# SchedulerPlatform.Core

## Business Overview

The Core project is the heart of the SchedulerPlatform, containing the fundamental business entities and rules that define what the system can do. Think of it as the “dictionary” of the application - it defines what a Schedule is, what a Job Execution means, and what types of jobs we can run.

**What It Provides:** - **Domain Entities**: Definitions of all business objects (schedules, clients, job executions, etc.) - **Business Rules**: Validation logic and entity behaviors - **Contracts (Interfaces)**: Defines what operations the system needs without implementing them - **Enumerations**: Standard lists of values (job types, statuses, frequencies)

**Why It Matters:** This project has zero dependencies on external frameworks or libraries - it’s pure business logic. This makes it easy to understand, test, and maintain without worrying about database connections, web APIs, or UI frameworks. Any changes to core business concepts start here.

## Key Components

### Domain Entities

All entities inherit from BaseEntity which provides common tracking fields:

#### BaseEntity (Abstract Base Class)

* Id (int): Unique identifier
* CreatedAt (DateTime): When the record was created
* UpdatedAt (DateTime?): When the record was last modified
* CreatedBy (string?): Username who created the record
* UpdatedBy (string?): Username who last updated the record
* IsDeleted (bool): Soft delete flag

#### Schedule

**Purpose**: Represents a scheduled job with its configuration and timing.

**Properties:** - Name (string): Display name for the schedule - Description (string?): Detailed description of what this schedule does - ClientId (int): Which client this schedule belongs to - JobType (JobType enum): Type of job (Process=1, StoredProcedure=2, ApiCall=3) - Frequency (ScheduleFrequency enum): How often it runs (Manual, Daily, Weekly, Monthly, etc.) - CronExpression (string): Quartz CRON expression defining when to run - NextRunTime (DateTime?): When the job will execute next - LastRunTime (DateTime?): When the job last executed - IsEnabled (bool): Whether the schedule is active - MaxRetries (int): How many times to retry on failure - RetryDelayMinutes (int): Minutes to wait between retries - TimeZone (string): Timezone for CRON expression evaluation - TimeoutMinutes (int?): Maximum execution time in minutes before job is terminated - JobConfiguration (string): JSON configuration specific to the job type

**Navigation Properties:** - Client: The client who owns this schedule - JobExecutions: History of all executions for this schedule - JobParameters: Dynamic parameters for job execution - NotificationSetting: Email notification configuration

#### Client

**Purpose**: Represents a tenant/customer in the multi-tenant system.

**Properties:** - ClientName (string): Full name of the client organization - ClientCode (string): Unique code identifier - IsActive (bool): Whether the client account is active - ContactEmail (string?): Primary contact email - ContactPhone (string?): Primary contact phone

**Navigation Properties:** - Users: Users belonging to this client - Schedules: All schedules owned by this client - VendorCredentials: Third-party API credentials for this client

#### JobExecution

**Purpose**: Records the result of a single job execution.

**Properties:** - ScheduleId (int): Which schedule this execution belongs to - StartTime (DateTime): When execution started - EndTime (DateTime?): When execution completed - Status (JobStatus enum): Current status (Scheduled, Running, Completed, Failed, etc.) - Output (string?): Success output or result data - ErrorMessage (string?): Error message if failed - StackTrace (string?): Full stack trace for debugging failures - RetryCount (int): How many times this execution has been retried - DurationSeconds (int): How long the execution took - TriggeredBy (string): Who/what triggered this execution (Scheduler, Manual, Retry) - CancelledBy (string?): Username who manually cancelled this execution (if cancelled)

**Navigation Properties:** - Schedule: The schedule this execution belongs to

#### JobParameter

**Purpose**: Defines dynamic parameters for job execution.

**Properties:** - ScheduleId (int): Which schedule this parameter belongs to - ParameterName (string): Name of the parameter (e.g., “AccountId”) - ParameterType (string): Data type (int, string, datetime, etc.) - ParameterValue (string?): Static value or default value - SourceQuery (string?): SQL query to get dynamic value at runtime - SourceConnectionString (string?): Database connection for dynamic query - IsDynamic (bool): Whether to execute SourceQuery at runtime - DisplayOrder (int): Order to display in UI

