Using .Net core. Can you please provide implementation for a school management platform that can handle many schools. The platform should allow teachers to upload school results, students, their grade/class, course/subjects, assign students to courses/subjects. We must have attendance register, notifications to parents, social media interaction including WhatsApp. Parents should be able to print term results, school must be able to do the same.. Please provide the service layer implementations, models/entities, ef core relationships

can we accommodate the grade to be dynamic for both symbols and unit. in other countries like zimbabwe they might use units to score/grade they must be able to set what each grade/score is what unit(1,2,3,4..etc) and what symbol it is (A,B C.D,etc).... can we add timetable generation for a class or per student of subject specific which can then be corrected according to the school rules.... Add a functionality for student application and admission. Functionality for transfers when students are transferring to another school if the school they are transferring to uses the platform their records can be accessed with the platform else it can generate a transfer letter and current student result for previous term(semester) and current term(semester)

Can the grading scheme be defined by one body and then all school entities can use those scales.

If capturing the answers as handwritten and determine what they were trying to say can be done on the device level instead of going to the cloud that would be ideal and the cloud will just mark to answer against teacher saved answers but it is not possible to determine the handwritting on the device then it can compare with the stored model... If possible can you define all the steps need to store this model training data and how to use it in the app. Please be very detailed because this can be the grand feature of the app. Can we then have an implementation in detail of the system/Ai generating alphabets, sentences, numbers, etc then the student captures it using handwritten responses so to personalize the models once all has been captured if there were any not clear or model failed to recognize the capture handwritting it can the assign to respective class teacher to confirm. Once all is confirmed it can the be trained on the clouded and the stored locally and on the cloud. Which will allow for the checking for the answers to happen locally before being sent to the cloud to be compared to the teacher's answers. The implementation must be comprehensive and detailed.

You have already provide these models and Service interface definitions, Look at what is currently here if you think it needs to be improved you only give me stuff that needs to be improved and only new stuff. We going to approach this in a structured way. Once you have provide the new models and DbContext, we can look at the implementation of the schoolservice,studentservice,teacherservice,teacheservice and the gradingservices. If any of these do not need changes or new implementation do mention that so that we can move to other service implementations. These implementations must be in detailed do not use placeholder use what you see would be an ideal implementation for those placeholders. I have added some implementation of the services you had previously given me if you see that there is a changed needed then change it but where no changes are needed do not respond. Always make sure you define where the implementation is going(class name). New implementation must also be in detail

namespace SmartXul.Api.Models

{

namespace SchoolManagement.Core.Entities

{

public class GradingScheme : BaseEntity

{

[Required]

[StringLength(100)]

public string Name { get; set; } = string.Empty;

[StringLength(500)]

public string Description { get; set; } = string.Empty;

[StringLength(50)]

public string Country { get; set; } = string.Empty;

public bool IsActive { get; set; } = true;

// Navigation Properties

public virtual ICollection<GradingScale> GradingScales { get; set; } = new List<GradingScale>();

public virtual ICollection<School> Schools { get; set; } = new List<School>();

}

}

// Models/Entities/GradingScale.cs

namespace SchoolManagement.Core.Entities

{

public class GradingScale : BaseEntity

{

public Guid GradingSchemeId { get; set; }

public virtual GradingScheme GradingScheme { get; set; } = null!;

[StringLength(10)]

public string Symbol { get; set; } = string.Empty; // A, B, C, D, etc.

public int Unit { get; set; } // 1, 2, 3, 4, etc.

public decimal MinPercentage { get; set; }

public decimal MaxPercentage { get; set; }

[StringLength(100)]

public string Description { get; set; } = string.Empty; // Excellent, Good, Fair, etc.

public int SortOrder { get; set; }

}

}

}

namespace SmartXul.Api.Models.Exams

{

public class Exam : BaseEntity

{

public Guid SchoolId { get; set; }

public virtual School School { get; set; } = null!;

public Guid SubjectId { get; set; }

public virtual Subject Subject { get; set; } = null!;

public Guid GradeId { get; set; }

public virtual Grade Grade { get; set; } = null!;

public Guid TermId { get; set; }

public virtual Term Term { get; set; } = null!;

public Guid SchoolYearId { get; set; }

public virtual SchoolYear SchoolYear { get; set; } = null!;

public Guid CreatedByTeacherId { get; set; }

public virtual Teacher CreatedByTeacher { get; set; } = null!;

[Required]

[StringLength(200)]

public string Title { get; set; } = string.Empty;

[StringLength(1000)]

public string Description { get; set; } = string.Empty;

public ExamType Type { get; set; }

public DateTime ExamDate { get; set; }

public TimeSpan Duration { get; set; }

public TimeSpan StartTime { get; set; }

public TimeSpan EndTime { get; set; }

public decimal TotalMarks { get; set; }

public decimal PassingMarks { get; set; }

[StringLength(100)]

public string Venue { get; set; } = string.Empty;

public ExamStatus Status { get; set; } = ExamStatus.Draft;

public bool IsPublished { get; set; } = false;

public DateTime? PublishedAt { get; set; }

public bool AllowHandwritingRecognition { get; set; } = false;

public bool IsOnline { get; set; } = false;

public bool RequiresSupervision { get; set; } = true;

[StringLength(500)]

public string Instructions { get; set; } = string.Empty;

public string MaterialsRequired { get; set; } = string.Empty; // JSON list

// Navigation Properties

public virtual ICollection<ExamQuestion> Questions { get; set; } = new List<ExamQuestion>();

public virtual ICollection<ExamRegistration> Registrations { get; set; } = new List<ExamRegistration>();

public virtual ICollection<ExamSession> Sessions { get; set; } = new List<ExamSession>();

public virtual ICollection<ExamResult> Results { get; set; } = new List<ExamResult>();

public virtual ICollection<ExamSupervisor> Supervisors { get; set; } = new List<ExamSupervisor>();

}

}

namespace SmartXul.Api.Models.Exams

{

public class ExamAnswer : BaseEntity

{

public Guid ExamSessionId { get; set; }

public virtual ExamSession ExamSession { get; set; } = null!;

public Guid ExamQuestionId { get; set; }

public virtual ExamQuestion ExamQuestion { get; set; } = null!;

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public string AnswerText { get; set; } = string.Empty;

// For file uploads or handwritten answers

public string AttachmentPath { get; set; } = string.Empty;

public string HandwritingImagePath { get; set; } = string.Empty;

// AI Recognition results

public string RecognizedText { get; set; } = string.Empty;

public decimal AiConfidenceScore { get; set; }

public bool IsAiProcessed { get; set; } = false;

// Marking

public decimal? MarksObtained { get; set; }

public bool IsMarked { get; set; } = false;

public Guid? MarkedByTeacherId { get; set; }

public virtual Teacher? MarkedByTeacher { get; set; }

public DateTime? MarkedAt { get; set; }

public string TeacherFeedback { get; set; } = string.Empty;

// Answer metadata

public DateTime AnsweredAt { get; set; } = DateTime.UtcNow;

public TimeSpan TimeSpent { get; set; }

public int AttemptNumber { get; set; } = 1;

public bool IsFlagged { get; set; } = false;

public string FlagReason { get; set; } = string.Empty;

}

}

using SmartXul.Shared.Enums;

using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models.Exams

{

public class ExamIncident : BaseEntity

{

public Guid ExamId { get; set; }

public virtual Exam Exam { get; set; } = null!;

public Guid? StudentId { get; set; }

public virtual Student? Student { get; set; }

public Guid ReportedByTeacherId { get; set; }

public virtual Teacher ReportedByTeacher { get; set; } = null!;

public IncidentType Type { get; set; }

public IncidentSeverity Severity { get; set; }

[Required]

public string Description { get; set; } = string.Empty;

public DateTime IncidentTime { get; set; }

public string Location { get; set; } = string.Empty;

public string ActionTaken { get; set; } = string.Empty;

public bool RequiresFollowUp { get; set; } = false;

public string FollowUpNotes { get; set; } = string.Empty;

public string AttachmentsPath { get; set; } = string.Empty; // Evidence photos/videos

public IncidentStatus Status { get; set; } = IncidentStatus.Open;

public DateTime? ResolvedAt { get; set; }

public Guid? ResolvedByTeacherId { get; set; }

public virtual Teacher? ResolvedByTeacher { get; set; }

}

}

using SmartXul.Shared.Enums;

using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models.Exams

{

public class ExamQuestion : BaseEntity

{

public Guid ExamId { get; set; }

public virtual Exam Exam { get; set; } = null!;

public int QuestionNumber { get; set; }

[Required]

public string QuestionText { get; set; } = string.Empty;

public QuestionType Type { get; set; }

public decimal Marks { get; set; }

public bool IsRequired { get; set; } = true;

public TimeSpan? TimeLimit { get; set; }

// For multiple choice questions

public string Options { get; set; } = string.Empty; // JSON array

public string CorrectAnswer { get; set; } = string.Empty;

// For essay/written questions

public int? WordLimit { get; set; }

public string SampleAnswer { get; set; } = string.Empty;

public string MarkingRubric { get; set; } = string.Empty; // JSON

// For handwriting recognition

public bool EnableHandwritingRecognition { get; set; } = false;

public string ExpectedKeywords { get; set; } = string.Empty; // JSON array

// Media attachments

public string AttachmentPath { get; set; } = string.Empty;

public int SortOrder { get; set; }

// Navigation Properties

public virtual ICollection<ExamAnswer> Answers { get; set; } = new List<ExamAnswer>();

}

}

using SmartXul.Shared.Enums;

using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models.Exams

{

public class ExamRegistration : BaseEntity

{

public Guid ExamId { get; set; }

public virtual Exam Exam { get; set; } = null!;

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public DateTime RegistrationDate { get; set; } = DateTime.UtcNow;

public RegistrationStatus Status { get; set; } = RegistrationStatus.Registered;

public bool IsPresent { get; set; } = false;

public DateTime? CheckInTime { get; set; }

public DateTime? CheckOutTime { get; set; }

[StringLength(100)]

public string SeatNumber { get; set; } = string.Empty;

public bool RequiresSpecialAccommodation { get; set; } = false;

public string SpecialAccommodations { get; set; } = string.Empty;

[StringLength(500)]

public string Remarks { get; set; } = string.Empty;

}

}

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models.Exams

{

public class ExamResult : BaseEntity

{

public Guid ExamId { get; set; }

public virtual Exam Exam { get; set; } = null!;

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public Guid ExamSessionId { get; set; }

public virtual ExamSession ExamSession { get; set; } = null!;

public decimal TotalMarks { get; set; }

public decimal MarksObtained { get; set; }

public decimal Percentage { get; set; }

public string Grade { get; set; } = string.Empty;

public int GradeUnit { get; set; }

public ResultStatus Status { get; set; } = ResultStatus.Pending;

public bool IsPassed { get; set; }

public string Remarks { get; set; } = string.Empty;

// Marking details

public bool IsFullyMarked { get; set; } = false;

public int QuestionsMarked { get; set; }

public int TotalQuestions { get; set; }

public Guid? MarkedByTeacherId { get; set; }

public virtual Teacher? MarkedByTeacher { get; set; }

public DateTime? MarkedAt { get; set; }

// Result publication

public bool IsPublished { get; set; } = false;

public DateTime? PublishedAt { get; set; }

// Analytics

public int Rank { get; set; }

public decimal? ClassAverage { get; set; }

public decimal? HighestScore { get; set; }

public decimal? LowestScore { get; set; }

public string DetailedAnalysis { get; set; } = string.Empty; // JSON

}

}

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models.Exams

{

public class ExamSession : BaseEntity

{

public Guid ExamId { get; set; }

public virtual Exam Exam { get; set; } = null!;

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public DateTime StartTime { get; set; }

public DateTime? EndTime { get; set; }

public TimeSpan ActualDuration { get; set; }

public SessionStatus Status { get; set; } = SessionStatus.NotStarted;

public bool IsSubmitted { get; set; } = false;

public DateTime? SubmittedAt { get; set; }

public bool IsAutoSubmitted { get; set; } = false;

public int TotalQuestions { get; set; }

public int AnsweredQuestions { get; set; }

// Proctoring data

public string DeviceInfo { get; set; } = string.Empty; // JSON

public string IPAddress { get; set; } = string.Empty;

public string BrowserInfo { get; set; } = string.Empty;

// Security flags

public bool HasViolations { get; set; } = false;

public string ViolationDetails { get; set; } = string.Empty; // JSON

public decimal? Score { get; set; }

public decimal? Percentage { get; set; }

// Navigation Properties

public virtual ICollection<ExamAnswer> Answers { get; set; } = new List<ExamAnswer>();

public virtual ICollection<ExamSessionLog> SessionLogs { get; set; } = new List<ExamSessionLog>();

}

}

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models.Exams

{

public class ExamSessionLog : BaseEntity

{

public Guid ExamSessionId { get; set; }

public virtual ExamSession ExamSession { get; set; } = null!;

public LogType Type { get; set; }

public string Event { get; set; } = string.Empty;

public string Details { get; set; } = string.Empty; // JSON

public DateTime Timestamp { get; set; } = DateTime.UtcNow;

public string IPAddress { get; set; } = string.Empty;

public string UserAgent { get; set; } = string.Empty;

public bool IsSuspicious { get; set; } = false;

public string SuspicionReason { get; set; } = string.Empty;

}

}

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models.Exams

{

public class ExamSupervisor : BaseEntity

{

public Guid ExamId { get; set; }

public virtual Exam Exam { get; set; } = null!;

public Guid TeacherId { get; set; }

public virtual Teacher Teacher { get; set; } = null!;

public SupervisorRole Role { get; set; }

public DateTime AssignedAt { get; set; } = DateTime.UtcNow;

public bool IsPresent { get; set; } = false;

public DateTime? CheckInTime { get; set; }

public DateTime? CheckOutTime { get; set; }

public string Responsibilities { get; set; } = string.Empty;

public string Notes { get; set; } = string.Empty;

// Navigation Properties

public virtual ICollection<ExamIncident> ReportedIncidents { get; set; } = new List<ExamIncident>();

}

}

using SmartXul.Shared;

using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class Attendance : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public Guid SubjectId { get; set; }

public virtual Subject Subject { get; set; } = null!;

public Guid TeacherId { get; set; }

public virtual Teacher Teacher { get; set; } = null!;

public Guid TermId { get; set; }

public virtual Term Term { get; set; } = null!;

public DateTime Date { get; set; }

public AttendanceStatus Status { get; set; }

[StringLength(200)]

public string Remarks { get; set; } = string.Empty;

}

}

using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public abstract class BaseEntity

{

[Key]

public Guid Id { get; set; } = Guid.NewGuid();

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

public string CreatedBy { get; set; } = string.Empty;

public string UpdatedBy { get; set; } = string.Empty;

public bool IsDeleted { get; set; } = false;

}

}

namespace SmartXul.Api.Models

{

public class ClassTeacher : BaseEntity

{

public Guid TeacherId { get; set; }

public virtual Teacher Teacher { get; set; } = null!;

public Guid GradeId { get; set; }

public virtual Grade Grade { get; set; } = null!;

public Guid SchoolYearId { get; set; }

public virtual SchoolYear SchoolYear { get; set; } = null!;

public bool IsActive { get; set; } = true;

}

}

using System.ComponentModel.DataAnnotations;

using System.Xml;

namespace SmartXul.Api.Models

{

public class Grade : BaseEntity

{

public Guid SchoolId { get; set; }

public virtual School School { get; set; } = null!;

[Required]

[StringLength(50)]

public string Name { get; set; } = string.Empty; // Grade 1, Form 1, Year 7, etc.

