

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 17: Informatika

Vývoj sociální sítě mesh.sk

**Tomáš Kebrle
Ústecký kraj**

Louny 2023

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 17: Informatika

Vývoj sociální sítě mesh.sk

Development of a social site mesh.sk

Autor: Tomáš Kebrle
Škola: Gymnázium Václava Hlavatého, Poděbradova 661,
440 01 Louny
Kraj: Ústecký kraj
Konzultant: Zdeněk Zákutný

Louny 2023

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 Lounech dne datum

.....

Kebrle Tomáš

Anotace

Práce popisuje vývoj a tvorbu aplikace mesh.sk. Mým cílem bylo vytvořit sociální síť, platformu pro učení a organizaci školy.

Klíčová slova

Sociální síť e-learning, umělá inteligence, škola

Annotation

This paper talks about the development and creation of app mesh.sk. My goal was to create a social network, platform for learning and organizing school.

Keywords

Social network, e-learning, artificial intelligence, school

PODĚKOVÁNÍ

Nejprve bych chtěl poděkovat Milanu Šedivému za pomoc se psaním samotné práce, jeho připomínky mi velice pomohli při zhotovení této práce. Dále bych chtěl poděkovat panu učiteli Zdeňkovi Zákutnému, za zodpovězení mých otázek ohledně soutěže SOČ. A také bych chtěl poděkovat Martinu Mayovi za konzultace ohledně „e-learningu“.

Úvod

V rámci své práce pro SOČ, jsem se rozhodl vytvořit aplikaci mesh.sk. Za cíl jsem si dal vytvořit aplikaci která by pomáhala studentům s učením, pomocí nejnovějších technologií v podobě umělé inteligence. Dále jsem chtěl vytvořit místo pro organizaci věcí týkající se školy, ve formě nástěnky s přehledem všech důležitých informací, jako jsou nejnovější výpisky, nebo úkoly, na jednom místě. V neposlední řadě jsem chtěl také vytvořit sociální síť, která by umožňovala se lépe propojit s ostatními spolužáky.

Obsah

Poděkování.....	10
Úvod.....	11
1 Koncept aplikace.....	14
1.1 Sociální síť pro „e-learning“	14
1.2 Pomoc při učení.....	15
2 Technologie	17
2.1 <i>Frameworky a programovací jazyky</i>	17
2.1.1 Svelte	17
2.1.2 SvelteKit	22
2.1.3 Typescript	22
2.1.4 Pocketbase a GO.....	22
2.2 Infrastruktura.....	24
2.2.1 PNPM Workspaces a Turborepo	24
2.2.2 Docker.....	25
2.2.3 Digital Ocean	26
2.2.4 Vercel.....	26
2.2.5 Git a Github	27
2.3 Design aplikace	29
2.3.1 Figma	29
2.3.2 TailwindCSS.....	31
3 Výsledná aplikace	32
3.1 Autentizace.....	32
3.2 Osobní nástěnka	33
3.3 Psaní výpisků	35
3.4 Automatické vytváření kartiček na učení.....	36
3.5 Správce úkolů.....	38

3.6	Třídy	39
3.6.1	Nástěnka třídy	39
3.6.2	Osobní profily	41
3.6.3	Připojení do třídy	41
3.7	Lokalizace	42
4	Závěr	44
4.1	Budoucí plány	44
4.2	Zkušenosti	44
5	Literatura.....	46

1 KONCEPT APLIKACE

1.1 Sociální síť pro „e-learning“

Prvotním návrhem pro mojí aplikaci byla sociální síť, zaměřena ovšem pouze na školu, přesněji by to byla sociální síť pro třídy.

Sociální síť jsem se snažil navrhnout tak aby podporovala studenty ve vzdělávání, ale také aby jim pomáhala naučit se pracovat ve skupině. Tohle je již známý koncept tzv. „e-learning“.

Hlavní funkce, které jsem chtěl, aby aplikace měla jsou:

- Možnost vytvořit si vlastní profil, a ten dále doplnit o osobní informace, například koníčky, odkazy na další sociální sítě, oblíbené filmy, seriály, hudební skupiny atd...
- Vytváření skupin neboli tříd, kam by se jednotliví studenti mohli připojit
- Komunikační nástěnka pro každou třídu. Na této nástěnce by každý student mohl vytvořit vlastní příspěvek. Ostatní členové třídy by mohli pod příspěvkem vést diskusi ve formě chatu.
- List všech členů skupiny, aby se ostatní členové mohli podívat na profily svých spolužáků.
- Možnost vytvářet si předměty, pro kategorizování poznámek a úkolů.
- Interaktivní textový editor pro zapisování si výpisků z jednotlivých předmětů.
- Plánovač úkolů pro jednodušší organizaci domácích úkolů.
- Možnost hromadně sdílet úkoly a výpisky s jednotlivými třídami.
- Automatické vytváření testovacích kartiček z probrané látky pomocí analyzování výpisků.
- Grafické prostředí pro vyzkoušení si znalostí z těchto kartiček

Aplikace by těmito funkcemi pokládala dobrý základ pro rozvíjení studentů pomocí skupinového učení, jak ukazují autoři (Binesh Sarwar, 2019). V tomto článku autoři také zmiňují že tento způsob výuky může mít pozitivní dopad na studenty, v podobě zvýšené motivace k učení, lepšímu zapojení studentů s ostatními spolužáky a zlepšení výkonu studentů.

Výše zmíněná studie vyznačuje hlavní body, které by aplikace měla mít, aby bylo dosaženo výše zmíněných benefitů. Těmito body jsou:

- Nástroje pro komunikaci a spolupráci. Tyto nástroje v aplikaci jsou obsaženy formou příspěvků na společnou nástěnkou, a možností přidávat komentáře pod příspěvek.
- Nástroje pro tvorbu a sdílení obsahu. Obsah jde v aplikaci vytvářet pomocí textového editoru který dále umožňuje výpisky sdílet pomocí odkazu, jednotlivým třídám. Pro pokročilejší obsah lze využít specializovaných nástrojů, tento obsah poté exportovat do souborů, které jdou sdílet pomocí příspěvku.
- Integrace systému řízení výuky a nástroje pro hodnocení a zpětnou vazbu, by mohli být v budoucnu implementovány vytvořením speciální učitelské role která by mohla rozdávat body.

1.2 Pomoc při učení

Vzhledem k již představeným funkcím aplikace by dále mohla pomáhat při učení látky ze školy, to je také již známý koncept tzv. „blended learning“, který kombinuje formu prezenční výuky s konceptem „e-learningu“. (Eger, 2004)

Studenti by se naučili látku ve škole, kde by si jí také zapsali do aplikace, poté by si doma látku mohli zopakovat pomocí vytváření tzv „učících se kartiček“. To jsou speciální kartičky kdy na jedné straně kartičky se nachází otázka na něco, co se snažíme naučit, na druhé straně je poté napsaná odpověď. Tyto kartičky by byly vytvořené automaticky pomocí umělé inteligence, která by analyzovala výpisky ze školy, a podle nich vytvořila otázky. Umělá inteligence by také navíc vytvářela špatné odpovědi, které by byli použity v kvízu, ve kterém by si studenti otestovali svoje znalosti.

Po vytvoření kartiček by si je studenti dále mohli exportovat do specializovaného softwaru na učení se pomocí tzv. metody „rozloženého opakování“. Tento styl učení se ukázal jako efektivnější, dle jedné studie (Kornell, 2009), kde byli testováni 2 skupiny lidí, které se měli naučit pokročilejší anglické slova. První skupina se učila po 5 slovech v jeden den, tak že ke každé kartičce došli 4krát. Druhá skupina využívala metodu rozloženého opakování a učili se všechny kartičky postupně v průběhu 4 dnů, každou kartičku také viděli 4krát.

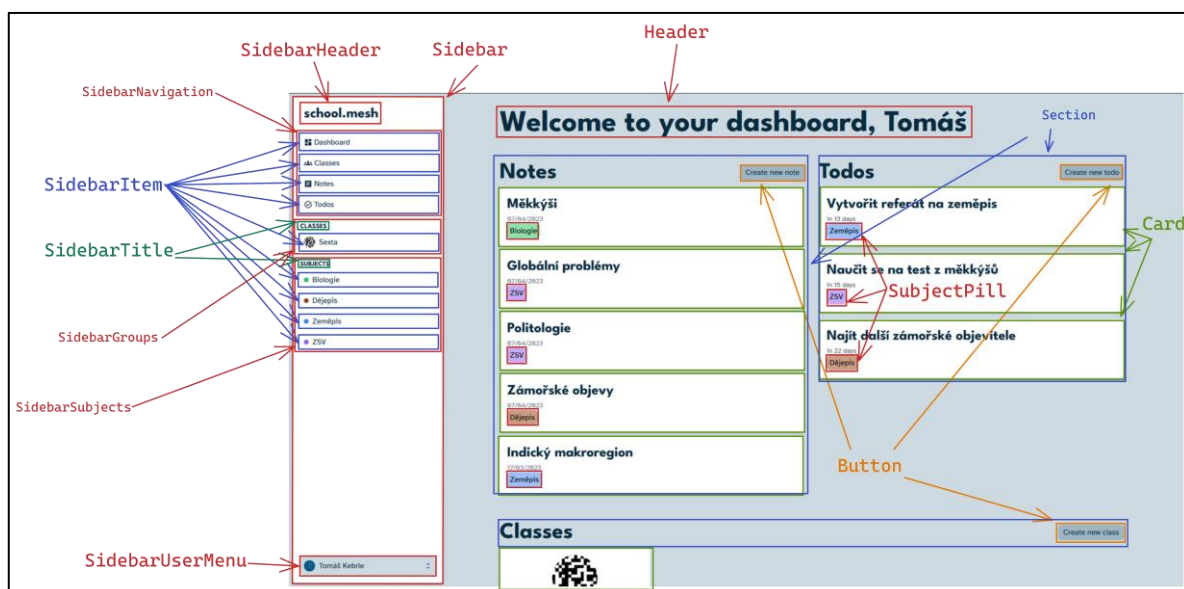
Skupina, která se učila právě pomocí této metody, měla průměrné skóre 49 %, zatímco druhá skupina měla průměrné skóre 36 %.

