Programowanie aplikacji klient-serwer

Egzamin testowy (Temat 6)

Projekt

(wersja 1.1)

AUTORZY PROJEKTU:

Jakub Żmuda

Margarita Chirillova

Spis treści

1 Wstęp

1.1 Treść zadania

Aplikacja umożliwia tworzenie i przeprowadzanie egzaminu testowego. 3 rodzaje użytkowników: administrator, egzaminator i student. Administrator tworzy grupy studenckie i przypisuje do nich studentów. Egzaminator wprowadza egzamin testowy (wszystkie pytania zamknięte) podając dla wszystkich pytań: Treść pytania, Propozycje odpowiedzi (co najwyżej 5), Poprawną odpowiedź (tylko 1). Egzaminator przeprowadza egzamin udostępniając go wybranej grupie studentów na określony czas oraz uruchamia automatyczne sprawdzanie wyników testu dla poszczególnych studentów. Po sprawdzeniu wynik testu jest przypisywany poszczególnym studentom. Student loguje się, wprowadza odpowiedzi na poszczególne pytania i ogląda wyniki.

1.2 Dodatkowe założenia

1.2.1 Serwer

Serwer wykonany w języku „C”. Dane o klientach, plikach i egzaminach przechowywane w MongoDB, dokumentowej bazie danych. Serwer udostępnia metody według standardu REST.

1.2.2 Klient

Klient to aplikacja napisaną w React. React to aktualnie popularny framework Javascriptowy.Korzysta z udostępnionych przez serwer metod za pomocą protokołu HTTP.

Słownik pojęć

Klient - aplikacja frontowa napisana w react

Serwer - aplikacja backendowa realizująca połączenie z klientem i odczyt/zapis do bazy danych

Użytkownik - osoba korzystająca z serwisu

Rola - zakres możliwych funkcji użytkownika

Egzaminator - rola użytkownika udostępniająca podgląd listy studentów, tworzenie grup, egzaminów, wyników

Student - rola użytkownika pozwalająca na podgląd wyników oraz wypełnienie egzaminu

Administrator - rola użytkownika pozwalająca na zarządzanie użytkownikami i testami

2 Użyte protokoły

2.1 Protokół komunikacji klienta z serwerem

2.1.1 Wstęp

Protokołem komunikacji klient-server będzie dla nas HTTP. Jest to bardzo dobry protokół dla aplikacji internetowych, używany w znakomitej większości podobnych aplikacji.

2.1.1.1 Technologia

Do obsługi żądań http planujemy użyć biblioteki LibHTTP.

Funkcjonalność:

Protokół ten realizuje następującą funkcjonalność:

Protokół pozwala na ustrukturyzowaną komunikację klient-serwer. Pozwala na przykład łatwo przekazywać token sesji, przesyłać parametry żądań i zwracane przez serwer dane.

Lp	Treść komunikatu	Spodziewana reakcja
1	HTTP GET /users	server zwraca listę użytkowników
2	HTTP POST /exams	tworzenie egzaminu zgodnie z treścią zapytania
3	HTTP GET /groups	serwer zwraca listę grup

2.1.2 Dokładny opis funkcjonalny poszczególnych metod/komend

2.1.2.1

Sygnatura:

- HTTP method (GET/POST/PUT/EDIT)
- path (np. /users lub /exams)
- path params (np. /users?include=students)
- body params (np. {name: “Jan Kowalski”, role: “admin”})

Deklaracja (wartość zwracana, nazwa, parametry i ich znaczenie:

- Linia kodu statusu
- Jeden lub więcej nagłówek (header)
- Pusta linia oznaczająca koniec nagłówków
- Opcjonalnie ciało odpowiedzi

Semantyka:

The Hypertext Transfer Protocol (HTTP) to bezstanowy protokół warstwy aplikacji. Protokół ten jest połączeniowy, czyli działa wyłącznie z protokołem TCP/IP. Wiadomości są wymieniane na podstawie żądań i odpowiedzi zgodnych z deklaracją powyżej.

