DELTA - Střední škola informatiky a ekonomie, s.r.o.
Ke Kamenci 151, Pardubice

Webová aplikace pro rodinné úkoly

Adam Pečenka
4.A
Informační technologie (18-20-M/01)

2023/2024

Poděkování

Chtěl bych poděkovat Ing. Monice Borkovcové, Ph.D. za vedení tohoto maturitního projektu a konzultaci problémů. Dále bych chtěl poděkovat Jakubovi Hamplovi za pomoc se složením použitých technologií, Nele Bartákové za pomoc se vzhledem aplikace a Martině Kozlové s Nelou Halfarovou za cennou zpětnou vazbu v rozličných oblastech.

Anotace

Cílem projektu je tvorba aplikace pro zadávání a plnění úkolů s webovým rozhraním. Systém zahrnuje uživatele, kteří jsou rozděleni do skupin, které si sami zakládají, v rámci skupin si mohou sami zadat úkol sami sobě nebo jinému uživateli ve skupině. Dalším požadavkem je, aby úkoly bylo možné dělit do kategorií, které vytváří uživatelé ve skupině. Skupinu může upravit pouze její zakladatel, úkol a kategorii pouze její autor a zakladatel rodičovské skupiny. Součástí aplikace je i práce s kalendářem, kdy je možné zadat úkoly na různá období.

Klíčová slova

Next.js, Typescript, Prisma, tRPC, PostgreSQL, Webová aplikace, Úkoly

Annotation

The aim of the project is to create an application for task management with a web interface. The system includes users who are divided into groups that they create themselves. Within these groups, users can assign tasks to themselves or to other users in the group. Another requirement is that tasks can be categorized, which users within the group create. Only the founder of a group can modify it, and only the author and the founder of the parent group can modify a task and its category. The application also features calendar integration, allowing tasks to be scheduled for different time periods.

Keywords

Next.js, Typescript, Prisma, tRPC, PostgreSQL, Web Application, Tasks

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení:	*Adam Pečenka*
Školní rok:	*2022/2023*
Třída:	*3.A*
Obor:	*Informační technologie 18-20-M/01*
Téma práce:	*Webová aplikace pro rodinné úkoly*
Vedoucí práce:	*Ing. Monika Borkovcová, Ph.D.*

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cílem projektu je vytvoření webové aplikace, která bude umožňovat zadávat jednotlivým členům rodiny či jiné skupině osob různé úkoly.

Popis funkčnosti webové aplikace, která umožňuje:

- založit skupinu a členy skupiny (každý člen vidí jen svoji skupinu a členy),
- zobrazit kalendář a zadávat jednotlivé úkoly buď na celý den nebo více dní, nebo jen s časovým omezením (například oslava narozenin, návštěva lékaře apod.),
- každý úkol bude mít svůj název a popis a bude přiřazen konkrétnímu časovému období
- každý člen může zadat splnění vlastního úkolu, který mu může potvrdit jiný člen skupiny
- u každého člena je evidováno, jak plní jednotlivé úkoly formou jednoduchého procentuálního vyjádření
- každý člen vidí přehled, které úkoly má splněné a které ještě nikoliv nebo které jsou v budoucnu
- úkoly mohou být zařazeny do různých kategorií, kategorie může spravovat každý člen skupiny, ale měnit již stávající kategorie může pouze ten, který sám kategorii vytvořil
- každý člen skupiny může zadávat i svá přání, která mu mohou plnit ostatní členové skupiny, každé takové přání lze převzít a splnit, pokud člen skupiny přání převezme je a splní, vlastník přání jej následně může potvrdit jako splněné.

Stručný časový harmonogram (s daty a konkretizovanými úkoly):

Září: Analýza současných obdobných webových aplikací a návrh architektury vlastní aplikace

Říjen-leden: Práce na webové aplikaci

Leden: Zahájení zpracování dokumentace aplikace a teoretické části práce

Únor-Březen: Testování, ladění chyb, dokončení dokumentace a teoretické části práce

Prohlašuji, že jsem maturitní projekt vypracoval samostatně, výhradně s použitím uvedené literatury.

V Pardubicích, 4.2.2024

Adam Pečenka

Obsah

1. Úvod

V současné době je k dispozici velké množství TODO aplikací. Tyto aplikace slouží k správě seznamů úkolů a organizaci pracovních, osobních nebo jiných aktivit. Jedná se o běžně používané softwarové aplikace k usnadnění plánování, sledování a dokončení úkolů. TODO aplikace jsou převážně dostupné ve formě mobilních nebo webových aplikací.
Základní funkce, které takové aplikace obvykle obsahují, zahrnují správu úkolů, stanovení termínů a upomínek k úkolům, možnosti kategorizace a štítkování, sdílení seznamů úkolů a například i integraci s dalšími aplikacemi. Často bývá součástí TODO aplikace i funkcionalita "štítkování", která v některých aplikacích může nahrazovat kategorizaci nebo ji může doplňovat.
TODO aplikace cílí buď na jednotlivce nebo na skupiny. Problém nastává, jakmile má uživatel zájem o zapisování vlastních úkolů a zároveň úkolování skupiny, ať už na úrovni pracovní nebo osobní. V tomto případě je nucen často používat dvě oddělené aplikace. Aplikace na trhu také bývají velmi zjednodušené nebo naopak příliš komplexní, což práci s takovou aplikací může paradoxně až znevýhodnit. Komplexnější aplikace bývají také placené nebo poskytují jen velmi omezenou dostupnost funkcionalit v základní neplacené verzi.

Cílem maturitní práce je vytvořit intuitivní TODO aplikaci s webovým rozhraním, která umožňuje rychlou správu úkolů, přání a jejich dělení do různých kategorií. Uživatelé aplikace se vzájemně přidávají do skupin, ve kterých si mohou úkoly či přání zadávat. Úkoly mohou mít časová omezení a uživatelé, kteří plní tyto úkoly, je vidí ve vestavěné funkcionalitě kalendáře.
Tato práce je rozdělena do dvou základních částí. První část se soustředí na vysvětlení základních pojmů a podobné aplikace, ty popisuje kapitola 2. Následující kapitola pak seznamuje se "základním technologickým stackem". V práci bude použit Typescript, React a Next.js, tRPC, Prisma, NextAuth, TailwindCSS, Prettier a další.
Druhá část se již prakticky věnuje samotnému vývoji webové aplikace. Popis tvorby zahrnující jednotlivé kroky vývoje webové aplikace od analýzy až po implementaci je popsán v kapitole 4 Aplikace.

2. Porovnání s ostatními aplikacemi na trhu

V rámci teoretické části práce jsem se před psaním vlastní "TODO" aplikace rozhodl porovnat "TODO" aplikace, které již byly uvedeny na trh. Průzkum probíhal v týdnu 15.09.2023 - 22.09.2023. Na trhu se již vyskytuje několik velkých hráčů na trhu takzvaných "TODO"[1] aplikací. Níže jsou popsané nejčastěji používané tzv. "TODO"[2] aplikace na trhu v ČR.

2.1 Microsoft TODO

MS TODO aplikace je jednou z hojně používaných řešení pro jednotlivce. Je velmi jednoduchá na ovládání, ale nedisponuje funkcionalitou správy skupin nebo kalendářem. Uživatel si může rozdělit úkoly do jednotlivých kategorií a nastavit si upozornění na jejich dokončení. Při práci s Microsoft TODO uživatel pracuje v off-line režimu (.exe), což může být pro některé výhodou. Dále aplikace disponuje určitou formou propojení s Microsoft Office 365 a synchronizaci v cloudu.

2.2 Trello

Trello cílí především na práci v týmu. Je komplexnější než MS TODO a disponuje kalendářem. Trello disponuje prací se skupinami, respektive tabulemi, do kterých může majitel přidávat ostatní uživatele. Uživatelé si mohou přiřadit úkoly sami nebo je může přiřadit majitel. V aplikaci také lze zapnout sledování jednotlivých úkolů a Trello je poté informuje upozorněními na stránce.

2.3 Google Keep

Google Keep je zaměřen na osobní řešení. Uživatel má možnost provádět základní CRUD[3] operace na tzv. poznámkách a štítcích. Google Keep nemá funkci kalendáře či skupin, ale místo ní můžeme přidávat jako "spolupracovníky" ostatní osoby s Google účtem. I přesto, že nemá aplikace kalendář tak si uživatelé mohou zapnout upozornění na jednotlivé poznámky.

[1] Úkolníky
[2] Úkolníky
[3] CRUD - Create Retrieve Update Delete - Tvořit Číst Aktualizovat Mazat

2.4 Freelo

Freelo je z porovnávaných aplikací ta nejkomplexnější, ale také jediná, která není pro náročnější uživatele zdarma a uzavírá některé funkce za platební brány. Freelo nabízí 5 tarifů; tarify jsou seřazeny dle jejich měsíční ceny, Freelo nabízí 10% slevu na každý měsíc, po volbě předplatného na rok dopředu, ceny jsou uvedeny bez DPH[1].

1. Free
    - Zdarma
    - Maximálně 3 projekty
    - Maximálně 4 uživatelé, včetně majitele
    - Různé pohledy na úkoly, včetně kalendáře
    - Maximálně 500 MB souborů
2. Freelance
    - 990 Kč za měsíc
    - Přidaný pohled na úkoly pomocí "Mind mapy"
    - Stejné funkce, co tarif "Free", nicméně bez omezení
3. Team
    - 2 190 Kč za měsíc
    - Role "Správce" - umožňuje delegovat spravování projektu na jiné uživatele než majitele
    - Role "Projekťák" - umožňuje delegovat zakládání projektů na jiné uživatele než majitele
    - Vazby mezi úkoly napříč projekty [4]
4. Business
    - Cena se odvíjí dle počtu uživatelů
        - Do 5 uživatelů: 1 690 Kč za měsíc
        - Do 10 uživatelů: 2 690 Kč za měsíc
        - Do 15 uživatelů: 3 590 Kč za měsíc
        - Do 20 uživatelů: 4 180 Kč za měsíc

[4] Například úkol X navažuje na úkol Y apod.