**Navigation Properties:** - Schedule: The schedule this parameter belongs to

#### NotificationSetting

**Purpose**: Email notification configuration for job execution results.

**Properties:** - ScheduleId (int): Which schedule these settings apply to - EnableSuccessNotifications (bool): Send email on successful execution - EnableFailureNotifications (bool): Send email on failed execution (default: true) - SuccessEmailRecipients (string?): Comma-separated email addresses for success - FailureEmailRecipients (string?): Comma-separated email addresses for failure - SuccessEmailSubject (string?): Custom subject line for success emails - FailureEmailSubject (string?): Custom subject line for failure emails - IncludeExecutionDetails (bool): Include execution time, duration in email - IncludeOutput (bool): Include job output in email body

**Navigation Properties:** - Schedule: The schedule these settings belong to (one-to-one)

#### User

**Purpose**: Represents a user account in the system.

**Properties:** - Username (string): Login username - Email (string): User’s email address (unique) - FirstName (string): First name - LastName (string): Last name - ClientId (int): Which client this user belongs to - IsActive (bool): Whether the account is active - ExternalUserId (string?): ID from external identity provider (if using SSO)

**Navigation Properties:** - Client: The client organization this user belongs to - Permissions: List of permissions assigned to this user

#### UserPermission

**Purpose**: Defines what actions a user can perform.

**Properties:** - UserId (int): Which user this permission is for - PermissionName (string): Name of the permission (e.g., “ViewSchedules”, “EditSchedules”) - ResourceType (string?): Type of resource (e.g., “Schedule”, “Client”) - ResourceId (int?): Specific resource ID (for row-level security)

**Navigation Properties:** - User: The user this permission belongs to

#### VendorCredential

**Purpose**: Stores encrypted credentials for third-party API integrations.

**Properties:** - ClientId (int): Which client owns these credentials - VendorName (string): Name of the vendor/service - VendorUrl (string): API base URL - Username (string): API username or client ID - EncryptedPassword (string): Encrypted password or API key - LastVerified (DateTime?): When credentials were last validated - IsValid (bool): Whether credentials are currently valid - AdditionalData (string?): JSON for extra vendor-specific config

**Navigation Properties:** - Client: The client who owns these credentials

#### ScheduleSyncSource

**Purpose**: Stores external schedule source data for synchronization with third-party systems.

**Properties:** - ClientId (int): Which client owns this sync source - Vendor (string): Name of the external vendor/system - AccountNumber (string): Account or identifier in the external system - ScheduleFrequency (int): How often the schedule should run (maps to ScheduleFrequency enum) - ScheduleDate (DateTime): Specific date/time for the scheduled execution - CreatedAt (DateTime): When this sync source was created - UpdatedAt (DateTime?): When this sync source was last updated - CreatedBy (string?): Username who created this record - UpdatedBy (string?): Username who last updated this record - IsDeleted (bool): Soft delete flag

**Navigation Properties:** - Client: The client who owns this sync source

**Use Case:** - Import schedules from external billing systems, vendor portals, or data warehouses - Bulk schedule creation from external date/time lists - Maintain mapping between internal schedules and external system identifiers

#### AuditLog

**Purpose**: Tracks all changes to entities for compliance and debugging.

**Properties:** - EventType (string): Type of event (e.g., “EntityChanged”, “LoginAttempt”) - EntityType (string): Type of entity changed (e.g., “Schedule”) - EntityId (int?): ID of the entity changed - Action (string): Action performed (Create, Update, Delete) - OldValues (string?): JSON snapshot of entity before change - NewValues (string?): JSON snapshot of entity after change - UserName (string): Username who made the change - ClientId (int?): Client context for the change - IpAddress (string?): IP address of the user - UserAgent (string?): Browser/client user agent - Timestamp (DateTime): When the event occurred - AdditionalData (string?): Extra context data in JSON

### Enumerations

#### JobType

Defines the types of jobs that can be scheduled: - Process = 1: Execute a Windows process/executable - StoredProcedure = 2: Execute a SQL Server stored procedure - ApiCall = 3: Call an external REST API

