

2.6.3 Prisma

Prisma slouží jako ORM (Object–relational mapping), což umožňuje získávat data z databáze pomocí JavaScriptu bez přímého použití jazyka SQL. Prisma se skládá z klientské části, která komunikuje se serverovou částí pomocí protokolu založeného na JSON. Na serverové straně se následně vykonává SQL dotaz a odpověď je odeslána zpět klientovi. Prisma také řeší problémy s otypováním dotazů. Je třeba definovat model databáze v souboru schema.prisma, který popisuje její strukturu. Tento model může být používán jak pro generování dotazů, tak pro typování dat v aplikaci.

Tato ukázka kódu zobrazuje SQL operaci, kterou Prisma vytvoří podle této objektové struktury, kterou jsem sestavil. Cílem je získat unikátní test pomocí jeho ID společně s přidruženými tabulkami. Druhá ukázka poté vytváření modelu.

```

1 const test = await ctx.prisma.test.findUnique({
2   where: { id: input.id },
3   include: {
4     subcategories: input.includeGroupSubcategories || false,
5     owner: true,
6     tags: { include: { tag: true } },
7     testVersions: {
8       include: { questions: { include: { type: true } } },
9       orderBy: { version: "desc" },
10      take: 1
11    }
12  },
13})

```

Kód 2.4: Získání testu podle id a přidání dat ze spojených tabulek

```

1 model Question {
2   id      String          @id @default(uuid())
3   title   String
4   createdAt DateTime        @default(now())
5   updatedAt DateTime        @updatedAt
6   typeId  String
7   testId  String
8   content  Json
9   points   Int            @default(0)
10  type     QuestionType   @relation(fields: [typeId], references: [id], onDelete:
11    Cascade)
12  test      TestVersion    @relation(fields: [testId], references: [versionId],
13    onDelete: Cascade)
14  records   QuestionRecord[]
15
16}

```

Kód 2.5: Schéma modelu verze testu

2.6.4 Zod

Zod je validační knihovna, podobná například Yup. To znamená, že jejím úkolem je kontrolovat vložené vstupy. Knihovna vrací úspěšnost a také chyby, na které během kontroly narazí.

Její výhodou je možnost využít validačních schémat Zodu jako typů. Samotná validace funguje i jako „type guard“, což znamená, že kontroluje a nastavuje typy u vložených proměnných.

Zod je v Effiu využíván jak na backendu, tak na frontendu. V backendové části je umíštěn pouze proto, že validační operace častěji probíhají na serveru. Díky Zodu jsem schopen přehledným a spolehlivým způsobem kontrolovat data testů, formulářů apod. Zásluhou hezký formátovaných vrácených chyb je poté mohu bezproblémově zobrazovat uživatelům.

Tento kód kontroluje zdali vložený input odpovídá struktuře „answerSchema“, popřípadě nastaví error do proměnné, která se poté zobrazí klientovi.

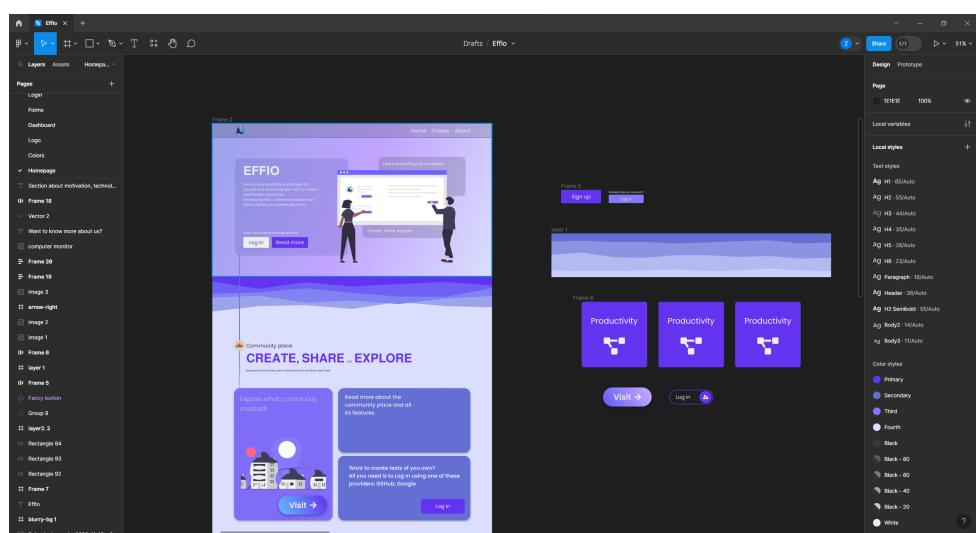
```
1 const answerSchema = z.string().min(ANSWER_MIN, `Answer has to be at least ${ANSWER_MIN} character long.`).max(ANSWER_MAX, `Answer can be max ${ANSWER_MAX} characters long.`)
2 const result = answerSchema.safeParse(content.answers[item].answer)
3 if (result.success === false) {
4   content.answers[item].error = result.error.errors[0].message
5 }
```

Kód 2.6: Validace vstupu pomocí validační knihovny Zod

3 FRONTEND

3.1 DESIGN

Důležitým záměrem projektu Effio bylo vytvořit na pohled přívětivou aplikaci, proto se návrh designu stál klíčovou částí pro stylově propracovanější prvky stránky. K návrhu designu, stejně jako k vytváření a úpravě potřebných obrázků, jsem využil aplikaci Figma.

