BÁO CÁO ĐỒ ÁN CUỐI KỲ

ỨNG DỤNG QUẢN LÝ SINH VIÊN

Nguyễn Nam Khánh _ 23110183

THÔNG TIN CHUNG

Tên đề tài: Xây dựng ứng dụng Quản lý Sinh viên

Nền tảng: Desktop Application

Ngôn ngữ lập trình: C# .NET Framework 4.7.2

Công nghệ: Windows Forms, Entity Framework 6

Cơ sở dữ liệu: Microsoft SQL Server

Kiến trúc: Mô hình 3 lớp (Entities – DAL – BLL – GUI)

MỤC LỤC

---

1. GIỚI THIỆU

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, việc ứng dụng các giải pháp phần mềm vào quản lý giáo dục đã trở thành xu hướng tất yếu. Công tác quản lý sinh viên là một trong những hoạt động quan trọng của các cơ sở giáo dục, đòi hỏi sự chính xác, nhanh chóng và hiệu quả trong việc lưu trữ, truy xuất và xử lý thông tin.

Đề tài "Xây dựng ứng dụng Quản lý Sinh viên" được thực hiện nhằm xây dựng một giải pháp phần mềm desktop toàn diện, hỗ trợ công tác quản lý thông tin sinh viên, lớp học, môn học, điểm số cũng như các báo cáo thống kê liên quan. Ứng dụng được phát triển trên nền tảng .NET Framework sử dụng ngôn ngữ C#, áp dụng các công nghệ hiện đại như Entity Framework cho tầng truy cập dữ liệu, lập trình bất đồng bộ để tối ưu hiệu năng, và Dependency Injection để giảm sự phụ thuộc giữa các thành phần.

Báo cáo này trình bày chi tiết về quá trình phân tích, thiết kế và cài đặt hệ thống, đồng thời đánh giá kết quả đạt được cũng như hướng phát triển của đề tài trong tương lai.

---

2. MỤC TIÊU ĐỀ TÀI

2.1. Mục tiêu chung

Đề tài hướng đến việc xây dựng một ứng dụng quản lý sinh viên hoàn chỉnh, có khả năng đáp ứng các nhu cầu cơ bản trong công tác quản lý giáo vụ của một cơ sở đào tạo. Ứng dụng cần đảm bảo tính chính xác, ổn định và dễ dàng trong việc bảo trì, mở rộng.

2.2. Mục tiêu cụ thể

Về mặt chức năng, hệ thống cần thực hiện đầy đủ các nghiệp vụ quản lý thông tin sinh viên bao gồm thêm, sửa, xóa và tìm kiếm; quản lý thông tin lớp học và môn học; quản lý điểm số của sinh viên theo từng môn học; cũng như cung cấp các báo cáo thống kê như danh sách sinh viên theo lớp, xếp hạng sinh viên theo điểm trung bình, và thống kê theo các tiêu chí khác nhau.

Về mặt kỹ thuật, đề tài tập trung vào việc áp dụng kiến trúc 3 lớp (Three-tier Architecture) để tách biệt rõ ràng giữa tầng giao diện người dùng, tầng xử lý nghiệp vụ và tầng truy cập dữ liệu. Điều này giúp hệ thống dễ dàng bảo trì, mở rộng và tái sử dụng code. Đề tài cũng đặt mục tiêu áp dụng Entity Framework 6 làm công cụ ORM (Object-Relational Mapping) cho phép thao tác với cơ sở dữ liệu thông qua các đối tượng .NET, giảm thiểu việc viết câu lệnh SQL thủ công và tăng tính bảo mật cho ứng dụng.

Bên cạnh đó, đề tài hướng đến việc áp dụng lập trình bất đồng bộ (Asynchronous Programming) sử dụng cơ chế async/await nhằm cải thiện khả năng đáp ứng của giao diện người dùng, đặc biệt trong các thao tác truy cập cơ sở dữ liệu có thể mất nhiều thời gian. Việc ứng dụng Dependency Injection thông qua thư viện Microsoft.Extensions.DependencyInjection cũng được đặt ra như

một mục tiêu quan trọng, giúp giảm sự phụ thuộc giữa các thành phần và tăng khả năng kiểm thử của hệ thống.

Cuối cùng, đề tài cần đảm bảo tính toàn vẹn dữ liệu thông qua việc áp dụng cơ chế Transaction trong các thao tác phức tạp, đảm bảo rằng tất cả các thao tác trong một giao dịch đều được thực hiện thành công hoặc không có thao tác nào được thực hiện.

---

3. CÔNG NGHỆ VÀ CÔNG CỤ SỬ DỤNG

3.1. Nền tảng và ngôn ngữ lập trình

Ứng dụng được phát triển trên nền tảng .NET Framework 4.7.2, một framework mạnh mẽ và ổn định của Microsoft, cung cấp đầy đủ các thư viện và công cụ cần thiết cho việc phát triển ứng dụng desktop. Ngôn ngữ lập trình được sử dụng là C#, một ngôn ngữ hướng đối tượng hiện đại, có cú pháp rõ ràng và được hỗ trợ tốt bởi Visual Studio.

3.2. Công nghệ giao diện

Windows Forms (WinForms) được lựa chọn làm công nghệ xây dựng giao diện người dùng. Mặc dù WPF (Windows Presentation Foundation) là công nghệ mới hơn, WinForms vẫn được ưa chuộng trong các dự án yêu cầu sự đơn giản, ổn định và tốc độ phát triển nhanh. WinForms cung cấp đầy đủ các controls cơ bản như Button, TextBox, ComboBox, DataGridView, giúp xây dựng giao diện trực quan và dễ sử dụng.

3.3. Entity Framework 6

Entity Framework 6 (EF6) là công cụ ORM (Object-Relational Mapping) được sử dụng trong tầng truy cập dữ liệu. EF6 cho phép ánh xạ các bảng trong cơ sở dữ liệu thành các lớp đối tượng trong C#, giúp lập trình viên thao tác với dữ liệu thông qua các đối tượng thay vì viết các câu lệnh SQL trực tiếp. Điều này không chỉ giảm thiểu lỗi do viết SQL sai cú pháp mà còn tăng tính bảo mật cho ứng dụng thông qua việc tự động tham số hóa các truy vấn, ngăn chặn các cuộc tấn công SQL Injection.

