



Politechnika Wrocławska

Platformy programistyczne .Net i Java
Aplikacja do zarządzania korepetycjami

Prowadzący: dr inż. Aneta Górniak
Grupa Wtorek, 18:55 - 20:35

Wojciech Buła, Przemysław Erbert

23.04.2024

Spis treści

1	Wprowadzenie	2
1.1	Wykorzystane technologie	2
2	Zasada działania aplikacji	2
2.1	Rejestracja użytkownika	2
2.2	Logowanie	3
2.3	Okno ucznia	3
2.4	Okno korepetytora	5
3	Baza danych	7
3.1	Struktura bazy danych	7
3.2	Modele Entity Framework	7
3.3	Data Context	8
3.4	Dodawanie do bazy	9
3.4.1	Dodawanie użytkownika	9
3.4.2	Dodawanie lekcji	11
3.5	Aktualizacja danych w bazie	11
4	Komunikacja z API	12
4.1	Google Calendar API	12
4.2	Nuget Packages	12
4.3	Credential.json	13
4.4	Implementacja metody do wyświetlania Calendar Events	13
4.5	Działanie	14
5	Dokumentacja Doxygen	15
6	Dokumentacja Swagger	15
7	Obsługa wyjątków	16
8	Wymagania projektowe	17

1. Wprowadzenie

Stworzyliśmy aplikację *TutorManager* która pozwala na zapisywanie si, planowanie i monitorowanie korepetycji zarówno po stronie ucznia jak i korepetytora.

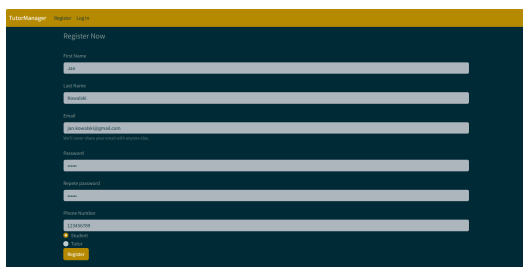
1.1. Wykorzystane technologie

- **ASP.NET** wraz ze wzorcem projektowym MVC
- **Entity framework** - system ORM do mapowania bazy danych
- **Google Calendar API** - pobieranie danych z kalendarza google przez API
- **Doxygen** - dokumentacja XML, HTML
- **Swagger** - dokumentacja json
- **Bootstrap** - frontend

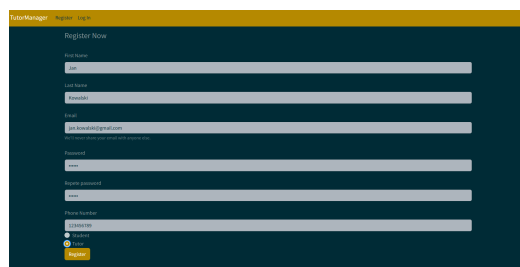
2. Zasada działania aplikacji

2.1. Rejestracja użytkownika

- Formularz html do rejestracji użytkownika w bazie aplikacji zawiera pola:
 - **First name** - pierwsze imię użytkownika (text)
 - **Last name** - nazwisko użytkownika (text)
 - **Email** - mail, który posłuży również jako login (text zawierający znak '@')
 - **Password** - hasło (text)
 - **Repete Password** - powtórzenie hasła do sprawdzenia czy zostało poprawnie wpisane
 - **Phone Number** - numer telefonu
 - **Student/Tutor** - oznaczenie typu konta i bazy do której skierowane zostaną dane



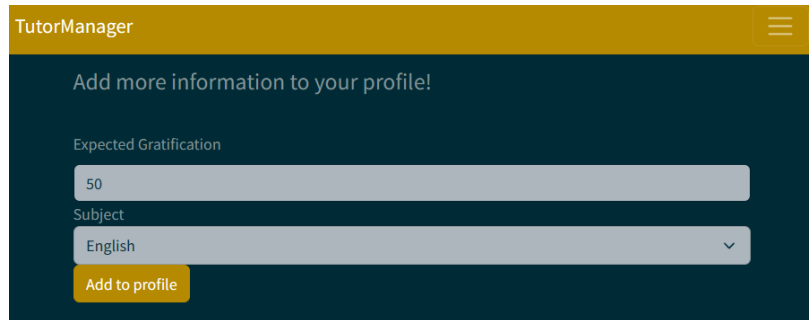
Rys. 1: Formularz rejestracji dla ucznia



Rys. 2: Formularz rejestracji dla korepetytora

- W przypadku korepetytora po początkowej rejestracji wymagane są jeszcze dodatkowe dane.
 - **Expected Gratification** - stawka za godzinę lekcji (numer)

- **Subject** - nauczany przedmiot (wybór z pośród *Maths*, *Physics*, *Biology*, *English*, *German*)