2 TECHNOLOGIE

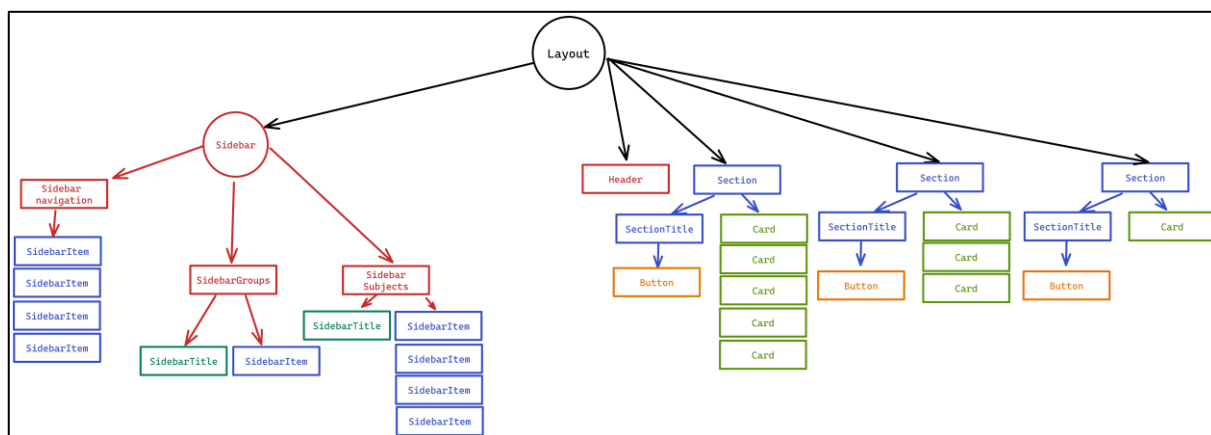
2.1 Frameworky a programovací jazyky

2.1.1 Svelte

Pro vytváření vzhledu a funkcionality mé aplikace jsem se rozhodl použít komponentový framework Svelte. Ten umožňuje rozdělit různé části aplikace do jednotlivých komponentů, a ty potom používat skrz aplikaci. (Wikipedia, 2023) Komponenty také můžou mít v sobě další komponenty, tím stavíme hierarchii neboli strom komponentů aplikace.



Obrázek 1: Vyznačení komponentů v aplikaci. Zdroj: Autor



Obrázek 2: Příklad hierarchie komponentů, vyznačena podle obrázku 1 Zdroj: Autor

Svelte jsem vybral pro jeho deklarativní způsob vytváření aplikaci, oproti imperativnímu přístupu Javascriptu, který je těžší jak na psaní kódu, tak i na následné čtení a jeho údržbu

2.1.1.1 Porovnání Javascriptu, Reactu a Svelte

Pro porovnání vytvořím jednoduchou aplikaci, která bude zobrazovat tlačítko, po jehož kliknutí se proměnná `count` zvýší o 1 a zobrazí se zpátky na stránce. Na stránce také bude ukázaná hodnota `count` ale vynásobena dvakrát.

```
1 <button id="my-button">
2   Current count: 0
3 </button>
4
5 <p id="doubled">Count doubled is 0</p>
```

```
1 let count = 0;
2 const button = document.getElementById("my-button");
3 const doubled = document.getElementById("doubled")
4
5 button.addEventListener("click", () =>{
6   count++;
7   button.innerHTML = `Current count: ${count}`
8   doubled.innerHTML = `Count doubled is ${count * 2}`
9 })
```

Ukázka kódu 1: Provedení pomocí běžného Javascriptu Zdroj: Autor

```

1  export default function Counter() {
2      const [count, setCount] = useState(0);
3      const doubled = useMemo(() => count * 2, [count])
4
5
6      return (
7          <>
8              <button onClick={() => {setCount(count + 1)}}>
9                  Current count: {count}
10             </button>
11             <p>Doubled count is: {doubled}</p>
12         </>
13     )
14 }

```

Ukázka kódu 2: Provedení pomocí Reactu. Zdroj: Autor

```

1  <script>
2      let count = 0;
3      $: doubled = count * 2;
4  </script>
5
6  <button on:click={() => (count++)}>
7      Count is {count}
8  </button>
9  <p>Doubled count is {doubled}</p>

```

Ukázka kódu 3: Provedení pomocí Svelte. Zdroj: Autor

Jak lze vidět syntaxe Svelte je mnohem jednodušší, také je to jeden z cílů Svelte vytvořit co nejminimálnější syntax pro své komponenty. Protože věří v to, že počet chyb v kódu je přímo propojený s počtem řádků kódu. (Harris, 2019)

Obrovským rozdílem mezi běžným Javascriptem, je také fakt že Svelte i React spojují logiku a vzhled stránky do jednoho souboru neboli komponentu. Javascriptový příklad je rozdělený na dvě části, které obě představují jeden soubor. Pokud se koukáme pouze na HTML soubor nemůžeme se nijak dozvědět co kliknutí na tlačítko dělá. Zatímco v Reactu a Svelte jde na první pohled vidět, že kliknutí na tlačítko něco udělá.

Name	svelte-v3.50.1	vue-v3.2.47	react-v18.2.0
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	1,876.7 ± 0.5 (1.04)	2,077.0 ± 49.6 (1.15)	2,527.4 ± 48.5 (1.40)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	145.7 ± 0.0 (1.02)	197.0 ± 0.0 (1.38)	281.6 ± 0.0 (1.97)
geometric mean of all factors in the table	1.03	1.26	1.66

¹ Pro porovnání výsledků pouze mezi Svelte, Vue a React navštivte https://krausest.github.io/js-framework-benchmark/2023/table_chrome_112.0.5615.49.html#eyJmcmFiZXZvcmtzIjpbIm5vbi1rZXIlZC9yZWJfdCIsIm5vbi1rZXIlZC9zdmVsdGUlLCJub24ta2V5ZWQvdnVlIl0sImJlbmNobWVya3MiOlsiMDFfcjVuMWSiLCIwMl9yZXBsYWNIWWSiLCIwM19iCGRhdGUxMHRoMWFfeDE2IiwiMDRfc2VsZWNOmWSiLCIwNV9zd2FwMWSiLCIwNl9yZW1vdmUtb25lTFRlwiMDdfY3JlYXRlMTBrlwiMDhfY3JlYXRlMWstYWZ0ZXIxa194MiIsIjA5X2NsZWYyMWtfeDgiLCIyMV9yZWFeS1tZW1vcnkiLCIyMl9ydW4tbWVtb3J5IiwiMjNfdXBkYXRlNS1tZW1vcnkiLCIyNV9ydW4tY2xiYXItbWVtb3J5IiwiMjZfcjcnVuLTEway1tZW1vcnkiLCIzMV9zdGFyYHVvLWNpIiwiMzRfc3Rhcnc1cC10b3RhbgJ5dGVzIl0sImRpc3BsYXIuNb2RlIjowLCJjYXRlZ29yaWVzIjpbMSwYLDMsNCw1XX0=

2.1.2 SvelteKit

Svelte samotný je ale dobrý pouze na vytváření samostatných komponentů, pro vytváření aplikace je ale dobré mít i pokročilejší funkce jako je tzv. „router“, neboli systém, který podle URL adresy zobrazí správnou stránku, v našem případě správný Svelte komponent. SvelteKit router nám ale dále dovoluje vytvářet tzv. „layouty“, což jsou komponenty, které budou zobrazeny na vícero stránkách. Například v mojí aplikaci mám hlavní layout s postranní navigací, který jde vidět navíce stránkách, a při změně adresy zůstává na místě. (Svelte, 2023)

SvelteKit dále obsahuje ale i jiné funkce, jako je načítání dat pomocí tzv. „load“ funkcí, ve kterých získáme data na serveru, které jsou poté předány Svelte komponentu, tyto data mají také automaticky vygenerované typy díky Typescriptu, díky čemuž nám editor poté dokáže pomoci data zpracovat.

Dále můžeme i definovat tzv. „form actions“, neboli funkce, které na serveru zpracují data, které byli předány pomocí form elementu. (Svelte, 2023)

2.1.3 Typescript

Ačkoliv jsem výše porovnával Javascript a Svelte, a došel jsem k závěru, že Svelte je lepší, tak se stejně nevyhneme Javascriptu, Svelte je jenom rozšíření pro Javascript, a musíme pamatovat, že prohlížeče nerozumí Sveltu, a tak ve finále Svelte vygeneruje optimalizovaný Javascript pro prohlížeč. Také kód, který se nachází ve script tagu jakéhokoliv Svelte komponentu, je napsaný v Javascriptu.

