Документация

по

CSCB532 Практика по програмиране и интернет технологии

НА ТЕМА:

Логистична Компания Backend

Изготвил:

Александър Мартинов Стоянов (F099867)

Съдържание

1.Описание на проекта

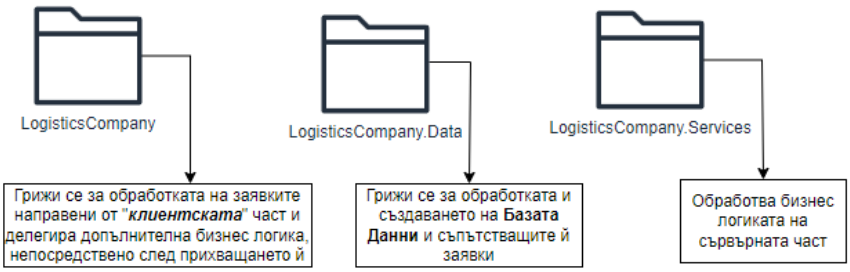
Сървърното приложение за “Логистична Компания“ играе ролята на мост между сървърът и клиента.

По-конкретно, прави връзка с база данни и връща данни, които са провокирани от потребителско взаимодействие с “фронтовата” клиенска част.

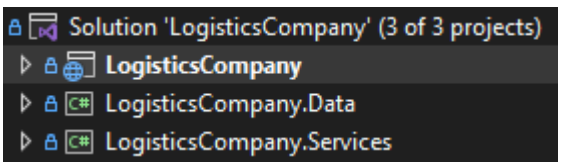
Този модул се обуславя с RESTful архитектура.

2. Архитектура на приложението

Архитектурата на сървърното приложение има фрагментарен характер:



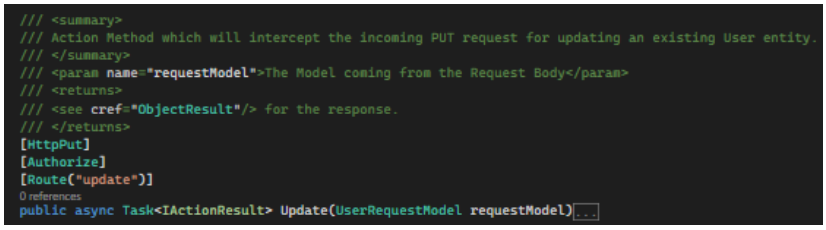
От гледна точка на код:



2.1 LogisticsCompany

Основната задача на LogisticsCompany проекта е да:

1. Прихваща заявките направени от клиента.



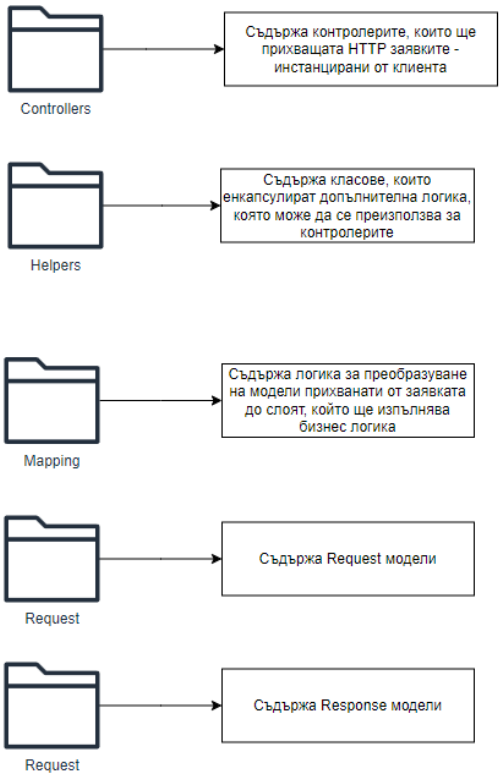
2. Проверява за ауторизиран потребител дали има дадени права за достъпване на ресурсите, от отговора на заявката и дали моделът (ако има такъв) е валиден спрямо своеобразни изисквания:

```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}

var header = HttpContext.Request.Headers["Authorization"];

if (!IsAuthorized("Admin", header))
{
    return Unauthorized();
}
```

