



Uniwersytet Technologiczno – Przyrodniczy



DOKUMENTACJA PROJEKTU – NAUKA SZYBKIEGO PISANIA NA KLAWIATURZE

DOMINIK WIŚNIEWSKI | INFORMATYKA STOSOWANA V SEMESTR | INDEKS 113131

MICHAEL SZYCHULSKI | INFORMATYKA STOSOWANA V SEMESTR | INDEKS 113115

Cała aplikacja dostępna na:

<https://github.com/plywakd/WindowsProject>

Główne założenia realizacji projektu z wykorzystaniem języka C#:

Zamysłem projektowym aplikacji było utworzenie w pełni działającej aplikacji webowej za wykorzystaniem platformy ASP.NET. Dzięki temu rozwiązaniu, wytworzony został szkielet aplikacji backendowej, obsługiwanej w pełni przez język C#, i aplikacja kliencka wykonana w Javascript z pomocą frameworku React.

Aplikacja startowo jest na porcie 44306 z protokołem https i 50400 z protokołem http.

Nauka szybkiego pisania na klawiaturze odbywa się poprzez zjawisko „grywalizacji”. Nasz projekt zakłada, że chęć uzyskiwania coraz lepszych wyników w tabeli rankingowych, będzie motywatorem do dalszego pisania tekstów i tym samym nauki szybkiego pisania.

Na podstawie projektu sprawdziliśmy działanie współpracy backendu C# poprzez ASP.NET oraz Entity Framework do zarządzania połączeniem z warstwą frontendu i wysyłaniem danych pomiędzy nimi i przekazywaniem ich do bazy danych.

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	<pre>{ "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" }</pre>
Typ metody: GET /Accounts/find?id={id}	Pobranie konta po jego numerze id	<pre>{ "id": 1, "username": "testowy", "password": "test",</pre>



		<pre>"last_logged": "2021-01-12T18:30:10.47843", "created": "2021-01-12T18:30:10.480785" }</pre>
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 201 Created 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	<pre>{ "id": 2, "game": null, "wordSpeed": 10, "account": { "id": 3, "username": "drugi", "password": "test", "last_logged": "2021-01-12T18:59:22.412457", "created": "2021-01-12T18:59:22.412477" }, "finish_date": "2021-01-16T17:23:37.327277", "isPassed": false, "mistakes": 1 }, { "id": 3, "game": null, "wordSpeed": 107.33, "account": { "id": 3, "username": "drugi", "password": "test", "last_logged": "2021-01-12T18:59:22.412457", "created": "2021-01-12T18:59:22.412477" } }</pre>



		<pre> }, "finish_date": "2021-01- 16T18:04:18.098837", "isPassed": true, "mistakes": 0 }, }</pre>
Typ metody: GET /games/find?id={id}	Pobranie gry o danym id	<pre>{ "id": 2, "gameName": "tryme", "textToWrite": { "id": 2, "text": "to jest tekst który musimy wrzucic do bazy danych ze spacjami.", "source": "moja glowa", "wordCount": 11, "topSpeed": 0, "averageSpeed": 0 }, "wordCount": 0, "minWordspeed": 20, "maxMistakes": 1, "difficulty": 0 }</pre>
Typ metody: POST /results/add Request body: { "gameID": { gameID }, "username": { username }, "wordSpeed": { wordSpeed }, "mistakes": { mistakes } }	Utworzenie wyniku użytkownika o podanym username, dla danego ID gry, wysyłane są statystyki, czyli prędkość wpisywanych słów oraz liczba pomyłek	Response 201 Created Response 404 Not Found
Typ metody: GET /results	Pobranie wszystkich wyników gier dla wszystkich użytkowników.	<pre>{ "id": 10, "game": { "id": 2, "gameName": "tryme", "textToWrite": null, "wordCount": 0, "minWordspeed": 20, "maxMistakes": 1, "difficulty": 0 }, "wordSpeed": 87.93, "account": { "id": 3, "username": "drugi", "password": "test", "last_logged": "2021-01- 12T18:59:22.412457", "created": "2021-01- 12T18:59:22.412477" }, "finish_date": "2021-01- 17T01:13:20.739607", "isPassed": true, "mistakes": 0 }, { "id": 11, "game": { "id": 8, "gameName": "Lorem welcome", </pre>



