

**SVEUČILIŠTE U SPLITU**  
**SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE**

Stručni diplomski studij Primijenjeno računarstvo

**ANAMARIJA PAPIĆ**

**D I P L O M S K I   R A D**

**RAZVOJ WEB APLIKACIJE ZA REPETICIJE**

Split, veljača 2026.

**SVEUČILIŠTE U SPLITU**  
**SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE**

Stručni diplomski studij Primijenjeno računarstvo

**Predmet:** Kriptovalute

**D I P L O M S K I R A D**

**Kandidat:** Anamarija Papić

**Naslov rada:** Razvoj web aplikacije za repeticije

**Mentor:** Nikola Grgić, viši predavač

Split, veljača 2026.

# Sadržaj

<b>Sažetak</b>	<b>1</b>
<b>1 Uvod</b>	<b>2</b>
<b>2 Tehnologije i teoretska podloga</b>	<b>4</b>
2.1 Prezentacijski i aplikacijski sloj . . . . .	4
2.2 Sloj korisničkog sučelja . . . . .	6
2.3 Podatkovni sloj . . . . .	8
2.4 Komunikacijski sloj . . . . .	9
2.4.1 WebRTC . . . . .	9
2.4.1.1 Topologije sustava WebRTC . . . . .	11
2.4.1.2 Faze uspostave WebRTC veze . . . . .	12
2.5 Ostali alati . . . . .	16
<b>3 Opis implementacije i prikaz praktičnog dijela rada</b>	<b>17</b>
3.1 Instalacija i početno postavljanje . . . . .	17
3.2 Struktura Next.js projekta . . . . .	17
3.2.1 Struktura početnog direktorija . . . . .	17
3.2.2 Struktura direktorija app/ . . . . .	20
3.3 Dijagram arhitekture aplikacije . . . . .	21
3.4 Rad s bazom podataka . . . . .	22
3.4.1 Postavljanje i postupci nad bazom podataka . . . . .	22
3.4.2 Dijagram relacijske baze podataka . . . . .	24
3.5 Repozitoriji u sloju pristupa podatcima . . . . .	25
3.6 Poslovna logika u servisnom sloju . . . . .	26
3.7 Usmjeravanje zahtjeva . . . . .	28
3.8 Implementacija autentikacije i autorizacije . . . . .	31
3.8.1 Administratorski dodatak, uloge i impersonacija korisnika . . . . .	34
3.8.2 <i>Proxy, middleware</i> i zaštita API ruta . . . . .	36
3.9 Validacija vremenske zone . . . . .	39
3.10 Integracija Google Maps API-ja . . . . .	39
3.11 Implementacija podrške za <i>online</i> repeticije . . . . .	40

3.11.1	Inicijalizacija sobe za sastanke . . . . .	41
3.11.2	Integracija s WebRTC-om i implementacija video poziva . . . . .	41
3.12	Slanje obavijesti putem e-pošte i Resend integracija . . . . .	42
3.13	Automatizirani <i>cron</i> zadatci . . . . .	43
3.14	Prezentacija funkcionalnosti aplikacije kroz korisničko sučelje . . . . .	44
<b>4</b>	<b>Zaključak</b>	<b>54</b>
	<b>Literatura</b>	<b>55</b>
	<b>Dodatci</b>	<b>58</b>

## Sažetak

Cilj ovog diplomskog rada je razviti intuitivnu web aplikaciju koja će služiti kao platforma za povezivanje studenata i tutora s ciljem olakšavanja organizacije i održavanja repeticija, pojednostavljajući proces pružanja i pohađanja repeticija uživo ili *online*. Aplikacija omogućuje studentima jednostavno pronalaženje i rezervaciju repeticija, dok tutori mogu učinkovito upravljati vlastitim rasporedom, ponudama i rezervacijama. Sudjelovanje u *online* repeticijama podržano je integracijom *peer-to-peer* tehnologije WebRTC za video pozive. *Full-stack* web aplikacija razvijena je u razvojnem okviru za React – Next.js-u, dok se za pohranu podataka koristi relacijska baza podataka PostgreSQL upravljana putem Prisma ORM-a i smještena na platformi Supabase. U ovom pisanom radu predstavljene su korištene tehnologije, arhitektura sustava, poslovna logika te implementacija pojedinih funkcionalnosti aplikacije.

**Ključne riječi:** *audio-video pozivi, Next.js, sustav za upravljanje rezervacijama, WebRTC*

## Summary

### Development of a Web Application for Tutoring

The aim of this thesis is to develop an intuitive web application that serves as a platform for connecting students and tutors, with the goal of facilitating the organization and conduct of tutoring sessions and simplifying the process of providing and attending lessons, both in-person and online. The application enables students to easily find and book tutoring sessions, while tutors can efficiently manage their schedules, offers, and reservations. Participation in online tutoring sessions is supported through integrated WebRTC peer-to-peer technology for video calls. The full-stack web application is developed using the Next.js framework for React, while data storage is handled by a relational PostgreSQL database through Prisma ORM and managed on the Supabase platform. This thesis presents the technologies used, the system architecture, business logic, and the implementation of individual application functionalities.

**Keywords:** *audio-video calls, booking management system, Next.js, WebRTC*

## 1. Uvod

U hrvatskom se obrazovanju posljednjih dvadesetak godina jasno potvrđuje snažan rast „sjene obrazovanja” – razgranat obrazovni biznis plaćenih privatnih repeticija i organiziranih priprema za državnu maturu i prijemne ispite, koji funkcionira paralelno s formalnim obrazovnim sustavom. Dodatne poduke postale su uobičajen dio školovanja i svakodnevica mnogih hrvatskih đaka svih uzrasta – od osnovne škole do fakulteta, a ne iznimka, unatoč tome što ovakav oblik obrazovanja otvara niz pitanja jednakosti pristupa znanju (navedene usluge dostupnije su učenicima iz socioekonomski povoljnijih obitelji te iz većih geografskih centara) te stvara pritisak na učenike i komercijalizira obrazovanje. Istraživanja Borisa Jokića i Zrinke Ristić Dedić pokazuju da je gotovo 40,0% učenika u svakoj ispitanoj generaciji (8. razred osnovne te 2. i završni razred srednje škole) koristilo privatne repeticije u školskoj godini 2020./2021., a među maturantima ih je 38,0%. Svaki drugi maturant gimnazije poхађa repeticije (i to 17,1% njih redovito), dok 39,2% učenika završnih razreda strukovnih i umjetničkih škola također koristi takvu pomoć, najčešće iz matematike i ključnih predmeta za upis na studij. Većina učenika i dalje preferira poхађanje repeticija uživo, međutim 20,6% maturanata i 15,9% osmaša poхађalo je privatne repeticije u šk. god. 2020./2021. u *online* obliku [1]. 56,0% maturanata poхађalo je pripremne tečajeve za državnu maturu, a njih 36,1% poхађalo ih je u nekom obliku kod privatnih tvrtki. 62,9% gimnazijalaca i 50,3% učenika strukovnih škola koji planiraju upisati studij poхађalo je pripremne tečajeve u šk. god. 2020./2021., pri čemu 48,6% gimnazijalaca i 26,4% *strukovnjaka* kod privatnih tvrtki [2].

Trenutno se ponuda privatnih repeticija u Hrvatskoj najčešće pronalazi putem usmenih preporuka, *online* oglasnika ili grupa na društvenim mrežama, kao i na zalijepljenim oglasima na javnim površinama, gdje svoje usluge podučavanja najčešće nude uspješniji studenti i (umirovljeni) nastavnici kako bi zaradili dodatni prihod. U domaćem se medijskom prostoru povremeno izvještava da su tutori stoga i česta meta Državnog inspektorata zbog rada „na crno” jer mnogi od njih nisu registrirani kao obrtnici, a s obzirom na popularnost ovakvog oblika dodatne zarade, nadzori su učestali. Razvijena aplikacija zasad ipak ne nalazi u ovu pravnu i poreznu problematiku, već samo ističe cijene usluga te plaćanje usluga ostaje između tutora i učenika izvan same aplikacije.

Praktični dio ovog rada, aplikacija *repeticije-hr*, nastaje upravo unutar prethodno pred-

stavljenog konteksta, kao pokušaj da se postojeće, često netransparentno i nejednako tržište privatnih repeticija učini pravednijim, preglednijim i dostupnijim većem broju učenika i studenata. Temeljna ideja aplikacije je digitalno povezati učenike i tuteure na jednom mjestu, uz jasan uvid u kvalifikacije, ponudu predmeta, cijene, termine, lokaciju (*uživo/online*) i povratne informacije drugih korisnika, čime se smanjuje ovisnost o „vezi“ i uskim krugovima preporuka. Time se barem djelomično adresiraju problemi koje ističu Jokić i Ristić Dedić – neravnomjerna dostupnost repeticija, netransparentnost ponude i snažna socijalna selektivnost – jer učenici iz različitih dijelova Hrvatske, neovisno o tome žive li u većim centrima ili manjim sredinama, dobivaju strukturiran digitalni prostor u kojem mogu lakše pronaći podršku koja im je potrebna.

U poglavljima koja slijede prvo će kratko biti predstavljene korištene tehnologije pri izradi aplikacije i objasnjena teoretska podloga, a zatim će se detaljno i pregledno opisati poslovna logika i način na koji je sama aplikacija implementirana.

## 2. Tehnologije i teoretska podloga

U ovom poglavlju detaljno su opisane tehnologije i alati korišteni u razvoju, kao i teoretska podloga koja je oblikovala arhitekturu i dizajn sustava. Polazi se od klasične troslojne arhitekture (prezentacijski, aplikacijski i podatkovni sloj), ali su za potrebe jasnijeg opisa korištenih tehnologija ti slojevi dodatno razrađeni u zajednički prezentacijski i aplikacijski sloj, sloj korisničkog sučelja, podatkovni sloj te komunikacijski sloj. Na kraju poglavlja kratko su opisani i ostali alati koji su korišteni u procesu razvoja.

### 2.1. Prezentacijski i aplikacijski sloj

U klasičnoj troslojnoj arhitekturi prezentacijski sloj zadužen je za prikaz podataka i interakciju s korisnikom, dok aplikacijski sloj implementira poslovnu logiku i orkestrira pristup podatkovnom sloju. U ovoj aplikaciji Next.js, temeljen na Reactu, istovremeno preuzima ulogu prezentacijskog sloja (renderiranje korisničkog sučelja u pregledniku) i dijela aplikacijskog sloja (renderiranje na strani poslužitelja, API rute, obrada zahtjeva), dok TypeScript, Node.js i npm čine temeljni tehnološki ekosustav u kojem se taj sloj razvija i izvršava.

**Next.js** je razvojni okvir (engl. *framework*) otvorenog kôda za React koji omogućuje objedinjeni (engl. *full-stack*) razvoj web aplikacija, što znači da obuhvaća razvoj i klijentskog (engl. *front-end*) i poslužiteljskog (engl. *back-end*) dijela aplikacije. Razvijen je od strane tvrtke Vercel, a izvorni kôd dostupan je na platformi GitHub [3] gdje je prva verzija objavljena 2016. godine.

Temeljen je na JavaScriptu i izvršava se u okruženju **Node.js**, pri čemu Node.js omogućuje pokretanje JavaScript kôda izvan preglednika, na poslužitelju, što je preduvjet za poslužiteljsko renderiranje i izgradnju aplikacije [4]. U praksi se Next.js projekti gotovo uvijek razvijaju uz **TypeScript**, nadskup JavaScripta razvijen od strane Microsofta, koji uvodi statičku tipizaciju i provjeru kôda u vrijeme prevođenja, čime se smanjuje broj pogrešaka u složenijim aplikacijama [5].

Upravljanje ovisnostima obavlja se putem **npm**-a (engl. *Node Package Manager*) ili alternativno drugim upraviteljima paketa kao što su pnpm/yarn/bun, koji omogućuju jednostavnu instalaciju, ažuriranje i skriptiranje razvojnih zadataka u okruženju Node.js. Paketi (biblioteke) se preuzimaju iz javnog repozitorija [npmjs.com](https://www.npmjs.com), a definiraju se u datoteci

`package.json` unutar projekta, a pri instalaciji se kreira i datoteka `package-lock.json` koja zaključava točne verzije ovisnosti radi konzistentnog okruženja. U direktoriju `node_modules` pohranjuju se sve instalirane ovisnosti projekta [6].

**React** (`React.js`) je slobodan softver otvorenog kôda tzv. FOSS (engl. *Free and Open Source Software*). Riječ je o JavaScript biblioteci za izgradnju korisničkih sučelja koju je razvio Facebook (sada Meta). Programski kôd javno je objavljen 2013. godine, a od tada je prigrljen od strane zajednice i pozicionirao se kao jedna od najpopularnijih biblioteka za razvoj web aplikacija [7].

React se često opisuje kroz tri osnovne značajke: *declarative*, *component-based* i sloganom „*learn once, write anywhere*”. Deklarativan pristup znači da razvojni programer opisuje kakvo korisničko sučelje želi u određenom stanju aplikacije, a React se brine za učinkovit prikaz i osvježavanje DOM-a (engl. *Document Object Model*), što olakšava razumijevanje i održavanje kôda. Komponentni pristup podrazumijeva da se sučelje sastoji od malih, ponovno iskoristivih komponenti koje enkapsuliraju logiku i prikaz pa se iste komponente mogu koristiti na više mjesta i u različitim projektima. Načelo „*learn once, write anywhere*” odnosi se na činjenicu da se ista znanja i koncepti mogu primijeniti u web aplikacijama, ali i u izradi mobilnih (React Native) i *desktop* aplikacija, bez ponovnog učenja potpuno novog programskog modela [8].

React se temelji na konceptu komponenti, pri čemu se u povijesnom razvoju razlikuju klasne (engl. *class*) komponente i funkcjske (engl. *function*) komponente, koje su trenutno preporučeni način pisanja komponenti. Uvođenjem kuka (engl. *hooks*) preko Hook API-ja (engl. *Application Programming Interface*) funkcjske komponente dobine su mogućnost upravljanja stanjem i *side* efektima bez potrebe za klasama, čime se dodatno pojednostavilo strukturiranje logike i ponovna iskoristivost kôda. React koristi virtualni DOM – laganu, u memoriji pohranjenu reprezentaciju strukture korisničkog sučelja kako bi učinkovito uspoređivao prethodno i novo stanje te primjenjivao samo nužne promjene u stvarnom DOM-u, što značajno poboljšava performanse složenijih aplikacija. JSX (engl. *JavaScript XML*) je sintaksno proširenje za JavaScript koje omogućuje zapis komponenti u obliku HTML-u slične sintakse, pri čemu se JSX u pozadini prevodi u pozive funkcije `React.createElement`, a razvojnom programeru olakšava čitanje i strukturiranje kôda [9]. U kombinaciji s TypeScriptom koristi se proširenje TSX (engl. *TypeScript JSX*). U novijim verzijama React uvodi

i Server Components – komponente koje se izvršavaju isključivo na poslužitelju, generiraju HTML bez slanja JavaScript kôda na klijent i tako smanjuju veličinu paketa te povećavaju sigurnost i performanse, osobito u kombinaciji s razvojnim okvirima poput Next.js-a.

