DOKUMENTACJA PROJEKTU – NAUKA SZYBKIEGO PISANIA NA KLAWIATURZE

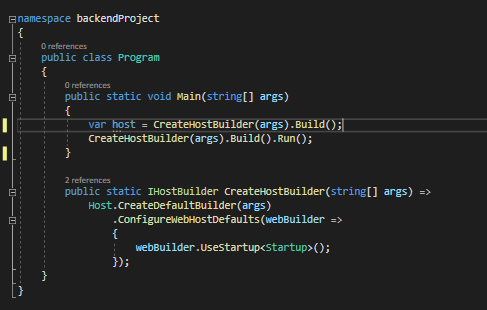
DOMINIK WIŚNIEWSKI | INFORMATYKA STOSOWANA V SEMESTR | INDEKS 113131

MICHAEL SZYCHULSKI | INFORMATYKA STOSOWANA V SEMESTR | INDEKS 113115

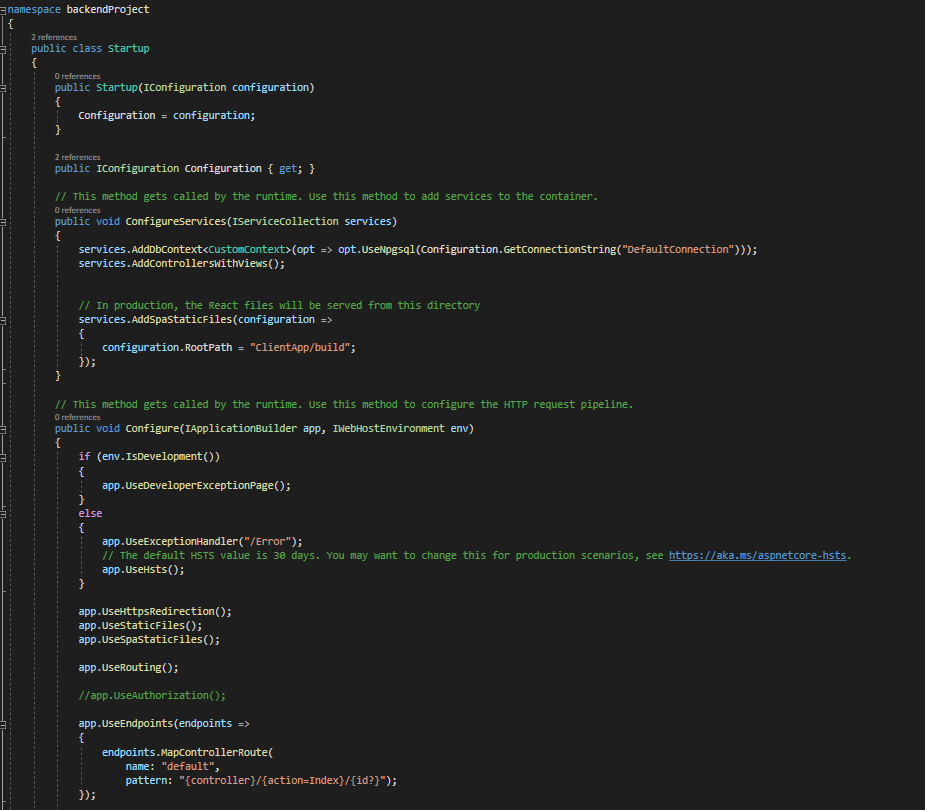
## Opis dostępnego API obsługiwanego przez backend w C#:

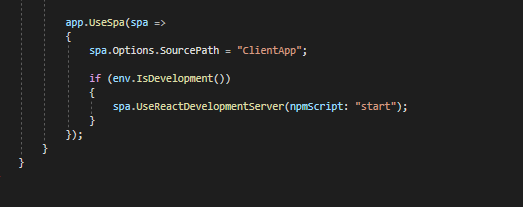
|  |  |  |
| --- | --- | --- |
| Składania zapytania: http://{localhost}:{port}/{model}/{x} Preambuła : http://{localhost}:{port}/ | Opis | Typ wyniku |
| Typ metody: GET  /Accounts | Pobranie wszystkich dostępnych kont | {{  "id": 1,  "username": "testowy",  "password": "test",  "last\_logged": "2021-01-12T18:30:10.47843",  "created": "2021-01-12T18:30:10.480785"  },{  "id": 2,  "username": "pierwszy",  "password": "TEST",  "last\_logged": "2021-01-12T18:58:11.251023",  "created": "2021-01-12T18:58:11.2531"  }} |
| Typ metody: GET  / Accounts /find?id={id} | Pobranie konta po jego numerze id | {JSON}  {  "id": 1,  "username": "testowy",  "password": "test",  "last\_logged": "2021-01-12T18:30:10.47843",  "created": "2021-01-12T18:30:10.480785"  } |
| Typ metody: POST  / Accounts /login  Request body:  {“username”:{username},  “password”:{password}} | Sprawdzenie czy dany użytkownik istnieje i czy jego hasło jest poprawne | Response 200 OK  Response 404 Not Found |
| Typ metody: POST  / Accounts /add  Request body:  {“username”:{username},  “password”:{password}} | Dodanie nowe konta użytkownika do serwisu | Response 200 OK  Response 404 Not Found |
| Typ metody: DELETE  / Accounts /delete?id={id} | Usuwa użytkownika o podanym ID | Response 200 OK  Response 404 Not Found |
| Typ metody: PUT  / Accounts /update?id={id}  Request body:  {“username”:{username},  “password”:{password}} | Zmienia Username i Password użytkownika o podanym ID | Response 200 OK  Response 404 Not Found |
| Typ metody: PUT  /Accounts/lastlogged | Aktualizuje czas ostatniego zalogowania użytkownika o podanym id na obecny | Response 200 OK  Response 404 Not Found |
| Typ metody: GET  /results/player?accId={accId} | Pobranie wszystkich wyników dla gracza o podanym id | {JSON} |
| Typ metody: GET  /games/find?id={id} | Pobranie gry o danym id | {JSON} |
| Typ metody: POST  /results/add  Request body:  {“gameID”:{ gameID },  “username”:{username},  “wordSpeed”:{ wordSpeed },  “mistakes”:{ mistakes}} | Zapisanie wyników danej gry. | Response 200 OK  Response 404 Not Found |
| Typ metody: GET  /results | Pobranie wszystkich wyników gier dla wszystkich użytkowników. | {JSON} |
| Typ metody: POST  results/player/criteria  {“accountID”:{ accountID },  “isPassed”:{ isPassed },  “startDate”:{ startDate },  “endDate”:{ endDate }} | Pobranie wszystkich wyników spełniających kryteria podane w request body | {JSON} |
| Typ metody: DELETE  /results/delete?id={id} | Usuń wynik o podanym id | Response 200 OK  Response 404 Not Found |
| Typ metody: GET  /results/scoretable | Pobranie wszystkich wyników w formacie dostosowanym jako wiersze tabeli wyników | {JSON}  Response 404 Not Found |
| Typ metody: GET  /results/scoretable/player?accId={accId} | Pobranie wszystkich wyników dla danego użytkownika w formacie dostosowanym jako wiersze tabeli wyników | {JSON}  Response 404 Not Found |
| Typ metody: GET  /results/scoretable/game?gameId={gameId} | Pobranie wszystkich wyników dla danej gry w formacie dostosowanym jako wiersze tabeli wyników | {JSON}  Response 404 Not Found |
| Typ metody: GET  /games | Pobranie wszystkich gier | {JSON} |
| Typ metody: POST  /games/add  Request body:  {“gameName”:{ gameName },  “textToWrite”:{ textToWrite },  “difficulty”:{ difficulty }} | Dodanie gry o podanych parametrach do bazy | Response 200 OK  Response 404 Not Found |
| Typ metody: GET  /games/find?id={id} | Pobranie gry o podanym id | {JSON} |
| Typ metody: DELETE  /games/delete?id={id} | Usunięcie gry o podanym id | Response 200 OK  Response 404 Not Found |
| Typ metody: PUT  /games/update?id={id}  Request body:  {“gameName”:{ gameName },  “textToWrite”:{ textToWrite },  “difficulty”:{ difficulty }} | Zaktualizowanie danych gry o podanym id według danych podanych w request body | Response 200 OK  Response 404 Not Found |
| Typ metody: GET  /writingtexts | Pobranie wszystkich zapisanych tekstów | {JSON} |
| Typ metody: POST  /writingtexts/add  Request body:  {“text”:{ text },  “source”:{source}} | Dodanie tekstu z podanymi parametrami | Response 200 OK  Response 404 Not Found |
| Typ metody: GET  /writingtexts/find?id={id} | Pobranie tekstu o podanym id | {JSON} |
| Typ metody: DELETE  /writingtexts/delete?id={d} | Usunięcie tekstu o podanym id | Response 200 OK  Response 404 Not Found |
| Typ metody: PUT  / writingtexts /update?id={id}  Request body:  {“text”:{ text },  “source”:{source}} | Zaktualizowanie tekstu o podanym id według podanych parametrów | Response 200 OK  Response 404 Not Found |
| Typ metody: PUT  / writingtexts /updateSpeeds?id={id} | Zaktualizowanie TopSpeed I AvarageSpeed dla tekstu o podanym id | Response 200 OK  Response 404 Not Found |
| Typ metody: GET  /custom/get/{words} | Pobranie {words} losowych słów, gdzie {words} to liczba słów które chcemy otrzymać | {JSON} |

