

Catalog online

ABSOLVENT

Borze Andrei

2023

CUPRINS

1. INTRODUCERE	3
2. DESCRIEREA APLICAȚIEI – DATA PROJECT	4
2.1 Introducere	4
2.1 Models Folder	6
2.2 Exceptions Folder	9
2.3 DAL folder - Data Access Layer	10
3. DESCRIEREA APLICAȚIEI – ONLINE CATALOG APPLICATION	12
3.1 Introducere	12
3.3 Controllers.....	15
3.3 Dtos	17
3.4 Utils	18
3.4 Filters	19
CONCLUZII.....	20
IMAGINI CU APLICAȚIA	21

1. INTRODUCERE

Această lucrare se concentrează pe dezvoltarea și implementarea unui proiect de tip ASP.NET Web API pentru gestionarea studenților, adresei și notelor lor. Scopul principal al proiectului este de a oferi o soluție eficientă și flexibilă pentru administrarea informațiilor legate de studenți, inclusiv detaliile personale, adresa și performanța academică.

Proiectul își propune să ofere o interfață API robustă, ușor de utilizat și scalabilă, care să permită utilizatorilor să acceseze și să manipuleze datele despre studenți, adrese și note prin intermediul cererilor HTTP. Prin intermediul acestei aplicații, administratorii, profesorii și alți utilizatori autorizați vor putea obține informații detaliate despre studenți, să adauge sau să actualizeze date, să acorde note și să efectueze operațiuni de gestionare a datelor.

O altă caracteristică importantă a proiectului constă în utilizarea de DTO-uri (Data Transfer Objects) pentru a asigura o separare clară între obiectele de domeniu și datele transferate prin rețea. Astfel, se va încuraja un design modular și un flux eficient al datelor între client și server.

Lucrarea se va concentra, de asemenea, pe optimizarea performanței aplicației, utilizând tehnici precum cache-ul datelor și optimizarea cererilor către baza de date. De asemenea, se va implementa documentația API-ului utilizând XML comments, astfel încât dezvoltatorii să poată înțelege și utiliza în mod corespunzător funcționalitățile expuse.

În partea a doua a proiectului, se va extinde funcționalitatea pentru a permite adăugarea notelor pentru studenți. Notele vor fi caracterizate de valoarea numerică, data și ora acordării și cursul pentru care au fost acordate. Vor fi implementate funcționalități pentru adăugarea de cursuri, acordarea de note, obținerea tuturor notelor unui student, obținerea notelor pentru un anumit curs și calculul mediilor per materie ale unui student. De asemenea, se va implementa funcționalitatea pentru obținerea listei de studenți în ordinea mediilor, care poate fi configurată în mod ascendent sau descendent.

Prin finalizarea acestui proiect, se va oferi o soluție completă și robustă pentru gestionarea eficientă a informațiilor despre studenți, adrese și note într-un mediu academic sau instituțional.

2. DESCRIEREA APLICAȚIEI – DATA PROJECT

2.1 Introducere

Proiectul este structurat în jurul a patru clase principale și o interfață pentru comunicarea cu baza de date. Aceasta include clasa Address, care reprezintă adresa unui student, clasa Course, care reprezintă un curs, clasa Grade, care reprezintă o notă acordată unui student pentru un anumit curs, și clasa Student, care reprezintă un student. Interfața IStudentsDbContext definește un contract pentru accesul la baza de date și include proprietăți pentru fiecare entitate.

În cadrul proiectului, este implementată o interfață de servicii numită IDataAccessLayerService care gestionează operațiunile de bază legate de student, adresă și note. Aceasta oferă metode pentru a obține toți studenții, a obține un student după ID, a crea un student nou, a actualiza un student existent și a șterge un student. De asemenea, oferă metode pentru a actualiza sau a crea o adresă pentru un student, a obține adresa unui student, a obține lista de studenți în ordinea notelor și a acorda note unui student pentru un anumit curs. Interfața IDataAccessLayerService include și metode pentru gestionarea cursurilor. Aceasta permite adăugarea unui curs nou și obținerea tuturor cursurilor disponibile.

De asemenea, interfața IDataAccessLayerService oferă metode pentru gestionarea notelor. Aceasta permite acordarea unei note unui student pentru un anumit curs, obținerea tuturor notelor unui student, obținerea notelor unui student pentru un anumit curs și obținerea mediei pe materie pentru un student. Metodele implementate asigură adăugarea automată a datei și orei acordării notei.

Structura proiectului urmărește un design modular și bine organizat, în care fiecare clasă și interfață își îndeplinește rolul specific. Aceasta facilitează dezvoltarea, întreținerea și extensibilitatea aplicației, oferind o structură clară pentru gestionarea informațiilor despre studenți, adrese, cursuri și note.

Prin utilizarea interfețelor și implementarea contractelor bine definite, proiectul promovează principiile dezvoltării orientate pe interfețe și asigură separarea responsabilităților în cadrul codului sursă.