Rys. 3: Dodatkowe dane dla konta korepetytora

2.2. Logowanie

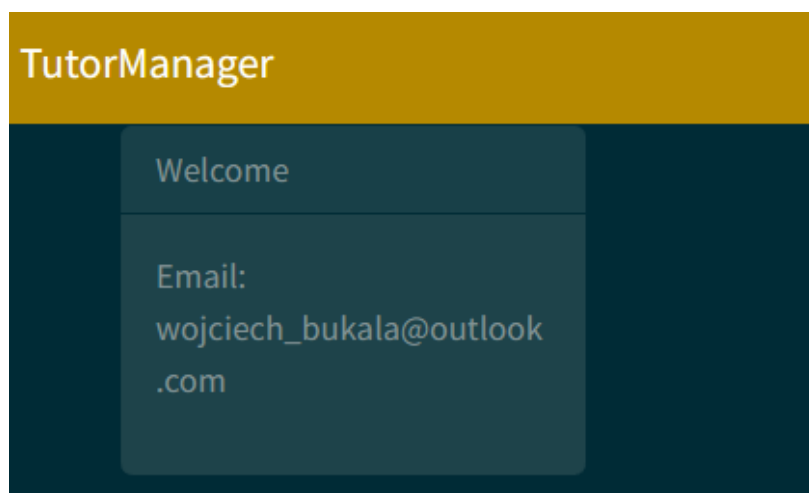
- Logowanie powiedzie się w przypadku gdy istnieje w bazie konto o podanym email u hasle. Jeden widok logowania dla ucznia i korepetytora



Rys. 4: Okno logowania do aplikacji

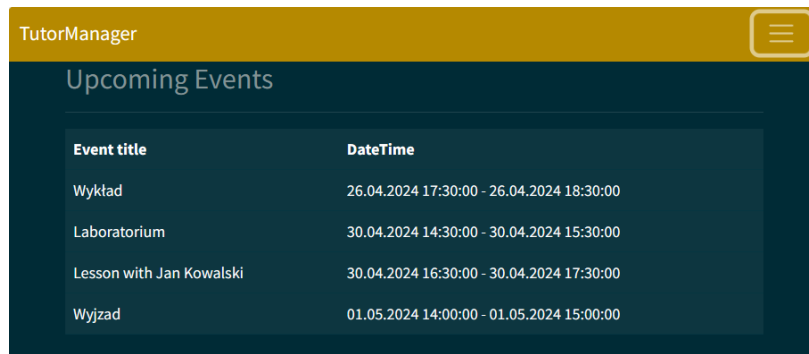
2.3. Okno ucznia

- Okno ucznia zawiera:
 - **Home** - strona domowa



Rys. 5: Strona domowa

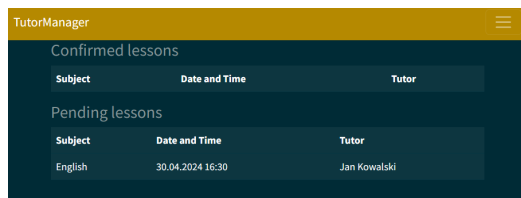
- **Schedule** - harmonogram: lekcje z aplikacji plus wydarzenia z Google Calendar



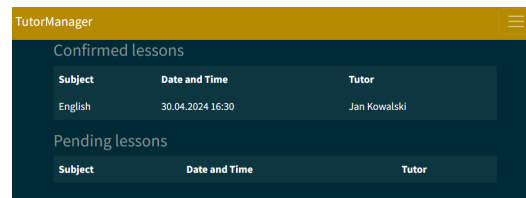
Event title	DateTime
Wykład	26.04.2024 17:30:00 - 26.04.2024 18:30:00
Laboratorium	30.04.2024 14:30:00 - 30.04.2024 15:30:00
Lesson with Jan Kowalski	30.04.2024 16:30:00 - 30.04.2024 17:30:00
Wyjazd	01.05.2024 14:00:00 - 01.05.2024 15:00:00

Rys. 6: Dodatkowe dane dla konta korepetytora

- **Lessons** - lekcje: potwierdzone i oczekujące na zatwierdzenie korepetytora



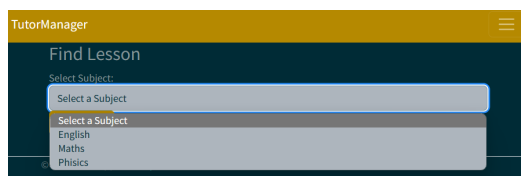
Subject	Date and Time	Tutor
English	30.04.2024 16:30	Jan Kowalski



Subject	Date and Time	Tutor
English	30.04.2024 16:30	Jan Kowalski

Rys. 7: Lekcje przed potwierdzeniem przez ko-Rys. 8: Lekcje po potwierdzeniu przez korepetytora

- **FindLesson** - panel do zapisów na lekcję



Select Subject:

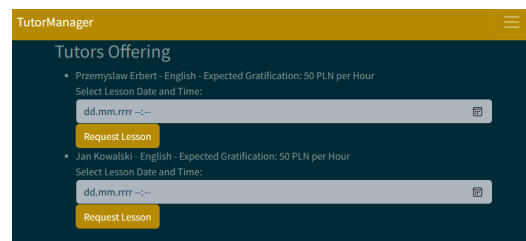
Select a Subject

English

Maths

Physics

Rys. 9: Wybranie przedmiotu z dostępnych (tych dla których są korepetytorzy w bazie



Tutors Offering

- Przemyslaw Erbert - English - Expected Gratification: 50 PLN per Hour

Select Lesson Date and Time:

dd.mm.rrrr--:--

Request Lesson

- Jan Kowalski - English - Expected Gratification: 50 PLN per Hour

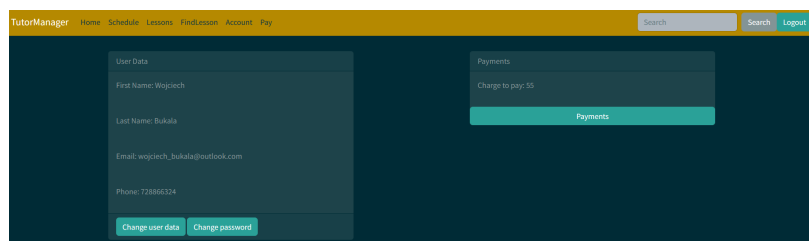
Select Lesson Date and Time:

dd.mm.rrrr--:--

Request Lesson

Rys. 10: Zaproponowanie daty lekcji

- **Acount** - informacje o koncie z możliwością zmiany danych



User Data

First Name: Wojciech

Last Name: Bukala

Email: wojciech.bukala@outlook.com

Phone: 728866324

Change user data

Change password

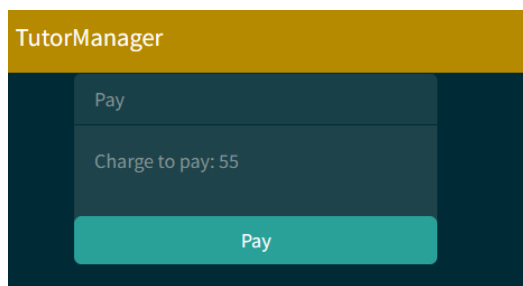
Payments

Charge to pay: \$0

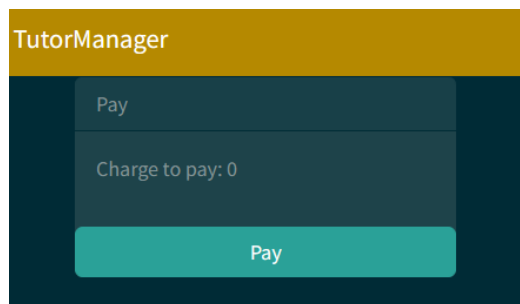
Payments

Rys. 11: Dodatkowe dane dla konta korepetytora

- **Pay** - panel opłaty za lekcję



Rys. 12: Widok przed zapłaceniem

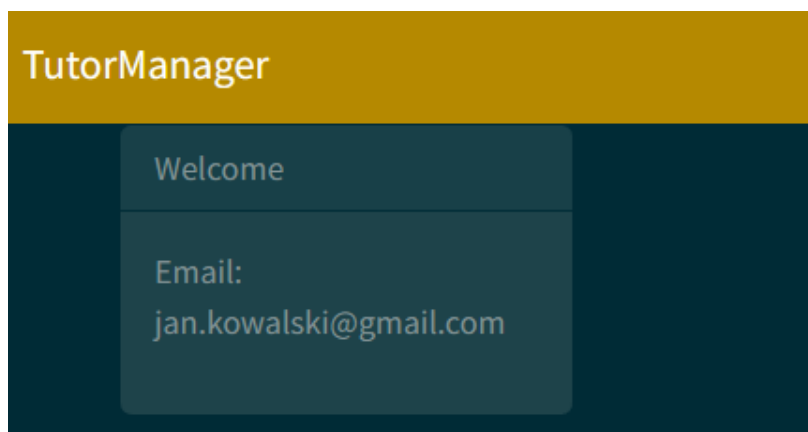


Rys. 13: Widok po zapłaceniu

2.4. Okno korepetytora

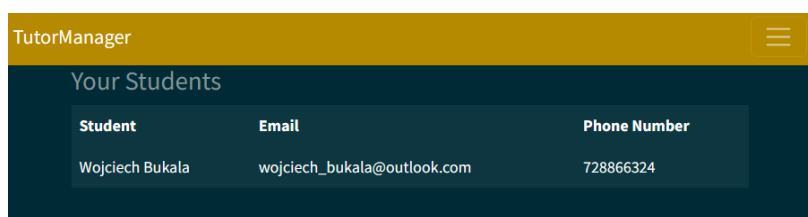
- Okno korepetytora zawiera:

- **Home** - strona domowa



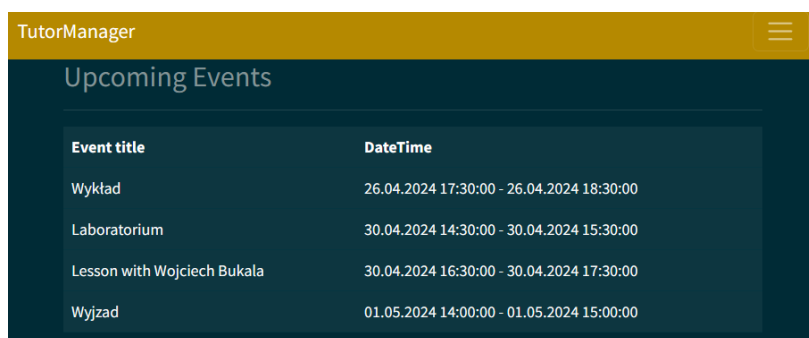
Rys. 14: Strona domowa

- **Student list** - lista studentów danego korepetytora



Rys. 15: Lista studentów korepetytora

- **Schedule** - harmonogram: lekcje z aplikacji plus wydarzenia z Google Calendar

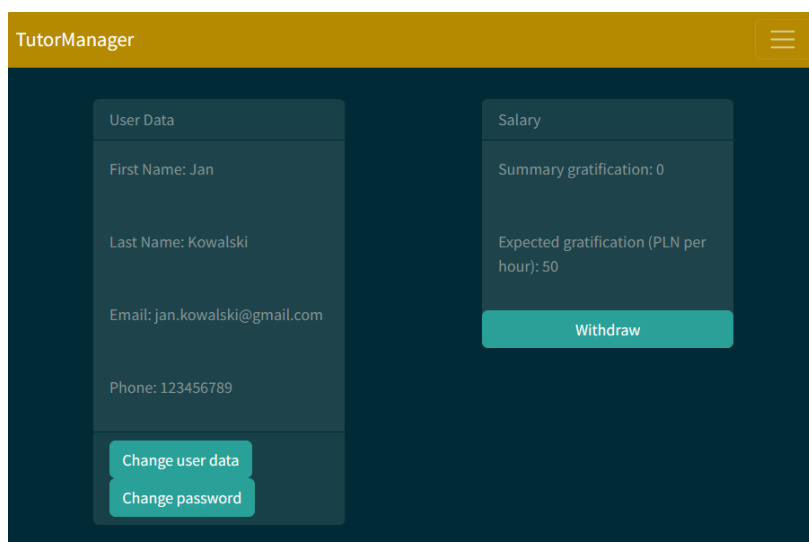


The screenshot shows the 'Upcoming Events' section of the TutorManager application. It features a table with two columns: 'Event title' and 'DateTime'. The events listed are: 'Wykład' (Lecture) on 26.04.2024 from 17:30:00 to 18:30:00; 'Laboratorium' (Laboratory) on 30.04.2024 from 14:30:00 to 15:30:00; 'Lesson with Wojciech Bukala' on 30.04.2024 from 16:30:00 to 17:30:00; and 'Wyjazd' (Trip) on 01.05.2024 from 14:00:00 to 15:00:00.

Event title	DateTime
Wykład	26.04.2024 17:30:00 - 26.04.2024 18:30:00
Laboratorium	30.04.2024 14:30:00 - 30.04.2024 15:30:00
Lesson with Wojciech Bukala	30.04.2024 16:30:00 - 30.04.2024 17:30:00
Wyjazd	01.05.2024 14:00:00 - 01.05.2024 15:00:00

Rys. 16: Dodatkowe dane dla konta korepetytora

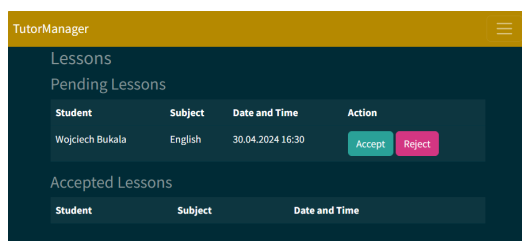
- **Acconut** - informację o koncie z możliwością zmiany danych



The screenshot shows the 'Acconut' (Account) page of the TutorManager application. It is divided into two main sections: 'User Data' and 'Salary'. The 'User Data' section contains fields for 'First Name: Jan', 'Last Name: Kowalski', 'Email: jan.kowalski@gmail.com', and 'Phone: 123456789'. Below these fields are two buttons: 'Change user data' and 'Change password'. The 'Salary' section shows 'Summary gratification: 0' and 'Expected gratification (PLN per hour): 50', with a 'Withdraw' button at the bottom.

