QuizStream

1. Uvod

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

Autor: Iva Bazo

Aplikacija QuizStream je real-time kviz platforma inspirirana web aplikacijom Slido (https://www.slido.com/). Glavni cilj projekta bio je implementirati sustav koji omogućuje:

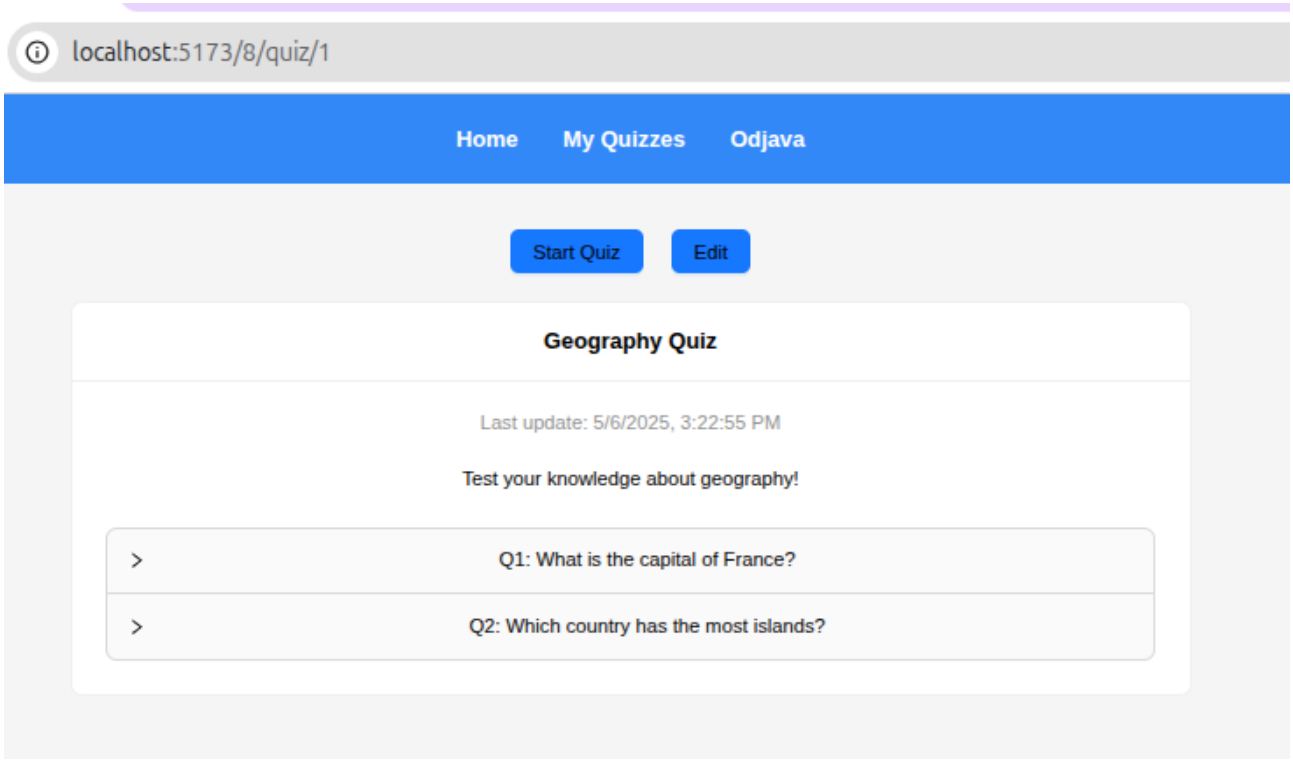
- Kreiranje kvizova s ABCD pitanjima
- Pridruživanje kvizu putem jedinstvenog koda
- Real-time odgovaranje na pitanja i prikaz statistike
- Upravljanje tokom kviza

