МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

ОТЧЕТ

по лабораторной работе №10

по дисциплине «Объектно-ориентированное программирование»

ТЕМА: «Организация многопоточных приложений»

Студент гр. 3314 _____ Кокорев С.С.

Преподаватель _____ Гречухин М.Н.

Санкт-Петербург

2024

Цель работы.

Знакомство с правилами и классами построения параллельных приложений в языке C#.

Задание на лабораторную работу.

1. Описать параллельные потока, которые будет загружать данные из XML-файла, редактировать данные, строить отчет в PDF - формате.
2. С помощью конструктора подготовить шаблон для отчета.
3. Запустить приложение и убедиться, что сформирован PDF-файл..

Задание курсовой работы.

Задание 1. Разработать ПК для работников библиотеки. В ПК должны храниться сведения об имеющихся в библиотеке книгах и о читателях библиотеки. Библиотекарю могут потребоваться следующие сведения:

- какие книги закреплены за читателем;
- кто автор и как называется книга с заданным шифром.

Библиотекарь может вносить следующие изменения:

- запись нового читателя в библиотеку;
- пополнение библиотеки;
- списывание старой книги;
- изменение шифра книги.

Необходимо предусмотреть возможность выдачи справки о количестве читателей библиотеки и работе библиотеки за месяц (количество выданных книг, число записавшихся читателей).

Замечание.

В своём приложении, вместо потоков, я использую асинхронность, т.к. её вполне достаточно для реализации всего параллельного функционала, и при этом она гораздо проще в использовании, чем потоки.

Асинхронный метод генерации PDF-отчёта.

```
public async Task GenerateReportAsync()
{
    var books = _manager.Books;

    await Task.Run(async () =>
    {
        Document.Create(container =>
        {
            container.Page(page =>
            {
                page.Size(PageSizes.A4);
                page.Margin(2, Unit.Centimetre);
                page.PageColor(Colors.White);
                page.DefaultTextStyle(x => x.FontSize(16));

                page.Header()
                    .Text("Library report")
                    .SemiBold().FontSize(36);

                page.Content().PaddingVertical(10).Column(column =>
                {
                    column.Spacing(10);

                    column.Item().Text("Books in library").SemiBold().AlignCenter();
                    column.Item().Table(table =>
                    {
                        table.ColumnsDefinition(columns =>
                        {
                            columns.RelativeColumn();
                            columns.RelativeColumn();
                            columns.RelativeColumn();
                            columns.RelativeColumn();
                        });

                        for (int i = 0; i < _manager.Books.Count; i++)
                        {
                            table.Cell().Text(books[i].Id);
                            table.Cell().Text(books[i].Name);
                            table.Cell().Text(books[i].BookAuthor.FullName);
                            table.Cell().Text(books[i].Year);
                        }
                    });
                });

                page.Footer()
                    .AlignCenter()
                    .Text(x =>
                    {
                        x.Span("Page ");
                        x.CurrentPageNumber();
                    });
            });
        })
        .GeneratePdfAndShow();
    });
}
```

Асинхронный метод генерации PDF-отчёта.