Next.js nadograđuje React dodavanjem brojnih značajki i optimizacija koje su olakšale izgradnju skalabilnih i visokoučinkovitih web aplikacija. Uvodi višestruke načine renderiranja stranica i napredne optimizacije koje su posebno važne za performanse i SEO. Osim klasičnog renderiranja na strani klijenta – CSR (engl. *Client-Side Rendering*), Next.js podržava renderiranje na strani poslužitelja – SSR (engl. *Server-Side Rendering*), statičko generiranje stranica – SSG (engl. *Static Site Generation*) te inkrementalno statičko obnavljanje – ISR (engl. *Incremental Static Regeneration*), pri čemu se većina sadržaja isporučuje kao statički HTML, a pojedine stranice se povremeno regeneriraju kada se podatci promijene. Next.js također automatski dijeli kôd u manje dijelove (engl. *code splitting*) i koristi usmjeravanje (engl. *routing*) bazirano na strukturi direktorija, poznato kao *file-based routing*. To znači da se samo nužni dijelovi kôda učitavaju za svaku stranicu, čime se smanjuje vrijeme učitavanja i poboljšava korisničko iskustvo. U novijim verzijama uveden je App Router s podrškom za React Server Components, strujanje (engl. *streaming*) sadržaja i Server Actions, što omogućuje da se dio logike obrade podataka izvršava na poslužitelju uz manju količinu potrebnog JavaScript kôda na klijentu, što pruža brži prikaz sadržaja korisniku [10].

Osim spomenutih značajki, Next.js sadrži integrirane alate za optimizaciju slika automatski generirajući različite veličine, koristi lijeno učitavanje (engl. *lazy loading*) i moderne formate slika, čime se poboljšavaju Core Web Vitals metrike i ukupne performanse web aplikacija. U kombinaciji s podrškom za TypeScript „*out-of-the-box*”, integracijom s Vercelom za jednostavnu isporuku te bogatim ekosustavom dodataka, Next.js se pozicionira kao cjelovit razvojni okvir za izradu skalabilnih, produksijski spremnih aplikacija [10].

U trenutku izrade rada najnovija stabilna verzija Next.js-a je 16, dok je React dosegao verziju 19 te su iste korištene u izradi praktičnog rada, uz TypeScript verzije 5.x. Korištena verzija Node.js-a je 24.x LTS (engl. *Long Term Support*) uz npm verzije 11.x.

## 2.2. Sloj korisničkog sučelja

Sloj korisničkog sučelja (engl. UI – *User Interface*) aplikacije temelji se na kombinaciji biblioteka shadcn/ui, Radix UI i Tailwind CSS, koje su usko povezane i nadograđuju se

jedna na drugu.

**shadcn/ui** je zbirka unaprijed pripremljenih React komponenti (npr. avatar, botuni, bedževi, kartice za prikaz sadržaja, dijaloški okviri, padajući izbornici, polja i grupe za unos, navigacijski izbornici, tablice itd.) koje su izgrađene na Radix UI „primitivima” i stilizirane pomoću Tailwind CSS-a. Umjesto da se isporučuje kao klasična biblioteka ovisnosti, shadcn/ui generira stvarne izvorišne datoteke komponenti koje se kopiraju u projekt, čime razvojni programer zadržava potpunu kontrolu nad kôdom, može ga prilagoditi specifičnim potrebama aplikacije i izbjegava dodatni teret zasebne UI biblioteke dok se aplikacija izvršava (engl. *runtime*). Ovo je osobito korisno u većim projektima, gdje se očekuju dugočrno održavanje, prilagodba dizajna i dosljedan vizualni identitet [11].

**Radix UI** predstavlja skup pristupačnih (engl. *accessible*) niskorazinskih komponenti tzv. „primitiva” poput dijaloških okvira, skočnih prozora, kartica (engl. *tabs*), skočnih izbornika i slično. Fokus Radix UI-ja je na ispravnom ponašanju i pristupačnosti: osigurava ispravne ARIA (engl. *Accessible Rich Internet Applications*) atribute, upravljanje fokusom i podršku za tipkovničku navigaciju u skladu s WAI-ARIA (engl. *Web Accessibility Initiative – ARIA*) smjernicama, dok izgled i stilizacija ostaju u potpunosti u domeni razvojnih timova [12].

**Tailwind CSS** je utilitaristički (engl. *utility-first*) razvojni okvir za CSS koji umjesto gotovih vizualnih komponenti nudi velik broj malih, jednoznačnih klasa za oblikovanje (npr. razmaci, boje, tipografija, raspored – fleksibilni i mrežni (engl. *grid*) rasporedi). Te se klase kombiniraju izravno u markup-u ili JSX/TSX kôdu, čime se smanjuje potreba za pisanjem zasebnih CSS datoteka i olakšava održavanje dosljednog dizajna u komponentnom pristupu Reacta. Tailwind se konfigurira putem konfiguracijske datoteke koja definira dizajnerske tokene (boje, tipografiju, radijuse, razmake), a u produksijskom okruženju alat automatski uklanja neiskorištene klase (engl. *tree shaking*), što rezultira malim konačnim CSS paketom i boljim performansama [13].

U razvijenoj aplikaciji kombinacija predstavljenih UI biblioteka omogućuje izradu modernog, responzivnog i pristupačnog korisničkog sučelja uz dobru ravnotežu između brzine razvoja, kontrole nad dizajnom i tehničke kvalitete.

## 2.3. Podatkovni sloj

Za pohranu podataka u aplikaciji koristi se relacijska baza podataka PostgreSQL, uz objektno-relacijsko mapiranje pomoću Prisma ORM-a (engl. *Object-Relational Mapper*) i upravljeni *hosting* preko platforme Supabase.

**PostgreSQL** (Postgres) snažan je sustav za upravljanje relacijskim bazama podataka (engl. *RDBMS – Relational Database Management System*) otvorenog kôda s dugom poviješću razvoja, visokom razinom usklađenosti sa SQL (engl. *Structured Query Language*) standardom te podrškom za napredne tipove podataka i transakcijske operacije. Kao objektno-relacijski sustav, PostgreSQL osim klasičnih relacijskih tablica podržava i pohranu polustrukturiranih podataka (npr. JSON/JSONB tipovi, polja, geografski tipovi), što ga čini pogodnim za moderne web aplikacije koje kombiniraju strogo strukturirane podatke (npr. korisnici, termini rezervacija, recenzije) s fleksibilnjim strukturama [14].

**Prisma ORM** je biblioteka za objektno-relacijsko mapiranje za Node.js i TypeScript koja omogućuje tipno siguran (engl. *type-safe*) pristup bazi podataka. U središtu Prisma pristupa nalazi se datoteka `schema.prisma` u kojoj se deklarativno definira shema podataka putem Prisma Schema Language (PSL) – definiraju se modeli, relacije i mapiranja na tablice u bazi, a Prisma na temelju te sheme automatski generira klijenta za pristup podatcima, zajedno s odgovarajućim TypeScript tipovima. Time se postiže da je sve upite prema bazi moguće statički provjeriti u vrijeme prevođenja, uz bogatu podršku za automatisko dovršavanje i rano otkrivanje neispravnih upita prije pokretanja aplikacije, što smanjuje broj pogrešaka i ubrzava razvoj. Uz Prisma Client za rad s podatcima, Prisma nudi i alate za migracije (Prisma Migrate) te vizualni pregled podataka (Prisma Studio) koji olakšavaju upravljanje razvojnim i produkcijskim okruženjima [15].

**Supabase** je platforma otvorenog kôda tipa „*backend kao usluga*” (engl. *Backend-as-a-Service*) izgrađena na PostgreSQL-u, koja nudi upravljanu bazu podataka, autentikaciju, pohranu datoteka, *real-time* funkcionalnosti i poslužiteljske funkcije. U praksi to znači da se PostgreSQL baza podataka može vrlo brzo podići u oblaku, uz automatski izložene API-je, ugrađenu podršku za sigurnost na razini retka (engl. *Row-Level Security*) i mogućnost „osluškivanja” promjena u tablicama u stvarnom vremenu putem WebSocketa. Supabase pruža i CLI alat te mogućnost lokalnog razvoja putem Docker okruženja, što olakšava repliciranje produkcijske baze u lokalnom okruženju i testiranje sheme podataka [16].

U ovoj aplikaciji PostgreSQL služi kao temeljna baza podataka, Prisma ORM kao tipno siguran sloj pristupa podatcima u okruženju Next.js/TypeScript, dok Supabase pruža *hosting* baze i *real-time* funkcionalnosti, čime se dobiva skalabilan i moderni podatkovni sloj prilagođen potrebama web aplikacije za repeticije. Korištena su sljedeća izdanja tehnologija: PostgreSQL 17.x, Prisma ORM 7.x i Supabase 2.x.

## 2.4. Komunikacijski sloj

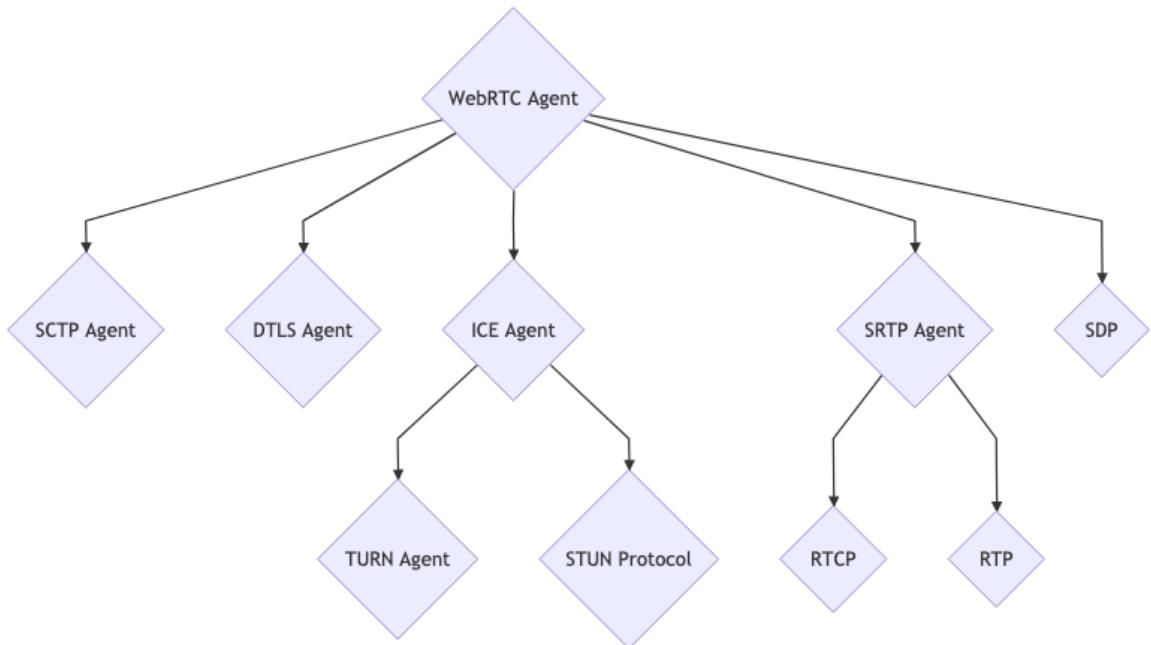
Komunikacijski sloj u ovoj aplikaciji zadužen je za uspostavljanje i održavanje komunikacije u stvarnom vremenu između tutora i učenika tijekom *online* repeticija. U tu svrhu koristi se **WebRTC** (engl. *Web Real-Time Communication*).

WebRTC preuzima brigu o dohvaćanju audio i video tokova, uspostavi šifriranog kanala te prilagodbi kvalitete prijenosa uvjetima mreže, dok se logika aplikacije fokusira na upravljanje sobama, sudionicima i korisničkim sučeljem. U nastavku su detaljnije opisane ključne značajke i funkcionalnosti tehnologije WebRTC.

### 2.4.1. WebRTC

Tehnologija WebRTC je skup standardiziranih tehnologija otvorenog kôda (FOSS) koja omogućuje izravnu *peer-to-peer* komunikaciju u stvarnom vremenu putem interneta, i to kroz moderne web preglednike ili nativne klijente u slučaju mobilnih aplikacija. Krajnji korisnici mogu izmjenjivati audio, video i podatke bez potrebe za dodatcima ili posebnim softverom. WebRTC se sastoji od WebRTC protokola i WebRTC API-ja specificiranog za JavaScript, dok je za nativne klijente dostupna biblioteka koja pruža iste funkcionalnosti [17]. Izvorni kôd dostupan je u repozitoriju na poveznici [webrtc.googlesource.com/src](https://webrtc.googlesource.com/src).

WebRTC protokol u suštini je sastavljen od niza standardiziranih protokola i stoga je često opisivan kao kolekcija protokola jer orkestrira različite protokole i mehanizme za uspostavu, održavanje i osiguravanje komunikacije u stvarnom vremenu, što je prikazano na slici 1 [18].



**Slika 1:** WebRTC kao kolekcija protokola [19]

Nastao je kao pokušaj da se tehnologije za govor i video u stvarnom vremenu, koje su ranije bile zatvorene i vlasničke, upgrade izravno u web preglednike kao otvoren standard. Google je 2010. kupio tvrtku Global IP Solutions (GIPS) te otvoreno licencirao kôd njihovih RTC komponenti, što je bio ključni korak u razvoju WebRTC-a. Nakon toga, Google je 2011. pokrenuo projekt WebRTC, a standardizacija je trajala nekoliko godina i iziskivala velike napore i suradnju između dvaju standardizacijskih tijela: W3C-a (engl. *World Wide Web Consortium*) i IETF-a (engl. *Internet Engineering Task Force*) te velikog broja tvrtki i pojedinaca iz industrije. Uz Google, u razvoj specifikacija i implementaciju uključile su se i tvrtke poput Mozille, Microsofta, Applea, Intel Corporations, Cisca i Ericssona, a podrška je postupno uvedena u svim modernim web preglednicima, čime je WebRTC postao standardni mehanizam za *in-browser* audio-video komunikaciju [20]. Danas se primjenjuje u širokom spektru aplikacija, od video konferencijskih sistema, do *online* igara, udaljenog upravljanja, IoT (engl. *Internet of Things*) komunikacije i drugih scenarija koji zahtijevaju nisku latenciju i visoku kvalitetu prijenosa.

#### 2.4.1.1. Topologije sustava WebRTC

WebRTC topologije mogu se podijeliti na *peer-to-peer* (P2P) i *client/server* pristup [18].

U najjednostavnijem slučaju dvije krajne točke uspostavljaju izravnu vezu „1 na 1” te međusobno razmjenjuju medijske tokove i podatke. Za grupne scenarije moguće je koristiti više različitih topologija, ovisno o broju sudionika, dostupnoj propusnosti i zahtjevima aplikacije.

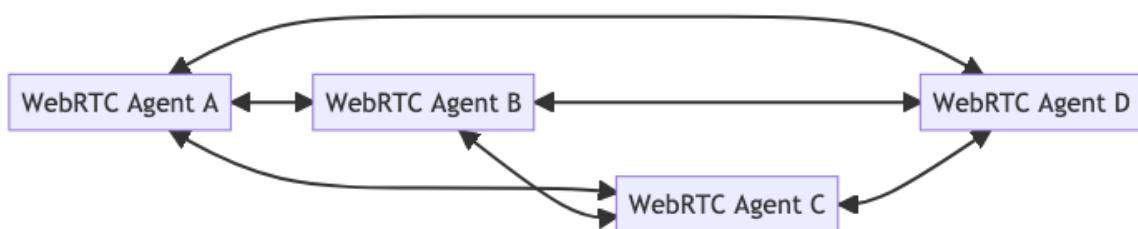
Topologija *full mesh* podrazumijeva da je svaki sudionik povezan izravno sa svakim drugim sudionikom. Svaki klijent tada mora zasebno kodirati i slati svoj video tok prema svakom udaljenom sudioniku, što povećava opterećenje mreže i ograničava praktičan broj sudionika, ali uz relativno jednostavnu implementaciju i bez dodatne poslužiteljske infrastrukture. Alternativno, *hybrid mesh* smanjuje broj izravnih veza tako da se podatci prosljeđuju preko pojedinih sudionika.

Topologije *client/server* tipa koriste specijalizirane poslužitelje: SFU (engl. *Selective Forwarding Unit*) koji selektivno prosljeđuje pristigle tokove svim klijentima te MCU (engl. *Multipoint Conferencing Unit*) koji višestruke ulazne tokove spaja u jedan kompozitni izlazni tok.