Javascript má ale stále i jiné nedostatky, tím hlavním je právě jeho dynamický typový systém, který je sice skvělý pro začátečníky, ale v komplexnějších aplikacích se stává tohle nevýhodou, jelikož je potom mnohem jednodušší vytvořit chyby v kódu. Typescript tento problém řeší tím, že přidává statický typový systém, který kontroluje, jestli je vše v pořádku. (Typescript, 2023)

2.1.4 Pocketbase a GO

Pro rychlejší development a škálovatelnost backendu, jsem zvolil framework Pocketbase. Pocketbase automaticky vytváří REST API pro SQLite databázi. Přes REST API poté

můžeme jednoduše získávat data z databáze a zobrazit je uživateli. Pocketbase navíc nabízí tzv. „SDK“ které nám tuto práci ještě usnadní.

```
1 const record = await pb
2   .collection("users")
3   .getOne<UsersResponse>("<id uživatele>");
```

Ukázka kódu 4: Získání dat určitého uživatele pomocí Pocketbase SDK. Zdroj: Autor

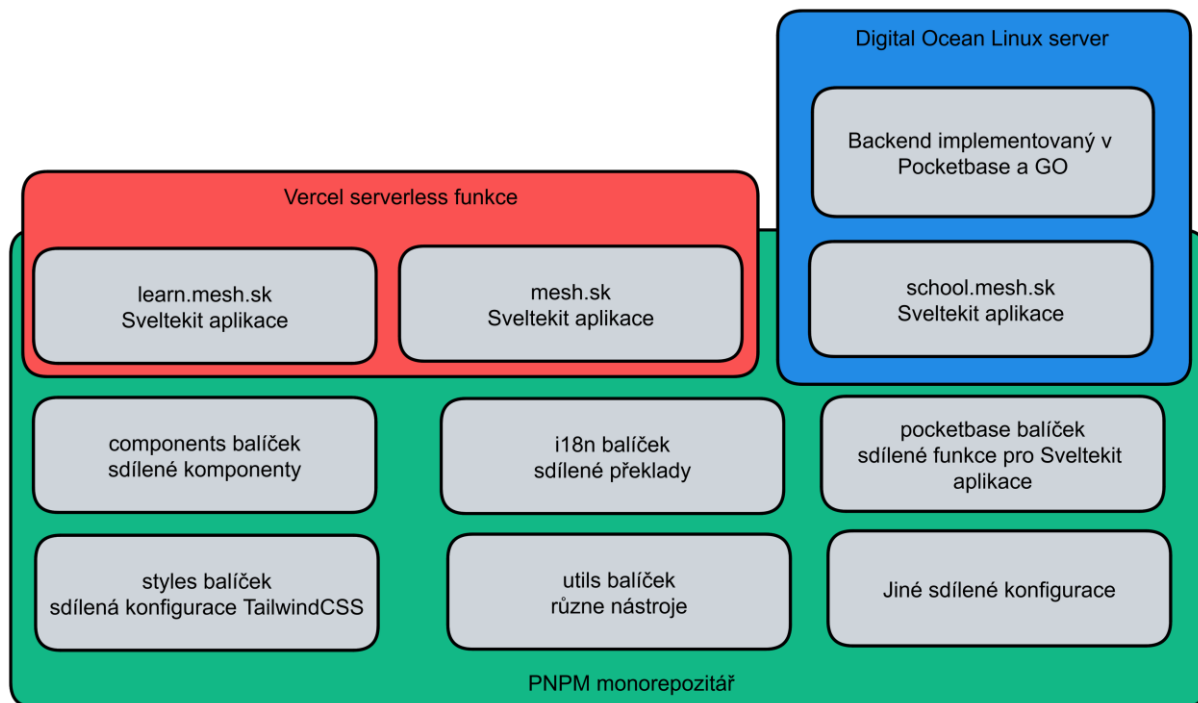
Dále vytváří administrační panel, kde jde pomocí grafického rozhraní nastavit struktura dat v databázi, a také pravidla které určují kdo má přístup ke kterým datům, díky tomu je aplikace zabezpečená.

Dále je taky automaticky vytvořeno tzv. „realtime“ API, které v reálném čase posílá informace o změnách klientovi, pomocí tzv. „Server-Sent Events“ technologie, díky čemuž je například implementovaný chat v této aplikaci.

Pocketbase je vytvořený v GO, a tímto jazykem jde potom také dále rozšířit o vlastní funkce. Tyto funkce můžou přidat funkcionalitu různým akcím které mohou proběhnout na serveru, jako například před nebo po přidání dat do databáze.

2.2 Infrastruktura

2.2.1 PNPM Workspaces a Turborepo



Obrázek 5: Ukázka struktury monorepozitáře. Zdroj: Autor.

Jelikož je celá aplikace vytvořena jako více aplikací dohromady které spolu komunikují přes společný backend rozhodl jsem se aplikaci vytvořit jako monorepozitář. To mi dovoluje sdílet různé knihovny neboli balíčky, napříč různými aplikacemi. Abych tyhle balíčky mohl sdílet využil jsem technologie PNPM Workspaces. Tato technologie jednotlivé balíčky v repozitáři propojuje mezi sebou. V aplikaci poté můžu jednotlivé části kódu z různých balíčků importovat.

```
1 import {Button} from "@mesh/components"
```

Ukázka kódu 5: Ukázka importování tlačítka z balíčku s komponentami. Zdroj: Autor

Pro optimalizaci monorepozitáře dále využívám Turborepo, které mi dovoluje spustit jednotlivé aplikace vzájemně na různých jádrech procesoru, aby byl vývoj aplikací rychlejší.

2.2.2 Docker

Docker je nástroj pro vytváření a spouštění aplikací v izolovaném a standardizovaném prostředí, které se nazývá kontejner. Nejprve vytvoříme instrukce pro Docker, ve kterém popíšeme, co všechno potřebuje naše aplikace, aby mohla být spuštěna, například specifikujeme, jaký operační systém, které knihovny má docker nainstalovat a více. Poté můžeme vytvořit obraz našeho Docker kontejneru, který můžeme spustit s tím, že si můžeme být jistí, že naše aplikace se nerozbije vlivem externích podmínek, protože je izolovaná. Náš kontejner stále sdílí jádro s hostitelským systémem, ale je izolovaný od ostatních procesů čímž zajišťuje větší výkonnost oproti simulování virtuálního počítače. (Docker, 2023)

```
1 FROM golang:1.19-alpine
2
3 RUN apk add build-base
4
5 WORKDIR /app
6
7 # download Go modules and dependencies
8 COPY go.mod ./
9 RUN go mod download
10
11 COPY ./ ./
12
13 RUN go build
14
15 EXPOSE 8080
16
17 # start PocketBase
18 CMD ["/app/backend", "serve", "--http=0.0.0.0:8080"]
```

Ukázka kódu 6: Dockerfile pro budování backendu

Pro deployování, tj. nasazení aplikace na server a její zpřístupnění na internetu, využívám právě docker. Pomocí něho nasazuji na server backend implementovaný v GO, a school.mesh aplikaci.

Díky dockeru můžu kontejnerizovat svojí aplikaci, a jelikož replikuje konzistentní prostředí tak si můžu být jistý, že se aplikace nerozbije vlivem externích podmínek.

Pro orchestraci Docker kontejnerů dále využívám Docker compose, pomocí něho spouštím navzájem backend v GO, school.mesh frontend v Javascriptu, a Litestream na zálohování dat v SQLite databázi do S3 úložiště.

2.2.3 Digital Ocean

Aby aplikace byla přístupná světu, musí také být spuštěna na nějakém serveru. Jelikož jsem neměl prostředky na využívání vlastního domácího serveru, rozhodl jsem se využít možnosti pronajmout si server. Dnes existuje velká řada firem, které nabízejí tyto služby, jako například Amazon AWS, Microsoft Azure nebo Google Cloud Platform. Nakonec jsem se rozhodl pro platformu Digital Ocean, jelikož nastavení serveru bylo jednoduché, a ceník byl na rozdíl od výše zmíněných služeb, více přehledný. Navíc tato platforma nabízela další funkce, které jsem také potřeboval jako je S3 kompatibilní blokové úložiště, nebo Docker container registry kam ukládám jednotlivé obrazy Docker kontejnerů.

2.2.4 Vercel

Pro zpřístupnění ostatních aplikací jsem využil firmy Vercel. Konfigurace byla vcelku jednoduchá, platforma automaticky detekovala že používám Turborepo, což je produkt této firmy, a na mně už zbývalo přidat adaptér pro Vercel do mých SveltKit aplikací. Platforma se poté připojí k mému GitHub repozitáři a při každém novém commitu automaticky vybuduje a nasadí novou verzi aplikace.

Aplikace jsou nasazeny pomocí tzv. „serverless“ infrastrukturu, při které není stále v provozu jeden server, ale server je nastartován jen když přijde požadavek na načtení stránky. Díky této infrastruktuře navíc platím pouze za čas, který serverless funkce využijí, a ne za celou dobu co je server v provozu, jako například u Digital Ocean. Další výhodou je také jednodušší škálování místo nastavování dalšího serveru, instalování dockeru, a psaní GitHub akce která by kód automaticky nasadila do produkce na novém serveru, se prostě může upravit maximální počet serverless funkcí které můžou běžet, což je mnohem jednodušší.