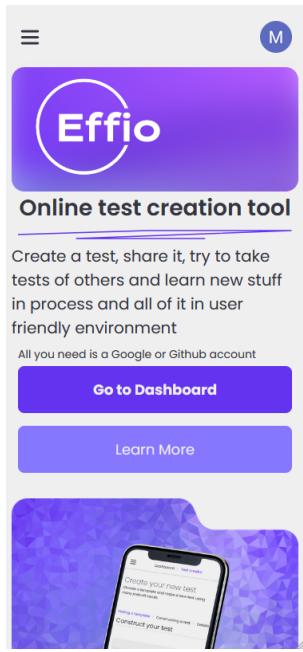


Obrázek 3.1: Prvotní návrh domovské stránky.

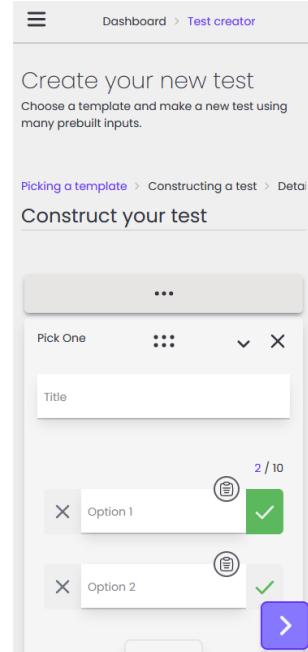
3.2 RESPONZIVITA

Celá aplikace je uzpůsobená jak pro počítače, tak pro mobilní zařízení. Zajištění responzivity není vůbec lehká práce, ale mně pomohl Tailwind. V něm se CSS media query dělají snadněji společně s moderními CSS kontejnery, které umožňují responzivní breakpointy odvozovat nejen od velikosti stránky, ale také od rozměrů rodičovských elementů.

Problematika však nespočívá pouze v zobrazení prvků, ale také v jejich funkcionalitě, která musí fungovat s myší i s dotekem. Příkladem tohoto problému je jeden z typů otázek, a to „Connect“, kde se musí jednotlivé spoje přesouvat jak pohybem myši, tak při dotykovém vstupu.



Obrázek 3.2: Domovská stránka na mobilním zařízení



Obrázek 3.3: Generátor testů na mobilním zařízení

3.3 PŘÍSTUPNOST

3.3.1 Světlý a tmavý režim

S ohledem na uživatele, kteří preferují tmavý režim, jsem se také rozhodl vytvořit tmavý režim. Ten je možné vidět na obrázku 4.3. Oba tyto režimy vyžadovaly vlastní paletu barev a byly mnohokrát přepracovány, aby jednotlivé barvy co nejlépe ladily k sobě.

3.3.2 Dostupnost

Stránky jsem se také snažil tvořit s ohledem na lidi využívající „Tab“ k navigaci stejně jako pro lidi s omezeními, aktuálně ale podpora pro čtečky není kompletní a je ještě spoustu prostoru pro zlepšení.

3.4 VYUŽITÉ FRONTENDOVÉ TECHNOLOGIE

3.4.1 Svelte

Svelte je open-source JavaScriptový framework vyvíjený týmem Richa Harrise od roku 2016. Jeho hlavní výhodou je rychlosť a intuitivnosť, protože jazyk, ktorý používá, se snaží vypadat ako JavaScript, zatiaľco rozširuje jeho možnosti. Tím se odlišuje od iných frameworkov, ako sú React, Angular, Solid alebo Qwik. Kód sa zapisuje do súborov s príponou .svelte, ktoré sú následne kompilované do veľmi efektívneho JavaScriptu. Díky tomu Svelte získáva stále rostúcu popularitu mezi webovými vývojáři.

Kód v Svelte se skládá ze tří hlavních částí.

1. Script. Jedná se o část, kde se vypisují funkce, promenné a řeší se reaktivní deklarace. Celá tato část je obklopená <script> tagy jako v HTML dokumentu.

```
1  export let inputValue: HTMLTextAreaElement['value'] = '';
2
3  let setError = getContext('setError');
4
5  let inputRef: HTMLTextAreaElement;
6
7  const dispatch = createEventDispatcher();
8
9  function validateInput() {
10    const result = validationSchema?.safeParse(inputValue);
11    if (!result?.success) {
12      dispatch('error', result?.error.errors[0].message);
13      if (typeof setError === 'function')
14        setError(result?.error.errors[0].message);
15    } else {
16      dispatch('error', null);
17      if (typeof setError === 'function') setError('');
18    }
19  }
20
21  function dispatchInputChange() {
22    dispatch('inputChange', inputRef.value);
23  }
```

Kód 3.1: Ukázka Svelte kódu ve script tagu

2. Style. Tato část slouží ako CSS pro daný dokument. Výhodou je lokální rozsah aplikovaných stylů (podobný principu CSS modules), ale umožňuje i použití globálních stylů

pomocí `:global`. Mně osobně vyhovuje, že se styly nacházejí v jednom souboru. I když jsem tento způsob pro stylování Effia převážně nevyužíval, osobně mi připadá velmi dobře provedený. Následující kód je formátován jako v HTML a je obsažen v tagu `<style>`.

```

1  <style>
2      .grid_cover {
3          display: grid;
4          grid-template-rows: auto 1fr;
5      }
6      :global(.dark) .fading {
7          background-color: var(--dark-light_grey);
8      }
9  </style>