3. Преобразява данните получени от HTTP заявката до модел годен за обработка от бизнес логиката.

```
var dto = _mapper.Map<UserRequestModel, UserDto>(requestModel);

await _commandService.Update(dto);

return Ok("");
```

Инфраструктура на този модул бива от следния порядък:



2. 2 LogisticsCompany.Data

В сравнение с предишния композиционен модул, този е съставен които обхващат концепции наложени в софтуерната архитектура като DesignPatterns.

Както и създаването на база данни, чрез познати сурови Structured Query Language заявки.

Създаване на базата

Създаването на базата данни става от строго централизирано място, което се явява посредник между приложението и активния SQL Server.

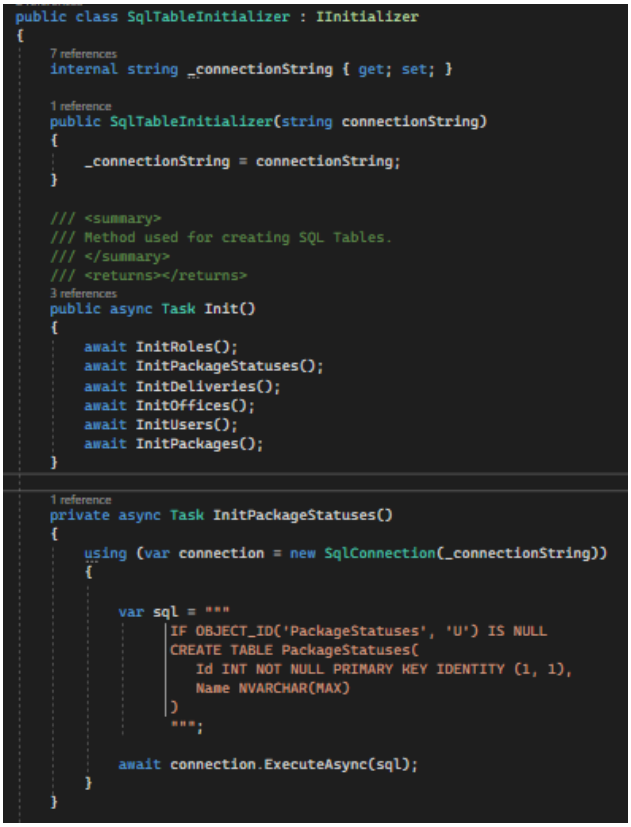
```
/// <summary>
/// Class used for creating the Database Context.
/// </summary>
19 references
public class LogisticsCompanyContext
{
    3 references
    internal IConfiguration _configuration { get; set; }

    /// <summary>
    /// Creates a <see cref="LogisticsCompanyContext" /> instance with the passed <paramref name="configuration"/>
    /// </summary>
    /// <param name="configuration"></param>
    0 references
    public LogisticsCompanyContext(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    /// <summary>
    /// Database constructor for context for an SQL Server.
    /// </summary>
    /// <param name="configuration"></param>
    /// <param name="dbFactory"></param>
    /// <param name="tableFactory"></param>
    0 references
    public LogisticsCompanyContext(IConfiguration configuration, SqlDbFactory dbFactory)
    {
        _configuration = configuration;
        var connectionString = GetConnectionString();

        ConstructDatabase(dbFactory, connectionString);
    }

    1 reference
    private static void ConstructDatabase(SqlDbFactory dbFactory, string connectionString)...

    /// <summary>
    /// Method that retrieves the connection string from the Application Settings.
    /// </summary>
    3 references
    public string GetConnectionString()...
}
```

"***ConstructDatabase***" метода ползва Abstract Factory Design Pattern за:

- Създаване на базата:

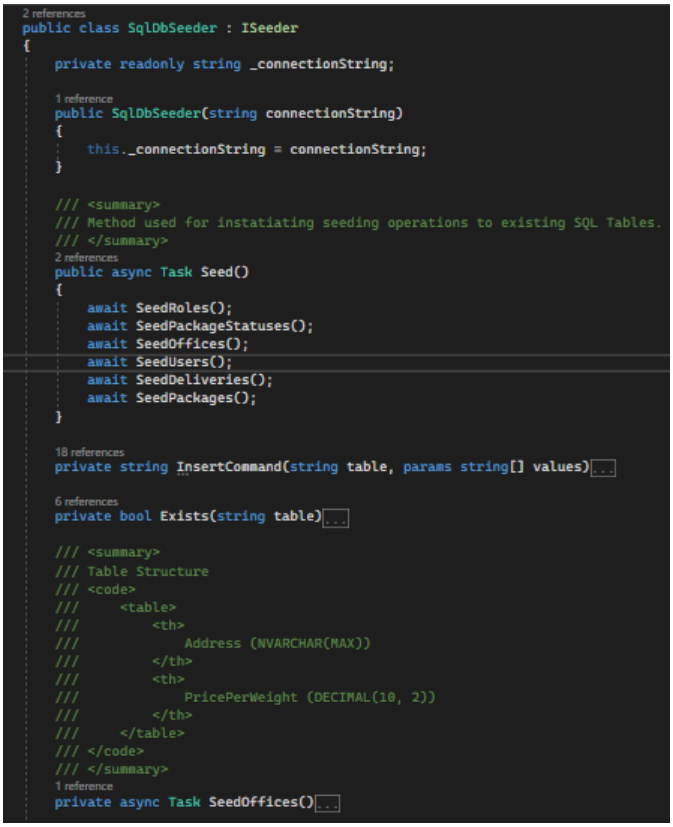
```
/// <summary>
/// Method used constructing the Database through a newly created SQL Connection.
/// </summary>
1 reference
public async Task Init()
{
    using (var connection = new SqlConnection(_connectionString))
    {
        var sql = $"IF NOT EXISTS (SELECT * FROM sys.databases WHERE name = 'LogisticsCompany') CREATE DATABASE LogisticsCompany";
        await connection.ExecuteAsync(sql);

        sql = "USE LogisticsCompany";
        await connection.ExecuteAsync(sql);
    }
}
```

- Създаване на схемата на базата:



```
public class SqlTableInitializer : IInitializer
{
    7 references
    internal string _connectionString { get; set; }

    1 reference
    public SqlTableInitializer(string connectionString)
    {
        _connectionString = connectionString;
    }

    /// <summary>
    /// Method used for creating SQL Tables.
    /// </summary>
    /// <returns></returns>
    3 references
    public async Task Init()
    {
        await InitRoles();
        await InitPackageStatuses();
        await InitDeliveries();
        await InitOffices();
        await InitUsers();
        await InitPackages();
    }

    1 reference
    private async Task InitPackageStatuses()
    {
        using (var connection = new SqlConnection(_connectionString))
        {

            var sql = """
                IF OBJECT_ID('PackageStatuses', 'U') IS NULL
                CREATE TABLE PackageStatuses(
                   Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
                   Name NVARCHAR(MAX)
                )
                """;

            await connection.ExecuteAsync(sql);
        }
    }
```