Rys. 17: Informacje o koncie wraz z możliwością zmiany danych

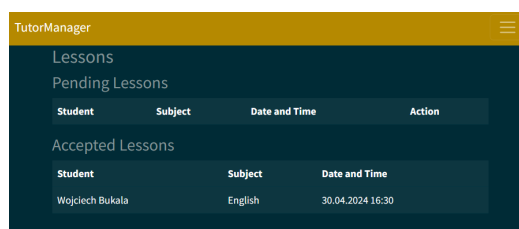
- **Lessons** - lekcje z możliwością potwierdzenia przyjęcia lekcji



The screenshot shows the 'Lessons' page of the TutorManager application, specifically the 'Pending Lessons' section. It displays a table with columns: 'Student', 'Subject', 'Date and Time', and 'Action'. A single lesson is listed for 'Wojciech Bukala' in 'English' on '30.04.2024 16:30'. The 'Action' column contains two buttons: 'Accept' (green) and 'Reject' (red). Below this is a section for 'Accepted Lessons' with a table header: 'Student', 'Subject', and 'Date and Time'.

Student	Subject	Date and Time	Action
Wojciech Bukala	English	30.04.2024 16:30	Accept Reject

Rys. 18: Lekcje przez zaakceptowaniem



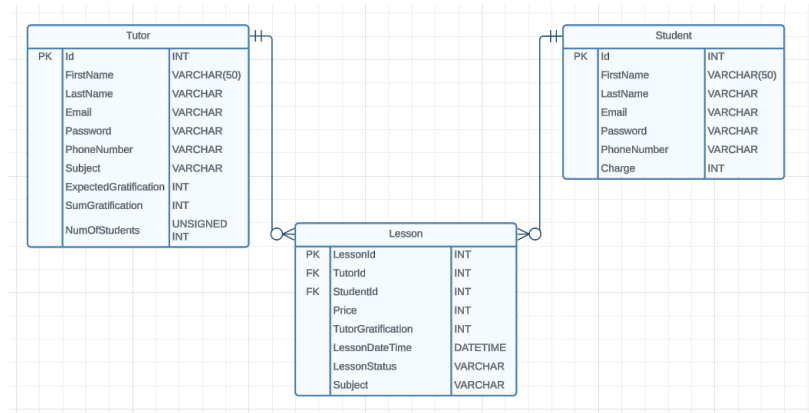
The screenshot shows the 'Lessons' page of the TutorManager application, specifically the 'Accepted Lessons' section. It displays a table with columns: 'Student', 'Subject', and 'Date and Time'. A single lesson is listed for 'Wojciech Bukala' in 'English' on '30.04.2024 16:30'.

Student	Subject	Date and Time
Wojciech Bukala	English	30.04.2024 16:30

Rys. 19: Lekcje po zaakceptowaniu

3. Baza danych

3.1. Struktura bazy danych



Rys. 20: Entities Relationship Diagram

3.2. Modele Entity Framework

- Model *User* na podstawie którego, poprzez dziedziczenie, utworzone zostały modele *Tutor* i *Student*

```
public class UserModel
{
    [Key, Column(Order = 1)]
    18 references
    public int Id { get; set; }

    [StringLength(50)]
    25 references
    public string? FirstName { get; set; }

    [StringLength(50)]
    25 references
    public string? LastName { get; set; }
    26 references
    public string? Email { get; set; }
    21 references
    public string? Password { get; set; }
    17 references
    public string? PhoneNumber { get; set; }

    [NotMapped]
    [Compare("Password")]
    12 references
    public string? ConfirmPassword { get; set; }
    0 references
    public string FullName()
    {
        return this.FirstName + " " + this.LastName;
    }
}
```

Rys. 21: Model User


```

public class TutorModel : UserModel
{
    8 references
    public string? Subject { get; set; }
    12 references
    public int ExpectedGratification { get; set; }
    5 references
    public int SumGratification { get; set; }
    3 references
    public uint NumOfStudents { get; set; }
}

```

Rys. 22: Model Tutor

```

public class StudentModel : UserModel
{
    6 references
    public int Charge { get; set; }
}

```

Rys. 23: Model Student

- Model połączony relacyjnie z *Tutor* i *Student* do zapisywania lekcji (potwierdzonych i niepotwierdzonych)

```

public class LessonModel
{
    [Key]
    2 references
    public int LessonId { get; set; }

    [ForeignKey("Tutor")]
    6 references
    public int TutorId { get; set; }
    7 references
    public virtual TutorModel? Tutor { get; set; }

    [ForeignKey("Student")]
    4 references
    public int StudentId { get; set; }
    7 references
    public virtual StudentModel? Student { get; set; }
    2 references
    public int Price { get; set; }
    1 reference
    public int TutorGratification { get; set; }
    5 references
    public DateTime LessonDateTime { get; set; }
    7 references
    public string? LessonStatus { get; set; }
    1 reference
    public string? Subject { get; set; }
}

```

Rys. 24: Lesson model

3.3. Data Context

```

29 references
public class DataContext : DbContext
{
    0 references
    public DataContext(DbContextOptions<DataContext> options) : base(options)
    {
    }
    10 references
    public DbSet<TutorModel> TutorTable { get; set; }
    10 references
    public DbSet<StudentModel> StudentTable { get; set; }
    9 references
    public DbSet<LessonModel> LessonTable { get; set; }
}

```

Rys. 25: Context

3.4. Dodawanie do bazy

Dodawanie elementów do bazy odbywa się poprzez rejestrację użytkownika (ucznia lub korepetytora) i dodawanie lekcji przez ucznia.