    - Nad 20 uživatelů: 4 180 Kč + 209 Kč za 21. uživatele a dál[5]
  - Vše, co předchozí tarify
  - Vlastní role
  - Rozšíření vazeb
  - Přehledy
  - Rozšířené zabezpečení
  - Ganttův diagram
5. Enterprise
  - Od 45 900 Kč za měsíc
  - Vše, co předchozí tarify
  - Integrace na míru
  - Osobní komunikace s Freelem

[5]Za každých dalších 20 uživatelů se snižuje poplatek za nadbytečné uživatele až na 160 Kč za uživatele

3. Technologie

3.1 Licence

Jednou z prvních věcí, kterou je vhodné zvážit ještě než na projektu začneme pracovat, je licence pod kterou projekt bude zpracován. Pro projekt byla zvolena MIT Licence, také známá jako Expat nebo X11[2]. MIT Licence vznikla na Massachusettském technologickém institutu roku 1988 a umožňuje ostatním uživatelům software jakkoliv upravovat a vydávat nové verze programu, i pod jinou licencí, pokud uvedou autora a ponechají původní kód pod MIT Licencí[3].

3.2 Stack

V dnešní době je prakticky neomezeně různých způsobů, jak si sestavit technologický stack[1] pro tvorbu full-stackové[2] webové aplikace. Pro tvorbu jsem zvolil T3 Stack, T3 Stack skládá dohromady různé poměrně populární technologie na tvorbu full-stack aplikací, ale zároveň při tvoření projektu, za použití `create-t3-app` si umožňuje Stack jakkoliv upravit[4].

3.2.1 Typescript

T3 Stack obsahuje Typescript[3], to je jedna z jeho hlavních předností. Typescript je nad stavbou nad jazykem Javascript[4]. Typescript se od JS liší, jak již název napovídá, přidáním typů, byť se může zdát, že se jedná pouze o "syntaktický cukr"[5], tak je TS, na rozdíl od JS, který je interpretovaný jazyk, kompilovaný jazyk. Typescript umožňuje definovat typy proměnných, funkcí a dalších objektů, což umožňuje efektivní ladění kódu a typové kontroly.

3.2.1.1 Interpretované a Kompilované jazyky

Interpretované jazyky jsou spouštěny postupným čtením kódu od 0. řádku až na konec programu, to má za následek, že některé chyby jdou zachytit až po spuštění samotného programu. Kompilované jazyky před spuštěním projdou kompilací a mohou tak zachytit všechny chyby, které by mohly nastat.

[1] Stack je označení pro technologie, které jsou na projektu použity

[2] Termín full-stack označuje aplikace, které se skládají jak z klientské části, tedy "Frontend", a ze serverové části, tedy "Backend". Backend a Frontend spolu komunikují za pomoci API.

[3] Zkráceně TS

[4] Zkráceně JS

[5] Slangový termín pro nadbytečné funkce programovacích jazyků

Proces kompilace dovoluje Typescriptu zachytávat chyby, které jsou spojeny s daty co by byly typu `undefined` nebo `null`. Typ `undefined` se v kódu vyskytne hlavně při čekání na data z databáze, typ `null` když nám databáze nic nevrátí. Typescript je kompilovaný jazyk poměrně netradičním způsobem, protože TS se kompiluje na JS a ne na machine code[6].

3.2.2 React a Next.js

Celá uživatelská část aplikace je psaná pomocí populárního frameworku Next.js, který je zároveň nadstavba snad ještě populárnějšího frameworku React. T3 Stack obsahuje Next.js, který poskytuje velmi příjemné funkce, které ulehčují vývoj aplikací, například automatickou optimalizaci routování nebo cachování[5].

3.2.3 tRPC

Pro řešení API, tedy komunikačního kanálu mezi uživatelskou a serverovou částí, T3 Stack volí tRPC. tRPC je elegantní způsob jak psát endpointy[7]. Pro typovou validaci dat volí T3 stack knihovnu Zod[6].

```
// Nerelevantní části byly nahrazeny *pseudokódem*
deleteById: protectedProcedure
    .input(z.object({ id: z.string() }))
    .mutation(async ({ ctx, input }) => {
      //*Dostaneme z databáze skupinu s identifikátorem o hodnotě id a uložíme jí do proměnné group*
      //Pokud je group o typu null nebo undefined vrátíme error
       throw new TRPCError({ code: "NOT_FOUND", message: "Group not found " });
      //Pokud je id majitele skupiny jiné, než id přihlášeného uživatele vrátíme error
      if (group.ownerId !== ctx.session.user.id)
        throw new TRPCError({
          code: "UNAUTHORIZED",
          message: "Must be owner of the group",
        });
      //*Smažeme skupinu v databázi*
    }),
```

Ukázka kódu 3.1 Úryvek z "groups" routeru zobrazující mazání skupiny

Kapitola 4.2 obsahuje podrobnější popis celé API.

[6]Kód, který již dokáže přečíst počítač.
[7]Endpoint = část API, se kterou se dá komunikovat "z venčí"

3.2.4 Prisma a Databáze

Datová vrstva je v projektu řešená pomocí PostgreSQL, která se řadí mezi relační databáze.

Pro připojení na databázi volí T3 Stack databázové ORM Prisma. Prisma je páteřní technologie, která umožňuje sdílení typů mezi serverem a klientskou částí aplikace. Další bezpochybnou výhodou Prismy je její bezpečnost, psaní čistých SQL příkazů do aplikace může vést někdy až ke kritickým chybám či ztrátě dat. Prisma ve vývojovém prostředí poskytuje našeptávání přímo z databázového schématu[8] a tak je mnohem těžší udělat chyby. Pro design databázového schématu používá Prisma vlastní jazyk.

```
model GroupMembership {
  id      String @id @default(cuid())
  userId  String
  user    User   @relation(fields: [userId], references: [id], onDelete:
    ↪ Cascade)
  groupId String
  group   Group  @relation(fields: [groupId], references: [id], onDelete:
    ↪  Cascade)

  @@unique([userId, groupId])
  @@index([userId])
  @@index([groupId])
}
```

Ukázka kódu 3.2 Úryvek z Databázové schématu zobrazující strukturu tabulky pro členství ve skupině

Kapitola 4.1 obsahuje podrobnější dokumentaci databázového schématu.

3.2.5 NextAuth

Autentikaci a autorizaci uživatelů T3 Stack řeší za pomocí NextAuth. NextAuth nám umožňuje, místo standardního přihlašování formou email-heslo, použít takzvané *Providery*[9]. To má mnoho výhod, například, nemusíme ukládat uživatelovo heslo a přihlašování pro uživatele je jednodušší, postačí kliknout na jedno tlačítko a prohlížeč provede automatické přihlášení, za předpokladu, že už na něj uživatel je přihlášený. Pokud přihlášený není, bude přesměrován na přihlašovací stránku providera. NextAuth nabízí přes 30 Providerů[7], já se rozhodl pro použití přihlášení od společnosti Google, vzhledem k rozšířenosti Google účtů ve společnosti. Jednou z nevýhod tohoto řešení je, že pokud u Providera dojde k úniku dat budou v ohrožení i data uživatelů.

[8] Databázové schéma určuje strukturu databáze.
[9] Provider = Poskytovatel přihlášení

3.2.6 TailwindCSS, PostCSS a NextUI

Na stylování projektu používá T3 Stack TailwindCSS. Tailwind CSS je moderní CSS framework, který umožňuje vytvářet responzivní uživatelská rozhraní pomocí předdefinovaných tříd. Tailwind CSS nabízí rychlou a flexibilní alternativu k psaní vanilla[10] CSS.

```
1 //CSS soubor:
2 .vanilla {
3   margin-top: 12px,
4   margin-bottom: 12px,
5   padding-left: 12px,
6   padding-right: 12px
7 }
8 //HTML soubor:
9 <div class="vanilla"></div>
```

Ukázka kódu 3.3 Ukázka stylování HTML prvku za použití vanilla CSS[8]

```
1     <div class="mx-3 py-3"></div>
```

Ukázka kódu 3.4 Ukázka stylování HTML prvku za použití TailwindCSS[8]

Než budeme moct Tailwind vůbec použít musíme vytvořit konfigurační soubor `tailwind.config.ts`.

Tailwind má sám o sobě velikost přibližně `3645 kB` to se může zdát v porovnání s dnešními velikostmi souborů jako málo, ale na webu se jedná o velký soubor. Proto jsem se rozhodl použít technologii PostCSS, které při vystavení projektu do produkce vymaže nepoužitá stylovací pravidla z Tailwindu a sníží tím velikost souboru. PostCSS lze definovat jako nástroj pro transformaci CSS s pomocí Javascriptu umožňující provádět optimalizaci kódu. PostCSS se nastavuje v konfiguračním souboru `postcss.config.jcs`.

Tailwind nám poskytuje velkou kontrolu nad naším stylováním, ale na rozdíl od CSS frameworku Bootstrap neposkytuje předem postavené komponenty, abych tedy urychlil vývoj aplikace, rozhodl jsem se použít předem stavěné komponenty za použití Tailwindu od NextUI[9], jmenovitě jsem použil především tabulky.

3.2.7 Prettier

Správné formátování kódu a vyhýbání se špatným vývojovým praktikám je důležitá část jakéhokoliv projektu. Na formátování používá T3 Stack Prettier, ten, v kombinaci s přídavným balíčkem do VS Code, upravuje formátování při uložení souboru. Dobře naformáto-

[10] Vanilla = Označení pro neupravenou verzi technologie

vaný kód ulehčuje čitelnost a znovuvyužitelnost. Prettier se nastavuje v konfiguračním souboru `prettier.config.mjs`.