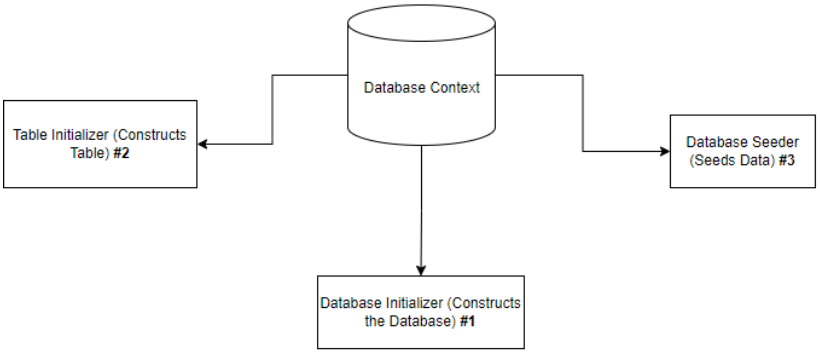
- Наливането на готови данни:



```
2 references
public class SqlDbSeeder : ISeeder
{
    private readonly string _connectionString;

    1 reference
    public SqlDbSeeder(string connectionString)
    {
        this._connectionString = connectionString;
    }

    /// <summary>
    /// Method used for instatiating seeding operations to existing SQL Tables.
    /// </summary>
    2 references
    public async Task Seed()
    {
        await SeedRoles();
        await SeedPackageStatuses();
        await SeedOffices();
        await SeedUsers();
        await SeedDeliveries();
        await SeedPackages();
    }

    18 references
    private string InsertCommand(string table, params string[] values)...

    6 references
    private bool Exists(string table)...

    /// <summary>
    /// Table Structure
    /// <code>
    ///     <table>
    ///         <th>
    ///             Address (NVARCHAR(MAX))
    ///         </th>
    ///         <th>
    ///             PricePerWeight (DECIMAL(10, 2))
    ///         </th>
    ///     </table>
    /// </code>
    /// </summary>
    1 reference
    private async Task SeedOffices()...
```

Целта е да се позволи създаването и на други типове бази данни - като нерелационни

От гледна точка на архитектура, нещата изглеждат по следния начин:



Създаването на заявки

Заявките се създават чрез допълнителна SqlBuilder абстракция, която има за цел да:

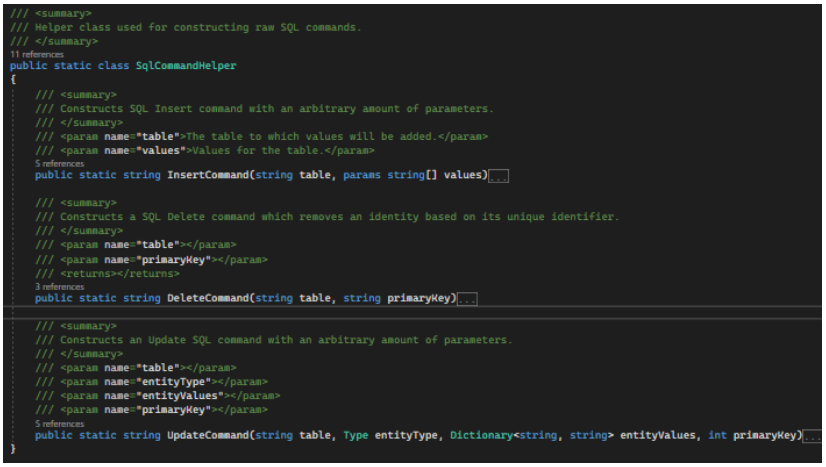
- Конструира Select заявки.
- Създаването на по-комплексни заявки от порядъка на JoinClauses.

Целта е заявката да се създава чрез извикването на метода с цел по-добра четимост. Ползвайки [Builder Design Pattern-a]

Конструкцията на тази абстракция се състои от следните компоненти:

Създаването на команди

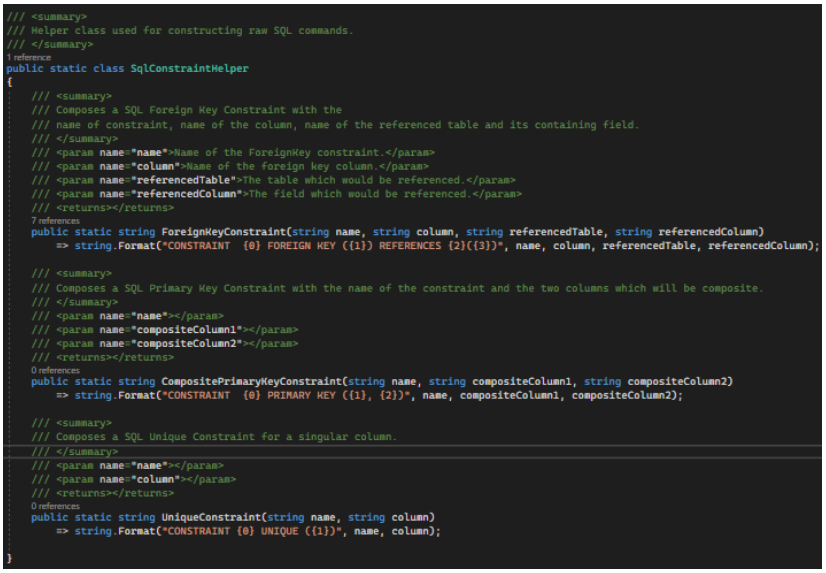
Създаването на командите е от значително по-примитивно естество, като това става чрез инструменти за обработка на символни низове, която се позволява от езика



```
/// <summary>
/// Helper class used for constructing raw SQL commands.
/// </summary>
11 references
public static class SqlCommandHelper
{
    /// <summary>
    /// Constructs SQL Insert command with an arbitrary amount of parameters.
    /// </summary>
    /// <param name="table">The table to which values will be added.</param>
    /// <param name="values">Values for the table.</param>
    5 references
    public static string InsertCommand(string table, params string[] values)...

    /// <summary>
    /// Constructs a SQL Delete command which removes an identity based on its unique identifier.
    /// </summary>
    /// <param name="table"></param>
    /// <param name="primaryKey"></param>
    /// <returns></returns>
    3 references
    public static string DeleteCommand(string table, string primaryKey)...

    /// <summary>
    /// Constructs an Update SQL command with an arbitrary amount of parameters.
    /// </summary>
    /// <param name="table"></param>
    /// <param name="entityType"></param>
    /// <param name="entityValues"></param>
    /// <param name="primaryKey"></param>
    5 references
    public static string UpdateCommand(string table, Type entityType, Dictionary<string, string> entityValues, int primaryKey)...
}
```