3.4.1. Dodawanie użytkownika

Metoda obsługi formularza HTML do rejestracji nowego użytkownika:

```
public IActionResult Index(UserModel model, string UserRole)
{
    if (ModelState.IsValid)
    {
        if (UserRole == "Student")
        {
            var student_model = new StudentModel
            {
                // Map properties from UserModel to StudentModel
                Id = model.Id,
                FirstName = model.FirstName,
                LastName = model.LastName,
                Email = model.Email,
                Password = model.Password,
                PhoneNumber = model.PhoneNumber,
                ConfirmPassword = model.ConfirmPassword,
                Charge = 0
            };
            var check = _db_con.StudentTable.FirstOrDefault(u => u.Email ==
                student_model.Email);
            if (check == null)
            {
                _db_con.StudentTable.Add(student_model);
                _db_con.SaveChanges();
                _session.SetInt32("StudentID", student_model.Id);
                _session.SetString("FirstName", student_model.FirstName);
                _session.SetString("LastName", student_model.LastName);
                _session.SetString("UserEmail", student_model.Email);
                _session.SetString("Phone", student_model.PhoneNumber);
                _session.SetString("Password", student_model.Password);
                _session.SetInt32("Charge", student_model.Charge);
                return RedirectToAction("Index", "Student");
            }
            else
            {
                ViewBag.error = "Email already exists";
                return View();
            }
        }
    }
}
```

```

    }
    else if (UserRole == "Tutor")
    {
        var tutor_model = new TutorModel
        {
            // Map properties from UserModel to TutorModel
            Id = model.Id,
            FirstName = model.FirstName,
            LastName = model.LastName,
            Email = model.Email,
            Password = model.Password,
            PhoneNumber = model.PhoneNumber,
            ConfirmPassword = model.ConfirmPassword,
            SumGratification = 0,
            NumOfStudents = 0

        };
        var check = _db_con.TutorTable.FirstOrDefault(u => u.Email ==
            tutor_model.Email);
        if (check == null)
        {
            _db_con.TutorTable.Add(tutor_model);
            _db_con.SaveChanges();
            _session.SetInt32("TutorID", tutor_model.Id);
            _session.SetString("FirstName", tutor_model.FirstName);
            _session.SetString("LastName", tutor_model.LastName);
            _session.SetString("UserEmail", tutor_model.Email);
            _session.SetString("Phone", tutor_model.PhoneNumber);
            _session.SetString("Password", tutor_model.Password);
            _session.SetInt32("SumGratification", tutor_model.SumGratification);
            _session.SetInt32("NumOfStudents", Convert.ToInt32(tutor_model.
                NumOfStudents));
            _session.SetInt32("ExpGratification", tutor_model.
                ExpectedGratification);
            return RedirectToAction("MoreInfo");
        }
        else
        {
            ViewBag.error = "Email already exists";
            return View();
        }
    }
}
return View(model);
}

```

3.4.2. Dodawanie lekcji

Student w oknie *FindLesson* może poprosić o lekcję, jest ona wtedy dodawana do bazy ze statusem *Pending* za pomocą poniższej metody:

```
public IActionResult RequestLesson(int tutorId, DateTime lessonDateTime)
{
    var userEmail = HttpContext.Session.GetString("UserEmail");
    var student = _db_con.StudentTable.FirstOrDefault(u => u.Email == userEmail);
    var tutor = _db_con.TutorTable.FirstOrDefault(t => t.Id == tutorId);

    if (student != null && tutor != null)
    {
        var newLesson = new LessonModel
        {
            StudentId = student.Id,
            TutorId = tutor.Id,
            Student = student,
            Tutor = tutor,
            Subject = tutor.Subject,
            LessonStatus = "Pending",
            LessonDateTime = lessonDateTime
        };

        _db_con.LessonTable.Add(newLesson);
        _db_con.SaveChanges();
    }
    return RedirectToAction("FindLesson");
}
```