Entity Framework hỗ trợ ba phương pháp tiếp cận chính: Database First, Model First và Code First. Trong đề tài này, phương pháp Database First được áp dụng, nghĩa là cơ sở dữ liệu được thiết kế trước, sau đó các lớp Entity được sinh ra hoặc viết thủ công để ánh xạ với cấu trúc database. Phương pháp này phù hợp khi cấu trúc database đã được xác định rõ ràng và cần sự kiểm soát chặt chẽ về thiết kế cơ sở dữ liệu.

3.4. LINQ (Language Integrated Query)

LINQ được sử dụng rộng rãi trong toàn bộ ứng dụng để truy vấn dữ liệu từ các collection và từ Entity Framework. LINQ cung cấp cú pháp truy vấn tích hợp ngay trong ngôn ngữ C#, giúp code trở nên ngắn gọn, dễ đọc và dễ bảo trì hơn so với việc sử dụng các vòng lặp truyền thống.

3.5. Lập trình bất đồng bộ (Async/Await)

Trong các ứng dụng desktop, việc thực hiện các thao tác I/O như truy cập cơ sở dữ liệu, đọc/ghi file có thể mất nhiều thời gian. Nếu các thao tác này được thực hiện đồng bộ trên UI thread, giao diện người dùng sẽ bị đóng băng (freeze), gây trải nghiệm không tốt cho người dùng. Để giải quyết vấn đề này, đề tài áp dụng lập trình bất đồng bộ sử dụng từ khóa async và await.

Khi một phương thức được đánh dấu async, nó có thể chứa các lệnh await để tạm dừng thực thi tại điểm đó mà không chặn thread hiện tại. Trong thời gian chờ đợi, UI thread được giải phóng để xử lý các sự kiện khác, giúp giao diện vẫn phản hồi mượt mà. Khi thao tác bất đồng bộ hoàn thành, phương thức sẽ tiếp tục thực thi từ điểm await. Tất cả các phương thức truy cập cơ sở dữ liệu trong tầng DAL và BLL đều được triển khai dưới dạng bất đồng bộ, sử dụng các phương thức như ToListAsync(), FindAsync(), SaveChangesAsync() của Entity Framework.

3.6. Dependency Injection

Dependency Injection (DI) là một design pattern quan trọng trong phát triển phần mềm hiện đại, giúp giảm sự phụ thuộc chặt chẽ (tight coupling) giữa các thành phần trong hệ thống. Thay vì một class tự tạo ra các đối tượng phụ thuộc của nó, các đối tượng này sẽ được "inject" (tiêm) vào từ bên ngoài thông qua constructor hoặc property.

Trong đề tài này, thư viện Microsoft.Extensions.DependencyInjection được sử dụng để cấu hình và quản lý các dependency. Tại điểm khởi đầu của ứng dụng (file Program.cs), một ServiceCollection được tạo ra và các service được đăng ký với container. Các Repository được đăng ký với interface tương ứng, các Service tương tự, và cuối cùng các Form cũng được đăng ký. Khi cần sử dụng một Form, thay vì tạo trực tiếp bằng từ khóa new, Form sẽ được lấy từ ServiceProvider, và tất cả các dependency của nó sẽ được tự động inject.

Lợi ích của DI bao gồm khả năng dễ dàng thay đổi implementation mà không cần sửa code của client, dễ dàng viết unit test thông qua việc mock các dependency, và code trở nên rõ ràng hơn về các phụ thuộc của nó.

3.7. Transaction Management

Trong các hệ thống quản lý dữ liệu, việc đảm bảo tính toàn vẹn dữ liệu là vô cùng quan trọng. Có những nghiệp vụ yêu cầu thực hiện nhiều thao tác cơ sở dữ liệu liên tiếp, và tất cả các thao tác này phải được thực hiện thành công hoặc không có thao tác nào được thực hiện. Đây là nơi Transaction phát huy vai trò.

Transaction là một đơn vị công việc logic bao gồm một hoặc nhiều thao tác cơ sở dữ liệu, tuân theo nguyên tắc ACID (Atomicity, Consistency, Isolation, Durability). Entity Framework hỗ trợ Transaction thông qua phương thức Database.BeginTransaction(). Trong đề tài này, Transaction

được áp dụng trong tình huống nhập điểm cho nhiều sinh viên cùng một lúc. Nếu có bất kỳ lỗi nào xảy ra trong quá trình nhập liệu, toàn bộ giao dịch sẽ được rollback, đảm bảo dữ liệu không bị hỏng.

3.8. Hệ quản trị cơ sở dữ liệu

Microsoft SQL Server được lựa chọn làm hệ quản trị cơ sở dữ liệu cho ứng dụng. SQL Server là một trong những RDBMS phổ biến nhất, được tích hợp tốt với .NET Framework và Entity Framework. SQL Server cung cấp nhiều tính năng mạnh mẽ như stored procedures, triggers, views, và các công cụ quản lý đồ họa thông qua SQL Server Management Studio (SSMS).

---

4. THIẾT KẾ CƠ SỞ DỮ LIỆU

4.1. Tổng quan

Cơ sở dữ liệu StudentManagementDB được thiết kế theo mô hình quan hệ (Relational Model), bao gồm bốn bảng chính: Classes, Students, Subjects và Scores. Thiết kế này đảm bảo tính chuẩn hóa dữ liệu, giảm thiểu sự trùng lặp và duy trì tính toàn vẹn thông qua các ràng buộc khóa ngoại.

4.2. Bảng Classes (Lớp học)

Bảng Classes lưu trữ thông tin về các lớp học trong trường. Mỗi bản ghi trong bảng đại diện cho một lớp học cụ thể.