[StringLength(200)]

public string Description { get; set; } = string.Empty;

public int Level { get; set; } // 1, 2, 3, etc. for ordering

public int MaxStudents { get; set; } = 50;

// Navigation Properties

public virtual ICollection<Student> Students { get; set; } = new List<Student>();

public virtual ICollection<GradeSubject> GradeSubjects { get; set; } = new List<GradeSubject>();

public virtual ICollection<ClassTeacher> ClassTeachers { get; set; } = new List<ClassTeacher>();

public virtual ICollection<Timetable> Timetables { get; set; } = new List<Timetable>();

}

}namespace SmartXul.Api.Models

{

public class GradeSubject : BaseEntity

{

public Guid GradeId { get; set; }

public virtual Grade Grade { get; set; } = null!;

public Guid SubjectId { get; set; }

public virtual Subject Subject { get; set; } = null!;

public bool IsCompulsory { get; set; } = true;

public int SortOrder { get; set; }

}

}using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class HandwritingCharacterData : BaseEntity

{

public Guid HandwritingTrainingDataId { get; set; }

public virtual HandwritingTrainingData HandwritingTrainingData { get; set; } = null!;

[Required]

public string Character { get; set; } = string.Empty; // The actual character (A, B, 1, 2, etc.)

[Required]

public string ImageSegmentPath { get; set; } = string.Empty; // Path to individual character image

// Bounding box coordinates

public int X { get; set; }

public int Y { get; set; }

public int Width { get; set; }

public int Height { get; set; }

public decimal ConfidenceScore { get; set; }

public bool IsVerified { get; set; } = false;

// Feature vectors for ML model

public string FeatureVector { get; set; } = string.Empty; // JSON array of features

public int SequenceOrder { get; set; } // Order in the original text

}

}using System.ComponentModel.DataAnnotations;

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models

{

public class HandwritingModel : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

[Required]

[StringLength(100)]

public string ModelName { get; set; } = string.Empty;

[Required]

[StringLength(50)]

public string Version { get; set; } = string.Empty;

public string ModelFilePath { get; set; } = string.Empty; // Local device model path

public string CloudModelPath { get; set; } = string.Empty; // Cloud backup path

public ModelType Type { get; set; }

public decimal Accuracy { get; set; }

public int TrainingDataCount { get; set; }

public bool IsActive { get; set; } = true;

public bool IsDeployedLocally { get; set; } = false;

public bool IsDeployedCloud { get; set; } = false;

public DateTime TrainingStarted { get; set; }

public DateTime? TrainingCompleted { get; set; }

public DateTime? LastUsed { get; set; }

// Model metadata

public string TrainingParameters { get; set; } = string.Empty; // JSON

public string PerformanceMetrics { get; set; } = string.Empty; // JSON

// Navigation Properties

public virtual ICollection<HandwritingRecognitionResult> RecognitionResults { get; set; } = new List<HandwritingRecognitionResult>();

}

}using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class HandwritingRecognitionResult : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public Guid? SubjectId { get; set; }

public virtual Subject? Subject { get; set; }

public Guid? ResultId { get; set; }

public virtual Result? Result { get; set; }

public Guid HandwritingModelId { get; set; }

public virtual HandwritingModel HandwritingModel { get; set; } = null!;

[Required]

public string OriginalImagePath { get; set; } = string.Empty;

[Required]

public string RecognizedText { get; set; } = string.Empty;

public decimal OverallConfidence { get; set; }

public bool ProcessedLocally { get; set; } = true;

public bool ProcessedInCloud { get; set; } = false;

public TimeSpan ProcessingTime { get; set; }

public string CharacterConfidences { get; set; } = string.Empty; // JSON array

public string AlternativeTexts { get; set; } = string.Empty; // JSON array of alternatives

// For answer verification

public string ExpectedAnswer { get; set; } = string.Empty;

public bool IsCorrect { get; set; }

public decimal PartialCreditScore { get; set; }

public DateTime ProcessedAt { get; set; } = DateTime.UtcNow;

}

}using System.ComponentModel.DataAnnotations;

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models

{

public class HandwritingTrainingData : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public Guid? TeacherId { get; set; }

public virtual Teacher? Teacher { get; set; }

public TrainingDataType DataType { get; set; }

[Required]

public string OriginalContent { get; set; } = string.Empty; // What the student was supposed to write

[Required]

public string ImagePath { get; set; } = string.Empty; // Path to handwritten image

public string ProcessedText { get; set; } = string.Empty; // AI interpreted text

public string VerifiedText { get; set; } = string.Empty; // Teacher verified text

public decimal ConfidenceScore { get; set; }

public TrainingStatus Status { get; set; } = TrainingStatus.Pending;

public bool IsVerified { get; set; } = false;

public bool IsTrainingComplete { get; set; } = false;

public DateTime CapturedAt { get; set; } = DateTime.UtcNow;

public DateTime? VerifiedAt { get; set; }

public DateTime? TrainedAt { get; set; }

// Metadata for training

public string BoundingBoxes { get; set; } = string.Empty; // JSON array of character bounding boxes

//public string CharacterData { get; set; } = string.Empty; // JSON array of individual character data

public string ModelVersion { get; set; } = string.Empty;

// Navigation Properties

public virtual ICollection<HandwritingCharacterData> CharacterData { get; set; } = new List<HandwritingCharacterData>();

}

}

using System.ComponentModel.DataAnnotations;

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models

{

public class ModelTrainingJob : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

[Required]

[StringLength(100)]

public string JobName { get; set; } = string.Empty;

public JobType Type { get; set; }

public JobStatus Status { get; set; } = JobStatus.Queued;

public int TotalDataPoints { get; set; }

public int ProcessedDataPoints { get; set; }

public decimal Progress { get; set; }

public DateTime QueuedAt { get; set; } = DateTime.UtcNow;

public DateTime? StartedAt { get; set; }

public DateTime? CompletedAt { get; set; }

public string ErrorMessage { get; set; } = string.Empty;

public string ResultPath { get; set; } = string.Empty;

// Training parameters

public string TrainingParameters { get; set; } = string.Empty; // JSON

// Navigation Properties

public virtual HandwritingModel? ResultingModel { get; set; }

}

} using System.ComponentModel.DataAnnotations;

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models

{

public class Notification : BaseEntity

{

public Guid? ParentId { get; set; }

public virtual Parent? Parent { get; set; }

public Guid? StudentId { get; set; }

public virtual Student? Student { get; set; }

public Guid? TeacherId { get; set; }

public virtual Teacher? Teacher { get; set; }

[Required]

[StringLength(200)]

public string Title { get; set; } = string.Empty;

[Required]

public string Message { get; set; } = string.Empty;

public NotificationType Type { get; set; }

public NotificationChannel Channel { get; set; }

public bool IsRead { get; set; } = false;

public bool IsSent { get; set; } = false;

public DateTime? SentAt { get; set; }

public DateTime? ReadAt { get; set; }

[StringLength(100)]

public string ExternalId { get; set; } = string.Empty; // For WhatsApp, SMS tracking

public string Metadata { get; set; } = string.Empty; // JSON for additional data

}

} using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class Parent : BaseEntity

{

[Required]

[StringLength(100)]

public string FirstName { get; set; } = string.Empty;

[Required]

[StringLength(100)]

public string LastName { get; set; } = string.Empty;

[EmailAddress]

[StringLength(100)]

public string Email { get; set; } = string.Empty;

[StringLength(20)]

public string PhoneNumber { get; set; } = string.Empty;

[StringLength(20)]

public string WhatsAppNumber { get; set; } = string.Empty;

[StringLength(200)]

public string Address { get; set; } = string.Empty;

[StringLength(50)]

public string Relationship { get; set; } = string.Empty; // Father, Mother, Guardian

[StringLength(100)]

public string Occupation { get; set; } = string.Empty;

// Navigation Properties

public virtual ICollection<Student> Students { get; set; } = new List<Student>();

public virtual ICollection<Notification> Notifications { get; set; } = new List<Notification>();

}

} using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class Result : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public Guid SubjectId { get; set; }

public virtual Subject Subject { get; set; } = null!;

public Guid TeacherId { get; set; }

public virtual Teacher Teacher { get; set; } = null!;

public Guid TermId { get; set; }

public virtual Term Term { get; set; } = null!;

public Guid SchoolYearId { get; set; }

public virtual SchoolYear SchoolYear { get; set; } = null!;

public decimal Score { get; set; }

public decimal TotalMarks { get; set; }

public decimal Percentage { get; set; }

[StringLength(10)]

public string Grade { get; set; } = string.Empty; // A, B, C, etc.

public int GradeUnit { get; set; } // 1, 2, 3, etc.

[StringLength(50)]

public string AssessmentType { get; set; } = string.Empty; // Test, Exam, Assignment

[StringLength(500)]

public string Comments { get; set; } = string.Empty;

public DateTime AssessmentDate { get; set; }

// AI Answer Verification

public bool IsAiVerified { get; set; } = false;

public decimal AiConfidenceScore { get; set; }

public string AiProcessedAnswers { get; set; } = string.Empty; // JSON

public object PartialCreditScore { get; internal set; }

}

} using System.ComponentModel.DataAnnotations;

using System.Diagnostics;

using System.Xml;

using SmartXul.Api.Models.SchoolManagement.Core.Entities;

namespace SmartXul.Api.Models

{

public class School : BaseEntity

{

[Required]

[StringLength(200)]

public string Name { get; set; } = string.Empty;

[StringLength(500)]

public string Address { get; set; } = string.Empty;

[StringLength(20)]

public string PhoneNumber { get; set; } = string.Empty;

[EmailAddress]

[StringLength(100)]

public string Email { get; set; } = string.Empty;

[StringLength(100)]

public string Website { get; set; } = string.Empty;

[StringLength(50)]

public string RegistrationNumber { get; set; } = string.Empty;

public string Logo { get; set; } = string.Empty;

// Navigation Properties

public virtual ICollection<Teacher> Teachers { get; set; } = new List<Teacher>();

public virtual ICollection<Student> Students { get; set; } = new List<Student>();

public virtual ICollection<Grade> Grades { get; set; } = new List<Grade>();

public virtual ICollection<Subject> Subjects { get; set; } = new List<Subject>();

public virtual ICollection<SchoolYear> SchoolYears { get; set; } = new List<SchoolYear>();

public virtual ICollection<Timetable> Timetables { get; set; } = new List<Timetable>();

// Grading Scheme Reference

public Guid? GradingSchemeId { get; set; }

public virtual GradingScheme? GradingScheme { get; set; }

}

} using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class SchoolYear : BaseEntity

{

public Guid SchoolId { get; set; }

public virtual School School { get; set; } = null!;

[Required] [StringLength(50)] public string Name { get; set; } = string.Empty; // 2024/2025

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public bool IsActive { get; set; } = true;

public bool IsCurrent { get; set; } = false;

// Navigation Properties

public virtual ICollection<Term> Terms { get; set; } = new List<Term>();

public virtual ICollection<SubjectTeacher> SubjectTeachers { get; set; } = new List<SubjectTeacher>();

public virtual ICollection<StudentSubject> StudentSubjects { get; set; } = new List<StudentSubject>();

public virtual ICollection<ClassTeacher> ClassTeachers { get; set; } = new List<ClassTeacher>();

public virtual ICollection<Result> Results { get; set; } = new List<Result>();

}

public class Term : BaseEntity

{

public Guid SchoolYearId { get; set; }

public virtual SchoolYear SchoolYear { get; set; } = null!;

[Required] [StringLength(50)] public string Name { get; set; } = string.Empty; // Term 1, Semester 1, etc.

public int TermNumber { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public bool IsActive { get; set; } = true;

public bool IsCurrent { get; set; } = false;

// Navigation Properties

public virtual ICollection<Result> Results { get; set; } = new List<Result>();

public virtual ICollection<Attendance> AttendanceRecords { get; set; } = new List<Attendance>();

}

} using System.ComponentModel.DataAnnotations;

using System.Diagnostics;

namespace SmartXul.Api.Models

{

public class Student : BaseEntity

{

public Guid SchoolId { get; set; }

public virtual School School { get; set; } = null!;

[Required]

[StringLength(100)]

public string FirstName { get; set; } = string.Empty;

[Required]

[StringLength(100)]

public string LastName { get; set; } = string.Empty;

[StringLength(50)]

public string StudentNumber { get; set; } = string.Empty;

public DateTime DateOfBirth { get; set; }

[StringLength(10)]

public string Gender { get; set; } = string.Empty;

[StringLength(200)]

public string Address { get; set; } = string.Empty;

public string ProfilePicture { get; set; } = string.Empty;

// Current Grade

public Guid CurrentGradeId { get; set; }

public virtual Grade CurrentGrade { get; set; } = null!;

// Navigation Properties

public virtual ICollection<Parent> Parents { get; set; } = new List<Parent>();

public virtual ICollection<StudentSubject> StudentSubjects { get; set; } = new List<StudentSubject>();

public virtual ICollection<Attendance> AttendanceRecords { get; set; } = new List<Attendance>();

public virtual ICollection<Result> Results { get; set; } = new List<Result>();

public virtual ICollection<HandwritingTrainingData> HandwritingTrainingData { get; set; } = new List<HandwritingTrainingData>();

public virtual ICollection<StudentTimetable> StudentTimetables { get; set; } = new List<StudentTimetable>();

}

} namespace SmartXul.Api.Models

{

public class StudentSubject : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public Guid SubjectId { get; set; }

public virtual Subject Subject { get; set; } = null!;

public Guid SchoolYearId { get; set; }

public virtual SchoolYear SchoolYear { get; set; } = null!;

public DateTime EnrollmentDate { get; set; } = DateTime.UtcNow;

public bool IsActive { get; set; } = true;

}

} using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class StudentTimetable : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public Guid TimetableSlotId { get; set; }

public virtual TimetableSlot TimetableSlot { get; set; } = null!;

public bool IsActive { get; set; } = true;

// For individual student customizations

[StringLength(200)]

public string CustomNotes { get; set; } = string.Empty;

}

} using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models

{

public class StudentTrainingSession : BaseEntity

{

public Guid StudentId { get; set; }

public virtual Student Student { get; set; } = null!;

public Guid TrainingExerciseId { get; set; }

public virtual TrainingExercise TrainingExercise { get; set; } = null!;

public DateTime StartedAt { get; set; }

public DateTime? CompletedAt { get; set; }

public SessionStatus Status { get; set; } = SessionStatus.InProgress;

public int TotalItems { get; set; }

public int CompletedItems { get; set; }

public int CorrectItems { get; set; }

public decimal OverallAccuracy { get; set; }

public TimeSpan TotalTime { get; set; }

// Navigation Properties

public virtual ICollection<HandwritingTrainingData> TrainingData { get; set; } = new List<HandwritingTrainingData>();

}

} using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class Subject : BaseEntity

{

public Guid SchoolId { get; set; }

public virtual School School { get; set; } = null!;

[Required]