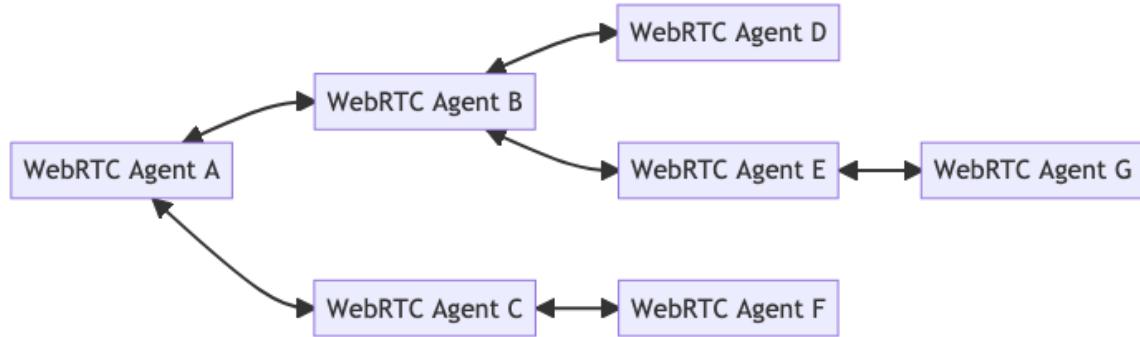
Na slikama 2 – 6 prikazane su različite WebRTC topologije.



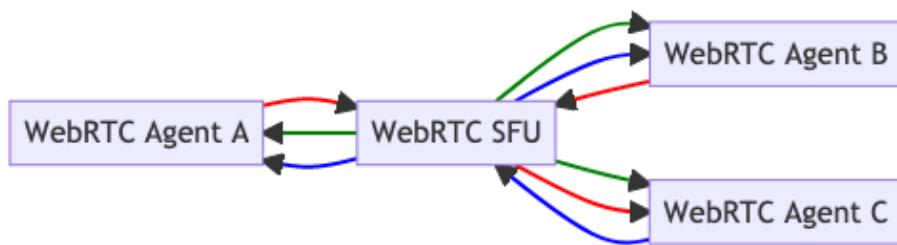
**Slika 2:** Topologija „1 na 1” [21]



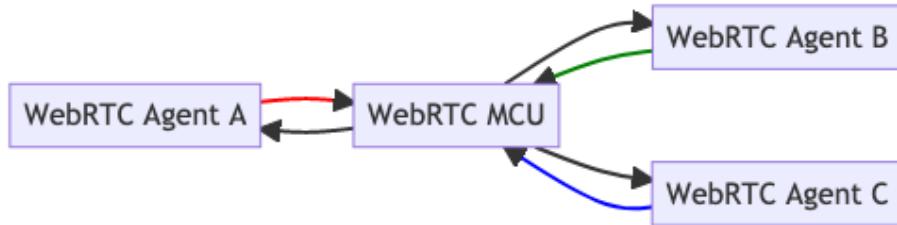
**Slika 3:** Topologija *full mesh* [22]



**Slika 4:** Topologija *hybrid mesh* [23]



**Slika 5:** Topologija SFU [24]



**Slika 6:** Topologija MCU [25]

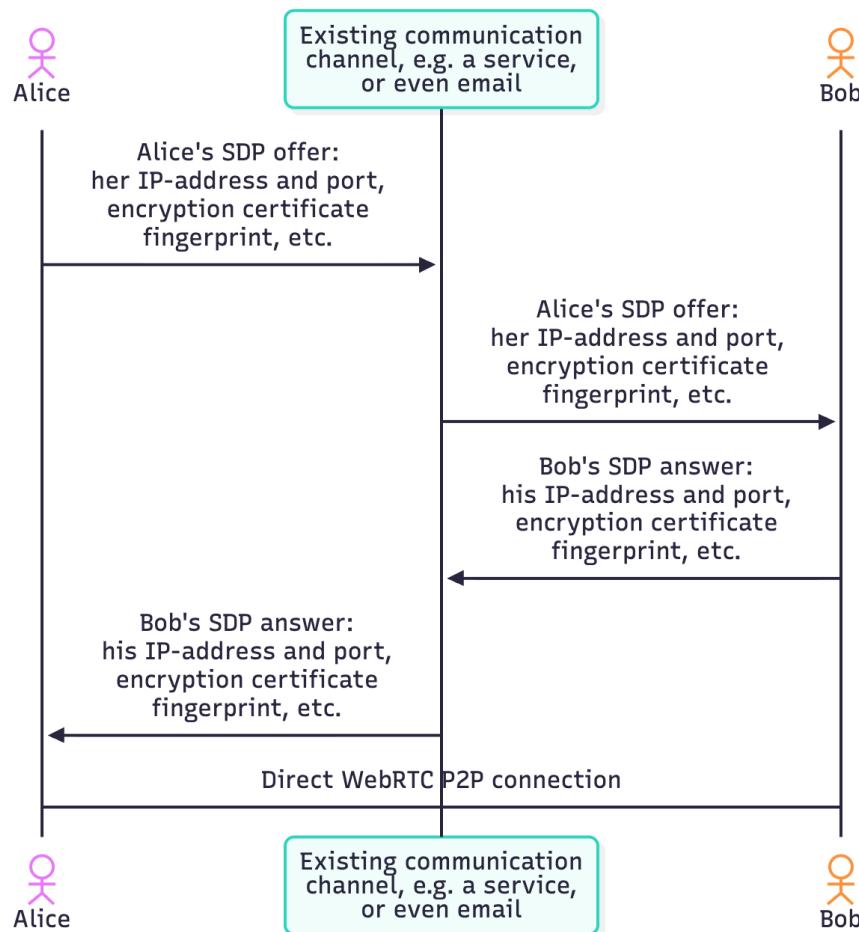
U ovoj aplikaciji koristi se *peer-to-peer* topologija *full mesh* što znači da svaki klijent u sobi uspostavlja po jednu vezu RTCPeerConnection prema svakom drugom sudioniku.

#### 2.4.1.2. Faze uspostave WebRTC veze

Uspostava WebRTC veze može se konceptualno rastaviti na četiri uzastopna koraka: signalizaciju, uspostavu povezivanja, osiguravanje veze i samu komunikaciju, pri čemu svaki sljedeći korak započinje tek kada je prethodni u potpunosti uspješno dovršen.

Prva faza, **signalizacija**, obuhvaća sve korake kojima se dva (ili više) sudionika međusobno „pronađu” i dogovore kako će komunicirati, prije nego što krene stvarni prijenos audio-video podataka. Signalizacija nije standardizirana u samom WebRTC-u, već se odvija

„out-of-band” i smatra dijelom aplikacijske logike – razvojni tim slobodno bira protokole i formate poruka (npr. REST, WebSocket i sl.). Najprije sudionici uspostave vezu sa zajedničkim signalizacijskim kanalom i na neki način identificiraju sobu ili razgovor u kojem sudjeluju. Zatim preko tog kanala razmjenjuju SDP (engl. *Session Description Protocol*) poruke s informacijama o tome koje medijske tokove žele slati te koje *kodeke* i konfiguracije podržavaju, kao i pripadajuće signalizacijske poruke (ponuda/odgovor, kandidati i sl.) potrebne za uspostavu veze [18]. Signalizacija je ilustrirana na slici 7.



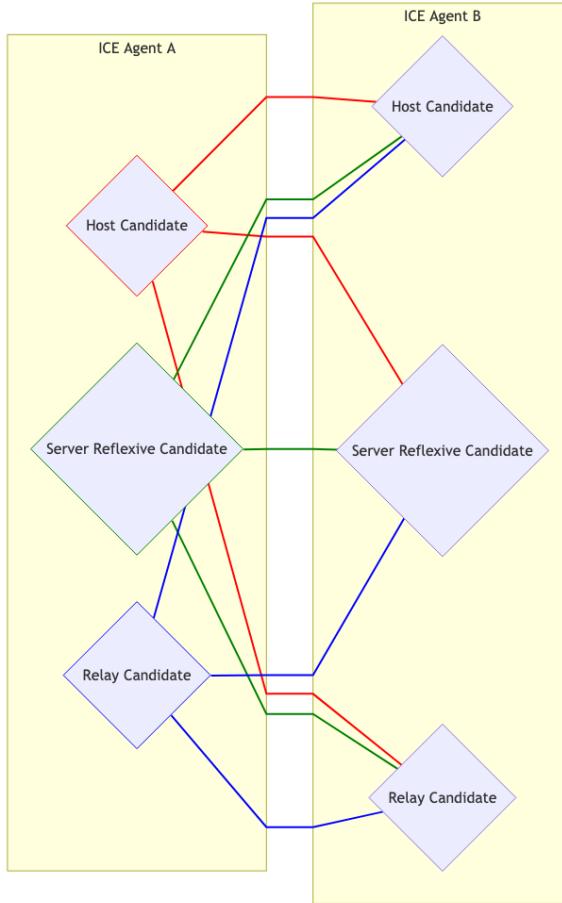
**Slika 7:** Signalizacija [26]

Budući da WebRTC ne specificira način signalizacije, ova aplikacija koristi već spomenute *real-time* funkcionalnosti platforme Supabase kao signalizacijski kanal. To znači da se poruke potrebne za uspostavu veze razmjenjuju preko Supabase *real-time* kanala, dok se sami medijski tokovi (audio i video) nakon dogovora o parametrima uspostavljaju izravno između preglednika sudionika. Na taj se način podatkovni sloj (Supabase/PostgreSQL) i

komunikacijski sloj (WebRTC) integriraju u cjelinu – Supabase osigurava pouzdanu i skalabilnu razmjenu signalnih poruka, a WebRTC preuzima prijenos multimedije u stvarnom vremenu.

Nakon što je signalizacijom razmijenjena osnovna konfiguracija sesije, slijedi druga faza – **povezivanje**, u kojoj WebRTC pokušava uspostaviti izvedivu mrežnu putanju između sudionika. U stvarnim uvjetima većina klijenata ne posjeduje javnu IP adresu, već se nalazi iza usmјernika (engl. *router*), NAT (engl. *Network Address Translation*) uređaja i vatrozida, što onemogućuje izravno adresiranje između dvaju računala. Kako bi unatoč tome ostvario *peer-to-peer* komunikaciju, WebRTC koristi kombinaciju okvira ICE (engl. *Interactive Connectivity Establishment*) te STUN (engl. *Session Traversal Utilities for NAT*) i TURN (engl. *Traversal Using Relays around NAT*) infrastrukture za tzv. NAT *traversal*, odnosno pronalaženje puteva kroz mrežnu konfiguraciju [18].

ICE na svakoj strani prikuplja različite mrežne „kandidate” — tzv. ICE kandidate (engl. *ICE candidates*) — što uključuje lokalne adrese, javne adrese dobivene putem STUN poslužitelja te, po potrebi, relejske (engl. *relay*) adrese dobivene preko TURN poslužitelja. Ti se kandidati zatim razmjenjuju signalizacijskim kanalom i sustavno testiraju kako bi se pronašao par kandidata preko kojih je moguće uspostaviti dvosmjernu komunikaciju uz što manju latenciju. Testiranje parova kandidata (engl. *connectivity checks*) prikazano je na slici 8. Na slici se nalaze dva ICE agenta, svaki s po tri različita kandidata (što podrazumijeva ukupno devet parova kandidata). U idealnom slučaju odabire se izravna P2P putanja (npr. putem javne adrese otkrivene STUN-om), a ako to nije moguće zbog ograničenja NAT-a ili vatrozida, komunikacija se preusmjerava preko TURN poslužitelja koji djeluje kao reley između sudionika. Na taj način faza povezivanja dinamički prilagođava mrežnu putanju kako bi osigurala da se medijski tokovi mogu prenositi i u složenim mrežnim okruženjima, pri čemu se uvijek preferiraju najmanje „skupe” putanje [18].



**Slika 8:** ICE kandidati i testiranje parova [27]

U trećoj fazi, **osiguravanju veze**, WebRTC osigurava da se svi medijski i podatkovni tokovi prenose sigurno. U tu svrhu koristi kombinaciju dvaju protokola: DTLS (engl. *Data-gram Transport Layer Security*) i SRTP (engl. *Secure Real-time Transport Protocol*). DTLS je UDP inačica TLS-a koji se koristi u HTTPS-u, zadužena za sigurno pregovaranje kriptografskih parametara i razmjenu ključeva preko nepouzdanog UDP transporta, pri čemu štiti od prisluskivanja, krivotvorena poruka i njihove izmjene u prijenosu [18].

SRTP nadograđuje RTP tako da dodaje enkripciju sadržaja medijskih paketa, provjeru integriteta i zaštitu od ponovnog reproduciranja paketa (engl. *replay attacks*), ali pritom ostaje dovoljno lagan da zadovolji zahtjeve komunikacije u stvarnom vremenu. U WebRTC-u se ključevi koji se koriste u SRTP-u ne šalju kao *plain-text*, već se izvode iz DTLS sesije u postupku poznatom kao DTLS-SRTP – najprije se između sudionika uspostavlja siguran DTLS kanal, zatim se iz njega generira potrebnii ključni materijal za SRTP, a nakon toga se audio i video tokovi šifrirano prenose upravo preko SRTP-a. Time WebRTC zahtijeva da sav medijski promet bude enkriptiran i autenticiran, bez mogućnosti isključenja enkripcije

u produkcijskim implementacijama, čime se krajnjim korisnicima osigurava visoka razina povjerljivosti i zaštite komunikacije [18].

Konačno, završna faza odnosi se na **stvarnu komunikaciju**. WebRTC koristi dva protokola za stvarnu razmjenu podataka između sudionika: RTP i SCTP. RTP je zadužen za prijenos audio i video tokova u stvarnom vremenu: paketi s medijskim okvirima nose vremenske oznake i redne brojeve, što omogućuje sinkronizaciju, detekciju gubitka paketa i prilagodbu reprodukcije mrežnim uvjetima. U WebRTC-u se RTP gotovo uvijek koristi u kombinaciji sa SRTP-om, koji dodaje enkripciju i zaštitu integriteta, pa se stvarni medijski promet prenosi putem SRTP paketa preko ranije uspostavljenog sigurnog kanala. Za proizvoljne podatkovne tokove WebRTC koristi SCTP (engl. *Stream Control Transmission Protocol*) inkapsuliran unutar DTLS-a, što omogućuje pouzdanu, redoslijednu komunikaciju između sudionika, uz logičko dijeljenje prometa u više neovisnih strujanja (engl. *streams*) [18].

## 2.5. Ostali alati

Osim glavnih tehnologija koje su temelj aplikacije, u procesu razvoja korišteni su i brojni drugi alati koji su olakšali razvoj, verzioniranje, isporuku i održavanje kôda.

Kao uređivač kôda korišten je **Visual Studio Code**, uz popularne ekstenzije za već prethodno spomenute tehnologije. **Git** je korišten za verzioniranje kôda, uz **GitHub** kao odabranu platformu za objavu repozitorija. **Vercel** je odabran kao platforma za *hosting* i *deployment* Next.js aplikacije, zbog svoje duboke integracije s Next.js-om, automatskog skaliranja, globalne mreže isporuke sadržaja (CDN) i jednostavnog procesa isporuke putem integracije s GitHub-om. Platforma **Resend** korištena je za slanje obavijesti korisnicima e-poštom, zahvaljujući svojoj jednostavnosti, pouzdanosti i modernom API-ju koji se lako integrira s okruženjem Node.js.