Cấu trúc bảng:

- ClassId (INT, PRIMARY KEY, IDENTITY): Mã định danh duy nhất của lớp học, được tự động tăng.
- ClassName (NVARCHAR(100), NOT NULL, UNIQUE): Tên lớp học, không được trùng lặp.
- Description (NVARCHAR(500), NULL): Mô tả về lớp học, có thể để trống.

Ràng buộc UNIQUE trên cột ClassName đảm bảo không có hai lớp nào trùng tên, giúp tránh nhầm lẫn trong quản lý.

4.3. Bảng Students (Sinh viên)

Bảng Students lưu trữ thông tin cá nhân của sinh viên, là bảng trung tâm của hệ thống.

Cấu trúc bảng:

- StudentId (INT, PRIMARY KEY, IDENTITY): Mã định danh duy nhất của sinh viên.
- StudentName (NVARCHAR(200), NOT NULL): Họ và tên đầy đủ của sinh viên.

- DateOfBirth (DATE, NOT NULL): Ngày sinh của sinh viên.
- Gender (NVARCHAR(10), NOT NULL): Giới tính của sinh viên.
- ClassId (INT, NOT NULL, FOREIGN KEY): Khóa ngoại tham chiếu đến bảng Classes, xác định sinh viên thuộc lớp nào.

Ràng buộc khóa ngoại giữa Students.ClassId và Classes.ClassId đảm bảo rằng mỗi sinh viên phải thuộc về một lớp học hợp lệ tồn tại trong hệ thống. Khi một lớp bị xóa, các sinh viên thuộc lớp đó cũng sẽ bị xóa theo (CASCADE DELETE), đảm bảo tính nhất quán của dữ liệu.

4.4. Bảng Subjects (Môn học)

Bảng Subjects lưu trữ thông tin về các môn học được giảng dạy trong trường.

Cấu trúc bảng:

- SubjectId (INT, PRIMARY KEY, IDENTITY): Mã định danh duy nhất của môn học.
- SubjectName (NVARCHAR(200), NOT NULL, UNIQUE): Tên môn học, không được trùng lặp.
- Credit (INT, NOT NULL, CHECK > 0): Số tín chỉ của môn học, phải là số dương.

Ràng buộc CHECK trên cột Credit đảm bảo rằng số tín chỉ luôn là một giá trị dương, phù hợp với quy định của hệ thống giáo dục.

4.5. Bảng Scores (Điểm)

Bảng Scores lưu trữ điểm số của sinh viên cho từng môn học, thể hiện mối quan hệ nhiều-nhiều giữa Students và Subjects.

Cấu trúc bảng:

- ScoreId (INT, PRIMARY KEY, IDENTITY): Mã định danh duy nhất của bản ghi điểm.
- StudentId (INT, NOT NULL, FOREIGN KEY): Khóa ngoại tham chiếu đến bảng Students.
- SubjectId (INT, NOT NULL, FOREIGN KEY): Khóa ngoại tham chiếu đến bảng Subjects.
- ScoreValue (DECIMAL(4,2), NOT NULL, CHECK >= 0 AND <= 10): Điểm số, nằm trong khoảng từ 0 đến 10.

Ràng buộc UNIQUE trên cặp (StudentId, SubjectId) đảm bảo rằng mỗi sinh viên chỉ có một điểm duy nhất cho mỗi môn học, tránh tình trạng nhập trùng lặp. Ràng buộc CHECK trên ScoreValue đảm bảo điểm số nằm trong thang điểm hợp lệ.

4.6. Quan hệ giữa các bảng

Cơ sở dữ liệu được thiết kế với các quan hệ sau:

Quan hệ 1-N giữa Classes và Students: Một lớp học có thể có nhiều sinh viên, nhưng mỗi sinh viên chỉ thuộc về một lớp. Quan hệ này được thể hiện thông qua khóa ngoại ClassId trong bảng Students tham chiếu đến ClassId trong bảng Classes.

Quan hệ N-M giữa Students và Subjects thông qua bảng Scores: Một sinh viên có thể học nhiều môn, một môn có thể được nhiều sinh viên học. Quan hệ nhiều-nhiều này được thực hiện thông qua bảng trung gian Scores, trong đó mỗi bản ghi đại diện cho điểm của một sinh viên trong một môn học cụ thể.

Thiết kế này tuân theo các nguyên tắc chuẩn hóa cơ sở dữ liệu, đạt mức chuẩn hóa thứ ba (3NF), giúp giảm thiểu dư thừa dữ liệu và đảm bảo tính toàn vẹn tham chiếu.

---

5. KIẾN TRÚC HỆ THỐNG

5.1. Mô hình 3 lớp

Ứng dụng được xây dựng theo kiến trúc 3 lớp (Three-tier Architecture), một mô hình thiết kế phần mềm phổ biến, tách biệt ứng dụng thành ba tầng logic riêng biệt: Presentation Layer (GUI), Business Logic Layer (BLL), và Data Access Layer (DAL). Ngoài ra, còn có tầng Entities để định nghĩa các thực thể dữ liệu.

5.2. Tầng Entities (StudentManagement.Entities)

Tầng Entities chứa các lớp đại diện cho các thực thể trong cơ sở dữ liệu. Mỗi bảng trong database tương ứng với một lớp Entity. Các lớp này chỉ chứa các thuộc tính (properties) ánh xạ với các cột trong bảng và các navigation properties để thể hiện quan hệ giữa các bảng.

Trong project này, các Entity được định nghĩa sử dụng Data Annotations để chỉ định các ràng buộc như khóa chính, độ dài tối đa, required fields. Các lớp Entity bao gồm: Class.cs (ánh xạ bảng Classes), Student.cs (ánh xạ bảng Students), Subject.cs (ánh xạ bảng Subjects), và Score.cs (ánh xạ bảng Scores).

Các navigation properties như Student.Class hoặc Class.Students cho phép dễ dàng truy cập các đối tượng liên quan mà không cần viết các câu lệnh JOIN phức tạp. Entity Framework sẽ tự động xử lý việc load các đối tượng liên quan này khi cần thiết.

5.3. Tầng Data Access Layer (StudentManagement.DAL)

Tầng DAL chịu trách nhiệm tương tác trực tiếp với cơ sở dữ liệu thông qua Entity Framework. Tầng này bao gồm DbContext và các Repository classes.