```

Kód 3.2: Ukázka Svelte CSS kódu

3. Obsahová část. Vše, co se nenachází v jednom z těchto tagů, představuje HTML reprezentaci dané stránky. Nejedná se však o čisté HTML, nýbrž obohacenou verzi. Podobně jako v jiných frameworcích je možné přidávat „event listeners“ k jednotlivým elementům, podmínkově je zobrazovat, pracovat s asynchronním kódem nebo dokonce „svazovat“ element nebo hodnotu elementu s proměnnou ve skriptové části.

```

1  {#await data}
2      <!-- Zde se nachází další kód -->
3  {:then awaitedData}
4      <div class="@container h-full">
5          <div
6              bind:this={scrollerDiv}
7              class="flex w-full h-full py-1 scroller flex-nowrap"
8              style="--translate-x: 0%;"
9          >
10         {#each awaitedData as item}
11             <div
12                 class="min-w-[calc(100%/var(--items-count))] relative aspect-[4/5]"
13             >
14                 <CardAlternative
15                     navigationLink={'/tests/' + item.id}
16                     type={item.type}
17                     data={{...item}}
18                 />
19             </div>
20         {/each}
21     </div>
22  {/await}

```

Kód 3.3: Ukázka Svelte kódu

3.4.2 TypeScript

TypeScript lze považovat za nadstavbu JavaScriptu s jednou výraznou výhodou - typy. Díky nim je mnohem snazší odhalovat chyby, vracet se k dříve napsanému kódu a celkově zlepšit "developer experience" při vytváření aplikace. Sám o sobě dokáže pomoci s otypováním jednotlivých částí kódu, ale nemá schopnost pracovat s API endpointy nebo databázovými dotazy, které je nutné otypovat ručně. Z tohoto důvodu jsou v tomto projektu využívány další knihovny, jako jsou Prisma, tRPC a Zod.

V Effiu jsem se pro TypeScript rozhodl z důvodů, které vysvětluji už v podkapitole 1.2. Šlo hlavně o možnost efektivně psát kód, který bude obsahovat méně produkčních chyb, způsob jak se lépe vracet k dřívějšímu kódu, také rychlosť a jistota psaní, díky automatickému doplňování IDE, v mém případě Visual Studio Code.

Ukázka kódu zobrazuje vytváření různých typů, které poté v aplikaci využívám.

```
1 // Carousel.svelte
2 export type IdCardAlternativeProps = CardAlternativeProps & {
3   id: string;
4   type: TestType;
5 };
6
7 export type CarouselItemInput =
8 | IdCardAlternativeProps[]
9 | Promise<IdCardAlternativeProps[]>;
10
11 // customUtilities.d.ts
12
13 // Ručně vytvořený generic, který mi umožňuje zkombinovat funkcionality Partial a Pick
14 // genericu.
15 type PartialPick<T extends Record<unknown, unknown>, K extends keyof T> = {
16   [Key in Exclude<keyof T, K>]: T[Key];
17 } & {
18   [Key in K]?: T[Key];
19 }
```

Kód 3.4: Ukázka TypeScriptového typu

3.4.3 Tailwind CSS

Tailwind CSS je utility knihovna pro práci s CSS, která se odlišuje od frameworků jako Bootstrap nebo Material UI. Na rozdíl od nich neposkytuje celé předpřipravené komponenty, ale nabízí sadu připravených CSS tříd, které se aplikují na HTML elementy. Hlavní výhody Tailwind CSS spočívají v absolutní kontrole nad chováním aplikovaných stylů, přehlednosti a

rychlosti, s jakou lze vytvářet styly.

Pro Effio jsem se rozhodl využít Tailwind především díky jeho flexibilitě a rychlosti použití. Pro některé komponenty jsem využil také Daisy UI, což je komponentová knihovna integrovaná s Tailwind, pracující pouze s CSS. Tato knihovna slouží jako plugin pro Tailwind.

Následující kód ukazuje, jak lze využít Tailwind tříd pro stylizaci určitého HTML elementu. I když jsem zde záměrně vybral rozsáhlejší příklad, většinou si vystačím s jedním až dvěma řádky tříd.

