

Iva Bazo
Fakultet elektrotehnike i računarstva, Zagreb

**PROGRAMSKA POTPORA KOMUNIKACIJSKIM
SUSTAVIMA**

QuizStream

v1.0

U Zagrebu, ožujak 2025.

Sadržaj

1.	Uvod.....	3
2.	Korištene tehnologije i arhitektura.....	4
2.1.	HTTP (Hypertext Transfer Protocol).....	4
2.2.	REST API (Representational State Transfer)	4
2.3.	WebSocket	5
2.4.	STOMP (Simple Text Oriented Messaging Protocol)	5
2.5.	React i Typescript	6
2.6.	Spring Boot	6
2.7.	PostgreSQL	7
2.8.	Arhitektura sustava	8
3.	Opis funkcionalnosti	9
3.1.	Registracija	10
3.2.	Prijava	11
3.3.	Kreiranje novog kviza.....	13
3.4.	Uređivanje kviza	15
3.5.	Pregled kreiranih kvizova	17
3.6.	Pregled pojedinog kviza.....	19
3.7.	Brisanje kviza.....	20
3.8.	Povezivanje moderatora na WebSocket.....	21
3.9.	Povezivanje sudionika na WebSocket	25
3.10.	Emitiranje pitanja sudionicima	28
3.11.	Odgovaranje sudionika	30
3.12.	Prikaz krajnje statistike odgovaranja sudionicima.....	32
3.13.	Kraj emitiranja instance kviza	33
4.	Upute za pokretanje za OS Linux	35
4.1.	Baza podataka	35
4.2.	Poslužitelj.....	35
4.3.	Klijent	35

1. Uvod

Autor: Iva Bazo

GitHub repozitorij: <https://github.com/ivabazo7/QuizStream>

Web aplikacija QuizStream je real-time kviz platforma inspirirana web aplikacijom [Slido](#). Glavni cilj projekta bio je implementirati sustav koji omogućuje:

- Kreiranje kvizova s ABCD pitanjima
- Pridruživanje kvizu putem jedinstvenog koda
- Real-time odgovaranje na pitanja i prikaz statistike odgovora
- Upravljanje tokom kviza

Ključne tehnologije korištene u razvoju su:

- WebSocket (STOMP) za real-time komunikaciju
- REST API za CRUD operacije
- React s Typescript-om za frontend
- SpringBoot za backend
- PostgreSQL za relacijsku bazu podataka

2. Korištene tehnologije i arhitektura

Svi korišteni protokoli koriste TCP/IP kao transportni sloj. REST API se gradi na HTTP-u, a STOMP se koristi preko WebSocketeta. HTTP i REST API su bez stanja i zahtijevaju inicijaciju od klijentske strane, dok su WebSocket i STOMP sa stanjem, odnosno održavaju stalnu vezi i omogućuju publish/subscribe prijenos podataka.

Za klijentsko sučelje aplikacije (frontend) korišten je React i Typescript, za poslužiteljski dio korišten je Spring Boot, dok je za bazu podataka korišten PostgreSQL.

2.1. HTTP (Hypertext Transfer Protocol)

HTTP je komunikacijski protokol koji koristi klijent-poslužitelj model, gdje klijent šalje zahtjev, a poslužitelj vraća odgovor. Komunikacija klijenta i poslužitelja odvija se iniciranjem klijentske strane, odnosno poslužitelj nikada prvi ne šalje zahtjev klijentu. HTTP se koristi za standardne web stranice, odnosno za dohvaćanje i slanje podataka između klijentske i poslužiteljske aplikacije.

Ključne karakteristike HTTP protokola:

- Bez stanja (stateless) - zahtjevi su međusobno nezavisni i ne pohranjuju se na poslužitelju
- Jednosmjerna komunikacija – komunikacija se uvijek odvija od klijenta prema poslužitelju (koji samo vraća odgovor klijentu)
- Privremena veza – zatvara se nakon odgovora
- Metode:
 - GET – dohvat podataka
 - POST – pohrana podataka
 - PUT - ažuriranje podataka
 - DELETE – brisanje podataka
 - Itd.

2.2. REST API (Representational State Transfer)

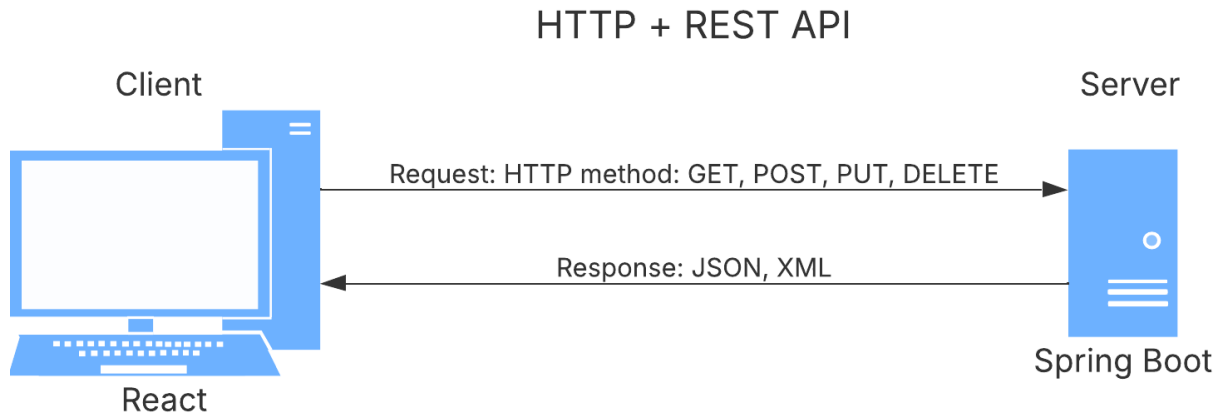
REST API je arhitekturni stil koji koristi HTTP protokol za upravljanje podacima. Radi sa resursima (URI, npr. /quiz /user) i HTTP metodama. REST API se koristi kod aplikacija kojima nije potrebno ništa više od jednostavne povremene komunikacije, odnosno za povremene CRUD operacije (Create, Read, Update, Delete).

Primjeri:

- GET /quiz - dohvaća sve kvizove
- GET /quiz/{quizId} - dohvaća jedan kviz sa odgovarajućim identifikatorom
- DELETE /quiz/{quizId} - briše kviz sa odgovarajućim identifikatorom
- PUT /quiz/{quizId} - ažurira vrijednosti kviza koje se predaju u tijelu zahtjeva
- POST /quiz - kreira kviz koji se predaje u tijelu zahtjeva, identifikator se ne navodi jer se dinamički kreira na poslužiteljskoj strani ili u bazi podataka

Karakteristike REST API arhitekturnog stila:

- Bez stanja (stateless), kao i HTTP
- Koristi HTTP metode (GET, POST, PUT, DELETE, ...) za obavljanje operacija
- Podaci se najčešće šalju u formatima kao što su JSON i XML, a manje kao HTML
- Privremena veza
- Jednosmjerna komunikacija



2.3. WebSocket

WebSocket je protokol za dvosmjernu komunikaciju preko jedne TCP veze, odnosno stalnog kanala između klijentske i poslužiteljske strane. Stalna dvosmjerna TCP veza razlog je zašto se ovaj protokol koristi kod stvarnovremenske komunikacije, kao što je chat, igre uživo, praćenje stanja na burzi i slično.

Karakteristike:

- Stalna veza
- Dvosmjerna komunikacija
- Trenutno slanje podataka, bez čekanja na zahtjev
- Efikasnije od HTTP polinga
- Sa stanjem (stateful)

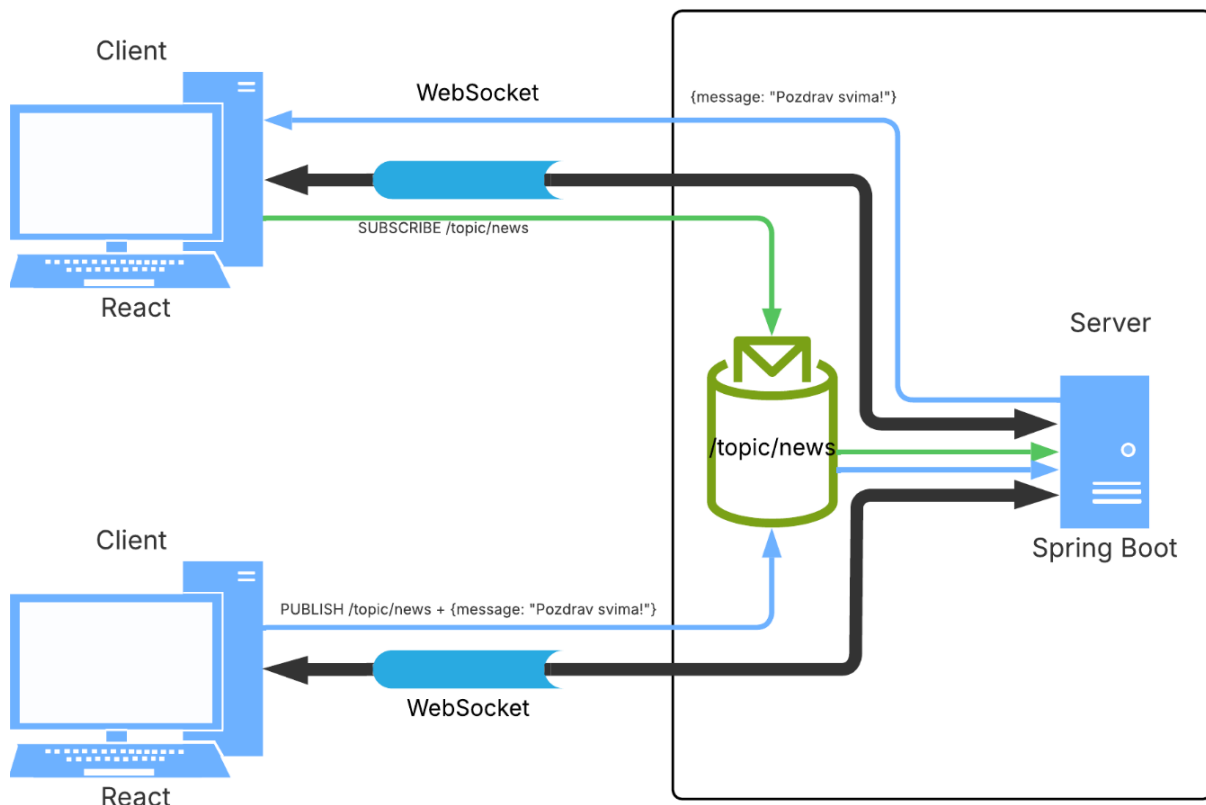
2.4. STOMP (Simple Text Oriented Messaging Protocol)

STOMP je tekstualni protokol za razmjenu poruka koji se često koristi u kombinaciji s WebSocket-om. Definira format i semantiku poruka (PUBLISH; SUBSCRIBE). Primjerice, klijent se sa SUBSCRIBE /topic/news pretplaćuje na taj kanal, te dolaskom nove vijesti na kanal (PUBLISH /topic/news), automatski prima tu vijest jer se prethodno pretplatilo.