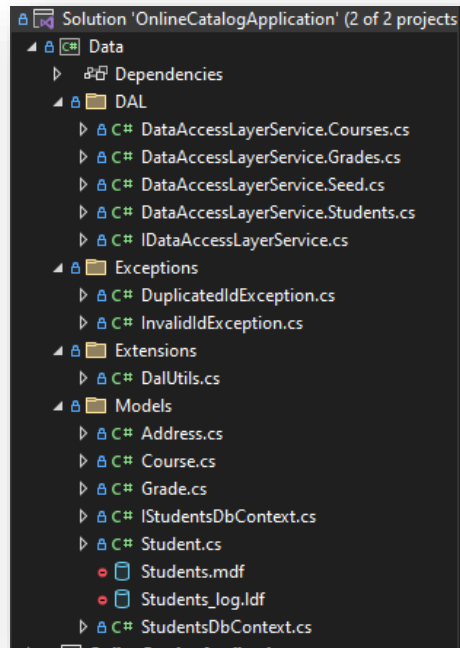


Figura nr. 2.1 Structură - Data Project

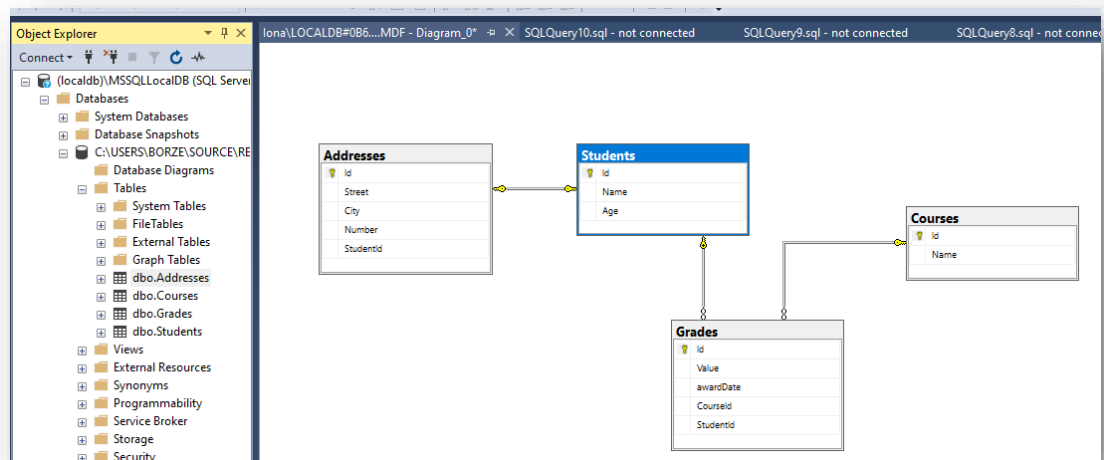


Figura nr. 2.2 High Level - UML Diagram

2.1 Models Folder

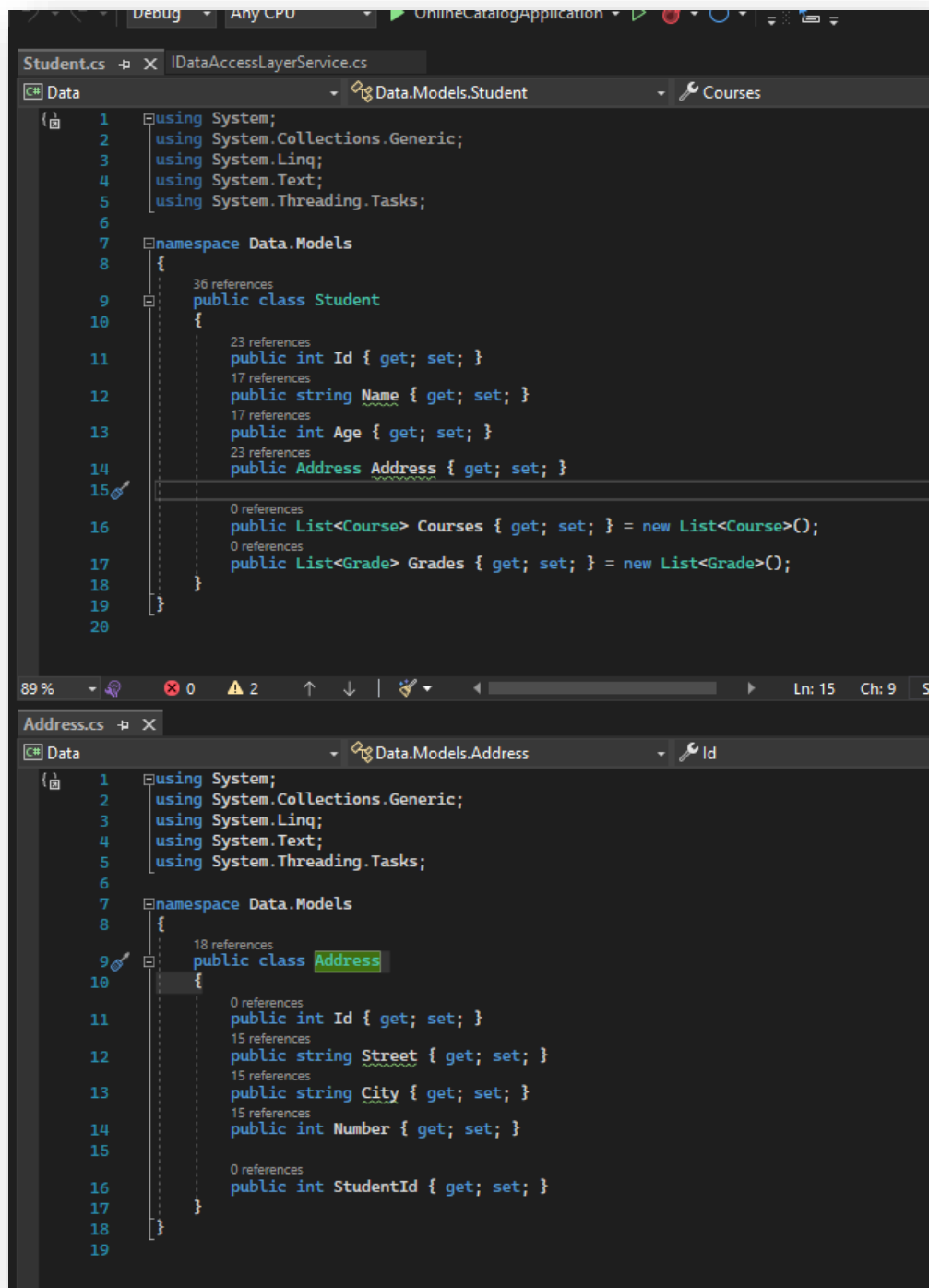


Figura nr. 2.3 Clasa Student și clasa Address

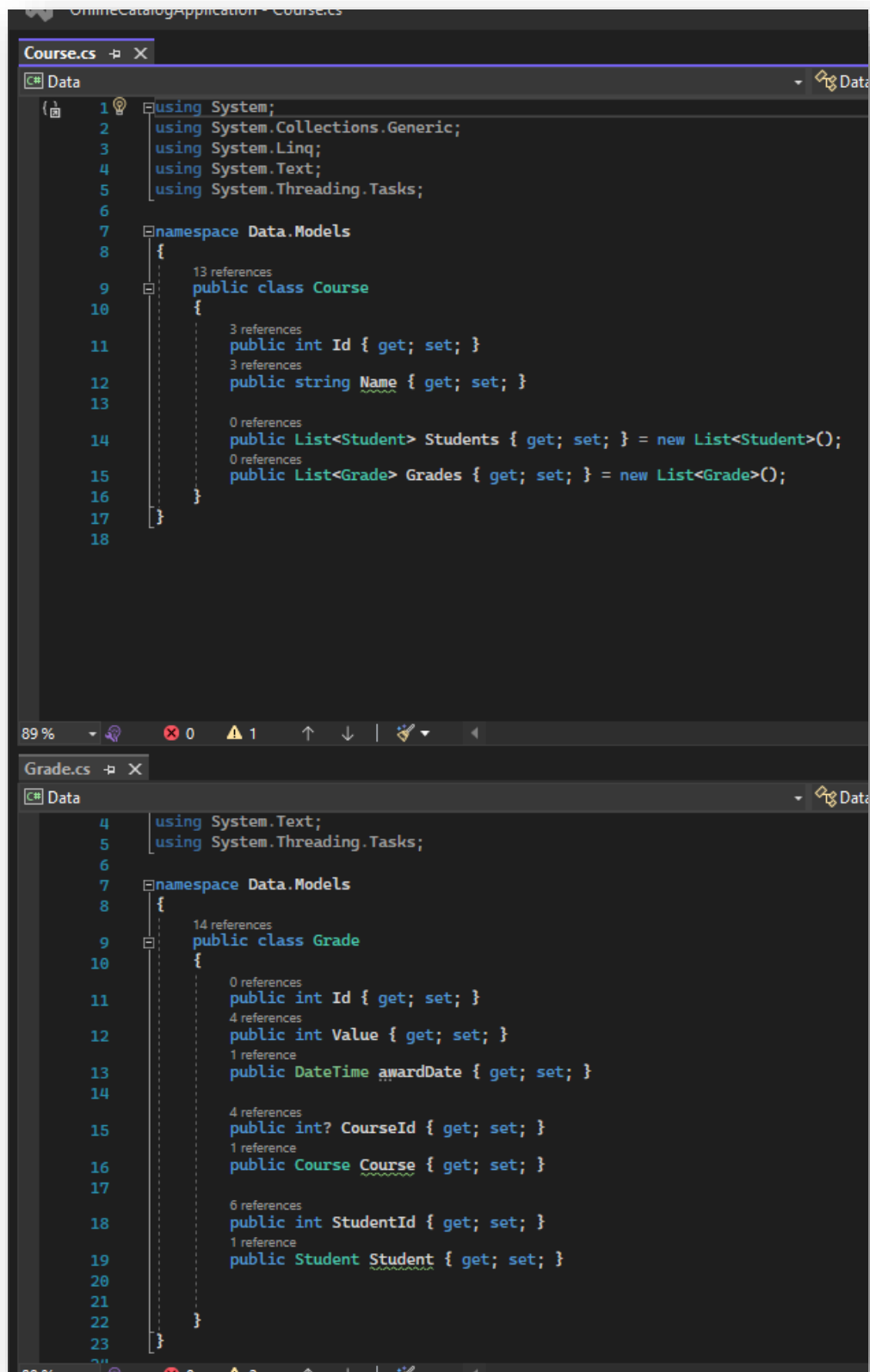


Figura nr. 2.4 Clasa Course și clasa Grade

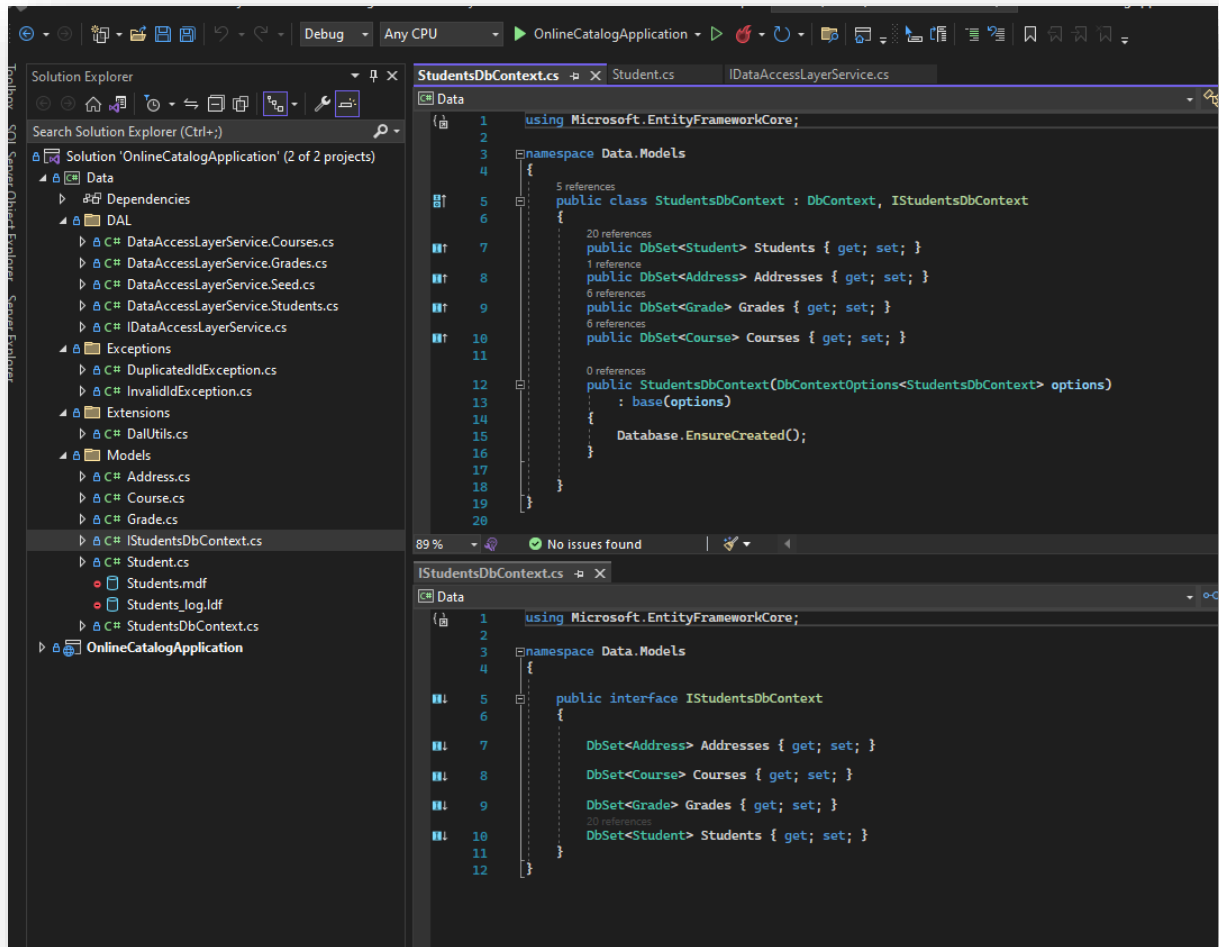


Figura nr. 2.5 Clasa StudentDbContext și interfața acestei clase

2.2 Exceptions Folder

Folder "Exceptions" în cadrul unui proiect are rolul de a gestiona și trata situațiile excepționale sau erorile care pot apărea în timpul execuției aplicației. Acesta este destinat să conțină clase specifice care extind clasa de bază pentru excepții, oferind astfel o abordare structurată și modulară pentru gestionarea excepțiilor în cod.

