**Техническое задание: Рефакторинг и улучшение приложения для торговли облигациями**

**Обзор проекта**

Приложение для торговли облигациями — это система на базе WPF, использующая библиотеку StockSharp для торговли российскими ОФЗ на Московской бирже (ММВБ). Приложение интегрировано с Telegram-ботом для удаленного управления и использует кривую доходности (OfzCurve) для стратегий маркет-мейкинга. Текущая реализация имеет ряд проблем, включая жестко закодированные значения, закомментированный код, нереляционную структуру базы данных и отсутствие хранения транзакций и параметров в базе данных. Данное задание описывает требования к рефакторингу кода, улучшению пользовательского интерфейса и внедрению хранения данных в базе данных.

**Цели**

1. **Рефакторинг кода**:
   * Повысить качество, читаемость и поддерживаемость кода.
   * Удалить неиспользуемый или закомментированный код.
   * Устранить технический долг (например, жестко закодированные значения, слабая обработка ошибок).
   * Обеспечить безопасность потоков и оптимизировать производительность.
2. **Улучшение пользовательского интерфейса**:
   * Добавить поле поиска для фильтрации облигаций в интерфейсе.
   * Повысить удобство использования существующих компонентов интерфейса.
3. **Улучшение базы данных**:
   * Перепроектировать базу данных для использования реляционной структуры.
   * Сохранять транзакции в базе данных.
   * Сохранять параметры торговли (например, объемы, лимиты, отступы) в базе данных вместо файла.

**Требования**

**1. Рефакторинг кода**

**Общие требования**

* **Удаление неиспользуемого кода**:
  + Удалить закомментированный код (например, старый массив OfzCodes, методы OnProcessMDepthChangedForSizeQuoteStrategy, OnProcessMDepthChangedForSmallQuoteStrategy).
  + Удалить неиспользуемые переменные и словари (например, SFutList, \_ordersForFutBuyReregister, \_ordersForFutSellReregister).
  + Удалить метод StartClick с атрибутом [Obsolete] или переработать его в поддерживаемый метод.
* **Управление конфигурацией**:
  + Заменить жестко закодированные значения (PortName, ClCode, ProgramPath, токен Telegram-бота) на конфигурацию, хранимую в файле appsettings.json.
  + Использовать фреймворк внедрения зависимостей (например, Microsoft.Extensions.DependencyInjection) для управления конфигурацией и сервисами.
* **Обработка ошибок**:
  + Добавить комплексную обработку ошибок для операций ввода-вывода (например, доступ к базе данных, наблюдение за файлами).
  + Логировать исключения с использованием структурированного фреймворка логирования (например, Serilog) с подробным контекстом (например, временные метки, идентификаторы ордеров).
  + Обрабатывать ошибки Telegram-бота (например, HttpRequestException) с логикой повторных попыток и логированием вместо диалоговых окон.
* **Безопасность потоков**:
  + Убедиться, что все обновления интерфейса используют Dispatcher.Invoke или GuiAsync для предотвращения проблем с потоками.
  + Последовательно использовать async/await для операций, связанных с вводом-выводом (например, запросы к базе данных, вызовы Telegram API).
* **Оптимизация производительности**:
  + Оптимизировать метод OnProcessMDepthChangedForQuoteStrategy, избегая ненужного клонирования depth и используя эффективные структуры данных.
  + Использовать LINQ или параллельную обработку для итераций BondList при работе с большими списками.
  + Периодически очищать TradesList (например, ежедневно или по сессиям) для предотвращения роста потребления памяти.
* **Структура кода**:
  + Разделить класс MainWindow на меньшие, специализированные классы (например, TradingService, BondManager, TelegramBotHandler) для повышения поддерживаемости.
  + Использовать интерфейсы для ключевых компонентов (например, IConnector, ILogger) для упрощения тестирования.
  + Добавить модульные тесты для критических методов (например, OnProcessMDepthChangedForQuoteStrategy, OnProcessNewTrade) с использованием фреймворка, такого как xUnit.

