

AASS - Elektronický edukačný softvér

Bc. Martin Šváb, Bc. Patrik Harmaňoš

Máj 2023

Contents

1	Úvod	3
2	Návrh	4
2.1	Funkčné požiadavky	4
2.2	Aktéri a stakeholderi	4
2.2.1	Systémoví aktéri	4
2.2.2	Ľudskí aktéri/stakeholderi	4
2.3	Wireframy	5
2.3.1	Registrácia	5
2.3.2	Prihlásenie	6
2.3.3	Úlohy	6
3	Architektúra	8
3.1	Diagram požiadaviek	8
3.2	Motivačný diagram	9
3.3	Biznis spôsobilosti	10
3.4	Tok hodnôt	10
3.5	Biznis architektúra	11
3.6	Biznis proces	12
3.7	Aplikačná architektúra	13
3.8	Technologická architektúra	14
3.9	Dátová architektúra	15
3.10	Prípady použitia	16
4	Implementácia	17
4.1	Front-end aplikácia	17
4.1.1	Registrácia	17
4.1.2	Prihlásenie	18
4.1.3	Úlohy	19
4.2	Back-end aplikácia	21
4.2.1	Mikroslužby - implementácia	21
4.2.2	Camunda - implementácia	22
4.2.3	Kafka - implementácia	27
5	Záver	32

1 Úvod

Cieľom nášho projektu je vytvoriť webovú aplikáciu, ktorá umožňuje učiteľom pridávať úlohy a študentom ich prezeranie a následne vypracovanie. Učitelia môžu úlohy v prípade potreby zmeniť. Študenti sú priradení do kurzov v ktorých majú úlohy, ktoré môžu riešiť a vidieť výstup svojho riešenia. V projekte sa pozrieme na návrh riešenia, implementáciu klientskej časti a taktiež rôzne spôsoby implementácie serverovej časti aplikácie.

2 Návrh

2.1 Funkčné požiadavky

1 Manažment používateľských účtov
1.1 Autentifikácia
1.2 Registrácia
1.3 Typy používateľov s rôznymi oprávneniami

2 Manažment úloh
2.1 Tvorba úloh
2.2 Úprava úloh
2.3 Zmazanie úlohy
2.4 Zobrazenie úlohy
2.5 Riešenie úlohy

2.2 Aktéri a stakeholderi

2.2.1 Systémoví aktéri

E-shop	Webová stránka, cez ktorú si môžu záujemcovia prezerať a kupovať kurzy
Vzdelávacia platforma	Webová aplikácia, cez ktorú sa budú používatelia vzdelávať, prezerať materiály a písať testy
Server	Slúži na poskytovanie služieb a obsluhu databázy
Databáza	Slúži na uchovanie všetkých dát pre e-shop a vzdelávaciu platformu
Zariadenie pre učiteľa	Učiteľ cez neho pristupuje na vzdelávaciu platformu
Zariadenie pre študenta	Žiaci cez neho pristupujú na platformu

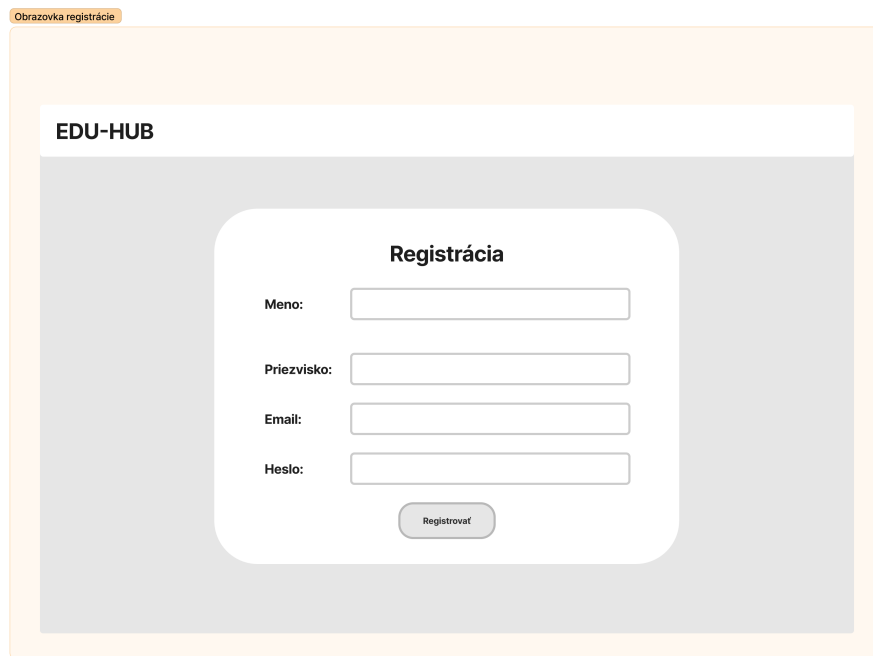
2.2.2 Ľudskí aktéri/stakeholderi

Majiteľ	Vlastník a riaditeľ celého vzdelávacieho systému
Administrátor systému	Administrátor systému, má za úlohu správu kurzov, úloh, používateľov a údržbu systému
Učiteľ	Stará sa o vzdelávanie študentov, tvorbu materiálov a testov
Používateľ	Žiak, ktorý sa vzdeláva, je súčasťou kurzov

2.3 Wireframy

Pred implementáciou jednotlivých obrazoviek vo webovej aplikácii sme si ich navrhli pomocou nasledujúcich wireframov vytvorených vo Figma. Oproti finálnej implementácii boli spravené len mierne zmeny ako opakovanie zadávania hesla v registrácii, zmena farebnej palety a umiestnenie niektorých objektov na obrazovke.

2.3.1 Registrácia



Obrazovka registrácie

EDU-HUB

Registrácia

Meno:

Priezvisko:

Email:

Heslo:

Registrovať

Figure 1: Návrh obrazovky registrácie

2.3.2 Prihlásenie

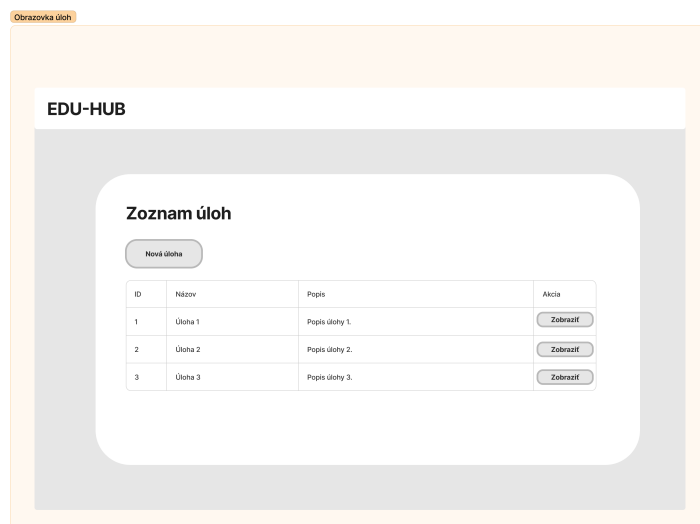


Figure 2: Návrh obrazovky Prihlásenia

2.3.3 Úlohy

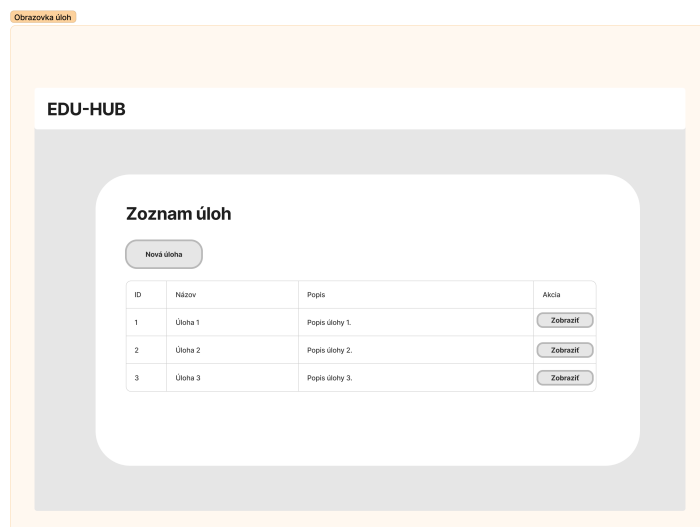


Figure 3: Návrh obrazovky úloh

Obrazovka tvorby úlohy

EDU-HUB

Späť

Nová úloha

Názov:

Zadanie:

Vytvoriť

Figure 4: Návrh obrazovky tvorby úloh

Obrazovka riešenia úlohy

EDU-HUB

Späť

Riešenie úlohy

Názov: názov úlohy

Zadanie: zadanie úlohy

Riešenie:

Odozvať

Figure 5: Návrh obrazovky riešenia úloh

3 Architektúra

3.1 Diagram požiadaviek

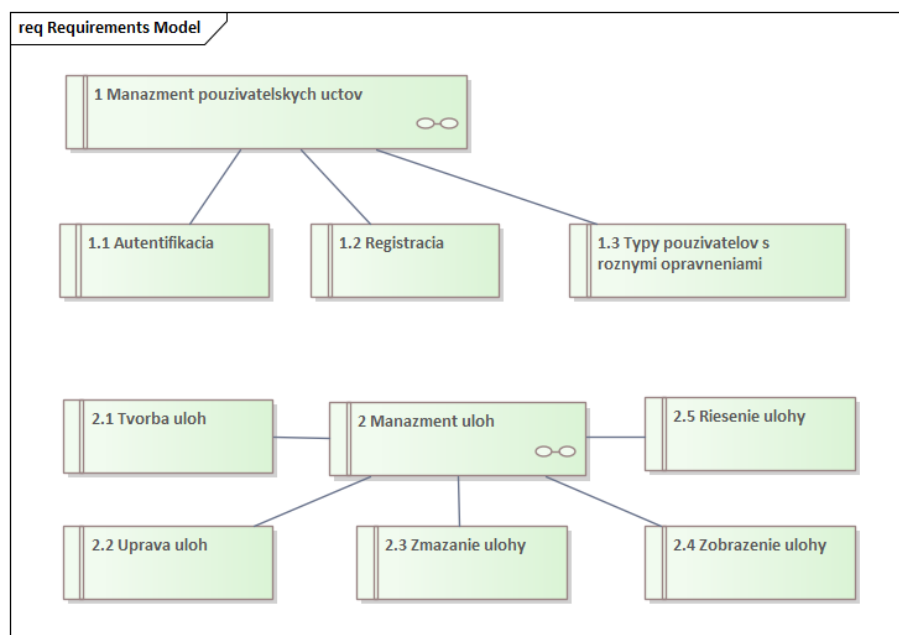


Figure 6: Diagram požiadaviek

V diagrame požiadaviek máme zadefinované funkčné požiadavky pre našu webovú aplikáciu. Máme tu 2 hlavné funkčné požiadavky - manažment používateľských účtov a manažment úloh. Manažment používateľských úloh sa skladá z autentifikácie, registrácie a typov používateľov s rôznymi oprávneniami. Manažment úloh sa skladá z tvorby úloh, úprav úloh, vymazania úlohy, zobrazenia úlohy a riešenia úlohy.

3.2 Motivačný diagram

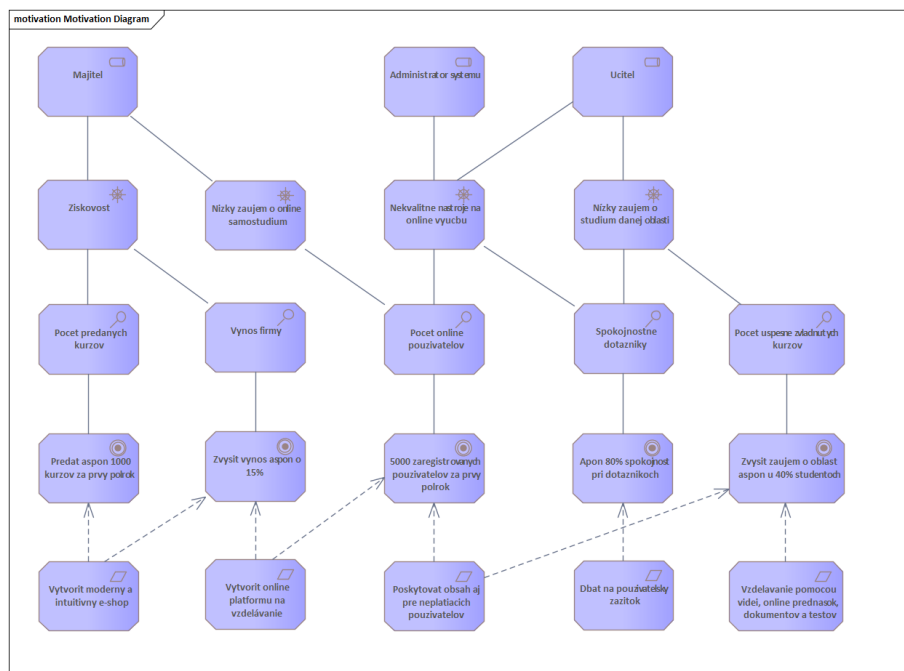


Figure 7: Motivačný diagram

V tomto diagrame máme zadaných 3 stakeholderov - majiteľa, administrátora systému a učiteľa.

Učiteľ je motivovaný nízkym záujmom o štúdiu v danej oblasti a nekvalitným nástrojom na online výučbu. Administrátor je motivovaný nekvalitnými nástrojmi na online výučbu. Majiteľ je motivovaný ziskovosťou a nízkym záujmom o online samostúdium. Z týchto motivácií máme ciele, ktoré môžeme vyčítať z diagramu.

Na spode diagramu máme nasledujúce požiadavky - vytvoriť moderný a intuitívny e-shop, vytvoriť online platformu na vzdelávanie, poskytovať obsah aj pre neplatiacich používateľov, dbať na používateľský zážitok a vzdelávanie pomocou videí, online prednášok, dokumentov a testov. Splnenie týchto požiadaviek môžeme overiť pomocou zadaní, ktoré môžeme vyčítať z diagramu.

3.3 Biznis spôsobilosti

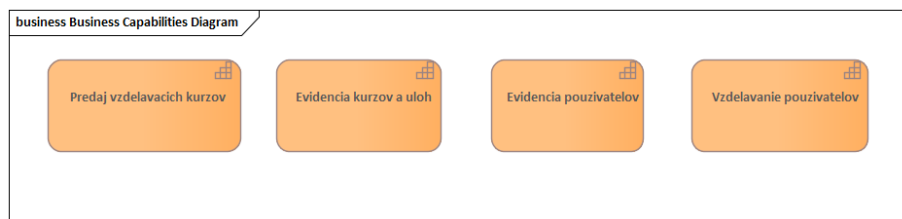


Figure 8: Biznis spôsobilosti

V tomto diagrame máme zadané 4 spôsobilosti, ktoré využívame vo viacerých diagramoch - predaj vzdelávacích kurzov, evidencia kurzov a úloh, evidencia používateľov a vzdelávanie používateľov.

3.4 Tok hodnôt

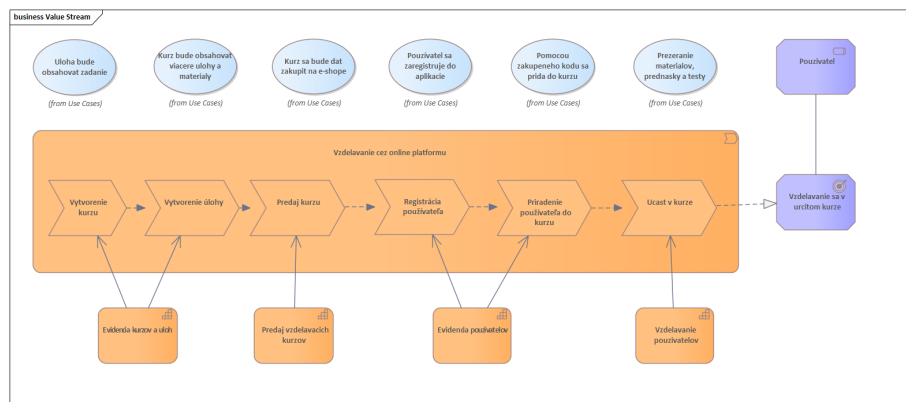


Figure 9: Tok hodnôt

V toku hodnôt máme na spodu diagramu spôsobilosti z diagramu 8. V toku hodnôt sekvenčne nasledujú vytvorenie kurzu, vytvorenie úlohy, predaj kurzu, registrácia používateľa, priradenie používateľa do kurzu a účasť v kurze. z posledného toku získavame dosiahnutý cieľ "vzdelávanie sa v určitom kurze". Tento cieľ chce dosiahnuť používateľ aplikácie.