Karakteristike:

- Stalna veza - koristi WebSocket, odnosno jednu TCP vezu koja je stalno otvorena
- Dvosmjerna komunikacija porukama, koje se šalju u obliku okvira na određeni kanal
- Podržava slanje poruke na kanal, te pretplatu na kanal
- Sa stanjem

WebSocket + STOMP



2.5. React i Typescript

React je JavaScript biblioteka za razvoj korisničkih sučelja koja je razvijena od strane Facebook-a. Olakšava izgradnju sučelja koristeći komponente i deklarativni pristup.

Karakteristike:

- Koristi Virtual DOM pomoću kojeg ažurira sučelje
- Komponente – ponovno iskoristivi dijelovi koda
- Jednosmjerni tok podataka – od komponente roditelja do komponente djeteta
- JSX sintaksa koja omogućuje pisanje HTML-a unutar JavaScripta

Uz React se često koristi Typescript, koji je nadogradnja JavaScripta koja dodaje statičko tipiziranje, te se na taj smanjuje broj pogreški koje nastanu zbog različitih tipova u podacima.

2.6. Spring Boot

Spring Boot je Java framework koji olakšava izgradnju aplikacija. Koristi se često za enterprise aplikacije, mikroservise, REST API itd.

Karakteristike:

- Automatska konfiguracija ovisnosti (baze podataka, sigurnosti, ...)
- Ugrađeni server - najčešće Tomcat
- Jednostavno dodavanje funkcionalnosti

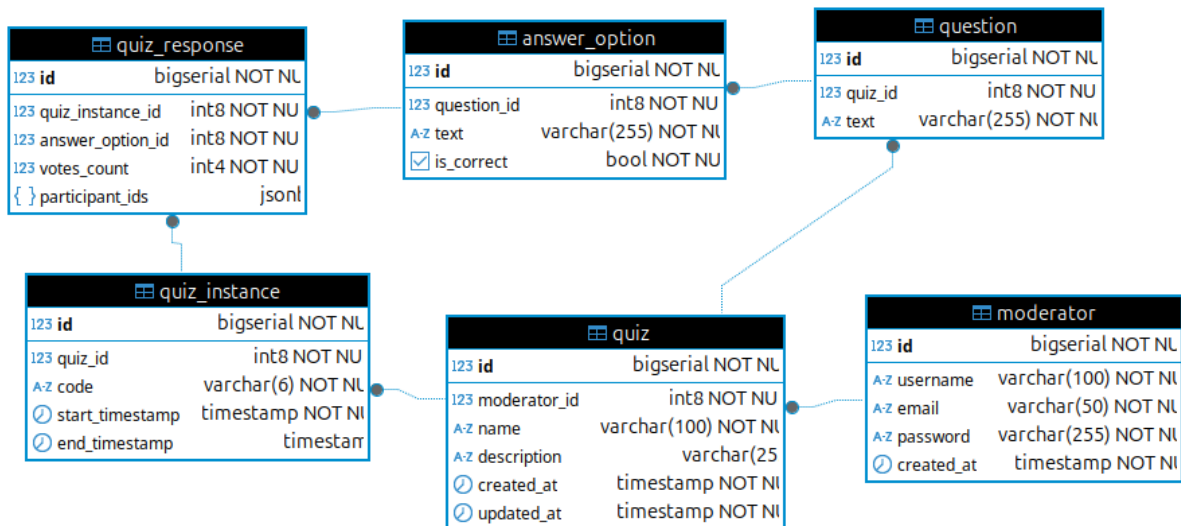
2.7. PostgreSQL

PostgreSQL je relacijski sustav za upravljanje bazama podataka.

Karakteristike:

- Osigurava ACID svojstva
- Open-source

ERD dijagram QuizStream baze podataka:



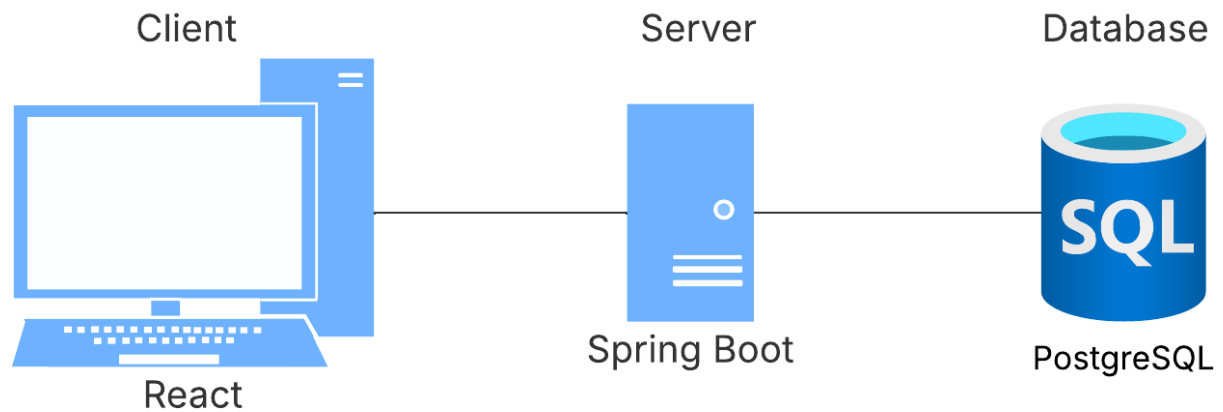
Entitet “*moderator*” može imati više kvizova “*quiz*”, a entitet “*quiz*” može imati više pitanja “*question*”, te više instanci kviza “*quiz_instance*”.

Entitet “*question*” može imati više odgovora “*answer_option*”, a entitet “*quiz_response*” se odnosi na neku instancu kviza “*quiz_instance*” i na neki odgovor “*answer_option*”, odnosno “*quiz_instance*” pohranjuje krajnju statistiku odgovaranja sudionika na nekoj instanci kviza na neki ponuđeni odgovor (koliko sudionika je glasalo za neki odgovor u toj instanci kviza).

2.8. Arhitektura sustava

Sustav se sastoji od 3 glavne komponente:

- Klijent - sučelje aplikacije, dio poslovne logike (React)
- Poslužitelj - poslovna logika, dohvat podataka (Spring Boot)
- Baza podataka – pohrana podataka (PostgreSQL)



3. Opis funkcionalnosti

U ovom poglavlju opisana je funkcionalnost aplikacije, te programski kod uz neke funkcionalnosti, dok uz neke nema programskog koda jer su pozivi klijentske strane i obrada na poslužitelju slični.

Korisnik aplikacije ima mogućnost registracije, te nakon što se registrira i prijavi u sustav, može upravljati svojim kvizovima. Upravljanje kvizovima uključuje kreiranje kviza, zajedno s kreiranjem pitanja od kojih se kviz sastoji, kreiranjem ponuđenih odgovora na pitanja, te označavanjem točnih odgovora na pojedino pitanje. Upravljanje kvizovima uključuje i mogućnost naknadnog uređivanja već postojećeg kviza, te brisanje i dohvrat svih kvizova koje je korisnik kreirao i dohvrat pojedinog kviza.

Navedene korisničke aktivnosti odvijaju se putem REST API sučelja, odnosno HTTP komunikacijom klijentske i poslužiteljske stanje koristeći HTTP metode GET, POST, PUT i DELETE, te prijenosom podataka u formatu JSON.

Nakon što korisnik kreira kviz, može uživo ispitati znanje sudionika ili prikupiti potrebne informacije. Takva interaktivna komunikacija odvija se putem WebSocket protokola uz korištenje STOMP-a. STOMP omogućuje pretplate i objave na određene kanale (teme). U slučaju QuizStream aplikacije, bitno je odvojiti pojedinačno emitiranje različitih kvizova od različitih korisnika, što se postiže kreiranjem jedinstvenog pristupnog koda instance kviza, te slanjem i preusmjeravanjem poruka na točno određeni kanal (instancu kviza) koristeći STOMP.

Napomena: kratica FE u daljnjem tekstu se odnosu na korisničko sučelje (en. frontend), a kratica BE se odnosni na poslužitelj (en. backend).