[StringLength(100)]

public string Name { get; set; } = string.Empty;

[StringLength(10)]

public string Code { get; set; } = string.Empty;

[StringLength(500)]

public string Description { get; set; } = string.Empty;

public int Credits { get; set; } = 1;

public bool IsActive { get; set; } = true;

// Navigation Properties

public virtual ICollection<GradeSubject> GradeSubjects { get; set; } = new List<GradeSubject>();

public virtual ICollection<SubjectTeacher> SubjectTeachers { get; set; } = new List<SubjectTeacher>();

public virtual ICollection<StudentSubject> StudentSubjects { get; set; } = new List<StudentSubject>();

public virtual ICollection<Result> Results { get; set; } = new List<Result>();

public virtual ICollection<TimetableSlot> TimetableSlots { get; set; } = new List<TimetableSlot>();

}

} namespace SmartXul.Api.Models

{

public class SubjectTeacher : BaseEntity

{

public Guid SubjectId { get; set; }

public virtual Subject Subject { get; set; } = null!;

public Guid TeacherId { get; set; }

public virtual Teacher Teacher { get; set; } = null!;

public Guid GradeId { get; set; }

public virtual Grade Grade { get; set; } = null!;

public Guid SchoolYearId { get; set; }

public virtual SchoolYear SchoolYear { get; set; } = null!;

public bool IsActive { get; set; } = true;

}

} using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class Teacher : BaseEntity

{

public Guid SchoolId { get; set; }

public virtual School School { get; set; } = null!;

[Required]

[StringLength(100)]

public string FirstName { get; set; } = string.Empty;

[Required]

[StringLength(100)]

public string LastName { get; set; } = string.Empty;

[EmailAddress]

[StringLength(100)]

public string Email { get; set; } = string.Empty;

[StringLength(20)]

public string PhoneNumber { get; set; } = string.Empty;

[StringLength(200)]

public string Address { get; set; } = string.Empty;

[StringLength(50)]

public string EmployeeId { get; set; } = string.Empty;

public DateTime DateOfBirth { get; set; }

public DateTime HireDate { get; set; }

[StringLength(100)]

public string Qualification { get; set; } = string.Empty;

public string ProfilePicture { get; set; } = string.Empty;

// Navigation Properties

public virtual ICollection<SubjectTeacher> SubjectTeachers { get; set; } = new List<SubjectTeacher>();

public virtual ICollection<ClassTeacher> ClassTeachers { get; set; } = new List<ClassTeacher>();

public virtual ICollection<Attendance> AttendanceRecords { get; set; } = new List<Attendance>();

public virtual ICollection<Result> Results { get; set; } = new List<Result>();

public virtual ICollection<TimetableSlot> TimetableSlots { get; set; } = new List<TimetableSlot>();

}

} using System.ComponentModel.DataAnnotations;

namespace SmartXul.Api.Models

{

public class Timetable : BaseEntity

{

public Guid SchoolId { get; set; }

public virtual School School { get; set; } = null!;

public Guid GradeId { get; set; }

public virtual Grade Grade { get; set; } = null!;

public Guid SchoolYearId { get; set; }

public virtual SchoolYear SchoolYear { get; set; } = null!;

public Guid TermId { get; set; }

public virtual Term Term { get; set; } = null!;

[Required]

[StringLength(100)]

public string Name { get; set; } = string.Empty;

public bool IsActive { get; set; } = true;

public bool IsGenerated { get; set; } = false;

public DateTime GeneratedAt { get; set; }

// Navigation Properties

public virtual ICollection<TimetableSlot> TimetableSlots { get; set; } = new List<TimetableSlot>();

}

public class TimetableSlot : BaseEntity

{

public Guid TimetableId { get; set; }

public virtual Timetable Timetable { get; set; } = null!;

public Guid SubjectId { get; set; }

public virtual Subject Subject { get; set; } = null!;

public Guid TeacherId { get; set; }

public virtual Teacher Teacher { get; set; } = null!;

public DayOfWeek DayOfWeek { get; set; }

public TimeSpan StartTime { get; set; }

public TimeSpan EndTime { get; set; }

[StringLength(50)]

public string Room { get; set; } = string.Empty;

public int PeriodNumber { get; set; }

[StringLength(200)]

public string Notes { get; set; } = string.Empty;

}

} using System.ComponentModel.DataAnnotations;

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Models

{

public class TrainingExercise : BaseEntity

{

public Guid SchoolId { get; set; }

public virtual School School { get; set; } = null!;

public Guid? GradeId { get; set; }

public virtual Grade? Grade { get; set; }

public Guid? SubjectId { get; set; }

public virtual Subject? Subject { get; set; }

[Required]

[StringLength(200)]

public string Title { get; set; } = string.Empty;

[StringLength(1000)]

public string Description { get; set; } = string.Empty;

public ExerciseType Type { get; set; }

public DifficultyLevel Difficulty { get; set; }

[Required]

public string Content { get; set; } = string.Empty; // JSON containing exercise data

public bool IsActive { get; set; } = true;

public bool IsSystemGenerated { get; set; } = false;

public int EstimatedMinutes { get; set; } = 10;

// Navigation Properties

public virtual ICollection<StudentTrainingSession> StudentSessions { get; set; } = new List<StudentTrainingSession>();

}

} using Microsoft.EntityFrameworkCore.ChangeTracking;

using Microsoft.EntityFrameworkCore;

using SmartXul.Api.Models;

using SmartXul.Api.Models.SchoolManagement.Core.Entities;

using System.Linq.Expressions;

using SmartXul.Api.Models.Exams;

namespace SmartXul.Api

{

public class SmartXulDbContext : DbContext

{

public SmartXulDbContext(DbContextOptions<SmartXulDbContext> options)

: base(options)

{

}

// Core Entities

public DbSet<School> Schools { get; set; }

public DbSet<GradingScheme> GradingSchemes { get; set; }

public DbSet<GradingScale> GradingScales { get; set; }

public DbSet<Teacher> Teachers { get; set; }

public DbSet<Student> Students { get; set; }

public DbSet<Parent> Parents { get; set; }

public DbSet<Grade> Grades { get; set; }

public DbSet<Subject> Subjects { get; set; }

public DbSet<SchoolYear> SchoolYears { get; set; }

public DbSet<Term> Terms { get; set; }

// Relationship Entities

public DbSet<GradeSubject> GradeSubjects { get; set; }

public DbSet<SubjectTeacher> SubjectTeachers { get; set; }

public DbSet<StudentSubject> StudentSubjects { get; set; }

public DbSet<ClassTeacher> ClassTeachers { get; set; }

// Academic Records

public DbSet<Attendance> Attendances { get; set; }

public DbSet<Result> Results { get; set; }

public DbSet<Notification> Notifications { get; set; }

// Timetable

public DbSet<Timetable> Timetables { get; set; }

public DbSet<TimetableSlot> TimetableSlots { get; set; }

public DbSet<StudentTimetable> StudentTimetables { get; set; }

// AI Handwriting Recognition

public DbSet<HandwritingTrainingData> HandwritingTrainingData { get; set; }

public DbSet<HandwritingCharacterData> HandwritingCharacterData { get; set; }

public DbSet<HandwritingModel> HandwritingModels { get; set; }

public DbSet<HandwritingRecognitionResult> HandwritingRecognitionResults { get; set; }

public DbSet<TrainingExercise> TrainingExercises { get; set; }

public DbSet<StudentTrainingSession> StudentTrainingSessions { get; set; }

public DbSet<ModelTrainingJob> ModelTrainingJobs { get; set; }

// Exams

public DbSet<Exam> Exams { get; set; }

public DbSet<ExamQuestion> ExamQuestions { get; set; }

public DbSet<ExamAnswer> ExamAnswers { get; set; }

public DbSet<ExamRegistration> ExamRegistrations { get; set; }

public DbSet<ExamSession> ExamSessions { get; set; }

public DbSet<ExamResult> ExamResults { get; set; }

public DbSet<ExamSupervisor> ExamSupervisors { get; set; }

public DbSet<ExamIncident> ExamIncidents { get; set; }

public DbSet<ExamSessionLog> ExamSessionLogs { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

// Configure entity relationships and constraints

ConfigureSchoolEntities(modelBuilder);

ConfigureGradingEntities(modelBuilder);

ConfigureUserEntities(modelBuilder);

ConfigureAcademicEntities(modelBuilder);

ConfigureTimetableEntities(modelBuilder);

ConfigureAIEntities(modelBuilder);

ConfigureIndexes(modelBuilder);

ConfigureSoftDelete(modelBuilder);

}

private void ConfigureSchoolEntities(ModelBuilder modelBuilder)

{

modelBuilder.Entity<School>(entity =>

{

entity.HasIndex(e => e.RegistrationNumber).IsUnique();

entity.HasIndex(e => e.Email).IsUnique();

entity.Property(e => e.Name).IsRequired();

});

modelBuilder.Entity<SchoolYear>(entity =>

{

entity.HasOne(sy => sy.School)

.WithMany(s => s.SchoolYears)

.HasForeignKey(sy => sy.SchoolId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.SchoolId, e.Name }).IsUnique();

entity.HasIndex(e => new { e.SchoolId, e.IsCurrent });

});

modelBuilder.Entity<Term>(entity =>

{

entity.HasOne(t => t.SchoolYear)

.WithMany(sy => sy.Terms)

.HasForeignKey(t => t.SchoolYearId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.SchoolYearId, e.TermNumber }).IsUnique();

});

}

private void ConfigureGradingEntities(ModelBuilder modelBuilder)

{

modelBuilder.Entity<GradingScheme>(entity =>

{

entity.HasIndex(e => new { e.Name, e.Country }).IsUnique();

entity.Property(e => e.Name).IsRequired();

});

modelBuilder.Entity<GradingScale>(entity =>

{

entity.HasOne(gs => gs.GradingScheme)

.WithMany(g => g.GradingScales)

.HasForeignKey(gs => gs.GradingSchemeId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.GradingSchemeId, e.Symbol }).IsUnique();

entity.HasIndex(e => new { e.GradingSchemeId, e.Unit }).IsUnique();

entity.Property(e => e.MinPercentage).HasColumnType("decimal(5,2)");

entity.Property(e => e.MaxPercentage).HasColumnType("decimal(5,2)");

});

modelBuilder.Entity<School>()

.HasOne(s => s.GradingScheme)

.WithMany(gs => gs.Schools)

.HasForeignKey(s => s.GradingSchemeId)

.OnDelete(DeleteBehavior.SetNull);

}

private void ConfigureUserEntities(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Teacher>(entity =>

{

entity.HasOne(t => t.School)

.WithMany(s => s.Teachers)

.HasForeignKey(t => t.SchoolId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.SchoolId, e.Email }).IsUnique();

entity.HasIndex(e => new { e.SchoolId, e.EmployeeId }).IsUnique();

entity.Property(e => e.FirstName).IsRequired();

entity.Property(e => e.LastName).IsRequired();

});

modelBuilder.Entity<Student>(entity =>

{

entity.HasOne(s => s.School)

.WithMany(sch => sch.Students)

.HasForeignKey(s => s.SchoolId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(s => s.CurrentGrade)

.WithMany(g => g.Students)

.HasForeignKey(s => s.CurrentGradeId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(e => new { e.SchoolId, e.StudentNumber }).IsUnique();

entity.Property(e => e.FirstName).IsRequired();

entity.Property(e => e.LastName).IsRequired();

// Many-to-many relationship with Parents

entity.HasMany(s => s.Parents)

.WithMany(p => p.Students)

.UsingEntity(j => j.ToTable("StudentParents"));

});

modelBuilder.Entity<Parent>(entity =>

{

entity.HasIndex(e => e.Email).IsUnique();

entity.Property(e => e.FirstName).IsRequired();

entity.Property(e => e.LastName).IsRequired();

});

}

private void ConfigureAcademicEntities(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Grade>(entity =>

{

entity.HasOne(g => g.School)

.WithMany(s => s.Grades)

.HasForeignKey(g => g.SchoolId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.SchoolId, e.Name }).IsUnique();

entity.HasIndex(e => new { e.SchoolId, e.Level });

entity.Property(e => e.Name).IsRequired();

});

modelBuilder.Entity<Subject>(entity =>

{

entity.HasOne(s => s.School)

.WithMany(sch => sch.Subjects)

.HasForeignKey(s => s.SchoolId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.SchoolId, e.Code }).IsUnique();

entity.Property(e => e.Name).IsRequired();

});

// Configure relationship entities

ConfigureRelationshipEntities(modelBuilder);

// Configure attendance

modelBuilder.Entity<Attendance>(entity =>

{

entity.HasOne(a => a.Student)

.WithMany(s => s.AttendanceRecords)

.HasForeignKey(a => a.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(a => a.Subject)

.WithMany()

.HasForeignKey(a => a.SubjectId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(a => a.Teacher)

.WithMany(t => t.AttendanceRecords)

.HasForeignKey(a => a.TeacherId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(a => a.Term)

.WithMany(t => t.AttendanceRecords)

.HasForeignKey(a => a.TermId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(e => new { e.StudentId, e.SubjectId, e.Date }).IsUnique();

});

// Configure results

modelBuilder.Entity<Result>(entity =>

{

entity.HasOne(r => r.Student)

.WithMany(s => s.Results)

.HasForeignKey(r => r.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(r => r.Subject)

.WithMany(s => s.Results)

.HasForeignKey(r => r.SubjectId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(r => r.Teacher)

.WithMany(t => t.Results)

.HasForeignKey(r => r.TeacherId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(r => r.Term)

.WithMany(t => t.Results)

.HasForeignKey(r => r.TermId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(r => r.SchoolYear)

.WithMany(sy => sy.Results)

.HasForeignKey(r => r.SchoolYearId)

.OnDelete(DeleteBehavior.Restrict);

entity.Property(e => e.Score).HasColumnType("decimal(6,2)");

entity.Property(e => e.TotalMarks).HasColumnType("decimal(6,2)");

entity.Property(e => e.Percentage).HasColumnType("decimal(5,2)");

entity.Property(e => e.AiConfidenceScore).HasColumnType("decimal(5,4)");

entity.Property(e => e.PartialCreditScore).HasColumnType("decimal(5,2)");

});

}

private void ConfigureRelationshipEntities(ModelBuilder modelBuilder)

{

modelBuilder.Entity<GradeSubject>(entity =>

{

entity.HasOne(gs => gs.Grade)

.WithMany(g => g.GradeSubjects)

.HasForeignKey(gs => gs.GradeId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(gs => gs.Subject)

.WithMany(s => s.GradeSubjects)

.HasForeignKey(gs => gs.SubjectId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.GradeId, e.SubjectId }).IsUnique();

});

modelBuilder.Entity<SubjectTeacher>(entity =>

{

entity.HasOne(st => st.Subject)

.WithMany(s => s.SubjectTeachers)

.HasForeignKey(st => st.SubjectId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(st => st.Teacher)

.WithMany(t => t.SubjectTeachers)

.HasForeignKey(st => st.TeacherId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(st => st.Grade)

.WithMany()

.HasForeignKey(st => st.GradeId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(st => st.SchoolYear)

.WithMany(sy => sy.SubjectTeachers)

.HasForeignKey(st => st.SchoolYearId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(e => new { e.SubjectId, e.TeacherId, e.GradeId, e.SchoolYearId }).IsUnique();