3.5 Biznis architektúra

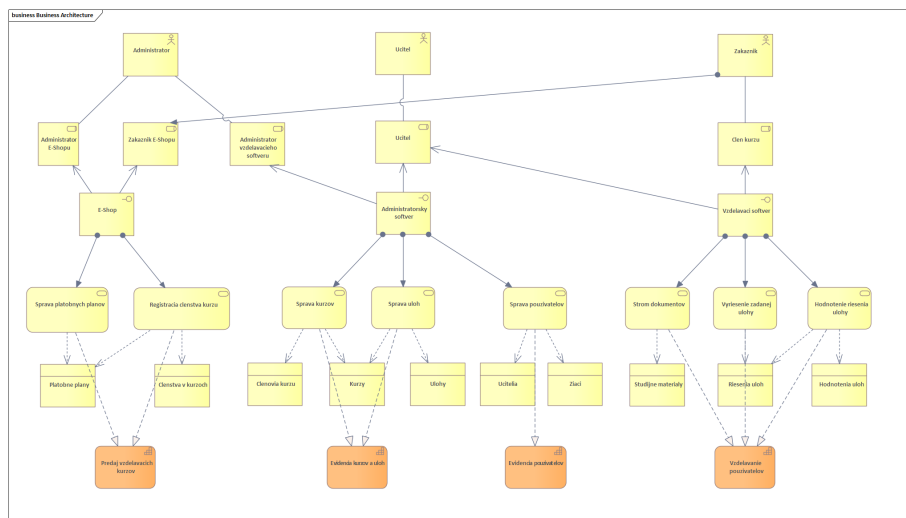


Figure 10: Biznis architektúra

V našej biznis architektúre máme na spode diagramu zadané biznis spôsobilosti z diagramu 8. Predaj vzdelávacích kurzov má k sebe priradené funkcie správa platobných plánov a registrácia členstva kurzu. Evidencia kurzov a úloh má k sebe priradené funkcie správa kurzov a správa úloh. Evidencia používateľov má k sebe priradenú funkciu správa používateľov. Vzdelávanie používateľov má k sebe priradené funkcie strom dokumentov, vyriešenie zadanej úlohy a hodnotenie vyriešenej úlohy. Spomenuté funkcie pristupujú k potrebným objektom, ktoré môžete vyčítať z diagramu.

Na vrchu diagramu máme zadaných 3 aktérov - administrátor, učiteľ a zákazník. Administrátor môže zastávať rolu administrátora E-Shopu a administrátora vzdelávacieho softvéru. Učiteľ môže zastávať rolu učiteľa. Zákazník môže zastávať rolu zákazníka E-Shopu a člena kurzu. Títo aktéri pristupujú k biznis rozhraniam - E-Shop, administrátorský softvér a vzdelávací softvér. Pomocou týchto rozhraní pristupujú k spomenutým biznis funkciám čo môžete vyčítať z diagramu.

3.6 Biznis proces

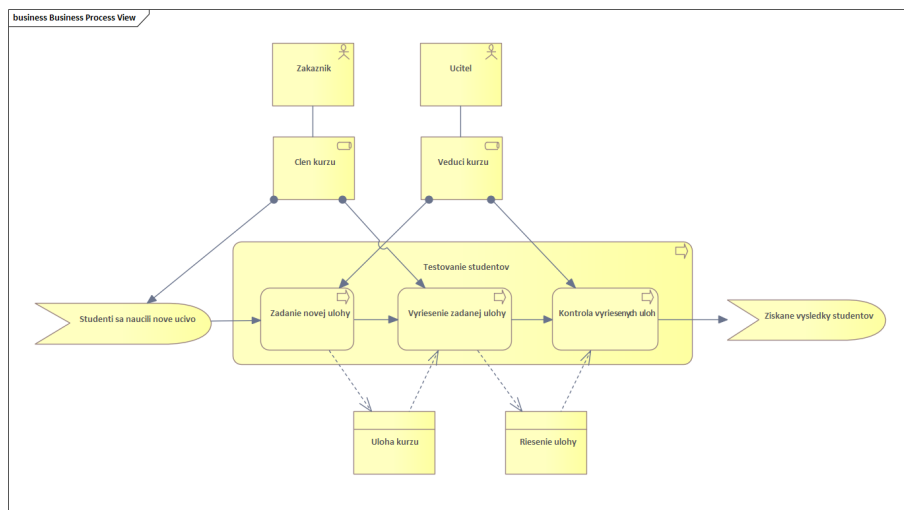


Figure 11: Biznis proces

V tomto diagrame je opísaný proces testovania študentov. Tento proces sa začína tým, že študenti sa naučia nové učivo. Následne sa zadá nová úloha, ktorú študenti vyriešia a tieto riešenie učiteľia skontrolujú a hodnotia. Proces sa končí tým, že získame výsledky študentov. Do tohto procesu pristupujú učiteľ ako vedúci kurzu a študenti ako členovia kurzu.

3.7 Aplikačná architektúra

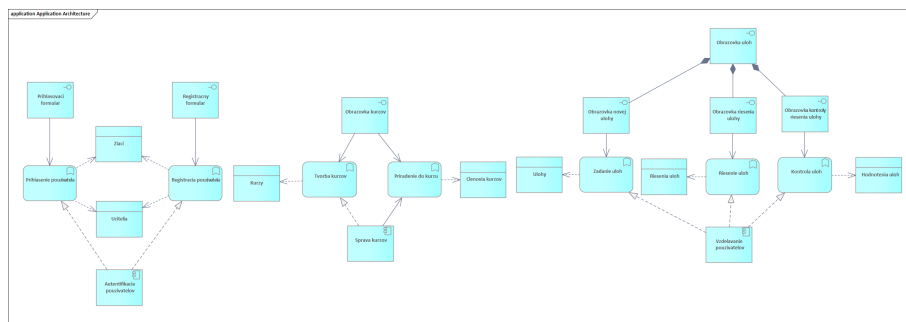


Figure 12: Aplikačná architektúra

V našej aplikačnej architektúre máme na spode diagramu zadané 3 aplikačné komponenty - autentifikácia používateľov, správa kurzov a vzdelávanie používateľov.

Komponent autentifikácie používateľov poskytuje funkcie registrácie používateľa a prihlásenia používateľa. Tieto funkcie prístupujú ku objektom žiakov a učiteľov.

Komponent správy kurzov poskytuje funkcie tvorby kurzov a priradenia do kurzu. Funkcia tvorby kurzov prístupuje ku objektu kurzov. Funkcia priradenia do kurzu prístupuje ku objektu členov kurzu.

Komponent vzdelávania používateľov poskytuje funkcie zadania úloh, riešenia úloh a kontroly úloh. Funkcia zadania úloh prístupuje ku objektu úloh. Funkcia riešenia úloh prístupuje ku objektu riešenia úloh. Funkcia kontroly úloh prístupuje ku objektu hodnotenia úloh.

Na vrchu diagramu sa nachádzajú rozhrania pomocou ktorých prístupuje ku funkciám aplikačnej architektúry. Rozhranie obrazovky úloh sa skladá z 3 rozhraní - obrazovka úloh, ktorá prístupuje k funkcii zadania úloh, obrazovka riešenia úloh, ktorá prístupuje ku funkcii riešenia úloh a obrazovka kontroly riešenia úloh, ktorá prístupuje ku funkcii kontroly úloh. Rozhranie obrazovky kurzov prístupuje ku funkciám tvorby kurzov a priradenia do kurzu. Rozhranie registračného formuláru prístupuje ku funkcii registrácie používateľa. Rozhranie prihlasovacieho formuláru prístupuje ku funkcii prihlásenia používateľa.

3.8 Technologická architektúra

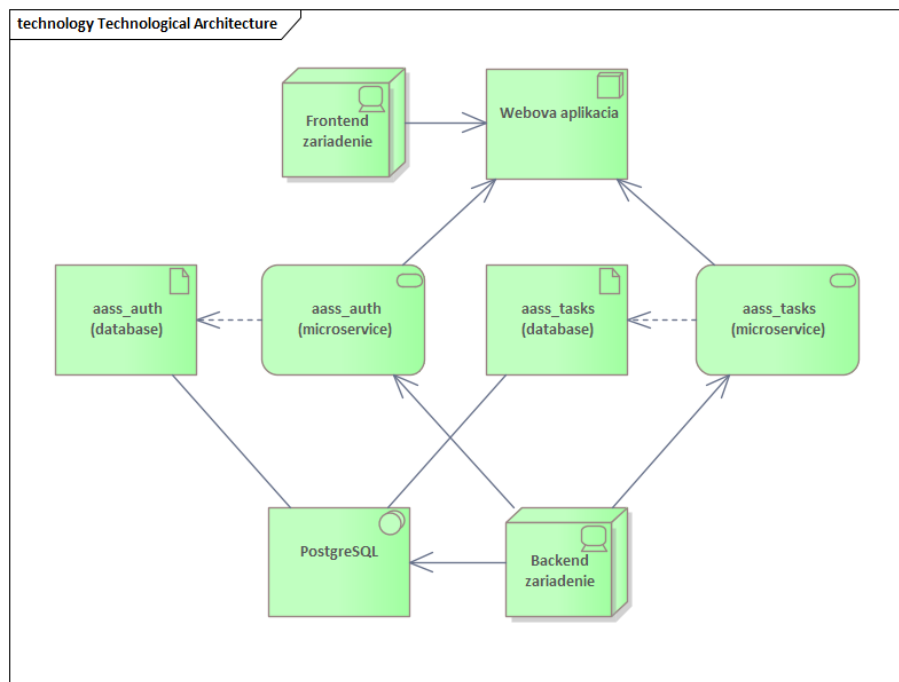


Figure 13: Technologická architektúra

V technologickej architektúre máme 2 zariadenia. Na prvom zariadení označenom ako "Backend zariadenie" je zavedený databázový server PostgreSQL a sú tam dostupné mikroslužby. Mikroslužby prístupujú ku databázam. Na druhom zariadení označenom ako "Frontend zariadenie" je zavedená webová služba, ktorá prístupuje k mikroslužbám.