### 3. Opis implementacije i prikaz praktičnog dijela rada

#### 3.1. Instalacija i početno postavljanje

Instalacija nove Next.js aplikacije jednostavna je i započinje izvršavanjem naredbe `npx create-next-app@latest` u terminalu uz željeni naziv projekta kao argument. Nakon pokretanja slijede interaktivne upute u terminalu, dok se pri korištenju zastavice `--yes` projekt automatski inicijalizira s predefiniranim postavkama (omogućen TypeScript, Tailwind CSS, ESLint, App Router, Turbopack te alias za uvoz `@/*`) [10]. Nakon uspješne instalacije, aplikacija je spremna za pokretanje i razvoj.

Naredba `next dev` pokreće razvojni poslužitelj i koristi Turbopack kao zadani *bundler*, što omogućuje vrlo brze inkrementalne *buildove* i osvježavanje promjena tijekom razvoja. `next build` priprema aplikaciju za produkciju optimiziranjem kôda, generiranjem statičkih stranica i pakiranjem resursa, dok `next start` pokreće produkcijski poslužitelj nad prethodno izgrađenom verzijom aplikacije. Skripta `eslint` služi za pokretanje alata ESLint, koji automatski provjerava stil i potencijalne pogreške u JavaScript/TypeScript kôdu.

Dobra je praksa i odmah inicijalizirati Git repozitorij u projektu, što se može učiniti na redbom `git init`. To omogućuje praćenje promjena i upravljanje verzijama kôda tijekom razvoja.

Također, preporučeno je postaviti i formatiranje kôda, što se može postići instalacijom i konfiguracijom alata poput Prettiera kako bi se osigurao dosljedan stil kôda. EditorConfig datoteka također može pomoći u održavanju dosljednosti postavki između različitih uređivača kôda i razvojnih okruženja.

#### 3.2. Struktura Next.js projekta

##### 3.2.1. Struktura početnog direktorija

Osnovna struktura *root* direktorija implementirane Next.js aplikacije jest sljedeća:

```
repeticije-hr/
└── .next/
└── app/
└── calendar/
└── components/
```

```
|- data/
|- lib/
|- middleware/
|- node_modules/
|- prisma/
|- public/
|- scripts/
|- services/
|- supabase/
|- .editorconfig
|- .env
|- .env.example
|- .gitignore
|- .prettierrc.json
|- components.json
|- eslint.config.mjs
|- next-env.d.ts
|- next.config.ts
|- package-lock.json
|- package.json
|- postcss.config.mjs
|- prisma.config.ts
|- proxy.ts
|- README.md
|- tsconfig.json
|- vercel.json
```

- `.next/` – Automatski generiran direktorij koji sadrži rezultat procesa izgradnje (kompljirane stranice, *bundleove* i *cache*) te se ne verzionira u sustavu za kontrolu verzija.
- `app/` – Sadrži rute, *layoute* i API rute temeljene na Next.js App Routeru, odnosno predstavlja glavni kôd korisničkog sučelja i poslužiteljske logike unutar okvira Next.js.
- `calendar/` – Izdvojeni paket s komponentama, kontekstima i pomoćnim funkcijama za rad s kalendarom (preuzet s [github.com/lramos33/big-calendar](https://github.com/lramos33/big-calendar)) i prilagođen za potrebe aplikacije.
- `components/` – Sadrži višekratno iskoristive React komponente korisničkog sučelja, od osnovnih UI elemenata do složenijih komponenti.
- `data/` – Implementira sloj pristupa podatcima (engl. *repository layer*) nad bazom podataka, s modulima za korisnike, ponude repeticija, rezervacije itd.
- `lib/` – Zajedničke pomoćne biblioteke i konfiguracije, uključujući autentikaciju, Prisma klijent, Supabase klijent, WebRTC signalizaciju i opće *utility* funkcije.

- `middleware/` – Sadrži middleware kôd za zaštitu ruta; sadrži implementirane funkcije za provjeru autentikacije i autorizacije nad dolaznim zahtjevima.
- `node_modules/` – Automatski generiran direktorij s instaliranim npm paketima, tj. *runtime* i *build* ovisnostima projekta.
- `prisma/` – Sadrži datoteku `schema.prisma` s modelom baze podataka, SQL migracije te skriptu `seed.ts` za inicijalno popunjavanje baze.
- `public/` – Statički resursi (slike, ikone, SVG-ovi i sl.) koji su javno dostupni bez dodatne obrade na poslužiteljskoj strani.
- `scripts/` – Pomoćne skripte za održavanje i pozadinske zadatke (npr. *cron* skripta za automatsko ažuriranje statusa rezervacija).
- `services/` – Implementacija poslovne logike aplikacije, koja koristi repozitorije za pristup podatcima i implementira funkcionalnosti specifične za domenu.
- `supabase/` – Konfiguracijske datoteke i dokumentacija za lokalno okruženje Supabase i migracije baze.
- `.editorconfig` – Datoteka s pravilima formatiranja kôda za različite uređivače kôda.
- `.env, .env.example` – Datoteke s konfiguracijskim varijablama okruženja (stvarna konfiguracija i ogledni primjer).
- `.gitignore` – Definira koje datoteke i direktoriji se isključuju iz verzioniranja.
- `.prettierrc.json` – Konfiguracija za Prettier formatiranje kôda.
- `components.json` – Konfiguracijska datoteka za sustav komponenti.
- `eslint.config.mjs` – Konfiguracija ESLint-a za statičku analizu kôda.
- `next-env.d.ts` – TypeScript deklaracije specifične za Next.js, automatski generirane.
- `next.config.ts` – Glavna konfiguracijska datoteka Next.js aplikacije (*build* postavke, usmjeravanje, integracije).
- `package-lock.json` – Zaključane verzije npm paketa za determinističke instalacije.

cije.

- `package.json` – Metapodatci o projektu te popis npm skripti i ovisnosti.
- `postcss.config.mjs` – Konfiguracija za PostCSS *pipeline* (npr. Tailwind CSS).
- `prisma.config.ts` – Pomoćne postavke i funkcije povezane s Prismom i bazom podataka.
- `proxy.ts` – Implementira *proxy* logiku i zaštitu API ruta; koristi *middleware* funkcije za kontrolu pristupa.
- `README.md` – Osnovna projektna dokumentacija i upute za pokretanje.
- `tsconfig.json` – Konfiguracija TypeScript kompjajlera.
- `vercel.json` – Konfiguracija *deploymenta* na platformu Vercel.

### 3.2.2. Struktura direktorija `app/`

Struktura direktorija `app/` zasebno je predstavljena zbog svoje ključne uloge u organizaciji kôda i implementaciji poslovne logike aplikacije. Unutar direktorija `app/` nalaze se sljedeći poddirektoriji i datoteke:

```
app/
├── (auth) /
├── (main) /
└── api/
    ├── generated/
    ├── apple-icon.png
    ├── favicon.ico
    ├── globals.css
    ├── icon0.svg
    ├── icon1.png
    └── layout.tsx
└── manifest.json
```

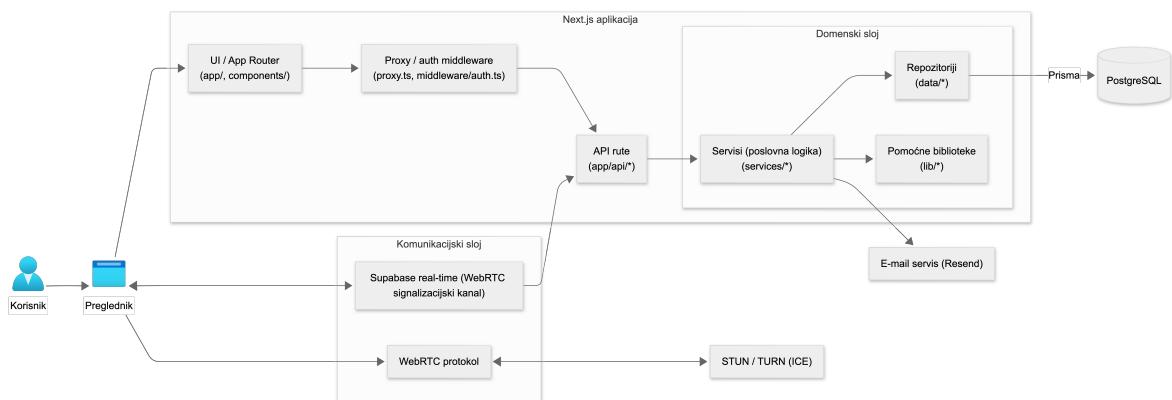
- `(auth) /` – Grupa ruta koja sadrži stranice vezane uz autentikaciju (prijava, registracija, zaboravljena lozinka, ponovno postavljanje lozinke); ove rute su odvojene od ostatka aplikacije radi jasne organizacije.
- `(main) /` – Glavna grupa ruta. Sadrži sve ostale stranice aplikacije koje nisu vezane uz autentikaciju, poput početne stranice, stranica s ponudama repeticija, profilom

korisnika, stranica nadzorne ploče itd.

- `api/` – Sadrži API route datoteke koje implementiraju REST sučelje za rad s korisnicima, tutorima, ponudama repeticija, rezervacijama, recenzijama, predmetima i sobama.
- `generated/` – Direktorij u koji Prisma generira klijentski kôd (Prisma Client) i povezane tipove, koji se potom koriste u poslužiteljskim dijelovima Next.js aplikacije za pristup bazi podataka.
- `apple-icon.png`, `favicon.ico`, `icon0.svg`, `icon1.png` – Ikone aplikacije koje se koriste u preglednicima i pri instalaciji kao progresivna web aplikacija (PWA), npr. za prikaz na početnom zaslonu mobilnog uređaja.
- `globals.css` – Datoteka s globalnim CSS stilovima koji se učitavaju jednom te vrijede za cijelu aplikaciju.
- `layout.tsx` – Korijenski *layout* App Routera koji definira zajedničku strukturu stranica (npr. zaglavje, podnožje i pružatelje konteksta).
- `manifest.json` – PWA manifest koji opisuje osnovne metapodatke aplikacije te omogućuje „instaliranje“ aplikacije na uređaj.

### 3.3. Dijagram arhitekture aplikacije

Na slici 9 prikazan je dijagram arhitekture implementirane aplikacije, koji nastoji ilustrirati ključne komponente i slojeve sustava te njihove međusobne odnose i komunikaciju.



Slika 9: Dijagram arhitekture aplikacije

## 3.4. Rad s bazom podataka

Aplikacija koristi bazu podataka PostgreSQL upravljanu na platformi Supabase kao centralno spremište svih trajnih podataka o korisnicima, ponudama repeticija, rezervacijama termina, recenzijama i ostalim entitetima domene. Za pristup bazi koristi se Prisma ORM, koji se na Supabase Postgres spaja putem stringa za povezivanje (engl. *connection string*) preuzetog iz Supabase sučelja i definiranog u varijablama okruženja aplikacije. Na taj se način kombiniraju prednosti Supabasea (upravljeni PostgreSQL, sigurnost, nadzor) i Prisma ORM-a (tipizirani upiti, migracije, generirani klijent), bez potrebe za održavanjem vlastite baze.

### 3.4.1. Postavljanje i postupci nad bazom podataka

Baza podataka kreirana je kao Supabase projekt, pri čemu Supabase osigurava PostgreSQL instancu. U Supabase sučelju generirani su stringovi za povezivanje s bazom, koji su zatim spremljeni u `.env` datoteku Next.js aplikacije kao varijable `DATABASE_URL` i `DIRECT_URL`. Te se varijable koriste za konfiguraciju Prisma ORM-a pri inicijalizaciji, omogućujući mu da se poveže na PostgreSQL bazu unutar Supabase okruženja.

Prisma ORM služi kao apstrakcijski sloj između domenskog kôda i baze podataka PostgreSQL. U datoteci `schema.prisma` definiraju se modeli koji preslikavaju domenske entitete na tablice u bazi podataka, uključujući tipove stupaca, primarne i strane ključeve te kardinalitet veza. Na temelju te definicije Prisma generira tipizirani klijent (`@prisma/client`) koji se koristi u repozitoriskom sloju aplikacije (`data/`) za izvođenje upita i transakcija nad bazom, dok servisni sloj aplikacije (`services/`) ovisi o repozitorijima umjesto o izravnim SQL upitim, što omogućuje jasnu separaciju poslovne logike i pristupa podatcima.

U ispisu 1 prikazane su naredbe koje su korištene za instalaciju Prisma ORM-a u projektu.

```
1 npm install prisma tsx @types/pg --save-dev  
2 npm install @prisma/client @prisma/adapter-pg dotenv pg
```

#### Ispis 1: Naredbe za instalaciju Prisma ORM-a

Nakon instalacije potrebno je inicijalizirati Prismu naredbom: `npx prisma init`

`-db -output ../app/generated/prisma`. Ova naredba kreira osnovnu strukturu datoteka i direktorija potrebnih za rad s Prismom, uključujući datoteku `schema.prisma`.

U ispisu 2 prikazan je primjer definicije modela baze podataka u datoteci `schema.prisma`.

```
1 model TutorSubject {  
2   tutorId String  
3   subjectId String  
4   lessonOffers LessonOffer[]  
5   subject Subject @relation(fields: [subjectId], references: [id])  
6   tutor User @relation(fields: [tutorId], references: [id])  
7  
8   @@id([tutorId, subjectId])  
9 }
```

### Ispis 2: Primjer definicije modela baze podataka u datoteci `schema.prisma`

Svaki model odgovara jednoj tablici u bazi podataka, a polja unutar modela definiraju stupce tablice, njihove tipove i odnose s drugim modelima.

Migracije baze podataka kreirane su pomoću Prisma Migrate te se izvršavaju naredbom: `npx prisma migrate dev` nakon definiranja ili izmjene modela u `schema.prisma`. Ova naredba generira SQL migracijske datoteke koje se zatim primjenjuju na bazu podataka, stvarajući potrebne tablice i odnose.

Prisma klijent generira se naredbom: `npx prisma generate` nakon definiranja modela, čime se omogućuje korištenje tipiziranog API-ja za pristup bazi podataka unutar aplikacije.

Punjjenje baze podataka inicijalnim testnim podatcima izvršeno je naredbom: `npx prisma db seed` koja pokreće skriptu `prisma/seed.ts` za umetanje testnih zapisa u bazu.