#### JobStatus

Tracks the lifecycle of a job execution: - Scheduled = 1: Job is scheduled but not started yet - Running = 2: Job is currently executing - Completed = 3: Job finished successfully - Failed = 4: Job failed with an error - Retrying = 5: Job is being retried after failure - Cancelled = 6: Job was cancelled by user

#### ScheduleFrequency

Defines common scheduling patterns (used for UI display): - Manual = 0: Only runs when manually triggered - Daily = 1: Runs every day - Weekly = 2: Runs every week - Monthly = 3: Runs every month - Quarterly = 4: Runs every quarter - Annually = 5: Runs every year - Custom = 6: Uses custom CRON expression

**Note**: The actual scheduling logic uses CRON expressions; this enum helps users select common patterns in the UI.

### Interfaces

#### IRepository

Generic repository interface for data access operations: - GetByIdAsync(int id): Retrieve entity by ID - GetAllAsync(): Retrieve all entities - FindAsync(Expression<Func<T, bool>> predicate): Query entities - AddAsync(T entity): Add new entity - UpdateAsync(T entity): Update existing entity - DeleteAsync(T entity): Delete entity - CountAsync(Expression<Func<T, bool>>? predicate): Count entities matching criteria

#### IScheduleRepository : IRepository

Extended repository for schedule-specific operations: - GetByClientIdAsync(int clientId): Get all schedules for a client - GetEnabledSchedulesAsync(): Get all active schedules - GetSchedulesDueForExecutionAsync(DateTime currentTime): Get schedules ready to run - UpdateNextRunTimeAsync(int scheduleId, DateTime nextRunTime): Update next execution time - GetByIdWithNotificationSettingsAsync(int id): Get schedule with notification config - GetPagedAsync(...): Get paginated list with filtering

#### IJobExecutionRepository : IRepository

Extended repository for execution tracking: - GetByScheduleIdAsync(int scheduleId): Get execution history for a schedule - GetByStatusAsync(JobStatus status): Get all executions with specific status - GetFailedExecutionsAsync(int scheduleId): Get failed executions for retry analysis - GetLastExecutionAsync(int scheduleId): Get most recent execution

#### IEmailService

Interface for sending email notifications: - SendJobExecutionNotificationAsync(int jobExecutionId, bool isSuccess): Send notification after job completes

### Services

#### CronExpressionGenerator

**Purpose**: Utility service for generating Quartz CRON expressions from dates/times.

**Methods:** - GenerateCronExpression(DateTime dateTime, bool includeYear = true): Generates a Quartz CRON expression for a specific date/time - Format: "0 {minute} {hour} {day} {month} ? [{year}]" - Example: "0 30 14 25 10 ? 2025" for October 25, 2025 at 2:30 PM - GenerateCronDescription(DateTime dateTime): Generates a human-readable description of the CRON schedule - Example: “Runs once on October 25, 2025 at 2:30 PM”

**Use Cases:** - Bulk schedule creation from external date/time lists - CRON expression preview in UI before schedule creation - Integration with external scheduling systems

#### IUnitOfWork

Manages transactions and provides access to all repositories: - Schedules: IScheduleRepository property - JobExecutions: IJobExecutionRepository property - Clients: IRepository property - Users: IRepository property - UserPermissions: IRepository property - VendorCredentials: IRepository property - JobParameters: IRepository property - NotificationSettings: IRepository property - SaveChangesAsync(): Commit all changes in transaction - BeginTransactionAsync(): Start explicit transaction - CommitTransactionAsync(): Commit explicit transaction - RollbackTransactionAsync(): Rollback explicit transaction

## For Developers

### Architecture Patterns

**Domain-Driven Design (DDD)**: - Pure domain layer with no infrastructure concerns - Rich domain entities with behavior (not anemic data models) - Clear separation between domain logic and infrastructure

**Repository Pattern**: - Abstracts data access behind interfaces - Allows swapping data sources without changing business logic - Supports unit testing with mock repositories

**Unit of Work Pattern**: - Manages transactions across multiple repository operations - Ensures atomic commits (all-or-nothing) - Centralizes SaveChanges logic

**Dependency Inversion**: - Core defines interfaces, Infrastructure implements them - No dependencies on external libraries or frameworks - Enables testability and flexibility

