Zaawansowane techniki internetowe

APLIKACJA QUIZ’OWA

07.04.2025

Zuzanna Flis, nr. Albumu 410837

SPIS TREŚCI

Wstęp 2

Co robi aplikacja? 2

Zastosowane Technologie 2

Funkcjonalność - Frontend 3

Funkcjonalność – Backend 5

Użycie Hibernate w projekcie 8

Baza Danych - MySQL 12

Tabele i ich Atrybuty 12

Relacje między Tabelami 14

Sekwencje 14

Wygląd – Demo 14

Dla użytkownika 15

Dla admina 18

Podsumowanie 20

WSTĘP

Aplikacja quizowa została zaprojektowana z myślą o zapewnieniu użytkownikom wygodnego i interaktywnego sposobu uczestniczenia w quizach oraz zarządzania profilem. Poniżej przedstawiam kluczowe funkcje i cechy aplikacji.

CO ROBI APLIKACJA?

DLA UŻYTKOWNIKÓW

- **Przeglądanie i uczestniczenie w quizach**: Użytkownicy mogą przeglądać dostępne quizy i w nich uczestniczyć.
- **Strona profilowa**: Aplikacja zapewnia spersonalizowaną stronę profilową do wyświetlania danych użytkownika.
- **Intuicyjny interfejs**: Zapewnia czytelny i intuicyjny interfejs do interakcji z quizami.

DLA ADMINISTRATORÓW

Aplikacja umożliwia administratorom zarządzanie quizami, np. dodawanie, edytowanie lub usuwanie quizów, a także kategorii. Dodatkowo dostarcza funkcje takie jak wyświetlanie czasu, punktów i osób wykonujących dany quiz.

POTENCJALNE PRZYPADKI UŻYCIA

- **Platformy edukacyjne**: Do testowania wiedzy.
- **Szkolenia korporacyjne i oceny**: W celu szkoleń i ocen pracowników.
- **Gry quizowe**: Dla zabawy i interakcji użytkowników.

ZASTOSOWANE TECHNOLOGIE

Frontend:

1. **Angular**:
    - Framework do budowy aplikacji frontendowych.
    - Wersja Angulara: 16.2.0
    - Używane moduły Angulara:
        - @angular/core, @angular/common, @angular/router, @angular/forms, @angular/animations.
2. **Angular Material**:
    - Biblioteka komponentów UI dla Angulara.
    - Wersja: 16.2.4.

- Używane komponenty, np. mat-card, mat-button, mat-card-header.

3. **Bootstrap**:
    - Framework CSS do stylizacji i responsywności.
    - Wersja: 5.3.1.
    - Używana wersja bootstrap-grid-only-css dla siatki.
4. **RxJS**:
    - Biblioteka do obsługi programowania reaktywnego.
    - Wersja: 7.8.0.
5. **SweetAlert2**:
    - Biblioteka do wyświetlania ładnych okien dialogowych.
    - Wersja: 11.7.27.
6. **ngx-ui-loader**:
    - Biblioteka do wyświetlania loaderów w aplikacji Angular.
    - Wersja: 13.0.0.

Backend:

- Język programowania: Java
- Framework: Spring Boot
- Zarządzanie zależnościami: Maven
- Baza danych: MySQL (za pomocą mysql-connector-j)
- Hibernate jako implementacja JPA(Java Persistance API)
- Bezpieczeństwo: Spring Security
- Logowanie: SLF4J, Logback
- Autoryzacja i uwierzytelnianie: JWT (JSON Web Token) za pomocą biblioteki jjwt
- Inne biblioteki: Lombok (do generowania kodu), Spring Data JPA (do operacji na bazie danych)

FUNKCJONALNOŚĆ - FRONTEND

Zarządzanie Quizami

- **Lista dostępnych quizów**: Użytkownicy mogą przeglądać listę quizów w różnych kategoriach.
- **Szczegóły quizu**: Każdy quiz wyświetla następujące informacje:
    - Tytuł
    - Kategoria
    - Opis

        - Maksymalna liczba punktów
        - Liczba pytań
        - Status aktywności quizu (czy jest aktywny, czy nie)
        - Dodatkowe zdjęcie
- **Rozpoczęcie quizu**: Użytkownicy mogą rozpocząć quiz klikając przycisk „Start”, który przenosi ich do interfejsu rozwiązywania quizu.
- **Instrukcje quizu**: Dostępna jest również opcja wyświetlenia instrukcji quizu przed rozpoczęciem.

Zarządzanie Profilem

- **Informacje o profilu**: Użytkownicy mogą wyświetlić swoje dane profilowe, w tym:
    - Zdjęcie profilowe (wyświetlane w okrągłym kontenerze).
    - Imię i nazwisko oraz inne dane osobowe.
- **Stylizacja profilu**: Strona profilu została zaprojektowana tak, aby wyświetlać dane użytkownika w czytelny i zorganizowany sposób.

Projektowanie Responsywne

- **Angular Material**: Aplikacja wykorzystuje komponenty Angular Material (np. mat-card, mat-button), co zapewnia nowoczesny i responsywny interfejs użytkownika.
- **Karty**: Quizy i informacje o profilu są wyświetlane na kartach, co gwarantuje atrakcyjny układ wizualny.

Obsługa Stanów Pustych

- **Brak quizów**: Jeśli w danej kategorii nie ma dostępnych quizów, aplikacja wyświetla komunikat: „Brak quizów w tej kategorii.”

Działania Użytkownika

- **Przyciski akcji**: Przyciski są dostępne dla akcji takich jak rozpoczęcie quizu lub wyświetlenie instrukcji.
- **Efekty hover**: Przyciski są stylizowane z efektami hover, co poprawia komfort użytkowania.

Stylizacja

- **CSS niestandardowy**: Aplikacja wykorzystuje dodatkowy CSS do stylizacji, takiej jak:
    - Cienie i zaokrąglone rogi kart.
    - Flexbox do wyrównywania układu.
    - Efekty hover dla przycisków.
    - Czysty i profesjonalny design tabel i szczegółów profilu.

FUNKCJONALNOŚĆ – BACKEND

Część funkcjonalności backend'owej:

- Rejestracja użytkowników: Endpoint /api/v1/auth/register pozwala na rejestrację nowych użytkowników.
- Uwierzytelnianie użytkowników: Endpoint /api/v1/auth/authenticate umożliwia uwierzytelnianie użytkowników i generowanie tokenów JWT.

```java
@RequiredArgsConstructor
@RequestMapping("/api/v1/auth")
public class AuthenticationController {
    2 usages
    private final AuthenticationService authenticationService;
    1 usage
    private static final Logger logger = LoggerFactory.getLogger(AuthenticationController.class);

    no usages  zuzaflis
    @PostMapping("/register")
    public ResponseEntity<AuthenticationResponse> register(
            @RequestBody RegisterRequest registerRequest)
    {
       return ResponseEntity.ok(authenticationService.register(registerRequest));
    }
    no usages  zuzaflis
    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponse> authenticate(
            @RequestBody AuthenticationRequest authRequest)

    {
        logger.info("Received authentication request for username: {}", authRequest.getUsername());

        return ResponseEntity.ok(authenticationService.authenticate(authRequest));
    }

    no usages  zuzaflis
    @GetMapping("/current-user")
    public Object getCurrentUser(Authentication authentication) { return authentication.getPrincipal(); }

}
```