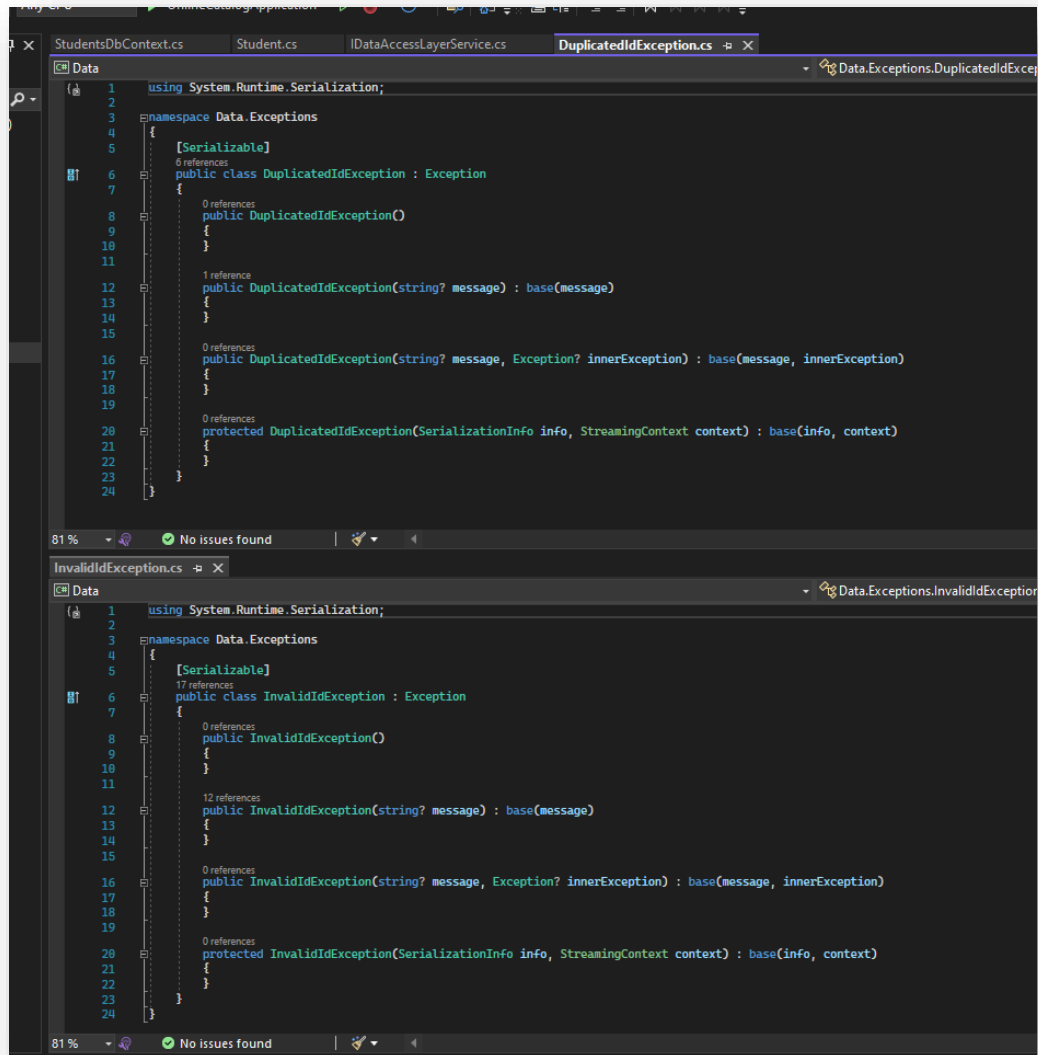


Figura nr. 2.6 Clasa `InvalidIdException` și clasa `DuplicatedIdException`

2.3 DAL folder - Data Access Layer

```
5
6 namespace Data.DAL
7 {
8     5 references
9     internal partial class DataAccessLayerService : IDataAccessLayerService
10     {
11         private readonly StudentsDbContext ctx;
12         0 references
13         public DataAccessLayerService(StudentsDbContext ctx)
14         {
15             this.ctx = ctx;
16
17         2 references
18         public IEnumerable<Student> GetAllStudents() => ctx.Students.ToList();
19
20         2 references
21         public Student GetStudentById(int studentId)
22         {
23             if (!ctx.Students.Any(x => x.Id == studentId))
24             {
25                 throw new InvalidIdException($"Invalid student id {studentId}");
26             }
27             return ctx.Students.FirstOrDefault(x => x.Id == studentId);
28         }
29
30         2 references
31         public Student CreateStudent(Student student)
32         {
33             if (ctx.Students.Any(x => x.Id == student.Id))
34             {
35                 throw new DuplicatedIdException($"Duplicated student id ");
36             }
37         }
38     }
39 }
```

```
5 namespace Data.DAL
6 {
7     5 references
8     internal partial class DataAccessLayerService : IDataAccessLayerService
9     {
10
11         2 references
12         public Grade AwardGrade(int value, int studentId, int courseId)
13         {
14             if (!ctx.Students.Any(x => x.Id == studentId))
15             {
16                 throw new InvalidIdException($"Invalid student ID {studentId}");
17             }
18             if (!ctx.Courses.Any(x => x.Id == courseId))
19             {
20                 throw new InvalidIdException($"Invalid course ID {courseId}");
21             }
22             var course = ctx.Courses.FirstOrDefault(x => x.Id == courseId);
23             var student = ctx.Students.FirstOrDefault(x => x.Id == studentId);
24             var grade = new Grade { Value = value, StudentId = studentId, CourseId = courseId, awardDate = DateTime.Now, Course = course, Student = student };
25             ctx.Grades.Add(grade);
26             ctx.SaveChanges();
27             return grade;
28         }
29
30         2 references
31         public IEnumerable<Grade> GetAllGradeForStudent(int studentId)
32         {
33             if (!ctx.Students.Any(x => x.Id == studentId))
34             {
35                 throw new InvalidIdException($"Invalid student ID {studentId}");
36             }
37             return ctx.Grades.Where(x => x.StudentId == studentId).ToList();
38         }
39
40         2 references
41         public IEnumerable<Grade> GetAllGradeForStudentForCourse(int studentId, int courseId)
42         {
43             if (!ctx.Students.Any(x => x.Id == studentId))
44             {
45                 throw new InvalidIdException($"Invalid student ID {studentId}");
46             }
47             if (!ctx.Courses.Any(x => x.Id == courseId))
48             {
49                 throw new InvalidIdException($"Invalid course ID {courseId}");
50             }
51             return ctx.Grades.Where(x => x.StudentId == studentId && x.CourseId == courseId).ToList();
52         }
53
54         2 references
55         public IDictionary<int, double> GetAllCourseAverageForStudent(int studentId)
56         {
57             if (!ctx.Students.Any(x => x.Id == studentId))
58             {
59                 throw new InvalidIdException($"Invalid student ID {studentId}");
60             }
61         }
62     }
63 }
```

Figura nr. 2.7 Clasa DataAccessLayerService

```

namespace Data.DAL
{
    13 references
    public interface IDataAccessLayerService
    {
        /// <summary>
        /// Seeds the database with initial data.
        /// </summary>
        2 references
        void Seed();

        /// <summary>
        /// Retrieves all students.
        /// </summary>
        /// <returns>An enumerable collection of Student objects.</returns>
        2 references
        IEnumerable<Student> GetAllStudents();

        /// <summary>
        /// Retrieves a student by their ID.
        /// </summary>
        /// <param name="id">The ID of the student.</param>
        /// <returns>The Student object.</returns>
        2 references
        Student GetStudentById(int id);

        /// <summary>
        /// Creates a new student.
        /// </summary>
        /// <param name="student">The Student object to create.</param>
        /// <returns>The created Student object.</returns>
        2 references
        Student CreateStudent(Student student);

        /// <summary>
        /// Updates a student.
        /// </summary>
        /// <param name="studentToUpdate">The updated Student object.</param>
        /// <returns>The updated Student object.</returns>
        2 references
        Student UpdateStudent(Student studentToUpdate);

        /// <summary>
        /// Deletes a student by their ID.
        /// </summary>
        /// <param name="studentId">The ID of the student to delete.</param>
        2 references
        void DeleteStudent(int studentId);

        /// <summary>
        /// Updates or creates a new address for a student.
        /// </summary>
        /// <param name="studentId">The ID of the student.</param>
        /// <param name="newAddress">The new address for the student.</param>
        /// <returns>True if the address was updated or created successfully, false otherwise.</returns>
        2 references
        bool UpdateOrCreateStudentAddress(int studentId, Address newAddress);

        /// <summary>
        /// Retrieves the address for a student by their ID.
        /// </summary>
        /// <param name="studentId">The ID of the student.</param>
        /// <returns>The Address object associated with the student.</returns>
    }
}

```

Figura nr. 2.8 Interfața IDataAccessLayerService

3. DESCRIEREA APLICAȚIEI – ONLINE CATALOG APPLICATION

3.1 Introducere

Proiectul web descris este o aplicație de tip ASP.NET, construită pe o arhitectură care utilizează controloare (Controllers), obiecte de transfer de date (DTOs), filtre (Filters) și un dosar "Util" pentru gestionarea transformărilor între obiectele DTO și clasele de bază.

În cadrul acestui proiect, dosarul "Controllers" conține clasele responsabile de gestionarea rutei și a cererilor primite de la client. Acestea acționează ca intermediari între cereri și logica de afaceri a aplicației. Fiecare clasă de controler este asociată cu o rută specifică și conține metode care definesc acțiunile specifice pe care aplicația le poate efectua, precum obținerea, crearea, actualizarea sau ștergerea datelor.

În dosarul "DTOs" se găsesc clasele DTO (Obiecte de Transfer de Date) utilizate pentru transferul datelor între client și server. Acestea sunt folosite pentru a defini structura și formatul datelor transmise sau primite prin cererile API. DTO-urile furnizează o abstracție a datelor și permit controlul informațiilor trimise sau extrase de la client, evitând expunerea directă a structurii interne a claselor de bază.

Dosarul "Filters" conține clasele care implementează filtrele aplicației. Aceste filtre sunt utilizate pentru a captura și manipula cererile și răspunsurile către și de la server. În cazul proiectului descris, filtrele sunt concepute pentru a prinde și trata excepțiile care apar în dosarul "Data" într-un mod specific, cum ar fi înregistrarea și gestionarea lor corespunzătoare. Filtrele pot fi, de asemenea, utilizate pentru validarea și verificarea cererilor sau pentru a aplica logica suplimentară de securitate.

Dosarul "Util" (Utilities) este destinat să conțină clase de utilitate care oferă funcționalități auxiliare utilizate în diverse părți ale aplicației. În cadrul proiectului, aceste clase sunt folosite pentru gestionarea și realizarea transformărilor între obiectele DTO și clasele de bază. Acestea pot include metode pentru maparea proprietăților între obiectele DTO și obiectele de bază, validarea datelor, conversia de tipuri etc. Utilizarea acestor clase de utilitate contribuie la menținerea unui cod modular și reutilizabil.

