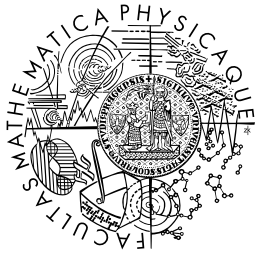


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Michaela Valentová

EdLab – webová aplikácia pre online vzdelávanie

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D.
Konzultant: Mgr. Tereza Hannemann, Ph.D.
Studijní program: Informatika
Studijní obor: Programování a vývoj software

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Za pomoc pri vytváraní tejto bakalárskej práce by som chcela poďakovať vedúcemu práce Mgr. Pavlovi Ježkovi, Ph.D. za jeho cenné rady, skúsenosti, konštruktívnu kritiku, ochotu a čas. Taktiež by som chcela poďakovať Mgr. Tereze Hannemann, Ph.D., ktorá myšlienku práce vymyslela, dodala množstvo nápadov a inšpirácií a pomohla mi lepšie porozumieť zámeru projektu.

V neposlednom rade by som chcela poďakovať svojim rodičom, súrodencom a prieteľovi za ich podporu, motiváciu a poskytnutie prostredia, v ktorom sa dobre žije, študuje a tvorí.

Název práce: EdLab – webová aplikácia pre online vzdelávanie

Autor: Michaela Valentová

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D., Katedra distribuovaných a spolehlivých systémů

Konzultant: Mgr. Tereza Hannemann, Ph.D., Katedra softwaru a výuky informatiky

Abstrakt: Oblasť vzdelávania sa rýchlo vyvíja, pričom vzdelávanie na diaľku sa stáva čoraz populárnejším spôsobom výučby. S príchodom technológií učitelia hľadajú efektívne spôsoby, ako vyučovať aj bez fyzickej prítomnosti svojich študentov. V tejto bakalárskej práci vytvárame webovú aplikáciu, ktorá umožní učiteľom vytvárať vlastné online vzdelávacie kurzy.

Navrhovaná webová aplikácia poskytne učiteľom užívateľsky prívetivé rozhranie na vytváranie a prispôsobenie kurzov podľa ich špecifických potrieb a preferencií. Učitelia budú môcť jednoducho navrhnúť obsah kurzu, rozčlenenie kurzu na kapitoly, interaktívne komponenty a mnoho ďalšieho. Aplikácia vytvorí interaktívne vzdelávacie prostredie tak pre študentov ako aj pre širokú verejnosť zapojenú v kurze *„AI v kontextu“*, ktorý je zadávateľom tohto projektu. Flexibilita aplikácie poskytuje študentom možnosť zvoliť si kedy a kde sa zapoja do kurzu, pričom rýchlosť osvojovania si vedomostí z kurzu sa prispôsobuje ich vlastnému tempu učenia sa. Atraktivitu štúdia zvyšuje možnosť zapájať sa do spoločných aktivít – zúčastňovať sa hlasovaní, overovať získané vedomosti a dostávať spätnú väzbu prostredníctvom vedomostných kvízov.

Aplikácia je vyvinutá vo frameworku React a využíva online databázu Firebase Firestore, jej rozhranie bolo navrhnuté na základe princípov Material Design.

Klíčová slova: online vzdelávanie Firebase Firestore React Material Design

Title: EdLab – web application for management of online courses

Author: Michaela Valentová

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Ježek, Ph.D., Department of Distributed and Dependable Systems

Consultant: Mgr. Tereza Hannemann, Ph.D., Department of Software and Computer Science Education

Abstract: The field of education is rapidly evolving, with distance learning becoming an increasingly popular method of teaching. With the advent of technology, teachers are seeking effective ways to educate their students even without physical presence. In this bachelor's thesis, we are developing a web application that will allow teachers to create their own online educational courses.

The proposed web application will provide teachers with a user-friendly interface for creating and customizing courses according to their specific needs and preferences. Teachers will be able to easily design course content, divide the course into chapters, incorporate interactive components, and much more. The application will create an interactive learning environment for both students and the wider public involved in the course *"AI v kontextu"*, which is the focus of this project. The flexibility of the application will allow students to choose when and where they participate in the course, while the pace of acquiring knowledge will adapt to their individual learning speed. The attractiveness of studying is enhanced by the ability to engage in shared activities such as polling, verifying acquired knowledge, and receiving feedback through knowledge quizzes.

The application is developed using the React framework and utilizes the online Firebase Firestore database. Its interface has been designed based on the principles of Material Design.

Keywords: online learning Firebase Firestore React Material Design

Obsah

1. Úvod

V dnešnej dobe sa oblasť vzdelávania neustále mení, vyvíja a rýchlo napreduje. Dostáva sa do popredia metóda výučby na diaľku, nakoľko technológie sú čoraz dostupnejšie. S príchodom nových možností učitelia hľadajú nové a efektívne spôsoby ako učiť aj bez fyzickej prítomnosti svojich študentov.

V tejto bakalárskej práci sme sa zaoberali vytvorením webovej aplikácie, ktorá umožní učiteľom vytvárať vlastné online vzdelávacie kurzy. Podnet pre tento projekt priniesol jeden z členov projektu **AI v kontextu** [1], ktorý je organizovaný Ústavom formálnej a aplikovanej lingvistiky (ÚFAL) na MFF UK [2]. Projekt okrem iného poskytuje dovzdelávacie kurzy pre učiteľov (informatiky) základných a stredných škôl v oblasti umelej inteligencie a jej presahu do bežného života.

Táto práca si kladie za cieľ vytvoriť rozhranie, v ktorom bude vzdelávanie na diaľku dostupné pre každého, poskytne podporu pre už existujúcu výuku a rozšíri tak možnosti vzdelávania pre širokú verejnosť.

1.1 Slovník pojmov

V úvode sme naznačili základné ciele bakalárskej práce. Aby sme mohli presnejšie popísať všetky požiadavky, analyzovať existujúce riešenia a opísať výsledok práce, bolo potrebné si zaviesť slovník pojmov na základe konzultácií zo zadávateľom projektu a zjednotiť tak význam niektorých výrazov, ktoré síce čitateľ pozná ale v kontexte tejto práce môžu mať odlišný alebo inak špecifický význam.

- Kurz – vzdelávací okruh, ktorý sa typicky venuje jednej hlavnej téme. Má určité príznaky (*tagy*) a obsahuje sériu menších vzdelávacích celkov – kapitol.
- Kapitola – časť kurzu, ktorá sa zameriava na konkrétnu tému. Obsahuje vzdelávacie a interaktívne prvky – komponenty.
- Komponent – súčasť kapitoly, ktorá predstavuje určitý obsah alebo aktivitu. Neskôr v tejto práci budú komponenty rozobrané viac do detailov. Druhy komponentov:
  - ◇ Paragraf – blok náučného textu
  - ◇ Obrázok
  - ◇ Video
  - ◇ Hlasovanie
- Zdieľaný komponent – druh komponentu, ktorý sa môže nachádzať vo viacerých kapitolách a jeho editácia sa prejaví na všetkých miestach. Druhy zdieľaných komponentov:
  - ◇ Kvíz
  - ◇ Pojmové pexeso

- Tag – anglické slovo, ktoré bude reprezentovať príznak – štítok nejakej veci, typicky kurzu. Vďaka nemu budeme schopní kurzy kategorizovať a vyhľadávať (filtrovať).

1.2 Typy užívateľov

Vytvorená webová aplikácia by mala podporovať dva základné typy / role používateľov pre interakciu s platformou – učiteľov a študentov. Oba typy používateľov majú dôležitú úlohu v rámci webovej aplikácie a ich potreby a očakávania sú dôležité pre jej úspešnú implementáciu. Požiadavky a funkcionality webovej aplikácie sú navrhnuté tak, aby zodpovedali potrebám oboch typov používateľov aj s ohľadom na ich vek alebo skúsenosti, s cieľom zabezpečiť efektívny a interaktívny proces vzdelávania.

V nasledujúcich častiach tejto bakalárskej práce budeme bližšie skúmať požiadavky a očakávania oboch typov používateľov a analyzovať, ako sú tieto požiadavky zohľadnené v návrhu a implementácii webovej aplikácie.

1.2.1 Učiteľ (osoba, ktorá tvorí vzdelávací obsah)

Učiteľ je hlavným tvorcom vzdelávacieho obsahu – kurzov v tejto aplikácii. Učitelia majú možnosť vytvárať a prispôsobovať svoje vlastné online vzdelávacie kurzy v rámci webovej aplikácie. Môžu pridávať rôzne typy obsahu, ako sú texty, obrázky, videá, kvízy, hlasovania a iné interaktívne prvky. Sú zodpovední za riadenie obsahu, jeho aktuálnosť a korektnosť.

Ako sme už stanovili v úvode práce, primárne sa zameriavame na pedagógov vyučujúcich na MFF UK v projekte AI v kontextu a očakávame, že sú dostatočne technicky vzdelaní. Bakalárska práca však má potenciál byť dostupná aj pre iné fakulty alebo univerzity, preto musíme rolu učiteľa definovať ako akéhokoľvek pedagogického zamestnanca bez nutnosti širšieho technického vzdelania, ktorý má záujem vytvárať online vzdelávací obsah a poskytnúť tak podporu pre svoju už existujúcu výuku.

1.2.2 Študent (osoba, ktorá sa chce učiť)

Študenti sú používatelia, ktorí majú možnosť zúčastniť sa kurzov vytvorených učiteľmi. Majú prístup k obsahu kurzov, pracujú s interaktívnymi aktivitami, vypĺňajú kvízy. Študenti si môžu vyberať kurzy na základe svojich záujmov, potrieb a preferencií a môžu sa učiť vlastným tempom.

V úvode sme zmienili, že projekt bude slúžiť aj ako podpora pre sériu vzdelávacích kurzov AI v kontextu, kde v roli študenta vystupujú učitelia základných a stredných škôl. Pre presnejšie popísanie typu používateľa – študenta je potrebné získať informácie o priemernom veku, technickom vzdelaní a pod.

Z prieskumu *Ministerstva školství, mládeže a tělovýchovy ČR* [3] z roku 2019 vyplýva, že priemerný vek pedagógov na školách je 47,2 rokov.

1.3 Typické použitie – scenáre

V predošlej sekcii sme si stanovili hlavné koncepty, z ktorých by mal systém vychádzať. Na základe nich by sme chceli predstaviť typické príklady použitia našej aplikácie. Následne budeme vedieť podrobnejšie určiť všetky funkcie a komponenty aplikácie a vzťahy medzi nimi. Ako sme už spomenuli v sekcii 1.2, naša aplikácia by mala mať dva typy užívateľov (študent a učiteľ) a teda scenáre rozdelíme do skupín podľa primárneho typu a zámeru užívateľa.

1.3.1 Scenáre pre rolu Učiteľ

1. **Prihlásenie**

   Užívateľ, ktorý je tvorcom kurzu, sa chce prihlásiť do aplikácie, aby získal prístup k funkcionalitám určeným pre tvorcov kurzu. Chce sa prihlásiť pomocou svojho pracovného alebo univerzitného účtu, ktorý mu umožní spravovať a vytvárať kurzy.
   Ako sme spomenuli v časti 1.2.1, primárny typ učiteľa bude predstavovať zamestnanca MFF UK, ktorý pracuje na projekte AIvK a vlastní pracovný / univerzitný účet. Potenciálne by mohlo byť prihlásenie umožnené aj iným fakultám alebo univerzitám.

2. **Vytvorenie kurzu**

   Užívateľ chce vedieť, ako vytvoriť vlastný vzdelávací *kurz*. Chce mať možnosť nastaviť obsah kurzu, pridať rôzne materiály, ako sú texty, videá, obrázky a iné. Chce mať prístup k nástrojom na správu a organizáciu kurzu.

3. **Priraďovanie tagov kurzu**

   Užívateľ má vytvorené nejaké *kurzy*, každý je obsahovo niečím odlišný, ale predsa len sa v niektorých častiach ich obsah môže prelínať. Chcel by vedieť svojim kurzom prideliť nejaké štítky – *tagy*, ktoré by vedeli kurz jednoducho kategorizovať a následne by podľa nich vedel kurzy vyhľadávať (vytriediť ich podľa týchto príznakov).

4. **Vytvorenie kapitoly**

   Užívateľ má vytvorený *kurz* s nejakou širšou tématikou, ktorú vieme prirovnať k okruhu nejakého predmetu na univerzite. V danom kurze si chce užívateľ vedieť vytvoriť menšie vzdelávacie časti – *kapitoly*, ktoré vieme logicky prirovnať k obsahu jednej prednášky zvoleného predmetu.

5. **Vytvorenie komponentov kapitoly**

   Užívateľ má vytvorenú *kapitolu* a chcel by do nej vedieť pridať rôzne vzdelávacie prvky. Chcel by vedieť pridávať bloky náučného textu, obrázky a videá. Takisto má záujem urobiť kapitolu zaujímavú, prehľadnú a interaktívnu, takže do nej chce pridávať prvky, s ktorými vie študent pracovať, overovať si svoju získanú znalosť a dostávať spätnú väzbu.

6. **Vytvorenie hlasovania**

   Užívateľ má záujem vedieť, ako sa jeho študentom páčila jeho kapitola. Na konci kapitoly chce vytvoriť *hlasovanie* s otázkou *„Páčila sa Vám táto kapitola?“* s možnosťami odpovedí *„Áno“* a *„Nie“*. Potrebuje teda vytvoriť v kapitole prvok, ktorý mu zobrazí názory študentov alebo prehľad ich odpovedí na nejakú tému. Na hlasovanie budú mať možnosť študenti odpovedať výberom odpovede zo zoznamu, ktorý tvorca kapitoly vytvoril.

7. **Vytvorenie kvízu**

   Užívateľ chce poskytnúť svojim študentom možnosť si overiť získané znalosti. Potrebuje mať možnosť vytvoriť sériu kvízových otázok a odpovedí. Pri každej sérii odpovedí má možnosť určiť ktoré odpovede sú správne.

8. **Vytvorenie pojmového pexesa**

   Tvorca kurzu by chcel vytvoriť interaktívny *komponent*, v ktorom môžu jeho študenti spájať novo naučené odborné pojmy a ich významy. Takýmto spôsobom budú schopní si učivo lepšie a zábavnejšie zapamätať.

9. **Editácia a odstránenie**

   Užívateľ má rozsiahlu databázu *kurzov, kapitol a komponentov*, ktoré vytvoril. V prípade, že zistí, že akákoľvek časť jeho obsahu je nepresná, neaktuálna alebo obsahuje preklep, má možnosť ju editovať. Zmena zdieľaných komponentov sa prejaví na všetkých miestach kde je komponent použitý.
   Ak sa obsah jeho práce/tvorby stane nepotrebný alebo inak nevyužiteľný, má užívateľ možnosť kurz alebo akúkoľvek jeho časť natrvalo odstrániť. Na nezvratnosť akcie odstránenia chce byť vopred upozornený aby sa mu to nestalo nechtiac.

1.3.2 Scenáre pre rolu Student

1. **Výber kurzu**

   Užívateľ má možnosť si z ponuky *kurzov* vybrať ten, ktorý ho zaujíma a jednoducho kliknutím si vie zobraziť jeho obsah. Po výbere konkrétneho kurzu sa mu zobrazia podrobnosti o kurze, a jeho kapitoly.

2. **Vyhľadávanie kurzu**

   Užívateľ sa nachádza na stránke ponuky všetkých *kurzov*, kde je ich dostupné veľké množstvo. Kurzov je však príliš veľa na to, aby sa v nich vedel jednoducho orientovať. Preto by mal rád možnosť vyhľadávať kurzy podľa ich názvov alebo podľa kategórií – *tagov*, ktoré ho zaujímajú. Užívateľ môže zadávať kľúčové slová, úseky z názvov alebo *tagy*, aby zúžil výber kurzov podľa svojich preferencií a potrieb.

3. **Výber kapitoly**

   Užívateľ má otvorený konkrétny *kurz* a v ňom vidí jeho základné informácie a všetky kapitoly, ktoré obsahuje. Chce mať možnosť z ponuky kapitol si vybrať tú, ktorá ho zaujíma, a kliknutím si zobraziť jej obsah. Po výbere konkrétnej *kapitoly* môže vidieť názov kapitoly, popis, a jej obsah.

4. **Zúčastnenie sa hlasovania**

   Užívateľ sa nachádza na stránke *kapitoly*, prečítal si všetok náučný obsah a teraz by sa rád zúčastnil *hlasovania*, ktoré táto kapitola obsahuje. Jednoducho klikne na odpoveď, s ktorou sa stotožňuje a následne má možnosť vidieť ako hlasovali ostatní študenti.

5. **Vyplnenie kvízu**

   Užívateľ sa nachádza na stránke *kapitoly*, naštudoval si celý obsah a rád by si svoju znalosť overil. Daná kapitola obsahuje *kvíz*, ktorý pozostáva z viacerých odpovedí kde je aspoň jedna správna. Užívateľ chce mať možnosť sa kvízu zúčastniť, vie odpovedať na všetky otázky a priebežne vidí svoj progres. Následne po zobrazení vidí získané skóre – počet odpovedí. V prípade, že je s výsledkom nespokojný má možnosť kvíz opakovať.

6. **Interakcia s pojmovým pexesom**

   V kapitole sa bude nachádzať komponent predstavujúci *pojmové pexeso* pozostávajúce z náhodne rozložených kariet dvojíc pojmov a ich významov. Užívateľ bude schopný si toto pexeso „zahrať“ a overiť tak svoju získanú znalosť. Cieľom tohto komponentu je správne si zapamätať spojenie dvojíc a ich pozíciu za čo najmenší počet pokusov. Počet pokusov bude užívateľovi jasne zobrazený aby bol schopný sledovať svoj progres. Užívateľ bude mať umožnené pexeso opakovať.

1.4 Funkčné požiadavky

V tejto sekcii si zhrnieme funkčné požiadavky systému vychádzajúce zo základných scenárov (sekcia 1.3)

1. Užívateľ musí byť schopný sa prihlásiť.
2. Prihlásený užívateľ musí byť schopný sa odhlásiť.
   2.1. Pred odhlásením musí byť vyzvaný, aby svoju akciu odhlásenia potvrdil.
3. Užívateľ musí byť schopný sa v aplikácii navigovať.
4. Prihlásený užívateľ si vie zobraziť ponuku kurzov, ktoré vytvoril.
5. Prihlásený užívateľ vie vytvoriť nový kurz.
   5.1. Prihlásený užívateľ vie vyplniť názov kurzu.
   5.2. Prihlásený užívateľ vie vyplniť podnadpis kurzu.
   5.3. Prihlásený užívateľ vie pridať a vytvoriť tagy, ktoré sú pridelené ku kurzu.
6. Prihlásený užívateľ vie editovať kurz, ktorý vytvoril.
   6.1. Prihlásený užívateľ vie editovať názov kurzu.
   6.2. Prihlásený užívateľ vie editovať podnadpis kurzu.

6.3. Prihlásený užívateľ vie editovať pridelené tagy.

7. Prihlásený užívateľ vie vymazať kurz, ktorý vytvoril.
8. Prihlásený užívateľ vie otvoriť zoznam kapitol, ktoré kurz obsahuje kliknutím na položku kurzu.
9. Prihlásený užívateľ vie vytvoriť novú kapitolu.
   9.1. Prihlásený užívateľ vie vyplniť názov kapitoly.
   9.2. Prihlásený užívateľ vie vyplniť podnadpis kapitoly.
10. Prihlásený užívateľ vie editovať kapitolu, ktorú vytvoril.
    10.1. Prihlásený užívateľ vie editovať názov kapitoly.
    10.2. Prihlásený užívateľ vie editovať podnadpis kapitoly.
11. Prihlásený užívateľ vie vymazať kapitolu, ktorú vytvoril.
12. Prihlásený užívateľ vie v kapitole, ktorú vytvoril, pridať nový *komponent*.
    12.1. Prihlásený užívateľ vie vytvoriť náučný blok textu – *paragraf*
        12.1.1. Prihlásený užívateľ vie vytvoriť názov *paragrafu*.
        12.1.2. Prihlásený užívateľ vie vytvoriť obsah *paragrafu*.
    12.2. Prihlásený užívateľ vie pridať komponent obrázka.
    12.3. Prihlásený užívateľ vie pridať komponent videa.
    12.4. Prihlásený užívateľ vie vytvoriť hlasovanie.
        12.4.1. Prihlásený užívateľ vie pridať hlasovaciu otázku do hlasovania, ktoré vytvára.
        12.4.2. Prihlásený užívateľ vie pridať možnosti odpovedí ku hlasovacej otázke.
        12.4.3. Prihlásený užívateľ vie vytvoriť z hlasovania šablónu.
        12.4.4. Prihlásený užívateľ vie použiť nejakú šablónu hlasovania, ktorú vytvoril.
13. Prihlásený užívateľ vie v kapitole, ktorú vytvoril, editovať každý *komponent*.
14. Prihlásený užívateľ vie v kapitole, ktorú vytvoril, vytvoriť *zdieľaný komponent*.
    14.1. Prihlásený užívateľ vie vytvoriť *kvíz*.
        14.1.1. Prihlásený užívateľ vie vytvoriť kvízové otázky.
        14.1.2. Prihlásený užívateľ vie pridať možné odpovede ku každej kvízovej otázke.
        14.1.3. Prihlásený užívateľ vie použiť kvíz, ktorý už vytvoril.
    14.2. Prihlásený užívateľ vie vytvoriť *pojmové pexeso*.
        14.2.1. Prihlásený užívateľ vie vytvoriť dvojice pojmov pexesa.
        14.2.2. Prihlásený užívateľ vie použiť *pojmové pexeso*, ktoré už vytvoril.