- Zarządzanie kategoriami: Endpointy w kontrolerze CategoryController pozwalają na dodawanie, pobieranie, aktualizowanie i usuwanie kategorii egzaminów. Dostęp do tych operacji jest ograniczony do użytkowników z rolą ADMIN.

```java
@RestController
@RequestMapping("/category")
@PreAuthorize("hasRole('ADMIN')")
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:4200", allowedHeaders = "*", allowCredentials = "true")
public class CategoryController {
    5 usages
    private final CategoryService categoryService;

    //addCategory
    no usages  zuzaflis
    @PostMapping("/")
    public ResponseEntity<Category> addCategory(@RequestBody Category category){
        Category cat1 = this.categoryService.addCategory(category);
        return ResponseEntity.ok(cat1);
    }
    //getCategory
    no usages  zuzaflis
    @Transactional
    @GetMapping("/{categoryId}")
    public Category getCategory(@PathVariable("categoryId") Long categoryId){
        return this.categoryService.getCategory(categoryId);
    }
    //get all categories
    no usages  zuzaflis
    @Transactional
    @GetMapping("/")
    public ResponseEntity<?> getCategories() { return ResponseEntity.ok(this.categoryService.getCategories()); }
    //updateCategory
    no usages  zuzaflis
    @PutMapping("/")
    public Category updateCategory(@RequestBody Category category){
        return this.categoryService.updateCategory(category);
```

JWT i Spring Security:

- JWT (JSON Web Token): JWT jest używany do bezpiecznego przekazywania informacji między klientem a serwerem. Tokeny są generowane podczas uwierzytelniania użytkownika i zawierają zakodowane informacje o użytkowniku. Tokeny te są następnie używane do autoryzacji dostępu do chronionych zasobów.
- Spring Security: Spring Security jest używany do zarządzania uwierzytelnianiem i autoryzacją użytkowników. W projekcie zaimplementowano filtr JwtAuthenticationFilter, który przechwytuje żądania HTTP, sprawdza obecność tokena JWT w nagłówku Authorization, a następnie weryfikuje jego ważność. Jeśli token jest ważny, użytkownik jest uwierzytelniany i autoryzowany do dostępu do chronionych zasobów.

```java
private final UserDetailsService userDetailsService;
no usages  zuzaflis
@Override
protected void doFilterInternal(
        @NonNull HttpServletRequest request,
        @NonNull  HttpServletResponse response,
        @NonNull FilterChain filterChain)
         throws ServletException, IOException {

    final String authHeader = request.getHeader( s: "Authorization");
    final String jwt;
    final String userName;
    if(authHeader == null || !authHeader.startsWith("Bearer ")){
        filterChain.doFilter(request, response);
        return;
    }
    jwt = authHeader.substring( beginIndex: 7);
    userName = jwtService.extractUserName(jwt);

    if(userName != null && SecurityContextHolder.getContext().getAuthentication()==null){
        UserDetails userDetails = this.userDetailsService.loadUserByUsername(userName);
        if(jwtService.isTokenValid(jwt,userDetails)){
            UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                    userDetails,
                    credentials: null,
                    userDetails.getAuthorities()
            );
            authToken.setDetails(
                    new WebAuthenticationDetailsSource().buildDetails(request)
            );
            //update security context holder
            SecurityContextHolder.getContext().setAuthentication(authToken);
        }
    }
    filterChain.doFilter(request, response);
}
```

Dodatkowe funkcjonalności projektu:

Zarządzanie quizami:

- Dodawanie quizów: Endpoint POST /quiz/ pozwala na dodawanie nowych quizów.
- Aktualizowanie quizów: Endpoint PUT /quiz/ umożliwia aktualizowanie istniejących quizów.
- Pobieranie quizów: Endpoint GET /quiz/ zwraca listę wszystkich quizów.
- Pobieranie quizów według kategorii: Endpoint GET /quiz/category/{cid} zwraca quizy przypisane do określonej kategorii.
- Pobieranie pojedynczego quizu: Endpoint GET /quiz/{Id} zwraca szczegóły pojedynczego quizu.
- Usuwanie quizów: Endpoint DELETE /quiz/{Id} pozwala na usunięcie quizu.

Zarządzanie pytaniami:

- Dodawanie pytań: Endpoint POST /question/ pozwala na dodawanie nowych pytań do quizów.
- Aktualizowanie pytań: Endpoint PUT /question/ umożliwia aktualizowanie istniejących pytań.
- Pobieranie pytań: Endpoint GET /question/ zwraca listę wszystkich pytań.
- Pobieranie pytań z quizu: Endpoint GET /question/quiz/{Id} zwraca pytania przypisane do określonego quizu.

- Pobieranie pojedynczego pytania: Endpoint GET /question/{Id} zwraca szczegóły pojedynczego pytania.
- Usuwanie pytań: Endpoint DELETE /question/{Id} pozwala na usunięcie pytania.

Zarządzanie użytkownikami:

- Pobieranie użytkownika: Endpoint GET /api/v1/user/{username} zwraca szczegóły użytkownika na podstawie nazwy użytkownika.

Przykładowe pliki:

- QuizController.java: Kontroler REST, który obsługuje operacje CRUD na quizach.
- QuestionController.java: Kontroler REST, który obsługuje operacje CRUD na pytaniach.
- UserController.java: Kontroler REST, który obsługuje operacje związane z użytkownikami.

UŻYCIE HIBERNATE W PROJEKCIE

W projekcie używam Hibernate jako implementacji JPA (Java Persistence API) do zarządzania operacjami na bazie danych. Hibernate umożliwia mapowanie obiektowo-relacyjne (ORM), co pozwala na łatwe mapowanie klas Java na tabele w bazie danych.

Konfiguracja Hibernate

Hibernate jest skonfigurowany za pomocą Spring Boot Starter Data JPA, który automatycznie integruje Hibernate z projektem.

Definiowanie encji

Encje są definiowane jako klasy Java z adnotacjami JPA. Przykładowa encja *Question*:

Repozytoria

Repozytoria są interfejsami, które rozszerzają JpaRepository, co umożliwia wykonywanie operacji CRUD na encjach. Przykładowe repozytorium *QuestionRepository*:

```
2 usages  zuzaflis
public interface QuestionRepository extends JpaRepository<Question, Long> {
    1 usage  zuzaflis
    Set<Question> findByQuiz(Quiz quiz);
}
```

Serwisy

Serwisy zawierają logikę biznesową i korzystają z repozytoriów do wykonywania operacji na bazie danych. Przykładowy fragment serwisu *QuestionService* :