3.5. Aktualizacja danych w bazie

Aktualizacje danych można wykonać za pomocą przycisku *Change user data* lub *Change password* w widoku *Account* dla ucznia lub korepetytora.

```
public IActionResult ChangePassword(UserModel model)
{
    if (ModelState.IsValid)
    {
        var userEmail = HttpContext.Session.GetString("UserEmail");
        var _student = _db_con.StudentTable.FirstOrDefault(u => u.Email ==
            userEmail);

        if (_student != null)
        {
            _student.Password = model.Password;
            _student.ConfirmPassword = model.ConfirmPassword;
            _db_con.SaveChanges();
        }
    }
}
```

```

        ViewBag.SuccessMessage = "Data updated";
    }
}

return View();
}

public IActionResult ChangeData(UserModel model)
{
    if (ModelState.IsValid)
    {
        var userEmail = HttpContext.Session.GetString("UserEmail");
        var _student = _db_con.StudentTable.FirstOrDefault(u => u.Email ==
            userEmail);

        if (_student != null)
        {
            _student.FirstName = model.FirstName;
            _student.LastName = model.LastName;
            _student.PhoneNumber = model.PhoneNumber;
            _db_con.SaveChanges();
            ViewBag.SuccessMessage = "Data updated";
        }
    }
    return View(model);
}

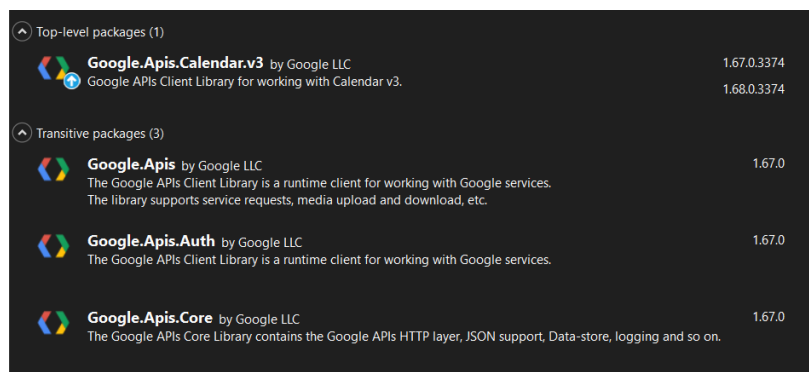
```

4. Komunikacja z API

4.1. Google Calendar API

4.2. Nuget Packages

Aby korzystać z Google Apis należy pobrać następujące pakiety:



Rys. 26: Google API packages

4.3. Credential.json

- Na stronie <https://console.cloud.google.com/> należy utworzyć projekt dla danego API i wygenerować klucze dostępu
- Klucze można pobrać jako Credential.json

4.4. Implementacja metody do wyświetlania Calendar Events

```
public void CalendarEvents()
{
    UserCredential credential;
    //string path = Server.MapPath("credentail.json");

    using( var stream =
        new FileStream("Credential2.json", FileMode.Open, FileAccess.Read))
    {
        string credPath = "token.json";
        credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
            GoogleClientSecrets.Load(stream).Secrets,
            Scopes,
            "user",
            CancellationToken.None,
            new FileDataStore(credPath, true)).Result;
    }

    var service = new CalendarService(new BaseClientService.Initializer()
    {
        HttpClientInitializer = credential,
        ApplicationName = ApplicationName,
    });

    EventsResource.ListRequest request = service.Events.List("primary");
    request.TimeMin = DateTime.Now;
    request.ShowDeleted = false;
    request.SingleEvents = true;
    request.MaxResults = 10;
    request.OrderBy = EventsResource.ListRequest.OrderByEnum.StartTime;

    Events events = request.Execute();
    if(events.Items != null && events.Items.Count > 0)
    {
        foreach(var eventItem in events.Items)
        {
            GoogleEvents.Add(new
            {
                Title = eventItem.Summary,
```

```

        Start = eventItem.Start.DateTime,
        End = eventItem.End.DateTime,
    });
}
}

var userEmail = HttpContext.Session.GetString("UserEmail");
var tutor = _db_con.TutorTable.FirstOrDefault(t => t.Email == userEmail);
var lessons = _db_con.LessonTable
    .Include(l => l.Student)
    .Where(l => l.TutorId == tutor.Id).ToList();

foreach (var lesson in lessons)
{
    GoogleEvents.Add(new
    {
        Title = $"Lesson with {lesson.Student.FirstName} {lesson.Student.
            LastName}",
        Start = lesson.LessonDateTime,
        End = lesson.LessonDateTime.AddHours(1),
    });
}

GoogleEvents = GoogleEvents.OrderBy((dynamic e) => e.Start).ToList();
}

}

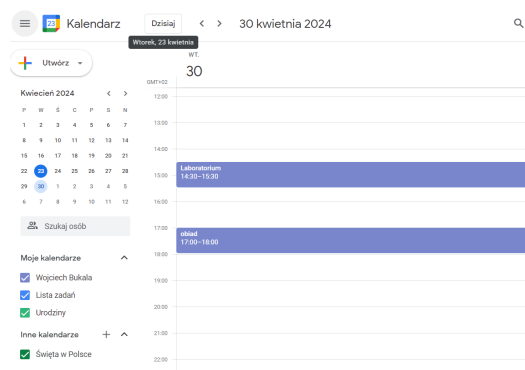
```