Uniwersytet Technologiczno – Przyrodniczy

		<pre>"textToWrite": null, "wordCount": 0, "minWordspeed": 20, "maxMistakes": 6, "difficulty": 0 }, "wordSpeed": 44.58, "account": { "id": 3, "username": "drugi", "password": "test", "last_logged": "2021-01- 12T18:59:22.412457", "created": "2021-01- 12T18:59:22.412477" }, "finish_date": "2021-01- 17T01:38:25.282066", "isPassed": true, "mistakes": 0 }</pre>
<p>Typ metody: POST results/player/criteria</p> <p>{ "accountID": { accountID }, "isPassed": { isPassed }, "startDate": { startDate }, "endDate": { endDate } }</p>	<p>Pobranie wszystkich wyników spełniających kryteria podane w request body</p>	<pre>{ "id": 1643, "game": "Game Name 104", "wordSpeed": 72.95795824330204, "account ": "User1", "finish_date": "2021-01- 14T00:00:00", "isPassed": false, "mistakes": 5 } , { "id": 1026, "game": "Game Name 192", "wordSpeed": 35.68951470390405, "account ": "User1", "finish_date": "2020-12- 23T00:00:00", "isPassed": false, "mistakes": 9 } , { "id": 1517, "game": "Game Name 161", "wordSpeed": 37.61114346264449, "account ": "User1", "finish_date": "2020-11- 01T00:00:00", "isPassed": false, "mistakes": 6 } , { "id": 1094, "game": "Game Name 63", "wordSpeed": 52.03203132005037, "account ": "User1", "finish_date": "2020-08- 22T00:00:00", "isPassed": false, "mistakes": 4 } , { "id": 1590, "game": "Game Name 105", "wordSpeed": 71.54519159791302, "account ": "User1", "finish_date": "2020-07- 25T00:00:00", "isPassed": false, "mistakes": 6 } , { "id": 1996, "game": "Game Name 212", "wordSpeed": 59.70471355072442, "account ": "User1", "finish_date": "2020-07- 10T00:00:00", "isPassed": false, "mistakes": 7 } , { "id": 1131, "game": "Game Name 140", "wordSpeed": 46.57670505604553, "account ": "User1", "finish_date": "2020-06- 23T00:00:00", "isPassed": false, "mistakes": 3 }</pre>
<p>Typ metody: DELETE /results/delete?id={id}</p>	<p>Usuń wynik o podanym id</p>	<p>Response 200 OK Response 404 Not Found</p>
<p>Typ metody: GET /results/scoretable</p>	<p>Pobranie wszystkich wyników w formacie dostosowanym jako wiersze tabeli wyników</p>	<pre>{ "id": 12, "game": "Lorem welcome", "wordSpeed": 54.64, "account": "drugi", "finish_date": "2021-01- 17T13:03:31.156853", "isPassed": true, "mistakes": 0 }, { "id": 11,</pre>



Uniwersytet Technologiczno – Przyrodniczy

		<pre>"game": "Lorem welcome", "wordSpeed": 44.58, "account": "drugi", "finish_date": "2021-01- 17T01:38:25.282066", "isPassed": true, "mistakes": 0 }, Response 404 Not Found</pre>
Typ metody: GET /results/scoretable/player?acclId={acclId}	Pobranie wszystkich wyników dla danego użytkownika w formacie dostosowanym jako wiersze tabeli wyników	<pre>{{ "id": 1, "game": "testGame", "wordSpeed": 12, "account": "pierwszy", "finish_date": "2021-01- 12T23:48:39.281193", "isPassed": false, "mistakes": 0 } Response 404 Not Found</pre>
Typ metody: GET /results/scoretable/game?gameId={gameId}	Pobranie wszystkich wyników dla danej gry w formacie dostosowanym jako wiersze tabeli wyników	<pre>{ "id": 10, "game": "tryme", "wordSpeed": 87.93, "account": "drugi", "finish_date": "2021-01- 17T01:13:20.739607", "isPassed": true, "mistakes": 0 }, { "id": 9, "game": "tryme", "wordSpeed": 53.86, "account": "drugi", "finish_date": "2021-01- 17T01:13:08.444502", "isPassed": true, "mistakes": 0 }, Response 404 Not Found</pre>
Typ metody: GET /games	Pobranie listy wszystkich dostępnych obiektów gier wraz z powiązanymi obiektami tekstów, który zawiera dodatkowe informacje	<pre>{ "id": 1, "gameName": "testGame", "textToWrite": { "id": 1, "text": "testowy_tekst_do_gry_bo_tak_trzeba", "source": "ja", "wordCount": 1, "topSpeed": 0, "averageSpeed": 0 }, "wordCount": 0, "minWordspeed": 20, "maxMistakes": 0, "difficulty": 0 }, { "id": 7, "gameName": "tryme7", "textToWrite": { "id": 2,</pre>