Nevýhodou je že tyto funkce mají omezenou dobu životnosti, kvůli tomu je aplikace school.mesh na serveru, jelikož generování kartiček pomocí umělé inteligence trvá déle než je maximální doba po kterou můžou být serverless funkce spuštěny. Dále nemůžou ukládat žádná data, proto je nutné, aby databáze byla na samostatném serveru.

2.2.5 Git a GitHub

Pro verzování kódu jsem využil technologie Git. Ta mi dovolovala ukládat historii mého kódu v tzv. „commitech“. Pro zálohování této historie jsem využil služby GitHub, kam sem kód nahrával. Tato služba mně dále dovolovala organizovat úkoly na projektu pomocí funkce GitHub issues, kde pro každou věc, kterou jsem chtěl přidat nebo opravit jsem si vytvořil tzv. „issue“, ke každé issue jsem poté vytvořil v Gitu větev, na které jsem přidával commity, dokud issue nebyla hotová. Poté jsem využil funkce Pull request, která kód z této větve sjednotila zpátky do hlavní větve.

Dále GitHub nabízí tzv. „GitHub Actions“, které dovolují spustit různé akce, podle toho co se změnilo v Git repozitáři. V mém případě spouštím GitHub akci při každé změně kódu na hlavní větvi.

```
1 name: Deploy pocketbase to Digital Ocean
2 on:
3   push:
4     branches: [main]
5
6 jobs:
7   build-and-push-image:
8     runs-on: ubuntu-latest
9
10    steps:
11      - name: Checkout the repo
12        uses: actions/checkout@master
13
14      - name: Install `doctl`
15        uses: digitalocean/action-doctl@v2
16        with:
17          token: ${ secrets.DIGITAL_OCEAN }
18
19      - name: Log in to DO Container Registry
20        run: doctl registry login --expiry-seconds 600
21
22      - name: Set up Docker Buildx
23        uses: docker/setup-buildx-action@v2
24
25      - name: Build @mesh/backend
26        uses: docker/build-push-action@v4
27        with:
28          context: ./backend
29          push: true
30          tags: 'registry.digitalocean.com/mesh-containers/mesh-
```

```

31 backend:latest'
32     cache-from: type=gha
33     cache-to: type=gha,mode=max
34
35     - name: Install PNPM
36       uses: pnpm/action-setup@v2
37       with:
38         version: latest
39         run_install: |
40           - recursive: true
41             args: [--no-frozen-lockfile]
42
43     - name: Build dependencies
44       run: pnpm run build --filter=@mesh/school
45
46     - name: Build @mesh/school
47       uses: docker/build-push-action@v4
48       with:
49         context: ./apps/school
50         push: true
51         tags: 'registry.digitalocean.com/mesh-containers/mesh-
52 school:latest'
53     cache-from: type=gha
54     cache-to: type=gha,mode=max
55
56   deploy:
57     runs-on: ubuntu-latest
58     needs: build-and-push-image
59     steps:
60       - name: Restart docker-compose
61         uses: appleboy/ssh-action@v0.1.7
62         with:
63           key: ${ secrets.SSH_PRIVATE_KEY }
64           username: ${ secrets.SSH_USERNAME }
65           host: ${ secrets.SSH_HOST }
66           passphrase: ${ secrets.SSH_PASSPHRASE }
67           script_stop: true
68           script: |
69             cd ~/mesh
70             git pull --force
71             docker-compose pull
72             docker-compose up -d
73             cp /root/mesh/Caddyfile /etc/caddy/Caddyfile
74             caddy reload
75             echo "Successfully deployed, hooray!"

```

Ukázka kódu 7: GitHub akce která nasadí kód do produkce. Zdroj: Autor

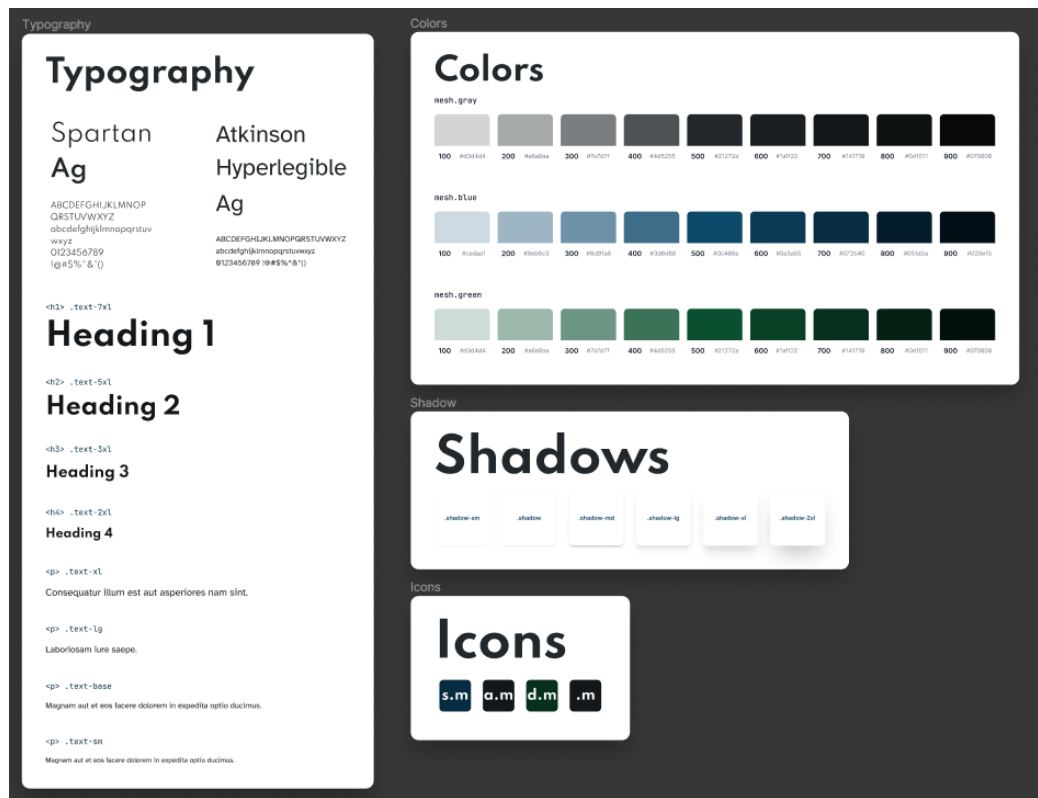
Tato akce nejdříve zpřístupní kód z repozitáře ve svém prostředí, aby se s ním dalo dále pracovat. Poté nastaví Docker buildx, který využívám na budování docker kontejnerů, a

přihlásí se do privátní docker registry, kam se poté nahraje obraz docker kontejneru, který bude v rámci akce vybudován. Po vybudování obou docker kontejnerů a jejich nahrání do docker registry, se akce připojí pomocí tzv. „SSH“ protokolu na server na kterém běží backend a school.mesh aplikace, tam nejdříve stáhne nové obrazy docker kontejnerů a poté restartuje docker compose, který už spustí novou verzi aplikace. Dále ještě restartuje Caddy což je webserver, který používám jako tzv. „reverse proxy“, což znamená že všechny požadavky jdou nejdřív přes tuto proxy, která je poté nasměruje na správný proces spuštěný v Dockeru. Caddy hlavně využívám díky funkci automatického vytváření SSL certifikátu pro zabezpečení mé aplikace pomocí HTTPS protokolu.

2.3 Design aplikace

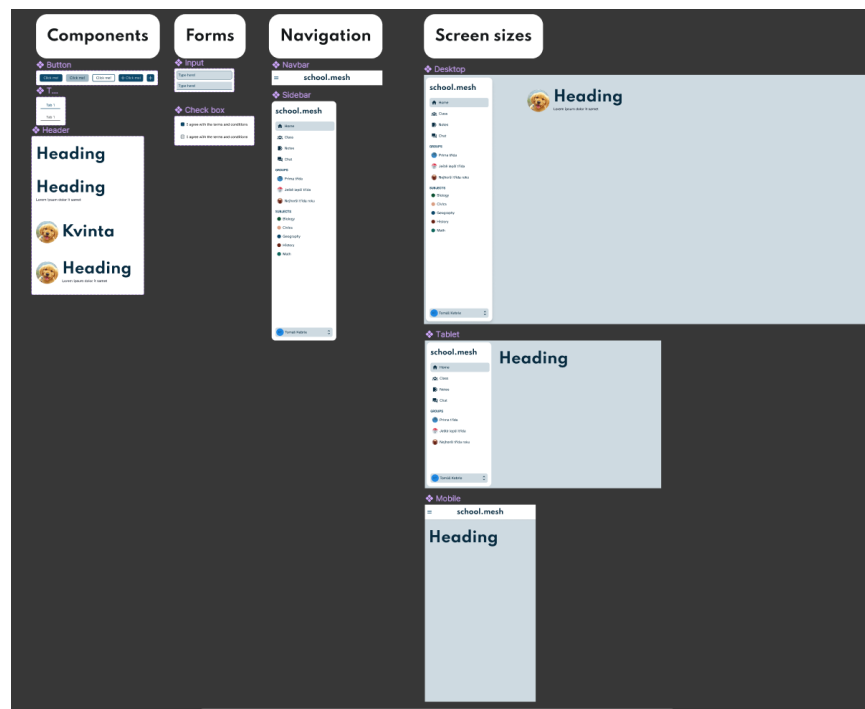
2.3.1 Figma

Pro implementaci vzhledu jsem využil software Figma, pomocí něho jsem navrhl designový systém neboli sadu pravidel, která určují, jak postupovat při návrhu jednotlivých částí aplikace. V tomto systému jsem určil pravidla jako třeba které fonty bude aplikace používat, jaké barvy se v aplikaci budou objevovat, velikost stínů, také jsem si určil, jak se aplikace má měnit když bude spuštěna na menších obrazovkách.