**Специфические рефакторинги**

* **Управление ордерами**:
  + Упростить логику размещения ордеров в OnProcessMDepthChangedForQuoteStrategy, вынося общий код в переиспользуемые методы.
  + Объединить дублирующуюся логику для ордеров на покупку и продажу.
  + Убедиться, что IsMarketMaker устанавливается последовательно в зависимости от типа облигаций (OfzCodes, OfzCodesNew).
* **Наблюдатель за файлами**:
  + Добавить механизм подавления дребезга в OnDatabaseChanged для предотвращения быстрого срабатывания событий.
  + Логировать ошибки доступа к файлам и реализовать логику повторных попыток для доступа к базе данных.
* **Telegram-бот**:
  + Проверять команды более тщательно (например, проверять коды облигаций, числовые входные данные).
  + Добавить аутентификацию для ограничения доступа к командам только для авторизованных пользователей (например, проверять message.From.Id по белому списку в базе данных).
  + Переработать BotOnMessageReceived, используя шаблон команды для улучшения поддерживаемости.

**2. Улучшение пользовательского интерфейса**

**Поле поиска**

* Добавить поле поиска в панель MyProgram для фильтрации облигаций по коду (например, SU26234RMFS3) или имени (например, OFZ\_26234).
* Реализовать фильтрацию в реальном времени по мере ввода текста, обновляя отображаемый список облигаций (CbSecurities.ItemsSource).
* Поддерживать частичное совпадение (например, поиск "26234" показывает все облигации, содержащие "26234").
* Убедиться, что поле поиска доступно и стилизовано в соответствии с другими элементами интерфейса (например, с использованием XAML и элементов управления WPF, таких как TextBox).

**Улучшения интерфейса**

* Стандартизировать имена элементов управления (например, Price\_26234, WorkingVolume\_26234) и обеспечить их единообразие для всех облигаций.
* Добавить всплывающие подсказки к элементам управления, объясняющие их назначение (например, "LowLimit: минимальное отклонение цены в базисных пунктах").
* Отключать неактуальные элементы управления (например, btnStart), когда система не подключена или не инициализирована.
* Добавить строку состояния, отображающую статус подключения, время последнего обновления базы данных и общий объем торгов.
* Убедиться, что обновления интерфейса безопасны для потоков с использованием Dispatcher.Invoke.

**3. Улучшение базы данных**

**Текущие проблемы**

* База данных (RedArrowData.db) имеет нереляционную структуру и используется только для кривой доходности (OfzCurve).
* Транзакции не сохраняются в базе данных, а отслеживаются только в памяти (TradesList).
* Параметры (например, объемы, лимиты, отступы) сохраняются в текстовом файле (Parameters.txt), что подвержено ошибкам и не поддерживает версионность.

**Новая структура базы данных**

* **Тип базы данных**: Использовать SQLite (поскольку RedArrowData.db, вероятно, является SQLite) для простоты и совместимости.
* **Схема**:  
  Создать следующие реляционные таблицы:
  1. **Облигации (Bonds)**:
     + Id (Первичный ключ, автоинкремент)
     + Code (Текст, например, SU26234RMFS3)
     + Name (Текст, например, OFZ\_26234)
     + ISIN (Текст, например, RU000A101QE0)
     + IssueDate (Дата, например, 2020-07-22)
     + MaturityDate (Дата, например, 2025-07-16)
     + CouponRate (Десятичное, например, 0.045)
     + Board (Текст, например, MicexTqob)
  2. **Транзакции (Transactions)**:
     + Id (Первичный ключ, автоинкремент)
     + BondId (Внешний ключ, ссылается на Bonds.Id)
     + TradeId (Текст, уникальный, из MyTrade.Trade.Id)
     + Side (Текст, например, Buy или Sell)
     + Price (Десятичное)
     + Volume (Десятичное)
     + Strategy (Текст, например, Quote, OfRStrategy)
     + Timestamp (Дата и время)
     + ClientCode (Текст, например, SEACB\_01/)
  3. **Параметры (Parameters)**:
     + Id (Первичный ключ, автоинкремент)
     + BondId (Внешний ключ, ссылается на Bonds.Id)
     + SmallBidVolume (Длинное целое, например, 2000)
     + SmallOfferVolume (Длинное целое, например, 2000)
     + WorkingVolume (Длинное целое, например, 1000)
     + SmallOffset (Целое, например, 0)
     + Offset (Целое, например, 0)
     + LowLimit (Целое, например, 19)
     + HighLimit (Целое, например, 25)
     + IsMarketMaker (Булево, например, true для OfzCodes)
  4. **Кривая доходности (YieldCurve)** (заменяет текущую нереляционную структуру):
     + Id (Первичный ключ, автоинкремент)
     + BondId (Внешний ключ, ссылается на Bonds.Id)
     + CurveDate (Дата)
     + ModelPrice (Десятичное)
     + Yield (Десятичное, необязательное, для справки)
* **Реализация**:
  1. Использовать ORM, такой как Entity Framework Core (с провайдером SQLite), для операций с базой данных.
  2. Перенести существующие данные об облигациях из логики создания SBondList в таблицу Bonds.
  3. Обновить OfzCurve для чтения/записи из таблицы YieldCurve вместо текущей нереляционной структуры.
  4. В OnProcessNewTrade сохранять каждую сделку в таблицу Transactions.
  5. Заменить SaveParamsClick и LoadParamsClick на операции с базой данных для хранения/извлечения параметров из таблицы Parameters.
  6. Обеспечить потокобезопасный доступ к базе данных с использованием async/await и правильного управления соединениями.
* **Валидация данных**:
  1. Проверять коды облигаций, цены и объемы перед сохранением в базу данных.
  2. Обеспечить соблюдение ограничений внешних ключей (например, BondId должен существовать в Bonds).
  3. Добавить индексы на часто запрашиваемые столбцы (например, Transactions.TradeId, Bonds.Code).
* **Миграция**:
  1. Создать скрипт миграции для преобразования существующей базы RedArrowData.db в новую схему.
  2. Импортировать существующие параметры из Parameters.txt в таблицу Parameters во время миграции.