DbContext (StudentDbContext.cs): Là lớp trung tâm của Entity Framework, đại diện cho một phiên làm việc với cơ sở dữ liệu. StudentDbContext kế thừa từ DbContext và chứa các DbSet

cho mỗi Entity (DbSet<Class>, DbSet<Student>, DbSet<Subject>, DbSet<Score>). Mỗi DbSet đại diện cho một bảng trong database và cung cấp các phương thức để truy vấn và lưu dữ liệu. Trong OnModelCreating, các ràng buộc và quan hệ có thể được cấu hình sử dụng Fluent API nếu cần.

Repository Pattern: Để tách biệt logic truy cập dữ liệu và tăng khả năng tái sử dụng code, pattern Repository được áp dụng. Mỗi Entity có một Repository tương ứng với một interface định nghĩa các phương thức truy cập dữ liệu cơ bản (CRUD operations). Ví dụ, IStudentRepository định nghĩa các phương thức như GetAllAsync(), GetByIdAsync(), AddAsync(), UpdateAsync(), DeleteAsync(). Class StudentRepository implement interface này và chứa code cụ thể để tương tác với DbContext.

Tất cả các phương thức trong Repository đều là bất đồng bộ (async), sử dụng các phương thức async của Entity Framework như ToListAsync(), FirstOrDefaultAsync(), SaveChangesAsync(). Điều này đảm bảo rằng các thao tác I/O với cơ sở dữ liệu không chặn UI thread.

Repository không chứa bất kỳ logic nghiệp vụ nào. Nó chỉ đơn giản là thực hiện các thao tác truy cập dữ liệu thuần túy. Ví dụ, StudentRepository không kiểm tra xem tên sinh viên có hợp lệ hay không, việc đó là trách nhiệm của BLL.

5.4. Tầng Business Logic Layer (StudentManagement.BLL)

Tầng BLL chứa toàn bộ logic nghiệp vụ của ứng dụng. Đây là tầng trung gian giữa GUI và DAL, nhận yêu cầu từ GUI, xử lý logic, gọi DAL để truy cập dữ liệu, và trả kết quả về GUI.

Các Service classes trong BLL tương ứng với các nghiệp vụ cụ thể. Ví dụ, StudentService chứa các phương thức nghiệp vụ liên quan đến sinh viên như AddStudentAsync(), UpdateStudentAsync(), CalculateAverageScoreAsync(). Mỗi Service phụ thuộc vào một hoặc nhiều Repository thông qua interface, được inject vào constructor.

Trước khi gọi Repository để lưu dữ liệu, Service sẽ thực hiện validation để đảm bảo dữ liệu hợp lệ. Ví dụ, khi thêm sinh viên mới, StudentService sẽ kiểm tra xem tên có rỗng không, tuổi có hợp lệ không (phải từ 16 tuổi trở lên), lớp học có tồn tại không. Nếu có lỗi, Service sẽ throw exception với thông điệp lỗi cụ thể, GUI sẽ bắt exception này và hiển thị thông báo cho người dùng.

Tầng BLL cũng chứa các logic tính toán phức tạp. Ví dụ, phương thức CalculateAverageScoreAsync() trong StudentService sẽ lấy tất cả điểm của sinh viên từ ScoreRepository, tính điểm trung bình và trả về. ReportService chứa các phương thức tạo báo cáo thống kê như GetTopStudentsAsync(), GetClassStatisticsAsync().

Một nguyên tắc quan trọng là BLL không phụ thuộc vào GUI. Service không biết gì về Form, Button hay TextBox. Nó chỉ nhận dữ liệu đầu vào dưới dạng các đối tượng Entity hoặc primitive types, xử lý và trả về kết quả. Điều này cho phép BLL có thể được tái sử dụng trong các loại giao diện khác nhau (WinForms, WPF, Web API) mà không cần thay đổi code.

5.5. Tầng Presentation Layer (StudentManagement.GUI)

Tầng GUI là tầng giao diện người dùng, chịu trách nhiệm hiển thị dữ liệu và nhận tương tác từ người dùng. Trong project này, Windows Forms được sử dụng để xây dựng giao diện.

Các Form trong GUI chỉ nên chứa code liên quan đến giao diện như khởi tạo controls, xử lý sự kiện click, hiển thị dữ liệu lên DataGridView. Mọi logic nghiệp vụ và truy cập dữ liệu đều được ủy thác cho BLL thông qua các Service.

Mỗi Form nhận các Service cần thiết thông qua constructor injection. Ví dụ, StudentForm nhận IStudentService và IClassService. Khi người dùng click nút "Thêm", Form sẽ thu thập dữ liệu từ các TextBox, tạo đối tượng Student, và gọi phương thức AddStudentAsync() của StudentService. Nếu có exception, Form sẽ bắt và hiển thị thông báo lỗi bằng MessageBox.

Nguyên tắc quan trọng nhất là GUI không được truy cập trực tiếp DbContext hay Repository. Mọi tương tác với dữ liệu phải thông qua BLL. Điều này đảm bảo sự tách biệt rõ ràng giữa các tầng và dễ dàng bảo trì.

5.6. Luồng xử lý trong hệ thống

Khi người dùng thực hiện một thao tác, luồng xử lý diễn ra như sau: Người dùng tương tác với GUI (ví dụ: nhập thông tin sinh viên và click nút Thêm). GUI thu thập dữ liệu, tạo đối tượng Entity và gọi phương thức tương ứng trong BLL (ví dụ: StudentService.AddStudentAsync()). BLL thực hiện validation dữ liệu, nếu hợp lệ, gọi phương thức tương ứng trong DAL (ví dụ: StudentRepository.AddAsync()). DAL sử dụng Entity Framework để thực hiện thao tác với cơ sở dữ liệu. Kết quả được trả về ngược lại từ DAL lên BLL, rồi từ BLL lên GUI. GUI hiển thị kết quả hoặc thông báo lỗi cho người dùng.