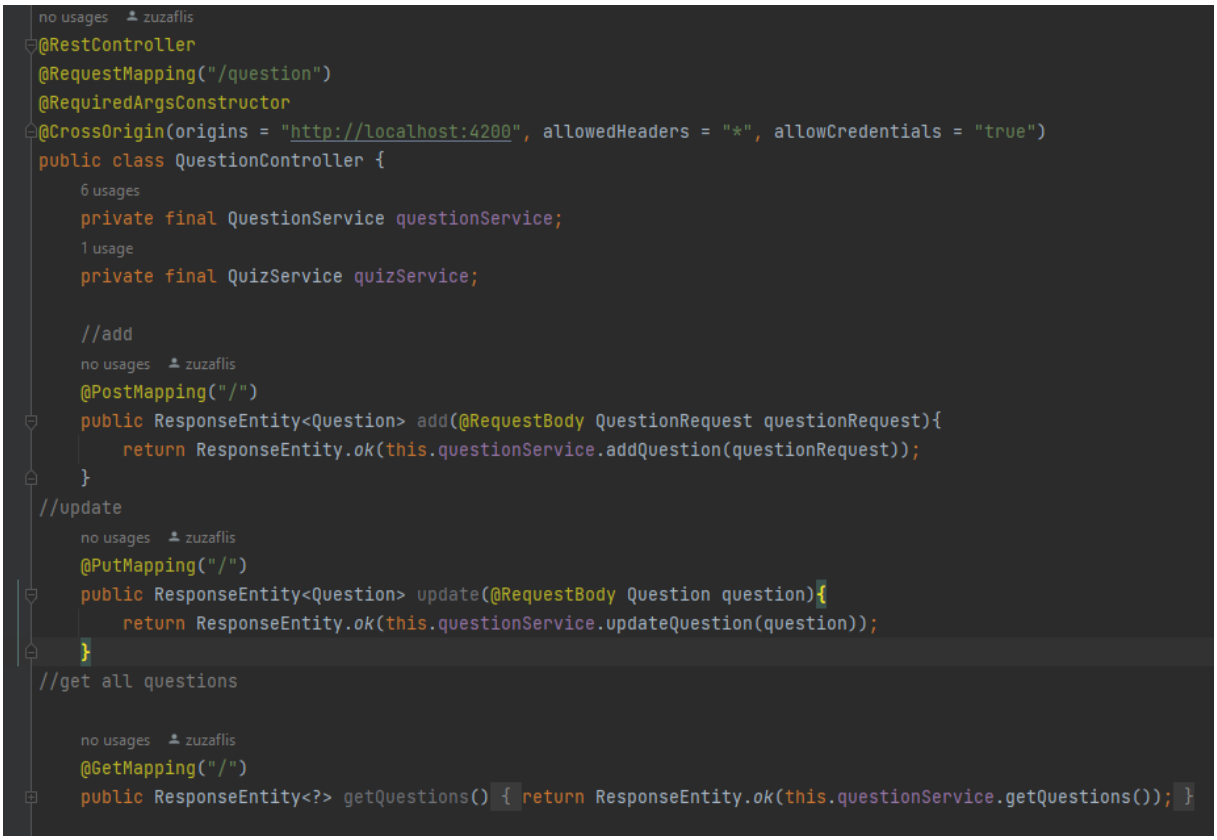
```
@Service
@RequiredArgsConstructor
public class QuestionService {
    1 usage
    private final QuizRepository quizRepository;
    7 usages
    private final QuestionRepository questionRepository;
    1 usage
    private final UserRepository userRepository;
    1 usage
    private final ResultRepository resultRepository;
    1 usage
    @PersistenceContext
    private EntityManager entityManager;
    1 usage  zuzaflis
    @Transactional
    public Question addQuestion(QuestionRequest questionRequest){
        Long quizId =questionRequest.getQuizId();
        Optional<Quiz> optionalQuiz = this.quizRepository.findById(quizId);

        if(optionalQuiz.isPresent()){
            Quiz quiz = optionalQuiz.get();
            quiz = entityManager.merge(quiz);
            Question question = Question.builder()
                    .content(questionRequest.getContent())
                    .image(questionRequest.getImage())
                    .option1(questionRequest.getOption1())
                    .option2(questionRequest.getOption2())
                    .option3(questionRequest.getOption3())
                    .option4(questionRequest.getOption4())
                    .answer(questionRequest.getAnswer())
                    .quiz(quiz)
                    .build();

            return this.questionRepository.save(question);
        }
        else throw new EntityNotFoundException("Quiz not found");
    }
```

Kontrolery

Kontrolery obsługują żądania HTTP i korzystają z serwisów do wykonywania operacji na bazie danych. Przykładowy kontroler *QuestionController*:



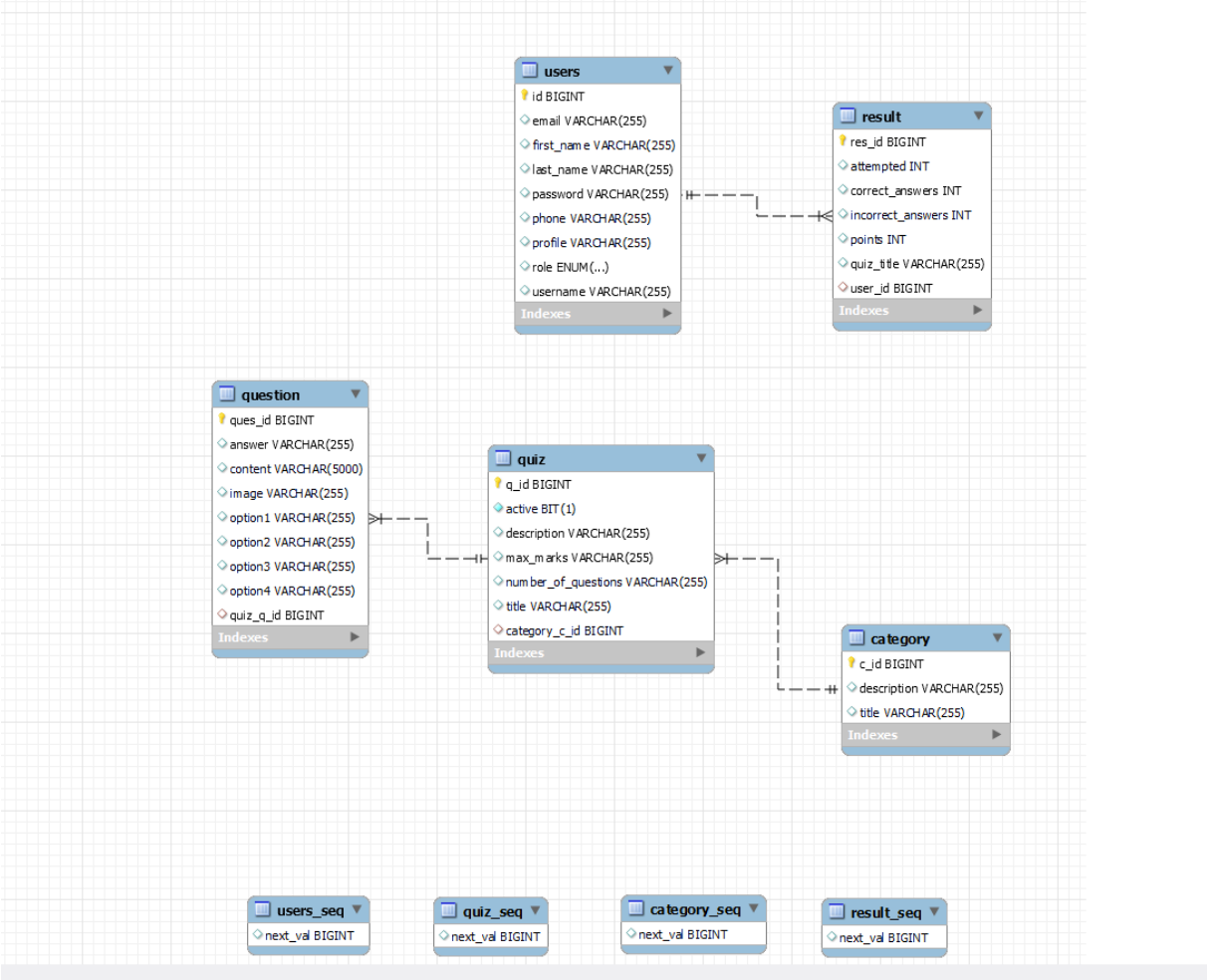
```
no usages  zuzaflis
@RestController
@RequestMapping("/question")
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:4200", allowedHeaders = "*", allowCredentials = "true")
public class QuestionController {
    6 usages
    private final QuestionService questionService;
    1 usage
    private final QuizService quizService;

    //add
    no usages  zuzaflis
    @PostMapping("/")
    public ResponseEntity<Question> add(@RequestBody QuestionRequest questionRequest){
        return ResponseEntity.ok(this.questionService.addQuestion(questionRequest));
    }
//update
    no usages  zuzaflis
    @PutMapping("/")
    public ResponseEntity<Question> update(@RequestBody Question question){
        return ResponseEntity.ok(this.questionService.updateQuestion(question));
    }
//get all questions

    no usages  zuzaflis
    @GetMapping("/")
    public ResponseEntity<?> getQuestions() { return ResponseEntity.ok(this.questionService.getQuestions()); }
```