});

modelBuilder.Entity<StudentSubject>(entity =>

{

entity.HasOne(ss => ss.Student)

.WithMany(s => s.StudentSubjects)

.HasForeignKey(ss => ss.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(ss => ss.Subject)

.WithMany(s => s.StudentSubjects)

.HasForeignKey(ss => ss.SubjectId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(ss => ss.SchoolYear)

.WithMany(sy => sy.StudentSubjects)

.HasForeignKey(ss => ss.SchoolYearId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(e => new { e.StudentId, e.SubjectId, e.SchoolYearId }).IsUnique();

});

modelBuilder.Entity<ClassTeacher>(entity =>

{

entity.HasOne(ct => ct.Teacher)

.WithMany(t => t.ClassTeachers)

.HasForeignKey(ct => ct.TeacherId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(ct => ct.Grade)

.WithMany(g => g.ClassTeachers)

.HasForeignKey(ct => ct.GradeId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(ct => ct.SchoolYear)

.WithMany(sy => sy.ClassTeachers)

.HasForeignKey(ct => ct.SchoolYearId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(e => new { e.GradeId, e.SchoolYearId }).IsUnique();

});

}

private void ConfigureTimetableEntities(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Timetable>(entity =>

{

entity.HasOne(t => t.School)

.WithMany(s => s.Timetables)

.HasForeignKey(t => t.SchoolId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(t => t.Grade)

.WithMany(g => g.Timetables)

.HasForeignKey(t => t.GradeId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(t => t.SchoolYear)

.WithMany()

.HasForeignKey(t => t.SchoolYearId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(t => t.Term)

.WithMany()

.HasForeignKey(t => t.TermId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(e => new { e.GradeId, e.SchoolYearId, e.TermId }).IsUnique();

});

modelBuilder.Entity<TimetableSlot>(entity =>

{

entity.HasOne(ts => ts.Timetable)

.WithMany(t => t.TimetableSlots)

.HasForeignKey(ts => ts.TimetableId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(ts => ts.Subject)

.WithMany(s => s.TimetableSlots)

.HasForeignKey(ts => ts.SubjectId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(ts => ts.Teacher)

.WithMany(t => t.TimetableSlots)

.HasForeignKey(ts => ts.TeacherId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(e => new { e.TimetableId, e.DayOfWeek, e.PeriodNumber }).IsUnique();

entity.HasIndex(e => new { e.TeacherId, e.DayOfWeek, e.StartTime, e.EndTime });

});

modelBuilder.Entity<StudentTimetable>(entity =>

{

entity.HasOne(st => st.Student)

.WithMany(s => s.StudentTimetables)

.HasForeignKey(st => st.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(st => st.TimetableSlot)

.WithMany()

.HasForeignKey(st => st.TimetableSlotId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.StudentId, e.TimetableSlotId }).IsUnique();

});

}

private void ConfigureAIEntities(ModelBuilder modelBuilder)

{

modelBuilder.Entity<HandwritingTrainingData>(entity =>

{

entity.HasOne(htd => htd.Student)

.WithMany(s => s.HandwritingTrainingData)

.HasForeignKey(htd => htd.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(htd => htd.Teacher)

.WithMany()

.HasForeignKey(htd => htd.TeacherId)

.OnDelete(DeleteBehavior.SetNull);

entity.HasIndex(e => new { e.StudentId, e.Status });

entity.HasIndex(e => e.IsVerified);

entity.Property(e => e.ConfidenceScore).HasColumnType("decimal(5,4)");

});

modelBuilder.Entity<HandwritingCharacterData>(entity =>

{

entity.HasOne(hcd => hcd.HandwritingTrainingData)

.WithMany(htd => htd.CharacterData)

.HasForeignKey(hcd => hcd.HandwritingTrainingDataId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.HandwritingTrainingDataId, e.SequenceOrder });

entity.Property(e => e.ConfidenceScore).HasColumnType("decimal(5,4)");

});

modelBuilder.Entity<HandwritingModel>(entity =>

{

entity.HasOne(hm => hm.Student)

.WithMany()

.HasForeignKey(hm => hm.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.StudentId, e.Version }).IsUnique();

entity.HasIndex(e => new { e.StudentId, e.IsActive });

entity.Property(e => e.Accuracy).HasColumnType("decimal(5,4)");

});

modelBuilder.Entity<HandwritingRecognitionResult>(entity =>

{

entity.HasOne(hrr => hrr.Student)

.WithMany()

.HasForeignKey(hrr => hrr.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(hrr => hrr.Subject)

.WithMany()

.HasForeignKey(hrr => hrr.SubjectId)

.OnDelete(DeleteBehavior.SetNull);

entity.HasOne(hrr => hrr.Result)

.WithMany()

.HasForeignKey(hrr => hrr.ResultId)

.OnDelete(DeleteBehavior.SetNull);

entity.HasOne(hrr => hrr.HandwritingModel)

.WithMany(hm => hm.RecognitionResults)

.HasForeignKey(hrr => hrr.HandwritingModelId)

.OnDelete(DeleteBehavior.Restrict);

entity.Property(e => e.OverallConfidence).HasColumnType("decimal(5,4)");

entity.Property(e => e.PartialCreditScore).HasColumnType("decimal(5,2)");

});

modelBuilder.Entity<TrainingExercise>(entity =>

{

entity.HasOne(te => te.School)

.WithMany()

.HasForeignKey(te => te.SchoolId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(te => te.Grade)

.WithMany()

.HasForeignKey(te => te.GradeId)

.OnDelete(DeleteBehavior.SetNull);

entity.HasOne(te => te.Subject)

.WithMany()

.HasForeignKey(te => te.SubjectId)

.OnDelete(DeleteBehavior.SetNull);

entity.HasIndex(e => new { e.SchoolId, e.Type, e.Difficulty });

});

modelBuilder.Entity<StudentTrainingSession>(entity =>

{

entity.HasOne(sts => sts.Student)

.WithMany()

.HasForeignKey(sts => sts.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(sts => sts.TrainingExercise)

.WithMany(te => te.StudentSessions)

.HasForeignKey(sts => sts.TrainingExerciseId)

.OnDelete(DeleteBehavior.Cascade);

entity.Property(e => e.OverallAccuracy).HasColumnType("decimal(5,4)");

});

modelBuilder.Entity<ModelTrainingJob>(entity =>

{

entity.HasOne(mtj => mtj.Student)

.WithMany()

.HasForeignKey(mtj => mtj.StudentId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasIndex(e => new { e.Status, e.QueuedAt });

entity.Property(e => e.Progress).HasColumnType("decimal(5,2)");

});

}

private void ConfigureIndexes(ModelBuilder modelBuilder)

{

// Additional performance indexes

modelBuilder.Entity<Result>()

.HasIndex(e => new { e.StudentId, e.TermId, e.SubjectId });

modelBuilder.Entity<Attendance>()

.HasIndex(e => new { e.Date, e.Status });

modelBuilder.Entity<Notification>()

.HasIndex(e => new { e.Type, e.IsSent, e.CreatedAt });

}

private void ConfigureSoftDelete(ModelBuilder modelBuilder)

{

foreach (var entityType in modelBuilder.Model.GetEntityTypes())

{

if (typeof(BaseEntity).IsAssignableFrom(entityType.ClrType))

{

// Create parameter expression for the entity type

var parameter = Expression.Parameter(entityType.ClrType, "e");

// Convert parameter to BaseEntity and access IsDeleted property

var convertExpression = Expression.Convert(parameter, typeof(BaseEntity));

var propertyAccess = Expression.Property(convertExpression, nameof(BaseEntity.IsDeleted));

// Create negation expression (!IsDeleted)

var notExpression = Expression.Not(propertyAccess);

// Build lambda expression

var lambda = Expression.Lambda(notExpression, parameter);

// Set query filter using the dynamically built expression

modelBuilder.Entity(entityType.ClrType).HasQueryFilter(lambda);

}

}

}

public override int SaveChanges()

{

UpdateTimestamps();

return base.SaveChanges();

}

public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

{

UpdateTimestamps();

return base.SaveChangesAsync(cancellationToken);

}

private void UpdateTimestamps()

{

var entries = ChangeTracker.Entries<BaseEntity>();

foreach (var entry in entries)

{

switch (entry.State)

{

case EntityState.Added:

entry.Entity.CreatedAt = DateTime.UtcNow;

entry.Entity.UpdatedAt = DateTime.UtcNow;

break;

case EntityState.Modified:

entry.Entity.UpdatedAt = DateTime.UtcNow;

break;

}

}

}

}

} using AutoMapper;

using SmartXul.Api.Models.SchoolManagement.Core.Entities;

using SmartXul.Api.Models;

using SmartXul.Shared.DTOs;

namespace SmartXul.Api

{

public class AutoMapperProfile : Profile

{

public AutoMapperProfile()

{

// School mappings

CreateMap<School, SchoolDto>().ReverseMap();

CreateMap<CreateSchoolDto, School>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore());

CreateMap<UpdateSchoolDto, School>()

.ForAllMembers(opts => opts.Condition((src, dest, srcMember) => srcMember != null));

// Student mappings

CreateMap<Student, StudentDto>().ReverseMap();

CreateMap<CreateStudentDto, Student>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore())

.ForMember(dest => dest.Parents, opt => opt.MapFrom(src => src.Parents));

CreateMap<UpdateStudentDto, Student>()

.ForAllMembers(opts => opts.Condition((src, dest, srcMember) => srcMember != null));

// Teacher mappings

CreateMap<Teacher, TeacherDto>().ReverseMap();

CreateMap<CreateTeacherDto, Teacher>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore());

CreateMap<UpdateTeacherDto, Teacher>()

.ForAllMembers(opts => opts.Condition((src, dest, srcMember) => srcMember != null));

// Parent mappings

CreateMap<Parent, ParentDto>().ReverseMap();

CreateMap<CreateParentDto, Parent>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore());

// Grading mappings

CreateMap<GradingScheme, GradingSchemeDto>().ReverseMap();

CreateMap<CreateGradingSchemeDto, GradingScheme>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore())

.ForMember(dest => dest.IsActive, opt => opt.MapFrom(src => true));

CreateMap<UpdateGradingSchemeDto, GradingScheme>()

.ForAllMembers(opts => opts.Condition((src, dest, srcMember) => srcMember != null));

CreateMap<GradingScale, GradingScaleDto>().ReverseMap();

CreateMap<CreateGradingScaleDto, GradingScale>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore());

CreateMap<UpdateGradingScaleDto, GradingScale>()

.ForAllMembers(opts => opts.Condition((src, dest, srcMember) => srcMember != null));

// Core entity mappings

CreateMap<Grade, GradeDto>().ReverseMap();

CreateMap<Subject, SubjectDto>().ReverseMap();

CreateMap<Term, TermDto>().ReverseMap();

CreateMap<SchoolYear, SchoolYearDto>().ReverseMap();

// Result mappings

CreateMap<Result, ResultDto>().ReverseMap();

CreateMap<CreateResultDto, Result>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore())

.ForMember(dest => dest.Percentage, opt => opt.Ignore())

.ForMember(dest => dest.Grade, opt => opt.Ignore())

.ForMember(dest => dest.GradeUnit, opt => opt.Ignore());

CreateMap<UpdateResultDto, Result>()

.ForAllMembers(opts => opts.Condition((src, dest, srcMember) => srcMember != null));

// Attendance mappings

CreateMap<Attendance, AttendanceDto>().ReverseMap();

CreateMap<CreateAttendanceDto, Attendance>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore());

CreateMap<UpdateAttendanceDto, Attendance>()

.ForAllMembers(opts => opts.Condition((src, dest, srcMember) => srcMember != null));

// Timetable mappings

CreateMap<Timetable, TimetableDto>().ReverseMap();

CreateMap<TimetableSlot, TimetableSlotDto>().ReverseMap();

CreateMap<UpdateTimetableSlotDto, TimetableSlot>()

.ForAllMembers(opts => opts.Condition((src, dest, srcMember) => srcMember != null));

// Notification mappings

CreateMap<Notification, NotificationDto>().ReverseMap();

CreateMap<CreateNotificationDto, Notification>()

.ForMember(dest => dest.Id, opt => opt.Ignore())

.ForMember(dest => dest.CreatedAt, opt => opt.Ignore())

.ForMember(dest => dest.UpdatedAt, opt => opt.Ignore())

.ForMember(dest => dest.IsRead, opt => opt.MapFrom(src => false))

.ForMember(dest => dest.IsSent, opt => opt.MapFrom(src => false));

// AI Handwriting mappings

CreateMap<HandwritingTrainingData, HandwritingTrainingDataDto>().ReverseMap();

CreateMap<HandwritingRecognitionResult, HandwritingRecognitionResultDto>()

.ForMember(d => d.AlternativeTexts, opt => opt.MapFrom(src => JsonHelpers.ToStringList(src.AlternativeTexts)))

.ForMember(d => d.CharacterConfidences, opt => opt.MapFrom(src => JsonHelpers.ToDecimalList(src.CharacterConfidences)));

CreateMap<TrainingExercise, TrainingExerciseDto>()

.ForMember(d => d.Content, opt => opt.MapFrom(src => JsonHelpers.ToExerciseContent(src.Content)));

CreateMap<StudentTrainingSession, StudentTrainingSessionDto>().ReverseMap();

CreateMap<ModelTrainingJob, ModelTrainingJobDto>().ReverseMap();

CreateMap<HandwritingModel, HandwritingModelDto>()

.ForMember(d => d.PerformanceMetrics, opt => opt.MapFrom(src => JsonHelpers.ToDictionary(src.PerformanceMetrics)));

}

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api

{

public static class JsonHelpers

{

public static Dictionary<string, object> ToDictionary(string json)

{

if (string.IsNullOrWhiteSpace(json)) return new Dictionary<string, object>();

return System.Text.Json.JsonSerializer.Deserialize<Dictionary<string, object>>(json)

?? new Dictionary<string, object>();

}

public static List<string> ToStringList(string json)

{

if (string.IsNullOrWhiteSpace(json)) return new List<string>();

return System.Text.Json.JsonSerializer.Deserialize<List<string>>(json)

?? new List<string>();

}

public static List<decimal> ToDecimalList(string json)

{

if (string.IsNullOrWhiteSpace(json)) return new List<decimal>();

return System.Text.Json.JsonSerializer.Deserialize<List<decimal>>(json)

?? new List<decimal>();

}

public static ExerciseContentDto ToExerciseContent(string json)

{

if (string.IsNullOrWhiteSpace(json)) return new ExerciseContentDto();

return System.Text.Json.JsonSerializer.Deserialize<ExerciseContentDto>(json)

?? new ExerciseContentDto();

}

}

}

Service Interfaces

using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface IAttendanceService

{

Task<AttendanceDto> MarkAttendanceAsync(CreateAttendanceDto createAttendanceDto);

Task<IEnumerable<AttendanceDto>> GetAttendanceByClassAsync(Guid gradeId, DateTime date);

Task<IEnumerable<AttendanceDto>> GetStudentAttendanceAsync(Guid studentId, Guid termId);

Task<AttendanceReportDto> GetAttendanceReportAsync(Guid studentId, Guid termId);

Task<bool> BulkMarkAttendanceAsync(List<CreateAttendanceDto> attendanceList);