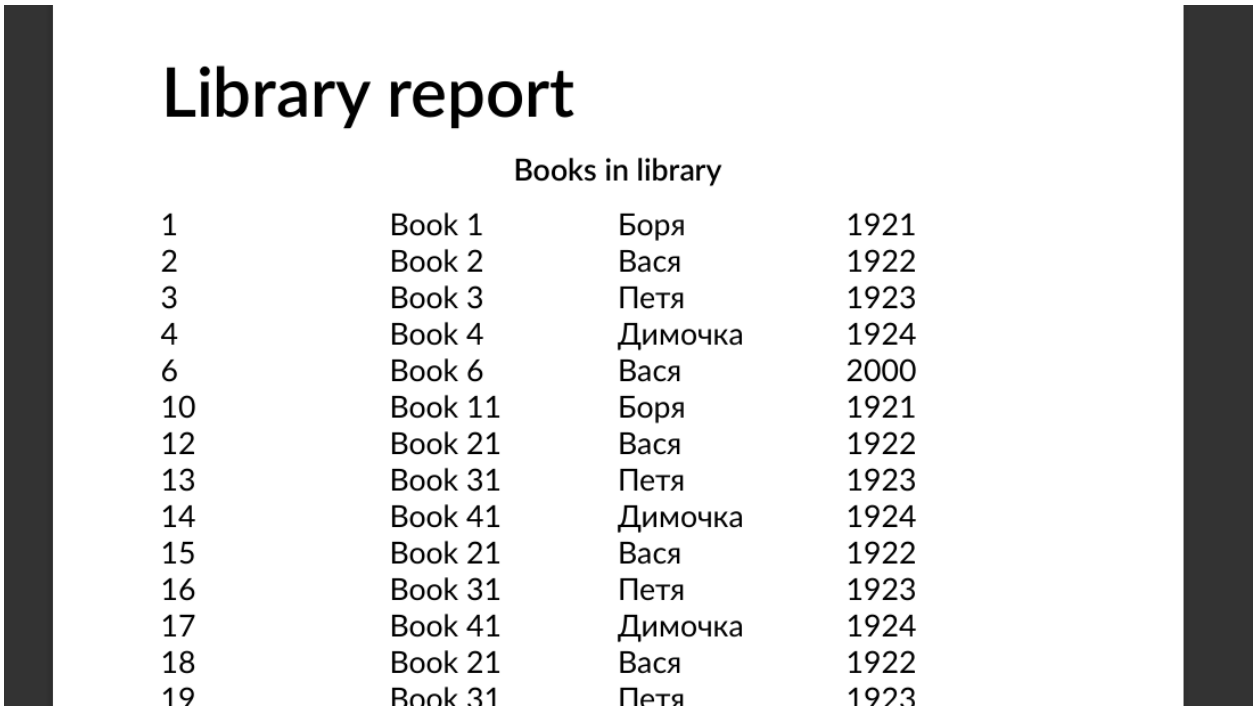
Также я сделал все взаимодействия с базой данных (через EF Core) асинхронными. Вот несколько примеров:

```
public async Task AddVisitorAsync(Visitor visitor)
{
    Visitors.Add(visitor);
    _dbContext.Visitors.Add(visitor);
    await _dbContext.SaveChangesAsync();
}
public async Task AddBookAsync(Book book)
{
    Books.Add(book);
    _dbContext.Books.Add(book);
    await _dbContext.SaveChangesAsync();
}

public async Task EditVisitorAsync(Visitor oldVer, Visitor newVer)
{
    int index = Visitors.IndexOf(oldVer);
    if (index != -1)
    {
        var existingVisitor = await _dbContext.Visitors.FindAsync(oldVer.Id);

        if (existingVisitor != null)
        {
            _dbContext.Entry(existingVisitor).CurrentValues.SetValues(newVer);
            await _dbContext.SaveChangesAsync();
            Visitors[index] = newVer;
        }
    }
}
```

Пример сгенерированного PDF-отчёта.



Library report

Books in library

1	Book 1	Боря	1921
2	Book 2	Вася	1922
3	Book 3	Петя	1923
4	Book 4	Димочка	1924
6	Book 6	Вася	2000
10	Book 11	Боря	1921
12	Book 21	Вася	1922
13	Book 31	Петя	1923
14	Book 41	Димочка	1924
15	Book 21	Вася	1922
16	Book 31	Петя	1923
17	Book 41	Димочка	1924
18	Book 21	Вася	1922
19	Book 31	Петя	1923

Вывод

В ходе лабораторной работы была изучена и применена на практике асинхронность в C# (async await). Все затратные по времени функции, сделаны асинхронными, чтобы не блокировать UI.