3.9 Dátová architektúra

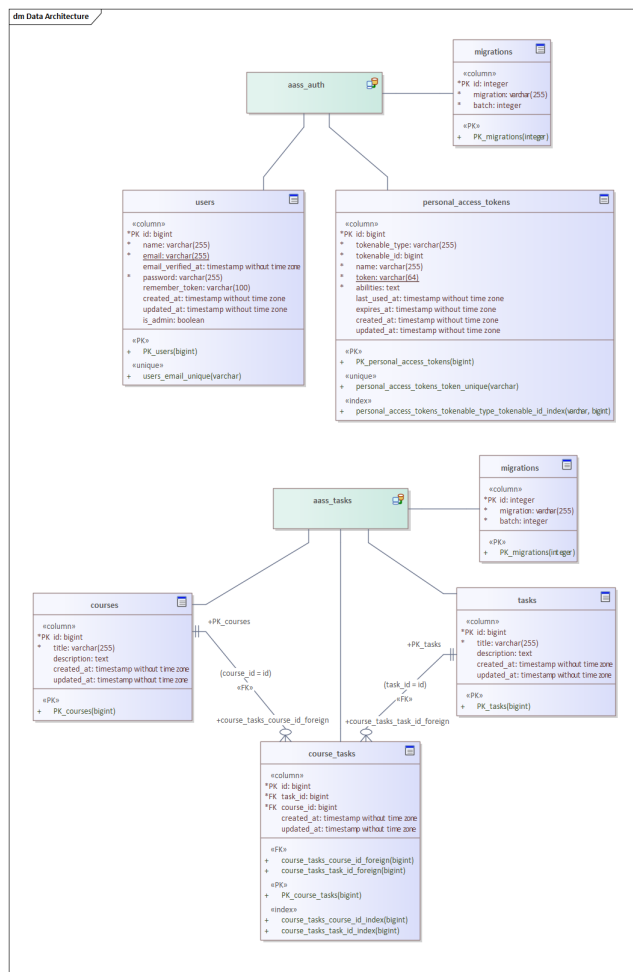


Figure 14: Dátová architektúra

V dátovej architektúre máme 2 databázy PostgreSQL. Všetky databázy obsahujú migračnú tabuľku. S prvou databázou označenou ako "aass_auth" pracuje mikroslužba autentifikácie. Nachádzajú sa tu 2 tabuľky (+ migračná tabuľka) - tabuľka používateľov a tabuľka prístupových tokenov. S druhou databázou označenou ako "aass_tasks" pracuje mikroslužba autentifikácie. Nachádzajú sa tu 3 tabuľky (+ migračná tabuľka) - tabuľka úloh, tabuľka kurzov a tabuľka kurzových úloh. Pomocou tabuľky kurzových úloh môžeme vytvoriť spojenie medzi kurzami a úlohami. Je tu implementovaná M:M relácia - jeden kurz môže mať viac úloh a jedna úloha môže patriť viacerým kurzom.

3.10 Prípady použitia

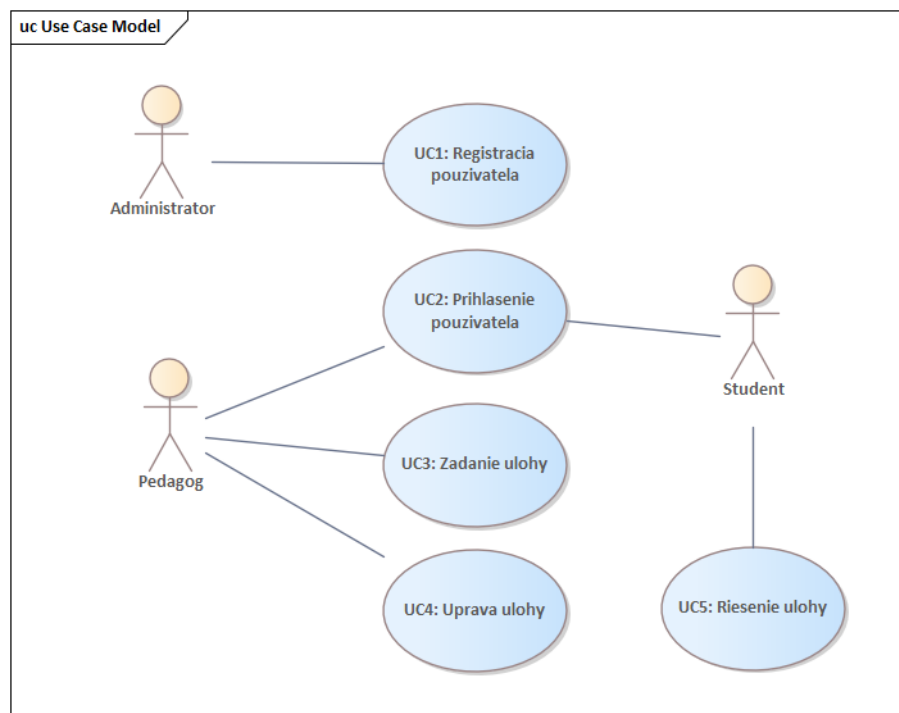


Figure 15: Prípady použitia

V prípadoch použitia máme zadaných 3 aktérov - administrátora, pedagóga a študenta. Administrátor môže zaregistrovať používateľov. Učiteľ sa môže prihlásiť, zadať úlohy a upraviť úlohy. Študent sa môže prihlásiť a riešiť úlohy. Spolu máme 5 prípadov použitia.

4 Implementácia

4.1 Front-end aplikácia

Klientská časť našej aplikácie je implementovaná pomocou Javascriptového frameworku Vue.js. Pri implementácii sme využili taktiež jazyk TypeScript a komponentovú knižnicu Quasar. Aplikácia je napojiteľná na akýkoľvek backend. Aplikácia komunikuje so serverom pomocou REST API. Na https komunikáciu sme použili knižnicu axios.

4.1.1 Registrácia

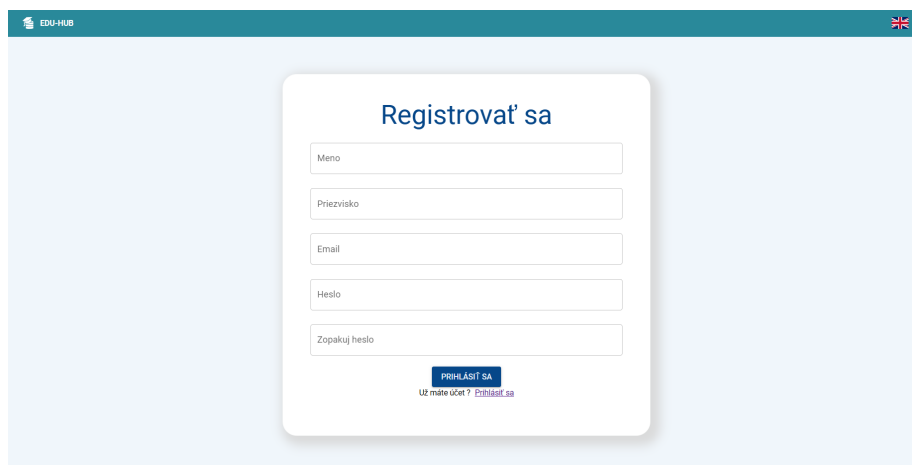


Figure 16: Obrazovka registrácie

Ak používateľ nemá v aplikácii konto, tak sa musí registrovať v obrazovke registrácie. V tejto obrazovke sa nachádza registračný formulár. Všetky jeho polia sú povinné. Pri registrácii musí používateľ zadať svoje meno a priezvisko, emailovú adresu a heslo. Validita emailovej adresy je kontrolovaná. Heslo sa musí zadať dvakrát aby sa zabránilo chybe pri vytváraní konta. Pri registrácii je vynútená politika hesiel. Heslo musí mať minimálne 8 znakov a musí obsahovať aspoň 1 malé písmeno, 1 veľké písmeno a 1 číslo. Ak používateľ zadá heslo, ktoré sa neriadi nedodržia tieto pravidlá, tak sa registrácia zamietne. Heslo je pri zadávaní skryté, ak chcete prepnúť čitateľnosť hesla, tak stlačte ikonu oka vpravo v poli zadaného hesla. Po vyplnení korektných údajov v registračnom formulári sa môžete registrovať stlačením tlačidla "REGISTROVAŤ SA". Ak registrácia zlyhá, tak sa vám na spode registračného formulára zobrazí chybová hláška. Ak už ste registrovaný, tak sa môžete dostať na obrazovku prihlásenia cez hyperlink pod tlačidlom registrácie "Už máte účet ? [Prihlásiť sa](#)".

