InfoSymbolServer.Presentation

Overview

The Presentation layer exposes REST API that allow clients to interact with the system. This layer is responsible for:

- Exposing HTTP endpoints through controllers
- Converting HTTP requests to application DTOs
- Converting application responses to HTTP responses
- Handling exceptions and generating appropriate HTTP status codes
- Configuring middleware components for cross-cutting concerns
- Providing API documentation through Swagger

API Endpoints

Exchange Controller

Method	Endpoint	Description
GET	`/api/v1/exchanges`	Gets all exchanges
GET	`/api/v1/exchanges/supported`	Gets all supported exchanges names
GET	`/api/v1/exchanges/{name}`	Gets an exchange by its name
POST	`/api/v1/exchanges`	Creates a new exchange
DELETE	`/api/v1/exchanges/{name}`	Deletes an exchange

Symbol Controller

Method	Endpoint	Desc
GET	`/api/v1/exchanges/{exchangeName}/symbols`	Gets for ar (pagi
GET	`/api/v1/exchanges/{exchangeName}/symbols-list`	Gets for ar (no p
GET	`/api/v1/exchanges/{exchangeName}/active-symbols`	Gets symb excha (pagi
GET	`/api/v1/exchanges/{exchangeName}/active-symbols-list`	Gets symb excha pagin
GET	`/api/v1/exchanges/{exchangeName}/symbols/{symbolName}`	Gets symb
POST	`/api/v1/exchanges/{exchangeName}/symbols`	Adds symb an ex status Adde
POST	`/api/v1/exchanges/{exchangeName}/symbols/{symbolName}/revoke`	Revo symb status chan from Remo to Ad
DELETE	`/api/v1/exchanges/{exchangeName}/symbols/{symbolName}`	Mark delet (sets Remo

Status Controller

Method	Endpoint	Desc
GET	`/api/v1/exchanges/{exchangeName}/symbols/history`	Gets histo all sy in an

Method	Endpoint	Desc
		exch (pagi
GET	`/api/v1/exchanges/{exchangeName}/symbols-list/history`	Gets histo all sy in an exch (no pagir
GET	`/api/v1/exchanges/{exchangeName}/active-symbols/history`	Gets histo activ symb an exch (pagi
GET	`/api/v1/exchanges/{exchangeName}/active-symbols-list/history`	Gets histo activ symb an exch (no pagir
GET	`/api/v1/exchanges/{exchangeName}/symbols/{symbolName}/history`	Gets histo spec symb

Notification Settings Controller

Method	Endpoint	Description
GET	`/api/v1/notification-settings`	Gets current notification settings
PUT	`/api/v1/notification-settings`	Updates notification settings

Key Components

API Controllers

The Presentation layer implements RESTful API controllers that handle HTTP requests and responses. All controllers follow a consistent pattern and use dependency injection for their dependencies.

ExchangeController

The `ExchangeController` manages exchange-related operations:

```
[ApiController]
[Route("api/v1")]
[Produces("application/json")]
[Tags("Exchanges")]
public class ExchangeController : ControllerBase
{
    private readonly IExchangeService _exchangeService;
    private readonly IMapper _mapper;

    public ExchangeController(IExchangeService exchangeService, IMapper
mapper)
    {
        _exchangeService = exchangeService;
        _mapper = mapper;
    }

    [HttpGet("exchanges")]
    [ProducesResponseType(StatusCodes.Status200OK, Type =
typeof(IEnumerable<ExchangeDto>))]
    public async Task<ActionResult<IEnumerable<ExchangeDto>>> GetAll(
        CancellationToken cancellationToken = default)
    {
        var exchanges = await
_exchangeService.GetAllAsync(cancellationToken);
        return Ok(exchanges);
    }

    // Other endpoints for CRUD operations...
}
```