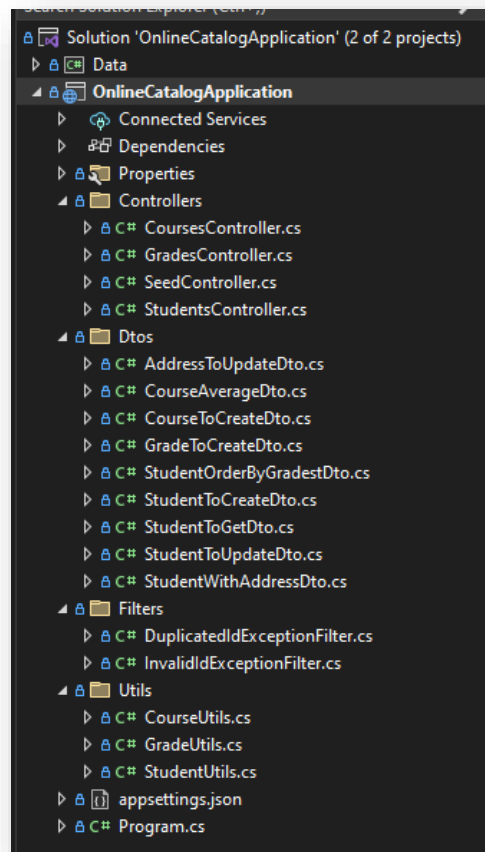


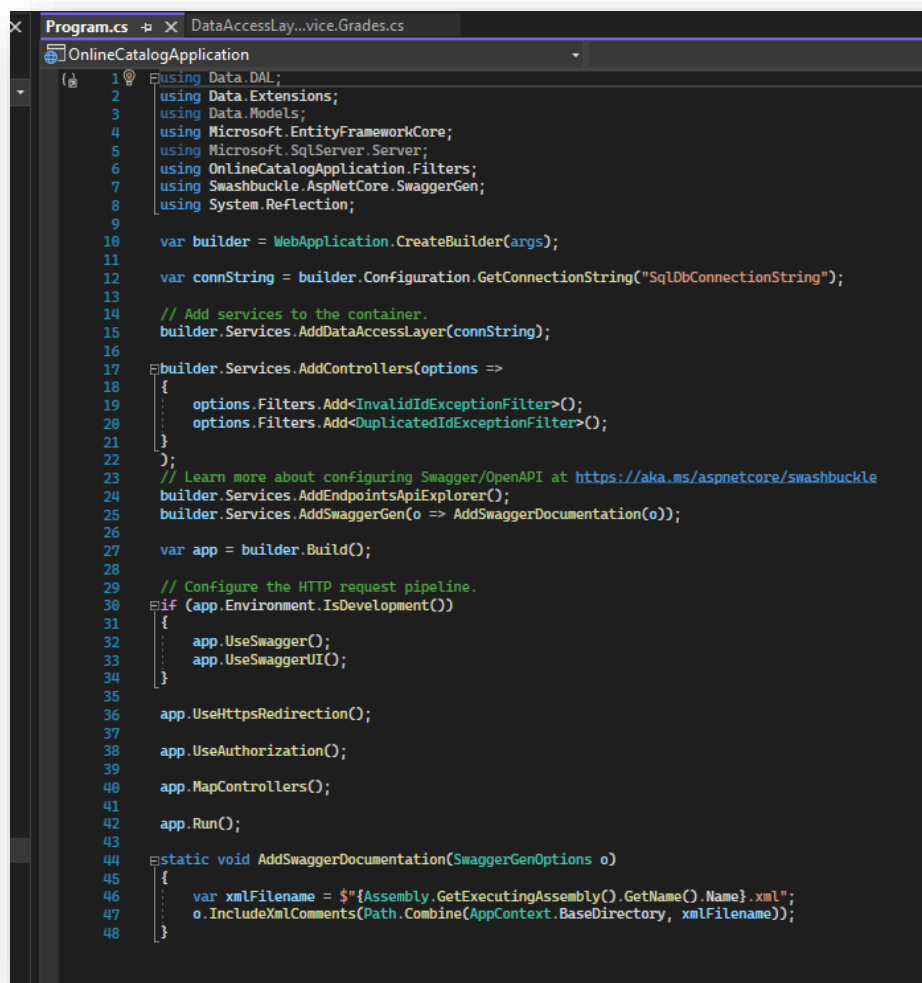
Figura nr. 3.1 Structura – Online Catalog Application

În clasa Program(Figura nr. 3.2), aplicația web folosește principiul injecției de dependențe pentru a crea o legătură între proiectul principal și proiectul Data. Iată cum funcționează acest proces:

- Se definește o variabilă `connString` care obține stringul de conexiune la baza de date din configurările aplicației.
- Se adaugă serviciile necesare în containerul de dependențe utilizând metoda `AddDataAccessLayer(connString)`. Aceasta face ca implementarea interfeței `IDataAccessLayerService` din proiectul Data să fie disponibilă în cadrul aplicației web. Aceasta înseamnă că orice clasă care are nevoie de servicii de acces la date poate solicita o instanță a `IDataAccessLayerService` prin intermediul mecanismului de injecție a dependențelor.
- Se adaugă controller-ele utilizând metoda `AddControllers()`. Controller-ele sunt configurate să utilizeze două filtre personalizate, `InvalidIdExceptionFilter` și

DuplicatedIdExceptionHandler. Aceste filtre vor fi aplicate înainte și după executarea acțiunilor din controller-e pentru a trata excepțiile specifice.

- Se adaugă suport pentru generarea documentației Swagger/OpenAPI utilizând metodele `AddEndpointsApiExplorer()` și `AddSwaggerGen(o => AddSwaggerDocumentation(o))`. Aceasta permite generarea automată a documentației API-ului pe baza adnotărilor XML și a comentariilor din codul sursă.
- În funcție de mediul de dezvoltare, se configurează aplicația pentru a utiliza Swagger și HTTPS redirection prin apelarea metodelor `UseSwagger()` și `UseSwaggerUI()` în mediu de dezvoltare.
- Se configurează pipeline-ul de procesare al cererilor HTTP prin intermediul metodelor `UseHttpsRedirection()`, `UseAuthorization()`, `MapControllers()` și `Run()`. Aceasta asigură că cererile HTTP sunt redirecționate către HTTPS, se aplică autorizarea, se mapează rutele către controller-e și se rulează aplicația.



```
1 using Data.DAL;
2 using Data.Extensions;
3 using Data.Models;
4 using Microsoft.EntityFrameworkCore;
5 using Microsoft.SqlServer.Server;
6 using OnlineCatalogApplication.Filters;
7 using Swashbuckle.AspNetCore.SwaggerGen;
8 using System.Reflection;
9
10 var builder = WebApplication.CreateBuilder(args);
11
12 var connString = builder.Configuration.GetConnectionString("SqlDbConnectionString");
13
14 // Add services to the container.
15 builder.Services.AddDataAccessLayer(connString);
16
17 builder.Services.AddControllers(options =>
18 {
19     options.Filters.Add<InvalidIdExceptionHandler>();
20     options.Filters.Add<DuplicatedIdExceptionHandler>();
21 });
22
23 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
24 builder.Services.AddEndpointsApiExplorer();
25 builder.Services.AddSwaggerGen(o => AddSwaggerDocumentation(o));
26
27 var app = builder.Build();
28
29 // Configure the HTTP request pipeline.
30 if (app.Environment.IsDevelopment())
31 {
32     app.UseSwagger();
33     app.UseSwaggerUI();
34 }
35
36 app.UseHttpsRedirection();
37
38 app.UseAuthorization();
39
40 app.MapControllers();
41
42 app.Run();
43
44 static void AddSwaggerDocumentation(SwaggerGenOptions o)
45 {
46     var xmlFilename = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
47     o.IncludeXmlComments(Path.Combine(AppContext.BaseDirectory, xmlFilename));
48 }
```

Figura nr. 3.2 Program.cs

3.3 Controllers

```
9
10 namespace OnlineCatalogApplication.Controllers
11 {
12     [Route("api/[controller]")]
13     [ApiController]
14     public class StudentsController : ControllerBase
15     {
16         private readonly IDataAccessLayerService dal;
17
18         public StudentsController(IDataAccessLayerService dataAccessLayer)
19         {
20             this.dal = dataAccessLayer;
21         }
22
23         /// <summary>
24         /// Returns all students in the database.
25         /// </summary>
26         /// <returns>A list of students</returns>
27         [HttpGet()]
28         public IEnumerable<StudentToGetDto> GetAllStudents() =>
29             dal.GetAllStudents().Select(s => s.ToDto()).ToList();
30
31         /// <summary>
32         /// Get a student by ID.
33         /// </summary>
34         /// <param name="id">The ID of the student</param>
35         /// <returns>The student data</returns>
36         [HttpGet("/{id}/{id}")]
37         [ProducesResponseType(StatusCodes.Status200OK, Type = typeof(StudentToGetDto))]
38         [ProducesResponseType(StatusCodes.Status400BadRequest, Type = typeof(string))]
39         [ProducesResponseType(StatusCodes.Status404NotFound, Type = typeof(string))]
40         public ActionResult<StudentToGetDto> GetStudentById([Range(1, int.MaxValue)] int id) =>
41             Ok(dal.GetStudentById(id).ToDto());
42
43         /// <summary>
44         /// Create a student.
45         /// </summary>
46         /// <param name="studentToCreate">Student data to create</param>
47         /// <returns>The created student data</returns>
48         [HttpPost]
49         public StudentToGetDto CreateStudent([FromBody] StudentToCreateDto studentToCreate) =>
50             dal.CreateStudent(studentToCreate.ToEntity().ToDto());
51
52         /// <summary>
53         /// Update a student.
```