**Операции с базой данных**

* **Транзакции**:
  + В OnProcessNewTrade вставлять новую запись в таблицу Transactions для каждой сделки, включая TradeId, BondId, Side, Price, Volume, Strategy, Timestamp и ClientCode.
  + Проверять наличие дубликатов TradeId перед вставкой.
* **Параметры**:
  + В SaveParamsClick выполнять обновление или вставку (upsert) параметров в таблицу Parameters для каждой облигации.
  + В LoadParamsClick извлекать параметры из таблицы Parameters и обновлять элементы управления интерфейса (SmallBidVolume, WorkingVolume и т.д.).
  + Убедиться, что применяются значения по умолчанию (например, quoteSmallStrategyBidVolume, lowLimit), если параметры отсутствуют в базе данных.
* **Кривая доходности**:
  + Обновить OfzCurve.GetCurveFromDb для чтения из таблицы YieldCurve.
  + Убедиться, что OnDatabaseChanged обновляет таблицу YieldCurve и элементы управления интерфейса.

**Результаты работы**

1. **Переработанная кодовая база**:
   * Чистый, модульный код с удаленными неиспользуемыми частями и улучшенной структурой.
   * Конфигурация, управляемая через appsettings.json.
   * Комплексная обработка ошибок и логирование.
   * Модульные тесты для критических методов.
2. **Улучшенный интерфейс**:
   * Поле поиска для фильтрации облигаций.
   * Повышенное удобство использования с подсказками, строкой состояния и потокобезопасными обновлениями.
3. **Реализация базы данных**:
   * Реляционная база данных SQLite с таблицами Bonds, Transactions, Parameters и YieldCurve.
   * Скрипт миграции для существующих данных.
   * Операции CRUD для транзакций и параметров.
   * Обновленная логика OfzCurve для использования новой схемы.
4. **Документация**:
   * Обновленный README с инструкциями по настройке, включая конфигурацию базы данных.
   * Комментарии к коду для ключевых методов и классов.
   * Руководство пользователя для нового поля поиска и функций базы данных.

**Критерии приемки**

* Приложение компилируется и запускается без ошибок.
* Все облигации из OfzCodes, OfzCodesNew и OfzCodesIlliquid загружаются в таблицу Bonds.
* Транзакции сохраняются в таблицу Transactions без дубликатов.
* Параметры сохраняются в таблицу Parameters и загружаются из нее, заменяя Parameters.txt.
* Кривая доходности корректно хранится в таблице YieldCurve и извлекается из нее.
* Поле поиска фильтрует облигации в реальном времени с поддержкой частичного совпадения.
* Обновления интерфейса потокобезопасны и отзывчивы.
* Ошибки логируются с достаточной детализацией, а команды Telegram-бота безопасны и проверены.
* Модульные тесты покрывают не менее 80% критических методов.