Uniwersytet Technologiczno – Przyrodniczy

		<pre>"text": "to jest tekst ktory musimy wrzucic do bazy danych ze spacjami.", "source": "moja glowa", "wordCount": 11, "topSpeed": 0, "averageSpeed": 0 }, "wordCount": 11, "minWordspeed": 60, "maxMistakes": 0, "difficulty": 2 } }</pre>
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	{ "id": 2, "gameName": "tryme", "textToWrite": { "id": 2, "text": "to jest tekst ktory musimy wrzucic do bazy danych ze spacjami.", "source": "moja glowa", "wordCount": 11, "topSpeed": 0, "averageSpeed": 0 }, "wordCount": 0, "minWordspeed": 20, "maxMistakes": 1, "difficulty": 0 }
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	{ "id": 2, "text": "to jest tekst ktory musimy wrzucic do bazy danych ze spacjami.", "source": "moja glowa", "wordCount": 11, "topSpeed": 0, "averageSpeed": 0 }, {



Uniwersytet Technologiczno – Przyrodniczy

		<pre>"id": 3, "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. "source": "lorem ipsum", "wordCount": 69, "topSpeed": 0, "averageSpeed": 0 }</pre>
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	{ "id": 2, "text": "to jest tekst ktory musimy wrzucic do bazy danych ze spaczami.", "source": "moja glowa", "wordCount": 11, "topSpeed": 0, "averageSpeed": 0 }
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 AverageSpeed 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	["determiner", "descendible", "brusk", "mournings", "calderas", "claystones", "macks", "ganglion", "ruffled", "swearword"]



Opis kodu całej aplikacji ASP.NET:

Klasa Program i metoda Main. Inicjalizuje ona cały rozruch aplikacji. Uruchamiana jest od razu aplikacja po stronie backendu i aplikacja klienta, dostępne pod localhostem dla protokołów http/https o określonych portach na samym początku dokumentacji.

```
namespace backendProject
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateHostBuilder(args).Build();
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
        }
    }
}
```

Klasa Startup. Odpowiada ona za całą konfigurację backendu aplikacji i ustanawia połączenie z bazą danych.

```
namespace backendProject
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<CustomContext>(opt => opt.UseNpgsql(Configuration.GetConnectionString("DefaultConnection")));
            services.AddControllersWithViews();

            // In production, the React files will be served from this directory
            services.AddSpaStaticFiles(configuration =>
            {
                configuration.RootPath = "ClientApp/build";
            });
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseSpaStaticFiles();

            app.UseRouting();

            //app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller}/{action=Index}/{id?}");
            });
        }
    }
}
```



```

app.UseSpa(spa =>
{
    spa.Options.SourcePath = "clientApp";

    if (env.IsDevelopment())
    {
        spa.UseReactDevelopmentServer(npmScript: "start");
    }
});
}

```

Tworzony jest Context, który odpowiada za połączenie z bazą danych dzięki metodzie:

```

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

```

Poprzez pakiety NuGet – EntityFrameworkCore i Npgsql dla PostgreSQL – Context utworzony dla połączenia podanego w ustawieniach „DefaultConnection”:

```

1  {
2      "iisSettings": {
3          "windowsAuthentication": false,
4          "anonymousAuthentication": true,
5          "iisExpress": {
6              "applicationUrl": "http://localhost:50400",
7              "sslPort": 44306
8          }
9      },
10     "profiles": {
11         "IIS Express": {
12             "commandName": "IISExpress",
13             "launchBrowser": true,
14             "environmentVariables": {
15                 "ASPNETCORE_ENVIRONMENT": "Development"
16             }
17         },
18         "backendProject": {
19             "commandName": "Project",
20             "launchBrowser": true,
21             "applicationUrl": "https://localhost:5001;http://localhost:5000",
22             "environmentVariables": {
23                 "ASPNETCORE_ENVIRONMENT": "Development"
24             }
25         }
26     }
27 }
28

```

Służą one 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.

```
namespace backendProject.Models
{
    public class CustomContext : DbContext
    {
        public CustomContext(DbContextOptions<CustomContext> options) : base(options) { }

        public DbSet<Account> Accounts { get; set; }
        public DbSet<Game> Games { get; set; }
        public DbSet<Result> Results { get; set; }
        public DbSet<WritingText> WritingTexts { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Account>().ToTable("Account");
            modelBuilder.Entity<Game>().Property(g => g.difficulty).HasConversion(d => d.ToString(), d => (Difficulty)Enum.Parse(typeof(Difficulty), d));
            modelBuilder.Entity<Game>().ToTable("Game");
            modelBuilder.Entity<Result>().ToTable("Result");
            modelBuilder.Entity<WritingText>().ToTable("WritingText");
        }
    }
}
```