Figura nr. 3.3 StudentsController

```
SeedController.cs  StudentsController.cs  Program.cs  DataAccessLayerServiceGrade
OnlineCatalogApplication
1 using Data.DAL;
2 using Microsoft.AspNetCore.Http;
3 using Microsoft.AspNetCore.Mvc;
4
5 namespace OnlineCatalogApplication.Controllers
6 {
7     [Route("api/[controller]")]
8     [ApiController]
9     public class SeedController : ControllerBase
10     {
11         private readonly IDataAccessLayerService dal;
12
13         public SeedController(IDataAccessLayerService dataAccessLayer)
14         {
15             this.dal = dataAccessLayer;
16         }
17
18         /// <summary>
19         /// Initialize the database by seeding data.
20         /// </summary>
21         [HttpPost()]
22         public void Seed() => dal.Seed();
23     }
24
25 }
26
```

Figura nr. 3.4 SeedController

```

1  using Data.DAL;
2  using Data.Exceptions;
3  using Microsoft.AspNetCore.Http;
4  using Microsoft.AspNetCore.Mvc;
5  using OnlineCatalogApplication.Dtos;
6  using OnlineCatalogApplication.Utils;
7
8  namespace OnlineCatalogApplication.Controllers
9  {
10     [Route("api/[controller]")]
11     [ApiController]
12     public class CoursesController : ControllerBase
13     {
14         private readonly IDataAccessLayerService dal;
15
16         public CoursesController(IDataAccessLayerService dataAccessLayer)
17         {
18             this.dal = dataAccessLayer;
19         }
20
21         /// <summary>
22         /// Add a new course.
23         /// </summary>
24         /// <param name="courseToCreate">The course to create.</param>
25         /// <returns>The new course.</returns>
26         [HttpPost]
27         [ProducesResponseType(StatusCodes.Status200OK, Type = typeof(string))]
28         [ProducesResponseType(StatusCodes.Status201Created, Type = typeof(string))]
29         public CourseToCreateDto AddCourse([FromBody] CourseToCreateDto courseToCreate) =>
30             dal.AddCourse(courseToCreate.Name).ToDto();
31
32         /// <summary>
33         /// Get all courses.
34         /// </summary>
35         /// <returns>The List of all courses.</returns>
36         [HttpGet]
37         [ProducesResponseType(StatusCodes.Status200OK, Type = typeof(string))]
38         [ProducesResponseType(StatusCodes.Status400BadRequest, Type = typeof(string))]
39         [ProducesResponseType(StatusCodes.Status404NotFound, Type = typeof(string))]
40         public IEnumerable<CourseToCreateDto> GetAllCourses() =>
41             dal.GetAllCourses().Select(s => s.ToDto()).ToList();
42     }
43 }
44
45

```

Figura nr. 3.5 CoursesController

```

8
9  namespace OnlineCatalogApplication.Controllers
10 {
11     [Route("api/[controller]")]
12     [ApiController]
13     public class GradesController : ControllerBase
14     {
15         private readonly IDataAccessLayerService dal;
16
17         public GradesController(IDataAccessLayerService dataAccessLayer)
18         {
19             this.dal = dataAccessLayer;
20         }
21
22         /// <summary>
23         /// Award a grade to a student for a course.
24         /// </summary>
25         /// <param name="grade">The grade to award.</param>
26         /// <returns>The awarded grade.</returns>
27         [HttpPost]
28         [ProducesResponseType(StatusCodes.Status200OK, Type = typeof(GradeToCreateDto))]
29         [ProducesResponseType(StatusCodes.Status400BadRequest, Type = typeof(string))]
30         public GradeToCreateDto AwardGrade([FromBody] GradeToCreateDto gradeToCreate) =>
31             dal.AwardGrade(gradeToCreate.StudentId, gradeToCreate.CourseId, gradeToCreate.Grade).ToDto();
32     }
33 }
34
35