Task<AttendanceDto> UpdateAttendanceAsync(Guid attendanceId, UpdateAttendanceDto updateDto);

}

} using SmartXul.Shared.DTOs;

using SmartXul.Shared.DTOs.Exams;

using SmartXul.Shared.Enums;

namespace SmartXul.Api.Services.Interfaces

{

public interface IExamService

{

// Exam Management

Task<ExamDto> CreateExamAsync(CreateExamDto createExamDto, Guid teacherId);

Task<ExamDto> UpdateExamAsync(Guid examId, UpdateExamDto updateExamDto);

Task<bool> DeleteExamAsync(Guid examId);

Task<ExamDto> GetExamByIdAsync(Guid examId);

Task<IEnumerable<ExamDto>> GetExamsBySchoolAsync(Guid schoolId);

Task<IEnumerable<ExamDto>> GetExamsBySubjectAsync(Guid subjectId, Guid termId);

Task<IEnumerable<ExamDto>> GetExamsByGradeAsync(Guid gradeId, Guid termId);

Task<IEnumerable<ExamDto>> GetUpcomingExamsAsync(Guid schoolId, DateTime fromDate, DateTime toDate);

// Exam Publication

Task<bool> PublishExamAsync(Guid examId);

Task<bool> UnpublishExamAsync(Guid examId);

// Student Registration

Task<bool> RegisterStudentForExamAsync(Guid examId, Guid studentId);

Task<bool> RegisterStudentsForExamAsync(Guid examId, List<Guid> studentIds);

Task<bool> UnregisterStudentFromExamAsync(Guid examId, Guid studentId);

Task<IEnumerable<ExamRegistrationDto>> GetExamRegistrationsAsync(Guid examId);

// Exam Sessions

Task<ExamSessionDto> StartExamSessionAsync(StartExamSessionDto startSessionDto);

Task<ExamSessionDto> GetExamSessionAsync(Guid sessionId);

Task<bool> EndExamSessionAsync(Guid sessionId);

Task<bool> PauseExamSessionAsync(Guid sessionId);

Task<bool> ResumeExamSessionAsync(Guid sessionId);

// Answer Submission

Task<bool> SubmitAnswerAsync(SubmitExamAnswerDto submitAnswerDto);

Task<bool> SaveAnswerDraftAsync(SubmitExamAnswerDto saveAnswerDto);

Task<bool> SubmitExamAsync(Guid sessionId);

// Marking & Results

Task<bool> MarkAnswerAsync(Guid answerId, decimal marks, string feedback);

Task<bool> AutoMarkExamAsync(Guid examId);

Task<ExamResultDto> GenerateExamResultAsync(Guid sessionId);

Task<bool> PublishResultsAsync(Guid examId);

Task<IEnumerable<ExamResultDto>> GetExamResultsAsync(Guid examId);

Task<ExamResultDto> GetStudentExamResultAsync(Guid examId, Guid studentId);

// Statistics & Analytics

Task<ExamStatisticsDto> GetExamStatisticsAsync(Guid examId);

Task<IEnumerable<QuestionAnalysisDto>> GetQuestionAnalysisAsync(Guid examId);

Task<byte[]> GenerateExamReportAsync(Guid examId);

Task<byte[]> GenerateStudentExamResultAsync(Guid examResultId);

// Supervision & Security

Task<bool> AssignSupervisorAsync(Guid examId, Guid teacherId, SupervisorRole role);

Task<bool> RemoveSupervisorAsync(Guid examId, Guid teacherId);

Task<bool> CheckInSupervisorAsync(Guid examId, Guid teacherId);

Task<bool> CheckOutSupervisorAsync(Guid examId, Guid teacherId);

Task<bool> ReportIncidentAsync(CreateExamIncidentDto incidentDto);

Task<IEnumerable<ExamIncidentDto>> GetExamIncidentsAsync(Guid examId);

// Student Experience

Task<IEnumerable<ExamDto>> GetStudentUpcomingExamsAsync(Guid studentId);

Task<IEnumerable<ExamResultDto>> GetStudentExamHistoryAsync(Guid studentId);

Task<ExamDto> GetStudentExamDetailsAsync(Guid examId, Guid studentId);

Task<bool> ValidateExamAccessAsync(Guid examId, Guid studentId);

// Handwriting Integration

Task<bool> ProcessHandwrittenAnswersAsync(Guid examId);

Task<bool> ReviewHandwritingRecognitionAsync(Guid answerId, string correctedText);

}

} namespace SmartXul.Api.Services.Interfaces

{

public interface IFileStorageService

{

Task<string> SaveImageAsync(byte[] imageData, string relativePath);

Task<string> SaveFileAsync(byte[] fileData, string relativePath, string contentType);

Task<byte[]> GetFileAsync(string filePath);

Task<bool> DeleteFileAsync(string filePath);

Task<string> DownloadModelAsync(string cloudPath, Guid modelId);

Task<string> UploadModelAsync(string localPath, string cloudPath);

Task<List<string>> GetFilesInDirectoryAsync(string directoryPath);

Task<bool> FileExistsAsync(string filePath);

Task<long> GetFileSizeAsync(string filePath);

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface IGradingService

{

Task<IEnumerable<GradingSchemeDto>> GetAllGradingSchemesAsync();

Task<GradingSchemeDto> CreateGradingSchemeAsync(CreateGradingSchemeDto createDto);

Task<GradingSchemeDto> UpdateGradingSchemeAsync(Guid schemeId, UpdateGradingSchemeDto updateDto);

Task<bool> DeleteGradingSchemeAsync(Guid schemeId);

Task<GradingScaleDto> AddGradingScaleAsync(Guid schemeId, CreateGradingScaleDto createDto);

Task<GradingScaleDto> UpdateGradingScaleAsync(Guid scaleId, UpdateGradingScaleDto updateDto);

Task<bool> DeleteGradingScaleAsync(Guid scaleId);

Task<GradeCalculationDto> CalculateGradeAsync(Guid gradingSchemeId, decimal percentage);

Task<IEnumerable<GradingScaleDto>> GetGradingScalesAsync(Guid schemeId);

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface IHandwritingRecognitionService

{

Task<HandwritingRecognitionResultDto> ProcessHandwritingAsync(ProcessHandwritingDto processDto);

Task<TrainingExerciseDto> GenerateTrainingExerciseAsync(GenerateTrainingExerciseDto generateDto);

Task<StudentTrainingSessionDto> StartTrainingSessionAsync(Guid studentId, Guid exerciseId);

Task<StudentTrainingSessionDto> SubmitTrainingDataAsync(SubmitTrainingDataDto submitDto);

Task<bool> VerifyTrainingDataAsync(Guid trainingDataId, string verifiedText);

Task<ModelTrainingJobDto> StartModelTrainingAsync(Guid studentId);

Task<ModelTrainingJobDto> GetTrainingJobStatusAsync(Guid jobId);

Task<HandwritingModelDto> GetStudentModelAsync(Guid studentId);

Task<bool> DeployModelLocallyAsync(Guid modelId);

Task<TrainingProgressDto> GetStudentTrainingProgressAsync(Guid studentId);

Task<IEnumerable<HandwritingTrainingDataDto>> GetUnverifiedTrainingDataAsync(Guid teacherId);

}

} using SmartXul.Api.Models.SupportingModels;

namespace SmartXul.Api.Services.Interfaces

{

public interface IImageProcessingService

{

Task<ImageProcessingResult> ProcessImageAsync(string imagePath);

Task<TextExtractionResult> ExtractTextAsync(string imagePath);

Task<List<CharacterSegment>> SegmentCharactersAsync(string imagePath);

Task<string> PreprocessImageAsync(string imagePath);

Task<bool> ValidateImageQualityAsync(string imagePath);

}

} using SmartXul.Api.Models;

using SmartXul.Api.Models.SupportingModels;

namespace SmartXul.Api.Services.Interfaces

{

public interface IMachineLearningService

{

Task<LocalRecognitionResult> RecognizeTextLocallyAsync(string modelPath, string imagePath);

Task<CloudRecognitionResult> RecognizeTextInCloudAsync(string imagePath);

Task<ModelTrainingResult> TrainPersonalModelAsync(List<HandwritingTrainingData> trainingData);

Task<bool> ValidateModelAsync(string modelPath);

Task<ModelMetrics> EvaluateModelAsync(string modelPath, List<HandwritingTrainingData> testData);

Task<string> DeployModelToCloudAsync(string localModelPath);

Task<bool> UpdateModelAsync(string existingModelPath, List<HandwritingTrainingData> newTrainingData);

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface INotificationService

{

Task<NotificationDto> CreateNotificationAsync(CreateNotificationDto createDto);

Task<bool> SendNotificationAsync(Guid notificationId);

Task<bool> SendBulkNotificationAsync(List<Guid> recipientIds, CreateNotificationDto notificationDto);

Task<IEnumerable<NotificationDto>> GetUserNotificationsAsync(Guid userId, string userType);

Task<bool> MarkAsReadAsync(Guid notificationId);

Task<bool> SendWhatsAppMessageAsync(string phoneNumber, string message);

Task<bool> SendEmailNotificationAsync(string email, string subject, string message);

Task<NotificationStatsDto> GetNotificationStatsAsync(Guid schoolId);

}

} using SmartXul.Shared.DTOs;

using SmartXul.Shared.DTOs.Exams;

namespace SmartXul.Api.Services.Interfaces

{

public interface IPdfGenerationService

{

Task<byte[]> GenerateReportCardPdfAsync(StudentReportCardDto reportCard);

Task<byte[]> GenerateClassResultsPdfAsync(ClassResultSummaryDto classSummary);

Task<byte[]> GenerateTimetablePdfAsync(TimetableDto timetable);

Task<byte[]> GenerateAttendanceReportPdfAsync(AttendanceReportDto attendanceReport);

Task<byte[]> GenerateCertificatePdfAsync(StudentDto student, string certificateType);

Task<byte[]> GenerateTranscriptPdfAsync(StudentDto student, List<ResultDto> results);

Task<byte[]> GenerateStudentExamResultAsync(ExamResultDto resultDto);

Task<byte[]> GenerateExamAnalysisReportAsync(ExamStatisticsDto stats);

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface IResultService

{

Task<ResultDto> CreateResultAsync(CreateResultDto createResultDto);

Task<ResultDto> UpdateResultAsync(Guid resultId, UpdateResultDto updateResultDto);

Task<IEnumerable<ResultDto>> GetStudentResultsAsync(Guid studentId, Guid termId);

Task<IEnumerable<ResultDto>> GetClassResultsAsync(Guid gradeId, Guid subjectId, Guid termId);

Task<StudentReportCardDto> GenerateReportCardAsync(Guid studentId, Guid termId);

Task<bool> UploadResultsAsync(Guid teacherId, List<CreateResultDto> results);

Task<ClassResultSummaryDto> GetClassResultSummaryAsync(Guid gradeId, Guid termId);

Task<byte[]> PrintTermResultsAsync(Guid studentId, Guid termId);

Task<byte[]> PrintClassResultsAsync(Guid gradeId, Guid termId);

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface ISchoolService

{

Task<IEnumerable<SchoolDto>> GetAllSchoolsAsync();

Task<SchoolDto> GetSchoolByIdAsync(Guid schoolId);

Task<SchoolDto> CreateSchoolAsync(CreateSchoolDto createSchoolDto);

Task<SchoolDto> UpdateSchoolAsync(Guid schoolId, UpdateSchoolDto updateSchoolDto);

Task<bool> DeleteSchoolAsync(Guid schoolId);

Task<IEnumerable<SchoolDto>> GetSchoolsByGradingSchemeAsync(Guid gradingSchemeId);

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface IStudentService

{

Task<IEnumerable<StudentDto>> GetStudentsBySchoolAsync(Guid schoolId);

Task<IEnumerable<StudentDto>> GetStudentsByGradeAsync(Guid gradeId);

Task<StudentDto> GetStudentByIdAsync(Guid studentId);

Task<StudentDto> CreateStudentAsync(CreateStudentDto createStudentDto);

Task<StudentDto> UpdateStudentAsync(Guid studentId, UpdateStudentDto updateStudentDto);

Task<bool> DeleteStudentAsync(Guid studentId);

Task<bool> AssignStudentToSubjectsAsync(Guid studentId, List<Guid> subjectIds);

Task<IEnumerable<SubjectDto>> GetStudentSubjectsAsync(Guid studentId, Guid schoolYearId);

Task<StudentTimetableDto> GetStudentTimetableAsync(Guid studentId);

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface ITeacherService

{

Task<IEnumerable<TeacherDto>> GetTeachersBySchoolAsync(Guid schoolId);

Task<TeacherDto> GetTeacherByIdAsync(Guid teacherId);

Task<TeacherDto> CreateTeacherAsync(CreateTeacherDto createTeacherDto);

Task<TeacherDto> UpdateTeacherAsync(Guid teacherId, UpdateTeacherDto updateTeacherDto);

Task<bool> DeleteTeacherAsync(Guid teacherId);

Task<bool> AssignTeacherToSubjectAsync(AssignTeacherSubjectDto assignDto);

Task<bool> AssignClassTeacherAsync(AssignClassTeacherDto assignDto);

Task<IEnumerable<SubjectDto>> GetTeacherSubjectsAsync(Guid teacherId, Guid schoolYearId);

Task<IEnumerable<GradeDto>> GetTeacherClassesAsync(Guid teacherId, Guid schoolYearId);

}

} using SmartXul.Shared.DTOs;

namespace SmartXul.Api.Services.Interfaces

{

public interface ITimetableService

{

Task<TimetableDto> GenerateTimetableAsync(GenerateTimetableDto generateDto);

Task<TimetableDto> GetGradeTimetableAsync(Guid gradeId, Guid termId);

Task<StudentTimetableDto> GetStudentTimetableAsync(Guid studentId);

Task<TeacherTimetableDto> GetTeacherTimetableAsync(Guid teacherId, Guid termId);

Task<TimetableSlotDto> UpdateTimetableSlotAsync(Guid slotId, UpdateTimetableSlotDto updateDto);

Task<bool> ValidateTimetableAsync(Guid timetableId);

Task<TimetableConflictDto> CheckTimetableConflictsAsync(Guid timetableId);

Task<byte[]> PrintTimetableAsync(Guid timetableId);

}

} using SmartXul.Api.Models.SupportingModels;

namespace SmartXul.Api.Services.Interfaces

{

public interface IWhatsAppService

{

Task<bool> SendMessageAsync(string phoneNumber, string message);

Task<bool> SendMediaMessageAsync(string phoneNumber, string mediaUrl, string caption);

Task<bool> SendTemplateMessageAsync(string phoneNumber, string templateName, Dictionary<string, string> parameters);

Task<List<WhatsAppMessageStatus>> GetMessageStatusAsync(List<string> messageIds);

}

}

Services Implementation

using AutoMapper;

using Microsoft.EntityFrameworkCore;

using SmartXul.Api.Models.SchoolManagement.Core.Entities;

using SmartXul.Api.Services.Interfaces;

using SmartXul.Shared.DTOs;

using SmartXul.Shared.Exceptions;

namespace SmartXul.Api.Services.Implementations

{

public class GradingService : IGradingService

{

private readonly SmartXulDbContext \_context;

private readonly IMapper \_mapper;

public GradingService(SmartXulDbContext context, IMapper mapper)