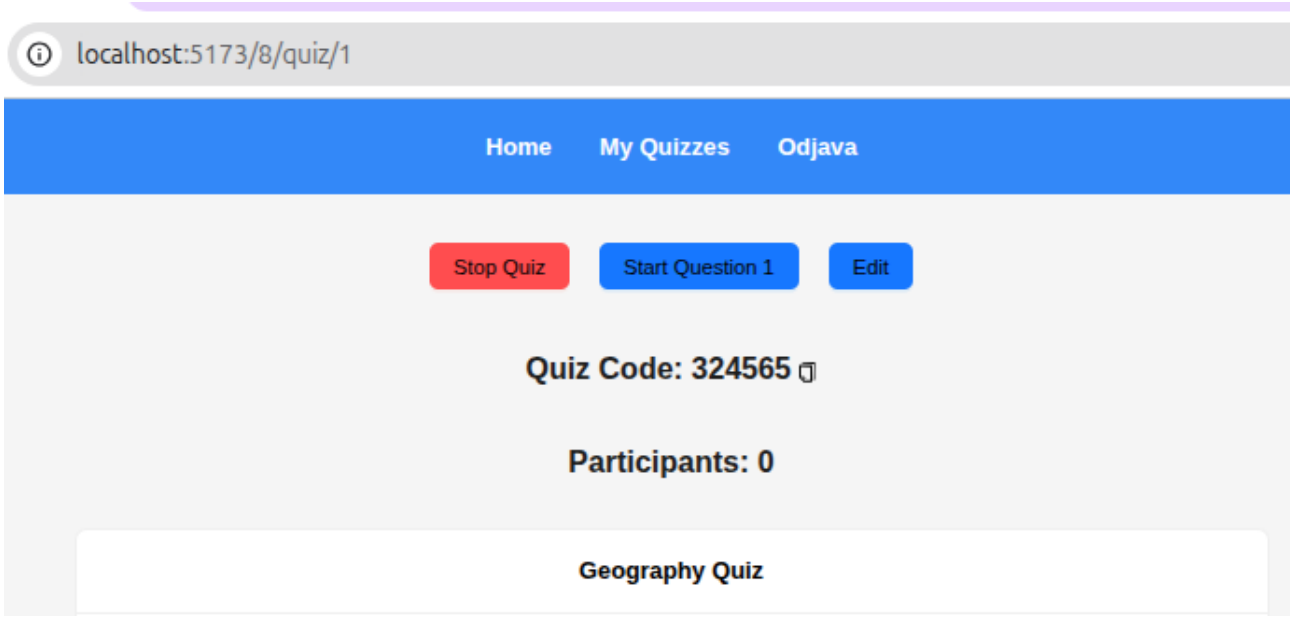
Ključne tehnologije korištene u razvoju:
- WebSocket (STOMP) za real-time komunikaciju
- REST API za CRUD operacije
- React s TypeScript-om za frontend
- Spring Boot za backend
- PostgreSQL za relacijsku bazu podataka