3.1. Registracija

- FE “/register” poziva BE “**POST /api/auth/register**” sa potrebnim u podacima u tijelu zahtjeva za kreiranje korisničkog računa, odnosno s unesenim podacima iz forme

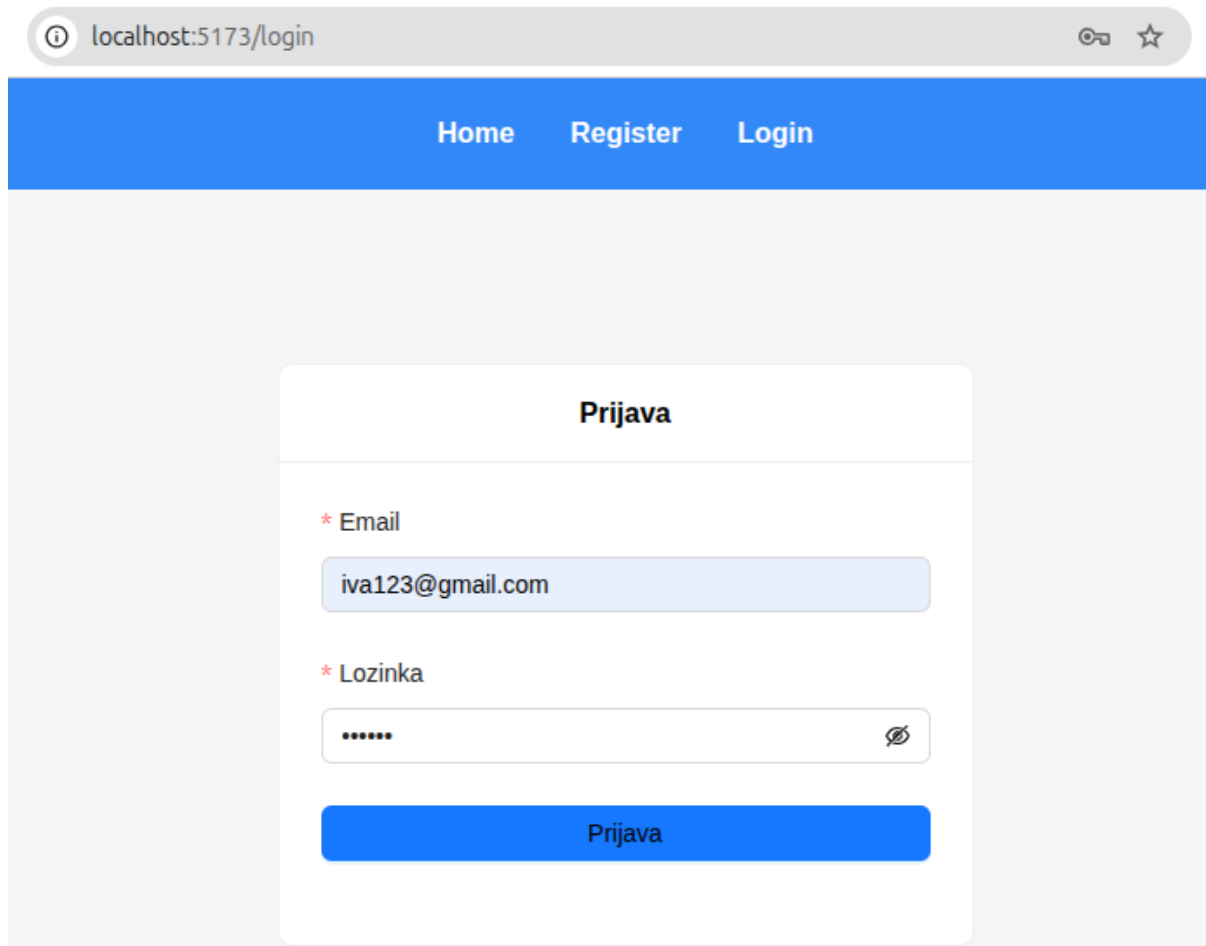
The screenshot shows a web browser window with the address bar displaying "localhost:5173/register". The page has a blue header with navigation links: "Home", "Register", and "Login". The main content area is light gray and contains a white registration form titled "Registracija". The form has three input fields, each preceded by a red asterisk indicating a required field:

- * Korisničko ime: Input field containing "iva123".
- * Email: Input field containing "iva123@gmail.com".
- * Lozinka: Input field containing masked characters "*****" and a toggle icon.

At the bottom of the form is a blue button labeled "Registriraj se".

3.2. Prijava

- FE “/login” poziva BE “**POST /api/auth/login**” sa potrebnim podacima za prijavu u tijelu zahtjeva, odnosno s unesenim podacima iz forme



The screenshot shows a web browser window with the address bar displaying 'localhost:5173/login'. The browser's address bar also shows a key icon and a star icon. The web page has a blue header with three links: 'Home', 'Register', and 'Login'. Below the header, there is a light gray background. In the center, there is a white card titled 'Prijava'. Inside the card, there are two input fields. The first field is labeled '* Email' and contains the text 'iva123@gmail.com'. The second field is labeled '* Lozinka' and contains a series of dots, indicating a password. To the right of the password field is a small icon of an eye with a slash through it, used for toggling password visibility. Below the input fields is a blue button labeled 'Prijava'.

Poziv klijentske strane:

```
const login = async (email: string, password: string) => {  
  const response = await api.post('/auth/login', { email, password });  
  const userData = {  
    id: response.data.id,  
    username: response.data.username,  
    email: response.data.email,  
  };  
  setUser(userData);  
  localStorage.setItem('user', JSON.stringify(userData));  
  message.success('Login successful!');  
  
  return userData;  
};
```

Obrada poslužiteljske strane:

```
@RestController
@RequestMapping("/api/auth")
public class AuthController {

    ...

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest request) {
        Optional<Moderator> optModerator =
        moderatorRepository.findByEmail(request.getEmail());
        if (optModerator.isEmpty()) {
            return ResponseEntity.badRequest().body("Pogreška prijave");
        }
        Moderator moderator = optModerator.get();

        if (!request.getPassword().equals(moderator.getPassword())) {
            return ResponseEntity.badRequest().body("Pogreška prijave.");
        }

        ModeratorDTO moderatorDTO = new ModeratorDTO(moderator);
        return ResponseEntity.ok(moderatorDTO);
    }

    ...
}
```

3.3. Kreiranje novog kviza

- Kreiranje novog kviza se pokreće na pritisak gumba “*Create Quiz*”
- Poziva se BE “***POST /quiz***” + kviz u tijelu zahtjeva u JSON formatu, odnosno uneseni podaci iz forme za kreiranje kviza
- Kviz može imati više pitanja, a svako pitanje može imati više ponuđenih odgovora od kojim je jedan ili više točno.

The top screenshot shows the web application interface at `localhost:5173/9/quiz`. It features a blue navigation bar with links for 'Home', 'My Quizzes', and 'Odjava'. Below the navigation bar, there is a large blue button labeled 'Create Quiz'.

The bottom screenshot shows the 'Start Quiz' form at `localhost:5173/9/quiz/12`. It includes a blue 'Start Quiz' button and a red 'Cancel' button. The form contains the following fields:

- Name:** A text input field containing 'Kviz općeg znanja za programere'.
- Description:** A text input field containing 'Kviz koji testira opće znanje programera o tehnologijama'.
- Question 1:** A modal window with the following content:
 - Question:** A text input field containing 'Što znači kratica HTML?'.
 - Answer Options:** A list of three options, each with a checkbox and a close button (X):
 - HighText Machine Language (checkbox unchecked)
 - HyperText Markup Language (checkbox checked)
 - HyperTool Multi Language (checkbox unchecked)
 - + Add Answer Option:** A dashed border button to add more options.

localhost:5173/9/quiz/12

+ Add Answer Option

Question 5

×

Question

Koji SQL upiti se koriste za manipulaciju podacima?

Answer Options

CREATE

☐

×

UPDATE

☒

×

DELETE

☒

×

INSERT

☒

×

REMOVE

☐

×

READ

☐

×

+ Add Answer Option

+ Add Question

Save

3.4. Uređivanje kviza

- Izgleda isto kao kreiranje, uz izmjenu poziva prema poslužiteljskoj strani.
- Poziva se BE “**PUT /quiz/{quizId}**”+ kviz u tijelu zahtjeva u JSON formatu

The image shows two screenshots of the QuizStream application. The top screenshot is a modal window titled "Question 3" for editing a quiz question. It contains a text input for the question, "Koja tehnologija se koristi za upravljanje verzijama koda?", and a list of answer options: Docker, NGINX, Git (selected), and Linux. Each option has a checkbox and a delete icon. The bottom screenshot shows the main quiz interface for "Kviz općeg znanja za programere". It displays a list of five questions, with the third question expanded to show the same answer options as in the modal. The interface includes a navigation bar with "Home", "My Quizzes", and "Odjava", and buttons for "Start Quiz" and "Edit".

localhost:5173/9/quiz/12

+ Add Answer Option

Question 3

Question

Koja tehnologija se koristi za upravljanje verzijama koda?

Answer Options

Docker

NGINX

Git

Linux

+ Add Answer Option

localhost:5173/9/quiz/12

Home My Quizzes Odjava

Start Quiz Edit

Kviz općeg znanja za programere

Last update: 6/1/2025, 1:14:18 PM

Kviz koji testira opće znanje programera o tehnologijama

Q1: Što znači kratica HTML?

Q2: Koji je operacijski sustav temeljen na Unixu i koristi se na većini web servera?

Q3: Koji tehnologija se koristi za upravljanje verzijama koda?

Docker

NGINX

Git

Linux

Q4: Što znači osnovni princip "DRY" u programiranju?

Q5: Koji SQL upiti se koriste za manipulaciju podacima?

Poziv klijentske strane za kreiranje ili uređivanje kviza:

```

const handleSave = async () => { try { const values = await form.validateFields();
  const payload: QuizEdit = {
    moderatorId: Number(moderatorId),
    name: values.name,
    description: values.description,
    questions: values.questions.map((q: QuestionEdit) => ({
      text: q.text,
      answerOptions: q.answerOptions.map((a: AnswerEdit) => ({
        text: a.text,
        correct: a.correct,
      })),
    })),
  });

  const url = existingQuiz ? `${API_BASE_URL}/quiz/${existingQuiz.id}` :
`${API_BASE_URL}/quiz`;

  const method = existingQuiz ? 'PUT' : 'POST';

  const response = await fetch(url, {
    method,
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload),
  });

  if (!response.ok) throw new Error('Failed to save quiz');

  const savedQuiz = await response.json();
  onQuizCreated(savedQuiz as Quiz);
} catch (error) {
  console.error('Validation failed:', error);
}
};