Opis wykorzystywanych modeli i ich metod:

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.

```
namespace backendProject.Models
{
    33 references
    public class Account
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        4 references
        public int ID { get; set; }

        14 references
        public string Username { get; set; }
        11 references
        public string Password { get; set; }
        8 references
        public DateTime Last_logged { get; set; }
        8 references
        public DateTime Created { get; set; }

        1 reference
        public Account(string username, string password)
        {
            Username = username;
            Password = password;
            Last_logged = DateTime.Now;
            Created = DateTime.Now;
        }

        0 references
        public Account() { }

        0 references
        public override string ToString()
        {
            return "Account : " + Username + ", created at: " + Created + ", last logged: " + Last_logged;
        }
    }

    6 references
    public class AccountJSON {
        2 references
        public string username { get; set; }
        2 references
        public string password { get; set; }

        2 references
        public Account getAccount()
        {
            return new Account(username, password);
        }
    }
}
```



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 tekście
- minWordSpeed – minimalna wartość słów na minutę potrzebna do ukończenia gry
- maxMistakes – maksymalna ilość błędów popełnionych w trakcie gry
- difficulty – stopień trudności gry. Na jego podstawie ustalane są wartości minWordSpeed i maxMistakes dzięki metodzie difficultySetting()

```
public class Game
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    4 references
    public int ID { get; set; }
    11 references
    public string gameName { get; set; }
    13 references
    public WritingText textToWrite { get; set; }
    6 references
    public int wordCount { get; set; }
    10 references
    public double minWordSpeed { get; set; }
    10 references
    public int maxMistakes { get; set; }
    13 references
    public Difficulty difficulty { get; set; }

    0 references
    public Game() { }
    1 reference
    public Game(string gn, WritingText ttw, Difficulty d) //remove wc
    {
        gameName = gn;
        textToWrite = ttw;
        difficulty = d;
        difficultySetting(d);
    }

    2 references
    public void difficultySetting(Difficulty d)
    {
        switch (d)
        {
            case Difficulty.EASY:
            {
                minWordSpeed = 20.0d;
                maxMistakes = (int)(textToWrite.wordCount * 0.1);
                break;
            }
            case Difficulty.MEDIUM:
            {
                minWordSpeed = 30.0d;
                maxMistakes = (int)(textToWrite.wordCount * 0.05);
                break;
            }
            case Difficulty.HARD:
            {
                minWordSpeed = 60.0d;
                maxMistakes = (int)(textToWrite.wordCount * 0.01);
                break;
            }
            default: break;
        }
    }
}
```



Uniwersytet Technologiczno – Przyrodniczy

Posiada ona podklasę GameJSON utworzoną w celu łatwiejszego serializowania i deserializowania danych z i do frontu aplikacji. Dodatkowo enum Difficulty ze stopniami trudności.

```
4 references
public class GameJSON
{
    1 reference
    public string gameName { set; get; }
    2 references
    public int textToWrite { set; get; }
    1 reference
    public int difficulty { set; get; }

    2 references
    public Game getGame( WritingText wt )
    {
        return new Game(gameName, wt, (Difficulty)difficulty);
    }
}

9 references
public enum Difficulty
{
    EASY,
    MEDIUM,
    HARD,
}
}
```

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

```
14 references
public class Result
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    1 reference
    public int ID { get; set; }
    10 references
    public Game game { get; set; }
    4 references
    public double wordSpeed { get; set; }
    7 references
    public Account account { get; set; }
    7 references
    public DateTime finish_date { get; set; }
    3 references
    public bool isPassed { get; set; }
    1 reference
    public int mistakes { get; set; }

    0 references
    public Result() { }
    1 reference
    public Result(Game g, double ws, Account acc, DateTime fd, int m)
    {
        game = g;
        wordSpeed = ws;
        account = acc;
        finish_date = fd;
        mistakes = m;
        isPassed = ws >= game.minWordspeed && mistakes <= game.maxMistakes;
    }

    3 references
    public ResultTableJSON GetResultTableJSON(Game game, Account account)
    {
        return new ResultTableJSON(this, game, account);
    }
}

2 references
public class ResultJSON
{
    2 references
    public int gameID { get; set; }
    2 references
    public string username { get; set; }
    2 references
    public double wordSpeed { get; set; }
    2 references
    public int mistakes { get; set; }

    1 reference
    public Result getResult(Game game, Account account)
    {
        return new Result(game, wordSpeed, account, DateTime.Now, mistakes);
    }
}
```