15. Prihlásený užívateľ vie v kapitole, ktorú vytvoril editovať každý *zdieľaný komponent*.

    15.1. Editáciou *zdieľaného komponentu* sa zmení jeho obsah na všetkých miestach kde je použitý.

16. Neprihlásený užívateľ má prístup ku vzdelávaciemu obsahu bez nutnosti prihlásenia.

17. Neprihlásený užívateľ má prístup ku vzdelávaciemu obsahu zadarmo.

18. Neprihlásený užívateľ si vie zobraziť ponuku všetkých kurzov.

19. Neprihlásený užívateľ vie vyhľadávať kurzy podľa názvu alebo *tagov*.

20. Neprihlásený užívateľ si vie zobraziť konkrétny kurz a kapitoly, ktoré obsahuje.

21. Neprihlásený užívateľ si vie zvoliť kapitolu so vzdelávacím obsahom.

    21.1. Neprihlásený užívateľ má prístup ku obsahu kapitoly a všetkým jej komponentom.

    21.2. Neprihlásený užívateľ má možnosť hlasovať v hlasovaní.

        21.2.1. Neprihlásený užívateľ má možnosť vybrať možnosť z odpovedí v hlasovaní.

        21.2.2. Neprihlásený užívateľ má možnosť po hlasovaní vidieť výsledok celého hlasovania.

    21.3. Neprihlásený užívateľ má možnosť zúčastniť sa kvízu.

        21.3.1. Neprihlásený užívateľ má možnosť vybrať z možností odpovedí v kvízových otázkach.

        21.3.2. Neprihlásený užívateľ má možnosť po zodpovedaní kvízových otázok vidieť svoj výsledok.

        21.3.3. Neprihlásený užívateľ má možnosť zopakovať vyplnenie kvízu.

    21.4. Neprihlásený užívateľ má možnosť zúčastniť sa *pojmového pexesa*.

        21.4.1. Neprihlásený užívateľ má možnosť vidieť svoj výsledok – celkový počet ťahov.

        21.4.2. Neprihlásený užívateľ má možnosť hru zopakovať.

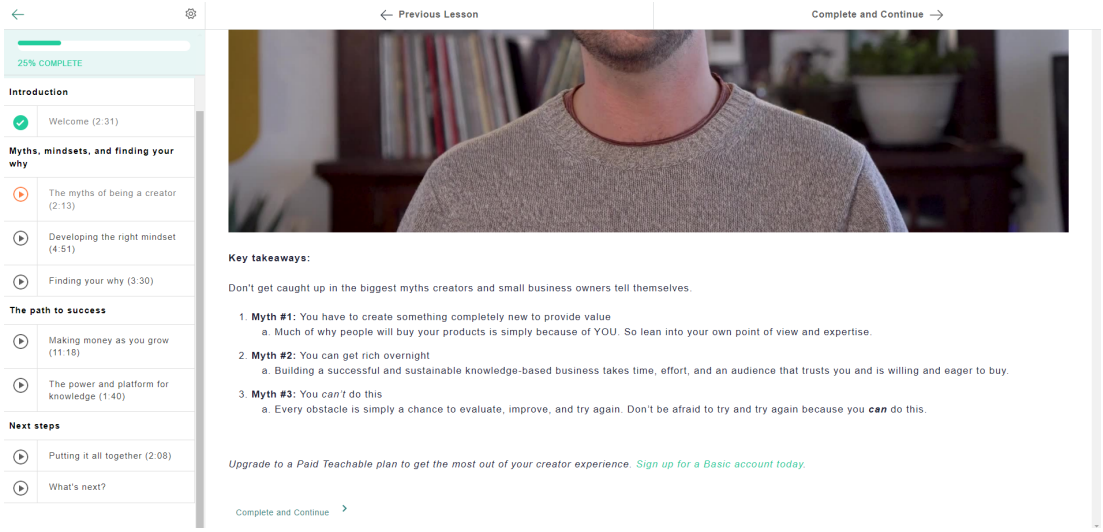
1.5 Konkurenčné riešenia

V obsahu tejto kapitoly sa zameriame na iné webové platformy, ktoré sa venujú online vzdelávaniu. Porovnáme základné funkcionality v kontraste s funkčnými požiadavkami špecifikovanými v predošlej kapitole. Vo výbere sa nachádzajú najpopulárnejšie aplikácie alebo tie, ktoré sa svojím fungovaním najviac približujú zámeru našej práce.

1.5.1 Teachable

Teachable je komplexná vzdelávacia platforma pre vytváranie a správu kurzov, je možné v nej vytvoriť širokú škálu vzdelávacieho obsahu s rôznym zameraním (programovanie, maľovanie vodovými farbami, varenia a pečenie,atď.). Medzi jej hlavné výhody patrí to, že okrem vzdelávacích komponentov dáva možnosť tvorcovi kurzu pridávať aj audiozáznamy a úseky kódu, okrem toho poskytuje aj podporu pre rôzne marketingové nástroje. Aplikácia je komplexnejšia, dajú sa v nej využívať preddefinované motívy, vytvárať predajnú stratégiu a mnoho iného. Na obrázku 1.1 je zobrazený príklad kurzu vytvoreného v tejto aplikácii.

Ku nevýhodám by sme však museli zaradiť podstatný fakt, že aplikácia je spoplatnená, respektíve obsahuje len obmedzený bezplatný plán. Následne sú spoplatnené aj všetky kurzy, ktoré sú dostupné, takže ide skôr o platformu produkujúcu zisk. Celkovo je aplikácia až príliš komplexná a vytvorenie nejakého kurzu zaberie násobne viac času v porovnaní s požiadavkami.



Obr. 1.1: Screenshot kurzu z aplikácie Teachable

1.5.2 Thinkific

Thinkific je jedna z ďalších platforiem poskytujúcich možnosť vytvoriť si vlastné vzdelávacie kurzy. Podobne ako predošlá aplikácia, aj táto sa sústreďuje na generovanie zisku. Ponúka integráciu rôznych nástrojov ako sú automatické rozposielanie emailov, vytváranie darčekových poukážok, reklama, analýza správania a výsledkov študentov a mnoho ďalšieho nad rámec nášho zadania. Vytváranie kurzov je spoplatnené od počtu kurzov (resp. od zvoleného platobného plánu – viď obrázok nižšie). Takisto prístup študentov ku kurzom je spoplatnený.

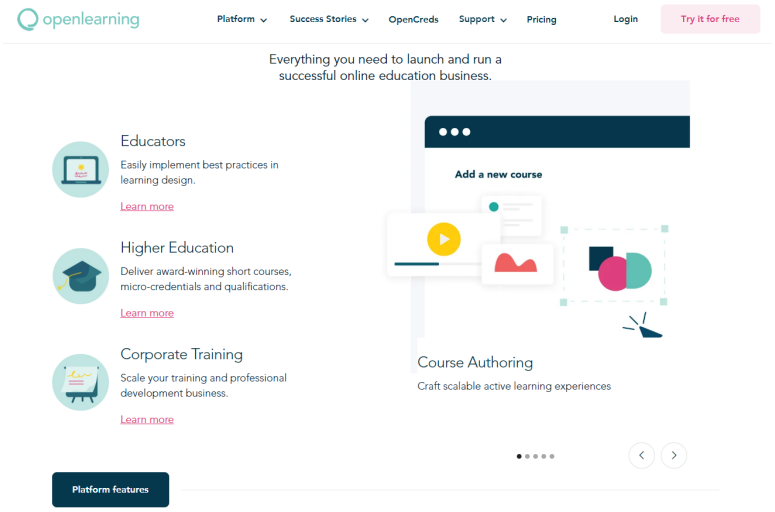
1.5.3 OpenLearning

OpenLearning je škálovateľná online vzdelávacia platforma, ktorej cieľom je umožniť pedagógom a inštitúciám pripraviť svojich študentov na budúcu prácu. Častokrát je využívaná väčšími inštitúciami (Sunway University, Charles Sturt

University, UNSW Sydney, Australian Catholic University, a pod.). Aplikácia obsahuje aj videonávod [4] s prehľadom funkcionalít.

Ku hlavým výhodám by sme zaradili možnosť spolupráce celej univerzity s aplikáciou, a nielen jej zamestnancov individuálne. Takisto poskytuje možnosti správy študentov a dovoľuje vytvoriť celú vzdelávaciu platformu. Na obrázku 1.2 môže čitateľ vidieť rozhranie webovej aplikácie.

Medzi nevýhody patrí to, že väčšina dostupných kurzov je spoplatnená a časovo veľmi náročná (jeden kurz môže trvať aj 16 týždňov). Zámerom aplikácie je nahradiť nutnosť univerzitného vzdelania pomocou týchto certifikovaných kurzov, čo nespĺňa požiadavky stanovené vyššie (požadované sú dostupné jednoduché dovzdelávacie kurzy určené aj pre laickú verejnosť).



Obr. 1.2: Screenshot aplikácie OpenLearning

1.5.4 Moodle

Moodle je výkonný a všestranný open-source systém na správu výučby (Learning Management System – LMS), ktorý si získal široké uplatnenie v rôznych vzdelávacích prostrediach. Kľúčové funkcie zahŕňajú správu kurzov (vytváranie úloh, kvízov, fór a sledovania pokroku študentov), možnosť komunikácie a spolupráce (správy, fóra a chat), vytváranie a zdieľanie obsahu (text, multimédiá a interaktívne zdroje), hodnotenie a spätnú väzbu (kvízy, úlohy a prieskumy).

Moodle ponúka niekoľko výhod, ktoré z neho robia obľúbenú voľbu pre e-learning. Je populárny vďaka tomu že je open-source, dá sa dobre prispôsobiť a škálovať a má veľkú aktívnu komunitu používateľov.

Aj keď má Moodle mnoho výhod, má aj určité obmedzenia, ako je napríklad krivka učenia (v porovnaní s niektorými inými platformami online vzdelávania má strmšiu krivku učenia, pretože vyžaduje, aby sa pedagógovia oboznámili s jeho funkciami a možnosťami prispôsobenia), vyžadovaná technická odbornosť, obmedzený predvolený dizajn používateľského rozhrania a nemožnosť vytvárania kurzov dostupných pre verejnosť bez prihlásenia.

1.5.5 Google Classroom

Google Classroom je ďalšia bezplatná platforma pre online výučbu a správu tried od spoločnosti Google. Táto platforma umožňuje učiteľom vytvárať triedy a zdieľať s nimi materiály, ako sú dokumenty, prezentácie, videá a iné multimédiá. Študenti majú prístup k týmto materiálom a môžu odovzdávať úlohy, ktoré sú automaticky triedené a hodnotené. Učitelia môžu tiež komunikovať so študentami pomocou rôznych nástrojov, ako sú diskusné fóra a správy.

Táto platforma má však aj nedostatky. Pri vytváraní novej triedy, musí učiteľ vytvoriť unikátny účet pre každého študenta. Výuka je limitovaná pre určitú skupinu ľudí, ktorí vopred dostanú vhodnú akreditáciu pre povolenie k prístupu do danej triedy – to znamená, že aplikácia neposkytuje možnosť prístupu ku vzdelaniu pre verejnosť. Aj v prípade, že študent dostane možnosť sa pripojiť k triede, musí mať účet Google. Keďže sa jedná o súkromné prostredie, je nemožné zdieľať obsah so širšou komunitou.

1.5.6 Zhrnutie

Webové aplikácie, ktoré sme vyskúšali, majú spoločné to, že očakávajú veľké množstvo užívateľov, dávajú si záležať na vizuálnom spracovaní, sú globalizované a stoja za nimi spoločnosti s veľkým kapitálom. Dali by sa primárne dali rozdeliť do dvoch skupín, tie ktoré sa zameriavajú na generovanie zisku (Teachable, Thinkific, OpenLearning) a tie, ktoré slúžia ako podpora pre vzdelávanie vrámci nejakej inštitúcie (Moodle, Google Classroom).

Platformy z prvej kategórie boli typické tým, že obsahovali iba čiastočnú bezplatnú verziu pre tvorcov kurzov, ale pre študentov boli spoplatnené čo nespĺňa primárny cieľ práce a funkčnú požiadavku 17. Častokrát boli komplexnejšie a obsahovali funkcionality nad rámec nášho zadania, ako je marketing, štruktúrovaný prehľad progresu študentov a iné. Počas prieskumu sme narazili na mnoho ďalších podobných platforiem s rovnakým účelom ako sú napríklad Coursera, Udemy, Skillshare a mnoho ďalších.

Druhá kategória spomínaných aplikácií sa zameriavala primárne na univerzity alebo iné vzdelávacie inštitúcie, ktorých cieľom bolo poskytnúť obsah pre svojich študentov. Zvyčajne sa vyžadovalo prihlásenie do platformy (nespĺňa požiadavku 16) alebo iné spôsoby overenia prístupu ku vzdelávaciemu obsahu. Spomínané aplikácie boli robustné, vytváranie obsahu zaberalo množstvo času a bolo technicky náročnejšie. Na druhú stranu poskytovali možnosti komunikácie a podporu pre testovanie znalostí.

Žiadne z uvedených a rozobratých riešení neposkytovalo splnenie všetkých podstatných požiadaviek špecifikovaných v predošlých sekciách.

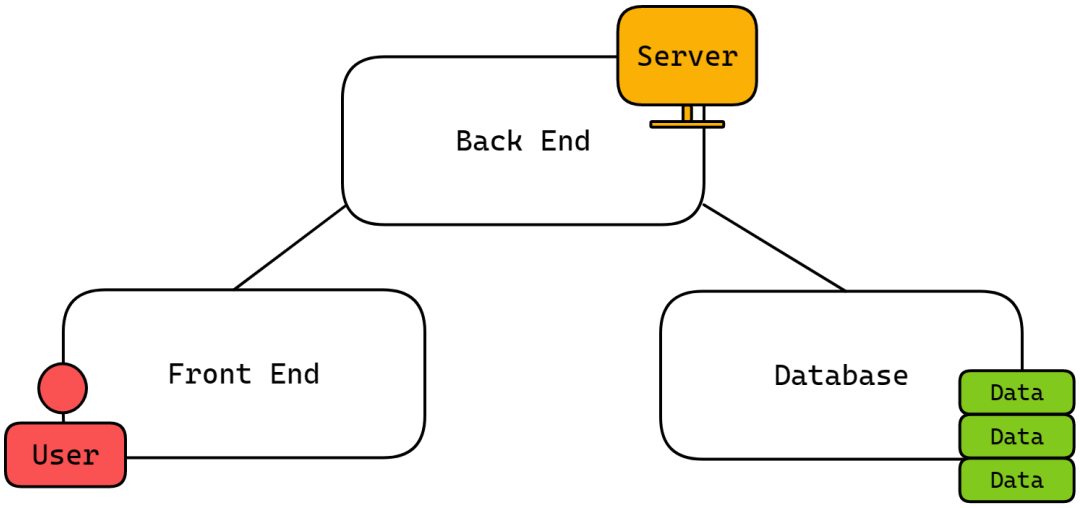
1.6 Ciele práce

Na záver prvej kapitoly si zhrnieme ciele bakalárskej práce na základe predošlých sekcií. Ciele sú nasledovné:

1. Vytvorenie webovej aplikácie pre online vzdelávanie.
2. Používanie technológií, ktoré sú zadarmo.
3. Vytvorenie prívetivého užívateľského rozhrania, ktoré je jednoduché a intuitívne.
4. Splnenie zoznamu funkčných požiadaviek (sekcia 1.4)

2. Analýza technických riešení

Vytváranie webovej aplikácie typicky pozostáva z návrhu užívateľského rozhrania, programovania a správy databázy [5]. V tejto kapitole podrobne rozoberieme každú časť, bližšie sa pozrieme na to aké sú možnosti pri výbere programovacieho jazyka a frameworku, aké databázy sú k dispozícii vzhľadom na naše požiadavky a podobne.



Obr. 2.1: Časti webového vývoja

Pri tvorbe webového rozhrania je zaužívané rozdelenie na tri základne časti (viď obrázok 2.1), ktoré so sebou vzájomne komunikujú:

- *Databáza* je softvér, ktorý sa používa na ukladanie a správu údajov. Existuje niekoľko typov databáz, najpoužívanejšie sú grafové, relačné a NoSQL. Typ použitej databázy závisí od požiadaviek aplikácie.
- *Frontend* je súčasťou vývoja webu na strane klienta. Zahŕňa vytvorenie používateľského rozhrania a logiky, ktorá beží v prehliadači. Vývojári frontendu používajú HTML, CSS a JavaScript alebo pokročilejšie frameworky ako napríklad React, Angular alebo Vue na vytváranie interaktívnych webových stránok. Interagujú s backendom pomocou rozhraní API.
- *Backend* sa stará o logiku a funkčnosť aplikácie, ako napríklad riadenie autentifikácie a autorizácie užívateľov, spracovávanie požiadaviek a pripojenie k databázam. Je zodpovedný za bezpečnosť, presnosť a aktuálnosť údajov a informácií poskytnutých pre frontend.

2.1 Alternatíva ku klasickému backendu

Pri klasickom vývoji backendu sa kód na strane servera píše pomocou PHP, Pythonu, ASP.NET, Ruby alebo Node.js. Server spracováva požiadavky od klienta a odosiela odpovede. K dátam pristupuje pomocou SQL dotazov.

Okrem vývoja klasického backendu a jeho napojenia na databázu poznáme aj prístup pomocou **Backend as a Service** (BaaS), čo je typ cloudovej služby, ktorá umožňuje vývojárom vytvárať webové a mobilné aplikácie bez potreby vlastného backendu. BaaS poskytuje sadu funkcií a nástrojov pre prepojenie s poskytovanou databázou, ktoré uľahčujú vývoj a správu aplikácií a zároveň poskytuje aj integrovanú funkcionalitu autentifikácie.

Výhody používania BaaS spočívajú v úspore času a peňazí, rýchlom získaní určitej funkcionality, škálovateľnosti infraštruktúry a knižnice na prenos dát medzi klientom a serverom. Ďalšou výhodou takto spracovaného backendu a databázy je možnosť zamerať sa primárne na frontend a jeho funkcionalitu pri interakcii s užívateľom a backend nechať na poskytovateľovi a využívať jeho hotové riešenia. Príkladom BaaS je Firebase, AWS Amplify a Parse.

Vývoj pomocou BaaS však nie je vhodný za každých okolností. Hodí sa primárne na menšie projekty, alebo tie, ktoré nemajú veľkú základňu backend developerov a nepotrebujú riešiť špecifickú funkcionalitu na strane servera. Ak sa očakáva, že projekt bude rýchlo rásť, môžu nastať zvýšené poplatky za využívanie tejto služby vo väčšom rozsahu.

Čo sa týka nášho projektu, z hľadiska backendu potrebujeme pokryť iba funkcionalitu autentifikácie a prácu s databázou a inak je všetko ostatné možné riadiť na frontende.

2.2 Voľba frontendového frameworku

Určenie vhodného frameworku pre vývoj frontendu webových aplikácií môže byť zložitý proces. Existuje veľa možností a každý framework má svoje výhody aj nevýhody, preto treba zvážiť všetky koncepty, ktoré sú dôležité pri výbere. Medzi tie najhlavnejšie patria jazyk, výkon, bezpečnosť, flexibilita a komunita.

Na vytváranie webových aplikácií je najbežnejším jazykom JavaScript, ktorý je stále populárnou voľbou. Existuje niekoľko frameworkov postavených na JavaScripte, napríklad Angular, React a Vue. Každý ponúka rôzne výhody a nevýhody a výber do značnej miery závisí od konkrétnych potrieb projektu.

2.2.1 Frameworky založené na Javascripte

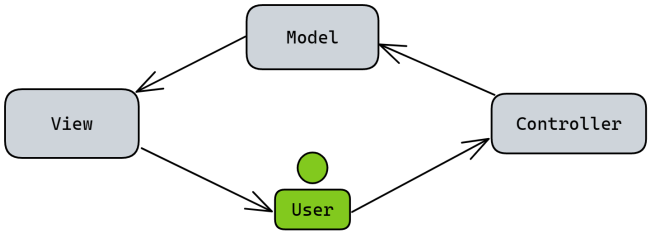
React[1] je open-source framework vyvíjaný spoločnosťou Facebook. Jeho hlavným zameraním je tvorba užívateľského rozhrania (UI) pre webové aplikácie. V Reacte sa používa tzv. *komponentový prístup*, kde každá časť UI je reprezentovaná komponentom. Tento prístup umožňuje jednoduchú a efektívnu tvorbu a správu webových aplikácií.

Angular[2] je open-source framework vyvíjaný spoločnosťou Google. Podobne ako React, aj Angular sa zameriava na tvorbu webových aplikácií, no jeho prí-

[1] https://react.dev/
[2] https://angular.io/

stup je trochu odlišný. Angular používa tzv. *model-view-controller* (MVC) architektúru, ktorú môžete vidieť na obrázku 2.2, kde model predstavuje dáta, view reprezentuje UI a controller riadi interakciu medzi modelom a view. V Angulari sa používa výlučne *TypeScript* – nadmnožina JavaScriptu, čo zabezpečuje statickú typovú kontrolu a robí kód menej chybovým, využíva statické aj dynamické typovanie a IntelliSense. Okrem toho je ľahšie debugovateľný, identifikuje problémy počas kompilácie a dá sa jednoducho refaktorovať. Na druhej strane je Angular robustnejší a menej výkonný pri optimalizácii vykresľovania DOM (Document Object Model) elementov.