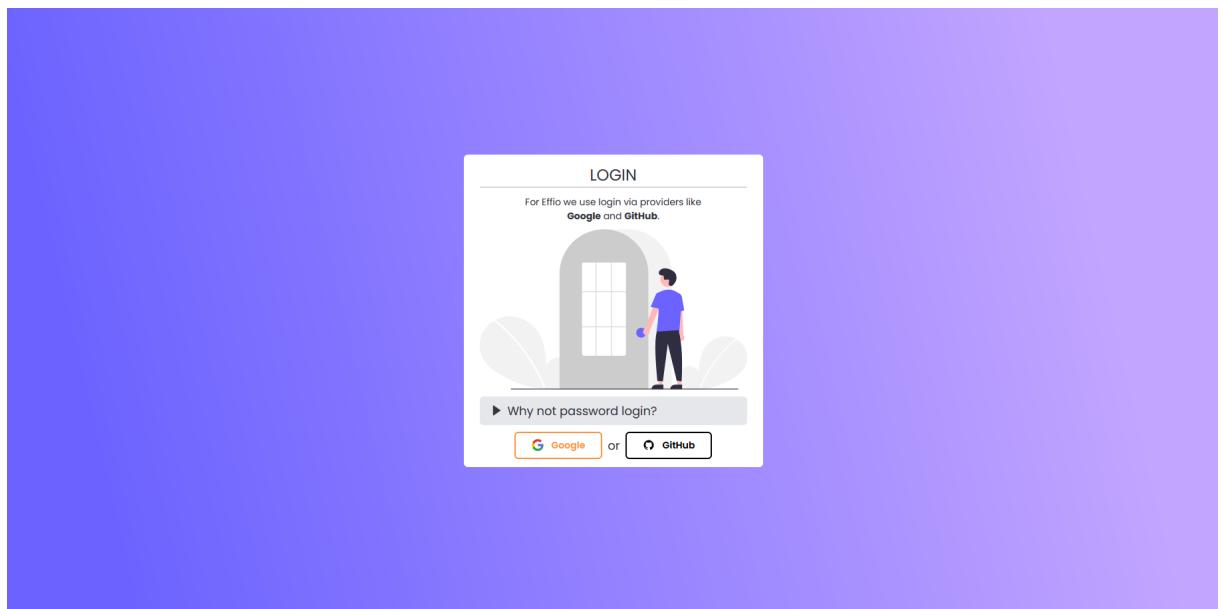
```
1 <button
2   type="button"
3   on:click={starTest}
4   disabled={canStarTest === false || isSubmittingStar === true}
5   class={`absolute flex items-center z-[2] gap-1 px-2 py-1 rounded-lg right-1 top-1 bg
6     -light_white dark:bg-dark_grey shadow-md duration-100 ${'
7       isStarred ? 'bg-yellow-100 dark:bg-yellow-700' : ''
8     } hover:bg-light_secondary dark:hover:bg-dark_secondary disabled:bg-
9     light_grey_dark dark:disabled:bg-slate-600
10    text-light_text_black dark:text-dark_text_white hover:text-light_whiter disabled:
11      hover:text-light_text_black
12      dark:disabled:hover:text-dark_text_white`}
13    >
14  </button>
```

Kód 3.5: Ukázka Tailwind kódu

4 FUNKCE APLIKACE

4.1 PŘIHLÁŠENÍ

Po kliknutí na tlačítko „Login“ se dostanete na přihlašovací obrazovku, kde si můžete vybrat mezi přihlášením přes Google a GitHub účet. Po úspěšném přihlášení se uživatel dostane do dashboardu, který je spolu s možností vytváření testů, prohlížením historie a správou skupin dostupný pouze pro přihlášeného uživatele. Pokus o načtení stránky, pokud je uživatel nepřihlášený, vyústí v přesměrování na přihlašovací stránku.



Obrázek 4.1: Domovská stránka Effia pro přihlášeného uživatele.

4.1.1 Testy a jejich vlastnosti

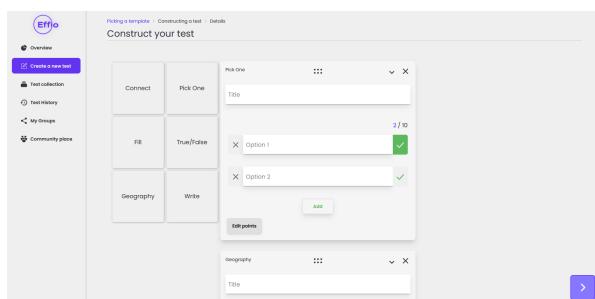
Jednou z esenciálních funkcionalit Effia je možnost vytvořit test. K tomu existuje mnoho různých přístupů. Nejprve jsem se zamýšlel nad danou problematikou, a až poté jsem napsal funkční nástroj pro jejich vytváření. Nicméně, i přes tuto předběžnou úvahu, bylo nakonec nutné téměř celou funkcionalitu při vytváření testu přepsat.

Tvorba testu

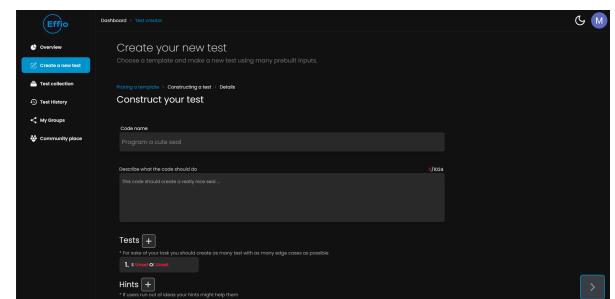
Jako první si uživatel zvolí mezi kvízovým a programovacím testem, u obou si poté vybere šablonu.

- Kvízový test: Po výběru šablony, která umožňuje i import z formátu GIFT, se uživatel dostane do tvorby testu. Zde má možnost vybírat z 8 typů otázek: *Pick One*, *True/False*, *Connect*, *Write*, *Fill*, *Geography*, *Image* a *Bitmap*. Uživatel může libovolně měnit pořadí otázek, přidávat komentáře k odpovědím a upravovat počet získaných bodů.
- Programovací test: Po výběru šablony se uživatel dostane do tvorby programovacího testu. Zde pojmenuje problém, popíše, co má uživatel řešit, nadefinuje kontrolní vstupy a očekávané výstupy. Dále může přidat návodů.