2.1.3 Spis komunikatów błędów

Nr błędu	Komunikat
404	Nie znaleziono
500	Wewnętrzny błąd serwera
400	Błędne zapytanie
401	Nieuwierzytelniony

3 Przypadki użycia

3.1.1 Klient tworzy nowego użytkownika w bazie serwera

Użytkownicy są predefiniowane i załadowane wprost do bazy za pomocą pliku .json

3.1.2 Klient loguje się do serwera

Logujemy się do panelu wpisując logini hasło w kliencie, następnie wykonywane jest zapytanie POST na endpoint ‘/login’ przesyłające do serwera dane, w odpowiedzi dostajemy token zaszyfrowany za pomocą base64 z informacjami o użytkowniku oraz jego sesji

3.1.3 Klient wylogowuje się z serwera

Klient wywołuje logout token sesji usuwamy tracimy możliwość odczytu roli i danych o użytkowniku w związku z tym nie możemy wyświetlać mu treści serwisu

3.1.4 Klient przegląda listę studentów

Użytkownik z odpowiednimi uprawnieniami(egzaminator i administrator) po zalogowaniu może obejrzeć listę studentów (userController)

3.1.5 Klient przegląda grupy

Użytkownik z odpowiednimi uprawnieniami(egzaminator i administrator) po zalogowaniu może obejrzeć listę istniejących grup (groupsController)

3.1.6 Klient tworzy grupy

Administrator może utworzyć grupę oraz przypisać do niej studentów(create group oraz assign students) dostępna również opcja usunięcia studentów z grupy (groupsController)

3.1.7 Klient tworzy i przegląda egzamin

Egzaminator ma możliwość utworzenia oraz przejrzenia egzaminu (examController)

4 Aplikacja klienta

4.1 Wstęp

Aplikacja klienta zostanie wykonana w React, framerowku Javascript. Framework ten jest popularny i wygodny dla szybkiego renderowania widoków i komunikacji z serwerem. W naszym przypadku będzie ona wykorzystywałam protokół http dla połączenia się oraz przesyłania danych do aplikacji serwera. Przewidywanych jest kilka endpointów dla komunikacji, przykładowe: /login, /test

4.2 Moduł logowania

Strona logowania zawiera formę i przycisk, pozwala na identyfikacje użytkownika

4.3 Moduł panelu użytkownika

W zależności od roli użytkownika pozwala na przeglądanie wyników/list studentów oraz grup lub egzaminów

5 Aplikacja serwera

5.1 Wstęp

Aplikacja serwera została wykonana w języku C, komunikuje się z aplikacją klienta za pomocą protokołu HTTP. Na serwerze wystawione są endpointy, na których serwer nasłuchuje ruch (np /login, /test). W przypadku otrzymania zapytań na tych przykładowych endpointach otrzymywane dane są przetwarzane i zapisywane w bazie danych.

5.2 Moduł zarządzania grupami

Użytkowniak o odpowiednich uprawnieniach(egzamintaroe/administrator) może stworzyć

grupę studentów lub edytować ją

5.3 Moduł zarządzania testami

Użytkownik o odpowiednich uprawnieniach może stworzyć test/egzamin, oraz udostępnić go grupie studentów

6 Przechowywane dane

6.1 Wstęp

W celu przechowywania informacji wybraliśmy MongoDB. Jest to dokumentowa baza danych, w której będziemy przechowywać informacje o użytkownikach, ich rolach, wykonanych testach oraz wynikach testów. Nasz wybór jest podyktowany elastycznością schematu danych oraz prostotą użycia.

6.2 Tabela/ plik1

Przykładowe dane użytkownika w bazie:

```
{

          "id": "1000",

          "name": "Albert
       Wright",

          "role": "admin",

          "login": "admin",

          "password": "admin1"

        },
```

6.3 Tabela/ plik2

Przykładowe dane niezbędne do dodania grupy do bazy danych, automatycznie tworzy się indeks

```
{

        "groupName": "nazwa nowej grupy",

       }
```

7 Interfejs danych

7.1 findUserByLogin

przyjmuje argument w postaci stringu i zwraca użytkownika o danym loginie z bazy

7.2 findAllStudents

Zwraca wszystkich użytkowników z bazy danych

7.3 createGroup

Tworzy nową grupę w bazie danych na podstawie przekazanych od klienta danych

7.4 assignStudentToGroup

przypisuje studentów do grupy na podstawie id grupy oraz id studenta

7.5 removeUserFromGroup

Usuwa wskazanego użytkownika z grupy

7.6 findAllGroups

zwraca listę istniejących grup

7.7 createExam

tworzy zapis z nowym egzaminem w bazie danych jako argument przyjmuje obiekt Json zawierający dane o zawartości egzaminu

7.8 findAllExams

zwraca listę istniejących egzaminów

7.9 findCompleteExams

zwraca studentom wypełnione egzaminy

7.10 Spis komunikatów błędów

Error Inserting Documents - wyświetla się przy błędnej próbie zapisania danych do bazy

8 Planowany podział prac

Aplikacja serwera	Jakub Żmuda/Margarita Chirillova
Aplikacja klienta	Jakub Żmuda/Margarita Chirillova

Baza danych	Jakub Żmuda/Margarita Chirillova
Konspekt projektu	Jakub Żmuda/Margarita Chirillova