SymbolController

The `SymbolController` handles symbol-related operations:

```
[ApiController]
[Route("api/v1/exchanges")]
[Produces("application/json")]
[Tags("Symbols")]
public class SymbolController : ControllerBase
{
    private readonly ISymbolService _symbolService;
    private readonly IMapper _mapper;

    public SymbolController(ISymbolService symbolService, IMapper mapper)
```

```
    {
        _symbolService = symbolService;
        _mapper = mapper;
    }

    [HttpGet("{exchangeName}/symbols")]
    [ProducesResponseType(StatusCodes.Status200OK, Type =
typeof(IEnumerable<SymbolDto>))]
    public async Task<ActionResult<IEnumerable<SymbolDto>>> GetAll(
        [FromRoute] string exchangeName,
        [FromQuery] int? pageNumber = null,
        [FromQuery] int? pageSize = null,
        CancellationToken cancellationToken = default)
    {
        var symbols = await _symbolService.GetForExchangeAsync(
            exchangeName, pageNumber, pageSize, cancellationToken);
        return Ok(symbols);
    }

    // other endpoints for symols lookup

    [HttpPost("{exchangeName}/symbols")]
    [ProducesResponseType(StatusCodes.Status200OK, Type =
typeof(SymbolDto))]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<ActionResult<SymbolDto>> Add(
        [FromRoute] string exchangeName,
        [FromBody] AddSymbolRequest request,
        CancellationToken cancellationToken = default)
    {
        var dto = _mapper.Map<AddSymbolDto>(request);
        dto.ExchangeName = exchangeName;

        var result = await _symbolService.AddAsync(dto,
cancellationToken);
        return Ok(result);
    }

    [HttpDelete("{exchangeName}/symbols/{symbolName}")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<ActionResult> Delete(
        [FromRoute] string exchangeName,
        [FromRoute] string symbolName,
        CancellationToken cancellationToken = default)
    {
        await _symbolService.DeleteAsync(symbolName, exchangeName,
cancellationToken);
        return NoContent();
```

```
    }
}
```

StatusController

The `StatusController` manages symbol status history:

```
[ApiController]
[Route("api/v1/exchanges")]
[Produces("application/json")]
[Tags("Status History")]
public class StatusController : ControllerBase
{
    private readonly IStatusService _statusService;

    public StatusController(IStatusService statusService)
    {
        _statusService = statusService;
    }

    [HttpGet("{exchangeName}/symbols/history")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<ActionResult<IEnumerable<SymbolHistoryDto>>>
GetExchangeSymbolsHistory(
        [FromRoute] string exchangeName,
        [FromQuery] int? pageNumber = null,
        [FromQuery] int? pageSize = null,
        CancellationToken cancellationToken = default)
    {
        var history = await _statusService.GetExchangeSymbolsHistoryAsync(
            exchangeName, pageNumber, pageSize, cancellationToken);
        return Ok(history);
    }

    // Other endpoints for symbols history...
}
```

NotificationSettingsController

The `NotificationSettingsController` manages notification settings:

```
[ApiController]
[Route("api/v1/notification-settings")]
[Produces("application/json")]
[Tags("Notification Settings")]
public class NotificationSettingsController : ControllerBase
{
```

```
    private readonly INotificationSettingsService
_notificationSettingsService;
    private readonly IMapper _mapper;

    public NotificationSettingsController(INotificationSettingsService
notificationSettingsService, IMapper mapper)
    {
        _notificationSettingsService = notificationSettingsService;
        _mapper = mapper;
    }

    [HttpGet]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<ActionResult<NotificationSettingsDto>> GetSettings()
    {
        var settings = await _notificationSettingsService.GetAsync();
        return Ok(settings);
    }

    [HttpPut]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    public async Task<ActionResult<NotificationSettingsDto>>
UpdateSettings(
        [FromBody] UpdateNotificationSettingsRequest request)
    {
        var updateDto = _mapper.Map<UpdateNotificationSettingsDto>
(request);
        var updatedSettings = await
_notificationSettingsService.UpdateAsync(updateDto);
        return Ok(updatedSettings);
    }
}
```

Request/Response DTOs

The controllers use several DTOs for request and response handling:

1. **Exchange DTOs**
   - `ExchangeDto` : Response DTO for exchange data
   - `CreateExchangeRequest` : Request DTO for creating exchanges
2. **Symbol DTOs**
   - `SymbolDto` : Response DTO for symbol data
   - `AddSymbolRequest` : Request DTO for adding or updating symbols
   - Includes market type, status, and trading parameters
3. **Status DTOs**
   - `SymbolHistoryDto` : Response DTO for symbol status history

- Tracks status changes over time

4. **Notification Settings DTOs**
   - `NotificationSettingsDto`: Response DTO for notification settings
   - `UpdateNotificationSettingsRequest`: Request DTO for updating notification settings

Request Models

Request models define the structure of incoming HTTP requests.