Za administraciju baze i razvoj u lokalnom okruženju korišten je i pomoćni alat za naredbeni redak Supabase CLI (engl. *Command Line Interface*), koji omogućuje upravljanje Supabase projektom i bazom podataka iz terminala. Putem njega je bilo moguće povezati lokalni projekt s udaljenim Supabase projektom, povući ili primijeniti promjene

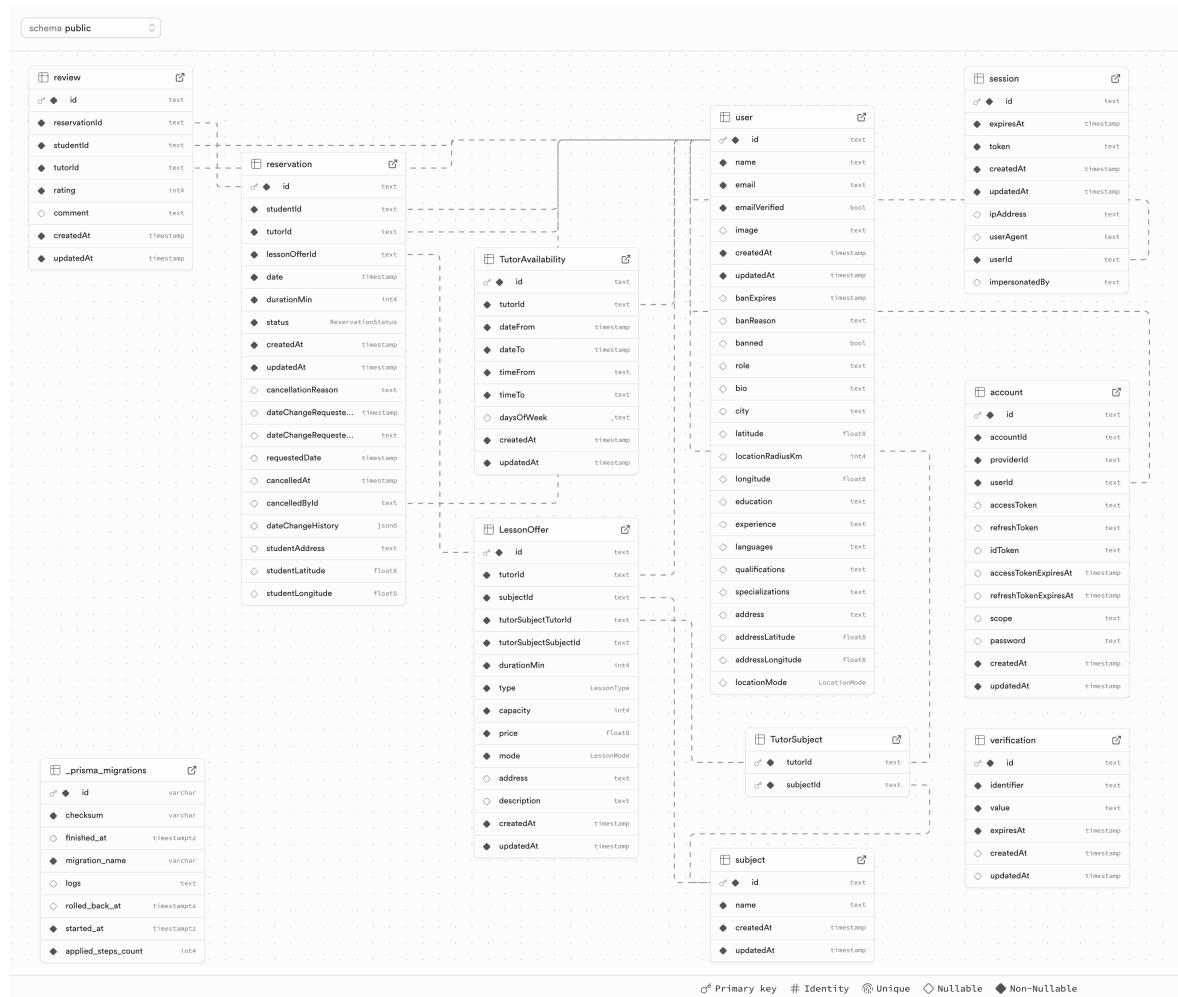
nad bazom (supabase db pull, supabase db push) te izraditi sigurnosne kopije i izvoze sheme baze podataka. Za lokalni razvoj korištene su naredbe supabase start i supabase stop, kojima se pokreće, odnosno zaustavlja kompletan lokalni Supabase stack, što olakšava reprodukciju produkcijskog okruženja na razvojnem računalu [16].

### 3.4.2. Dijagram relacijske baze podataka

Na slici 10 prikazan je ER (engl. *Entity Relationship*) dijagram relacijske baze podataka u sučelju platforme Supabase, na kojem se mogu vidjeti tablice u koje su podatci spremani i njihove međusobne relacije.

Relacijska baza podataka sastoji se od 11 tablica:

- `_prisma_migrations` – tablica koju koristi Prisma za praćenje izvršenih migracija baze podataka.
- `User` – glavna tablica koja pohranjuje osnovne podatke o korisnicima.
- `Session`, `Account`, `Verification` – tablice koje, uz `User`, koristi autentifikacijska biblioteka Better Auth.
- `Subject` – tablica koja pohranjuje informacije o predmetima.
- `TutorSubject` – povezna tablica koja modelira mnogostruku vezu između tablica `User (tutor)` i `Subject`.
- `TutorAvailability` – tablica koja pohranjuje dostupnost tutora za pojedine dane i vremenske intervale.
- `LessonOffer` – tablica koja pohranjuje ponude repeticija koje su tutori kreirali.
- `Reservation` – tablica koja pohranjuje rezervacije koje su korisnici napravili na ponude repeticija.
- `Review` – tablica koja pohranjuje recenzije koje su korisnici ostavili tutorima nakon održanih repeticija.



Slika 10: Dijagram relacijske baze podataka

### 3.5. Repozitoriji u sloju pristupa podatcima

Aplikacija koristi sloj pristupa podatcima (engl. *data access layer*) koji se nalazi u direktoriju `data/` i implementira repozitorije za različite domenske entitete. Repozitoriji su odgovorni za izvođenje operacija nad bazom podataka putem Prisma klijenta. Pojedini repozitoriji implementiraju funkcije za dohvati, kreiranje, ažuriranje i brisanje zapisa u bazi, a servisni sloj aplikacije koristi te repozitorije za implementaciju poslovne logike bez direktne ovisnosti o detaljima pristupa podatcima.

U ispisu 3 prikazan je primjer implementacije repozitorija za entitet `Subject`.

```

1 import prisma from "@lib/prisma"
2
3 export class SubjectRepository {

```

```

4   static async findAll() {
5     return prisma.subject.findMany({
6       orderBy: { createdAt: "desc" },
7     })
8   }
9
10  static async findById(id: string) {
11    return prisma.subject.findUnique({
12      where: { id },
13    })
14  }
15
16  static async create(name: string) {
17    return prisma.subject.create({
18      data: { name },
19    })
20  }
21
22  static async update(id: string, name: string) {
23    return prisma.subject.update({
24      where: { id },
25      data: { name },
26    })
27  }
28
29  static async delete(id: string) {
30    return prisma.subject.delete({
31      where: { id },
32    })
33  }
34 }

```

**Ispis 3:** Primjer implementacije repozitorija za entitet Subject

### 3.6. Poslovna logika u servisnom sloju

Poslovna logika aplikacije implementirana je u servisnom sloju, koji se nalazi u direktoriju `services/`. Servisi koriste repozitorije za pristup podatcima i implementiraju

funkcionalnosti koje su specifične za poslovne procese aplikacije, poput kreiranja ponuda repeticija, rezervacija termina, ostavljanja recenzija i sl. Servisi su dizajnirani tako da budu neovisni o specifičnim detaljima baze podataka, što omogućuje fleksibilnost u slučaju promjena u sloju pristupa podatcima ili migracije na drugi sustav za pohranu.

U ispisu 4 prikazan je primjer implementacije servisne funkcije za kreiranje ponude repeticija.

```
1 import { LessonOfferRepository, TutorRepository } from "@/data"
2 import { ApiError } from "@/lib/utils/error-handler"
3 import { LessonMode, LessonType, LocationMode } from "@/app/generated/
prisma/enums"
4
5 export class LessonOfferService {
6   static async createLessonOffer(data: {
7     tutorId: string
8     subjectId: string
9     durationMin: number
10    type: LessonType
11    capacity: number
12    price: number
13    mode: LessonMode
14    address?: string
15    description?: string
16  }) {
17    // Validacija numeričkih polja
18    if (Number.isNaN(data.durationMin) || data.durationMin <= 0) {
19      throw new ApiError(400, "Invalid durationMin, capacity or price")
20    }
21
22    // Provjera tutorovih postavki
23    const tutor = await TutorRepository.findById(data.tutorId)
24    if (!tutor) throw new ApiError(400, "Tutor nije pronađen")
25
26    if (tutor.locationMode === LocationMode.ONLINE &&
27        data.mode === LessonMode.ONSITE) {
28      throw new ApiError(400, "Online tutor ne može kreirati uživo ponude
")
```

```

29     }
30
31     // Postavljanje adrese za HOST tutore
32     const address = (tutor.locationMode === LocationMode.HOST &&
33         data.mode === LessonMode.ONBSITE)
34         ? tutor.address : null
35
36     return await LessonOfferRepository.create({ ...data, address })
37 }
38 }
```

**Ispis 4:** Primjer implementacije servisne funkcije za kreiranje ponude repeticija

### 3.7. Usmjeravanje zahtjeva

Next.js App Router koristi datotečnu strukturu unutar direktorija `app/` za definiranje ruta i API krajeva (engl. *endpoints*). Struktura direktorija i datoteka unutar `app/` određuje URL putanje koje aplikacija podržava, dok datoteke unutar direktorija `app/api/` definiraju REST API krajeve [10]. Na primjer, datoteka `app/api/subjects/route.ts` implementira API rutu `/api/subjects` koja podržava HTTP metode GET i POST za rad s predmetima.

App Router koristi koncept segmenata ruta, pri čemu svaka podmapa unutar `app/` predstavlja jedan segment URL putanje, a konačni segment koji sadrži datoteku `page.tsx` definira stranicu koja se prikazuje korisniku. Na primjer, struktura `app/dashboard/page.tsx` definira rutu `/dashboard`, dok se višerazinske rute ostvaruju ugnježđivanjem direktorija, npr. `app/dashboard/reservations/page.tsx` definirala bi rutu `/dashboard/reservations`.

Uz stranice, App Router uvodi i *layout* datoteke koje omogućuju dijeljenje zajedničkog izgleda i elemenata sučelja između više ruta. Datoteka `app/layout.tsx` definira korijenski izgled koji se primjenjuje na sve rute u aplikaciji, dok se dodatne `layout.tsx` datoteke mogu nalaziti u podmapama kako bi se definirali specifični izgledi za pojedine sekcije, npr. za korisničko sučelje nakon prijave ili za administratorsko sučelje. Svaki *layout* obavija priпадne stranice, čime se postiže nasljeđivanje i ponovno korištenje zajedničkih elemenata kao što su navigacija, zaglavljje ili podnožje.

Dinamički segmenti ruta definiraju se korištenjem uglatih zagrada u nazivima direkto-rija, npr. `app/subjects/[id]/page.tsx` definira rutu `/subjects/{id}`, gdje se `{id}` u stvarnom URL-u zamjenjuje konkretnim identifikatorom predmeta. Na taj način App Router omogućuje jednostavnu izgradnju detaljnih stranica za pojedine entitete (predmete, ponude, repeticija, rezervacije i sl.) bez potrebe za ručnim definiranjem ruta.

Opisani mehanizam usmjeravanja omogućuje da se korisničko sučelje i API krajevi organiziraju na konzistentan način unutar istog direktorija `app/`, pri čemu se čitljivost i održavanje kôda poboljšavaju zahvaljujući jasnom preslikavanju između strukture datoteka i URL putanja aplikacije.

U ispisu 5 prikazan je primjer implementacije API rute za dohvat svih predmeta.

```
1 import { NextResponse } from "next/server"
2 import { SubjectRepository } from "@/data"
3
4 import { ApiError, handleApiError } from "@/lib/utils/error-handler"
5
6 // GET /api/subjects
7 export async function GET() {
8     try {
9         const subjects = await SubjectRepository.findAll()
10        return NextResponse.json(subjects)
11    } catch (err) {
12        return handleApiError(err)
13    }
14 }
```

**Ispis 5:** Primjer implementacije API rute za dohvat svih predmeta

Tablica 1 prikazuje glavne rute korisničkog sučelja koje su definirane u direktoriju `app/`:

**Tablica 1:** Pregled glavnih ruta korisničkog sučelja u App Routeru

Funkcija / tip	Ruta	Tip komponente
function	/ (auth) / (routes) /forget-password/	use client
function	/ (auth) / (routes) /reset-password/	use client
function	/ (auth) / (routes) /sign-in/	server
function	/ (auth) / (routes) /sign-up/	server
function	/ (main) / (routes) /account/	use client
function	/ (main) / (routes) /dashboard/	use client
function	/ (main) / (routes) /offers/	use client
function	/ (main) / (routes) /room/[id] /	server
function	/ (main) / (routes) /tutors/[id] /	use client
Home	/ (main) /	server

REST API krajevi implementirani su u direktoriju `app/api/`, pri čemu svaka datoteka `route.ts` unutar odgovarajuće podmape definira HTTP metode podržane na toj ruti.

Pregled API krajeva dan je u tablici 2:

**Tablica 2:** Pregled API krajeva implementiranih u direktoriju `app/api/`

Metoda	Ruta
GET	/api/analytics
GET	/api/auth/ [...all]
GET	/api/auth/me
PATCH	/api/auth/me
POST	/api/auth/update-role
GET	/api/cron/complete-reservations
DELETE	/api/lesson-offers/ [id]
GET	/api/lesson-offers
POST	/api/lesson-offers
PATCH	/api/lesson-offers
GET	/api/reservations/ [id]
PATCH	/api/reservations/ [id]
GET	/api/reservations

Metoda	Ruta
POST	/api/reservations
DELETE	/api/reviews/[id]
PATCH	/api/reviews/[id]
GET	/api/reviews
POST	/api/reviews
GET	/api/rooms/[id]
GET	/api/subjects/[id]
DELETE	/api/subjects/[id]
PATCH	/api/subjects/[id]
GET	/api/subjects
POST	/api/subjects
GET	/api/tutor/availability
POST	/api/tutor/availability
DELETE	/api/tutor/availability
GET	/api/tutor/location
POST	/api/tutor/location
GET	/api/tutor/reservations
PATCH	/api/tutor/reservations
GET	/api/tutor/subjects
POST	/api/tutor/subjects
GET	/api/tutors/[id]
GET	/api/users

### 3.8. Implementacija autentikacije i autorizacije

Za implementaciju autentikacije i autorizacije korisnika u aplikaciji koristi se biblioteka **Better Auth**, namijenjena modernim JavaScript i TypeScript aplikacijama te integrirana s Next.js okvirom i Prisma ORM-om. Better Auth pruža funkcionalnosti za upravljanje korisničkim računima, *heširanje* lozinki, upravljanje sesijama i različite strategije autentikacije (e-pošta/lozinka, OAuth i dr.), pri čemu sav osjetljivi podatkovni sloj ostaje u PostgreSQL-u, a biblioteka generira tipizirani API za autentikaciju i autorizaciju [28].

Instalacija biblioteke Better Auth obavlja se dodavanjem paketa u projekt naredbom `npm install better-auth`, nakon čega se u datoteci `.env` definiraju varijable okruženja `BETTER_AUTH_SECRET` i `BETTER_AUTH_URL` koje biblioteka koristi za potpisivanje sesija i određivanje baznog URL-a aplikacije. Zatim se u datoteci `lib/auth.ts` (pričazanoj u ispisu 6) inicijalizira Better Auth instanca, pri čemu se konfigurira povezivanje s Prisma ORM-om i bazom podataka te odabiru metode autentikacije koje će se koristiti u aplikaciji (u ovom slučaju e-pošta/lozinka). Na temelju te konfiguracije Better Auth generira potrebne strukture u bazi podataka (tablice i polja za korisnike, sesije i povezane entitete) te izlaže rutu `/api/auth/ [...].all` kojom se centralizira sav promet vezan uz autentifikaciju.