Po dokončení těchto úprav se uživatel přesune do konečných úprav testu, kde zadá jméno, popisek a přidá obrázek testu. Má možnost volitelně zařadit test do skupin, přidat tagy, rozhodnout se pro použití vlastního známkovacího systému (který si může upravit), zvolit náhodné třídění otázek a nakonec se rozhodnout, zda test uložit jako návrh nebo ho publikovat.



Obrázek 4.2: Kvízový test



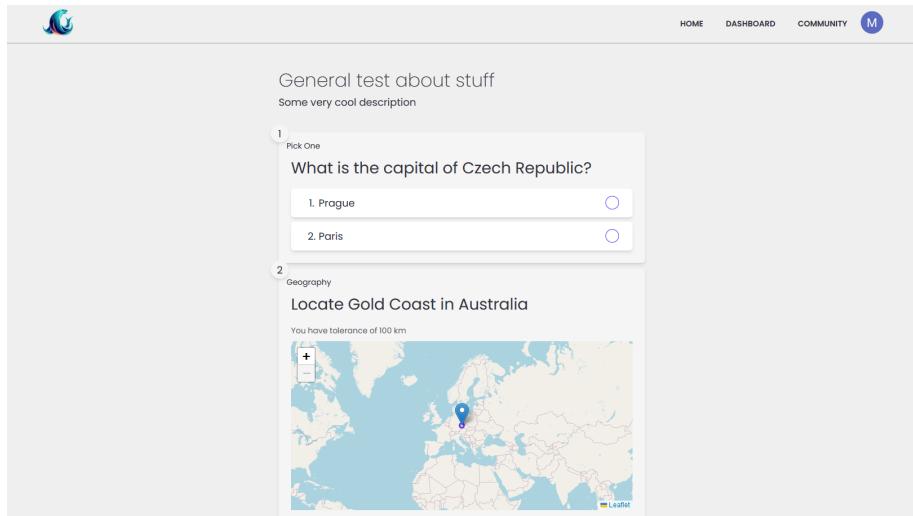
Obrázek 4.3: Programovací test

4.1.2 Vyplňování testu

Vyplňování kvízu

Vytvořený test si poté může kdokoliv s přístupem k němu vyplnit (testy jsou základně dostupné pro všechny, po úpravě mohou být zveřejněny pouze pro členy skupin).

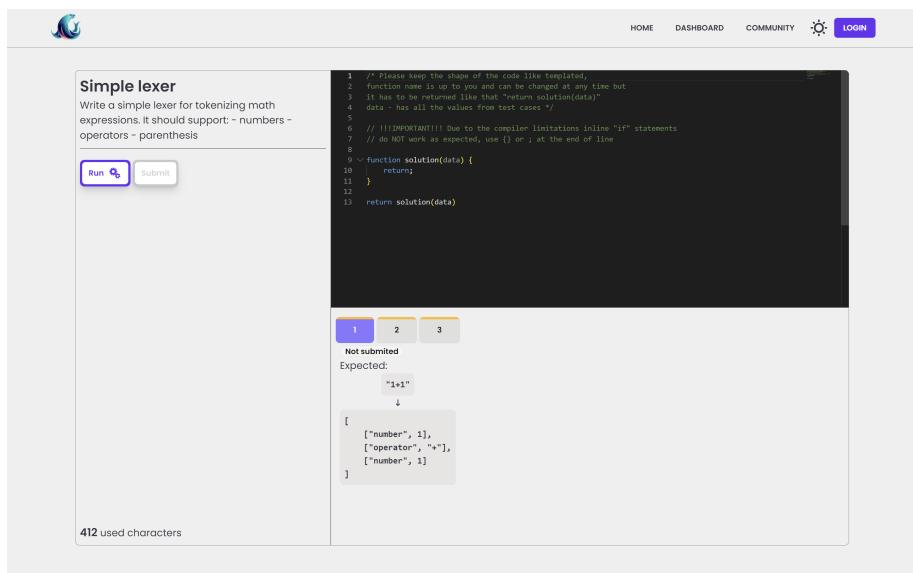
Otázky jsou náhodně seřazeny, a uživatel musí odpovědět na všechny z nich. Odpovědi některých typů otázek jsou náhodně zamíchány. Po vyplnění všech otázek uživatel odevzdá test, který se následně zkонтroluje. Uživateli jsou zobrazeny správné odpovědi, počet získaných bodů, udělená známka a pokud je uživatel přihlášený, záznam o vyplnění se uloží do databáze. Uživatel si poté může zobrazit záznam v sekci *Test history*.



Obrázek 4.4: Obrázek kvízu.