3.2.8 EsLint

EsLint se používá na varování před praktikami, které by mohly vést k chybě a rozšiřuje tak "varovný systém", který už samotný Typescript poskytuje. EsLint se nastavuje v konfiguračním souboru `.eslintrc.cjs`. Jedny z pravidel, které se v aplikaci používají jsou například `no-unused-vars` nebo `no-misused-promises`.

Eslint má v konfiguraci 3 následující možnosti hlášení chyb[10].

- `off`
  - Používá se na přepsání základních pravidel EsLintu
- `warn`
  - Vytvoří varování, které nikterak nebrání kompilaci kódu
- `error`
  - Zabrání kompilaci kódu a používá se pouze na nejzávažnější chyby

3.3 Dodatečné technlogie

Projekt je spouštěn pomocí node.js v kombinaci s NPM[11]. Celý kód je verzován verzovacím systémem Git a psán v textovém editoru Visual Studio Code (VS Code).

3.3.1 package.json

Tento konfigurační soubor je často přehlížen, ale jedná se o stěžejní konfigurační soubor celého projektu. V `package.json` jsou definovaný takzvané "závislosti" projektu. `Package.json` funguje jako nákupní seznam, kde jsou právě tyto závislost napsané. Po naklonování projektu je nutné spustit příkaz `npm i`, ten vezme `package.json` a z platformy NPM[11] stáhne všechny balíčky, na kterých projekt závisí a automaticky je nainstaluje do složky `node_module`.

[11] Node Package Manager - správce balíčků pro Javascript standardně používán v ekosystému Node.js

3.4 Konfigurace

Ještě než se podaří spustit projekt, je dalším krokem vytvoření konfiguračního souboru `.env`[12]. Ten z bezpečnostních důvodů není obsažen v repozitáři, protože obsahuje velmi citlivé informace (přístupové údaje k databázi nebo klíče k autentikaci uživatel). Vzhledem k tomu, že `.env` soubor musí mít pro správné fungování aplikace jistou strukturu, je v repozitáři šablona `.env.example`.

```
...
# Prisma
...
DATABASE_URL=""
DATABASE_NON_POOLING_URL=""

# Next Auth
...
NEXTAUTH_SECRET=""
NEXTAUTH_URL=""

# Next Auth Providers
GOOGLE_CLIENT_ID=""
GOOGLE_CLIENT_SECRET=""

# Custom Image API
...
IMAGE_API=""
```

Ukázka kódu 3.5 Šablona konfiguračního souboru .env

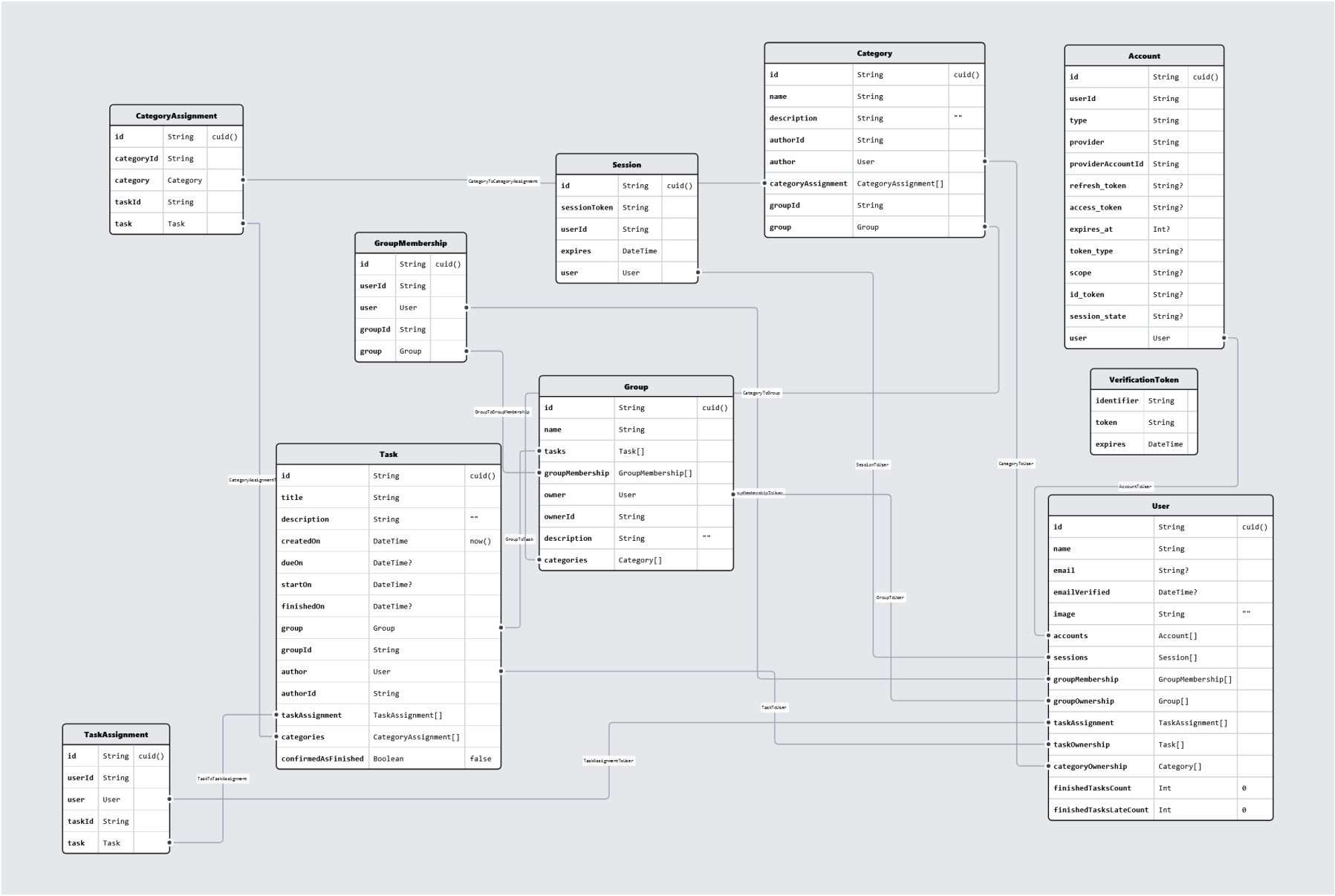
[12]Z anglického "environment", tedy "prostředí"

4. Aplikace

Vzhledem ke komplexnosti aplikace je pro přehlednost rozdělena do několika složek. Podobným způsobem byly rozděleny Class diagramy projektu, v Class diagramech jsou popsány veškeré podpůrné funkce, které projekt používá, stránky, komponenty a API. V Class diagramech je například popsaná podpůrná funkce `roundToPreviousHour`, stránka `TaskCalendarPage` nebo komponenta `GroupCreate`. Class diagramy se nachází v kapitole 6.4 v Příloze.

4.1 Prisma

Databázové schéma v jazyku Prisma a databázové migrace se nachází ve složce `/prisma/`. Po úpravě databázové schématu je vhodné i vytvořit databázovou migraci, ta umožní manuálně spustit SQL příkazy, které Prisma vykonala, pokud to bude potřeba. Díky použití Prismy lze jak na API tak na frontendové části aplikace používat identické datové typy, nemusíme, především na Frontendu, definovat vlastní "Interface" pro každý objekt v databázi, který bychom museli pokaždé aktualizovat, když se změní databázové schéma. Následuje vizualizace databázové schématu vygenerovaná pomocí webové aplikace Prismaliser [12].



Obrázek 4.1: Databázové schéma

4.2 API

Celé API je psané za pomocí tRPC, to se nachází ve složce `/src/server/api/`. API je inicializováno v souborech `trpc.ts` a `root.ts`. Konkretně v souboru `trpc.ts` inicializujeme objekt tRPC a v `root.ts` se na něj napojují jednotlivé Routery. Samotné "endpointy", tedy přístupy, jsou rozděleny do "routerů", ty se nachází ve složce `/src/server/api/routers` a dělí se na jednotlivé "hlavní objekty" v databázi. Tyto routery jsou děleny na soubory `categories.ts`, `groups.ts`, `tasks.ts` a `users.ts`. V kódu se "endpointy" označují jako "procedury", v princpu existují dva druhy procedur; public[1] a protected[2]. Jediným rozdílem mezi nimi je, že protected procedura zaručuje, že ji může úspěšně zavolat pouze přihlášený uživatel. Následuje úryvek z routeru pro skupiny v souboru `groups.ts`, ostatní procedury byly z ukázky odstraněny.

```
1  import { z } from "zod";
2
3  import {
4    createTRPCRouter,
5    protectedProcedure,
6    publicProcedure,
7  } from "~/server/api/trpc";
8
9  export const groupsRouter = createTRPCRouter({
10 ...
11 addMember: protectedProcedure
12     .input(z.object({ userId: z.string(), groupId: z.string() }))
13     .mutation(async ({ ctx, input }) => {
14       const group = await ctx.prisma.group.findFirst({
15         where: { id: input.groupId },
16       });
17       if (!group)
18         throw new TRPCError({ code: "NOT_FOUND", message: "Group not
     ↪ found" });
19       const user = await ctx.prisma.user.findFirst({
20         where: { id: input.userId },
21       });
22       if (!user)
23         throw new TRPCError({ code: "NOT_FOUND", message: "User not found
     ↪ " });
24       if (group.ownerId !== ctx.session.user.id)
25         throw new TRPCError({
26           code: "UNAUTHORIZED",
27           message: "Must be owner of the group",
28         });
29       const membership = await ctx.prisma.groupMembership.create({
```

[1] Veřejná
[2] Chráněná

```
30          data: {
31            groupId: input.groupId,
32            userId: input.userId,
33          },
34        });
35        if (membership) return true;
36        return false;
37      }),
38  })
```

Ukázka kódu 4.1 Procedura na přidání uživatele do skupiny

Jako první je nutné definovat název Procedury, v tomto případě "addMember", tedy "přidatUživatele". Dále přidáme `.input()` zde bude procedura přijímat data pro své provedení. Pokud bude Procedura data měnit tak přidáme `.mutation`, pokud data pouze předá z databáze, tak přidáme `.query`. Do nich poté předáme vstup z `.input` a `ctx`[3]. Kontext je globálně zpracovávaný a nachází se v něm `session` přihlášeného uživatele a `prisma`, tou se napojujeme na databázi.