```

Obrada poslužiteljske strane za kreiranje ili uređivanje kviza:

```

@RestController
@RequestMapping("/quiz")
public class QuizController {

  ...

  @PostMapping
  public ResponseEntity<QuizDTO> saveQuiz(@RequestBody CreateQuizDTO createQuizDTO) {
    QuizDTO createdQuiz = quizService.createQuiz(createQuizDTO);
    if (createdQuiz == null) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
    }
    return ResponseEntity.status(HttpStatus.CREATED).body(createdQuiz);
  }

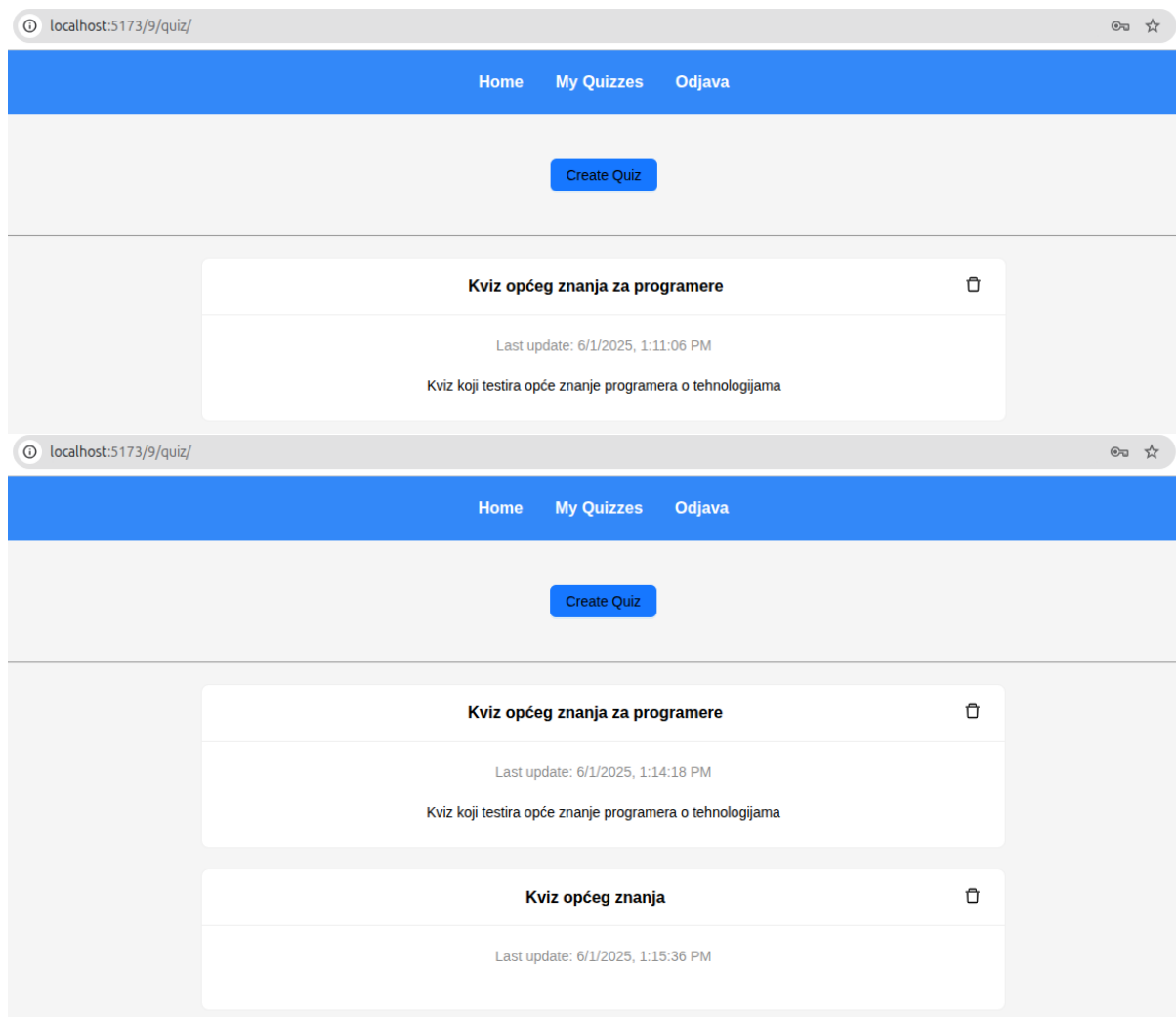
  @PutMapping("/{quizId}")
  public ResponseEntity<QuizDTO> updateQuiz(@PathVariable("quizId") Long quizId,
@RequestBody CreateQuizDTO createQuizDTO) {
    QuizDTO createdQuiz = quizService.updateQuiz(quizId, createQuizDTO);
    if (createdQuiz == null) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
    }
    return ResponseEntity.status(HttpStatus.CREATED).body(createdQuiz);
  }

  ...
}

```


3.5. Pregled kreiranih kvizova

- FE “`/moderatorId/quiz`” poziva BE “`GET /{moderatorId}/quiz`”



Poziv klijentske strane:

```
const fetchQuizzes = async () => {
  try {
    const response = await fetch(`${API_BASE_URL}/moderator/${moderatorId}/quiz`);
    if (!response.ok) throw new Error('Failed to fetch quizzes');
    const data = await response.json();
    setQuizzes(data);
  } catch (error) {
    console.error('Error fetching quizzes:', error);
  }
};
```

Obrada poslužiteljske strane za kreiranje ili uređivanje kviza:

```
@RestController
@RequestMapping("/moderator")
public class ModeratorController {

    @Autowired
    private ModeratorService moderatorService;

    @GetMapping("/{moderatorId}/quiz")
    public ResponseEntity<List<QuizDTO>> getAllQuizzes(@PathVariable("moderatorId") Long
moderatorId) {
        System.out.println("GET /moderator/{moderatorId}/quiz");
        List<QuizDTO> quizList = moderatorService.getAll(moderatorId);
        if (quizList == null) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
        }
        return ResponseEntity.status(HttpStatus.OK).body(quizList);
    }
}
```

3.6. Pregled pojedinog kviza

- FE “`{/moderatorId}/quiz/{quizId}`” poziva BE “`GET /quiz/{quizId}`”

The image displays two screenshots of a web application interface for a quiz titled "Kviz općeg znanja za programere". The browser address bar shows "localhost:5173/9/quiz/12". The application has a blue header with navigation links: "Home", "My Quizzes", and "Odjava". Below the header, there are two buttons: "Start Quiz" and "Edit".

The quiz content is displayed in a white box with a light gray border. It includes the title "Kviz općeg znanja za programere", the last update time "Last update: 6/1/2025, 1:11:06 PM", and a description "Kviz koji testira opće znanje programera o tehnologijama".

The quiz consists of five questions, each with a right arrow icon (>) on the left:

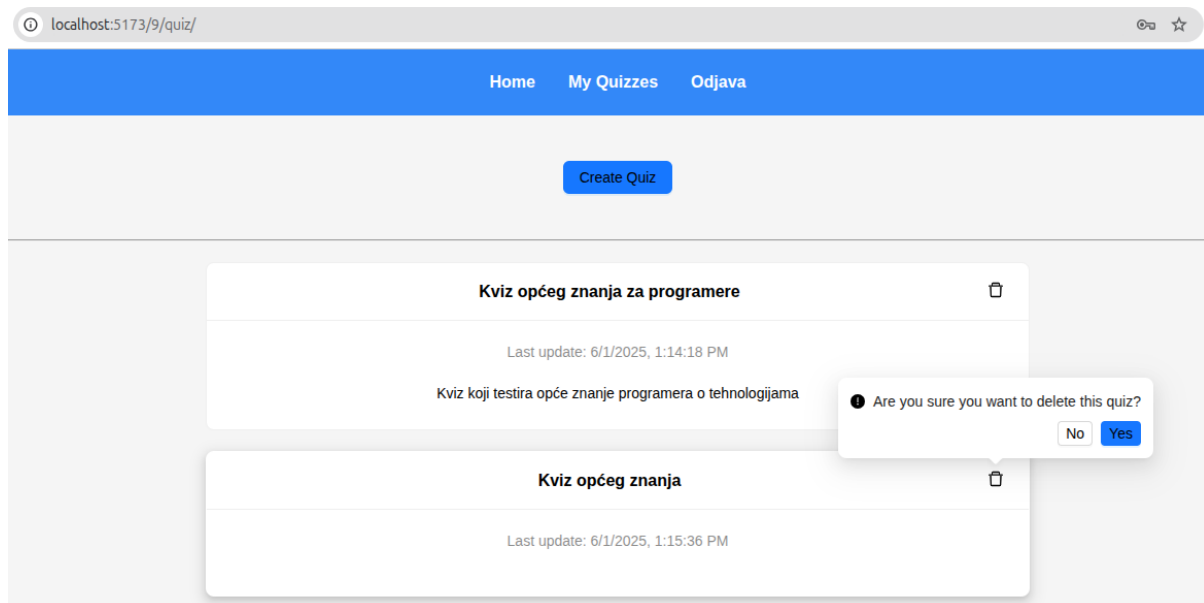
- Q1: Što znači kratica HTML?
- Q2: Koji je operacijski sustav temeljen na Unixu i koristi se na većini web servera?
- Q3: Koja tehnologija se koristi za upravljanje verzijama koda?
- Q4: Što znači osnovni princip "DRY" u programiranju?
- Q5: Koji SQL upiti se koriste za manipulaciju podacima?

The bottom screenshot shows the same quiz, but with the third question (Q3) selected, indicated by a downward arrow icon (v) on the left. The options for Q3 are displayed in a white box with a light gray border:

- Docker
- NGINX
- Git (highlighted in green)

3.7. Brisanje kviza

- Poziva se BE “***DELETE /quiz/{quizId}***”

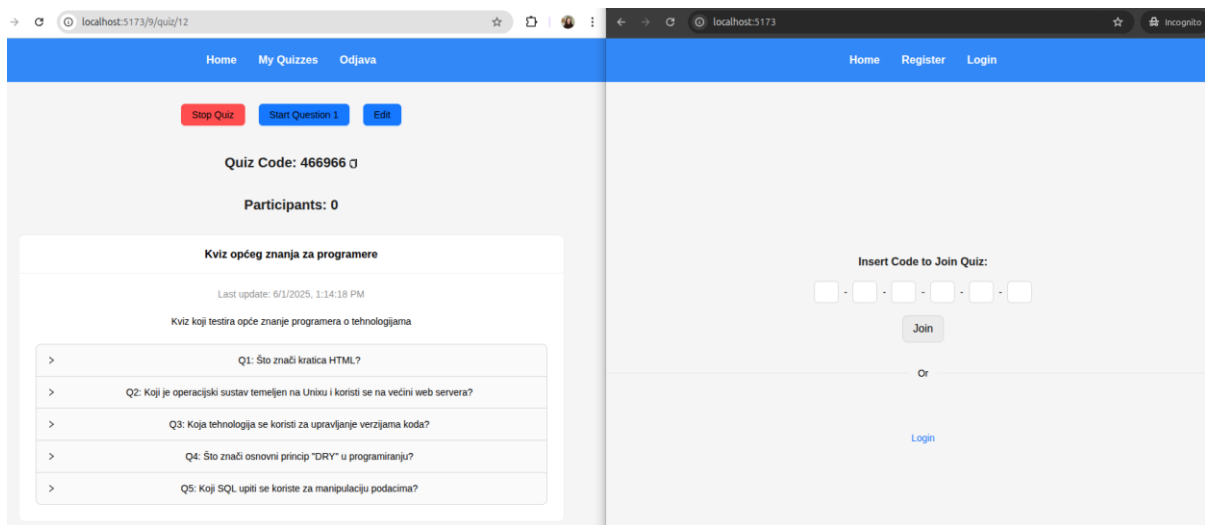


3.8. Povezivanje moderatora na WebSocket

Prijavljeni korisnik koji je kreirao kviz naziva se moderator.