Vue.js[3] je ďalší open-source framework pre vývoj webových aplikácií. Vue.js sa zameriava na jednoduchosť a flexibilitu. Jeho prístup je podobný ako v Reacte – používajú sa komponenty a virtuálny DOM, čo zabezpečuje rýchlosť a efektivitu. Vue.js je považovaný za ľahší a jednoduchší na naučenie sa v porovnaní s Angularom, avšak oproti Reactu je menej populárny [6].



Obr. 2.2: Model View Controller

2.2.2 React

Autorka tejto práce je názoru, že aplikáciu by bolo možne vytvoriť v každom zo spomínaných frameworkov. Po zvážení možností a požiadaviek sme však vybrali React, keďže je najpopulárnejší [6] a jeden z najrozšírenejších frameworkov pre vývoj webových aplikácií. Jeho výhodou je komponentový prístup, vďaka ktorému vieme opakovane použiť komponenty naprieč celou aplikáciou, čo nám umožní písať lepšie udržateľný kód.

Na základe predošlej analýzy a výhod využitia Typescriptu v Angulari, sme sa rozhodli ho použiť namiesto JavaScriptu, keďže sa jedná o staticky typovaný jazyk, čo prinesie výhody spomenuté v predošlej sekcii a zaručí bezpečnejší kód.

[3]https://vuejs.org/

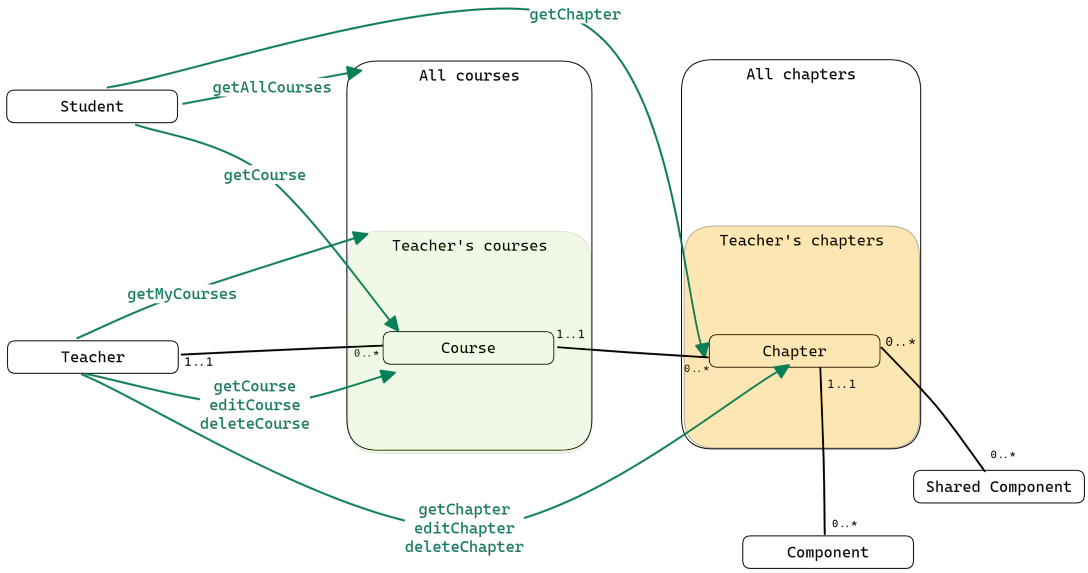
2.3 Dátový model

Na základe analýzy požiadaviek (sekcia 1.4) sme vytvorili konceptuálny model, ktorý zobrazuje základne zaznačené entity a vzťahy medzi nimi.

2.3.1 Model

Ak sa pozrieme na obrázok 2.3, môžeme vidieť užívateľov (Učiteľ a Študent), ktorí pracujú s kolekciou kurzov a kapitol. Kolekcia kurzov obsahuje zároveň aj subkolekciu kurzov, ktoré vytvoril špecifický Učiteľ (rovnako aj v prípade kapitol). Učiteľ vie vytvárať, editovať a mazať kurzy a kapitoly, ktoré vytvoril. Študent vie pristupovať ku všetkým kurzom, zároveň si vie otvoriť konkrétny kurz a kapitoly v ňom. Tieto interakcie sú na obrázku zaznamenané zelenou farbou.

Zjednodušené vzťahy medzi základnými inštanciami sú označené čiernou farbou a je k nim zaznačená ich arita. Čitateľ si môže všimnúť, že väčšina vzťahov má aritu (1..1) a (0..*), čo v konkrétnom príklade znamená, že učiteľ môže mať 0 alebo nekonečne veľa kurzov, ktoré vytvoril, ale kurz nemôže existovať bez toho aby bol priradený nejakému učiteľovi. Zaujímavý vzťah predstavuje prepojenie medzi *kapitolou* a *zdieľaným komponentom* – kde sa jeden zdielaný komponent môže nachádzať vo viacerých alebo žiadnej kapitole.



Obr. 2.3: Konceptuálny model aplikácie

2.3.2 Špecifické obmedzenia na dátový model

Na obrázku 2.3 v predošlej časti sme vytvorili iba základnú predstavu o tom ako by dáta mali byť medzi sebou prepojené a nerozoberali sme to viac do hĺbky. Avšak v tejto podsekcii sa pozrieme na detaily, ktoré je nutné si uvedomiť počas analýzy a neskôr pri výbere konkrétneho typu databázy.

Nijako sme nešpecifikovali ako presne budú komponenty v kapitole uložené, ale na základe prvej kapitoly tejto bakalárskej práce (sekcia 1.1) vieme, že budú mať rôzne typy a zároveň sa budú deliť na zdieľané a tie, ktoré patria iba jednej

kapitole. Zároveň musia byť tieto komponenty previazané s ich autorom, aby sme ich na základe toho vedeli filtrovať a zobrazovať užívateľovi. Je teda potrebné si uvedomiť túto komplikáciu a myslieť na ňu pri výbere vhodnej reprezentácie.

Ďalší detail, na ktorý je potrebné nezabúdať je možnosť vytvárania šablóny hlasovania a jej použite v jednotlivých kapitolách. Hlasovanie však samo o sebe nie je zdieľaný komponent, keďže musí obsahovať záznam o odpovediach pre každú jeho inštanciu samostatne. Z čoho nám vyplýva potreba ukladať si šablóny hlasovaní cez užívateľa, ktorý ich vytvoril.

Celkovo teda potrebujeme vedieť komponenty rozdeliť podľa typov, spojiť s kapitolou, zároveň potrebujeme vytvoriť podporu pre zdieľané komponenty a prepojiť ich s ich autorom a podporovať možnosť vytvárania hlasovacích šablón a ich konkrétnych inštancií a výsledkov v kapitole.

2.3.3 Dotazy

Na základe analýzy požiadaviek zo sekcie 1.4 nám vyplynuli tieto typické dotazy:

- Zobraz všetky kurzy
- Zobraz všetky kapitoly pre kurz s id ID
- Zobraz konkrétnu kapitolu s id ID a jej komponenty
- Zobraz všetky kurzy učiteľa s id ID
- Zobraz všetky zdieľané komponenty daného typu učiteľa s id ID

a bezpečnostné opatrenia:

- Učiteľ môže editovať a mazať len prvky, ktoré vytvoril
- Učiteľ môže používať len zdieľané komponenty, ktoré vytvoril

2.4 Databáza a Backend

Na základe dátového modelu sa pokúsime vytvoriť konkrétny návrh reprezentácie dát v rôznych druhoch databáz aby sme zistili, ktorá bude najviac vyhovovať. Pozrieme sa aj na potreby v kontexte backendu a zanalyzujeme jeho výber.

2.4.1 Relačná

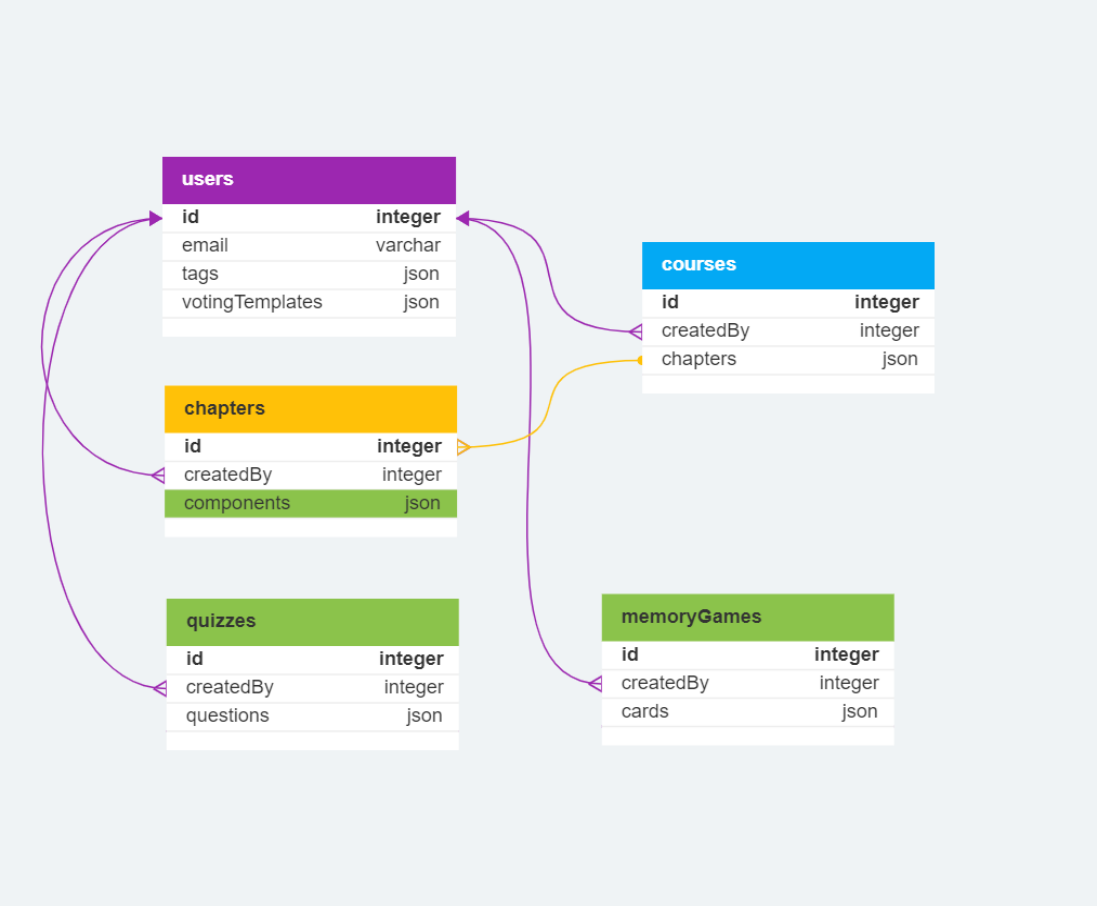
Relačné databázy sú tradičným typom databázového systému, ktorý používa štruktúrované tabuľky. Dáta v relačných databázach sú usporiadané do riadkov a stĺpcov, kde každý stĺpec má svoj dátový typ a každý riadok reprezentuje konkrétny záznam.

Pre náš projekt by sme potrebovali vytvoriť samostatnú tabuľku pre každý typ komponentu. Ak by sme chceli zaručiť previazanie komponentu s kapitolou (viď obmedzenia v sekcii 2.3.2), museli by sme každému nezdieľanému komponentu priradiť ID kapitoly. Pre zdieľané komponenty by sme museli vytvoriť ešte

samostatné tabuľky pre prepojenie komponent - kapitola. Komplikácia by bola v tom, že by sme mali celkovo 6 tabuliek pre komponenty podľa typu a dve ďalšie určujúce prepojenie zdieľaných komponentov. Ak by sme sa potom chceli dotázať na všetky komponenty v jednej kapitole (podľa typických dotazov v časti 2.3.3), museli by sme prechádzať všetky tieto tabuľky a hľadať správny cudzí kľúč. Z týchto dôvodov je relačný typ databázy pre naše požiadavky nevhodný.

2.4.2 Relačná s využitím JSON

Rozšírením relačnej databázy o použitie JSONu ako dátového typu by sme vedeli znížiť negatívne aspekty obmedzení relačných databáz. Komponenty by sme vedeli reprezentovať ako JSON objekty, ktoré nemusia mať preddefinovanú schému. Ako môžeme vidieť na obrázku 2.4, všetky potrebné zložitejšie typy by sme reprezentovali ako json. Na obrázku sa nám podarilo zaznačiť vzťahy medzi kurzami a kapitolami, ktoré budú reprezentované ako JSON array (budú obsahovať id kapitol). Takisto by sme komponenty reprezentovali podľa potreby – buď ako konkrétne dáta alebo ako cudzie kľúče zdieľaných komponentov, čím by sme vyriešili všetky obmedzenia zo sekcie 2.3.2.



Obr. 2.4: návrh v relačnej databáze s využitím JSONu

2.4.3 NoSQL

NoSQL databázy sú alternatívou k tradičným relačným databázam a sú navrhnuté na ukladanie a spracovanie veľkých objemov neštrukturovaných alebo menej štruktúrovaných dát (je možné ich ukladať bez potreby definície presnej schémy). Dáta su typicky uložené v dokumentoch, key-value pároch, stĺpcoch alebo grafoch.

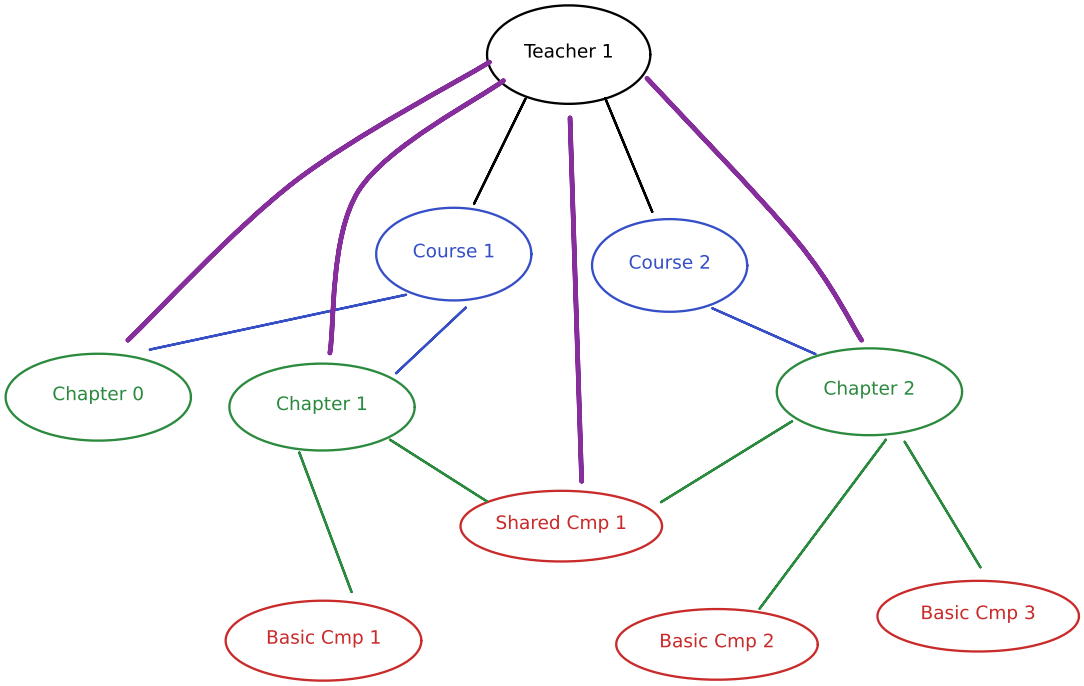
Pre potreby nášho projektu by sme vytvorili 5 kolekcií (učitelia, kurzy, kapitoly, pexesá a kvízy), kde by každý dokument v kolekcii obsahoval všetky potrebné dáta a vzťahy medzi nimi by boli reprezentované pomocou ID. Keďže NoSQL nepotrebuje presnú databázovú schému, kapitola by obsahovala komponenty, kde by boli uložené buď dáta nezdieľaných komponentov alebo referencie (id) na zdieľané komponenty. Každý komponent by obsahoval aj svoj typ, aby sme v prípade zdieľaných komponentov vedeli, v ktorej kolekcii hľadať.

```
//Teachers collection
Teacher {
  id: string,
  email: string,
  voting_templates : VotingTemplate [ ] // voting
     template objects
}
//Courses collection
Course  {
  id: string,
  title: string,
  chapters: string [ ], //chapter ids
  createdBy: string; // teacher id
};
//Chapters collection
Chapter  {
  id: string;
  title: string;
  createdBy: string; // teacher_id
  components: Component [ ]; //paragraph, video, image,
     voting or memory game / quiz ID
};
//MemoryGames collection
MemoryGame {
  id: string;
  createdBy: string; // teacher id
  cards: string [ ],
};
//Quizzes collection
Quiz {
  id: string;
  createdBy: string; // teacher id
  questions: string [ ],
};
```

2.4.4 Grafová

Grafové databázy sa špecializujú na ukladanie a spracovanie dát v grafovom modeli, čo znamená, že dáta sú reprezentované ako vrcholy a hrany, kde vrcholy predstavujú entity a hrany reprezentujú vzťahy medzi nimi.

Pre náš projekt by sme vytvorili 4 typy uzlov (učiteľ, kurz, kapitola, komponent), avšak pre reprezentáciu vzťahov by sme museli mať množstvo hrán, ktoré by ich spájali a komplikácia by nastala keď potrebujeme spojiť uzly (kapitoly, komponenty) s ich autorom – učiteľom (viď fialové hrany na obrázku 2.5), aby sme vedeli použiť jednoduchý dotaz *zoznam všetkých kapitol, ktoré vytvoril užívateľ s ID* zo sekcie 2.3.3.



Obr. 2.5: návrh v grafovej databáze

2.4.5 Autentifikácia

Ako sme stanovili v sekcii 1.4 aplikácia potrebuje mať integrovanú funkcionalitu autentifikácie. Na základe tejto požiadavky nám v sekcii 2.3.3, vyplynuli bezpečnostné opatrenia.

Autentifikácia sa zvyčajne implementuje pomocou užívateľského mena a hesla a ich overenie má na zodpovednosti backend, následne po overení akreditácie má užívateľ prístup k obsahu, ktorý vytvoril. Autentifikácia sa môže nachádzať ako ponúkaná funkcionalita spomínaných BaaS služieb v sekcii 2.1.

2.4.6 Záver

Ako sme stanovili v úvodnej kapitole, aplikácia potrebuje mať databázu pre správu registrovaných užívateľov – učiteľov, a vzdelávacieho obsahu. Pri výbere

vhodnej technológie je potrebné dbať predovšetkým na štruktúru dát a poskytovanú funkcionalitu, ktorá zahŕňa možnosť autentifikácie. Na základe predošlej analýzy je vidieť, že dáta sa dajú reprezentovať relačne s využitím JSONu aj pomocou NoSQL. Výhodou použitia NoSQL v kontexte našej práce je väčšia variácia typov, ako napríklad uloženie tagov ako zoznam stringov alebo uloženie odkazov na kapitoly ako zoznam id.

Na základe doterajšej analýzy a požiadaviek (sekcia 1.4) sme zhodnotili, že vývoj samostatného backendu nie je nevyhnutný a preto sme sa rozhodli pre využitie BaaS (spomínanú v úvode tejto kapitoly, sekcia 2.1), ktorá zabezpečí potrebnú funkcionalitu (prepojenie s databázou a autentifikáciu) bez nutnosti vývoja vlastného backendu.

V ďalšej kapitole teda budeme analyzovať možnosti výberu BaaS služby, ktorá ponúka NoSQL alebo relačnú databázu s JSON rozšírením.

2.5 BaaS

Definíciu BaaS sme spomenuli na začiatku kapitoly v sekcii 2.1. Medzi hlavnými požiadavkami aplikácie, bola rola Učiteľa a s ním spojená funkcionalita prihlásenia sa (pomocou univerzitného Microsoft konta) do účtu. Pri výbere služby bolo teda kľúčové vyberať tak, aby bola integrácia tejto funkcionality zahrnutá.

2.5.1 Firebase

Firebase[4] je cloudová platforma od spoločnosti Google, poskytujúca rôzne nástroje pre vývoj webových a mobilných aplikácií. Jedným z jej najväčších benefitov je poskytnutá databáza a autentifikácia, ktorá umožňuje používateľom prihlásiť sa pomocou emailu, hesla, telefónneho čísla a ďalšieho [7]. Zároveň má podporu pre autentifikáciu pomocou externých poskytovateľov, ako napríklad Google, Facebook, Twitter a pre nás dôležitý – Microsoft, v ktorom sú vytvorené univerzitné kontá zamestnancov MFF UK. Ďalšou výhodou reflektujúcou požiadavky na bezpečnostné opatrenia zo sekcie 2.3.3 je funkcionalita nastavenia vlastných *security rules*, ktoré vedia obmedziť možnosť read a write operácií na základe stanovených kritérií. Spomínaná integrovaná aplikácia je veľmi jednoduchá na používanie, obsahuje množstvo užitočných funkcií a NoSQL databázu Firestore.