4.1.2 Prihlásenie

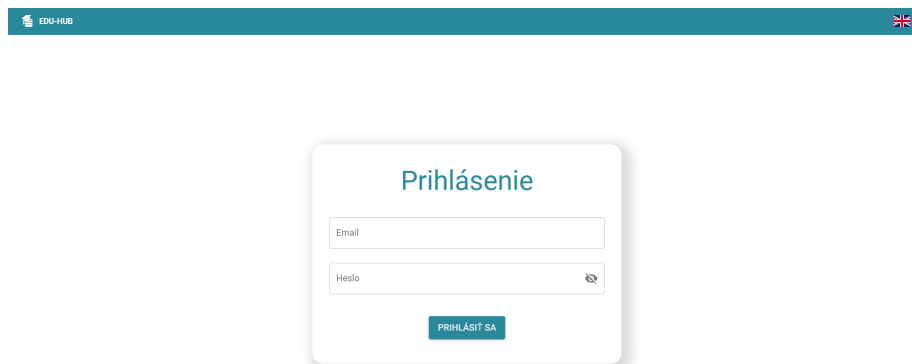
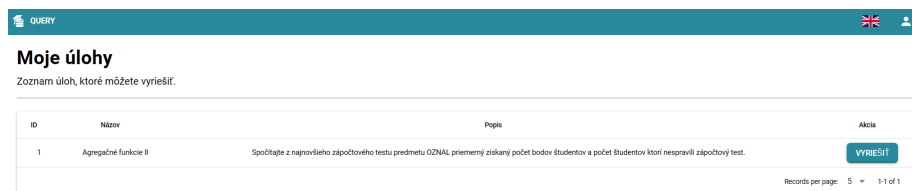
The image shows a login form titled "Prihlásenie" (Login) in a teal header bar. The form itself is a white box with a teal border. It contains two input fields: "Email" and "Heslo" (Password). The "Heslo" field has a small eye icon to its right for toggling visibility. Below the fields is a teal button with the text "PRIHLÁSIŤ SA" (Login) in white. The header bar also contains a small "EDU-HUB" logo on the left and a small red flag icon on the right.

Figure 17: Obrazovka prihlásenia

Ak používateľ už je registrovaný, tak sa môže prihlásiť v obrazovke prihlásenia. V tejto obrazovke sa nachádza prihlasovací formulár. Pri prihlasovaní musí používateľ zadať emailovú adresu konta do ktorého sa chce prihlásiť a správne heslo. Heslo je pri zadávaní skryté, ak chcete prepnúť čitateľnosť hesla, tak stlačte ikonu oka vpravo v poli zadaného hesla. Po vyplnení prihlasovacích údajov v prihlasovacom formulári sa môžete prihlásiť stlačením tlačidla "PRIHLÁSIŤ SA". Ak používateľ zadá emailovú adresu nesprávneho konta alebo zadá nesprávne heslo pre zadaný účet, tak sa prihlásenie zamietne. Ak prihlásenie zlyhá, tak sa vám na spode prihlasovacieho formulára zobrazí chybová hláška. Pri úspešnom prihlásení sa presuniete do obrazovky úloh.

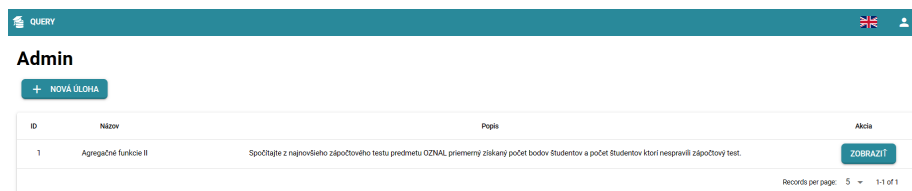
Po prihlásení sa môžete z účtu odhlásiť stlačením ikony postavy v navigačnom paneli vpravo hore a nasledovne stlačením tlačidla v zobrazenom zozname "Odhlásiť sa".

4.1.3 Úlohy



ID	Názov	Popis	Akcia
1	Agregačné funkcie II	Spočítajte z najnovšieho zápočtového testu predmetu OZNAL priemerný získaný počet bodov študentov a počet študentov ktorí nespravili zápočtový test.	VYRIEŠIť

Figure 18: Obrazovka úloh (študent)



ID	Názov	Popis	Akcia
1	Agregačné funkcie II	Spočítajte z najnovšieho zápočtového testu predmetu OZNAL priemerný získaný počet bodov študentov a počet študentov ktorí nespravili zápočtový test.	ZOBRAZIť

Figure 19: Obrazovka úloh (učiteľ)

V obrazovke úloh sa nachádza zoznam úloh. Študenti a učitelia majú prístup k rozdielnym verziám tejto obrazovky. Učitelia majú v tejto obrazovke aj tlačidlo na vytvorenie úloh. Zoznam úloh je stránkovaný. Ak chcete zmeniť zobrazovanú stránku použít tlačidlá so šípkami vpravo dole. Ak chcete zmeniť počet záznamov v stránke, tak zmeňte hodnotu čísla vpravo dole označenú ako "Record per page". Pre vyriešenie úlohy stlačte tlačidlo "ZOBRAZIť" vpravo pri konkrétnej úlohe. Pre vytvorenie novej úlohy stlačte tlačidlo vľavo hore "NOVÁ ÚLOHA".

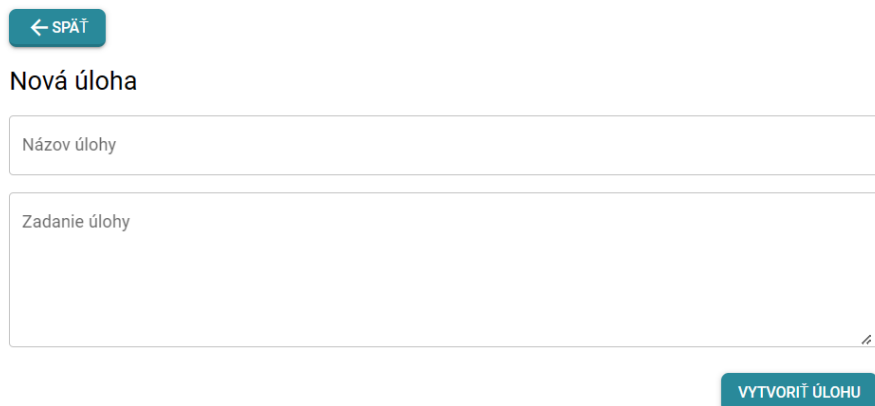


Figure 20: Obrazovka tvorby úlohy

V obrazovke tvorby úlohy môže učiteľ vytvoriť novú úlohu. Úloha musí mať zadaný názov. Jej popis je dobrovoľný ale odporúča sa ho zadať. Po zadaní informácii o novej úlohe, tvorbu úlohu môžete potvrdiť stlačením tlačidla vpravo dole "VYTVORIŤ ÚLOHU". Ak chcete tvorbu úlohy zrušiť, tak sa môžete vrátiť do obrazovky úloh stlačením tlačidla vľavo hore "SPÄŤ".