Pri pokretanju kviza pritiskom na gumb “*Start Quiz*”, klijentska strana moderatora otvara WebSocket komunikaciju s poslužiteljskom stranom, poslužiteljska strana kreira jedinstveni pristupni kod za trenutno pokrenutu instancu kviza, šalje ga klijentskoj strani kako bi ga moderator mogao podijeliti sa sudionicima.

U narednim slikama na lijevoj strani je prikazana klijentska strana moderatora, a na desnoj klijentska strana sudionika.



Konfiguracija poslužiteljske strane:

- Endpoint za kreiranje veze je “*/stomp-endpoint*”
- Broker koji prima pretplate je “*/topic*”, odnosno sve pretplate (en. *subscribe*) moraju imati taj prefiks kako bi se osiguralo da dolaskom poruke na pojedini kanal pretplatnik dobije odgovor od brokera
- Sve objave (en. *publish*) moraju započeti prefiksom “*/app*”

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void registerStompEndpoints(final StompEndpointRegistry registry) {
        registry.addEndpoint("/stomp-endpoint") // u JS: var socket = new SockJS("/stomp-endpoint")
            .setAllowedOrigins("http://localhost:5173")
            .withSockJS();
    }

    @Override
    public void configureMessageBroker(final MessageBrokerRegistry registry) {
        registry.enableSimpleBroker("/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }
}
```

Koraci:

- Klijentska strana otvara vezu sa poslužiteljskom stranom na endpoint poslužitelja ***“/stomp-endpoint”*** jer je tako konfigurirano na poslužiteljskoj strani
- Prilikom spajanja, događa se i REST API poziv na BE ***“POST /quiz-instance/{quizId}/start”*** što vraća jedinstveni pristupni kod. Razlog za POST zahtjev je semantički, odnosno kreira se instanca kviza, iako se u tijelu zahtjeva ne šalje ništa.

Poslužiteljska strana za kreiranje instance kviza:

```
@RestController
@RequestMapping("/quiz-instance")
public class QuizInstanceController {

    ...

    @PostMapping("/{quizId}/start")
    public ResponseEntity<QuizCodeResponse> startQuiz(@PathVariable Long quizId) {
        QuizInstance instance = quizInstanceService.createInstance(quizId);
        return ResponseEntity.ok(new QuizCodeResponse(instance.getCode()));
    }
    ...
}

@Service
public class QuizInstanceService {

    ...

    public QuizInstance createInstance(Long quizId) {
        Quiz quiz = quizRepository.findById(quizId)
            .orElseThrow(() -> new RuntimeException("Quiz not found"));

        String code = generateUniqueCode();

        QuizInstance instance = new QuizInstance();
        instance.setQuiz(quiz);
        instance.setCode(code);
        instance.setStartTimestamp(LocalDateTime.now());

        return quizInstanceRepository.save(instance);
    }

    ...
}
```

- Klijentska strana moderatora se pretplaćuje na ***“SUBSCRIBE /topic/quiz/{quizCode}/moderatorState”***, te se tom pretplatom prati stanje trenutne instance kviza, odnosno odvija se:
 - praćenje indeksa trenutnog pitanja
 - praćenje broja spojenih sudionika na instancu kviza s kodom ***“{quizCode}”***
 - praćenje statistike odgovaranja sudionika na trenutno pitanje koje se emitira
 - praćenje zastavice da li treba spojenim sudionicima prikazati rezultate ili ne
- Klijentska strana moderatora objavljuje poruku da odmah želi dobiti trenutno stanje, kako se ne bi čekala akcija samog moderatora ili nekog od sudionika, odnosno odvija se objava na ***“PUBLISH /topic/quiz/{quizCode}/getModeratorState”***.

- Dohvaćanje stanja na poslužitelju:
 - Postavlja se stanje koje se treba vratiti klijentskoj strani, te se isto šalje na pretplatu koju je moderator prethodno napravio

```

    @PostMapping("/quiz/{quizCode}/getModeratorState")
    public void getModeratorState(@DestinationVariable String quizCode, @Payload
    String empty) {
        quizStateService.sendModeratorState(quizCode);
    }

    ...

    public void sendModeratorState(String quizCode) {
        ModeratorStateDTO moderatorStateDTO = new ModeratorStateDTO();
        moderatorStateDTO.setCurrentQuestionIndex(getCurrentQuestionIndex(quizCode));
        moderatorStateDTO.setParticipantCount(getParticipantCount(quizCode));
        moderatorStateDTO.setShowResults(getShowResults(quizCode));

        QuestionDTO currentQuestion = getCurrentQuestion(quizCode);
        if (currentQuestion != null) {
            // Postavlja inicijalnu statistiku glasanja
            List<AnswerCorrectResultDTO> results = getVotingStats(quizCode,
            currentQuestion.getId().toString());
            setResultsStat(quizCode, currentQuestion.getId().toString(), results);
            moderatorStateDTO.setResultsStat(results);
        } else {
            moderatorStateDTO.setResultsStat(null);
        }
        messagingTemplate.convertAndSend("/topic/quiz/" + quizCode +
        "/moderatorState", moderatorStateDTO);
    }

```

Otvaranje veze na klijentskoj strani moderatora:

```

function QuizDetailsPage() {
    const [participantCount, setParticipantCount] = useState<number>(0);
    const [currentQuestionIndex, setCurrentQuestionIndex] = useState(0);
    const [showResults, setShowResults] = useState(false);
    const [resultsStat, setResultsStat] = useState<AnswerResultStat[] | null>(null);

    const stompClientRef = useRef<Client | null>(null);

    const handleStateMessage = (message: any) => {
        const state: ModeratorState = JSON.parse(message.body);
        if (state.participantCount !== participantCount) {
            setParticipantCount(state.participantCount);
        }
        if (state.currentQuestionIndex !== currentQuestionIndex) {
            setCurrentQuestionIndex(state.currentQuestionIndex);
        }
        setResultsStat(state.resultsStat); // AnswerResultStat[]
        if (state.showResults !== showResults) {
            setShowResults(state.showResults);
        }
    };

    const connectToWebSocket = (isRefresh: boolean) => {
        const storedCode = localStorage.getItem('quizCode');
        if (isRefresh && !storedCode) {
            // ako se dogodio refresh, a nema pohranjenog koda, ne spajaj se
            return;
        }
        // inače se spoji ili sa pohranjenim kodom ako postoji ili kreiraj kod
        const socket = new SockJS(`${API_BASE_URL}/stomp-endpoint`);
        const client = new Client({
            websocketFactory: () => socket,
            reconnectDelay: 5000,
            onConnect: async () => {
                console.log('Moderator connected to WebSocket');
                try {
                    let code;
                    if (storedCode) {

```

```
    setQuizCode(storedCode);
    code = storedCode;
  } else {
    const response = await fetch(`${API_BASE_URL}/quiz-instance/${quizId}/start`, {
      method: 'POST',
    });
    const data = await response.json();
    setQuizCode(data.quizCode);
    localStorage.setItem('quizCode', data.quizCode);
    code = data.quizCode;
  }

  // Pretplata na stanje
  client.subscribe(`/topic/quiz/${code}/moderatorState`, handleStateMessage);
  // Dohvat stanja
  client.publish({
    destination: `/app/quiz/${code}/getModeratorState`, // trigerira moderatorState
    body: JSON.stringify({}),
  });
} catch (error) {
  console.error('Failed to start quiz instance:', error);
}
},
onStompError: frame => {
  console.error('STOMP protocol error:', frame.headers.message);
},
});

client.activate();
stompClientRef.current = client;

return () => {
  if (client.connected) {
    client.deactivate();
  }
};
};