K nevýhodám patrí to, že Firebase je zdarma iba pre určité množstvo operácií a ďalej je spoplatnená podľa cenníka [8].

Firebase má k dispozícii 50 000 read, 20 000 write a 30 000 delete operácií denne a 50 000 autentifikovaných užívateľov mesačne bez poplatku. V prípade našej aplikácie a dát uložených podľa modelu z predošlej sekcie si vieme prepočítať približnú cenu za túto službu.

Na základe diskusií so zadávateľom sme vyhodnotili približné počty užívateľov a kurzov aj s určitou rezervou. Ak očakávame 500 užívateľov denne a v aplikácii máme 30 kurzov kde každý kurz má priemerne 5 kapitol a každá má priemerne 2 zdieľané komponenty (pexeso a kvíz), tak ak si chce každý užívateľ prečítať jednu kapitolu denne, to predstavuje
*počUžívateľov * (počKurzov + počKapitol + počZdieľanýchKomponentov)*,

[4] https://firebase.google.com/

čiže v našom príklade je to *500 * (30 + 5 + 2) = 18 500* read operácií denne. Write a delete operácií bude násobne menej, keďže sa očakáva menší počet učiteľov – tvorcov kurzov a aj to, že obsah nebudú pridávať každý deň.

Na základe výpočtu vieme posúdiť, že počet očakávaných operácií je menší ako limity nastavené Firebase a ešte obsahuje aj dostatočnú rezervu, takže by nemali nastať žiadne poplatky za využívanie tejto služby v stanovenom rozsahu.

2.5.2 AWS

AWS[5] je databázová služba vyvinutá spoločnosťou Amazon, ktorá okrem iného ponúka aj NoSQL databázu DynamoDB. Vyznačuje sa širokou škálou produktov, ktoré ponúka. AWS poskytuje službu s názvom AWS Identity and Access Management (IAM), ktorá umožňuje nastaviť bezpečnostné opatrenia a spravovať tak riadenie prístupu. Integrácia týchto pravidiel má oproti Firebase inú implementáciu a syntax, ale ich funkčnosť je ekvivalentná.

V porovnaní s Firebase je však AWS komplexnejšia, má strmšiu krivku učenia a práca s ňou je časovo náročná. Takisto ponúka len obmedzený bezplatný program, ktorý ale nie je presne špecifikovaný, preto nebolo možné poskytnúť odhad.

2.5.3 Záver

V prípade nášho projektu sme sa rozhodli pre Firebase a jej NoSQL databázu (ktorá spĺňuje požiadavky a je najvhodnejšou na základe analýzy v sekcii 2.4) z dôvodov ako jednoduchosť používania, pozitívna predošlá skúsenosť, priateľské rozhranie a poskytnutá autentifikácia, ktorá spĺňa kľúčovú požiadavku a zároveň, ako sme ukázali vyššie, by nemala byť spoplatnená.

2.6 Voľba dizajnového systému

Pre projekt bakalárskej práce sme potrebovali vybrať dizajnový systém, ktorý by spĺňal požiadavky projektu (sekcia 1.4) a pomohol vytvoriť úspešné používateľské rozhranie pre náš produkt. Rozhodnutie, ktorý dizajnový systém si vybrať, nebolo jednoduché, pretože existuje veľa možností, z ktorých každá má svoje silné a slabé stránky. Museli sme starostlivo zhodnotiť dostupné možnosti a vybrať dizajnový systém, ktorý by najlepšie vyhovoval potrebám projektu a zodpovedal jeho cieľom.

2.6.1 Prečo je dizajnový systém dôležitý

Dizajnový systém je súbor princípov, smerníc a komponentov, ktoré sa používajú konzistentne v celom produkte alebo organizácii. Zahŕňa kolekciu vizuálnych a interaktívnych prvkov, ako je typografia, farebné palety, ikony, tlačidlá, formuláre a ďalšie, ktoré možno opätovne použiť a kombinovať na vytvorenie konzistentných a súdržných používateľských rozhraní [9]. Tieto pravidlá sa špecializujú primárne pre stolné počítače, tablety a mobilné zariadenia.

[5]https://aws.amazon.com

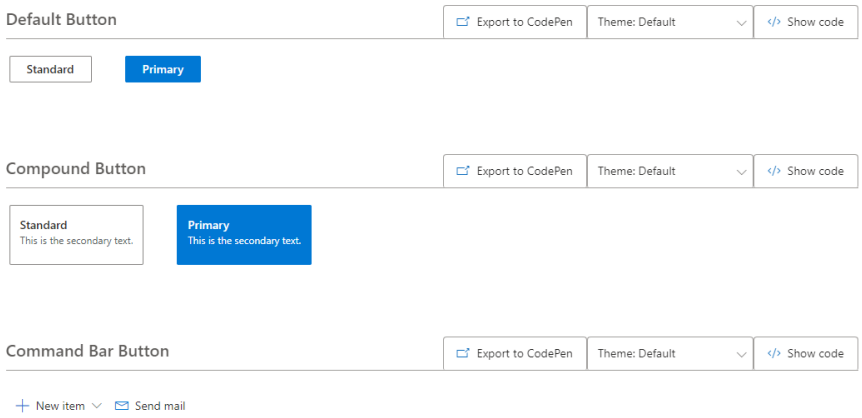
Výber dizajnového systému je podstatnou časťou moderného produktového vývoja, jeho benefit je v poskytovaní dizajnu, ktorý je konzistentný a efektívny na všetkých platformách a zariadeniach. Má zásadný dopad na obľúbenosť a celkový úspech produktu. Okrem iného tým, že poskytuje sadu preddefinovaných dizajnových vzorov a komponentov, urýchľuje proces navrhovania a vývoja [10]. V tejto sekcii sa teda zameriame na porovnanie a výber vhodného dizajnového systému spĺňajúceho požiadavky nášho projektu.

2.6.2 Fluent Design System

Fluent Design System [11] bol vyvinutý spoločnosťou Microsoft. Je známy z platforiem Windows, Xbox alebo programov balíka Micsrosoft Office.

Medzi výhody systému patrí jeho konzistencia – poskytuje súbor princípov dizajnu a vizuálnych prvkov, ktoré možno použiť na rôznych platformách a aplikáciách spoločnosti Microsoft, čo umožňuje súdržnú používateľskú skúsenosť.

Zatiaľ čo systém Fluent Design System poskytuje sadu preddefinovaných dizajnových prvkov (viď dizajn tlačidiel podľa Fluent Design systému na obrázku 2.6), prispôsobenie nad rámec týchto možností si môže vyžadovať ďalšie úsilie a odborné znalosti.

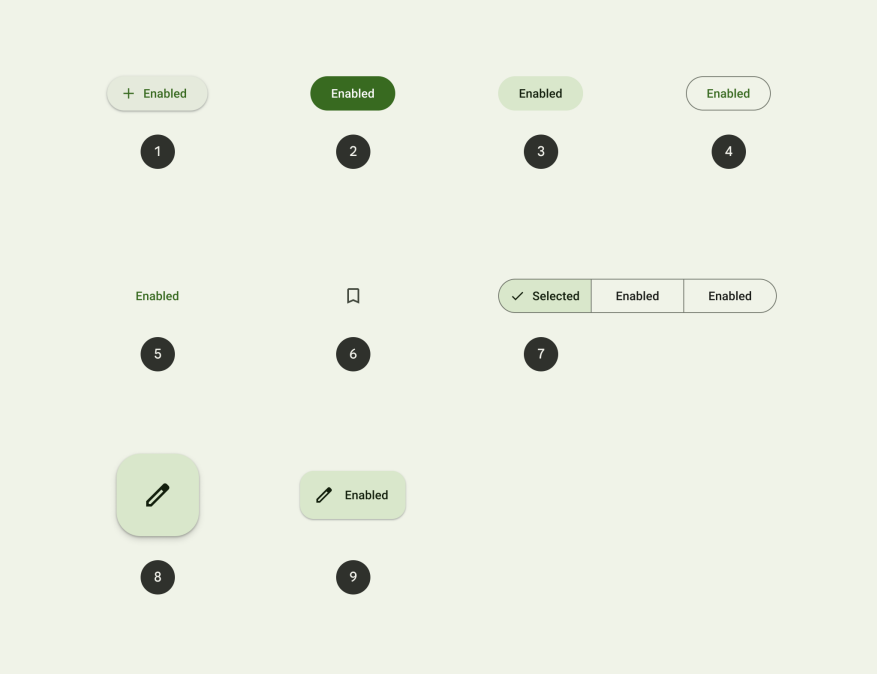


Obr. 2.6: screenshot návrhu tlačidiel Fluent UI

2.6.3 Material Design

Material Design [12] bol vyvinutý spoločnosťou Google v roku 2014 a aktuálne je v najnovšej verzii 3. Jednou z jeho výhod je vynikajúca dokumentácia a tzv. *guidelines*, ktoré poskytujú dizajnérom súbor odporúčaní, ako si vytvoriť vlastný konzistentný dizajn podľa stanovených pravidiel. Tieto odporúčania sú dôkladne prepracované a umožňujú vytvoriť výnimočný dizajn, ktorý bude zároveň úspešný na trhu. V súčasnosti je Material Design najpoužívanejším [13] dizajnovým systémom, má prepracované UI frameworky a autorka s ním má skúsenosti, preto sme sa rozhodli ho použiť pre účely tohto projektu.

Na obrázku 2.7 si môže čitateľ pozrieť výzor tlačidiel podľa Material Design a porovnať s predchádzajúcim obrázkom 2.6.

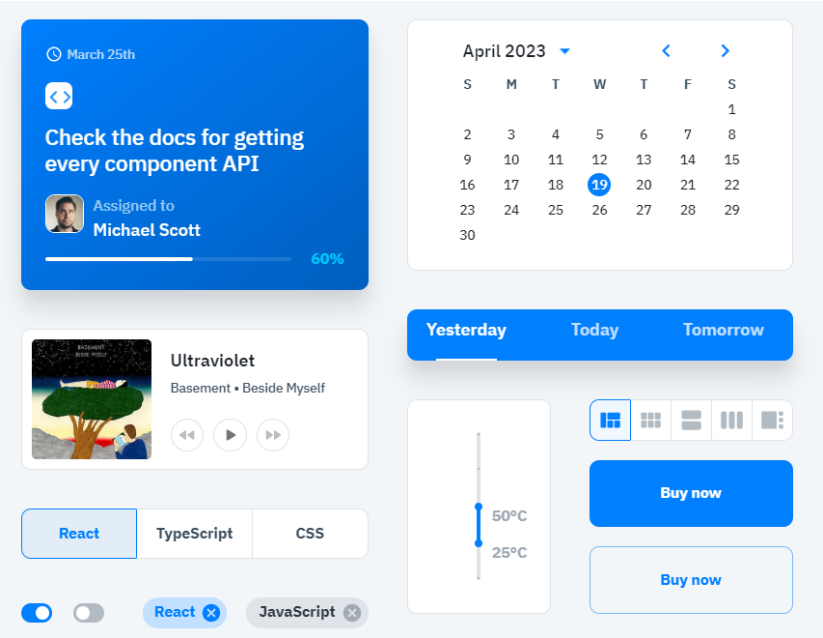


Obr. 2.7: screenshot návrhu tlačidiel Material

2.7 UI framework

Ako sme zmienili v predošlých kapitolách, rozhodli sme sa pre kombináciu Reactu a Material Designu. Preto bolo potrebné si zvoliť UI knižnicu, ktorá umožňuje ich kombináciu. Jasnou voľbou sa teda stala knižnica MUI[6], ktorá patrí medzi najpoužívanejšie [14] a obsahuje Reactovské komponenty navrhnuté podľa pravidiel Material Designu (viď obrázok 2.8). Knižnica okrem iného umožňuje aj predefinovanie si celkovej témy aplikácie, nastavenia vlastnej farebnej schémy, typografie a vlastných štýlov komponentov.

[6]https://mui.com/



Obr. 2.8: Screenshot komponentov knižnice MUI

3. Užívateľské rozhranie

V sekcii 2.6 sme spomínali dôležitosť správneho výberu dizajnového systému a jeho vplyv na celkový úspech produktu. V tejto kapitole si podrobne rozoberieme návrh užívateľského rozhrania, budeme analyzovať výber farebnej schémy a typografie, návrh konkrétnych komponentov a ich spracovanie do obrazoviek.

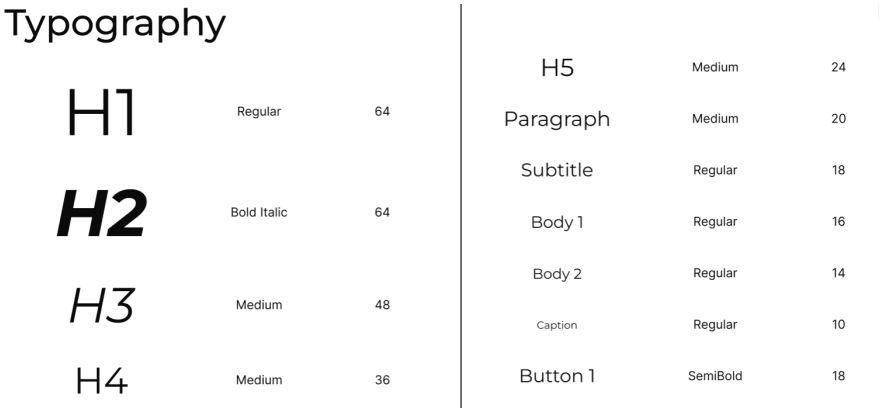
Návrh užívateľského rozhrania bol robený vo Figme[1], nakoľko je to populárny bezplatný nástroj pre vytváranie UI a autorka práce s ním má predošlú pozitívnu skúsenosť.

3.1 Farebná schéma a typografia

Pri návrhu užívateľského rozhrania je dôležité zvoliť vhodnú farebnú schému a typografiu. Farebná schéma by mala byť zladená s cieľovou skupinou a povahou produktu. Farby môžu byť použité na zdôraznenie niektorých častí rozhrania, vytvorenie hierarchie a emocionálneho pôsobenia na používateľa. Schéma, ktorú sme zvolili (viď obrázok 3.1), obsahuje jednoduché kombinácie farieb, ktoré nie sú nijako výstredné a neodvádzajú pozornosť od učenia.

Obr. 3.1: screenshot farebnej schémy projektu

Typografia je ďalším dôležitým prvkom, ktorý môže ovplyvniť čitateľnosť a celkovú estetiku produktu. V našej práci používame font Montserrat (viď obrázok 3.2), nakoľko má čistý a moderný dizajn, je open-source a ľahko čitateľný aj v menších veľkostiach. Medzi ďalšie jeho výhody patrí to, že podporuje množstvo jazykov – medzi nimi aj češtinu, ktorá sa v aplikácií používa.

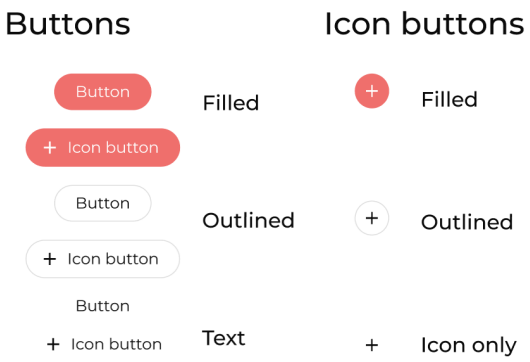


Obr. 3.2: screenshot typografie použitej v projekte

[1]https://www.figma.com/

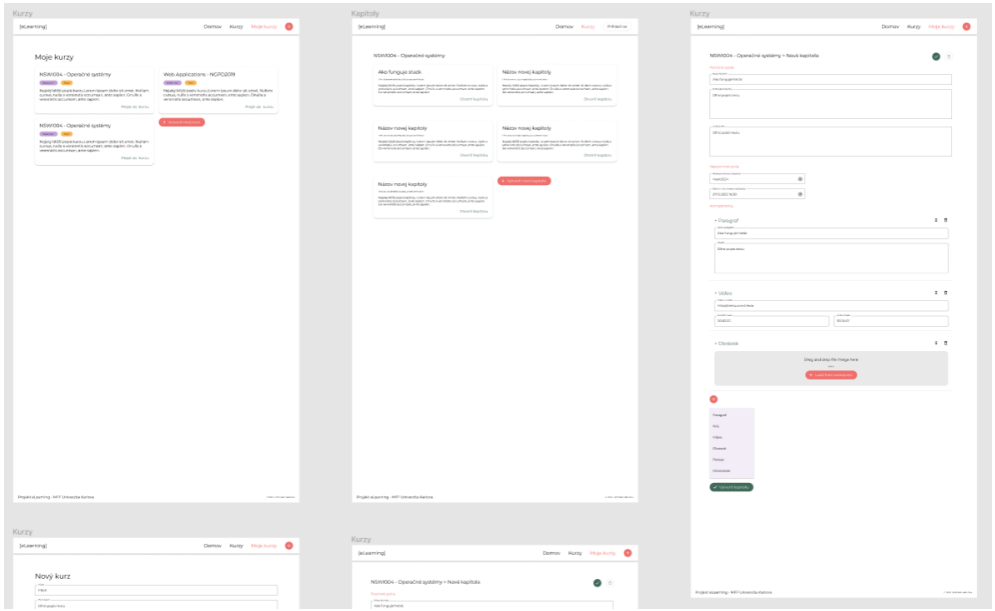
3.2 Komponenty a obrazovky

Ako sme rozoberali v sekcii 2.6, rozhodli sme sa pre použitie Material Designu a jeho pravidiel pre tvorbu komponentov[15]. Na obrázku 3.3 je zobrazený návrh komponentov – tlačidiel pre našu aplikáciu vo Figme.



Obr. 3.3: screenshot návrhu tlačidiel

Pri návrhu konkrétnych obrazoviek aplikácie sme sa zamerali na jednoduchosť a zaužívaný dizajn. Vytvorili sme obrazovky (viď príklad na obrázku 3.4), ktoré obsahujú rovnaké komponenty, ako napríklad navigačnú lištu, tlačidlá, klikateľné karty a ďalšie. Naprieč celou aplikáciou sme sa snažili dodržiavať správny *user flow*, čo zaručuje intuitívne používanie a prezentovanie správnych informácií v správnom čase. Týmto spôsobom môžu užívatelia dokončiť požadovanú úlohu s minimálnym počtom krokov. Celý návrh UI je možné si pozrieť v elektronickej prílohe.



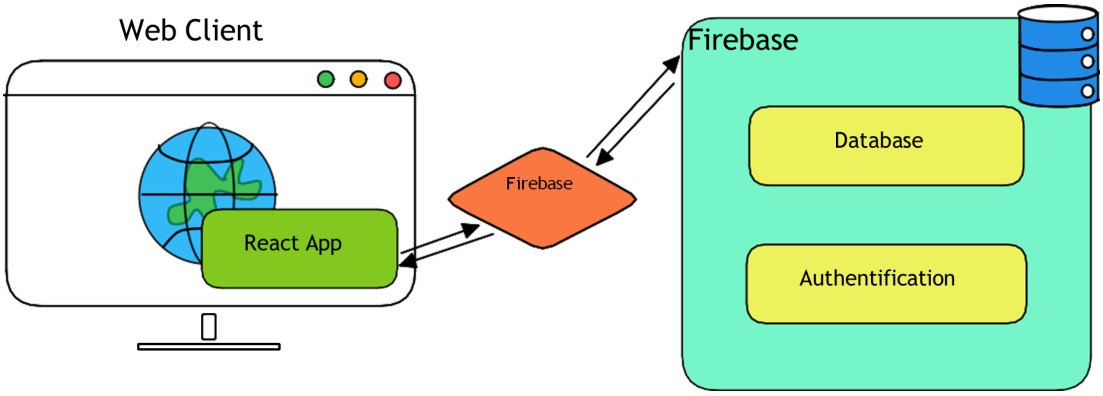
Obr. 3.4: screenshot návrhu obrazoviek v programe Figma

4. Vývojová dokumentácia

V tejto kapitole si bližšie popíšeme vývoj aplikácie, pozrieme sa na konkrétne riešenie databázového modelu a autentifikáciu a ich prepojene s webom. Podrobne rozoberieme každú časť vývoja.

4.1 Architektúra aplikácie

Ako sme uviedli v analýze v kapitole 2, využívame React framework a TypeScript pre vývoj frontendu. Hoci nemáme tradičný backend, používame Firebase databázu a jej možnosť autentifikácie. Náčrt architektúry a vzájomnej komunikácie jednotlivých jej častí môžeme vidieť na obrázku 4.1.



Obr. 4.1: Architektúra aplikácie

Frontend vyvinutý v Reacte umožňuje vytváranie opakovane použiteľných UI komponentov, čím podporuje modularitu a udržateľnosť kódu. Komponenty sú napísané v TypeScripte, staticky typovanej nadmnožine JavaScriptu, ktorá poskytuje rozšírenú kontrolu typu a lepšiu kvalitu kódu. Riadenie frontendu je primárne zabezpečované pomocou zabudovaných reactovských konceptov (Props, States).