```

Figura nr. 3.6 GradesController

3.3 Dtos

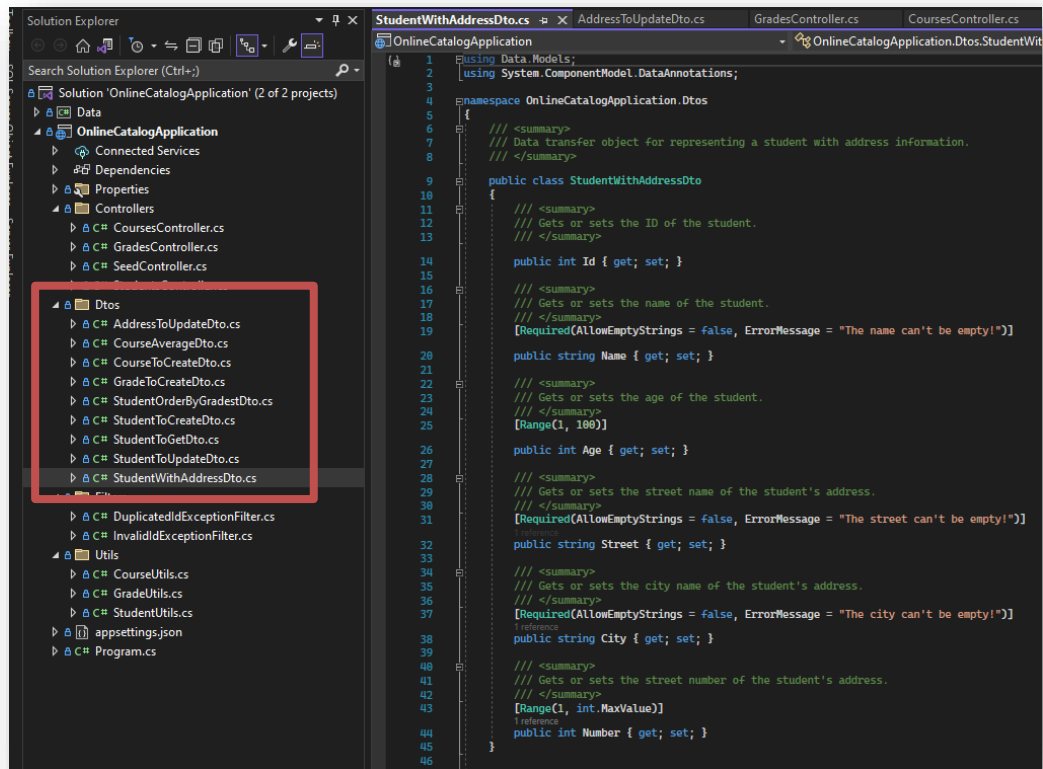


Figura nr. 3.7 StudentWithAddressDto.cs

3.4 Utils

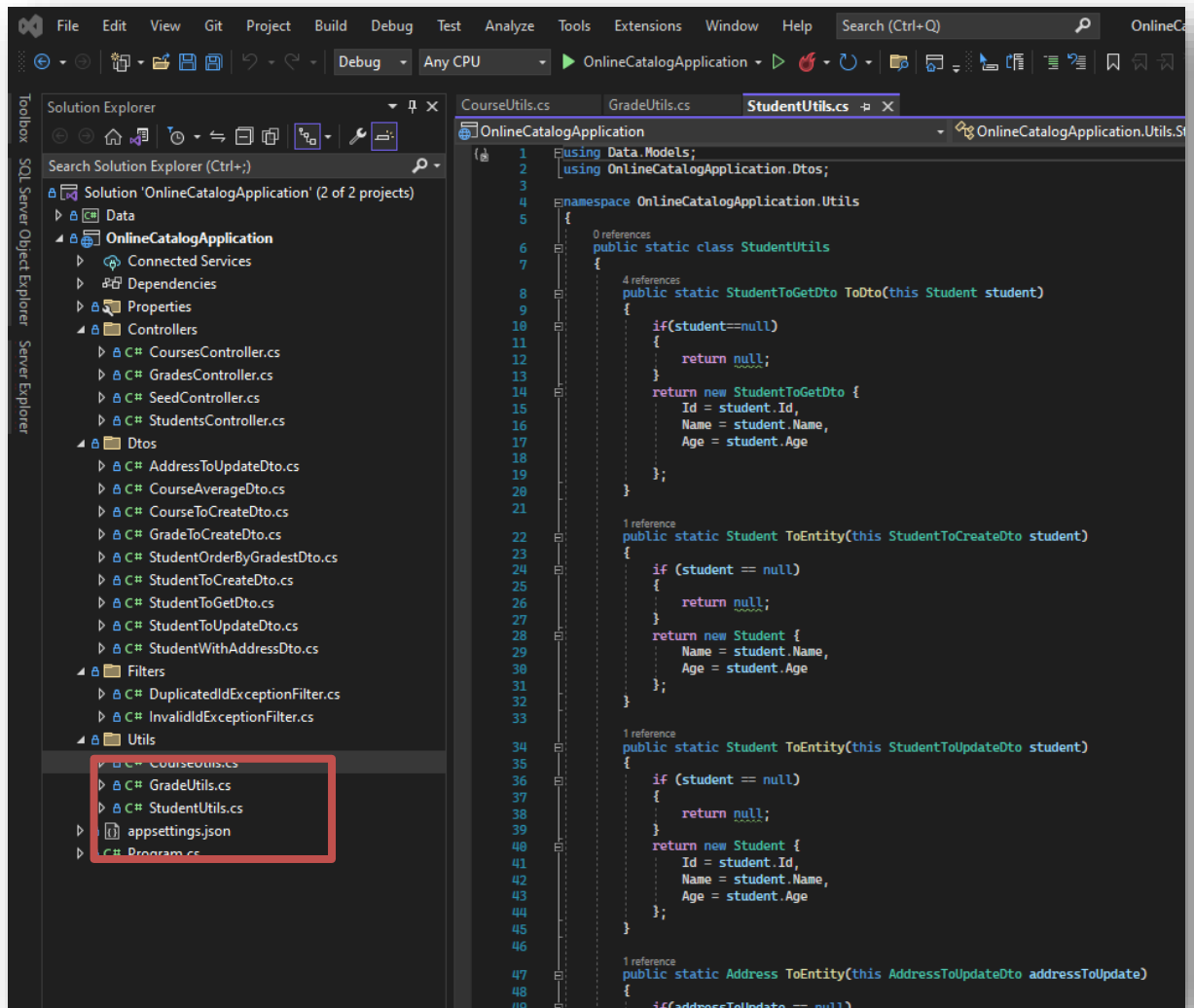


Figura nr. 3.8 CourseUtils.cs

3.4 Filters

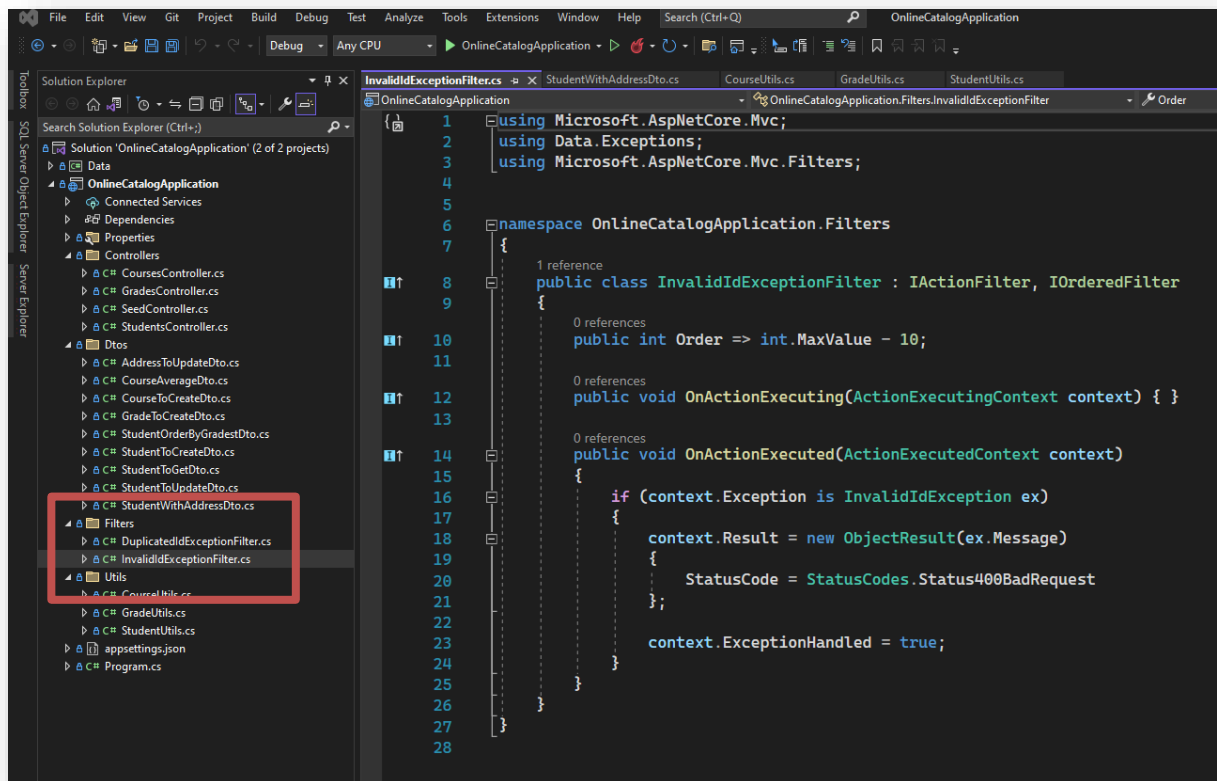


Figura nr. 3.9 InvalidIdExceptionFilter

CONCLUZII

Proiectul descris este o aplicație web bazată pe ASP.NET, care utilizează limbajul C# și urmărește o structură arhitecturală modulară și scalabilă. În cadrul proiectului, se utilizează principii precum injecția de dependențe (dependency injection) și separarea responsabilităților pentru a obține un cod bine organizat și ușor de întreținut.

Prin intermediul limbajului C#, se implementează clasele de controlere (Controllers) care gestionează rutele și cererile primite de la client. Aceste controlere acționează ca intermediari între cererile HTTP și logica de afaceri a aplicației, permițând manipularea și procesarea datelor.

