**Registry API - Application Context Dump**

Version: 1.0

Date: May 12, 2025

**1. Overview & Purpose**

* **Application Name:** Registry API
* **Purpose:** To provide a backend RESTful API for managing data specifications stored in the RegistryDatabase. It allows client applications to perform Create, Read, Update, and Delete (CRUD) operations on specification metadata (SpecificationIdentifyingInformation) and their associated core (SpecificationCore) and extension (SpecificationExtensionComponents) elements.
* **Primary Goal:** Enable users/systems to define a specification and populate it by selecting and potentially customizing elements from predefined models (CoreInvoiceModel, ExtensionComponentModelElements).

**2. Core Technologies**

* **Backend Framework:** ASP.NET Core on .NET 8 (expected forward compatibility with .NET 9)
* **Language:** C#
* **Database:** Microsoft SQL Server
* **Data Access:** Entity Framework Core (EF Core) 8 - Used as the Object-Relational Mapper (ORM).
* **Database Server:** [YourServerHere] (for development)
* **API Style:** RESTful HTTP API

**3. Database (**RegistryDatabase**)**

* **Source Script:** The schema was initially created using the CreateRegistryDatabaseAndTables2.sql script.
* **Schema Management:** EF Core Migrations have been introduced to manage subsequent schema changes after establishing a baseline against the existing script-created database.
* **Developer Documentation:** A detailed description of the tables, columns, and relationships is available in the RegistryDatabase Developer Report.docx document.
* **Key Tables:**
  + **Model Tables (Reference Data):**
    - CoreInvoiceModel: Stores standard "Core Invoice" elements.
    - ExtensionComponentsModelHeader: Defines available extension components.
    - ExtensionComponentModelElements: Stores detailed elements belonging to each extension component. Data likely populated from RegistryModelData.xls - \*.csv files.
  + **Specification Tables (User Data):**
    - SpecificationIdentifyingInformation: The main header table for a user-defined specification (PK: IdentityID).
    - SpecificationCore: Links a specification to selected CoreInvoiceModel elements, storing usage details (FKs: IdentityID, BusinessTermID).
    - SpecificationExtensionComponents: Links a specification to selected ExtensionComponentModelElements, storing usage details (FKs: IdentityID, composite key (ExtensionComponentID, BusinessTermID)).
* **Key Relationships & Constraints:**
  + SpecificationCore and SpecificationExtensionComponents have a foreign key back to SpecificationIdentifyingInformation. Cascade delete is configured in EF Core (deleting a header deletes its core/extension items).
  + SpecificationCore links to CoreInvoiceModel via BusinessTermID. Restricted delete.
  + SpecificationExtensionComponents links to ExtensionComponentModelElements via a composite foreign key (ExtensionComponentID, BusinessTermID), which targets a unique constraint on the corresponding columns in ExtensionComponentModelElements. Restricted delete.
* **Recent Schema Changes (via Migrations):**
  + SpecificationCore.UsageNote changed from nchar(10) to text.
  + SpecificationCore.TypeOfChange changed from text to nvarchar(50).

**4. Backend API Architecture & Implementation**

* **Architecture:** Layered Architecture
  + **Controllers (**/Controllers**):** Handle HTTP requests/responses, route to services. Use TypedResults (.NET 8+) for responses. (SpecificationsController.cs)
  + **Services (**/Services**):** Contain business logic, orchestrate repository calls, perform validation, map data (using AutoMapper). Return specific result enums (DeleteResult, ServiceResult) for clarity. (SpecificationService.cs, ISpecificationService.cs)
  + **Repositories (**/Repositories**):** Abstract data access using EF Core. Implement specific repository interfaces inheriting from a generic interface. (SpecificationIdentifyingInformationRepository.cs, ISpecificationIdentifyingInformationRepository.cs, etc.)
* **Key Design Patterns:**
  + **REST:** Standard HTTP verbs, status codes, resource-based URLs.
  + **Dependency Injection (DI):** Used throughout (configured in Program.cs) to inject dependencies (DbContext, Repositories, Services, Logger, AutoMapper) primarily via primary constructors (.NET 8+).
  + **Repository Pattern:** Isolates data access logic. Services depend on repository interfaces.
  + **Unit of Work:** Implicitly handled by EF Core's DbContext.SaveChangesAsync() within repository methods.
  + **Data Transfer Objects (DTOs):** Used for API request/response models to decouple internal entities from the external contract. Records (.NET 8+) are used for many DTOs. (/DTOs folder)
* **Framework Features Used:**
  + .NET 8 Minimal Hosting Model (Program.cs)
  + Primary Constructors (DbContext, Repositories, Services, some DTOs/Helpers)
  + required modifier (Models, some DTOs)
  + Collection Expressions ([]) (Model navigation properties)
  + TypedResults (Controller action results)
* **Data Access:**
  + EF Core 8 is used.
  + Models (/Models) represent database tables using attributes ([Table], [Key], [MaxLength], [Column], [ForeignKey]) and navigation properties.
  + DbContext (/Data/RegistryDbContext.cs) configures entity mappings, relationships (using Fluent API in OnModelCreating), and database connection.
* **Mapping:** AutoMapper is used for mapping between Entities (Models) and DTOs. A profile (/Mappings/SpecificationProfile.cs) defines these mappings.
* **API Functionality (Endpoints in** SpecificationsController**):**
  + SpecificationIdentifyingInformation **CRUD:**
    - GET /api/specifications: Retrieves a paginated list of specification headers.
    - GET /api/specifications/{id}: Retrieves details of a single specification, including paginated lists of its core and extension elements.
    - POST /api/specifications: Creates a new specification header.
    - PUT /api/specifications/{id}: Updates an existing specification header.
    - DELETE /api/specifications/{id}: Deletes a specification header **only if** it has no associated core or extension elements (returns 409 Conflict otherwise).
  + SpecificationCore **CRUD (relative to a specification):**
    - GET /api/specifications/{specId}/coreElements: Retrieves a paginated list of core elements for a specific specification.
    - POST /api/specifications/{specId}/coreElements: Adds a core element to a specification (validates parent spec and CoreInvoiceModel element exist).
    - PUT /api/specifications/{specId}/coreElements/{coreId}: Updates a specific core element within a specification.
    - DELETE /api/specifications/{specId}/coreElements/{coreId}: Removes a specific core element from a specification.
  + SpecificationExtensionComponents **CRUD (relative to a specification):**
    - GET /api/specifications/{specId}/extensionElements: Retrieves a paginated list of extension elements for a specific specification.
    - POST /api/specifications/{specId}/extensionElements: Adds an extension element to a specification (validates parent spec and ExtensionComponentModelElement exists via composite key).
    - PUT /api/specifications/{specId}/extensionElements/{extId}: Updates a specific extension element within a specification.
    - DELETE /api/specifications/{specId}/extensionElements/{extId}: Removes a specific extension element from a specification.
* **Pagination:** Implemented using helper classes (/Helpers/PaginationParams.cs, /Helpers/PagedList.cs) and applied to list endpoints (GET /api/specifications, GET /api/specifications/{id}/coreElements, GET /api/specifications/{id}/extensionElements). Pagination metadata is included in responses.
* **Error Handling:** Primarily uses standard HTTP status codes returned via TypedResults. Service layer uses result enums (DeleteResult, ServiceResult) to communicate outcomes (Success, NotFound, Conflict, RefNotFound, BadRequest) back to the controller. Standard ILogger assumed for logging errors.
* **Authentication/Authorization:** None specifically implemented yet. API is currently open.
* **Logging:** Uses standard ASP.NET Core ILogger. No additional third-party logging libraries configured yet.