[3] Kontext

4.2.1 Příklad API Endpointů

Následující tabulka zobrazuje vybrané API Endpointy, z každého routeru, který se v aplikaci nachází dva Endpointy. Tyto Endpointy byly vybrány vzhledem k jejich četnosti výskytu v aplikaci, ale také vzhledem k jejich rozmanitosti přijímaných a vracených dat.

Router	**Název**	**Popis**	**Vstup**	**Výstup**
categories.ts	create	Vytvoří kategorii	Název, Popis, ID autora, ID skupiny	Category
categories.ts	getTasks	Vrátí úkoly s kategorií	ID kategorie	Task[]
tasks.ts	assignUser	Přiřadí uživatele k úkolu	ID uživatele, ID úkolu	True / False
tasks.ts	getCategories	Vrátí pole kategorií úkolu	ID úkolu	Category[]
groups.ts	getById	Vrátí objekt skupiny, pokud je přihlášený uživatel člen	ID skupiny	Group
groups.ts	leave	Odstraní přihlášeného uživatele ze skupiny	ID skupiny	True / False
users.ts	locateByName	Vrátí pole uživatelů s podobným jménem k vstupu	Text	User[]
users.ts	editName	Upraví jméno uživatele	ID uživatele, Nové jméno	User

Tabulka 4.1: Příklad API Endpointů

4.3 Next.js

Kód, ze kterého se staví frontendová část aplikace pomocí Next.js se nachází ve složkách `src/pages` a `src/components`. V `pages` se nachází webové stránky, které vidí uživatel, v `components` jsou uloženy komponenty, ze kterých se staví jednotlivé stránky. Obě složky jsou rozděleny do podsložek podle jednotlivých objektů, kterých se týkají, například v `src/components/groups` se nachází pouze komponenty, které se týkají skupin.

4.3.1 src/components/

Složka je rozdělena na následujících podsložek:

```
components
├── auth //komponenty související s autorizací uživatelů
├── categories //komponenty související s kategoriemi
├── groups //komponenty související se skupinami
├── layout //komponenty související s rozvržením stránek
├── tasks //komponenty související s úkoly
└── users //komponenty související s uživateli
```

Jednotlivé podsložky jsou následně rozděleny na komponenty, které mají každá svou vlastní složku. Ve složce pro komponentu se nachází soubor `index.tsx` a `názevKomponenty.tsx`.
Finálně tedy struktura adresáře vypadá následovně:

```
/
└── src
    ├── components
    │   └── příbuznéKomponenty
    │       └── názevKomponenty
    │           ├── index.tsx
    │           └── názevKomponenty.tsx
    └── pages
```

Jednotlivé komponenty jsou vnořeny ve vlastních složkách, aby bylo možné je odděleně stylovat, pokud to bude nutné, pomocí css, nebo scss, tohoto se využívá v komponentě `taskCalendar`.

4.3.1.1 Proč přidat do složky i index.tsx

Abychom mohli komponentu na stránce použít musíme ji importovat pomocí příkazu `import`, pro ukázku jsem se rozhodl použít komponentu `pageHeader`

```
1    import PageHeader from "~/components/layout/pageHeader/pageHeader";
```

Zde je patrné, že v příkazu máme zdvojeně název komponenty, protože máme komponenty rozdělené do jednotlivých složek a poté je máme pojmenované jménem. Jedním z řešení, jak dosáhnout `import`u, co vypadá následovně je přejmenovat komponenty z názvu na `index.tsx`.

```
1     import PageHeader from "~/components/layout/pageHeader/";
```

Adresář by tedy vypadal následovně:

```
/
└── src
    ├── components
    │   └── příbuznéKomponenty
    │       └── názevKomponenty
    │           └── index.tsx
    └── pages
```

Pokud bychom přejmenovali komponenty na `index.tsx` tak sice dosáhneme požadovaného výsledku, ale pokud bychom ve vývojovém prostředí měli otevřeno několik komponent, byl by problém se v projektu zorientovat. Proto místo přejmenování komponent na `index.tsx` ponecháme soubor pojmenovaný názvem komponent a vytvoříme pro každou komponentu jejich vlastní soubor `index.tsx`, tímto způsobem docílíme nezdvojené adresy v `import`u, protože když `import` dostane jako cestu složku automaticky vybere soubor s názvem `index.tsx`. Následně v souboru `index.tsx` musíme importovat komponentu, které se týká a tu dále exportovat. Složka komponenty a její soubory vypadají následovně, irelevantní části kódu byly vynechány.

```
1 ...
2 const PageHeader: React.FC<Props> = (props: Props) => {
3   return (
4     ...
5   );
6 };
7
8 export default PageHeader;
```

Ukázka kódu 4.2 Deklarace komponenty pageHeader

```
1 import PageHeader from "./pageHeader";
2 export default PageHeader;
```

Ukázka kódu 4.3 Soubor index.tsx komponenty pageHeader

4.3.2 /src/pages/

Struktura adresáře `pages` vypadá následovně:

```
pages
├── api
├── groups
├── tasks
├── users
├── _app.tsx
├── 404.tsx
├── 500.tsx
└── index.tsx
```

Ve složce `api` se nachází soubory pro inicializaci API a NextAuth na Frontendu, dále se zde nachází stránka `panel.ts`. K té je povolen přístup pouze ve vývojovém prostředí, jinak vždy vrací kód 404. Stránka byla použita během vývoje na testování API. NextAuth je incializován v souboru `/auth/[...nextauth].ts` a tRPC je inicializováno v `/trpc/[trpc].ts`.

4.3.2.1 _app.tsx

Soubor `_app.tsx` je stavební kámen celé uživatelské části aplikace, tato stránka funguje, jako šablona pro všechny stránky aplikace. Na stránce pomocí komponenty `checkAuth` aplikace kontroluje na všech stránkách, kromě stránky index, zda je uživatel přihlášen a pokud ne, zobrazí uživateli pouze žádost o to, aby se přihlásil.

```
1  ...
2  type Props = {
3    children: ReactNode;
4  };
5
6  const CheckAuth: FC<Props> = ({ children }) => {
7    const router = useRouter();
8    const isIndexPage = router.pathname === "/";
9    const { status } = useSession();
10   if (isIndexPage) return <>{children}</>;
11   if (status === "loading") return <Loading />;
12   if (status === "unauthenticated") return <SignIn />;
13   if (status === "authenticated") return <>{children}</>;
14   return <Code500 />;
15 };
16
17 export default CheckAuth;
```

Ukázka kódu 4.4 Komponenta CheckAuth

CheckAuth přijímá jako parametr `children`, tedy děti, ty komponentě nepředáme stejným způsobem jako ostatní parametry, ale vložením dětí mezi otevírací a uzavírací značky.

```
1     <PageHeader name="Calendar" breadcrumbs={[]} />
```

Ukázka kódu 4.5 Standardní předání parametrů komponentě

```
1     <CheckAuth>
2         <Component {...pageProps} />
3     </CheckAuth>
```

Ukázka kódu 4.6 Předání dětí komponentě

4.4 Rozvržení uživatelských práv

Uživatelé se nacházejí, po přidání vlastníkem, ve skupinách, skupiny mají pod sebou úkoly a kategorie. Aby byl uživateli vylepšen prvotní zážitek při používání aplikace, tak mu je automaticky s účtem i vytvořena skupina. Samozřejmě si uživatelé mohou v aplikaci spravovat nové skupiny. Ve skupinách poté mohou vytvářet úkoly a kategorie, kategorie mohou být přiřazeny jednotlivým úkolům. Majitelé skupin mohou upravovat členství uživatelů ve skupině, vlastnosti skupiny, jako například název nebo převést skupinu na dalšího uživatele.

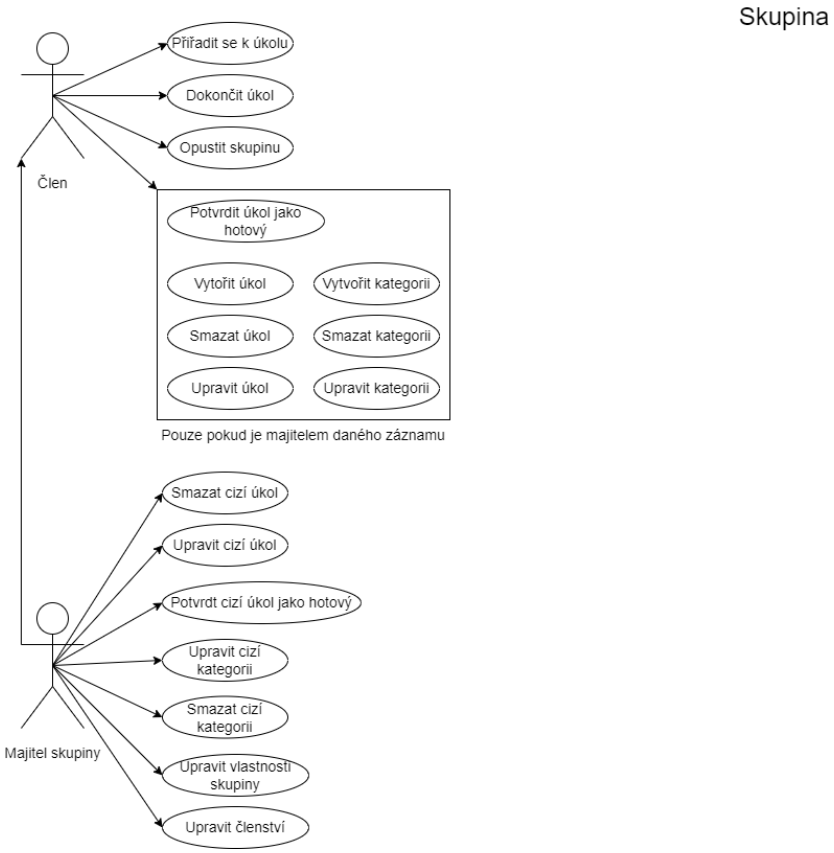
U úkolů a kategorií mohou úpravy provázet jen autoři a také majitel skupiny, pod kterou spadají. Po odebrání, nebo odchodu uživatele ze skupiny, se veškeré úkoly převedou na majitele skupiny.