Obrázek 6: Přehled design systému ve Figmě. Zdroj: Autor

Dále jsem ve Figmě navrhl design některých komponentů, který jsem poté vytvořil ve Svelte.



Obrázek 7: Ukázka navržení některých komponentů. Zdroj: Autor

2.3.2 TailwindCSS

Ve Figmě jsem sice udělal návrhy, ale aby tyto komponenty bylo možné využít v aplikaci musí se i vytvořit pomocí kódu. Běžně se pro popisování vzhledu ve webových aplikacích používá CSS, já jsem si ale vybral TailwindCSS, protože mi dovozoval jednotlivé komponenty budovat rychleji a ušetřil mi také čas nad přemýšlením, jak co pojmenovat díky tomu že se všechny styly jsou rovnou definované na HTML tagu pro který je chceme aplikovat, takže ho nemusíme speciálně pojmenovávat a v CSS poté definovat.

S TailwindCSS bylo také jednodušší definovat různá pravidla které jsem vytvořil jako třeba barvy a fonty, a díky monorepozitáři bylo jednoduché tyto pravidla i sdílet napříč aplikacemi

```
1 <!-- TailwindCSS -->
2 <div class="bg-white p-4 my-2 rounded-md
3     shadow-md hover:shadow-lg duration-300">
4   <h4 class="text-2xl font-bold text-mesh-700">Nadpis</h4>
5   <p> Lorem, ipsum dolor sit amet.</p>
6 </div>
7
8 <!-- CSS -->
9 <div class="card">
10   <h4 class="title">Nadpis</h4>
11   <p> Lorem ipsum dolor sit amet.</p>
12 </div>
13
14 <style>
15   .card {
16     background-color: white;
17     padding: 16px;
18     margin-top: 8px;
19     margin-bottom: 8px;
20     border-radius: 16px;
21     box-shadow: 0 2px 4px -1px rgba(0, 0, 0, 0.06);
22     transition: all;
23     transition-duration: 300ms;
24   }
25
26   .card:hover {
27     box-shadow: 0 4px 6px -2px rgba(0, 0, 0, 0.05);
28   }
29
30   .title {
31     font-size: 24px;
32     line-height: 28px;
33     color: #0d47a1;
34     font-family: "Spartan", sans-serif;
35   }
36 </style>
```

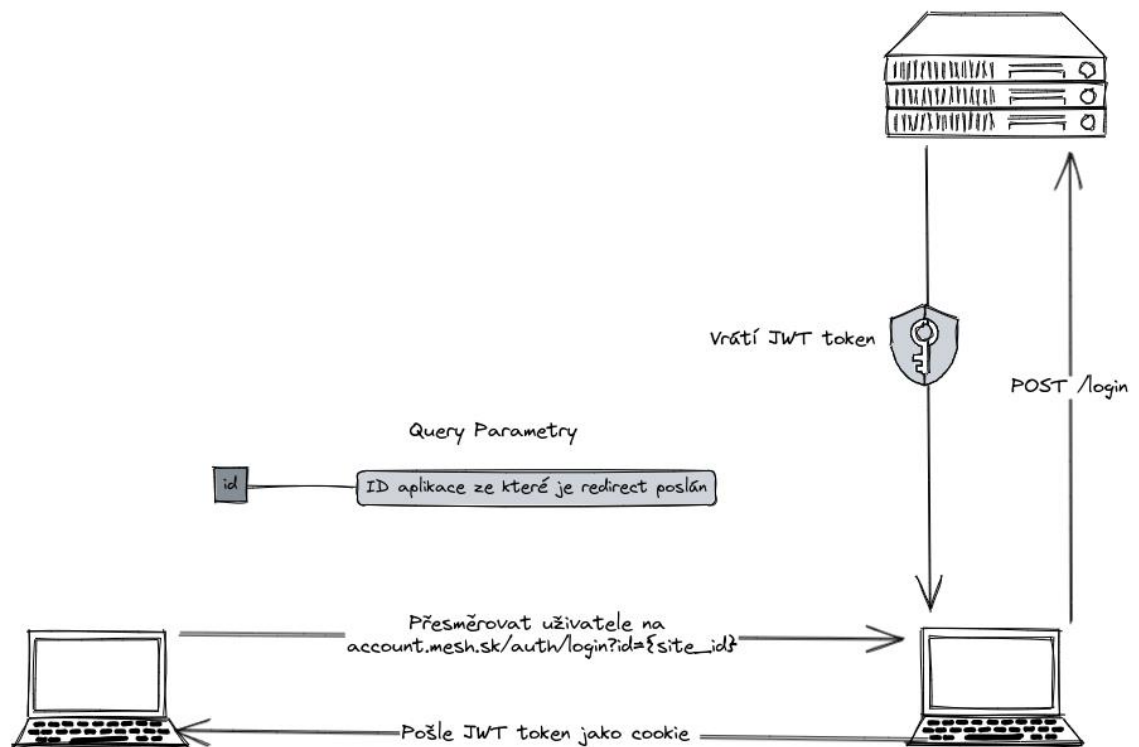
3 VÝSLEDNÁ APLIKACE

3.1 Autentizace

Jelikož jsem v rámci projektu vybudoval více aplikací, které ale všechny sdílí společný backend, potřeboval jsem vytvořit autentizaci tak aby fungovala napříč všemi aplikacemi. To jsem vyřešil vybudováním další aplikace, která slouží jenom k registraci uživatelů, a přihlašování. Protože tato aplikace je spuštěna na doméně mesh.sk může nastavovat tzv. „cookies“ pro všechny aplikace které existují na stejné doméně, nebo subdoméně. Což znamená že každá aplikace, která je spuštěna na doméně mesh.sk, má přístup k cookies, které byli nastaveny právě touto aplikací.

Uživatel se na této stránce registruje nebo přihlásí, tím pošle požadavek serveru, který vygeneruje JWT token. To je speciální token, který je kryptograficky podepsán speciálním heslem, tak aby backend vždycky mohl verifikovat, že uživatel, který poslal požadavek je opravdu ten, za kterého se vydává. (Auth 0, 2022)

Aplikace samotné jsou poté zabezpečeny pomocí speciálního souboru `hooks.server.ts` tento soubor exportuje speciální funkce `Handle`, která je spuštěna při každém požadavku ve SvelteKitu, jako je načtení nové stránky, nebo odeslání dat pomocí form akce. (Svelte, 2023) V této funkci je poté pomocí Pocketbase uživatel verifikován, pokud jeho token neprojde kryptografickou verifikací, což znamená že podpis se neshoduje s tím na tokenu, nebo je token už prošlý, tak je uživatel přesměrován na stránku kde se může přihlásit.



Obrázek 8: Diagram autentizace. Zdroj: Autor

3.2 Osobní nástěnka

Po přihlášení uvítá každého uživatele jeho osobní nástěnka. Na této nástěnce se každému uživatele zobrazují všechny důležité informace, jako například jsou nejnovější výpisky, nejbližší úkoly, všechny třídy, do kterých se připojil a také předměty které si vytvořil.

Tato nástěnka načítá dost informací, a tak je důležité, aby se načetla rychle. Proto jsem přepsal kód načítání, tak aby nezískával data z databáze postupně, ale aby je získal všechny najednou. Pocketbase SDK pro získávání dat používá standardní Javascriptové Promise API, takže stačilo poté použít metodu `Promise.all`, které spustila všechny funkce na získání dat současně. (MDN, 2023)

```

1 async function load({ locals }) {
2   const userRecord = locals.pb
3     .collection('users')
4     .getOne<UsersResponse>(locals.user.id, {
5       expand: 'groups'
6     });
7
8   const subjects = locals.pb.collection('subjects')
9     .getFullList<SubjectsResponse>();
10
11  const latestNotes = locals.pb.collection('notes')
12    .getList<NotesResponse>(0, 5, {
13      sort: '-created'
14    });
15
16  const latestTodos = locals.pb.collection('todos')
17    .getList<TodosResponse>(0, 5, {
18      sort: 'date',
19      filter: 'status = false',
20      expand: 'user'
21    });
22
23  const promises = [userRecord,
24    subjects,
25    latestNotes,
26    latestTodos]
27
28  const [userData, subjectsData, notesData, todosData] = await
29    Promise.all(promises)
30
31  return {
32    user: userData,
33    subjects: subjectData,
34    notes: notesData,
35    todos: todosData
36  }
37 }

```

Ukázka kódu 9: Optimalizovaná load funkce Zdroj: Autor

3.3 Psaní výpisků

Jedna z hlavních funkcí mé aplikace je možnost poznamenávat si výpisky. Jelikož jsem chtěl, aby aplikace mohla být použitelná i ve škole, je důležité, aby výpisky šlo snadno formátovat, protože by potom studenti, kteří píšou pomalu nemuseli stíhat, pokud by ještě museli klikat na tlačítka pro formátování. Proto jsem zvolil možnost udělat editor minimální, a k formátování použít tzv. „markdown“, který umožňuje rychle formátovat text pomocí speciálních symbolů. Například pokud chceme vytvořit nadpis napíšeme znak „#“, přidáním více „##“ se poté zvětšuje úroveň nadpisu. Pokud chceme text zvýraznit lze využít klávesových zkratk jako jsou Ctrl + B pro tučné písmo, nebo Ctrl + I pro kurzívu. Dále jde vytvářet listy s odrážkami napsáním „-“ a zmačknutím mezerníku, při každém zmačknutí klávesy Enter se vytvoří nová odrážka, to samé platí pro očíslované listy, pro jejich první vyznačení ale musíme použít číslovku s tečkou, například „1.“, poté již fungují stejně.