Plnění programovacího testu

Uživatel obdrží popis toho, co by měl kód schopný vykonat, sadu testů, které mají ověřit funkčnost kódu, a případné návodky. V případě programovacích testů je k dispozici vlastní editor, do kterého uživatel píše kód. Pro kontrolu správnosti kódu může použít tlačítko „Run“ a v případě, že testy procházejí, může odevzdat své řešení. Jednotlivý běh kódu probíhá na klientovi v rámci „web workeru“, které přesouvají zátěž na jiné vlákno procesoru. Hlavní výhodou ale je, že pokud dojde k nějaké situaci, kterou nelze snadno vyřešit (třeba while true), tak je možné tento worker zrušit a spustit nový.

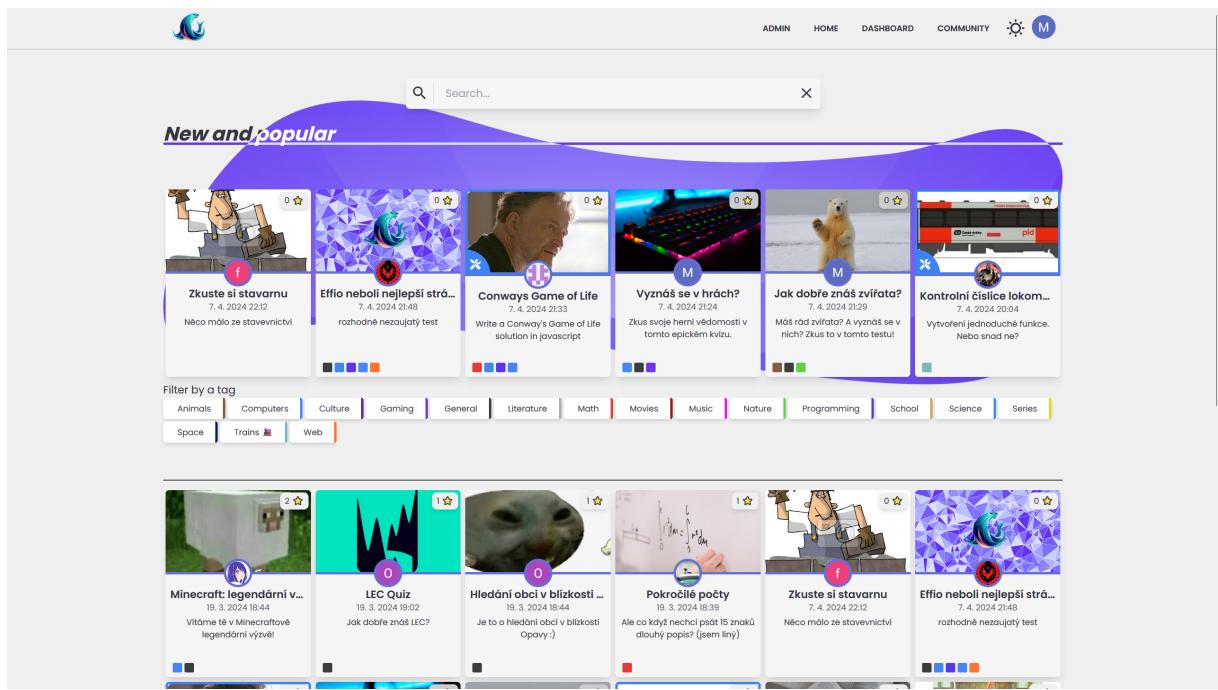


Obrázek 4.5: Obrázek programovací úlohy.

4.2 ZOBRAZOVÁNÍ TESTŮ

4.2.1 Komunitní místo

Na této stránce může uživatel objevit nové a populární testy, včetně všech dostupných testů. Testy se zobrazují postupně pomocí techniky „infinite scrolling“, kde využívám JavaScript API Intersection Observer k detekci, kdy uživatel dosáhl posledního prvku a následně si vyžádá další testy. Implementoval jsem vyhledávací pole, které umožňuje filtrovat zobrazené testy, a další možností je filtrace pomocí tagů. Vizuálně jsou testy rozlišeny mezi kvízy a programovacími úlohami. Uživatel může hodnotit testy hvězdičkou s využitím principu „optimistic update“. To znamená, že po přidání hvězdičky ji uživatel ihned vidí, přestože se zároveň ověřuje jeho oprávnění a vytváří se záznam hvězdičky v databázi. V případě neúspěchu se automaticky hvězdička odebere.



Obrázek 4.6: Komunitní místo.

4.2.2 Kolekce testů

Zde si uživatel může zobrazit jím vytvořené testy, aplikovaná je stejná funkcionality vyhledávacího pole a „infinite scrolling“ . Každý test má ale také další možnosti, a to úpravu, export a smazání.

- Úprava - uživatel se přesune na stránku úprav, tam může celý test přepracovat.