Na klijentskoj strani definiran je pomoćni modul (npr. `lib/auth-client.ts`) u kojem se, korištenjem `createAuthClient` funkcije, inicijalizira klijentska instanca Better Auth-a i konfiguriraju dodaci koje će koristiti korisničko sučelje. Ovaj modul izlaže funkcije za registraciju, prijavu, odjavu i dohvati podataka o trenutno prijavljenom korisniku, koje se pozivaju iz React komponenti stranica za prijavu, registraciju i reset lozinke, čime se smanjuje duplicitiranje kôda i pojednostavljuje upravljanje autentifikacijom u korisničkom sučelju.

```
1 import { betterAuth } from "better-auth";
2 import { prismaAdapter } from "better-auth/adapters/prisma";
3 import { adminPlugin } from "better-auth/plugins/admin";
4
5 import { prisma } from "./db";
6 import { ac, student, tutor, admin } from "./permissions";
7
8 export const auth = betterAuth({
9   appName: "repeticije-hr",
10  database: prismaAdapter(prisma, {
11    provider: "postgresql",
12  }),
13  emailAndPassword: {
14    enabled: true,
15  },
16  plugins: [
17    adminPlugin({
```

```

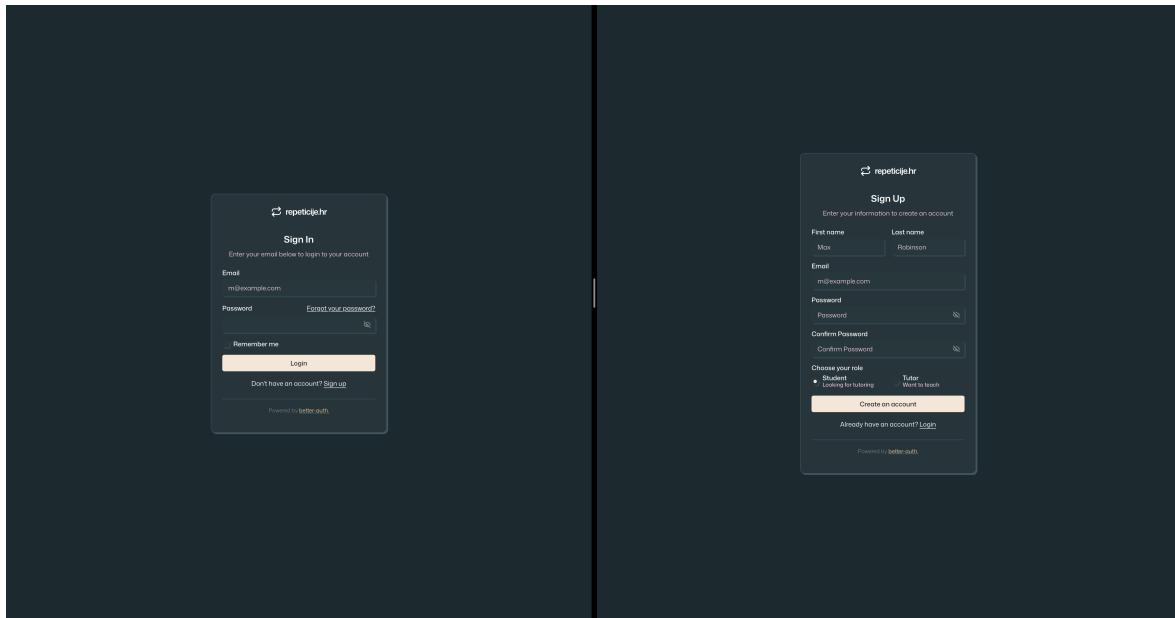
18     ac,
19     roles: {
20       student,
21       tutor,
22       admin,
23     },
24     defaultRole: "student",
25     adminRoles: ["admin"],
26   } ),
27 ],
28 } );

```

### Ispis 6: Konfiguracija biblioteke Better Auth i Admin dodatka

Autorizacija se temelji na podacima o korisniku i ulozi pohranjenima u sesiji. Na poslužiteljskoj strani posredničke funkcije provjeravaju postoji li valjana sesija i ima li korisnik odgovarajuću ulogu (tutor, student, administrator) za pristup određenoj ruti ili API krajevima; u suprotnom se zahtjev preusmjerava na prijavu ili vraća odgovarajući HTTP status.

Na slici 11 prikazane su stranice za prijavu i registraciju koje koriste Better Auth. Na slici je također vidljiva tamna tema, koja je omogućena globalnim CSS-om i može se primijeniti na cijelu aplikaciju.



Slika 11: Stranice za prijavu i registraciju

### 3.8.1. Administratorski dodatak, uloge i impersonacija korisnika

Za naprednije upravljanje korisnicima i ulogama koristi se *Admin* dodatak Better Auth biblioteke, koji dodaje administratorske operacije poput kreiranja računa, izmjene uloga, zabrane i impersonacije korisnika [28]. Prava pristupa definiraju se u zasebnoj datoteci permissions.ts (ispis 7), korištenjem pristupnog kontrolera (createAccessControl), gdje se opisuju dopuštene akcije nad resursima i definiraju uloge student, tutor i admin.

```
1 import { createAccessControl } from "better-auth/plugins/access"
2 import {
3   adminAc,
4   defaultStatements,
5   userAc,
6 } from "better-auth/plugins/admin/access"
7
8 const statement = {
9   ...defaultStatements,
10  subject: ["create", "list", "update", "delete"],
11  tutorSubject: ["create", "list", "update", "delete"],
12  tutorAvailability: ["create", "list", "update", "delete"],
13  lessonOffer: ["create", "list", "update", "delete"],
14  reservation: [
15    "create",
16    "list",
17    "update",
18    "delete",
19    "approve",
20    "reject",
21    "cancel",
22  ],
23  review: ["create", "list", "update", "delete"],
24  profile: ["view", "update"],
25 } as const
26
27 export const ac = createAccessControl(statement)
28
29 export const student = ac.newRole({
30   ...userAc.statements,
```

```

31   // Definiranje prava pristupa za studente
32 })
33
34 export const tutor = ac.newRole({
35   ...userAc.statements,
36   // Definiranje prava pristupa za tutore
37 })
38
39 export const admin = ac.newRole({
40   // Admins have full access to all models
41   ...adminAc.statements,
42   // Definiranje dodatnih prava pristupa za administratore, ako je
43   // potrebno
43 })

```

### Ispis 7: Definiranje uloga i dopuštenih akcija u datoteci permissions.ts

Ovaj model pristupa koristi se u konfiguraciji auth.ts pri inicijalizaciji *Admin* podataka, čime se centralno definira da aplikacija razlikuje studente, tutore i administratore te da administratori imaju proširene mogućnosti, uključujući i impersonaciju korisnika.

Ključna funkcionalnost u ovoj aplikaciji je *impersonacija* korisnika, kojom se administrator privremeno prijavljuje kao drugi korisnik radi reprodukcije i analize problema. Povizom metode authClient.admin.impersonateUser s identifikatorom korisnika stvara se nova sesija koja se u ostatku aplikacije tretira kao sesija tog korisnika, uz dodatno polje koje bilježi identitet administratora koji je impersonaciju pokrenuo. Trajanje takve sesije ograničeno je opcijom impersonationSessionDuration, nakon čijeg isteka se sesija automatski poništava.

Zahvaljujući ovoj korisnoj funkcionalnosti nije implementirana zasebna administratorska nadzorna ploča (osim za upravljanje korisnicima i predmetima), već su administrativne funkcionalnosti integrirane u postojeće korisničko sučelje s obzirom na to da se aplikacija oslanja na impersonaciju kao glavni mehanizam za administraciju i dijagnostiku.

### 3.8.2. Proxy, middleware i zaštita API ruta

Kako bi se zaštitile odabrane stranice i API rute, u aplikaciji se koristi *proxy* sloj i pomoćne *middleware* funkcije koje provjeravaju postojanje korisničke sesije i, po potrebi, dodatne dozvole pristupa.

*Proxy* sloj definiran je u datoteci `proxy.ts` (ispis 8). On presreće zahtjeve prema odabranim putanjama i prije renderiranja provjerava je li korisnik prijavljen, koristeći funkciju `requireAuth` iz modula `middleware/auth.ts`. Ako korisnik nije prijavljen, zahtjev se preusmjerava na stranicu za prijavu:

```
1 import { NextResponse, type NextRequest } from "next/server"
2 import { requireAuth } from "@/middleware/auth"
3
4 export async function proxy(request: NextRequest) {
5   const protectedPaths = ["/dashboard", "/account", "/room"]
6   const isProtected = protectedPaths.some((path) =>
7     request.nextUrl.pathname.startsWith(path)
8   )
9
10  if (isProtected) {
11    const { error } = await requireAuth(request.headers)
12
13    if (error) {
14      return NextResponse.redirect(new URL("/sign-in", request.url))
15    }
16  }
17
18  return NextResponse.next()
19 }
20
21 export const config = {
22   matcher: ["/dashboard/:path*", "/account/:path*", "/room/:path*"],
23 }
```

Ispis 8: Proxy sloj za zaštitu odabralih ruta

Stvarna provjera sesije i dozvola implementirana je u datoteci `middleware/auth.ts`.

Funkcija `requireAuth` (ispis 9) dohvaca sesiju trenutnog korisnika putem Better Auth API-ja i, ako sesija ne postoji ili nema korisnički identifikator, vraća HTTP odgovor s pogreškom:

```
1 import { headers } from "next/headers"
2 import { NextResponse, type NextRequest } from "next/server"
3
4 import { auth } from "@/lib/auth"
5
6 export async function requireAuth(requestHeaders?: NextRequest["headers"]
7   ] ) {
8   const session = await auth.api.getSession({
9     headers: requestHeaders || (await headers()),
10   })
11
12   if (!session?.user?.id) {
13     return {
14       error: NextResponse.json({ error: "Unauthorized" }, { status: 401
15     }) ,
16       session: null,
17     }
18   }
19 }
```

### Ispis 9: Provjera autentikacije u middleware/auth.ts

Na temelju ove funkcije gradi se i `requirePermission` (ispis 10), koja dodatno provjerava ima li korisnik pravo izvođenja tražene akcije nad određenim resursima. Nakon što potvrdi postojanje valjane sesije, funkcija poziva metodu `userHasPermission` Better Auth API-ja te vraća odgovarajući status (401, 403) ili dopušta nastavak izvršavanja API rute:

```
1 export async function requirePermission(permissions: Record<string,
2   string[]> {
```

```

2  const { error, session } = await requireAuth()
3  if (error) return { error, session: null }
4
5  if (!session) {
6    return {
7      error: NextResponse.json({ error: "Unauthorized" }, { status: 401
8    }),
9      session: null,
10    }
11
12  const allowed = await auth.api.userHasPermission({
13    body: {
14      userId: session.user.id,
15      permissions,
16    },
17  })
18
19  if (!allowed) {
20    return {
21      error: NextResponse.json({ error: "Forbidden" }, { status: 403 }),
22      session: null,
23    }
24  }
25
26  return { error: null, session }
27 }

```

### Ispis 10: Provjera dozvola za izvođenje akcija

U praksi se opisane funkcije koriste na dva načina. *Proxy* sloj (`proxy.ts`) poziva `requireAuth` kako bi zaštitio čitave stranice prije renderiranja, dok se u pojedinačnim API rutama na početku *handlera* poziva `requirePermission` s odgovarajućim skupom dozvola. Time se osigurava da samo prijavljeni korisnici s odgovarajućim ovlastima mogu pristupiti zaštićenim resursima i izvršavati osjetljive operacije unutar aplikacije.

### **3.9. Validacija vremenske zone**

Aplikacija je prvenstveno namijenjena korisnicima u Hrvatskoj i susjednim zemljama te podržava vremenske zone koje koriste CET/CEST (UTC+1 zimi, UTC+2 ljeti), poput Europe/Zagreb, Europe/Belgrade, Europe/Sarajevo, Europe/Ljubljana, Europe/Skopje i Europe/Podgorica. Sva vremena se u sučelju prikazuju u lokalnoj vremenskoj zoni preglednika, s hrvatskom lokalizacijom hr-HR, dok se u bazi podataka pohranjuju u UTC formatu, čime se izbjegavaju razlike između klijenta i poslužitelja.

Pri dizajnu ovog sustava validacije primijenjen je tzv. YAGNI princip (*You Aren't Gonna Need It*). Umjesto da se unaprijed implementira potpuna međunarodna podrška (spremanje korisničke vremenske zone u model korisnika, konverzije za svakog sudionika, globalni selektori zona), početna verzija ograničena je na skup CET/CEST zona koji pokriva ciljanu skupinu korisnika.

Centralna logika za rad s vremenskim zonama nalazi se u modulu lib/utils/timezone-helper.ts, koji sadrži funkcije koje detektiraju vremensku zonu preglednika, provjeravaju pripada li podržanom skupu te upozoravaju korisnika ako nije u očekivanoj CET/CEST zoni i po potrebi blokiraju kritične radnje. Na temelju tih funkcija implementirana je komponenta TimezoneWarning koja se koristi u različitim dijelovima sučelja (npr. globalni *banner*, kompaktan prikaz na stranici rezervacija) te je validacija vremenske zone integrirana u ključne ekrane aplikacije.

### **3.10. Integracija Google Maps API-ja**

Integracija servisa Google Maps u aplikaciji koristi Maps JavaScript API, Geocoding i Places kako bi se omogućilo automatsko popunjavanje (engl. *autocomplete*) adresa, geokodiranje na poslužitelju te filtriranje i validacija ponuda prema udaljenosti.

Aplikacija je podijeljena na klijentski i poslužiteljski sloj. Klijentski sloj učitava biblioteke Maps JavaScript API i Places preko React *provider* komponenti, prikazuje *autocomplete* polje adrese i prikuplja odabrane lokacije (tekst + koordinate) iz korisničkog preglednika. Poslužiteljski sloj koristi Google Geocoding REST API za pretvaranje tekstualnih adresa i gradova u koordinate, a zatim te koordinate koristi u poslovnoj logici (npr. provjera je li student unutar radijusa koji je tutor definirao).

Na klijentskoj strani stranica za pretraživanje ponuda koristi biblioteku `@vis.gl/react-google-maps` [29], koja pruža komponente `APIProvider`, `Map` i pripadne kuke (npr. `useMapsLibrary`) za učitavanje biblioteka Maps JavaScript API i Places. Na temelju toga implementirano je polje s *autocomplete* funkcionalnošću: korisnik upisuje adresu ili grad, komponenta prikazuje prijedloge iz Places API-ja, a nakon odabira dohvaća normalizirani tekst i geografske koordinate koje se šalju *backendu* kao numeričke vrijednosti (*latitude/longitude*). Te koordinate služe i za prikaz lokacije na karti i za lokalne UI filtre.

Na poslužiteljskoj strani servisi za ponude i tutore koriste Geocoding API kako bi iz korisničkih upita (npr. „Split”) i tekstualnih adresa tutora dobili koordinate. Pri filtriranju ponuda po lokaciji servis najprije geokodira korisnički upit i dobiva referentni centar, a zatim za tutorove adrese koje još nemaju koordinate radi *batch* geokodiranje. Stringovi adresa se dedupliciraju, geokodiraju u ograničenom paralelnom setu i spremaju u privremenu mapu kako bi se izbjegli dupli pozivi unutar istog zahtjeva.

Za izračun udaljenosti između dvije točke koristi se tzv. Haversinova formula implementirana u pomoćnoj funkciji `getDistanceFromLatLonInKm`, koja vraća rezultat u kilometrima. Ona se koristi za filtriranje ponuda unutar zadanog radijusa te pri validaciji rezervacija koje se održavaju kod studenta u odnosu na `locationRadiusKm` koji tutor ima postavljen. Prije samog izračuna provjerava se da su koordinate i radijus definirani.

API ključ za Google Maps čuva se u varijabli okruženja `NEXT_PUBLIC_GOOGLE_MAPS_API_KEY`. Iako se ključ mora učitati na klijentu zbog Maps JavaScript API-ja, njegova je upotreba ograničena konfiguracijom u Google Cloud konzoli (ograničenje po domeni / HTTP *refereru*) kako bi se spriječila zlonamjerna ili neovlaštena upotreba.

### 3.11. Implementacija podrške za *online* repeticije

Podrška za *online* repeticije u domeni aplikacije modelirana je kroz enumeraciju `LessonMode` s barem dvije vrijednosti: `ONLINE` i `ONSITE`. Na taj se način u poslovnoj logici i korisničkom sučelju jasno razlikuju *online* i nastava uživo, iako dijele zajedničke entitete kao što su ponude, rezervacije i profili korisnika. U sučelju filtriranje i prikaz ponuda repeticija jasno razdvaja *online* i *onsite* ponude, a kod slanja obavijesti e-poštom za *online* lekcije adresa lokacije je `null` te se u predlošku prikazuje oznaka „*Online*” umjesto fizičke adrese.