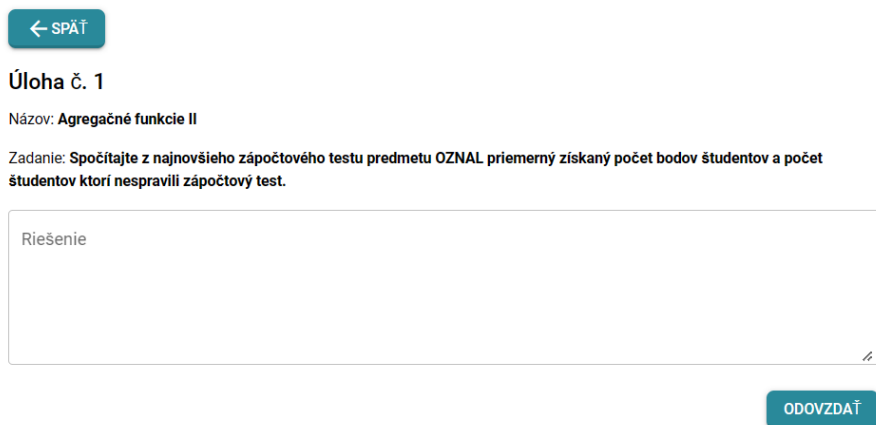


Figure 21: Obrazovka riešenia úlohy

V obrazovke riešenia úlohy môže študent vyriešiť zadanú úlohu. Tu môžete vidieť názov a popis zobrazenej úlohy. Dole máte pole na zadanie riešenia úlohy. Po zadaní riešenia môžete potvrdiť odovzdanie riešenia stlačením tlačidla vpravo dole "ODOVZDAŤ". Ak chcete riešenie úlohy, tak sa môžete vrátiť do obrazovky úloh stlačením tlačidla vľavo hore "SPÄŤ".

4.2 Back-end aplikácia

Serverovú časť našej aplikácie implementujeme najprv ako architektúru mikroslužieb, následne s využitím Cammunda a nakoniec s využitím Kafky. Frontendovú časť aplikácie bude možné napojiť na ktorúkoľvek implementáciu serverovaj časti.

4.2.1 Mikroslužby - implementácia

Serverovú časť aplikácie sme najprv implementovali ako architektúru mikroslužieb pomocou frameworku Laravel v jazyku PHP. Využili sme databázu PostgreSQL. Aplikácia má jednu hlavnú službu s názvom MainService, ktorá bude komunikovať s ďalšími dvoma službami: AuthService a TasksService.

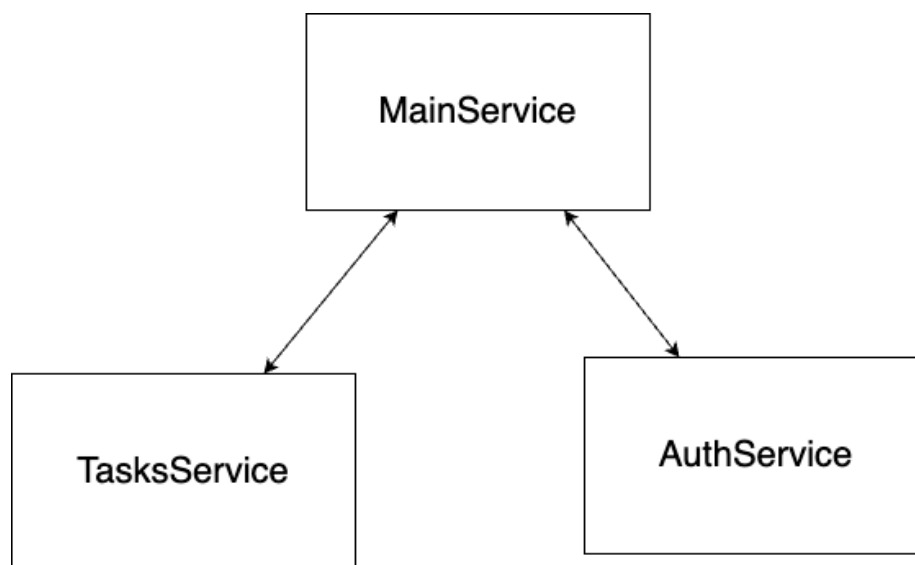


Figure 22: Jednotlivé mikroslužby

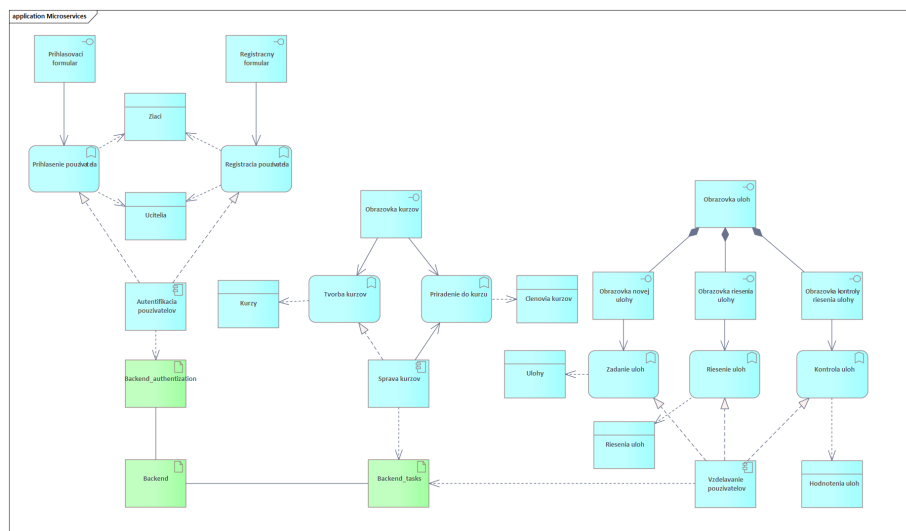


Figure 23: Aplikačná architektúra mikroslužieb

Frontend bude posilať požiadavky pomocou REST API na Main Service, ktorý v prípade potreby bude komunikovať s ďalšími mikroslužbami. Každá z týchto mikroslužieb je vlastný Laravel projekt a je napojená na samostatnú databázu. V nasledujúcom príklade vidíme funkciu login v službe MainService, ktorá volá REST API z Auth Service mikroslužby.

```
public function login(LoginRequest $request): JsonResponse
{
    try {
        $response = Http::post('http://localhost:8001/api/auth/login', [
            'email' => $request->email,
            'password' => $request->password
        ]);
        $data = $response->json();

        return $this->responseSuccess($data['data']);
    } catch (\Exception $e) {
        return $this->responseError(['errors' => null, $e->getMessage(), 'status_code' => Response::HTTP_INTERNAL_SERVER_ERROR]);
    }
}
```

Figure 24: Ukážka funkcie v Laraveli

4.2.2 Cammunda - implementácia

Pri implementácii Cammundy sme nedokázali znovupoužiť projekt v Laraveli z dôvodu nekompatibility s Cammundou. Museli sme teda zvoliť iné riešenie v podobe node.js, v ktorom sme vytvorili nový projekt. Teraz budú viaceré služby v rámci jedného projektu pre jednoduchosť implementácie.