### Entity Relationships

erDiagram  
 BaseEntity {  
 int Id PK  
 DateTime CreatedAt  
 DateTime UpdatedAt  
 string CreatedBy  
 string UpdatedBy  
 bool IsDeleted  
 }  
   
 Client ||--o{ User : "has many"  
 Client ||--o{ Schedule : "owns"  
 Client ||--o{ VendorCredential : "has many"  
   
 Client {  
 int Id PK  
 string ClientName  
 string ClientCode UK  
 bool IsActive  
 string ContactEmail  
 string ContactPhone  
 }  
   
 User ||--o{ UserPermission : "has many"  
 User }o--|| Client : "belongs to"  
   
 User {  
 int Id PK  
 string Username  
 string Email UK  
 string FirstName  
 string LastName  
 int ClientId FK  
 bool IsActive  
 string ExternalUserId  
 }  
   
 UserPermission }o--|| User : "belongs to"  
   
 UserPermission {  
 int Id PK  
 int UserId FK  
 string PermissionName  
 string ResourceType  
 int ResourceId  
 }  
   
 Schedule }o--|| Client : "owned by"  
 Schedule ||--o{ JobExecution : "has many"  
 Schedule ||--o{ JobParameter : "has many"  
 Schedule ||--o| NotificationSetting : "has one"  
   
 Schedule {  
 int Id PK  
 string Name  
 string Description  
 int ClientId FK  
 int JobType  
 int Frequency  
 string CronExpression  
 DateTime NextRunTime  
 DateTime LastRunTime  
 bool IsEnabled  
 int MaxRetries  
 int RetryDelayMinutes  
 string TimeZone  
 string JobConfiguration  
 }  
   
 JobExecution }o--|| Schedule : "belongs to"  
   
 JobExecution {  
 int Id PK  
 int ScheduleId FK  
 DateTime StartTime  
 DateTime EndTime  
 int Status  
 string Output  
 string ErrorMessage  
 string StackTrace  
 int RetryCount  
 int DurationSeconds  
 string TriggeredBy  
 }  
   
 JobParameter }o--|| Schedule : "belongs to"  
   
 JobParameter {  
 int Id PK  
 int ScheduleId FK  
 string ParameterName  
 string ParameterType  
 string ParameterValue  
 string SourceQuery  
 string SourceConnectionString  
 bool IsDynamic  
 int DisplayOrder  
 }  
   
 NotificationSetting ||--|| Schedule : "belongs to"  
   
 NotificationSetting {  
 int Id PK  
 int ScheduleId FK  
 bool EnableSuccessNotifications  
 bool EnableFailureNotifications  
 string SuccessEmailRecipients  
 string FailureEmailRecipients  
 string SuccessEmailSubject  
 string FailureEmailSubject  
 bool IncludeExecutionDetails  
 bool IncludeOutput  
 }  
   
 VendorCredential }o--|| Client : "belongs to"  
   
 VendorCredential {  
 int Id PK  
 int ClientId FK  
 string VendorName  
 string VendorUrl  
 string Username  
 string EncryptedPassword  
 DateTime LastVerified  
 bool IsValid  
 string AdditionalData  
 }  
   
 AuditLog {  
 int Id PK  
 string EventType  
 string EntityType  
 int EntityId  
 string Action  
 string OldValues  
 string NewValues  
 string UserName  
 int ClientId  
 string IpAddress  
 string UserAgent  
 DateTime Timestamp  
 string AdditionalData  
 }

### UML Class Diagrams

#### Core Domain Entities

classDiagram  
 class BaseEntity {  
 <<abstract>>  
 +int Id  
 +DateTime CreatedAt  
 +DateTime? UpdatedAt  
 +string? CreatedBy  
 +string? UpdatedBy  
 +bool IsDeleted  
 }  
   
 class Schedule {  
 +string Name  
 +string? Description  
 +int ClientId  
 +JobType JobType  
 +ScheduleFrequency Frequency  
 +string CronExpression  
 +DateTime? NextRunTime  
 +DateTime? LastRunTime  
 +bool IsEnabled  
 +int MaxRetries  
 +int RetryDelayMinutes  
 +string TimeZone  
 +string JobConfiguration  
 +Client Client  
 +ICollection~JobExecution~ JobExecutions  
 +ICollection~JobParameter~ JobParameters  
 +NotificationSetting? NotificationSetting  
 }  
   