Integrácia Firebase s aplikáciou je sprostredkovaná pomocou typického prepojenia – *Firebase SDK* a je riadená na strane klienta. Komponenty na frontende môžu žiadať o načítanie údajov z databázy alebo vykonávať CRUD operácie (Create, Read, Update, Delete). Funkciu overenia totožnosti zabezpečuje Firebase a umožňuje tak užívateľom sa zaregistrovať (napríklad pomocou konta Microsoft).

4.2 Architektúra databázy

V analýze tejto bakalárskej práce v sekcii 2.3 sme na základe požiadaviek vytvorili jednoduchý konceptuálny model. Neskôr v sekcii 2.4 sme zhodnotili, že pre aplikáciu je vhodný relačný aj noSQL typ databázy a rozhodli sme sa pre implementáciu vo Firebase a jej noSQL Firestore databáze, ktorá je tvorená pomocou kolekcií a dokumentov.

V tejto sekcii sa budeme odvolávať na obrázok návrhu našej databázy 4.2. Keďže využívame NoSQL databázu, ktorá nevynucuje presnú schému, pomocou

security rules, Typescriptu a dátového modelu, rozobraného neskôr, zaručíme, že sa v databáze budú nachádzať len dáta správneho typu a štruktúry. V databáze máme 5 kolekcií - užívatelia, kurzy, kapitoly, pexesá a kvízy.

4.2.1 Users

Kolekcia užívateľov obsahuje dokumenty pre každého prihláseného užívateľa – Učiteľa. Každý dokument obsahuje polia:

```
authProvider: string - poskytovateľ autentifikácie (Microsoft)
email: string
name: string
tags: string[] - zoznam tagov, ktoré užívateľ vytvoril
uid: string - identifikačné číslo užívateľa
votingTemplates: {
question: string - hlasovacia otázka
options: string[] - zoznam hlasovacích možností
}[] -- zoznam hlasovacích šablón, ktoré užívateľ vytvoril
```

4.2.2 Courses

Kolekcia kurzov obsahuje záznamy všetkých kurzov. Každý kurz obsahuje:

```
id: string - identifikačné číslo kurzu
title: string - názov kurzu
subtitle: string - stručný popis kurzu
chapters: string[] - zoznam id kapitol, ktoré kurz obsahuje
createdBy: string - id užívateľa, ktorý kurz vytvoril
tags: string[] - zoznam tagov patriacich danému kurzu
```

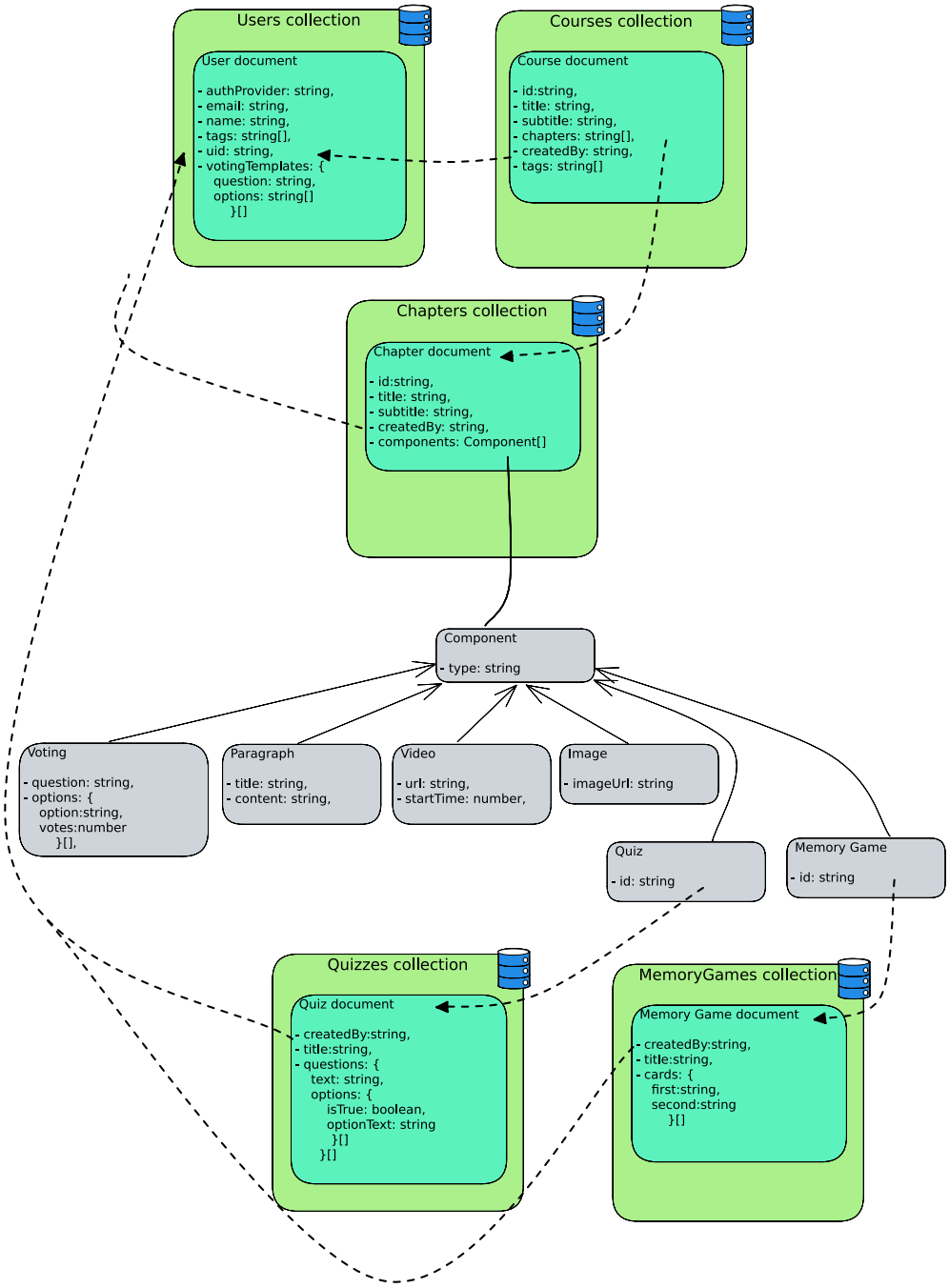
4.2.3 Chapters

Kolekcia kapitol obsahuje všetky kapitoly, kde každá z nich obsahuje polia:

```
id: string - identifikačné číslo kapitoly
title: string - názov kapitoly
subtitle: string - stručný popis kapitoly
createdBy: string - id užívateľa, ktorý kapitolu vytvoril
components: Component[] - zoznam komponentov (vysvetlený nižšie)
```

Ako môžeme vidieť na obrázku 4.2, zoznam komponentov má nejednoznačný typ, čo je využitie funkcionality, ktorú ponúkajú NoSQL databázy. Aby sme vedeli štruktúru na obrázku lepšie popísať, využili sme tradičný zápis dedičnosti (viď sivé obdĺžniky na obrázku). Každý komponent musí obsahovať typ (type) reprezentovaný reťazcom.

V prípade, že sa jedná o zdieľaný komponent (pexeso a kvíz), tak daná inštancia obsahuje id, teda referenciu na zdieľaný komponent v konkrétnej kolekcii (určenej podľa typu). Ostatné komponenty (hlasovania, bloky textu, obrázky a videá) obsahujú polia popísané na obrázku.



Obr. 4.2: Architektúra databázy

4.2.4 Quizzes

Každý kvíz v kolekcii obsahuje:

```
createdBy: string - id užívateľa, ktorý kvíz vytvoril
title: string - názov kvízu
questions: {
  text: string - kvízová otázka
  options: {
    isTrue: boolean - pravdivosť možnosti
```

```
    optionText: string - text možnosti
  } [] - zoznam možností
} []-- zoznam kvízových otázok
```

4.2.5 MemoryGames

Každý pexeso v kolekcii obsahuje:

```
createdBy: string - id užívateľa, ktorý pexeso vytvoril
title: string - názov pexesa
cards: {
  first: string
  second: string
}[] - zoznam kariet, ktoré obsahujú dvojicu pojmov
```

4.2.6 Bezpečnostné opatrenia

Na základe analýzy požiadaviek a z nich vyplývajúcich bezpečnostných opatreniach spomínaných v sekcii 2.3.3 sme vytvorili ich konkrétnu implementáciu vo Firebase. Pravidlá sa zapisujú v jazyku CEL (Common Expression Language) a vyzerajú následovne.

```
service cloud.firestore {
  match /databases/{database}/documents {
  match /courses/{courseId} {
    allow read: if true;
    allow write: if request.auth != null;
    allow update: if request.auth != null &&
        request.auth.uid == resource.data.createdBy;
    }
  match /quizzes/{quizId} {
    allow read: if true;
    allow write: if request.auth != null;
    allow update: if request.auth != null &&
        request.auth.uid == resource.data.createdBy;
      }
  match /memoryGames/{quizId} {
    allow read: if true;
    allow write: if request.auth != null;
    allow update: if request.auth != null &&
        request.auth.uid == resource.data.createdBy;
      }
  match /chapters/{chapterId} {
    allow read: if true;
    allow write: if request.auth != null;
    allow update: if (request.auth != null &&
        request.auth.uid == resource.data.createdBy) ||
        (request.resource.data.diff(resource.data)
        .affectedKeys().hasOnly(['components']));
```

```
      }
  match /users/{userId} {
    allow read: if request.auth != null;
    allow write: if true;
      }
  }
}
```

Ako môžeme vidieť, čítanie dokumentov je povolené pre všetkých (okrem čítania dát užívateľov) – `allow read: if true`. Zapisovanie je podmienené autentifikáciou užívateľa, updatovanie je povolené len v prípade, že sa jedná o rovnakého autora – `request.auth.uid == resource.data.createdBy`.

Výnimku predstavuje updatovanie komponentov v kapitole – to je možné aj v prípade neprihláseného užívateľa a pokryje to funkcionalitu ukladania hlasovacích odpovedí.

```
request.resource.data - dáta, ktoré sme dostali
.diff(resource.data) - rozdiel oproti starým dátam
.affectedKeys() - zoznam kľúčov, ktorých dáta sa menili
.hasOnly(['components']) - zmenili sa len komponenty
```

4.3 Štruktúra priečinka

Adresár bakalárskej práce je štruktúrovaný ako typický projekt v Reacte. Koreňový adresár obsahuje priečinok *edlab*, ktorý ma dva podpriečinky – public a src. Priečinok public obsahuje vstupnú html stránku – index.html. Adresár src obsahuje všetky zdrojové kódy. Podpriečinky sú rozdelené na komponenty, privátne a verejné stránky, obrázky, autentifikáciu, dátové typy a prácu s databázou.

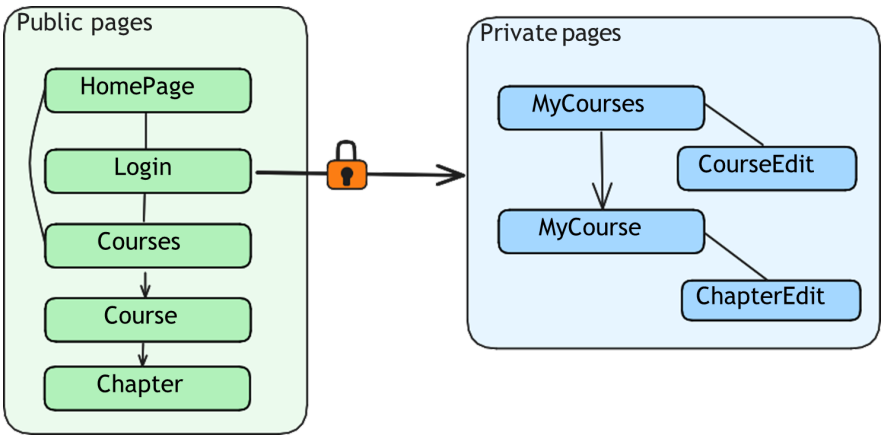
```
edlab
├── public
│   └── index.html
└── src
    ├── App.js
    ├── auth
    ├── components
    ├── images
    ├── private_pages
    │   ├── chapter_form
    │   ├── course_form
    │   ├── user_course
    │   └── user_courses
    ├── public_pages
    │   ├── chapter
    │   ├── course
    │   ├── courses
    │   ├── home
    │   └── login
    ├── types
    └── database
```

4.4 Rozdelenie aplikácie

V tejto sekcii sa bližšie pozrieme ako je webová aplikácia štruktúrovaná, aké je jej rozdelenie na jednotlivé stránky a z akých komponentov pozostávajú.

4.4.1 Rozdelenie na stránky

Aplikácia je rozdelená na takzvanú verejnú a privátnu časť (viď obrázok 4.3). Pre prístup ku privátnej časti je potrebné prihlásenie.



Obr. 4.3: Rozdelenie na stránky

Verejná časť obsahuje tri hlavné stránky – Domov, Kurzy a Prihlásenie, medzi ktorými je možné sa navigovať pomocou navigačnej lišty. Stránka *Domov* obsahuje úvodné informácie o aplikácii a odkazy na jednotlivé jej časti. Stránka *Prihlásenia* obsahuje funkcionalitu prihlásenia sa. Najdôležitejšia časť aplikácie je stránka ponuky *Kurzov*, ktorá obsahuje všetky vzdelávacie kurzy. Z tejto stránky je potom možné sa dostať na stránku konkrétneho kurzu a jeho kapitol.

Privátna časť obsahuje stránku *Moje kurzy*, na ktorej sú zobrazené všetky kurzy, ktoré užívateľ vytvoril. V tomto zobrazení je pridaná funkcionalita vytváranie nových kurzov, ich editácie a odstránenia. Takisto ako vo verejnej časti, aj tu sú odkazy na konkrétne kurzy a ich kapitoly, ktoré vie ich autor editovať a mazať.

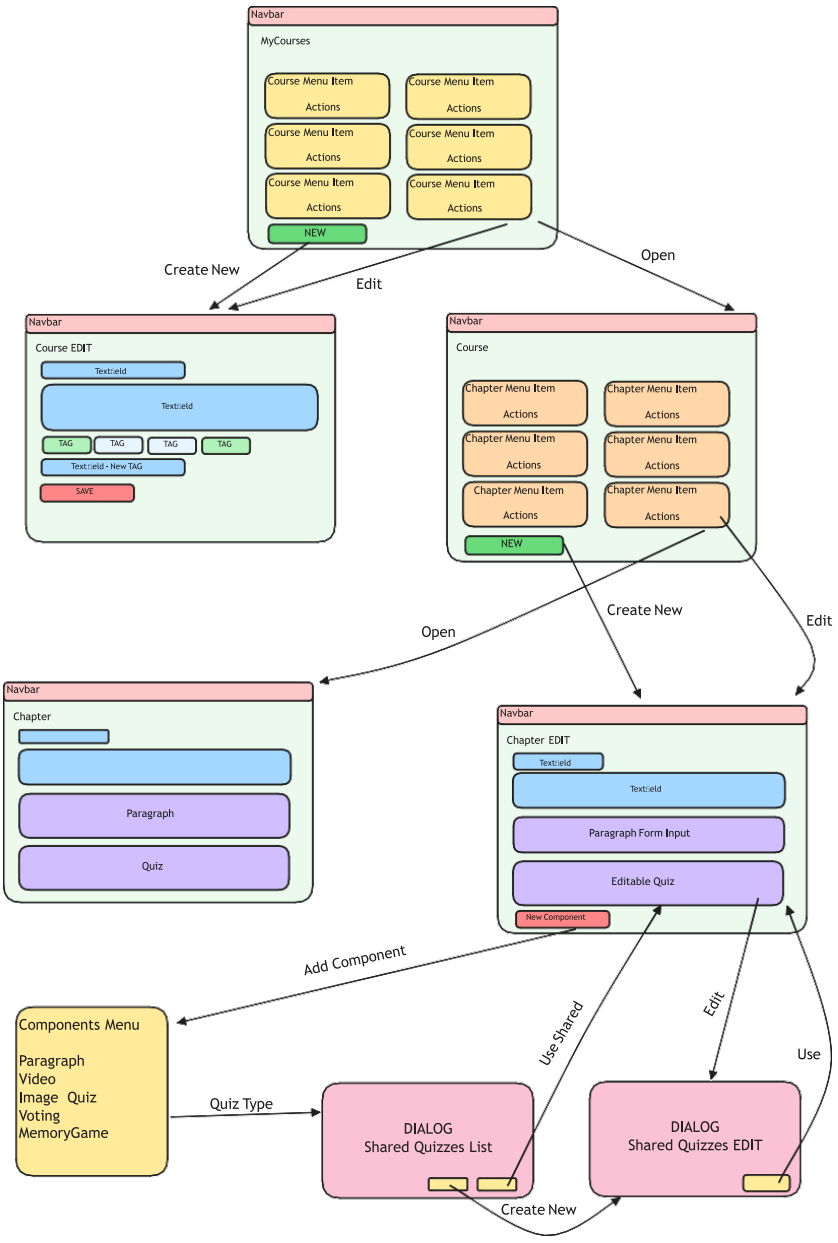
4.4.2 Komponenty

Na obrázku 4.4 sme načrtli rozloženie a prepojenie komponentov v určitej časti aplikácie – zachytili sme 4 stránky ktoré pokrývajú zoznam kurzov (MyCourses), ktoré užívateľ vytvoril, editáciu kurzov (CourseEDIT), ponuku kapitol konkrétneho kurzu (Course) aj ich editáciu (ChapterEDIT). Čitateľ si môže všimnúť opakované použitie určitých komponentov. Na obrázku sú pomocou pomenovaných šípok zobrazené aj tzv. *workflows*, ktoré vysvetľujú ako sú jednotlivé komponenty a stránky prepojené.

Dôležité je podotknúť, že stránka vytvorenia novej kapitoly alebo kurzu je totožná s editáciou, čo sme zabezpečili použitím jedného React komponentu s odlišnými údajmi.

Ak sa pozrieme bližšie na stránku editovania kapitoly – ChapterEDIT, môžeme si všimnúť ako funguje pridávanie zdieľaných komponentov. Po výbere typu zdieľaného komponentu v ComponentsMenu sa zobrazí dialóg so zoznamom už vytvorených komponentov daného typu (napr. Dialog Shared Quizzes List). Vieme tento komponent použiť alebo vytvoriť nový pomocou dialógu na vytváranie a editáciu špecifického komponentu. Nezdieľané komponenty (bloky textu, obrázky, videá) sú pridávané ako formuláre obsahujúce konkrétne polia.

Každý komponent v aplikácii má svoj vlastný priečinok, v ktorom sa nachádza súbor **.tsx**, z ktorého sa exportuje práve jeden Reactovský komponent. V prípade potreby priečinok komponentu obsahuje aj súbor so špecifikáciou štýlov.



Obr. 4.4: Rozdelenie na komponenty a ich prepojenie

4.5 Dátový model v TypeScripte

Na to aby sme vedeli pracovať s NoSQL databázou bezpečne, vytvorili sme si dátový model s typmi, ktoré používame naprieč aplikáciou a aj pri načítaní a ukladaní dát do Firestore databázy.

Model vychádza z architektúry databázy a možností jazyka TypeScript. V priečinku **./src/types** je súbor **./src/types/firebaseTypes.ts**, ktorý obsahuje typy v reprezentácii ako sú uložené vo firebase.

```
export type FirebaseCourse = {
  id: string;
  title: string;
  subtitle: string;
  chapters: string[];
  createdBy: string;
  tags: string[];
};
export type FirebaseChapter = {
  title: string;
  subtitle: string;
  createdBy: string;
  components: Array<Component>;
  id: string;
};
export interface FirebaseComponent {}
export interface FirebaseParagraph extends
   FirebaseComponent {
  title: string;
  content: string;
  type: ComponentTypes;
}
export interface FirebaseImage extends FirebaseComponent
   {
  imageUrl: string;
  type: ComponentTypes;
}
export interface FirebaseVideo extends FirebaseComponent
   {
  url: string;
  type: ComponentTypes;
}
...
```

Na základe týchto Firebase typov, sme potrebovali vytvoriť rozšírenie, s ktorým by sme jednoducho vedeli pracovať v aplikácii. V následujúcom texte sú zobrazené časti súboru **./src/types/components.ts**, v ktorom sa nachádzajú triedy predstavujúce jednotlivé komponenty, ktoré dedia z hlavnej abstraktnej base triedy Component.

Abstraktná trieda obsahuje všetky povinné polia a metódy, ktoré musia jej potomkovia obsahovať. Metóda`setReactComponent` uloží do triedy konkrétny Rec-

tovský komponent, ktorý bude jeho reprezentáciou pri prezeraní kapitoly. Obdobnú funkcionalitu má aj `setEditableComponent`, ktorá zabezpečí prepojenie s komponentom, ktorý zabezpečí jeho editáciu alebo vytváranie na stránke.

Abstraktná metóda `exportToFirebase` zabezpečí to, že každý komponent musí obsahovať vlastnú implementáciu vďaka čomu sa dá exportovať do konkrétneho Firebase typu, ktorý je potomkom FirebaseComponent, pri ukladaní do databázy.