2. Opis funkcionalnosti

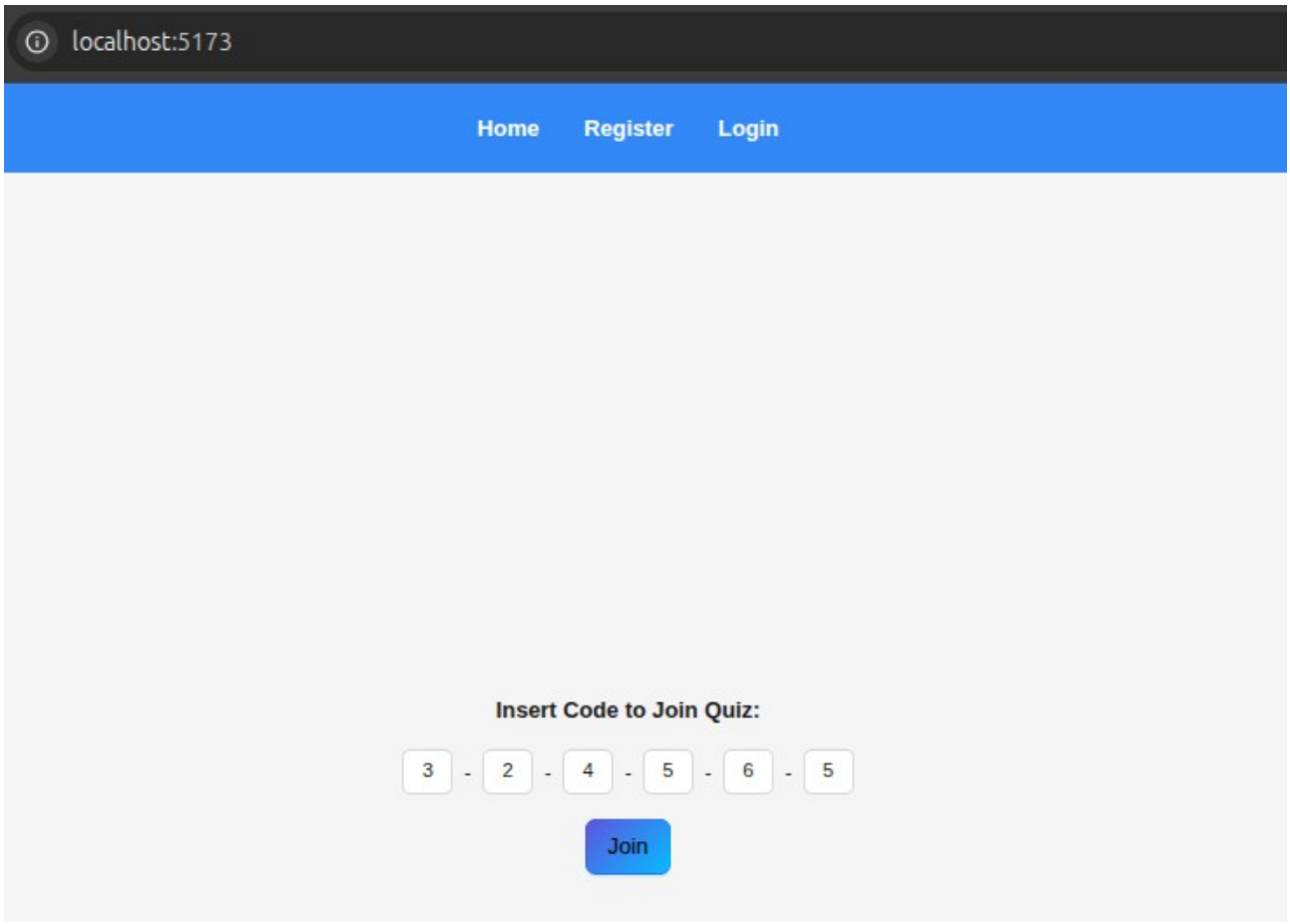
Korisnik ima mogućnost registracije, te nakon što se registrira i prijavi u sustav, može kreirati kvizove. Kreiranje kviza uključuje i kreiranje pitanja, mogućih odgovora na pitanje, te označavanje točnog odgovora. Isto tako, korisnik može uređivati i brisati već postojeće kvizove. To se odvija putem REST API, odnosno HTTP komunikacijom koristeći metode GET, POST, PUT i DELETE.

Nakon što korisnik kreira kvizove, može uživo ispitati znanje sudionika ili prikupiti informacije. Interaktivna komunikacija odvija se putem WebSocket protokola. Korisnik pokretanjem kviza implicitno kreira pristupni kod kvizu, koji podijeli sa sudionicima.