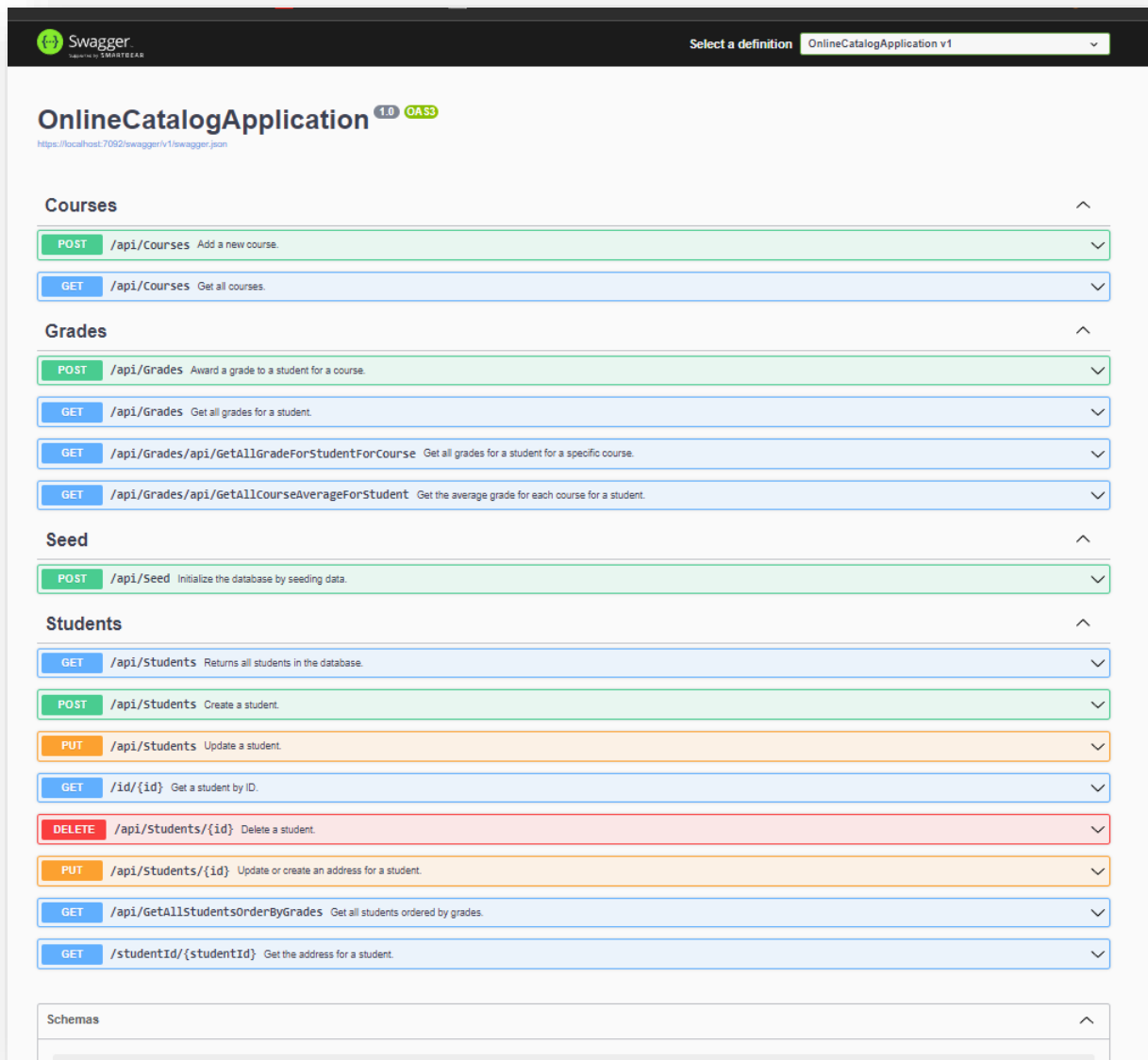
Pentru transferul datelor între client și server, se utilizează obiecte de transfer de date (DTOs). Acestea sunt clase care definesc structura și formatul datelor care sunt trimise sau primite prin intermediul cererilor API. Utilizarea DTO-urilor permite controlul asupra informațiilor transmise către client și extrase din acesta, evitând expunerea directă a structurii interne a claselor de bază.

Aplicația utilizează și filtre (Filters) pentru a captura și trata excepțiile care pot apărea în dosarul "Data". Aceste filtre sunt responsabile de gestionarea și înregistrarea corectă a excepțiilor, oferind o abordare coerentă și sigură în ceea ce privește manipularea erorilor.

Un aspect important al proiectului este utilizarea serviciilor și interfețelor definite în proiectul Data, care sunt adăugate în containerul de dependențe al aplicației web. Aceasta permite controller-elor să solicite o instanță a interfeței `IDataAccessLayerService`, care este implementată în proiectul Data. Această abordare elimină nevoia de a crea manual și de a gestiona obiectele necesare pentru accesul la date și facilitează dezvoltarea modulară și reutilizabilă a aplicației.

În concluzie, proiectul evidențiază utilizarea limbajului C# într-o aplicație web bazată pe ASP.NET, care adoptă o structură modulară și scalabilă prin intermediul utilizării controloarelor, obiectelor de transfer de date, filtrelor și serviciilor de acces la date. Această abordare permite o dezvoltare eficientă, testare și întreținere a aplicației, urmărind cele mai bune practici în dezvoltarea de aplicații web.

IMAGINI CU APLICAȚIA



Courses

POST

/api/Courses

Add a new course.

GET

/api/Courses

Get all courses.

Parameters

No parameters

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7092/api/Courses' \
  -H 'accept: text/plain'
```

Request URL

https://localhost:7092/api/Courses

Server response

Code

Details

200

Response body

```
[
  {
    "name": "Romana"
  },
  {
    "name": "Math"
  },
  {
    "name": "History"
  },
  {
    "name": "English Language and Literature"
  },
  {
    "name": "Geography"
  },
  {
    "name": "Computer Science"
  },
  {
    "name": "Fine Art"
  }
]
```

Response headers

```
content-type: application/json; charset=utf-8
date: Sun, 11 Jun 2023 20:26:06 GMT
server: Kestrel
```

Responses

Code

Description

Links

200

Success

No links

Media type

text/plain

22

Grades

POST

/api/Grades

Award a grade to a student for a course.

GET

/api/Grades

Get all grades for a student.

GET

/api/Grades/api/GetAllGradeForStudentForCourse

Get all grades for a student for a specific course.

GET

/api/Grades/api/GetAllCourseAverageForStudent

Get the average grade for each course for a student.

Parameters

Cancel

Name	Description
<div>studentId</div> <div>integer(\$int32)</div> <div>(query)</div>	The student ID.

2

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7092/api/Grades/api/GetAllCourseAverageForStudent?studentId=2' \
  -H 'accept: text/plain'
```

Request URL

https://localhost:7092/api/Grades/api/GetAllCourseAverageForStudent?studentId=2

Server response

Code	Details
200	<div>Response body</div> <div> <pre>{ "courseId": 3, "averageGrade": 6.25 }, { "courseId": 3, "averageGrade": 6 } }</pre> </div> <div> <div>Download</div> </div>

Response headers

```
content-type: application/json; charset=utf-8
date: Sun, 11 Jun 2023 20:27:27 GMT
server: Kestrel
```

Responses

Code	Description	Links
------	-------------	-------

Students

- GET /api/Students Returns all students in the database.
- POST /api/Students Create a student.
- PUT /api/Students Update a student.
- GET /id/{id} Get a student by ID.
- DELETE /api/Students/{id} Delete a student.
- PUT /api/Students/{id} Update or create an address for a student.
- GET /api/GetAllStudentsOrderByGrades Get all students ordered by grades.
- GET /studentId/{studentId} Get the address for a student.

Parameters

Name

Description

studentId * required

Integer (\$int32)

(path)

The ID of the student

1

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7892/studentId/1' \
  -H 'accept: text/plain'
```

Request URL

https://localhost:7892/studentId/1

Server response

Code

Details

200

Response body

```
{
  "id": 1,
  "name": "Marin Chitac",
  "age": 43,
  "street": "Revoluției",
  "city": "Iasi",
  "number": 32
}
```

Download

24

DELETE
/api/Students/{id}
Delete a student.

PUT
/api/Students/{id}
Update or create an address for a student.

GET
/api/GetAllStudentsOrderByGrades
Get all students ordered by grades.

Parameters
Cancel

No parameters

Execute
Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7092/api/GetAllStudentsOrderByGrades' \
  -H 'Accept: text/plain'
```

Request URL
https://localhost:7092/api/GetAllStudentsOrderByGrades

Server response

Code
Details

200

Response body

```
[
  {
    "studentId": 1,
    "averageGrades": 6.142857142857143
  },
  {
    "studentId": 2,
    "averageGrades": 9.5
  },
  {
    "studentId": 3,
    "averageGrades": 10
  },
  {
    "studentId": 4,
    "averageGrades": 10
  },
  {
    "studentId": 5,
    "averageGrades": 10
  },
  {
    "studentId": 6,
    "averageGrades": 10
  },
  {
    "studentId": 7,
    "averageGrades": 10
  }
]
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Sun, 11 Jun 2023 20:25:23 GMT
server: Kestrel
```

Responses

Code
Description
Links

200