InfoSymbolServer.Infrastructure.Notifications

Overview

The Notification System in InfoSymbolServer provides a comprehensive mechanism for alerting administrators about symbol changes and system status. It consists of two main notification types:

1. Regular notifications about symbol changes (via Telegram and Email)
2. Emergency notifications about system issues (via Email)

The system also includes notification settings feature that allows administrators to enable or disable specific notification types through the API.

> **Important**: Notifications are optional and will be automatically disabled if not properly configured in the application settings. Administrator can skip configuring notifications if they don't need to receive updates about symbol changes or system issues.

Table of Contents

Architecture

The notification system follows a clean architecture pattern with clear separation of concerns:

- **Interfaces**: Define contracts for notification services, validation services, and message formatters
- **Services**: Implement the actual notification delivery
- **Formatters**: Handle message formatting for different output types
- **Options**: Configure notification services
- **Settings**: Control global notification behavior
- **Validation**: Verify notification configuration status

Notification Services

Regular Notifications

InfoSymbolServer supports two types of regular notifications:

Telegram Notifications

The `TelegramNotificationService` handles regular notifications about symbol changes via Telegram:

```
public interface INotificationService
{
    Task<bool> AreNotificationsEnabledAsync();
    Task SendNewSymbolsNotificationAsync(IEnumerable<Symbol> symbols,
string exchangeName);
    Task SendUpdatedSymbolsNotificationAsync(IEnumerable<Symbol> symbols,
string exchangeName);
```

```
    Task SendDelistedSymbolsNotificationAsync(IEnumerable<Symbol> symbols,
string exchangeName);
    Task SendCombinedSymbolChangesNotificationAsync(
        IEnumerable<Symbol> newSymbols,
        IEnumerable<Symbol> updatedSymbols,
        IEnumerable<Symbol> delistedSymbols,
        string exchangeName,
        string marketType);
}
```

Key features:

- Sends notifications via Telegram Bot API
- Supports individual notifications for new, updated, and delisted symbols
- Provides combined notifications for all symbol changes
- Includes configurable message formatting options
- Respects global notification settings
- Automatically disabled if not properly configured
- Uses validation service to check configuration status

Email Notifications

The `EmailNotificationService` also implements the same `INotificationService` interface but sends notifications via email:

Key features:

- Sends regular notifications via SMTP
- Uses HTML message formatter for rich message display
- Supports sending to multiple recipients
- Respects global notification settings
- Automatically disabled if not properly configured
- Uses validation service to check configuration status

Emergency Notifications (Email)

The `EmailEmergencyNotificationService` handles critical system notifications:

```
public interface IEmergencyNotificationService
{
    Task SendExchangeApiErrorNotificationAsync(string exchangeName, string
errorMessage, Exception? exception = null);
    Task SendDatabaseErrorNotificationAsync(string operation, string
errorMessage, Exception? exception = null);
    Task SendSystemErrorNotificationAsync(string component, string
errorMessage, Exception? exception = null);
}
```

Key features:

- Sends emergency notifications via SMTP
- Handles three types of emergencies:
  - Exchange API errors
  - Database errors
  - System errors
- Includes detailed error information and stack traces
- Supports multiple recipients
- Automatically disabled if not properly configured
- Uses validation service to check configuration status

Notification Settings

The notification system includes settings feature that allows administrators to enable or disable specific notification types:

NotificationSettings Model

```
public class NotificationSettings
{
    public Guid Id { get; set; }
    public bool IsTelegramEnabled { get; set; }
    public bool IsEmailEnabled { get; set; }
}
```

Repository Interface

```
public interface INotificationSettingsRepository
{
    Task<NotificationSettings?> GetAsync();
    Task<NotificationSettings> UpdateAsync(NotificationSettings settings);
}
```

Returned DTO

The notification settings DTO also includes information about whether each notification type is properly configured:

```
public record NotificationSettingsDto
{
    public bool IsTelegramEnabled { get; init; }
    public bool IsEmailEnabled { get; init; }
    public bool IsTelegramConfigured { get; init; }
    public bool IsEmailConfigured { get; init; }
}
```

Message Formatters

Markdown Formatter

The `MarkdownNotificationMessageFormatter` formats regular notifications for Telegram:

- Supports detailed and concise message formats
- Limits the number of symbols per message
- Includes timestamps and summaries

Example format:

```
📊 *Symbol changes on BinanceSpot (Spot)*
_2024-03-14 15:30:00 UTC_

🆕 *New Symbols (2)*
- BTCUSDT (Spot)
  Base: BTC, Quote: USDT
  Status: Trading

🔄 *Updated Symbols (1)*
- ETHUSDT (Spot)
  Base: ETH, Quote: USDT
  Status: Break

⛔ *Delisted Symbols (1)*
- XRPUSDT (Spot)

*Total changes: 4*
```

HTML Notification Formatter

The `HtmlNotificationMessageFormatter` formats regular notifications for email:

- Uses HTML formatting for better readability in email clients
- Applies consistent styling with CSS
- Provides structured content with sections for different symbol types
- Respects the same symbol limits as the Markdown formatter