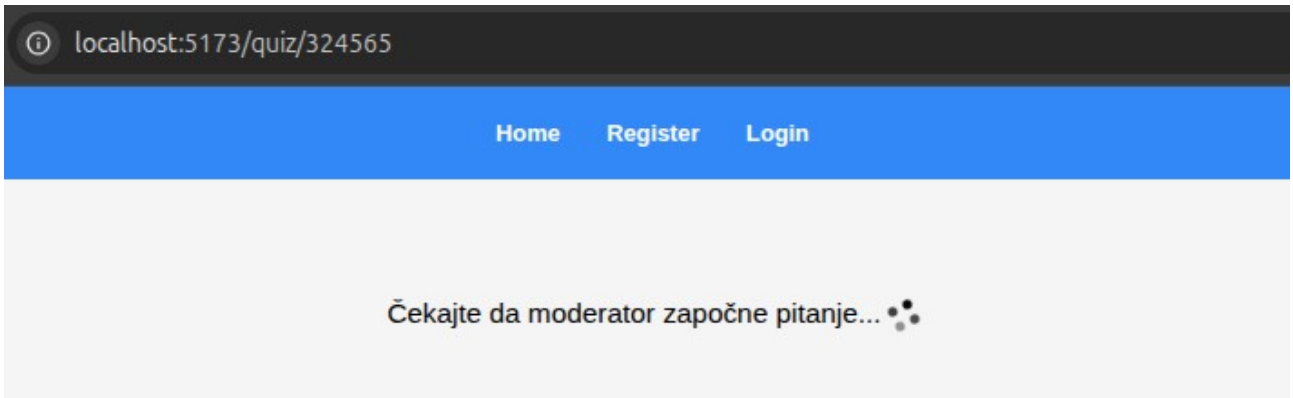




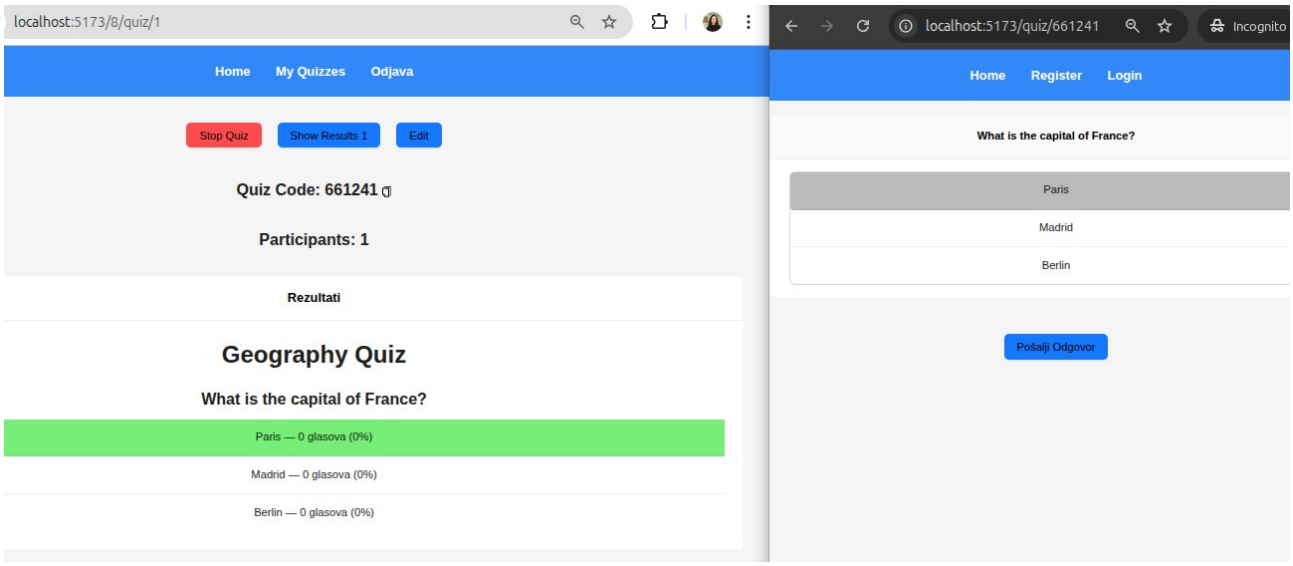
Sudionici su anonimni, odnosno nije potreban prethodna registracija i prijava kako bi sudjelovali u kvizu, potreban je samo kod koji dobiju od moderatora.
Upisivanjem pristupnog koda otvara se WebSOcket komunikacija, klijentska aplikacija pretplaćuje se na određene teme kako bi se dobila poruka koja stigne na određenu temu.