Dodatna pravila u domenskom modelu (npr. polje `tutor.locationMode`) sprječavaju kontradiktorne kombinacije poput situacije da je tutor označen kao isključivo *online*, a istovremeno ima aktivne `ONSITE` ponude. Time se konzistentnost podataka osigurava na razini modela, a ne samo u korisničkom sučelju.

### 3.11.1. Inicijalizacija sobe za sastanke

*Online* repeticije odvijaju se kroz WebRTC video pozive u tzv. sobama za sastanke. Inicijalizacija sobe implementirana je u klijentskoj React komponenti `RoomCall`, koja upravlja signalizacijskim klijentom, lokalnim medijskim tokovima i mapom WebRTC veza prema ostalim sudionicima. Pri pokretanju poziva komponenta najprije pribavlja lokalni audio i video tok korisnika, kao u sljedećem ispisu kôda:

```
1 const stream = await navigator.mediaDevices.getUserMedia ( {  
2   audio: true,  
3   video: true,  
4 } )  
5 // Postavi lokalni video element i dodaj medijske tokove u  
RTCPeerConnection
```

**Ispis 11:** Pribavljanje lokalnog audio i video toka

Dobiveni `MediaStream` se postavlja na lokalni video element radi pregleda, a pojedinačni `MediaStreamTrack` objekti dodaju se na `RTCPeerConnection` instance. U sklopu inicijalizacije definiraju se ICE poslužitelji (STUN/TURN) i medijska ograničenja (npr. omogućavanje mikrofona i kamere), čime se priprema okruženje za uspostavu WebRTC veza.

### 3.11.2. Integracija s WebRTC-om i implementacija video poziva

Kako bi se omogućilo održavanje grupnih *online* repeticija, video pozivi su implementirani u *full-mesh* topologiji, pri čemu svaki sudionik uspostavlja zasebnu `RTCPeerConnection` vezu prema svakom drugom sudioniku. Za razmjenu signalnih poruka koristi se signalni kanal temeljen na Supabase kanalima; iznad njega je definiran modul sa shemama poruka (npr. `OfferMessage`, `AnswerMessage`, `CandidateMessage`) koji brine o serijalizaciji i

validaciji.

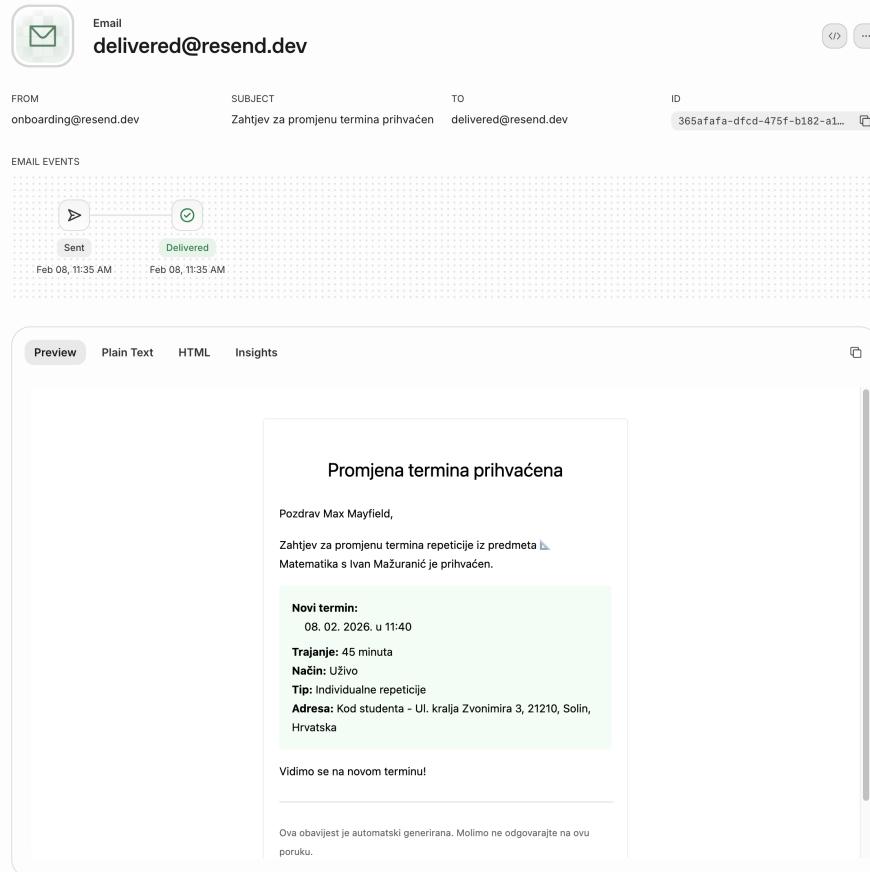
Životni ciklus WebRTC sesije slijedi standardni obrazac: inicijator poziva kreira ponudu (`createOffer`), postavlja je kao lokalni opis (`setLocalDescription`) i šalje drugoj strani preko signaliziranja. Primatelj postavlja ponudu kao udaljeni opis (`setRemoteDescription`), kreira odgovor (`createAnswer`), postavlja ga lokalno i vraća inicijatoru, dok se paralelno razmjenjuju ICE kandidati dok se ne uspostavi stabilna veza. Nakon uspostave veze lokalni audio i video tokovi dodaju se na sve relevantne veze, a pristigle udaljene *streamove* komponenta RoomCall prikazuje u sučelju. Kod završetka poziva implementiran je postupak čišćenja: zaustavljanje lokalnih MediaStreamTrack objekata, uklanjanje udaljenih *streamova* i zatvaranje svih RTCPeerConnection instanci.

### 3.12. Slanje obavijesti putem e-pošte i Resend integracija

Sustav obavijesti putem e-pošte temelji se na predlošcima i centralnom servisu EmailService. Ovaj servis renderira React predloške za pojedine scenarije (potvrda rezervacije, novi zahtjev, odobrenje ili odbijanje termina, promjena i otkazivanje) te preko konfiguiriranog pružatelja usluge šalje poruke korisnicima. Svaki predložak prima potrebne podatke (ime sudionika, predmet, datum i vrijeme, način izvođenja itd.) i generira poruku e-pošte u skladu s potrebama aplikacije.

U trenutnoj fazi razvoja koristi se zamjenska konfiguracija za lokalno i testno slanje e-pošte na platformi Resend. Podešavanje uključuje definiranje varijabli okruženja `RESEND_API_KEY` i `RESEND_FROM_EMAIL` [30], nakon čega se u EmailService implementira funkcija `sendEmail` koja koristi Resend API za slanje poruka.

Primjer testne dolazne poruke e-pošte prikazan je na slici 12.



**Slika 12:** Primjer testne poruke e-pošte poslane preko Resend API-ja

### 3.13. Automatizirani cron zadatci

Za ponavljajuće poslove u sustavu predviđeni su automatizirani *cron* zadatci koji periodično pozivaju metode u servisnom sloju. Tipičan primjer je kompletiranje prošlih rezervacija – servis za rezervacije pronalazi sve zapise čiji je izračunati kraj termina manji ili jednak trenutnom vremenu te im ažurira status na COMPLETED.

Na producijskom okruženju periodično pokretanje ovih zadataka ostvareno je pomoću Vercel *cron* mehanizma. U konfiguracijskoj datoteci `vercel.json` definiran je zadatak koji jednom dnevno poziva API rutu zaduženu za kompletiranje rezervacija (ispis 12) [31]:

```

1  {
2    "crons": [

```

```
3     {
4         "path": "/api/cron/complete-reservations",
5         "schedule": "0 0 * * *"
6     }
7 ]
8 }
```

**Ispis 12:** Konfiguracija Vercel *cron* zadatka

Ruta `/api/cron/complete-reservations` poziva servisnu metodu koja odrađuje opisanu logiku nad bazom podataka. Vercel u ovom slučaju služi samo kao pouzdani okidač u zadanoj vremenskoj shemi, dok se sva poslovna pravila i transakcijska sigurnost nalaze unutar same aplikacije.

### 3.14. Prezentacija funkcionalnosti aplikacije kroz korisničko sučelje

Aplikacija je dizajnirana tako da različiti tipovi korisnika (neprijavljeni, studenti, tutori, administratori) imaju pristup funkcionalnostima sukladno svojim potrebama i ovlastima.

Neprijavljeni korisnici imaju pristup početnoj stranici s osnovnim informacijama o aplikaciji, mogu posjetiti tutor profile i pregledavati te filtrirati cjelokupnu ponudu repeticija uz dohvaćanje dostupnih termina za određenu ponudu, međutim nisu u mogućnosti rezervirati termine.

repeticije.hr      Ponuda repeticija

Filtriraj ponude

Nalj. odgovorenog  
Odaberite razinu  
Šir nobira

Predmet  
Naziv predmeta  
npr. Matematika

Razine  
Sve razine

Termin  
Datum i vrijeme  
dd mm yyyy, --:

Aktuelni sortiranj  
Sortiraj po  
Nizno

Prihvati      Odustani

Slika 13: Stranica s oglasima različitih ponuda repeticija

**Slika 14:** Javni profil tutora

Svaki prijavljeni korisnik, bez obzira na ulogu, ima pristup stranici gdje može pregledati i urediti svoje osobne podatke i avatar, promijeniti lozinku i upravljati postavkama računa.

Studenti, nakon prijave, mogu pristupiti nadzornoj ploči gdje imaju izbor između sljedećih funkcionalnosti:

- Analitika: Prikaz osnovnih statistika atraktivnih za studente.
- Moj kalendar: Prikaz studentovih rezervacija u kalendarskom prikazu (slika 15).
- Moje rezervacije: Detaljan popis svih rezervacija s mogućnošću otkazivanja, promjene termina i ocjenjivanja (slike 16 – 18).

**Slika 15:** Kalendarski prikaz studentovih rezervacija

**Slika 16:** Nadolazeće rezervacije studenta

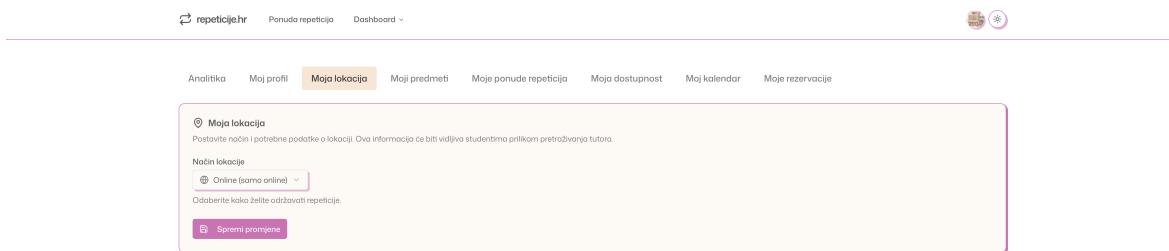
**Slika 17:** Prethodne rezervacije studenta



**Slika 18:** Neostvarene rezervacije studenta

Tutori, nakon prijave, imaju pristup nadzornoj ploči s funkcionalnostima:

- Analitika: Prikaz statistika relevantnih za tutore.
- Moj profil: Mogućnost uređivanja osobnih podataka za tutor profil.
- Moja lokacija: Postavljanje načina rada (*online/onsite* uz varijacije „Domaćin” ili „Putujem”) i ovisno o tome, adrese i radiusa djelovanja (slike 19–21).
- Moji predmeti: Upravljanje predmetima koje tutor podučava (slika 22).
- Moje ponude repeticija: Kreiranje, uređivanje i brisanje ponuda repeticija (slika 23).
- Moja dostupnost: Postavljanje vremenskih intervala kada je tutor dostupan za rezervacije (slika 24).
- Moj kalendar: Prikaz tutorovih rezervacija i dostupnosti u kalendarskom prikazu (slika 25).
- Moje rezervacije: Detaljan popis svih rezervacija s mogućnošću otkazivanja i promjene termina.



**Slika 19:** Lokacija: *Online*

[repeticije.hr](#) Ponuda repeticija Dashboard

Analitika Moj profil **Moja lokacija** Moji predmeti Moje ponude repeticija Moja dostupnost Moj kalendar Moje rezervacije

**Moja lokacija**

Postavite način i potrebne podatke o lokaciji. Ova informacija će biti vidljiva studentima prilikom pretraživanja tutora.

Način lokacije

Domaćin (studenti dolaze kod mene)  Putujem (dolazim kod studentu)

Odselite kako želite održavati repeticije.

Adresa određivanja

Kopilica ul. 1, 21000 Split, Hrvatska

Točna adresa gdje održavate repeticije (provjerite točnost ukoliko koristite automatski unos i po potrebi ispravite, npr. dodajte kućni broj, kat, ston)

Koristi moju lokaciju Koordinate: (43.5260, 16.4596)

Pregled lokacije

Karta  Satelit

Tiskovni prelazi Podaci karte ©2020 Google Uredi Projavi pogrešku na karti

Plava zona pokazuje područje unutar kojeg ste dostupni za repeticije

**Slika 20:** Lokacija: *Onsite* (Domaćin)

[repeticije.hr](#) Ponuda repeticija Dashboard

Analitika Moj profil **Moja lokacija** Moji predmeti Moje ponude repeticija Moja dostupnost Moj kalendar Moje rezervacije

**Moja lokacija**

Postavite način i potrebne podatke o lokaciji. Ova informacija će biti vidljiva studentima prilikom pretraživanja tutora.

Način lokacije

Putujem (dolazim kod studentu)  Domaćin (studenti dolaze kod mene)

Odselite kako želite održavati repeticije.

Grad

Solin, Hrvatska

Radijus (km)

3 km

Maksimalno udaljenje za dolazak (1 - 50 km)

Pregled lokacije

Karta  Satelit

Tiskovni prelazi Podaci karte ©2020 Google Uredi Projavi pogrešku na karti

Plava zona pokazuje područje unutar kojeg ste dostupni za repeticije

**Slika 21:** Lokacija: *Onsite* (Putujem)

[repeticije.hr](#) Ponuda repeticija Dashboard

Analitika Moj profil Moja lokacija **Moji predmeti** Moje ponude repeticija Moja dostupnost Moj kalendar Moje rezervacije

**Moji predmeti**

Odselite predmete koje želite predavati. Ova informacija će biti vidljiva studentima prilikom pretraživanja tutora.

<input checked="" type="checkbox"/> Ekonomija	<input checked="" type="checkbox"/> Informatika	<input checked="" type="checkbox"/> Geografija
<input checked="" type="checkbox"/> Povijest	<input checked="" type="checkbox"/> Hrvatski	<input checked="" type="checkbox"/> Engleski
<input checked="" type="checkbox"/> Biologija	<input checked="" type="checkbox"/> Kemija	<input checked="" type="checkbox"/> Fizika
<input checked="" type="checkbox"/> Matematika		

3 od 10 predmeta

**Slika 22:** Upravljanje predmetima koje tutor predaje

**Slika 23:** Popis i upravljanje ponudama za tuteure

**Slika 24:** Postavke dostupnosti za tuteure

**Slika 25:** Kalendarski prikaz tutorovih rezervacija i dostupnosti

Administratori, nakon prijave, imaju pristup nadzornoj ploči s funkcionalnostima:

- Analitika: Prikaz sveobuhvatnih statistika o korištenju aplikacije (slika 26).
- Upravljanje korisnicima: Pregled, kreiranje, uređivanje i brisanje korisnika, uključujući mogućnost impersonacije s ciljem upravljanja umjesto pojedinog korisnika i brže dijagnostike problema (slika 27).
- Upravljanje predmetima: Pregled, kreiranje, uređivanje i brisanje predmeta koji su dostupni na platformi (slika 28).