BAZA DANYCH - MYSQL

Relacje w bazie danych:



TABELE I ICH ATRYBUTY

1. **users**

   - id (BIGINT): Klucz główny, unikalny identyfikator użytkownika.
   - email (VARCHAR(255)): Adres e-mail użytkownika.
   - first_name (VARCHAR(255)): Imię użytkownika.
   - last_name (VARCHAR(255)): Nazwisko użytkownika.
   - password (VARCHAR(255)): Zaszyfrowane hasło użytkownika.
   - phone (VARCHAR(255)): Numer telefonu użytkownika.
   - profile (VARCHAR(255)): Dodatkowe informacje profilowe (np. adres).
   - role (ENUM(...)): Rola użytkownika (np. administrator, zwykły użytkownik).
   - username (VARCHAR(255)): Nazwa użytkownika.

2. **quiz**

    - q_id (BIGINT): Klucz główny, unikalny identyfikator quizu.
    - active (BIT(1)): Flaga wskazująca, czy quiz jest aktywny.
    - description (VARCHAR(255)): Krótki opis quizu.
    - max_marks (VARCHAR(255)): Maksymalna liczba punktów do zdobycia w quizie.
    - number_of_questions (VARCHAR(255)): Liczba pytań w quizie.
    - title (VARCHAR(255)): Tytuł quizu.
    - category_c_id (BIGINT): Klucz obcy, odwołujący się do tabeli category, identyfikator kategorii, do której należy quiz.

3. **question**

    - ques_id (BIGINT): Klucz główny, unikalny identyfikator pytania.
    - answer (VARCHAR(255)): Prawidłowa odpowiedź na pytanie.
    - content (VARCHAR(5000)): Treść pytania.
    - image (VARCHAR(255)): Ścieżka do obrazka związanego z pytaniem (opcjonalne).
    - option1 (VARCHAR(255)): Pierwsza opcja odpowiedzi.
    - option2 (VARCHAR(255)): Druga opcja odpowiedzi.
    - option3 (VARCHAR(255)): Trzecia opcja odpowiedzi.
    - option4 (VARCHAR(255)): Czwarta opcja odpowiedzi.
    - quiz_q_id (BIGINT): Klucz obcy, odwołujący się do tabeli quiz, identyfikator quizu, do którego należy pytanie.

4. **category**

    - c_id (BIGINT): Klucz główny, unikalny identyfikator kategorii quizów.
    - description (VARCHAR(255)): Opis kategorii.
    - title (VARCHAR(255)): Nazwa kategorii.

5. **result**

    - res_id (BIGINT): Klucz główny, unikalny identyfikator wyniku.
    - attempted (INT): Liczba pytań, na które użytkownik odpowiedział.
    - correct_answers (INT): Liczba poprawnych odpowiedzi.
    - incorrect_answers (INT): Liczba niepoprawnych odpowiedzi.

- points (INT): Liczba punktów zdobytych przez użytkownika.
- quiz_title (VARCHAR(255)): Tytuł quizu, którego dotyczy wynik.
- user_id (BIGINT): Klucz obcy, odwołujący się do tabeli users, identyfikator użytkownika, który rozwiązywał quiz.

RELACJE MIĘDZY TABELAMI

- **users** i **result**: Relacja jeden-do-wielu. Jeden użytkownik może mieć wiele wyników w quizach.
- **quiz** i **question**: Relacja jeden-do-wielu. Jeden quiz może zawierać wiele pytań.
- **quiz** i **category**: Relacja wiele-do-jednego. Wiele quizów może należeć do jednej kategorii.
- **quiz** i **result**: Relacja jeden-do-wielu. Jeden quiz może mieć wiele wyników.
- **users** i **password**: Relacja jeden do jednego. Jeden użytkownik ma jedno hasło.

SEKWENCJE

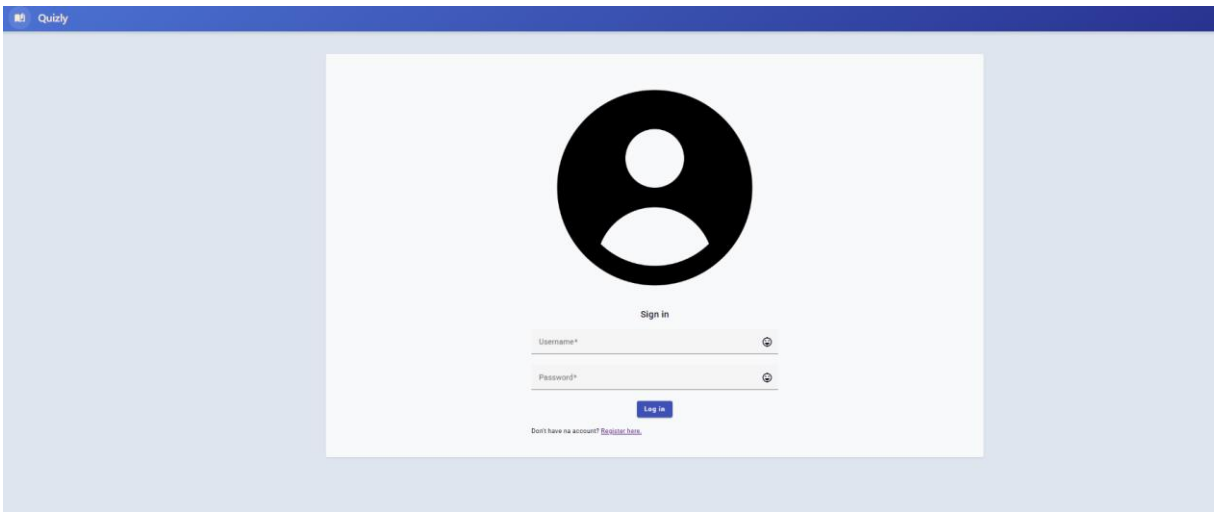
Dodatkowo, w bazie danych znajdują się tabele sekwencji:

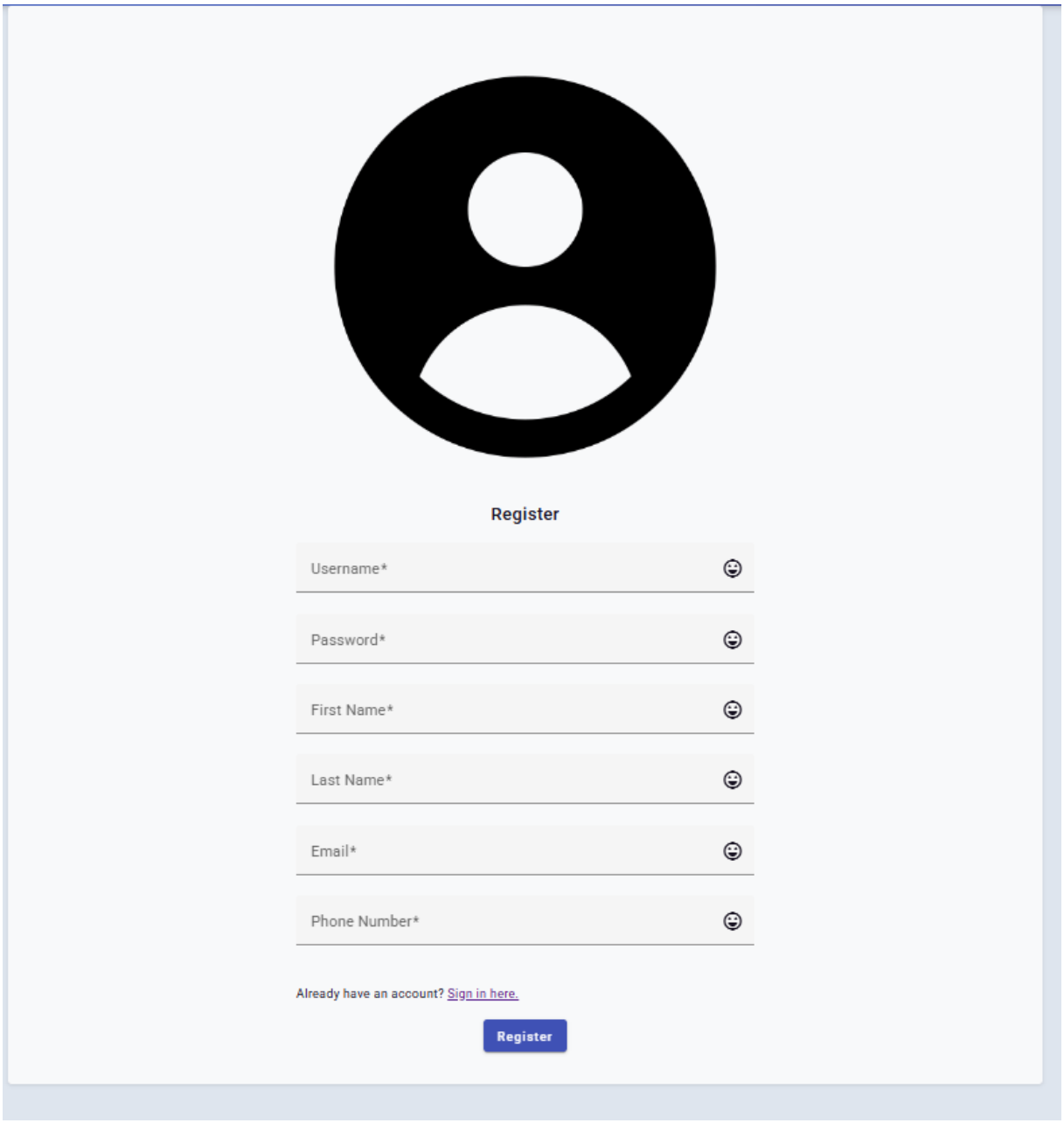
- users_seq: Przechowuje następną dostępną wartość dla identyfikatora użytkownika.
- quiz_seq: Przechowuje następną dostępną wartość dla identyfikatora quizu.
- category_seq: Przechowuje następną dostępną wartość dla identyfikatora kategorii.
- result_seq: Przechowuje następną dostępną wartość dla identyfikatora wyniku.

Te tabele służą do automatycznego generowania unikalnych identyfikatorów dla nowych rekordów w odpowiednich tabelach.