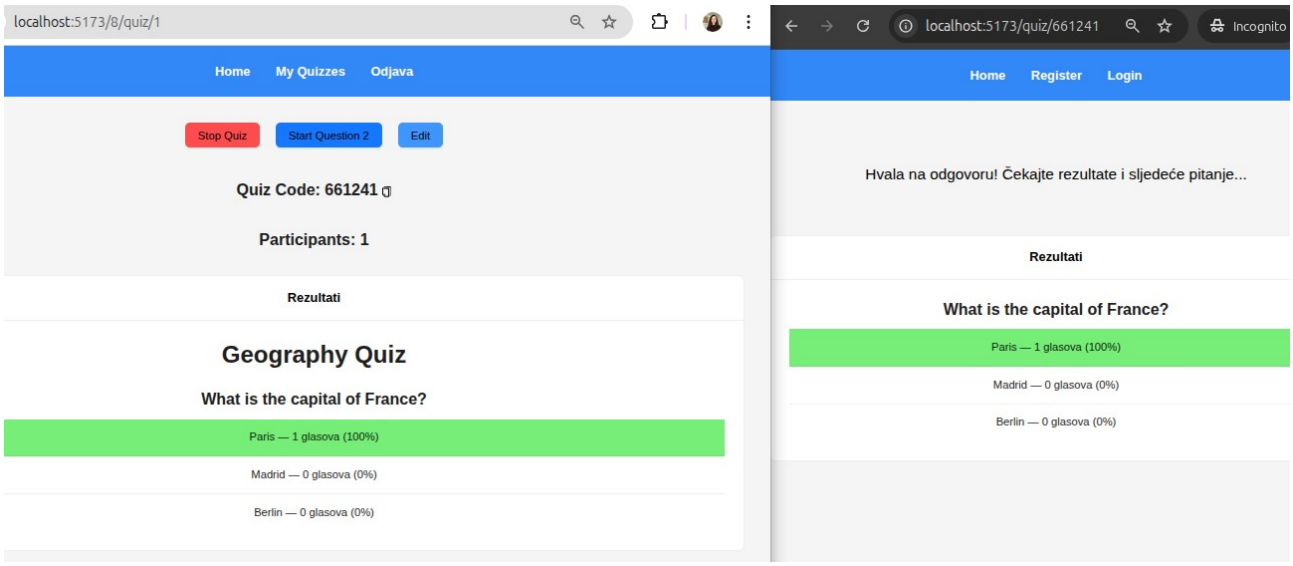
Nakon spajanja, klijentska aplikacija na stranici sudionika čeka da moderator pokrene pitanje.



Moderator pokreće pitanje pritiskom na gumb “Start Question 1”, te se istovremeno sudionicima prikazuje sučelje za odgovor, jer su se pretplatili na tu temu.