Obrázek 9: Ukázka editoru Zdroj: Autor

Výpisky jsou ukládány v podobě JSON formátu do databáze. JSON formát jsem zvolil oproti HTML, jelikož u něho nehrozí tzv. „XSS Injection“ útok, ke kterému by mohlo dojít při sdílení výpisků. XSS je typ útoku kde útočník uloží do databáze škodlivý kód, který by potom byl spuštěn při navštívení stránky napadeným uživatelem. Tento kód by za uživatele poté mohl provádět různé akce, jako například smazat jeho výpisky, úkoly, nebo změnit jeho profil. (MDN, 2023)

Ukládání probíhá automaticky, takže se uživatel nemusí o nic starat. Při každé změně v editoru je poslán požadavek na server se změněným obsahem poznámky. Avšak pokud v určitém časovém intervalu dojde ke změně je požadavek zrušen, a je vyslán nový s aktuálním obsahem. Tohle rušení je implementováno, aby server nezačal být přetížený, což by mohlo způsobit nepříjemné výpadky aplikace.

Výpisky je také možné sdílet do tříd. Uživatel vybere třídu, které chce zpřístupnit poznámku, a získá odkaz který může poslat ostatním členům třídy. Sdílení funguje tak že potom co uživatel zpřístupní svoji poznámku, tak je v databázi upravena hodnota `shared_to`, která v sobě drží unikátní identifikátor třídy, které poznámku sdílel. Jelikož na tokenu každého uživatele, který je poslán při každém požadavku na server je obsažena i informace o tom v kterých skupinách je členem, backendu stačí pouze porovnat, jestli hodnota `shared_to` z poznámky je obsažena v listu všech tříd kterých je uživatel členem, pokud ano tak je uživateli poznámka zpřístupněna.

3.4 Automatické vytváření kartiček na učení

Hlavní funkcí aplikace je možnost z výpisků automaticky vytvořit kartičky na učení se. Kartičky jsou vytvářeny pomocí umělé inteligence, přesněji využívám textový model `gpt-3.5-turbo`, tento model také stojí za dalšími aplikacemi jako například ChatGPT. Základem pracování s těmito textovými modely je tzv. „prompt“, neboli příkaz, který předáme umělé inteligenci, aby věděla, co vlastně po ní chceme. V průběhu vývoje aplikace jsem zkoušel více těchto příkazů, a pak vybíral ty které fungovali nejlépe. Největším problémem je přinutit umělou inteligenci, aby odpovídala v nějakém standardizovaném formátu, který bych potom mohl využít. Nejdříve jsem zkoušel umělou inteligenci namluvit, že je REST API, a proto může odpovídat pouze ve formátu JSON, stále ale občas odpovídala s textem navíc, což dělalo problémy při zpracování. Dalším problémem bylo donutit umělou inteligenci, aby vytvářela co nejvíce otázek, občas se totiž stávalo že odpověď nebyla kompletní, kvůli limitaci na 2000 tokenů, cca 1750 slov.

První problém se mi podařilo vyřešit dodáním „CODE ONLY“ do příkazu, to donutí umělou inteligenci odpovídat pouze s kódem. Druhý problém se stále může objevit, jelikož se do počtu tokenů počítá i zpráva ve který jsou obsaženy výpisky (OpenAI, 2023), podle kterých umělá inteligence generuje svoji odpověď. Vyměnil jsem ale JSON za formát CSV, který je mnohem úspornější, jelikož se nemusí opakovat data. Díky tomu se vejde do zprávy více dat,

ale také klesne cena za použití umělé inteligence, protože se platí podle počtu tokenů, a méně dat znamená méně tokenů. Na rozdíl od JSONu, CSV také nemusí být nijak speciálně zakončeno, takže by mělo docházet k méně chybám v procesu zpracování.

Momentální příkaz, který používám pro nastavení umělé inteligence vypadá takto: „*You are a system that generates cards for learning based only on information you will be given in the message sent by the user. The response should be in a CSV format. Each card consists of question for which an answer can be found in the provided notes, answer to that question and wrong1, wrong2 and wrong3, which will all be wrong answers. For the delimiter use a semicolon. The language of the cards must be the same as the language of the notes you will be sent! Try to generate as much cards as you can. CODE ONLY*“

Příkaz je v angličtině, jelikož z mého testování jsem dostával lepší výsledky, pokud byl příkaz v angličtině. V příkazu říkám umělé inteligenci, že je systém na generování kartiček pro učení. Kartičky musí být vygenerovány pouze podle informací, které budou zaslány v příští zprávě. Odpověď by měla být v CSV formátu. Každá kartička se potom skládá z otázky, jejíž odpověď lze nalézt v poznámce, dále má kartička obsahovat odpověď na tuto otázku a wrong1, wrong2 a wrong3, což jsou všechno špatné odpovědi. Jako tzv. „delimiter“, neboli znak, který bude oddělovat data od sebe má použít středník. Ten jsem vybral, jelikož občas umělá inteligence chtěla do odpovědi použít čárku, což je ale znak běžně používaný pro oddělování hodnot v CSV formátu, což rozbilo zpracovávání odpovědi. Poté umělé inteligenci zdůrazníme že odpověď musí být ve stejném jazyce jako poznámka která jí bude poslána. Nakonec jí ještě řeknu, aby vygenerovala co nejvíce kartiček, a také že odpověď má být pouze ve formě kódu, což zajistí že odpověď bude obsahovat pouze data které potřebuji.

Umělá inteligence ale stále není perfektní, a občas může vytvořit chybné kartičky, nebo použít špatný jazyk, proto si stále musíme dávat pozor při využívání této funkce, a také bychom měli zkontrolovat to co vyprodukuje.

Po vytvoření kartiček si je můžou studenti exportovat do formátu pro aplikaci Anki. Tato aplikace využívá metodu „rozloženého učení“ popsanou v sekci *Koncept aplikace*. Poté co se pomocí této aplikace naučili vše, co potřebují, mohou využít testovací funkce, ve které si vyzkouší své znalosti.

3/11 Jak je vysoká hustota zalidnění v tomto makroregionu?

- A. 350 ob/km²
- B. 100 ob/km²
- C. 500 ob/km²
- D. 1000 ob/km²

Obrázek 10: Ukázka testovacího prostředí

Dokončeno

Správné odpovědi: 8

Špatné odpovědi: 3

Jaké státy jsou součástí tohoto makroregionu?

Odpověď jste : 2 Vnitrozemské, 2 Přímorské, 3 Ostrovní

Správná odpověď : 3 Vnitrozemské, 3 Přímorské, 2 Ostrovní

Jak je vysoká hustota zalidnění v tomto makroregionu?

Odpověď jste : 1000 ob/km²

Správná odpověď : 350 ob/km²

Který je charakterizován jako druhý nejchudší makroregion?