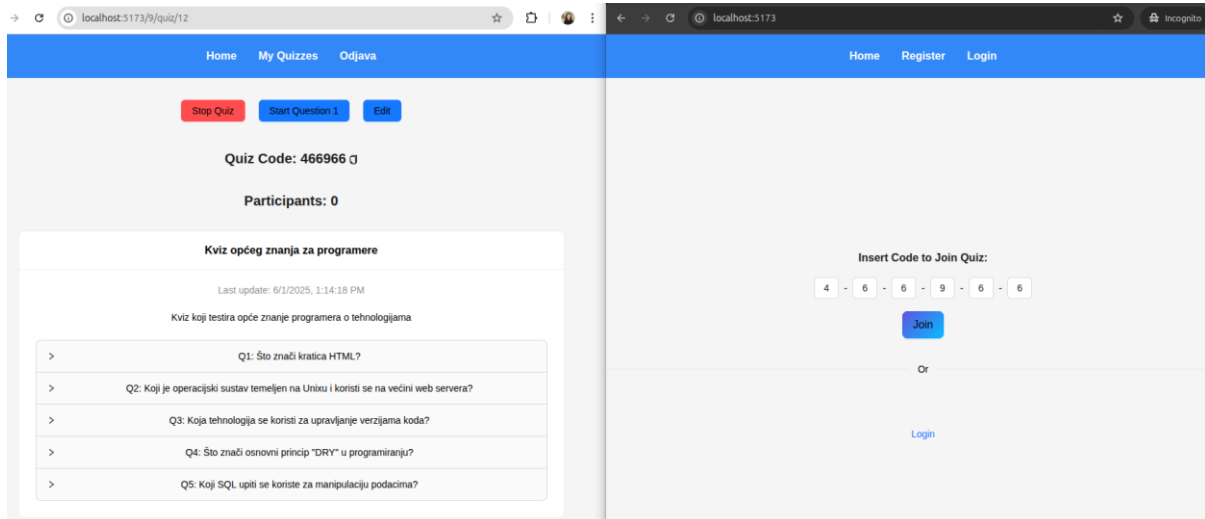
useEffect(() => {
  // ponovno spajanje na WebSocket ako se dogodi refresh stranice
  connectToWebSocket(true);
}, []);

return (
  <div style={{ padding: '2rem' }}>
...
    <Button
      type="primary"
      onClick={() => connectToWebSocket(false)}
      disabled={!quiz?.questions || quiz.questions.length === 0}
    >
      Start Quiz
    </Button>
...
  </div>
);
}

export default QuizDetailsPage;
```


3.9. Povezivanje sudionika na WebSocket

Klijentska strana sudionika otvara WebSocket vezu sa poslužiteljskom stranom kada sudionik unese kod instance kviza koju dobije od moderatora i pritisne gumb “Join”.



Otvaranje veze na klijentskoj strani sudionika slično je kao i kod klijentske strane moderatora, osim što se u ovom slučaju ne događa REST API poziv, te se događaju različite pretplate i objave:

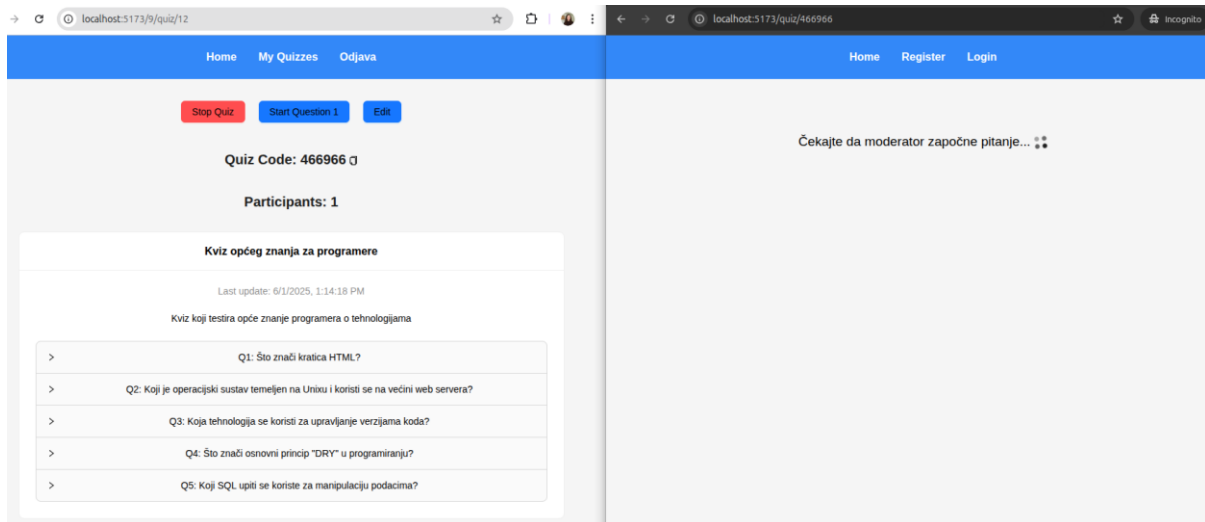
- Klijentska strana sudionika se pretplaćuje na **“SUBSCRIBE /topic/quiz/{quizCode}/participantState”**, te se tom pretplatom prati stanje trenutne instance kviza, odnosno odvija se:
 - praćenje trenutnog pitanja (ne indeksa kao kod moderatora, već pitanja s ponuđenim odgovorima)
 - praćenje krajnje statistike odgovaranja na trenutno pitanje koje se emitira
 - praćenje zastavice da li treba prikazati statistiku ili ne
- Klijentska strana sudionika se pretplaćuje na **“SUBSCRIBE /topic/quiz/{quizCode}/end”**, te se time prati da li je došlo do kraja emitiranja kviza (da li je moderator pritisnuo “End Quiz”), kako bi se poduzele potrebne akcije

```
...
// Pretplata na kraj kviza
client.subscribe(`/topic/quiz/${quizCode}/end`, _msg => {
  if (stompClientRef.current) {
    stompClientRef.current.deactivate();
    console.log('WebSocket disconnected.');

```

- Klijentska strana sudionika objavljuje poruku da odmah želi dobiti trenutno stanje, kako se ne bi čekala akcija samog moderatora, odnosno odvija se objava na **“PUBLISH /topic/quiz/{quizCode}/getParticipantState”**.

U trenutku kada sudionik pristupi kvizu pomoću koda, moderatoru se ažurira stanje, odnosno ažurira se broj spojenih sudionika:



Za to je na poslužiteljskoj strani napravljen posebni event listener koji osluškuje spajanje i odspajanje moderatora i sudionika (prati se pretplata na stanja), koji na promjenu ažurira stanje i šalje ažurirano stanje moderatoru (jer se prethodno pretplatio):

```
@Slf4j
@Component
public class QuizWebSocketEventListener {

    @Autowired
    private QuizStateService quizStateService;

    private final Map<String, Set<String>> participantSessions = new ConcurrentHashMap<>();
    private final Map<String, String> moderatorSessions = new ConcurrentHashMap<>();

    @EventListener
    public void handleSessionSubscribe(SessionSubscribeEvent event) {
        SimpMessageHeaderAccessor headers =
            SimpMessageHeaderAccessor.wrap(event.getMessage());
        String sessionId = headers.getSessionId();
        String destination = headers.getDestination();

        // /topic/quiz/{quizCode}/
        if (destination != null) {
            String quizCode = extractQuizCode(destination);
            if (destination.contains("/moderatorState")) {
                // obrada moderatora
                moderatorSessions.put(sessionId, quizCode);
                log.info("Moderator {} subscribed to quiz {}", sessionId, quizCode);
                // slanje trenutnog broja sudionika
                sendParticipantCount(quizCode);
            } else if (destination.contains("/participantState")) {
                // obrada sudionika
                participantSessions.computeIfAbsent(quizCode, k ->
                    ConcurrentHashMap.newKeySet()).add(sessionId);
                log.info("Participant {} subscribed to quiz {}", sessionId, quizCode);
                // slanje trenutnog broja sudionika
                sendParticipantCount(quizCode);
            }
        }
    }

    ...

    private void sendParticipantCount(String quizCode) {
        int count = participantSessions.getOrDefault(quizCode, Collections.emptySet()).size();
        quizStateService.updateParticipantCount(quizCode, count);
    }
}
```

```
...
}

...
@Service
public class QuizStateService {

    ...

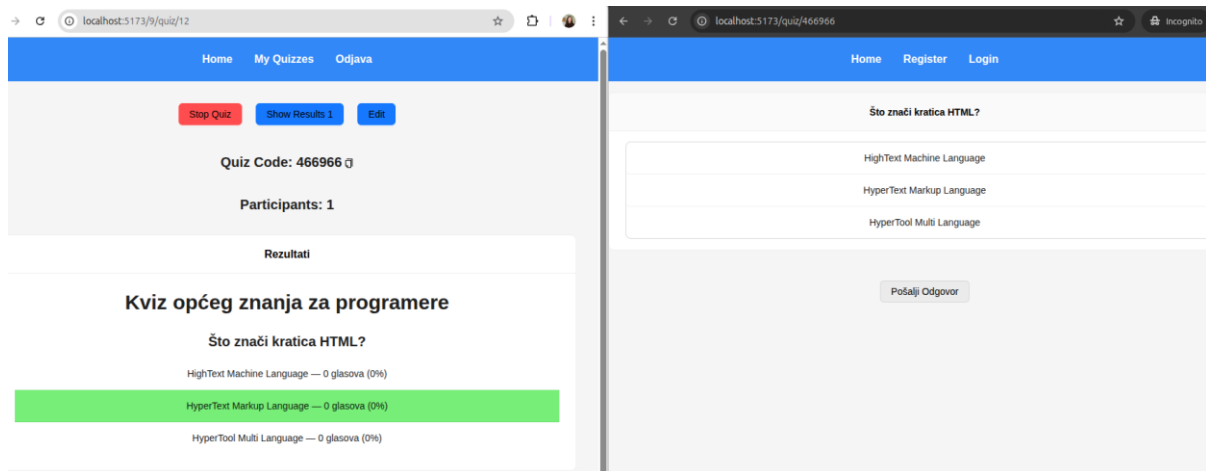
    private final Map<String, Integer> currentQuestionIndexMap = new ConcurrentHashMap<>(); //
    Map<quizCode, currentQuestionIndex>
    private final Map<String, QuestionDTO> currentQuestionMap = new ConcurrentHashMap<>(); //
    Map<quizCode, currentQuestion>
    private final Map<String, Boolean> showResultsMap = new ConcurrentHashMap<>(); //
    Map<quizCode, showResults>
    private final Map<String, Map<String, List<AnswerCorrectResultDTO>>> resultsStatMap = new
    ConcurrentHashMap<>(); // Map<quizCode, Map<questionId, results>>
    private final Map<String, Integer> participantCountMap = new ConcurrentHashMap<>(); //
    Map<quizCode, count>

    public void updateParticipantCount(String quizCode, int count) {
        participantCountMap.put(quizCode, count);
        sendModeratorState(quizCode);
    }

    ...
}
```

3.10. Emitiranje pitanja sudionicima

Moderator pritiskom na gumb “*Start Question I*” pokreće emitiranje pitanja sudionicima koji su spojeni na instancu kviza s odgovarajućim kodom. U istom trenutku sudionici na svom sučelju vide pitanje s ponuđenim odgovorima, a moderator vidi statistiku glasanja sudionika uživo.



Sudionik dobije pitanje jer se prethodno pretplatio na stanje, a stanje prati i koje je trenutno pitanje koje se emitira. Pitanje koje se trenutno emitira određuje moderator, odnosno pritiskom na gumb “*Start Question I*” pokreće se emitiranje prvog pitanja objavom istog pitanja (“**PUBLISH** /app/quiz/{quizCode}/question”).

Klijentska strana moderatora:

```
...
const startQuestion = () => {
...
  // Objava novog pitanja
  stompClientRef.current.publish({
    destination: `/app/quiz/${quizCode}/question`,
    body: JSON.stringify(question),
  });

  setCurrentQuestionIndex(prev => prev + 1);
  setShowResults(false);
  setResultsStat([]);
};
...
```

Poslužiteljska strana prima tu objavu, obrađuje novo stanje, te isto šalje moderatoru i sudionicima:

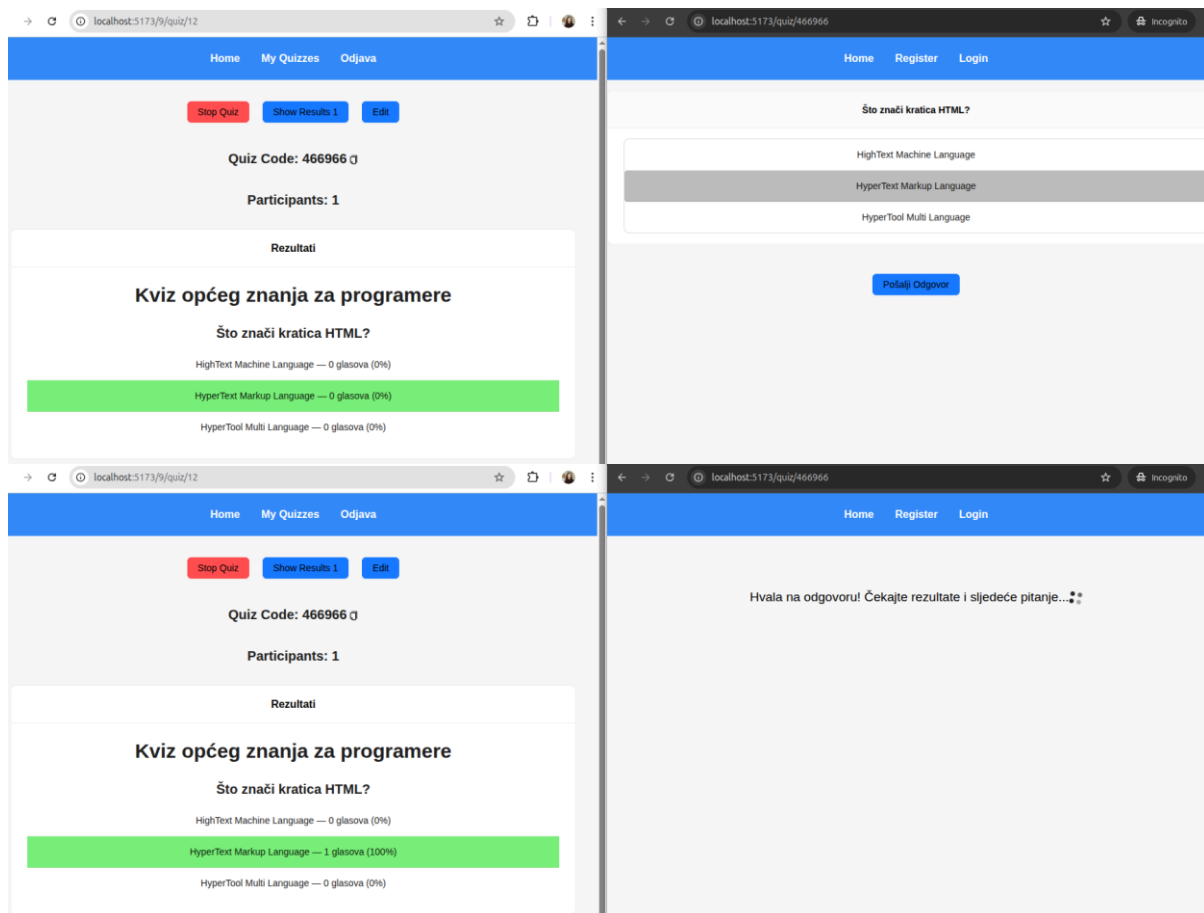
```
@Controller
public class QuizWebSocketController {
...
  @PostMapping("/quiz/{quizCode}/question")
  public void sendQuestion(@DestinationVariable String quizCode, @Payload QuestionDTO
question) {

    // Ažuriraj trenutno stanje
    int currentIndex = quizStateService.getCurrentQuestionIndex(quizCode);
    quizStateService.setCurrentQuestionIndex(quizCode, currentIndex + 1);
    quizStateService.setCurrentQuestion(quizCode, question);
    List<AnswerCorrectResultDTO> results = quizStateService.getVotingStats(quizCode,
quizStateService.getCurrentQuestionIndex(quizCode).toString());
    quizStateService.setResultsStat(quizCode,
quizStateService.getCurrentQuestionIndex(quizCode).toString(), results);
    quizStateService.setShowResults(quizCode, false);

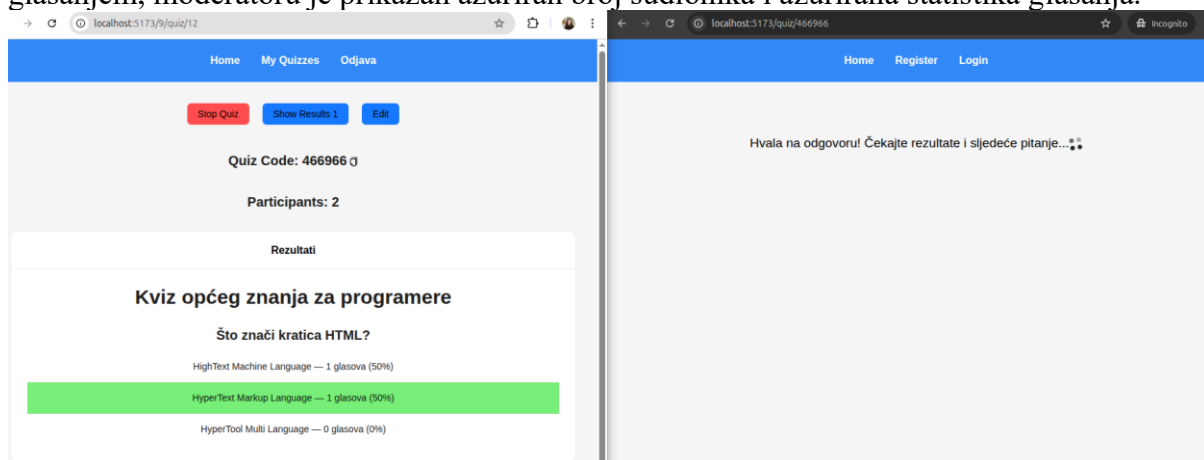
    // Pošalji stanje sudionicima i moderatoru
    quizStateService.sendParticipantState(quizCode);
    quizStateService.sendModeratorState(quizCode);
  }
...
}
```

3.11. Odgovaranje sudionika

Sudionik na svom sučelju bira odgovore za koje misli da su točni, te pritiskom na gumb “*Pošalji odgovor*” (trebalo je biti na engleskom jeziku, kao i neke druge poruke, isprike), objavljuje svoj odgovor, te se tom objavom moderatoru šalje ažurirano stanje, u kojem je ažurirana statistika glasanja.



Otvaranjem još jedne kartice u pregledniku, spajanjem na instancu kviza pomoću koda, te glasanjem, moderatoru je prikazan ažuriran broj sudionika i ažurirana statistika glasanja.



Klijentska strana sudionika:

- Radi “**PUBLISH** /app/quiz/{quizCode}/answer” - objava odgovora
- Pohranjuje u local storage informaciju da je na to pitanje već dan odgovor, kako ne bi prilikom osvježavanja stranice došlo do ponovnog odgovaranja na isto pitanje

```
...
const sendAnswer = () => {
  if (!participantAnswer || !stompClientRef.current?.connected) return;

  // Slanje odgovora
  stompClientRef.current.publish({
    destination: `/app/quiz/${quizCode}/answer`,
    body: JSON.stringify(participantAnswer),
  });

  // Spremi stanje glasanja u lokalnu pohranu
  const hasVotedKey = `quiz_${quizCode}_question_${question?.id}`;
  localStorage.setItem(hasVotedKey, 'true');

  setHasAnswered(true); // Spriječi ponovni odgovor
  message.success('Hvala na odgovoru! Čekajte sljedeće pitanje...');
};
...
```

Poslužiteljska strana obrađuje pristiglo pitanje, odnosno mijenja stanje statistike glasanja i ažurirano stanje šalje moderatoru, a sudionicima samo ako je stanje takvo da sudionicima treba biti prikazana statistika odgovaranja:

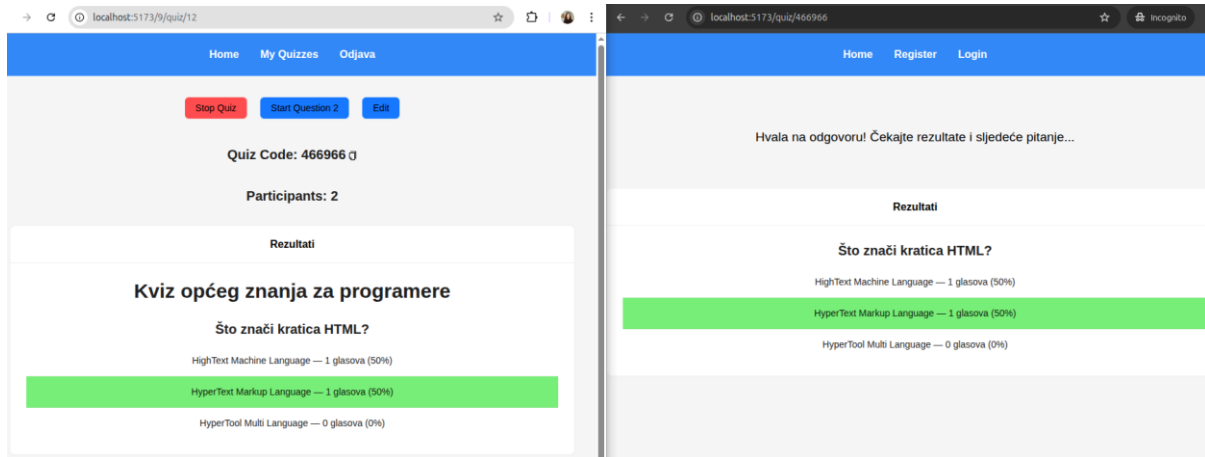
```
@Controller
public class QuizWebSocketController {
  ...
  @MessageMapping("/quiz/{quizCode}/answer")
  public void receiveAnswer(@DestinationVariable String quizCode, @Payload
  ParticipantAnswersDTO answer) {
    // Pohrani odgovor lokalno
    quizStateService.saveAnswer(quizCode, answer);