- Export - vytvoří z testu textový soubor ve formátu GIFT se všemi otázkami daného testu, které jsou podporovány Moodle
- Delete - smazání testu z databáze

4.2.3 Testová historie

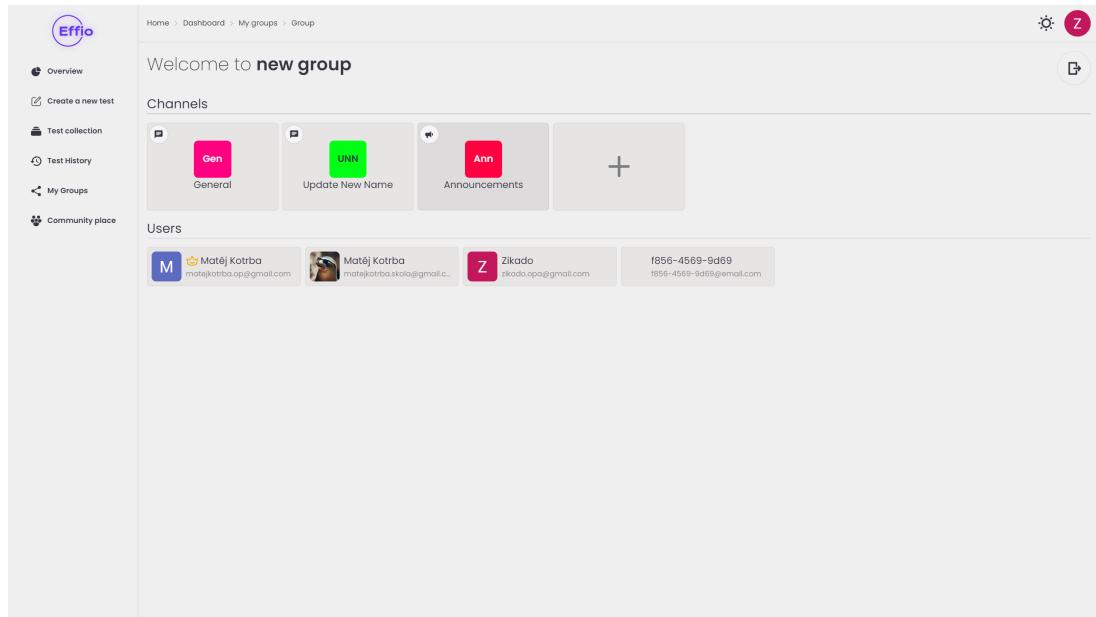
Toto místo slouží pro zobrazení dříve vyplněných testů, na výpis testů je zde pro změnu využita metoda stránkování ("pagination"), která rozděluje jejich zobrazení na jednotlivé části, mezi nimiž se přepíná. Po kliknutí na test se zobrazí záznam testu přesně v tom stavu, jak ho uživatel dříve vyplnil.

Name	Taken at	Description
Cool test	1.4.2024	Really cool one
Cool test	1.4.2024	Really cool one
Cool test	1.4.2024	Really cool one
Cool test	1.4.2024	Really cool one
Simple programming	19.3.2024	Can you complete this insane test
LEC Quiz	19.3.2024	Jak dobře znáš LEC?
Všeobecná fakta	19.3.2024	Úžasný text o spoustě zajímavých věcech
LEC Quiz	19.3.2024	Jak dobře znáš LEC?

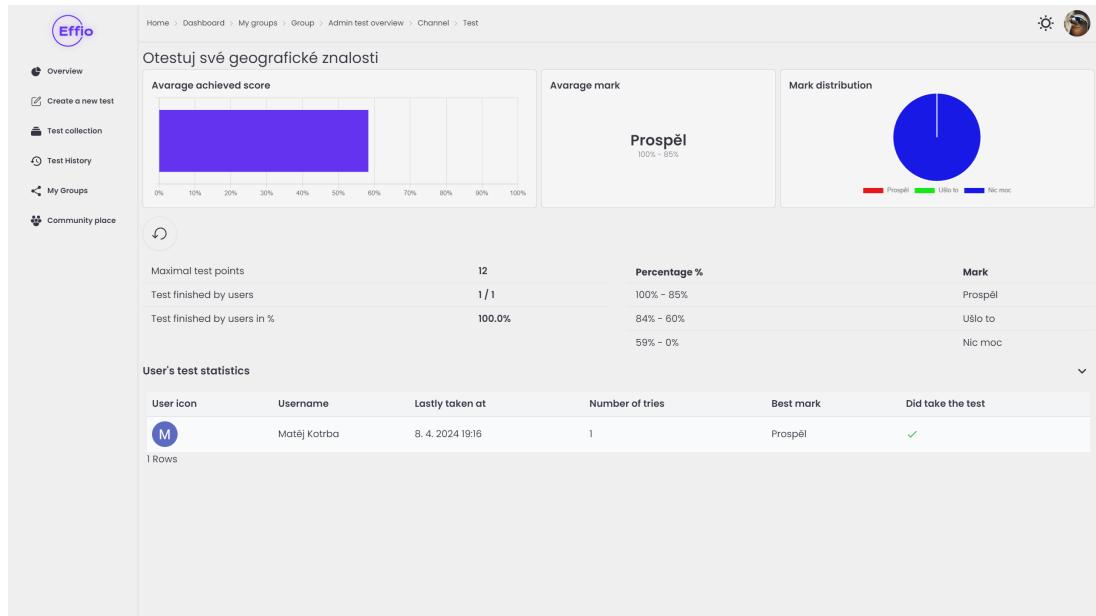
Obrázek 4.7: Testová historie

4.3 SKUPINY

Každý přihlášený uživatel si může vytvořit vlastní skupinu, k níž se pomocí generovaného kódu mohou připojit ostatní uživatelé; tyto kódy se poté pomocí cronu mažou. Skupina obsahuje jednotlivé kanály, které fungují jednak jako chat (nebo jako oznámení a psát do něj může pouze majitel), jednak jako místo pro sdílení testů vlastníka. U těchto testů poté může vlastník kontrolovat vyplnění jednotlivými členy skupiny, grafy výsledků a také si zobrazit jednotlivá řešení uživatelů (na testy přidané do skupin lze aplikovat limit počtu vyplnění pro uživatele). Majitel má také možnost zakázat přístup jednotlivým členům.