CreateExchangeRequest

Used for creating new exchanges:

```
public record CreateExchangeRequest
{
    public string Name { get; init; } = null!;
}
```

UpdateNotificationSettingsRequest

Used for updating notification settings:

```
public record UpdateNotificationSettingsRequest
{
    /// <summary>
    /// Gets or sets whether Telegram notifications are enabled
    /// </summary>
    public bool IsTelegramEnabled { get; init; }

    /// <summary>
    /// Gets or sets whether Email notifications are enabled
    /// </summary>
    public bool IsEmailEnabled { get; init; }
}
```

AddSymbolRequest

Used for adding or updating symbols:

```
public record AddSymbolRequest
{
    public string SymbolName { get; init; } = null!;
    public MarketType MarketType { get; init; }
    public string BaseAsset { get; init; } = null!;
    public string QuoteAsset { get; init; } = null!;
```

```
    public int PricePrecision { get; init; }
    public int QuantityPrecision { get; init; }
    public ContractType? ContractType { get; init; }
    public DateTime? DeliveryDate { get; init; }
    public string MarginAsset { get; init; } = null!;
    public decimal MinQuantity { get; init; }
    public decimal MinNotional { get; init; }
    public decimal MaxQuantity { get; init; }
}
```

Object Mapping

The Presentation layer uses `AutoMapper` to map between request models and application DTOs:

```
public class MappingProfile : Profile
{
    public MappingProfile()
    {
        CreateMap<CreateExchangeRequest, CreateExchangeDto>();
        CreateMap<UpdateNotificationSettingsRequest,
UpdateNotificationSettingsDto>();
    }
}
```

Custom Filters

ValidatePaginationAttribute

It is used to validate pagination parameters in one place. This filter is registered as a global action filter, but is also available to use as an attribute for action methods or entire controller classes.

```
[AttributeUsage(AttributeTargets.Method | AttributeTargets.Class)]
public class ValidatePaginationAttribute : ActionFilterAttribute
{
    private const int MinPageNumber = 1;
    private const int MinPageSize = 1;

    public override void OnActionExecuting(ActionExecutingContext context)
    {
        if (context.ActionArguments.TryGetValue("page", out var pageObj)
&&
            pageObj is int page)
        {
            if (page < MinPageNumber)
            {
```

```
                context.ModelState.AddModelError(
                    "page", $"Page must be greater than or equal to
{MinPageNumber}");
            }
        }

        if (context.ActionArguments.TryGetValue("pageSize", out var
pageSizeObj) &&
            pageSizeObj is int pageSize)
        {
            if (pageSize < MinPageSize)
            {
                context.ModelState.AddModelError(
                    "pageSize", $"PageSize must be greater than or equal
to {MinPageSize}");
            }
        }

        if (!context.ModelState.IsValid)
        {
            context.Result = new
BadRequestObjectResult(context.ModelState);
        }
    }
}
```

Global Exception Handling

The Presentation layer includes middleware for handling exceptions and converting them to appropriate HTTP responses:

```
public class GlobalExceptionHandlingMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<GlobalExceptionHandlingMiddleware> _logger;
    private readonly IWebHostEnvironment _environment;

    public GlobalExceptionHandlingMiddleware(
        RequestDelegate next,
        ILogger<GlobalExceptionHandlingMiddleware> logger,
        IWebHostEnvironment environment)
    {
        _next = next;
        _logger = logger;
        _environment = environment;
    }

    public async Task InvokeAsync(HttpContext context)
```

```
    {
        try
        {
            await _next(context);
        }
        catch (Exception ex)
        {
            var traceId = context.TraceIdentifier;
            _logger.LogError(ex, "An unhandled exception occurred.
TraceId: {TraceId}", traceId);
            var problemDetailsFactory =
context.RequestServices.GetRequiredService<ProblemDetailsFactory>();
            await HandleExceptionAsync(context, ex,
problemDetailsFactory);
        }
    }

    private async Task HandleExceptionAsync(
        HttpContext context,
        Exception exception,
        ProblemDetailsFactory problemDetailsFactory)
    {
        context.Response.ContentType = "application/problem+json";

        // Handle validation error separately, cause ProblemDetails does
not have Problems
        // property unlike ValidationProblemDetails.
        if (exception is ValidationException validationException)
        {
            var validationProblemDetails =
CreateValidationProblemDetails(context, validationException);
            context.Response.StatusCode =
validationProblemDetails.Status!.Value;
            await
context.Response.WriteAsJsonAsync(validationProblemDetails);
            return;
        }

        // Handle not found error separately, cause it has a specific
status code.
        if (exception is NotFoundException notFoundException)
        {
            var problemDetails = CreateNotFoundProblemDetails(context,
problemDetailsFactory, notFoundException);
            context.Response.StatusCode = problemDetails.Status!.Value;
            await context.Response.WriteAsJsonAsync(problemDetails);
            return;
        }

        // Handle all other exceptions.
```

```
        var generalProblemDetails = CreateProblemDetails(context,
exception, problemDetailsFactory);
        context.Response.StatusCode = generalProblemDetails.Status!.Value;
        await context.Response.WriteAsJsonAsync(generalProblemDetails);
    }

    // Helper methods for creating problem details...
}
```