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 wypełnienia tabeli wyników.

```
5 references
public class ResultTableJSON
{
    1 reference
    public int ID { get; set; }
    1 reference
    public string game { get; set; }
    1 reference
    public double wordSpeed { get; set; }
    1 reference
    public string account { get; set; }
    1 reference
    public DateTime finish_date { get; set; }
    1 reference
    public bool isPassed { get; set; }
    1 reference
    public int mistakes { get; set; }

    public ResultTableJSON (Result result, Game game, Account account)
    {
        this.ID = result.ID;
        this.game = game.gameName;
        this.wordSpeed = result.wordSpeed;
        this.account = account.Username;
        this.finish_date = result.finish_date;
        this.isPassed = result.isPassed;
        this.mistakes = result.mistakes;
    }
}
```

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 tekstu
- wordCount – liczba słów w tekście
- topSpeed – najwyższa uzyskana szybkość pisania
- averageSpeed – średnia uzyskana szybkość pisania

```
namespace backendProject.Models
{
    public class WritingText
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        0 references
        public int ID { get; set; }
        5 references
        public string text { set; get; }
        4 references
        public string source { set; get; }
        4 references
        public int wordCount { set; get; }
        2 references
        public double topSpeed { set; get; }
        2 references
        public double averageSpeed { set; get; }

        0 references
        public WritingText() { }
        1 reference
        public WritingText(string t, string s)
        {
            text = t;
            source = s;
            wordCount = countWords();
            topSpeed = 0.0d;
            averageSpeed = 0.0d;
        }

        1 reference
        public int countWords()
        {
            return text.Count(c => c == ' ') + 1;
        }
    }
}
```



Uniwersytet Technologiczno – Przyrodniczy

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

```
4 references
public class WritingTextJSON
{
    1 reference
    public string text { get; set; }
    1 reference
    public string source { get; set; }

    2 references
    public WritingText getWritingText()
    {
        return new WritingText(text, source);
    }
}
```

Opis działania kontrolerów serwisu do komunikacji:

Do komunikacji z serwisem wykorzystujemy `Controllery`. Służą one do obsługi zapytań poprzez end-pointy. Lista dostępnych end-pointów i ich opis wraz z przykładowymi wynikami znajdują się w tabeli API. Jak na aplikację webową API jest wystawione przez serwer i do tych adresów wysyłane są zapytania, które obsługują wspomniane wcześniej kontrolery.

```
namespace backendProject.Controllers
{
    [ApiController]
    [Route("[controller]")]
    1 reference
    public class AccountsController : ControllerBase
    {
        private readonly CustomContext _context;

        0 references
        public AccountsController(CustomContext context)
        {
            _context = context;
        }

        0 references
        public ActionResult<IEnumerable<Account>> Index()
        {
            return _context.Accounts.ToList();
        }

        [HttpGet("/[controller]/find")]
        0 references
        public ActionResult<Account> FindById(int id)
        {
            return _context.Accounts.Find(id);
        }

        [HttpPost("/[controller]/login")]
        0 references
        public async Task<ActionResult> LogInUserAsync()
        {
            try
            {
                using (StreamReader stream = new StreamReader(HttpContext.Request.Body))
                {
                    string body = await stream.ReadToEndAsync();
                    AccountJSON accJSON = JsonSerializer.Deserialize<AccountJSON>(body);
                    Account acc = _context.Accounts.Where(s => s.Username.Equals(accJSON.username)).FirstOrDefault();
                    if (acc != null && acc.Password.Equals(accJSON.password))
                    {
                        return StatusCode(200);
                    }
                    return StatusCode(404);
                }
            } catch { return StatusCode(404); }
        }
    }
}
```

```
[HttpPost("/[controller]/add")]
0 references
public async Task<ActionResult> AddAsync()
{
    try
    {
        using (StreamReader stream = new StreamReader(HttpContext.Request.Body))
        {
            string body = await stream.ReadToEndAsync();
            AccountJSON accJSON = JsonSerializer.Deserialize<AccountJSON>(body);
            Account acc = accJSON.getAccount();
            _context.Accounts.Add(acc);
            _context.SaveChanges();
            return StatusCode(200);
        }
    }
    catch (Exception e)
    {
        return StatusCode(404);
    }
}

[HttpDelete("/[controller]/delete")]
0 references
public ActionResult Delete(int id)
{
    try
    {
        Account acc = _context.Accounts.Find(id);
        _context.Accounts.Remove(acc);
        _context.SaveChanges();
        return StatusCode(200);
    }
    catch
    {
        return StatusCode(404);
    }
}
```

```
[HttpPut("/[controller]/update")]
0 references
public async Task<ActionResult> UpdateAsync(int id)
{
    try
    {
        using (StreamReader stream = new StreamReader(HttpContext.Request.Body))
        {
            string body = await stream.ReadToEndAsync();
            AccountJSON accJSON = JsonSerializer.Deserialize<AccountJSON>(body);
            Account acc = accJSON.getAccount();
            Account newAcc = _context.Accounts.Find(id);
            if (acc.Username != "") newAcc.Username = acc.Username;
            if (acc.Password != "") newAcc.Password = acc.Password;
            _context.Accounts.Update(newAcc);
            _context.SaveChanges();
            return StatusCode(200);
        }
    }
    catch (Exception e)
    {
        return StatusCode(404);
    }
}
```

```
namespace backendProject.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class GameController : ControllerBase
    {
        private readonly CustomContext _context;

        public GameController(CustomContext context)
        {
            _context = context;
        }

        public ActionResult<IEnumerable<Game>> Index()
        {
            try
            {
                return _context.Games.Include(game => game.textToWrite).ToList();
            } catch { return StatusCode(404); }
        }

        [HttpPost("/[controller]/add")]
        public async Task<ActionResult> AddAsync()
        {
            try
            {
                using (StreamReader stream = new StreamReader(HttpContext.Request.Body))
                {
                    string body = await stream.ReadToEndAsync();
                    GameJSON gameJSON = JsonSerializer.Deserialize<GameJSON>(body);
                    Game game = gameJSON.getGame(_context.WritingTexts.Find(gameJSON.textToWrite));
                    _context.Games.Add(game);
                    _context.SaveChanges();
                    return StatusCode(200);
                }
            } catch { return StatusCode(404); }
        }
    }
}
```

```
[HttpGet("/[controller]/find")]
public ActionResult<Game> Find(int id)
{
    return _context.Games.Find(id);
}

[HttpDelete("/[controller]/delete")]
public ActionResult Delete(int id)
{
    try
    {
        Game game = _context.Games.Find(id);
        _context.Games.Remove(game);
        _context.SaveChanges();
        return StatusCode(200);
    } catch { return StatusCode(404); }
}

[HttpPut("/[controller]/update")]
public async Task<ActionResult> UpdateAsync(int id)
{
    try
    {
        using (StreamReader stream = new StreamReader(HttpContext.Request.Body))
        {
            string body = await stream.ReadToEndAsync();
            GameJSON gameJSON = JsonSerializer.Deserialize<GameJSON>(body);
            Game game = gameJSON.getGame(_context.WritingTexts.Find(gameJSON.textToWrite));
            Game newGame = _context.Games.Find(id);
            if (game.gameName != "") newGame.gameName = game.gameName;
            if (game.difficulty != newGame.difficulty)
            {
                newGame.difficulty = game.difficulty;
                newGame.difficultySetting(newGame.difficulty);
            }
            if (game.textToWrite != newGame.textToWrite) newGame.textToWrite = game.textToWrite;
            _context.Games.Update(newGame);
            _context.SaveChanges();
            return StatusCode(200);
        }
    } catch { return StatusCode(404); }
}
}
```



```

namespace backendProject.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class ResultsController : ControllerBase
    {
        private readonly CustomContext _context;

        0 references
        public ResultsController(CustomContext context)
        {
            _context = context;
        }

        0 references
        public ActionResult<IEnumerable<Result>> Index()
        {
            return _context.Results.Include(res => res.game).Include(res => res.account).ToList();
        }

        [HttpPost("/[controller]/add")]
        0 references
        public async Task<ActionResult> AddAsync()
        {
            try
            {
                using (StreamReader stream = new StreamReader(HttpContext.Request.Body))
                {
                    string body = await stream.ReadToEndAsync();
                    ResultJSON resultJSON = JsonSerializer.Deserialize<ResultJSON>(body);
                    Console.WriteLine("Check body json: " + resultJSON.gameID + "," + resultJSON.wordSpeed + "," + resultJSON.mistakes + "," + resultJSON.username);
                    Account acc = _context.Accounts.Where(a => a.Username.Equals(resultJSON.username)).FirstOrDefault();
                    Result result = resultJSON.getResult(_context.Games.Find(resultJSON.gameID), acc);
                    _context.Results.Add(result);
                    _context.SaveChanges();

                    return StatusCode(200);
                }
            }
            catch { return StatusCode(404); }
        }

        [HttpGet("/[controller]/find")]
        0 references
        public ActionResult<Result> Find(int id)
        {
            return _context.Results.Find(id);
        }

        [HttpGet("/[controller]/player")]
        0 references
        public ActionResult<IEnumerable<Result>> ForAccount(int accId)
        {
            try { return findByAccount(accId).ToList(); }
            catch { return StatusCode(404); }
        }
    }
}