Odpověď jste : Jiný makroregion

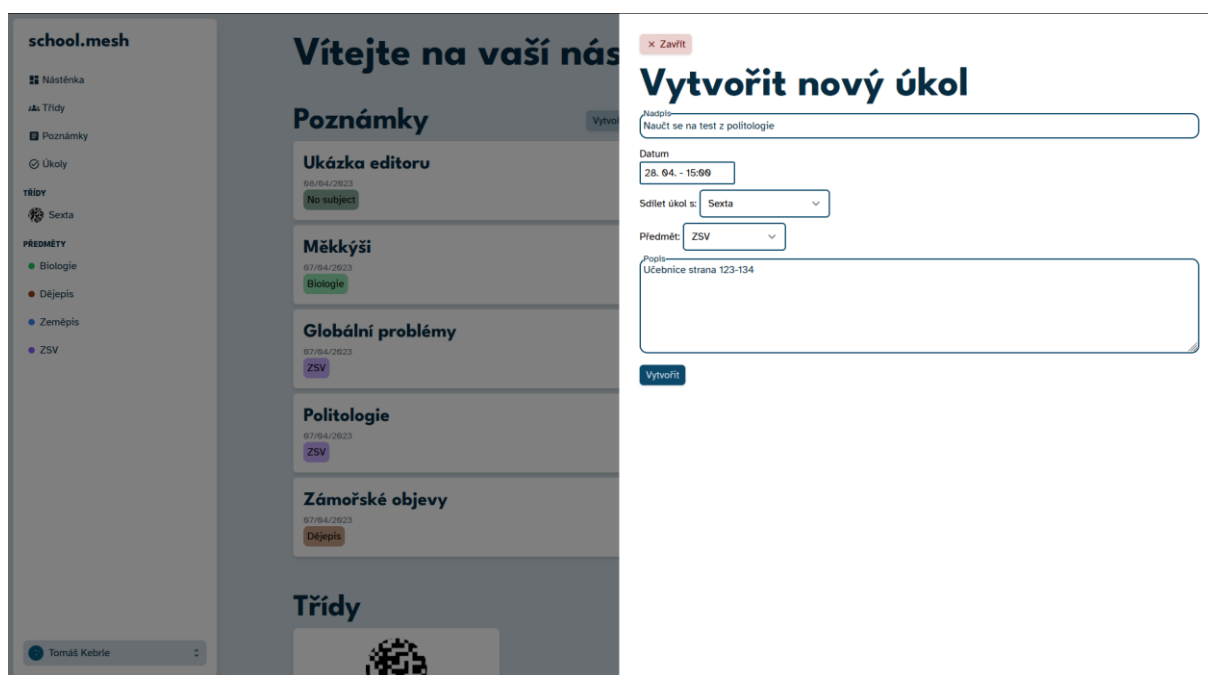
Správná odpověď : Tento makroregion

Obrázek 11: Ukázka dokončení testu

3.5 Správce úkolů

Se školou se dost často také pojí úkoly, proto jsem do aplikace přidal funkci na přidávání úkolů. Uživatel si může zapsat úkol přidat k němu popis, datum do kdy musí být splněn, a také je může kategorizovat pomocí předmětů. Úkoly jdou také sdílet s třídami, díky čemuž

stačí aby si zapsal úkol pouze jeden student ze třídy, a i tak o něm budou vědět ostatní. Sdílení je implementováno stejně jako sdílení výpisků, což znamená že u každého úkolu v databázi je uložena informace `shared_to`, která určuje, do jaké skupiny student sdílel úkol. Na rozdíl od výpisků, kde student musí po sdílení zkopírovat odkaz a poté ho přeposlat, úkoly jsou automaticky přidány do speciální záložky ve třídě s názvem úkoly, a také se automaticky přidávají do seznamu úkolů ostatních členů třídy.



The screenshot displays the 'school.mesh' interface. On the left is a sidebar with navigation links: 'Nástěnka', 'Třídy', 'Poznámky', 'Úkoly', 'Třídy', 'Sexta', 'PŘEDMĚTY', 'Biologie', 'Dějepis', 'Zeměpis', and 'ZSV'. The main content area is titled 'Vítejte na vaší stránce' and 'Poznámky'. It features a 'Ukázka editoru' section with a 'No subject' placeholder. Below this are several task cards: 'Měkkýši' (Biologie), 'Globální problémy' (ZSV), 'Politologie' (ZSV), and 'Zámořské objevy' (Dějepis). On the right, a modal form titled 'Vytvořit nový úkol' is open. It contains fields for 'Nápis' (Title), 'Datum' (Date), 'Sdílet úkol s:' (Share task with), 'Předmět:' (Subject), and a large text area for 'Popis' (Description). The 'Datum' field is set to '28. 04. - 15:00'. The 'Sdílet úkol s:' dropdown is set to 'Sexta'. The 'Předmět:' dropdown is set to 'ZSV'. The 'Popis' field contains the text 'Učebnice strana 123-134'. A 'Vytvořit' (Create) button is at the bottom of the form.

Obrázek 12: Ukázka vytváření úkolu. Zdroj: Autor

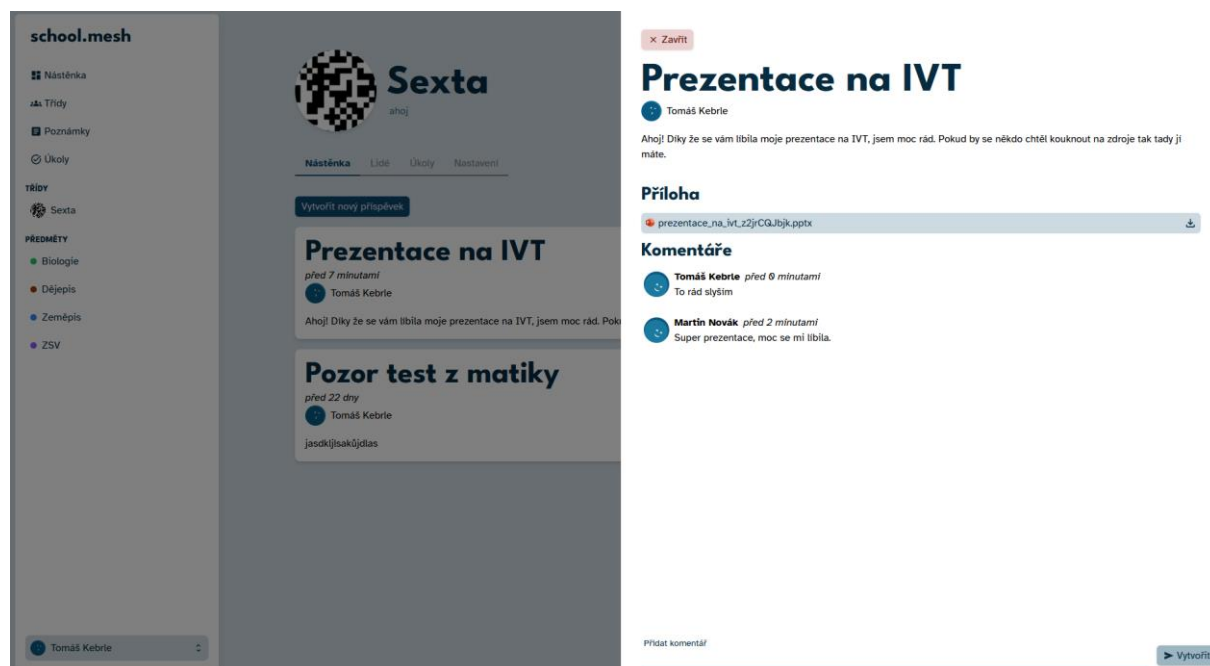
3.6 Třídy

Pro část aplikace, která funguje jako sociální síť jsem přidal funkci vytváření tříd. Každý student může vytvořit svojí vlastní třídu, nebo se připojit do již existující třídy. Třídy mají dvě hlavní funkce, diskusní, kde pro každou třídu je vytvořena nástěnka, na kterou může každý člen něco napsat a socializační která spočívá v tom, že každý uživatel si může vytvořit svůj osobní profil, který potom můžou ostatní členové třídy vidět.

3.6.1 Nástěnka třídy

Nástěnka třídy funguje na principu příspěvků, každý člen třídy může vytvořit svůj vlastní příspěvek, stačí zadat nápis, přidat svojí zprávu a pokud autor příspěvku chce tak může přidat i soubory. Tyto soubory jsou poté uloženy do S3 objektového úložiště na Digital

Ocean. Pod každým příspěvkem jsou také komentáře, které fungují formou chatu, tím pádem vše probíhá v reálném čase.



Obrázek 13: Ukázka zobrazení příspěvku s diskusí

Implementace komentářů v reálném čase je udělána pomocí Pocketbase, nejdříve musím backendu říct, že čekáme od něj zprávy o změnách, tzv. „Server sent events“. Poté vytvoříme funkci, která bude spuštěna pokaždé když server vyšle zprávu o novém komentáři.

```
1 pb.collection("comments").subscribe<CommentsResponse>("*", (e) => {
2   if (e.record.post !== post.id) return;
3
4   comments = [...comments, e.record];
5 });
```

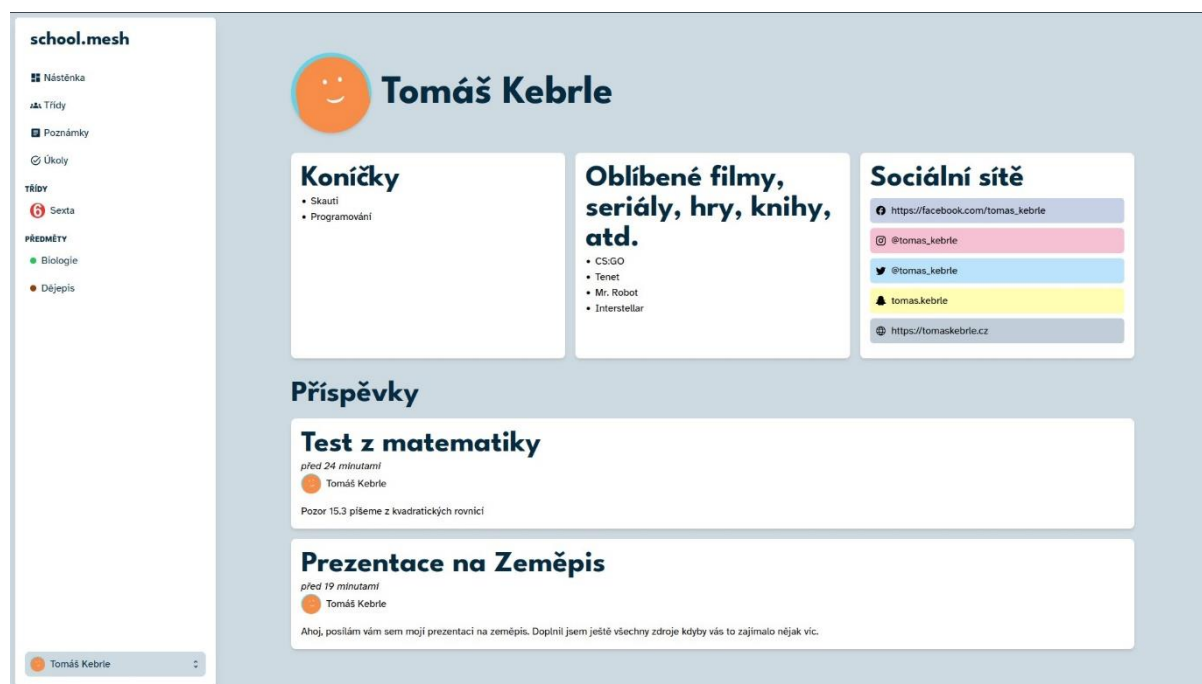
Ukázka kódu 10: Implementace komentářů v reálném čase