---

6. ENTITY FRAMEWORK VÀ LẬP TRÌNH BẤT ĐỒNG BỘ

6.1. DbContext và DbSet

Trong Entity Framework, DbContext đóng vai trò như một cầu nối giữa code C# và cơ sở dữ liệu. StudentDbContext kế thừa từ DbContext và được cấu hình để kết nối đến SQL Server thông qua connection string được lưu trong file App.config. Trong constructor của DbContext, tên connection string được truyền vào, Entity Framework sẽ tự động đọc và sử dụng connection string này để kết nối database.

Mỗi DbSet trong DbContext đại diện cho một bảng trong database. DbSet cung cấp các phương thức để query dữ liệu (sử dụng LINQ), thêm mới (Add), cập nhật, và xóa dữ liệu. Tuy nhiên, các thay đổi chỉ được lưu xuống database khi phương thức SaveChanges() hoặc

SaveChangesAsync() được gọi. Điều này cho phép thực hiện nhiều thao tác và chỉ commit một lần, tăng hiệu năng.

6.2. CRUD operations với Entity Framework

Create (Thêm mới): Để thêm một Entity mới, Entity đó được tạo ra, sau đó được add vào DbSet tương ứng bằng phương thức Add(), và cuối cùng gọi SaveChangesAsync() để lưu xuống database. Entity Framework sẽ tự động generate câu lệnh INSERT tương ứng.

Read (Đọc dữ liệu): Dữ liệu được query từ DbSet sử dụng LINQ. Entity Framework hỗ trợ nhiều phương thức query như ToListAsync() (lấy tất cả), FirstOrDefaultAsync() (lấy phần tử đầu tiên hoặc null), FindAsync() (tìm theo khóa chính), Where() (lọc theo điều kiện). Khi query, Entity Framework sẽ tự động generate câu lệnh SELECT tương ứng.

Update (Cập nhật): Để cập nhật, Entity cần được load từ database trước, sau đó các thuộc tính của nó được thay đổi, và cuối cùng gọi SaveChangesAsync(). Entity Framework sẽ theo dõi các thay đổi (change tracking) và chỉ update các cột thực sự thay đổi.

Delete (Xóa): Để xóa, Entity cần được load từ database, sau đó gọi phương thức Remove() trên DbSet, và cuối cùng gọi SaveChangesAsync(). Entity Framework sẽ generate câu lệnh DELETE.

6.3. Eager Loading và Include

Mặc định, Entity Framework sử dụng lazy loading, nghĩa là các navigation properties không được load ngay lập tức mà chỉ được load khi được truy cập. Tuy nhiên, trong ứng dụng này, lazy loading được tắt để tránh N+1 query problem và các vấn đề về serialization.

Thay vào đó, eager loading được sử dụng thông qua phương thức Include(). Khi query Student, nếu cần thông tin Class của Student đó, phương thức Include() được sử dụng để load luôn cả Class trong cùng một query. Điều này giúp giảm số lượng query xuống database và tăng hiệu năng.

6.4. Lập trình bất đồng bộ với async/await

Trong ứng dụng desktop, khi thực hiện các thao tác I/O như truy cập database, nếu thực hiện đồng bộ, UI sẽ bị đóng băng cho đến khi thao tác hoàn thành. Điều này tạo trải nghiệm không tốt cho người dùng. Lập trình bất đồng bộ giải quyết vấn đề này bằng cách cho phép UI thread tiếp tục xử lý các sự kiện khác trong khi chờ đợi thao tác I/O hoàn thành.

Tất cả các phương thức trong DAL và BLL đều được định nghĩa là async và trả về Task hoặc Task<T>. Trong thân hàm, các thao tác Entity Framework như ToListAsync(), SaveChangesAsync() được gọi với await. Khi gặp await, phương thức sẽ trả control về cho caller ngay lập tức mà không chặn thread. Khi thao tác async hoàn thành, phương thức sẽ tiếp tục từ điểm await.

Trong GUI, khi xử lý sự kiện như button click, event handler được đánh dấu async và gọi các phương thức async của Service với await. Điều này đảm bảo UI luôn mượt mà và phản hồi nhanh.

Việc sử dụng async/await đúng cách không chỉ cải thiện trải nghiệm người dùng mà còn tăng khả năng mở rộng của ứng dụng. Trong tương lai, nếu chuyển sang kiến trúc client-server, code async này có thể được tái sử dụng mà không cần thay đổi nhiều.

---

7. DEPENDENCY INJECTION

7.1. Khái niệm và lợi ích

Dependency Injection là một kỹ thuật thực hiện Inversion of Control (IoC), trong đó các dependency của một object được cung cấp từ bên ngoài thay vì object tự tạo ra chúng. Điều này giảm sự phụ thuộc giữa các class và làm cho code dễ test, dễ bảo trì và dễ mở rộng hơn.

Khi một class phụ thuộc vào một interface thay vì một concrete class, nó không cần biết implementation cụ thể là gì. Điều này cho phép thay đổi implementation mà không cần sửa code của class đó. Ví dụ, nếu sau này cần chuyển từ SQL Server sang MySQL, chỉ cần tạo repository implementation mới và thay đổi registration trong DI container, các Service không cần thay đổi gì.

7.2. Cấu hình Dependency Injection

Trong file Program.cs, tại hàm Main, một ServiceCollection được tạo ra. ServiceCollection là một container chứa các service registration. Các service được đăng ký với container thông qua các phương thức như AddTransient, AddScoped, AddSingleton, tùy thuộc vào lifecycle mong muốn.

AddTransient: Mỗi lần request, một instance mới được tạo ra. Phù hợp cho các service stateless, lightweight.

AddScoped: Một instance được tạo ra cho mỗi scope. Trong ứng dụng desktop, mỗi Form có thể coi là một scope.

AddSingleton: Chỉ một instance duy nhất được tạo ra và tái sử dụng cho toàn bộ ứng dụng.

Trong dự án này, AddTransient được sử dụng cho DbContext, Repository và Service. Điều này đảm bảo mỗi request có một DbContext riêng, tránh các vấn đề về concurrency.