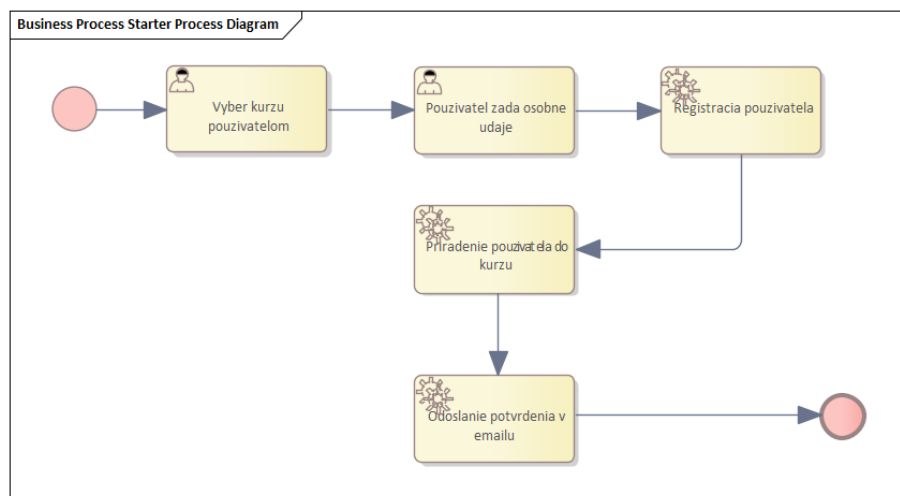


Figure 25: Diagram procesu v Camunde

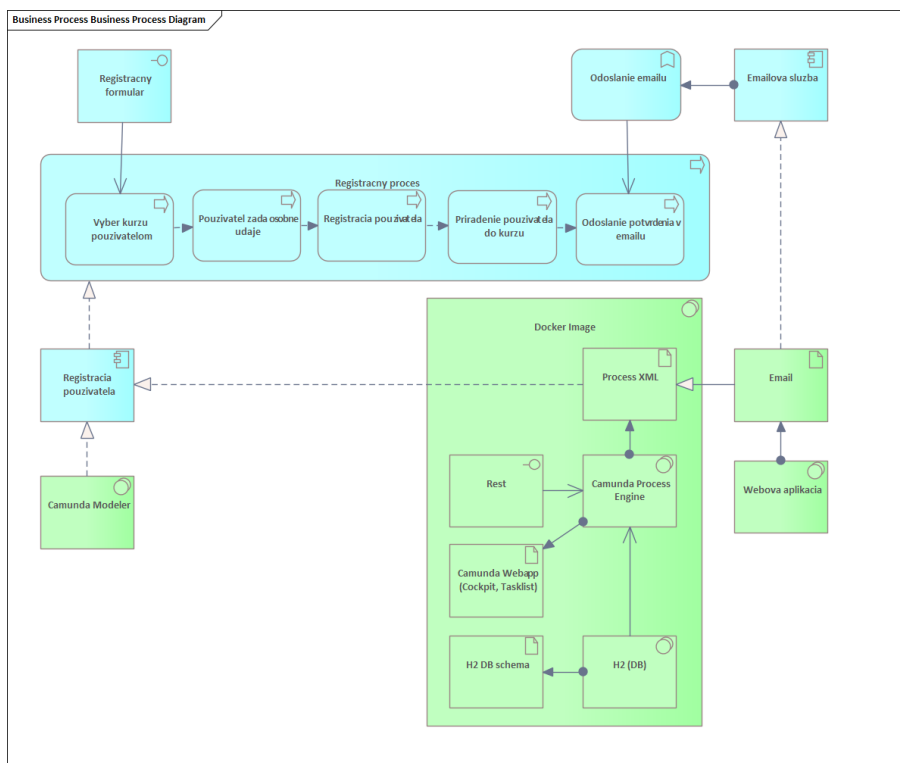


Figure 26: Rozšírený diagram procesu v Camunde

V tomto diagrame máme zobrazený proces registrácie používateľa a následného priradenia do kurzu. Proces sa začína výberom kurzu používateľom a následne používateľ zadá osobné údaje. Potom nasledujú automatické systémove akcie. Používateľ je systémom registrovaný, je priradený do kurzu a na jeho emailovú adresu sa odošle potvrdenie. Cammunda sme spúšťali cez Docker a diagram sme namodelovali v programe Cammunda Modeler.



Figure 27: Cammunda Modeler proces

Na začiatku sa nachádza formulár, ktorý natiahneme zo súboru registration.form a obsahuje dáta, ktoré používateľ zadá na frontende.

The screenshot shows the Cammunda Modeler interface with two tabs: 'register.form' and 'payment.form'. The 'Form Definition' view is active, displaying a form with the following fields:

- firstName: Text input field
- lastName: Text input field
- email: Text input field
- password: Text input field
- courseId: Dropdown menu

A 'Form Preview' tab is visible on the right side of the interface.

Figure 28: Cammunda Modeler formulár

Ukážeme si simuláciu cez aplikáciu Postman, ktorá bude odosielať požiadavky na backend namiesto frontendovej aplikácie. Najprv zavoláme endpoint na inicializáciu procesu pomocou názvu procesu. Zavoláme endpoint `http://localhost:8080/engine-rest/process-definition/key/payment-retrieval/start`, kde pošleme v body údaje z formulára. V odpovedi sa nám vráti process ID.

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/engine-rest/process-definition/key/payment-retrieval/start`. The request body is a JSON object with variables for user information and course details. The response body is a JSON object containing a link to the process instance and its ID.

```
1 {
2   "variables": {
3     "firstName": {
4       "value": "Patrik"
5     },
6     "lastName": {
7       "value": "Harmanos"
8     },
9     "email": {
10      "value": "patrik@gmail.com"
11    },
12    "password": {
13      "value": "patrik123"
14    },
15    "courseId": {
16      "value": 1,
17      "type": "integer"
18    }
19  }
20 }
```

```
1 {
2   "links": [
3     {
4       "method": "GET",
5       "href": "http://localhost:8080/engine-rest/process-instance/65f9118a-e738-11ed-9bef-0242ac110002",
6       "rel": "self"
7     }
8   ],
9   "id": "65f9118a-e738-11ed-9bef-0242ac110002",
10 }
```

An arrow points from the text "Process ID" to the `"id"` field in the response JSON.

Figure 29: Inicializácia camunda procesu

Následne si zavoláme endpoint `http://localhost:8080/engine-rest/task/?processInstanceId=65f9118a-e738-11ed-9bef-0242ac110002`, kde pošleme ID procesu a vráti sa nám zoznam taskov v danom procese. Ďalším volaním endpointu vieme skompletizovať task: `http://localhost:8080/engine-rest/task/aa51cb82-e73a-11ed-9bef-0242ac110002/complete`. V nástroji camunda cockpit môžeme sledovať v akej fáze je process.

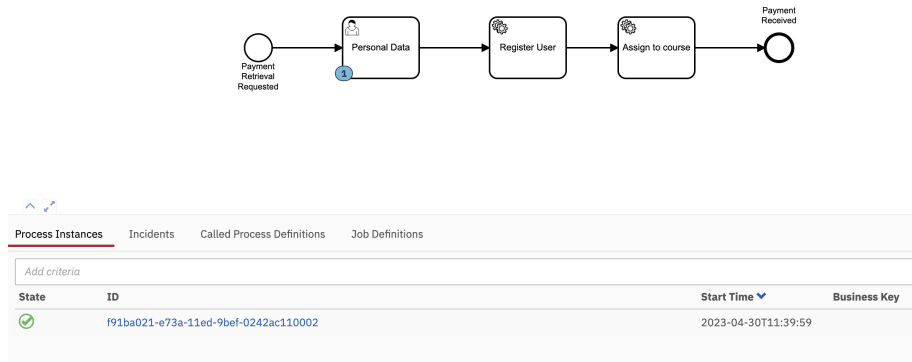


Figure 30: Camunda Cockpit

Kód na strane servera sme implementovali vo frameworku Express.js, ktorý je postavený nad Node.js. Pôvodnú implementáciu v PHP sme museli vymeniť za JavaScript z dôvodu nekompatibility PHP s Cammundou. V JavaScripte sme na napojenie s Cammundou použili knižnicu camunda-external-task-client-js. Vytvorili sme dve funkcie, ktoré sú subsribnute na extrerné cammudan tasky. Prvá funkcia počúva na task "register-user". V nej sa vytvorí v databáze nový používateľ a označí task ako "complete".

```
client.subscribe("register-user", async function ({ task, taskService }) {
  // Execute your logic for the external task here, using the task and taskService objects
  const firstName = task.variables.get("firstName");
  const lastName = task.variables.get("lastName");
  const email = task.variables.get("email");
  const password = task.variables.get("password");

  // UserController.test();

  const data = {
    id: 55,
    first_name: firstName,
    last_name: lastName,
    email: email,
    password: password,
    is_admin: false,
  };

  UserController.registerUser(data);

  // Complete the task
  await taskService.complete(task);
});
```

Figure 31: Externý task v node.js (register-user)

Druhá funkcia počúva na externý task "assign-to-course", kde sa používateľ priradí do kurzu.

```
client.subscribe("assign-to-course", async function ({ task, taskService }) {
  // Execute your logic for the external task here, using the task and taskService objects
  const courseId = task.variables.get("courseId");

  const data = {
    course_id: courseId,
    user_id: 55,
  };

  UserController.assignUserToCourse(data);

  // Complete the task
  await taskService.complete(task);
});
```

Figure 32: Externý task v node.js (assign-to-course)

4.2.3 Kafka - implementácia

Na implementáciu pomocou Kafky sme použili projekt implementovaný taktiež v Node.js a frameworku Express.