{

\_context = context;

\_mapper = mapper;

}

public async Task<IEnumerable<GradingSchemeDto>> GetAllGradingSchemesAsync()

{

var schemes = await \_context.GradingSchemes

.Include(gs => gs.GradingScales.OrderBy(scale => scale.SortOrder))

.Where(gs => gs.IsActive)

.ToListAsync();

return \_mapper.Map<IEnumerable<GradingSchemeDto>>(schemes);

}

public async Task<GradingSchemeDto> CreateGradingSchemeAsync(CreateGradingSchemeDto createDto)

{

var scheme = \_mapper.Map<GradingScheme>(createDto);

\_context.GradingSchemes.Add(scheme);

await \_context.SaveChangesAsync();

// Add grading scales

if (createDto.GradingScales.Any())

{

var scales = createDto.GradingScales.Select(scaleDto =>

{

var scale = \_mapper.Map<GradingScale>(scaleDto);

scale.GradingSchemeId = scheme.Id;

return scale;

}).ToList();

\_context.GradingScales.AddRange(scales);

await \_context.SaveChangesAsync();

}

return await GetGradingSchemeByIdAsync(scheme.Id);

}

public async Task<GradingSchemeDto> UpdateGradingSchemeAsync(Guid schemeId, UpdateGradingSchemeDto updateDto)

{

var scheme = await \_context.GradingSchemes.FindAsync(schemeId);

if (scheme == null)

throw new NotFoundException($"Grading scheme {schemeId} not found");

\_mapper.Map(updateDto, scheme);

await \_context.SaveChangesAsync();

return await GetGradingSchemeByIdAsync(schemeId);

}

public async Task<bool> DeleteGradingSchemeAsync(Guid schemeId)

{

var scheme = await \_context.GradingSchemes.FindAsync(schemeId);

if (scheme == null)

return false;

// Check if scheme is being used by any schools

var isInUse = await \_context.Schools.AnyAsync(s => s.GradingSchemeId == schemeId);

if (isInUse)

{

throw new ConflictException("Cannot delete grading scheme that is in use by schools");

}

scheme.IsActive = false;

await \_context.SaveChangesAsync();

return true;

}

public async Task<GradingScaleDto> AddGradingScaleAsync(Guid schemeId, CreateGradingScaleDto createDto)

{

var scheme = await \_context.GradingSchemes.FindAsync(schemeId);

if (scheme == null)

throw new NotFoundException($"Grading scheme {schemeId} not found");

var scale = \_mapper.Map<GradingScale>(createDto);

scale.GradingSchemeId = schemeId;

\_context.GradingScales.Add(scale);

await \_context.SaveChangesAsync();

return \_mapper.Map<GradingScaleDto>(scale);

}

public async Task<GradingScaleDto> UpdateGradingScaleAsync(Guid scaleId, UpdateGradingScaleDto updateDto)

{

var scale = await \_context.GradingScales.FindAsync(scaleId);

if (scale == null)

throw new NotFoundException($"Grading scale {scaleId} not found");

\_mapper.Map(updateDto, scale);

await \_context.SaveChangesAsync();

return \_mapper.Map<GradingScaleDto>(scale);

}

public async Task<bool> DeleteGradingScaleAsync(Guid scaleId)

{

var scale = await \_context.GradingScales.FindAsync(scaleId);

if (scale == null)

return false;

\_context.GradingScales.Remove(scale);

await \_context.SaveChangesAsync();

return true;

}

public async Task<GradeCalculationDto> CalculateGradeAsync(Guid gradingSchemeId, decimal percentage)

{

var gradingScale = await \_context.GradingScales

.Where(gs => gs.GradingSchemeId == gradingSchemeId &&

percentage >= gs.MinPercentage &&

percentage <= gs.MaxPercentage)

.FirstOrDefaultAsync();

if (gradingScale == null)

{

// Return default grade if no matching scale found

return new GradeCalculationDto

{

Symbol = "F",

Unit = 0,

Description = "Fail",

Percentage = percentage

};

}

return new GradeCalculationDto

{

Symbol = gradingScale.Symbol,

Unit = gradingScale.Unit,

Description = gradingScale.Description,

Percentage = percentage

};

}

public async Task<IEnumerable<GradingScaleDto>> GetGradingScalesAsync(Guid schemeId)

{

var scales = await \_context.GradingScales

.Where(gs => gs.GradingSchemeId == schemeId)

.OrderBy(gs => gs.SortOrder)

.ToListAsync();

return \_mapper.Map<IEnumerable<GradingScaleDto>>(scales);

}

private async Task<GradingSchemeDto> GetGradingSchemeByIdAsync(Guid schemeId)

{

var scheme = await \_context.GradingSchemes

.Include(gs => gs.GradingScales.OrderBy(scale => scale.SortOrder))

.FirstOrDefaultAsync(gs => gs.Id == schemeId);

if (scheme == null)

throw new NotFoundException($"Grading scheme {schemeId} not found");

return \_mapper.Map<GradingSchemeDto>(scheme);

}

}

} using AutoMapper;

using Microsoft.EntityFrameworkCore;

using SmartXul.Api.Models;

using SmartXul.Api.Services.Interfaces;

using SmartXul.Shared.DTOs;

using SmartXul.Shared.Exceptions;

namespace SmartXul.Api.Services.Implementations

{

public class SchoolService : ISchoolService

{

private readonly SmartXulDbContext \_context;

private readonly IMapper \_mapper;

public SchoolService(SmartXulDbContext context, IMapper mapper)

{

\_context = context;

\_mapper = mapper;

}

public async Task<IEnumerable<SchoolDto>> GetAllSchoolsAsync()

{

var schools = await \_context.Schools

.Include(s => s.GradingScheme)

.ToListAsync();

return \_mapper.Map<IEnumerable<SchoolDto>>(schools);

}

public async Task<SchoolDto> GetSchoolByIdAsync(Guid schoolId)

{

var school = await \_context.Schools

.Include(s => s.GradingScheme)

.FirstOrDefaultAsync(s => s.Id == schoolId);

if (school == null)

throw new NotFoundException($"School with ID {schoolId} not found");

return \_mapper.Map<SchoolDto>(school);

}

public async Task<SchoolDto> CreateSchoolAsync(CreateSchoolDto createSchoolDto)

{

var school = \_mapper.Map<School>(createSchoolDto);

\_context.Schools.Add(school);

await \_context.SaveChangesAsync();

return \_mapper.Map<SchoolDto>(school);

}

public async Task<SchoolDto> UpdateSchoolAsync(Guid schoolId, UpdateSchoolDto updateSchoolDto)

{

var school = await \_context.Schools.FindAsync(schoolId);

if (school == null)

throw new NotFoundException($"School with ID {schoolId} not found");

\_mapper.Map(updateSchoolDto, school);

await \_context.SaveChangesAsync();

return \_mapper.Map<SchoolDto>(school);

}

public async Task<bool> DeleteSchoolAsync(Guid schoolId)

{

var school = await \_context.Schools.FindAsync(schoolId);

if (school == null)

return false;

school.IsDeleted = true;

await \_context.SaveChangesAsync();

return true;

}

public async Task<IEnumerable<SchoolDto>> GetSchoolsByGradingSchemeAsync(Guid gradingSchemeId)

{

var schools = await \_context.Schools

.Where(s => s.GradingSchemeId == gradingSchemeId)

.Include(s => s.GradingScheme)

.ToListAsync();

return \_mapper.Map<IEnumerable<SchoolDto>>(schools);

}

}

} using AutoMapper;

using Microsoft.EntityFrameworkCore;

using SmartXul.Api.Models;

using SmartXul.Api.Services.Interfaces;

using SmartXul.Shared.DTOs;

using SmartXul.Shared.Exceptions;

namespace SmartXul.Api.Services.Implementations

{

public class StudentService : IStudentService

{

private readonly SmartXulDbContext \_context;

private readonly IMapper \_mapper;

public StudentService(SmartXulDbContext context, IMapper mapper)

{

\_context = context;

\_mapper = mapper;

}

public async Task<IEnumerable<StudentDto>> GetStudentsBySchoolAsync(Guid schoolId)

{

var students = await \_context.Students

.Where(s => s.SchoolId == schoolId)

.Include(s => s.CurrentGrade)

.Include(s => s.Parents)

.ToListAsync();

return \_mapper.Map<IEnumerable<StudentDto>>(students);

}

public async Task<IEnumerable<StudentDto>> GetStudentsByGradeAsync(Guid gradeId)

{

var students = await \_context.Students

.Where(s => s.CurrentGradeId == gradeId)

.Include(s => s.CurrentGrade)

.Include(s => s.Parents)

.ToListAsync();

return \_mapper.Map<IEnumerable<StudentDto>>(students);

}

public async Task<StudentDto> GetStudentByIdAsync(Guid studentId)

{

var student = await \_context.Students

.Include(s => s.CurrentGrade)

.Include(s => s.Parents)

.Include(s => s.School)

.FirstOrDefaultAsync(s => s.Id == studentId);

if (student == null)

throw new NotFoundException($"Student with ID {studentId} not found");

return \_mapper.Map<StudentDto>(student);

}

public async Task<StudentDto> CreateStudentAsync(CreateStudentDto createStudentDto)

{

var student = \_mapper.Map<Student>(createStudentDto);

// Generate student number if not provided

if (string.IsNullOrEmpty(student.StudentNumber))

{

student.StudentNumber = await GenerateStudentNumberAsync(student.SchoolId);

}

\_context.Students.Add(student);

await \_context.SaveChangesAsync();

return \_mapper.Map<StudentDto>(student);

}

public async Task<StudentDto> UpdateStudentAsync(Guid studentId, UpdateStudentDto updateStudentDto)

{

var student = await \_context.Students.FindAsync(studentId);

if (student == null)

throw new NotFoundException($"Student with ID {studentId} not found");

\_mapper.Map(updateStudentDto, student);

await \_context.SaveChangesAsync();

return \_mapper.Map<StudentDto>(student);

}

public async Task<bool> DeleteStudentAsync(Guid studentId)

{

var student = await \_context.Students.FindAsync(studentId);

if (student == null)

return false;

student.IsDeleted = true;

await \_context.SaveChangesAsync();

return true;

}

public async Task<bool> AssignStudentToSubjectsAsync(Guid studentId, List<Guid> subjectIds)

{

var student = await \_context.Students.FindAsync(studentId);

if (student == null)

return false;

var currentSchoolYear = await \_context.SchoolYears

.Where(sy => sy.SchoolId == student.SchoolId && sy.IsCurrent)

.FirstOrDefaultAsync();

if (currentSchoolYear == null)

return false;

// Remove existing assignments for current school year

var existingAssignments = await \_context.StudentSubjects

.Where(ss => ss.StudentId == studentId && ss.SchoolYearId == currentSchoolYear.Id)

.ToListAsync();

\_context.StudentSubjects.RemoveRange(existingAssignments);

// Add new assignments

var newAssignments = subjectIds.Select(subjectId => new StudentSubject

{

StudentId = studentId,

SubjectId = subjectId,

SchoolYearId = currentSchoolYear.Id,

EnrollmentDate = DateTime.UtcNow

});

\_context.StudentSubjects.AddRange(newAssignments);

await \_context.SaveChangesAsync();

return true;

}

public async Task<IEnumerable<SubjectDto>> GetStudentSubjectsAsync(Guid studentId, Guid schoolYearId)

{

var subjects = await \_context.StudentSubjects

.Where(ss => ss.StudentId == studentId && ss.SchoolYearId == schoolYearId && ss.IsActive)

.Include(ss => ss.Subject)

.Select(ss => ss.Subject)

.ToListAsync();

return \_mapper.Map<IEnumerable<SubjectDto>>(subjects);

}

public async Task<StudentTimetableDto> GetStudentTimetableAsync(Guid studentId)

{

var student = await \_context.Students

.Include(s => s.CurrentGrade)

.FirstOrDefaultAsync(s => s.Id == studentId);

if (student == null)

throw new NotFoundException($"Student with ID {studentId} not found");

var currentTerm = await \_context.Terms

.Where(t => t.SchoolYear.SchoolId == student.SchoolId && t.IsCurrent)

.FirstOrDefaultAsync();

if (currentTerm == null)

throw new NotFoundException("No current term found");

var timetable = await \_context.Timetables

.Include(t => t.TimetableSlots)

.ThenInclude(ts => ts.Subject)

.Include(t => t.TimetableSlots)

.ThenInclude(ts => ts.Teacher)

.FirstOrDefaultAsync(t => t.GradeId == student.CurrentGradeId && t.TermId == currentTerm.Id);

return \_mapper.Map<StudentTimetableDto>(timetable);

}

private async Task<string> GenerateStudentNumberAsync(Guid schoolId)

{

var year = DateTime.Now.Year;

var lastStudent = await \_context.Students

.Where(s => s.SchoolId == schoolId && s.StudentNumber.StartsWith(year.ToString()))

.OrderByDescending(s => s.StudentNumber)

.FirstOrDefaultAsync();

var sequence = 1;

if (lastStudent != null && int.TryParse(lastStudent.StudentNumber.Substring(4), out var lastSequence))

{

sequence = lastSequence + 1;

}

return $"{year}{sequence:D4}";

}

}

} using AutoMapper;

using Microsoft.EntityFrameworkCore;

using SmartXul.Api.Models;

using SmartXul.Api.Services.Interfaces;

using SmartXul.Shared.DTOs;

using SmartXul.Shared.Enums;

using SmartXul.Shared.Exceptions;

namespace SmartXul.Api.Services.Implementations

{

public class TimetableService : ITimetableService

{

private readonly SmartXulDbContext \_context;

private readonly IMapper \_mapper;

private readonly ILogger<TimetableService> \_logger;

private readonly IPdfGenerationService \_pdfService;

public TimetableService(

SmartXulDbContext context,

IMapper mapper,

ILogger<TimetableService> logger,

IPdfGenerationService pdfService)

{

\_context = context;

\_mapper = mapper;

\_logger = logger;

\_pdfService = pdfService;

}

public async Task<TimetableDto> GenerateTimetableAsync(GenerateTimetableDto generateDto)

{

try

{

\_logger.LogInformation("Starting timetable generation for Grade {GradeId}", generateDto.GradeId);

// Validate input data

await ValidateGenerationInputAsync(generateDto);

// Get required data

var gradeSubjects = await GetGradeSubjectsAsync(generateDto.GradeId, generateDto.SchoolYearId);

var teachers = await GetAvailableTeachersAsync(generateDto.SchoolId, generateDto.SchoolYearId);

// Create timetable entity

var timetable = new Timetable

{

SchoolId = generateDto.SchoolId,

GradeId = generateDto.GradeId,

SchoolYearId = generateDto.SchoolYearId,

TermId = generateDto.TermId,

Name = generateDto.Name,

IsActive = true,

IsGenerated = true,

GeneratedAt = DateTime.UtcNow

};

\_context.Timetables.Add(timetable);

await \_context.SaveChangesAsync();

// Generate timetable slots

var generatedSlots = await GenerateTimetableSlotsAsync(

timetable.Id,

gradeSubjects,

teachers,

generateDto.Rules);

timetable.TimetableSlots = generatedSlots;

await \_context.SaveChangesAsync();