Jelikož nejde pomocí Pocketbase poslouchat změny jenom na určitém typu kolekcí musíme poslouchat všechny změny v kolekci comments, respektive všechny změny, ke kterým máme přístup, což znamená že když někdo přidá komentář na příspěvek ve třídě kde nejsme tak se o tom nedozvíme, ale kvůli této limitaci se může stát že někdo přidá komentář na jiný příspěvek než na kterém právě jsme proto v druhém řádku porovnáváme jestli je nový komentář určen pro příspěvek na kterém právě jsme. Ve 4. řádku poté jenom přidáváme nový komentář do

listu, Svelte poté automaticky přidá nový komentář na obrazovku. Nový komentář přidáváme do listu pomocí tzv. „spread operátoru“, protože při běžnějším stylu přidávání do listu pomocí metody `.push()` (MDN, 2023), Svelte nemůže detekovat změnu a tím pádem neví že se naše data změnily.

3.6.2 Osobní profily

V každé třídě jde také vidět list všech členů. Kliknutím na jednoho člena se dá dostat na jeho osobní profil. Do svého osobního profilu si každý uživatel může přidat informace o tom jaké má koníčky, oblíbené filmy, seriály atd. a odkazy na jiné sociální sítě. Také se na jeho profilu zobrazují příspěvky, které vytvořil, uživatel, který si ale jeho profil prohlíží, uvidí pouze příspěvky vytvořené ve třídách, které má s vlastníkem profilu společné.



Obrázek 14: Ukázka osobního profilu

3.6.3 Připojení do třídy

Jelikož připojením do třídy se automaticky zpřístupní informace o všech členech třídy, je důležité, aby se do třídy mohli připojit jen ty lidi který jsme pozvali. To je vyřešené vytvářením speciálních odkazů na připojení. Po vytvoření třídy je vždycky vytvořen záznam v kolekci `join`, který má svoje vlastní `id` a uchovává informaci o tom pro kterou třídu je určen.

```

1 app.OnRecordAfterCreateRequest()
2 .Add(func(e *core.RecordCreateEvent) error {
3     if e.Record.Collection().Name == "groups" {
4         collection, err := app.Dao()
5             .FindCollectionByNameOrId("join")
6
7         if err != nil {
8             return err
9         }
10
11         expireRecord := models.NewRecord(collection)
12         form := forms.NewRecordUpsert(app, expireRecord)
13
14         form.LoadData(map[string]any{
15             "group": e.Record.Id,
16
17             if err := form.Submit(); err != nil {
18                 return err
19             }
20
21         })
22
23         return nil
24 })

```

Ukázka kódu 11: Vytváření join záznamu před vytvořením třídy

Administrátor třídy neboli ten, kdo jí vytvořil poté v nastavení může zkopírovat odkaz a poslat ho spolužákům, kteří se po jeho rozkliknutí můžou připojit. Poté co se připojí všichni spolužáci stačí odkaz smazat, tím se zablokuje přístup komukoliv novému k připojení do třídy.

3.7 Lokalizace

Aby aplikace mohla být využívána co největším publikem je důležité, aby byla lokalizována neboli přeložena do více jazyků. V průběhu vývoje aplikace jsem vyzkoušel 2 možnosti, jak lokalizovat aplikaci. První byla pomocí knihovny `svelte-i18n`, od té jsem ale odstoupil poté co jsem předělal projekt na monorepozitář, od té doby používám knihovnu `typesafe-i18n`, její hlavní výhodou, jak už název napovídá je typová bezpečnost, tím pádem si můžu

být jistý, že překlady, které chceme použít existují, navíc nabízí také spoustu dalších funkcí, jako automatické detekování jazyka podle nastavení počítače. Dále má tato knihovna skvělé příkladové projekty pro SvelteKit, takže přidání knihovny bylo docela jednoduché. (Hofer, 2023)

4 ZÁVĚR

Cílem mé práce bylo vytvořit aplikaci, která by usnadňovala všechny aspekty školy, a věřím tomu, že se mi to alespoň z části povedlo. Hlavní využití mé aplikace je již popsáno v samotné práci, ale dále by aplikace mohla být vhodná třeba pro studenty kteří nastoupí do nové školy a tím pádem i do nové třídy, kde by poté mohli využít osobních profilů, aby se lépe poznali.

Aplikaci jsem již představil třem třídám u nás na škole, ale zájem nebyl moc veliký. Myslím si, že je to způsobeno obtížností zaujmout studenty aplikací určenou pro školu. Dostal jsem ale zpětnou vazbu v podobě toho, že by aplikace mohla být užitečnější pro učitele.

4.1 Budoucí plány

Líbila se mi zpětná vazba o tom, že by aplikace byla možná lepší pro učitele, a určitě bych chtěl v budoucnosti tuhle myšlenku dále rozvíjet, třeba navázáním spolupráce s některými učiteli a vytvořením specifických funkcí pro učitele, které by jim mohli usnadnit práci.

Dále bych chtěl vytvořit mobilní aplikaci. Webová verze sice jde používat i na telefonu, ale vzhled aplikace není 100 % přizpůsoben mobilům, a tak by dedikovaná mobilní aplikace byla mnohem lepší. Třeba by to dokonce pomohlo s adopcí aplikace a byla by více používaná, vzhledem k tomu že sociální sítě jsou dnes primárně využívány na telefonech.

4.2 Zkušenosti

Prvotní nápad na aplikaci jsem měl již v roce 2021, tehdy jsem uměl maximálně základy HTML a CSS, a trochu Pythonu, ale rozhodně jsem neměl dostatek znalostí na to abych vytvořil celou aplikaci. Od toho momentu jsem aplikaci zkoušel udělat třikrát, u prvních dvou pokusů jsem ale nevydržel déle jak 2 měsíce. Při vytváření této aplikace jsem se naučil, jak se pracuje se systémem Git, jak se využívá SvelteKit, pro backend jsem vyzkoušel Firebase, Supabase až nakonec jsem se rozhodl využít Pocketbase. Také jsem se naučil programovat v Javascriptu, Typescriptu i v GO.

Nakonec trochu statistiky finální verze má přes 560 commitů ve své Git historii, více jak 5700 řádků kódu a přes 300 souborů. Projekt je z 58 % napsaný ve Svelte, 40 % je v Typescriptu, a zbylé 2 % jsou různé konfigurační soubory.

5 LITERATURA

Auth 0, 2022. *JSON Web Token Introduction* - *jwt.io*. [Online]

Available at: <https://jwt.io/>

[Přístup získán 09 04 2023].

Binesh Sarwar, S. Z. S. A. E. C., 2019. Usage of Social Media. *Journal of Educational Computing*, Svazek 57(1), p. 246–279.

Docker, 2023. *Docker overview*. [Online]

Available at: <https://docs.docker.com/get-started/overview/>

[Přístup získán 09 04 2023].

Eger, L., 2004. Blended Learning. *Aula*, Svazek 03, pp. 21-24.

Harris, R., 2018. *Virtual DOM is pure overhead*. [Online]

Available at: <https://svelte.dev/blog/virtual-dom-is-pure-overhead>

[Přístup získán 09 04 2023].

Harris, R., 2019. *Write less code*. [Online]

Available at: <https://svelte.dev/blog/write-less-code>

[Přístup získán 09 04 2023].

Hofer, I., 2023. *Typesafe i18n*. [Online]

Available at: <https://github.com/ivanhofer/typesafe-i18n>

[Přístup získán 09 04 2023].

Kornell, N., 2009. Optimising learning using flashcards: Spacing is more effective than cramming. *Applied Cognitive Psychology*, Svazek 23, pp. 1297-1317.

MDN, 2023. *MDN*. [Online]

Available at: <https://developer.mozilla.org/en-US/>

[Přístup získán 09 04 2023].

OpenAI, 2023. *What are tokens and how to count them*. [Online]

Available at: <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count->

them

[Přístup získán 09 04 2023].

Svelte, 2023. *SvelteKit - Web development, streamlined*. [Online]

Available at: <https://kit.svelte.dev/>

[Přístup získán 09 04 2023].

Typescript, 2023. *TypeScript is JavaScript with syntax for types*. [Online]

Available at: <https://www.typescriptlang.org/>

[Přístup získán 07 04 2023].

Wikipedia, 2023. *Wikipedia*. [Online]

Available at: <https://en.wikipedia.org/wiki/Svelte>

[Přístup získán 09 04 2023].