WYGLĄD – DEMO

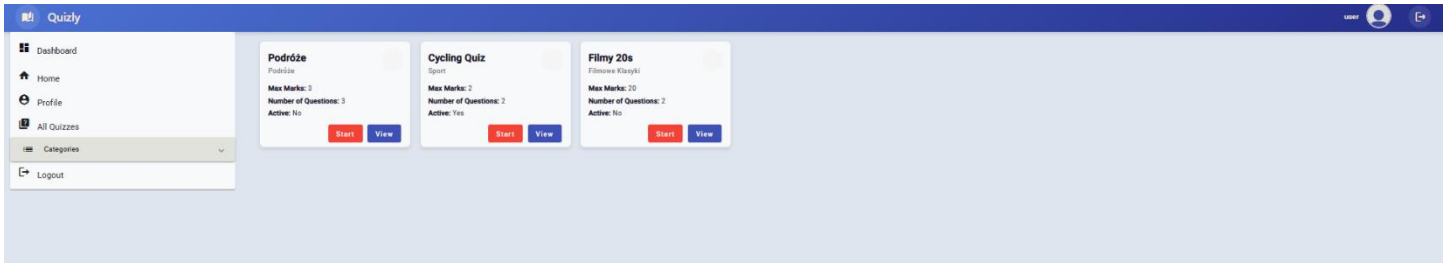
- Logowanie/Rejestracja



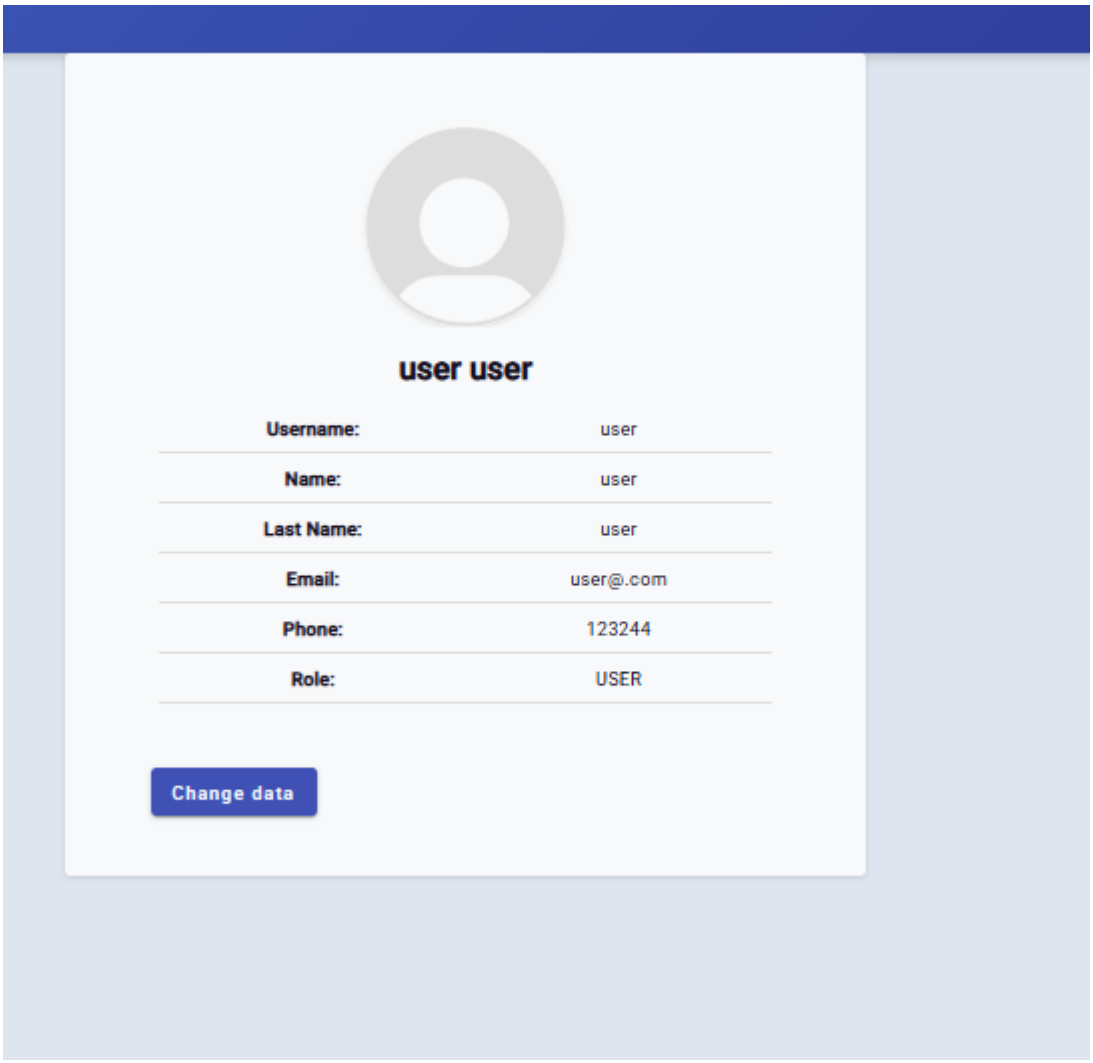


DLA UŻYTKOWNIKA

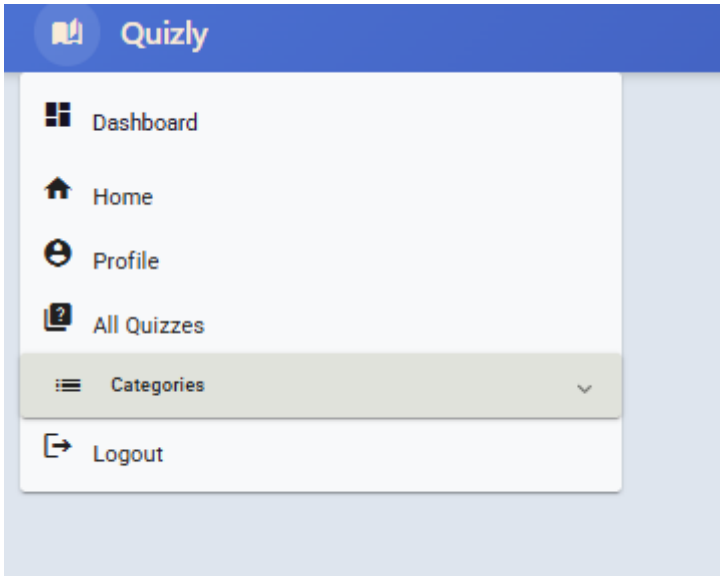
- Wszystkie quizy



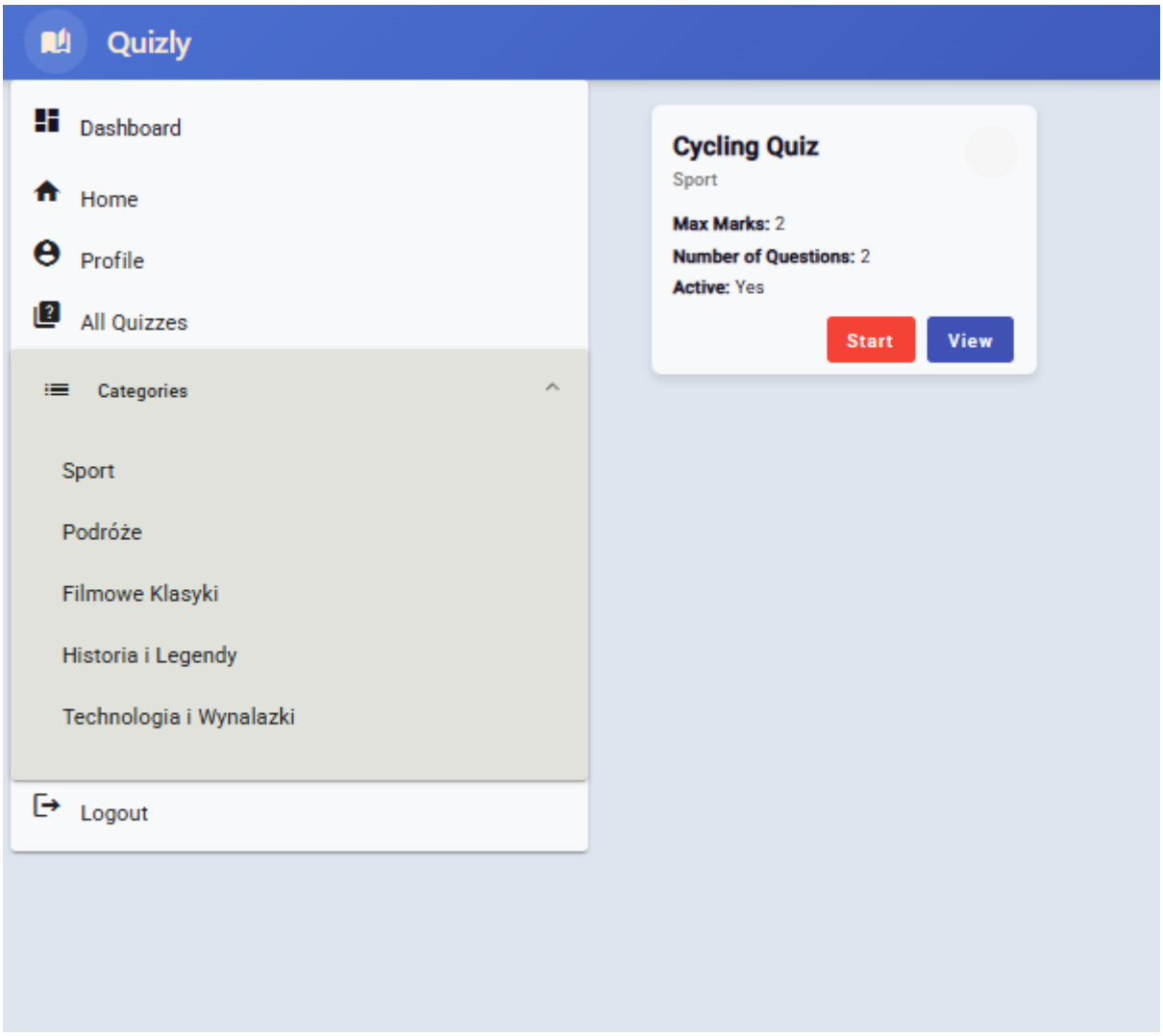
- Profil wraz z mozliwoscia zmiany danych