```

```

[HttpDelete("/[controller]/delete")]
0 references
public ActionResult Delete(int id)
{
    try
    {
        Result result = _context.Results.Find(id);
        _context.Results.Remove(result);
        _context.SaveChanges();
        return StatusCode(200);
    }
    catch (Exception e) { return StatusCode(404); }
}

[HttpGet("/[controller]/scoretable")]
0 references
public ActionResult<IEnumerable<ResultTableJSON>> getScoreTable()
{
    //try
    //{
    //    return _context.Results.OrderByDescending(r => r.finish_date).Select(r => r.GetResultTableJSON(r.game, r.account)).ToList();
    //}
    //catch { return StatusCode(404); }
}

[HttpGet("/[controller]/scoretable/player")]
0 references
public ActionResult<IEnumerable<ResultTableJSON>> getScoreTableByAccount(int accId)
{
    try
    {
        return _context.Results.OrderByDescending(r => r.finish_date).Where(r => r.account == _context.Accounts.Find(accId)).Select(r => r.GetResultTableJSON(r.game, r.account)).ToList();
    }
    catch { return StatusCode(404); }
}

[HttpGet("/[controller]/scoretable/game")]
0 references
public ActionResult<IEnumerable<ResultTableJSON>> getScoreTableByGame(int gameId)
{
    try
    {
        return _context.Results.OrderByDescending(r => r.finish_date).Where(r => r.game == _context.Games.Find(gameId)).Select(r => r.GetResultTableJSON(r.game, r.account)).ToList();
    }
    catch { return StatusCode(404); }
}

2 references
private IEnumerable<Result> findByAccount(int accId)
{
    return _context.Results.Where(r => r.account == _context.Accounts.Find(accId));
}
}

```

```

namespace backendProject.Controllers
{
    [ApiController]
    [Route("[controller]")]
    1 reference
    public class WritingTextsController : ControllerBase
    {
        private readonly CustomContext _context;

        0 references
        public WritingTextsController(CustomContext context)
        {
            _context = context;
        }

        0 references
        public ActionResult<IEnumerable<WritingText>> Index()
        {
            return _context.WritingTexts.ToList();
        }

        [HttpPost("/[controller]/add")]
        0 references
        public async Task<ActionResult> AddAsync()
        {
            try
            {
                using (StreamReader stream = new StreamReader(HttpContext.Request.Body))
                {
                    string body = await stream.ReadToEndAsync();
                    WritingTextJSON writingTextJSON = JsonSerializer.Deserialize<WritingTextJSON>(body);
                    WritingText writingText = writingTextJSON.getWritingText();
                    _context.WritingTexts.Add(writingText);
                    _context.SaveChanges();
                    return StatusCode(200);
                }
            }
            catch (Exception e)
            {
                return StatusCode(404);
            }
        }

        [HttpGet("/[controller]/find")]
        0 references
        public ActionResult<WritingText> Find(int id)
        {
            return _context.WritingTexts.Find(id);
        }
    }
}

```

```
[HttpDelete("/[controller]/delete")]
0 references
public ActionResult Delete(int id)
{
    try
    {
        WritingText wt = _context.WritingTexts.Find(id);
        _context.Games.Where(g => g.textToWrite == wt).ToList().ForEach(g => g.textToWrite = null);
        _context.WritingTexts.Remove(wt);
        _context.SaveChanges();
        return StatusCode(200);
    }
    catch { return StatusCode(404); }
}

[HttpPut("/[controller]/update")]
0 references
public async System.Threading.Tasks.Task<ActionResult> UpdateAsync(int id)
{
    try
    {
        using (StreamReader stream = new StreamReader(HttpContext.Request.Body))
        {
            string body = await stream.ReadToEndAsync();
            WritingTextJSON writingTextJSON = JsonSerializer.Deserialize<WritingTextJSON>(body);
            WritingText writingText = writingTextJSON.getWritingText();
            WritingText newWritingText = _context.WritingTexts.Find(id);
            if (newWritingText != null) {
                if (writingText.text != "") newWritingText.text = writingText.text;
                if (writingText.source != "") newWritingText.source = writingText.source;
                _context.WritingTexts.Update(newWritingText);
                _context.SaveChanges();
                return StatusCode(200);
            } else return StatusCode(404);
        }
    }
    catch (Exception e)
    {
        return StatusCode(404);
    }
}

[HttpPut("/[controller]/updateSpeeds")]
0 references
public ActionResult UpdateSpeeds(int id)
{
    try
    {
        WritingText wt = _context.WritingTexts.Find(id);
        wt.topSpeed = getTopSpeed(id);
        wt.averageSpeed = getAvgSpeed(id);
        _context.WritingTexts.Update(wt);
        _context.SaveChanges();
        return StatusCode(200);
    }
    catch { return StatusCode(404); }
}
```