Klasa Program i metoda Main. Inicjalizuje ona cały rozruch aplikacji.



Klasa Startup. Odpowiada ona za całą konfigurację backendu aplikacji.

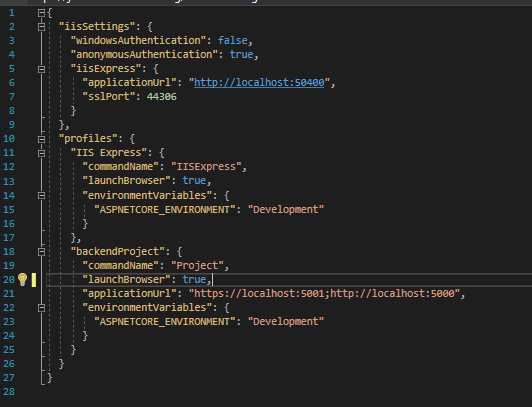




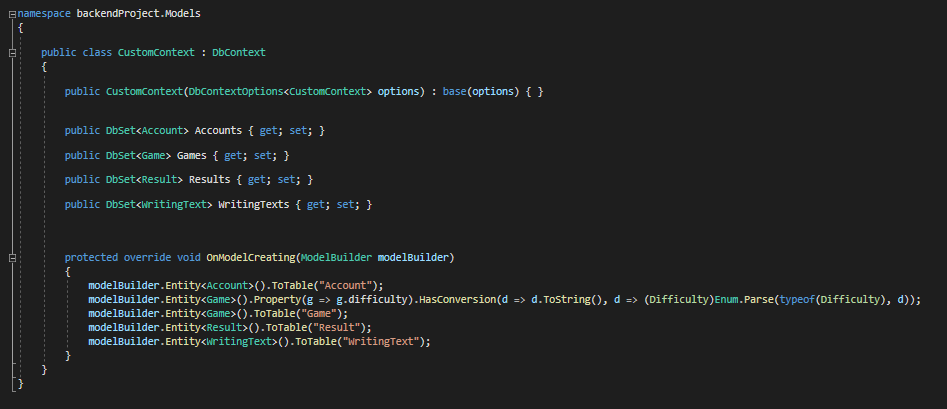
Jako bazy danych użyta została baza PostgreSQL. Tworzymy Context połączenia z bazą danych dzięki metodzie:

services.AddDbContext<CustomContext>(opt => opt.UseNpgsql(Configuration.GetConnectionString("DefaultConnection")));

Dzięki pakietom NuGet – EntityFrameworkCore i Npsql dla PostgreSQL – Context utworzony dla połączenia podanego w ustawieniach „DefaultConnection”:



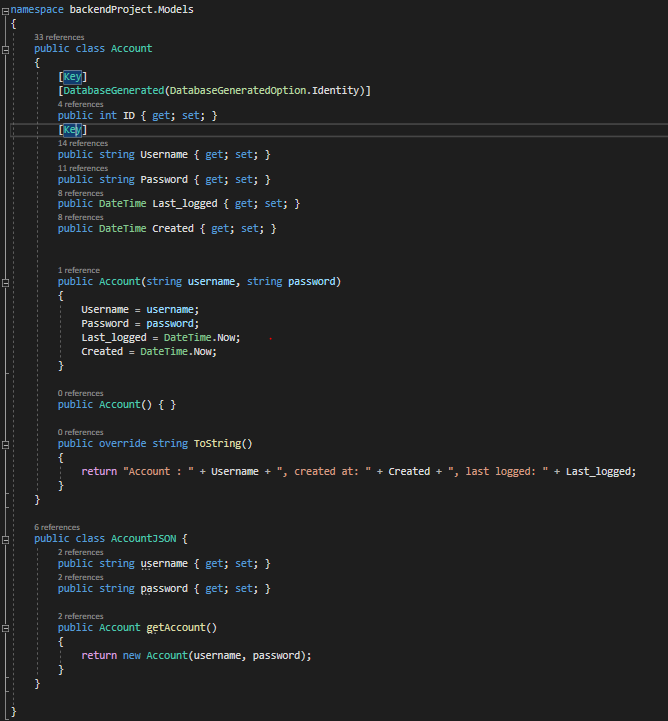
Służy on do zarządzania zmianami w bazie danych ze strony serwisu. Tworzone są obiekty DbSet<T>, który każdy służy do reprezentowania stanu powiązanej tabeli w bazie danych. Poniżej klasa CustomContext wykorzystana w projekcie. Każda tabela ma swój odpowiednik DbSet. Metoda OnModelCreating zapewnia utworzenie tabel oraz ich ustawień.