Các interface và implementation được đăng ký theo cặp. Ví dụ: services.AddTransient<IStudentRepository, StudentRepository>(). Khi một class yêu cầu IStudentRepository, DI container sẽ tự động cung cấp một instance của StudentRepository.

7.3. Constructor Injection

Trong các class Service và Form, dependency được inject thông qua constructor. Constructor nhận các interface cần thiết làm tham số, và DI container sẽ tự động cung cấp các instance tương ứng khi tạo object.

Ví dụ, StudentService có constructor nhận IStudentRepository và IScoreRepository. Khi ServiceProvider.GetRequiredService<StudentService>() được gọi, DI container sẽ tự động tạo StudentRepository và ScoreRepository, inject vào StudentService, rồi trả về StudentService instance.

7.4. Sử dụng ServiceProvider

Sau khi tất cả service được đăng ký, ServiceCollection.BuildServiceProvider() được gọi để tạo ra ServiceProvider. ServiceProvider là object được sử dụng để resolve các service. Trong Main, MainForm được lấy từ ServiceProvider và được truyền vào Application.Run().

Khi một Form cần mở Form khác, nó sẽ lấy Form đó từ ServiceProvider thay vì tạo trực tiếp bằng new. Điều này đảm bảo tất cả dependency của Form con cũng được inject đúng cách.

---

8. QUẢN LÝ GIAO DỊCH (TRANSACTION)

8.1. Tại sao cần Transaction

Trong nhiều nghiệp vụ, một thao tác logic có thể bao gồm nhiều thao tác database. Ví dụ, khi nhập điểm cho nhiều sinh viên cùng lúc, mỗi điểm là một bản ghi trong bảng Scores. Nếu quá trình nhập gặp lỗi ở giữa chừng (ví dụ: vi phạm ràng buộc, mất kết nối), một số điểm đã được lưu trong khi các điểm còn lại chưa được lưu. Điều này dẫn đến dữ liệu không nhất quán.

Transaction giải quyết vấn đề này bằng cách đảm bảo rằng tất cả các thao tác trong một đơn vị logic phải được thực hiện thành công hoặc không có thao tác nào được thực hiện (all-or-nothing). Transaction tuân theo nguyên tắc ACID: Atomicity (tính nguyên tử), Consistency (tính nhất quán), Isolation (tính độc lập), Durability (tính bền vững).

8.2. Sử dụng Transaction trong Entity Framework

Entity Framework hỗ trợ Transaction thông qua DbContext.Database.BeginTransaction(). Phương thức này trả về một DbContextTransaction object, cho phép kiểm soát transaction.

Trong ví dụ cụ thể của ScoreRepository, phương thức AddRangeWithTransactionAsync() được triển khai để thêm nhiều điểm cùng lúc. Phương thức này bắt đầu một transaction bằng BeginTransaction(), sau đó lặp qua danh sách Score và thêm vào DbSet. Sau khi tất cả điểm đã được thêm vào DbSet, SaveChangesAsync() được gọi để lưu xuống database. Nếu không có lỗi,

transaction.Commit() được gọi để commit các thay đổi. Nếu có exception xảy ra, transaction.Rollback() được gọi trong khối catch để hoàn tác tất cả các thay đổi, đảm bảo database quay về trạng thái ban đầu.

8.3. Ví dụ thực tế

Trong ScoreService, phương thức AddMultipleScoresAsync() sử dụng Repository để thêm nhiều điểm. Trước khi gọi Repository, Service thực hiện validation cho tất cả điểm để đảm bảo chúng hợp lệ. Sau đó, ScoreRepository.AddRangeWithTransactionAsync() được gọi, tất cả điểm sẽ được thêm trong một transaction duy nhất. Nếu bất kỳ điểm nào không hợp lệ hoặc có lỗi xảy ra, toàn bộ giao dịch sẽ bị rollback, không có điểm nào được lưu.

Việc sử dụng Transaction trong trường hợp này không chỉ đảm bảo tính toàn vẹn dữ liệu mà còn cải thiện hiệu năng, vì nhiều thao tác được thực hiện trong một lần kết nối và commit duy nhất.

---

9. CHỨC NĂNG HỆ THỐNG

9.1. Quản lý sinh viên

Module quản lý sinh viên cung cấp đầy đủ các chức năng CRUD (Create, Read, Update, Delete) cùng với chức năng tìm kiếm. Người dùng có thể thêm sinh viên mới bằng cách nhập thông tin cơ bản như họ tên, ngày sinh, giới tính và chọn lớp từ ComboBox. Khi người dùng click nút Thêm, StudentForm sẽ thu thập dữ liệu, tạo đối tượng Student và gọi StudentService.AddStudentAsync(). Service sẽ kiểm tra dữ liệu (tên không rỗng, tuổi hợp lệ, lớp tồn tại), nếu hợp lệ sẽ gọi Repository để lưu vào database.

Chức năng sửa cho phép cập nhật thông tin sinh viên đã có. Khi người dùng chọn một sinh viên từ DataGridView, thông tin của sinh viên đó sẽ được hiển thị lên các TextBox. Người dùng có thể thay đổi thông tin và click nút Sửa để lưu. StudentService.UpdateStudentAsync() sẽ được gọi để thực hiện validation và cập nhật.

Chức năng xóa cho phép xóa sinh viên khỏi hệ thống. Khi xóa một sinh viên, tất cả điểm số của sinh viên đó cũng sẽ bị xóa theo (cascade delete). Trước khi xóa, một MessageBox xác nhận sẽ hiển thị để tránh xóa nhầm.

Chức năng tìm kiếm cho phép tìm sinh viên theo tên hoặc theo lớp. Người dùng nhập từ khóa vào TextBox tìm kiếm và click nút Tìm, kết quả sẽ được hiển thị trên DataGridView.

9.2. Quản lý lớp học

Module quản lý lớp học cho phép thêm, sửa, xóa các lớp học. Mỗi lớp có thông tin cơ bản như tên lớp và mô tả. Khi thêm lớp mới, ClassService sẽ kiểm tra xem tên lớp đã tồn tại chưa để tránh trùng lặp.