Každý komponent má aj svoj typ, ktorý je reprezentovaný ako enum stringov.

```
// Define component types as an enum
export enum ComponentTypes {
    paragraph = "paragraph",
    video = "video",
    image = "image",
    memoryGame = "memoryGame",
    quiz = "quiz",
    vote = "vote",
}

// Abstract base class for components
export abstract class Component {
  reactComponent: FunctionComponent = React.Fragment;
  reactEditableComponent: FunctionComponent = React.
     Fragment;
  public id: number = 0;
  public type: ComponentTypes;

  //base class constructor
  constructor(type: ComponentTypes) {
      this.type = type;
  }
  public setEditableComponent(reactComponent:
     FunctionComponent) {
    this.reactEditableComponent = reactComponent;
  }
  public setComponent(reactComponent: FunctionComponent)
     {
    this.reactComponent = reactComponent;
  }
  //Exports the component to the specific Firebase
     Component
  public abstract exportToFirebase(): FirebaseComponent;
}
```

4.5.1 Príklad použitia dátového modelu

V príklade nižšie sa môžeme pozrieť na konkrétny komponent `ParagraphType` a jeho implementáciu. Komponent má pridané nové polia `title` a `content`.

V konštruktore sa tieto polia inicializuje podľa vstupných údajov, volá sa konštruktor rodičovskej triedy, kde nastaví svoj typ (`ComponentTypes.paragraph`) a komponent na `<Paragraph ... />`. V implementácii metódy `exportToFirebase` vytvára objekt typu `FirebaseParagraph`, ktorý ho bude reprezentovať v databáze.

```
// Paragraph component type
export class ParagraphType extends Component {
  public title: string;
  public content: string;
  constructor(title: string, content: string,
     paragraphCount: number) {
    //call base class constructor
      super(ComponentTypes.paragraph);
      this.title = title;
      this.content = content;
      this.setComponent(() => (
        <Paragraph
          order={paragraphCount}
          content={this.content}
          title={this.title}
        />
      ));
  }
  exportToFirebase(): FirebaseParagraph {
    //creates a new FirebaseParagraph
    return { title: this.title, content: this.content,
       type: this.type };
  }
}
```

4.6 Práca s databázou

Ako sme zmienili v úvode kapitoly, Firebase poskytuje takzvané Firebase SDK, ktoré zaručí prepojenie medzi projektom a databázou.

Súbory obsahujúce funkcie práce s databázou sa nachádzajú v priečinku **./src/database** a sú rozdelené podľa toho s čím pracujú.

4.6.1 Kapitoly

Ak sa pozrieme napríklad na kapitoly (**./src/database/chapters.tsx**), funkcionalita je rozdelená na 4 časti – získavanie všetkých kapitol vrámci kurzu, pridanie novej kapitoly, editácia kapitoly a odstránenie.

V príklade vidíme napríklad funkciu `getChapterData`, ktorá získa z databázy údaje o kapitole na základe jej id, skonvertuje ich do správneho typu a vráti ich.

Funkcia je asynchrónna a vracia `Promise` na základe úspešnosti výsledku – ak bola kapitola nájdená, vráti jej dáta, ak nie, vráti null.

```
export async function getChapterData(chapterId: string):
   Promise<types.ChapterType | null> {
  const docRef = doc(db, "chapters", chapterId);
  const docSnap = await getDoc(docRef);
  if (docSnap.exists()) {
    const chapterData = await convertChapterFromFirebase(
       docSnap);
    return chapterData;
  } else {
    console.log(`No chapter found with id: \${chapterId
       }`);
    return null;
  }
}
```

4.6.2 Obrázky

V tejto sekcii sa pozrieme ako sa nahrávajú obrázky do Firestore databázy. V nasledujúcom kóde je ukázaný príklad funkcie pre nahrávanie obrázkov zo súboru `src/database/imageUpload.tsx`. Najprv importujeme potrebne funkcie pre prácu s obrázkami. Funkcia uploadImage je asynchrónna, prijíma parameter image (obrázok), a vracia Promise<string | null>, ktorý je URL adresou nahraného obrázka alebo hodnotou null v prípade chyby.

Vrámci tela funkcie vytvoríme referenciu na obrázok v úložisku pomocou `ref` funkcie, kde zadávame referenciu na úložisko a určujeme cestu k obrázku. V tomto prípade sme použili názov adresára `images` a pridali náhodne generovaný identifikátor k názvu obrázku.

Následne použijeme funkciu `uploadBytes` na nahranie bytov obrázka do referencie úložiska. S pomocou `getDownloadURL` získame URL adresu pre stiahnutie nahraného obrázka. V prípade úspešného nahrania obrázka, vrátime získanú URL adresu.

V prípade chyby ju vypíšeme v konzole a vrátime hodnotu `null`.

Získaná url obrázka sa potom uloží do FirebaseImage.

```
import { getDownloadURL, ref, uploadBytes } from "
   firebase/storage";
import { v4 } from "uuid";
import { storage } from '../../firebase-config';

export const uploadImage = async (image): Promise<string
   | null> => {
  try {
    // Create a reference to the image in the storage
    const imageRef = ref(storage, `images/\${image.name +
        v4()}`);
    // Upload the image bytes to the storage reference
    const snapshot = await uploadBytes(imageRef, image);
    // Get the download URL of the uploaded image
    const url : string = await getDownloadURL(snapshot.
       ref);
    // Return the URL of the uploaded image
    return url;
  } catch (error) {
    console.log(error);
    // Return null in case of error
    return null;
  }
};
```

4.6.3 Prihlásenie

Prihlásenie a odhlásenie sa rieši v súbore `./src/auth/firebaseAuth.ts`. Prebieha pomocou autentifikácie, ktorú poskytuje Firebase.

V našom prípade podporujeme prihlásenie pomocou konta Microsoft, kon-

krétne s využitím poskytnutej async funkcie `signInWithPopup`, ktorá užívateľa vyzve aby zadal svoje prihlasovacie údaje v pop up okne. Následne spracujeme výsledok a ak došlo k chybe v priebehu prihlasovania, nahlásime ju. V prípade že bolo prihlásenie úspešné, overíme či sa užívateľ už v databáze nachádza a ak nie, tak vytvoríme nového užívateľa (FirebaseUser) so získanými údajmi.

Výhodou takto spracovaného prihlásenia a registrácie je, že neriešime žiadne heslá ale spoliehame sa na konkrétnych poskytovateľov autentifikácie.

```
export const signInWithMicrosoft = async () => {
  try {
    const result = await signInWithPopup(auth, provider);
    // Get the OAuth access token and ID Token
    const credentials = OAuthProvider.
       credentialFromResult(result);
    if (credentials == null) {
        throw Error("Credentials is null");
    }

    const user = result.user;
    const q = query(collection(db, "users"), where("uid",
        "==", user.uid));
    const docs = await getDocs(q);

    if (docs.docs.length === 0) {
      //create new user
      const newUser: FirebaseUser = {
        uid: user.uid,
        name: user.displayName ? user.displayName : "",
        tags: [],
        authProvider: "Microsoft",
        votingTemplates: [],
        email: user.email!,
      };
        //register new user
      await setDoc(doc(db, "users", user.uid), newUser);
    }
  } catch (error) {
    console.error(error);
    // Provide user-friendly error message to the user
    alert("Sign-in failed");
    logOut();
  }
};
```

4.7 Implementačné detaily špecifických komponentov

V tejto sekcii sa zameriame na určité komponenty alebo skupiny komponentov, ktoré sú svojou funkcionalitou špecifické.

4.7.1 Formuláre

V aplikácii formuláre slúžia primárne pre vytváranie alebo editáciu nových kapitol, kurzov a komponentov.

Ako môžeme vidieť v príklade nižšie, po pridaní nového komponentu – odseku textu (`ParagraphInput.tsx`), sa pridá Reactovský komponent obsahujúci formulár pre vytváranie. Takýto komponent obsahuje textové polia (`<TextField>`) pre vkladanie obsahu a funkciu (`handleParagraphChange`), ktorá reaguje na ich zmeny.

```
//ParagraphInput.tsx

export default function ParagraphInput({ title, content,
   setComponents, index }: {
  title: string; content: string; setComponents: React.
     Dispatch<React.SetStateAction<Component[]>>;
  index: number;
}) {
  const [_title, setTitle] = useState(title);
  const [_content, setContent] = useState(content);

  const handleParagraphChange = (title: string, content:
     string) => {
    setComponents((prevData) => {
      const updatedData = [...prevData];
      (updatedData[index] as ParagraphType).title = title
         ;
      (updatedData[index] as ParagraphType).content =
         content;
      return updatedData;
    });
  };

  return (
    <div className="content-padding">
      <Typography variant="h5" color="secondary">+ Blok
         textu</Typography>
      <TextField ... />
      <TextField ... />
    </div>
  );
}
```

4.7.2 Zdieľané komponenty

V tejto časti si bližšie popíšeme ako je implementovaná funkcionalita zdieľaných komponentov. Zameriame sa na ich vytváranie a editáciu a ako príklad použijeme Kvíz.

Ak chceme kapitole pridať nový zdieľaný komponent, z ponuky typov komponentov vyberieme Kvíz. Zdrojový kód spomínanej časti sa nachádza v súbore `ChapterFormComponents.tsx`. Následne po výbere, sa nám otvorí dialóg s ponukou všetkých už existujúcich kvízov, ktoré vieme použiť (`SharedQuizDialog.tsx`).

Ak si vyberieme konkrétny kvíz z ponuky, do kapitoly sa pridá karta odkazujúca sa na tento komponent (`<QuizCard>`).

```
// QuizCard.tsx

export default function QuizCard({ id, quiz, onSubmit,
   firebaseID }) {
  const [quizInputDialogOpen, setQuizInputDialogOpen] =
     useState<boolean>(false)
  ...
  //editing opens QuizInputDialog
  const handleEditButtonClick = () => {
     setQuizInputDialogOpen(true);}
  ...
  return (
    <div>
      <div>
          <Typography >+ Kviz</Typography>
          <IconButton onClick={handleEditButtonClick}>
              <EditIcon />
          </IconButton>
      </div>
      <div>
        <Typography>{quiz.title}</Typography>
        {
          quiz.questions.map((q,index) => <Typography> -
             {q.text}</Typography>)
        }
      </div>
      <QuizInputDialog  quiz={quiz} open={
         quizInputDialogOpen} onClose={onClose} onSubmit
         ={submitWithIndex} firebaseID={firebaseID}/>
    </div>
    );
}
```

Kvíz vieme editovať tlačidlom na karte a funkcia `handleEditButtonClick` otvorí editačný dialóg nachádzajúci sa v `QuizInputDialog.tsx`, ktorý obsahuje potrebné údaje.

V dialógu s ponukou zdieľaných kvízov vieme vybrať aj možnosť vytvorenia nového kvízu, čo otvorí rovnaký dialóg ako pri editácii (`<QuizInputDialog>`),

akurát s inými údajmi, resp. s prázdnym kvízom a nedefinovaným firebaseID (viď príklad nižšie).

```
// from SharedQuizDialog.tsx
<QuizInputDialog  quiz={{ title: "", questions: [],
   createdBy: userId }} open={newDialogOpen} onClose={
   onClose} onSubmit={onSubmit} firebaseID={undefined}/>
```

4.7.3 Dialógy

Dialógy sa v aplikácii používajú primárne na tri účely – potvrdenie akcie (napríklad odstránenie), výber z ponuky už existujúcich zdieľaných komponentov a hlasovacích šablón a pre vytváranie a editáciu týchto komponentov.

V príklade vidíme dialóg potvrdzujúci vymazanie, ktorý dostane ako vstupné údaje – `DeleteConfirmationProps`, správu, ktorú sa má opýtať a delete funkciu, ktorú po potvrdení zavolá. Vstupné údaje sú reprezentované ako interface, teda musia spĺňať požadované typy a obsah.

```
// DeleteConfirmationDialog.tsx


interface DeleteConfirmationProps {
    message: string;
    onDelete: () => Promise<State>;
  }

const DeleteConfirmationDialog: React.FC<
   DeleteConfirmationProps> = ({ message, onDelete }) =>
   {...}

export default DeleteConfirmationDialog;
```

4.7.4 Navigácia

Navigácia v aplikácii je riešená pomocou funkcionality `react-router-dom` verzie 6. Navigáciu medzi hlavnými stránkami sme zabezpečili pomocou komponentu navigačnej lišty (v `Navbar.tsx`), ktorá obsahuje jednotlivé linky na stránky. Konkrétne linkovanie je potom zaručené pomocou Routes elementu (viď príklad nižšie), ktorý obsahuje cesty k jednotlivým stránkam.

Stránky, ktoré sú prístupné len pre prihlásených užívateľov sú zaobalené v `PrivateRoutes`, ktoré kontrolujú autentifikovaného užívateľa, čím je vyriešená bezpečnosť prístupu.

Časti url začínajúce dvojbodkou (napr. `:chapter_id`) označujú miesto pre vloženie konkrétneho identifikátoru, na základe neho môžeme po prístupe na stránku načítať z databázy konkrétne dáta.

```
// from App.tsx

<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/courses" element={<Courses />} />
  <Route path="/courses/:course_id" element={<Course />}
     />
  <Route path="/courses/:course_id/chapters/:chapter_id"
     element={<Chapter />} />
  <Route element={<PrivateRoutes />}>
     <Route path="/user_courses" element={<UserCourses
        />} />
    <Route path="/course_form" element={<CourseForm />}
       />
    <Route path="/user_courses/:course_id" element={<
       UserCourse />} />
    <Route path="/user_courses/:course_id/chapter_form/:
       chapter_id" element={<ChapterForm isNew={false}
       />} />
    <Route path="/user_courses/:course_id/chapter_form"
       element={<ChapterForm isNew={true} />}/>
  </Route>
  <Route path="/login" element={<LogIn />} />
</Routes>
```

4.7.5 Drag and Drop

Funkcionalita Drag and Drop umožňuje pri vytváraní alebo editácii kapitoly jednoducho meniť poradie komponentov, ktoré obsahuje. Zmena poradia je možná potiahnutím karty konkrétneho komponentu a presunutím jej na nové miesto.

Implementácia je realizovaná pomocou knižnice DND Kit[1] a pozostáva z dvoch častí – vytvorenie posúvateľného objektu (u nás SortableItem.tsx) a priestoru ktorý tieto objekty obsahuje.

Nižšie môžeme vidieť časť kódu z implementácie nachádzajúceho sa v súbore `./src/components/sortable_item/SortableItem.tsx`. Tento komponent obsahuje tri podstatné časti – tlačidlo s ikonou odstránenia, ktoré zavolá príslušnú funkciu, oblasť obsahujúcu `DragIndicatorIcon`, ktorá zabezpečí drag and drop funkcionalitu a následne `ChildComponent`, ktorý do karty vloží komponent slúžiaci pre editáciu.

```
// SortableItem.tsx

export function SortableItem({ sortableId, deleteId,
   child, deleteComponentFunction }: { sortableId: number
   , deleteId: number, child: FunctionComponent,
   deleteComponentFunction: (index: number) => void; }) {
```

[1] https://dndkit.com/

```
  ...

  const ChildComponent = child;
  return (
    <Paper style={style} {...attributes}>
      <div>
          <IconButton onClick={() => {
             deleteComponentFunction(deleteId)}}>
                  <DeleteIcon />
          </IconButton>
          <div ref={setNodeRef}  {...listeners}>
              <DragIndicatorIcon />
          </div>
      </div>
      <ChildComponent />
    </Paper>
  )
}
```

V ďalšej časti je zobrazené použitie `DndContext`, ktoré určuje oblasť, v ktorej vieme komponenty presúvať. Zobrazená časť kódu sa nachádza v súbore `./src/private_pages/chapter_form/ChapterComponents.tsx`.

Do komponentu `SortableItem` vkladáme potrebné údaje, teda funkciu na odstránenie, indexy a potom konkrétny komponent predstavujúci editáciu (viď sekcia 4.5). Ako si môže čitateľ všimnúť, do komponentu pridávame indexy od 1 a nie od 0 ako býva zvykom, je to kvôli návrhu knižnice, ktorú používame.

```
<DndContext collisionDetection={closestCenter} onDragEnd
   ={handleDragEnd}>
  <div >
    <Typography>Komponenty</Typography>
    <SortableContext items={components} strategy={
       verticalListSortingStrategy} >
      {components.map((component, index) => (
        <SortableItem
          deleteComponentFunction={deleteComponent}
          deleteId={component.id}
          key={index + 1}
          sortableId={index + 1}
          child={component.reactEditableComponent}
        />
      ))}
    </SortableContext>
  </div>
</DndContext>
```

4.8 Pridanie nového typu komponentu

V tejto sekcii si rozoberieme postup pre pridanie nového typu komponentu.

- V prvom rade je potrebné spraviť podporu priamo vo Firebase konzole, ak ide o zdieľaný komponent je potrebné vytvoriť mu vlastnú kolekciu a nastaviť security rules (viď sekcia 4.2.6).
- Následne je potrebné pridať nový typ do ComponentTypes enum, vytvoriť nový Firebase interface pre potrebný typ a takisto aj triedu, ktorá bude rozširovať abstract base Component (viď sekcia 4.5). Napr. pomocou `export class NewComponentType extends Component {...}`
- Potom je potrebné vytvoriť Reactovský komponent pre zobrazenie komponentu a pridať ho do konštruktora triedy ako `reactComponent`. Okrem toho musíme vytvoriť aj Reactovský komponent, ktorý bude podporovať editáciu a ten pridať vo funkcii `convertComponents` v súbore ChapterFormComponents.tsx pomocou `setEditableComponent()`.
- Posledným krokom je upraviť funkcie pracujúce s databázou (podľa toho či sa jedná o zdieľaný komponent alebo nie) na základe príkladu ostatných komponentov.

4.9 Požiadavky pre lokálne spustenie

Na to aby bolo možné aplikáciu spustiť, je potrebné mať nainštalovaný nástroj **npm** verzie aspoň 8.12.2 a **Node.js** verzie 17.7.2 (návod na inštaláciu nájdete napríklad na `https://nodejs.org/en/download`).

Ak máme tieto potrebné nástroje nainštalované, v priečinku edlab vieme nainštalovať pomocou príkazu `npm install --legacy-peer-deps` a následne spustiť pomocou `npm start`. Po obhájení bakalárskej práce nevieme zaručiť, že bude Firebase databáza stále dostupná, preto v kapitole 6 uvedieme postup, ako ju prepojiť s vlastnou Firebase databázou.

5. Užívateľská dokumentácia - študent a učiteľ

5.1 Prístup na stránku

V čase tvorby bakalárskej práce a v určitom časovom obmedzení po jej odovzdaní bude webová aplikácia dostupná na adrese *https://edlab-9e22c.web.app/*, v prípade, že je odkaz neaktuálny je možné sa na stránku dostať pomocou návodu v kapitole 6.

5.2 Základné user stories – študent

V nasledujúcich častiach si prejdeme základné *user stories* typické pre rolu študenta. V prípade potreby zobrazíme snímky obrazoviek aplikácie spolu s označením potrebných krokov.

5.2.1 Výber kurzu

Pre prechod na stránku ponuky kurzov je potrebné vybrať položku **Kurzy** (1), ktorá sa nachádza v navigačnej lište (viď obrázok 5.1). Užívateľ je automaticky presmerovaný a zobrazia sa mu všetky existujúce kurzy. Na to aby užívateľ otvoril konkrétny kurz je potrebné stlačiť tlačidlo **Otevřít kurz** (2) na karte kurzu.

Obr. 5.1: Výber kurzu

5.2.2 Vyhľadávanie kurzu

Ak chce užívateľ vyhľadávať v ponuke kurzov, môže použiť vyhľadávacie pole (1), ktoré sa nachádza v hornej časti stránky (viď obrázok 5.2). Automaticky počas písania sa zaháji vyhľadávanie podľa názvu kurzu alebo jeho tagov.

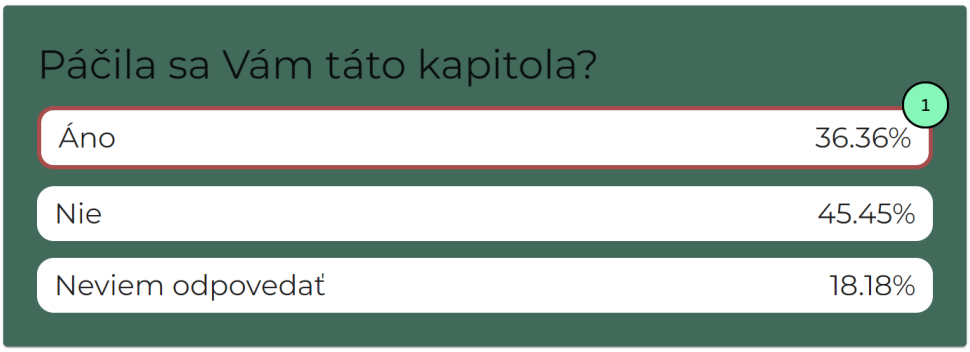
5.2.3 Výber kapitoly

Predpokladáme, že užívateľ sa nachádza na stránke konkrétneho kurzu, ktorý obsahuje kapitoly. Pre výber kapitoly je potrebné stlačiť tlačidlo **Otevřít kapitolu**, ktoré sa nachádza na karte každej kapitoly.