// Validate the generated timetable

var conflicts = await CheckTimetableConflictsAsync(timetable.Id);

if (conflicts.HasConflicts)

{

\_logger.LogWarning("Generated timetable has conflicts: {ConflictCount}", conflicts.Conflicts.Count);

// Attempt to resolve conflicts

await ResolveConflictsAsync(timetable.Id, conflicts);

}

// Create student timetable assignments

await CreateStudentTimetableAssignmentsAsync(timetable.Id, generateDto.GradeId);

var result = await GetTimetableWithDetailsAsync(timetable.Id);

\_logger.LogInformation("Timetable generation completed for Grade {GradeId}", generateDto.GradeId);

return \_mapper.Map<TimetableDto>(result);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Error generating timetable for Grade {GradeId}", generateDto.GradeId);

throw;

}

}

public async Task<TimetableDto> GetGradeTimetableAsync(Guid gradeId, Guid termId)

{

var timetable = await \_context.Timetables

.Include(t => t.School)

.Include(t => t.Grade)

.Include(t => t.SchoolYear)

.Include(t => t.Term)

.Include(t => t.TimetableSlots)

.ThenInclude(ts => ts.Subject)

.Include(t => t.TimetableSlots)

.ThenInclude(ts => ts.Teacher)

.FirstOrDefaultAsync(t => t.GradeId == gradeId && t.TermId == termId && t.IsActive);

if (timetable == null)

throw new NotFoundException($"No active timetable found for grade {gradeId} in term {termId}");

return \_mapper.Map<TimetableDto>(timetable);

}

public async Task<StudentTimetableDto> GetStudentTimetableAsync(Guid studentId)

{

var student = await \_context.Students

.Include(s => s.CurrentGrade)

.Include(s => s.School)

.FirstOrDefaultAsync(s => s.Id == studentId);

if (student == null)

throw new NotFoundException($"Student {studentId} not found");

var currentTerm = await \_context.Terms

.Where(t => t.SchoolYear.SchoolId == student.SchoolId && t.IsCurrent)

.FirstOrDefaultAsync();

if (currentTerm == null)

throw new NotFoundException("No current term found");

var timetable = await GetGradeTimetableAsync(student.CurrentGradeId, currentTerm.Id);

// Get student-specific assignments (for elective subjects)

var studentSpecificSlots = await \_context.StudentTimetables

.Where(st => st.StudentId == studentId && st.IsActive)

.Include(st => st.TimetableSlot)

.ThenInclude(ts => ts.Subject)

.Include(st => st.TimetableSlot)

.ThenInclude(ts => ts.Teacher)

.Select(st => st.TimetableSlot)

.ToListAsync();

// Filter timetable based on student's subjects

var studentSubjects = await \_context.StudentSubjects

.Where(ss => ss.StudentId == studentId && ss.SchoolYearId == timetable.SchoolYear.Id && ss.IsActive)

.Select(ss => ss.SubjectId)

.ToListAsync();

var filteredSlots = timetable.TimetableSlots

.Where(slot => studentSubjects.Contains(slot.Subject.Id))

.Concat(studentSpecificSlots.Select(slot => \_mapper.Map<TimetableSlotDto>(slot)))

.Distinct()

.OrderBy(slot => slot.DayOfWeek)

.ThenBy(slot => slot.PeriodNumber)

.ToList();

return new StudentTimetableDto

{

Student = \_mapper.Map<StudentDto>(student),

TimetableSlots = filteredSlots,

TimetableName = timetable.Name

};

}

public async Task<TeacherTimetableDto> GetTeacherTimetableAsync(Guid teacherId, Guid termId)

{

var teacher = await \_context.Teachers

.Include(t => t.School)

.FirstOrDefaultAsync(t => t.Id == teacherId);

if (teacher == null)

throw new NotFoundException($"Teacher {teacherId} not found");

var term = await \_context.Terms.FindAsync(termId);

if (term == null)

throw new NotFoundException($"Term {termId} not found");

var teacherSlots = await \_context.TimetableSlots

.Where(ts => ts.TeacherId == teacherId && ts.Timetable.TermId == termId)

.Include(ts => ts.Subject)

.Include(ts => ts.Teacher)

.Include(ts => ts.Timetable)

.ThenInclude(t => t.Grade)

.OrderBy(ts => ts.DayOfWeek)

.ThenBy(ts => ts.PeriodNumber)

.ToListAsync();

return new TeacherTimetableDto

{

Teacher = \_mapper.Map<TeacherDto>(teacher),

Term = \_mapper.Map<TermDto>(term),

TimetableSlots = \_mapper.Map<List<TimetableSlotDto>>(teacherSlots)

};

}

public async Task<TimetableSlotDto> UpdateTimetableSlotAsync(Guid slotId, UpdateTimetableSlotDto updateDto)

{

var slot = await \_context.TimetableSlots.FindAsync(slotId);

if (slot == null)

throw new NotFoundException($"Timetable slot {slotId} not found");

// Check for conflicts before updating

if (updateDto.TeacherId.HasValue || updateDto.StartTime.HasValue || updateDto.EndTime.HasValue)

{

var tempSlot = new TimetableSlot

{

Id = slot.Id,

TimetableId = slot.TimetableId,

TeacherId = updateDto.TeacherId ?? slot.TeacherId,

SubjectId = updateDto.SubjectId ?? slot.SubjectId,

DayOfWeek = slot.DayOfWeek,

StartTime = updateDto.StartTime ?? slot.StartTime,

EndTime = updateDto.EndTime ?? slot.EndTime,

Room = updateDto.Room ?? slot.Room,

PeriodNumber = slot.PeriodNumber

};

var conflicts = await ValidateSlotConflictsAsync(tempSlot);

if (conflicts.Any())

{

throw new ConflictException($"Update would create conflicts: {string.Join(", ", conflicts)}");

}

}

\_mapper.Map(updateDto, slot);

await \_context.SaveChangesAsync();

return \_mapper.Map<TimetableSlotDto>(slot);

}

public async Task<bool> ValidateTimetableAsync(Guid timetableId)

{

var conflicts = await CheckTimetableConflictsAsync(timetableId);

return !conflicts.HasConflicts;

}

public async Task<TimetableConflictDto> CheckTimetableConflictsAsync(Guid timetableId)

{

var conflicts = new List<ConflictDto>();

var slots = await \_context.TimetableSlots

.Where(ts => ts.TimetableId == timetableId)

.Include(ts => ts.Teacher)

.Include(ts => ts.Subject)

.ToListAsync();

// Check teacher double booking

var teacherConflicts = slots

.GroupBy(s => new { s.TeacherId, s.DayOfWeek, s.StartTime, s.EndTime })

.Where(g => g.Count() > 1)

.Select(g => new ConflictDto

{

Type = ConflictType.TeacherDoubleBooking,

Description = $"Teacher {g.Key.TeacherId} is double-booked on {g.Key.DayOfWeek} from {g.Key.StartTime} to {g.Key.EndTime}",

AffectedSlotIds = g.Select(s => s.Id).ToList(),

Severity = "High"

});

conflicts.AddRange(teacherConflicts);

// Check room conflicts

var roomConflicts = slots

.Where(s => !string.IsNullOrEmpty(s.Room))

.GroupBy(s => new { s.Room, s.DayOfWeek, s.StartTime, s.EndTime })

.Where(g => g.Count() > 1)

.Select(g => new ConflictDto

{

Type = ConflictType.RoomDoubleBooking,

Description = $"Room {g.Key.Room} is double-booked on {g.Key.DayOfWeek} from {g.Key.StartTime} to {g.Key.EndTime}",

AffectedSlotIds = g.Select(s => s.Id).ToList(),

Severity = "Medium"

});

conflicts.AddRange(roomConflicts);

// Check consecutive period violations

var consecutiveViolations = CheckConsecutivePeriodViolations(slots);

conflicts.AddRange(consecutiveViolations);

return new TimetableConflictDto

{

HasConflicts = conflicts.Any(),

Conflicts = conflicts

};

}

public async Task<byte[]> PrintTimetableAsync(Guid timetableId)

{

var timetable = await GetTimetableWithDetailsAsync(timetableId);

var timetableDto = \_mapper.Map<TimetableDto>(timetable);

return await \_pdfService.GenerateTimetablePdfAsync(timetableDto);

}

// Private helper methods

private async Task ValidateGenerationInputAsync(GenerateTimetableDto generateDto)

{

var school = await \_context.Schools.FindAsync(generateDto.SchoolId);

if (school == null)

throw new NotFoundException($"School {generateDto.SchoolId} not found");

var grade = await \_context.Grades.FindAsync(generateDto.GradeId);

if (grade == null)

throw new NotFoundException($"Grade {generateDto.GradeId} not found");

var term = await \_context.Terms.FindAsync(generateDto.TermId);

if (term == null)

throw new NotFoundException($"Term {generateDto.TermId} not found");

// Check if timetable already exists

var existingTimetable = await \_context.Timetables

.FirstOrDefaultAsync(t => t.GradeId == generateDto.GradeId &&

t.TermId == generateDto.TermId &&

t.IsActive);

if (existingTimetable != null)

{

throw new ConflictException($"Active timetable already exists for this grade and term");

}

}

private async Task<List<GradeSubject>> GetGradeSubjectsAsync(Guid gradeId, Guid schoolYearId)

{

return await \_context.GradeSubjects

.Where(gs => gs.GradeId == gradeId)

.Include(gs => gs.Subject)

.Include(gs => gs.Grade)

.ToListAsync();

}

private async Task<List<Teacher>> GetAvailableTeachersAsync(Guid schoolId, Guid schoolYearId)

{

return await \_context.Teachers

.Where(t => t.SchoolId == schoolId)

.Include(t => t.SubjectTeachers.Where(st => st.SchoolYearId == schoolYearId))

.ThenInclude(st => st.Subject)

.ToListAsync();

}

private async Task<List<TimetableSlot>> GenerateTimetableSlotsAsync(

Guid timetableId,

List<GradeSubject> gradeSubjects,

List<Teacher> teachers,

TimetableGenerationRulesDto rules)

{

var slots = new List<TimetableSlot>();

var random = new Random();

// Calculate periods per day

var totalMinutesPerDay = (int)(rules.SchoolEndTime - rules.SchoolStartTime).TotalMinutes;

var breakMinutes = (int)rules.BreakDuration.TotalMinutes;

var lunchMinutes = (int)rules.LunchDuration.TotalMinutes;

var periodMinutes = (int)rules.PeriodDuration.TotalMinutes;

var availableMinutes = totalMinutesPerDay - breakMinutes - lunchMinutes;

var periodsPerDay = Math.Min(availableMinutes / periodMinutes, rules.MaxPeriodsPerDay);

// Generate time slots for each day

var timeSlots = GenerateTimeSlots(rules, periodsPerDay);

foreach (var gradeSubject in gradeSubjects)

{

// Get assigned teacher for this subject

var assignedTeacher = teachers

.FirstOrDefault(t => t.SubjectTeachers

.Any(st => st.SubjectId == gradeSubject.SubjectId));

if (assignedTeacher == null)

{

\_logger.LogWarning("No teacher assigned to subject {SubjectId}", gradeSubject.SubjectId);

continue;

}

// Determine periods per week for this subject

var periodsPerWeek = rules.SubjectPeriodsPerWeek.GetValueOrDefault(gradeSubject.SubjectId, 3);

// Get preferred days and periods for this subject

var preferredDays = rules.SubjectPreferredDays.GetValueOrDefault(

gradeSubject.SubjectId,

rules.WorkingDays);

var preferredPeriods = rules.SubjectPreferredPeriods.GetValueOrDefault(

gradeSubject.SubjectId,

Enumerable.Range(1, periodsPerDay).ToList());

// Schedule periods for this subject

var scheduledPeriods = 0;

var attempts = 0;

const int maxAttempts = 100;

while (scheduledPeriods < periodsPerWeek && attempts < maxAttempts)

{

attempts++;

// Select random day and period from preferred options

var day = preferredDays[random.Next(preferredDays.Count)];

var period = preferredPeriods[random.Next(preferredPeriods.Count)];

var timeSlot = timeSlots[day][period - 1];

// Check if slot is available

if (IsSlotAvailable(slots, assignedTeacher.Id, day, timeSlot.StartTime, timeSlot.EndTime))

{

var room = rules.SubjectRoomPreferences.GetValueOrDefault(gradeSubject.SubjectId, "");

var slot = new TimetableSlot

{

TimetableId = timetableId,

SubjectId = gradeSubject.SubjectId,

TeacherId = assignedTeacher.Id,

DayOfWeek = day,

StartTime = timeSlot.StartTime,

EndTime = timeSlot.EndTime,

Room = room,

PeriodNumber = period

};

slots.Add(slot);

scheduledPeriods++;

}

}

if (scheduledPeriods < periodsPerWeek)

{

\_logger.LogWarning("Could only schedule {Scheduled}/{Required} periods for subject {SubjectId}",

scheduledPeriods, periodsPerWeek, gradeSubject.SubjectId);

}

}

return slots;

}

private Dictionary<DayOfWeek, List<TimeSlotDto>> GenerateTimeSlots(

TimetableGenerationRulesDto rules,

int periodsPerDay)

{

var timeSlots = new Dictionary<DayOfWeek, List<TimeSlotDto>>();

foreach (var day in rules.WorkingDays)

{

var daySlots = new List<TimeSlotDto>();

var currentTime = rules.SchoolStartTime;

for (int period = 1; period <= periodsPerDay; period++)

{

// Add break after certain periods

if (period == 3) // After 2nd period

{

currentTime = currentTime.Add(rules.BreakDuration);

}

else if (period == 6) // After 5th period (lunch)

{

currentTime = currentTime.Add(rules.LunchDuration);

}

var slot = new TimeSlotDto

{

DayOfWeek = day,

StartTime = currentTime,

EndTime = currentTime.Add(rules.PeriodDuration)

};

daySlots.Add(slot);

currentTime = slot.EndTime;

}

timeSlots[day] = daySlots;

}

return timeSlots;

}

private bool IsSlotAvailable(

List<TimetableSlot> existingSlots,

Guid teacherId,

DayOfWeek day,

TimeSpan startTime,

TimeSpan endTime)

{

return !existingSlots.Any(slot =>

slot.TeacherId == teacherId &&

slot.DayOfWeek == day &&

((startTime >= slot.StartTime && startTime < slot.EndTime) ||

(endTime > slot.StartTime && endTime <= slot.EndTime) ||

(startTime <= slot.StartTime && endTime >= slot.EndTime)));

}

private async Task<List<string>> ValidateSlotConflictsAsync(TimetableSlot slot)

{

var conflicts = new List<string>();

// Check teacher conflicts

var teacherConflict = await \_context.TimetableSlots

.AnyAsync(ts => ts.Id != slot.Id &&

ts.TeacherId == slot.TeacherId &&

ts.DayOfWeek == slot.DayOfWeek &&

((slot.StartTime >= ts.StartTime && slot.StartTime < ts.EndTime) ||

(slot.EndTime > ts.StartTime && slot.EndTime <= ts.EndTime) ||

(slot.StartTime <= ts.StartTime && slot.EndTime >= ts.EndTime)));

if (teacherConflict)

conflicts.Add("Teacher scheduling conflict");