Khi xóa một lớp, hệ thống sẽ kiểm tra xem lớp có sinh viên hay không. Nếu lớp còn sinh viên, hệ thống sẽ không cho phép xóa và hiển thị thông báo yêu cầu chuyển sinh viên sang lớp khác trước. Điều này đảm bảo tính toàn vẹn dữ liệu.

9.3. Quản lý môn học

Module quản lý môn học tương tự như quản lý lớp học, cung cấp các chức năng thêm, sửa, xóa môn học. Mỗi môn học có tên và số tín chỉ. SubjectService sẽ đảm bảo tên môn không trùng lặp và số tín chỉ phải là số dương.

9.4. Quản lý điểm

Module quản lý điểm cho phép nhập và cập nhật điểm cho sinh viên. Người dùng chọn sinh viên từ ComboBox, chọn môn học từ ComboBox khác, nhập điểm và click nút Thêm. ScoreService sẽ kiểm tra xem sinh viên đã có điểm môn này chưa để tránh nhập trùng. Điểm phải nằm trong khoảng từ 0 đến 10.

Module này cũng cung cấp chức năng tính điểm trung bình. Khi người dùng chọn một sinh viên và click nút Tính điểm TB, ScoreService sẽ lấy tất cả điểm của sinh viên đó, tính trung bình và hiển thị kết quả.

9.5. Báo cáo và thống kê

Module báo cáo cung cấp nhiều loại báo cáo khác nhau thông qua ReportService.

Thống kê theo lớp: Hiển thị danh sách các lớp với số lượng sinh viên và điểm trung bình của lớp. Đây là báo cáo tổng quan giúp đánh giá chất lượng học tập của từng lớp.

Sinh viên theo lớp: Khi người dùng chọn một lớp, hệ thống sẽ hiển thị danh sách sinh viên trong lớp đó kèm theo điểm trung bình và xếp loại học lực. Xếp loại được tính dựa trên thang điểm: Xuất sắc (>= 8.5), Giỏi (>= 7.0), Khá (>= 5.5), Trung bình (>= 4.0), Yếu (< 4.0).

Top sinh viên giỏi: Hiển thị danh sách N sinh viên có điểm trung bình cao nhất trong toàn trường. Người dùng có thể nhập số lượng N vào TextBox. Báo cáo này giúp nhà trường dễ dàng nhận diện các sinh viên xuất sắc để khen thưởng.

Lọc theo khoảng điểm: Cho phép tìm các sinh viên có điểm trung bình trong một khoảng nhất định. Người dùng nhập điểm từ và điểm đến, hệ thống sẽ hiển thị danh sách sinh viên thỏa mãn. Báo cáo này hữu ích khi cần tìm các sinh viên cần học bổ túc hoặc các sinh viên đủ điều kiện thi học bổng.

---

10. GIAO DIỆN NGƯỜI DÙNG

10.1. MainForm (Trang chủ)

MainForm là Form chính của ứng dụng, hiển thị khi người dùng khởi động chương trình. Form này có giao diện đơn giản với năm nút lớn, mỗi nút dẫn đến một module chức năng: Quản lý Sinh viên, Quản lý Lớp học, Quản lý Môn học, Quản lý Điểm, và Báo cáo Thống kê. Khi người dùng click vào một nút, Form tương ứng sẽ được mở dưới dạng dialog.

10.2. StudentForm (Quản lý sinh viên)

StudentForm cung cấp giao diện để quản lý sinh viên. Phần trên của Form chứa các TextBox để nhập thông tin sinh viên (mã, tên, ngày sinh), ComboBox để chọn giới tính và lớp, cùng với các nút Thêm, Sửa, Xóa, Làm mới. Phần giữa có một TextBox tìm kiếm và nút Tìm. Phần dưới là một DataGridView hiển thị danh sách sinh viên với các cột: Mã SV, Họ tên, Ngày sinh, Giới tính, Lớp. Khi người dùng click vào một dòng trong DataGridView, thông tin sinh viên đó sẽ được hiển thị lên các TextBox ở trên, sẵn sàng cho việc sửa hoặc xóa.

10.3. ClassForm (Quản lý lớp học)

ClassForm có giao diện tương tự StudentForm nhưng đơn giản hơn. Có các TextBox để nhập mã lớp, tên lớp, mô tả, các nút Thêm, Sửa, Xóa, Làm mới, và một DataGridView hiển thị danh sách lớp với các cột: Mã lớp, Tên lớp, Mô tả, Số SV.

10.4. SubjectForm (Quản lý môn học)

SubjectForm cung cấp giao diện để quản lý môn học với các TextBox nhập mã môn, tên môn, số tín chỉ, các nút chức năng và DataGridView hiển thị danh sách môn học.

10.5. ScoreForm (Quản lý điểm)

ScoreForm có giao diện đặc biệt với hai ComboBox để chọn sinh viên và môn học, một TextBox nhập điểm, các nút Thêm, Sửa, Xóa, Làm mới, và một nút đặc biệt là Tính điểm TB. DataGridView hiển thị danh sách điểm với các cột: Mã điểm, Sinh viên, Môn học, Điểm.

10.6. ReportForm (Báo cáo thống kê)

ReportForm có bố cục khác biệt so với các Form khác. Phần trên có bốn nút lớn tương ứng với bốn loại báo cáo. Phần giữa có các TextBox và ComboBox để nhập tham số cho báo cáo (chọn lớp, nhập số top, nhập khoảng điểm). Phần dưới là DataGridView lớn để hiển thị kết quả báo cáo. Tùy theo loại báo cáo được chọn, DataGridView sẽ hiển thị các cột khác nhau.

Tất cả các Form đều được thiết kế theo nguyên tắc đơn giản, rõ ràng, dễ sử dụng. Các controls được sắp xếp hợp lý, có khoảng cách phù hợp. DataGridView được cấu hình để tự động điều chỉnh kích thước theo Form (Anchor property), đảm bảo giao diện luôn hiển thị tốt ở các kích thước màn hình khác nhau.