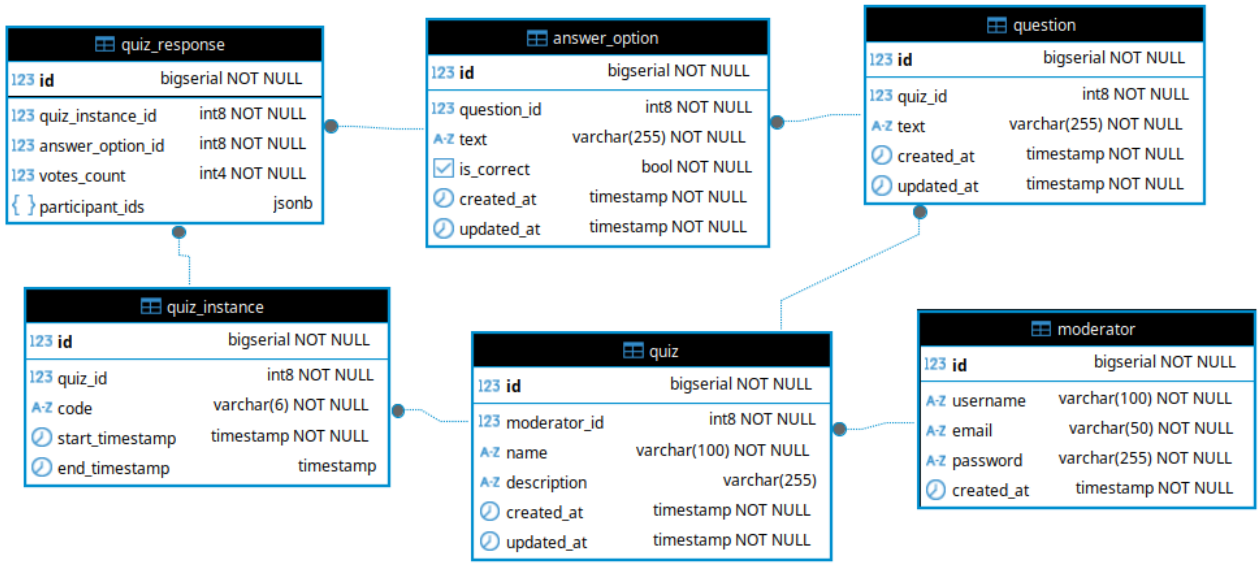
Istovremeno, moderator vidi koliko sudionika je trenutno priključeno, te vidi uživo statistiku glasanja. Nakon što moderator pritisne gumb “Show Results 1”, sudionicima se također prikaže statistika glasanja za to pitanje s označenim točnim odgovorom, te ne mogu više odgovarati na to pitanje.



Na kraju kviza, kada moderator pritisne gumb “Stop Quiz”, odspaja se sa WebSocket komunikacije, ali isto tako šalje i sudionicoma da se odspoje, odnosno sudionici su se prethodno pretplatili na tu temu, te kada moderator objavi da se odspoje, oni to i učine.

4. Baza podataka

Korištena je relacijska PostgreSQL baza podataka sa sljedećom shemom:



Entitet moderator može imati više kvizova, a entitet quiz može imati više pitanja (question), te više pokrenutih instanci (qioz_instance). Entitet question može imati više odgovora (answer_option), a entitet quiz_response se odnosi na glasanje sudionika za neku instancu kviza i odgovor na pitanje.

3. Tehnologije i protokoli uz primjere

3.1 WebSocket komunikacija na temelju primjera iz koda

Konfiguracija backenda:

```
@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");
    }
}
```

Ovom konfiguracijom određuje se da svaka pretplata mora započeti sa “/topic” kako bi kada nešto stigne na određeni broker bilo preusmjereno i pretplatniku, te da svaka objava treba započeti sa “/app”.