Klasa modelu Account. Jej pola to:

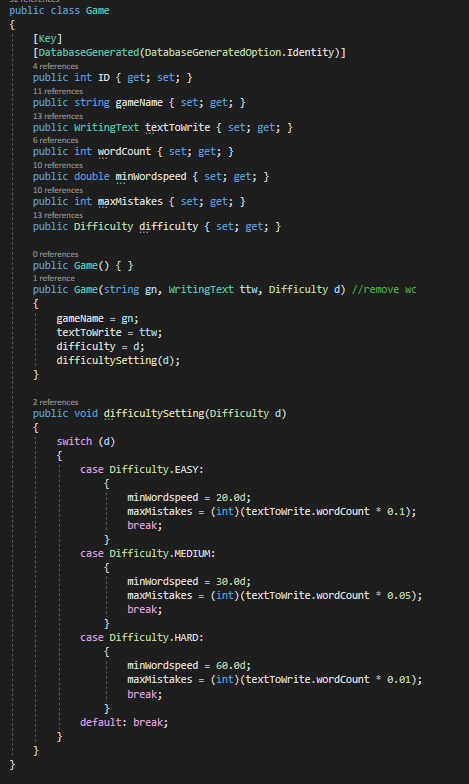
* ID – Unikalny klucz główny modelu, automatycznie generowany dla każdego kolejnego zapisanego w bazie obiektu
* Username – Unikalna nazwa użytkownika
* Password – Hasło za pomocą którego użytkownik loguje się do aplikacji
* Last\_logged – Data ostatniego zalogowania się użytkownika
* Created – Data utworzenia konta użytkownika

Posiada ona podklasę AccountJSON utworzoną w celu łatwiejszego serializowania i deserializowania danych z i do frontu aplikacji.



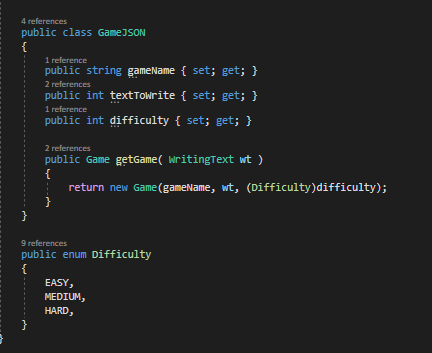
Klasa modelu Game. Posiada ona pola:

* ID - Unikalny klucz główny modelu, automatycznie generowany dla każdego kolejnego zapisanego w bazie obiektu
* gameName – Nazwa gry
* textToWrite – powiązanie Tekstu do danej gry. W bazie danych jest to klucz obcy do tabeli WritingText
* wordCount – liczba słów w tekscie
* minWordSpeed – minimalna wartość słów na minute potrzebna do ukończenia gry
* maxMistakes – maksymalna ilość błędów popełnionych w trakcie gry
* difficulty – stopnień trudności gry. Na jego podstawie ustalane są wartości minWordSpeed i maxMistakes dzięki metodzie difficultySetting()



Posiada ona podklasę GameJSON utworzoną w celu łatwiejszego serializowania i deserializowania danych z i do frontu aplikacji.

Dodatkowo enum Difficulty ze stopniami trudności.