4.4.1 UseCase Diagramy

Pro přehlednost byl UseCase diagram rozdělen na dvě části, UseCase diagram aplikace a UseCase diagram v zobrazení skupiny.

Obrázek 4.2: UseCase Diagram aplikace

Následující UseCase diagram zobrazuje akce, které mohou uživatelé provádět ve skupině.



Obrázek 4.3: UseCase Diagram skupiny

4.5 Uživatelské rozhraní

Uživatelské rozhraní aplikace je poměrně jednoduché v horní části aplikace se nachází navigační panel. V navigačním panelu se nachází název aplikace, odkaz na úkoly, kalendář a skupiny přihlášeného uživatele. Pokud uživatel není přihlášen tak se v navigačním panelu nachází i tlačítko pro přihlášení, jakmile se uživatel přihlásí, tak tlačítko zmizí a je nahrazeno jeho profilovým obrázkem, ten slouží jako odkaz na stránku uživatele.

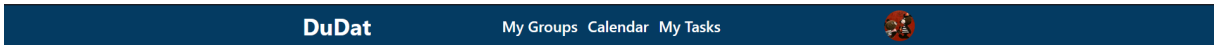
Výchozí nastavení, které používá T3 Stack, je uložit odkaz na profilový obrázek z účtu, pomocí kterého se přihlásil uživatel do aplikace. Toto řešení je poměrně "nešťastné", protože při každém vstupu do aplikace se musí provést volání na externí službu, kde aplikace uživatelův obrázek stáhne. Z tohoto důvodu jsem se rozhodl použít pro generování profilových obrázků knihovnu `DiceBear`[13]. Tato knihovna poskytuje rozhraní pro vytváření různých avatarů formátu SVG.

4.5.1 Registrace do aplikace

Jakmile NextAuth zachytí událost typu registrace nového uživatele, tak zavolá tuto knihovnu a ta mu vytvoří nový profilový obrázek, ten je následně převeden na text a uložen do databáze. Jako "seed" pro generaci je použito uživatelovo `ID` a jako skupinu avatarů aplikace používá "lorelei-neutral"[14]. Při odchytnutí nového uživatele je také pro tohoto uživatele automaticky vytvořena nová skupina.

Jakmile je na serveru zachycena událost[4] o vytvoření nového uživatele, NextAuth tento Event vyvolá pokaždé, co se zaregistruje nový uživatel do aplikace vytvoří se nový záznam v databázi. Při vytvoření uživatele je vygenerován nový uživatelský obrázek pomocí knihovny DiceBear. Po vygenerování obrázku je obrázek převeden na `DataURI` a uložen do databáze. Po uložení obrázku je uživatelovi dále vytvořena první skupina, ta je vytvořena pro ulehčení a zjednodušení prvního vstupu do aplikace.

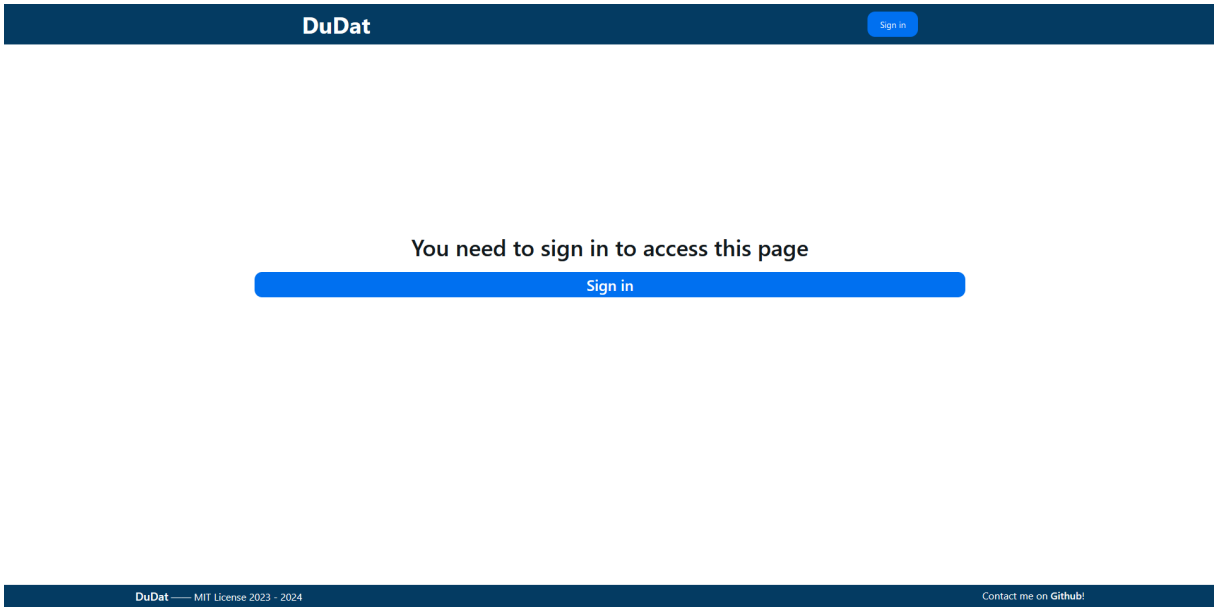
Obrázek 4.4: Navigační panel nepřihlášeného uživatele



Obrázek 4.5: Navigační panel přihlášeného uživatele

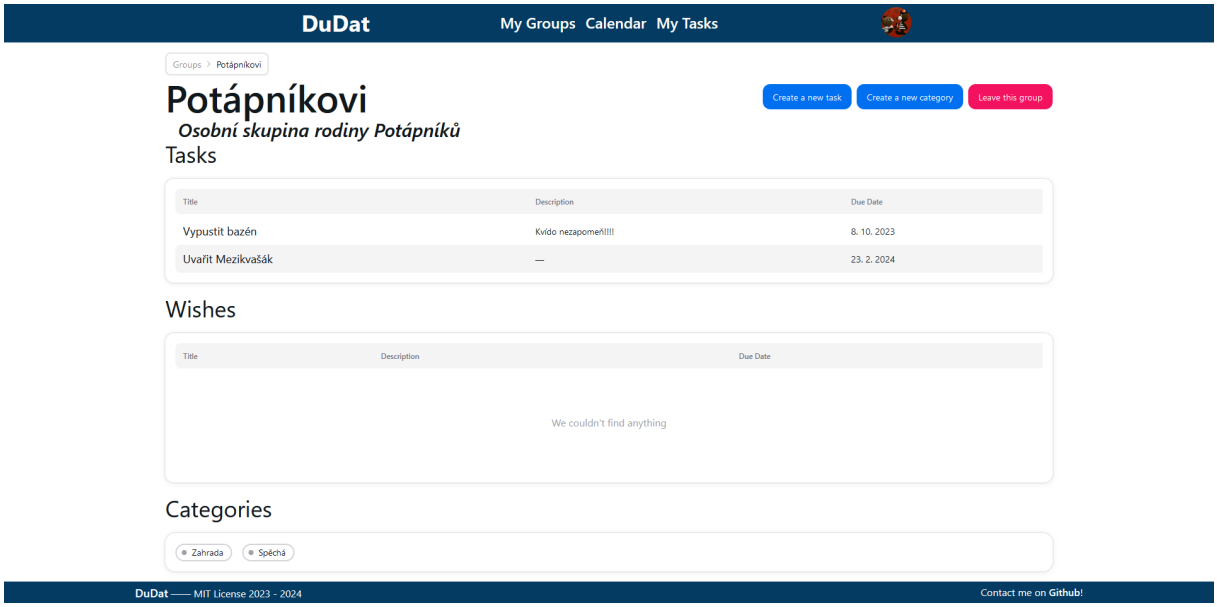
[4] Tedy "Event"

Na každé stránce, která není úvodní, je nepřihlášený uživatel požádán o přihlášení a není vpuštěn do aplikace.



Obrázek 4.6: Stránka se žádostí o přihlášení

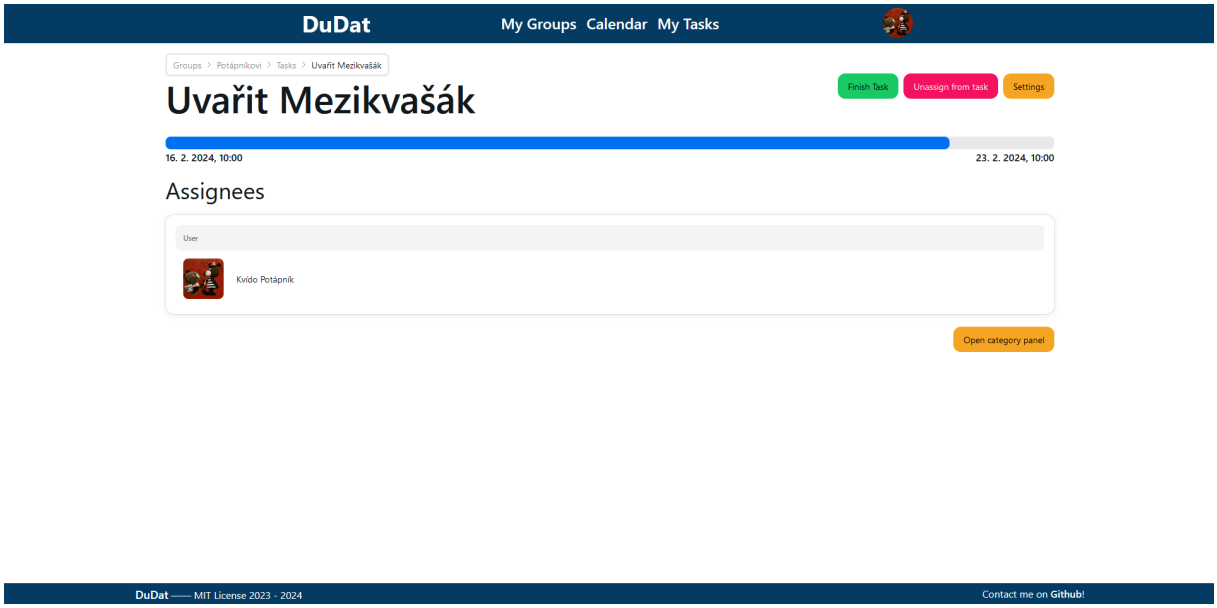
Na stránku skupiny, úkolu a kategorie se dostanou pouze členové dané skupiny, vlastníkovi se místo tlačítka pro opuštění skupiny zobrazí tlačítko pro vstup do nastavení skupiny. Na