Konfiguracija frontenda na primjeru sudionika:

```
useEffect(() => {
if (!quizCode || !isValidQuiz || stompClientRef.current?.connected) return;

const socket = new SockJS(`${API_BASE_URL}/stomp-endpoint`);
const client = new Client({
webSocketFactory: () => socket,
reconnectDelay: 5000,
onConnect: () => {
console.log('Connected to WebSocket');

// Pretplata na trenutno pitanje
client.subscribe(`/topic/quiz/${quizCode}/question`, msg => {
const newQuestion: Question = JSON.parse(msg.body);
setQuestion(newQuestion);
setHasAnswered(false); // resetira status za novo pitanje
setResultsStat([]);
setShowStat(false);
```

```
});

// Pretplata na rezultate
client.subscribe(`/topic/quiz/${quizCode}/finalResults`, message => {
const parsedResults: AnswerResultStat[] = JSON.parse(message.body);
setResultsStat(parsedResults);
setShowStat(true);
});

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

// Pretplata na trenutno stanje
client.subscribe(`/topic/quiz/${quizCode}/state`, msg => {
const state = JSON.parse(msg.body);
if (state.currentQuestion) {
setQuestion(state.currentQuestion);
}
if (state.showResults) {
// Zatraži rezultate ako su dostupni
client.publish({
destination: `/app/quiz/${quizCode}/getResults`,
body: state.currentQuestion?.id || '',
});
}
});

// Zatraži trenutno stanje
client.publish({
destination: `/app/quiz/${quizCode}/getState`, // trigera /state
body: JSON.stringify({}),
});
},
onDisconnect: () => {
console.log('Disconnected from WebSocket');
},
onStompError: frame => {
console.error('STOMP error:', frame.headers.message);
message.error('Connection error. Please try again.');
navigate('/');
```

```
},
});

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

return () => {
client.deactivate();
console.log('WebSocket client deactivated');
};
}, [quizCode, isValidQuiz]);
```

Klijent se sa subscribe pretplaćuje na neku temu, a sa publish objavljuje nešto na neku temu, isto čini i moderator.

Na primjer, vidimo da se sudionik pretplaćuje na `client.subscribe(`/topic/quiz/${quizCode}/question`` , te kada moderator objavi pitanje (pritisne gumb “Start Question”), sudioniku se prikaže to pitanje.

Moderator:

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

Backend dio:

```
@MessageMapping("/quiz/{quizCode}/question")
public void sendQuestion(@DestinationVariable String quizCode, @Payload QuestionDTO question) {

    // Spremi trenutno stanje
    int currentIndex = quizStateService.getCurrentQuestionIndex(quizCode);
    quizStateService.setCurrentQuestionIndex(quizCode, currentIndex + 1);
    quizStateService.setCurrentQuestion(quizCode, question);
    quizStateService.setShowResults(quizCode, false);

    // Pošalji stanje svim klijentima
    getCurrentState(quizCode);

    // Pošalji pitanje svim sudionicima koji slušaju
    messagingTemplate.convertAndSend("/topic/quiz/" + quizCode + "/question", question);
}
```

“@MessageMapping” definira na što moderator treba slati. (publish) “@Payload” je objekt koji se očekuje u tijelu, što je u ovom slučaju pitanje.

“messagingTemplate.convertAndSend” preusmjerava poslano na ono na što se sudionici trebaju pretplatiti (subscribe).