---

11. ĐÁNH GIÁ VÀ KẾT QUẢ ĐẠT ĐƯỢC

11.1. Hoàn thành mục tiêu đề tài

Đề tài đã hoàn thành đầy đủ các mục tiêu đề ra ban đầu. Ứng dụng đã được xây dựng thành công với đầy đủ các chức năng quản lý sinh viên, lớp học, môn học, điểm số và báo cáo thống kê. Tất cả các chức năng đều hoạt động ổn định, dữ liệu được xử lý chính xác, giao diện người dùng thân thiện và dễ sử dụng.

11.2. Áp dụng kiến trúc và công nghệ

Về mặt kỹ thuật, đề tài đã thành công trong việc áp dụng kiến trúc 3 lớp một cách đúng đắn và nhất quán. Sự tách biệt rõ ràng giữa các tầng Entities, DAL, BLL và GUI giúp code dễ đọc, dễ bảo trì và có khả năng mở rộng cao. Việc sử dụng Entity Framework 6 đã đơn giản hóa việc truy cập cơ sở dữ liệu, giảm thiểu lỗi và tăng tính bảo mật.

Lập trình bất đồng bộ đã được áp dụng toàn diện trong toàn bộ tầng DAL và BLL, giúp giao diện người dùng luôn mượt mà và phản hồi nhanh ngay cả khi thực hiện các thao tác database phức tạp. Dependency Injection đã được cấu hình đúng cách, giúp giảm sự phụ thuộc giữa các thành phần và làm cho code dễ test hơn.

Cơ chế Transaction đã được triển khai thành công trong các nghiệp vụ cần thiết, đảm bảo tính toàn vẹn dữ liệu trong mọi tình huống. Các ví dụ về Transaction trong ScoreRepository đã chứng minh hiệu quả của cơ chế này trong việc xử lý các thao tác phức tạp.

11.3. Chất lượng code

Code trong dự án tuân thủ các nguyên tắc SOLID và các best practices của C#. Tên biến, tên hàm được đặt rõ ràng, mô tả đúng chức năng. Code được tổ chức thành các phương thức nhỏ, mỗi phương thức thực hiện một nhiệm vụ cụ thể. Exception handling được thực hiện đầy đủ ở mọi tầng, đảm bảo ứng dụng không bị crash khi gặp lỗi.

11.4. Hiệu năng

Ứng dụng hoạt động mượt mà với lượng dữ liệu vừa và nhỏ (vài trăm đến vài nghìn sinh viên). Việc sử dụng async/await đảm bảo UI không bị đóng băng. Eager loading với Include giúp giảm số lượng query xuống database. Tuy nhiên, với lượng dữ liệu rất lớn (hàng chục nghìn bản ghi), có thể cần áp dụng thêm các kỹ thuật như paging, caching để tối ưu hiệu năng.

11.5. Hạn chế

Mặc dù đã đạt được nhiều kết quả tích cực, đề tài vẫn còn tồn tại một số hạn chế nhất định. Thứ nhất, ứng dụng hiện chưa triển khai chức năng đăng nhập và phân quyền người dùng, do đó mọi người sử dụng đều có quyền truy cập và thao tác trên toàn bộ dữ liệu hệ thống. Điều này có thể gây ra rủi ro về bảo mật trong môi trường sử dụng thực tế.

Thứ hai, chức năng báo cáo và thống kê hiện mới dừng ở mức hiển thị dữ liệu trên giao diện, chưa hỗ trợ xuất báo cáo ra các định dạng phổ biến như PDF hoặc Excel. Ngoài ra, ứng dụng chưa áp dụng cơ chế phân trang (paging) cho DataGridView, nên khi dữ liệu lớn có thể ảnh hưởng đến hiệu năng và trải nghiệm người dùng.

Bên cạnh đó, dự án chưa triển khai các kỹ thuật nâng cao như logging, caching hoặc unit testing, do phạm vi đồ án và thời gian thực hiện còn hạn chế.

---

12. HƯỚNG PHÁT TRIỂN

Trong tương lai, hệ thống có thể được mở rộng và nâng cấp theo nhiều hướng để đáp ứng tốt hơn nhu cầu thực tế:

- Bổ sung chức năng **đăng nhập và phân quyền người dùng** (Admin, Giáo vụ, Giảng viên).
- Phát triển tính năng **xuất báo cáo ra PDF hoặc Excel** phục vụ công tác thống kê và lưu trữ.
- Áp dụng **phân trang dữ liệu (paging)** và **tìm kiếm nâng cao** để cải thiện hiệu năng khi dữ liệu lớn.
- Chuyển đổi giao diện từ WinForms sang **WPF** hoặc phát triển phiên bản **Web Application**.
- Áp dụng **Unit Test** cho tầng BLL để tăng độ tin cậy và khả năng kiểm thử.
- Tối ưu hiệu năng bằng cách áp dụng **caching** và **lazy loading** trong những trường hợp phù hợp.

---

13. KẾT LUẬN

Đề tài "Xây dựng ứng dụng Quản lý Sinh viên" đã hoàn thành đầy đủ các yêu cầu đặt ra ban đầu của đồ án học phần. Thông qua việc thực hiện đề tài, sinh viên đã vận dụng hiệu quả các kiến thức đã học về lập trình C#, cơ sở dữ liệu SQL Server, Entity Framework, kiến trúc phần mềm nhiều tầng, lập trình bất đồng bộ, Dependency Injection và Transaction.

Ứng dụng được xây dựng có tính thực tiễn cao, giao diện thân thiện, chức năng đầy đủ và hoạt động ổn định. Quan trọng hơn, đề tài giúp sinh viên hiểu rõ hơn về quy trình xây dựng một ứng dụng phần mềm hoàn chỉnh theo hướng chuyên nghiệp, từ khâu thiết kế kiến trúc đến triển khai và kiểm thử.

Mặc dù vẫn còn một số hạn chế nhất định, nhưng với những kết quả đạt được, đề tài có thể xem là đã hoàn thành tốt mục tiêu đề ra và là nền tảng vững chắc để tiếp tục phát triển, mở rộng trong các nghiên cứu và dự án sau này.