Obrázek 4.7: Zobrazení skupiny z pohledu člena

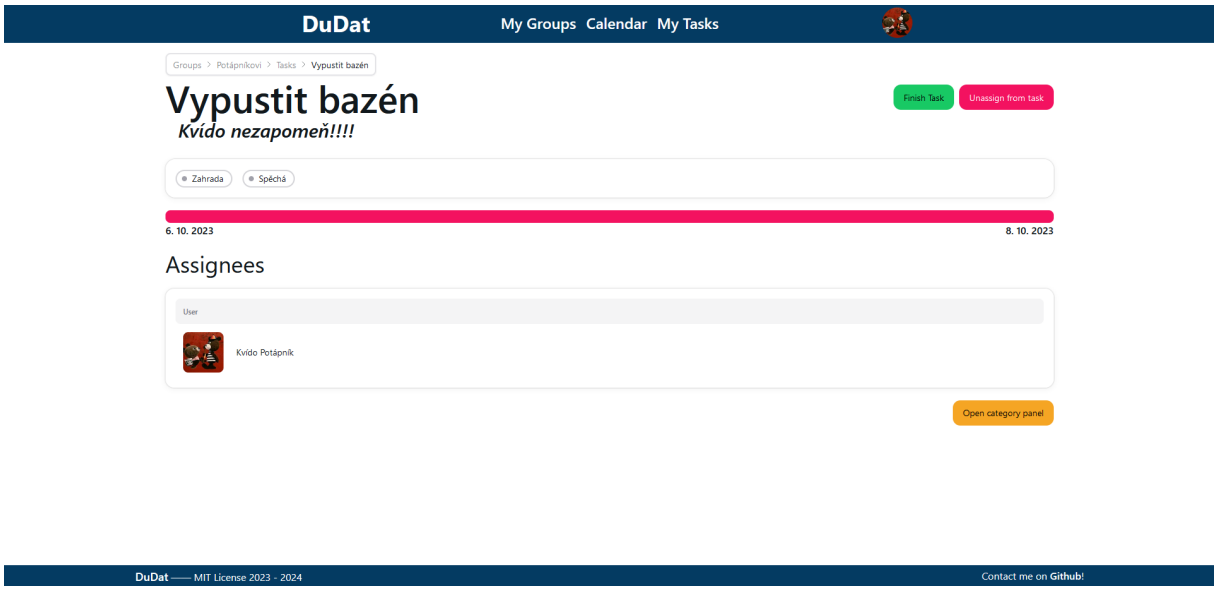
stránkách skupiny vidíme výpis úkolů, přání, kategorií a ostatních členů v dané skupině. Na stránkách úkolu vidíme přiřazené uživatele, možnost se "odhlásit" nebo "přihlásit" k plnění úkolu, tlačítko pro dokončení úkolu, kategorie a indikátor do kdy má být daný úkol hotov.

Autor úkolu a majitel skupiny dále vidí panel pro nastavení, kde může úkol upravit a potvrdit ho jako vypracovaný.



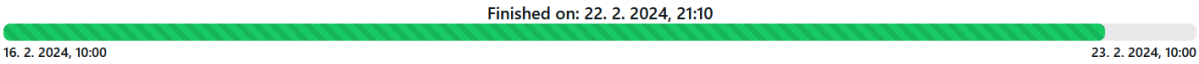
Obrázek 4.8: Zobrazení úkolu z pohledu majitele[15]

Když je na vypracování úkolu stále čas, tak je indikátor modrý, jakmile se blíží konec lhůty na vypracování, konkrétně když zbývá méně než $1/10$ času, indikátor zežloutne a po překročení lhůty je indikátor červený.



Obrázek 4.9: Zobrazení opožděného úkolu z pohledu uživatele

Po potvrzení autorem, nebo majitelem skupiny, že byl úkol vypracován, tak je indikátor vyšrafován. Pokud byl úkol dokončen v čas tak zezelená, jinak zůstane červený a nad indikátorem se zobrazí text, kdy byl úkol dokončen, ne kdy byl potvrzen jako dokončený.

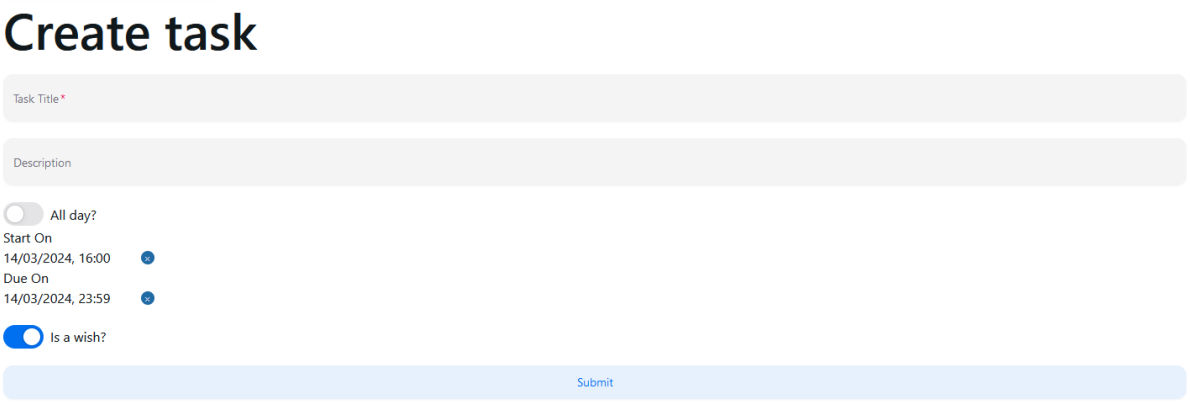


Obrázek 4.10: Zobrazení indikátoru u úkolu, který byl potvrzen a včas odevzdán

Dále se v aplikaci nachází i zobrazení kalendáře na stránce `tasks/calendar/`. Kalendář je řešen knihovnou `react-big-calendar`. Položky v kalendáři odkazují na svou stránku, dále se mění jejich zbarvení stejně jako v indikátoru na stránce úkolů.

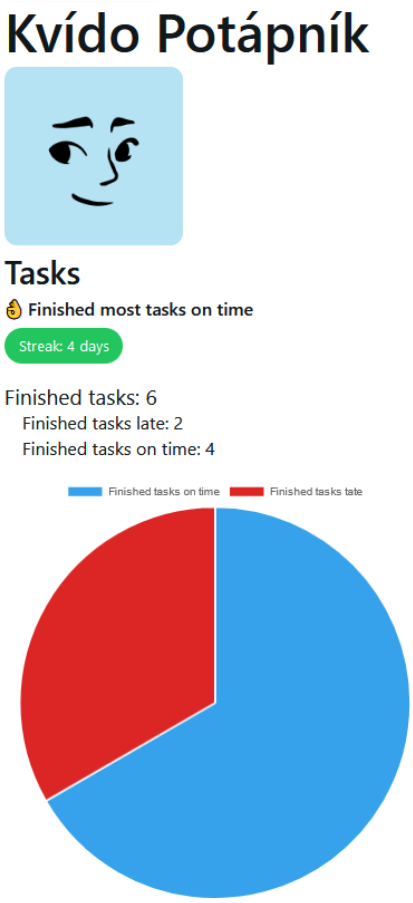
4.5.2 Úkoly x Přání

Jako přání je v aplikaci brán jakýkoliv úkol, který nemá nikoho přiřazeného k sobě, zobrazují se tedy odděleně. Úkol je automaticky převeden na přání jakmile se odhlásí poslední člen od jeho plnění nebo pokud při vytváření nového úkolu zaškrtneme možnost, že se jedná o přání, v tom případě není po vytvoření automaticky přiřazen autor k plnění úkolu.



Obrázek 4.11: Formulář pro vytvoření úkolu, který bude považován za přání

Na stránce uživatele je kromě jejich jména a profilového obrázku zobrazeno grafové vyjádření, které porovnává počet úkolů, které splnil daný uživatel včas a počet úkolů, které odevzdal pozdě. Dále se zde nachází zobrazení takzvaného "Streak" systému, "Streak" se započne se včasným odevzdáním úkolu a je resetován jakmile uživatel odevzdá úkol pozdě. Pokud je uživatel na vlastní stránce tak se zde nachází tlačítko pro odhlášení a vstup na nastavení. V nastavení si může uživatel smazat svůj účet a změnit jméno.



Obrázek 4.12: Zobrazení uživatelské stránky

Než bude uživateli dovoleno smazat svůj účet je potřeba, aby převedl veškeré skupiny, které vlastní na jiného uživatele nebo, aby skupinu smazal. Skupinu majitel převede na jiného uživatele v nastavení skupiny. Jakmile si uživatel smaže účet jsou všechny úkoly a kategorie, které vytvořil převedeny na majitele mateřské skupiny.