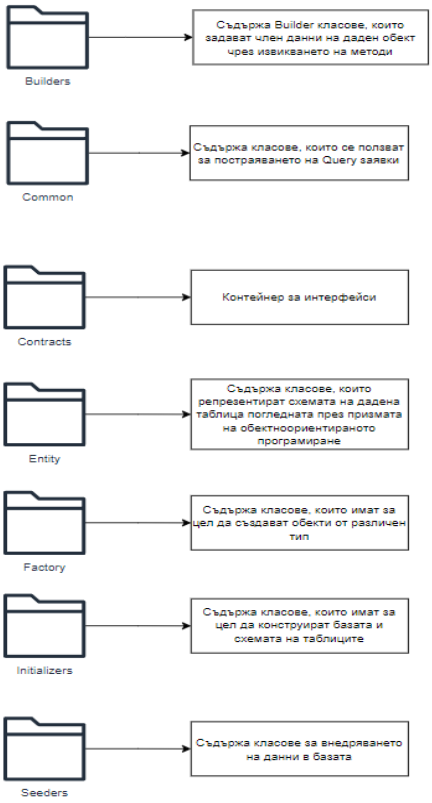
Създаването на ограничения

Ограниченията се създават от същия порядък:



```
/// <summary>
/// Helper class used for constructing raw SQL commands.
/// </summary>
1 reference
public static class SqlConstraintHelper
{
    /// <summary>
    /// Composes a SQL Foreign Key Constraint with the
    /// name of constraint, name of the column, name of the referenced table and its containing field.
    /// </summary>
    /// <param name="name">Name of the ForeignKey constraint.</param>
    /// <param name="column">Name of the foreign key column.</param>
    /// <param name="referencedTable">The table which would be referenced.</param>
    /// <param name="referencedColumn">The field which would be referenced.</param>
    /// <returns></returns>
    7 references
    public static string ForeignKeyConstraint(string name, string column, string referencedTable, string referencedColumn)
        => string.Format("CONSTRAINT  {0} FOREIGN KEY ({1}) REFERENCES {2}({3})", name, column, referencedTable, referencedColumn);

    /// <summary>
    /// Composes a SQL Primary Key Constraint with the name of the constraint and the two columns which will be composite.
    /// </summary>
    /// <param name="name"></param>
    /// <param name="compositeColumn1"></param>
    /// <param name="compositeColumn2"></param>
    /// <returns></returns>
    0 references
    public static string CompositePrimaryKeyConstraint(string name, string compositeColumn1, string compositeColumn2)
        => string.Format("CONSTRAINT  {0} PRIMARY KEY ({1}, {2})", name, compositeColumn1, compositeColumn2);

    /// <summary>
    /// Composes a SQL Unique Constraint for a singular column.
    /// </summary>
    /// <param name="name"></param>
    /// <param name="column"></param>
    /// <returns></returns>
    0 references
    public static string UniqueConstraint(string name, string column)
        => string.Format("CONSTRAINT {0} UNIQUE ({1})", name, column);

}
```

Гледайки от призмата на инфраструктурата, нещата изглеждат по следния начин:



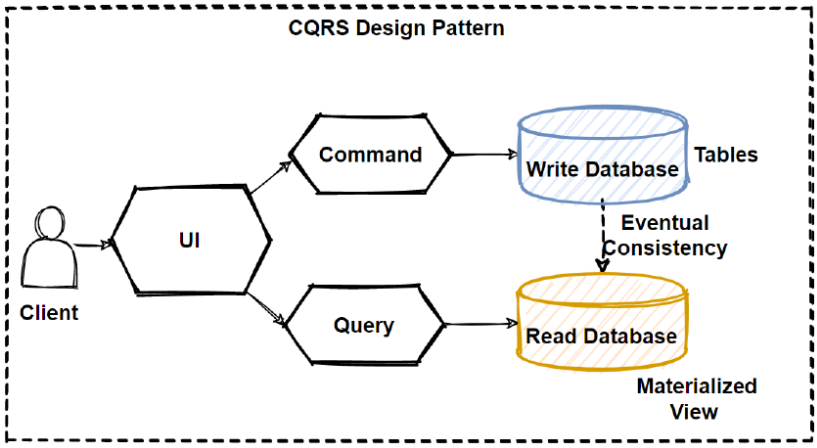
2.3 LogisticsCompany.Services

Този модул има за цел да обработки бизнес логиката на приложението.

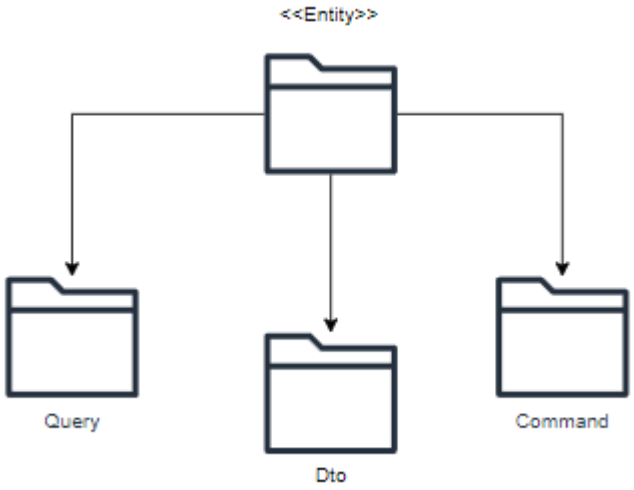
Като от гледната точка на задачи, нещата се делегират чрез:

- Сървиси, които обработват само заявки.
- Сървиси, които обработват само командите.

Или още така наречено Command Query Responsibility Segregation:



Инфраструктурно, нещата в този модул са композирани по следния начин:



3. Описание на ползваните технологии

- SQL Server
- AutoMapper
- Dapper
- JWT Identity Model
- BCrypt