// Check room conflicts

if (!string.IsNullOrEmpty(slot.Room))

{

var roomConflict = await \_context.TimetableSlots

.AnyAsync(ts => ts.Id != slot.Id &&

ts.Room == slot.Room &&

ts.DayOfWeek == slot.DayOfWeek &&

((slot.StartTime >= ts.StartTime && slot.StartTime < ts.EndTime) ||

(slot.EndTime > ts.StartTime && slot.EndTime <= ts.EndTime) ||

(slot.StartTime <= ts.StartTime && slot.EndTime >= ts.EndTime)));

if (roomConflict)

conflicts.Add("Room scheduling conflict");

}

return conflicts;

}

private List<ConflictDto> CheckConsecutivePeriodViolations(List<TimetableSlot> slots)

{

var violations = new List<ConflictDto>();

const int maxConsecutive = 3; // Maximum consecutive periods

var teacherSlots = slots

.GroupBy(s => s.TeacherId)

.ToDictionary(g => g.Key, g => g.ToList());

foreach (var teacherGroup in teacherSlots)

{

var teacherId = teacherGroup.Key;

var teacherPeriods = teacherGroup.Value

.GroupBy(s => s.DayOfWeek)

.ToDictionary(g => g.Key, g => g.OrderBy(s => s.PeriodNumber).ToList());

foreach (var dayGroup in teacherPeriods)

{

var dayPeriods = dayGroup.Value;

var consecutiveCount = 1;

for (int i = 1; i < dayPeriods.Count; i++)

{

if (dayPeriods[i].PeriodNumber == dayPeriods[i - 1].PeriodNumber + 1)

{

consecutiveCount++;

if (consecutiveCount > maxConsecutive)

{

violations.Add(new ConflictDto

{

Type = ConflictType.ConsecutivePeriodViolation,

Description = $"Teacher has {consecutiveCount} consecutive periods on {dayGroup.Key}",

AffectedSlotIds = dayPeriods.Skip(i - consecutiveCount + 1).Take(consecutiveCount).Select(s => s.Id).ToList(),

Severity = "Medium"

});

break;

}

}

else

{

consecutiveCount = 1;

}

}

}

}

return violations;

}

private async Task ResolveConflictsAsync(Guid timetableId, TimetableConflictDto conflicts)

{

// Implement basic conflict resolution logic

foreach (var conflict in conflicts.Conflicts.Where(c => c.Severity == "High"))

{

try

{

await AttemptConflictResolutionAsync(conflict);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Failed to resolve conflict: {ConflictDescription}", conflict.Description);

}

}

}

private async Task AttemptConflictResolutionAsync(ConflictDto conflict)

{

if (conflict.Type == ConflictType.TeacherDoubleBooking)

{

// Try to reschedule one of the conflicting slots

var conflictingSlots = await \_context.TimetableSlots

.Where(ts => conflict.AffectedSlotIds.Contains(ts.Id))

.ToListAsync();

if (conflictingSlots.Count >= 2)

{

var slotToReschedule = conflictingSlots.Last(); // Reschedule the last one

await TryRescheduleSlotAsync(slotToReschedule);

}

}

}

private async Task<bool> TryRescheduleSlotAsync(TimetableSlot slot)

{

var timetable = await \_context.Timetables.FindAsync(slot.TimetableId);

if (timetable == null) return false;

// Try to find an alternative time slot

var workingDays = new[] { DayOfWeek.Monday, DayOfWeek.Tuesday, DayOfWeek.Wednesday, DayOfWeek.Thursday, DayOfWeek.Friday };

var periods = Enumerable.Range(1, 8).ToList();

foreach (var day in workingDays)

{

foreach (var period in periods)

{

var startTime = new TimeSpan(8, 0, 0).Add(TimeSpan.FromMinutes((period - 1) \* 45)); // 45-minute periods

var endTime = startTime.Add(TimeSpan.FromMinutes(40));

var tempSlot = new TimetableSlot

{

Id = slot.Id,

TimetableId = slot.TimetableId,

TeacherId = slot.TeacherId,

SubjectId = slot.SubjectId,

DayOfWeek = day,

StartTime = startTime,

EndTime = endTime,

Room = slot.Room,

PeriodNumber = period

};

var conflicts = await ValidateSlotConflictsAsync(tempSlot);

if (!conflicts.Any())

{

// Update the slot

slot.DayOfWeek = day;

slot.StartTime = startTime;

slot.EndTime = endTime;

slot.PeriodNumber = period;

await \_context.SaveChangesAsync();

return true;

}

}

}

return false;

}

private async Task CreateStudentTimetableAssignmentsAsync(Guid timetableId, Guid gradeId)

{

var students = await \_context.Students

.Where(s => s.CurrentGradeId == gradeId)

.ToListAsync();

var timetableSlots = await \_context.TimetableSlots

.Where(ts => ts.TimetableId == timetableId)

.ToListAsync();

var studentTimetables = new List<StudentTimetable>();

foreach (var student in students)

{

// Get student's enrolled subjects

var studentSubjects = await \_context.StudentSubjects

.Where(ss => ss.StudentId == student.Id && ss.IsActive)

.Select(ss => ss.SubjectId)

.ToListAsync();

// Assign relevant timetable slots to student

var relevantSlots = timetableSlots

.Where(slot => studentSubjects.Contains(slot.SubjectId))

.ToList();

foreach (var slot in relevantSlots)

{

studentTimetables.Add(new StudentTimetable

{

StudentId = student.Id,

TimetableSlotId = slot.Id,

IsActive = true

});

}

}

\_context.StudentTimetables.AddRange(studentTimetables);

await \_context.SaveChangesAsync();

}

private async Task<Timetable> GetTimetableWithDetailsAsync(Guid timetableId)

{

return await \_context.Timetables

.Include(t => t.School)

.Include(t => t.Grade)

.Include(t => t.SchoolYear)

.Include(t => t.Term)

.Include(t => t.TimetableSlots)

.ThenInclude(ts => ts.Subject)

.Include(t => t.TimetableSlots)

.ThenInclude(ts => ts.Teacher)

.FirstOrDefaultAsync(t => t.Id == timetableId)

?? throw new NotFoundException($"Timetable {timetableId} not found");

}

}

} using SmartXul.Api.Services.Interfaces;

using SmartXul.Shared.DTOs;

using SmartXul.Shared.DTOs.Exams;

namespace SmartXul.Api.Services.Implementations

{

public class PdfGenerationService : IPdfGenerationService

{

public async Task<byte[]> GenerateReportCardPdfAsync(StudentReportCardDto reportCard)

{

// Implementation using iTextSharp, PdfSharp, or similar library

// This is a placeholder implementation

await Task.Delay(100); // Simulate processing time

return new byte[0]; // Return generated PDF bytes

}

public async Task<byte[]> GenerateClassResultsPdfAsync(ClassResultSummaryDto classSummary)

{

await Task.Delay(100);

return new byte[0];

}

public async Task<byte[]> GenerateTimetablePdfAsync(TimetableDto timetable)

{

await Task.Delay(100);

return new byte[0];

}

public async Task<byte[]> GenerateAttendanceReportPdfAsync(AttendanceReportDto attendanceReport)

{

await Task.Delay(100);

return new byte[0];

}

public async Task<byte[]> GenerateCertificatePdfAsync(StudentDto student, string certificateType)

{

await Task.Delay(100);

return new byte[0];

}

public async Task<byte[]> GenerateTranscriptPdfAsync(StudentDto student, List<ResultDto> results)

{

await Task.Delay(100);

return new byte[0];

}

public Task<byte[]> GenerateStudentExamResultAsync(ExamResultDto resultDto)

{

throw new **NotImplementedException**();

}

public Task<byte[]> GenerateExamAnalysisReportAsync(ExamStatisticsDto stats)

{

throw new **NotImplementedException**();

}

}

} using AutoMapper;

using Microsoft.EntityFrameworkCore;

using SmartXul.Api.Models;

using SmartXul.Api.Services.Interfaces;

using SmartXul.Shared.DTOs;

using SmartXul.Shared.Enums;

using SmartXul.Shared.Exceptions;

using SmartXul.Shared;

namespace SmartXul.Api.Services.Implementations

{

public class AttendanceService : IAttendanceService

{

private readonly SmartXulDbContext \_context;

private readonly IMapper \_mapper;

private readonly INotificationService \_notificationService;

public AttendanceService(

SmartXulDbContext context,

IMapper mapper,

INotificationService notificationService)

{

\_context = context;

\_mapper = mapper;

\_notificationService = notificationService;

}

public async Task<AttendanceDto> MarkAttendanceAsync(CreateAttendanceDto createAttendanceDto)

{

// Check if attendance already exists for this date

var existingAttendance = await \_context.Attendances

.FirstOrDefaultAsync(a => a.StudentId == createAttendanceDto.StudentId &&

a.SubjectId == createAttendanceDto.SubjectId &&

a.Date.Date == createAttendanceDto.Date.Date);

if (existingAttendance != null)

{

// Update existing attendance

existingAttendance.Status = createAttendanceDto.Status;

existingAttendance.Remarks = createAttendanceDto.Remarks;

await \_context.SaveChangesAsync();

return \_mapper.Map<AttendanceDto>(existingAttendance);

}

// Create new attendance record

var attendance = \_mapper.Map<Attendance>(createAttendanceDto);

\_context.Attendances.Add(attendance);

await \_context.SaveChangesAsync();

// Send notification to parents if absent

if (attendance.Status == AttendanceStatus.Absent)

{

await NotifyParentsAboutAbsenceAsync(attendance);

}

return \_mapper.Map<AttendanceDto>(attendance);

}

public async Task<IEnumerable<AttendanceDto>> GetAttendanceByClassAsync(Guid gradeId, DateTime date)

{

var attendances = await \_context.Attendances

.Where(a => a.Student.CurrentGradeId == gradeId && a.Date.Date == date.Date)

.Include(a => a.Student)

.Include(a => a.Subject)

.Include(a => a.Teacher)

.Include(a => a.Term)

.ToListAsync();

return \_mapper.Map<IEnumerable<AttendanceDto>>(attendances);

}

public async Task<IEnumerable<AttendanceDto>> GetStudentAttendanceAsync(Guid studentId, Guid termId)

{

var attendances = await \_context.Attendances

.Where(a => a.StudentId == studentId && a.TermId == termId)

.Include(a => a.Student)

.Include(a => a.Subject)

.Include(a => a.Teacher)

.Include(a => a.Term)

.OrderBy(a => a.Date)

.ToListAsync();

return \_mapper.Map<IEnumerable<AttendanceDto>>(attendances);

}

public async Task<AttendanceReportDto> GetAttendanceReportAsync(Guid studentId, Guid termId)

{

var student = await \_context.Students

.Include(s => s.CurrentGrade)

.Include(s => s.School)

.FirstOrDefaultAsync(s => s.Id == studentId);

if (student == null)

throw new NotFoundException($"Student {studentId} not found");

var term = await \_context.Terms.FindAsync(termId);

if (term == null)

throw new NotFoundException($"Term {termId} not found");

var attendances = await GetStudentAttendanceAsync(studentId, termId);

var subjectAttendance = attendances

.GroupBy(a => new { a.Subject.Id, a.Subject.Name })

.Select(g => new AttendanceSummaryDto

{

SubjectId = g.Key.Id,

SubjectName = g.Key.Name,

TotalClasses = g.Count(),

PresentClasses = g.Count(a => a.Status == AttendanceStatus.Present),

AbsentClasses = g.Count(a => a.Status == AttendanceStatus.Absent),

LateClasses = g.Count(a => a.Status == AttendanceStatus.Late),

AttendancePercentage = g.Count() > 0 ?

(decimal)g.Count(a => a.Status == AttendanceStatus.Present) / g.Count() \* 100 : 0

})

.ToList();

var totalClasses = subjectAttendance.Sum(s => s.TotalClasses);

var totalPresent = subjectAttendance.Sum(s => s.PresentClasses);

return new AttendanceReportDto

{

Student = \_mapper.Map<StudentDto>(student),

Term = \_mapper.Map<TermDto>(term),

SubjectAttendance = subjectAttendance,

TotalClassesTaken = totalClasses,

TotalClassesAttended = totalPresent,

OverallAttendancePercentage = totalClasses > 0 ? (decimal)totalPresent / totalClasses \* 100 : 0

};

}

public async Task<bool> BulkMarkAttendanceAsync(List<CreateAttendanceDto> attendanceList)

{

var attendances = new List<Attendance>();

var notifications = new List<CreateNotificationDto>();

foreach (var createDto in attendanceList)

{

// Check for existing attendance

var existing = await \_context.Attendances

.FirstOrDefaultAsync(a => a.StudentId == createDto.StudentId &&

a.SubjectId == createDto.SubjectId &&

a.Date.Date == createDto.Date.Date);

if (existing != null)

{

existing.Status = createDto.Status;

existing.Remarks = createDto.Remarks;

}

else

{

var attendance = \_mapper.Map<Attendance>(createDto);

attendances.Add(attendance);

// Queue absence notifications

if (attendance.Status == AttendanceStatus.Absent)

{

var student = await \_context.Students

.Include(s => s.Parents)

.FirstOrDefaultAsync(s => s.Id == createDto.StudentId);

if (student?.Parents.Any() == true)

{

foreach (var parent in student.Parents)

{

notifications.Add(new CreateNotificationDto

{

ParentId = parent.Id,

StudentId = student.Id,

Title = "Student Absence",

Message = $"{student.FirstName} {student.LastName} was absent on {createDto.Date:dd/MM/yyyy}",

Type = NotificationType.Attendance,

Channel = NotificationChannel.WhatsApp

});

}

}

}

}

}

if (attendances.Any())

\_context.Attendances.AddRange(attendances);

await \_context.SaveChangesAsync();

// Send notifications

foreach (var notification in notifications)

{

try

{

await \_notificationService.CreateNotificationAsync(notification);

}

catch (Exception ex)

{

// Log but don't fail the attendance marking

// \_logger.LogError(ex, "Failed to send attendance notification");

}

}

return true;

}

public async Task<AttendanceDto> UpdateAttendanceAsync(Guid attendanceId, UpdateAttendanceDto updateDto)

{

var attendance = await \_context.Attendances.FindAsync(attendanceId);

if (attendance == null)

throw new NotFoundException($"Attendance record {attendanceId} not found");

\_mapper.Map(updateDto, attendance);

await \_context.SaveChangesAsync();

return \_mapper.Map<AttendanceDto>(attendance);

}

private async Task NotifyParentsAboutAbsenceAsync(Attendance attendance)

{

var student = await \_context.Students

.Include(s => s.Parents)

.FirstOrDefaultAsync(s => s.Id == attendance.StudentId);

if (student?.Parents.Any() == true)

{

foreach (var parent in student.Parents)

{

var notification = new CreateNotificationDto

{

ParentId = parent.Id,

StudentId = student.Id,

Title = "Student Absence Alert",

Message = $"Dear {parent.FirstName}, {student.FirstName} {student.LastName} was absent from school on {attendance.Date:dd/MM/yyyy}. Please contact the school if this absence was not planned.",

Type = NotificationType.Attendance,

Channel = NotificationChannel.WhatsApp

};

await \_notificationService.CreateNotificationAsync(notification);

}

}

}

}

}