Obrázek 4.8: Ukázka skupin



Obrázek 4.9: Ukázka přehledu testu vlastníka skupiny

4.4 ADMINISTRÁTORSKÁ ČÁST

Pro administrátory aplikace je dostupná také stránka se správou testů a uživatelů; zde si je mohou zobrazit v tabulce, odstraňovat je, měnit uživatelské role apod. Další funkcí je také zobrazování logů akcí, které byly administrátory provedeny.

5 ZÁVĚR

Cílem projektu bylo vytvořit webovou aplikaci pro vytváření a vyplňování testů. Aplikace je postavená na frameworku SvelteKit, psaný v jazyce TypeScript. Přihlašování stojí na knihovně Auth.js, PostgreSQL databáze je hostovaná službou Supabase, jako ORM je použita Prisma. Frontend řeší framework Svelte s CSS utility knihovnou Tailwind, pro validaci využívá Zod.

Základem aplikace je generátor testů, které registrovaný uživatel vytváří z předpřipravených typů otázek. Nepřihlášený uživatel může testy pouze vyplňovat, přihlásit se uživatel může pomocí Google nebo GitHub účtu. Uživatelé se mohou sdružovat do skupin, mohou využívat chat, sdílet testy společně s výsledky a grafy, apod. Možnosti využití aplikace sahají od nenáročného občasného návštěvníka, který si chce zkoušit nějaký kvíz, až po implementaci v rámci třídy, kde učitel zadá kvíz, který následně žáci musejí vyplnit.

Využité technologie činí aplikaci lehce škálovatelnou a velice výkonnou. Díky zvoleným technologiím si dovolují aplikaci prohlásit za produkčně způsobilou, jednotlivé služby lze snadno škálovat, implementace je z velké části stabilní. Responzivní design zajišťuje možnost použití aplikace na různých zařízeních.

Aplikace je zálohovaná na GitHubu na adrese <https://github.com/matej-kotrba/effio>. Je také volně přístupná na adrese <https://effio.vercel.app/>

POUŽITÉ ZDROJE INFORMACÍ

- [1] Microsoft [online]. Choose Between Traditional Web Apps and Single Page Apps (SPAs). Microsoft. 2023 [cit. 2024-04-10]. Dostupné z: <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps>.
- [2] OpenReplay [online]. Serverless vs. Edge Functions. Goodluck Woha. 2023 [cit. 2024-04-10]. Dostupné z: <https://blog.openreplay.com/serverless-vs-edge-functions/>.
- [3] Prismic [online]. Understanding the JavaScript Meta-framework Ecosystem. Ben Holmes. 2022 [cit. 2024-04-10]. Dostupné z: <https://prismic.io/blog/javascript-meta-frameworks-ecosystem>.
- [4] Svelte [online]. 2023 [cit. 2023-12-27]. Dostupné z: [https://svelte.dev/](https://svelte.dev)
- [5] SvelteKit [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://kit.svelte.dev/>
- [6] tRPC [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://trpc.io/>
- [7] Prisma [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://www.prisma.io/>
- [8] Zod [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://zod.dev/>
- [9] Auth.js [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://authjs.dev/>
- [10] Tailwind CSS [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://tailwindcss.com/>
- [11] tRPC-SvelteKit [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://icflorescu.github.io/trpc-sveltekit/>
- [12] ChatGPT [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://chat.openai.com/>
- [13] Stackoverflow [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://stackoverflow.com/>
- [14] Joy of Code. Youtube kanál. <Https://www.youtube.com/> [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://www.youtube.com/@JoyofCodeDev>

- [15] Huntabyte. Youtube kanál. [Https://www.youtube.com/](https://www.youtube.com/) [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://www.youtube.com/@Huntabyte>
- [16] BROWNE, Theo. Youtube kanál. [Https://www.youtube.com/](https://www.youtube.com/) [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://www.youtube.com/@t3dotgg>
- [17] POWELL, Kevin. Youtube kanál. [Https://www.youtube.com/](https://www.youtube.com/) [online]. 2023 [cit. 2023-12-27]. Dostupné z: <https://www.youtube.com/@KevinPowell>

Seznam obrázků

1.1	Rozdíl mezi serverless a edge.	8
1.2	Databázový model Effia. Vytvořeno pomocí Prismaliser	10
2.1	Jednoduchý přehled backendové části Effia.	11
3.1	Prvotní návrh domovské stránky.	20
3.2	Domovská stránka na mobilním zařízení	21
3.3	Generátor testů na mobilním zařízení	21
4.1	Domovská stránka Effia pro přihlášeného uživatele.	26
4.2	Kvízový test	27
4.3	Programovací test	27
4.4	Obrázek kvízu.	28
4.5	Obrázek programovací úlohy.	28
4.6	Komunitní místo.	29
4.7	Testová historie	30
4.8	Ukázka skupin	31
4.9	Ukázka přehledu testu vlastníka skupiny	31