 class Client {  
 +string ClientName  
 +string ClientCode  
 +bool IsActive  
 +string? ContactEmail  
 +string? ContactPhone  
 +ICollection~User~ Users  
 +ICollection~Schedule~ Schedules  
 +ICollection~VendorCredential~ VendorCredentials  
 }  
   
 class JobExecution {  
 +int ScheduleId  
 +DateTime StartTime  
 +DateTime? EndTime  
 +JobStatus Status  
 +string? Output  
 +string? ErrorMessage  
 +string? StackTrace  
 +int RetryCount  
 +int DurationSeconds  
 +string TriggeredBy  
 +Schedule Schedule  
 }  
   
 class JobParameter {  
 +int ScheduleId  
 +string ParameterName  
 +string ParameterType  
 +string? ParameterValue  
 +string? SourceQuery  
 +string? SourceConnectionString  
 +bool IsDynamic  
 +int DisplayOrder  
 +Schedule Schedule  
 }  
   
 class NotificationSetting {  
 +int ScheduleId  
 +bool EnableSuccessNotifications  
 +bool EnableFailureNotifications  
 +string? SuccessEmailRecipients  
 +string? FailureEmailRecipients  
 +string? SuccessEmailSubject  
 +string? FailureEmailSubject  
 +bool IncludeExecutionDetails  
 +bool IncludeOutput  
 +Schedule? Schedule  
 }  
   
 class User {  
 +string Username  
 +string Email  
 +string FirstName  
 +string LastName  
 +int ClientId  
 +bool IsActive  
 +string? ExternalUserId  
 +Client Client  
 +ICollection~UserPermission~ Permissions  
 }  
   
 class UserPermission {  
 +int UserId  
 +string PermissionName  
 +string? ResourceType  
 +int? ResourceId  
 +User User  
 }  
   
 class VendorCredential {  
 +int ClientId  
 +string VendorName  
 +string VendorUrl  
 +string Username  
 +string EncryptedPassword  
 +DateTime? LastVerified  
 +bool IsValid  
 +string? AdditionalData  
 +Client Client  
 }  
   
 class AuditLog {  
 +string EventType  
 +string EntityType  
 +int? EntityId  
 +string Action  
 +string? OldValues  
 +string? NewValues  
 +string UserName  
 +int? ClientId  
 +string? IpAddress  
 +string? UserAgent  
 +DateTime Timestamp  
 +string? AdditionalData  
 }  
   
 BaseEntity <|-- Schedule  
 BaseEntity <|-- Client  
 BaseEntity <|-- JobExecution  
 BaseEntity <|-- JobParameter  
 BaseEntity <|-- NotificationSetting  
 BaseEntity <|-- User  
 BaseEntity <|-- UserPermission  
 BaseEntity <|-- VendorCredential  
 BaseEntity <|-- AuditLog  
   
 Client "1" --> "\*" Schedule : owns  
 Client "1" --> "\*" User : has  
 Client "1" --> "\*" VendorCredential : has  
 Schedule "1" --> "\*" JobExecution : has  
 Schedule "1" --> "\*" JobParameter : has  
 Schedule "1" --> "0..1" NotificationSetting : has  
 User "1" --> "\*" UserPermission : has

#### Domain Enums

classDiagram  
 class JobType {  
 <<enumeration>>  
 Process = 1  
 StoredProcedure = 2  
 ApiCall = 3  
 }  
   
 class JobStatus {  
 <<enumeration>>  
 Scheduled = 1  
 Running = 2  
 Completed = 3  
 Failed = 4  
 Retrying = 5  
 Cancelled = 6  
 }  
   
 class ScheduleFrequency {  
 <<enumeration>>  
 Manual = 0  
 Daily = 1  
 Weekly = 2  
 Monthly = 3  
 Quarterly = 4  
 Annually = 5  
 Custom = 6  
 }  
   
 Schedule --> JobType : uses  
 Schedule --> ScheduleFrequency : uses  
 JobExecution --> JobStatus : uses