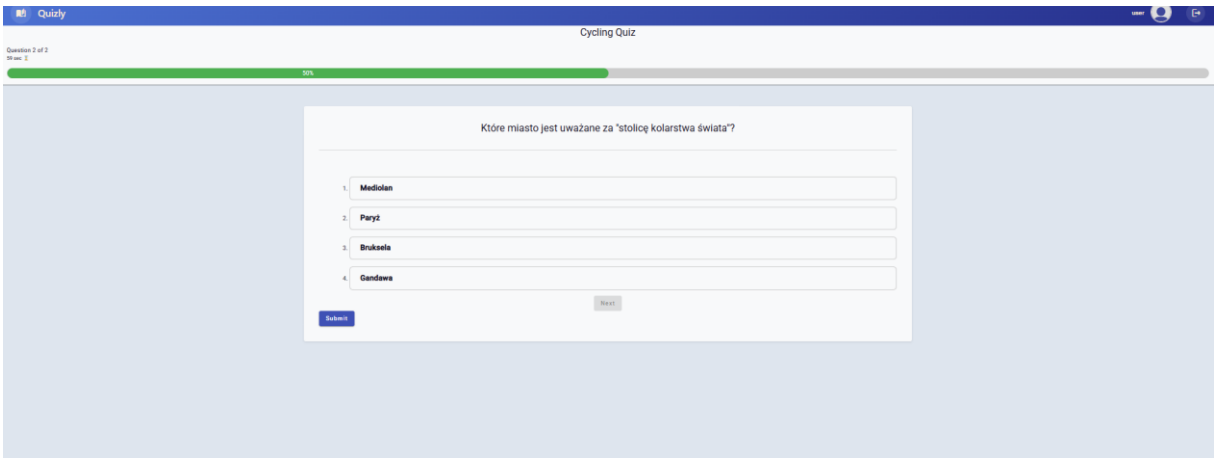
- Dashboard



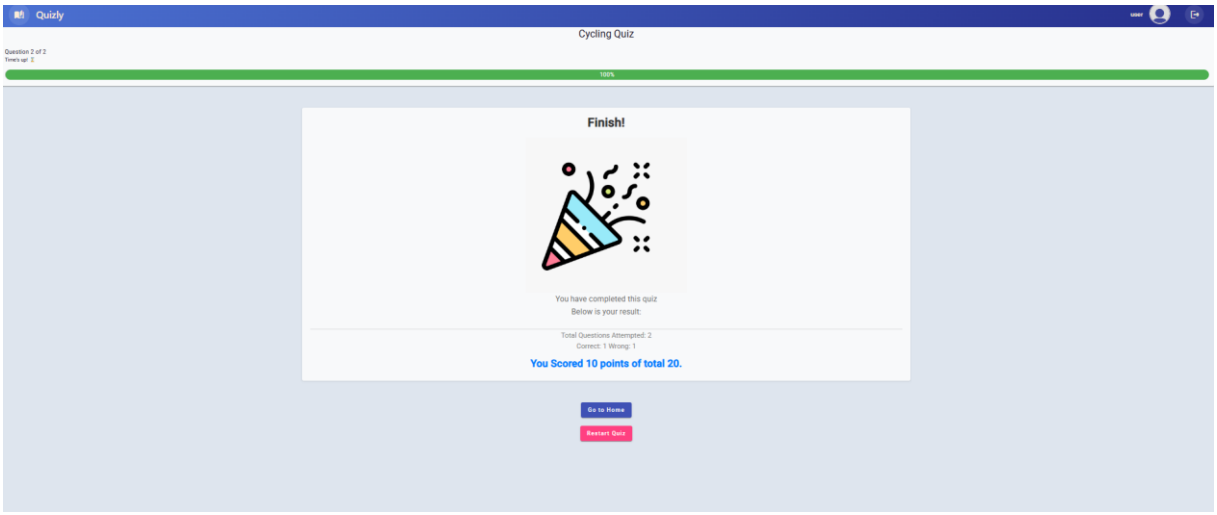
- Kategorie



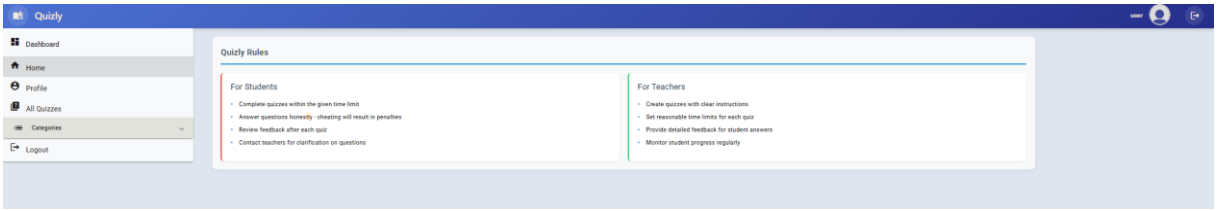
- Test



- Wynik końcowy

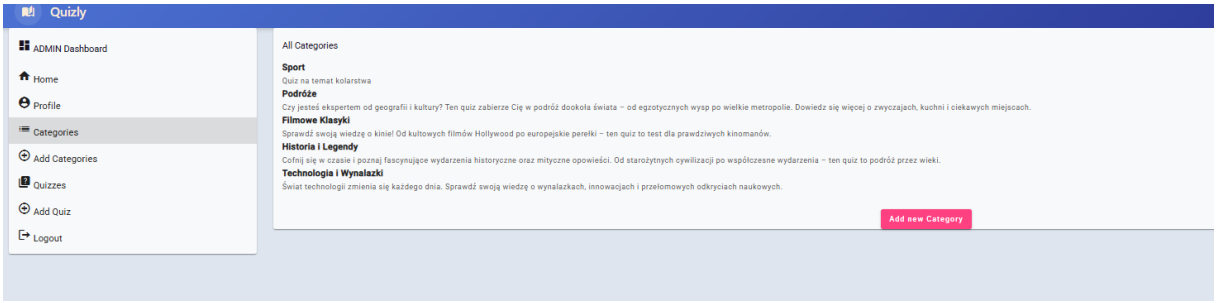


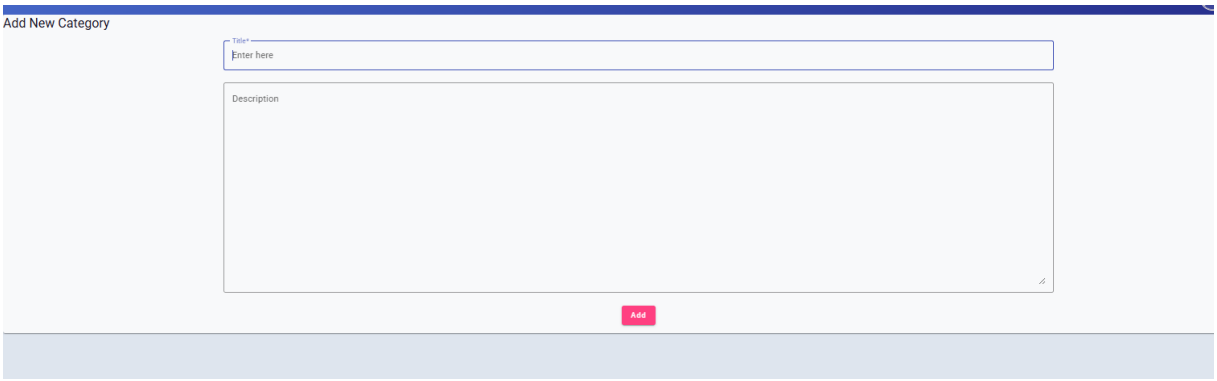
- Homepage



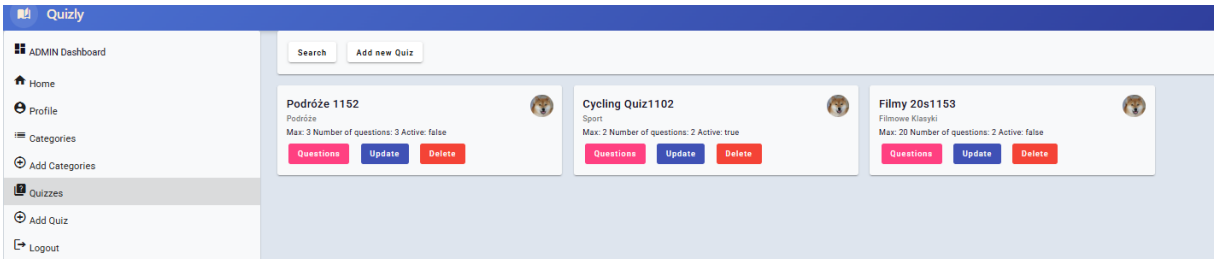
DLA ADMINA

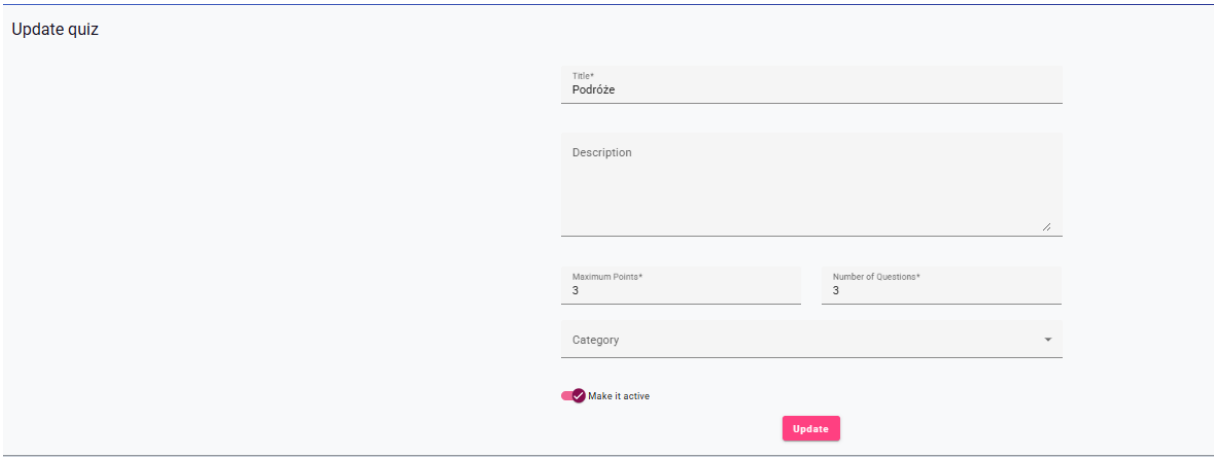
- Dodawanie/usuwanie kategorii

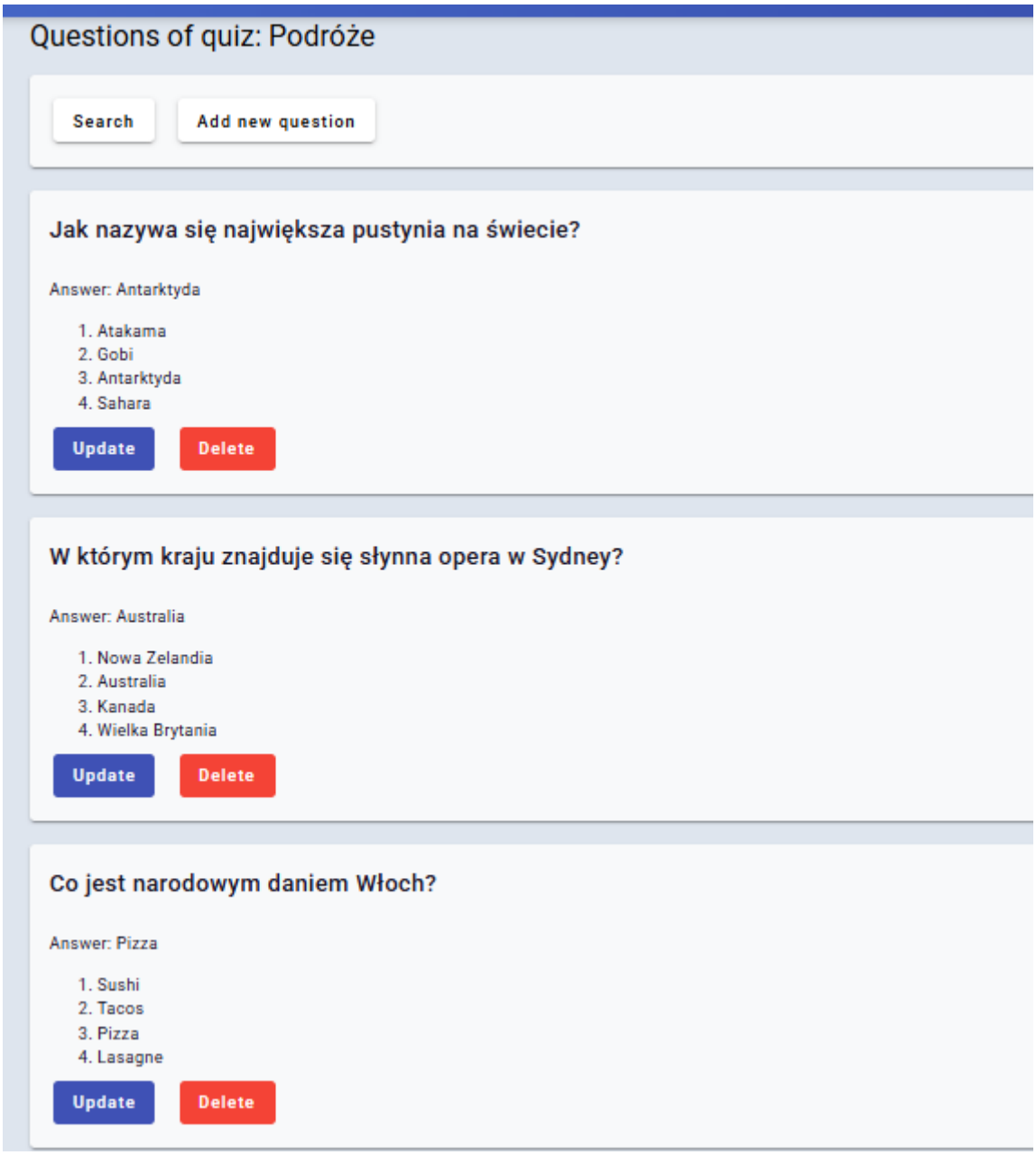




- Dodawanie/ usuwanie/ update’owanie Quizów


Quizly
ADMIN Dashboard
Home
Profile
Categories
Add Categories
Quizzes
Add Quiz
Logout
Search
Add new Quiz
Podróże 1152
Podróże
Max: 3 Number of questions: 3 Active: false
Questions
Update
Delete
Cycling Quiz1102
Sport
Max: 2 Number of questions: 2 Active: true
Questions
Update
Delete
Filmy 20s1153
Filmowe Klasyki
Max: 20 Number of questions: 2 Active: false
Questions
Update
Delete


Update quiz
Title*
Podróże
Description
Maximum Points*
3
Number of Questions*
3
Category
Make it active
Update



PODSUMOWANIE

Aplikacja webowa do zarządzania quizami to kompleksowe rozwiązanie, które integruje nowoczesny frontend z solidnym backendem i efektywną bazą danych. Zapewnia użytkownikom interaktywne doświadczenie, a administratorom narzędzia do zarządzania treścią i użytkownikami. Wykorzystanie nowoczesnych technologii gwarantuje wydajność, bezpieczeństwo i skalowalność aplikacji.