**Slika 26:** Administratorska analitika

**Slika 27:** Upravljanje korisnicima

Name	Created	Actions
Ekonomija	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Informatika	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Geografija	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Povijest	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Hrvatski	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Engleski	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Biofizika	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Kemijski	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Fizika	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>
Matematika	13 siječnja 2020.	<a href="#">Edit</a> <a href="#">Delete</a>

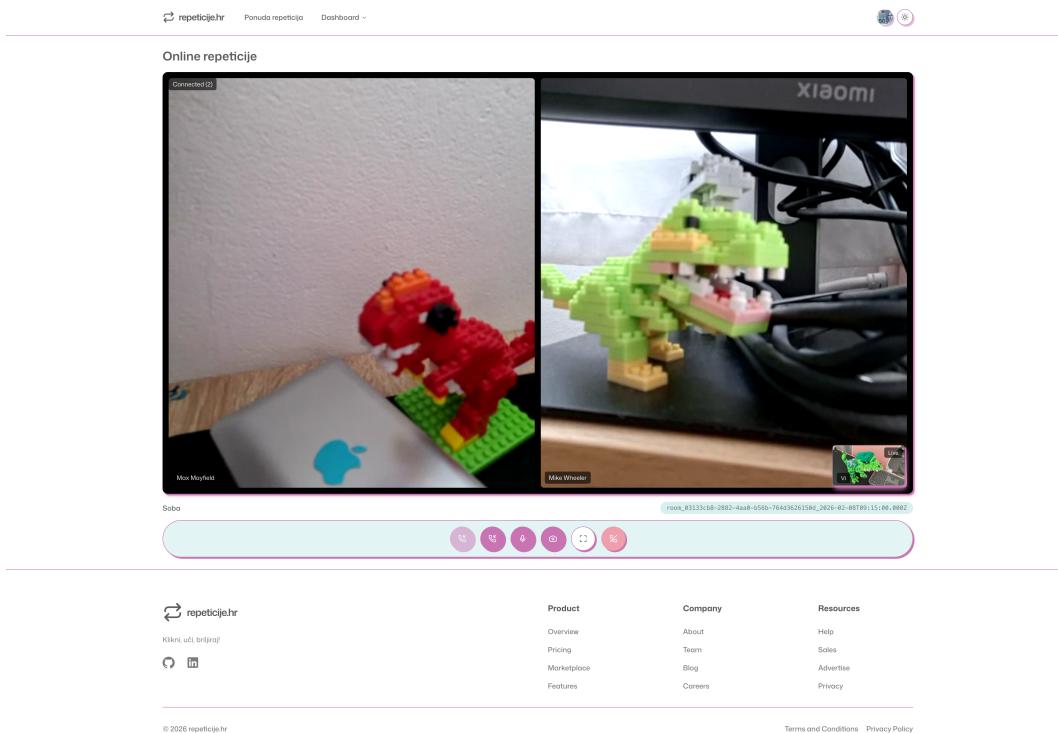
**Slika 28:** Upravljanje predmetima

Na slikama 29 i 30 prikazana je soba za sastanke za održavanje *online* repeticija pri audio-video pozivu između dvaju sudionika (jednog tutora i jednog studenta). Vidljivo je da je sučelje prilagođeno i desktop i mobilnim uređajima.

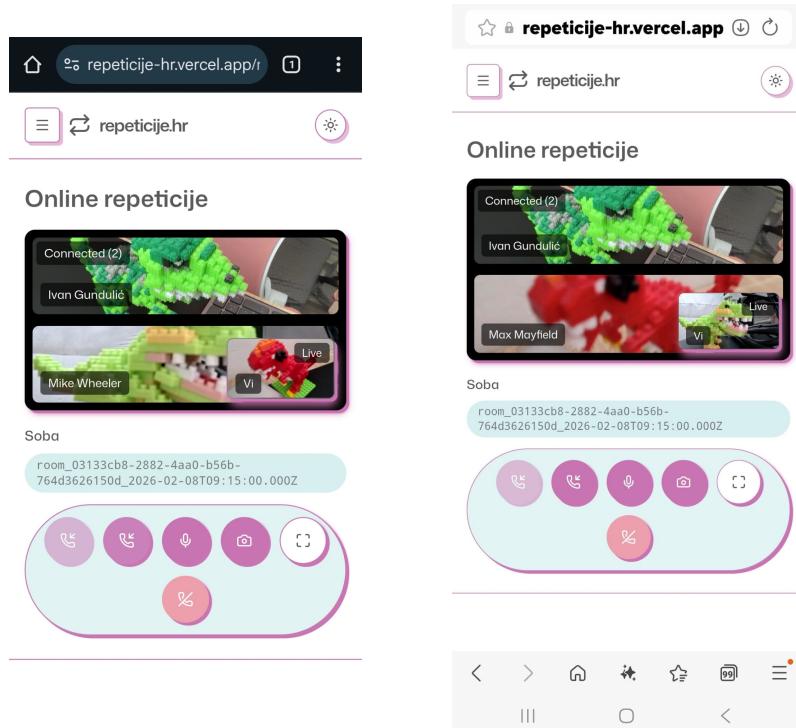
**Slika 29:** Soba za sastanke pri audio-video pozivu između dvaju sudionika (desktop, macOS, Chrome)

**Slika 30:** Soba za sastanke pri audio-video pozivu između dvaju sudionika (mobile, Android, Chrome)

Na slikama 31 – 33 prikazana je soba za sastanke pri audio-video pozivu između više sudionika (jedan tutor i dva studenta), što ilustrira funkcionalnost grupnih *online* repeticija kroz različite preglednike, operativne sustave, uređaje i mrežne uvjetne.



**Slika 31:** Soba za sastanke pri audio-video pozivu između više sudionika (*desktop, macOS, Chrome*)



**Slika 32:** Soba za sastanke pri audio-video pozivu između više sudionika (*mobile, Android, Chrome*)

**Slika 33:** Soba za sastanke pri audio-video pozivu između više sudionika (*mobile, Android, Samsung Internet*)

## 4. Zaključak

U ovom diplomskom radu predstavljen je razvoj web aplikacije za povezivanje studenta i tutora koja pokriva proces od formiranja ponuda i rezervacije termina do održavanja repeticija, uz implementiranu podršku za online video pozive. Sustav je podijeljen na prezentacijski, aplikacijski, podatkovni i komunikacijski sloj, pri čemu odabrane tehnologije omogućuju skalabilnu i modularnu arhitekturu.

Posebna pažnja posvećena je sigurnosti i pouzdanosti: implementirani su autentikacija i autorizacija s ulogama studenta, tutora i administratora, impersonacija radi lakšeg otklanjanja problema, zaštita API ruta preko *proxy* i *middleware* sloja, validacija vremenske zone te integracija Google Maps API-ja i automatiziranih *cron* zadataka. Uz to, podrška za online repeticije putem WebRTC-a, sustav obavijesti putem e-pošte i filtriranje ponuda prema lokaciji i dostupnosti tutora čine aplikaciju praktičnim alatom za organizaciju repeticija.

Odabrani dizajn omogućuje daljnje proširenje funkcionalnosti (primjerice integracija plaćanja, internacionalizacija i sl.), ali je u prvoj verziji sustava svjesno primijenjen YAGNI princip kako bi rješenje ostalo razumljivo, održivo i fokusirano na ključne potrebe korisnika na lokalnom, hrvatskom tržištu.

## Literatura

- [1] B. Jokić and Z. Ristić Dedić, “Privatne instrukcije u Republici Hrvatskoj: biznis iz sjene kojeg pandemija nije ugrozila,” <http://idiprints.knjiznica.idi.hr/id/eprint/1048> (posjećeno 30.1.2026.), Institut za društvena istraživanja, Zagreb, Project Report, 2022.
- [2] ——, “Organizirane pripreme za državnu maturu i prijemne ispite: obrazovni biznis koji u Hrvatskoj i dalje raste,” <http://idiprints.knjiznica.idi.hr/id/eprint/1046> (posjećeno 30.1.2026.), Institut za društvena istraživanja, Zagreb, Project Report, 2022.
- [3] <https://github.com/vercel/next.js> (posjećeno 30.1.2026.).
- [4] OpenJS Foundation, “About Node.js®,” <https://nodejs.org/en/about/> (posjećeno 30.1.2026.).
- [5] Microsoft, “TypeScript Docs,” <https://www.typescriptlang.org/docs/> (posjećeno 30.1.2026.).
- [6] npm, Inc., “npm Docs,” <https://docs.npmjs.com/> (posjećeno 30.1.2026.).
- [7] <https://github.com/facebook/react> (posjećeno 30.1.2026.).
- [8] Meta Platforms, Inc., “React,” <https://legacy.reactjs.org/> (posjećeno 30.1.2026.).
- [9] ——, “React Reference Overview,” <https://react.dev/reference/react> (posjećeno 30.1.2026.).
- [10] Vercel, Inc., “Next.js Docs,” <https://nextjs.org/docs> (posjećeno 30.1.2026.).
- [11] shadcn at Vercel, “shadcn/ui Docs,” <https://ui.shadcn.com/docs> (posjećeno 30.1.2026.).
- [12] Radix by WorkOS, “Radix UI - Primitives: Introduction,” <https://www.radix-ui.com/primitives/docs/overview/introduction> (posjećeno 30.1.2026.).
- [13] Tailwind Labs, Inc., “Tailwind CSS Docs - Core Concepts: Styling with Utility Classes,” <https://tailwindcss.com/docs/styling-with-utility-classes> (posjećeno 30.1.2026.).
- [14] The PostgreSQL Global Development Group, “PostgreSQL - About,” <https://www.postgresql.org/about/> (posjećeno 30.1.2026.).
- [15] Prisma Data, Inc., “Prisma Docs,” <https://www.prisma.io/docs/> (posjećeno 30.1.2026.).
- [16] Supabase Inc., “Supabase Docs,” <https://supabase.com/docs> (posjećeno 30.1.2026.).

- [17] Google for Developers, “WebRTC - Real-Time Communication for the Web,” <https://webrtc.org/> (posjećeno 30.1.2026.).
- [18] S. DuBois, C. Mogren, A. Zhukov, and webrtcforthechair.com team, *WebRTC for the Curious*. webrtcforthechair.com, 2020, <https://github.com/webrtc-for-the-chair/webrtc-for-the-chair> (posjećeno 30.1.2026.).
- [19] webrtcforthechair.com, <https://webrtcforthechair.com/docs/images/01-webrtc-agent.png> (posjećeno 30.1.2026.).
- [20] World Wide Web Consortium, “Web Real-Time Communications (WebRTC) transforms the communications landscape; becomes a World Wide Web Consortium (W3C) Recommendation and multiple Internet Engineering Task Force (IETF) standards,” *W3C Press Releases*, 2021, <https://www.w3.org/press-releases/2021/webrtc-rec/> (posjećeno 30.1.2026.).
- [21] webrtcforthechair.com, <https://webrtcforthechair.com/docs/images/08-one-to-one.png> (posjećeno 30.1.2026.).
- [22] ——, <https://webrtcforthechair.com/docs/images/08-full-mesh.png> (posjećeno 30.1.2026.).
- [23] ——, <https://webrtcforthechair.com/docs/images/08-hybrid-mesh.png> (posjećeno 30.1.2026.).
- [24] ——, <https://webrtcforthechair.com/docs/images/08-sfu.png> (posjećeno 30.1.2026.).
- [25] ——, <https://webrtcforthechair.com/docs/images/08-mcu.png> (posjećeno 30.1.2026.).
- [26] ——, <https://webrtcforthechair.com/docs/images/01-signaling-sequence-overview.png> (posjećeno 30.1.2026.).
- [27] ——, <https://webrtcforthechair.com/docs/images/03-connectivity-checks.png> (posjećeno 30.1.2026.).
- [28] Better Auth, “Better Auth Docs,” <https://www.better-auth.com/docs/introduction> (posjećeno 30.1.2026.).

[29] OpenJS Foundation, “React Google Maps,” <https://visgl.github.io/react-google-maps/> (posjećeno 30.1.2026.).

[30] Resend, “Resend Docs,” <https://resend.com/docs> (posjećeno 30.1.2026.).

[31] Vercel, Inc., “Vercel Documentation,” <https://vercel.com/docs> (posjećeno 30.1.2026.).

# Dodatci

## Popis slika

1	WebRTC kao kolekcija protokola [19] . . . . .	10
2	Topologija „1 na 1” [21] . . . . .	11
3	Topologija <i>full mesh</i> [22] . . . . .	11
4	Topologija <i>hybrid mesh</i> [23] . . . . .	12
5	Topologija SFU [24] . . . . .	12
6	Topologija MCU [25] . . . . .	12
7	Signalizacija [26] . . . . .	13
8	ICE kandidati i testiranje parova [27] . . . . .	15
9	Dijagram arhitekture aplikacije . . . . .	21
10	Dijagram relacijske baze podataka . . . . .	25
11	Stranice za prijavu i registraciju . . . . .	33
12	Primjer testne poruke e-pošte poslane preko Resend API-ja . . . . .	43
13	Stranica s oglasima različitih ponuda repeticija . . . . .	45
14	Javni profil tutora . . . . .	46
15	Kalendarski prikaz studentovih rezervacija . . . . .	46
16	Nadolazeće rezervacije studenta . . . . .	47
17	Prethodne rezervacije studenta . . . . .	47
18	Neostvarene rezervacije studenta . . . . .	48
19	Lokacija: <i>Online</i> . . . . .	48
20	Lokacija: <i>Onsite</i> (Domaćin) . . . . .	49
21	Lokacija: <i>Onsite</i> (Putujem) . . . . .	49
22	Upravljanje predmetima koje tutor predaje . . . . .	49
23	Popis i upravljanje ponudama za tutore . . . . .	50
24	Postavke dostupnosti za tutore . . . . .	50
25	Kalendarski prikaz tutorovih rezervacija i dostupnosti . . . . .	50
26	Administratorska analitika . . . . .	51
27	Upravljanje korisnicima . . . . .	51
28	Upravljanje predmetima . . . . .	52

29	Soba za sastanke pri audio-video pozivu između dvaju sudionika ( <i>desktop</i> , macOS, Chrome) . . . . .	52
30	Soba za sastanke pri audio-video pozivu između dvaju sudionika ( <i>mobile</i> , Android, Chrome) . . . . .	52
31	Soba za sastanke pri audio-video pozivu između više sudionika ( <i>desktop</i> , macOS, Chrome) . . . . .	53
32	Soba za sastanke pri audio-video pozivu između više sudionika ( <i>mobile</i> , Android, Chrome) . . . . .	53
33	Soba za sastanke pri audio-video pozivu između više sudionika ( <i>mobile</i> , Android, Samsung Internet) . . . . .	53

## **Popis tablica**

1	Pregled glavnih ruta korisničkog sučelja u App Routeru . . . . .	30
2	Pregled API krajeva implementiranih u direktoriju app/api/ . . . . .	30

## **Popis ispisa kôda**

1	Naredbe za instalaciju Prisma ORM-a . . . . .	22
2	Primjer definicije modela baze podataka u datoteci schema.prisma . . .	23
3	Primjer implementacije repozitorija za entitet Subject . . . . .	25
4	Primjer implementacije servisne funkcije za kreiranje ponude repeticija . .	27
5	Primjer implementacije API rute za dohvat svih predmeta . . . . .	29
6	Konfiguracija biblioteke Better Auth i Admin dodatka . . . . .	32
7	Definiranje uloga i dopuštenih akcija u datoteci permissions.ts . . . . .	34
8	Proxy sloj za zaštitu odabranih ruta . . . . .	36
9	Provjera autentikacije u middleware/auth.ts . . . . .	37
10	Provjera dozvola za izvođenje akcija . . . . .	37
11	Pribavljanje lokalnog audio i video toka . . . . .	41
12	Konfiguracija Vercel cron zadatka . . . . .	43