#### Repository Interfaces

classDiagram  
 class IRepository~T~ {  
 <<interface>>  
 +GetByIdAsync(int id) Task~T?~  
 +GetAllAsync() Task~IEnumerable~T~~  
 +FindAsync(Expression predicate) Task~IEnumerable~T~~  
 +AddAsync(T entity) Task~T~  
 +UpdateAsync(T entity) Task  
 +DeleteAsync(T entity) Task  
 +CountAsync(Expression predicate) Task~int~  
 }  
   
 class IScheduleRepository {  
 <<interface>>  
 +GetByClientIdAsync(int clientId) Task~IEnumerable~Schedule~~  
 +GetEnabledSchedulesAsync() Task~IEnumerable~Schedule~~  
 +GetSchedulesDueForExecutionAsync(DateTime currentTime) Task~IEnumerable~Schedule~~  
 +UpdateNextRunTimeAsync(int scheduleId, DateTime nextRunTime) Task  
 +GetByIdWithNotificationSettingsAsync(int id) Task~Schedule?~  
 +GetPagedAsync(int pageNumber, int pageSize, int? clientId, string? searchTerm) Task  
 }  
   
 class IJobExecutionRepository {  
 <<interface>>  
 +GetByScheduleIdAsync(int scheduleId) Task~IEnumerable~JobExecution~~  
 +GetByStatusAsync(JobStatus status) Task~IEnumerable~JobExecution~~  
 +GetFailedExecutionsAsync(int scheduleId) Task~IEnumerable~JobExecution~~  
 +GetLastExecutionAsync(int scheduleId) Task~JobExecution?~  
 }  
   
 class IEmailService {  
 <<interface>>  
 +SendJobExecutionNotificationAsync(int jobExecutionId, bool isSuccess) Task  
 }  
   
 class IUnitOfWork {  
 <<interface>>  
 +IScheduleRepository Schedules  
 +IJobExecutionRepository JobExecutions  
 +IRepository~Client~ Clients  
 +IRepository~User~ Users  
 +IRepository~UserPermission~ UserPermissions  
 +IRepository~VendorCredential~ VendorCredentials  
 +IRepository~JobParameter~ JobParameters  
 +IRepository~NotificationSetting~ NotificationSettings  
 +SaveChangesAsync() Task~int~  
 +BeginTransactionAsync() Task  
 +CommitTransactionAsync() Task  
 +RollbackTransactionAsync() Task  
 +Dispose() void  
 }  
   
 IRepository~T~ <|.. IScheduleRepository : extends  
 IRepository~T~ <|.. IJobExecutionRepository : extends  
 IUnitOfWork --> IScheduleRepository : provides  
 IUnitOfWork --> IJobExecutionRepository : provides  
 IUnitOfWork --> IRepository~T~ : provides

### Design Decisions

**Why No Business Logic in Entities?** - Current implementation uses anemic domain model (data-only entities) - Business logic is in service layer (Jobs, API controllers) - Could be refactored to rich domain model with entity methods - Trade-off: Simplicity vs. encapsulation

**Why Soft Deletes (IsDeleted)?** - Audit trail: Never lose historical data - Regulatory compliance: Maintain records for legal requirements - Data recovery: Easily restore accidentally deleted items - Queries must filter on IsDeleted = false

**Why String JSON for JobConfiguration?** - Flexibility: Each job type has different configuration needs - No schema changes: Add new config fields without migrations - Easy to version: Store different config versions side-by-side - Trade-off: Type safety vs. flexibility

**Why CRON Expressions Instead of Simple Schedules?** - Power: CRON expressions handle complex schedules (e.g., “2nd Tuesday of each month”) - Industry standard: Quartz.NET uses CRON, familiar to ops teams - Flexibility: Single format for all schedule types - UI provides helpers: CronBuilder component makes it user-friendly

## Dependencies

**None!** This is a pure domain layer with zero external dependencies.

**Framework**: .NET 8.0 (System.Text.Json for serialization attributes)

## Integration

**Referenced By:** - SchedulerPlatform.Infrastructure: Implements repositories and data access - SchedulerPlatform.Jobs: Uses entities for job execution - SchedulerPlatform.API: Uses entities and interfaces for controllers - SchedulerPlatform.IdentityServer: Uses User and Client entities