**5. Setup, Configuration & Development Workflow**

* **Prerequisites:** .NET 8 SDK, IDE (VS 2022, VS Code, Rider), SQL Server instance.
* **Project Setup:** Standard dotnet new webapi -f net8.0 project.
* **NuGet Packages:** Microsoft.EntityFrameworkCore.SqlServer, Microsoft.EntityFrameworkCore.Design, Microsoft.EntityFrameworkCore.Tools (for PMC), AutoMapper.Extensions.Microsoft.DependencyInjection.
* **Configuration (**appsettings.json**):** Contains ConnectionStrings.
* **Secrets Management (Dev):** .NET Secret Manager is used to store the development database connection string securely (dotnet user-secrets set "ConnectionStrings:RegistryDatabaseConnection" "..."). The connection string is removed from appsettings.json.
* **Dependency Injection (**Program.cs**):** Configures DbContext, Repositories, Services, AutoMapper, Controllers, Swagger.
* **EF Core Migrations:**
  + Used for schema changes *after* the initial database creation.
  + A baseline migration (InitialCreateForExistingDB) was established against the existing database schema (created by the SQL script) using the "fake apply" / baseline migration strategy (manually inserting history record). This involved temporarily aligning models with the script schema, generating the baseline, manually marking it applied in \_\_EFMigrationsHistory, then re-applying desired model changes and generating incremental migrations.
  + Commands are run via Package Manager Console (Add-Migration, Update-Database, Remove-Migration) or dotnet ef CLI.
* **Source Control:** Git is used. A .gitignore file is configured to exclude bin, obj, .vs, user secrets, logs, etc. Previously tracked bin/obj folders were removed using git rm -r --cached <folder>.
* **Running the App:** dotnet run.
* **API Testing:** Swagger UI is configured (accessible at the root URL / in development) for viewing endpoints and testing requests.

**6. Source Files Provided**

* CreateRegistryDatabaseAndTables2.sql: Initial database schema script.
* RegistryDatabase Developer Report.docx: Documentation for the database schema.
* RegistryModelData.xls - CoreInvoiceModel.csv, RegistryModelData.xls - ExtensionComponentModelElements.csv, RegistryModelData.xls - ExtensionComponentsModelHeader.csv: Sample data for the model tables.

**7. Key Decisions & Troubleshooting History**

* **Delete Behavior:** Deleting a SpecificationIdentifyingInformation record is prevented (409 Conflict) if child SpecificationCore or SpecificationExtensionComponents records exist. Children must be deleted first.
* **Pagination:** Implemented for all list endpoints, including child lists within the detail view of a specification.
* **Swagger UI URL:** Configured to run at the root URL (/) in the development environment due to options.RoutePrefix = string.Empty; in Program.cs.
* **Migrations Baseline:** Addressed issues with applying initial migrations to a pre-existing database by using the "fake apply" / baseline migration strategy (manually inserting history record).
* **Secret Management:** Adopted .NET User Secrets for local development connection string storage.
* .gitignore Setup: Configured standard .NET gitignore and cleaned previously tracked build artifacts.
* API Error (POST /extensionElements): Troubleshooted 400 Bad Request ("Referenced Extension Component element not found"), traced to validation check (ExtensionElementExistsAsync) failing due to missing data combination in ExtensionComponentModelElements table.

This document should provide a solid foundation for another developer (or AI assistant) to understand the Registry API project.