5. Závěr

Cílem práce bylo vytvořit intuitivní TODO aplikaci s webovým rozhraním. Aplikace disponuje všemi požadovanými systémy, kterými byly skupiny, úkoly, přání a kalendář. Uživatelé tvoří skupiny, do kterých se vzájemně přidávají, majitelé skupin mohou uživatelům skupinu předat nebo je ze skupiny odstranit. Uživatelé si v skupinách vzájemně zadávají úkoly nebo se hlásí k plnění přání. Časově omezené úkoly se zobrazují v zabudovaném kalendáři. Veškerá uživatelem generovaná data mohou být upravena autorem nebo majitelem skupiny, dále mohou všichni členové skupiny úkol či přání naklonovat. Na profilu každého uživatele je grafové znázornění kolik úkolů dokončil v čas, oproti kolik úkolů dokončil pozdě a jejich "Streak"; "Streak" systém byl přidán nad rámec zadání.

Aplikace je hostovaná na platformě Vercel[5] a databáze je hostovaná na platformě Neon[16].

V rámci práce jsem se naučil s novými technologiemi jako je Prisma, tRPC, NextAuth a TailwindCSS. Dále jsem získal hlubší porozumění s technologiemi Next.js, React a Typescript.

Stanovený výstup práce je funkční a cíl byl splněn. Webová aplikace byla ověřena testováním na uživatelích, kdy se během procesu testování, ale i po jednotlivých funkčních blocích v rámci vývoje aplikace podrobila vyzkoušení z pohledu uživatelů. Toto mi přineslo cennou zpětnou vazbu, která mi pomohla lépe pochopit uživatelské chování a preference. Jednou ze změn, které nastaly na popud zpětné vazby byla například přidání "Streak" systému nebo kopírování úkolů či přání. Hlavní přínos spatřuji v možnosti implementovat takové úpravy, které odpovídají potřebám uživatelů.

Do budoucna bych chtěl, například, umožnit uživatelům měnit časová omezení úkolů přímo v kalendáři pomocí "drag and drop"[1] systému.

[1] "Táhni a pusť"

6. Přílohy

6.1 Literatura

[1] Freelo Bay s.r.o. *Ceník*. 2. břez. 2024. URL: https://www.freelo.io/cs/cenik.

[2] Inc Free Software Foundation. *Různé licence a komentáře k nim*. 14. led. 2024. URL: https://www.gnu.org/licenses/license-list.cs.html#X11License.

[3] Inc. GitHub. *MIT License*. 14. led. 2024. URL: https://choosealicense.com/licenses/mit/.

[4] Theo Browne. *T3 Stack*. 14. led. 2024. URL: https://create.t3.gg/en/introduction.

[5] Inc. Vercel. *Introduction*. 15. led. 2024. URL: https://nextjs.org/docs.

[6] Colin McDonnell. *Introduction*. 19. ún. 2024. URL: https://zod.dev.

[7] Iain Collins. *Documentation*. 16. led. 2024. URL: https://next-auth.js.org/getting-started/introduction.

[8] Beau Coburn. *Should I Choose Tailwind or Vanilla CSS?* 16. led. 2024. URL: https://dev.to/beaucoburn/should-i-choose-tailwind-or-vanilla-css-ccl.

[9] NextUI inc. *Introduction*. 16. led. 2024. URL: https://nextui.org/docs/guide/introduction.

[10] OpenJS Foundation a ESLint. *Configure Rules*. 17. led. 2024. URL: https://eslint.org/docs/latest/use/configure/rules.

[11] *About*. 29. led. 2024. URL: https://www.npmjs.com/about.

[12] Michael 'Ovyerus' Mitchell. *Prismaliser*. 15. ún. 2024. URL: https://prismaliser.app/.

[13] Florian Körner. *DiceBear*. 4. břez. 2024. URL: https://www.dicebear.com/.

[14] Lisa Wischofsky. *Lorelei Neutral*. 4. břez. 2024. URL: https://www.dicebear.com/styles/lorelei-neutral/.

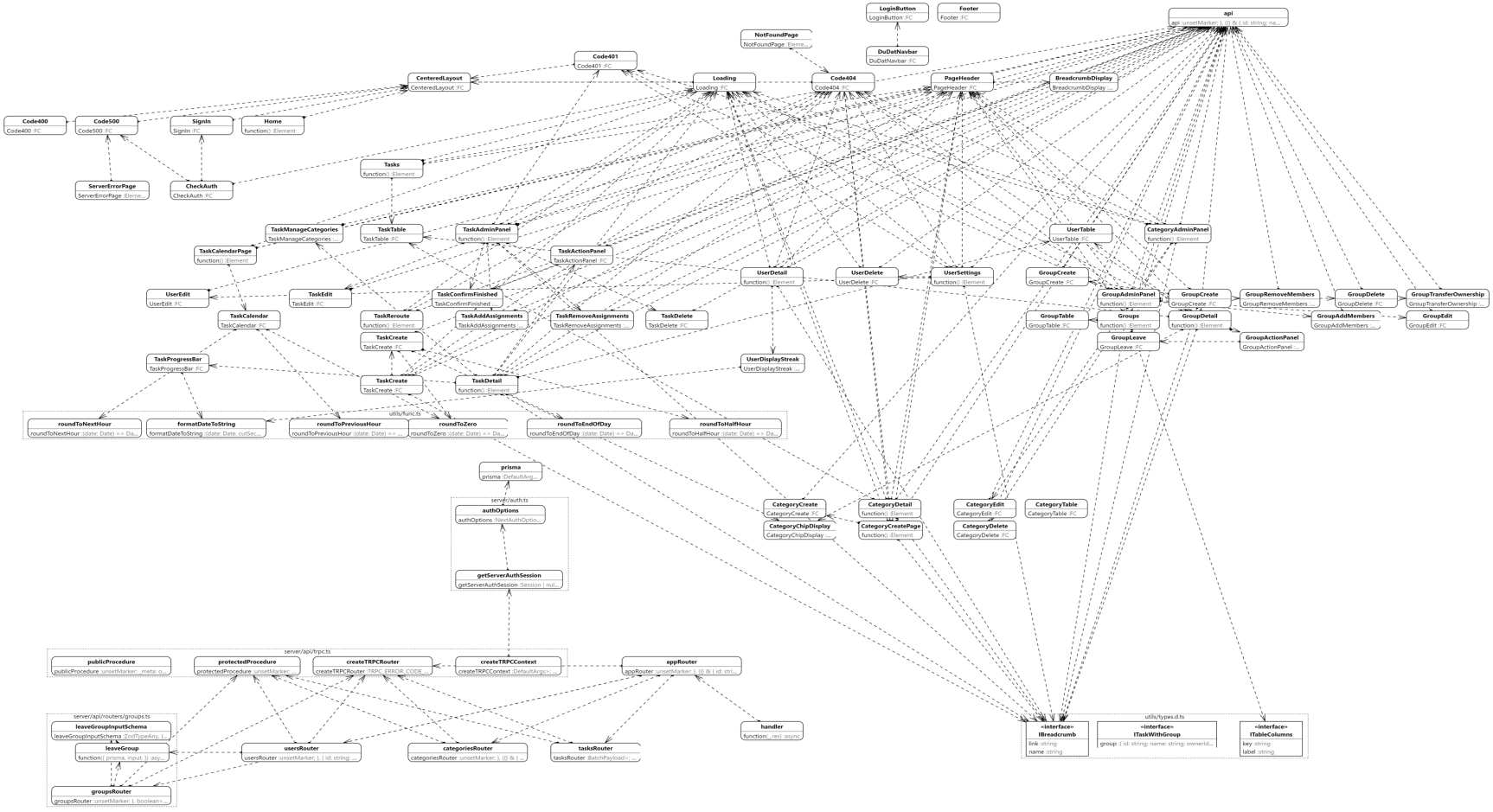
[15] Tomáš Sláma. *Mezikvašák*. 23. ún. 2024. URL: https://www.meziklasi.cz/recepty/mezikvasak/.

[16] Inc. Neon. *Neon*. 23. ún. 2024. URL: https://neon.tech/.