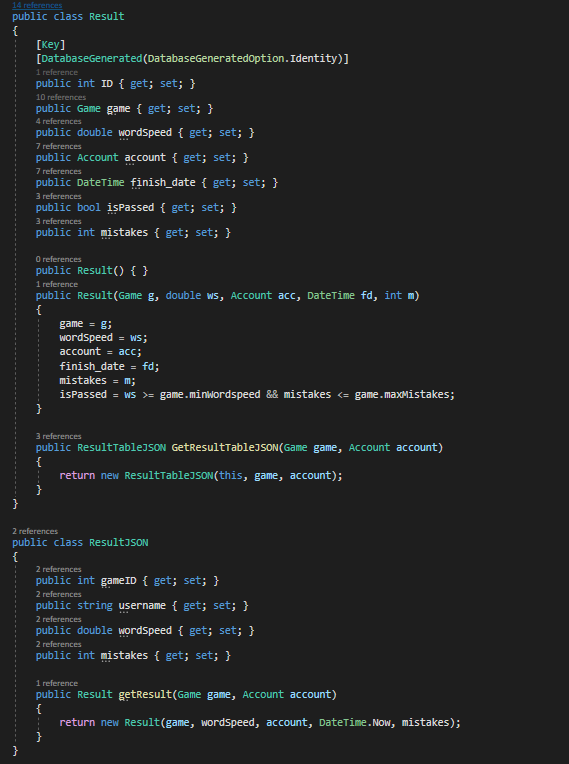


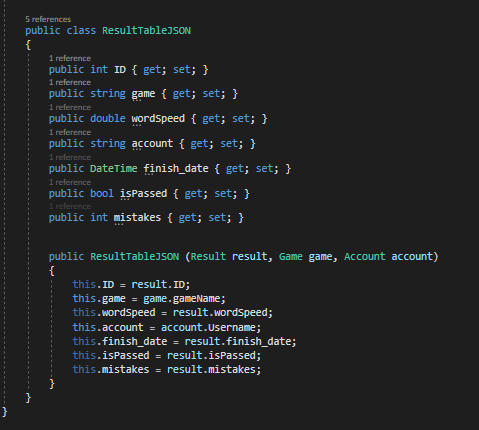
Klasa modelu Result. Posiada ona pola:

* ID – Unikalny klucz główny modelu, automatycznie generowany dla każdego kolejnego zapisanego w bazie obiektu
* Game – Klucz obcy do tabeli Game
* wordSpeed – uzyskana szybkość słów na minute na końcu gry
* account – klucz obcy do tabeli Account
* finish\_date – data ukończenia gry
* isPassed – osądzenie czy wordSpeed i mistakes były dostateczne do ukończenia gry z wynikiem pozytywnym
* mistakes – liczba błędów

Posiada również 3 podklasy:

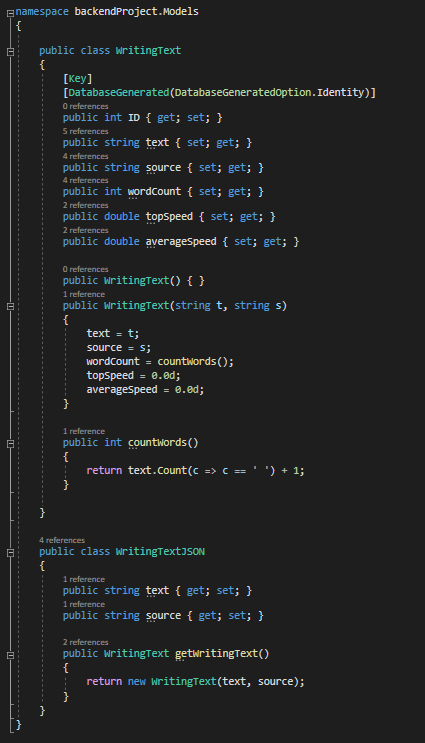
* ResultJSON – klasa do łatwej serializacji i deserializacji całego obiektu Result
* ResultSearchJSON – klasa do obsługi parametrów przekazywanych do filtrowania wyników
* ResultTableJSON – klasa służąca do mapowania obiektów Result na formę gotową do zapełnienia tabeli wyników.





Klasa modelu WritingText. Posiada ona pola:

* ID – Unikalny klucz główny modelu, automatycznie generowany dla każdego kolejnego zapisanego w bazie obiektu
* Text – Tekst który użytkownik będzie pisać
* Source – źródło tesktu
* wordCount – liczba słów w tekście
* topSpeed – najwyższa uzyskana szybkość pisania
* averageSpeed – średnia uzyskana szybkość pisania



Posiada ona również podklasę WritingTextJSON do łatwej serializacji i deserializacji obiektów WritingText.

Do komunikacji z serwisem wykorzystujemy Controllery. Służą one do obsługi zapytań poprzez end-pointy. Lista dostępnych end-pointów i ich opis znajduję się w tabeli API.