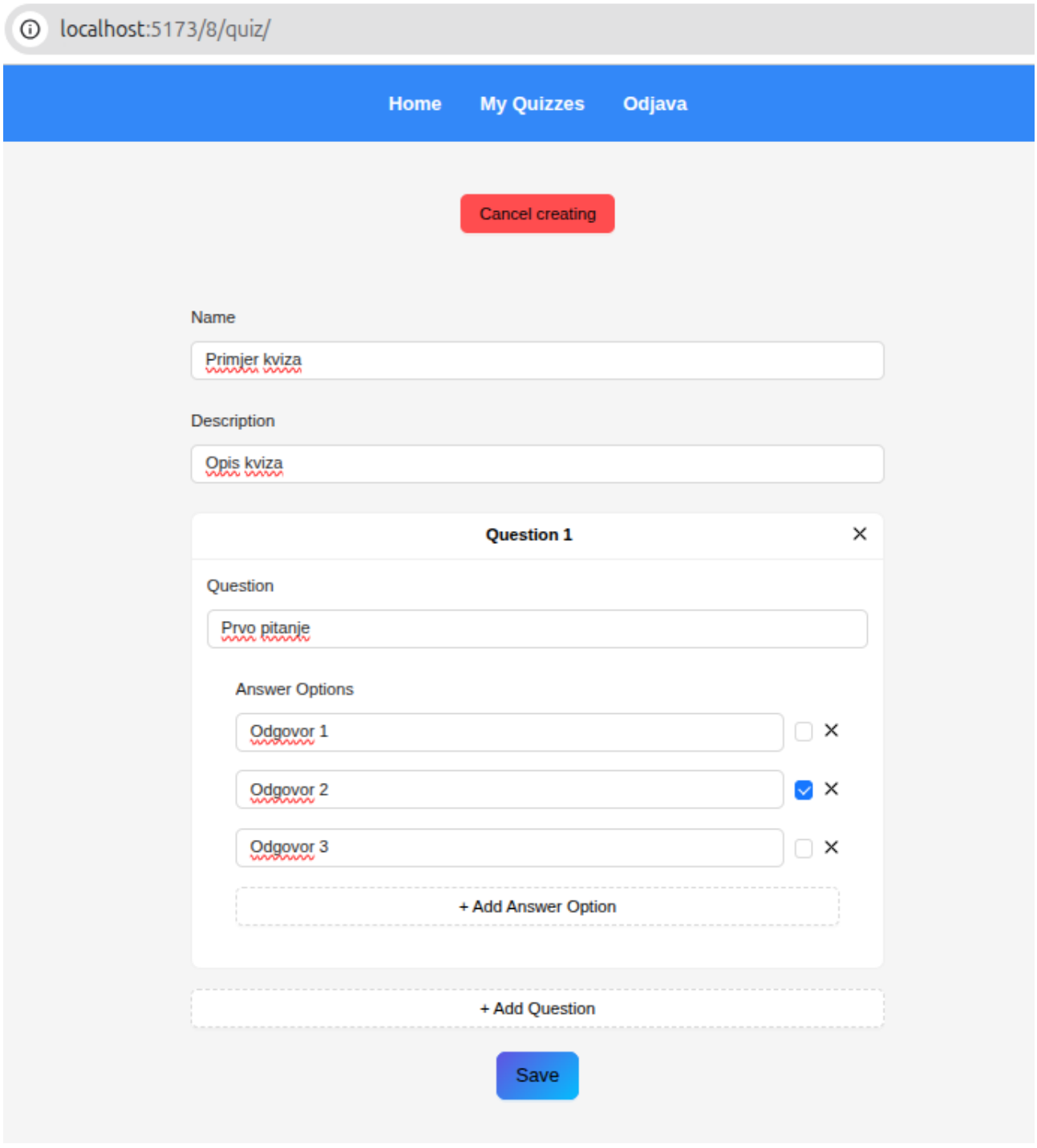
3.2 REST API na temelju primjera iz koda

Pomoću REST API se kreiraju kvizovi na endpoint “POST /quiz”, gdje se u tijelu zahtjeva šalju detalji kviza. Moguće je urediti postojeći kviz na endpoint “PUT /quiz/{quizId}”, gdje je quizId id kviza kojeg se uređuje, a u tijelu se zahtjeva objekt ažuriranog kviza. Moguć je dohvat hviza na “GET /quiz/{quizId}”, te brisanje kviza na “DELETE /quiz/{quizID}”. Ovo je primjer backend dijela kreiranja kviza, te dohvaćanja kviza:

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



@GetMapping("/{quizId}")
public ResponseEntity<QuizDTO> getQuiz(@PathVariable("quizId") Long quizId) {
    QuizDTO quiz = quizService.getQuiz(quizId);
    if (quiz == null) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
    }
    return ResponseEntity.status(HttpStatus.CREATED).body(quiz);
}
```

Slanje zahtjeva za kreiranje kviza sa frontenda na backend, te primjer forme kreiranja:

```
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);
```


localhost:5173/8/quiz/
Home
My Quizzes
Odjava
Cancel creating
Name
Primjer kviza
Description
Opis kviza
Question 1
Question
Prvo pitanje
Answer Options
Odgovor 1
Odgovor 2
Odgovor 3
+ Add Answer Option
+ Add Question
Save