Obr. 5.2: Vyhľadávanie kurzu

5.2.4 Práca s hlasovaním

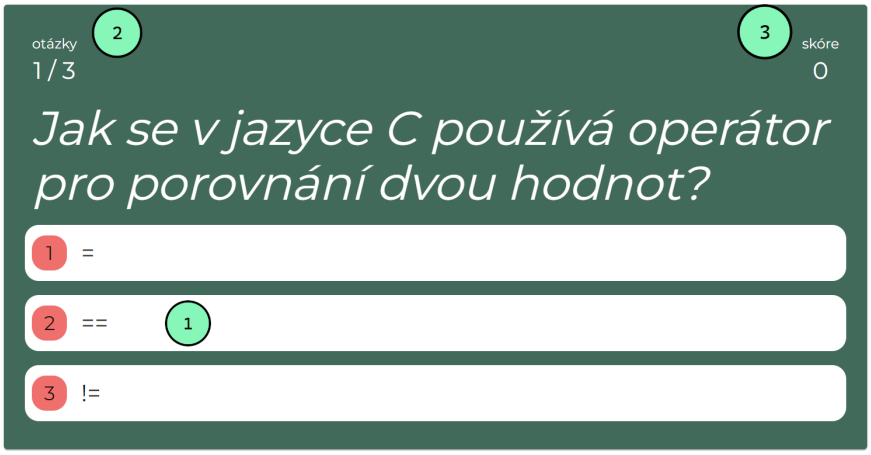
Predpokladáme, že užívateľ sa nachádza na stránke kapitoly, ktorá obsahuje hlasovanie (viď obrázok 5.3). Hlasovanie obsahuje hlasovaciu otázku a odpovede, ktoré pripravil autor kurzu. Ak chce užívateľ hlasovať označí svoju odpoveď. Následne mu budú zobrazené výsledky celkového hlasovania ①.



Obr. 5.3: Práca s hlasovaním

5.2.5 Práca s kvízom

V kapitole, v ktorej sa nachádza kvíz je možné sa ho zúčastniť odpovedaním na otázky, ktoré obsahuje (obrázok 5.4). Svoju odpoveď užívateľ zvolí stlačením jednej z ponúkaných možností ①. V priebehu odpovedania na otázky je mu zobrazený jeho progres ② a priebežný výsledok ③. Po ukončí kvízu (obrázok 5.5) užívateľ vidí svoj výsledok ④ a má možnosť kvíz zopakovať ⑤.

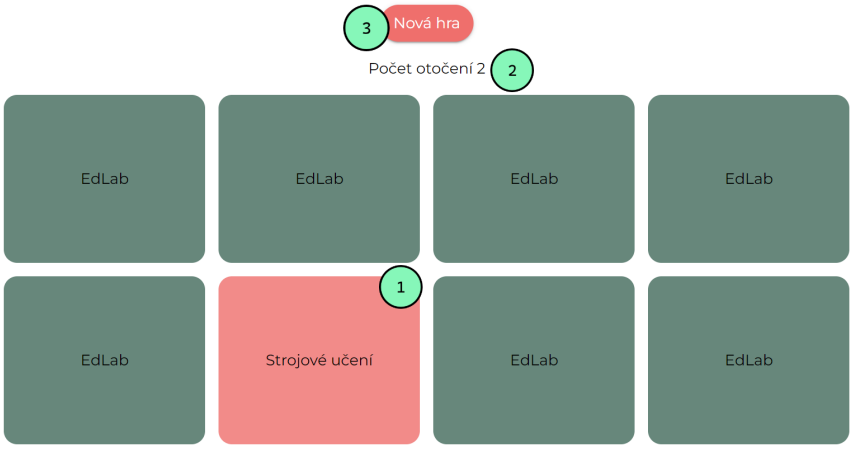


Obr. 5.4: Práca s kvízom

Obr. 5.5: Výsledky kvízu

5.2.6 Práca s pexesom

Pexeso obsahuje dvojice kariet, ktoré su náhodne rozmiestnené a otočené obsahom tak, že ho nie je vidieť (viď obrázok 5.6). Jeden tak spočíva v otočení dvoch kariet. Ak chceme otočiť kartu, klikneme na ňu (1) a po animácií sa zobrazí jej obsah. Ak otočíme 2 karty, ktoré nepatria do jednej dvojice, karty sa po chvíli otočia do pôvodnej pozície a začína nový ťah. V prípade, že ťah bol úspešný (karty sú z jednej dvojice), ostávajú karty otočené. V priebehu hry je zobrazený počet ťahov (2) a tlačidlo (3) pre opätovné spustenie hry.



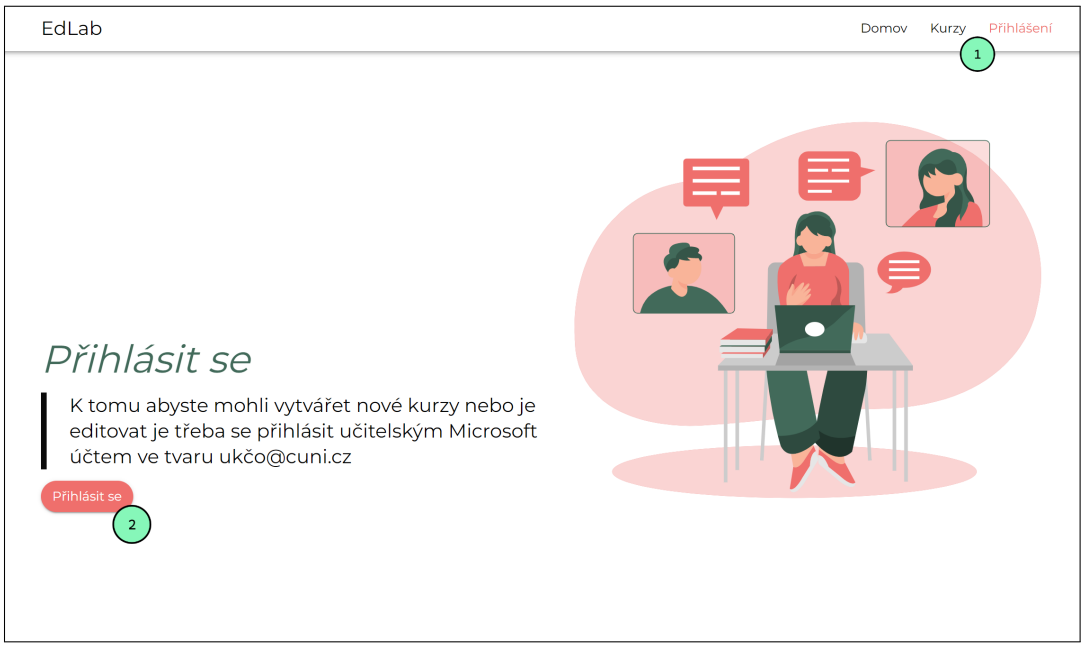
Obr. 5.6: Práca s pexesom

5.3 Základné user stories – Učiteľ

V nasledujúcich častiach si prejdeme základné *user stories* typické pre rolu učiteľa. V prípade potreby zobrazíme snímky obrazoviek aplikácie spolu s označením potrebných krokov.

5.3.1 Registrácia a prihlásenie

Pre registráciu a prihlásenie je potrebné sa dostať na stránku kliknutím na položku *Přihásení* ① v navigačnej lište (viď obrázok 5.7) alebo príslušným tlačidlom na úvodnej obrazovke Domov. Následne sa zobrazí príslušná stránka s tlačidlom **Přihlásit se** ③. Po kliknutí na spomínané tlačidlo sa užívateľovi zobrazí dialógové okno ktoré presmeruje prihlásenie na stranu poskytovateľa (v našom prípade Microsoft), kde je užívateľ vyzvaný aby zadal svoje prihlasovacie údaje. Po správnom vyplnení údajov je užívateľ úspešne prihlásený.



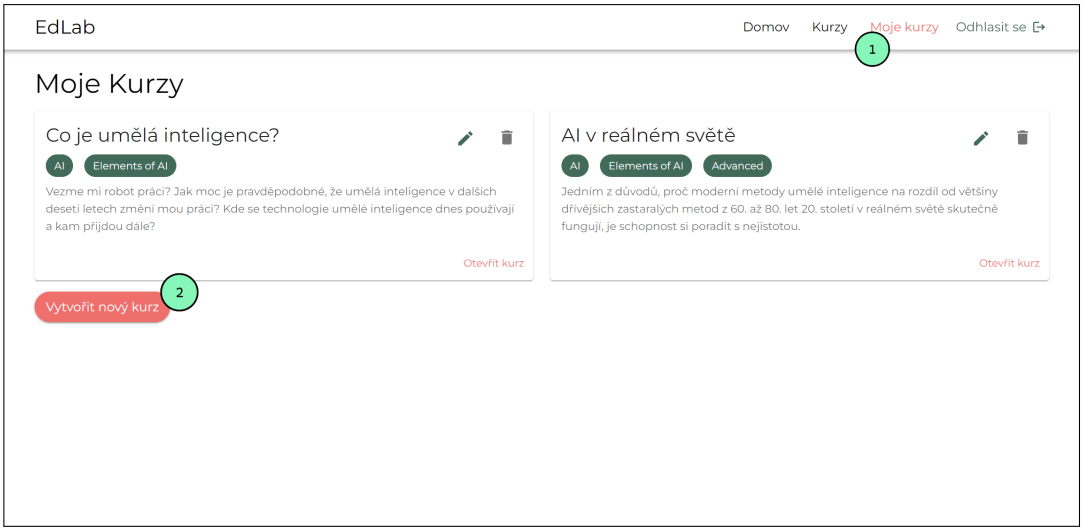
Obr. 5.7: Registrácia a prihlásenie

5.3.2 Vytvorenie nového kurzu

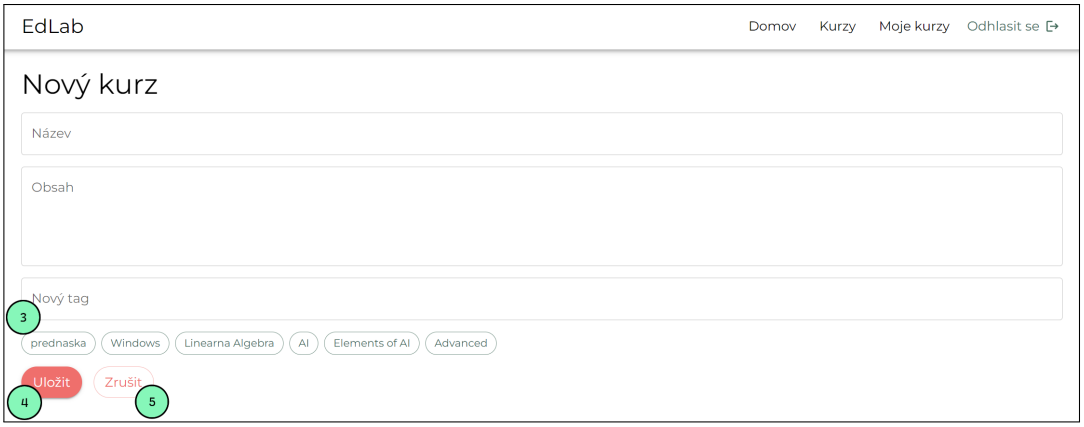
Kliknutím na položku *Moje kurzy* ① je užívateľ presmerovaný na stránku, so zoznamom všetkých kurzov, ktoré vytvoril (viď obrázok 5.8). Na konci tohto zoznamu sa nachádza tlačilo **Vytvořit nový kurz** ②, vďaka ktorému sa dostane na stránku vytvorenia nového kurzu.

Stránka obsahuje povinné textové polia, ako je názov a obsah kurzu (viď obrázok 5.9). Okrem toho vie užívateľ priradiť kurzu tagy③ - vyberie z ponuky už existujúcich tagov alebo vytvorí nový.

Pre uloženie novovytvoreného kurzu je potrebné stlačiť tlačidlo *Uložit* ④. Ak chceme ukončiť proces vytvárania bez uloženia, zvolíme možnosť *Zrušit* ⑤.



Obr. 5.8: Vytvorenie nového kurzu – stránka mojich kurzov

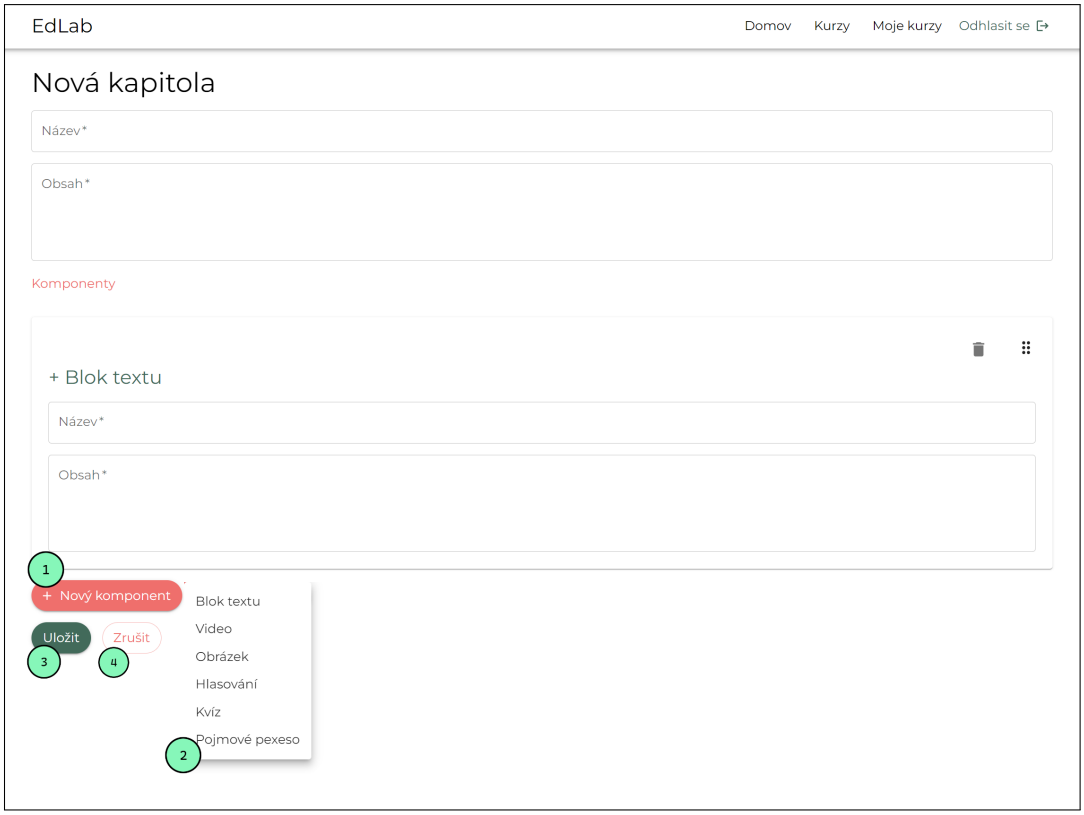


Obr. 5.9: Vytvorenie nového kurzu – formulár

5.3.3 Vytvorenie novej kapitoly

Predpokladá sa, že užívateľ má vytvorený *kurz* a nachádza sa na jeho stránke. Pre vytvorenie novej kapitoly je potrebné stlačiť tlačidlo **Vytvořit novou kapitolu**, ktoré sa nachádza na stránke. Následne bude užívateľ presmerovaný na stránku vytvárania novej kapitoly (viď obrázok 5.10). Na tejto stránke sa nachádzajú povinné textové polia ako je nadpis a podnadpis kapitoly. Okrem toho sa tam nachádza aj sekcia komponentov, ktorá obsahuje tlačidlo **Vytvořit novou komponentu** (1). Po stlačení tohto tlačidla sa zobrazí zoznam typov komponentov(2), ktoré je možné do kapitoly pridať.

Pre uloženie kapitoly je potrebné stlačiť tlačidlo *Uložit* (3). Ak chceme ukončiť proces vytvárania bez uloženia, zvolíme možnosť *Zrušit* (4).



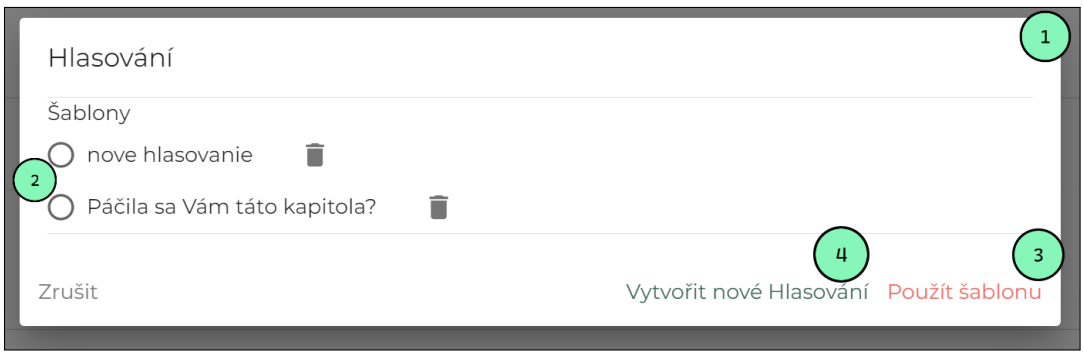
Obr. 5.10: Vytvorenie novej kapitoly

5.3.4 Vytváranie špecifických komponentov

V tejto sekcii si bližšie popíšeme návod ako vytvoriť špecifické komponenty, ktorých vytváranie môže byť pre užívateľa komplexnejšie.

5.3.4.1 Hlasovanie

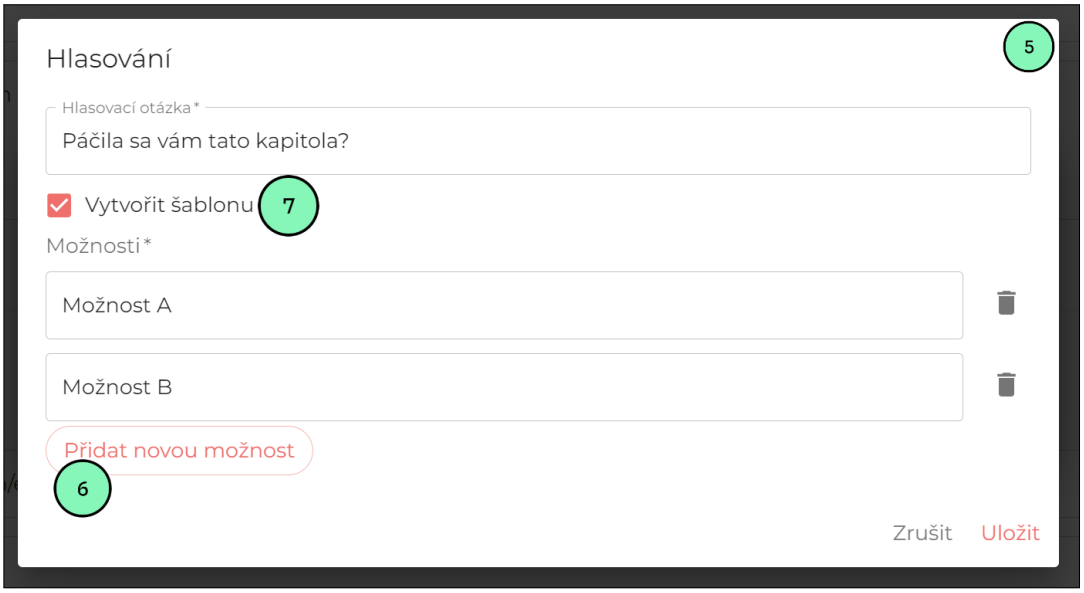
Užívateľ si v ponuke zvolí možnosť *Hlasování* a následne sa mu v dialógovom okne ① zobrazia šablóny hlasovaní, ktoré už v minulosti vytvoril (viď obrázok 5.11). Ak si praje vybrať nejakú z ponúkaných šablón, označí ju ② a potvrdí akciu stlačením tlačidla **Použít šablonu** ③.



Obr. 5.11: Hlasovanie - dialóg so šablónami

Ak chce užívateľ vytvoriť nové hlasovanie, stačí tlačidlo **Vytvořit nové hlasování** ④, ktoré otvorí dialóg s formulárom pre vytváranie hlasovania ⑤ (viď

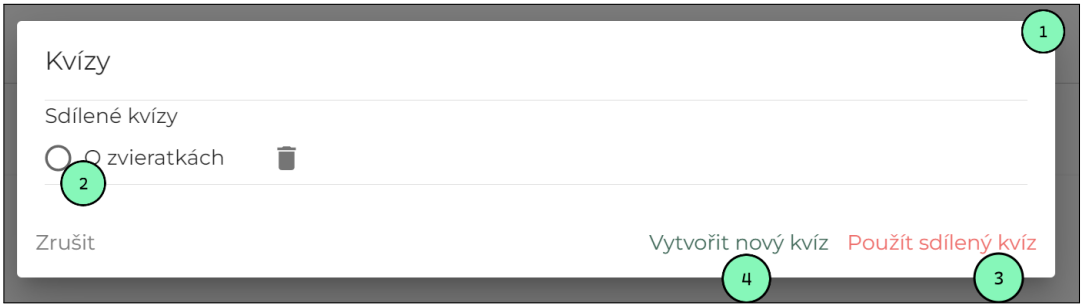
obrázok 5.12). V dialógu je potrebné vyplniť hlasovaciu otázku a pridať minimálne dve hlasovacie odpovede (hlasovacie odpovede sa pridávajú tlačidlom **Přidat novou možnost** ⑥). V dialógu sa takisto nachádza aj zaklikávacie okienko s možnosťou *Vytvořit šablonu* ⑦, v prípade, že si užívateľ túto možnosť zvolí, po uložení kapitoly sa vytvorí šablóna a bude možné ju použiť aj na iných miestach.