4.5. Działanie

Upcoming Events

Event title	DateTime
Wykład	26.04.2024 17:30:00 - 26.04.2024 18:30:00
Laboratorium	30.04.2024 14:30:00 - 30.04.2024 15:30:00
Lesson with Jan Kowalski	30.04.2024 16:30:00 - 30.04.2024 17:30:00
Wykład	30.04.2024 17:00:00 - 30.04.2024 18:00:00
Wykład	01.05.2024 14:00:00 - 01.05.2024 15:00:00

Rys. 27: Wydarzenia z kalendarz oraz lekcje z aplikacji wyświetlane w jednej tabeli



Rys. 28: Możliwość dodawania wydarzeń do kalendarza google

5. Dokumentacja Doxygen

Do wygenerowania dokumentacji XML wykorzystano aplikację *Doxygen*. Dokumentacja jest dostępna w repozytorium GitHub jako index.html

TutorManagerDoc

[Main Page](#) [Related Pages](#) [Packages ▾](#) [Classes ▾](#)

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

[detail level 1 2 3]

▾ N TutorManager	
▾ N Controllers	
C HomeController	
C LoginController	Kontroler do obsługi logowania
C RegisterController	Kontroler do obsługi akcji rejestracji
C StudentController	Kontroler panelu Studenta
C TutorController	Kontroler panelu Tutora
▸ N Data	
▸ N Migrations	
▾ N Models	
C ErrorViewModel	
C LessonModel	Model lekcji
C StudentModel	Model Student dziedziczący do User
C TutorModel	Model Tutor dziedziczący do User
C UserModel	Model użytkownika

Generated by **doxygen** 1.10.0

6. Dokumentacja Swagger

Do wygenerowania dokumentacji w postaci json wykorzystaliśmy narzędzie Swagger.

TutorManager API API OpenAPI

[swagger/v1/swagger.json](#)

Login

GET /api/Login/Index

POST /api/Login/Index

GET /api/Login/Logout

Register

GET /api/Register/Index

POST /api/Register/Index

GET /api/Register/MoreInfo

POST /api/Register/MoreInfo

Student			^
GET	/api/Student/Index		✓
GET	/api/Student/Schedule		✓
GET	/api/Student/Account		✓
GET	/api/Student/ChangeData		✓
POST	/api/Student/ChangeData		✓
GET	/api/Student/ChangePassword		✓
POST	/api/Student/ChangePassword		✓
GET	/api/Student/FindLesson		✓
POST	/api/Student/FindLesson		✓
POST	/api/Student/RequestLesson		✓
GET	/api/Student/Lesson		✓
GET	/api/Student/Pay		✓
GET	/api/Student/Payment		✓
GET	/api/Student/CalendarEvents		✓

Tutor			^
GET	/api/Tutor/Index		✓
GET	/api/Tutor/Account		✓
GET	/api/Tutor/Withdraw		✓
GET	/api/Tutor/Schedule		✓
GET	/api/Tutor/Student_list		✓
GET	/api/Tutor/Lessons		✓
POST	/api/Tutor/AcceptLesson		✓
POST	/api/Tutor/RejectLesson		✓
POST	/api/Tutor/CalendarEvents		✓

Schemas			^
TutorModel >			
UserModel >			

7. Obsługa wyjątków

Przykładowa obsługa braku odnalezienia konta google

```
public void CalendarEvents()
{
    UserCredential credential;
    try
    {
        OBSUGA  KALENDARZA
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred: {ex.Message}");
    }
}
```

8. Wymagania projektowe

- Aplikacja napisana w języku programowania C sharp.
- Interfejs użytkownika powinien wykorzystywać technologię WPF lub podobną (UWP, ASP.NET) - ASP.NET
- Należy zadbać o persystencję danych w aplikacji (dane nie powinny znikać po zamknięciu aplikacji, po ponownym uruchomieniu aplikacji dane powinny być dalej dostępne).
- Utworzenie bazy danych ORM i jej obsługa w technologii Entity Framework.
- Możliwość zapisu i odczytu danych w obrębie aplikacji oraz do i z bazy danych - zapis rejestracji, odczyt wyświetlanie danych i wiele innych
- Możliwość ręcznego wprowadzania danych do aplikacji, np. w formie formularza - rejestracja, zapisy na lekcję
- Przeprowadzenie walidacji wprowadzanych danych i danych pobieranych z sieci - podczas rejestracji
- Graficzna prezentacja danych w postaci wykresów, tabel, list, itp. - tabelki i listy HTML
- Wykorzystanie kolekcji obiektów do obsługi modeli danych (np. do filtrowania czy wyszukiwania odpowiednich danych w bazie). - wyszukiwanie danych do znajdowania właściwych tutorów i lekcji
- Obsługa kontrolek i wyjątków.
- Aplikacja powinna obsługiwać połączenie sieciowe i komunikację z zewnętrznym serwerem API. - Google API
- Wymiana danych z API powinna obsługiwać format JSON/HTML/XML lub podobny - json
- Wygenerowanie dokumentacji przy użyciu wybranego generatora. - Doxygen i Swagger