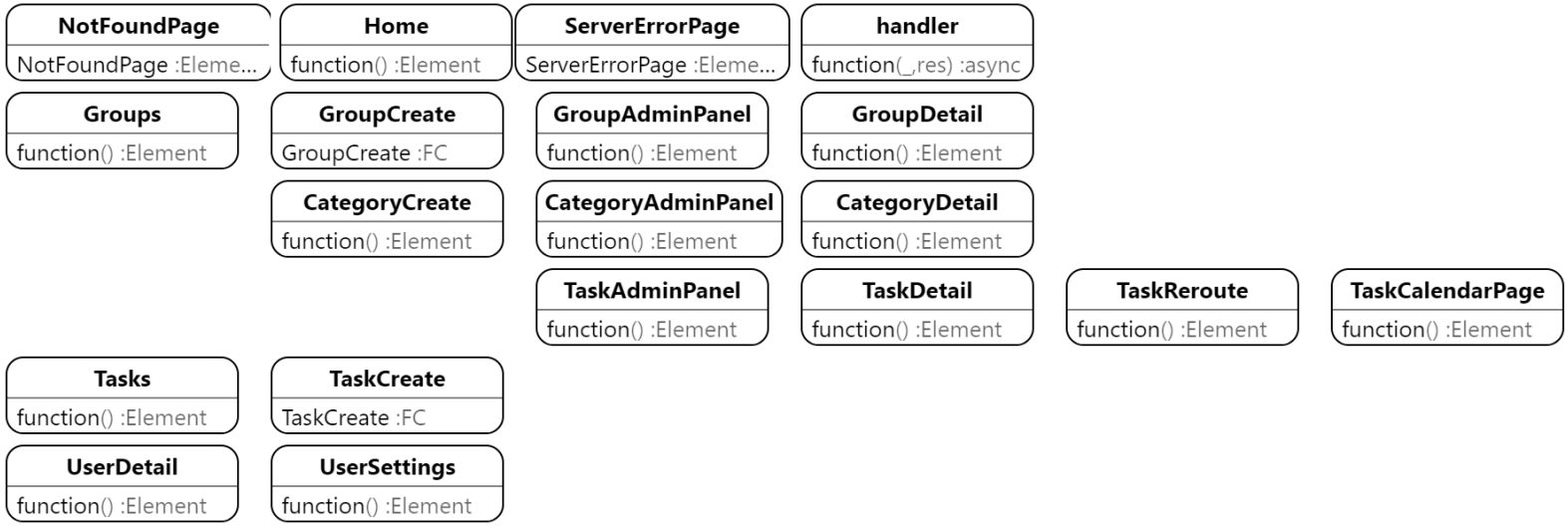
6.2 Obrázky

6.3 Ukázky

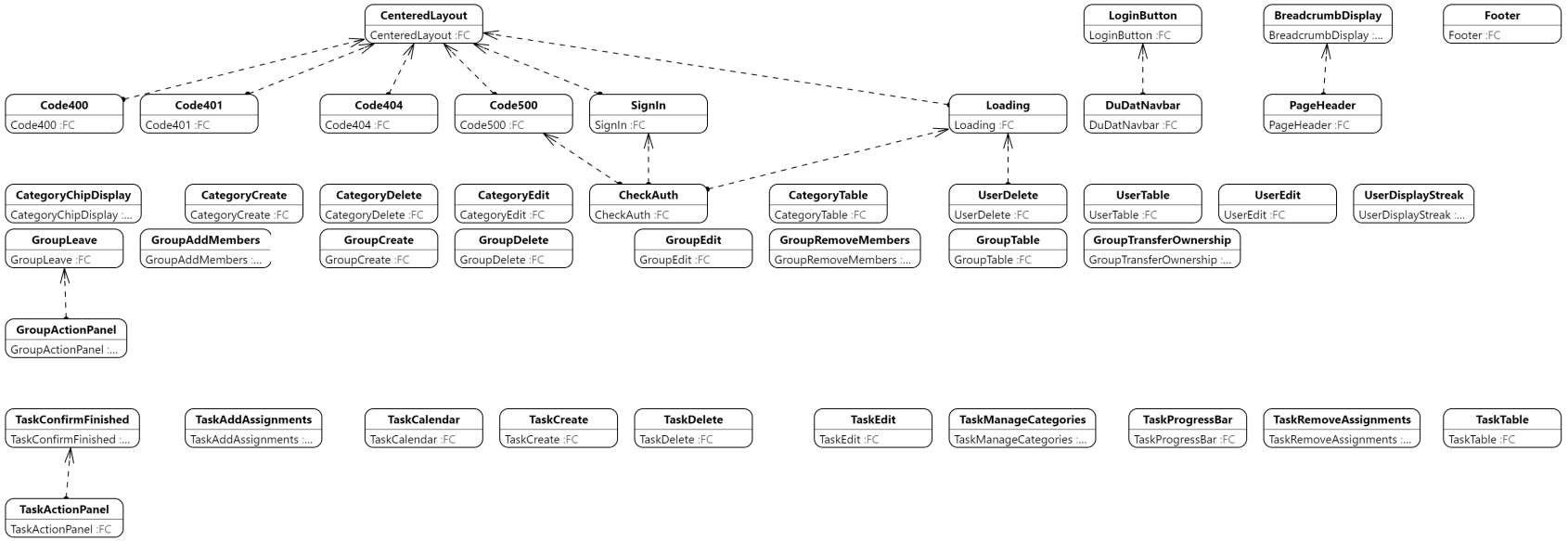
6.4 Class diagramy



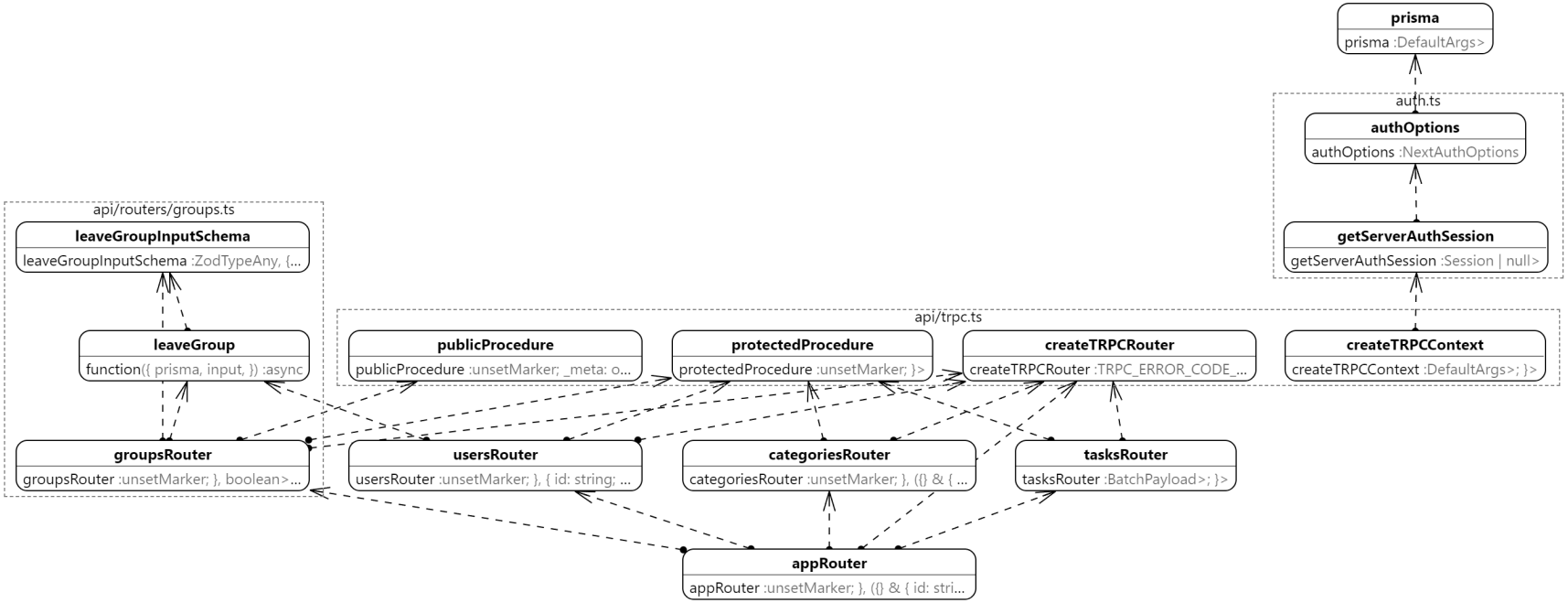
Obrázek 6.1: Class diagram projektu



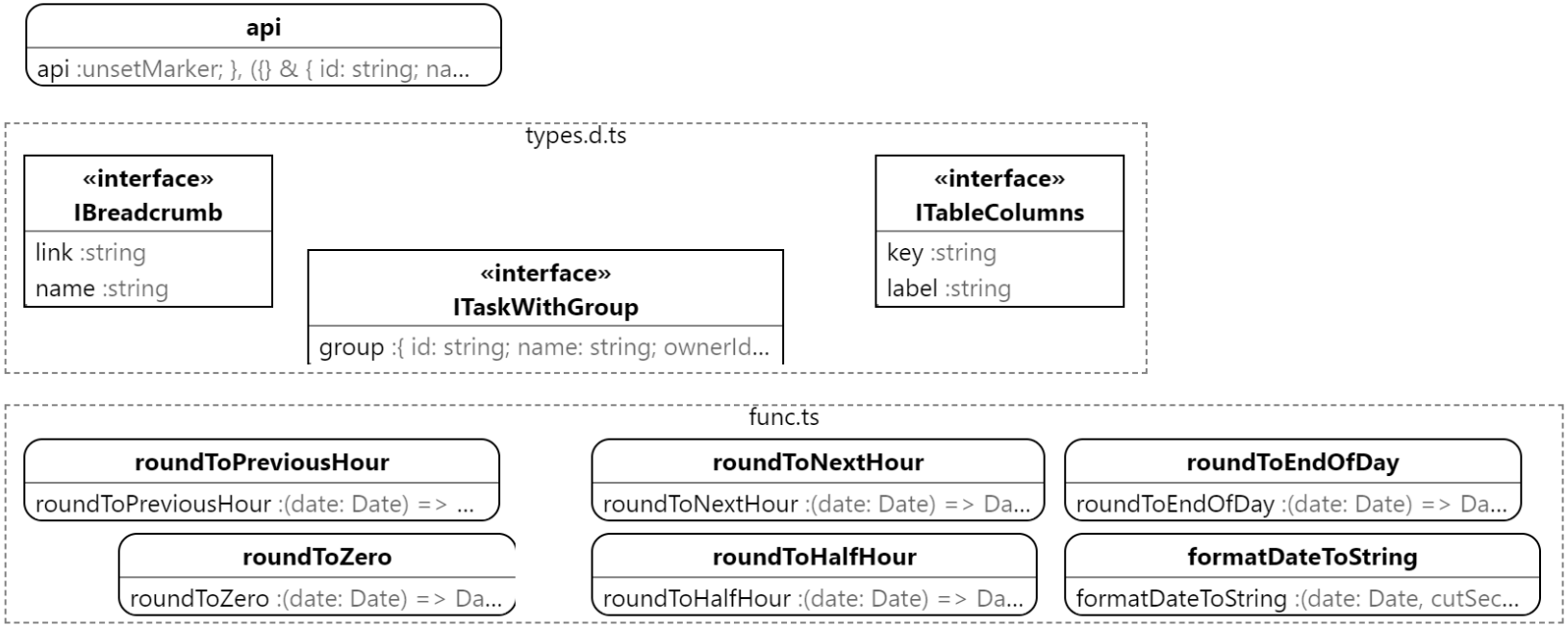
Obrázek 6.2: Class diagram stránek



Obrázek 6.3: Class diagram komponent



Obrázek 6.4: Class diagram serverové části aplikace



Obrázek 6.5: Class diagram podpůrných funkcí a rozhraní