**Ограничения**

* **База данных**: Использовать SQLite
* **Фреймворк**: Продолжать использовать WPF и StockSharp для торговых функций.
* **Сроки**: 7 дней
* **Среда**: Целевая платформа — .NET Framework.

**Зависимости**

* **Пакеты NuGet**:
  + StockSharp.Algo, StockSharp.Xaml, StockSharp.Messages, StockSharp.Logging
  + Microsoft.Extensions.Configuration.Json для конфигурации
  + Microsoft.EntityFrameworkCore.Sqlite для доступа к базе данных
  + Serilog для логирования
  + Telegram.Bot для интеграции с Telegram
  + xUnit для модульного тестирования
* **Внешние сервисы**:
  + API Telegram-бота (требуется действительный токен).
  + Подключение к Московской бирже через StockSharp.

**Пример реализации базы данных**

Ниже приведен пример настройки базы данных с использованием Entity Framework Core:

using Microsoft.EntityFrameworkCore;

public class TradingContext : DbContext

{

public DbSet<Bond> Bonds { get; set; }

public DbSet<Transaction> Transactions { get; set; }

public DbSet<Parameter> Parameters { get; set; }

public DbSet<YieldCurve> YieldCurves { get; set; }

public TradingContext(DbContextOptions<TradingContext> options) : base(options) { }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Bond>()

.HasKey(b => b.Id);

modelBuilder.Entity<Bond>()

.HasIndex(b => b.Code)

.IsUnique();

modelBuilder.Entity<Transaction>()

.HasKey(t => t.Id);

modelBuilder.Entity<Transaction>()

.HasOne(t => t.Bond)

.WithMany()

.HasForeignKey(t => t.BondId);

modelBuilder.Entity<Transaction>()

.HasIndex(t => t.TradeId)

.IsUnique();

modelBuilder.Entity<Parameter>()

.HasKey(p => p.Id);

modelBuilder.Entity<Parameter>()

.HasOne(p => p.Bond)

.WithMany()

.HasForeignKey(p => p.BondId);

modelBuilder.Entity<YieldCurve>()

.HasKey(y => y.Id);

modelBuilder.Entity<YieldCurve>()

.HasOne(y => y.Bond)

.WithMany()

.HasForeignKey(y => y.BondId);

}

}

public class Bond

{

public int Id { get; set; }

public string Code { get; set; }

public string Name { get; set; }

public string ISIN { get; set; }

public DateTime IssueDate { get; set; }

public DateTime MaturityDate { get; set; }

public decimal CouponRate { get; set; }

public string Board { get; set; }

}

public class Transaction

{

public int Id { get; set; }

public int BondId { get; set; }

public Bond Bond { get; set; }

public string TradeId { get; set; }

public string Side { get; set; }

public decimal Price { get; set; }

public decimal Volume { get; set; }

public string Strategy { get; set; }

public DateTime Timestamp { get; set; }

public string ClientCode { get; set; }

}

public class Parameter

{

public int Id { get; set; }

public int BondId { get; set; }

public Bond Bond { get; set; }

public long SmallBidVolume { get; set; }

public long SmallOfferVolume { get; set; }

public long WorkingVolume { get; set; }

public int SmallOffset { get; set; }

public int Offset { get; set; }

public int LowLimit { get; set; }

public int HighLimit { get; set; }

public bool IsMarketMaker { get; set; }

}

public class YieldCurve

{

public int Id { get; set; }

public int BondId { get; set; }

public Bond Bond { get; set; }

public DateTime CurveDate { get; set; }

public decimal ModelPrice { get; set; }

public decimal? Yield { get; set; }

}

**Пример реализации поля поиска**

Добавить поле поиска в XAML:

<TextBox x:Name="BondSearch" Width="200" Margin="5" TextChanged="BondSearch\_TextChanged" PlaceholderText="Поиск облигаций..." />

Обработать поиск в коде:

private void BondSearch\_TextChanged(object sender, TextChangedEventArgs e)

{

var searchText = BondSearch.Text.ToLower();

var filteredBonds = BondList.Where(b => b.Code.ToLower().Contains(searchText) || b.Name?.ToLower().Contains(searchText)).ToList();

CbSecurities.ItemsSource = filteredBonds;

}