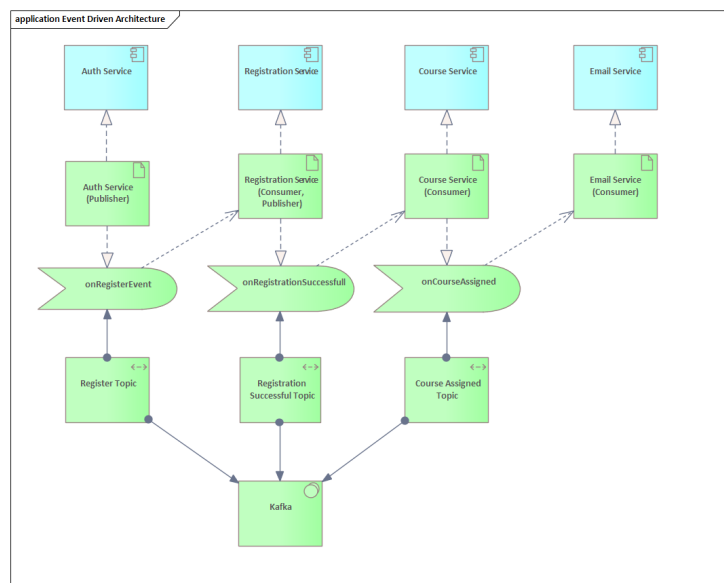


Figure 33: Diagram pre udalosti v Kafke

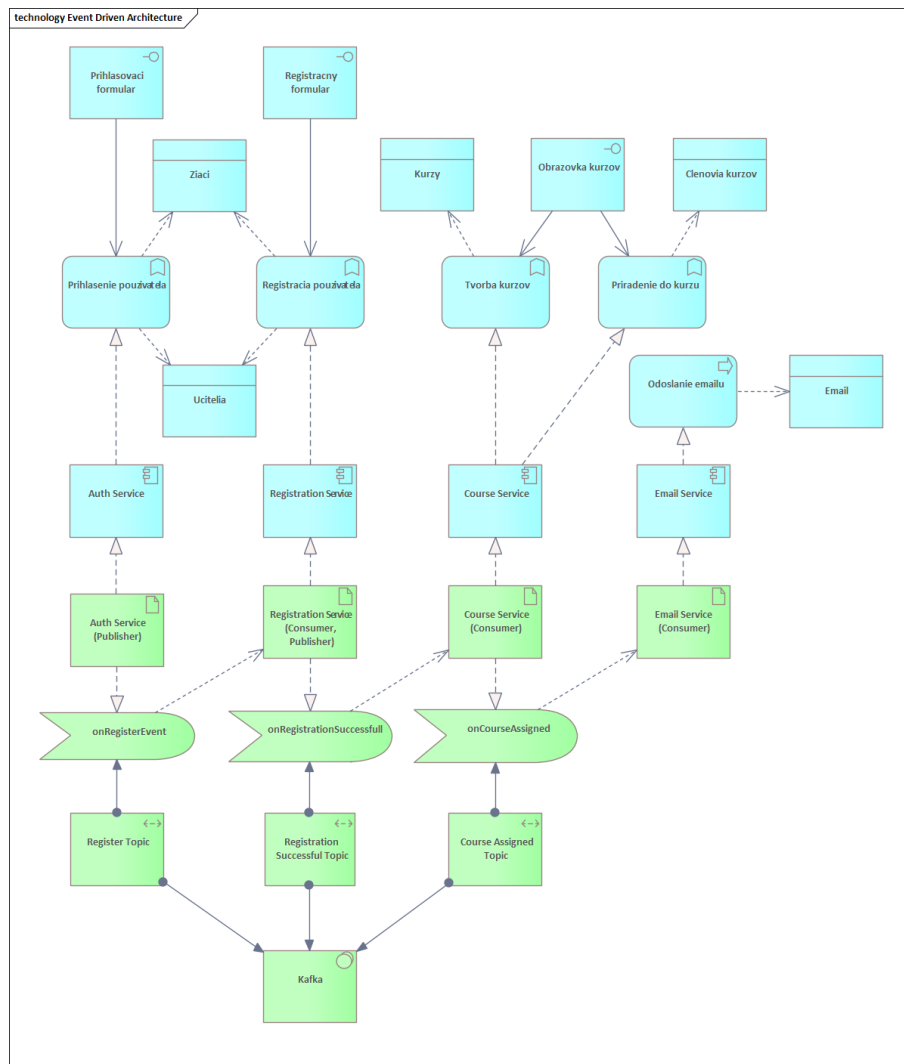


Figure 34: Rozšírený digram pre udalosti v Kafke

V diagrame máme zadané tri hlavné eventy - onRegisterEvent, onRegistrationSuccessfulEvent, onCourseAssignedEvent. V tomto diagrame máme zobrazený proces registrácie používateľa a pridelenie do kurzu. Každý event patrí jednému topicu. Na začiatku frontend zavolá auth service, ktorý vypublikuje onRegisterEvent, ktorý patrí do Register topicu. Registration service počúva na onRegisterEvent a vykoná sa registrácia používateľa. Následne vypublikuje onRegistrationSuccessfulEvent, na ktorý počúva Course service, ktorý priradí používateľa do kurzu. Po priradení vypublikuje onCourseAssignedEvent, na ktorý počúva Email service, ktorý odošle potvrdenie vo forme e-mailu.

Na začiatku potrebujeme spustiť Kafku v Dockeri pomocou príkazu: `docker exec -it kafka1 /bin/bash`. Na implementáciu pomocou Kafky sme použili Node.js, framework Express.js a databázu PostgreSQL. Využili sme knižnicu `kafkajs`. Na začiatku kódu sme inicializovali pripojenie na Kafku. Následne sme vytvorili jeden kafka producer a dva kafka consumery. Každý z týchto consumerov začne počúvať na konkrétny topic pomocou funkcie `subscribe`.

```
const kafka = new Kafka({
  clientId: "kafka-nodejs-starter",
  brokers: ["localhost:9092"],
});

const producer = kafka.producer();
const consumer = kafka.consumer({ groupId: "demoTopic-consumerGroup" });
const consumer2 = kafka.consumer({ groupId: "demoTopic-consumerGroup" });
await producer.connect();
await consumer.connect();
await consumer2.connect();

await consumer.subscribe({
  topic: "registrationSuccessfulTopic",
  fromBeginning: true,
});
await consumer2.subscribe({
  topic: "courseAssignedTopic",
  fromBeginning: true,
});
```

Figure 35: Inicializácia Kafka producerov a consumerov

Následne ak používateľ na frontende vyplní registračný formulár a pomocou tlačidla registrovať odošle formulár, tak na backende sa zavolá `RegisterUser` controller a funkcia `registreUser`. V tejto funkcii sa vytvorí v databáze nový používateľ a následne producer odošle správu "User created" na topic "registrationSuccessfulTopic".

```

const registerUser = async (request) => {
  const { id, first_name, last_name, password, email, is_admin } = request.body;

  pool.query(
    "INSERT INTO users (id, first_name, last_name, password, email, is_admin) VALUES ($1, $2, $3, $4, $5, $6)",
    [id, first_name, last_name, passwordHash, email, is_admin],
    (error) => {
      if (error) {
        throw error;
      }
      producer.send({
        topic: "registrationSuccessfulTopic",
        messages: [{ value: "User created" }],
      });
    }
  );
};

```

Figure 36: Registrácia používateľa a odoslanie správy do topicu

Prvý consumer, ktorý počúva na registrationSuccessfulTopic sa aktivuje a vypíše správu, ktorá bola odoslaná na topic. Následne zavolá ďalšiu funkciu na priradenie používateľa do kurzu.

```

await consumer.run({
  eachMessage: async ({ topic, partition, message }) => {
    console.log("Received1:" + message.value.toString());

    assignUserToCourse({
      user_id: 13,
      course_id: 1,
    });
  },
});

```

Figure 37: Vykonanie funkcie consumera

Zavolá sa funkcia na priradenie používateľa do kurzu, v ktorej sa vykonajú potrebné zmeny v databáze a po úspešnom zapísaní producer pošle správu do topicu s názvom courseAssignedTopic, na ktorý počúva nasledujúci consumer, ktorý vypíše potvrdzujúcu správu a bude volať ďalšie funkcie na odoslanie e-mailu.

```

const assignUserToCourse = async (request) => {
  const { user_id, course_id } = request;

  pool.query(
    "INSERT INTO courses_users (course_id, user_id) VALUES ($1, $2)",
    [course_id, user_id],
    (error) => {
      if (error) {
        throw error;
      }
      // Produce new message
      producer.send({
        topic: "courseAssignedTopic",
        messages: [{ value: "User assigned" }],
      });
    }
  );
};

```

Figure 38: Priradenie používateľa do kurzu

V terminále kde beží kafka si vieme zobrazíť všetky aktívne topicy. Pomocou príkazu: `kafka-topics --list --bootstrap-server localhost:9092` sa nám vypíšu aktívne topicy.

```

[appuser@kafka1 ~]$ kafka-topics --list --bootstrap-server localhost:9092
__consumer_offsets
courseAssignedTopic
registrationSuccessfulTopic

```

Figure 39: Zoznam aktívnych kafka topicov

5 Záver

V tejto práci sme opísali proces vývoj webovej aplikácie od návrhu až po implementáciu. Popísali sme jednotlivé diagramy navrhnuté v nástroji Enterprise Architect. Následne sme opísali implementáciu klientskej (frontend) časti aplikácie v jazyku Javascript s využitím Vue.js framweorku. Následne sme implementovali serverovú časť aplikácie troma rôznymi spôsobmi. Najprv sme implementovali backend s využitím mikroslužiev v jazyku PHP s využitím Laravel frameworku. Potom sme aplikáciu prerobili do frameworku Node.js s využitím Cammundy a nakoniec sme si vyskúšalai backend napojiť na Kafku. Takto sme si vyskúšali tri rôzne, populárne spôsoby implementácie veľkých serverových aplikácií.