Example format:

```
<div style='font-family: Arial, sans-serif; max-width: 800px;'>
  <h2>📊 Symbol changes on BinanceSpot (Spot)</h2>
  <p><em>2024-03-14 15:30:00 UTC</em></p>

  <h3>🆕 New Symbols (2)</h3>
  <ul style='list-style-type: none; padding-left: 10px;'>
    <li style='margin-bottom: 10px;'>
      <code style='background-color: #f5f5f5; padding: 2px 4px; border-
radius: 3px;'>BTCUSDT</code> (Spot)
      <div style='margin-left: 20px;'>
        Base: <code style='background-color: #f5f5f5; padding: 2px 4px;
border-radius: 3px;'>BTC</code>,
        Quote: <code style='background-color: #f5f5f5; padding: 2px 4px;
border-radius: 3px;'>USDT</code><br>
        Status: Trading
      </div>
    </li>
    <!-- More symbols here -->
  </ul>

  <!-- Updated and Delisted sections follow the same pattern -->

  <p><strong>Total changes: 4</strong></p>
</div>
```

HTML Emergency Formatter

The `HtmlEmergencyNotificationMessageFormatter` formats emergency notifications for email:

- Uses HTML formatting for better readability
- Includes styled headers and sections
- Provides detailed exception information
- Adds action items for each type of emergency

Example format:

```
<h2 style='color: #cc0000;'>Exchange API Error</h2>
<p><strong>Exchange:</strong> BinanceSpot</p>
<p><strong>Time:</strong> 2024-03-14 15:30:00 UTC</p>
<p><strong>Error Message:</strong> Failed to fetch symbol information</p>
<h3>Exception Details:</h3>
<pre style='background-color: #f8f8f8; padding: 10px; border-radius:
5px;'>
Type: System.Net.Http.HttpRequestException
```

```
Message: Connection timed out
Stack Trace: ...
</pre>
<p>Please take immediate action to resolve this issue.</p>
```

Validation Services

The notification system includes validation services to check if notifications are properly configured:

```
public interface INotificationValidationService
{
    bool IsTelegramValid();
    bool IsEmailValid();
    bool IsEmergencyEmailValid();
}
```

The implementation (`NotificationValidationService`) checks:

- If Telegram has a valid bot token and at least one chat ID
- If regular email has valid SMTP settings and at least one recipient
- If emergency email has valid SMTP settings and at least one recipient

These validation methods are used by:

- Notification services to determine if they should attempt to send notifications
- Application startup to log warnings about misconfigured notification channels
- Admin API to prevent enabling notifications that are not properly configured

Configuration

The notification system is configured through the `appsettings.json` file:

SMTP Configuration

```
"Notifications": {
  "Smtp": {
    "SmtpServer": "smtp.your-domain.com",
    "SmtpPort": 587,
    "UseSsl": true,
    "SenderEmail": "notifications@your-domain.com",
    "SenderName": "InfoSymbolServer Notifications",
    "Username": "notifications@your-domain.com",
    "Password": "your-smtp-password"
  }
}
```

Telegram Configuration

```
"Notifications": {
  "Telegram": {
    "BotToken": "your-telegram-bot-token",
    "ChatIds": ["chat-id-1", "chat-id-2"],
    "IncludeDetailedInfo": true,
    "MaxSymbolsPerMessage": 20
  }
}
```

Email Configuration

```
"Notifications": {
  "Email": {
    "Recipients": [
      "admin1@example.com",
      "admin2@example.com"
    ]
  }
}
```

Emergency Email Configuration

```
"Notifications": {
  "EmailEmergency": {
    "Recipients": [
      "admin1@example.com",
      "admin2@example.com"
    ]
  }
}
```

Dependency Registration

The notification system is registered in the DI container through the `AddInfrastructure` extension method:

```
// Configure shared SMTP settings
services.Configure<SmtpSettings>(
    configuration.GetSection("Notifications:Smtp"));

// Configure notification channel options
services.Configure<TelegramNotificationOptions>(
    configuration.GetSection("Notifications:Telegram"));
```

```
services.Configure<EmailNotificationOptions>(
    configuration.GetSection("Notifications:Email"));

services.Configure<EmergencyEmailOptions>(
    configuration.GetSection("Notifications:EmailEmergency"));

// Register notification validation service
services.AddScoped<INotificationValidationService,
NotificationValidationService>();

// Register notification services
services.AddScoped<INotificationService, TelegramNotificationService>();
services.AddScoped<INotificationService, EmailNotificationService>();
services.AddScoped<IEmergencyNotificationService,
EmailEmergencyNotificationService>();
```

Usage in Background Jobs

The notification services are used in background jobs to report symbol changes and system errors:

```
public class BaseBinanceSymbolSyncJob
{
    private readonly IEnumerable<INotificationService>
_notificationServices;
    private readonly IEmergencyNotificationService
_emergencyNotificationService;

    public async Task Execute(IJobExecutionContext context)
    {
        try
        {
            // Process symbols...

            // Send notifications about changes to all configured services
            foreach (var notificationService in _notificationServices)
            {
                await
notificationService.SendCombinedSymbolChangesNotificationAsync(
                    newSymbols, updatedSymbols, delistedSymbols,
exchange.Name, marketType);
            }
        }
        // catch blocks for specific errors, like API error or Database
error
        catch (Exception ex)
        {
            // Send emergency notification
            await
_emergencyNotificationService.SendSystemErrorNotificationAsync(
                nameof(BaseBinanceSymbolSyncJob), ex.Message, ex);
        }
    }
}
```