    // Ažuriraj stanje
    String currentQuestionId =
    quizStateService.getCurrentQuestion(quizCode).getId().toString();
    List<AnswerCorrectResultDTO> results = quizStateService.getVotingStats(quizCode,
    currentQuestionId);
    quizStateService.setResultsStat(quizCode, currentQuestionId, results);

    // Pošalji stanje moderatoru, a sudionicima samo ako treba prikazati rezultate
    quizStateService.sendModeratorState(quizCode);
    if (quizStateService.getShowResults(quizCode)) {
      quizStateService.sendParticipantState(quizCode);
    }
  }
  ...
}
```

3.12. Prikaz krajnje statistike odgovaranja sudionicima

Moderator pritiskom na gumb “*Show Results I*” označava da spojenim sudionicima treba prikazati krajnju statistiku odgovaranja na trenutno pitanje.



Klijentska strana moderatora u tom trenutku radi objavu “**PUBLISH** /app/quiz/{quizCode}/showResults” u kojoj se na poslužiteljskoj strani postavlja zastavica za prikaz rezultata sudionicima, te se sudionicima šalje ažurirano stanje.

```
...
const showVotingResults = () => {
...
  // Objava da se rezultati trebaju prikazati
  stompClientRef.current.publish({
    destination: `/app/quiz/${quizCode}/showResults`,
    body: JSON.stringify({}),
  });

  setShowResults(true);
};
...
```

Poslužiteljska strana:

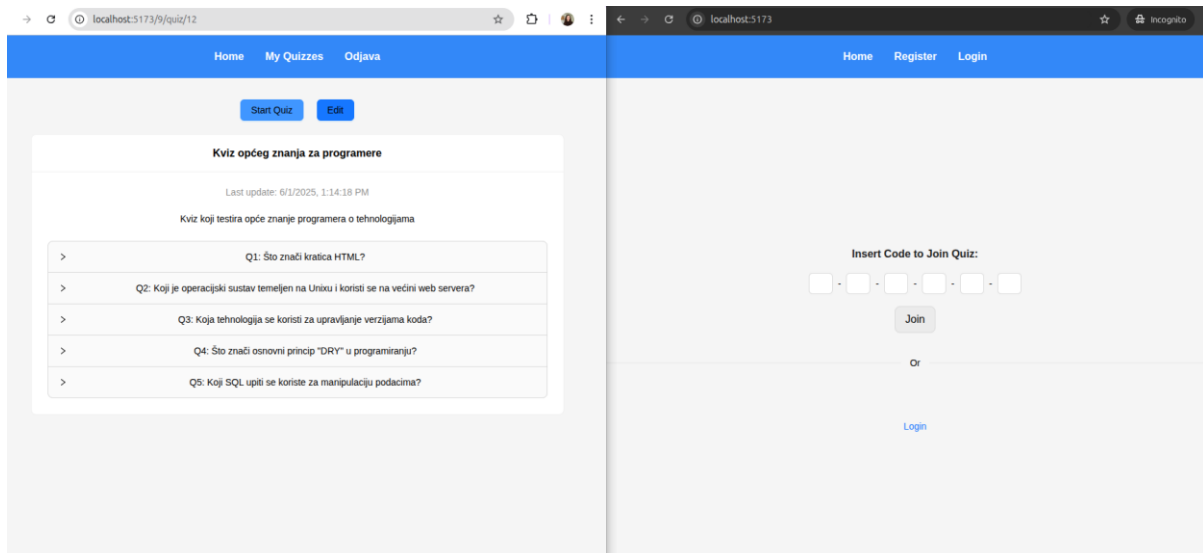
```
@Controller
public class QuizWebSocketController {
...
  @MessageMapping("/quiz/{quizCode}/showResults")
  public void sendShowResults(@DestinationVariable String quizCode, @Payload String empty) {

    // Ažuriraj trenutno stanje
    quizStateService.setShowResults(quizCode, true);

    // Pošalji stanje sudionicima
    quizStateService.sendParticipantState(quizCode);
  }
...
}
```


3.13. Kraj emitiranja instance kviza

Moderator pritiskom na “*Stop Quiz*” označava da se emitiranje instance kviza treba zaustaviti, odnosno radi “**PUBLISH** /app/quiz/\${quizCode}/end”, na koji se prekida WebSocket veza moderator-poslužitelj i sudionik-poslužitelj, te se briše stanje za tu instancu kviza. Sudionik prekida vezu jer se prethodno pretplatio na praćenje završetka kviza.



Klijentska strana moderatora:

```
const disconnectFromWebSocket = async () => {
  if (!quizCode) return;

  // Signal da se participants ugase
  if (stompClientRef.current?.connected) {
    stompClientRef.current.publish({
      destination: `/app/quiz/${quizCode}/end`,
      body: JSON.stringify({}),
    });
  }

  try {
    // Označi instancu kviza neaktivnom
    const response = await fetch(`${API_BASE_URL}/quiz-instance/${quizCode}/end`, {
      method: 'PUT',
    });

    if (!response.ok) throw new Error('Failed to end quiz instance.');
```

```
    setQuizCode(null);
    setParticipantCount(0);
    localStorage.removeItem('quizCode');
    setCurrentQuestionIndex(0);
    setResultsStat([]);

    if (stompClientRef.current) {
      stompClientRef.current.deactivate();
      console.log('WebSocket disconnected.');
```

```
    }
  } catch (error) {
    console.error('Error ending quiz instance:', error);
  }
};
```

Poslužiteljska strana:

```

@Controller
public class QuizWebSocketController {
    ...
    @MessageMapping("/quiz/{quizCode}/end")
    public void endQuiz(@DestinationVariable String quizCode, @Payload String empty) {
        // Pošalji kraj svim sudionicima koji slušaju
        // Stanje se čisti REST API pozivom
        quizStateService.clearState(quizCode);
        messagingTemplate.convertAndSend("/topic/quiz/" + quizCode + "/end", "Quiz ended");
    }
    ...
}

```

Događa se i REST API “***PUT /quiz-instance/\${quizCode}/end***” poziv na koji poslužiteljska strana označava prethodno kreiranu instancu kviza neaktivnom, pohranjuje statistiku odgovaranja u bazu podataka, te čisti lokalna pohrana statistike odgovaranja.

```

@RestController
@RequestMapping("/quiz-instance")
public class QuizInstanceController {
    ...
    @PutMapping("/{quizCode}/end")
    public ResponseEntity<Void> endQuizInstanceByCode(@PathVariable String quizCode) {
        quizInstanceService.endInstanceByCode(quizCode);
        return ResponseEntity.ok().build();
    }
    ...
}

@Service
public class QuizInstanceService {
    ...
    @Transactional
    public void endInstanceByCode(String quizCode) {
        QuizInstance instance =
        quizInstanceRepository.findByCodeAndEndTimestampIsNull(quizCode)
            .orElseThrow(() -> new NoSuchElementException("Active quiz instance not found"));

        // Počisti answerStore na kraju i pohrani u bazu što je potrebno
        // answerId, count
        Map<String, Integer> counts = new HashMap<>();
        String prefix = quizCode + ":";
        List<String> keysToRemove = new ArrayList<>();
        for (Map.Entry<String, List<ParticipantAnswersDTO>> entry : answerStore.entrySet()) {
            String key = entry.getKey();
            if (key.startsWith(prefix)) {
                for (ParticipantAnswersDTO answerDTO : entry.getValue()) {
                    for (String answerId : answerDTO.getAnswerIds()) {
                        counts.put(answerId, counts.getDefault(answerId, 0) + 1);
                    }
                }
                keysToRemove.add(key);
            }
        }
        for (String key : keysToRemove) {
            answerStore.remove(key);
        }
        ...
        instance.setEndTimestamp(LocalDateTime.now());
        quizInstanceRepository.save(instance);
    }
}

```

4. Upute za pokretanje za OS Linux

Na GitHub repozitoriju <https://github.com/ivabazo7/QuizStream> se nalazi implementacija klijentske i poslužiteljske strane, svaka u svom direktoriju, te SQL skripta za kreiranje korištene baze podataka (bez podataka, samo shema baze).

4.1. Baza podataka

U nekom klijentu za baze podataka (npr. pgAdmin) kreirajte PostgreSQL bazu podataka naziva “*quizstream*”, otvorite QueryTool te pokrenite priloženu SQL skriptu sa GitHub repozitorija koja će kreirati potrebne tablice i veze u bazi.

Možete odabrati i neko drugo ime za bazu, ali u tom slučaju će biti potrebno promijeniti podatke za spajanje na bazu u poslužiteljskoj aplikaciji:

- Pozicionirajte se u “*QuizStream-backend/src/main/resources/application.properties*”
- Promijenite naziv baze, lozinku za spajanje na PostgreSQL poslužitelj, korisnika i druge potrebne podatke

4.2. Poslužitelj

Poslužiteljska aplikacija nalazi se u direktoriju “*QuizStream-backend*”. Pozicionirajte se u taj direktorij i izvršite sljedeće naredbe u terminalu:

- “*./mvnw clean install*”
- “*./mvnw spring-boot:run*”

4.3. Klijent

Klijentska aplikacija nalazi se u direktoriju “*QuizStream-frontend*”. Pozicionirajte se u taj direktorij i izvršite sljedeće naredbe u terminalu:

- “*npm install*”
- “*npm run dev*”