Metody dla tekstu, do bieżącego aktualizowania najlepszego wyniku i średniej prędkości:

```
1 reference
private double getAvgSpeed(int textID)
{
    double sumSpeed = _context.Results.Where(r => r.game.textToWrite == _context.WritingTexts.Find(textID)).ToList().Aggregate(0.0d, (acc, x) => acc + x.wordSpeed);
    double resultsSpeed;
    if (sumSpeed != 0) resultsSpeed = sumSpeed / _context.Results.ToList().Count();
    else return 0;
    return resultsSpeed;
}

1 reference
private double getTopSpeed(int textID)
{
    try { return _context.Results.Where(r => r.game.textToWrite == _context.WritingTexts.Find(textID)).ToList().Max(r => r.wordSpeed); }
    catch { return 0; }
}
```



Uniwersytet Technologiczno – Przyrodniczy



Wygląd poszczególnych okien ze strony aplikacji klienckiej:

Menu główne po uruchomieniu aplikacji:

Speed up your typing!

Hello from userMenu

Login

Train Game!

Ranking

Username

Enter username

Password

Password

Login Register

Po udanym zalogowaniu:

Speed up your typing!

Hello from userMenu

Login

Train Game!

Ranking

Welcome drugi

Wybór gry po kliknięciu w przycisk Train Game!:

Speed up your typing!

Hello from userMenu

Login

Train Game!

Ranking

testGame

tryme7

tryme6

tryme5

tryme4

tryme2

tryme

Lorem welcome

Główna funkcjonalność, czyli wpisywanie tekstu na czas:

Speed up your typing!

Hello from userMenu

Login

Train Game!


Ranking

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

52.93 words/min



Po zakończeniu wpisywania, dane o grze wysyłane są do serwisu i zapisywane w bazie danych. Można je potem sprawdzić przechodząc do okna Ranking:



Speed up your typing!

Hello from userMenu		id	game	wordSpeed	account	finish_date	isPassed	mistakes
Login		1	testGame	12	pierwszy	Tue Jan 12 2021 23:48:39 GMT+0100 (Central European Standard Time)	false	0
Train Game!		2	tryme	10	drugi	Sat Jan 16 2021 17:23:37 GMT+0100 (Central European Standard Time)	false	1
Ranking		3	tryme5	107.33	drugi	Sat Jan 16 2021 18:04:18 GMT+0100 (Central European Standard Time)	true	0
		4	tryme	20	drugi	Sat Jan 16 2021 19:05:24 GMT+0100 (Central European Standard Time)	true	0
		5	tryme4	0	drugi	Sat Jan 16 2021 19:09:00 GMT+0100 (Central European Standard Time)	false	0
		6	tryme7	94.80170643071575	drugi	Sat Jan 16 2021 19:11:11 GMT+0100 (Central European Standard Time)	true	0
		7	tryme6	107.68	drugi	Sat Jan 16 2021 19:14:55 GMT+0100 (Central European Standard Time)	true	0
		8	tryme6	66.47	drugi	Sat Jan 16 2021 19:19:02 GMT+0100 (Central European Standard Time)	true	0
		9	tryme	53.86	drugi	Sun Jan 17 2021 01:13:08 GMT+0100 (Central European Standard Time)	true	0
		10	tryme	87.93	drugi	Sun Jan 17 2021 01:13:20 GMT+0100 (Central European Standard Time)	true	0
		11	Lorem welcome	44.58	drugi	Sun Jan 17 2021 01:38:25 GMT+0100 (Central European Standard Time)	true	0
		12	Lorem welcome	54.64	drugi	Sun Jan 17 2021 13:03:31 GMT+0100 (Central European Standard Time)	true	0
		13	tryme6	83.84	drugi	Sun Jan 17 2021 13:57:35 GMT+0100 (Central European Standard Time)	true	0