The middleware is registered in the application pipeline using an extension method:

```
public static class ApplicationBuilderExtensions
{
    public static IApplicationBuilder AddPresentation(
        this IApplicationBuilder app, IWebHostEnvironment environment)
    {
        app.UseMiddleware<GlobalExceptionHandlingMiddleware>();

        // If runs in development environment, configures Swagger to test
endpoints.
        if (environment.IsDevelopment())
        {
            app.UseSwagger(c =>
            {
                c.SerializeAsV2 = false;
                c.RouteTemplate = "swagger/{documentName}/swagger.json";
            });

            app.UseSwaggerUI(options =>
            {
                options.SwaggerEndpoint("/swagger/v1/swagger.json",
"InfoSymbolServer API v1");
            });
        }

        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });

        return app;
    }
}
```

Service Registration

The presentation layer registers its services through an extension method:

```
public static class ServiceCollectionExtensions
{
    public static IServiceCollection AddPresentation(this
IServiceCollection services)
    {
        services
            .AddControllers(options =>
options.Filters.Add<ValidatePaginationAttribute>())
            .AddJsonOptions(options =>
            {
                options.JsonSerializerOptions.PropertyNamingPolicy =
JsonNamingPolicy.CamelCase;
                options.JsonSerializerOptions.WriteIndented = true;
                options.JsonSerializerOptions.Converters.Add(new
JsonStringEnumConverter());
            });

        services.AddEndpointsApiExplorer();

        // Configures Swagger, that will be used in Development
environment.
        services.AddSwaggerGen(options =>
        {
            options.SwaggerDoc("v1", new OpenApiInfo
            {
                Title = "InfoSymbolServer API",
                Version = "v1",
                Description = "API for managing exchange symbols and
trading information"
            });

            // Configure xml documentation for Swagger endpoints and
objects.
            var presentationXmlFile = $"
{Assembly.GetExecutingAssembly().GetName().Name}.xml";
            var presentationXmlPath =
Path.Combine(AppContext.BaseDirectory, presentationXmlFile);
            if (File.Exists(presentationXmlPath))
            {
                options.IncludeXmlComments(presentationXmlPath);
            }

            var applicationXmlFile = "InfoSymbolServer.Application.xml";
            var applicationXmlPath =
Path.Combine(AppContext.BaseDirectory, applicationXmlFile);
            if (File.Exists(applicationXmlPath))
            {
```

```
                options.IncludeXmlComments(applicationXmlPath);
            }
        });

        return services;
    }
}
```

API Documentation

The Presentation layer provides API documentation through Swagger/OpenAPI:

- **Swagger UI**: Available at `/swagger` in development environment
- **OpenAPI Specification**: Provides machine-readable API documentation
- **Response Types**: Each endpoint documents its possible response types

Swagger Configuration

The Swagger UI is configured and made available in the development environment:

```
if (environment.IsDevelopment())
{
    app.UseSwagger(c =>
    {
        c.SerializeAsV2 = false;
        c.RouteTemplate = "swagger/{documentName}/swagger.json";
    });

    app.UseSwaggerUI(options =>
    {
        options.SwaggerEndpoint("/swagger/v1/swagger.json",
"InfoSymbolServer API v1");
    });
}
```