**References:** None (pure domain layer)

## Known Issues

### Design Issues

1. **Anemic Domain Model**
   * **Issue**: Entities are just data containers with no behavior
   * **Impact**: Business logic scattered across services instead of encapsulated in entities
   * **Recommendation**: Consider refactoring to rich domain model with entity methods (e.g., Schedule.CalculateNextRunTime())
2. **JobConfiguration as String JSON**
   * **Issue**: No compile-time type safety for job configurations
   * **Impact**: Runtime errors if JSON structure is wrong
   * **Mitigation**: Strict validation in job execution, schema documentation
   * **Alternative**: Consider polymorphic JobConfiguration base class with typed subclasses
3. **No Value Objects**
   * **Issue**: Primitives used for complex concepts (e.g., CronExpression, Email)
   * **Impact**: No validation at domain level, easy to pass invalid values
   * **Recommendation**: Create value objects: CronExpression, EmailAddress, TimeZone

### Missing Features

1. **No Audit Trail in Entities**
   * **Issue**: AuditLog is separate; entities don’t track their own history
   * **Impact**: Can’t easily see entity change history
   * **TODO**: Consider adding HistoryChanges collection to BaseEntity
2. **No Domain Events**
   * **Issue**: No way for entities to publish events (e.g., “ScheduleCreated”)
   * **Impact**: Hard to trigger side effects (send notifications, log events)
   * **Recommendation**: Implement domain events pattern
3. **Limited Validation**
   * **Issue**: Minimal validation logic in entity properties
   * **Impact**: Invalid data can be created
   * **TODO**: Add data annotation attributes or FluentValidation

### Multi-Tenancy

1. **Client Isolation Not Enforced**
   * **Issue**: No built-in query filters for ClientId
   * **Impact**: Developers must remember to filter by ClientId in every query
   * **Risk**: Potential data leaks between clients
   * **Recommendation**: Implement EF Core global query filters in Infrastructure layer
2. **No Client Context Awareness**
   * **Issue**: Entities don’t know current user’s ClientId
   * **Impact**: Must pass ClientId explicitly in every operation
   * **TODO**: Consider IClientContext service injected via constructor

### Security

1. **Passwords in VendorCredential**
   * **Issue**: Encryption handled outside Core layer
   * **Impact**: Core doesn’t enforce encryption
   * **TODO**: Document encryption requirements clearly
2. **No Permission Validation in Core**
   * **Issue**: UserPermission entities exist but no validation logic
   * **Impact**: Permission checks must be done in API/UI layers
   * **Recommendation**: Add User.HasPermission(string permissionName) method

### Performance

1. **Lazy Loading Not Configured**
   * **Issue**: All navigation properties must be explicitly included
   * **Impact**: Risk of N+1 query problems
   * **Mitigation**: Always use .Include() in repositories
   * **Alternative**: Enable lazy loading proxies (with caution)
2. **Large Collections**
   * **Issue**: Schedule.JobExecutions can grow to thousands of records
   * **Impact**: Loading a schedule with all executions is slow
   * **TODO**: Implement pagination in repository queries

### Testing

1. **No Unit Tests**
   * **Issue**: Core project has no test coverage
   * **Impact**: Risk of breaking changes
   * **TODO**: Add xUnit project with entity tests
2. **No Validation Tests**
   * **Issue**: Can’t verify business rules are enforced
   * **TODO**: Test invalid entity creation scenarios

## Best Practices for Using Core

1. **Never Reference Infrastructure**: Core should never depend on Infrastructure, API, or Jobs
2. **Use Interfaces**: Always program against IRepository, IUnitOfWork, never concrete implementations
3. **Keep It Pure**: No database concerns, no HTTP concerns, no UI concerns in Core
4. **Validate Early**: Add validation logic to entity constructors or factory methods
5. **Document Enums**: Add XML comments explaining when to use each enum value
6. **Test Entities**: Write unit tests for entity behavior and validation rules

## Future Improvements

1. Migrate to rich domain model with entity behaviors
2. Implement domain events for loose coupling
3. Add value objects for complex primitives
4. Implement specification pattern for complex queries
5. Add comprehensive validation using FluentValidation
6. Create factory methods for complex entity creation
7. Add integration events for cross-bounded-context communication