Obr. 5.12: Hlasovanie - vytvorenie nového hlasovania

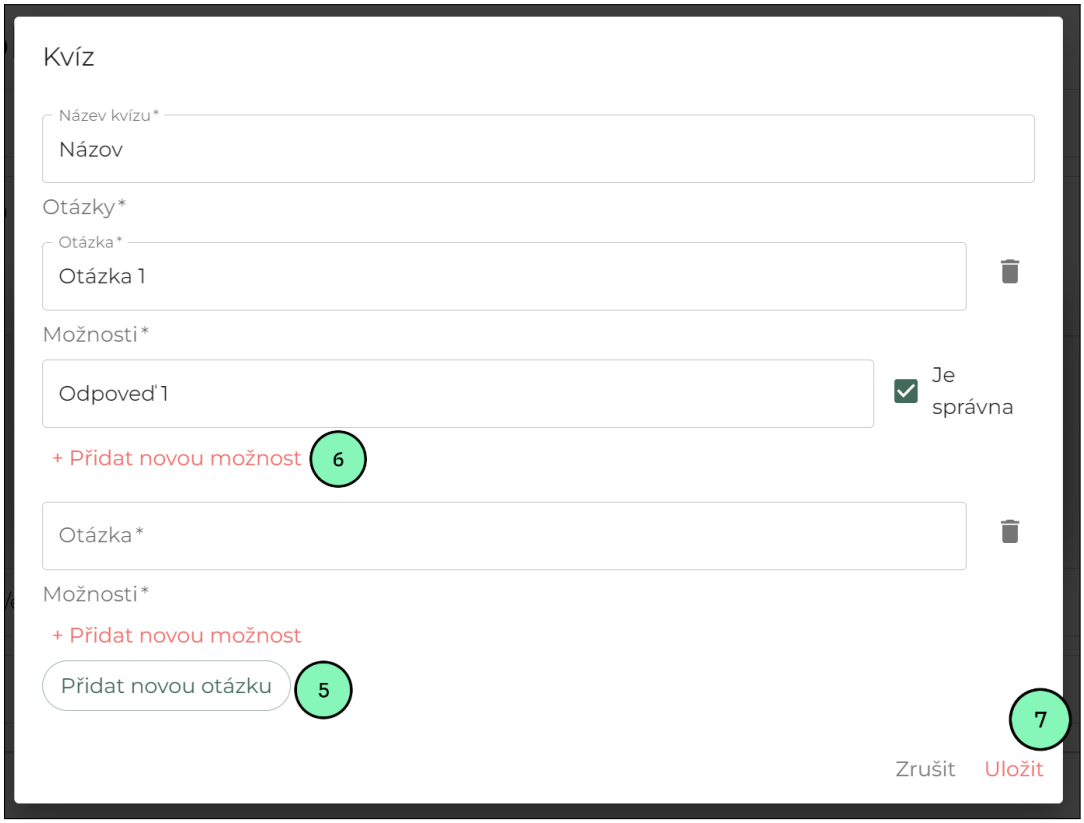
5.3.4.2 Kvíz

Užívateľ si v ponuke zvolí možnosť *Kvíz* a následne sa mu v dialógovom okne ① zobrazia všetky kvízy, ktoré daný užívateľ vytvoril (viď obrázok 5.13). V prípade, že chceme použiť nejaký z týchto kvízov, označíme ho ② a akciu potvrdíme stlačením tlačidla **Použít sdílený kvíz** ③.



Obr. 5.13: Kvíz 1

V prípade, že chceme vytvoriť úplne nový kvíz, zvolíme tlačidlo **Vytvořit nový kvíz** ④, ktorý otvorí dialóg s formulárom pre vytváranie kvízu (obrázok 5.14). Vo formulári sa nachádzajú textové polia pre vyplnenie názvu kvízu a tlačidlá **Přidat novou otázku** ⑤ a **Přidat možnost** ⑥. Pre uloženie novovytvoreného kvízu je potrebné stlačiť tlačidlo **Uložit** ⑦ čím sa obsah uloží a dialógové okno sa zatvorí.



Obr. 5.14: Kvíz 2

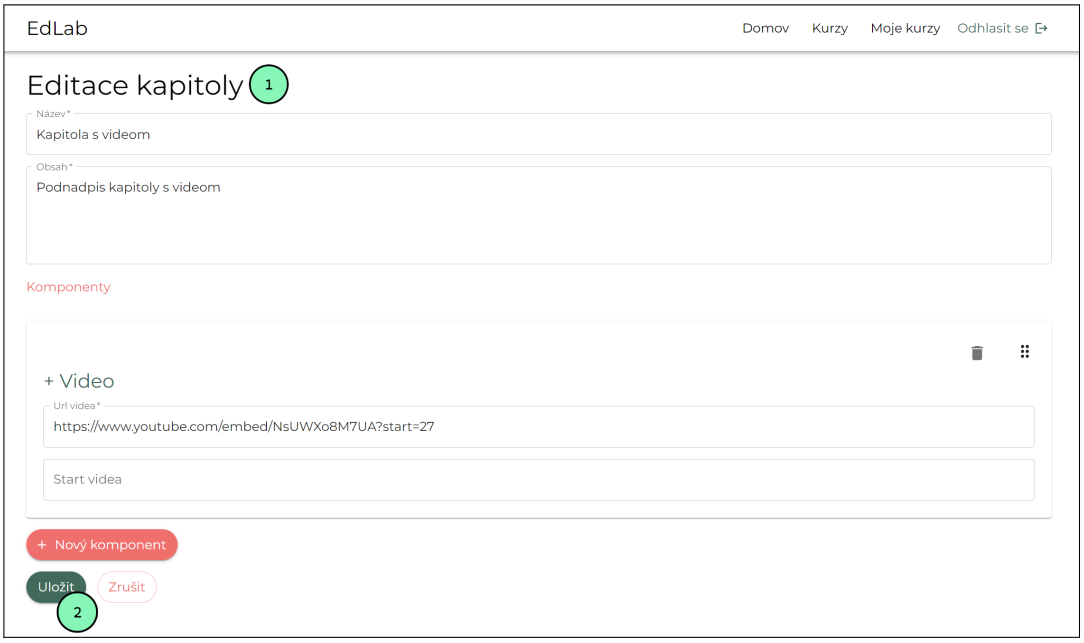
5.3.4.3 Pexeso

Pri vytváraní pexesa užívateľ postupuje obdobne ako pri vytváraní kvízu popísaného vyššie.

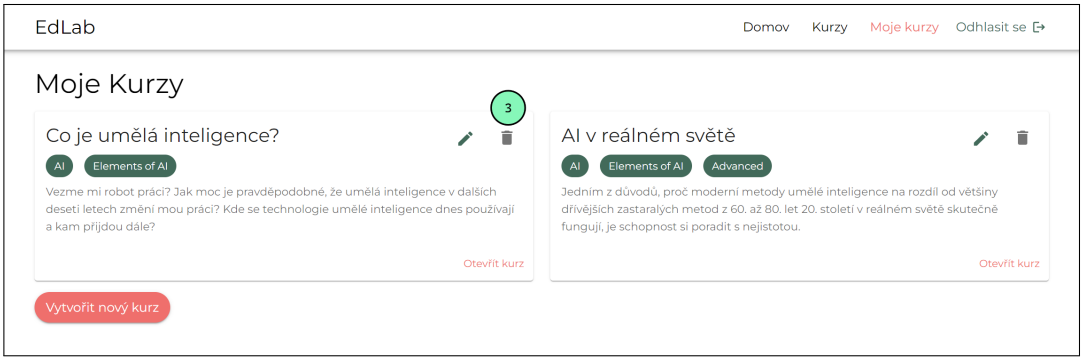
5.3.5 Editácia a odstránenie

Všetky komponenty, kapitoly a kurzy, ktoré daný užívateľ vytvoril môže editovať alebo mazať. Akciu editácie si užívateľ zvolí typicky kliknutím na ikonu Ceruzky. Táto akcia užívateľa presmeruje na stránku editácie (1), ktorá je podobná ako pri vytváraní nového prvku (viď obrázok 5.15). Aby boli zmeny ktoré pri editácii vykonal zaznamenané, je potrebné túto akciu uložiť stlačením tlačidla **Uložit** (2). Pri editácii Hlasovania sa zmení iba konkrétna inštancia. Pri zmene pexesa alebo kvízu (zdieľaného komponentu) sa zmena prejaví na všetkých miestach.

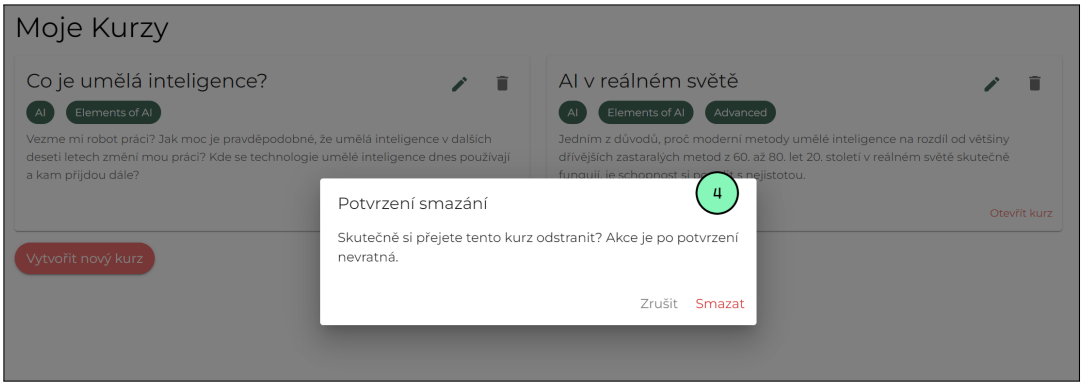
Ak chce užívateľ vymazať nejaký komponent, kurz alebo kapitolu, ktorú vytvoril musí kliknúť na ikonu Koša (3) (viď obrázok 5.16). V prípade vymazania komponentu z kapitoly, bude po uložení zmien komponent odstránený. Pri odstraňovaní kurzu alebo kapitoly, je užívateľ vyzvaný v dialógovom okne (4) aby svoju akciu potvrdil alebo zrušil (obrázok 5.17), po potvrdení sa daný prvok vymaže.



Obr. 5.15: Editácia



Obr. 5.16: Odstránenie 1



Obr. 5.17: Odstránenie 2

6. Užívateľská dokumentácia - admin

V tejto kapitole bližšie popíšeme všetky kroky, ktoré je potrebné urobiť, aby bolo možné aplikáciu prepojiť s vlastnou databázou Firebase a nasadiť na server.

6.1 Firebase

Webová aplikácia využíva Firebase na ukladanie dát a autentifikáciu užívateľov. Pre nasadenie aplikácie na server je potrebné mať vytvorený účet na Firebase a následne nastaviť Firebase konfiguráciu pre náš projekt. Po registrácii do firebase je potrebné Vytvoriť nový projekt. Po vytvorení projektu potrebujeme pridať prepojenie Firebase do projektu. Vyberieme si možnosť webovej aplikácie (v tomto kroku je možné pridať aj hosťovanie, ktoré Firebase ponúka). Následne postupujeme podľa pokynov, ktoré je možné nájsť na `https://firebase.google.com/docs/web/setup`. Skopírujeme obsah **firebaseConfig** a nahradíme ho v súbore `./edlab/src/firebase-config.js`. Týmto bude prepojenie Firebase s aplikáciou dokončené.

6.2 Úprava špecifických častí aplikácie

Firebase poskytuje rôzne metódy autentifikácie užívateľov, ako napríklad e-mail a heslo, sociálne siete, anonymný prihlasovací údaj atď. Pred nasadením aplikácie je potrebné nastaviť požadované metódy autentifikácie v konfigurácii aplikácie. V našom prípade je potrebné nastaviť autentifikáciu pomocou Microsoft účtu. Presný návod je popísaný v dokumentácií na stránke `https://firebase.google.com/docs/auth/web/microsoft-oauth`

6.3 Nasadenie aplikácie na server

Po vykonaní všetkých potrebných úprav je možné aplikáciu nasadiť na server. Aplikáciu je potrebné previesť do build štádia pomocou **npm build** a vygenerované súbory nasadiť na server alebo využiť ponúkanú možnosť hosťovania od Firebase (viď `https://firebase.google.com/docs/hosting`).

7. Záver

V závere našej práce si zhodnotíme naplnenie požiadaviek a cieľov aplikácie, ktoré sme si stanovili v úvodnej kapitole. Rozoberieme si užívateľské skúsenosti a spätnú väzbu a na základe nich vyvodíme ďalšie možnosti pre rozšírenie funkcionality aplikácie a spríjemnenie užívateľskej skúsenosti.

7.1 Užívateľská skúsenosť

V priebehu vývoja sme aplikáciu poskytli viacerým ľuďom v rôznom vekovom rozpätí v roli učiteľa aj študenta aby aplikáciu otestovali a poskytli nám spätnú väzbu. Celkovo sa aplikácia testerom používala jednoducho a priamočiaro.

Zopár študentov sa vyjadrilo, že by bolo dobré mať možnosť web prepnúť do tzv. *Dark Mode*. Takisto by aplikácia mohla slúžiť aj na overenie vedomostí, ted vypĺňanie testov a získavanie hodnotení.

Zo strany užívateľov v roli tvorcu obsahu sme dostali inšpiráciu, že by aplikácia po registrácii nového užívateľa mohla ponúknuť úvodný tutoriál, ako rýchlo a efektívne vytvárať a editovať obsah. To by sa mohlo primárne zísť technicky menej skúseným učiteľom vyššieho veku. Ďalším nápadom pre rozšírenie bola možnosť uzamykať kapitoly heslom alebo časovým zámkom.

7.2 Splnenie požiadaviek a cieľov

V tejto sekcii zhodnotíme naplnenie požiadaviek a cieľov práce definovaných v sekciách 1.4 a 1.6.

Vytvorili sme webovú aplikáciu, ktorá sprístupní online vzdelávanie pre verejnosť a tvorba jej obsahu bude jednoduchá a intuitívna, čím sme splnili prvý a tretí cieľ.

Čo sa týka splnenia druhého cieľa, ktorý hovorí o používaní bezplatných technológií, sme cieľ splnili čiastočne. Používanie technológie Firebase je síce aktuálne zadarmo, ale v prípade nadmierneho množstva denných užívateľov môže nastať spoplatnenie (viď sekcia 2.5.1). Všetky ostatné použité technológie sú zdarma.

Posledný bod stanovených cieľov hovorí o naplnení všetkých funkčných požiadaviek, čo sa nám podarilo v plnom rozsahu. Dokonca sme pridali aj možnosti nad rámec stanovených požiadaviek (napr. drag and drop funkcionalita spomínaná v 4.7.5).

7.3 Možnosti rozšírenia

Na základe spätnej väzby od užívateľov, ktorí aplikáciu testovali sme vytvorili zoznam možných rozšírení spolu s potenciálnou implementáciou.

- *Dark Mode.* Myšlienka možnosti výberu farebnej schémy by sa dala jednoducho navrhnúť fo Figme [1] a následne pomocou MUI témy [2] nakonfigurovať.

[1] https://www.figma.com/

[2] https://mui.com/material-ui/customization/theming/

- *Uzamykanie kapitoly heslom alebo časovým zámkom*, by sa mohlo dať jednoducho zadať počas vytvárania kapitoly. Overenie hesla by bolo potrebné overiť pred vstupom do kapitoly (napríklad pomocou dialógového okna) a následne úspešný výsledok overenia uložiť do cookies. Ak by kapitoly obsahovali časový zámok tak by sme pri ich načítaní museli filtrovať podľa ich prístupnosti. Problém, ktorý by bolo treba preskúmať je opätovné preverovanie, či nie sú dostupné ďalšie kapitoly v nejakom časovom intervale.

- *Testy.* V tomto prípade by sme museli vyriešiť ako by boli testy štruktúrované, či by patrili danej kapitole alebo existovali nezávisle. Ďalej by bolo potrebné vymyslieť, ako by sme zaručili to, že študent, ktorý test vyplnil je skutočne ten, za ktorého sa vydáva, nakoľko pre študentov nie je možné sa prihlásiť a tak overiť ich identitu.

- *Vytvorenie mobilnej aplikácie* je nápad autorky práce. Jej pridanú hodnotu by sme videli v lepšej dostupnosti pre študentov a napríklad takáto aplikácia by mohla vedieť fungovať čiastočne aj offline (jej obsah by sa updatoval počas pripojenia na internet a následne by ostal statický). Pravdepodobne by dávalo zmysel aby aplikácia slúžila len pre študentov a neposkytovala by možnosť vytvárania nového obsahu.

- *Úvodný tutoriál* pre novoregistrovaných užívateľov by sa dal jednoducho vytvoriť pomocou Reactovských komponentov a mohol by výrazne pomôcť menej technicky zdatným učiteľom pri vytváraní ich prvého obsahu.

- *Obnovenie hlasovania.* Táto funkcionalita je síce naplnená už v aktuálnej verzii, ale jej možnosť je len ako sekundárny dôsledok možnosti vytvárania hlasovacích šablón. Aktuálne vieme výsledky obnoviť tak, že vytvoríme novú inštanciu hlasovania z existujúcej šablóny a nahradíme pôvodné hlasovanie. Lepšie riešenie by však bolo pridať explicitné tlačidlo pre obnovenie výsledkov.

- *Rôzne behy kapitol.* S týmto nápadom prišiel vedúci práce a jeho myšlienka bola vedieť rozdeliť kapitoly na rôzne paralelné behy, kde by každý beh mal svoje vlastné dáta (napríklad výsledky hlasovaní a pod.) a bol by dostupný pomocou špeciálneho linku. Výhodu by bola možnosť mať pre nejakú špecifickú skupinu študentov takýto samostatný beh a vedieť oddeliť ich výsledky. Implementácia tejto funkcionality by však potrebovala hlbšiu analýzu možných riešení a pravdepodobne viaceré závažnejšie zásahy do architektúry databázy.

- *Preddefinované šablóny pre kapitoly* sú ďalším nápadom autorky a mohli by pomôcť pri rýchlejšom vytváraní náučného obsahu. Obsahovali by komponenty, ktoré sa najviac využívajú ale samozrejme by boli editovateľné.

Rozborom možných rozšírení sme chceli ukázať ako by sme jednotlivé časti vedeli implementovať. Z rozboru vyplýva, že aplikácia je navrhnutá tak, aby bola jednoducho rozšíriteľná čo je splnenie dôležitej kvalitatívnej požiadavky softwarového systému.

Seznam použité literatury

[1] MFF UK. Projekt AI v kontextu. https://ufal.mff.cuni.cz/AIvK/index.html. Accessed: 2023-4-24.

[2] MFF UK. Domovská stránka ÚFAL. https://ufal.mff.cuni.cz/home-page. Accessed: 2023-4-24.

[3] Ministerstvo školství, mládeže a tělovýchovy. Stav učitelů v regionálním školství 2019. https://www.msmt.cz/ministerstvo/novinar/ministerstvo-zjistovalo-stav-ucitelu-v-regionalnim-skolstvi, 2019. Accessed: 2023-5-29.

[4] OpenLearning. Openlearning - course creator walkthrough. https://www.youtube.com/watch?v=KT4OXyPvUgU&feature=youtu.be. Accessed: 2023-4-24.

[5] Aleksandra Czajka. Front-end, Back-end and Database-side – The Structure of an App. https://www.linkedin.com/pulse/front-end-back-end-database-side-structure-app-aleksandra-czajka/, 2018. Accessed: 2023-5-29.

[6] Stack Overflow. 2022 developer survey. https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe. Accessed: 2023-6-1.

[7] Google. Firebase Documentation. https://firebase.google.com/docs/, 2021. Accessed: 2023-4-19.

[8] Google. Firebase pricing plans. https://firebase.google.com/pricing. Accessed: 2023-4-24.

[9] Interaction Design Foundation. Design Systems. https://www.interaction-design.org/literature/topics/design-systems. Accessed: 2023-4-26.

[10] Therese Fessenden. Design Systems 101. https://www.nngroup.com/articles/design-systems-101/. Accessed: 2023-4-26.

[11] Microsoft. Fluent Design System. https://www.microsoft.com/design/fluent/. Accessed: 2023-4-26.

[12] Google. Material Design System. https://m3.material.io/. Accessed: 2023-4-26.

[13] UXPin. Top 8 Design System Examples. https://www.uxpin.com/studio/blog/best-design-system-examples/. Accessed: 2023-4-26.

[14] Bito. Best React UI frameworks to boost Productivity. https://www.linkedin.com/pulse/best-react-ui-frameworks-boost-productivity-bitodev/, 2023. Accessed: 2023-6-10.

[15] Google. Components - material design 3. https://m3.material.io/components. Accessed: 2023-4-19.

Prílohy

Prehľad elektronických príloh

- edlab – priečinok obsahujúci zdrojové kódy aplikácie, sekcia 4.3
  - public
  - src
- bakalarska_praca – adresár obsahujúci zdrojové súbory textu práce
  - export_figma – priečinok obsahujíci vyexportovaný dizajnový návrh obrazoviek z Figmy
  - img – priečinok obsahujúci obrázky použité v práci
  - sk – samotné .tex súbory obsahujúce text práce v slovenčine
  - praca.pdf – samtoný pdf súbor obsahujúci text práce vo formáte PDF/A
- README.md – súbor popisujúci celý adresár a prácu s ním